IOT – Sistemas Embebidos

Laboratorio 1

Integrantes del grupo:

- Nicolás Raposo
- Martín Da Rosa
- Rafael Durán

Docentes: Nicolás Calarco y Pablo Alonso

6/5/2025

Índice

Primera Parte

1) Creación de un proyecto en VSCode + Espressif IDE

a. Abrir el VSCode y crear un nuevo proyecto vacío

Como se pide, se creó un proyecto nuevo, cabe aclarar que a diferencia de la letra donde se indica que el nombre para un proyecto en blanco es “sample_project”, en la extensión de VScode, el proyecto en blanco se llama “template_app”.

b-c. Analizar el árbol de directorios creado en el navegador del proyecto. Analizar los archivos generados: ¿qué secciones tiene?

Como se puede observar en la ilustración 1, hay varias secciones (devcontainer, vscode y main), dentro de estas hay varios archivos .json que parecen contener configuraciones. Por otro lado, la carpeta build fue generada al seleccionar el modelo ESP32-S2 y contiene muchos archivos con prefijos como boot y flash, se estima que estos son para organizar la memoria a la hora de cargar el código.

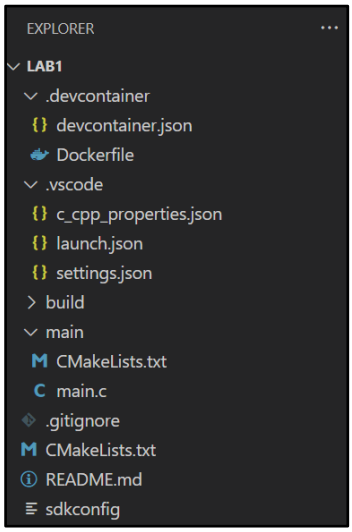
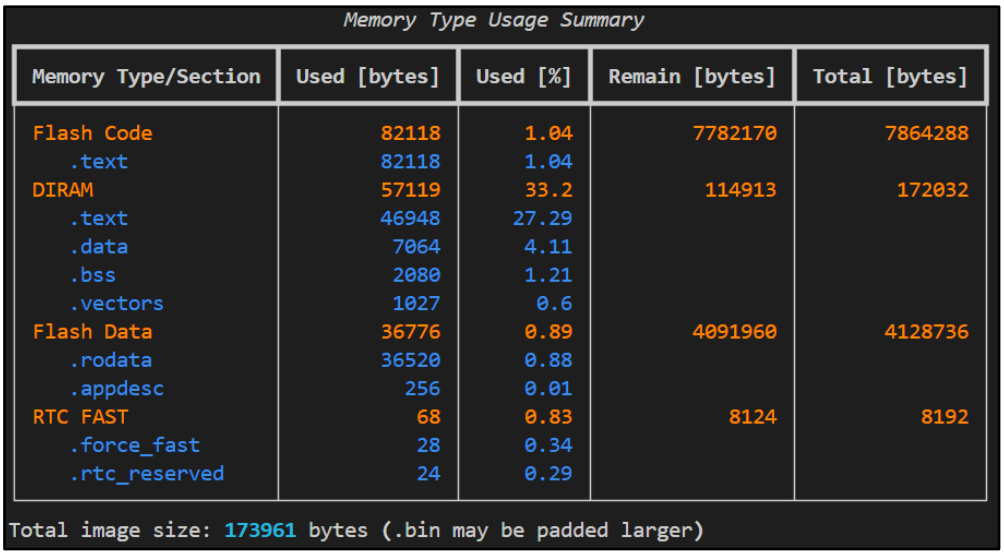


Ilustración 1. Archivos generados por un proyecto en blanco.

d. Compilar el proyecto tal cual está ¿Qué cosas cambian en el workspace del laboratorio? ¿Qué información nos brinda el compilador?

Al compilar el proyecto aparecen algunos archivos más adentro de la carpeta build, asimismo, el compilador muestra la tabla de la ilustración 2, la cual parece tener un resumen del uso de memoria, contiene tipo de memoria, el uso en bytes y en porcentaje, la memoria restante y la total.



Memory Type Usage Summary

Memory Type/Section	Used [bytes]	Used [%]	Remain [bytes]	Total [bytes]
Flash Code	82118	1.04	7782170	7864288
.text	82118	1.04		
DIRAM	57119	33.2	114913	172032
.text	46948	27.29		
.data	7064	4.11		
.bss	2080	1.21		
.vectors	1027	0.6		
Flash Data	36776	0.89	4091960	4128736
.rodata	36520	0.88		
.appdesc	256	0.01		
RTC FAST	68	0.83	8124	8192
.force_fast	28	0.34		
.rtc_reserved	24	0.29		

Total image size: 173961 bytes (.bin may be padded larger)

Ilustración 2. Tabla mostrada por el compilador sobre el uso de memoria.

2) Parámetros de Configuración

a-b-c-d. Espressif IDF nos permite modificar los parámetros de configuración a través de una interfaz gráfica, investigar qué opciones se pueden modificar en el sdkconfig y ver cómo cambia el archivo al guardar. Comparar diferencias con sdkconfig.old.

Como fue indicado, se entró en la configuración de los parámetros mediante el botón de engranaje proporcionado por la extensión, se pueden modificar gran variedad de opciones como la cantidad de información dada por el log, el formato, configuraciones de seguridad y de uso de memoria, opciones para compilar, parámetros para el ADC, DAC y conexión Wireless, distintas velocidades para el clock, etc.

Dado que el archivo sdkconfig tiene más de 2000 lineas de código, es difícil compáralo con el sdkconfig.old cambiando unas pocas opciones, así que se cambió el modelo de la placa del proyecto por otra cualquiera comprobando así que cambian bastantes opciones así como aparecen nuevas.

3) Compilación

a-b. Agregar al archivo main.c la siguiente definición y las siguientes variables: #define ARRAY_SIZE 12 int exampleData; char exampleArray[ARRAY_SIZE]; y analizar los archivos flasher_args.json y project_description.json. ¿Qué puede comentar acerca de estos archivos?

El archivo **flasher_args.json** tiene configuraciones para cargar el código en la placa como “flash_size”,”flash_mode” y “flash_freq”, asimismo parece contener direcciones de memoria como 0x1000 o 0x8000 .

Por otro lado, **project_description.json** como su nombre indica tiene las características del proyecto como el nombre, el chip, las rutas para los archivos de la extensión ESP-IDF,etc.

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?

Las variable exampleArray ocupa 0xc, lo cual es razonable porque es el 12 en hexadecimal (12 es el valor que se le había dado). Por otro lado exampleData ocupa 4 bytes, valor también correcto. Las direcciones están todas en 0, lo cual se supone que es porque dado que el código está siendo compilado sin la placa conectada, la memoria todavía no fue asignada. Ambas variables pueden verse en la ilustración 3.

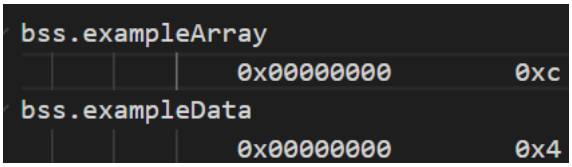


Ilustración 3. Archivo. Map con ARRAY_SIZE = 12

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

Al cambiar ARRAY_SIZE por 10, como se ve en la ilustración 4, el tamaño cambia a 0xa, numero hexadecimal que es 10 en decimal, lo cual es correcto.

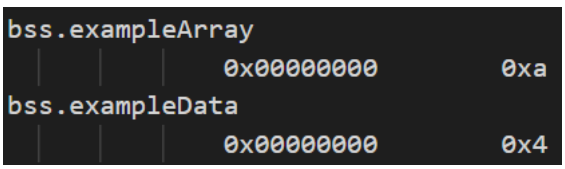


Ilustración 4. Archivo .map con ARRAY_SIZE = 10

Segunda Parte

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

a-b) Busque la hoja de datos del LED RGB y comente cómo funciona el componente. Plantee cómo controlaría dicho componente.

Como se puede ver en la ilustración 5, se realizó un diagrama de flujo indicando como se manejaría el LED RGB.

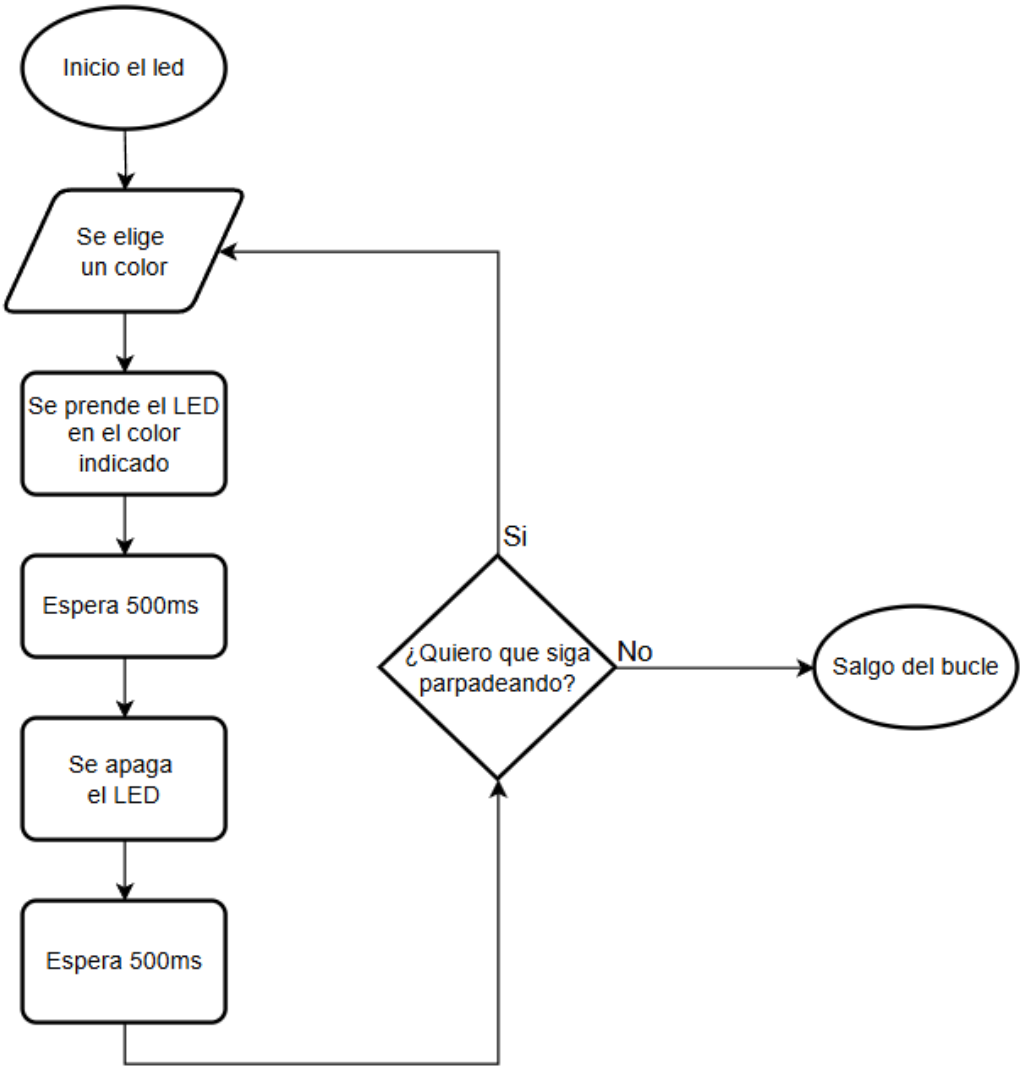


Ilustración 5. Diagrama de flujo para el control del LED.

Este diagrama de flujo muestra cómo funciona el programa al ejecutarse en la placa para hacer que un LED parpadee.

Primero, el programa se inicia y se prepara todo lo necesario para usar el LED. Luego, se elige un color con el que se va a encender el LED (al tratarse de un LED RGB, existen 3 posibles valores para configurar).

A continuación, el LED se enciende con el color seleccionado y se mantiene así durante medio segundo (500 milisegundos). Pasado ese tiempo, el LED se apaga y nuevamente se espera medio segundo.

A continuación, el programa pregunta si se desea que el LED siga parpadeando. Si la respuesta es "sí", se repite el ciclo desde la elección del color. Si la respuesta es "no", el programa sale del bucle y deja de parpadear.

2) Creación de una librería nueva

Como se puede observar en la ilustración 6, se creó la librería mi_led que funciona en base a los archivos provistos, dentro de mi_led.c está el código de la función que hará parpadear el LED

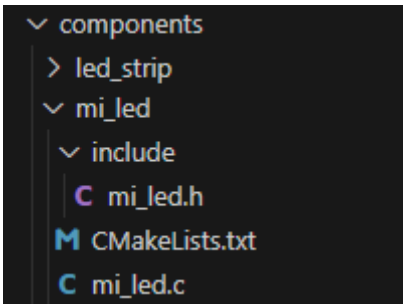


Ilustración 6. Librería para el control del LED RGB.

¿Cómo funciona el programa?

1. **Encender el LED en diferentes colores:**
    - Se elige el color rojo (`255, 0, 0`) y se enciende el LED.
    - Se mantiene encendido durante 500 ms.
    - Luego, el LED se apaga y se espera otros 500 ms.
    - Se repite el mismo proceso para el color verde (`0, 255, 0`) y el color azul (`0, 0, 255`).

2. **Bucle de parpadeo (función `blink_led`)**
    - La función `blink_led()` se encarga de encender el LED en tres colores (rojo, verde y azul), con pausas de 500 ms entre cada cambio.
    - Primero, enciende el LED en rojo, espera y luego lo apaga.
    - Después, enciende el LED en verde, espera y lo apaga.
    - Finalmente, enciende el LED en azul, espera y lo apaga.
    - Esta función se ejecuta en un ciclo continuo si es llamada repetidamente.

3. **Parpadeo rápido (función `one_blink`)**
    - La función `one_blink()` enciende el LED solo en color rojo por 250 ms y luego lo apaga por otros 250 ms.
    - Esta función permite hacer un solo destello en lugar de un parpadeo de múltiples colores como en `blink_led()`.

3) Creación de una función de delay

Como fue indicado, se creó la librería mi_delay (ver ilustración 7), la cual adapta la función nativa de delay (que toma µs) para hacerla funcionar en milisegundos y segundos.

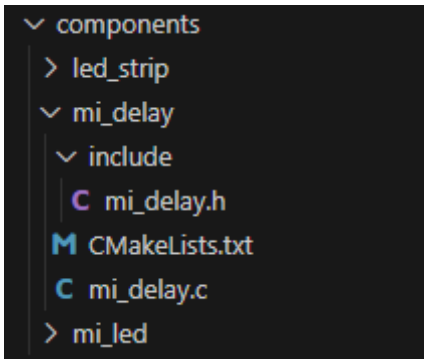


Ilustración 7. Librería de delay.

Funcionamiento general:

La función `delay()` recibe dos parámetros:

- Un valor entero sin signo (`uint32_t tiempo`) que representa la cantidad de tiempo a esperar.
- Un carácter (`char unidad`) que indica la unidad de tiempo utilizada. Esta unidad puede ser:
    - `'u'`: microsegundos,
    - `'m'`: milisegundos,
    - `'s'`: segundos.

Según la unidad especificada, la función convierte el valor recibido a microsegundos, ya que internamente utiliza la función `esp_rom_delay_us()`, la cual realiza retardos expresados en microsegundos.

Comportamiento específico:

- Si la unidad es `'u'`, se invoca directamente `esp_rom_delay_us(tiempo)`, generando un retardo del número exacto de microsegundos indicado.
- Si la unidad es `'m'`, el valor de `tiempo` se multiplica por 1.000, generando un retardo equivalente en microsegundos (es decir, mil microsegundos por cada milisegundo).
- Si la unidad es `'s'`, el valor se multiplica por 1.000.000, convirtiendo segundos a microsegundos.

Conclusión

A lo largo del presente trabajo se desarrolló un proyecto completo sobre la plataforma ESP32-S2 utilizando el entorno de desarrollo VSCode junto con la extensión ESP-IDF. Inicialmente, se realizó la configuración y análisis de un proyecto base, lo que permitió comprender la estructura de carpetas y archivos generados automáticamente, así como el comportamiento del sistema durante las fases de compilación y configuración. Particular atención fue dada al archivo `sdkconfig`, que centraliza los parámetros de configuración del sistema y cuya modificación incide directamente en el comportamiento del firmware.

Posteriormente, se integraron funcionalidades específicas orientadas al control de un LED RGB, lo cual implicó la creación de una librería modularizada. Esta librería permitió encapsular las funciones necesarias para manipular el estado del LED, brindando versatilidad tanto para efectos de parpadeo continuo como para destellos individuales. Asimismo, se diseñó e implementó una función de retardo personalizada que extiende la función nativa `esp_rom_delay_us()` para aceptar múltiples unidades de tiempo, lo que mejora la legibilidad y adaptabilidad del código.

El desarrollo progresivo de estas funcionalidades permitió consolidar conocimientos sobre el manejo de memoria, segmentación de código, compilación cruzada y la interacción con hardware mediante programación embebida. En conclusión, el trabajo constituye una base sólida para el desarrollo de aplicaciones más complejas sobre microcontroladores ESP32, promoviendo el uso de buenas prácticas de modularidad, documentación y abstracción funcional.

Webgrafía

Espressif Systems. (s.f.). *ESP-IDF Programming Guide – Get Started (ESP32-S2)*. Recuperado el 5 de mayo de 2025, de https://docs.espressif.com/projects/esp-idf/en/latest/esp32s2/get-started/index.html

Espressif Systems. (s.f.). *ESP-IDF Programming Guide – General Documentation (ESP32-S2)*. Recuperado el 5 de mayo de 2025, de https://docs.espressif.com/projects/esp-idf/en/latest/esp32s2/index.html

Espressif Systems. (2022). *Build System — CMakeLists.txt file (v4.4.2)*. En *ESP-IDF Programming Guide*. Recuperado el 5 de mayo de 2025, de https://docs.espressif.com/projects/esp-idf/en/v4.4.2/esp32s2/api-guides/build-system.html#project-cmakelists-file

Espressif Systems. (s.f.). *ESP32-S2-Kaluga-1 V1.2 – Esquemático del Hardware*. [Archivo PDF]. Recuperado de documentación técnica de Espressif.