SDK IDF ESP32
Jerarquia de carpetas y “hello world”

El Espressif Internet Development Framework (ESP-IDF) utiliza FreeRTOS para hacer un mejor uso de los dos procesadores de alta velocidad y administrar los numerosos periféricos incorporados. Se hace mediante la creación de tareas. Veamos el “hola mundo” , que se ve un poco diferente de los que usted pudo haber visto.
Este mundo hola imprime la cadena en UART (eventualmente en el terminal de computadora). Primero veremos la estructura de ESP-IDF. A continuación, mire el ejemplo de hola mundo y donde se mostrará el parpadeo de un LED mientras sigue imprimiendo continuamente la cadena 'hola mundo'.

```
1.
2.  +--esp-idf
3.          |
4.          |
5.          + - - components
6.          |
7.          + - - docs
8.          |
9.          + - - examples
.0.         |
.1.         + - - make
.2.         |
.3.         + - - tools
```

Directorios del folder “esp-idf”.

El directorio de componentes contiene todo el código 'C' para el ESP32. Contiene todos los "componentes" que conforman el ESP32. Incluye Drivers para numerosos periféricos, el gestor de arranque, bt (bluetooth), freeRTOS, etc.

De esta jerarquía de directorios podemos resaltar el folder “examples” donde encontraremos muchas plantillas por donde poder empezar nuestro código.

Includes.-

Al observar cada una de las carpetas de componente uno o más archivos .c y .h. Para incluir un componente debemos incluir el archivo .h correspondiente en nuestro código. Nuestro ejemplo hola mundo necesita 4 de estos componentes.

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

- **Freertos / FreeRTOS.h**: La inclusión de esta configuración de conjuntos es requerida para ejecutar freeRTOS en el ESP32.

- **Freertos / task.h**: Las tareas como se puede adivinar proporcionan la funcionalidad multitarea, que vamos a explorar en el ejemplo de blinky con "***hola mundo***" en algún momento.

- **Esp_system.h**: Esta inclusión configura los periféricos en el sistema ESP. Piense en ello como inicialización del sistema. Es como configurar todos los componentes de su bicicleta, antes de poder disparar el motor!

La funcion main().-

Por ahora, supongamos que la ejecución del programa comienza con ***app_main()***, igual que en la programación clásica llamábamos tradicionalmente al ***main()*** de nuestro C de toda a vida orientado a los pic, avr, etc. Esta es la primera función que se llama automáticamente.

```
void app_main()
{
    nvs_flash_init();
    xTaskCreate(&hello_task, "hello_task", 2048, NULL, 5, NULL);
}
```

funcion inicial, análoga al main() de toda la vida del C.

- **Inicializar Flash: nvs_flash_init()**: Como se dijo anteriormente los módulos ESP32 ejecutar código de un flash externo. A menos que esté utilizando directamente el chip ESP32, el fabricante del módulo / tarjeta tomará la inicialización y acceso al flash.

- **XTaskCreate():** Esta función tiene 5 argumentos. Olvídate de los NULL por ahora. El primer parámetro es la dirección de la función que se ejecuta, cuando esta tarea está programada para ejecutarse! Vuelva a leer la oración una vez. El segundo parámetro hello_task es el nombre dado a la tarea. El podría ser cualquier nombre. Se utiliza para obtener handle o mientras se depura la tarea. El siguiente parámetro 2048 es la memoria asignada a la tarea en palabra (2 Bytes).

Hello_task().-

La función que se llama desde la tarea creada anteriormente es una función simple como se muestra a continuación. Simplemente imprime la cadena al UART. El flujo de impresión está configurado en el UART0 del ESP32.

```
void hello_task(void *pvParameter)
{
    printf("Hello world!\n");
    for (int i = 10; i >= 0; i--) {
        printf("Restarting in %d seconds...\n", i);
        vTaskDelay(1000 / portTICK_RATE_MS);
    }
    printf("Restarting now.\n");
    fflush(stdout);
    esp_restart();
}
```

Ejecución de una tarea en un hilo

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

void hello_task(void *pvParameter)
{
    printf("Hello world!\n");
    for (int i = 10; i >= 0; i--) {
        printf("Restarting in %d seconds...\n", i);
        vTaskDelay(1000 / portTICK_RATE_MS);
    }
    printf("Restarting now.\n");
    fflush(stdout);
    esp_restart();
}

void app_main()
{
    nvs_flash_init();
    xTaskCreate(&hello_task, "hello_task", 2048, NULL, 5, NULL);
}
```

Código completo

Hagamos una multitarea: Blink with Hello.-

Deberías haber visto el código blinky numerosas veces donde-en; Encendemos un LED, esperamos un segundo, lo apagamos y esperamos otro segundo. ¿Qué pasa si quiero enviar una cadena continuamente, digamos cada 100 milisegundos mientras el LED sigue parpadeando?

Es fácil lograr eso usando el RTOS. Así que el código de abajo, crear dos tareas. Por ahora, basta con suponer que estos son dos bucles infinitos (while). Cada vez que se entra en el estado de retardo (función vTaskDelay), el otro se ejecuta y viceversa. Puesto que somos lentos y ESP32 es rápido, cambia entre las tareas numerosas veces y vemos que ambas tareas suceden simultáneamente.

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

#define BLINK_GPIO 13


void hello_task(void *pvParameter)
{

        while(1)
        {
            printf("Hello world!\n");
            vTaskDelay(100 / portTICK_RATE_MS);
        }
}
```

```c
void blinky(void *pvParameter)
{

    gpio_pad_select_gpio(BLINK_GPIO);
    /* Set the GPIO as a push/pull output */
    gpio_set_direction(BLINK_GPIO, GPIO_MODE_OUTPUT);
    while(1) {
        /* Blink off (output low) */
        gpio_set_level(BLINK_GPIO, 0);
        vTaskDelay(1000 / portTICK_RATE_MS);
        /* Blink on (output high) */
        gpio_set_level(BLINK_GPIO, 1);
        vTaskDelay(1000 / portTICK_RATE_MS);
    }
}


void app_main()
{
    nvs_flash_init();
    xTaskCreate(&hello_task, "hello_task", 2048, NULL, 5, NULL);
    xTaskCreate(&blinky, "blinky", 512,NULL,5,NULL );
}
```

Multi Task en el ESP32