Universidad Católica del Uruguay

Laboratorio 1

6 de Mayo de 2025

Diseño de IoT y Sistemas Embebidos

Integrantes del equipo:

Gonzalo Soto

Jerónimo Ventos

Santiago El Ters

Contenido

Primera Parte

1 - Creación de proyecto en VSCode + Espressif IDE

a) Proyecto de ejemplo: hello_world

Consigna:
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.

Se implementa un proyecto básico "hello_world" utilizando el ESP32-S2-Kaluga-1. El objetivo es mostrar el mensaje "Hello, World!" en la consola, lo que permite verificar que el sistema de compilación y el entorno de programación están correctamente configurados. Este ejercicio inicial es ideal para familiarizarse con la placa y comprobar la correcta instalación de las herramientas necesarias.

b) Análisis del árbol de directorios creado en el navegador del proyecto

Consigna:
Analizar el árbol de directorios creado en el navegador del proyecto (vista “Explorer”, Workspace del VSCode).

Cuando se crea un nuevo proyecto con ESP-IDF-Kaluga-1, se puede observar que ya viene con ejemplos cargados, lo cual permite probar distintas cosas desde el inicio. Hay ejemplos básicos como hello_world y blink para verificar que todo funciona bien, pero también hay un una variedad de proyectos más avanzados, entre ellos, cómo trabajar con Bluetooth, BLE, WiFi, Zigbee, y hasta BLE Mesh. También hay pruebas de rendimiento (throughput), seguridad, conexiones GATT, y coexistencia entre WiFi y Bluetooth.
Por otro lado, hay secciones para manejar sensores, pantallas, motores, almacenamiento, actualizaciones OTA, entre otros.

c) Análisis de los archivos generados

Consigna:
Analizar los archivos generados: ¿qué secciones tiene?

El proyecto tiene una estructura organizada: en la carpeta main está el código fuente (hello_world_main.c) junto con los archivos de configuración CMakeLists.txt necesarios para compilar e incluye un script de prueba (pytest_hello_world.py). Además, hay carpetas como .vscode y .devcontainer para configurar el entorno de desarrollo, un README.md con documentación del proyecto, y un archivo sdkconfig.ci para la configuración del proyecto. Todo esto indica que el proyecto fue creado y preparado correctamente para desarrollo con ESP-IDF.

d) Análisis de cambios en el directorio al compilar y datos en terminal

Consigna:
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?

Al compilar genera la carpeta build en el directorio del proyecto, esta carpeta contiene todas las librerías instaladas, y el archivo sdkconfig que tiene una configuración de la placa ESP-IDF.

Dentro de la carpeta build hay subdirectorios como bootloader y partition_table que guardan el arranque y la tabla de particiones. También hay carpetas como config y esp-idf que guardan configuraciones y dependencias de la placa.
Los archivos como build.ninja, .ninja_log y CMakeCache.txt son generados por el sistema de construcción (CMake y Ninja) y controlan el proceso de compilación. Otros archivos (flash_args, compile_commands.json, etc.) ayudan en la carga al dispositivo, depuración, entre otros.

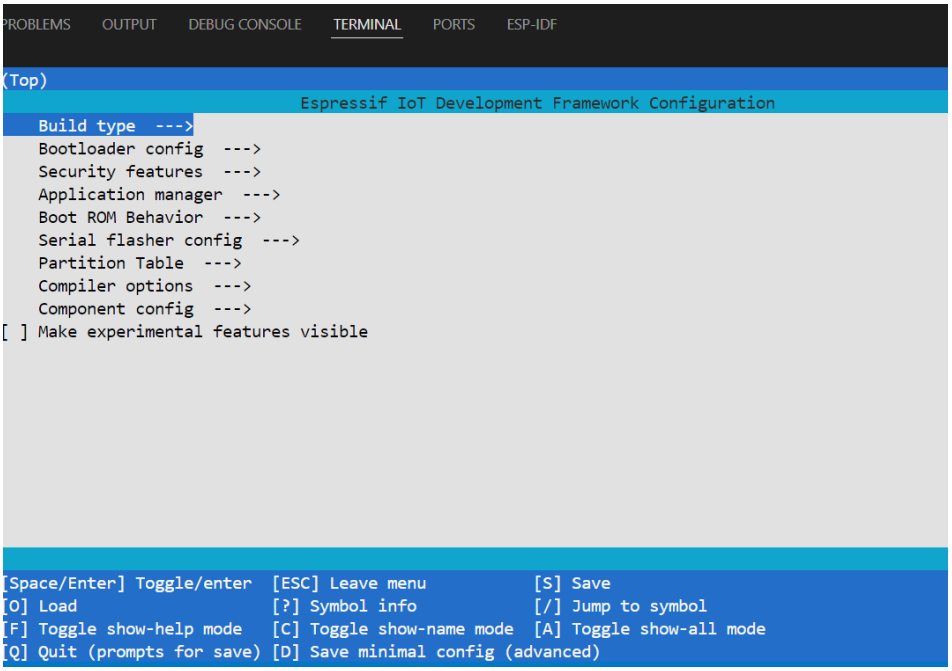
Por otro lado, la salida en la terminal nos muestra que el código se ejecutó correctamente en el ESP32-S2, indicando que la función app_main() fue llamada. También brinda información útil del chip, como que tiene 1 núcleo, soporte para WiFi, una revisión de silicio v0.0 y una memoria flash externa de 2MB. Además, nos muestra que el sistema tiene 250912 bytes de memoria heap libre. Finalmente, se ve un conteo regresivo que termina en un reinicio del sistema.

2 - Parámetros de configuración

a) idf.py menuconfig

Consigna:
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).

El ESP32 permite ajustar múltiples parámetros a través del archivo sdkconfig, donde se pueden activar o desactivar opciones como el nivel de logs, optimizar el rendimiento o el tamaño de la compilación. Estas configuraciones se pueden modificar en VSCode o usando idf.py menuconfig en la terminal como a continuación.



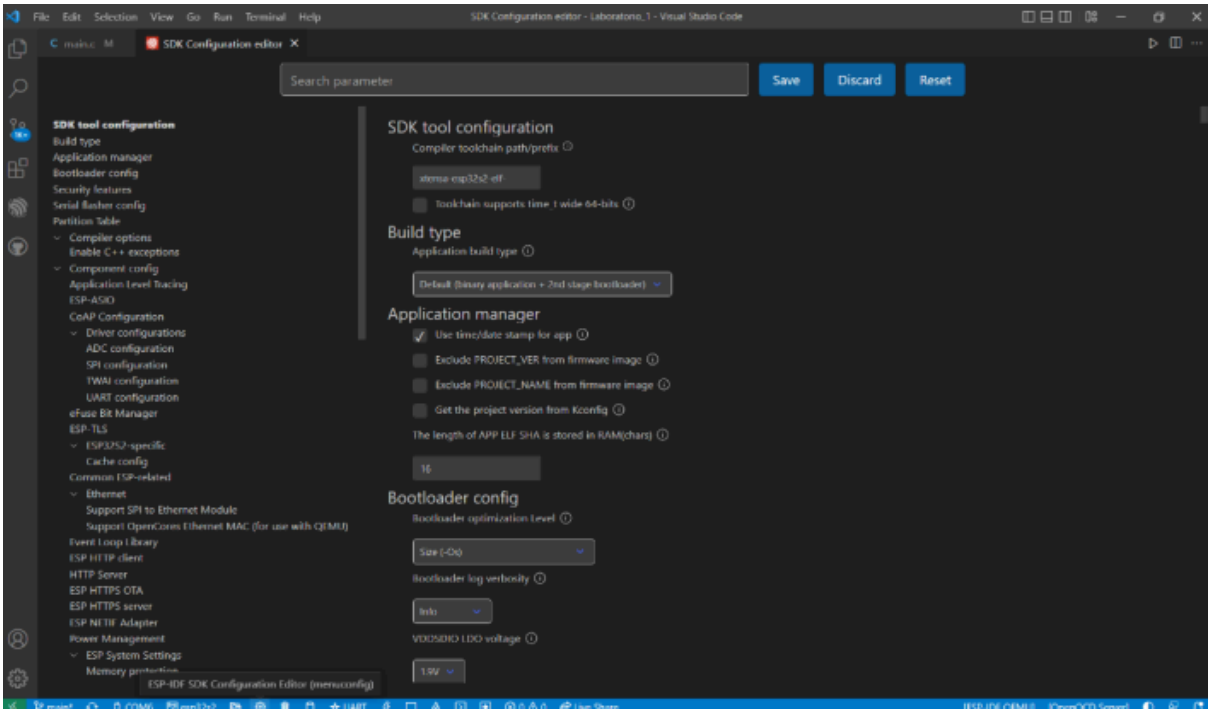
b) Análisis del archivo sdkconfig

Consigna:
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.

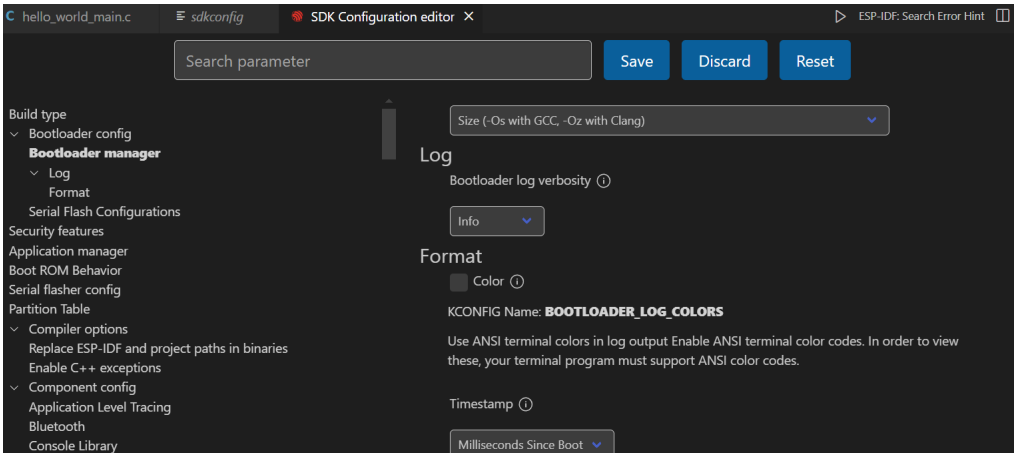
Al analizar el archivo sdkconfig generado en el proyecto, se pueden identificar varios grupos de parámetros interesantes. Primero, están las variables SOC (System-on-Chip), que definen qué módulos de hardware están disponibles en el ESP32-S2, como ADC, Wi-Fi, UART, USB, entre otros. Después, el grupo de FreeRTOS que nos permite configurar aspectos del sistema operativo que usa el chip, como la frecuencia de tics. También encontramos parámetros relacionados al sistema general (ESP_SYSTEM), donde se ajusta el comportamiento ante errores, manejo de interrupciones y control de memoria. En el caso de Wi-Fi, aunque no se use en el "Hello World", ya se habilitan configuraciones internas del stack para poder usarlo después si se necesita. Otro grupo muy importante es el de drivers de periféricos, donde se activan o desactivan módulos como UART, SPI, ADC, según lo que se vaya a utilizar. Finalmente, los parámetros de Build/Compilación definen cómo se optimiza el código, ya sea buscando velocidad, tamaño más chico o facilidad para depurar errores.

c) Interfaz de configuración

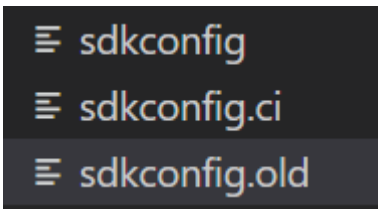
Consigna:
Afortunadamente, Espressif IDF nos permite modificar los parámetros de configuración a través de una interfaz gráfica con el botón (abajo a la izquierda en VSCode):



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.



Guardamos y generamos el archivo sdkconfig.old

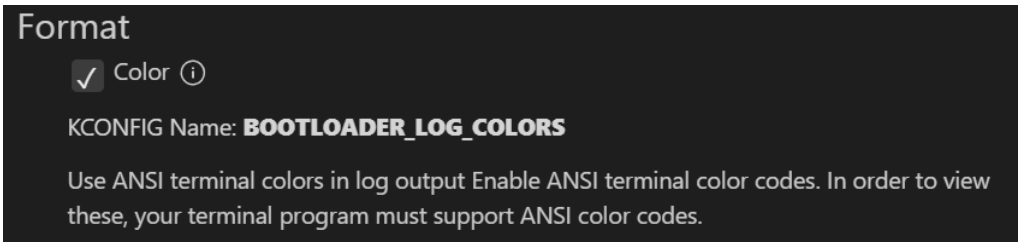


d) Análisis de cambios sdkconfig vs sdkconfig.old

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

Dentro del SDK Configuration Editor de ESP-IDF, se modificaron varias variables de configuración para observar los cambios generados entre el archivo sdkconfig.old y el nuevo sdkconfig una vez guardada la configuración.

1. Modificación en el formato de la terminal para logs del bootloader, en este caso el color.



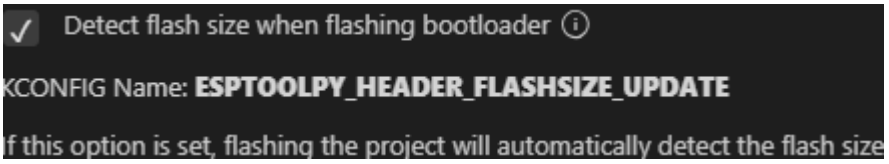
   nuevo: CONFIG_IDF_INIT_VERSION="5.4.1"
   viejo: CONFIG_IDF_INIT_VERSION="$IDF_INIT_VERSION"

   Se actualizó el valor de la versión de inicialización de IDF a una versión específica ("5.4.1") en lugar de usar una variable genérica. Esto asegura que el proyecto quede registrado exactamente con la versión utilizada al momento de la configuración.

   nuevo:
   # CONFIG_BOOTLOADER_LOG_COLORS is not set
   viejo:
   CONFIG_BOOTLOADER_LOG_COLORS=y

   Antes los mensajes del bootloader usaban códigos de colores ANSI para facilitar la lectura en terminales compatibles. Ahora, al desactivar esta opción, los logs del bootloader se verán en texto plano sin colores, lo cual puede hacer más difícil diferenciar los niveles de mensajes (info, warning, error) pero asegura compatibilidad con cualquier tipo de terminal.

2. Dentro de Serial Flash Configurations habilitamos “Detect flash size when flashing bootloader”

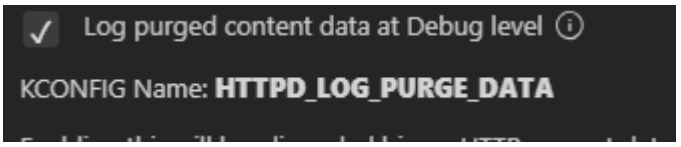


   nuevo:
   # CONFIG_ESPTOOLPY_HEADER_FLASHSIZE_UPDATE is not set
   viejo:
   CONFIG_ESPTOOLPY_HEADER_FLASHSIZE_UPDATE=y

   Antes estaba habilitada la detección automática del tamaño de la flash al grabar el bootloader. Al deshabilitar esta opción, ya no se detecta el tamaño en tiempo de grabación. Esto hace que el proceso de flashing sea más seguro pero menos

flexible.

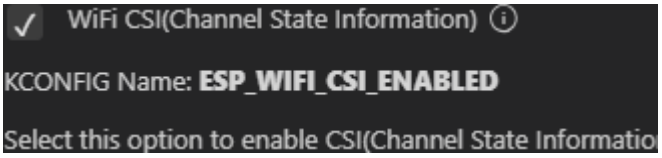
3. En HTTP Server se habilita “Log purged content data at Debug level”



nuevo:
CONFIG_HTTPD_LOG_PURGE_DATA is not set
viejo:
CONFIG_HTTPD_LOG_PURGE_DATA=y

Antes el servidor HTTP registraba a nivel Debug los datos binarios de solicitudes descartadas, lo cual podría saturar los logs si el contenido era muy grande. Al desactivar esta opción, se evita ese exceso de información, manteniendo el log más limpio y ligero.

4. En Wi-Fi se habilita “WiFi CSI(Channel State Information)”



nuevo:
CONFIG_ESP_WIFI_CSI_ENABLED is not set
CONFIG_ESP32_WIFI_CSI_ENABLED is not set
viejo:
CONFIG_ESP_WIFI_CSI_ENABLED=y
CONFIG_ESP32_WIFI_CSI_ENABLED=y

Antes se tenía habilitado el soporte para CSI, que permite obtener datos detallados sobre el estado del canal de comunicación WiFi, útil para análisis avanzados o aplicaciones específicas. Al deshabilitar, se ahorra memoria RAM, lo cual es beneficioso en proyectos como “hello_world” donde esta funcionalidad no es necesaria.

3 - Compilación

b) Análisis de archivos flasher_args.json y project_description.json

Consigna:
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?

El archivo flasher_args.json contiene los parámetros necesarios para realizar la carga del firmware en la placa ESP32-S2-Kaluga-1. El archivo se genera automáticamente durante la compilación del proyecto y especifica qué archivos binarios se deben grabar en la placa, y con las direcciones de memoria correspondientes.

En cuanto al archivo project_description.json describe las características generales del proyecto compilado. Se puede observar que detalla información como el nombre del proyecto, la versión de la aplicación, el chip (ESP32-S2), y la lista de componentes que fueron utilizados.

c) Análisis de las variables declaradas en archivo .map

Consigna:
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?

Dentro del archivo test.map que se encuentra en la carpeta build, se localizaron las variables exampleData y exampleArray, que habían sido declaradas en el hello_world_main.c. Estas variables aparecen en el archivo .bss, que es donde se almacenan los datos, exampleData ocupa 4 bytes y exampleArray, al ser un arreglo de tamaño 12, ocupa 12 bytes.

Respecto a la dirección de memoria, en el archivo test.map ambas variables muestran una dirección de 0x00000000. Esto no representa la dirección real en la memoria del dispositivo, sino que es una dirección relativa. En resumen, las variables están almacenadas correctamente y ocupan el espacio de memoria esperado.

d) Análisis de exampleArray nuevo

Consigna:
Sustituir ARRAY_SIZE por 10 y ver qué ocurre con exampleArray en la memoria.

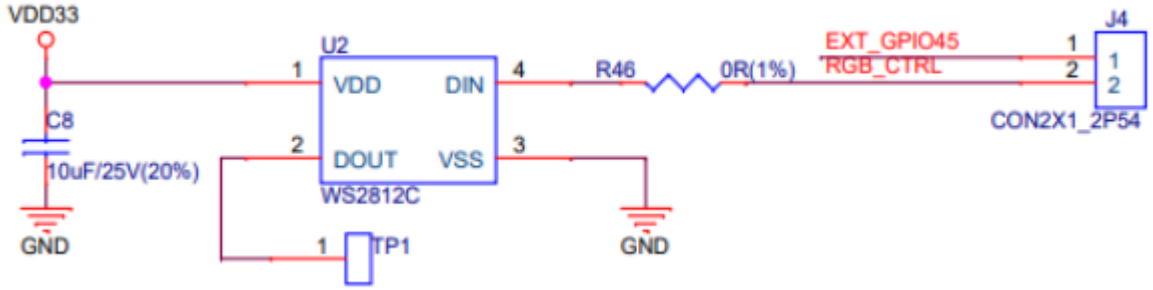
Tras modificar ARRAY_SIZE a 10, se observó que la variable exampleArray ocupa ahora 10 bytes en memoria, según lo indicado en el archivo .map. Esto es coherente ya que exampleArray es un arreglo de tipo char, donde cada elemento utiliza 1 byte. Al reducir la cantidad de elementos de 12 a 10, el espacio reservado disminuyó proporcionalmente.

Segunda Parte

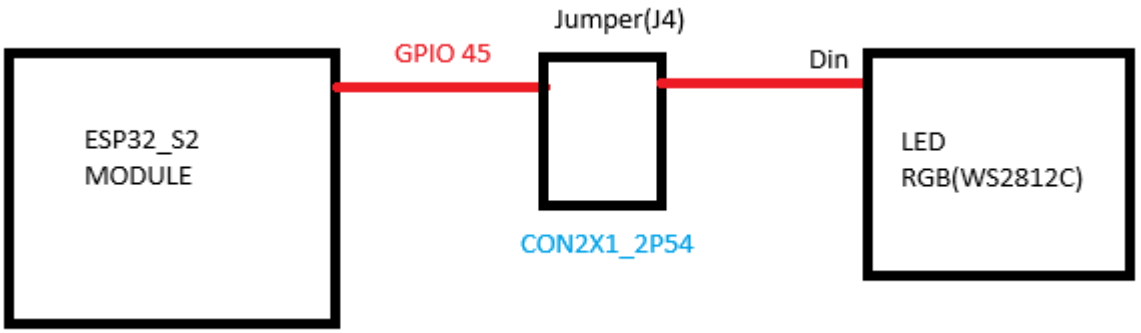
1 - Creación de librería de manejo de LED RGB del hardware

b) Análisis LED RGB y Pseudocódigo

Consigna:
Busque la hoja de datos del LED RGB y comente cómo funciona el componente.
Plantee (en pseudocódigo) cómo controlaría dicho componente.



El componente LED RGB WS2812C dentro de nuestro microcontrolador está dispuesto de tal manera:



Como se observa en el esquema, el pin de datos del LED (DIN) se conecta al pin EXT_GPIO45 del microcontrolador mediante un jumper ubicado en el conector J4. Esta conexión permite enviar señales de control desde el ESP32 hacia el LED a través del Aprotocolo de comunicación propio del WS2812.

Este LED inteligente recibe una cadena de 24 bits por cada píxel (LED). La cadena se compone de:

- 8 bits para el canal verde (G)
- 8 bits para el canal rojo (R)
- 8 bits para el canal azul (B)

en ese orden. Al variar la intensidad de cada canal (valores entre 0 y 255), el LED puede generar una amplia gama de colores.

Pseudocodigo de manejo del led:

```
inicializar_led()

mientras (verdadero):

    // Enviar color rojo
    enviar_color(255, 0, 0)
    esperar(tiempo_prendido)
    apagar_led()

    // Enviar color verde
    enviar_color(0, 255, 0)
    esperar(tiempo_prendido)
    apagar_led()

    // Enviar color azul
    enviar_color(0, 0, 255)
    esperar(tiempo_prendido)
    apagar_led()

fin mientras
```