

Diseño e Implementación de un Sistema de Seguridad Lógico basado en Arquitectura ARM Cortex-M3 (STM32)

SISTEMAS ELECTRÓNICOS DIGITALES

(2025-2026)

A-408 GRUPO 16

- Julia Magallares Verde (56449)
- Pablo Muñoz Moreno (56507)
- Victor Malumbres Muñoz (56451)

➤ Tutor: Rubén Nuñez

ÍNDICE

1. Introducción	4
2. Objetivos y Especificaciones del Sistema	4
2.1. Objetivos Generales y Requisitos Técnicos	4
2.2. Especificaciones Funcionales del Juego	5
2.3. Herramientas de Desarrollo (Toolchain).....	5
3. Metodología de Diseño y Arquitectura Software	6
3.1. Estrategia de Diseño Modular.....	6
3.2. Arquitectura del Firmware	6
3.3. Implementación de la Gestión de Tiempos	7
4. Descripción Detallada de Bloques (Hardware Abstraction Layer)	7
4.1. Configuración del Reloj del Sistema (RCC)	7
4.2. Adquisición de Datos (ADC1)	8
4.3. Interfaz de Comunicación I2C	8
4.4. Sistema de Temporización Hardware (TIM2).....	9
4.5. Interfaz de Entrada/Salida Digital (GPIO).....	9
5. Controlador Lógico del Juego (FSM)	10
5.1. Definición de la Máquina de Estados Finitos	10
5.2. Lógica de Transiciones.....	11
5.3. Algoritmos Específicos	12
5.3.1. Generación Pseudoaleatoria	12
5.3.2. Algoritmo de Comparación y Bloqueo (Feedback)	12
6. Implementación Física y Montaje	13
6.1. Mapa de Conexiones (Pinout).....	13
6.2. Topología de Alimentación	14
6.3. Descripción de los Circuitos de Interfaz.....	14
6.3.1. Acondicionamiento de Entradas	14
6.3.2. Etapas de Salida.....	15
7. Pruebas y Resultados Experimentales	15
7.1. Estrategia de Verificación.....	15
7.2. Verificación de Periféricos (Pruebas Unitarias).....	16
7.3. Validación Funcional del Juego	16

7.3.1. Caso de Prueba 1: Secuencia de Victoria	16
7.3.2. Caso de Prueba 2: Derrota por Agotamiento de Intentos	17
7.3.3. Caso de Prueba 3: Derrota por Tiempo Límite.....	17
7.4. Análisis de la Retroalimentación Visual (Feedback).....	17
8. Problemas Encontrados y Soluciones Adoptadas	18
8.1. Rebotes Mecánicos en las Entradas Digitales (Bouncing).....	18
8.2. Bloqueo del Temporizador por Retardos de CPU	18
8.3. Ruido y Escalado en la Conversión ADC.....	19
8.4. Gestión de Colores Mezclados (Amarillo)	19
9. Manual de Usuario	20
9.1. Puesta en Marcha.....	20
9.2. Interfaz de Juego (Visualización).....	20
9.3. Controles y Operación.....	21
9.4. Finalización de la Partida.....	21
10. Conclusiones y Líneas Futuras.....	22
10.1. Conclusiones	22
10.2. Líneas Futuras	23
11. Anexos	24
11.1. Archivos de Cabecera y Configuración (Headers)	24
11.2. Código Fuente Principal (Main y Lógica)	25
11.3. Drivers de Periféricos	38

1. Introducción

Este proyecto, titulado "Proyecto Cerrojo", consiste en el diseño y programación de un sistema electrónico basado en un microcontrolador STM32 de arquitectura ARM Cortex-M. La aplicación desarrollada es un juego de lógica similar al conocido *Mastermind* o a una caja fuerte electrónica, donde el usuario tiene el reto de adivinar una clave de cuatro dígitos generada aleatoriamente por el sistema.

La principal particularidad del diseño reside en su interfaz de entrada: en lugar de utilizar un teclado digital convencional, la selección de los dígitos se realiza mediante cuatro potenciómetros. Esto implica la necesidad de leer y convertir señales analógicas de forma continua para transformarlas en valores discretos del 0 al 9. Para la interacción con el usuario, el sistema incorpora una pantalla LCD, que muestra el tiempo de juego y los intentos restantes, junto con una serie de LEDs que proporcionan retroalimentación visual inmediata sobre los aciertos mediante un código de colores.

Para la implementación se ha optado por una arquitectura de 32 bits frente a soluciones más básicas de 8 bits. Esta decisión técnica se justifica por la necesidad de utilizar librerías de abstracción de hardware (HAL) para un desarrollo más profesional y escalable. Además, el STM32 ofrece la capacidad de procesamiento necesaria para gestionar simultáneamente la conversión de los cuatro canales analógicos y un sistema de interrupciones prioritarias, garantizando así que la cuenta atrás del juego sea precisa y no se vea afectada por la actualización de la pantalla o la lectura de sensores.

2. Objetivos y Especificaciones del Sistema

2.1. Objetivos Generales y Requisitos Técnicos

El objetivo principal de este proyecto es diseñar un sistema embebido capaz de gestionar múltiples periféricos de forma concurrente sin comprometer la estabilidad del procesador. Más allá de la funcionalidad lúdica, el sistema debe cumplir con una serie de requisitos técnicos propios de una aplicación de ingeniería en tiempo real:

- **Arquitectura No Bloqueante:** El software debe evitar el uso de retardos de CPU (delays) durante el ciclo de juego. La lectura de sensores y la actualización de la pantalla no deben detener la cuenta atrás del temporizador interno.
- **Gestión Eficiente de Periféricos:** El sistema debe controlar simultáneamente un bus de comunicación serie (I2C) para el display, cuatro canales de conversión analógico-digital (ADC) y múltiples líneas de entrada/salida digital (GPIO) para botones y LEDs.

- **Modularidad del Código:** La implementación debe seguir una estructura por capas, separando claramente los controladores de hardware (*drivers*) de la lógica de la aplicación. Esto facilita la depuración y permite modificar las reglas del juego sin necesidad de reescribir la gestión eléctrica de los componentes.

2.2. Especificaciones Funcionales del Juego

Desde el punto de vista del usuario, el sistema debe comportarse como un juego de lógica deductiva con las siguientes reglas y mecánicas de funcionamiento, las cuales han sido programadas en el módulo de lógica:

- **Mecánica de Entrada:** El usuario no introduce los números mediante un teclado, sino ajustando cuatro potenciómetros. El sistema debe traducir la señal de voltaje (0V - 3.3V) a un valor entero legible (0-9) en tiempo real.
- **Condiciones de Victoria y Derrota:** El jugador dispone de un máximo de 6 intentos y un tiempo límite de 120 segundos para adivinar la clave secreta. Si se agota cualquiera de estos recursos, el sistema pasa al estado de derrota.
- **Algoritmo de Retroalimentación:** Tras validar un intento, el sistema debe ofrecer pistas visuales inmediatas para cada uno de los cuatro dígitos, basándose en la proximidad numérica respecto a la clave secreta:
 - **Verde:** El número introducido es correcto.
 - **Amarillo:** El número está cerca (diferencia de ± 1 respecto al valor secreto).
 - **Rojo:** El número es incorrecto y no está próximo.
- **Interfaz de Control:** Se utilizan dos pulsadores físicos: uno para validar el intento actual y otro para reiniciar el sistema y generar una nueva clave aleatoria.

2.3. Herramientas de Desarrollo (Toolchain)

Para llevar a cabo la implementación del firmware se ha utilizado un conjunto de herramientas estándar en el desarrollo con microcontroladores STM32:

- **Entorno de Desarrollo (IDE):** Se ha empleado STM32CubeIDE, que integra la configuración gráfica del microcontrolador y la generación de código de inicialización.

- **Librerías de Abstracción (HAL):** El código se apoya en las librerías *Hardware Abstraction Layer* proporcionadas por el fabricante. Estas permiten controlar los periféricos (ADC, Timers, I2C) mediante funciones de alto nivel, garantizando la portabilidad del código entre diferentes modelos de la familia STM32.
- **Lenguaje de Programación:** Todo el firmware ha sido desarrollado en lenguaje C, utilizando estructuras de datos y punteros para la gestión eficiente de la memoria y los periféricos.

3. Metodología de Diseño y Arquitectura Software

3.1. Estrategia de Diseño Modular

Para cumplir con el requisito de mantenibilidad, la estructura del proyecto se ha fragmentado en archivos fuentes independientes, asignando responsabilidades específicas a cada uno para evitar un código monolítico en el archivo principal. La distribución implementada es la siguiente:

- **main.c (Capa de Sistema):** Se limita estrictamente a la inicialización de los relojes y la llamada a los configuradores de periféricos generados por el IDE. Su bucle infinito delega inmediatamente el control a la capa lógica, actuando únicamente como contenedor del flujo principal.
- **logica_juego.c (Capa de Aplicación):** Contiene la Máquina de Estados y las variables globales del juego (intentos, tiempo, clave secreta). Es el único archivo que toma decisiones; no accede a los registros del hardware directamente, sino que solicita acciones a la capa inferior.
- **hardware.c y i2c-lcd.c (Capa de Drivers):** Encapsulan las librerías HAL. Aquí se traducen las peticiones de alto nivel (ej. "Encender LED verde") en las operaciones de registros y pines necesarias (ej. HAL_GPIO_WritePin en el puerto PC0), aislando la lógica de los detalles eléctricos.

3.2. Arquitectura del Firmware

El software sigue un patrón de arquitectura basado en **Polling de Estados** dentro de un "Super-Bucle".

Tras la ejecución de HAL_Init() y SystemClock_Config(), el sistema entra en un ciclo infinito while(1). En cada iteración, se invoca a la función Logica_Ejecutar_Ciclo().

Esta función evalúa la variable global `estado_actual` y ejecuta el bloque de código correspondiente (switch-case). Esta arquitectura garantiza que en cada vuelta del procesador se verifiquen las condiciones de transición, permitiendo que el sistema reaccione a los cambios de estado sin necesidad de un sistema operativo en tiempo real (RTOS).

3.3. Implementación de la Gestión de Tiempos

Para materializar el requisito de ejecución no bloqueante, se han implementado dos mecanismos de temporización que conviven simultáneamente:

1. **Temporización Crítica (Interrupción):** Para el contador de juego (120 segundos), se utiliza el **TIM2** configurado para desbordarse cada segundo. La rutina de atención a la interrupción (`HAL_TIM_PeriodElapsedCallback`) decrementa la variable `tiempo_restante` en segundo plano, asegurando que la cuenta atrás sea independiente del flujo principal.
2. **Temporización de Interfaz (Software Ticks):** Para las pausas visuales (mensajes de victoria o parpadeos), se utiliza el contador de sistema `HAL_GetTick()`. El código captura la marca de tiempo al entrar en un estado (`t_inicio_estado`) y calcula en cada ciclo la diferencia con el tiempo actual (`t_actual`). Esto permite generar esperas de varios segundos sin detener la lectura de pulsadores ni el refresco del LCD, algo que sería imposible utilizando la instrucción estándar `HAL_Delay()`.

4. Descripción Detallada de Bloques (Hardware Abstraction Layer)

4.1. Configuración del Reloj del Sistema (RCC)

Para garantizar el funcionamiento síncrono de todos los periféricos, la configuración del reloj se establece en la función `SystemClock_Config`. Se ha optado por utilizar el oscilador externo de alta velocidad (**HSE**) como fuente primaria, ya que ofrece mayor estabilidad que los osciladores internos RC.

En el código se observa la activación del **PLL** (Phase Locked Loop) con los parámetros `PLLM=4`, `PLLN=192` y `PLLP=4`. Esta configuración es necesaria para elevar la frecuencia de entrada y distribuir una señal de reloj estable tanto a la CPU como a los

buses APB1 y APB2, donde se encuentran conectados el temporizador y el conversor analógico-digital.

4.2. Adquisición de Datos (ADC1)

La lectura de los potenciómetros se gestiona mediante el convertidor **ADC1**, configurado con una resolución de **12 bits** (valores de 0 a 4095). Según la función `MX_ADC1_Init`, se han habilitado cuatro canales específicos correspondientes a los pines físicos de los potenciómetros:

- Canal 0 (Rank 1)
- Canal 1 (Rank 2)
- Canal 4 (Rank 3)
- Canal 8 (Rank 4)

El modo de conversión seleccionado es **Discontinuous Mode**, lo que permite tener un control preciso sobre cuándo se leen los sensores. En el archivo `hardware.c`, la función `Leer_Potencios` realiza el escalado de la señal cruda. Es destacable que, aunque el ADC devuelve hasta 4095, en el código se divide por **4100** (`val * 10 / 4100`) para obtener un rango limpio de 0 a 9, asegurando que el valor máximo no desborde el dígito decimal por ruido en la alimentación.

4.3. Interfaz de Comunicación I2C

Para el control de la pantalla LCD se utiliza el periférico **I2C1** en modo maestro. La inicialización (`MX_I2C1_Init`) establece una velocidad de reloj de **100 kHz** (Standard Mode) y un Duty Cycle de 2.

A nivel de protocolo, la dirección del dispositivo esclavo se ha definido en el driver `i2c-lcd.c` como **0x4E**. La comunicación se realiza mediante tramas de 4 bits (nibbles) para controlar el controlador HD44780 de la pantalla, enviando primero la parte alta y luego la baja del byte, gestionando mediante software los pines de control *Enable* (EN) y *Register Select* (RS) integrados en la trama I2C.

4.4. Sistema de Temporización Hardware (TIM2)

La base de tiempo del juego no depende de bucles de software, sino del temporizador de propósito general **TIM2**. En la función `MX_TIM2_Init` se han cargado los siguientes valores en los registros:

- **Prescaler:** 16000
- **Period (ARR):** 3000

Estos valores dividen la frecuencia del reloj del bus APB1 para generar un evento de desbordamiento (Update Event) exactamente **cada 1 segundo**. Este evento dispara la interrupción gestionada en `main.c`, la cual invoca a `Logica_Timer_Callback()` para restar una unidad al tiempo de juego, garantizando una precisión cronométrica independiente de la carga de la CPU.

4.5. Interfaz de Entrada/Salida Digital (GPIO)

La interacción directa con el usuario se gestiona a través de los puertos de propósito general (GPIO). La configuración a nivel de registros se ha diseñado para garantizar la estabilidad eléctrica de las señales:

- **Entradas (Pulsadores):** Se han habilitado líneas de interrupción externa (EXTI) asociadas a los puertos A y C. A nivel eléctrico, es fundamental destacar que se ha activado la resistencia de **Pull-Down interna** del microcontrolador. Esta configuración fuerza un nivel lógico bajo ('0') cuando los pulsadores están en reposo, eliminando la necesidad de resistencias físicas externas en la placa y evitando lecturas erróneas por ruido electromagnético.
- **Salidas (Indicadores y Actuadores):**
 - **Matriz de LEDs:** Se ha dedicado un puerto completo de 8 bits (Puerto C) configurado en modo *Output Push-Pull* a baja velocidad. Esto permite controlar individualmente los ánodos de los LEDs de estado.
 - **Salida de Audio:** Para el zumbador, se utiliza una salida digital convencional en el Puerto B, la cual se conmuta por software para generar la frecuencia audible deseada.

5. Controlador Lógico del Juego (FSM)

5.1. Definición de la Máquina de Estados Finitos

El control del flujo del programa no se realiza mediante una ejecución lineal secuencial, sino a través de una Máquina de Estados Finitos (FSM) implementada en el archivo logica_juego.c. Este modelo permite que el sistema se comporte de manera determinista, reaccionando a eventos específicos (pulsaciones o temporizadores) dependiendo de la situación actual del juego.

Se han definido seis estados posibles mediante una enumeración (typedef enum), los cuales cubren todo el ciclo de vida de una partida:

1. **ESTADO_ESPERA:** Es el estado de reposo inicial. El sistema muestra un mensaje de bienvenida en el LCD ("Proyecto Cerrojo") y permanece a la espera de que el usuario inicie la actividad.
2. **ESTADO_GENERAR:** Fase de configuración previa a la partida. Aquí se reinician los contadores (vidas y tiempo) y se calcula la nueva clave secreta.
3. **ESTADO_JUGANDO:** Es el estado principal. El bucle permite la lectura continua de los potenciómetros, actualiza la pantalla con los valores seleccionados y monitoriza el tiempo restante.
4. **ESTADO_VERIFICAR:** Estado transitorio que se activa al pulsar el botón de validación. El sistema pausa momentáneamente la interacción para comparar la entrada del usuario con la clave secreta y actualizar los indicadores visuales.
5. **ESTADO_VICTORIA:** Se alcanza únicamente si los cuatro dígitos coinciden. El sistema bloquea el juego, activa la señal acústica de éxito y muestra la clave resuelta.
6. **ESTADO_DERROTA:** Se activa si el contador de tiempo llega a cero o si se agotan los 6 intentos disponibles.

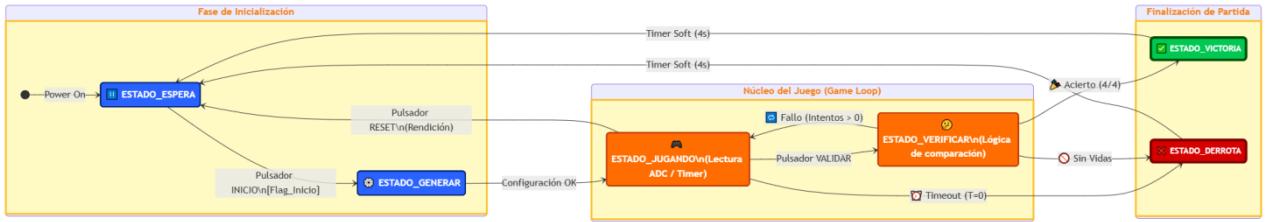


Figura 5.1: Diagrama de Estados del Controlador del Ascensor.

[\[Ver diagrama interactivo en alta resolución\]](#)

5.2. Lógica de Transiciones

La navegación entre los estados descritos anteriormente está gobernada por una serie de condiciones lógicas ("triggers") evaluadas en cada ciclo de reloj. Estas transiciones aseguran que el juego progrese de forma coherente:

- **Inicio de Partida (ESPERA → GENERAR):** Se dispara mediante la detección del *flag* flag_inicio, asociado a la interrupción del pulsador de "Inicio/Reset".
- **Validación de Intento (JUGANDO → VERIFICAR):** Ocurre cuando el usuario presiona el botón "Validar". El sistema entra en este estado para procesar la jugada y, tras 1,5 segundos de visualización de resultados, decide automáticamente si volver a JUGANDO (si quedan vidas y no ha ganado) o terminar la partida.
- **Condición de Derrota (JUGANDO → DERROTA):** Esta transición es prioritaria y automática. Se ejecuta si la variable tiempo_restante desciende a 0 (gestionada por el Timer) o si la variable intentos_restantes se agota tras una verificación fallida.

- **Reinicio Automático (VICTORIA/DERROTA → ESPERA):** Tras finalizar una partida, el sistema mantiene el resultado en pantalla durante 4 segundos antes de reinicializar las variables y volver al estado de reposo, quedando listo para un nuevo usuario.

5.3. Algoritmos Específicos

Dentro de la máquina de estados, residen dos algoritmos fundamentales que dotan de inteligencia al sistema: la generación de aleatoriedad y la lógica de comparación de la clave.

5.3.1. Generación Pseudoaleatoria

Al entrar en el estado GENERAR, el sistema debe crear una clave secreta que sea impredecible. Dado que los microcontroladores son sistemas deterministas, se utiliza la función srand(HAL_GetTick()) para inicializar la semilla del generador aleatorio.

Al utilizar el tiempo de ejecución actual (en milisegundos) como semilla, se garantiza que la secuencia de números generada por rand()%10 sea diferente en cada encendido o reinicio, evitando que el juego sea repetitivo o predecible.

5.3.2. Algoritmo de Comparación y Bloqueo (Feedback)

En el estado VERIFICAR, se ejecuta el algoritmo central del juego, el cual compara dígito a dígito el vector intento_actual[] con el vector clave_secreta[]. El sistema asigna un estado a cada posición basándose en la diferencia absoluta (abs()) entre ambos valores:

- **Acierto (Verde):** Si la diferencia es 0. En este caso, el software activa un *flag* de bloqueo (digito_bloqueado[i] = true). Esto es una característica de usabilidad importante: una vez que el usuario adivina un número, este se "congela" en la pantalla y deja de variar aunque se mueva el potenciómetro, facilitando que el jugador se centre en los dígitos restantes.
- **Aproximación (Amarillo):** Si la diferencia es exactamente 1 (por ejemplo, clave 5 y usuario 4 o 6). Indica al usuario que está muy cerca del valor correcto.
- **Fallo (Rojo):** Para cualquier diferencia mayor a 1.

Este algoritmo no solo actualiza los LEDs correspondientes, sino que determina el siguiente estado de la FSM: solo si el contador de aciertos es igual a 4, el sistema transita hacia VICTORIA.

6. Implementación Física y Montaje

6.1. Mapa de Conexiones (Pinout)

Para la materialización del sistema se ha utilizado una placa de prototipado (*protoboard*) donde se han centralizado todas las interconexiones. A continuación, se presenta la tabla de asignación de pines definitiva, la cual relaciona los periféricos físicos con los puertos del microcontrolador STM32 utilizados en el proyecto.

Periférico	Etiqueta / Función	Pin STM32	Puerto	Configuración Externa
Entradas Analógicas				
Potenciómetro 1	Selector Dígito 1	PA0	ADC1_IN0	Divisor de Tensión
Potenciómetro 2	Selector Dígito 2	PA1	ADC1_IN1	Divisor de Tensión
Potenciómetro 3	Selector Dígito 3	PA4	ADC1_IN4	Divisor de Tensión
Potenciómetro 4	Selector Dígito 4	PB0	ADC1_IN8	Divisor de Tensión
Entradas Digitales				
Pulsador 1	Acción: VALIDAR	PA10	GPIOA	Pull-Down (10kΩ)
Pulsador 2	Acción: INICIO/RESET	PC13	GPIOC	Pull-Down (10kΩ)

Salidas Visuales				
LED RGB 1	Estado Dígito 1	PC1 (R) / PC0 (G)	GPIOC	R=220Ω en serie
LED RGB 2	Estado Dígito 2	PC3 (R) / PC2 (G)	GPIOC	R=220Ω en serie
LED RGB 3	Estado Dígito 3	PC5 (R) / PC4 (G)	GPIOC	R=220Ω en serie
LED RGB 4	Estado Dígito 4	PC7 (R) / PC6 (G)	GPIOC	R=220Ω en serie
Buses y Otros				
Pantalla LCD	Bus de Datos I2C	PB9 (SDA) / PB8 (SCL)	I2C1	Módulo PCF8574
Zumbador	Señal Acústica	PB5	GPIOB	R=100Ω en serie

6.2. Topología de Alimentación

El sistema carece de fuentes de alimentación externas independientes; se aprovecha la regulación de tensión de la propia placa de desarrollo STM32.

- **Línea de Positivo (VCC):** Se ha extraído del pin de **5V** de la placa, conectándolo al carril rojo de la protoboard para alimentar tanto los potenciómetros como la pantalla LCD.
- **Referencia Común (GND):** Se ha unificado la tierra del sistema conectando el pin GND del microcontrolador al carril azul. Esta referencia es crítica para asegurar que las lecturas analógicas sean estables y libres de ruido de modo común.

6.3. Descripción de los Circuitos de Interfaz

Dado que el microcontrolador trabaja con niveles lógicos de 3.3V pero gestiona periféricos de distinta naturaleza, se han implementado los siguientes circuitos de acondicionamiento:

6.3.1. Acondicionamiento de Entradas

- **Sensores Analógicos:** Los cuatro potenciómetros están conectados en configuración de **divisor de tensión**. Los extremos fijos se alimentan entre VCC y GND, mientras que el terminal variable (wiper) entrega una tensión

proporcional al giro (0V a 3.3V aprox.) directamente a las entradas analógicas del Puerto A y B.

- **Pulsadores (Lógica Positiva):** A pesar de que el microcontrolador dispone de resistencias internas, se ha optado por implementar un circuito **Pull-Down externo** utilizando resistencias de **10kΩ**. Esto garantiza un '0' lógico robusto en reposo y protege la entrada frente a interferencias electromagnéticas cuando el botón no está presionado. Al pulsar, se cierra el circuito a VCC, enviando un '1' lógico al pin.

6.3.2. Etapas de Salida

- **Indicadores LED:** Se han empleado LEDs RGB de **Cátodo Común**. El cátodo de cada LED se conecta directamente a tierra, mientras que los ánodos correspondientes a los canales Rojo y Verde se gestionan desde el microcontrolador. Para proteger los puertos del STM32, se han intercalado resistencias limitadoras de **220Ω** en cada línea de señal, limitando la corriente a un nivel seguro (aprox. 10-15mA) sin comprometer el brillo. Cabe destacar que el canal Azul (B) se ha dejado desconectado (NC) ya que no es necesario para la lógica de colores del juego (Rojo/Verde/Amarillo).
- **Interfaz Acústica:** El zumbador se controla mediante el pin PB5. Para evitar picos de corriente inductiva que pudieran dañar el pin digital, se ha colocado una resistencia de **100Ω** en serie con el terminal positivo del componente.

7. Pruebas y Resultados Experimentales

7.1. Estrategia de Verificación

Una vez finalizado el montaje físico y la carga del firmware, se procedió a la validación del sistema siguiendo una estrategia *Bottom-Up* (de abajo hacia arriba). Primero se verificó el correcto funcionamiento eléctrico de cada periférico de forma aislada (Pruebas Unitarias) y, posteriormente, se evaluó la lógica del juego en su conjunto (Pruebas de Integración).

Las pruebas se realizaron en un entorno de laboratorio estándar, alimentando el sistema mediante el puerto USB de depuración (ST-Link) y monitorizando el comportamiento en tiempo real.

7.2. Verificación de Periféricos (Pruebas Unitarias)

Antes de iniciar el juego, se realizaron test específicos para asegurar la integridad de la capa de hardware (hardware.c):

- **Linealidad del ADC:** Se comprobó la respuesta de los cuatro potenciómetros. Se observó que el algoritmo de escalado implementado ($\text{valor} * 10 / 4100$) divide correctamente el rango de giro en 10 segmentos estables. A pesar del ruido eléctrico inherente a la protoboard, la lectura del dígito se mostró estable en la pantalla LCD, sin oscilaciones entre valores adyacentes (ej. saltos entre 4 y 5) salvo en los puntos críticos de transición.
- **Respuesta de los Pulsadores:** Se validó la eficacia de las resistencias *Pull-Down* de $10\text{k}\Omega$. Al pulsar los botones "VALIDAR" y "INICIO", las interrupciones se dispararon correctamente. El filtro de *debounce* por software incluido en las funciones *callback* demostró ser eficaz, eliminando rebotes mecánicos y evitando que una sola pulsación se interpretase como múltiples intentos.
- **Bus I2C y Display:** Se verificó la correcta inicialización de la pantalla LCD. Los comandos de limpieza (`lcd_clear`) y posicionamiento del cursor funcionaron según lo previsto, mostrando los caracteres con buen contraste y sin caracteres "basura", lo que confirma que la velocidad de 100 kHz del bus I2C es adecuada para la longitud de cable utilizada.

7.3. Validación Funcional del Juego

Para certificar el cumplimiento de los objetivos, se simularon partidas completas cubriendo los tres escenarios de finalización posibles contemplados en la Máquina de Estados:

7.3.1. Caso de Prueba 1: Secuencia de Victoria

Se introdujo deliberadamente la clave correcta tras varios intentos de aproximación.

- **Comportamiento observado:** Al validar la clave correcta, el sistema respondió instantáneamente:
 1. Los cuatro indicadores LED se iluminaron en **color verde fijo**.

2. El zumbador emitió un tono agudo y corto (tipo "Victoria" definido en el driver).
3. La pantalla LCD mostró el mensaje "GANASTE!!" seguido de la clave secreta.
4. El sistema bloqueó las entradas durante 4 segundos antes de reiniciar, tal como se programó en la FSM.

7.3.2. Caso de Prueba 2: Derrota por Agotamiento de Intentos

Se forzó el fallo del usuario introduciendo claves erróneas consecutivamente hasta gastar las 6 vidas.

- **Comportamiento observado:** El contador de vidas en el LCD decrementó correctamente tras cada validación fallida. Al llegar el contador a 0, el sistema transitó al estado ESTADO_DERROTA, mostrando el mensaje "FIN DE JUEGO" y revelando la clave que no se pudo adivinar. El zumbador emitió la secuencia de tonos graves y entrecortados programada para indicar error.

7.3.3. Caso de Prueba 3: Derrota por Tiempo Límite

Se dejó el sistema en reposo con una partida iniciada para verificar la precisión del temporizador.

- **Comportamiento observado:** El contador de tiempo en pantalla decreció de forma consistente (1 segundo real por cada decremento en pantalla). Al llegar a T:000, el sistema interrumpió la partida inmediatamente, ignorando cualquier posición de los potenciómetros y mostrando el mensaje "TIEMPO AGOTADO". Esto confirma que la interrupción del **TIM2** tiene la prioridad adecuada y es capaz de tomar el control del flujo del programa para forzar el final de la partida.

7.4. Análisis de la Retroalimentación Visual (Feedback)

Durante las pruebas de juego, se prestó especial atención a la lógica de colores de los LEDs (función Actualizar_LED_Dígito). La respuesta visual resultó intuitiva:

- Los LEDs **Rojos** indicaron claramente el error.

- La mezcla de colores para el **Amarillo** (activando pines Rojo y Verde simultáneamente) fue distingible, aunque se notó una ligera predominancia del tono verde debido a la eficiencia lumínica de los LEDs, algo corregible ajustando las resistencias, pero funcional para el propósito del prototipo.
- El bloqueo de los dígitos acertados (LED Verde fijo) funcionó correctamente, impidiendo que el usuario modificase accidentalmente una parte de la clave ya resuelta.

8. Problemas Encontrados y Soluciones Adoptadas

Durante la fase de integración del software con el montaje físico, surgieron varios desafíos técnicos que obligaron a refinar el código inicial. A continuación, se detallan los problemas más críticos y las soluciones de ingeniería implementadas para resolverlos.

8.1. Rebotes Mecánicos en las Entradas Digitales (Bouncing)

Uno de los primeros problemas detectados fue la inestabilidad en los pulsadores. Al presionar el botón de "Validar" una sola vez, el sistema a veces registraba múltiples pulsaciones consecutivas, provocando que el juego saltara estados involuntariamente. Esto se debe al fenómeno mecánico del "rebote", donde los contactos metálicos oscilan antes de estabilizarse.

- **Solución Adoptada:** Dado que no se incluyeron condensadores de filtrado en el hardware (circuito RC), se implementó un **filtro de debounce por software**. Dentro de la rutina de interrupción HAL_GPIO_EXTI_Callback, se añadió un pequeño retardo de ciclo (for loop con instrucción __NOP) seguido de una segunda lectura del pin.
 - *Lógica:* Si tras el retardo el pin sigue en estado alto, se considera una pulsación válida. Si ha bajado, se descarta como ruido espurio.

8.2. Bloqueo del Temporizador por Retardos de CPU

En las primeras versiones del código, se utilizaba la función estándar HAL_Delay() para generar las pausas visuales (por ejemplo, mostrar el mensaje de victoria durante 4 segundos). Sin embargo, se observó un error crítico: durante esos 4 segundos de espera,

el microcontrolador quedaba totalmente "congelado". Esto impedía que se atendieran otras tareas y, lo más grave, afectaba a la precisión de la cuenta atrás si se decidía implementar animaciones de fondo.

- **Solución Adoptada:** Se migró toda la lógica de espera a un modelo **no bloqueante**. En lugar de detener la CPU, se programó la Máquina de Estados para capturar el instante de inicio (HAL_GetTick) y comparar continuamente la diferencia con el tiempo actual. Esto permite que el bucle while(1) siga girando millones de veces por segundo, manteniendo el sistema vivo y reactivo incluso durante las pausas visuales.

8.3. Ruido y Escalado en la Conversión ADC

Al leer los potenciómetros, se detectó que el valor máximo del conversor (4095 en 12 bits) generaba problemas al intentar mapearlo al rango decimal (0-9). Matemáticamente, dividir 4095 entre 409.6 debería dar 9.99, pero el ruido eléctrico en la alimentación a veces provocaba desbordamientos o inestabilidad en los valores límite.

- **Solución Adoptada:** Se modificó la fórmula de normalización en el archivo hardware.c. En lugar de dividir por el valor teórico exacto, se aplicó un divisor ligeramente superior (4100).
 - *Código:* $\text{valores_0_9[i]} = (\text{val} * 10) / 4100;$
 - Este margen de seguridad garantiza que, incluso con el potenciómetro al máximo y con ruido positivo, el resultado nunca supere el valor 9, asegurando una lectura sólida y confiable.

8.4. Gestión de Colores Mezclados (Amarillo)

El control de los LEDs tricolores presentó un desafío en la gestión del color amarillo, necesario para indicar "aproximación". A diferencia de los colores puros (Rojo o Verde) que dependen de un solo pin, el amarillo requiere la activación simultánea de ambos canales.

- **Solución Adoptada:** Se creó una función de abstracción (Actualizar_LED_Dígito) que encapsula esta complejidad. Cuando la lógica solicita el color AMARILLO, el driver se encarga de poner a '1' tanto el pin del canal Rojo como el del Verde. Aunque la intensidad del verde suele predominar en estos componentes, la mezcla aditiva resultó suficiente para distinguir el estado de advertencia frente al acierto (solo verde) o el error (solo rojo)

9. Manual de Usuario

9.1. Puesta en Marcha

El sistema no dispone de un interruptor de encendido dedicado; la activación es automática al suministrar energía.

1. **Conexión:** Conecte el cable USB de la placa STM32 a una fuente de alimentación de 5V (puerto USB de un ordenador o cargador estándar).
2. **Inicialización:** Al recibir corriente, el sistema realizará un chequeo rápido de periféricos (los LEDs parpadearán brevemente si se ha configurado así en el hardware) y la pantalla LCD mostrará el mensaje de bienvenida: "PROYECTO CERROJO".
3. **Estado de Espera:** Tras un segundo, la pantalla cambiará a "PULSA INICIO". El sistema permanecerá en este estado de bajo consumo hasta que el usuario interactúe.

9.2. Interfaz de Juego (Visualización)

Durante la partida, la pantalla LCD actúa como el panel de instrumentos principal, proporcionando la siguiente información en tiempo real:

- **Fila Superior:**
 - T:xxx: Indica el **Tiempo Restante** en segundos (comienza en 120).
 - Vid:x: Indica el número de **Vidas o Intentos** disponibles (comienza en 6).
- **Fila Inferior:** Muestra los cuatro números seleccionados actualmente por los potenciómetros (ej. 3 5 0 9).

Código de Colores (LEDs)

Cada uno de los cuatro dígitos de la clave tiene asociado un indicador LED situado sobre el potenciómetro correspondiente. Tras pulsar el botón de validación, estos indicadores informan de la precisión del intento:

- **LED Verde:** El número es **CORRECTO** y está en la posición adecuada.

- *Nota:* Cuando un dígito se pone en verde, el sistema lo **bloquea**. Aunque gire el potenciómetro, ese número no cambiará, facilitando que se centre en los restantes.
- **● LED Amarillo:** El número es **APROXIMADO**. El valor introducido difiere en solo una unidad respecto al secreto (ej. si la clave es 5, el 4 y el 6 encenderán el amarillo).
- **● LED Rojo:** El número es **INCORRECTO** y no está cerca del valor real.

9.3. Controles y Operación

El sistema se controla mediante una interfaz híbrida analógica-digital:

1. **Selección de Clave (Potenciómetros):**

- Gire cada una de las 4 perillas para seleccionar un dígito del 0 al 9.
- El valor cambia instantáneamente en la pantalla LCD. Se recomienda girar suavemente hasta ver el número deseado.

2. **Botón "VALIDAR" (Pulsador Derecho - PA10):**

- Púlselo una vez cuando tenga la combinación deseada en pantalla.
- El sistema procesará la jugada, actualizará los LEDs y restará una vida.

3. **Botón "INICIO / RESET" (Pulsador Izquierdo - PC13):**

- En modo de espera: Inicia una nueva partida, generando una nueva clave aleatoria.
- Durante la partida: Fuerza el reinicio del sistema (Rendición), volviendo a la pantalla de título.

9.4. Finalización de la Partida

El juego puede terminar de dos formas:

- **Victoria:** Si consigue poner los 4 LEDs en verde. Escuchará un tono de victoria y la pantalla mostrará "GANASTE!!" junto con la clave descifrada.
- **Derrota:** Si el contador T llega a 0 o las Vid llegan a 0. Se emitirá un tono grave de error y la pantalla mostrará "FIN DE JUEGO", revelando cuál era la clave secreta que no consiguió adivinar.

En ambos casos, tras esperar 4 segundos mostrando el resultado, el sistema volverá automáticamente a la pantalla de "PULSA INICIO".

10. Conclusiones y Líneas Futuras

10.1. Conclusiones

La realización del "Proyecto Cerrojo" ha permitido integrar y validar de forma práctica los conceptos teóricos fundamentales de la asignatura de Sistemas Electrónicos Digitales. Tras finalizar el diseño, montaje y depuración del prototipo, se extraen las siguientes conclusiones técnicas:

- **Viabilidad de la Arquitectura No Bloqueante:** Se ha constatado que el uso de retardos tradicionales (HAL_Delay) es incompatible con sistemas que requieren interactividad en tiempo real. La implementación de una gestión de tiempos basada en marcas de sistema (HAL_GetTick) y máquinas de estados ha resultado ser la solución óptima, permitiendo que la interfaz de usuario, la lectura de sensores y la cuenta atrás del temporizador convivan sin interrupciones ni bloqueos mutuos.
- **Importancia de la Abstracción de Hardware:** La decisión de separar el código en capas (*Drivers vs Lógica*) ha simplificado enormemente la fase de pruebas. Poder invocar funciones como Actualizar_LED_Dígito() desde la lógica principal, sin tener que manipular los registros del puerto GPIO en cada línea, ha facilitado la legibilidad del código y la detección de errores.
- **Retos del Mundo Analógico:** La interfaz basada en potenciómetros, aunque funcional, ha demostrado las dificultades de digitalizar señales continuas en un entorno ruidoso (protoboard). La necesidad de ajustar el factor de escala a 4100 en lugar del valor teórico para evitar desbordamientos evidencia que, en ingeniería, el ajuste empírico es tan importante como el cálculo teórico.

En definitiva, se ha logrado un sistema funcional, robusto y jugable que cumple con todos los requisitos iniciales, demostrando la capacidad del microcontrolador STM32 para gestionar tareas concurrentes con soltura.

10.2. Líneas Futuras

Aunque el sistema actual es plenamente funcional, el diseño modular del software permitiría implementar una serie de mejoras y ampliaciones en futuras revisiones del proyecto:

1. **Persistencia de Datos (High Scores):** Actualmente, al reiniciar la placa, se pierden todos los datos. Una mejora significativa sería utilizar la memoria Flash interna del STM32 (o una EEPROM externa I2C) para guardar una tabla de puntuaciones máximas o estadísticas de aciertos, incentivando la competitividad.
2. **Menú de Configuración:** Se podría aprovechar la pantalla LCD para añadir un menú de inicio que permita al usuario modificar los parámetros del juego definidos en `logica_juego.c` (tiempo límite de 120s o número de vidas), adaptando la dificultad sin necesidad de reprogramar el firmware.
3. **Seguridad (Watchdog Independiente):** Para elevar el nivel de robustez a un estándar industrial, sería recomendable activar el *Independent Watchdog* (IWDG). Esto reiniciaría el sistema automáticamente en el caso hipotético de que el programa quedase atrapado en un bucle infinito por un fallo eléctrico o de software.
4. **Mejora de la Interfaz de Entrada:** Sustituir los potenciómetros analógicos por **Encoders Rotativos**. Al ser dispositivos digitales, eliminarían el ruido de la señal y permitirían una selección de números más precisa (con "clicks" táctiles), mejorando la experiencia de usuario y simplificando el código de filtrado del ADC.
5. **Generación de Aleatoriedad Real:** Actualmente la semilla aleatoria depende del tiempo de arranque. Se podría mejorar la entropía utilizando el ruido térmico de un canal ADC analógico desconectado (*Floating Pin*) para inicializar la semilla `rand()`, haciendo la clave secreta verdaderamente impredecible.

11. Anexos

En este apartado se adjunta la totalidad del código fuente desarrollado para el proyecto. El software se estructura en módulos funcionales para facilitar la comprensión y el mantenimiento del sistema.

11.1. Archivos de Cabecera y Configuración (Headers)

Archivo: i2c-lcd.h **Descripción:** Definiciones para el controlador de la pantalla LCD.

```
#ifndef INC_I2C_LCD_H_
#define INC_I2C_LCD_H_

#include "stm32f4xx_hal.h"

void lcd_init (void);      // Inicializar pantalla
void lcd_send_cmd (char cmd); // Enviar comando interno
void lcd_send_data (char data); // Enviar letra/número
void lcd_send_string (char *str); // Enviar frase completa
void lcd_put_cur(int row, int col); // Mover cursor
void lcd_clear (void); // Borrar pantalla

#endif
```

Archivo: hardware.h **Descripción:** Prototipos de funciones de la capa de abstracción de hardware y definición de tipos enumerados para los colores.

```
#ifndef INC_HARDWARE_H_
#define INC_HARDWARE_H_

#include "stm32f4xx_hal.h"

// Definición de Colores para facilitar la lectura
typedef enum {
    APAGADO,
    ROJO,
    VERDE,
    AMARILLO
} ColorLED;

// Prototipos de funciones
void Hardware_Init(ADC_HandleTypeDef* hadc, I2C_HandleTypeDef* hi2c);
void Leer_Potenciómetros(int* valores_0_9);
```

```

void Actualizar_LED_Dígito(int dígito_index, ColorLED color);
void Apagar_Todos_LEDs(void);
void Zumbador_Tono(int duración_ms, int tipo);

#endif

```

Archivo: logica_juego.h **Descripción:** Definición de la máquina de estados y variables externas compartidas.

```

#ifndef INC_LOGICA_JUEGO_H_
#define INC_LOGICA_JUEGO_H_

#include "main.h"
#include <stdbool.h>

typedef enum {
    ESTADO_ESPERA,
    ESTADO_GENERAR,
    ESTADO_JUGANDO,
    ESTADO_VERIFICAR,
    ESTADO_VICTORIA,
    ESTADO_DERROTA
} EstadoJuego;

extern EstadoJuego estado_actual;

// Funciones
void Logica_Inicializar(void);
void Logica_Ejecutar_Ciclo(void);
void Logica_Timer_Callback(void);

#endif /* INC_LOGICA_JUEGO_H_ */

```

11.2. Código Fuente Principal (Main y Lógica)

Archivo: main.c **Descripción:** Configuración del sistema, inicialización de periféricos y bucle principal.

```

/* USER CODE BEGIN Header */
/** 
 * ***** 
 * @file          : main.c
 * @brief         : Main program body
 * ***** 
 */

```

```

* @attention
*
* Copyright (c) 2026 STMicroelectronics.
* All rights reserved.
*
* This software is licensed under terms that can be found in the
LICENSE file
* in the root directory of this software component.
* If no LICENSE file comes with this software, it is provided AS-IS.
*
*****
*/
/* USER CODE END Header */
/* Includes -----
-----*/
#include "main.h"

/* Private includes -----
-----*/
/* USER CODE BEGIN Includes */
#include "i2c-lcd.h"
#include <stdio.h> // Para poder usar sprintf
/* USER CODE BEGIN Includes */
#include "hardware.h"
#include "logica_juego.h"
/* USER CODE END Includes */

/* Private typedef -----
-----*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----
-----*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -----
-----*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----
-----*/
ADC_HandleTypeDef hadc1;

I2C_HandleTypeDef hi2c1;

TIM_HandleTypeDef htim2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

```

```

/* Private function prototypes -----
-----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM2_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code -----
-----*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----
-----*/
    /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_I2C1_Init();
    MX_ADC1_Init();
    MX_TIM2_Init();
    /* USER CODE BEGIN 2 */
    Hardware_Init(&hadcl, &hi2c1);
    Logica_Inicializar();

    // ENCENDER LA INTERRUPCIÓN DEL TEMPORIZADOR
    HAL_TIM_Base_Start_IT(&htim2);

```

```

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    Logica_Ejecutar_Ciclo();
    HAL_Delay(10); // Estabilidad del sistema
}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Configure the main internal regulator output voltage
    */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /** Initializes the RCC Oscillators according to the specified
    parameters
     * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 4;
    RCC_OscInitStruct.PLL.PLLN = 192;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
    RCC_OscInitStruct.PLL.PLLQ = 8;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /** Initializes the CPU, AHB and APB buses clocks
    */
    RCC_ClkInitStruct.ClockType =
    RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
    |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
}

```

```

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
    {
        Error_Handler();
    }
}

/**
 * @brief ADC1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_ADC1_Init(void)
{
    /* USER CODE BEGIN ADC1_Init_0 */

    /* USER CODE END ADC1_Init_0 */

    ADC_ChannelConfTypeDef sConfig = {0};

    /* USER CODE BEGIN ADC1_Init_1 */

    /* USER CODE END ADC1_Init_1 */

    /** Configure the global features of the ADC (Clock, Resolution,
    Data Alignment and number of conversion)
    */
    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B;
    hadc1.Init.ScanConvMode = ENABLE;
    hadc1.Init.ContinuousConvMode = DISABLE;
    hadc1.Init.DiscontinuousConvMode = ENABLE;
    hadc1.Init.NbrOfDiscConversion = 1;
    hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
    hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.NbrOfConversion = 4;
    hadc1.Init.DMAContinuousRequests = DISABLE;
    hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
    if (HAL_ADC_Init(&hadc1) != HAL_OK)
    {
        Error_Handler();
    }

    /** Configure for the selected ADC regular channel its corresponding
    rank in the sequencer and its sample time.
    */
    sConfig.Channel = ADC_CHANNEL_0;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_144CYCLES;
    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
    {
        Error_Handler();
    }
}

```

```

    /** Configure for the selected ADC regular channel its corresponding
     * rank in the sequencer and its sample time.
     */
    sConfig.Channel = ADC_CHANNEL_1;
    sConfig.Rank = 2;
    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
    {
        Error_Handler();
    }

    /** Configure for the selected ADC regular channel its corresponding
     * rank in the sequencer and its sample time.
     */
    sConfig.Channel = ADC_CHANNEL_4;
    sConfig.Rank = 3;
    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
    {
        Error_Handler();
    }

    /** Configure for the selected ADC regular channel its corresponding
     * rank in the sequencer and its sample time.
     */
    sConfig.Channel = ADC_CHANNEL_8;
    sConfig.Rank = 4;
    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
    {
        Error_Handler();
    }
/* USER CODE BEGIN ADC1_Init 2 */

/* USER CODE END ADC1_Init 2 */

}

/***
 * @brief I2C1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_I2C1_Init(void)
{
    /* USER CODE BEGIN I2C1_Init 0 */

    /* USER CODE END I2C1_Init 0 */

    /* USER CODE BEGIN I2C1_Init 1 */

    /* USER CODE END I2C1_Init 1 */
    hi2c1.Instance = I2C1;
    hi2c1.Init.ClockSpeed = 100000;
    hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
    hi2c1.Init.OwnAddress1 = 0;
    hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
    hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
    hi2c1.Init.OwnAddress2 = 0;
}

```

```

hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
if (HAL_I2C_Init(&hi2c1) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN I2C1_Init 2 */

/* USER CODE END I2C1_Init 2 */

}

/**
 * @brief TIM2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM2_Init(void)
{

/* USER CODE BEGIN TIM2_Init 0 */

/* USER CODE END TIM2_Init 0 */

TIM_ClockConfigTypeDef sClockSourceConfig = {0};
TIM_MasterConfigTypeDef sMasterConfig = {0};

/* USER CODE BEGIN TIM2_Init 1 */

/* USER CODE END TIM2_Init 1 */
htim2.Instance = TIM2;
htim2.Init.Prescaler = 16000;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 3000;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */

/* USER CODE END TIM2_Init 2 */

```

```

}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* USER CODE BEGIN MX_GPIO_Init_1 */

    /* USER CODE END MX_GPIO_Init_1 */

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
    |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);

    /*Configure GPIO pin : PC13 */
    GPIO_InitStruct.Pin = GPIO_PIN_13;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStruct.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    /*Configure GPIO pins : PC0 PC1 PC2 PC3
                           PC4 PC5 PC6 PC7 */
    GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
    |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    /*Configure GPIO pin : PA10 */
    GPIO_InitStruct.Pin = GPIO_PIN_10;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStruct.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /*Configure GPIO pin : PB5 */
    GPIO_InitStruct.Pin = GPIO_PIN_5;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}

```

```

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */

/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

// --- INTERRUPCIÓN DEL TEMPORIZADOR (Cada 1 segundo) ---
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Instance == TIM2) {
        Logica_Timer_Callback(); // Llama a la lógica para restar
        tiempo
    }
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    // RETARDO ANTI-RUIDO (Debounce Hardware simulado)
    for(int i=0; i<2000; i++) { __NOP(); }

    // SEGURIDAD PULL-DOWN:
    // Solo aceptamos la interrupción si el pin sigue siendo ALTO
    (3.3V)
    // Si fuera ruido eléctrico, ya habría bajado a 0.

    if (GPIO_Pin == GPIO_PIN_10) { // Botón Validar (PA10)
        if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_10) == GPIO_PIN_SET) {
            Callback_Boton_Validar();
        }
    }
    else if (GPIO_Pin == GPIO_PIN_13) { // Botón Inicio (PC13)
        if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_SET) {
            Callback_Boton_Inicio();
        }
    }
}
/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error
    return state */
    __disable_irq();
    while (1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
}

```

```

}

#ifndef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line
number
 * where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and
line number,
       ex: printf("Wrong parameters value: file %s on line %d\r\n",
file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

```

Archivo: logica_juego.c **Descripción:** Implementación de la lógica de negocio, FSM y reglas del juego.

```

#include "logica_juego.h"
#include "hardware.h"
#include "i2c-lcd.h"
#include <stdlib.h>
#include <stdio.h>
#include "main.h"

// --- VARIABLE EXTERNA DEL TIMER (Para arreglar el error de
compilación) ---
extern TIM_HandleTypeDef htim2;

// Variables Globales del Juego
EstadoJuego estado_actual = ESTADO_ESPERA;
int clave_secreta[4];
int intento_actual[4];
bool digito_bloqueado[4];

// Variables de Juego
int intentos_restantes = 6;
int tiempo_restante = 120; // 120 segundos

// VARIABLES PARA LOGICA NO BLOQUEANTE (El sustituto de HAL_Delay)
uint32_t t_inicio_estado = 0; // Guarda la hora a la que entramos al
estado
bool es_inicio_estado = true; // Indica si acabamos de entrar (para
ejecutar cosas 1 sola vez)
uint32_t ultimo_refresco_lcd = 0;

```

```

// Variables Botones
volatile uint32_t t_btn_val = 0;
volatile uint32_t t_btn_ini = 0;
volatile bool flag_validar = false;
volatile bool flag_inicio = false;

// Función para cambiar de estado de forma segura y marcar el tiempo
void Cambiar_Estado(EstadoJuego nuevo_estado) {
    estado_actual = nuevo_estado;
    es_inicio_estado = true; // Marcamos que acabamos de llegar
    t_inicio_estado = HAL_GetTick(); // Guardamos la hora de llegada
}

// --- TIMER HARDWARE (Cuenta atrás 120s) ---
void Logica_Timer_Callback() {
    if (estado_actual == ESTADO_JUGANDO) {
        if (tiempo_restante > 0) {
            tiempo_restante--;
        } else {
            // Cambio directo si se acaba el tiempo
            Cambiar_Estado(ESTADO_DERROTA);
        }
    }
}

void Logica_Inicializar() {
    Apagar_Todos_LEDs();
    lcd_clear();
    lcd_put_cur(0,0);
    lcd_send_string("PROYECTO CERROJO");
    // Aquí sí podemos dejar un delay pequeño al arrancar,
    // pero si es estricto, mejor quítalo. Dejamos 1s.
    HAL_Delay(1000);

    tiempo_restante = 120;
    lcd_clear();
    lcd_put_cur(0,0);
    lcd_send_string("PULSA INICIO");

    Cambiar_Estado(ESTADO_ESPERA);
}

void Logica_Ejecutar_Ciclo() {
    char buffer[20];
    int lecturas_raw[4];
    uint32_t t_actual = HAL_GetTick(); // Hora actual del sistema

    switch (estado_actual) {

        // --- 1. ESPERA ---
        case ESTADO_ESPERA:
            if (es_inicio_estado) {
                // Cosas que se hacen solo al entrar en ESPERA
                es_inicio_estado = false;
            }
    }
}

```

```

    if (flag_inicio) {
        flag_inicio = false;
        Cambiar_Estado(ESTADO_GENERAR);
    }
    break;

// --- 2. GENERAR ---
case ESTADO_GENERAR:
    if (es_inicio_estado) {
        srand(HAL_GetTick());
        for(int i=0; i<4; i++) {
            clave_secreta[i] = rand() % 10;
            digito_bloqueado[i] = false;
        }
        intentos_restantes = 6;
        tiempo_restante = 120;

        lcd_clear();
        lcd_send_string("NUEVA PARTIDA");
        es_inicio_estado = false;
    }

    // SUSTITUTO DEL HAL_DELAY(1000):
    // Si han pasado 1000ms desde que entramos, cambiamos.
    if (t_actual - t_inicio_estado >= 1000) {
        lcd_clear();
        Cambiar_Estado(ESTADO_JUGANDO);
    }
    break;

// --- 3. JUGANDO ---
case ESTADO_JUGANDO:
    // Aquí no había delays, así que se queda casi igual
    if (es_inicio_estado) es_inicio_estado = false;

    // Leer Hardware
    Leer_Potenciómetros(lecturas_raw);
    for(int i=0; i<4; i++) {
        if(!digito_bloqueado[i]) intento_actual[i] =
lecturas_raw[i];
    }

    // Refrescar LCD
    if (t_actual - ultimo_refresco_lcd > 200) {
        sprintf(buffer, "T:%03d Vid:%d", tiempo_restante,
intentos_restantes);
        lcd_put_cur(0, 0); lcd_send_string(buffer);

        sprintf(buffer, " %d %d %d %d ",
intento_actual[0], intento_actual[1],
intento_actual[2], intento_actual[3]);
        lcd_put_cur(1, 0); lcd_send_string(buffer);

        ultimo_refresco_lcd = t_actual;
    }

    // Verificar tiempo agotado (por si el Timer se adelantó)
}

```

```

        if (tiempo_restante <= 0) Cambiar_Estado(ESTADO_DERROTA);

        // Botones
        if (flag_validar) { flag_validar = false;
Cambiar_Estado(ESTADO_VERIFICAR); }
            if (flag_inicio) { flag_inicio = false;
Cambiar_Estado(ESTADO_GENERAR); }
                break;

        // --- 4. VERIFICAR ---
case ESTADO_VERIFICAR:
    if (es_inicio_estado) {
        lcd_clear();
        lcd_send_string("VERIFICANDO...");

        intentos_restantes--;

        // Lógica de colores
        for(int i=0; i<4; i++) {
            int dif = abs(intento_actual[i] -
clave_secreta[i]);
            if (dif == 0) { Actualizar_LED_Dígito(i, VERDE);
dígito_bloqueado[i] = true; }
            else if (dif == 1) { Actualizar_LED_Dígito(i,
AMARILLO); }
            else { Actualizar_LED_Dígito(i, ROJO); }
        }
        es_inicio_estado = false;
    }

    // SUSTITUTO DE HAL_DELAY(1500) para ver los LEDs:
    if (t_actual - t_inicio_estado >= 1500) {
        // Decidir destino
        int aciertos = 0;
        for(int i=0; i<4; i++) if(intento_actual[i] ==
clave_secreta[i]) aciertos++;

        if (aciertos == 4) {
            Cambiar_Estado(ESTADO_VICTORIA);
        } else if (intentos_restantes <= 0 || tiempo_restante
<= 0) {
            Cambiar_Estado(ESTADO_DERROTA);
        } else {
            lcd_clear();
            Zumbador_Tono(100, 0); // Pitido corto
            Cambiar_Estado(ESTADO_JUGANDO);
        }
    }
    break;

        // --- 5. VICTORIA ---
case ESTADO_VICTORIA:
    if (es_inicio_estado) {
        lcd_clear();
        lcd_send_string("GANASTE!!!");
        lcd_put_cur(1,0);

```

```

        sprintf(buffer, "Clave: %d%d%d%d", clave_secreta[0],
clave_secreta[1], clave_secreta[2], clave_secreta[3]);
        lcd_send_string(buffer);
        Zumbador_Tono(100, 1);
        es_inicio_estado = false;
    }

    // SUSTITUTO DE HAL_DELAY(4000): Esperamos 4s y
reiniciamos
    if (t_actual - t_inicio_estado >= 4000) {
        Logica_Inicializar(); // Reinicia y va a ESPERA
    }
    break;

    // --- 6. DERROTA ---
case ESTADO_DERROTA:
    if (es_inicio_estado) {
        lcd_clear();
        if (tiempo_restante <= 0) lcd_send_string("TIEMPO
AGOTADO");
        else lcd_send_string("FIN DE JUEGO");

        lcd_put_cur(1,0);
        sprintf(buffer, "Era: %d%d%d%d", clave_secreta[0],
clave_secreta[1], clave_secreta[2], clave_secreta[3]);
        lcd_send_string(buffer);
        Zumbador_Tono(500, 0);
        es_inicio_estado = false;
    }

    // SUSTITUTO DE HAL_DELAY(4000)
    if (t_actual - t_inicio_estado >= 4000) {
        Logica_Inicializar();
    }
    break;
}

// Callbacks Botones
void Callback_Boton_Validar() {
    if (HAL_GetTick() - t_btn_val > 500) { flag_validar = true;
t_btn_val = HAL_GetTick(); }
}
void Callback_Boton_Inicio() {
    if (HAL_GetTick() - t_btn_ini > 500) { flag_inicio = true;
t_btn_ini = HAL_GetTick(); }
}

```

11.3. Drivers de Periféricos

Archivo: hardware.c **Descripción:** Funciones de abstracción para controlar LEDs, Zumbador y lectura de Potenciómetros.

```

#include "hardware.h"
#include <math.h> // Para abs()

extern ADC_HandleTypeDef hadc1;

// Arrays de configuración de pines (Verifica que coinciden con tus
conexiones)
GPIO_TypeDef* PUERTOS_LED[4][2] = {
    {GPIOC, GPIOC}, // Dígito 1 (Verde, Rojo)
    {GPIOC, GPIOC}, // Dígito 2
    {GPIOC, GPIOC}, // Dígito 3
    {GPIOC, GPIOC} // Dígito 4
};

uint16_t PINES_LED[4][2] = {
    {GPIO_PIN_0, GPIO_PIN_1}, // Dígito 1: PC0 (Verde), PC1 (Rojo)
    {GPIO_PIN_2, GPIO_PIN_3}, // Dígito 2: PC2, PC3
    {GPIO_PIN_4, GPIO_PIN_5}, // Dígito 3: PC4, PC5
    {GPIO_PIN_6, GPIO_PIN_7} // Dígito 4: PC6, PC7
};

// --- FUNCIÓN ZUMBADOR (UNIVERSAL) ---
void Zumbador_Tono(int duracion_ms, int tipo) {
    // TIPO 1: VICTORIA (Agudo) | TIPO 0: ERROR/DERROTA (Grave/Roto)
    if (tipo == 1) {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        HAL_Delay(duracion_ms);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
    }
    else {
        for(int i=0; i<3; i++) {
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
            HAL_Delay(duracion_ms / 3);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
            HAL_Delay(50);
        }
    }
}

// --- FUNCIÓN POTENCIÓMETROS (MODO DISCONTINUO) ---
void Leer_Potenciómetros(int* valores_0_9) {
    for(int i=0; i<4; i++) {
        HAL_ADC_Start(&hadc1); // Dispara una lectura
        if (HAL_ADC_PollForConversion(&hadc1, 50) == HAL_OK) {
            uint32_t val = HAL_ADC_GetValue(&hadc1);
            valores_0_9[i] = (val * 10) / 4100; // Escala 0-9
        } else {
            valores_0_9[i] = 0; // Error
        }
    }
}

// --- FUNCIÓN LEDS (CON AMARILLO) ---
void Actualizar_LED_Dígito(int index, ColorLED color) {
    // 1. Apagar todo

```

```

        HAL_GPIO_WritePin(PUERTOS_LED[index][0], PINES_LED[index][0],
GPIO_PIN_RESET);
        HAL_GPIO_WritePin(PUERTOS_LED[index][1], PINES_LED[index][1],
GPIO_PIN_RESET);

    // 2. Encender según color
    if (color == VERDE) {
        HAL_GPIO_WritePin(PUERTOS_LED[index][0], PINES_LED[index][0],
GPIO_PIN_SET);
    }
    else if (color == ROJO) {
        HAL_GPIO_WritePin(PUERTOS_LED[index][1], PINES_LED[index][1],
GPIO_PIN_SET);
    }
    else if (color == AMARILLO) {
        // Mezcla: Rojo + Verde
        HAL_GPIO_WritePin(PUERTOS_LED[index][0], PINES_LED[index][0],
GPIO_PIN_SET);
        HAL_GPIO_WritePin(PUERTOS_LED[index][1], PINES_LED[index][1],
GPIO_PIN_SET);
    }
}

void Apagar_Todos_LEDs() {
    for(int i=0; i<4; i++) {
        Actualizar_LED_Dígito(i, 99); // Color invalido apaga todo
    }
}

void Hardware_Init(ADC_HandleTypeDef* adc, I2C_HandleTypeDef* i2c) {
    lcd_init();
    Apagar_Todos_LEDs();
}

```

Archivo: i2c-lcd.c **Descripción:** Driver de comunicación I2C para pantalla LCD 16x2.

```

#include "i2c-lcd.h"
extern I2C_HandleTypeDef hi2c1; // Debe coincidir con tu i2c en main

#define SLAVE_ADDRESS_LCD 0x4E // Prueba 0x27 si no funciona

void lcd_send_cmd (char cmd)
{
    char data_u, data_l;
    uint8_t data_t[4];
    data_u = (cmd&0xf0);
    data_l = ((cmd<<4)&0xf0);
    data_t[0] = data_u|0x0C; //en=1, rs=0
    data_t[1] = data_u|0x08; //en=0, rs=0
    data_t[2] = data_l|0x0C; //en=1, rs=0
    data_t[3] = data_l|0x08; //en=0, rs=0
}

```

```

    HAL_I2C_Master_Transmit (&hi2c1, SLAVE_ADDRESS_LCD, (uint8_t *)
data_t, 4, 100);
}

void lcd_send_data (char data)
{
    char data_u, data_l;
    uint8_t data_t[4];
    data_u = (data&0xf0);
    data_l = ((data<<4)&0xf0);
    data_t[0] = data_u|0x0D; //en=1, rs=1
    data_t[1] = data_u|0x09; //en=0, rs=1
    data_t[2] = data_l|0x0D; //en=1, rs=1
    data_t[3] = data_l|0x09; //en=0, rs=1
    HAL_I2C_Master_Transmit (&hi2c1, SLAVE_ADDRESS_LCD, (uint8_t *)
data_t, 4, 100);
}

void lcd_init (void)
{
    lcd_send_cmd (0x33);
    lcd_send_cmd (0x32);
    HAL_Delay(50);
    lcd_send_cmd (0x28);
    HAL_Delay(50);
    lcd_send_cmd (0x01);
    HAL_Delay(50);
    lcd_send_cmd (0x06);
    HAL_Delay(50);
    lcd_send_cmd (0x0C);
    HAL_Delay(50);
}

void lcd_send_string (char *str)
{
    while (*str) lcd_send_data (*str++);
}

void lcd_put_cur(int row, int col)
{
    switch (row)
    {
        case 0: col |= 0x80; break;
        case 1: col |= 0xC0; break;
    }
    lcd_send_cmd (col);
}

void lcd_clear (void)
{
    lcd_send_cmd (0x01);
    HAL_Delay(2);
}

```