

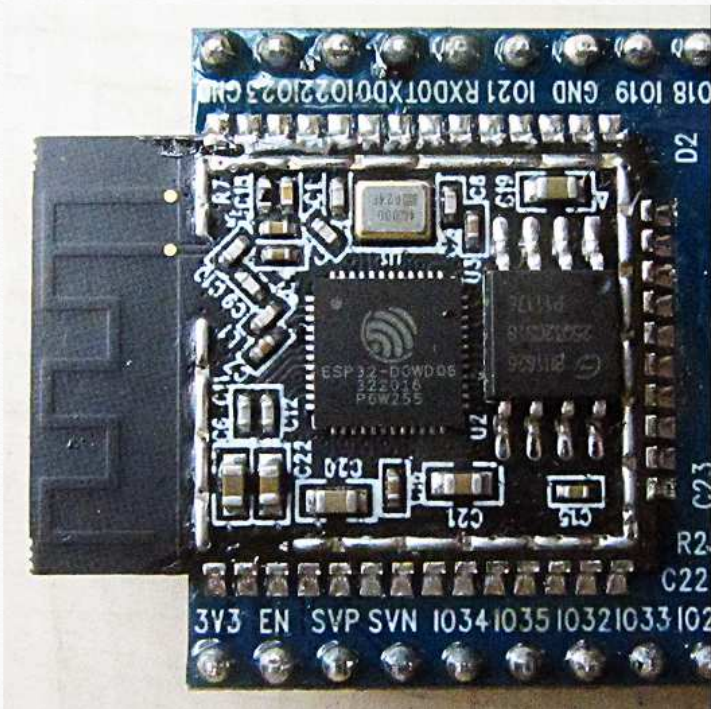
ESP32

YEISON JAVIER MONTAGUT FERIZZOLA



¿QUÉ ES ESP32?

ESP32



<https://www.espressif.com/>



[https://commons.wikimedia.org/wiki/File:ESP32_Espressif_ESP-WROOM-32_Shielded.jpg]
[https://upload.wikimedia.org/wikipedia/commons/7/7b/ESP32_Espressif_ESP-WROOM-32_Shielded.jpg]

ESP32

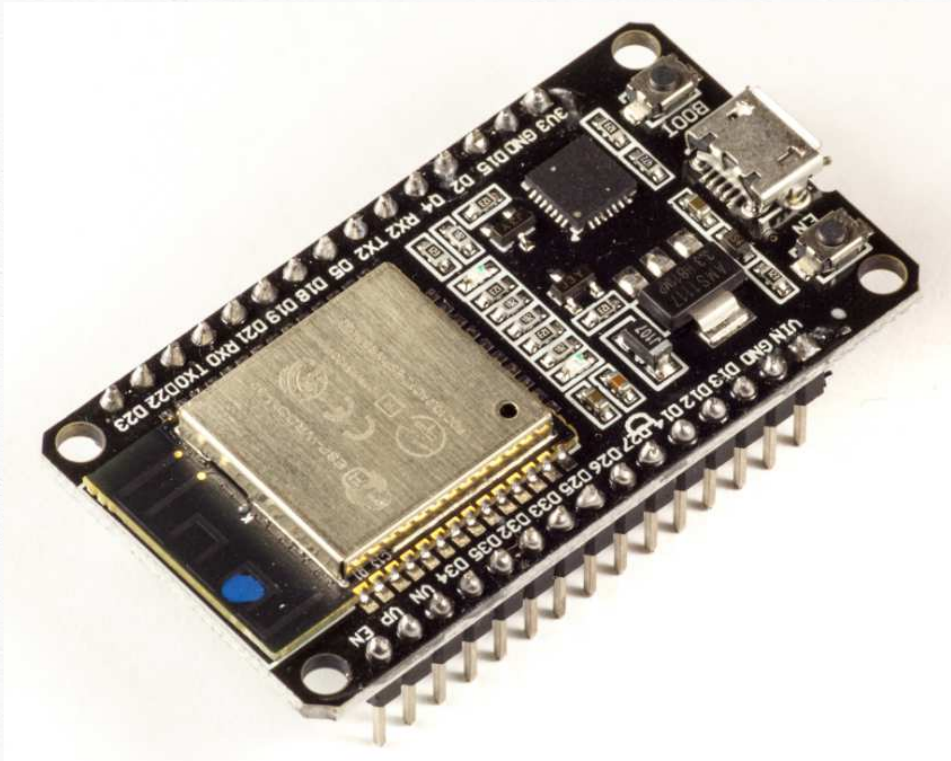


<https://www.espressif.com/>



[https://commons.wikimedia.org/wiki/File:ESP32_Espressif_ESP-WROOM-32_Shielded.jpg]
[https://upload.wikimedia.org/wikipedia/commons/7/7b/ESP32_Espressif_ESP-WROOM-32_Shielded.jpg]

ESP32



<https://www.espressif.com/>



[[https://commons.wikimedia.org/wiki/File:ESP32_Espressif_ESP-WROOM-32_Dev_Board_\(2\).jpg](https://commons.wikimedia.org/wiki/File:ESP32_Espressif_ESP-WROOM-32_Dev_Board_(2).jpg)]
[https://upload.wikimedia.org/wikipedia/commons/1/1d/ESP32_Espressif_ESP-WROOM-32_Dev_Board_%28%29.jpg]

¿POR QUÉ ESP32?

¿POR QUÉ ESP32?

- Bajo costo
- Wi-Fi y Bluetooth
- Disponible en un módulo.
- No requiere programador (cuando viene en un módulo)
- Aplicaciones IoT
- Usado en asignatura: Adquisición de señales electrofisiológicas (Ingeniería Biomédica), Procesamiento de señales (Maestría Ingeniería Biomédica) e IoT (Ingeniería Mecatrónica)



CARACTERÍSTICAS

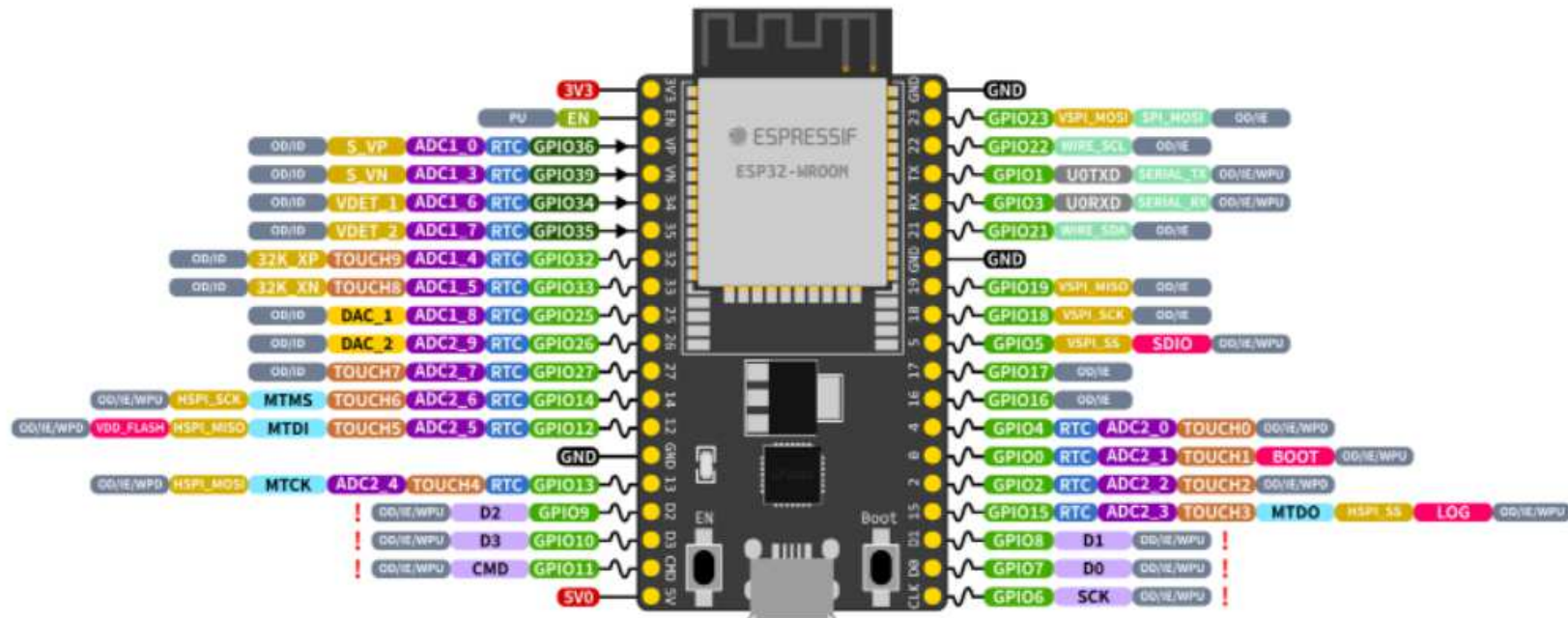
CARACTERÍSTICAS

- ✓ CPU 32-bits Xtensa LX6 dual-core 240 MHz
- ✓ ROM 448 KB
- ✓ RAM 520 KB
- ✓ Wi-Fi 802.11 n (2.4 GHz)
- ✓ Bluetooth v4.2
- ✓ GPIOs
- ✓ ADC
- ✓ DAC

CARACTERÍSTICAS

- ✓ SPI
- ✓ I2C
- ✓ I2S
- ✓ UART
- ✓ PWM
- ✓ CAN
- ✓ RTC (Real-Time clock)
- ✓ ULP (Ultra-Low-Power) 100uA

ESP32-DevKitC



ESP32 Specs

32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
Bluetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

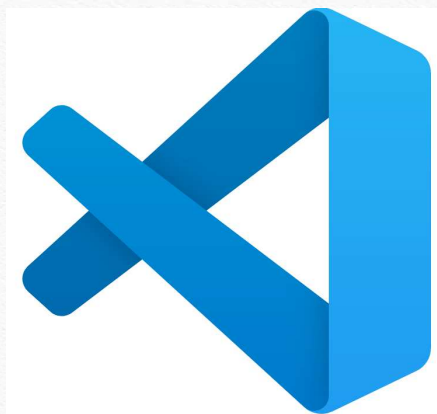


¿CÓMO SE PROGRAMA?

¿CÓMO SE PROGRAMA?

- ✓ Compilador C
- ✓ Arduino
- ✓ MicroPython

¿CÓMO SE PROGRAMA?

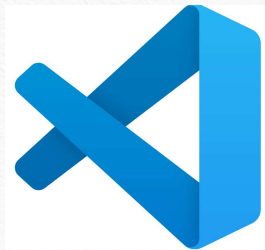


Visual Studio Code

[https://commons.wikimedia.org/wiki/File:Visual_Studio_Code_0.10.1_icon.png]

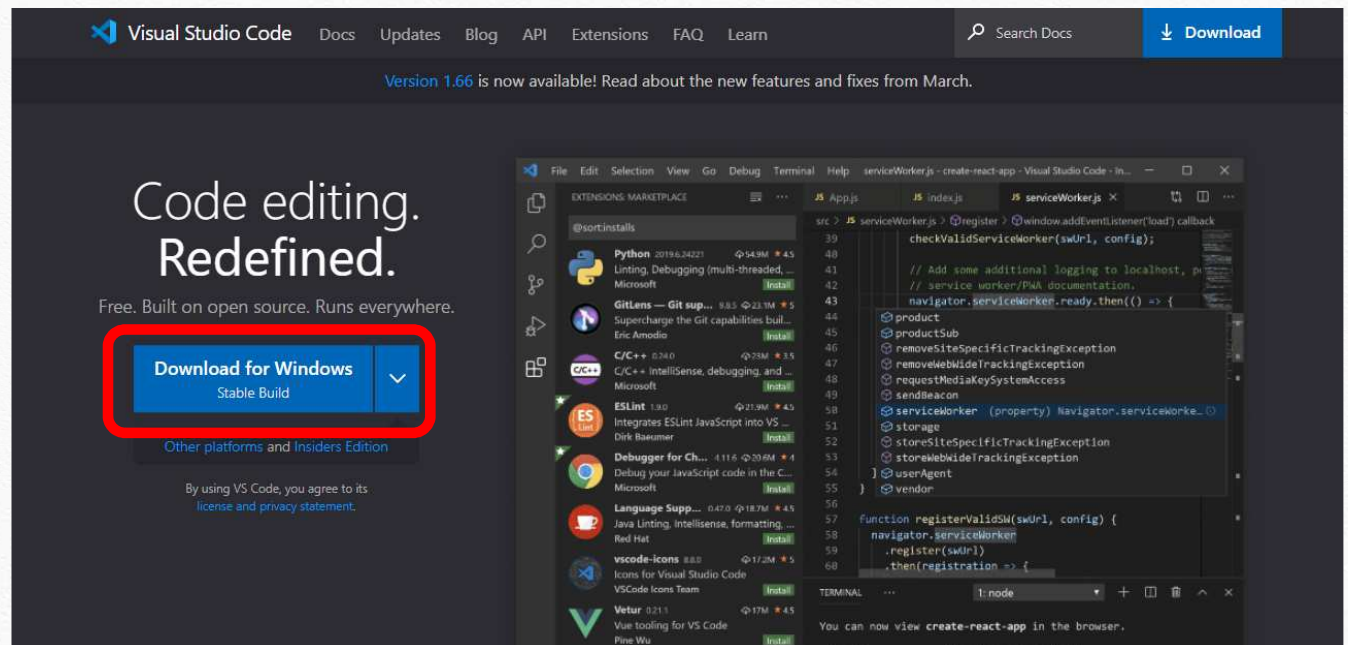


¿CÓMO SE PROGRAMA?



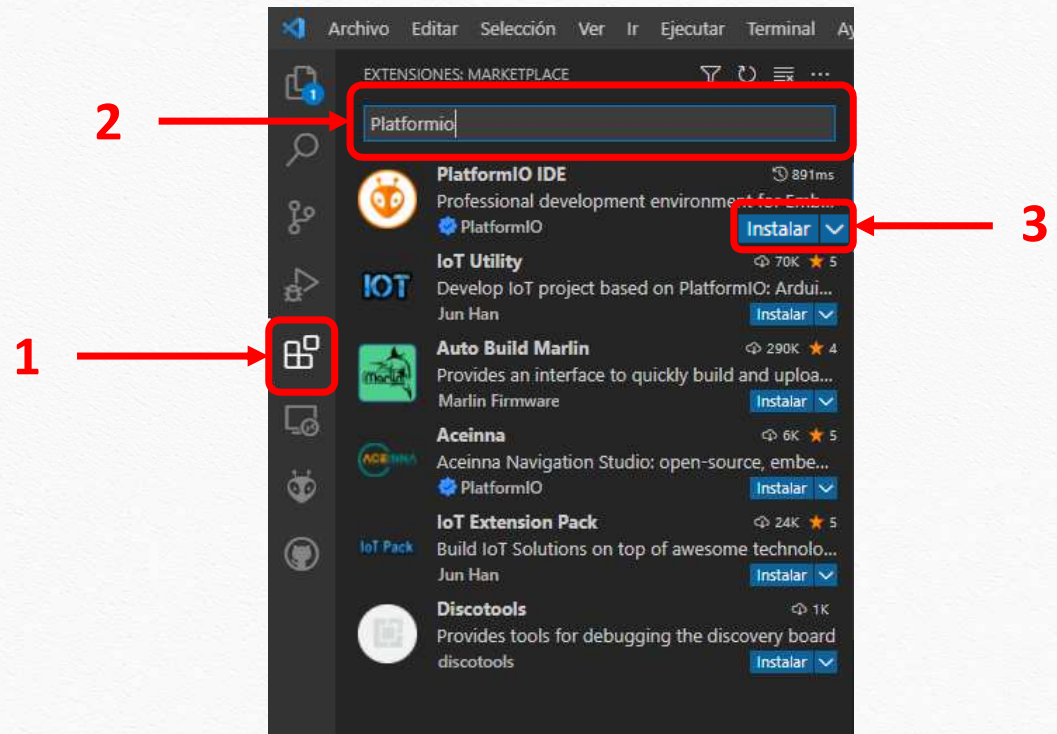
Visual Studio Code

[https://commons.wikimedia.org/wiki/File:Visual_Studio_Code_0.10.1_icon.png]



<https://code.visualstudio.com/>

¿CÓMO SE PROGRAMA?



SISTEMA OPERATIVO EN TIEMPO REAL

Es un sistema operativo ligero que se utiliza para facilitar la multitarea y la integración de tareas en sistemas que tienen recursos limitados, donde el manejo de tiempos es fundamental para la aplicación.

Tiempo real no significa velocidad, sino determinismo o previsibilidad en el tiempo

[<https://www.digikey.com/es/articles/real-time-operating-systems-and-their-applications>, revisado: 13/09/2022]

SISTEMA OPERATIVO EN TIEMPO REAL

Conceptos claves:

- Tareas
- Programador
- Comunicador entre tareas (ITC)
- TIC del sistema

[<https://www.digikey.com/es/articles/real-time-operating-systems-and-their-applications>, revisado: 13/09/2022]

SISTEMA OPERATIVO EN TIEMPO REAL

¿Por qué usar un sistema operativo en tiempo real?

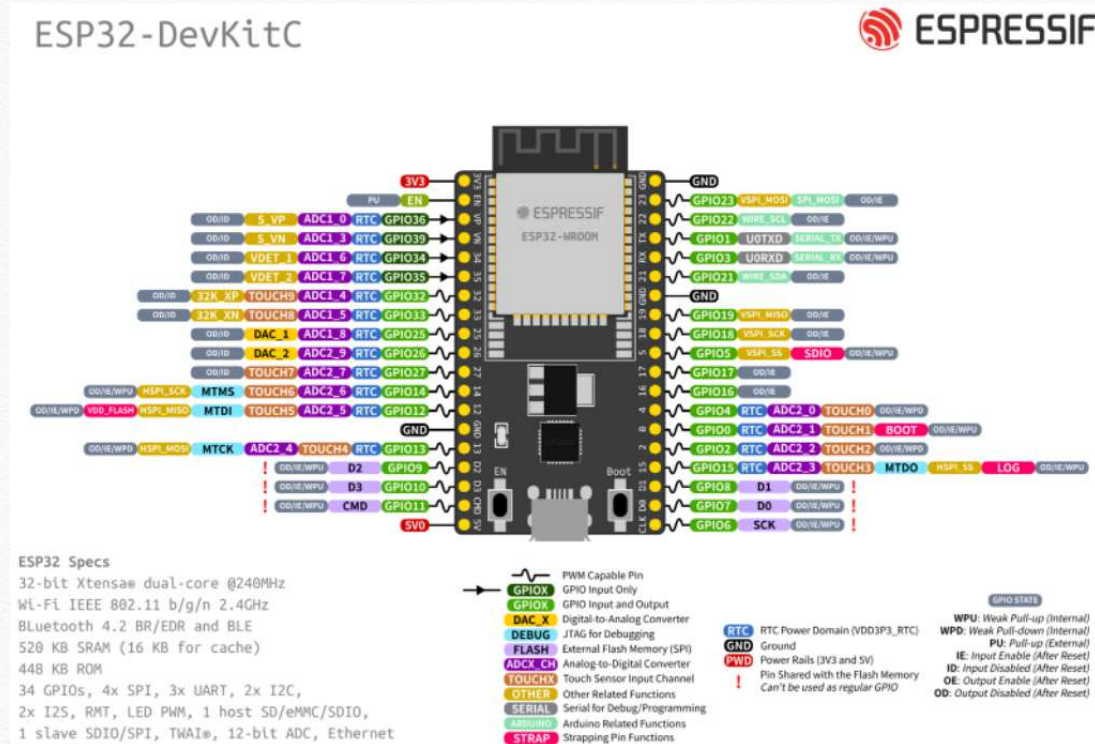
FreeRTOS www.freertos.org



GPIO

GPIO

Puerto de propósito general del microcontrolador el cual puede ser configurado como entrada o como salida



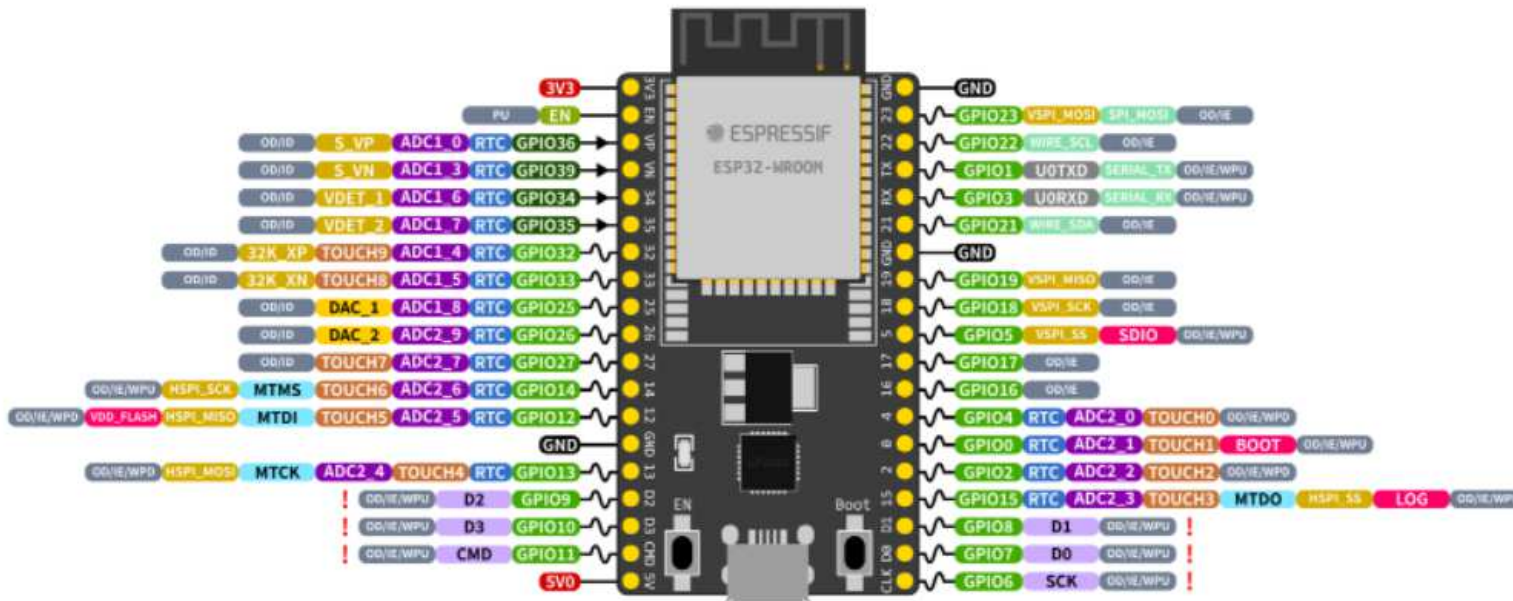
EJEMPLO 1

MANEJO DE PUERTOS - SALIDA

EJEMPLO 1

Programar en lenguaje C el microcontrolador ESP32 de tal manera que permita encender y apagar a una frecuencia de 1 Hz un led conectado al pin 2

ESP32-DevKitC



ESP32 Specs

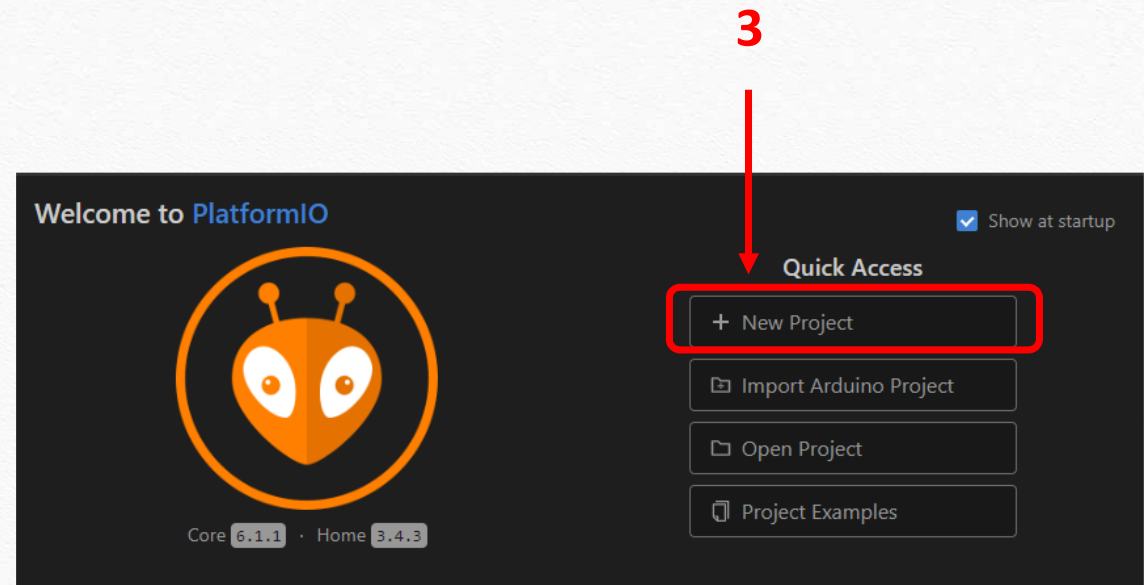
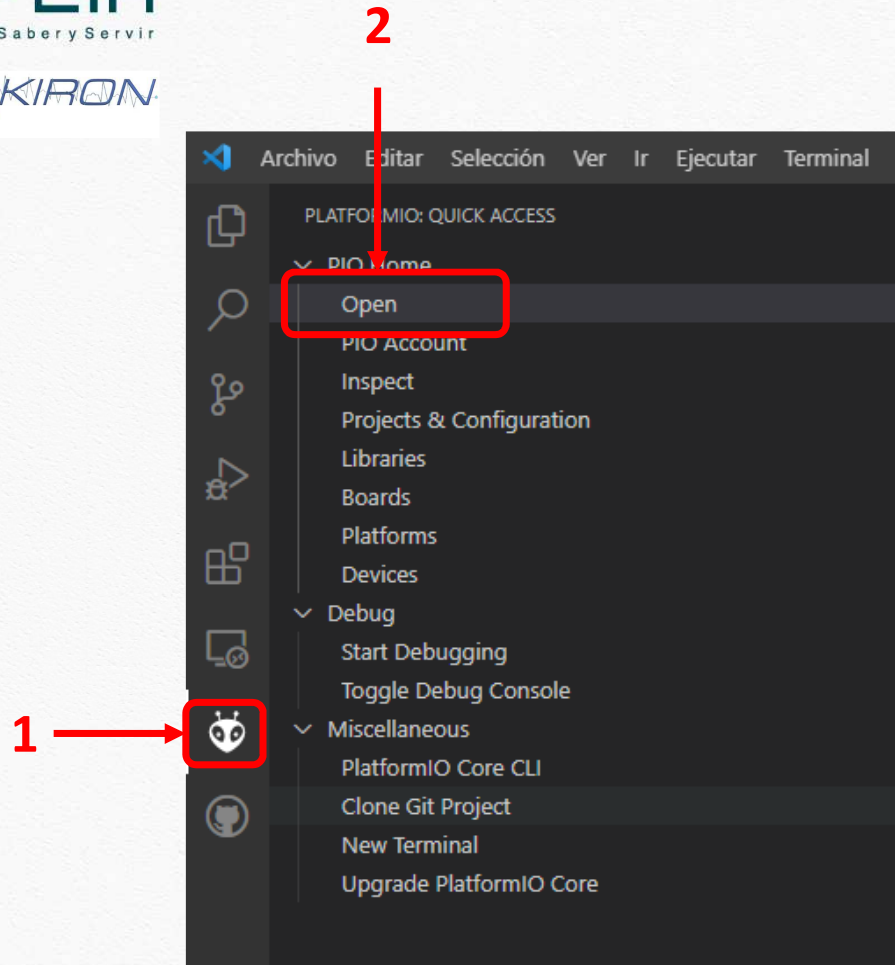
32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
Bluetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet



RTC: RTC Power Domain (VDD3P3_RTC)
GND: Ground
PWD: Power Rails (3V3 and 5V)
!: Pin Shared with the Flash Memory
Can't be used as regular GPIO

GPIO STATE

- WPU**: Weak Pull-up (Internal)
- WPD**: Weak Pull-down (Internal)
- PU**: Pull-up (External)
- IE**: Input Enable (After Reset)
- ID**: Input Disabled (After Reset)
- OE**: Output Enable (After Reset)
- OD**: Output Disabled (After Reset)



Project Wizard ✕

This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.

4 → **Name:** Ejemplo1_C

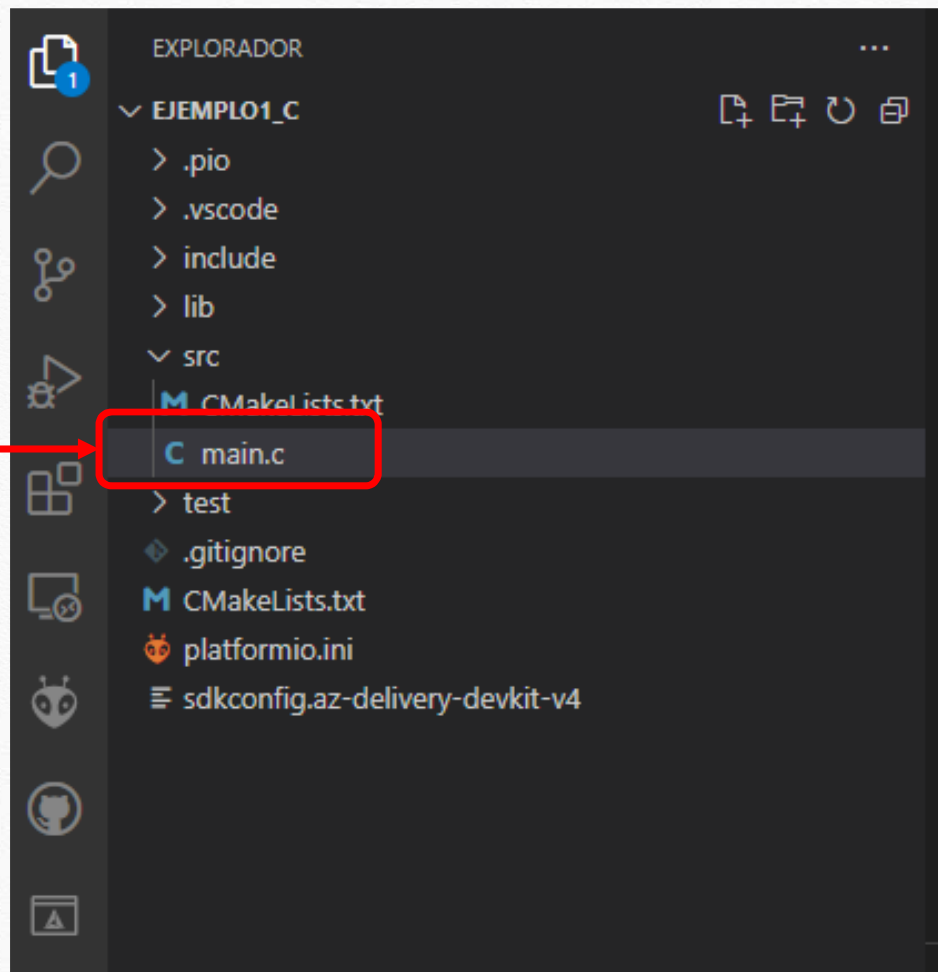
Board: AZ-Delivery ESP-32 Dev Kit C V4 5

Framework: Espressif IoT Development Framework 6

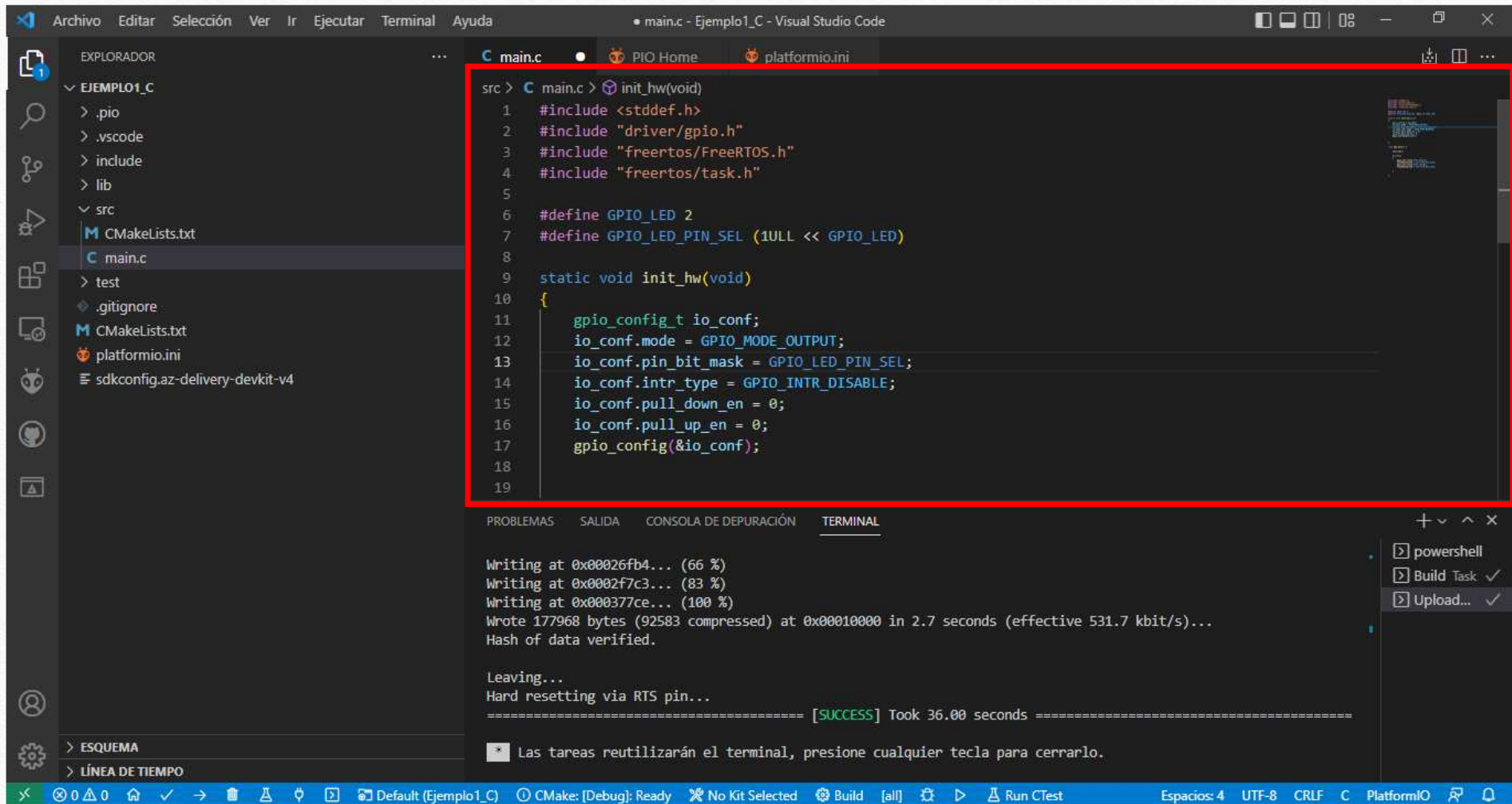
Location: ☒ Use default location ?

Cancel **Finish** 7

8



9. ESCRIBIR EL PROGRAMA



The screenshot shows the Visual Studio Code interface with the following components:

- EXPLORADOR (Explorer):** Shows the project structure for 'EJEMPLO1_C'. The 'src' folder is expanded, showing 'main.c' as the active file.
- main.c:** The code editor displays the following C code:

```
src > C main.c > init_hw(void)
1  #include <stddef.h>
2  #include "driver/gpio.h"
3  #include "freertos/FreeRTOS.h"
4  #include "freertos/task.h"
5
6  #define GPIO_LED 2
7  #define GPIO_LED_PIN_SEL (1ULL << GPIO_LED)
8
9  static void init_hw(void)
10 {
11     gpio_config_t io_conf;
12     io_conf.mode = GPIO_MODE_OUTPUT;
13     io_conf.pin_bit_mask = GPIO_LED_PIN_SEL;
14     io_conf.intr_type = GPIO_INTR_DISABLE;
15     io_conf.pull_down_en = 0;
16     io_conf.pull_up_en = 0;
17     gpio_config(&io_conf);
18
19
```
- TERMINAL:** Shows the output of the build and upload process:

```
Writing at 0x00026fb4... (66 %)
Writing at 0x0002f7c3... (83 %)
Writing at 0x000377ce... (100 %)
Wrote 177968 bytes (92583 compressed) at 0x00010000 in 2.7 seconds (effective 531.7 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
===== [SUCCESS] Took 36.00 seconds =====

Las tareas reutilizarán el terminal, presione cualquier tecla para cerrarlo.
```
- STATUS BAR:** Shows the current state: 'Default (Ejemplo1_C)', 'CMake: [Debug]: Ready', 'No Kit Selected', 'Build', 'Run CTest', 'Espacios: 4', 'UTF-8', 'CRLF', 'C', 'PlatformIO'.

```
1  #include <stddef.h>
2  #include "driver/gpio.h"
3  #include "freertos/FreeRTOS.h"
4  #include "freertos/task.h"
```

← **LIBRERÍAS**

```
6  #define GPIO_LED 2
7  #define GPIO_LED_PIN_SEL (1ULL << GPIO_LED)
```

← **IDENTIFICACIÓN PIN A USAR**

```
9  static void init_hw(void)
10 {
11     gpio_config_t io_conf;
12     io_conf.mode = GPIO_MODE_OUTPUT;
13     io_conf.pin_bit_mask = GPIO_LED_PIN_SEL;
14     io_conf.intr_type = GPIO_INTR_DISABLE;
15     io_conf.pull_down_en = 0;
16     io_conf.pull_up_en = 0;
17     gpio_config(&io_conf);
18 }
```

← **CONFIGURACIÓN PIN**


```
1  #include <stddef.h>
2  #include "driver/gpio.h"
3  #include "freertos/FreeRTOS.h"
4  #include "freertos/task.h"
```

```
6  #define GPIO_LED 2
7  #define GPIO_LED_PIN_SEL (1ULL << GPIO_LED)
```

```
9  static void init_hw(void)
10 {
11     gpio_config_t io_conf;
12     io_conf.mode = GPIO_MODE_OUTPUT;
13     io_conf.pin_bit_mask = GPIO_LED_PIN_SEL;
14     io_conf.intr_type = GPIO_INTR_DISABLE;
15     io_conf.pull_down_en = 0;
16     io_conf.pull_up_en = 0;
17     gpio_config(&io_conf);
18 }
```

PROGRAMA PRINCIPAL



```
20 void app_main() {
21
22     init_hw();
23
24     while(1)
25     {
26         gpio_set_level(GPIO_LED,1);
27         vTaskDelay(100 / portTICK_RATE_MS);
28         gpio_set_level(GPIO_LED,0);
29         vTaskDelay(100 / portTICK_RATE_MS);
30     }
31
32
33 }
```

COMPILAR Y PROGRAMAR



10

11. Conecte el sistema de desarrollo al puerto USB del PC



12

EJERCICIO 1

MANEJO DE PUERTOS - SALIDA

EJERCICIO 1

Programar en lenguaje C el microcontrolador ESP32 de tal manera que permita encender y apagar 4 leds formando una secuencia de luces.

EJERCICIO 2

MANEJO DE PUERTOS - SALIDA

EJERCICIO 2

Diseñe un contador de 0 a 99, muestre el resultado en dos displays 7 segmentos.

EJEMPLO 2

MANEJO DE PUERTOS – ENTRADAS y SALIDA DIGITALES

EJEMPLO 2

Programar en lenguaje C el microcontrolador ESP32 de tal manera que por medio de un pulsador conectado al pin 5 del microcontrolador se pueda controlar el encendido y apagado del led conectado al pin 2.


```
1  #include <stddef.h>
2  #include "driver/gpio.h"
3  #include "freertos/FreeRTOS.h"
4  #include "freertos/task.h"
```

← **LIBRERÍAS**

```
6  #define GPIO_LED 2
7  #define GPIO_LED_PIN_SEL (1ULL << GPIO_LED)
8  #define GPIO_PULSADOR 5
9  #define GPIO_PULSADOR_PIN_SEL (1ULL << GPIO_PULSADOR)
10 #define ESP_INTR_FLAG_DEFAULT 0
```

← **IDENTIFICACIÓN PIN A USAR**

```
12 static void pulsador_handler(void *arg);
```

```
14  static void init_hw(void)
15  {
16      gpio_config_t io_conf;
17      io_conf.mode = GPIO_MODE_OUTPUT;
18      io_conf.pin_bit_mask = GPIO_LED_PIN_SEL;
19      io_conf.intr_type = GPIO_INTR_DISABLE;
20      io_conf.pull_down_en = 0;
21      io_conf.pull_up_en = 0;
22      gpio_config(&io_conf);
23
24      io_conf.mode = GPIO_MODE_INPUT;
25      io_conf.pin_bit_mask = GPIO_PULSADOR_PIN_SEL;
26      io_conf.intr_type = GPIO_INTR_NEGEDGE;
27      io_conf.pull_up_en = 1;
28      gpio_config(&io_conf);
29
30      gpio_install_isr_service(ESP_INTR_FLAG_DEFAULT);
31      gpio_isr_handler_add(GPIO_PULSADOR, pulsador_handler, NULL);
32  }
```

← CONFIGURACIÓN DE
PINES


```
34 static TickType_t next = 0;
35 static bool led_state = false;
36
37 static void IRAM_ATTR pulsador_handler(void *arg)
38 {
39     TickType_t now = xTaskGetTickCountFromISR();
40     if (now > next)
41     {
42         led_state = ! led_state;
43         gpio_set_level(GPIO_LED, led_state);
44         next = now + 500 / portTICK_PERIOD_MS;
45     }
46 }
```

← **FUNCIÓN PULSADOR**

```
50 void app_main()
51 {
52     init_hw();
53
54     vTaskSuspend(NULL);
55 }
```

← **main**

```

1  #include <stddef.h>
2  #include "driver/gpio.h"
3  #include "freertos/FreeRTOS.h"
4  #include "freertos/task.h"

6  #define GPIO_LED 2
7  #define GPIO_LED_PIN_SEL (1ULL << GPIO_LED)
8  #define GPIO_PULSADOR 5
9  #define GPIO_PULSADOR_PIN_SEL (1ULL << GPIO_PULSADOR)
10 #define ESP_INTR_FLAG_DEFAULT 0

12 static void pulsador_handler(void *arg);

```

```

14 static void init_hw(void)
15 {
16     gpio_config_t io_conf;
17     io_conf.mode = GPIO_MODE_OUTPUT;
18     io_conf.pin_bit_mask = GPIO_LED_PIN_SEL;
19     io_conf.intr_type = GPIO_INTR_DISABLE;
20     io_conf.pull_down_en = 0;
21     io_conf.pull_up_en = 0;
22     gpio_config(&io_conf);
23
24     io_conf.mode = GPIO_MODE_INPUT;
25     io_conf.pin_bit_mask = GPIO_PULSADOR_PIN_SEL;
26     io_conf.intr_type = GPIO_INTR_NEGEDGE;
27     io_conf.pull_up_en = 1;
28     gpio_config(&io_conf);
29
30     gpio_install_isr_service(ESP_INTR_FLAG_DEFAULT);
31     gpio_isr_handler_add(GPIO_PULSADOR, pulsador_handler, NULL);
32 }

```

```

34 static TickType_t next = 0;
35 static bool led_state = false;
36
37 static void IRAM_ATTR pulsador_handler(void *arg)
38 {
39     TickType_t now = xTaskGetTickCountFromISR();
40     if (now > next)
41     {
42         led_state = ! led_state;
43         gpio_set_level(GPIO_LED, led_state);
44         next = now + 500 / portTICK_PERIOD_MS;
45     }
46 }

```

```

50 void app_main()
51 {
52     init_hw();
53
54     vTaskSuspend(NULL);
55 }

```


EJEMPLO

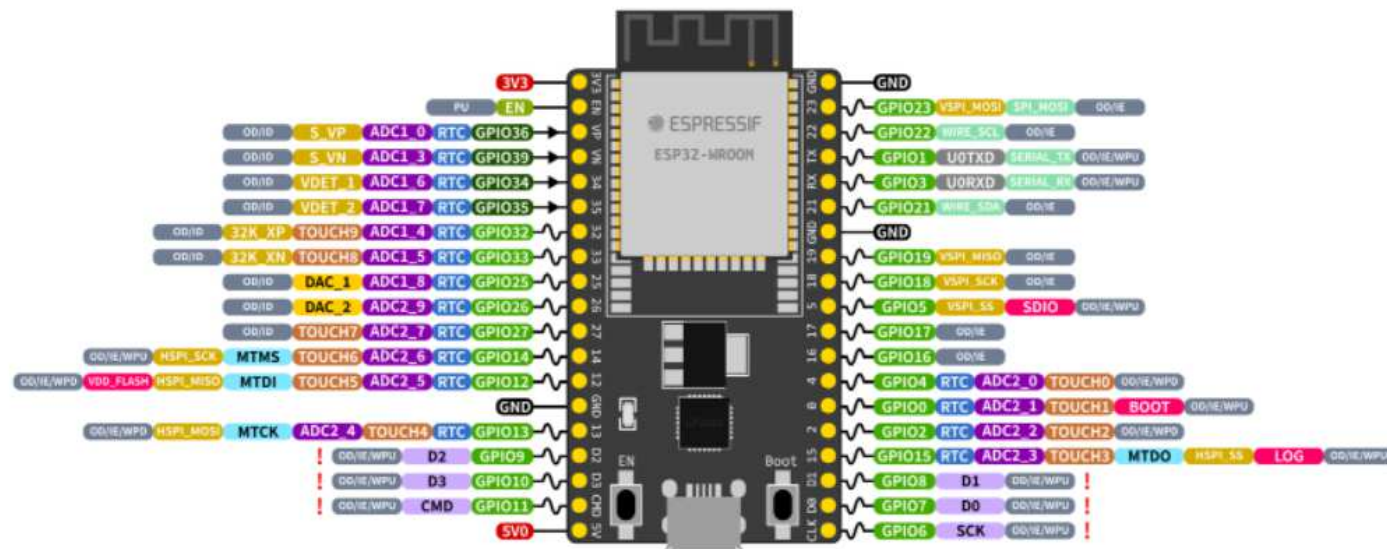
MANEJO DE PUERTOS – ENTRADAS y SALIDA DIGITALES

EJEMPLO 2

Se tiene dos pulsadores conectados a los pines 5 y 2 de un ESP32 y dos leds conectados a los pines 16 y 4 del mismo microcontrolador. Diseñar un programa en C de tal manera que cada pulsador controle el encendido y apagado de cada led.

EJEMPLO 2

ESP32-DevKitC



ESP32 Specs

32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
Bluetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet



RTC: RTC Power Domain (VDD3P3_RTC)
GND: Ground
PWD: Power Rails (3V3 and 5V)
!: Pin Shared with the Flash Memory Can't be used as regular GPIO

GPIO STATE
WPU: Weak Pull-up (Internal)
WPD: Weak Pull-down (Internal)
PU: Pull-up (External)
IE: Input Enable (After Reset)
ID: Input Disabled (After Reset)
OE: Output Enable (After Reset)
OD: Output Disabled (After Reset)

```
1  #include <stdint.h>
2  #include "driver/gpio.h"
3  #include "freertos/FreeRTOS.h"
4  #include "freertos/task.h"
```

← **LIBRERÍAS**

```
6  #define GPIO_LED1 16
7  #define GPIO_LED1_PIN_SEL (1ULL << GPIO_LED1)
8  #define GPIO_LED2 4
9  #define GPIO_LED2_PIN_SEL (1ULL << GPIO_LED2)
10 #define GPIO_MAS 5
11 #define GPIO_MAS_PIN_SEL (1ULL << GPIO_MAS)
12 #define GPIO_MENOS 2
13 #define GPIO_MENOS_PIN_SEL (1ULL << GPIO_MENOS)
14 #define ESP_INTR_FLAG_DEFAULT 0
```

← **IDENTIFICACIÓN
PINES A USAR**


```
static void init_hw(void)
{
    gpio_config_t io_conf;
    io_conf.mode = GPIO_MODE_OUTPUT;
    io_conf.pin_bit_mask = GPIO_LED1_PIN_SEL;
    io_conf.intr_type = GPIO_INTR_DISABLE;
    io_conf.pull_down_en = 0;
    io_conf.pull_up_en = 0;
    gpio_config(&io_conf);

    io_conf.mode = GPIO_MODE_OUTPUT;
    io_conf.pin_bit_mask = GPIO_LED2_PIN_SEL;
    io_conf.intr_type = GPIO_INTR_DISABLE;
    io_conf.pull_down_en = 0;
    io_conf.pull_up_en = 0;
    gpio_config(&io_conf);
}
```

**CONFIGURACIÓN DE
PUERTOS COMO
SALIDA**

**CONFIGURACIÓN DE
PUERTOS COMO
ENTRADAS**

```
io_conf.mode = GPIO_MODE_INPUT;
io_conf.pin_bit_mask = GPIO_MAS_PIN_SEL;
io_conf.intr_type = GPIO_INTR_DISABLE;
io_conf.pull_up_en = 1;
gpio_config(&io_conf);

io_conf.mode = GPIO_MODE_INPUT;
io_conf.pin_bit_mask = GPIO_MENOS_PIN_SEL;
io_conf.intr_type = GPIO_INTR_DISABLE;
io_conf.pull_up_en = 1;
gpio_config(&io_conf);
}
```

```
46 static bool x = 0;
47
48 void app_main() {
49     init_hw();
50
51     while (1)
52     {
53
54         x = gpio_get_level (GPIO_MAS);
55         gpio_set_level (GPIO_LED1, x);
56
57         x = gpio_get_level (GPIO_MENOS);
58         gpio_set_level (GPIO_LED2, x);
59
60
61
62         vTaskDelay (500 / portTICK_RATE_MS);
63     }
64
65 }
```

**PROGRAMA
PRINCIPAL**


```
1  #include <stddef.h>
2  #include "driver/gpio.h"
3  #include "freertos/FreeRTOS.h"
4  #include "freertos/task.h"
```

```
6  #define GPIO_LED1 16
7  #define GPIO_LED1_PIN_SEL (1ULL << GPIO_LED1)
8  #define GPIO_LED2 4
9  #define GPIO_LED2_PIN_SEL (1ULL << GPIO_LED2)
10 #define GPIO_MAS 5
11 #define GPIO_MAS_PIN_SEL (1ULL << GPIO_MAS)
12 #define GPIO_MENOS 2
13 #define GPIO_MENOS_PIN_SEL (1ULL << GPIO_MENOS)
14 #define ESP_INTR_FLAG_DEFAULT 0
```

```
static void init_hw(void)
{
    gpio_config_t io_conf;
    io_conf.mode = GPIO_MODE_OUTPUT;
    io_conf.pin_bit_mask = GPIO_LED1_PIN_SEL;
    io_conf.intr_type = GPIO_INTR_DISABLE;
    io_conf.pull_down_en = 0;
    io_conf.pull_up_en = 0;
    gpio_config(&io_conf);

    io_conf.mode = GPIO_MODE_OUTPUT;
    io_conf.pin_bit_mask = GPIO_LED2_PIN_SEL;
    io_conf.intr_type = GPIO_INTR_DISABLE;
    io_conf.pull_down_en = 0;
    io_conf.pull_up_en = 0;
    gpio_config(&io_conf);
}
```

```
io_conf.mode = GPIO_MODE_INPUT;
io_conf.pin_bit_mask = GPIO_MAS_PIN_SEL;
io_conf.intr_type = GPIO_INTR_DISABLE;
io_conf.pull_up_en = 1;
gpio_config(&io_conf);

io_conf.mode = GPIO_MODE_INPUT;
io_conf.pin_bit_mask = GPIO_MENOS_PIN_SEL;
io_conf.intr_type = GPIO_INTR_DISABLE;
io_conf.pull_up_en = 1;
gpio_config(&io_conf);
}
```

```
46 static bool x = 0;
47
48 void app_main() {
49     init_hw();
50
51     while (1)
52     {
53
54         x = gpio_get_level (GPIO_MAS);
55         gpio_set_level (GPIO_LED1, x);
56
57         x = gpio_get_level (GPIO_MENOS);
58         gpio_set_level (GPIO_LED2, x);
59
60
61         vTaskDelay (500 / portTICK_RATE_MS);
62     }
63 }
64
65 }
```

EJERCICIO

MANEJO DE PUERTOS – ENTRADAS y SALIDA DIGITALES

EJERCICIO

Diseñar un programa que permita controlar la secuencia de luces que se muestran en 4 leds por medio de un interruptor.

EJEMPLO

MANEJO DE ADC Y PWM

EJEMPLO

Usando un microcontrolador ESP32 diseñar un circuito de control de intensidad de iluminación para un led (GPIO18) por medio de un potenciómetro conectado al GPIO34.


```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "freertos/FreeRTOS.h"
4  #include "freertos/task.h"
5  #include "driver/ledc.h"
6  #include "driver/adc.h"

```

```

#define SAMPLE_CNT 32
static const adc1_channel_t adc_channel = ADC_CHANNEL_6;

```

```

#define LEDC_GPIO 18
static ledc_channel_config_t ledc_channel;

```

```

14 static void init_hw(void)
15 {
16     adc1_config_width(ADC_WIDTH_BIT_10);
17     adc1_config_channel_atten(adc_channel, ADC_ATTEN_DB_11);
18
19     ledc_timer_config_t ledc_timer = {
20         .duty_resolution = LEDC_TIMER_10_BIT,
21         .freq_hz = 1000,
22         .speed_mode = LEDC_HIGH_SPEED_MODE,
23         .timer_num = LEDC_TIMER_0,
24         .clk_cfg = LEDC_AUTO_CLK,
25     };

```

```

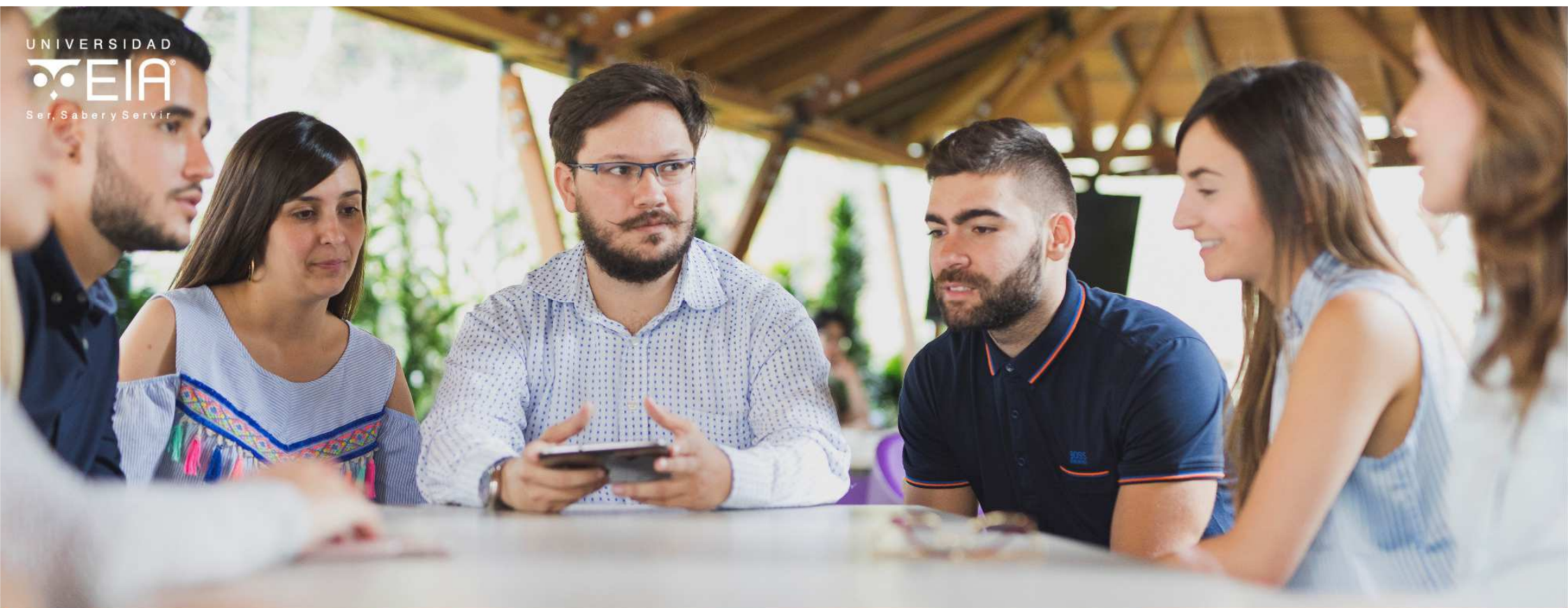
27     ledc_timer_config (&ledc_timer);
28
29     ledc_channel.channel = LEDC_CHANNEL_0;
30     ledc_channel.duty = 0;
31     ledc_channel.gpio_num = LEDC_GPIO;
32     ledc_channel.speed_mode = LEDC_HIGH_SPEED_MODE;
33     ledc_channel.hpoint = 0;
34     ledc_channel.timer_sel = LEDC_TIMER_0;
35     ledc_channel_config (&ledc_channel);
36
37 }

```

```

39 void app_main()
40 {
41     init_hw ();
42
43     while (1)
44     {
45         uint32_t adc_val = 0;
46         for (int i = 0; i < SAMPLE_CNT; ++i )
47         {
48             adc_val += adc1_get_raw(adc_channel);
49         }
50         adc_val /= SAMPLE_CNT;
51
52         ledc_set_duty(ledc_channel.speed_mode, ledc_channel.channel, adc_val);
53         ledc_update_duty(ledc_channel.speed_mode, ledc_channel.channel);
54         vTaskDelay(500 / portTICK_RATE_MS);
55     }
56
57
58 }

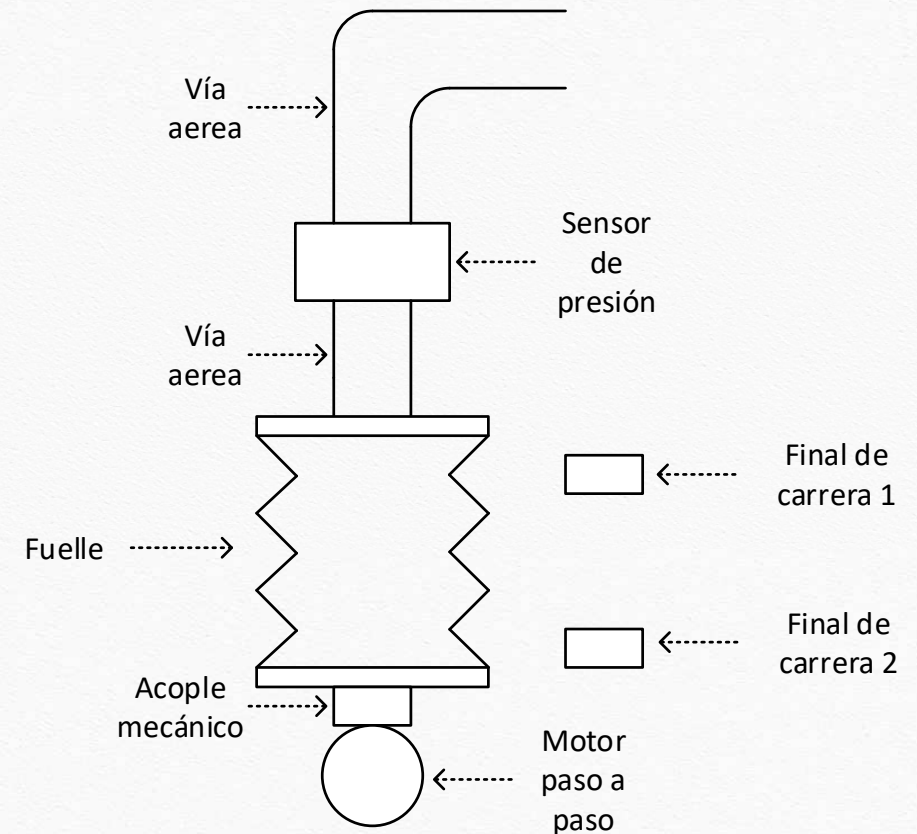
```



RETO DE DISEÑO

RETO DE DISEÑO

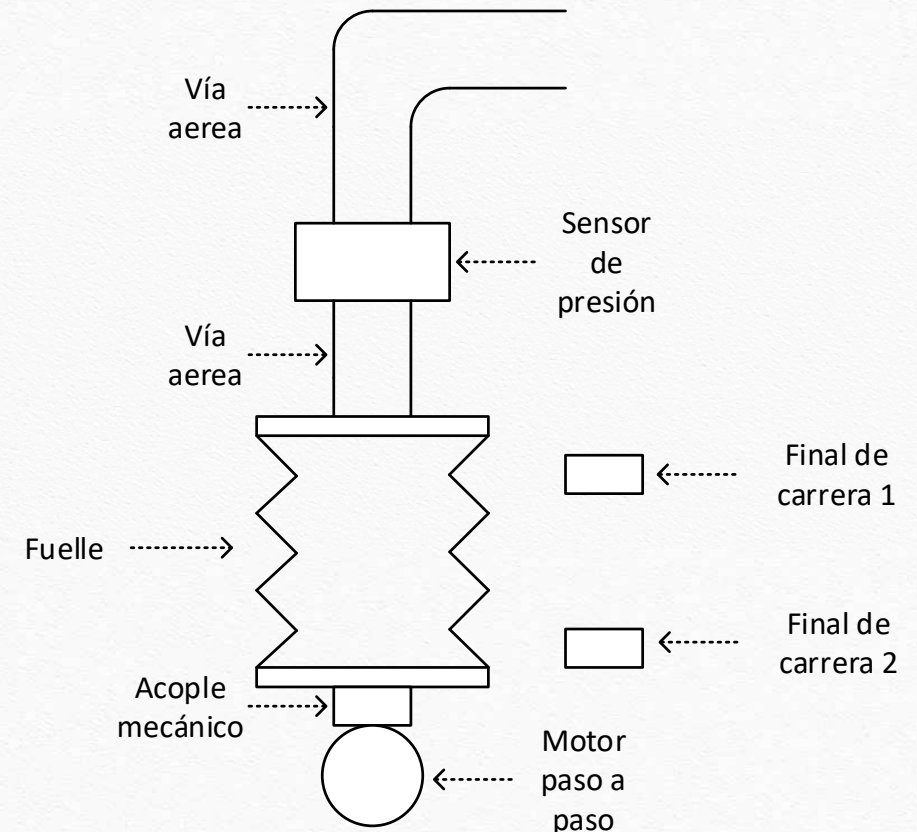
El semillero de investigación en Biomecatrónica, integrado por estudiantes de Ingeniería Biomédica e Ingeniería Mecatrónica, trabaja en el desarrollo de simuladores electromecánicos de procesos fisiológicos. En esta ocasión están desarrollando un simulador físico que representa la mecánica respiratoria. El diagrama de bloques del simulador que vienen desarrollando se muestra en la Fig. 1.



RETO DE DISEÑO

El simulador está formado por:

- Un fuelle que representa un pulmón.
- un motor paso a paso que representa el diafragma y que tiene como función expandir y contraer el fuelle con el objetivo de representar la respiración.
- Un acople mecánico que une el fuelle y el motor paso a paso y que permite transferir el movimiento del motor al fuelle.
- dos sensores finales de carrera tipo pulsador que se usan para indicar cuando el fuelle llegó a su máximo de compresión o expansión.

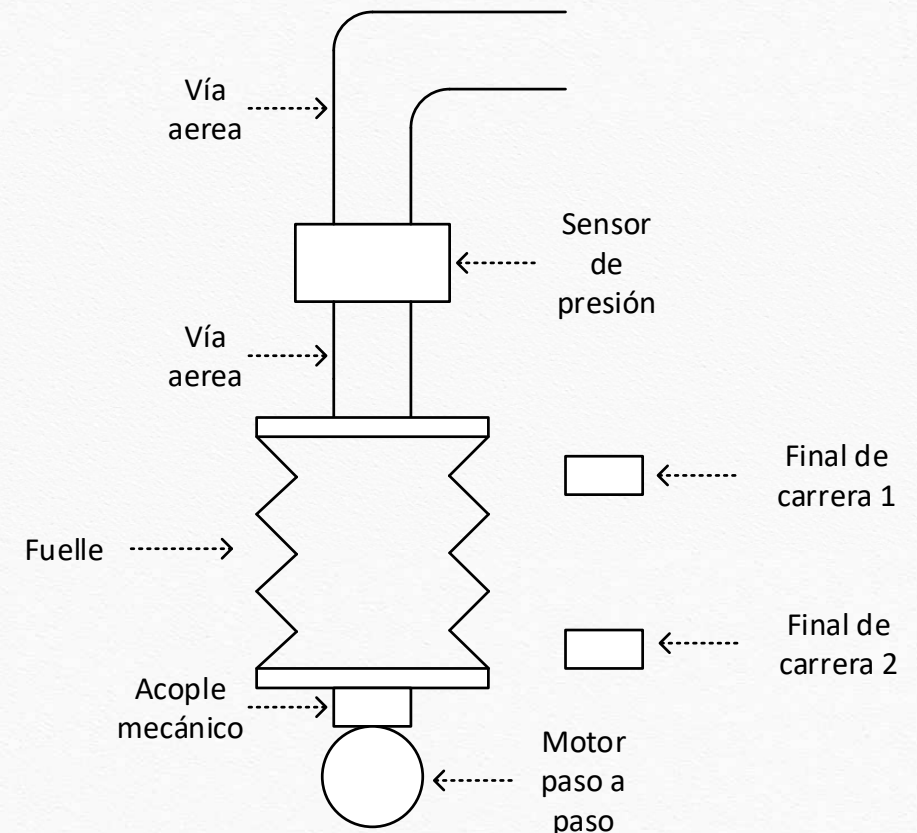


RETO DE DISEÑO

- Tubos que representan la vía aérea
- Un sensor de presión para medir la presión que genera el fuelle al expandirse o contraerse y que representa la respiración.

Además, el simulador cuenta con:

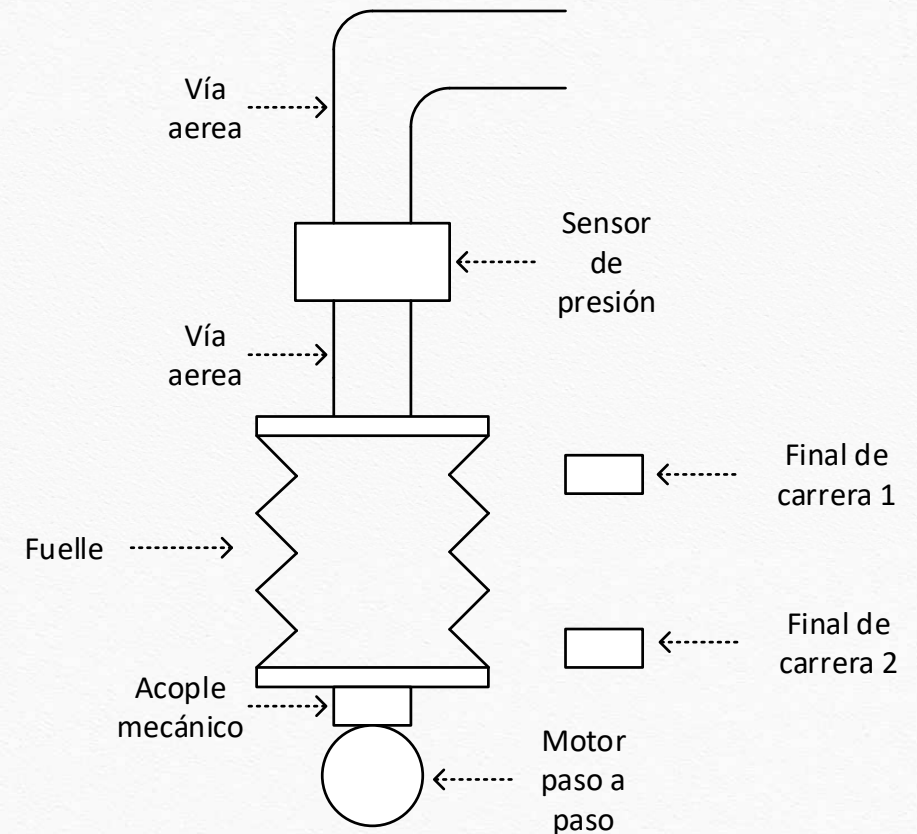
- Un potenciómetro de $10K\Omega$ que permite variar la velocidad del motor, con el objetivo de simular las variaciones de la frecuencia respiratoria
- Un interruptor de encendido y apagado.
- Un pulsador de inicio
- Un pulsador de pare.



RETO DE DISEÑO

El simulador funciona de la siguiente manera:

- El usuario enciende el simulador
- Con el potenciómetro el usuario selecciona la frecuencia respiratoria.
- El usuario presiona el pulsador de inicio y el simulador comienza a funcionar, esto es, el fuelle se expande y se contrae a la frecuencia determinada por el potenciómetro.
- El simulador se detiene cuando el usuario presiona el botón parar.
- Para continuar la simulación, el usuario puede presionar el pulsador inicio



RETO DE DISEÑO

Suponga que su trabajo en el semillero es diseñar un circuito basado en el microcontrolador ESP32 que permita controlar el movimiento del diafragma del simulador.

Para eso:

- 1) Diseñe el circuito
- 2) Diseñe el firmware del microcontrolador.

An aerial photograph showing five people (three men and two women) sitting around a long wooden table outdoors. They are working on papers, laptops, and other materials. The table is set on a stone-paved area, and the background is a dense, lush green forest with many ferns. The scene is brightly lit, suggesting daytime. The text 'TALLER' and 'Andrian Guerra – Yeison Montagut' is overlaid on the bottom right of the image.

TALLER

Andrian Guerra – Yeison Montagut

EJERCICIO 1

Por medio de un potenciómetro ($10\text{K}\Omega$) controlar la velocidad de un motor de corriente directa ($12\text{V} - 40\text{W}$)

EJERCICIO 2

Diseñar un circuito de control para un robot móvil. El control está formado por 5 pulsadores (adelante, atrás, derecha, izquierda y pare). Por medio de los pulsadores derecha e izquierda se controla la dirección del vehículo. Cuando se presiona el pulsador adelante, el vehículo se mueve hacia adelante, además si se deja presionado el pulsador adelante el robot incrementa su velocidad.

Cuando se presiona el pulsador atrás el robot se moverá para atrás, incrementando su velocidad si se deja presionado el pulsador.

El robot está formado por dos motores CD (12V – 10W) acopladas a sus ruedas traseras y una ruda loca en la parte delantera del robot.

EJERCICIO 3

Diseñar un sistema de control de iluminación.

El sistema está formado por una fotorresistencia como sensor de iluminación y un led de potencia (3.4V – 3W).

EJERCICIO 4

Diseñar un sistema de control de temperatura para ambientes.

El sistema está formado por un sensor LM35 como sensor, una resistencia CA (120V – 1500W) y un motor CD (12V – 60W) que funciona como ventilador.

El sistema funciona de la siguiente manera:

- Cuando la temperatura medida por el sensor es inferior a 23 °C el calefactor y el motor se encienden al tiempo, aumentando la temperatura.

(El aire que sale del ventilador pasa por la resistencia como lo hace un secador)

EJERCICIO 5

Diseñar un sistema de control de temperatura para ambientes.

El sistema está formado por un sensor LM35 como sensor, una resistencia CA (120V – 1500W) y un motor CD (12V – 60W) que funciona como ventilador.

El sistema funciona de la siguiente manera:

- Cuando la temperatura medida por el sensor es mayor a 23°C el calefactor y el motor están apagados.
- Cuando la temperatura medida por el sensor está entre 22°C y 23°C el calefactor se enciende a un 20% de su potencia.
- Cuando la temperatura medida por el sensor está entre 21°C y 22°C el calefactor se enciende a un 50% de su potencia.
- Cuando la temperatura medida por el sensor está entre 20°C y 21°C el calefactor se enciende a un 80% de su potencia.
- Cuando la temperatura medida por el sensor es menor que 20°C el calefactor se enciende a un 100% de su potencia.
- En los casos anteriores el motor se enciende a su máximo de potencia.

(El aire que sale del ventilador pasa por la resistencia como lo hace un secador)

EJERCICIO 6

Diseñar un sistema de control de temperatura para ambientes.

El sistema está formado por un sensor LM35 como sensor, una resistencia CA (120V – 1500W), un motor CD (12V – 60W) que funciona como ventilador, dos pulsadores (uno para aumentar y otro para disminuir) y dos displays 7 segmentos.

El sistema funciona de la siguiente manera:

- Por medio de los pulsadores el usuario fija la temperatura de control, la temperatura de control se muestra en los dos displays.
- Cuando la temperatura medida por el sensor es mayor que la temperatura de control la resistencia y el motor están apagados.
- Cuando la temperatura medida por el sensor es inferior a la temperatura de control la resistencia y el motor se encienden a su máxima potencia.

(El aire que sale del ventilador pasa por la resistencia como lo hace un secador)



UNIVERSIDAD



Ser, Saber y Servir

GRACIAS

VIGILADA MINEDUCACIÓN