

# Laboratorio 1

## *Introducción a la placa de desarrollo del curso y a VSCode + ESP-IDF plugin*

### Objetivo

Interiorizarse con el hardware de la placa de desarrollo del curso, el ESP32, y el entorno de desarrollo ESP-IDF en VSCode. Comenzar a escribir código en lenguaje de programación C para aplicaciones embebidas. Desarrollar funciones básicas de E/S. Emplear las herramientas de compilación y depuración que ofrece el entorno de desarrollo.

### Evaluación y metodología de trabajo

A partir de este laboratorio, y para todos los laboratorios restantes y el proyecto final, se trabajará en grupos (siempre el mismo). El docente asistirá a los estudiantes constantemente realizando sugerencias, devoluciones y aclarando conceptos cada vez que se necesite. Se evaluará el trabajo de cada grupo durante el laboratorio, y se considerarán para dicha evaluación los siguientes criterios:


- A. Organización del grupo, involucramiento de cada uno de los integrantes.
- B. Completitud y correctitud de las soluciones a las tareas indicadas en el laboratorio en el tiempo de clase.
- C. Respuestas a las preguntas planteadas en el laboratorio y/o preguntas que los docentes puedan realizar.
- D. Utilización de buenas prácticas de programación.
  - a. El código entregado **debe** compilar sin errores ni warnings. En caso de que por algún motivo excepcional sea necesario ignorar un warning, explicar porqué en el informe.
  - b. En el código no deben haber funciones, variables, comentarios, etc. que no se utilicen. Por ejemplo, si al crear un archivo hay funciones de ejemplo, estas se deben quitar antes de entregar el código.
- E. Entrega en fecha de las soluciones propuestas.

# Ejercicios


## Primera Parte

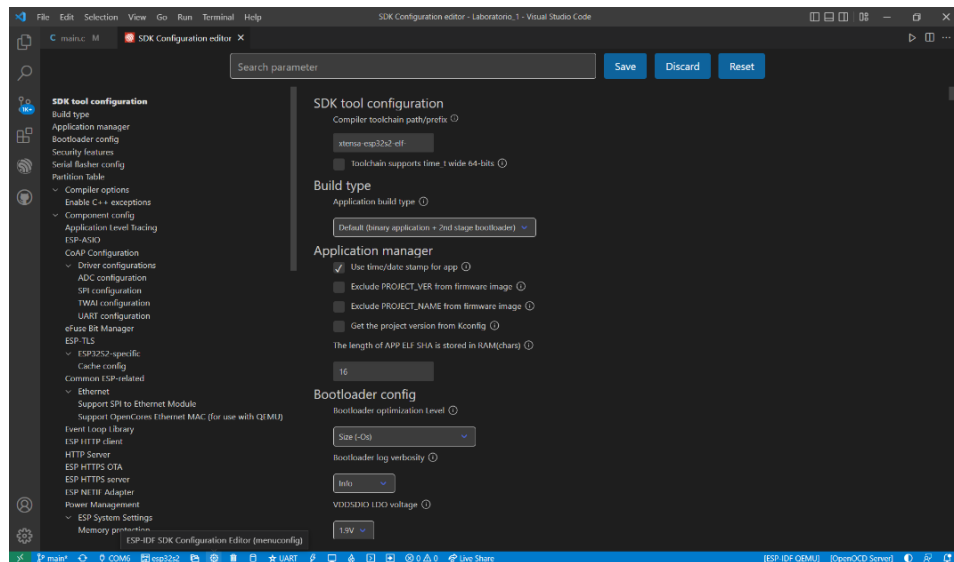
Introducción al ESP-IDF + VSCode, creación de proyectos y primeros pasos.

### 1. Creación de un proyecto en VSCode + Espressif IDE (shortcut CTRL + E N)

- a. Abrir el VSCode y crear un nuevo proyecto vacío (*View -> Command palette -> ESP-IDF: New Project*) y seleccionar *sample\_project* para el ESP32-S2-KALUGA-1. Darle un nombre al proyecto, por ejemplo "Laboratorio" y guardarlo en el directorio que deseen.
- b. Analizar el árbol de directorios creado en el navegador del proyecto (vista "Explorer", Workspace del VSCode).
- c. Analizar los archivos generados: ¿qué secciones tiene?
- d. Compilar el proyecto tal cual está (click  en la barra inferior del VSCode). ¿Qué cosas cambian en el workspace del laboratorio? ¿Qué información nos brinda el compilador?

### 2. Parámetros de Configuración (shortcut CTRL + E G)

- a. Los dispositivos ESP32 cuentan con múltiples **Parámetros de configuración** conocido como **sdkconfig**. Entre estos se pueden habilitar (o no) diversas configuraciones del ESP32, por ejemplo el nivel de log, priorizar la performance o el tamaño final de la compilación, habilitar algunas librerías, entre otras configuraciones.  
Estos **Parámetros de configuración** se deben configurar en VSCode o mediante el comando *idf.py menuconfig* en la terminal de ESP-IDF (sobre el proyecto a trabajar).
- b. Los **parámetros de configuración** y su valor son **específicos del proyecto** y pueden visualizarse en el documento **sdkconfig** (si se guardan nuevos parámetros los valores predeterminados están bajo el nombre **sdkconfig.old**) ubicado en el directorio del proyecto.  
Ver el archivo **sdkconfig** y analizar los distintos parámetros configurables.
- c. Afortunadamente, **Espressif IDF** nos permite modificar los parámetros de configuración a través de una interfaz gráfica con el botón  (abajo a la izquierda en VSCode):



Luego de seteados los parámetros, dar click en **SAVE** para guardar los cambios o **DISCARD** si no se desea realizar cambios. En caso de hacer click en **RESET** se restablecen los valores predeterminados en **sdkconfig.old**.

- d. Investigar qué opciones se pueden modificar en el **sdkconfig** y ver como cambia el archivo al guardar. Comparar diferencias con **sdkconfig.old**.

### 3. Compilación (shortcut CTRL + E B)

- a. Agregar al archivo **main.c** la siguiente definición y las siguientes variables:

```
#define ARRAY_SIZE 12
```

```
int exampleData;
char exampleArray[ARRAY_SIZE];
```

- b. Compilar el proyecto creado (en caso de errores corregirlos) y ver los archivos generados en la carpeta del proyecto **build**. Buscar y Analizar particularmente los archivos **flasher\_args.json** y **project\_description.json**.  
¿Qué puede comentar acerca de estos archivos?
- c. Buscar las variables antes declaradas en el archivo **.map** dentro de la carpeta **build**.  
¿Qué tamaño ocupan en la memoria y en qué direcciones están alojadas?
- d. Sustituir **ARRAY\_SIZE** por 10 y ver qué ocurre con **exampleArray** en la memoria.

## Segunda Parte

En esta parte del laboratorio se realizará el “Hola Mundo” de los Sistemas Embebidos: Encender un LED.

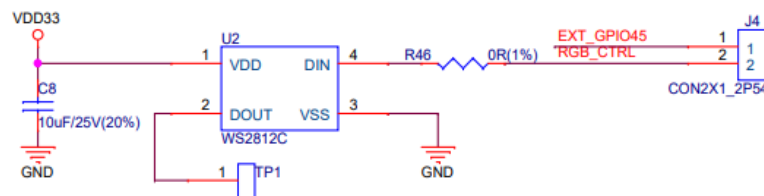
Luego se realizará el “parpadeo” de ambos LEDs de la placa, implementando una función de Delay (bloqueante).

### 1. Creación de librería de manejo de LED RGB del hardware

- Para comenzar a trabajar sobre el hardware disponible debemos conocer cómo está conectado este y como funciona cada parte en sí que la conforma.

#### RGB LED:

1. Put J4 jumper on if wanna control RGB.



En la imagen anterior podemos observar que es necesario conectar un jumper en el conector J4 (ubicar en la placa dicho jumper) y que al hacer dicha conexión conecta el pin EXT\_GPIO45 del ESP32 con RGB\_CTRL, entrada de datos del LED RGB (WS2812C).

- Busque la hoja de datos del LED RGB y comente cómo funciona el componente. Plantee (en pseudocódigo) cómo controlaría dicho componente.
- Para simplificar el uso del LED RGB se les proveerá una librería que utiliza un canal RMT del ESP32-S2. Para agregar dicha librería a su proyecto deberá crear una carpeta llamada **components** y colocar dentro de esta los archivos provistos.

### 2. Creación de una Librería nueva

- Dado que cuenta ahora cuenta con una librería que maneja tiras led, sería conveniente que cree una librería específica para el manejo del led embebido en el kit de desarrollo.
- Para ello, dentro de la carpeta **components** (creada previamente) crear una nueva carpeta con un nombre adecuado para la nueva librería.
- Dentro de esta nueva carpeta crear dos archivos: `<nombre_libreria>.c` y `CMakeLists.txt`, en el primero irán todas las funciones que manejan el LED y deberá incluir el cabezal (header, aún no creado). El archivo de CMakeLists solamente deberá incluir únicamente la siguiente directiva: `idf_component_register(SRCS "<nombre_libreria>.c" INCLUDE_DIRS "include" PRIV_REQUIRES "led_strip")`.

- d. Luego cree otra carpeta llamada **include**, y dentro de esta crear el archivo `<nombre_libreria>.h`, este archivo será el cabezal en el cual se definen las funciones, estructuras y/o tipos de datos utilizables por otros archivos. Este cabezal (header) es el que será incluido en el programa principal **main.c** para manejar el LED embebido.

### 3. Creación de una función de delay

- a. El ESP32-S2 cuenta con un delay pero el mismo toma valores en us (microsegundos). En esta oportunidad deberá crear una librería nueva que tome valores en ms (milisegundos) o en s (segundos) y adapte dicho valor a la función nativa de delay ( **`esp_rom_delay_us(uint32_t number_of_us);`** ).
- b. Para crear una librería nueva acceda a la paleta de comandos (CTRL + SHIFT + P) del VSCode y seleccione la opción de crear un “componente nuevo”(ESP-IDF: Create new ESP-IDF component) dentro de las funciones del ESP-IDF. Asígnele un nombre adecuado a la librería.
- c. Observe que en la carpeta **components** ahora hay una nueva carpeta con el nombre asignado previamente. Ubique dentro de la carpeta creada los archivos **.h** y **.c** de la librería. Modifique estos archivos para crear delays en ms y en s, y que puedan ser utilizados en otros archivos.

### 4. Implementación del parpadeo de LEDs

- a. Utilizando las librerías recién creadas, modificar el **main.c** para que en la placa quede el LED RGB parpadeando indefinidamente cambiando de color.