



IOT – Sistemas Embebidos

Laboratorio 1

Integrantes del grupo:

- Nicolás Raposo
- Martín Da Silva
- Rafael Durán

Docentes: Nicolás Calarco y Pablo Alonso

6/5/2025

Índice

Primera Parte	3
1) Creación de un proyecto en VSCode + Espressif IDE	3
a. Abrir el VSCode y crear un nuevo proyecto vacío	3
b-c. Analizar el árbol de directorios creado en el navegador del proyecto. Analizar los archivos generados: ¿qué secciones tiene?	3
d. Compilar el proyecto tal cual está ¿Qué cosas cambian en el workspace del laboratorio? ¿Qué información nos brinda el compilador?	3
2) Parámetros de Configuración	4
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.....	4
3) Compilación	5
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?	5
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?.....	5
d. Sustituir ARRAY_SIZE por 10 y ver qué ocurre con exampleArray en la memoria.....	5
Segunda Parte	6
1) Creación de librería de manejo de LED RGB del hardware	6
a-b) Busque la hoja de datos del LED RGB y comente cómo funciona el componente. Plantee cómo controlaría dicho componente.	6
2) Creación de una librería nueva	7
3) Creación de una función de delay	7

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.

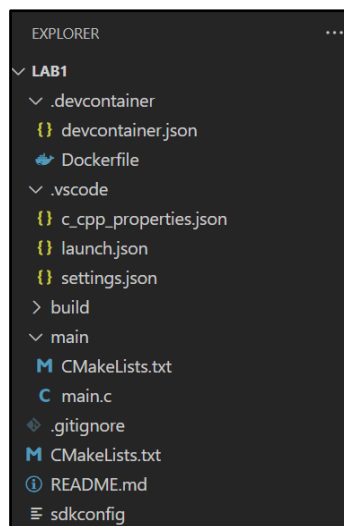


Ilustración 1. Archivos generados por un proyecto en blanco.

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 líneas de código, es difícil compárarlo 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.

bss.exampleArray			
		0x00000000	0xc
bss.exampleData			
		0x00000000	0x4

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, número hexadecimal que es 10 en decimal, lo cual es correcto.

bss.exampleArray			
		0x00000000	0xa
bss.exampleData			
		0x00000000	0x4

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.

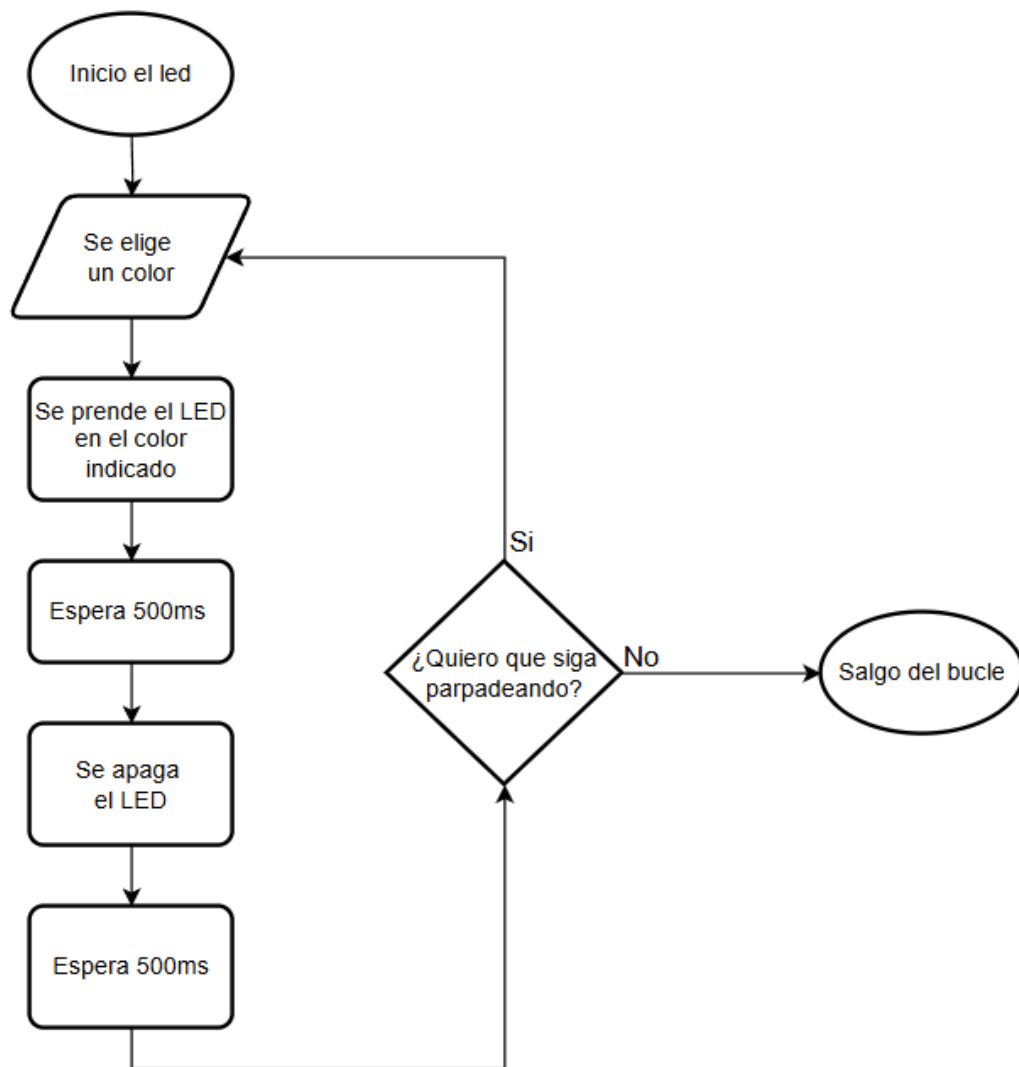


Ilustración 5. Diagrama de flujo para el control del LED.

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

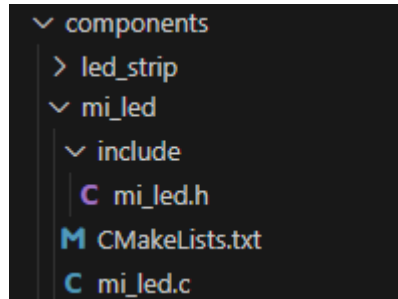


Ilustración 6. Librería para el control del LED RGB.

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 μ s) para hacerla funcionar en milisegundos y segundos.

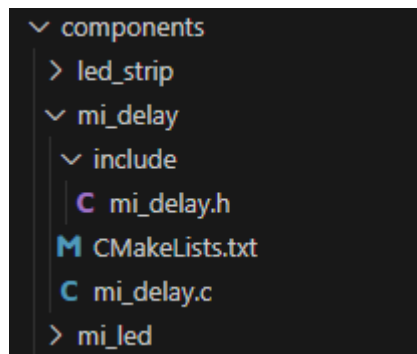


Ilustración 7. Librería de delay.