

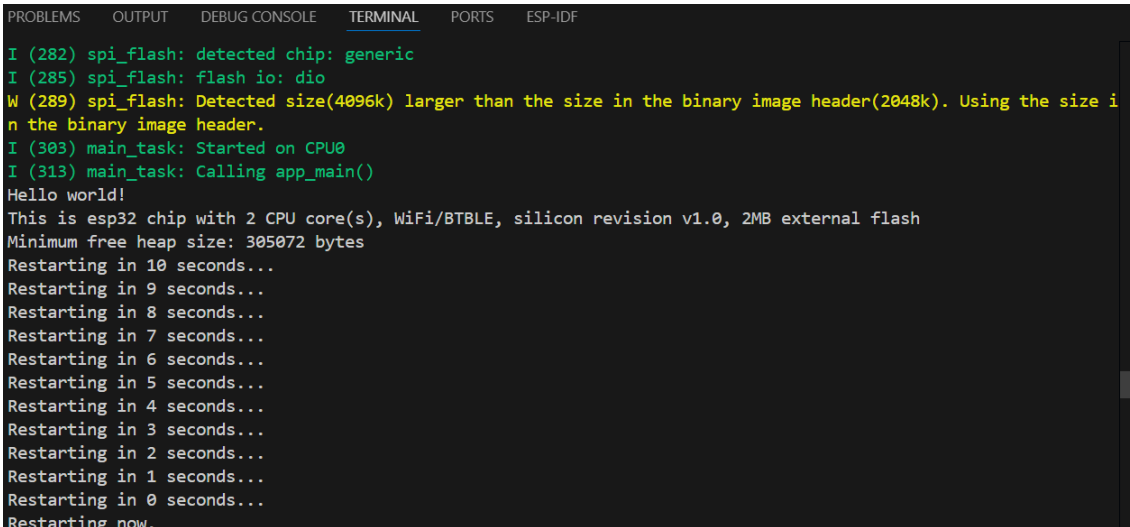
Actividad 1: Solución

Instala ESP-IDF en tu PC y crea un nuevo proyecto basado en el ejemplo *hello_world*. Puedes hacerlo a través del entorno VSCode y seguir el tutorial paso a paso de la presentación. Carga y ejecuta el programa *Hello_world* en tu ESP32. Monitoriza la salida serial y observa la información que aparece.

1. A partir del código principal del proyecto *hello_world* (*hello_world_main.c*), analiza las funcionalidades programadas y relaciónalas con el resultado observado a través de la monitorización.

Solución:

Al monitorizar la salida serial de la ESP32 se observa lo siguiente:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ESP-IDF
I (282) spi_flash: detected chip: generic
I (285) spi_flash: flash io: dio
W (289) spi_flash: Detected size(4096k) larger than the size in the binary image header(2048k). Using the size in the binary image header.
I (303) main_task: Started on CPU0
I (313) main_task: Calling app_main()
Hello world!
This is esp32 chip with 2 CPU core(s), WiFi/BT/BLE, silicon revision v1.0, 2MB external flash
Minimum free heap size: 305072 bytes
Restarting in 10 seconds...
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
Restarting in 6 seconds...
Restarting in 5 seconds...
Restarting in 4 seconds...
Restarting in 3 seconds...
Restarting in 2 seconds...
Restarting in 1 seconds...
Restarting in 0 seconds...
Restarting now.
```

En primer lugar, se observan una serie de mensajes de información (I) y de warning (W) comunes al lanzar la aplicación en la ESP32. Estos mensajes aparecen de forma iterativa porque en el propio código forzamos el reinicio de la ESP32 al final de nuestro código (lo veremos más adelante).

Por tanto, la funcionalidad realmente programada en nuestro código es el texto que aparece en blanco. Este código muestra:

- Un mensaje “Hello world!”.
- Información sobre la ESP32 relativa al número de cores, características de conexión (wifi y bluetooth BT/BLE), versión del chip, propiedades de la memoria flash y memoria heap disponible.
- Cuenta atrás de 10 segundos que concluye con el reseteo de la ESP32.

La secuencia anterior se repite de manera iterativa.

Vamos a analizar el código y ver cómo se ha desarrollado el programa:

```
/*
 * SPDX-FileCopyrightText: 2010-2022 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: CC0-1.0
 */

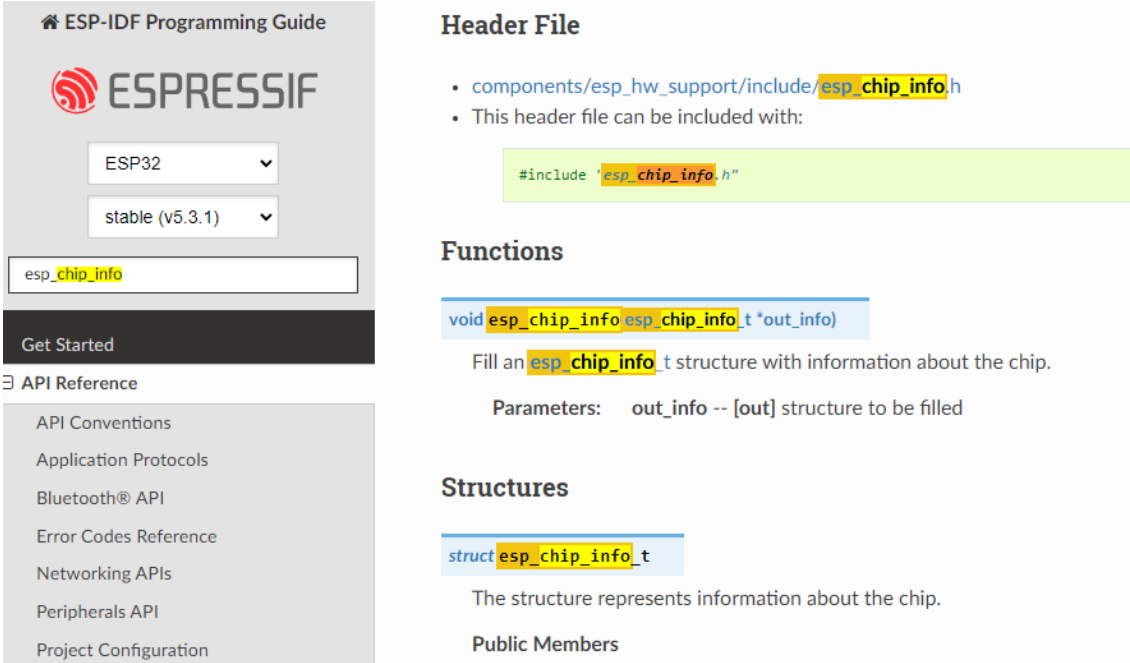
#include <stdio.h>
#include <inttypes.h>
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_chip_info.h"
#include "esp_flash.h"
#include "esp_system.h"
```

En primer lugar, se incluyen los archivos de cabecera (*header*) con las funcionalidades que se van a emplear en el código. Cabe destacar algunos archivos propios del SO en tiempo real (*freertos*) que se abordará de forma específica en próximos ejercicios, o archivos para interactuar con funcionalidades del sistema de ESP32 (*esp_chip_info.h*, *esp_flash.h*, *esp_system.h*).

Es posible consultar la definición de funciones, macros y tipos de datos que incluye cada fichero de ESP-IDF. Para eso, basta con poner el cursor en el nombre del fichero a consultar, y pulsar Alt+F12. En el siguiente ejemplo se muestra el contenido de *esp_chip_info.h*.

```
12 #include "esp_chip_info.h"
esp_chip_info.h C:\Users\laptop\esp\v5.3.1\esp-idf\components\esp_hw_support\include - Definitions (1)
1 /*
2  * SPDX-FileCopyrightText: 2021-2024 Espressif Systems (Shanghai) CO LTD
3  *
4  * SPDX-License-Identifier: Apache-2.0
5  */
6
7 #pragma once
8
9 #include <stdbool.h>
10 #include <stdint.h>
11 #include "sdkconfig.h"
12 #include "esp_bit_defs.h"
13
14 #ifndef __cplusplus
15 extern "C" {
```

Otra opción para consultar el contenido de un header es a través de la página web de espressif (<https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32/index.html>) e indicar en el buscador el contenido a consultar:



The screenshot shows the ESP-IDF Programming Guide API Reference for the `esp_chip_info` module. On the left, there is a navigation menu with options like 'ESP32', 'stable (v5.3.1)', and 'esp_chip_info'. The main content area is divided into sections: 'Header File', 'Functions', and 'Structures'. Under 'Header File', it lists the include path `components/esp_hw_support/include/esp_chip_info.h` and shows the preprocessor directive `#include "esp_chip_info.h"`. Under 'Functions', it shows the function signature `void esp_chip_info(esp_chip_info_t *out_info)` and explains that it fills an `esp_chip_info_t` structure. Under 'Structures', it shows the `struct esp_chip_info_t` and notes that it represents information about the chip.

Las siguientes líneas de código crean la función principal `app_main`:

```
void app_main(void)
{
    printf("Hello world!\n");
}
```

En este caso, nuestro programa está conformado por una única “tarea” (task) asociada al `app_main`. Lo primero que se hace en esta tarea es imprimir por pantalla el mensaje “Hello world!” tal y como se observaba en la monitorización de la salida de la ESP32.

```
/* Print chip information */
esp_chip_info_t chip_info;
uint32_t flash_size;
esp_chip_info(&chip_info);
printf("This is %s chip with %d CPU core(s), %s%s%s%s, ",
    CONFIG_IDF_TARGET,
    chip_info.cores,
    (chip_info.features & CHIP_FEATURE_WIFI_BGN) ? "WiFi/" : "",
    (chip_info.features & CHIP_FEATURE_BT) ? "BT" : "",
    (chip_info.features & CHIP_FEATURE_BLE) ? "BLE" : "",
    (chip_info.features & CHIP_FEATURE_IEEE802154) ? ", 802.15.4 (Zigbee/Thread)" : "");
```

A continuación, se crean dos variables:

- `chip_info`: variable tipo `esp_chip_info_t`. Se trata de una estructura con información sobre el chip como el número de núcleos, conexiones, entre otros.
- `flash_size`: variable tipo `uint32`.

A través de la función `esp_chip_info()` se completa la estructura `chip_info` con la información sobre el ESP32 que se está usando. Observa que el argumento de entrada de la función es un puntero a la variable.

El siguiente paso permite imprimir la información sobre el chip con el comando `printf`. En concreto, se obtiene la información del dispositivo usado (`CONFIG_IDF_TARGET`),

el número de cores (`chip_info.cores`) y se *comprueba* si tiene las siguientes funcionalidades (`chip_info.features`):

- Wifi 2.4GHz (`CHIP_FEATURE_WIFI_BGN`).
- Bluetooth clásico (`CHIP_FEATURE_BT`).
- Bluetooth LE (`CHIP_FEATURE_BLE`).
- IEEE 802.15.4 (`CHIP_FEATURE_IEEE802154`).

```
unsigned major_rev = chip_info.revision / 100;
unsigned minor_rev = chip_info.revision % 100;
printf("silicon revision v%d.%d, ", major_rev, minor_rev);
if(esp_flash_get_size(NULL, &flash_size) != ESP_OK) {
    printf("Get flash size failed");
    return;
}

printf("%" PRIu32 "MB %s flash\n", flash_size / (uint32_t)(1024 * 1024),
       (chip_info.features & CHIP_FEATURE_EMB_FLASH) ? "embedded" : "external");

printf("Minimum free heap size: %" PRIu32 " bytes\n", esp_get_minimum_free_heap_size());
```

En las siguientes líneas se calcula la versión del chip, adaptando el formato de `chip_info.revision` e imprimiendo la información por pantalla con el primer `printf`. A continuación se obtiene el tamaño de la memoria flash (`esp_flash_get_size()`). Si esta función se realiza con éxito, devuelve `ESP_OK`. En caso contrario, se imprime la sentencia “Get flash size failed”. Si la comprobación se ha llevado con éxito, se imprime por pantalla el tamaño de la memoria flash en MB y se especifica si es external o embedded.

Con el siguiente `printf` se muestra por pantalla el tamaño mínimo de memoria heap disponible mediante `esp_get_minimum_free_heap_size()`.

```
for (int i = 10; i >= 0; i--) {
    printf("Restarting in %d seconds...\n", i);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
}
printf("Restarting now.\n");
fflush(stdout);
esp_restart();
```

Por último, se implementa la cuenta atrás. Para ello se desarrolla un bucle *for* en el que el índice *i* va desde 10 hasta 0 en cada iteración. En primer lugar, se imprime el valor de *i*. A continuación, se emplea el comando `vTaskDelay`, clave en el SO FreeRTOS. Esta función se emplea para “retrasar” o “dormir” una tarea. Anteriormente, comentábamos que nuestro programa estaba conformado por una única tarea (`app_main`). Cada vez que se ejecuta esta sentencia en el bucle *for* la tarea queda “congelada” durante 1 segundo (`1000/portTICK_PERIOD_MS`), de forma que cada iteración se realiza cada segundo.

Una vez se completa el bucle, se imprime por pantalla el reseteo de la ESP32. A continuación se vacía cualquier contenido pendiente en el buffer (flush), y se reinicia el controlador (`esp_restart`). De esta forma, se comienza a ejecutar de nuevo el código como si el dispositivo volviera a iniciarse.