Videojuego portátil inspirado en consolas retro

Documento de diseño de software (SDD)

Autor:
Lic. Jezabel Danon (jezabel.danon@gmail.com)

09/07/2025
Versión A

Basado en el estándar IEEE 1016-2017.

Historial de cambios

Versión	Fecha	Descripción	Autor	Revisores
A	09/07/2025	Creación del documento	Lic. Jezabel Danon	

Índice

1. Introducción

1.1. Propósito

1. Este documento describe el diseño del software para el sistema embebido *Videojuego portátil inspirado en consolas retro*.

2. Está dirigido a los desarrolladores que se ocupen del análisis, diseño e implementación del software, así como también a quienes desarrollen el testing, validaciones y/o verificaciones del mismo.

1.2. Ámbito

1. El sistema está compuesto por un dispositivo embebido portátil que integra entradas físicas (botones, joystick, acelerómetro), salidas audiovisuales (pantalla, parlante, vibración) y una unidad de procesamiento encargada de coordinar la ejecución de un demo de juego simple.

2. El presente diseño de software abarca los módulos encargados de:

   - Control de entrada/salida digital y analógica.
   - Gestión del flujo del juego.
   - Renderizado gráfico en pantalla.
   - Generación de audio y control de vibración.
   - Manejo de estados internos y lógica de juego.

3. Este documento se enfoca exclusivamente en la descripción estructural y funcional del software embebido. No se incluyen aspectos de diseño físico, elección final de componentes electrónicos ni pruebas de verificación, los cuales son abordados en documentos específicos (como las especificaciones de hardware o el plan de validación).

4. El diseño cubre las capas de abstracción de hardware (drivers), lógica de control del juego, gestión de recursos gráficos y sonoros, así como el manejo de entradas y salidas del sistema.

5. Quedan fuera de este documento los detalles de bajo nivel sobre protocolos específicos de comunicación (como I^2C o SPI), que se encuentran documentados en los manuales de componentes o en la especificación de hardware.

6. Tampoco se incluyen aspectos de programación de la interfaz de usuario en cuanto a estilo o experiencia visual final, los cuales podrán ser definidos en una etapa posterior del desarrollo.

1.3. Definiciones y acrónimos

1. IEEE: Instituto de Ingenieros Eléctricos y Electrónicos.

2. ERS: Especificación de Requisitos de Software.

3. RTOS: Sistema Operativo de Tiempo Real (Real Time Operating System).

4. CMSIS: Arm's Common Microcontroller Software Interface Standard. Se compone por un conjunto de APIS, componentes de software, herramientas y flujos de trabajo provisto por el desarrollador de la arquitectura del microcontrolador a utilizar.

5. HAL: Hardware Abstraction Layer. Conjunto de bibliotecas de software provistas por STMicroelectronics para simplificar la interacción con el hardware de microcontroladores STM32.

6. UI: Interfaz de usuario (User Interface).

7. UART: Universal Asynchronous Receiver/Transmitter.
8. I2C: Inter-Integrated Circuit.
9. SPI: Serial Peripheral Interface.
10. ADC: Analogic to Digital Converter.
11. PWM: Pulse Width Modulation.
12. GPIO: General-purpose Input/Output.
13. ST-Link: programador y depurador integrado en placas STM32.
14. N/A: No Aplica.

1.4. Referencias

1. Estándar IEEE 1016-2017.
2. Plan de proyecto del trabajo práctico final para la *Carrera de Especialización en Sistemas Embebidos* (RETRO_GAME-PP-v5).
3. Especificaciones de requisitos de hardware: RETRO_GAME-RH-vA.
4. Especificaciones de requisitos de software: RETRO_GAME-RS-vA.

2. Visión general de la arquitectura

2.1. Perspectiva y motivación

1. Esta sección describe la estructura general del sistema *Videojuego portátil inspirado en consolas retro* desde una perspectiva arquitectónica.
2. Se presenta una vista de alto nivel que permite comprender los principales módulos de software, su organización y las interacciones entre ellos.
3. La arquitectura fue diseñada con el objetivo de facilitar el desacoplamiento entre módulos funcionales y la posibilidad de incorporar nuevas funcionalidades con un impacto mínimo en el resto del sistema.

2.2. Contexto del sistema

1. El software embebido diseñado es autónomo, no interactúa con sistemas externos ni depende de servicios de red.
2. La única interfaz externa habilitada será mediante UART vía ST-Link, utilizada para depuración durante el desarrollo.

2.3. Vista en capas

1. El sistema está estructurado en tres secciones o capas principales, siguiendo un enfoque jerárquico:
   - **Capa de Drivers:** contiene los controladores de hardware y sus rutinas de inicialización. Se encarga del acceso a periféricos como pantalla, audio, EEPROM, entradas físicas, temporizadores y comunicación serie.

- **Capa de lógica del sistema:** incluye los módulos que gestionan el estado global del sistema, la persistencia, los menús de usuario, la carga de recursos y la gestión de entradas. Esta capa implementa las FSMs de alto nivel del sistema y coordina el flujo entre modos.
- **Capa de lógica del juego:** implementa la lógica específica de gameplay, el renderizado de escenas y la interpretación de las entradas como acciones del juego.

2. Esta separación facilita la mantenibilidad y escalabilidad del sistema. La Figura 1 ilustra esta organización.

3. Cabe destacar que en la Figura 1 también se incluyen capas provistas por la plataforma, como CMSIS, HAL y RTOS. Estas capas no forman parte del diseño detallado presentado en este documento, pero se incluyen para brindar una visión completa del entorno sobre el cual se construye el software.

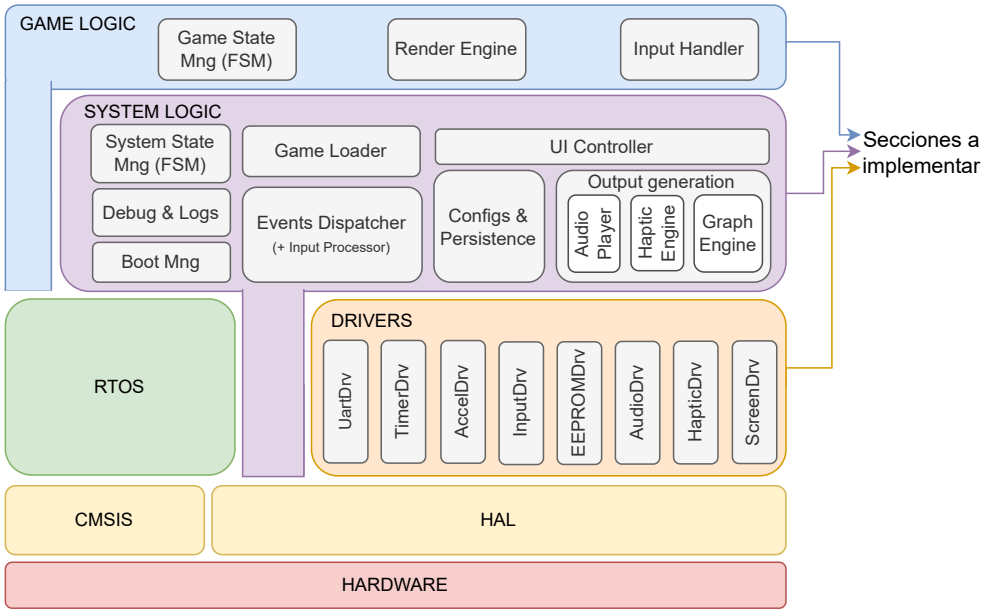

Figura 1: Diagrama de capas del sistema.

2.4. Estilos y patrones arquitectónicos

1. **Estilo por capas:** el sistema seguirá una arquitectura en capas, separando claramente la interacción con el hardware, la lógica de sistema y la lógica de juego. Esto permite modularidad, testeabilidad y mantenimiento desacoplado.

2. **Patrón de máquina de estados finitos (FSM):** tanto el flujo del sistema (menú, splash, juego, pausa) como el gameplay utilizarán FSMs explícitas para representar y gestionar los modos de operación y transición entre ellos.

3. **RTOS cooperativo:** el sistema se ejecutará sobre un RTOS (pen particular será FreeRTOS), con tareas cooperativas y temporizadas. Se definirán prioridades fijas para cada servicio.

4. **Modelo combinado productor-consumidor / publicación-suscripción:**

   - En los casos donde un módulo conoce explícitamente a su receptor, se adoptará un esquema de productor-consumidor (1:1).

- En cambio, para eventos globales que deben ser observados por múltiples componentes, se utilizará el patrón de publicación/suscripción (1:N).

5. **Separación lógica-física:** los servicios que gestionan el estado del juego y las interfaces de usuario están desacoplados de los drivers físicos, permitiendo una posible reutilización o simulación.

3. Descripción de módulos y servicios

3.1. Boot y drivers

3.1.1. AccelDrv

- **Identificador:** AccelDrv
- **Responsabilidad principal:** Lectura de acelerómetro por I^2C; genera IRQ "data ready".
- **Interfaces**
  - API: `AccelDrv_Init()`, `AccelDrv_Read(float* xyz)`.
  - Evento publicado: `ACCEL_DATA_READY(x,y,z)` a la cola de alto nivel.
- **Datos utilizados:** Buffer `int16_t raw[3]`, escala de sensado.
- **Dependencias:** I^2C HAL, TimerDrv (timeouts).
- **Errores y manejo:** NACK o timeoutm → evento `ACCEL_ERR`.
- **Notas especiales:** Tasa de muestreo fija a 100 Hz (configurable).

3.1.2. EEPROMDrv

- **Identificador:** EEPROMDrv
- **Responsabilidad principal:** Lectura/escritura de páginas vía I^2C; verificación de ACK y CRC.
- **Interfaces**
  - API síncrona: `EEPROMDrv_ReadPage(addr,buf)`, `EEPROMDrv_WritePage(addr,buf)`.
  - Evento asíncrono: `EEPROM_DONE` / `EEPROM_ERR`.
- **Datos utilizados:** Buffer página 64 B, mapa de snapshots, checksum.
- **Dependencias:** I^2C HAL.
- **Errores y manejo:** Reintenta 3 veces; si persiste → evento `EEPROM_ERR`.
- **Notas:** Opera en modo DMA para transferencias largas.

3.1.3. ScreenDrv

- **Identificador:** ScreenDrv
- **Responsabilidad principal:** Inicializa el controlador ST7735, gestiona ventana y transferencias DMA-SPI de píxeles (doble buffer).
- **Interfaces**

- API: `ScreenDrv_Init()`, `ScreenDrv_SwapBuffers()`.
- Evento `FRAME_DONE` al terminar DMA.

- **Datos utilizados:** Dos frame-buffers de $160 \times 128 \times 16$ bpp.
- **Dependencias:** SPI HAL, DMA2, TimerDrv (VSync simulado).
- **Errores y manejo:** Timeout SPI, re-init de controlador.

3.1.4. AudioDrv

- **Identificador:** AudioDrv
- **Responsabilidad principal:** Reproduce audio mediante PWM + DMA circular; controla volumen analógico (pin GPIO).
- **Interfaces**
  - API: `AudioDrv_Play(buf,len)`, `AudioDrv_SetVolume(uint8)`.
  - Evento `AUDIO_UNDERRUN` si el búfer se vacía.
- **Datos utilizados:** Búfer de 1024 muestras 8 bit.
- **Dependencias:** TIM PWM, DMA Stream 1.
- **Errores y manejo:** Underrun → rellena con silencio.

3.1.5. HapticDrv

- **Identificador:** HapticDrv
- **Responsabilidad principal:** Dispara patrones en el DRV2605L mediante I^2C.
- **Interfaces**
  - API: `HapticDrv_PlayPattern(uint8 id)`.
  - Notificación `HAPTIC_DONE` cuando finaliza el patrón.
- **Datos utilizados:** Cola circular de 8 patrones.
- **Dependencias:** I^2C HAL.
- **Errores y manejo:** Si no hay ACK → descarta patrón y genera `HAPTIC_ERR`.

3.1.6. UARTDrv

- **Identificador:** UARTDrv
- **Responsabilidad principal:** Comunicación serie (USART2) con DMA; expone `UartDrv_Write()` y `UartDrv_Read()`.
- **Dependencias:** DMA Stream 6/7, HAL UART.
- **Errores y manejo:** Overflow RX → descarta byte y cuenta error.

3.1.7. InputDrv

- **Identificador:** InputDrv
- **Responsabilidad principal:** Unifica lectura de botones (GPIO + EXTI) y joystick (ADC DMA continuo), aplica antirrebote y dead-zone.
- **Interfaces**
  - `InputDrv_RegisterCallback(cb)`.
  - Evento `INPUT_RAW(btnMask, joyX, joyY)` cada 10 ms.
- **Dependencias:** HAL GPIO, ADC DMA.

3.1.8. TimerDrv

- **Identificador:** TimerDrv
- **Responsabilidad principal:** Genera interrupción `FRAME_TICK` cada 50 ms (20 FPS) y timers de usuario.
- **Interfaces:** `TimerDrv_Start(freq)`, callback global.
- **Dependencias:** TIM3.

3.2. Lógica del sistema

3.2.1. Boot Manager

- **Identificador:** BootMgr
- **Responsabilidad principal:** Inicializa RTOS, crea colas, verifica drivers, muestra logo y transfiere a `SYS_MAIN_MENU`.
- **Interfaces**
  - Entradas: eventos `DRIVER_OK[i]`
  - Salidas: `ENTER_SPLASH`, `GO_MENU`
- **Dependencias:** Todos los drivers.

3.2.2. Resource Loader

- **Identificador:** ResLoader
- **Responsabilidad principal:** Carga assets (sprites, fuentes, jingles) en RAM desde EEPROM o Flash.
- **Interfaces**
  - `ResLoader_Request(type,id)`, callback `RES_READY(ptr)`.
- **Dependencias:** EEPROMDrv, Graphics Engine, Audio Player.
- **Notas:** Sólo lectura; nunca escribe EEPROM.

3.2.3. Config & Persistence

- **Identificador:** ConfigPersist
- **Responsabilidad principal:** Guarda / carga snapshot de juego y ajustes; publica `HAS_SAVE`, `SAVE_OK`, `SAVE_ERR`.
- **Interfaces**
  - API síncrona: `Persist_Save(ptr,len)`, `Persist_Load(ptr,len)`.
  - Eventos: véase arriba.
- **Dependencias:** EEPROMDrv, CRC util.

3.2.4. Event Dispatcher

- **Identificador:** EventDisp
- **Responsabilidad principal:** Recibe eventos de bajo nivel (inputs, ticks), los normaliza y redistribuye a colas de alto nivel.
- **Notas:** Punto único de entrada al sistema lógico.

3.2.5. UI Controller

- **Identificador:** UIController
- **Responsabilidad principal:** Maneja menús (Splash, Main Menu, Pausa), dibuja con Graphics Engine y envía acciones (`NEW_GAME`, `LOAD_SAVE`).
- **Dependencias:** Graphics Engine, Audio Player, Haptic Engine.

3.2.6. Graphics Engine

- **Identificador:** GfxEng
- **Responsabilidad principal:** Primitivas 2-D, doble buffer, envía display-list a ScreenDrv.

3.2.7. Audio Player

- **Identificador:** AudPlayer
- **Responsabilidad principal:** Mezcla música y SFX, gestiona mute global.

3.2.8. Haptic Engine

- **Identificador:** HapEng
- **Responsabilidad principal:** Cola de patrones y temporización para DRV2605L.

3.2.9. Debug & Logs

- **Identificador:** LogSink
- **Responsabilidad principal:** Enrutamiento de `Log_Put()` a UART, comandos `logon`/`logoff`.

3.2.10. System State Mng

- **Identificador:** SysState
- **Responsabilidad principal:** FSM del sistema (`SYS_SPLASH`, `SYS_MENU`, `SYS_IN_GAME`, `SYS_PAUSED`).

3.2.11. Game Loader

- **Identificador:** GameLoader
- **Responsabilidad principal:** Decide nueva partida vs. snapshot, solicita assets y variables iniciales.

3.3. Lógica del juego

3.3.1. Game State Mng

- **Identificador:** GameState
- **Responsabilidad principal:** Ejecuta bucle de juego; pasa a `GAME_FROZEN` al pausar, procesa `RESUME` o `EXIT`.

3.3.2. Render Engine (Juego)

- **Identificador:** GameRender
- **Responsabilidad principal:** Construye display-list cada frame y la envía a GfxEng.

3.3.3. Input Handler (Juego)

- **Identificador:** GameInput
- **Responsabilidad principal:** Traduce eventos normalizados a acciones de gameplay (mover, disparar, etc.).

4. Modelos dinámicos

4.1. Diagrama de secuencia 1 - Encendido hasta menú

Descripción narrativa de cada paso.

4.2. Diagrama de secuencia 2 - Pausa y opciones

5. Concurrencia y asignación a tareas

5.1. Mapa de tareas FreeRTOS

Tabla con tareas, prioridad, stack, frecuencia.

5.2. Sincronización y comunicación

Colas, semáforos, exclusiones mutuas usadas.

6. Modelo de datos

6.1. Estructuras persistentes

Formato del snapshot de partida, layout EEPROM, checksum.

6.2. Estructuras en RAM

Buffers de audio, frame buffer, colas de eventos.

7. Manejo de errores y logging

Política de códigos de error, niveles de log, rutas de fallos críticos.

8. Requisitos de diseño no funcionales

Rendimiento (FPS), footprint de Flash/RAM, respuesta a interrupciones, requisitos de seguridad.

9. Referencias

A. Apéndice A - Glosario completo de servicios

Tabla resumen: Nombre, capa, descripción de 1 línea.

B. Apéndice B - Diagramas adicionales

Coloca aquí otros diagramas UML o listas de mensajes.