Sistemas embebidos

Trabajo Practico – BareMetal



A blue letter on a black background

Description automatically generated

* **Alumno**
  + Castiglioni Jorge Luis - LU1126042
* **Profesor**
  + Amet, Leonardo Javier
* **Curso**
  + Miércoles – Noche
* **Cuatrimestre**
  + 1º Cuatrimestre
* **Fecha de entrega**
  + 01/06/2025

1. Compilación y veriﬁcación: Cloná el repositorio, compilá y ﬂasheá un ejemplo funcional. Adjuntá evidencia.

2. Comparación de acceso: Compará el acceso a un LED en PC13 mediante:

o Punteros directos (Práctica 1)

o CMSIS (GPIOC->ODR, RCC->APB2ENR, etc.)

3. Encapsulamiento con CMSIS: Escribí una función led\_toggle() que use CMSIS para alternar el estado de PC13.

4. Encendido condicional: Encendé el LED solo si PA0 está en bajo. Usá GPIOA-

> IDR para leer el pin.

5. Conﬁguración de múltiples pines: Encendé secuencialmente PC13, PC14 y PC15. Explicá cómo se conﬁgura cada pin con CMSIS.

6. Modiﬁcación de startup: Identiﬁcá el Reset\_Handler en startup.c. Veriﬁcá qué cambios hay respecto a la versión anterior.

7. Observación en objdump: Generá el binario y usá objdump -d para veriﬁcar:

o Dónde comienza main()

o Cómo se ve el acceso CMSIS (usando LDR, STR, etc.)

8. Retardo bloqueante con CMSIS: Usá un bucle for y observá el comportamiento al cambiar la cantidad de ciclos. Estimá la duración.

9. Documentación y estructuras CMSIS: Buscá en los encabezados CMSIS cómo está deﬁnida la estructura GPIO\_TypeDef. ¿Qué ventajas tiene este enfoque?

10. Exploración de registros RCC: Explicá el signiﬁcado de cada bit de RCC-

> APB2ENR. Identiﬁcá qué periféricos se pueden habilitar desde allí.

11. Análisis del mapa de memoria: Generá el archivo .map del proyecto y explicá cómo identiﬁcar:

o Tamaño total de .text, .data, .bss

o Dirección de carga de cada sección

12. Variables globales: Declarar y usar una variable global inicializada (int x = 42;) y una no inicializada (int y;). Usá objdump -h para identiﬁcar en qué sección del binario están.

13. Uso de NVIC con CMSIS: Investigá cómo se accede a NVIC con CMSIS. ¿Qué funciones o estructuras están disponibles para habilitar interrupciones?

14. Exploración de estructuras CMSIS adicionales: Observá la estructura TIM\_TypeDef. ¿Qué campos contiene? ¿Cómo podrías usarla para conﬁgurar un timer?

15. Modularización del proyecto: Separá la lógica de encendido del LED en un archivo led.c con su encabezado led.h. Mostrá cómo modiﬁcar el Makeﬁle, si fuera necesario, para compilarlo correctamente.

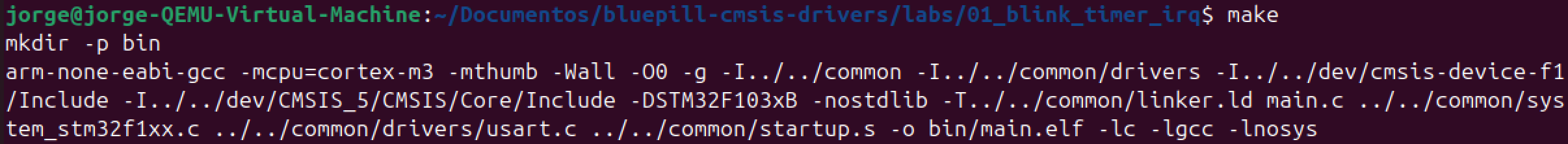
16. Inicialización estructurada: Escribí una función gpio\_conﬁg\_output(GPIO\_TypeDef \*gpio, uint8\_t pin) que reciba un puerto y número de pin, y conﬁgure ese pin como salida push-pull. Usala para conﬁgurar PC13 y PC14. Explicá cómo mejora la reutilización del código frente a hacer la conﬁguración "a mano" cada vez.

17. Lectura de múltiples pines: Leé el estado de PA0 y PA1 simultáneamente y encendé diferentes LEDs según su combinación. Implementalo de manera clara usando máscaras.

18. Uso de volatile: Investigá el signiﬁcado de la palabra clave volatile en C. ¿Por qué es importante usarla al declarar punteros a registros de periféricos? Escribí un pequeño ejemplo donde se note el efecto de omitir volatile (puede ser un fragmento de código comentado o explicado).

**Verificación del entorno**

Se utilizo la herramienta Makefile para compilar el proyecto, y la compilación se da sin errores



Luego se procedió a Flashear el proyecto y enviarlo via ST-LINK a la bluepill con el resultado esperado.

Texto

El contenido generado por IA puede ser incorrecto.

El led parpadea con el delay programado (VIDEO)

Una mesa de madera

El contenido generado por IA puede ser incorrecto.

**Comparacion de acceso:**

En el TP1 accedemos al LED en PC13 mediante una definición inicial en el main donde usando la instrucción #DEFINE se establecen los punteros a las direcciones de los registros de habilitación del clock para los periféricos, el registro del GPIOC base y sus OFFSET correspondientes para la activación del LED

Texto

El contenido generado por IA puede ser incorrecto.

Una vez Flasheado el micro con el nuevo firmware, el micro hace un reset

Siguiendo las instrucciones del Startup.c, lo primero que ocurre es que se inicializa el program counter en la direccion inicial de la flash 0x08000000, ahí se carga la dirección siguiente a la ultima dirección de la sección de RAM.

Seguido se inicializa el program counter en la dirección 0x08000004

La primer instrucción a ejecutar por el program counter es Reset\_handler, la cual copia las variables inicializadas a la sección .data de la FLASH a la RAM y las variables no inicializadas a la seccion .bss

Donde se inicializan todas a 0. Luego se ejecuta el Main

Por otro lado, en el TP 2 se utiliza la librería "stm32f1xx.h" donde ya se encuentran definidas todas las direcciones de los registros del Micro utilizado. Utilizando TYPEDEF se generaran las estructuras de la familia de los Micro STM32, facilitando el acceso al registro mediante “->”, simplificando la comprensión del código y agregando otra capa de abstracción al programador que se independiza de tener que escribir las direcciones de los registros y que el código sea utilizable para otro micro dentro de la familia STM32F1.

Texto

El contenido generado por IA puede ser incorrecto.

**Encapsulamiento con CMSIS:**

Se creo la nueva función led\_toggle () y se agregó en el main, luego será invocada en la interrupción para prender y apagar el led.

Texto

El contenido generado por IA puede ser incorrecto.

**Encendido condicional:**