

**Universidad Nacional de Misiones. Facultad de Ingeniería.
Carrera de Ingeniería en Computación**

Estudiantes
Becker, Ismael Lucas
Estehr, Mateo Nicolás

Diseño e implementación de una consola portátil de emulación de videojuegos retro basada en Raspberry Pi Zero 2 W

**Trabajo Final Integrador presentado para obtener el título de
“Ingeniero en Computación”**

“Este documento es resultado del financiamiento otorgado por el Estado Nacional, por lo tanto,
queda sujeto al cumplimiento de la Ley N° 26.899”.

Director
Dr. Roberto Esteban Carballo

Oberá, Misiones, marzo 2026



Esta obra está licenciado bajo Licencia Creative Commons (CC) Atribución-NoComercial-CompartirIgual 4.0 Internacional. <https://creativecommons.org/licenses/by-nc-sa/4.0/>

Facultad de Ingeniería

Asignatura:

Proyecto Final Integrador

Informe Final

Diseño e implementación de una consola portátil de
emulación de videojuegos retro basada en Raspberry Pi
Zero 2 W

Estudiante:

- **BECKER**, Ismael Lucas
- **STEHR**, Mateo Nicolás

Docente:

- **DR. CARBALLO**, Roberto Esteban

Oberá, Misiones - 13/3/26

Agradecimientos

A nuestras familias, por su apoyo incondicional y constante, y por ser el sostén fundamental que nos ha acompañado en cada etapa de nuestras vidas. Gracias por creer en nosotros, por motivarnos a seguir adelante en los momentos difíciles y por celebrar con nosotros cada logro alcanzado.

A nuestros amigos, con quienes formamos un grupo increíble durante este proceso, totalmente beneficioso tanto en lo académico como en lo personal. Agradecemos su compañía, sus palabras de aliento y los momentos de alegría compartidos, que hicieron más llevadero este camino. Su apoyo y confianza fueron una fuente fundamental de motivación para no rendirnos y continuar hasta alcanzar esta meta.

De manera especial, expresamos nuestro sincero agradecimiento a nuestro tutor elegido, el Dr. Ing. Roberto Carballo, por habernos brindado la oportunidad de desarrollar este tema y por confiar en nosotros desde el inicio. Destacamos su permanente disposición, por atender siempre nuestras consultas sin reservas, y su acompañamiento constante a lo largo de este proceso. Asimismo, agradecemos su calidad humana, su calidez y su excelente predisposición, las cuales generaron un ambiente de trabajo sumamente positivo.

A todos ellos, muchas gracias.

Índice General

Capítulo I - Contextualización	9
1.1 Resumen.....	10
1.2 Introducción	11
1.3 Análisis del Problema	12
1.4 Antecedentes	13
1.5 Justificación	20
1.6 Objetivos	21
Objetivo General.....	21
Objetivos Específicos.....	21
1.7 Análisis del Panorama Competitivo.....	22
1.8 Cronograma de Actividades.....	26
1.9 Marco Teórico.....	29
Fundamentos de la Emulación de Sistemas	29
Arquitectura de Sistemas Embebidos	30
Gestión del Sistema Operativo y Entorno Linux	32
Fundamentos de Hardware para Dispositivos Portátiles	34
Diseño Asistido por Computadora (CAD) e impresión 3D	35
Capítulo II - Desarrollo	36
2.1 Consultoría Técnica Inicial y Preparación del Proyecto.....	37
2.2 Configuración del <i>Backend</i> de Emulación.....	42
2.3 Diseño del Circuito de la Consola	44
2.3 Diseño del Circuito del <i>Gamepad</i>	48
2.4 Verificación y Montaje Efectivo de los Circuitos Diseñados.....	51
2.5 Desarrollo del <i>Frontend</i> Personalizado	55
Capítulo III - Resultados	59
3.1 Conexión a Internet del Módulo Principal.....	60

3.2 Carga de <i>ROMs</i> en el Sistema.....	61
3.3 Pruebas de Control de Volumen	64
Sega Génesis	65
Nintendo Entertainment System	66
Super Nintendo Entertainment System.....	66
Nintendo 64.....	67
Game Boy/Game Boy Color/Game Boy Advance	67
PlayStation 1	68
3.4 Autonomía de la Batería	68
Capítulo IV - Conclusiones.....	71
4.1 Características de la consola desarrollada.....	72
Viabilidad Económica y Comercial	73
Prestaciones de <i>Hardware</i>	73
4.2 Dificultades en el desarrollo	74
Filtro pasa bajos para mitigación de ruido en la salida de audio	74
Problemas de Alimentación	76
Problemas de integridad de señal en la interconexión HDMI	77
Problemas de vinculación <i>Bluetooth</i> y arquitectura del control	77
4.3 Mejoras Futuras	78
Videojuegos Exclusivos.....	78
Mejoras de <i>Hardware</i>	78
Mejoras de <i>Software</i>	79
Mejoras de diseño	79
Implementación de Modo Multijugador Local	79
4.4 Resumen de resultados obtenidos	81
Anexos	82
ANEXO A — Frontend	83

Arquitectura del <i>Frontend</i>	83
Estructura del proyecto	83
Códigos fuente	85
ANEXO B — <i>Gamepad</i>	122
Descripción general	122
Arquitectura del sistema	122
Código fuente.....	123
ANEXO C — Planos del encapsulado del dispositivo	126
Descripción general	126
Plano superior	127
Plano inferior	128
ANEXO D — Documentación de la placa de circuito impreso	129
Descripción general	129
Máscara de grabado utilizada.....	129
Referencias Bibliográficas	130

Índice de Imágenes

Imagen N° 1: Desarrollo de una consola de videojuegos-Alexander Bender.....	16
Imagen N° 2: DISEÑO Y DESARROLLO DE UNA CONSOLA RETRO-Daniel Fernández Arlandiz.....	17
Imagen N° 3: DESARROLLO DE UNA CONSOLA DE VIDEOJUEGOS RETRO-Cristian Ramírez Muñoz.	18
Imagen N° 4: Diagrama de bloques de LLE y HLE.	30
Imagen N° 5: Arquitectura general del sistema completo.	38
Imagen N° 6: Componentes de la Raspberry Pi Zero 2 W.	39
Imagen N° 7: Pinout del NODEMCU-32s-ESP32	39
Imagen N° 8: Flujo de Comunicación Bluetooth Low Energy.....	40
Imagen N° 9: Pantalla LCD HDMI 5’’.....	41
Imagen N° 10: Instalación de Raspberry Pi OS (64 bits) sobre la tarjeta SD.....	42
Imagen N° 11: RetroArch ejecutándose sobre Raspberry Pi OS (64 bits).	43
Imagen N° 12: Prueba de laboratorio para la determinación del consumo de corriente eléctrica del sistema principal.	44
Imagen N° 13: Prueba de laboratorio para la determinación del consumo de corriente del ESP-32.....	45
Imagen N° 14: Diagrama de bloques del sistema completo.	46
Imagen N° 15: Componentes utilizados: Raspberry Pi Zero 2W[59], TP 4056[60], PAM 8403[61], Display 5’’[62], Parlantes[63], Batería[64], MT3608[65], Ventilador[66], y ESP-32[67].	47
Imagen N° 16: Circuito del Gamepad.....	48
Imagen N° 17: Diagrama de flujo del firmware del Gamepad.	50
Imagen N° 18: Prueba en protoboard del sistema completo.....	51
Imagen N° 19: Captura del modelo 3d del encapsulado.....	52
Imagen N° 20: Diseño y resultado obtenido del circuito impreso del gamepad...	52
Imagen N° 21: Base con componentes dispuestos.....	53
Imagen N° 22: Disposición de los componentes sobre la base del encapsulado..	53
Imagen N° 23: Disposición de los componentes sobre la tapa del encapsulado. .	54
Imagen N° 24: Resultado Final.....	54
Imagen N° 25: Diagrama de flujo del frontend.	56
Imagen N° 26: Dispositivo sin acceso a internet.	60

Imagen N° 27: Interfaz de conexión a internet.	60
Imagen N° 28: Prueba de acceso a internet.	61
Imagen N° 29: Carga de una ROM desde el servidor de Flask.	61
Imagen N° 30: Carga de una ROM desde el servidor de Flask.	62
Imagen N° 31: Correcta carga de la ROM al frontend.	63
Imagen N° 32: Menú de eliminación de juegos.	63
Imagen N° 33: Menú gestor de canciones.	63
Imagen N° 34: Pantalla de Opciones.	64
Imagen N° 35: Correcto Funcionamiento del sistema.	64
Imagen N° 36: Sonic the Hedgehog – Sega Genesis	65
Imagen N° 37: Super Mario Bros – NES.	66
Imagen N° 38: Megaman X – SNES.	66
Imagen N° 39: Mario Kart 64 – Nintendo 64.	67
Imagen N° 40: Pokémon Edición Rojo Fuego – Game Boy/Game Boy Color/Game Boy Advance.	67
Imagen N° 41: J-League Jikkyō Winning Eleven – PlayStation 1.	68
Imagen N° 42: Prototipo con alimentación autónoma desmontado para la medición de tensiones.	69
Imagen N° 43: Pruebas de duración de la batería.	70
Imagen N° 44: Dispositivo final ensamblado sostenido por el usuario para evidenciar su ergonomía y factor de forma.	70
Imagen N° 45: Espectrograma de la señal de sonido en el tiempo.	74
Imagen N° 46: Diagrama de Bode del filtro pasa bajos.	75
Imagen N° 47: Disposición inicial (izquierda), y disposición final (derecha) del sistema de alimentación.	76
Imagen N° 48: Plano del modelo 3D de la pieza superior de la consola.	127
Imagen N° 49: Plano del modelo 3D de la pieza inferior de la consola.	128
Imagen N° 50: Diseño de la placa de circuito impreso.	129

Índice de Tablas

Tabla N° 1: Características ponderadas.	22
Tabla N° 2: Características ponderadas de la consola R36s Pro.	23
Tabla N° 3: Características ponderadas de la consola Trimui Pro.	23
Tabla N° 4: Características ponderadas de la consola Genérica.	24
Tabla N° 5: Características ponderadas de la consola Pocket 6 Pro.	24
Tabla N° 6: Características ponderadas de la consola M17.	25
Tabla N° 7: Diagrama de Gantt del cronograma de actividades desarrolladas.	28
Tabla N° 8: Características ponderadas de la consola desarrollada.	72
Tabla N° 9: Descripción de los módulos del frontend.	84

Capítulo I - Contextualización

1.1 Resumen

El proyecto consiste en el diseño e implementación de una consola portátil de emulación de videojuegos retro basada en *RetroArch* sobre *Raspberry Pi Zero 2 W*. El prototipo incluye una pantalla *LCD* de 5'', un amplificador para parlantes integrados, junto con un sistema de alimentación autónoma y recargable. Cuenta con un *gamepad* inalámbrico basado en *ESP-32*, que se conecta a la *Raspberry Pi Zero 2 W* vía *bluetooth*, permitiendo la detección y transferencia de hasta 8 botones digitales y un *stick* analógico. La plataforma *RetroArch* gestiona la emulación de múltiples sistemas clásicos, y posibilita la configuración de controles. Además, se cuenta con un *frontend* personalizado, que simplifica la navegación entre distintas consolas, la conexión *Wi-Fi*, la visualización de opciones, y la apertura de un servidor local sobre el cual se pueden transferir tanto los archivos de los videojuegos como canciones que sean propiedad del usuario, y que posteriormente se ejecutarán en el dispositivo. El diseño integra de manera cohesiva *hardware* y *software*, incluyendo sistemas de alimentación autónoma, control de entradas, salida de audio y video, y comunicación inalámbrica.

Palabras clave: Emulación, Videojuegos, RetroArch, Raspberry, ESP-32.

1.2 Introducción

En las últimas décadas, los videojuegos han trascendido su condición de simple entretenimiento para convertirse en un fenómeno cultural y tecnológico de alcance global. Las primeras generaciones de consolas domésticas definieron experiencias únicas para la época, que permanecen vigentes en la memoria colectiva. Sin embargo, el acceso al *hardware* original se ha vuelto cada vez más limitado debido a la obsolescencia, la escasez de repuestos y el incremento de su valor en el mercado.

En este contexto, la emulación surgió como una alternativa tecnológica que permite preservar y reproducir sistemas clásicos mediante plataformas modernas, garantizando compatibilidad y continuidad histórica.

El presente trabajo propone el diseño e implementación de una consola portátil de emulación de videojuegos retro basada en *Raspberry Pi Zero 2 W*[1], utilizando el *framework RetroArch*[2] como núcleo de ejecución. La solución integra *hardware* y *software* en un dispositivo autónomo que incorpora pantalla *LCD*, sistema de audio, alimentación recargable y un *gamepad* inalámbrico basado en *ESP32*[3] con conectividad *Bluetooth*[4].

El desarrollo contempla tanto la configuración del *backend* de emulación como la creación de un *frontend* personalizado orientado a simplificar la experiencia de usuario, facilitando la gestión de videojuegos, la conectividad *Wi-Fi*[5] y la transferencia de archivos.

En relación con este último aspecto, se adopta un paradigma en el cual la consola no incluye juegos preinstalados, en su lugar, es el propio usuario quien debe incorporar al sistema aquellos títulos de los que posea legítimamente los derechos.

Por otro lado, se contempla técnicamente la posibilidad de ejecutar videojuegos exclusivos desarrollados específicamente para esta consola, empleando motores de desarrollo como *GameMaker*[6], que, al igual que otras herramientas similares, permiten la exportación y adaptación de proyectos a sistemas basados en *Linux* [7], como el utilizado por la *Raspberry Pi Zero 2 W*, siendo su implementación efectiva un lineamiento previsto para el futuro.

El proyecto ha requerido abordar aspectos de diseño electrónico, gestión energética, integración de periféricos y fabricación del encapsulado, articulando conocimientos de sistemas embebidos, programación y electrónica aplicada, los cuales han sido adquiridos en el transcurso de la carrera de Ingeniería en Computación.

1.3 Análisis del Problema

Aquellos que durante su infancia o juventud experimentaron consolas clásicas como la *NES* [8], *SNES* [9], *Genesis* [10] o *PlayStation* [11], hoy forman parte de una generación adulta con mayor poder adquisitivo, capaz de invertir en productos que les permitan revivir esas experiencias. Para este sector, la emulación es una forma de reconectar con un legado cultural y tecnológico que marcaron sus vidas. Sin embargo, el acceso suele verse limitado a soluciones de *software* en computadoras o móviles, que, si bien reproducen los juegos, carecen del componente físico, la ergonomía y la inmediatez de una consola dedicada

Por otro lado, los emuladores en *software* presentan otras dificultades importantes en términos de usabilidad y accesibilidad práctica, ya que requieren configuraciones específicas, instalación de controladores y ajustes técnicos que no siempre resultan intuitivos para el usuario promedio. Esta dependencia de configuraciones externas contrasta con la experiencia directa que ofrece una consola física, donde el sistema se encuentra integrado y listo para su uso, eliminando barreras de acceso y reduciendo la distancia entre el usuario y la experiencia original que se busca emular. Los ordenadores ofrecen potencia, pero no resultan prácticos para el juego en movimiento, mientras que los dispositivos móviles sacrifican ergonomía, controles físicos y calidad de visualización en pantallas pequeñas. De esta forma, el usuario se ve obligado a elegir entre una experiencia limitada en portabilidad o una restringida al entorno fijo del escritorio.

Finalmente, la disponibilidad de *hardware* retro es limitada y costosa, lo que representa una barrera para la preservación cultural y tecnológica de estos sistemas. La falta de soluciones portátiles, y de emulación universal (o de una gran variedad de sistemas) en el mercado local impide que las personas puedan experimentar esta oportunidad existente, donde se combina la jugabilidad clásica con la flexibilidad moderna.

En consecuencia, existe la necesidad concreta, en un sector social determinado, de un producto físico de emulación capaz de ofrecer un entorno portátil que proporcione una experiencia tangible, posibilitando la experimentación y preservación de los sistemas clásicos que marcaron sus vidas.

1.4 Antecedentes

El concepto de emulación aplicado a sistemas de videojuegos tiene sus orígenes en los primeros años de la industria del entretenimiento electrónico. Durante la década de 1980, en un contexto de rápida evolución tecnológica y renovación constante de *hardware*, algunos fabricantes comenzaron a incorporar mecanismos de compatibilidad que permitían ejecutar títulos desarrollados para plataformas anteriores dentro de nuevas consolas [12]. Estas soluciones no estaban basadas en *software* interpretativo como en la actualidad, sino en la integración directa de componentes electrónicos capaces de reproducir el comportamiento del sistema previo a nivel físico. En esencia, se trataba de replicar circuitos o subsistemas completos dentro del nuevo dispositivo. Este enfoque permitía ampliar la biblioteca de juegos disponible y facilitar la transición generacional para los usuarios, aunque frecuentemente presentaba limitaciones en términos de fidelidad visual, sincronización de señales o desempeño general debido a diferencias arquitectónicas entre sistemas [12].

El avance de la informática personal a comienzos de la década de 1990 marcó un punto de inflexión. El incremento sostenido en la capacidad de procesamiento, la disponibilidad de memoria *RAM* y la mejora en las arquitecturas de los microprocesadores hicieron posible trasladar el concepto de compatibilidad desde el plano estrictamente físico hacia el ámbito del *software*. Comenzaron entonces a desarrollarse programas capaces de interpretar y traducir las instrucciones de una consola específica para que pudieran ejecutarse en una computadora de propósito general. Este proceso implicaba reproducir virtualmente la *CPU*, los sistemas gráficos, los controladores de sonido y los mecanismos de entrada del *hardware* original [13], [14]. Las primeras implementaciones surgieron principalmente en comunidades de programadores independientes y entusiastas de la informática, quienes, mediante técnicas de análisis estructural y estudio del comportamiento observable del sistema, lograron recrear parcialmente su funcionamiento. En muchos casos no existía documentación oficial detallada, por lo que el trabajo implicaba un proceso minucioso de deducción técnica. Estas primeras versiones presentaban errores de sincronización, problemas de compatibilidad con determinados títulos y limitaciones en la reproducción del audio o los gráficos. Sin embargo, establecieron las bases conceptuales de lo que posteriormente se convertiría en un campo consolidado dentro de la ingeniería de *software* aplicada [12], [15].

Los sistemas de 8 y 16 bits fueron los primeros en alcanzar niveles aceptables de reproducción debido a su menor complejidad arquitectónica. La relativa simplicidad de sus procesadores, la estructura más directa de sus unidades gráficas y la limitada cantidad de memoria facilitaban el modelado virtual.

A medida que los emuladores mejoraban en precisión y estabilidad, se amplió significativamente la base de usuarios interesados en preservar y experimentar nuevamente videojuegos clásicos. Esto dio lugar al surgimiento de comunidades colaborativas que contribuyeron con pruebas de compatibilidad, optimización de código y documentación técnica no oficial [12].

Con el paso del tiempo, la potencia de cálculo de los ordenadores continuó incrementándose, reduciendo progresivamente el intervalo entre la aparición de una consola y la posibilidad de reproducirla por *software*. Arquitecturas cada vez más complejas comenzaron a ser modeladas digitalmente, incluyendo sistemas de 32 bits y posteriores. La emulación dejó entonces de ser únicamente una curiosidad técnica para convertirse en una herramienta relevante tanto para la preservación histórica como para el estudio comparativo de arquitecturas computacionales [13].

El funcionamiento de un emulador se basa en la interpretación de archivos digitales que contienen la representación binaria de los datos almacenados originalmente en cartuchos, disquetes o discos ópticos. Estos archivos son cargados en memoria y procesados por el programa, que simula el comportamiento del sistema original a nivel de instrucciones de máquina [12]. La fidelidad de la experiencia depende de múltiples factores técnicos, entre ellos la correcta reproducción del temporizado de la *CPU*, la sincronización entre procesamiento gráfico y generación de audio, la gestión de interrupciones, el acceso a memoria y la latencia en la captura de entradas del usuario. La emulación puede implementarse mediante distintos enfoques, como la interpretación directa de instrucciones [16] o la recompilación dinámica [17], cada uno con ventajas y compromisos en términos de precisión y rendimiento.

En la actualidad, la emulación constituye un campo maduro dentro de la informática aplicada al entretenimiento digital y a la preservación tecnológica. La aparición de *frameworks* capaces de integrar múltiples núcleos de emulación dentro de una misma arquitectura de *software* ha permitido unificar criterios de entrada, salida y configuración.

A continuación, se presenta una revisión sintética de distintos desarrollos orientados a la construcción de consolas de videojuegos de carácter experimental, los cuales constituyen antecedentes académicos relevantes para el presente trabajo.

Estos proyectos abordan la problemática desde diferentes niveles de abstracción tecnológica: integración de *hardware* comercial, diseño de sistemas embebidos dedicados y descripción directa del *hardware* mediante lógica programable. Este análisis permite identificar enfoques metodológicos, decisiones de diseño y alcances funcionales aplicables a la propuesta desarrollada en esta investigación.

El proyecto *Desarrollo de una consola de videojuegos*[18], de Alexander Bender, presentado en la plantea el diseño integral de una consola portátil doméstica aprovechando la disponibilidad de computadoras de placa reducida. El trabajo parte del contexto tecnológico contemporáneo, en el cual la miniaturización del *hardware* permite integrar unidades de procesamiento gráfico y central en dispositivos compactos, posibilitando la recreación funcional de sistemas de entretenimiento interactivo sin necesidad de diseñar una placa base desde cero. En este sentido, la investigación adopta como núcleo computacional una placa de bajo costo, empleada como sistema embebido capaz de ejecutar *software* libre orientado a la emulación de videojuegos clásicos.

La metodología se estructura en cuatro etapas principales: análisis de mercado, definición de requisitos técnicos, diseño mecánico y fabricación. En primer lugar, se realiza un *benchmarking* de consolas portátiles comerciales como la *Nintendo Switch* y dispositivos similares, con el fin de identificar variables relevantes para el usuario final, tales como tamaño, autonomía energética, ergonomía y calidad visual. A partir de este estudio se establecen los parámetros funcionales que debe cumplir la consola casera, trasladando conceptos de la industria a un contexto académico experimental.

Posteriormente se seleccionan los componentes electrónicos necesarios, entre los que destacan la pantalla, el sistema de control, la batería recargable y la unidad de procesamiento basada en *Raspberry Pi 4 Model B* [19]. El sistema operativo empleado corresponde a una distribución libre especializada en emulación, lo cual evita el desarrollo de videojuegos propios y permite concentrar el esfuerzo en la integración *hardware-software*. De esta forma, la consola reproduce títulos de generaciones anteriores mediante técnicas de emulación, demostrando la viabilidad de recrear plataformas históricas con recursos actuales.

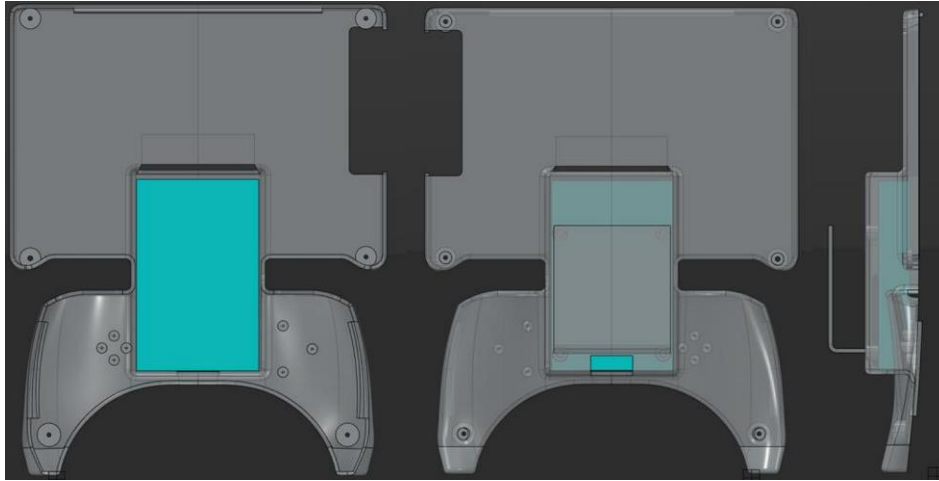


Imagen N° 1: Desarrollo de una consola de videojuegos-Alexander Bender.

Fuente: Desarrollo de una consola de videojuegos[18].

Una contribución significativa del trabajo radica en el diseño industrial del dispositivo. Mediante herramientas *CAD* se desarrolla la carcasa considerando tolerancias reales, disipación térmica y accesibilidad de puertos. Posteriormente se fabrica mediante impresión 3D, verificando la factibilidad de producir dispositivos electrónicos personalizados sin procesos industriales complejos. El ensamblaje final integra todos los subsistemas en un único producto portátil funcional.

Este antecedente resulta relevante porque demuestra que la construcción de consolas ya no depende exclusivamente de grandes fabricantes, sino que puede abordarse desde la ingeniería aplicada mediante la integración de componentes comerciales. Asimismo, evidencia la convergencia entre informática, electrónica y diseño mecánico, estableciendo una base metodológica para proyectos académicos de entretenimiento interactivo portátil.

El segundo trabajo analizado es *DISEÑO Y DESARROLLO DE UNA CONSOLA RETRO*, de Daniel Fernández Arlandiz[20], que se presenta en la Imagen N° 2, corresponde al desarrollo de una consola retro de baja potencia destinada a la ejecución de videojuegos simples programados específicamente para el sistema. A diferencia del proyecto anterior, que se centra en la emulación de plataformas existentes, este estudio aborda la creación de una arquitectura dedicada, inspirada en las primeras generaciones de videojuegos domésticos como *Pong* o *Tetris*.

El objetivo principal consiste en diseñar una plataforma de entretenimiento autocontenida basada en un microcontrolador, capaz de gestionar directamente los periféricos gráficos y de entrada sin un sistema operativo complejo.

Desde el punto de vista académico, el trabajo enfatiza la comprensión del *hardware* a bajo nivel: control de memoria, temporización, interrupciones y generación de señal gráfica.

El desarrollo comienza con la elección de un microcontrolador de arquitectura simple, al cual se conectan una pantalla de matriz de baja resolución, botones de entrada digital y un sistema de sonido básico por modulación de ancho de pulso. Posteriormente se implementa un conjunto de videojuegos programados en lenguaje *C*, diseñados específicamente para operar dentro de las limitaciones del sistema. Este enfoque permite estudiar la relación directa entre recursos computacionales y diseño lúdico, demostrando cómo las restricciones técnicas influyen en la jugabilidad. El trabajo también incluye el diseño de la carcasa y la ergonomía del dispositivo, reproduciendo la experiencia de las consolas portátiles clásicas como la *Game Boy*[21]. El resultado es una plataforma autónoma, no basada en emulación sino en ejecución nativa, donde cada programa es compilado para el *hardware* objetivo.

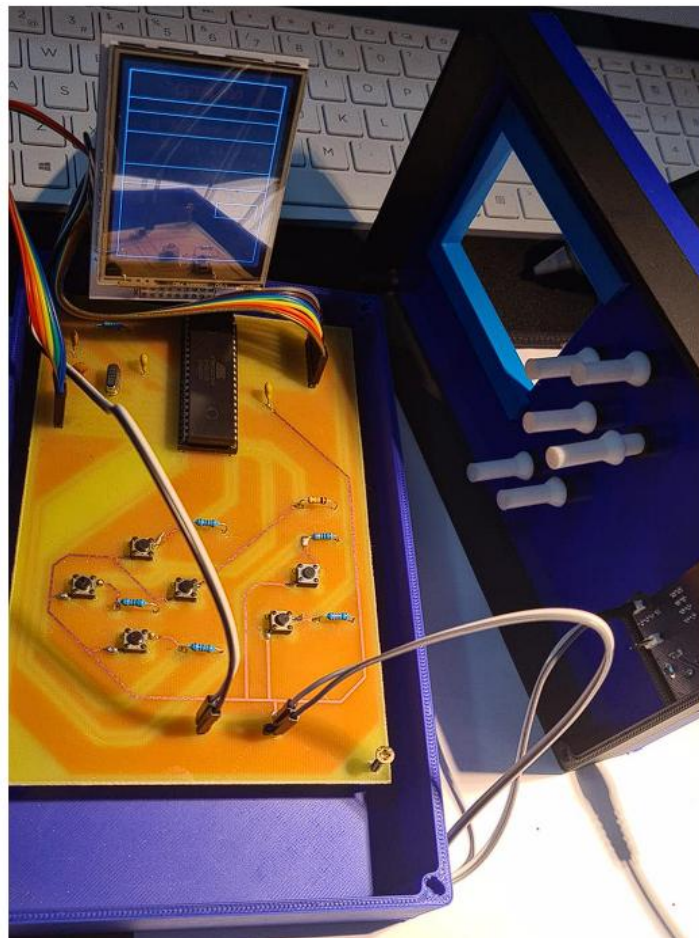


Imagen N° 2: DISEÑO Y DESARROLLO DE UNA CONSOLA RETRO-Daniel Fernández Arlandiz.

Fuente: DISEÑO Y DESARROLLO DE UNA CONSOLA RETRO[20].

Como antecedente, este proyecto es especialmente relevante porque introduce una aproximación pedagógica: la consola funciona simultáneamente como dispositivo de entretenimiento y herramienta educativa para comprender sistemas embebidos. Además, demuestra que la creación de videojuegos puede abordarse desde la ingeniería electrónica, permitiendo analizar el comportamiento del *software* directamente sobre el *hardware* físico sin capas de abstracción complejas.

El tercer antecedente, corresponde al trabajo denominado como *DESARROLLO DE UNA CONSOLA DE VIDEOJUEGOS RETRO*[22], de Cristian Ramírez Muñoz, presentado en la Imagen N° 3. Consiste en una consola implementada mediante lógica programable, donde el sistema no ejecuta un programa que simula una consola, sino que la propia consola es descrita en *hardware*. Este enfoque reproduce el funcionamiento de las primeras plataformas electrónicas, en las cuales el comportamiento del videojuego estaba determinado por circuitos digitales dedicados.

La investigación se basa en dispositivos *FPGA (Field-Programmable Gate Array)*[23], capaces de implementar circuitos digitales personalizados. Mediante lenguajes de descripción de *hardware* se diseñan módulos de procesamiento gráfico, controladores de entrada y generación de video. El sistema resultante puede ejecutar videojuegos simples directamente en *hardware*, sin *CPU* convencional ni sistema operativo, aproximándose al concepto de las primeras consolas domésticas como la *Magnavox Odyssey*[24].



Imagen N° 3: *DESARROLLO DE UNA CONSOLA DE VIDEOJUEGOS RETRO*-Cristian Ramírez Muñoz.

Fuente: *DESARROLLO DE UNA CONSOLA DE VIDEOJUEGOS RETRO*[22].

El trabajo se enfoca en la generación de señal *VGA*, temporización de píxeles y sincronización vertical y horizontal, aspectos fundamentales para comprender la electrónica de video. El juego implementado demuestra que la lógica digital, descrita estructuralmente en la *FPGA*, es suficiente para generar interacción visual en tiempo real. A diferencia de los microcontroladores, donde un programa secuencial ejecutado por una *CPU* gobierna los recursos de *hardware*, en la *FPGA* el comportamiento del sistema se materializa directamente como una arquitectura digital específica definida mediante un lenguaje de descripción de *hardware*.

Desde la perspectiva académica, el proyecto tiene gran valor formativo porque obliga a comprender el funcionamiento interno de una consola a nivel de compuertas lógicas. El estudiante debe diseñar registros, contadores, *buffers* de video y controladores de entrada, reconstruyendo conceptualmente la arquitectura de los sistemas interactivos primitivos.

Este antecedente aporta una visión histórica y tecnológica del desarrollo de videojuegos: muestra la transición desde circuitos dedicados hacia sistemas programables y posteriormente hacia la emulación. Además, permite analizar la eficiencia energética y temporal de los sistemas especializados frente a arquitecturas generales.

A partir del análisis de los tres trabajos presentados se adoptarán criterios de diseño que permitan integrar las ventajas de cada enfoque en un único sistema. El desarrollo no se limitará a la reproducción de plataformas existentes mediante emulación, sino que buscará sentar las bases de una arquitectura propia capaz de ejecutar aplicaciones desarrolladas específicamente para el dispositivo en el futuro. De los antecedentes analizados se considerará la estrategia de integración de *hardware* comercial de alta disponibilidad, empleando un sistema de cómputo compacto capaz de ejecutar sistemas operativos libres y entornos de emulación, junto con el diseño de un encapsulado propietario para el dispositivo. Este enfoque garantiza compatibilidad inicial con bibliotecas de videojuegos existentes, permitiendo validar tempranamente la funcionalidad del dispositivo y su interfaz de usuario, y con un mercado de rápido acceso. Sin embargo, a diferencia de una consola puramente emuladora, el sistema no se concibe como un producto cerrado, sino como una plataforma de transición hacia un ecosistema propio.

1.5 Justificación

La presente propuesta se justifica desde una perspectiva cultural al reconocer a los videojuegos retro como parte fundamental del patrimonio digital y de la memoria colectiva y popular de diferentes generaciones. Estas experiencias no solo marcaron un hito en la historia de la informática y el entretenimiento, sino que también influyeron en la forma en que muchas personas comprendieron la tecnología [25], [26], [27].

El desarrollo de una consola portátil contribuye a mantener vivas estas experiencias, ofreciendo un puente tangible entre pasado y presente, y asegurando que la cultura retro pueda ser transmitida y disfrutada por nuevas generaciones en un formato actual y accesible.

Por otra parte, pero como consecuencia de lo anterior, se logra una justificación comercial, dado que existe un mercado consolidado de personas con poder adquisitivo interesados en productos que combinan nostalgia y funcionalidad moderna. Esta propuesta, satisface demandas tanto de coleccionistas como de consumidores generales, posicionándose como una alternativa innovadora frente a las limitadas soluciones existentes en el mercado y con alto potencial de aceptación comercial.

Como parte de una justificación técnica o formativa, el tema elegido resulta altamente pertinente, dado que su desarrollo requiere de la combinación de conocimientos de programación y diseño de circuitos físicos en general, aspectos que representan desafíos técnicos alineados con las competencias que se espera de un profesional, por lo que involucra técnicas adquiridas durante el cursado de asignaturas específicas dentro de la carrera de Ingeniería en Computación, tales como Programación, Algoritmos y Estructura de Datos, Sistemas Embebidos, Sistemas Digitales, Electrónica Analógica, Materiales y Dispositivos Electrónicos, Internet de las Cosas, Redes o Gestión de Proyectos.

Finalmente, la propuesta resulta factible en términos técnicos y económicos, dado que los componentes seleccionados, como la *Raspberry Pi Zero 2 W*, *ESP-32*, pantalla LCD de 5'', baterías recargables y módulos de propósitos específicos, son de amplia disponibilidad, costo relativamente bajo y cuentan con extensa documentación comunitaria. Esto garantiza la viabilidad del proyecto dentro de un marco realista de recursos, al tiempo que ofrece una alternativa innovadora frente a las limitadas soluciones existentes en el mercado local.

1.6 Objetivos

Objetivo General

El objetivo general del presente proyecto consiste en desarrollar, en un plazo de cuatro meses, una consola portátil de emulación de videojuegos retro basada en *Raspberry Pi Zero 2 W* y *RetroArch*, usando una pantalla y un mando integrado, garantizando un funcionamiento estable, autonomía energética y compatibilidad con al menos el 90 % del catálogo de juegos de los sistemas retro seleccionados.

Objetivos Específicos

- Analizar y definir la arquitectura del sistema, seleccionando componentes clave como *Raspberry Pi Zero 2 W*, *ESP32*, pantalla *HDMI*[28], baterías recargables, amplificador de audio, y módulos de carga, asegurando documentación técnica completa.
- Configurar y optimizar el *backend* de emulación *RetroArch*, integrando emuladores de sistemas retro como *NES*, *SNES*, *Sega Genesis*, *PlayStation*, entre otros.
- Diseñar y desarrollar el *frontend* de emulación, utilizando *C*[29], con interfaz gráfica para gestión y ejecución de los videojuegos.
- Diseñar e implementar los circuitos de la consola y del mando, incluyendo alimentación autónoma, carga de baterías, refrigeración y sistema de sonido, con documentación detallada de componentes.
- Verificar, montar y validar el prototipo final, asegurando funcionalidad completa de la consola y los mandos, compatibilidad con *ROMs* [30], estabilidad de energía y experiencia de usuario.

1.7 Análisis del Panorama Competitivo

En este apartado se realizará un análisis detallado de los factores más relevantes identificados en los productos similares disponibles en el mercado nacional, con el fin de establecer la relación entre las prestaciones ofrecidas y su precio, y posteriormente compararla con la del producto final propuesto.

Para evaluar cada alternativa se utilizaron valoraciones subjetivas del 1 al 10 ponderadas en función de la Tabla N° 1:

Característica	Ponderación
Precio	15%
Almacenamiento (no volátil)	10%
Dimensiones de Pantalla	10%
Resolución	15%
Batería	10%
Conectividad	10%
Dimensiones	5%
Peso	5%
Emulación	10%
Exclusivos	10%

Tabla N° 1: Características ponderadas.

Fuente: Elaboración Propia.

	Consola: R36s Pro[31]		Valoración
	Precio	\$133.000	8.0 pts
	Almacenamiento	64 GB	7.0 pts
	Pantalla	3,5''	3.0 pts
	Resolución	640 x 480	7.0 pts
	Batería	3500 mAh	7.0 pts
	Wi-Fi	No	0.0 pts
	Dimensiones	13 x 8 x 2 cm	6.5 pts
	Peso	240 g	6.5 pts
	Emulación	Sí (hasta PS1)	9.0 pts
	Exclusivos	No	0.0 pts
Valoración Final Ponderada			5.5 pts

Tabla N° 2: Características ponderadas de la consola R36s Pro.

Fuente: Elaboración Propia.

	Consola: Trimui Pro[32]		Valoración
	Precio	\$215.000	7.5 pts
	Almacenamiento	256 GB	10.0 pts
	Pantalla	5''	8.5 pts
	Resolución	1280 x 720	9.0 pts
	Batería	5000 mAh	8.5 pts
	Wi-Fi	Si	10.0 pts
	Dimensiones	18,7 x 8 x 1,7 cm	7.5 pts
	Peso	229 g	7.5 pts
	Emulación	Sí (hasta NDS [33])	9.5 pts
	Exclusivos	No	0.0 pts
Valoración Final Ponderada			7.875 pts

Tabla N° 3: Características ponderadas de la consola Trimui Pro.

Fuente: Elaboración Propia.

	Consola: Genérica[34]		Valoración
	Precio	\$45.571	10.0 pts
	Almacenamiento	No	0.0 pts
	Pantalla	3,5’’	3.0 pts
	Resolución	1024 x 600	8.0 pts
	Batería	1200 mAh	2.0 pts
	Wi-Fi	No	0.0 pts
	Dimensiones	16 x 8 x 2 cm	7.0 pts
	Peso	120 g	9.0 pts
	Emulación	Sí (NES)	1.0 pts
	Exclusivos	No	0.0 pts
Valoración Final Ponderada			4.0 pts

Tabla N° 4: Características ponderadas de la consola Genérica.

Fuente: Elaboración Propia.

	Consola: Pocket 6 Pro [35]		Valoración
	Precio	\$1.200.000	2.0 pts
	Almacenamiento	256 GB	9.0 pts
	Pantalla	5,5’’	9.0 pts
	Resolución	1920 x 1080	9.5 pts
	Batería	6000 mAh	9.0 pts
	Wi-Fi	Si	10.0 pts
	Dimensiones	21 x 8,6 x 1,7 cm	7.5 pts
	Peso	320 g	5.0 pts
	Emulación	Sí (hasta Switch[36])	10.0 pts
	Exclusivos	Si	10.0 pts
Valoración Final Ponderada			8.08 pts

Tabla N° 5: Características ponderadas de la consola Pocket 6 Pro.

Fuente: Elaboración Propia.

	Consola: M17[37]		Valoración
	Precio	\$122.200	8.0 pts
	Almacenamiento	64 GB	7.0 pts
	Pantalla	4,3''	6.0 pts
	Resolución	480 x 272	4.0 pts
	Batería	1500 mAh	8.0 pts
	Wi-Fi	Si	10.0 pts
	Dimensiones	17 x 8 x 3 cm	7.5 pts
	Peso	250 g	6.5 pts
	Emulación	Sí (hasta PS1)	9.0 pts
	Exclusivos	No	0.0 pts
Valoración Final Ponderada			6.68 pts

Tabla N° 6: Características ponderadas de la consola M17.

Fuente: Elaboración Propia.

El análisis detallado de las consolas presentadas en la Tabla N° 2, Tabla N° 3, Tabla N° 4, Tabla N° 5, y Tabla N° 6 permite establecer un estudio comparativo (*benchmarking*) directo con las alternativas actualmente disponibles en el mercado argentino. Estos dispositivos representan la competencia directa y sirven como parámetro fundamental para definir las especificaciones técnicas mínimas y el rango de precio objetivo, garantizando el desarrollo de un producto comercialmente viable y competitivo. Sin embargo, el factor diferenciador más significativo de este proyecto radica en su origen de producción. La totalidad de las alternativas analizadas son manufacturadas en el exterior, lo que somete su adquisición a costos de importación, logística internacional y tiempos de espera prolongados. Al proponer una manufactura de carácter nacional, el proyecto adquiere una ventaja competitiva estratégica: la capacidad de ofrecer una garantía real, soporte técnico directo y un servicio postventa accesible. En el contexto de la electrónica importada, la gestión de reclamos o el reemplazo de componentes defectuosos resulta a menudo inviable para el consumidor local. Una producción nacional mitiga esta problemática estructural, asegurando la disponibilidad de repuestos y brindando un nivel de confianza y respaldo técnico que las alternativas extranjeras no logran satisfacer.

1.8 Cronograma de Actividades

A continuación, se presentan las diferentes etapas en las que se segmentó el desarrollo del proyecto:

1. Consultoría Técnica Inicial y Preparación del Proyecto:

- Análisis de requerimientos funcionales y técnicos.
- Definición de la arquitectura a implementar.
- Selección de componentes principales: *Raspberry Pi Zero 2 W*, *ESP-32*, Pantalla, módulos de carga y alimentación, baterías, Amplificador de Audio, etc.
- Recopilación de documentación de los componentes.
- **Entregable:** Descripción específica del objetivo del proyecto, de las funcionalidades incluidas, y de la forma de implementación, junto con los componentes principales a utilizar.

2. Configuración del *Backend* de Emulación

- Instalación y configuración de *Raspberry Pi OS Lite*, con *RetroArch*.
- Definición del conjunto de sistemas a emular (*Genesis*, *NES*, *SNES*, *GB/GBC/GBA*, *PSX*).
- Instalación de núcleos de emuladores seleccionados.
- Pruebas de compatibilidad con *ROMs*, y análisis del rendimiento.
- Configuración de salida de video dual: portátil + *HDMI* sobremesa.
- **Entregable:** Sistema de emulación con *RetroArch* operativo en *Raspberry Pi Zero 2 W*, con salida de video comprobada tanto en pantalla (modo portátil) como en salida *HDMI* (modo sobremesa).

3. Diseño del Circuito de la Consola

- Diseño de la Fuente de alimentación.
- Diseño del sistema de alimentación autónoma.
- Diseño del sistema de carga de baterías.
- Diseño del sistema de salida de audio.
- Diseño del sistema de salida de video.
- **Entregable:** Definición y documentación del circuito completo de la consola, junto con la especificación exacta de cada componente a utilizar.

4. Diseño del Circuito del *Gamepad*

- Integración con el sistema de alimentación central.
- Diseño del sistema de control mediante botones y *stick* analógico.
- Diseño y desarrollo del *firmware*.
- **Entregable:** Definición y documentación del circuito completo para el *gamepad*, junto con la especificación exacta de cada componente a utilizar.

5. Verificación y Montaje Efectivo de los Circuitos Diseñados

- Pruebas en *protoboard* del sistema completo.
- Implementación en *KiCad* de los circuitos.
- Montaje del Circuito Impreso de la Consola.
- Montaje del Circuito Impreso del *Gamepad*.
- **Entregable:** Circuitos impresos de la Consola y del *Gamepad*, con capacidad de integración funcional completa.

6. *Frontend* de Emulación

- Sistema de invocación de núcleos y *ROMs*.
- Diseño de Interfaz gráfica y menús.
- Sistema de configuración de controles.
- **Entregable:** *Frontend* instalado y funcional sobre la *Raspberry Pi Zero 2W*, capaz de iniciar la ejecución de juegos, y vincularse al *Gamepad*.

7. Documentación y Presentación

- Documentación.
- Defensa final.
- **Entregable:** Prototipo final listo para presentar.

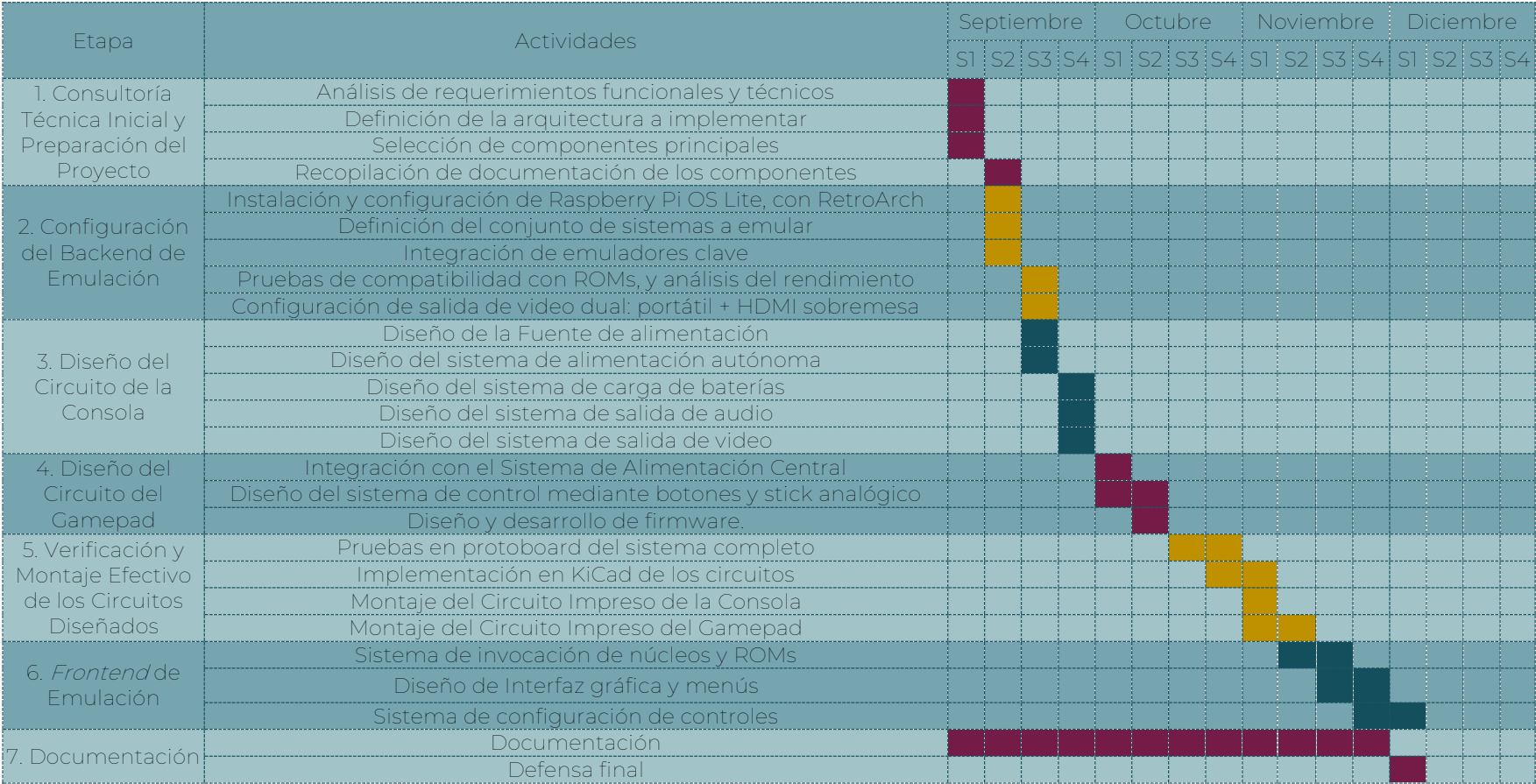


Tabla Nº 7: Diagrama de Gantt del cronograma de actividades desarrolladas.

Fuente: Elaboración Propia.

1.9 Marco Teórico

El diseño e implementación de una consola portátil de emulación de videojuegos retro es un proyecto interdisciplinario que se sustenta en principios de ingeniería de *software*, arquitectura de computadoras, electrónica analógica e interacción humano-computadora. Este marco teórico establece las bases conceptuales que fundamentan las decisiones de diseño y las tecnologías seleccionadas.

Fundamentos de la Emulación de Sistemas

La emulación es el proceso mediante el cual un sistema computacional (el anfitrión) reproduce el comportamiento de otro sistema (el huésped), permitiendo que el *software* del huésped se ejecute en el anfitrión. Es fundamental diferenciar este concepto de otros similares:

- **Emulación vs. Simulación:** Mientras que la simulación[38] modela un sistema a un nivel abstracto para estudiar su comportamiento, la emulación busca replicar su funcionamiento de manera precisa, a menudo ciclo a ciclo, para que sea funcionalmente indistinguible para el *software* que se ejecuta en él.
- **Emulación vs. Virtualización:** La virtualización[39] crea una versión virtual de un *hardware* que es idéntico al *hardware* real, gestionando el acceso a los recursos físicos. La emulación, en cambio, traduce las instrucciones de una arquitectura de *hardware* completamente diferente (ej. *MIPS*[40] de *PlayStation*) a una que el anfitrión pueda entender (ej. *ARM*[41] de *Raspberry*).

El núcleo de la emulación de consolas retro se divide en dos enfoques principales[42], cuyas arquitecturas se grafican en la Imagen N° 4:

- **Emulación de Bajo Nivel (*LLE - Low-Level Emulation*):** Este enfoque replica el *hardware* original con la máxima fidelidad posible, emulando cada componente (*CPU*[43], *PPU* [44], *APU* [45]) y su temporización. Aunque ofrece la mayor compatibilidad, posee una complejidad sumamente alta, ya que se debe estudiar cómo interactúa cada componente y cómo se comporta el sistema en su conjunto de manera perfecta, lo que requiere una gran documentación de los componentes que componen el *hardware* original o una ardua tarea de ingeniería inversa. Esto es más fácil en sistemas sencillos como por ejemplo la *Game Boy*, pero se vuelve cada vez más difícil dependiendo del *hardware* en cuestión.

- **Emulación de Alto Nivel (*HLE - High-Level Emulation*):** En lugar de emular el *hardware* directamente, el *HLE* intercepta y traduce llamadas a funciones del *BIOS*[46] o del sistema operativo del sistema original a funciones equivalentes en el sistema anfitrión. Es significativamente más rápido, aunque puede sacrificar precisión en ciertos casos. Para sistemas como *PlayStation* y posteriores, el *HLE* es una técnica común y efectiva debido a la alta complejidad que se empieza a manejarse a partir de estas consolas.

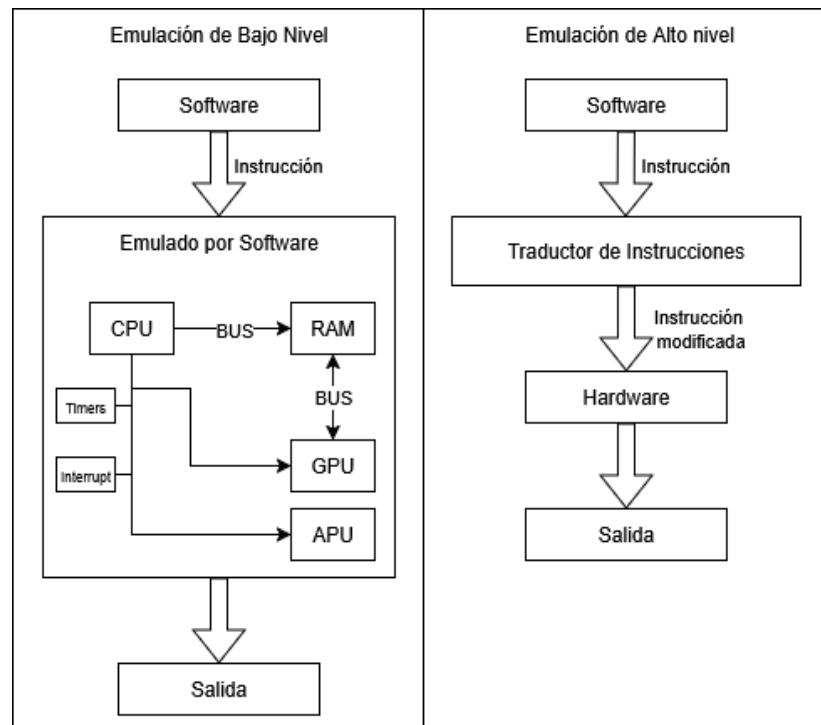


Imagen N° 4: Diagrama de bloques de LLE y HLE.

Fuente: Elaboración Propia.

Arquitectura de Sistemas Embebidos

Un sistema embebido es un sistema informático diseñado para una función dedicada. Esta especialización justifica la selección de componentes específicos en lugar de una computadora de propósito general, optimizando para factores como el costo, el consumo energético y el factor de forma.

- ***Single-Board Computer (SBC)*:** El concepto de *Single-Board Computer (SBC)* hace referencia a un sistema informático completo contenido en una sola placa de circuito impreso, condensando los elementos fundamentales de un ordenador tradicional, tales como la Unidad Central de Procesamiento (*CPU*), la Unidad de Procesamiento Gráfico (*GPU*), la memoria volátil (*RAM*) y las interfaces de entrada/salida.

- **Microcontrolador:** Un microcontrolador[47] es un circuito integrado que reúne *CPU*, memoria y periféricos en un solo *chip*, pensado para ejecutar tareas de control directas y determinísticas. A diferencia de un *SBC* no es de propósito general y normalmente se programa para realizar una tarea específica, estando optimizado para operar en tiempo real, reaccionar rápidamente a eventos del entorno y manejar *hardware* con precisión, normalmente con bajo consumo y recursos muy ajustados. Arquitectura de *Software* para Emulación

El *software* del proyecto adopta una arquitectura *Frontend/Backend*, en la cual se separan explícitamente las responsabilidades de interacción con el usuario de los procesos internos de cálculo y control.

- **Backend y la API Libretro[48]:** El *backend* es gestionado por *RetroArch*. Este no es un emulador en sí mismo, sino un *framework* que implementa la *API Libretro*. Esta interfaz de programación de aplicaciones (*API*) estandariza la comunicación entre los emuladores, denominados "núcleos" (*cores*), y la plataforma principal. Gracias a *Libretro*, *RetroArch* puede gestionar de forma centralizada tareas complejas como la configuración de mandos, el renderizado de vídeo, la salida de audio y los estados de guardado para una multitud de emuladores distintos, simplificando enormemente el desarrollo.
- **Frontend:** Es la interfaz gráfica con la que interactúa el usuario. Permite navegar por la biblioteca de juegos, ajustar configuraciones, cargar archivos y lanzar los "núcleos" de emulación correspondientes. Este proyecto utiliza un *frontend* que se ejecuta sobre el sistema operativo para invocar a *RetroArch*, ofreciendo una experiencia de usuario cohesionada y simplificada. El *frontend* desarrollado incorpora bibliotecas como *SDL* (*Simple DirectMedia Layer*), se logra gestionar los gráficos, el audio y la entrada de periféricos de manera altamente eficiente en sistemas con recursos limitados. *SDL* facilita la implementación de un ciclo de ejecución continuo (*main loop*) que captura los eventos del usuario y renderiza los elementos visuales directamente en pantalla, sin requerir un entorno de escritorio completo. Esto permite construir menús fluidos y detectar los botones del control con latencia mínima.
- **Sistema Operativo:** *Raspberry Pi OS Lite* es la versión mínima del sistema operativo oficial de *Raspberry Pi*, basada en una distribución *Debian GNU/Linux* y sin entorno gráfico.

Está pensada para proyectos donde se requiere bajo consumo de recursos, uso por terminal y ejecución de servicios o aplicaciones en segundo plano, sin embargo, al disponer de menos servicios activos, se maximiza la disponibilidad de recursos de *CPU* y *RAM* para la tarea principal, lo cual es crítico debido a que el *hardware* cuenta con recursos limitados.

- **ROMs:** Los *ROMs* (*Read Only Memory*) son copias digitales del contenido original almacenado en cartuchos o discos de consolas clásicas. Representan de manera exacta los datos presentes en el medio físico y permiten que los emuladores reproduzcan el comportamiento de los juegos tal como se ejecutaban en el *hardware* original. Estos archivos suelen conservar el formato propio de cada sistema, como *.nes*, *.sfc*, *.gb*, *.gba*, *.bin* o *.cue*. El proyecto no incorpora *ROMs* y solo provee la infraestructura para que el usuario cargue sus propios títulos, respetando así las restricciones legales asociadas a la distribución de estos contenidos.

Gestión del Sistema Operativo y Entorno Linux

- **Gestión de Paquetes y Dependencias (APT):** En sistemas operativos basados en Linux, particularmente en distribuciones derivadas de Debian, la administración del *software* se realiza mediante sistemas de gestión de paquetes como *APT* (*Advanced Package Tool*). La invocación de comandos con privilegios de administrador (mediante el uso de *sudo apt install*) permite la descarga, desempaquetado e instalación automatizada de bibliotecas, *frameworks* y aplicaciones (tales como bibliotecas gráficas *SDL*, entornos de servidor como *Flask* [49] o emuladores como *RetroArch*). Este sistema garantiza la integridad del sistema operativo al resolver y satisfacer automáticamente el árbol de dependencias requerido por cada componente.
- **Compilación de Software:** Cuando se desarrollan interfaces gráficas de usuario (*frontends*) a bajo nivel en lenguajes como *C*, el código fuente debe someterse a un proceso de compilación. Este proceso utiliza cadenas de herramientas (*toolchains*) como *GCC* para traducir las instrucciones legibles por el humano a código máquina ejecutable por la arquitectura específica del procesador, enlazando dinámica o estáticamente las bibliotecas necesarias (como *SDL*) para el renderizado gráfico y manejo de eventos.

- **Gestión de Autoarranque y Perfiles de Usuario:** En sistemas operativos *Linux* orientados a terminal, la inicialización automática de procesos puede gestionarse mediante el inicio de sesión automático (*autologin*) del usuario principal. Al configurar el sistema para acceder de forma desatendida a la consola virtual (*TTY*), se invoca el intérprete de comandos *Bash*, el cual lee y ejecuta secuencialmente las instrucciones alojadas en el archivo de entorno *.bash_profile*. Mediante la edición de este perfil de usuario, el sistema operativo automatiza el lanzamiento de aplicaciones compiladas o *scripts* inmediatamente tras completar la secuencia de encendido. Esto resulta esencial para dispositivos que operan bajo un modelo de consola dedicada, donde la interfaz de usuario debe cargarse de manera directa y sin intervención manual.
- **Arranque Silencioso (*Quiet Boot*):** En el diseño de dispositivos orientados al usuario final, es práctica común ocultar la complejidad de los procesos subyacentes del sistema operativo. Esto se logra modificando los parámetros del núcleo (*kernel*) pasados por el gestor de arranque. La implementación de directivas como *quiet* o *splash* suprime la impresión por pantalla de los registros de diagnóstico (*logs* de inicialización) durante la fase de encendido. Esta técnica previene la saturación visual de la terminal, ofreciendo una transición limpia y fluida desde el encendido del *hardware* hasta la carga de la interfaz gráfica principal.
- **Pila de Protocolos *Bluetooth* (*BlueZ*):** La gestión de las comunicaciones inalámbricas de corto alcance en entornos *Linux* se administra a través de la pila oficial de protocolos *BlueZ*. Para la interacción con esta pila desde el espacio de usuario, se emplea la utilidad de línea de comandos *bluetoothctl*. Esta herramienta expone una interfaz que permite automatizar o gestionar manualmente el escaneo, descubrimiento, emparejamiento (*pairing*) y conexión de dispositivos periféricos externos, tales como controladores de juego o interfaces de audio.

Fundamentos de Hardware para Dispositivos Portátiles

La funcionalidad portátil de un dispositivo embebido impone requerimientos estrictos sobre su arquitectura electrónica, obligando a mantener un equilibrio entre el rendimiento computacional y la eficiencia energética.

- **Sistema de Alimentación Autónoma:** La portabilidad se logra habitualmente mediante el uso de baterías de Polímero de Litio (*LiPo*), seleccionadas por su alta densidad energética y bajo peso. La carga y seguridad de estas celdas deben estar garantizadas por un Sistema de Gestión de Baterías (*BMS*). Este circuito resulta indispensable, ya que monitorea la tensión de las celdas y provee protección activa contra eventos de sobrecarga, sobredescarga y cortocircuitos.
- **Regulación de Voltaje:** La tensión suministrada por las celdas de litio no es constante, sino que desciende a medida que se descargan (típicamente desde los 4.2 V hasta los 3.0 V). Para alimentar de manera segura los componentes lógicos y digitales que requieren voltajes operativos estables (usualmente de 5 V), es obligatorio implementar una etapa de acondicionamiento de energía mediante reguladores conmutados tipo *Step-up* (o convertidores *Boost*). Esta topología de estado sólido eleva y estabiliza la tensión mediante la conmutación rápida de inductores y condensadores, logrando altas eficiencias que minimizan la disipación de calor y maximizan la autonomía.
- **Procesamiento e Interfaces de Audio y Vídeo:** La restitución multimedia en sistemas embebidos requiere una cadena estructurada de transmisión y conversión de señales. En el apartado visual, los gráficos generados por la Unidad de Procesamiento Gráfico (*GPU*) se alojan en un *framebuffer* digital. Para trasladar esta información al panel de visualización, se emplean protocolos de señal diferencial de alta velocidad (como el estándar HDMI o DSI). Estos enlaces serializan los datos y los envían hacia un controlador de pantalla (*Display Controller*), el cual decodifica la señal gráfica para excitar la matriz de píxeles correspondiente. Por otro lado, la cadena de audio inicia con un flujo de datos digitales procesados por el sistema operativo. Esta información binaria debe ser traducida a una señal eléctrica continua mediante un Conversor Digital-Analógico (*DAC*). Sin embargo, la onda analógica resultante opera a "nivel de línea", es decir, posee una amplitud de voltaje estándar pero una capacidad de corriente ínfima, incapaz de mover la bobina de un altavoz. Por ello, resulta obligatoria una etapa de amplificación de potencia.

En la teoría de diseño de dispositivos portátiles, se descartan los amplificadores lineales clásicos (como la Clase AB) por su ineficiencia térmica. En su lugar, el estándar dicta el uso de topologías de conmutación (amplificadores de Clase D), los cuales modulan la señal analógica en anchos de pulso (PWM) para hacer operar a los transistores en estados de corte y saturación. Este principio físico reduce drásticamente las pérdidas por calor, alcanzando eficiencias de conversión superiores al 85 %, un factor crítico para maximizar la autonomía energética del equipo.

Diseño Asistido por Computadora (CAD) e impresión 3D

El diseño estructural del dispositivo se fundamenta en herramientas de Diseño Asistido por Computadora (CAD), empleando el *software* de modelado paramétrico *FreeCAD*. Esta metodología permite construir geometrías tridimensionales basadas en restricciones matemáticas y un historial de operaciones, facilitando la modificación iterativa de cotas y garantizando el ajuste preciso de las dimensiones necesarias para el ensamblaje de los componentes electrónicos.

Para la fabricación de las piezas físicas se recurre a la impresión 3D, también conocida como tecnología de Modelado por Deposición Fundida (FDM). Este proceso consiste en calentar un filamento de plástico hasta su punto de fusión para luego extruirlo y depositarlo capa por capa, construyendo así el objeto tridimensional. En esta etapa de prototipado, el material elegido es el filamento de Ácido Poliláctico (*PLA*), un termoplástico seleccionado por su excelente estabilidad dimensional, su adecuada rigidez estructural y su baja tendencia a deformarse por cambios de temperatura durante la impresión.

Aunque el PLA presenta limitaciones mecánicas al superar su temperatura de transición vítrea (aproximadamente 60 °C), excede holgadamente los requerimientos de una carcasa portátil de baja disipación térmica. Su bajo costo y facilidad de extrusión lo convierten en el material idóneo para ciclos de prototipado rápido, permitiendo validar empíricamente la ergonomía y la funcionalidad del chasis de manera eficiente.

Capítulo II - Desarrollo

2.1 Consultoría Técnica Inicial y Preparación del Proyecto

El desarrollo del proyecto se llevó a cabo siguiendo el cronograma establecido en el diagrama de Gantt de la Tabla N° 7. Cada etapa fue definida con objetivos concretos y dependencias bien delimitadas, lo que permitió mantener un flujo de trabajo ordenado.

En la primera etapa, denominada *Consultoría Técnica Inicial y Preparación del Proyecto*, se desarrolló un análisis de requerimientos funcionales y técnicos, considerando aspectos de desempeño, autonomía, portabilidad, ergonomía y compatibilidad con distintos sistemas de videojuegos clásicos. A partir de este estudio, se determinaron requerimientos funcionales presentados a continuación:

- **RF-01:** El sistema deberá emplear una unidad principal de procesamiento que *garantice* equilibrio entre capacidad de procesamiento, consumo energético y conectividad inalámbrica integrada.
- **RF-02:** El sistema deberá permitir la emulación de plataformas de videojuegos retro de 8, 16 y 32 bits, asegurando una ejecución estable y con desempeño adecuado para los títulos compatibles.
- **RF-03:** El sistema deberá incorporar un dispositivo de control, capaz de comunicarse con el módulo principal mediante un enlace de baja latencia.
- **RF-04:** El sistema deberá operar en modo portátil, utilizando una pantalla integrada y un sistema de alimentación autónomo que permita su funcionamiento independiente de la red eléctrica.
- **RF-05:** El dispositivo deberá presentar un diseño físico propio, optimizado para la ergonomía del usuario, la portabilidad y la integración estética de sus componentes.
- **RF-06:** El sistema deberá soportar, principalmente emulación de plataformas preexistentes, y a modo de aditivo secundario para la expansión futura, deberá poder soportar la ejecución nativa de videojuegos exclusivos diseñados para la plataforma.

Posteriormente, se procedió a la definición de la arquitectura general del sistema, ilustrada en la Imagen N° 5. El diagrama representa de forma sintetizada la composición interna del dispositivo.

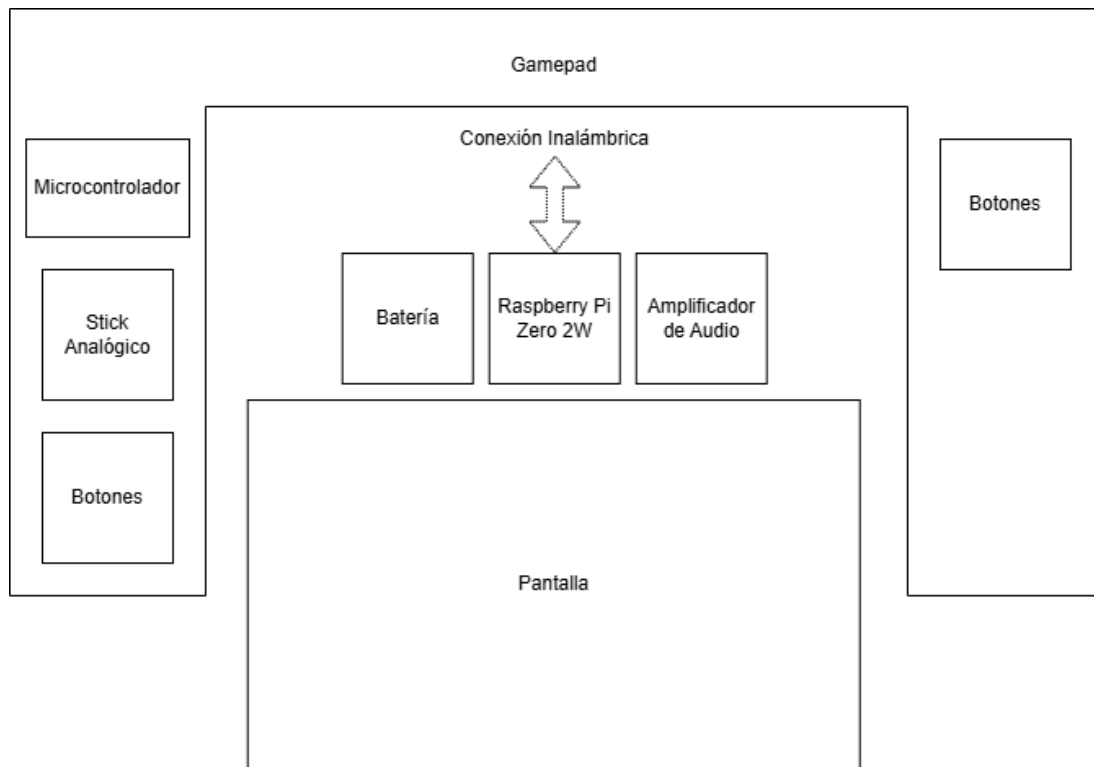


Imagen N° 5: Arquitectura general del sistema completo.

Fuente: Elaboración Propia.

Tras definir la arquitectura general del dispositivo, se procedió a la selección rigurosa de los componentes físicos, evaluando sus hojas de datos (*datasheets*) para garantizar el cumplimiento de los requerimientos establecidos y en función de su disponibilidad, bajo consumo energético y facilidad de integración, así como la recopilación y análisis de la documentación técnica de cada uno, asegurando su compatibilidad dentro del sistema.

Como núcleo de procesamiento (RF-01), se seleccionó la *Raspberry Pi Zero 2 W* como núcleo de procesamiento, la cual fue elegida por experiencia pasada, disponibilidad, y documentación existente. Su arquitectura y componentes se exhiben gráficamente en la Imagen N° 6. Este modelo maximiza la miniaturización mediante un encapsulado *System-in-Package* (SiP) *RP3A0* que apila 512 MB de memoria *RAM LPDDR2* sobre un *SoC Broadcom BCM2710A1*. Impulsado por un procesador *ARM Cortex-A53* de cuatro núcleos a 1.0 GHz y una *GPU VideoCore IV*, ofrece el equilibrio exacto entre capacidad de cálculo para emulación (RF-02), bajo consumo y la compatibilidad con un ecosistema de *software* robusto basado en distribuciones *Linux* [50], [51], [52], [53], [54].

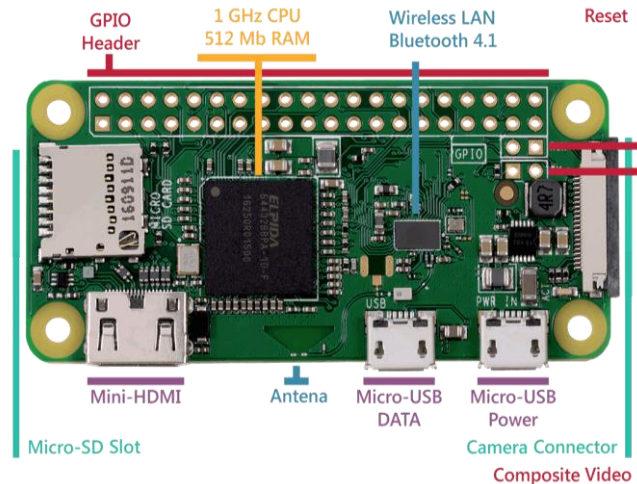


Imagen N° 6: Componentes de la Raspberry Pi Zero 2 W.

Fuente: Raspberry pi zero w headless setup wifi - jasmis[55]

En paralelo, para gestionar los periféricos de entrada, también llamados *gamepad*, (RF-03), se integró una placa *NodeMCU-32S* basada en el microcontrolador *ESP32* como el de la Imagen N° 7. La elección de este chip radica en la disponibilidad de conversores analógico-digitales (ADC) nativos, indispensables para la lectura de la palanca de direcciones, sumado a su capacidad de enviar telemetría a la unidad central mediante *Bluetooth* con latencias mínimas y alta eficiencia energética.

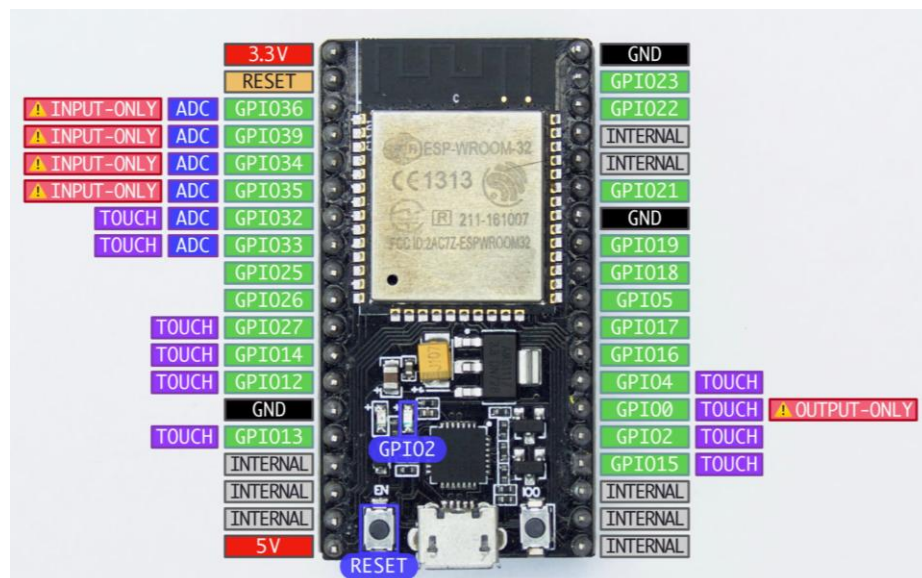


Imagen N° 7: Pinout del NODEMCU-32s-ESP32

Fuente: NodeMCU-32s – ESP32 ESP-WROOM-32 Development Board – CircuitSetup [56]

Para la conectividad del módulo ESP32 con el núcleo de procesamiento se utilizó una variante del estándar *Bluetooth*[4] llamada *Bluetooth Low Energy (BLE)* la cual fue diseñada para ofrecer comunicación inalámbrica con un consumo energético muy reducido. A diferencia de la versión clásica, que mantiene conexiones continuas y orientadas a un mayor ancho de banda, *BLE* utiliza un modelo basado en intervalos cortos de transmisión y reposo, lo que disminuye significativamente el gasto de energía en dispositivos portátiles.

Su funcionamiento se grafica en la Imagen N° 8, y se basa en un esquema de roles donde un dispositivo actúa como servidor, exponiendo servicios y características, mientras que otro opera como cliente, accediendo a esa información mediante operaciones de lectura, escritura o notificación. Esta estructura permite intercambiar datos con baja latencia, suficiente para interpretar pulsaciones de botones y movimientos analógicos en tiempo real. Además, *BLE* utiliza un protocolo simplificado y paquetes de menor tamaño, lo que mejora la eficiencia en microcontroladores con recursos limitados como el *ESP32*. Para su implementación en el firmware del *gamepad* se utiliza una librería llamada *ESP32-BLE-Gamepad*, la cual se encarga de manejar las conexiones de este tipo e implementa además varias funciones útiles para utilizar un *ESP32*. Esta librería empaqueta información como los *inputs* del usuario al presionar botones, mover los ejes o información adicional, como el nivel de batería, para que el sistema los interprete correctamente.

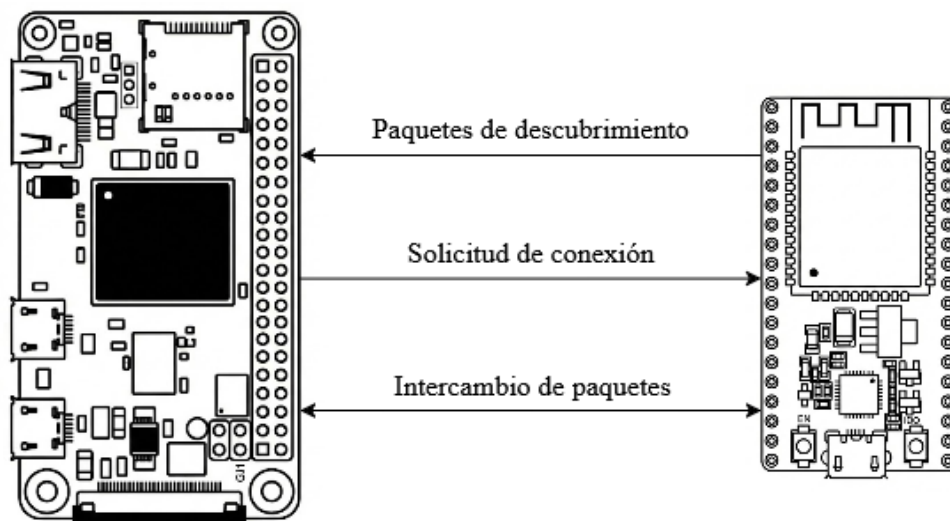


Imagen N° 8: Flujo de Comunicación Bluetooth Low Energy.

Fuente: Elaboración Propia.

Respecto al sistema de alimentación autónomo (RF-04), se diseñó una etapa de potencia fundamentada en celdas *LiPo*. La gestión de carga y seguridad quedó delegada a módulos basados en el circuito integrado *TP4056*. Para garantizar la alimentación estable a 5 V de toda la electrónica, se analizaron diversas topologías *Step-up*. Si bien existen chips combinados con el *TP4056* como el *ME2108A* este está limitado a 600 mA, esto solo sería suficiente para satisfacer la demanda de corriente de la *Raspberry Pi Zero 2 W*, por lo que la pantalla, controles, sistemas de audio y refrigeración requerían módulos dedicados de mayor capacidad. Por ello, se perfilaron opciones como el integrado *MT3608* (capaz de suministrar hasta 2 A a 5 V gracias a su alta frecuencia de conmutación) o alternativas para mayores exigencias como el chip *XL6009* (que soporta picos de hasta 4 A a 5 V), asegurando así un suministro ininterrumpido y seguro para los componentes críticos, eligiendo finalmente el primero debido a su menor tamaño.

Finalmente, para resolver la visualización y el sonido del modo portátil (RF-04 y RF-05), se optó por una pantalla *LCD HDMI* de 5 pulgadas como la de la Imagen N° 9. Este panel ofrece una resolución nativa de 800x480 píxeles, otorgando una densidad de 186.59 *PPI*, ideal para la nitidez de gráficos retro sin saturar el ancho de banda de la *GPU*. La placa controladora de este monitor integra un DAC que extrae la pista de audio del flujo *HDMI* hacia una salida auxiliar analógica (*AUX*). Para el funcionamiento de los altavoces de la consola, se acopló a esta salida un módulo amplificador de audio *PAM8403* de Clase D, seleccionado específicamente por su ínfimo factor de forma y su eficiencia superior al 85%, característica vital para preservar la carga de la batería del dispositivo.

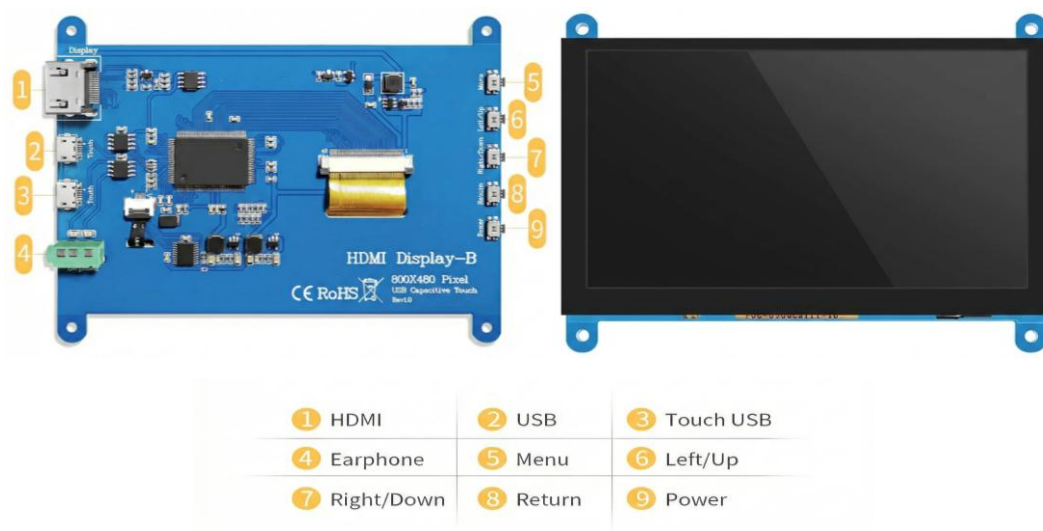


Imagen N° 9: Pantalla LCD HDMI 5''.

Fuente: 5 inch HDMI Display for Raspberry Pi[57].

2.2 Configuración del *Backend* de Emulación

La segunda etapa, denominada *Configuración del Backend de Emulación*, consistió en la instalación y personalización de *Raspberry Pi OS* en su versión de 64 bits, basada en arquitectura *AArch64*. La *Raspberry Pi Zero 2 W* incorpora un procesador *Broadcom BCM2710A1* de cuatro núcleos basado en la microarquitectura *ARM Cortex-A53*, funcionando a una frecuencia de 1,0 GHz. Este procesador implementa la arquitectura *ARMv8-A*, por lo que corresponde a una plataforma de 64 bits (*AArch64*). No obstante, mantiene compatibilidad con aplicaciones de 32 bits (*ARMv7*), permitiendo la ejecución tanto de sistemas operativos como de *software* desarrollado para ambas modalidades. Esta versión fue seleccionada por su mayor compatibilidad y estabilidad con los controladores y bibliotecas gráficas disponibles para la *Raspberry Pi Zero 2W*.

Sobre este sistema operativo se implementó *RetroArch* (Imagen N° 11) como *backend* de emulación, configurando los núcleos correspondientes a *NES*, *SNES*, *Sega Genesis*, *Nintendo Game Boy* (incluyendo *Game Boy Color* y *Game Boy Advance*), *Nintendo 64* y *PlayStation 1*. A partir de una instalación mínima del sistema operativo, se optimizó el proceso de arranque y se redujo el consumo de recursos mediante la desactivación de servicios no esenciales, garantizando un entorno ligero, eficiente y estable para la ejecución de los emuladores.

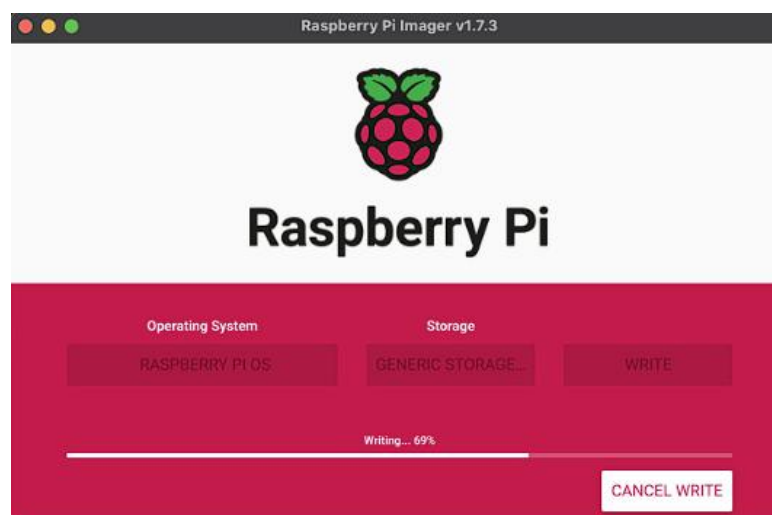


Imagen N° 10: Instalación de Raspberry Pi OS (64 bits) sobre la tarjeta SD.

Fuente: Elaboración Propia.



Imagen N° 11: RetroArch ejecutándose sobre Raspberry Pi OS (64 bits).

Fuente: Elaboración Propia.

En la Imagen N° 10, se presenta parte del proceso de creación de la imagen del sistema operativo mediante *Raspberry Pi Imager*, utilizado para grabar en la tarjeta *microSD* la distribución *Raspberry Pi OS* de 64 bits basada en *AArch64*[58].

Este procedimiento generó una instalación limpia, asegurando la correcta partición del sistema de archivos necesarias para la operación del dispositivo. En la Imagen N° 11 se observa *RetroArch* ejecutándose sobre dicho entorno, mostrando su interfaz de usuario, la cual centraliza la carga y gestión de los *cores* de emulación. Esta interfaz facilita la configuración de parámetros del sistema, asignación de controladores, administración de contenido y selección de núcleos de emulación.

Algunas configuraciones adicionales realizadas al sistema operativo en las etapas finales fueron orientadas a implementar un arranque silencioso (*Quiet Boot*). Para ello, se editaron los archivos de inicialización ubicados en el directorio */boot/firmware/* se añadieron los parámetros *consoleblank=1* y *loglevel=0* en el archivo *cmdline.txt*, y la directiva *disable_splash=1* en *config.txt*. El objetivo de esta modificación es suprimir la impresión en pantalla de los registros de diagnóstico del *kernel* y los logotipos de encendido, ocultando la complejidad del sistema para brindar una transición limpia y de aspecto profesional hacia la interfaz principal.

2.3 Diseño del Circuito de la Consola

La tercera etapa, denominada *Diseño del Circuito de la Consola*, se enfocó en el desarrollo del sistema electrónico de la arquitectura presentada. En primer lugar, se llevó a cabo el diseño del sistema de alimentación, analizando los requerimientos energéticos del módulo central y de los periféricos asociados. Para ello, se realizaron mediciones experimentales en laboratorio, con la intención de estimar los consumos de corriente de cada componente, y obtener la corriente total requerida por el sistema, para su correcto funcionamiento.

$$I_{Total} = I_{Raspberry} + I_{Pantalla} + I_{PAM-8403} + I_{Ventilador} + I_{ESP-32}$$

Se comenzó entonces cuantificando el consumo de tanto la *Raspberry Pi Zero 2 W*, como de la pantalla *LCD HDMI 5"* en conjunto. Como se presenta en la Imagen N° 12, se obtuvo un consumo promedio estable de aproximadamente 680 mA bajo condiciones normales de operación.

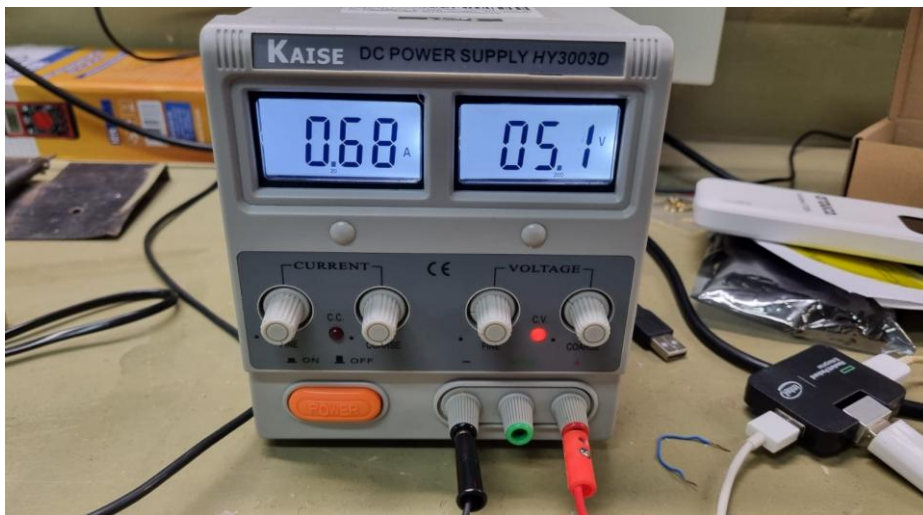


Imagen N° 12: Prueba de laboratorio para la determinación del consumo de corriente eléctrica del sistema principal.

Fuente: Elaboración Propia.

Además de estos componentes, se considera también la utilización de un amplificador clase D de 3 W, *PAM-8403*, elegido por su alta eficiencia y disponibilidad, el cual adiciona un consumo de corriente de hasta 600 mA (corriente máxima estimada en condiciones de máxima potencia de salida). Por otra parte, se incluye un ventilador de 5 V y 240 mA, que eleva el consumo contabilizado hasta los 1,52 A.

El último componente en consideración fue el *ESP-32*, que, como se indica en la Imagen N° 13, presentó un consumo de 80 mA durante las pruebas realizadas, con su funcionalidad de *Bluetooth 4.2* activada.

El consumo de corriente total, resultante de las mediciones presentadas, corresponde entonces a 1,6 A.

$$I_{Total} = 680 \text{ mA} + 600 \text{ mA} + 240 \text{ mA} + 80 \text{ mA} = 1600 \text{ mA}$$

Con un factor de seguridad de (1,5), que considera situaciones de estrés elevado de la *Raspberry* o *ESP-32*, el desgaste de las baterías, y variaciones en los consumos estimados, el resultado concluye en:

$$I_{Seguridad} = I_{Total} * 1,5 = 1600 \text{ mA} * 1,5 = 2400 \text{ mA}$$



Imagen N° 13: Prueba de laboratorio para la determinación del consumo de corriente del ESP-32.

Fuente: Elaboración Propia.

A partir de estos datos, se procedió al diseño del sistema de alimentación autónoma, seleccionando dos baterías de litio-polímero de 3,7 V y 4000 mAh, cuya capacidad permite alcanzar una autonomía superior a las 3 horas para el consumo estimado.

$$T_{Total} = \frac{C_{Baterías}}{I_{Seguridad}} = \frac{2 * 4000 \text{ mAh}}{2400 \text{ mA}} = 3,33 \text{ hs}$$

Esta batería fue elegida por su densidad energética, bajo peso y disponibilidad, características relevantes para el dispositivo portátil. Posteriormente, se desarrolló el sistema de carga y protección de la batería, integrando un módulo *TP4056 con step-up* incorporado, que combina el controlador de carga y protección para celdas de litio con un convertidor elevador de tensión a 5 V.

Este módulo fue seleccionado por su simplicidad de integración, tamaño reducido y capacidad para proporcionar una salida estable y segura desde la celda de 3,7 V hacia los componentes principales de 5 V, permitiendo además la recarga mediante puerto *micro USB*. El *TP4056* a parte de cargar las baterías, proporciona los 5 V que requiere la *Raspberry Pi Zero 2 W* para funcionar.

Por otra parte, se incluyen dos módulos *step-up MT3608* cuyas entradas comparten la misma línea de alimentación, que toman los 3,7 V de las baterías y lo elevan hasta 5 V. Uno de ellos alimenta a la pantalla, mientras que el otro alimenta al *ESP-32* y el ventilador. El *PAM-8403* se alimenta directamente con los 3,7 V de la batería.

En paralelo, se abordó el diseño del sistema de salida de audio, optándose por el amplificador *PAM8403*, un circuito integrado de clase D de alta eficiencia y baja disipación térmica, capaz de entregar hasta 3 W por canal con una distorsión armónica mínima. Este componente fue seleccionado por su tamaño reducido y capacidad de operar directamente con alimentación de 3,7 V. Se dimensionó además un parlante lateral de 32 Ω , integrados en la carcasa, optimizando su posición para mejorar la percepción del audio.

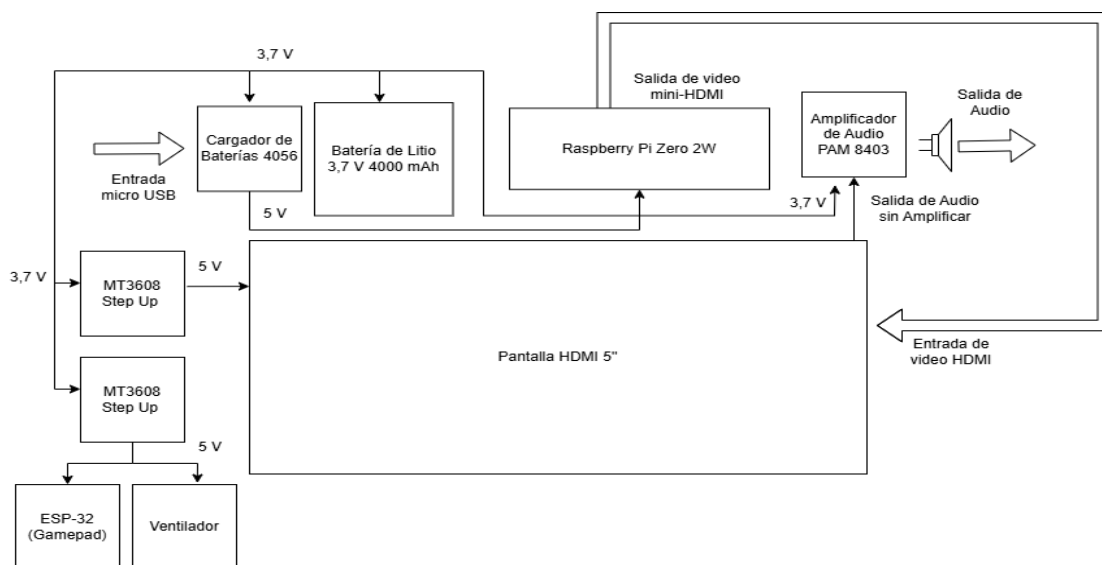


Imagen N° 14: Diagrama de bloques del sistema completo.

Fuente: Elaboración Propia.

Finalmente, se diseñó el sistema de salida de video, seleccionando una pantalla *LCD HDMI* de 5'' completamente compatible con la arquitectura de la *Raspberry Pi Zero 2W*. Este *display* proporciona una resolución en alta definición, bajo tiempo de respuesta y alimentación directa a 5 V, lo que simplifica su integración en el sistema.

Adicionalmente, la pantalla incorpora una salida de audio auxiliar (*jack* de 3,5 mm) integrada en el propio módulo *HDMI*, permitiendo la extracción del sonido digital directamente desde la señal *HDMI* sin necesidad de un convertidor *DAC* externo, optimizando así el diseño y reduciendo la complejidad del circuito de audio. El sistema completo se presenta en la Imagen N° 14.

A modo de cierre de la presente etapa, en la Imagen N° 15 se exponen los distintos componentes previamente descritos.



Imagen N° 15: Componentes utilizados: Raspberry Pi Zero 2W[59], TP 4056[60], PAM 8403[61], Display 5''[62], Parlantes[63], Batería[64], MT3608[65], Ventilador[66], y ESP-32[67].

Fuente: Elaboración Propia.

2.3 Diseño del Circuito del *Gamepad*

En la cuarta etapa, “*Diseño del Circuito del Gamepad*”, se seleccionó al *ESP-32* como microcontrolador principal para cada control, debido a su pequeño tamaño, su capacidad de conexión inalámbrica, y porque cuenta con la cantidad de terminales suficientes para los botones utilizados. Como se presentó anteriormente, la corriente requerida por el *Gamepad* con los botones, el *stick* y el *bluetooth* activado, resulta en 80 mAh, como se indica en la Imagen N° 13. Los 5 V necesarios para alimentarlo, provienen desde el *MT3608*. Por otro lado, se diseñó el sistema de control, compuesto por un conjunto de 8 botones de presión y un *stick* analógico de doble eje, cuyo posicionamiento fue definido en función de criterios ergonómicos y sus terminales se conectaron al *ADC* del microcontrolador. Además, se incorporó un divisor resistivo que se conecta a una de las terminales del *ADC*, cuyo voltaje es digitalizado y posteriormente transmitido vía *Bluetooth* hacia la *Raspberry Pi Zero 2 W*, para que esta la presente desde su *frontend* hacia el usuario.

Finalmente, se llevó a cabo el diseño y desarrollo del *firmware*, implementado en lenguaje C sobre una plataforma de desarrollo *Arduino*-compatible, que permite la comunicación del *gamepad* con la consola a través de protocolo *USB* o inalámbrico, interpretando las acciones del usuario y enviando los comandos correspondientes vía *Bluetooth*. El circuito diseñado se presenta en la Imagen N° 16.

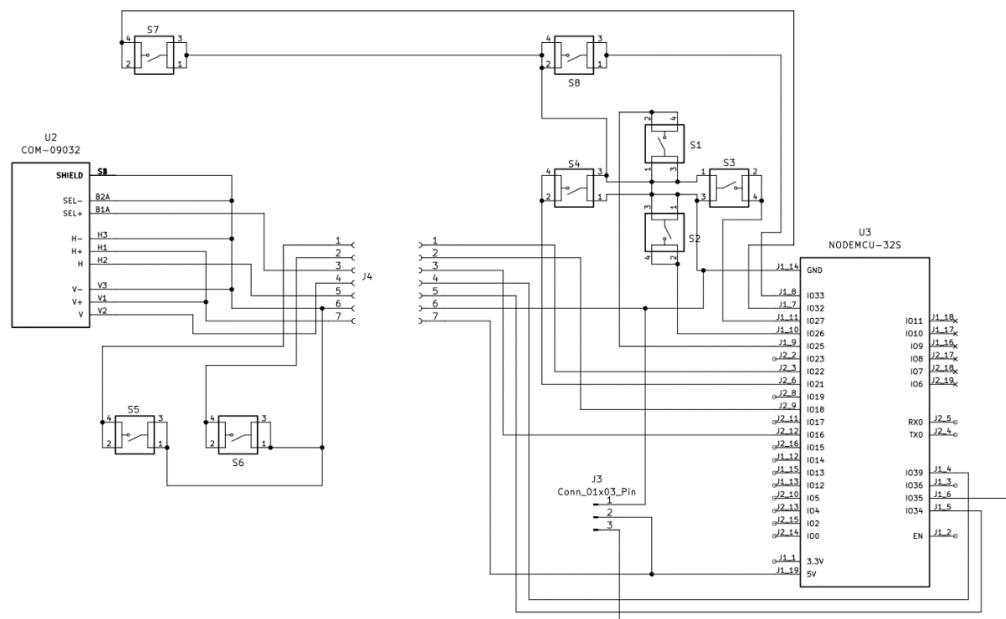


Imagen N° 16: Circuito del *Gamepad*.

Fuente: Elaboración Propia.

El *firmware* del *gamepad* funciona inicialmente mediante una rutina de calibración ejecutada durante el arranque del sistema, en la cual las posiciones analógicas de reposo de los ejes del *joystick* son medidas y almacenadas como el centro de referencia. Este procedimiento es implementado minuciosamente para compensar las tolerancias de fabricación de los componentes y evitar derivas mecánicas no deseadas. Posteriormente, se inicia el servicio *bluetooth* y comienza el bucle principal el cual está condicionado de manera estricta a la existencia de una conexión *bluetooth* activa.

Durante esta ejecución cíclica, el estado de los interruptores físicos es evaluado mediante un algoritmo de atenuación de rebotes, donde las oscilaciones electromecánicas transitorias son filtradas por retardos temporales, garantizando que cada accionamiento sea registrado como una única pulsación lógica. El cambio en el estado de los botones es detectado y emitido por *bluetooth*.

En la siguiente etapa se analizan las lecturas de *joystick* analógico. Un filtro de suavizado es aplicado sobre las lecturas brutas para suprimir el ruido eléctrico, estableciendo una zona muerta. Si la magnitud del desplazamiento es calculada como inferior a este umbral, el movimiento es descartado. Al superar esta zona de exclusión, se utiliza histéresis para discretizar las coordenadas analógicas en direcciones digitales fijas, previniendo fluctuaciones erráticas en los límites espaciales.

Finalmente, la monitorización energética es ejecutada mediante un pin analógico, donde múltiples muestras de voltaje de la batería son capturadas, promediadas y filtradas algorítmicamente. Este valor de tensión estabilizado es escalado a un porcentaje útil y transmitido de forma inalámbrica a intervalos controlados de medio segundo, preservando el ancho de banda y la baja latencia requerida por los comandos.

A continuación, en la *Imagen N° 17* se presenta un diagrama de flujo del *firmware* correspondiente al *gamepad*:

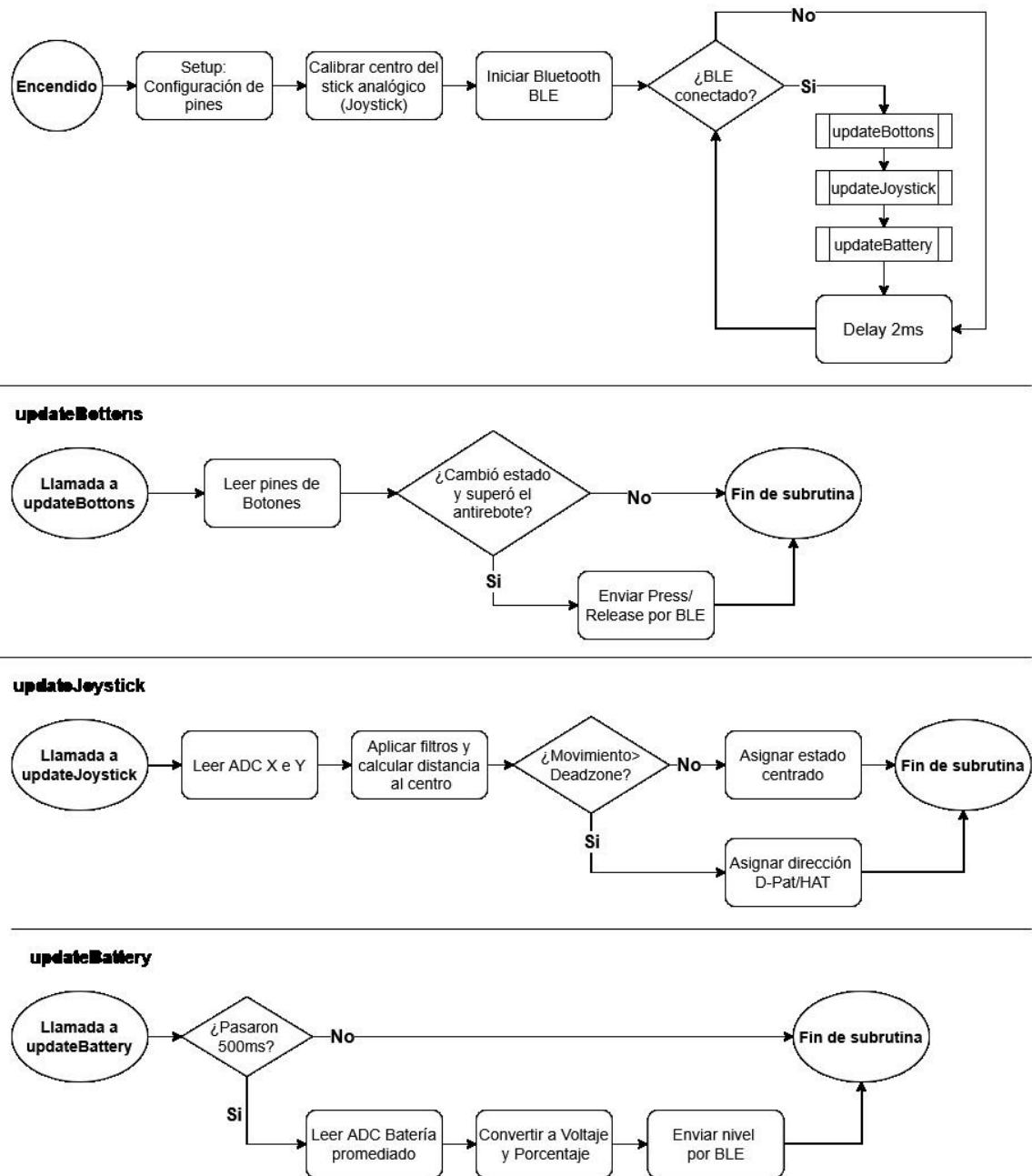


Imagen N° 17: Diagrama de flujo del firmware del Gamepad.

Fuente: Elaboración Propia.

2.4 Verificación y Montaje Efectivo de los Circuitos

Diseñados

En la quinta etapa, “*Verificación y Montaje Efectivo de los Circuitos Diseñados*”. Como se muestra en la Imagen N° 18, se realizaron pruebas en *protoboard* de cada subsistema: alimentación, carga, audio, video y control, lo que permitió evaluar la respuesta real de los componentes en forma aislada.

Una vez verificado el correcto funcionamiento, se procedió a la implementación de los esquemas en *KiCad*, generando los diagramas eléctricos y las pistas correspondientes para el diseño de las placas de circuito impreso (*PCB*). Posteriormente, se efectuó el montaje de la consola, integrando los módulos de la *Raspberry Pi*, la pantalla *LCD*, los conversores de tensión, el amplificador de audio y los conectores *HDMI* y *USB*, asegurando una disposición eficiente del cableado y la disipación térmica. En paralelo, se realizó el montaje del circuito impreso del *gamepad*, ensamblando los botones, *sticks* analógicos, microcontroladores y módulos de comunicación, con especial atención a la ergonomía y robustez mecánica. Durante toda esta etapa, y de forma simultánea se realizaron versiones de prueba del encapsulado, utilizando el programa de diseño y modelado en 3D *Freecad*[68], e imprimiendo estos modelos utilizando una impresora *Creality Ender 3 Pro*[69], y *PLA*[70] de 1,75 mm como filamento.

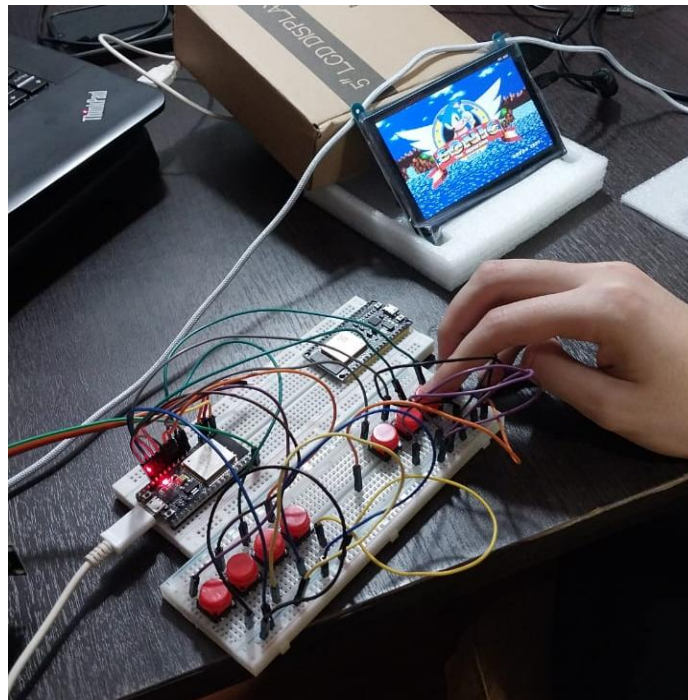


Imagen N° 18: Prueba en *protoboard* del sistema completo.

Fuente: Elaboración Propia.

El circuito impreso diseñado, y su resultado final, se presentan en la Imagen N° 20, mientras que el diseño del encapsulado en la Imagen N° 19.

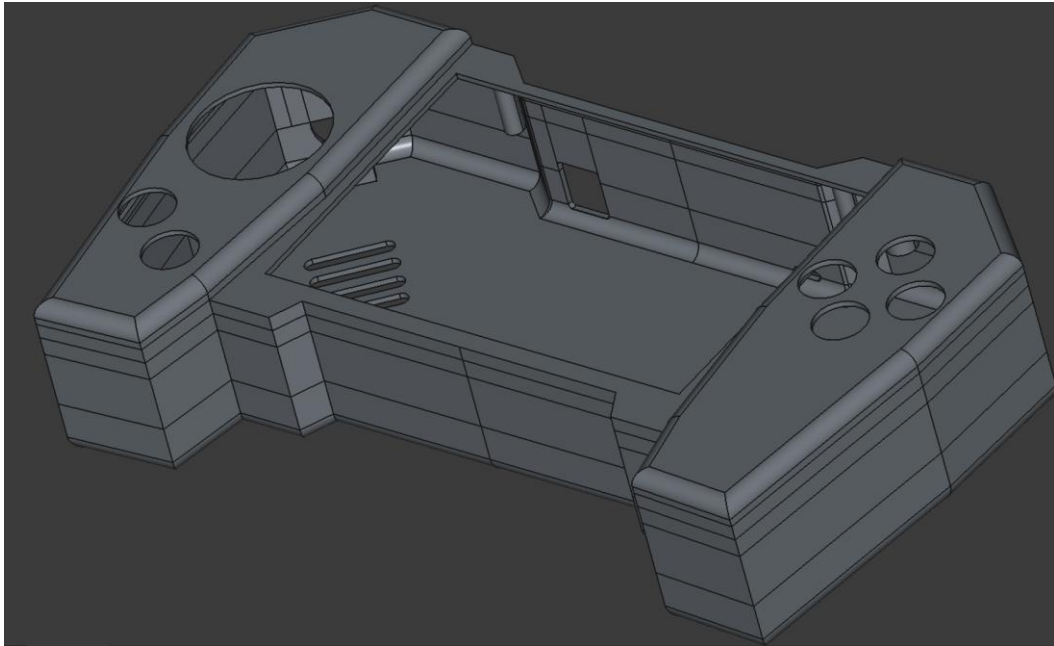


Imagen N° 19: Captura del modelo 3d del encapsulado.

Fuente: Elaboración Propia.

La consola completamente ensamblada se exhibe en la Imagen N° 24.

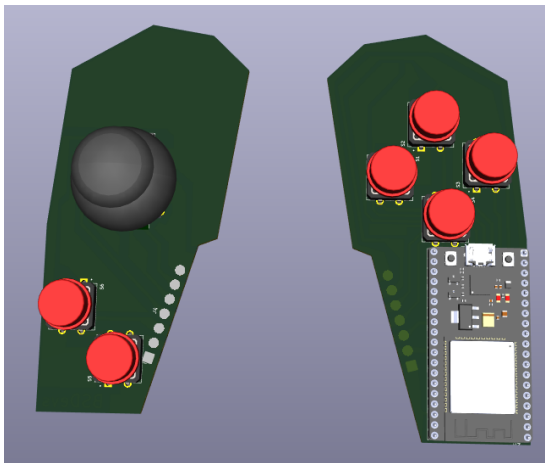


Imagen N° 20: Diseño y resultado obtenido del circuito impreso del gamepad.

Fuente: Elaboración Propia.

En la Imagen N° 19, se presenta una captura del modelo 3D de la estructura contenedora de la consola, la cual constituye el encapsulado físico destinado a alojar y proteger los distintos componentes electrónicos del sistema. Para su fabricación se optó por un diseño modular, imprimiéndose de manera independiente dos piezas diferenciadas: la tapa superior y la base inferior de la consola.

Posteriormente, y una vez impresos los modelos de la tapa y la base, se procedió a la disposición de los diferentes módulos sobre la base utilizando silicona, y siguiendo la disposición de la Imagen N° 14. Luego se soldaron los cables necesarios, utilizando cables de 0,5 mm² en las secciones de mayor corriente. Se asilaron los puntos expuestos y se gestionaron los cables utilizando precintos. Se obtuvo el resultado de la Imagen N° 21. En la Imagen N° 22 se aclaran las ubicaciones de los componentes ocultos.

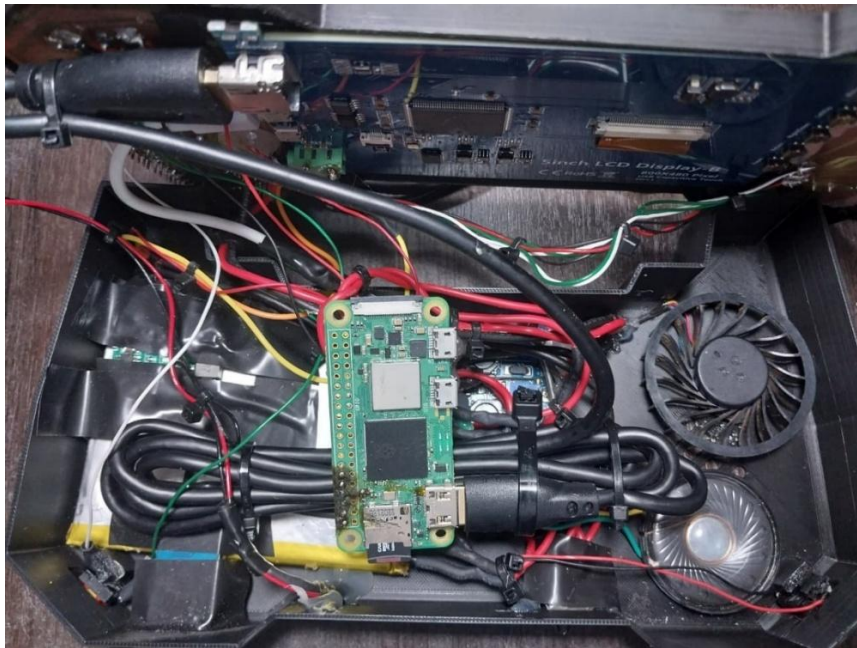


Imagen N° 21: Base con componentes dispuestos.

Fuente: Elaboración Propia.

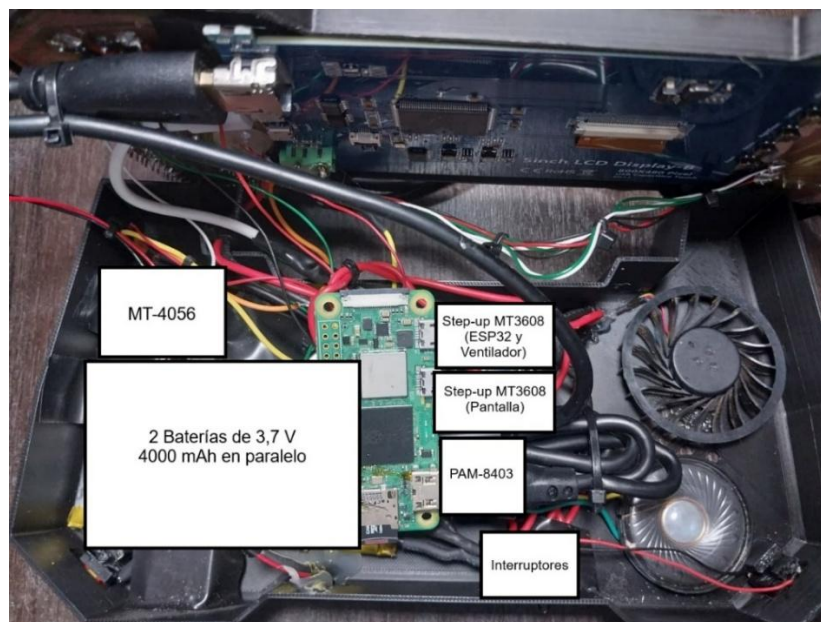


Imagen N° 22: Disposición de los componentes sobre la base del encapsulado.

Fuente: Elaboración Propia.

Por último, sobre la tapa se fijaron la pantalla y las placas correspondientes al *gamepad*. A continuación, se realizaron las conexiones eléctricas, enlazando los cables de alimentación y la señal *HDMI* provenientes desde la base, así como la provisión de energía hacia las placas impresas, completando de esta manera la integración del conjunto electrónico en el módulo superior. El resultado se expone en la Imagen N° 23.

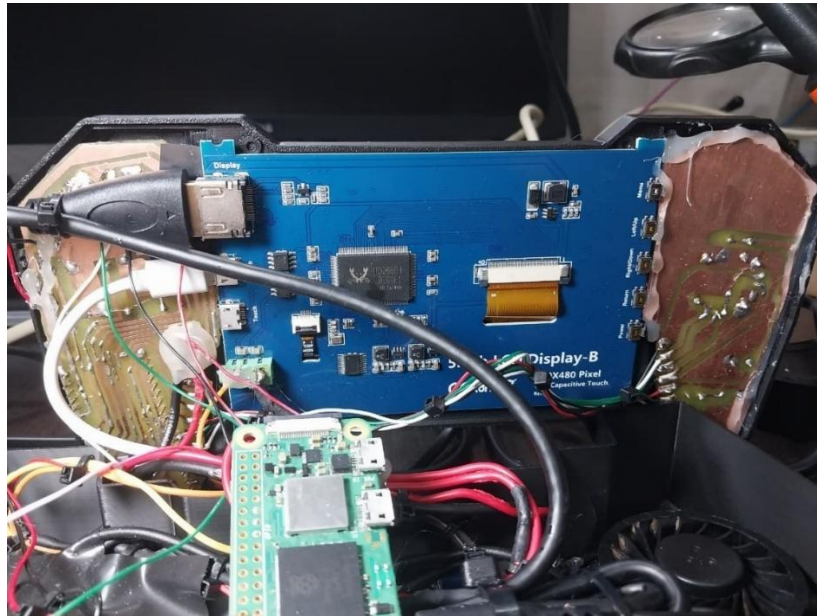


Imagen N° 23: Disposición de los componentes sobre la tapa del encapsulado.

Fuente: Elaboración Propia.



Imagen N° 24: Resultado Final.

Fuente: Elaboración Propia.

2.5 Desarrollo del *Frontend* Personalizado

En la sexta etapa, *Desarrollo del Frontend Personalizado*, se implementó una interfaz gráfica propia en C, diseñada específicamente para el dispositivo. El *frontend* es una aplicación que actúa como interfaz gráfica principal del sistema y se encarga de gestionar la interacción del usuario, la visualización de los menús y la ejecución de los programas externos, como el emulador *RetroArch* y el servidor de importación. Para explicar su funcionamiento se presenta el diagrama de flujo del programa en la **¡Error! No se encuentra el origen de la referencia..**

Se comienza con la inicialización de todos los recursos necesarios, incluyendo las variables de control, la biblioteca gráfica *SDL*, el sistema de audio, las fuentes tipográficas y la ventana principal. Esta etapa prepara el entorno de ejecución y deja el sistema listo para entrar en el bucle principal, que es el núcleo del funcionamiento del *frontend*.

Una vez completada la inicialización, el programa entra en el bucle principal, el cual se ejecuta continuamente mientras la variable de control “*run*” sea verdadera. Este bucle permite que el *frontend* funcione de forma persistente y reactiva. Dentro de este ciclo, el sistema primero verifica si existen eventos de entrada. Los eventos son señales generadas por el sistema o por el usuario, como la pulsación de botones del *gamepad*, teclas del teclado, o acciones del sistema. Si se detecta un evento, este se lee y se analiza para determinar su tipo y decidir qué acción realizar.

Cuando el evento corresponde a la pulsación de un botón, el *frontend* evalúa el estado actual de la interfaz para determinar el comportamiento adecuado. Dependiendo de la pantalla activa, el botón puede utilizarse para navegar entre consolas, desplazarse entre videojuegos, modificar opciones, interactuar con el teclado virtual o iniciar procesos externos. Por ejemplo, si el usuario se encuentra en la pantalla de selección de videojuegos y selecciona uno, el *frontend* lanza *RetroArch* en un proceso paralelo utilizando una llamada a *fork*[71].

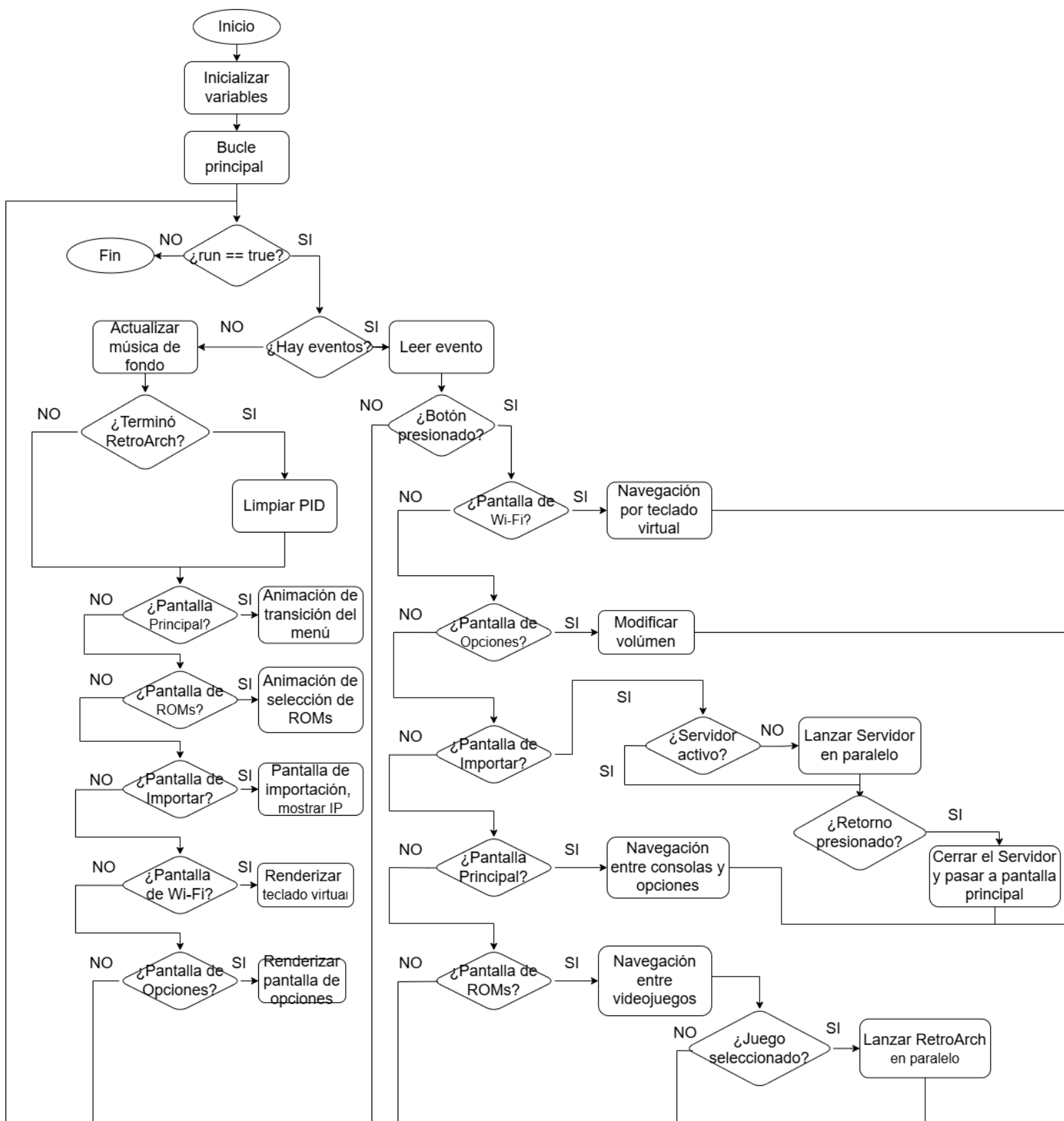


Imagen N° 25: Diagrama de flujo del frontend.

Fuente: Elaboración Propia.

Esto permite que el *frontend* y *RetroArch* existan como procesos independientes. El *frontend* registra el identificador del proceso hijo para poder monitorear su estado posteriormente.

Mientras el bucle principal continúa ejecutándose, el *frontend* actualiza constantemente la música de fondo y verifica si *RetroArch* sigue en ejecución. Esta verificación se realiza mediante el identificador del proceso almacenado.

Si el proceso ya terminó, el *frontend* limpia el identificador, lo que indica que el emulador ya no está activo y el control vuelve completamente a la interfaz. Este mecanismo permite que el *frontend* detecte automáticamente cuándo el usuario sale del emulador.

Además de gestionar los eventos de entrada provenientes del usuario, el *frontend* es responsable de actualizar continuamente el estado visual del sistema y renderizar la interfaz gráfica correspondiente en función del estado actual de la aplicación. Uno de los estados principales es el menú principal, donde se muestran las distintas consolas disponibles en el sistema. En esta pantalla, el *frontend* renderiza el fondo correspondiente, el logotipo de la consola seleccionada y su nombre, aplicando animaciones de transición cuando el usuario navega entre diferentes opciones. Estas animaciones se calculan en función del tiempo transcurrido, modificando progresivamente la posición y la transparencia de los elementos gráficos para generar un efecto visual suave de desplazamiento lateral. Este proceso implica el cálculo de *offsets* horizontales y valores de *alpha*, que permiten interpolar entre el elemento anterior y el nuevo elemento seleccionado.

Otro estado importante es la pantalla de lista de videojuegos (*ROMs*), donde el sistema muestra los juegos disponibles dentro de la consola seleccionada. En este caso, el *frontend* renderiza las imágenes de portada (*boxart*) de los juegos, así como sus nombres. De forma similar al menú principal, el cambio entre diferentes juegos también utiliza un sistema de animación basado en interpolación temporal, que permite desplazar la portada anterior fuera de la pantalla mientras la nueva portada entra progresivamente. Además, si no existen juegos disponibles, el sistema muestra un mensaje informativo indicando que no hay *ROMs*, evitando así un estado visual vacío o confuso.

El *frontend* también incluye una pantalla de importación de juegos, cuya función es permitir al usuario transferir nuevos archivos al sistema.

En este estado, el *frontend* renderiza un fondo específico, muestra un mensaje informativo indicando que el sistema está listo para importar juegos y obtiene dinámicamente la dirección *IP* del dispositivo mediante la ejecución de un comando del sistema operativo.

Esta dirección *IP* se muestra en pantalla para que el usuario pueda acceder al servidor de importación desde otro dispositivo a través de un navegador *web*. En caso de que no exista conexión a la red, el sistema muestra un mensaje alternativo indicando que es necesario conectarse a internet. Este proceso permite integrar funcionalidades de red dentro de la interfaz gráfica del *frontend*.

Otro estado corresponde a la pantalla de configuración *Wi-Fi*, donde el *frontend* muestra un teclado virtual que permite al usuario introducir datos de red utilizando el controlador. En esta pantalla, el *frontend* renderiza todos los elementos gráficos necesarios para representar el teclado, el texto introducido y las opciones disponibles. La interacción del usuario provoca la actualización del contenido mostrado, lo que requiere un renderizado continuo para reflejar los cambios en tiempo real.

Por último, el sistema incluye una pantalla de opciones, donde se muestra información del sistema como el volumen, el estado de la batería y la versión del *software*. En este estado, el *frontend* renderiza los textos correspondientes y permite modificar el volumen.

Finalmente, para garantizar la ejecución automática de la interfaz gráfica tras el encendido del dispositivo, se implementó una configuración de autoarranque basada en el perfil del usuario local. Aprovechando la configuración de inicio de sesión automático (*auto-login*) del sistema operativo hacia la terminal principal, se intervino el archivo de inicialización de entorno *.bash_profile*. Dentro de este archivo (el cual es interpretado y ejecutado por la terminal de comandos de forma desatendida inmediatamente al iniciar la sesión), se añadieron las instrucciones necesarias para navegar al directorio raíz del proyecto y realizar la invocación secuencial de los binarios compilados en C. Específicamente, se instruyó la ejecución en primera instancia de la animación gráfica de inicio (*./intro*), seguida de forma ininterrumpida por el lanzamiento del proceso principal del *frontend* (*./main*). Este método de configuración asegura que la plataforma opere como una consola dedicada, enlazando el encendido del *hardware* con el entorno interactivo sin requerir ningún tipo de intervención manual por parte del usuario final.

Capítulo III - Resultados

3.1 Conexión a Internet del Módulo Principal

El primer resultado obtenido corresponde a la verificación integral del sistema de conexión inalámbrica del módulo principal. Para garantizar una evaluación objetiva del comportamiento de la interfaz *Wi-Fi*, la prueba se inició con la desconexión forzada de la *Raspberry Pi Zero 2W*, operación realizada utilizando la herramienta de administración de redes *nmcli*. Mediante esta utilidad se eliminaron los perfiles previamente almacenados y se deshabilitó temporalmente el enlace inalámbrico, asegurando que no existiera ninguna asociación residual con redes conocidas. A continuación, como se indica en la Imagen N° 26, se ejecutó un comando de *ping* desde la terminal, obteniéndose inmediatamente el mensaje “*Network unreachable*”, lo cual confirmó la ausencia total de conectividad y estableció el estado inicial necesario para evaluar la capacidad del *frontend* de gestionar la reconexión.

```
ismael@raspberrypi:~/Frontend/project$ sudo nmcli connection delete Becker01
Connection 'Becker01' (22734f0d-8b90-44f8-b158-24c6345558ec) successfully deleted.
ismael@raspberrypi:~/Frontend/project$ ping 8.8.8.8
ping: connect: Network is unreachable
```

Imagen N° 26: Dispositivo sin acceso a internet.

Fuente: Elaboración Propia.

Posteriormente, se inició el *frontend* desarrollado específicamente para la consola y, desde su interfaz de configuración de red, en la Imagen N° 27, se procedió a cargar el *SSID* y la contraseña de la red *Wi-Fi* doméstica. Este proceso permitió evaluar no solo la capacidad del *frontend* para recibir y almacenar credenciales, sino también su integración con el sistema operativo para activar nuevamente la interfaz inalámbrica y solicitar la conexión correspondiente.



Imagen N° 27: Interfaz de conexión a internet.

Fuente: Elaboración Propia.

Tras completar la reconexión a través del *frontend*, se retornó a la terminal para realizar una verificación independiente del estado de la red.

Se ejecutó un *ping* hacia el servidor público 8.8.8.8, obteniendo respuestas exitosas y tiempos de latencia estables, como se indica en la Imagen N° 28, lo que confirmó que la *Raspberry Pi* había recuperado conectividad total a Internet.

Además, durante esta verificación se constató que la intensidad de la señal y la estabilidad del enlace no se vieron afectadas por el método de reconexión utilizado. El resultado general de la prueba permitió concluir que el sistema es capaz de desconectarse y reconectarse de manera controlada, tanto mediante herramientas de bajo nivel del sistema operativo como a través del *frontend*.

```
ismael@raspberrypi:~/Frontend/project$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=111 time=152 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=111 time=29.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=111 time=32.4 ms
```

Imagen N° 28: Prueba de acceso a internet.

Fuente: Elaboración Propia.

3.2 Carga de ROMs en el Sistema

Para evaluar el proceso de carga de contenido, se utilizó la funcionalidad del *frontend*, presentada en la Imagen N° 29, que habilita a la *Raspberry Pi* a comportarse como un servidor local accesible desde cualquier dispositivo conectado a la misma red.



Imagen N° 29: Carga de una ROM desde el servidor de Flask.

Fuente: Elaboración Propia.

Desde un navegador externo se ingresó a la interfaz de carga de la **¡Error! No se encuentra el origen de la referencia.**, donde el usuario puede seleccionar un archivo ROM de su propiedad, indicar a qué consola pertenece y enviarlo al módulo principal.

Este procedimiento se llevó a cabo exitosamente para cada una de las consolas soportadas, logrando transferir al menos un título de prueba por sistema. Una vez completada la carga de cada archivo, el *frontend* incorporó automáticamente el contenido a la biblioteca disponible, mostrándolo en el menú correspondiente de cada consola.

El proceso se caracterizó por tiempos de transferencia breves, una correcta identificación del contenido y total compatibilidad con los formatos utilizados. No se registraron errores de lectura ni fallos en la interfaz de usuario, confirmando que la herramienta de carga remota funciona de manera fiable y coherente con los requerimientos del proyecto.

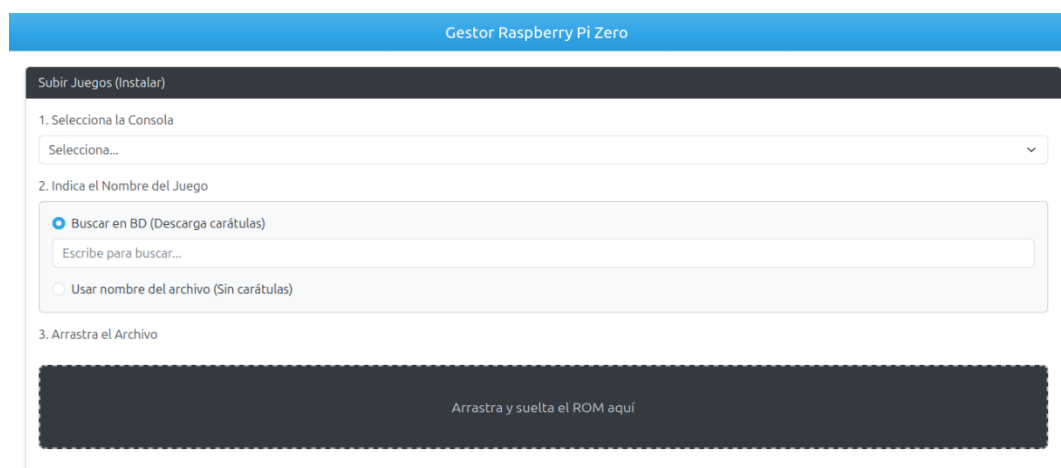


Imagen N° 30: Carga de una ROM desde el servidor de Flask.

Fuente: Elaboración Propia.

Desde un navegador externo se ingresó a la interfaz de carga, donde el usuario puede seleccionar un archivo *ROM* de su propiedad, indicar a qué consola pertenece y enviarlo al módulo principal. Este procedimiento se llevó a cabo exitosamente para cada una de las consolas soportadas, logrando transferir al menos un título de prueba por sistema. Una vez completada la carga de cada archivo, el *frontend* incorporó automáticamente el contenido a la biblioteca disponible, mostrándolo en el menú correspondiente de cada consola.

Para asegurar la estabilidad del sistema al procesar archivos de gran volumen, se implementó una arquitectura de transferencia por flujo continuo (*streaming*). En lugar de almacenar el archivo completo en la memoria *RAM* antes de su guardado, lo cual resultaba crítico en hardware de recursos limitados, el servidor ahora captura los datos entrantes en pequeños bloques (*chunks*) y los escribe de forma directa y secuencial en el almacenamiento secundario.

Gracias a esta optimización, el consumo de memoria volátil se mantiene mínimo y constante, en consecuencia, no se registraron errores por desbordamiento de memoria ni fallos en la interfaz de usuario, confirmando que la herramienta de carga remota funciona de manera fiable, escalable y coherente con los requerimientos del proyecto.

El proceso, como se muestra en la Imagen N° 31, se caracterizó por tiempos de transferencia breves, una correcta identificación del contenido y total compatibilidad con los formatos y tamaños utilizados.

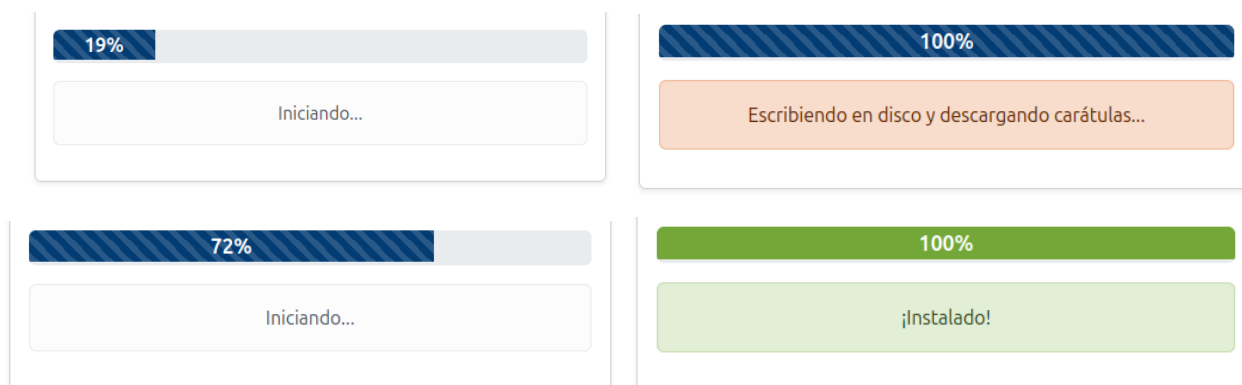


Imagen N° 31: Correcta carga de la ROM al frontend.

Fuente: Elaboración Propia.

Adicionalmente, se agregó al servidor de *Flask* las opciones necesarias para eliminar ROMs de la consola, así como un menú que permite al usuario agregar o quitar canciones al menú de inicio del *frontend*. Esto último se grafica con la Imagen N° 32 y en la Imagen N° 33.



Imagen N° 32: Menú de eliminación de juegos.

Fuente: Elaboración Propia.

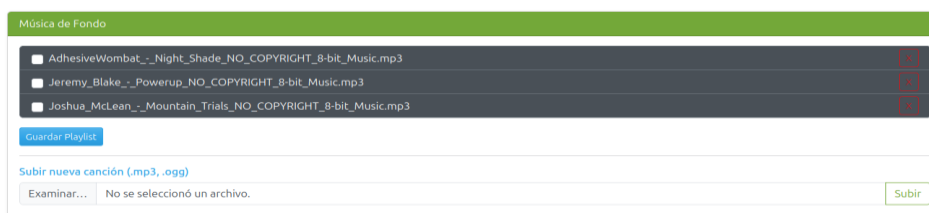


Imagen N° 33: Menú gestor de canciones.

Fuente: Elaboración Propia.

3.3 Pruebas de Control de Volumen

Con el fin de verificar el funcionamiento del sistema de modificación del volumen en la pantalla de opciones de la Imagen N° 34, se accedió a la misma y, mediante movimientos verticales del *stick*, se logró ajustar satisfactoriamente el nivel de sonido, tanto en su reducción como en su incremento. Asimismo, los niveles de carga de la batería coincidieron correctamente en situaciones en las que esta se encontraba totalmente cargada (próxima al 100 %) o a punto de apagarse (próxima al 0 %).

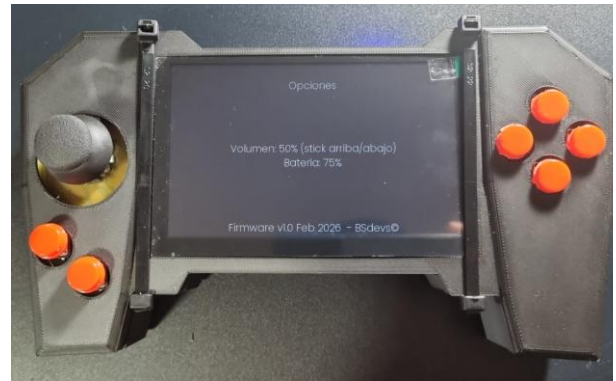


Imagen N° 34: Pantalla de Opciones.

Fuente: Elaboración Propia. Pruebas de Ejecución de Videojuegos

En la Imagen N° 35 se presenta la carátula del videojuego transferido mediante el sistema de servidor local. Como se puede notar, la misma corresponde al videojuego previamente seleccionado. El resultado del procedimiento ha sido efectivamente satisfactorio.



Imagen N° 35: Correcto Funcionamiento del sistema.

Fuente: Elaboración Propia.

Tras la carga de diferentes *ROM* 's desde el navegador, se procedió a ejecutar un juego de cada consola con el fin de evaluar el rendimiento del *backend* de emulación y la estabilidad general del sistema. En todos los casos, los juegos iniciaron correctamente desde el *frontend* personalizado, el cual invoca a *RetroArch*, cargando el núcleo correspondiente sin retrasos perceptibles. Los títulos de *NES* y *SNES* mostraron una ejecución fluida, sin caídas de fotogramas y con total sincronización entre imagen y sonido. Las pruebas realizadas en *Sega Genesis* y *Game Boy Advance* mostraron resultados equivalentes, manteniendo un funcionamiento estable incluso durante escenas con mayor demanda gráfica. En los casos de *PlayStation 1* y *Nintendo 64*, la ejecución del título de prueba evidenció un rendimiento adecuado, confirmando la correcta configuración del núcleo correspondiente y la compatibilidad con el *hardware* disponible. Durante todas las pruebas se validó la respuesta del *gamepad*, cuyo comportamiento inalámbrico mostró tiempos de reacción inmediatos y ausencia de desconexiones.

Sega Génesis

En primer lugar, en la Imagen N° 36 se indica que se probó la emulación de la *Sega Genesis* ejecutando el juego *Sonic the Hedgehog* con total fluidez, sin caídas de rendimiento, reflejando el correcto funcionamiento del núcleo de *Sega Genesis*.



Imagen N° 36: *Sonic the Hedgehog* – *Sega Genesis*

Fuente: *Elaboración Propia.*

Nintendo Entertainment System

En segundo lugar, en la Imagen N° 37, se presenta la evaluación la emulación de la NES mediante *Super Mario Bros*, el cual cargó de forma inmediata y mantuvo una ejecución estable durante toda la prueba, sin retrasos ni desincronizaciones perceptibles.



Imagen N° 37: Super Mario Bros – NES.

Fuente: Elaboración Propia.

Super Nintendo Entertainment System

A continuación, se verificó el rendimiento en SNES utilizando *Megaman X*, mostrando una ejecución suave, sin errores visuales y con una respuesta precisa del *gamepad*. Se adjunta la Imagen N° 38.



Imagen N° 38: Megaman X – SNES.

Fuente: Elaboración Propia.

Nintendo 64

Para *Nintendo 64* se empleó *Mario Kart 64*, comprobando un desempeño adecuado para un sistema más exigente, con estabilidad en los fotogramas y correcta adaptación del núcleo al *hardware*. Los pasos se presentan en la Imagen N° 39.



Imagen N° 39: Mario Kart 64 – Nintendo 64.

Fuente: Elaboración Propia.

Game Boy/Game Boy Color/Game Boy Advance

En el caso de *Game Boy Advance*, se probó *Pokémon Rojo Fuego*, observándose una emulación ligera y estable, con tiempos de carga instantáneos y una reproducción fiel de imagen. El resultado del procedimiento se muestra en la Imagen N° 40.



Imagen N° 40: Pokémon Edición Rojo Fuego – Game Boy/Game Boy Color/Game Boy Advance.

Fuente: Elaboración Propia.

PlayStation 1

Finalmente se evaluó la emulación de *PlayStation 1* con *Winning Eleven*, donde se constató un rendimiento sólido, tiempos de carga apropiados y emulación lo suficientemente estable para el núcleo más demandante de los integrados en la consola. El resultado es visible en la Imagen N° 41.



Imagen N° 41: *J-League Jikkyō Winning Eleven – PlayStation 1.*

Fuente: Elaboración Propia.

3.4 Autonomía de la Batería

Como parte de la validación del sistema, se efectuaron pruebas exhaustivas de autonomía con el propósito de evaluar de manera precisa la duración real de la batería del módulo principal bajo un escenario de uso continuo y exigente. Antes de iniciar la prueba prolongada, se decidió desmontar parcialmente el módulo para acceder físicamente al sistema de alimentación y realizar mediciones directas de tensión, tanto en los terminales de la batería como en la línea regulada que alimenta a la *Raspberry Pi Zero 2W*. Estas mediciones permitieron obtener datos para validar el desempeño eléctrico del sistema.

En primer lugar, como se indica en la Imagen N° 42, se constató que la batería de polímero de litio presentaba una tensión de 3,72 V, valor coherente con un nivel de carga intermedio y representativo del comportamiento típico de este tipo de celdas. En segundo lugar, se midió la tensión suministrada por el módulo *TP4056* con convertor *step-up* integrado, registrándose un valor estable de 5,53 V en la entrada de alimentación de la *Raspberry Pi*. Este resultado confirmó que el sistema regulador no solo elevaba correctamente el voltaje, sino que lo mantenía dentro de un rango seguro y continuo, adecuado para garantizar la estabilidad del procesador, la pantalla *HDMI* durante largos períodos de funcionamiento.

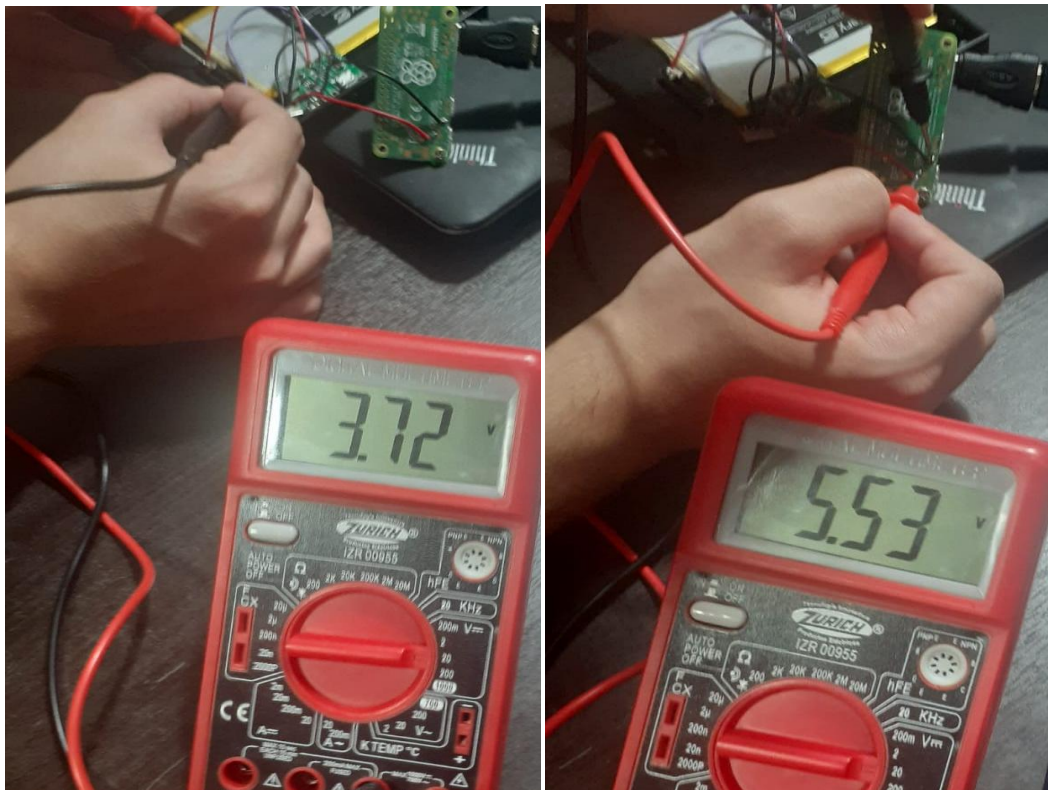


Imagen N° 42: Prototipo con alimentación autónoma desmontado para la medición de tensiones.

Fuente: Elaboración Propia.

Por último, en una etapa posterior del desarrollo, fueron realizadas pruebas de autonomía del sistema de alimentación, disponiéndose la consola completamente cargada y ejecutándose un juego de manera continua hasta que se produjo su apagado por descarga total de las baterías.

Estas pruebas resultan representativas de una condición real de uso, ya que el procesador fue sometido a carga sostenida, manteniéndose en funcionamiento simultáneo el ventilador, la pantalla, los controles y el sistema de audio.

En este escenario de operación continua, fue verificado que el sistema de alimentación autónoma permitió sostener el funcionamiento del dispositivo durante un período superior a tres horas antes de agotarse completamente, resultado que se encuentra en concordancia con las proyecciones teóricas previamente obtenidas a partir del consumo promedio estimado de aproximadamente 2400 mA, considerando el factor de seguridad aplicado en el dimensionamiento. El procedimiento se presenta en la Imagen N° 43. Además, fue registrado mediante video, y se encuentra disponible en *YouTube*[72].



Imagen N° 43: Pruebas de duración de la batería.

Fuente: Elaboración Propia.

Como conclusión de estas pruebas, el ensayo de descarga ininterrumpida arrojó un tiempo exacto de operación de 3 horas y 22 minutos bajo condiciones de uso práctico de manera sostenida. Este resultado exhibe un alto grado de correlación con la proyección calculada inicialmente, la cual estimaba una duración teórica de 3 horas y 33 minutos. La mínima desviación observada entre el cálculo de diseño y el comportamiento real del *hardware* permite validar definitivamente la robustez del sistema de alimentación. En consecuencia, se confirma que el dimensionamiento de las celdas de litio y la eficiencia de las etapas de regulación garantizan un rendimiento energético estable, predecible y acorde a las especificaciones planteadas para el dispositivo.



Imagen N° 44: Dispositivo final ensamblado sostenido por el usuario para evidenciar su ergonomía y factor de forma.

Fuente: Elaboración Propia.

Capítulo IV - Conclusiones

4.1 Características de la consola desarrollada

Tras el análisis del prototipo funcional y su contraste con las alternativas comerciales disponibles, se concluye que el desarrollo presenta resultados altamente satisfactorios, consolidándose como una alternativa sólida y capaz.

	Consola: Propia		Valoración
	Precio	\$250.000**	7.0 pts
	Almacenamiento	64 GB	7.0 pts
	Pantalla	5''	8.5 pts
	Resolución	800 x 640	7.5 pts
	Batería	8000 mAh	10.0 pts
	Wi-Fi	Si	10.0 pts
	Dimensiones	22,5 x 12,5 x 4,5 cm	8.5 pts
	Peso	581 g	7.0 pts
	Emulación	Sí (hasta PS1)	9.0 pts
	Exclusivos	Si	10.0 pts
Valoración Final Ponderada			8.40 pts

Tabla N° 8: Características ponderadas de la consola desarrollada.

Fuente: Elaboración Propia.

****Precio:** Este precio se calcula en función del costo minorista de materiales a la fecha del 24/02/2026, los cuales son los siguientes:

Raspberry Pi Zero 2 W: ARS 39.118,50

NodeMCU32-S: ARS 9.449,00

Pantalla LCD 5'' 800 X 480 HDMI: ARS 69.428,00

Batería Litio polímero 3.7v 4000mah 606090: 2 × ARS 26.939,00

Filamento Para Impresoras 3d Pla × 1kg: ARS 23.650,00

Placa cobre virgen 10 × 15 cm: ARS 5.000,00

Modulo PAM8403 y bocina: ARS 3.990,00

Stick Analógico + 8 Botones: ARS 3.512,60

Este dictamen se fundamenta en los siguientes aspectos:

Viabilidad Económica y Comercial

El dispositivo logra competitividad directa en el mercado nacional con un costo de fabricación adecuado. Es imperativo señalar que dicho costo está supeditado a la adquisición local de componentes. Si se implementara una cadena de suministro basada en la importación directa de los insumos, los costos de producción experimentarían una reducción significativa, posicionando al dispositivo competitivamente a escala internacional.

Prestaciones de *Hardware*

El equipo no presenta deficiencias técnicas frente a marcas consolidadas. Destaca por integrar una pantalla de excelente resolución y tamaño, junto con una capacidad computacional capaz de sostener la emulación fluida de sistemas de 32 bits (hasta *PlayStation 1*).

Adicionalmente, cuenta con un almacenamiento inicial de 64 GB (fácilmente expandible) y una autonomía energética superior a la media del segmento.

Experiencia de Usuario (*Software*): A diferencia de las demás consolas de la competencia que recurren a la interfaz genérica de *RetroArch*, el proyecto integra una interfaz gráfica de usuario (*frontend*) desarrollada a medida.

4.2 Dificultades en el desarrollo

Durante el proceso de diseño e implementación del sistema, se presentaron diversos desafíos técnicos que requirieron un análisis exhaustivo y la adopción de soluciones específicas para garantizar el funcionamiento estable del dispositivo.

Filtro pasa bajos para mitigación de ruido en la salida de audio

Uno de los primeros inconvenientes detectados estuvo relacionado con la integridad de la señal de audio analógica proveniente de la salida auxiliar de la pantalla, la cual presentaba un nivel significativo de ruido de alta frecuencia superpuesto a la señal útil, perceptible como un “pitido”.



Imagen N° 45: Espectrograma de la señal de sonido en el tiempo.

Fuente: Elaboración Propia.

Este fenómeno degradaba la calidad sonora e introducía componentes espurios no deseados, tal como se evidencia en los espectros de frecuencia obtenidos experimentalmente, donde se observan componentes de alta frecuencia con niveles cercanos a los -30 dBFS (*decibels Full Scale*). Este comportamiento es característico de interferencias de origen digital y acoplamientos electromagnéticos internos del propio módulo de visualización.

Para resolver este inconveniente, se diseñó e implementó un filtro pasa bajos pasivo, dimensionado específicamente para atenuar las componentes de alta frecuencia, seleccionando una frecuencia de corte próxima a los 750 Hz.

El efecto del filtro se puede percibir en la Imagen N° 45. Al momento de aplicar el filtro pasa bajos, las componentes de alta frecuencia que previamente poseían niveles cercanos a los -30 dBFS se atenúan hasta valores próximos a los -60 dBFS.

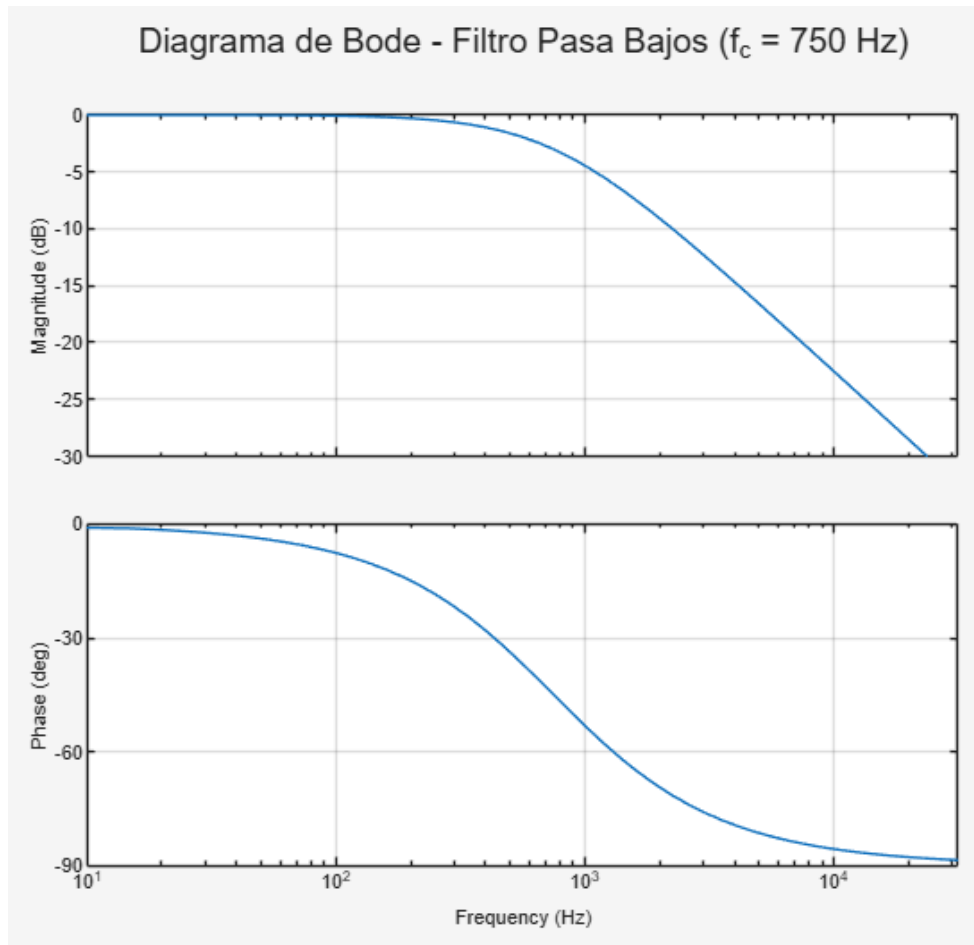


Imagen N° 46: Diagrama de Bode del filtro pasa bajos

Fuente: Elaboración Propia.

Si bien un filtro de 750 Hz puede aparentar muy agresivo, al ser este de primer orden como se ve en la Imagen N° 46: Diagrama de Bode del filtro pasa bajos, este reduce de manera leve las frecuencias bajas donde se encuentran los efectos de sonido y la música, y reduce de manera más drástica en frecuencias altas, cumpliendo el objetivo de mitigar los efectos del ruido.

Problemas de Alimentación

Inicialmente se intentó alimentar la totalidad del sistema mediante un único módulo basado en el circuito *TP4056*, el cual demostró ser capaz de suministrar energía de manera estable a la *Raspberry Pi* cuando operaba de forma exclusiva. Sin embargo, al incorporar cargas adicionales, como la pantalla y otros periféricos, se producía una caída significativa de tensión debido a la limitación de corriente del módulo, provocando reinicios intempestivos y la pérdida de estabilidad del sistema.

Por otro lado, los convertidores elevadores de tensión basados en el circuito *MT3608*, si bien eran adecuados para cargas de menor consumo, no lograban proporcionar una alimentación suficientemente estable para la *Raspberry Pi* debido a las fluctuaciones de tensión ante demandas transitorias de corriente.

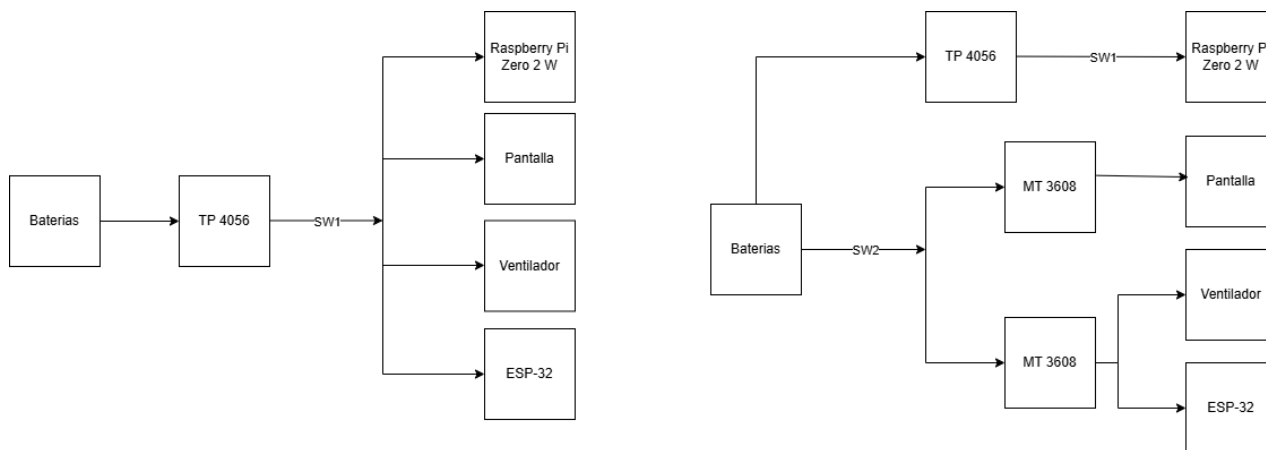


Imagen N° 47: Disposición inicial (izquierda), y disposición final (derecha) del sistema de alimentación.

Fuente: Elaboración Propia.

Como solución definitiva, se adoptó una arquitectura de alimentación distribuida, en la cual el módulo *TP4056* se destinó exclusivamente a la alimentación de la *Raspberry Pi*, garantizando su estabilidad operativa. De manera complementaria, se implementaron convertidores elevadores *MT3608* independientes para alimentar la pantalla y, de forma separada, el subsistema compuesto por los ventiladores y el microcontrolador *ESP32*.

Esta solución implica la necesidad de utilizar 2 interruptores diferentes para poder encender o apagar el sistema completo, a diferencia del planteo inicial, donde solo se requería un único interruptor. Además, se necesitan 2 módulos *MT-3608*, por lo cual se disminuye el margen de espacio disponible en el encapsulado. La situación inicial y final se representan con el diagrama de la Imagen N° 47.

Problemas de integridad de señal en la interconexión HDMI

Durante las primeras etapas de integración, se detectaron fallas intermitentes en la transmisión de video entre la *Raspberry Pi* y la pantalla. La configuración original utilizaba un cable *HDMI* estándar de 15 cm en conjunto con un adaptador a *mini HDMI*, necesario debido al formato del conector de la *Raspberry Pi*.

Esta disposición introducía una interfaz mecánica y eléctrica adicional que degradaba la integridad de la señal digital de alta velocidad, generando pérdidas intermitentes de sincronización y fallos en la inicialización de la pantalla. El análisis de este comportamiento permitió identificar al adaptador y al cable como el principal punto de falla, debido a posibles problemas de impedancia, calidad de contacto y susceptibilidad a interferencias. La solución implementada consistió en reemplazar este conjunto por un cable único con conectores nativos *HDMI* a *mini HDMI*, eliminando completamente el uso de adaptadores intermedios. Esta modificación permitió restablecer la continuidad de la señal y garantizar una transmisión confiable, eliminando por completo las fallas de inicialización y logrando un funcionamiento estable y consistente del subsistema de video.

Problemas de vinculación *Bluetooth* y arquitectura del control

El subsistema de control inalámbrico presentó inicialmente dificultades asociadas al establecimiento automático de la conexión *Bluetooth* entre la *Raspberry Pi* y el microcontrolador *ESP32-C3*[73]. En esta configuración, el proceso de vinculación debía realizarse manualmente en cada encendido mediante comandos específicos del gestor *Bluetooth* del sistema operativo, lo que comprometía la autonomía operativa del dispositivo.

Adicionalmente, se exploró el uso de comunicación serial mediante interfaz *UART*[74] como alternativa, sin embargo, esta implementación presentó problemas de estabilidad, manifestándose en desconexiones aleatorias y comportamiento inconsistente durante el funcionamiento.

Como solución definitiva, se optó por migrar hacia una plataforma basada en el microcontrolador *ESP32-NodeMCU*[75], el cual demostró una compatibilidad significativamente superior con el sistema anfitrión, transparente para el usuario, eliminando la necesidad de intervención manual y garantizando un funcionamiento robusto y confiable del sistema de control.

4.3 Mejoras Futuras

Videojuegos Exclusivos

Se ha desarrollado un prototipo básico de videojuego utilizando *GameMaker*, confirmando plenamente la compatibilidad de la consola con motores de desarrollo modernos. Esta demostración no solo valida la ejecución de juegos exclusivos y propietarios, sino que también abre un horizonte lleno de oportunidades para expandir el ecosistema de la plataforma, convirtiéndola en un terreno fértil para desarrolladores y proyectos innovadores en el mundo de los videojuegos. *GameMaker* permite exportar proyectos a múltiples sistemas operativos, incluyendo *Windows*, *Ubuntu*, *macOS*, *Android* y *HTML5*. El prototipo exportado para *Ubuntu* funcionó correctamente en *Raspberry Pi OS*, lo que permitió avanzar en las pruebas de compatibilidad y validar el comportamiento del motor de juegos en la plataforma objetivo.

Para la ejecución del prototipo en la *Raspberry Pi*, se utilizó el sistema de *SSH* de *GameMaker* para transferir el proyecto y establecer la conexión remota, facilitando la gestión del desarrollo desde el entorno de *GameMaker*.

Dado que la *Raspberry Pi* no cuenta por defecto con un escritorio gráfico activo, fue necesario instalar *startx* para habilitar la interfaz gráfica y permitir la ejecución del juego. Los resultados técnicos obtenidos fueron satisfactorios, el prototipo se ejecutó correctamente en la *Raspberry Pi Zero 2 W*, confirmando la viabilidad del motor de desarrollo para la consola. Además, se verificó la compatibilidad del *gamepad* con *GameMaker*, asegurando que los controles respondieran de manera adecuada y permitieran una experiencia de usuario funcional. La prueba fue documentada mediante registro de video en *YouTube*[76].

Mejoras de *Hardware*

En lo que respecta al *gamepad*, una mejora sustancial proyectada para futuras iteraciones consiste en la incorporación de un segundo par de botones de acción superior (equivalentes a los gatillos secundarios L2 y R2). Esta adición técnica permitiría replicar con mayor fidelidad la disposición estructural de los controladores originales pertenecientes a consolas de quinta generación, tales como *PlayStation 1* y *Nintendo 64*. Con ello, se garantizaría una retrocompatibilidad absoluta con el catálogo de *software* de dichas plataformas.

Otra mejora planteada sería implementar una salida de video desde la consola, permitiendo que se pueda utilizar la consola con una pantalla externa de mayor tamaño.

Mejoras de Software

Entre las mejoras de *software* proyectadas para futuras iteraciones, se plantea la implementación de un sistema que informe al usuario sobre el estado de carga de la batería durante la ejecución de los juegos. Si bien esta funcionalidad resulta conceptualmente sencilla, su desarrollo técnico reviste cierta complejidad gráfica. Al operar sobre un sistema operativo sin entorno de escritorio (donde la emulación toma control exclusivo de la salida de video), superponer elementos gráficos (*overlay*) sobre el renderizado activo requiere una manipulación de bajo nivel del *framebuffer*. Alternativamente, requeriría el desarrollo de un servicio en segundo plano que se comunique en tiempo real con la interfaz de programación (API) de *RetroArch*, inyectando los datos de telemetría energética a través de su sistema nativo de notificaciones en pantalla (OSD - *On-Screen Display*), garantizando así que no se penalicen los recursos de la *CPU* ni la fluidez de la emulación.

Mejoras de diseño

En cuanto a la arquitectura física del dispositivo, se identifican dos áreas principales de optimización para futuras revisiones. En primer lugar, a causa de las restricciones logísticas para la obtención de conectores de bajo perfil y cableado a medida en el mercado local, el enrutamiento interno de las conexiones eléctricas presenta un margen de mejora significativo. Una gestión de cableado más eficiente permitiría compactar el empaquetamiento electrónico y reducir de manera considerable el grosor general de la carcasa. En segundo lugar, desde la perspectiva del diseño mecánico asistido por computadora (*CAD*), existe viabilidad para refinar la geometría exterior de la consola. Mediante iteraciones avanzadas de modelado tridimensional, se proyecta perfeccionar la ergonomía del chasis, adaptando las superficies de agarre a parámetros antropométricos más precisos para maximizar la comodidad del usuario durante sesiones de uso prolongadas.

Implementación de Modo Multijugador Local

Se propone la incorporación de un modo de dos jugadores que amplíe las capacidades del sistema actualmente diseñado. Desde el punto de vista de *hardware*, una alternativa consiste en desarrollar un segundo mando exclusivo y dedicado (sin pantalla),

basado en una arquitectura similar a la implementada en el *gamepad* embebido en la consola portátil, pero con un sistema de alimentación autónoma. Esta solución permitiría mantener la topología actual sin modificaciones sustanciales en el módulo principal, aprovechando la compatibilidad nativa de la *Raspberry Pi Zero 2 W* con múltiples dispositivos *HID*. Su implementación requeriría el diseño de un nuevo encapsulado tridimensional para un mando externo, aprovechando el mismo circuito diseñado para el *gamepad* basado en el microcontrolador *ESP32-NodeMCU*[75], con la integración de un esquema de alimentación propio, de menor complejidad y demanda energética que el utilizado en la consola portátil. Otra alternativa que no requiere modificación alguna en el *hardware*, consiste en implementar el modo multijugador directamente entre dos consolas portátiles, donde mediante una opción de configuración una de ellas asuma el rol de jugador principal y la otra actúe como segundo controlador, transmitiendo únicamente eventos de entrada. Esta última estrategia reduciría la necesidad de *hardware* adicional y fomentaría una arquitectura distribuida basada en comunicación inalámbrica entre dispositivos homogéneos. Ambas propuestas resultan técnicamente viables dentro del marco estructural existente y representan una evolución coherente del sistema hacia experiencias multijugador locales sin comprometer el diseño base.

4.4 Resumen de resultados obtenidos

El sistema obtenido no se limita a la ejecución de emuladores sobre *hardware* preexistente. Constituye una plataforma diseñada para operar como una unidad compacta, estable y escalable. Las pruebas de funcionamiento evidencian que las decisiones de diseño tomadas durante el desarrollo resultaron técnicamente consistentes y adecuadamente fundamentadas. Se confirmó el desempeño fluido, respuesta adecuada en la captura de entradas y ausencia de fallas críticas en condiciones normales de operación. La integración del *gamepad* demostró solidez en términos de diseño electrónico y comunicación. Asimismo, la compatibilidad nativa del sistema con múltiples dispositivos *HID* abre posibilidades de expansión y no compromete la topología actual.

Desde el punto de vista formativo, el proyecto permitió consolidar competencias en diseño de *PCB*, integración de sistemas embebidos, programación tanto a nivel de aplicación como de *firmware*, implementación y configuración de servidores para intercambio de datos, configuración de entornos *Linux*, y resolución de problemas asociados a ruido eléctrico, alimentación y comunicación inalámbrica. La necesidad de articular *hardware* y *software* en un entorno real puso en evidencia la importancia de la planificación estructural, la validación incremental y el análisis sistemático de fallas, fortaleciendo la capacidad de diagnóstico y toma de decisiones técnicas fundamentadas.

En conjunto, los resultados obtenidos confirman no solo la viabilidad técnica de la propuesta, sino también la madurez alcanzada en el proceso de diseño e implementación. El resultado obtenido corresponda a una base estable para futuras iteraciones, y permite la potencial evolución hacia funcionalidades adicionales sin requerir rediseños estructurales significativos.

Anexos

ANEXO A — Frontend

Arquitectura del *Frontend*

El *frontend* constituye la capa de interacción principal entre el usuario y la consola, funcionando como un entorno de operación autónomo que se ejecuta automáticamente al iniciar el sistema. El proyecto completo se encuentra disponible en *GitHub*[77].

Estructura del proyecto

El *frontend* se organiza en una estructura jerárquica de archivos y carpetas, separando la lógica de ejecución, los recursos gráficos y los módulos externos. La organización general se muestra a continuación:

```
1. project/
2. |
3. |
4. └─ bios/           Archivos BIOS requeridos por ciertos sistemas
5. └─ cores/          Núcleos de emulación (.so)
6. └─ images/         Recursos gráficos generales
7. └─ songs/          Archivos de audio
8. |
9. └─ Nintendo - Game Boy/
10. └─ Nintendo - Nintendo 64/
11. └─ Nintendo - Nintendo Entertainment System/
12. └─ Nintendo - Super Nintendo Entertainment System/
13. └─ Sega - Mega Drive - Genesis/
14. └─ Sony - PlayStation/
15. |
16. └─ main.c          Núcleo principal del frontend
17. └─ intro.c         Pantalla de inicio
18. └─ main.py         Lanzamiento de sistemas
19. └─
20. └─
21. └─
22. |
23. └─ *.txt           Catálogos de juegos por sistema
24. └─ *.png           Recursos gráficos individuales
25. └─ Poppins-ExtraLight.ttf Fuente tipográfica
26.
```

Descripción de módulos

- **Núcleo del frontend**

Archivo	Función
main.c	Implementa el bucle principal de ejecución, renderizado y navegación.
intro.c	Gestiona la pantalla de inicio del sistema.
main.py	Ejecuta los sistemas seleccionados y gestiona llamadas externas.

Tabla N° 9: Descripción de los módulos del frontend.

Fuente: Elaboración Propia.

- **Núcleos de emulación**

La carpeta `cores/` contiene bibliotecas dinámicas compatibles con la arquitectura *Libretro*, responsables de ejecutar los distintos sistemas soportados. Los mismos se han obtenido desde un repositorio público de compilaciones en *github*[78].

- `gambatte_libretro.so`
- `genesis_plus_gx_libretro.so`
- `mupen64plus_next_libretro.so`
- `pcsx_rearmed_libretro.so`
- `snes9x2002_libretro.so`

- **Organización por sistema**

Cada sistema posee su propia carpeta estructurada de la siguiente manera:

```
1. [Sistema]/
2.   └─ roms/           Juegos del sistema
3.   └─ Named_Boxarts/  Imágenes de carátulas
4.   └─ Named_Logos/    Logotipos
5.   └─ Named_Snaps/    Capturas de pantalla
6.   └─ Named_Titles/   Títulos gráficos
7.   └─ logo.png        Identificador visual del sistema
```

Esta estructura permite que el *frontend* cargue dinámicamente los recursos asociados a cada juego.

- **Catálogos de juegos**

Los archivos `.txt` presentes en la raíz del proyecto contienen el listado de juegos disponibles por sistema. Estos archivos permiten al *frontend* generar dinámicamente el menú de selección sin necesidad de modificar el código fuente.

- *Nintendo - Game Boy.txt*
- *Nintendo - Nintendo 64.txt*
- *Sega - Mega Drive - Genesis.txt*
- *Sony - PlayStation.txt*

A.3.5 Recursos gráficos y tipográficos

Los archivos gráficos utilizados por la interfaz incluyen:

- *logo.png*
- *options.png*
- *wifi.png*
- *import.png*

Asimismo, se utiliza la fuente tipográfica `Poppins-ExtraLight.ttf` para la renderización de texto en pantalla.

Códigos fuente

En las secciones siguientes se presenta el código correspondiente a los módulos centrales del sistema:

- **Código Principal - *main.c***

Su propósito es proporcionar una interfaz gráfica simple, rápida y de bajo consumo de recursos que permita navegar entre los distintos sistemas disponibles, visualizar información relevante del dispositivo y ejecutar aplicaciones sin intervención del sistema operativo subyacente. El *software* fue diseñado bajo un esquema modular, donde cada componente cumple una función específica: renderizado gráfico, lectura de controles, gestión de contenidos y ejecución de núcleos de emulación.

1.

```

1. // Variables globales para el estado de conexión WiFi
2. int show_connecting = 0;
3. unsigned int connecting_start = 0;
4. #include <SDL2/SDL.h>
5. #include <SDL2/SDL_image.h>
6. #include <SDL2/SDL_ttf.h>
7. #include <SDL2/SDL_mixer.h>
8. #include <dirent.h>
9. #include <stdbool.h>
10. #include <stdio.h>
11. #include <errno.h>
12. #include <string.h>
13. #include <stdlib.h>
14. #include <unistd.h>
15. #include <sys/types.h>
16. #include <sys/wait.h>
17. #include <signal.h>
18. #include <sys/stat.h>
19. #include <sys/types.h>
20. #include <sys/stat.h>
21. #include <unistd.h>
22. #include <fcntl.h>
23. #include <time.h>
24. #include <ctype.h>
25.
26. #define SCREEN_W 800
27. #define SCREEN_H 480
28. #define MAX_ITEMS 50
29. #define MAX_ROMS 1024
30. #define MAX_SONGS 256
31. #define BOXART_PLACEHOLDER "no_boxart.png"
32. // Ajusta este valor para cambiar el tamaño de los boxarts renderizados
33. #define BOXART_SCALE 0.55f
34. // Desplazamiento vertical para que las portadas comiencen más arriba
35. #define BOXART_Y_OFFSET -190
36. // Ratio para la posición Y del nombre del juego (por ejemplo 0.90 = 90% del alto)
37. #define GAME_NAME_Y_RATIO 0.90f
38.
39. // Constantes del teclado
40. const int KEYBOARD_X = 100;
41. const int KEYBOARD_Y = 100;
42. const int KEY_W = 60; // separación horizontal
43. const int KEY_H = 50; // separación vertical
44. const int BOX_W = 60; // ancho del selector
45. const int BOX_H = 50; // alto del selector
46.
47. typedef struct
48. {
49.     char name[256];
50.     char logoPath[512];
51.     SDL_Texture *logo;
52.     SDL_Color bg;
53.     bool isSpecial;
54.     char core[256]; // core asociado a la consola (ej: genesisplusgx.so)
55. } Entry;
56.
57. // Global renderer so helper functions can draw before/after launching RetroArch
58. static SDL_Renderer *global_renderer = NULL;
59.
60. // Estructura para ROMs
61. typedef struct
62. {
63.     char name[512]; // nombre de archivo (con extension)

```

```
64.     char path[1024];    // ruta completa a la rom
65.     char baseName[512]; // nombre sin extension
66.     char boxartPath[1024];
67.     SDL_Texture *boxart;
68. } RomEntry;
69.
70. // Estructura para canciones
71. typedef struct
72. {
73.     char path[1024];    // ruta relativa o absoluta al archivo de audio
74. } SongEntry;
75.
76. // -----
77. // Música de fondo (playlist aleatoria sin repetir)
78. // -----
79. static SongEntry playlist[MAX_SONGS];
80. static int playlist_count = 0;
81. static int playlist_order[MAX_SONGS];
82. static int playlist_pos = -1; // índice dentro de playlist_order
83. static Mix_Music *current_music = NULL;
84. static Uint32 last_music_check = 0;
85. static int master_volume = 50; // volumen maestro en % (0-100)
86.
87. void free_playlist()
88. {
89.     playlist_count = 0;
90.     playlist_pos = -1;
91.     if (current_music)
92.     {
93.         Mix_HaltMusic();
94.         Mix_FreeMusic(current_music);
95.         current_music = NULL;
96.     }
97. }
98.
99. void shuffle_playlist()
100. {
101.     if (playlist_count <= 1) return;
102.     for (int i = 0; i < playlist_count; ++i)
103.         playlist_order[i] = i;
104.     for (int i = playlist_count - 1; i > 0; --i)
105.     {
106.         int j = rand() % (i + 1);
107.         int tmp = playlist_order[i];
108.         playlist_order[i] = playlist_order[j];
109.         playlist_order[j] = tmp;
110.     }
111.     playlist_pos = -1;
112. }
113.
114. void load_playlist_file(const char *unused)
115. {
116.     (void)unused;
117.     free_playlist();
118.
119.     const char *songs_dir = "songs";
120.     DIR *d = opendir(songs_dir);
121.     if (!d) return;
122.
123.     struct dirent *de;
124.     while ((de = readdir(d)) != NULL)
125.     {
126.         const char *name = de->d_name;
127.         if (!name || name[0] == '.') continue;
128.
```

```
129.         const char *dot = strrchr(name, '.');
130.         if (!dot) continue;
131.         char ext[8];
132.         strncpy(ext, dot, sizeof(ext)-1);
133.         ext[sizeof(ext)-1] = '\0';
134.         for (char *p = ext; *p; ++p) *p = (char)tolower((unsigned char)*p);
135.
136.         if (strcmp(ext, ".mp3") != 0 && strcmp(ext, ".ogg") != 0 && strcmp(ext, ".wav") != 0)
137.             continue;
138.
139.         if (playlist_count >= MAX_SONGS) break;
140.
141.         char fullpath[1024];
142.         snprintf(fullpath, sizeof(fullpath), "%s/%s", songs_dir, name);
143.         strncpy(playlist[playlist_count].path, fullpath, sizeof(playlist[playlist_count].path)-1);
144.         playlist[playlist_count].path[sizeof(playlist[playlist_count].path)-1] = '\0';
145.         playlist_count++;
146.     }
147.     closedir(d);
148.
149.     if (playlist_count > 0)
150.     {
151.         shuffle_playlist();
152.     }
153. }
154.
155. void play_next_song()
156. {
157.     if (playlist_count == 0) return;
158.
159.     if (current_music)
160.     {
161.         Mix_HaltMusic();
162.         Mix_FreeMusic(current_music);
163.         current_music = NULL;
164.     }
165.
166.     if (playlist_pos + 1 >= playlist_count)
167.     {
168.         shuffle_playlist();
169.     }
170.     playlist_pos++;
171.     int idx = playlist_order[playlist_pos];
172.     const char *path = playlist[idx].path;
173.
174.     current_music = Mix_LoadMUS(path);
175.     if (!current_music)
176.     {
177.         fprintf(stderr, "Mix_LoadMUS failed for '%s': %s\n", path, Mix_GetError());
178.         return;
179.     }
180.     if (Mix_PlayMusic(current_music, 1) == -1)
181.     {
182.         fprintf(stderr, "Mix_PlayMusic failed for '%s': %s\n", path, Mix_GetError());
183.         Mix_FreeMusic(current_music);
184.         current_music = NULL;
185.     }
186. }
187.
188. void update_music()
189. {
190.     if (playlist_count == 0) return;
191.     Uint32 now = SDL_GetTicks();
192.     if (now - last_music_check < 500) return; // comprobar ~2 veces por segundo
193.     last_music_check = now;
```

```
194.
195.     if (!Mix_PlayingMusic())
196.     {
197.         play_next_song();
198.     }
199. }
200.
201. // -----
202. // Helpers: carga y render
203. // -----
204. SDL_Texture* load_png(SDL_Renderer *r, const char *path)
205. {
206.     if (!path) return NULL;
207.     struct stat st;
208.     if (stat(path, &st) != 0)
209.     {
210.         // file does not exist
211.         //fprintf(stderr, "load_png: file not found: %s\n", path);
212.         return NULL;
213.     }
214.     SDL_Surface *img = IMG_Load(path);
215.     if (!img)
216.     {
217.         fprintf(stderr, "load_png: IMG_Load failed for '%s': %s\n", path, IMG_GetError());
218.         return NULL;
219.     }
220.     SDL_Texture *tex = SDL_CreateTextureFromSurface(r, img);
221.     SDL_FreeSurface(img);
222.     if (!tex)
223.     {
224.         fprintf(stderr, "load_png: SDL_CreateTextureFromSurface failed for '%s': %s\n", path, SDL_GetError());
225.         return NULL;
226.     }
227.     SDL_SetTextureBlendMode(tex, SDL_BLENDMODE_BLEND);
228.     return tex;
229. }
230.
231. void safe_DestroyTexture(SDL_Texture **tex)
232. {
233.     if (tex && *tex)
234.     {
235.         SDL_DestroyTexture(*tex);
236.         *tex = NULL;
237.     }
238. }
239.
240. void render_logo_at(SDL_Renderer *r, SDL_Texture *logo, int y, float scale, int x_offset, Uint8 alpha)
241. {
242.     if (!logo) return;
243.     int w,h;
244.     SDL_QueryTexture(logo,NULL,NULL,&w,&h);
245.     int ws=(int)(w*scale), hs=(int)(h*scale);
246.     SDL_Rect dst = {SCREEN_W/2 - ws/2 + x_offset, y, ws, hs};
247.     SDL_SetTextureAlphaMod(logo, alpha);
248.     SDL_RenderCopy(r, logo, NULL, &dst);
249. }
250.
251. void render_text_at(SDL_Renderer *r, TTF_Font *font, const char *text, int y, int x_offset, Uint8 alpha)
252. {
253.     if (!font || !text) return;
254.     SDL_Color col = {255,255,255, alpha};
255.     SDL_Surface *surf = TTF_RenderUTF8_Blended(font, text, col);
256.     if (!surf) return;
257.     SDL_Texture *tex = SDL_CreateTextureFromSurface(r, surf);
258.     SDL_FreeSurface(surf);
```

```
259.     if (!tex) return;
260.     SDL_SetTextureBlendMode(tex, SDL_BLENDMODE_BLEND);
261.     SDL_SetTextureAlphaMod(tex, alpha);
262.     int w,h;
263.     SDL_QueryTexture(tex,NULL,NULL,&w,&h);
264.     SDL_Rect dst = {SCREEN_W/2 - w/2 + x_offset, y, w, h};
265.     SDL_RenderCopy(r, tex, NULL, &dst);
266.     SDL_DestroyTexture(tex);
267. }
268.
269. void render_text_center(SDL_Renderer *r, TTF_Font *font, const char *text, int y)
270. {
271.     render_text_at(r, font, text, y, 0, 255);
272. }
273.
274. // -----
275. // Color por nombre (coincide con Paint)
276. // -----
277. SDL_Color colorForName(const char *name)
278. {
279.     if (strcmp(name, "Sega - Mega Drive - Genesis") == 0)
280.         return (SDL_Color){0,48,82,255};
281.     if (strcmp(name, "Sony - PlayStation") == 0)
282.         return (SDL_Color){46,46,46,255};
283.     if (strcmp(name, "Nintendo - Nintendo 64") == 0)
284.         return (SDL_Color){95,159,26,255};
285.     if (strcmp(name, "Nintendo - Game Boy") == 0)
286.         return (SDL_Color){55,24,125,255};
287.     if (strcmp(name, "Nintendo - Super Nintendo Entertainment System") == 0)
288.         return (SDL_Color){95,64,168,255};
289.     if (strcmp(name, "Nintendo - Nintendo Entertainment System") == 0)
290.         return (SDL_Color){255,41,55,255};
291.     if (strcmp(name, "Importar") == 0)
292.         return (SDL_Color){255,125,55,255};
293.     if (strcmp(name, "WiFi") == 0)
294.         return (SDL_Color){212,173,41,255};
295.     if (strcmp(name, "Options") == 0)
296.         return (SDL_Color){0,0,0,255};
297.     return (SDL_Color){0,0,0,255};
298. }
299.
300. // -----
301. // Mapeo consola -> core (arreglo interno)
302. // -----
303. const char* core_for_console(const char *name)
304. {
305.     // Mapear por coincidencias parciales (puedes ajustar nombres exactos)
306.     if (strstr(name, "Genesis") || strstr(name, "Mega Drive"))
307.         return "genesis_plus_gx_libretro.so";
308.     if (strstr(name, "PlayStation") || strstr(name, "PS1"))
309.         return "pcsx_rearmed_libretro.so";
310.     if (strstr(name, "Nintendo 64") || strstr(name, "N64"))
311.         return "parallel_n64_libretro.so";
312.     if (strstr(name, "Game Boy") && strstr(name, "Color")!=NULL)
313.         return "mgba_libretro.so";
314.     if (strstr(name, "Super Nintendo") || strstr(name, "SNES"))
315.         return "snes9x2002_libretro.so";
316.     if (strstr(name, "Nintendo Entertainment System") || strstr(name, "NES"))
317.         return "nestopia_libretro.so";
318.     // default
319.     return "lr-fbzero.so";
320. }
321.
322. // -----
323. // leer carpetas (consolas) - conserva la lógica original
```

```
324. // -----
325. int load_items(Entry items[], SDL_Renderer *r)
326. {
327.     DIR *d = opendir(".");
328.     if (!d) return 0;
329.     struct dirent *dir;
330.     int count = 0;
331.     while ((dir = readdir(d)) != NULL)
332.     {
333.         if (dir->d_type == DT_DIR)
334.         {
335.             if (strcmp(dir->d_name, ".")==0 || strcmp(dir->d_name, "..")==0) continue;
336.             if (dir->d_name[0]=='.') continue;
337.             if (!strstr(dir->d_name, "-")) continue; // Filtra solo carpetas con '-'
338.
339.             Entry *e = &items[count];
340.             strncpy(e->name, dir->d_name, sizeof(e->name)-1); e->name[sizeof(e->name)-1]='\0';
341.             snprintf(e->logoPath, sizeof(e->logoPath), "%s/logo.png", dir->d_name);
342.             e->logo = load_png(r, e->logoPath);
343.             e->bg = colorForName(e->name);
344.             e->isSpecial = false;
345.             // asignar core
346.             strncpy(e->core, core_for_console(e->name), sizeof(e->core)-1);
347.             e->core[sizeof(e->core)-1]='\0';
348.
349.             count++;
350.             if (count>MAX_ITEMS) break;
351.         }
352.     }
353.     closedir(d);
354.     return count;
355. }
356.
357. // -----
358. // util: extraer basename sin extension
359. // -----
360. void get_basename_noext(const char *fname, char *out, size_t outsize)
361. {
362.     // fname puede incluir ruta; aquí asumimos sólo archivo (ya lo manejamos antes)
363.     const char *base = fname;
364.     const char *slash = strrchr(fname, '/');
365.     if (slash) base = slash + 1;
366.     strncpy(out, base, outsize-1);
367.     out[outsize-1] = '\0';
368.     // quitar extension
369.     char *dot = strrchr(out, '.');
370.     if (dot) *dot = '\0';
371. }
372.
373. // -----
374. // Cargar lista de ROMs para una consola: busca en <console>/roms
375. // y boxart en <console>/boxart/<basename>.png
376. // -----
377. int load_roms_for_console(const char *console_dir, RomEntry roms[], SDL_Renderer *r)
378. {
379.     char roms_path[1024];
380.     snprintf(roms_path, sizeof(roms_path), "%s/roms", console_dir);
381.     DIR *d = opendir(roms_path);
382.     if (!d) return 0;
383.     struct dirent *dir;
384.     int count = 0;
385.     while ((dir = readdir(d)) != NULL) {
386.         if (dir->d_type == DT_REG || dir->d_type==DT_LNK || dir->d_type==DT_UNKNOWN) {
387.             // ignorar hidden
388.             if (dir->d_name[0]=='.') continue;
```

```

389.          // construir ruta completa
390.          RomEntry *reentry = &roms[count];
391.          snprintf(reentry->path, sizeof(reentry->path), "%s/%s", roms_path, dir->d_name);
392.          strncpy(reentry->name, dir->d_name, sizeof(reentry->name)-1);
393.          reentry->name[sizeof(reentry->name)-1]='\0';
394.          get_basename_noext(dir->d_name, reentry->baseName, sizeof(reentry->baseName));
395.          // boxart: probar varias ubicaciones y extensiones comunes (Named_Boxarts y boxart)
396.          char candidate[1024];
397.          const char *folders[] = {"Named_Boxarts", "boxart", NULL};
398.          const char *exts[] = {".png", ".PNG", NULL};
399.          SDL_Texture *found = NULL;
400.          for (int fi = 0; folders[fi] != NULL && !found; ++fi) {
401.              for (int ei = 0; exts[ei] != NULL && !found; ++ei) {
402.                  snprintf(candidate, sizeof(candidate), "%s/%s/%s%s", console_dir, folders[fi], reentry-
>baseName, exts[ei]);
403.                  found = load_png(r, candidate);
404.                  if (found) {
405.                      strncpy(reentry->boxartPath, candidate, sizeof(reentry->boxartPath)-1);
406.                      reentry->boxartPath[sizeof(reentry->boxartPath)-1] = '\0';
407.                      break;
408.                  }
409.                  // intentar con el nombre completo (incluida la extension original)
410.                  snprintf(candidate, sizeof(candidate), "%s/%s/%s%s", console_dir, folders[fi], reentry->name,
exts[ei]);
411.                  found = load_png(r, candidate);
412.                  if (found) {
413.                      strncpy(reentry->boxartPath, candidate, sizeof(reentry->boxartPath)-1);
414.                      reentry->boxartPath[sizeof(reentry->boxartPath)-1] = '\0';
415.                      break;
416.                  }
417.              }
418.          }
419.          reentry->boxart = found ? found : load_png(r, BOXART_PLACEHOLDER);
420.          count++;
421.          if (count>=MAX_ROMS) break;
422.      }
423.  }
424.  closedir(d);
425.  return count;
426.  }
427.
428.  // -----
429.  // Lanzar RetroArch con core y rom: fork + exec
430.  // Guarda pid para poder matarlo cuando el usuario pulse ESC
431.  // -----
432.
433.  // =====
434.  // LANZAR RETROARCH CON ENTORNO CORRECTO
435.  // =====
436.  pid_t launch_retroarch(const char *core_path, const char *rom_path) {
437.
438.
439.      // =====
440.      // OSCURECER TERMINAL EN WINDOWS ANTES DE LANZAR RETROARCH
441.      // =====
442.      #ifdef _WIN32
443.          system("color 00 && cls");
444.      #endif
445.      // MOSTRAR PANTALLA NEGRA EN SDL (opcional)
446.      if (global_renderer) {
447.          SDL_SetRenderDrawColor(global_renderer, 0, 0, 0, 255);
448.          SDL_RenderClear(global_renderer);
449.          SDL_RenderPresent(global_renderer);
450.      }
451.      SDL_Delay(150); // un pequeño delay para que se vea

```

```
452.
453.     pid_t pid = fork();
454.
455.     if (pid < 0) {
456.         perror("fork");
457.         return -1;
458.     }
459.
460.     // =====
461.     // CHILD PROCESS → ejecuta RetroArch
462.     // =====
463.     if (pid == 0) {
464.         // -----
465.         // SILENCIAR TERMINAL PARA QUE NO SE VEA NADA
466.         // -----
467.         int devnull = open("/dev/null", O_RDWR);
468.         if (devnull >= 0) {
469.             dup2(devnull, STDOUT_FILENO);
470.             dup2(devnull, STDERR_FILENO);
471.             // opcional → dup2(devnull, STDIN_FILENO);
472.         }
473.
474.         // NO cerrar ni destruir SDL en el hijo para evitar corrupción de contexto KMSDRM
475.
476.         setenv("DISPLAY", ":0", 1);
477.         setenv("XDG_RUNTIME_DIR", "/run/user/1000", 1);
478.
479.         // Resolver core absoluto
480.         static char abs_core[1024];
481.         if (core_path[0] == '/') {
482.             strncpy(abs_core, core_path, sizeof(abs_core)-1);
483.         } else {
484.             const char *pwd = getenv("PWD");
485.             if (!pwd) pwd = ".";
486.             snprintf(abs_core, sizeof(abs_core), "%s/cores/%s", pwd, core_path);
487.         }
488.
489.         struct stat st;
490.         if (stat(abs_core, &st) != 0) _exit(1);
491.
492.         char *args[] = {
493.             "retroarch",
494.             "-L", abs_core,
495.             (char *)rom_path,
496.             NULL
497.         };
498.
499.         execvp("retroarch", args);
500.
501.         _exit(127);
502.     }
503.
504.     // =====
505.     // PARENT PROCESS → front-end sigue
506.     // =====
507.     SDL_Init(SDL_INIT_VIDEO);
508.     IMG_Init(IMG_INIT_PNG);
509.     TTF_Init();
510.
511.     return pid;
512. }
513.
514. // -----
515. int try_init_sdl_video(void) {
516.     const char *drivers[] = {"x11", "kmsdrm", "fbcon", "wayland", "directfb", "dummy", NULL};
```

```
517.     for (int i = 0; drivers[i] != NULL; ++i) {
518.         const char *drv = drivers[i];
519.         setenv("SDL_VIDEODRIVER", drv, 1);
520.         if (SDL_Init(SDL_INIT_VIDEO) == 0) {
521.             fprintf(stderr, "SDL_Init succeeded with video driver '%s'\n", drv);
522.             return 0;
523.         } else {
524.             fprintf(stderr, "SDL_Init failed with driver '%s': %s\n", drv, SDL_GetError());
525.             SDL_Quit();
526.         }
527.     }
528.     return -1;
529. }
530.
531. // Funcion para dibujar rectangulo
532. void draw_rectangle(SDL_Renderer *renderer, int x, int y, int w, int h, SDL_Color color)
533. {
534.     SDL_SetRenderDrawColor(renderer, color.r, color.g, color.b, color.a);
535.     SDL_Rect rect = {x, y, w, h};
536.     SDL_RenderFillRect(renderer, &rect);
537. }
538.
539. void draw_font_in_pointer(SDL_Renderer *renderer, TTF_Font *font, const char *text, int x, int y)
540. {
541.     if (!font || !text) return;
542.     SDL_Color color = {255, 255, 255, 255};
543.     SDL_Surface *surface = TTF_RenderUTF8_Blended(font, text, color);
544.     if (!surface) return;
545.     SDL_Texture *texture = SDL_CreateTextureFromSurface(renderer, surface);
546.     SDL_FreeSurface(surface);
547.     if (!texture) return;
548.     int w, h;
549.     SDL_QueryTexture(texture, NULL, NULL, &w, &h);
550.     SDL_Rect dstRect = {x - w / 2, y - h / 2, w, h};
551.     SDL_RenderCopy(renderer, texture, NULL, &dstRect);
552.     SDL_DestroyTexture(texture);
553. }
554.
555. // Intentar leer nivel de batería del mando desde Raspberry Pi OS usando upower.
556. // Busca un dispositivo cuyo modelo contenga "ESP32Gamepad" y extrae el porcentaje.
557. // Devuelve 0 en éxito y deja el porcentaje en *percent_out, -1 en error.
558. static int read_esp32_battery_percent(int *percent_out)
559. {
560.     if (!percent_out) return -1;
561.
562.     // Comando: buscar en upower -d el bloque de ESP32Gamepad y leer "percentage".
563.     const char *cmd =
564.         "upower -d | awk '/ESP32Gamepad/,/percentage/ { if ($1==\"percentage:\") { gsub(\"%\", \"\", $2); print
$2; exit } }'";
565.
566.     FILE *fp = popen(cmd, "r");
567.     if (!fp) return -1;
568.
569.     char buf[64];
570.     if (!fgets(buf, sizeof(buf), fp))
571.     {
572.         pclose(fp);
573.         return -1;
574.     }
575.     pclose(fp);
576.
577.     int val = atoi(buf);
578.     if (val < 0 || val > 100) return -1;
579.
580.     *percent_out = val;
```

```
581.     return 0;
582. }
583.
584. // Refrescar gamepad periódicamente para manejar desconexiones/reconexiones
585. static void refresh_gamepad(SDL_GameController **gamepad, Uint32 *last_check_time_ms)
586. {
587.     Uint32 now = SDL_GetTicks();
588.     if (now - *last_check_time_ms < 1000) return; // comprobar cada ~1s
589.     *last_check_time_ms = now;
590.
591.     if (*gamepad)
592.     {
593.         if (SDL_GameControllerGetAttached(*gamepad))
594.         {
595.             return; // sigue conectado
596.         }
597.         SDL_GameControllerClose(*gamepad);
598.         *gamepad = NULL;
599.         printf("[DEBUG] Gamepad desconectado, cerrando handler.\n");
600.     }
601.
602.     int num_joy = SDL_NumJoysticks();
603.     if (num_joy <= 0) return;
604.
605.     for (int i = 0; i < num_joy; ++i)
606.     {
607.         if (SDL_IsGameController(i))
608.         {
609.             SDL_GameController *new_gp = SDL_GameControllerOpen(i);
610.             if (new_gp)
611.             {
612.                 *gamepad = new_gp;
613.                 printf("[DEBUG] Gamepad (re)abierto: %s (index %d)\n", SDL_GameControllerName(new_gp),
i);
614.                 break;
615.             }
616.         }
617.     }
618. }
619.
620. //Crear strings de password SSID
621. char* password = NULL;
622. char* ssid = NULL;
623. bool editing_ssid = true;
624.
625. int main(int argc, char **argv)
626. {
627.     (void)argc; (void)argv;
628.     if (try_init_sdl_video() != 0)
629.     {
630.         fprintf(stderr, "Fatal: could not initialize SDL video with any tested driver.\n");
631.         return 1;
632.     }
633.     // Inicializar subsistemas de joystick y gamecontroller
634.     if (SDL_InitSubSystem(SDL_INIT_GAMECONTROLLER) != 0) {
635.         fprintf(stderr, "[DEBUG] SDL_INIT_GAMECONTROLLER error: %s\n", SDL_GetError());
636.     } else {
637.         fprintf(stderr, "[DEBUG] SDL_INIT_GAMECONTROLLER OK\n");
638.     }
639.     if (SDL_InitSubSystem(SDL_INIT_JOYSTICK) != 0) {
640.         fprintf(stderr, "[DEBUG] SDL_INIT_JOYSTICK error: %s\n", SDL_GetError());
641.     } else {
642.         fprintf(stderr, "[DEBUG] SDL_INIT_JOYSTICK OK\n");
643.     }
644.     IMG_Init(IMG_INIT_PNG);
```

```
645.     TTF_Init();
646.     SDL_ShowCursor(SDL_DISABLE);
647.
648.     SDL_Window *win = SDL_CreateWindow("Frontend",
649.         SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
650.         SCREEN_W, SCREEN_H, SDL_WINDOW_SHOWN);
651.     SDL_Renderer *r = SDL_CreateRenderer(win,-1, SDL_RENDERER_ACCELERATED|SDL_RENDERER_PRESENTVSYNC);
652.     global_renderer = r;
653.
654.     // Inicializar audio (SDL_mixer) para música de fondo
655.     if (Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 2048) < 0)
656.     {
657.         fprintf(stderr, "Mix_OpenAudio error: %s\n", Mix_GetError());
658.     }
659.     else
660.     {
661.         master_volume = 50;
662.         Mix_VolumeMusic(MIX_MAX_VOLUME * master_volume / 100);
663.         srand((unsigned int)time(NULL));
664.         load_playlist_file("songs_playlist.txt");
665.     }
666.
667.     TTF_Font *fontSplash = NULL; // ya no usamos splash
668.     TTF_Font *fontSmall = TTF_OpenFont("Poppins-ExtraLight.ttf",28);
669.
670.     // Helper para reiniciar SDL y ventana
671.     #define REINIT_SDL_WINDOW() \
672.         SDL_Init(SDL_INIT_VIDEO); \
673.         IMG_Init(IMG_INIT_PNG); \
674.         TTF_Init(); \
675.         win = SDL_CreateWindow("Frontend", SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED, SCREEN_W, SCREEN_H,
SDL_WINDOW_SHOWN); \
676.         r = SDL_CreateRenderer(win,-1, SDL_RENDERER_ACCELERATED|SDL_RENDERER_PRESENTVSYNC); \
677.         global_renderer = r;
678.
679.     // ===== CARGAR ITEMS =====
680.     Entry items[MAX_ITEMS]; memset(items,0,sizeof(items));
681.     int total=load_items(items,r);
682.
683.     // agregar Importar
684.     if(total<MAX_ITEMS)
685.     {
686.         strncpy(items[total].name,"Importar",sizeof(items[total].name)-1);
687.         items[total].logo=load_png(r,"import.png");
688.         items[total].bg=colorForName("Importar");
689.         items[total].isSpecial=true;
690.         strncpy(items[total].core, core_for_console("Importar"), sizeof(items[total].core)-1);
691.         items[total].core[sizeof(items[total].core)-1] = '\0';
692.         total++;
693.         printf("[DEBUG] Item especial añadido: '%s' (total=%d)\n", items[total-1].name, total);
694.     }
695.
696.     // agregar WiFi
697.     if(total<MAX_ITEMS)
698.     {
699.         strncpy(items[total].name,"WiFi",sizeof(items[total].name)-1);
700.         items[total].logo=load_png(r,"wifi.png");
701.         items[total].bg=colorForName("WiFi");
702.         items[total].isSpecial=true;
703.         strncpy(items[total].core, core_for_console("WiFi"), sizeof(items[total].core)-1);
704.         items[total].core[sizeof(items[total].core)-1] = '\0';
705.         total++;
706.         printf("[DEBUG] Item especial añadido: '%s' (total=%d)\n", items[total-1].name, total);
707.     }
708.
```

```

709. // agregar Options
710. if(total<MAX_ITEMS)
711. {
712.     strncpy(items[total].name,"Options",sizeof(items[total].name)-1);
713.     items[total].logo=load_png(r,"options.png");
714.     items[total].bg=colorForName("Options");
715.     items[total].isSpecial=true;
716.     strncpy(items[total].core, core_for_console("Options"), sizeof(items[total].core)-1);
717.     items[total].core[sizeof(items[total].core)-1] = '\0';
718.     total++;
719.     printf("[DEBUG] Item especial añadido: '%s' (total=%d)\n", items[total-1].name, total);
720. }
721.
722. if(total==0)
723. {
724.     strncpy(items[0].name,"No Items",sizeof(items[0].name)-1);
725.     items[0].logo=NULL; items[0].bg=(SDL_Color){30,30,30,255}; items[0].isSpecial=true;
726.     strncpy(items[0].core, core_for_console("No Items"), sizeof(items[0].core)-1);
727.     items[0].core[sizeof(items[0].core)-1] = '\0';
728.     total=1;
729. }
730.
731. // ===== CONFIG ANIM =====
732. const int anim_time=350; float logo_scale=0.4f;
733. int index=0, oldIndex=0; bool run=true, animating=false; int direction=0; Uint32 anim_start=0;
734.
735. // Estados: 0 = menu consolas, 1 = lista roms, 2 = importar, 3 = wifi, 4 = options
736. enum {STATE_MAIN=0, STATE_ROMLIST=1, STATE_IMPORT=2, STATE_WIFI=3, STATE_OPTIONS=4} state = STATE_MAIN;
737.
738. // rom list storage
739. RomEntry roms[MAX_ROMS]; int rom_count = 0;
740. int rom_index = 0, rom_oldIndex = 0; bool rom_animating = false; int rom_direction = 0; Uint32 rom_anim_start
= 0;
741. char current_console_dir[512] = {0}; // carpeta seleccionada actual
742. pid_t retro_pid = 0; // PID de RetroArch si está corriendo
743. pid_t import_pid = 0; // PID de python3 main.py (servidor de importación)
744.
745. // Matriz de Teclado Virtual para SSID/Password
746. #define n 5
747. #define m 9
748. char virtual_keyboard_uppercase[5][9] =
749. {
750.     "ABCDEFGHI",
751.     "JKLMNOPQR",
752.     "STUVWXYZ0",
753.     "123456789",
754.     "-_@.#$!&*";
755. };
756.
757. char virtual_keyboard_lowercase[5][9] =
758. {
759.     "abcdefghi",
760.     "jklmnopqr",
761.     "stuvwxyz",
762.     "? /\\|,;:\"",
763.     "<~^={}[]"
764. };
765.
766.
767.
768. // Indice de teclado virtual
769. int vk_row = 0;
770. int vk_col = 0;
771. bool vk_uppercase = true;
772. // Botones efectivos para "entrar a menús" detectados dinámicamente

```

```
773.     int enter_btn_controller = -1; // último botón de controller usado para entrar
774.     int enter_btn_joy = -1;        // último botón raw joy usado para entrar
775.
776.     SDL_GameController *gamepad = NULL;
777.     Uint32 gamepad_last_check = 0;
778.     // Mostrar subsistemas activos
779.     printf("[DEBUG] SDL_WasInit(SDL_INIT_GAMECONTROLLER) = %d\n", SDL_WasInit(SDL_INIT_GAMECONTROLLER));
780.     printf("[DEBUG] SDL_WasInit(SDL_INIT_JOYSTICK) = %d\n", SDL_WasInit(SDL_INIT_JOYSTICK));
781.     // Buscar y abrir el primer gamepad disponible
782.     int num_joy = SDL_NumJoysticks();
783.     printf("[DEBUG] SDL_NumJoysticks() = %d\n", num_joy);
784.     if (num_joy > 0) {
785.         for (int i = 0; i < num_joy; ++i) {
786.             printf("[DEBUG] Joystick %d: %s\n", i, SDL_JoystickNameForIndex(i));
787.             if (SDL_IsGameController(i)) {
788.                 printf("[DEBUG] Joystick %d es compatible con GameController\n", i);
789.                 gamepad = SDL_GameControllerOpen(i);
790.                 if (gamepad) {
791.                     printf("[DEBUG] Gamepad abierto correctamente: %s\n", SDL_GameControllerName(gamepad));
792.                     break;
793.                 } else {
794.                     printf("[DEBUG] Error al abrir GameController %d: %s\n", i, SDL_GetError());
795.                 }
796.             } else {
797.                 printf("[DEBUG] Joystick %d NO es compatible con GameController\n", i);
798.             }
799.         }
800.     } else {
801.         printf("[DEBUG] No se detectaron joysticks/gamepads conectados.\n");
802.         printf("[DEBUG] SDL_NumJoysticks() retornó 0. Verifica que el dispositivo esté conectado y que el
usuario tenga permisos.\n");
803.         printf("[DEBUG] Si usas SSH, asegúrate de reenviar dispositivos USB correctamente.\n");
804.     }
805.     while(run)
806.     {
807.         // Reintentar abrir o reabrir el gamepad con frecuencia
808.         refresh_gamepad(&gamepad, &gamepad_last_check);
809.
810.         SDL_Event e;
811.         while(SDL_PollEvent(&e))
812.         {
813.             if(e.type==SDL_QUIT)
814.             {
815.                 run=false;
816.                 break;
817.             }
818.             // Gamepad BUTTON events (d-pad navegación y ENTER/ESC)
819.             if(e.type == SDL_CONTROLLERBUTTONDOWN) {
820.                 int button = e.button.button;
821.                 printf("[DEBUG] SDL_CONTROLLERBUTTONDOWN: button=%d, state=%d\n", button, state);
822.                 // D-pad derecha (invertido)
823.                 // Navegación por teclado virtual en Wifi
824.                 if(state == STATE_WIFI) {
825.                     // Corrige sentido: derecha suma, izquierda resta (al revés de menú)
826.                     if(button == SDL_CONTROLLER_BUTTON_DPAD_RIGHT) {
827.                         vk_col = (vk_col - 1 + m) % m;
828.                     } else if(button == SDL_CONTROLLER_BUTTON_DPAD_LEFT) {
829.                         vk_col = (vk_col + 1) % m;
830.                     } else if(button == SDL_CONTROLLER_BUTTON_DPAD_DOWN) {
831.                         vk_row = (vk_row + 1) % n;
832.                     } else if(button == SDL_CONTROLLER_BUTTON_DPAD_UP) {
833.                         vk_row = (vk_row - 1 + n) % n;
834.                     }
835.                     // Alternar mayúsculas/minúsculas/símbolos con botón 4 (ejemplo: LB)
836.                     if(button == 6) {
```

```

837.         vk_uppercase = !vk_uppercase;
838.     }
839. }
840. // Ajuste de volumen en menú Options con stick/D-Pad arriba/abajo
841. if(state == STATE_OPTIONS) {
842.     if(button == SDL_CONTROLLER_BUTTON_DPAD_UP) {
843.         master_volume += 5;
844.         if(master_volume > 100) master_volume = 100;
845.     } else if(button == SDL_CONTROLLER_BUTTON_DPAD_DOWN) {
846.         master_volume -= 5;
847.         if(master_volume < 0) master_volume = 0;
848.     }
849.     // Aplicar volumen en SDL_mixer y en el sistema (amixer) en segundo plano
850.     Mix_VolumeMusic(MIX_MAX_VOLUME * master_volume / 100);
851.     char cmd[128];
852.     snprintf(cmd, sizeof(cmd), "amixer set Master %d%%s", master_volume, " >/dev/null 2>&1
&");
853.     system(cmd);
854. }
855. // Navegación normal
856. if((button == SDL_CONTROLLER_BUTTON_DPAD_RIGHT) && state==STATE_MAIN && !animating) {
857.     oldIndex=index; index=(index-1+total)%total; direction=-1; animating=true;
anim_start=SDL_GetTicks();
858.     printf("[DEBUG] DPAD RIGHT (invertido), nuevo index=%d\n", index);
859. } else if((button == SDL_CONTROLLER_BUTTON_DPAD_LEFT) && state==STATE_MAIN && !animating) {
860.     oldIndex=index; index=(index+1)%total; direction=1; animating=true; anim_start=SDL_GetTicks();
861.     printf("[DEBUG] DPAD LEFT (invertido), nuevo index=%d\n", index);
862. } else if((button == SDL_CONTROLLER_BUTTON_DPAD_RIGHT) && state==STATE_ROMLIST && !rom_animating)
{
863.     if (rom_count > 0) {
864.         rom_oldIndex = rom_index; rom_index = (rom_index-1+rom_count)%rom_count; rom_direction
= -1; rom_animating = true; rom_anim_start = SDL_GetTicks();
865.         printf("[DEBUG] DPAD RIGHT (invertido) en ROMLIST, nuevo rom_index=%d\n", rom_index);
866.     }
867. } else if((button == SDL_CONTROLLER_BUTTON_DPAD_LEFT) && state==STATE_ROMLIST && !rom_animating)
{
868.     if (rom_count > 0) {
869.         rom_oldIndex = rom_index; rom_index = (rom_index+1)%rom_count; rom_direction = 1;
rom_animating = true; rom_anim_start = SDL_GetTicks();
870.         printf("[DEBUG] DPAD LEFT (invertido) en ROMLIST, nuevo rom_index=%d\n", rom_index);
871.     }
872. }
873. // Seleccionar letra en teclado virtual: usa el último botón de "entrar a menús"
874. // detectado fuera de WiFi. Si aún no se detectó, por defecto A (0).
875. if(state == STATE_WIFI && button == ((enter_btn_controller >= 0) ? enter_btn_controller : 0))
876. {
877.     const char *vk_row_ptr = vk_uppercase ? virtual_keyboard_uppercase[vk_row] :
virtual_keyboard_lowercase[vk_row];
878.     char ch = vk_row_ptr[vk_col];
879.     if(ch != '\\0' && ch != ' ') {
880.         size_t len = 0;
881.         if(editing_ssid) {
882.             len = ssid ? strlen(ssid) : 0;
883.             if(!ssid) ssid = calloc(64,1);
884.             if(len < 63) { ssid[len] = ch; ssid[len+1] = '\\0'; }
885.         } else {
886.             len = password ? strlen(password) : 0;
887.             if(!password) password = calloc(64,1);
888.             if(len < 63) { password[len] = ch; password[len+1] = '\\0'; }
889.         }
890.     }
891. }
892. else if(state == STATE_WIFI && button == 10) // Borrar último carácter con botón BACK
(SDL_CONTROLLER_BUTTON_BACK = 6)
893. {

```

```
894.         if(editing_ssid && ssid && strlen(ssid) > 0) {
895.             ssid[strlen(ssid)-1] = '\0';
896.         } else if(!editing_ssid && password && strlen(password) > 0) {
897.             password[strlen(password)-1] = '\0';
898.         }
899.     }
900.     else if(state != STATE_WIFI && (button == 0 || button == SDL_CONTROLLER_BUTTON_START || button
== 8 || button == 9)) // Abrir el Servidor
901.     {
902.         // Registrar botón de "entrar a menús" efectivo para usar en WiFi
903.         enter_btn_controller = button;
904.         if(state==STATE_MAIN && !animating)
905.         {
906.             //Abrir solamente si no es especial
907.             if(items[index].isSpecial) {
908.                 printf("DEBUG: ENTER en STATE_MAIN seleccion especial: index=%d name='%s'\n",
index, items[index].name); fflush(stdout);
909.                 if(strcmp(items[index].name, "Importar") == 0) {
910.                     // Lanzar servidor de importación (python3 main.py) si no está ya lanzado
911.                     if(import_pid == 0) {
912.                         pid_t child = fork();
913.                         if(child == 0) {
914.                             execlp("python3", "python3", "main.py", (char *)NULL);
915.                             _exit(127);
916.                         } else if(child > 0) {
917.                             import_pid = child;
918.                             printf("[DEBUG] Lanzado servidor de importación con PID %d\n",
(int)import_pid);
919.                         } else {
920.                             fprintf(stderr, "Error al hacer fork() para python3 main.py\n");
921.                         }
922.                     }
923.                     state = STATE_IMPORT;
924.                 } else if(strcmp(items[index].name, "WiFi") == 0) {
925.                     // Limpiar SSID y password al entrar a WiFi
926.                     if(ssid) { free(ssid); ssid = NULL; }
927.                     if(password) { free(password); password = NULL; }
928.                     editing_ssid = true;
929.                     state = STATE_WIFI;
930.                 } else if(strcmp(items[index].name, "Options") == 0) {
931.                     state = STATE_OPTIONS;
932.                 }
933.             } else {
934.                 printf("DEBUG: ENTER en STATE_MAIN seleccion: index=%d name='%s'\n", index,
items[index].name); fflush(stdout);
935.                 strncpy(current_console_dir, items[index].name, sizeof(current_console_dir)-1);
936.                 current_console_dir[sizeof(current_console_dir)-1] = '\0';
937.                 rom_count = load_roms_for_console(current_console_dir, roms, r);
938.                 printf("DEBUG: load_roms_for_console -> rom_count=%d for console='%s'\n",
rom_count, current_console_dir); fflush(stdout);
939.                 rom_index = 0; rom_oldIndex= 0; rom_animating = false; rom_direction = 0;
940.                 state = STATE_ROMLIST;
941.             }
942.         }
943.         else if(state==STATE_ROMLIST && !rom_animating)
944.         {
945.             if (rom_count > 0) {
946.                 const char *maybe_path = roms[rom_index].path[0] ? roms[rom_index].path :
"(empty)";
947.                 printf("DEBUG: ENTER en STATE_ROMLIST -> rom_count=%d rom_index=%d rom_path='%s'\n",
rom_count, rom_index, maybe_path); fflush(stdout);
948.             } else {
949.                 printf("DEBUG: ENTER en STATE_ROMLIST -> rom_count=0\n"); fflush(stdout);
950.             }
951.             if(rom_count>0 && roms[rom_index].path[0]!=0) {
```

```
952.         const char *core = items[index].core;
953.         const char *rompath = roms[rom_index].path;
954.         printf("Lanzando RetroArch: core=%s rom=%s\n", core?core:"(null)",
rompath?rompath:"(null)"); fflush(stdout);
955.         SDL_DestroyRenderer(r);
956.         SDL_DestroyWindow(win);
957.         TTF_CloseFont(fontSmall);
958.         TTF_CloseFont(fontSplash);
959.         TTF_Quit();
960.         IMG_Quit();
961.         SDL_Quit();
962.         global_renderer = NULL;
963.         r = NULL;
964.         win = NULL;
965.         fontSmall = NULL;
966.         fontSplash = NULL;
967.         pid_t child = fork();
968.         if(child == 0) {
969.             static char abs_core[1024];
970.             if (core[0] == '/') {
971.                 strncpy(abs_core, core, sizeof(abs_core)-1);
972.                 abs_core[sizeof(abs_core)-1] = '\0';
973.             } else {
974.                 const char *pwd = getenv("PWD");
975.                 if (!pwd) pwd = ".";
976.                 snprintf(abs_core, sizeof(abs_core), "%s/cores/%s", pwd, core);
977.             }
978.             struct stat st;
979.             if (stat(abs_core, &st) != 0) _exit(1);
980.             char *args[] = {
981.                 "retroarch",
982.                 "-L", abs_core,
983.                 (char *)rompath,
984.                 NULL
985.             };
986.             execvp("retroarch", args);
987.             _exit(127);
988.         } else if(child > 0) {
989.             int status = 0;
990.             int retro_running = 1;
991.             while(retro_running) {
992.                 SDL_Event ev;
993.                 SDL_Delay(16);
994.                 pid_t w = waitpid(child, &status, WNOHANG);
995.                 if(w == child) {
996.                     retro_running = 0;
997.                     break;
998.                 }
999.                 while(SDL_PollEvent(&ev)) {
1000.                     if(ev.type == SDL_CONTROLLERBUTTONDOWN && ev.cbutton.button == 1)
{
1001.                         kill(child, SIGTERM);
1002.                         waitpid(child, &status, 0);
1003.                         retro_running = 0;
1004.                         break;
1005.                     }
1006.                 }
1007.             }
1008.             // Reinicio total del frontend tras salir de RetroArch
1009.             char *argv0 = argv[0];
1010.             char *new_argv[argc+1];
1011.             for(int i=0; i<argc; i++) new_argv[i]=argv[i];
1012.             new_argv[argc]=NULL;
1013.             execvp(argv0, new_argv);
1014.             // Si execvp falla, salir
```

```
1015.         exit(0);
1016.     } else {
1017.         fprintf(stderr, "Error en fork() para RetroArch\n");
1018.     }
1019.     } else {
1020.         printf("No hay ROM seleccionada\n"); fflush(stdout);
1021.     }
1022. }
1023. }
1024. else if(button == 1) // Cerrar el Server
1025. {
1026.     if(state==STATE_MAIN && !animating) {
1027.         run=false;
1028.         break;
1029.     } else if(state==STATE_ROMLIST && !rom_animating) {
1030.         for(int i=0;i<rom_count;i++) safe_DestroyTexture(&roms[i].boxart);
1031.         rom_count = 0;
1032.         current_console_dir[0] = '\0';
1033.         state = STATE_MAIN;
1034.         retro_pid = 0;
1035.     } else if(state==STATE_IMPORT) {
1036.         // Al salir de la pantalla de importar, detener python3 main.py si sigue corriendo
1037.         if(import_pid != 0) {
1038.             kill(import_pid, SIGTERM);
1039.             waitpid(import_pid, NULL, 0);
1040.             printf("[DEBUG] Servidor de importación detenido (PID %d)\n", (int)import_pid);
1041.             import_pid = 0;
1042.         }
1043.         state = STATE_MAIN;
1044.     } else if(state==STATE_WIFI) {
1045.         // Si SSID y password no están vacíos, conectar
1046.         if(ssid && password && strlen(ssid) > 0 && strlen(password) > 0) {
1047.             // Mostrar mensaje 'Conectando...'
1048.             show_connecting = 1;
1049.             connecting_start = SDL_GetTicks();
1050.             Uint32 start_time = SDL_GetTicks();
1051.             while(SDL_GetTicks() - start_time < 3000) { // 3 segundos
1052.                 SDL_Color bg = {212,173,41,255};
1053.                 SDL_SetRenderDrawColor(r, bg.r, bg.g, bg.b, 255);
1054.                 SDL_RenderClear(r);
1055.                 render_text_center(r, fontSmall, "Conectando...", SCREEN_H / 2);
1056.                 SDL_RenderPresent(r);
1057.                 SDL_Delay(50);
1058.             }
1059.             // Ejecutar nmcli para conectar
1060.             char cmd[512];
1061.             snprintf(cmd, sizeof(cmd), "nmcli device wifi connect '%s' password '%s'", ssid,
password);
1062.             system(cmd);
1063.             show_connecting = 0;
1064.         }
1065.         state = STATE_MAIN;
1066.     } else if(state==STATE_OPTIONS) {
1067.         state = STATE_MAIN;
1068.     }
1069. }
1070. }
1071. // Eventos de joystick "crudos" (para botones que SDL no mapea a GameController)
1072. if(e.type == SDL_JOYBUTTONDOWN)
1073. {
1074.     int jbutton = e.jbutton.button;
1075.     printf("[DEBUG] SDL_JOYBUTTONDOWN: button=%d, state=%d\n", jbutton, state);
1076.
1077.     // WiFi: seleccionar letra con botón de "entrar" usando el último botón joy detectado
1078.     if(state == STATE_WIFI && jbutton == ((enter_btn_joy >= 0) ? enter_btn_joy : 0))
```

```

1079.         {
1080.             const char *vk_row_ptr = vk_uppercase ? virtual_keyboard_uppercase[vk_row] :
virtual_keyboard_lowercase[vk_row];
1081.             char ch = vk_row_ptr[vk_col];
1082.             if(ch != '\0' && ch != ' ') {
1083.                 size_t len = 0;
1084.                 if(editing_ssid) {
1085.                     len = ssid ? strlen(ssid) : 0;
1086.                     if(!ssid) ssid = calloc(64,1);
1087.                     if(len < 63) { ssid[len] = ch; ssid[len+1] = '\0'; }
1088.                 } else {
1089.                     len = password ? strlen(password) : 0;
1090.                     if(!password) password = calloc(64,1);
1091.                     if(len < 63) { password[len] = ch; password[len+1] = '\0'; }
1092.                 }
1093.             }
1094.         }
1095.         else if(state == STATE_WIFI && jbutton == 10) // WiFi: borrar último carácter (algunos
firmwares usan 10 para BACK)
1096.         {
1097.             if(editing_ssid && ssid && strlen(ssid) > 0) {
1098.                 ssid[strlen(ssid)-1] = '\0';
1099.             } else if(!editing_ssid && password && strlen(password) > 0) {
1100.                 password[strlen(password)-1] = '\0';
1101.             }
1102.         }
1103.         else if(state == STATE_WIFI && (jbutton == 4)) // WiFi: alternar SSID/PASS con botones 2 o
3
1104.         {
1105.             editing_ssid = !editing_ssid;
1106.         }
1107.         else if(state == STATE_WIFI && jbutton == 6) // WiFi: alternar mayúsculas/minúsculas con
hombro izquierdo (joy 4 común)
1108.         {
1109.             vk_uppercase = !vk_uppercase;
1110.         }
1111.         if(jbutton == 8 || jbutton == 9) // Tratar botones 8/9 como START/ENTER equivalente
1112.         {
1113.             // Registrar botón joy efectivo para "entrar a menús"
1114.             enter_btn_joy = jbutton;
1115.             if(state==STATE_MAIN && !animating) {
1116.                 //Abrir solamente si no es especial
1117.                 if(items[index].isSpecial) {
1118.                     printf("[DEBUG: ENTER (JOY) en STATE_MAIN seleccion especial: index=%d
name='%s'\n", index, items[index].name); fflush(stdout);
1119.                     if(strcmp(items[index].name, "Importar") == 0) {
1120.                         // Lanzar servidor de importación (python3 main.py) si no está ya lanzado
1121.                         if(import_pid == 0) {
1122.                             pid_t child = fork();
1123.                             if(child == 0) {
1124.                                 execlp("python3", "python3", "main.py", (char *)NULL);
1125.                                 _exit(127);
1126.                             } else if(child > 0) {
1127.                                 import_pid = child;
1128.                                 printf("[DEBUG] Lanzado servidor de importación con PID %d\n",
(int)import_pid);
1129.                             } else {
1130.                                 fprintf(stderr, "Error al hacer fork() para python3 main.py\n");
1131.                             }
1132.                         }
1133.                         state = STATE_IMPORT;
1134.                     } else if(strcmp(items[index].name, "WiFi") == 0) {
1135.                         printf("Funcionalidad WiFi no implementada aún. (JOY)\n"); fflush(stdout);
1136.                         // Limpiar SSID y password al entrar a WiFi
1137.                         if(ssid) { free(ssid); ssid = NULL; }

```

```
1138.         if(password) { free(password); password = NULL; }
1139.         editing_ssid = true;
1140.         state = STATE_WIFI;
1141.     } else if(strcmp(items[index].name, "Options") == 0) {
1142.         state = STATE_OPTIONS;
1143.     }
1144. } else {
1145.     printf("DEBUG: ENTER (JOY) en STATE_MAIN seleccion: index=%d name='%s'\n",
index, items[index].name); fflush(stdout);
1146.     strncpy(current_console_dir, items[index].name, sizeof(current_console_dir)-1);
1147.     current_console_dir[sizeof(current_console_dir)-1] = '\0';
1148.     rom_count = load_roms_for_console(current_console_dir, roms, r);
1149.     printf("DEBUG: load_roms_for_console (JOY) -> rom_count=%d for console='%s'\n",
rom_count, current_console_dir); fflush(stdout);
1150.     rom_index = 0; rom_oldIndex= 0; rom_animating = false; rom_direction = 0;
1151.     state = STATE_ROMLIST;
1152. }
1153. } else if(state==STATE_ROMLIST && !rom_animating) {
1154.     if (rom_count > 0) {
1155.         const char *maybe_path = roms[rom_index].path[0] ? roms[rom_index].path :
"(empty)";
1156.         printf("DEBUG: ENTER (JOY) en STATE_ROMLIST -> rom_count=%d rom_index=%d
rom_path='%s'\n", rom_count, rom_index, maybe_path); fflush(stdout);
1157.     } else {
1158.         printf("DEBUG: ENTER (JOY) en STATE_ROMLIST -> rom_count=0\n"); fflush(stdout);
1159.     }
1160.     if(rom_count>0 && roms[rom_index].path[0]!=0) {
1161.         const char *core = items[index].core;
1162.         const char *rompath = roms[rom_index].path;
1163.         printf("Lanzando RetroArch (JOY): core=%s rom=%s\n", core?core:"(null)",
rompath?rompath:"(null)"); fflush(stdout);
1164.         SDL_DestroyRenderer(r);
1165.         SDL_DestroyWindow(win);
1166.         TTF_CloseFont(fontSmall);
1167.         TTF_CloseFont(fontSplash);
1168.         TTF_Quit();
1169.         IMG_Quit();
1170.         SDL_Quit();
1171.         global_renderer = NULL;
1172.         r = NULL;
1173.         win = NULL;
1174.         fontSmall = NULL;
1175.         fontSplash = NULL;
1176.         pid_t child = fork();
1177.         if(child == 0) {
1178.             static char abs_core[1024];
1179.             if (core[0] == '/') {
1180.                 strncpy(abs_core, core, sizeof(abs_core)-1);
1181.                 abs_core[sizeof(abs_core)-1] = '\0';
1182.             } else {
1183.                 const char *pwd = getenv("PWD");
1184.                 if (!pwd) pwd = ".";
1185.                 snprintf(abs_core, sizeof(abs_core), "%s/cores/%s", pwd, core);
1186.             }
1187.             struct stat st;
1188.             if (stat(abs_core, &st) != 0) _exit(1);
1189.             char *args[] = {
1190.                 "retroarch",
1191.                 "-L", abs_core,
1192.                 (char *)rompath,
1193.                 NULL
1194.             };
1195.             execvp("retroarch", args);
1196.             _exit(127);
1197.         } else if(child > 0) {
```

```
1198.         int status = 0;
1199.         int retro_running = 1;
1200.         while(retro_running) {
1201.             SDL_Event ev;
1202.             SDL_Delay(16);
1203.             pid_t w = waitpid(child, &status, WNOHANG);
1204.             if(w == child) {
1205.                 retro_running = 0;
1206.                 break;
1207.             }
1208.             while(SDL_PollEvent(&ev)) {
1209.                 if(ev.type == SDL_CONTROLLERBUTTONDOWN && ev.cbutton.button == 1)
{
1210.                     kill(child, SIGTERM);
1211.                     waitpid(child, &status, 0);
1212.                     retro_running = 0;
1213.                     break;
1214.                 }
1215.             }
1216.         }
1217.         // Reinicio total del frontend tras salir de RetroArch
1218.         char *argv0 = argv[0];
1219.         char *new_argv[argc+1];
1220.         for(int i=0;i<argc;i++) new_argv[i]=argv[i];
1221.         new_argv[argc]=NULL;
1222.         execvp(argv0, new_argv);
1223.         // Si execvp falla, salir
1224.         exit(0);
1225.     } else {
1226.         fprintf(stderr, "Error en fork() para RetroArch (JOY)\n");
1227.     }
1228. } else {
1229.     printf("No hay ROM seleccionada (JOY)\n"); fflush(stdout);
1230. }
1231. }
1232. }
1233. }
1234. // Teclado (mantener como fallback)
1235. if(e.type==SDL_KEYDOWN)
1236. {
1237.     // Cerrar con ESC
1238.     if(e.key.keysym.sym == SDLK_ESCAPE)
1239.     {
1240.         run = false;
1241.         break;
1242.     }
1243. }
1244. }
1245. // Actualizar música de fondo (playlist aleatoria)
1246. update_music();
1247. // Si RetroArch se terminó de forma externa, detectar y limpiar PID
1248. if(retro_pid != 0)
1249. {
1250.     int status;
1251.     pid_t w = waitpid(retro_pid, &status, WNOHANG);
1252.     if(w == retro_pid) // el proceso terminó, si es 0 sigue corriendo, y si es -1 error
1253.     {
1254.         // terminó
1255.         retro_pid = 0;
1256.     }
1257. }
1258.
1259. // ----- RENDER -----
1260. if(state==STATE_MAIN)
1261. {
```

```
1262.         float t=1.0f;
1263.         if(animating)
1264.         {
1265.             Uint32 now=SDL_GetTicks();
1266.             t=(now-anim_start)/(float)anim_time;
1267.             if(t>=1.0f){t=1.0f; animating=false;}
1268.         }
1269.
1270.         int offsetNew=(int)(direction*SCREEN_W*(1.0-t));
1271.         int offsetOld=offsetNew-direction*SCREEN_W;
1272.         if(!animating){ offsetNew=0; offsetOld=-direction*SCREEN_W;}
1273.         Uint8 alphaNew=(Uint8)(255*t); Uint8 alphaOld=(Uint8)(255*(1.0-t));
1274.         if(!animating){ alphaNew=255; alphaOld=0;}
1275.
1276.         // RENDER fondo del nuevo
1277.         SDL_Color bg=items[index].bg;
1278.         SDL_SetRenderDrawColor(r,bg.r,bg.g,bg.b,255); SDL_RenderClear(r);
1279.
1280.         // dibujar nuevo
1281.         render_logo_at(r,items[index].logo,SCREEN_H/2-180,logo_scale,offsetNew,alphaNew);
1282.         render_text_at(r,fontSmall,items[index].name,(int)(SCREEN_H*0.1),offsetNew,alphaNew);
1283.
1284.         // dibujar viejo
1285.         if(animating || alphaOld>0)
1286.         {
1287.             SDL_SetRenderDrawBlendMode(r,SDL_BLENDMODE_BLEND);
1288.             SDL_Color bgo=items[oldIndex].bg;
1289.             Uint8 bgAlphaBlend=(Uint8)(128*(alphaOld/255.0f));
1290.             SDL_SetRenderDrawColor(r,bgo.r,bgo.g,bgo.b,bgAlphaBlend);
1291.             SDL_Rect full={0,0,SCREEN_W,SCREEN_H};
1292.             SDL_RenderFillRect(r,&full);
1293.             render_logo_at(r,items[oldIndex].logo,SCREEN_H/2-180,logo_scale,offsetOld,alphaOld);
1294.             render_text_at(r,fontSmall,items[oldIndex].name,(int)(SCREEN_H*0.1),offsetOld,alphaOld);
1295.             SDL_SetRenderDrawBlendMode(r,SDL_BLENDMODE_NONE);
1296.         }
1297.
1298.         // instrucciones ocultas por pedido del usuario (no mostrar mensajes en la parte inferior)
1299.
1300.         SDL_RenderPresent(r);
1301.         if(!animating) SDL_Delay(8); else SDL_Delay(1);
1302.     }
1303.     else if(state==STATE_ROMLIST)
1304.     {
1305.         // Carrousel de boxarts
1306.         float t=1.0f;
1307.         if(rom_animating)
1308.         {
1309.             Uint32 now=SDL_GetTicks();
1310.             t=(now-rom_anim_start)/(float)anim_time;
1311.             if(t>=1.0f){t=1.0f; rom_animating=false;}
1312.         }
1313.
1314.         int offsetNew=(int)(rom_direction*SCREEN_W*(1.0-t));
1315.         int offsetOld=offsetNew-rom_direction*SCREEN_W;
1316.         if(!rom_animating){ offsetNew=0; offsetOld=-rom_direction*SCREEN_W;}
1317.         Uint8 alphaNew=(Uint8)(255*t); Uint8 alphaOld=(Uint8)(255*(1.0-t));
1318.         if(!rom_animating){ alphaNew=255; alphaOld=0;}
1319.
1320.         // fondo basado en color de la consola
1321.         SDL_Color bg = {30,30,30,255};
1322.         // intentar recuperar color de la consola seleccionada
1323.         bg = items[index].bg;
1324.         SDL_SetRenderDrawColor(r,bg.r,bg.g,bg.b,255); SDL_RenderClear(r);
1325.
1326.         // logo/boxart escala distinto (usa la macro BOXART_SCALE)
```

```
1327.         float box_scale = BOXART_SCALE;
1328.
1329.         // dibujar nuevo (si hay ROMs)
1330.         if (rom_count == 0)
1331.         {
1332.             render_text_center(r, fontSmall, "No hay ROMs", SCREEN_H/2);
1333.         }
1334.         else
1335.         {
1336.             render_logo_at(r, roms[rom_index].boxart, SCREEN_H/2 + BOXART_Y_OFFSET, box_scale,
offsetNew, alphaNew);
1337.             render_text_at(r, fontSmall, roms[rom_index].baseName, (int)(SCREEN_H *
GAME_NAME_Y_RATIO), offsetNew, alphaNew);
1338.         }
1339.
1340.         // dibujar viejo (si animando)
1341.         if(rom_animating || alphaOld>0)
1342.         {
1343.             SDL_SetRenderDrawBlendMode(r,SDL_BLENDMODE_BLEND);
1344.             Uint8 bgAlphaBlend = (Uint8)(128*(alphaOld/255.0f));
1345.             SDL_SetRenderDrawColor(r, bg.r, bg.g, bg.b, bgAlphaBlend);
1346.             SDL_Rect full = {0,0,SCREEN_W,SCREEN_H};
1347.             SDL_RenderFillRect(r,&full);
1348.             render_logo_at(r, roms[rom_oldIndex].boxart, SCREEN_H/2 + BOXART_Y_OFFSET, box_scale,
offsetOld, alphaOld);
1349.             render_text_at(r, fontSmall, roms[rom_oldIndex].baseName, (int)(SCREEN_H *
GAME_NAME_Y_RATIO), offsetOld, alphaOld);
1350.             SDL_SetRenderDrawBlendMode(r,SDL_BLENDMODE_NONE);
1351.         }
1352.
1353.         // Mostrar nombre de la consola arriba
1354.         char top_label[512];
1355.         snprintf(top_label, sizeof(top_label), "%s - ROMs (%d)", current_console_dir, rom_count);
1356.         render_text_at(r, fontSmall, top_label, 8, 0, 255);
1357.
1358.         // no mostramos mensajes de instrucción en la parte inferior
1359.
1360.         SDL_RenderPresent(r);
1361.         if(!rom_animating) SDL_Delay(8); else SDL_Delay(1);
1362.     }
1363.     else if (state == STATE_IMPORT)
1364.     {
1365.         // pantalla de importación (simple mensaje)
1366.         SDL_Color bg = {255,125,55,255};
1367.         SDL_SetRenderDrawColor(r, bg.r, bg.g, bg.b, 255);
1368.         SDL_RenderClear(r);
1369.
1370.         render_text_center(r, fontSmall, "Importando juegos...", SCREEN_H / 2);
1371.
1372.         // Obtener IP de la Raspberry Pi
1373.         char ip[128] = {0};
1374.         FILE *fp = popen("hostname -I | awk '{print $1}'\"", "r");
1375.         if(fp)
1376.         {
1377.             if(fgets(ip, sizeof(ip), fp) != NULL)
1378.             {
1379.                 // eliminar salto de línea al final
1380.                 size_t len = strlen(ip);
1381.                 if(len > 0 && ip[len-1] == '\n') ip[len-1] = '\0';
1382.             }
1383.             pclose(fp);
1384.         }
1385.
1386.         // mostrar la IP si hay, si no mostrar mensaje de conexión
1387.         if(strlen(ip) > 0)
```

```

1388.         {
1389.             char msg[256];
1390.             snprintf(msg, sizeof(msg), "Abre el navegador en http://%s:5000", ip);
1391.             render_text_center(r, fontSmall, msg, SCREEN_H / 2 + 40);
1392.         }
1393.     else
1394.     {
1395.         render_text_center(r, fontSmall, "Primero debes conectarte a internet", SCREEN_H / 2 + 40);
1396.     }
1397.
1398.     render_text_center(r, fontSmall, "Presiona BACKSPACE para volver", SCREEN_H - 40);
1399.     SDL_RenderPresent(r);
1400.     SDL_Delay(100);
1401. }
1402.
1403. else if (state == STATE_WIFI)
1404. {
1405.     SDL_Color bg = {212,173,41,255};
1406.     SDL_SetRenderDrawColor(r, bg.r, bg.g, bg.b, 255);
1407.     SDL_RenderClear(r);
1408.
1409.     render_text_center(r, fontSmall, "Configuración WiFi", 20);
1410.     render_text_center(r, fontSmall, "Usa las flechas para seleccionar caracteres", 60);
1411.
1412.
1413.     // Dibujar caracteres del teclado
1414.     for(int row = 0; row < n; row++) {
1415.         const char *vk_row_ptr = vk_uppercase ? virtual_keyboard_uppercase[row] :
virtual_keyboard_lowercase[row];
1416.         for(int col = 0; col < m; col++) {
1417.             char ch = vk_row_ptr[col];
1418.             int x = KEYBOARD_X + col * KEY_W;
1419.             int y = KEYBOARD_Y + row * KEY_H;
1420.             // Si es la letra seleccionada, dibujar marco rojo
1421.             if(row == vk_row && col == vk_col) {
1422.                 SDL_Rect marco = {x-3, y-3, BOX_W, BOX_H};
1423.                 SDL_SetRenderDrawColor(r, 255, 0, 0, 255);
1424.                 SDL_RenderDrawRect(r, &marco);
1425.                 SDL_RenderDrawRect(r, &(SDL_Rect){x-2, y-2, BOX_W-2, BOX_H-2});
1426.             }
1427.             draw_font_in_pointer(r, fontSmall, (char[]){ch, '\0'}, x + KEY_W / 2, y + KEY_H /
2);
1428.         }
1429.     }
1430.
1431.
1432.     // Mostrar mensaje de estado solo brevemente tras presionar START
1433.     // show_connecting y connecting_start ahora son globales
1434.     if(show_connecting) {
1435.         render_text_center(r, fontSmall, "Conectando...", SCREEN_H - 40);
1436.         if(SDL_GetTicks() - connecting_start > 1500) { // 1.5 segundos
1437.             show_connecting = 0;
1438.         }
1439.     }
1440.
1441.     // Mostrar SSID y Password actuales con flecha en el campo activo
1442.     char ssid_display[520];
1443.     char password_display[520];
1444.     if(editing_ssid) {
1445.         snprintf(ssid_display, sizeof(ssid_display), ">SSID: %s", ssid ? ssid : "(vacío)");
1446.         snprintf(password_display, sizeof(password_display), " Password: %s", password ? password
: "(vacío)");
1447.     } else {
1448.         snprintf(ssid_display, sizeof(ssid_display), " SSID: %s", ssid ? ssid : "(vacío)");

```

```

1449.                 snprintf(password_display, sizeof(password_display), ">Password: %s", password ? password
: "(vacío)");
1450.             }
1451.             render_text_center(r, fontSmall, ssid_display, SCREEN_H - 120);
1452.             render_text_center(r, fontSmall, password_display, SCREEN_H - 80);
1453.
1454.
1455.             SDL_RenderPresent(r);
1456.             SDL_Delay(100);
1457.         }
1458.         else if (state == STATE_OPTIONS)
1459.         {
1460.             SDL_Color bg = colorForName("Options");
1461.             SDL_SetRenderDrawColor(r, bg.r, bg.g, bg.b, 255);
1462.             SDL_RenderClear(r);
1463.
1464.             render_text_center(r, fontSmall, "Opciones", 40);
1465.
1466.             // Mostrar volumen actual
1467.             char vol_text[64];
1468.             snprintf(vol_text, sizeof(vol_text), "Volumen: %d%% (stick arriba/abajo)", master_volume);
1469.             render_text_center(r, fontSmall, vol_text, SCREEN_H / 2 - 20);
1470.
1471.             // Mostrar nivel de batería del gamepad.
1472.             // 1º: intentar leer porcentaje real desde upower (ESP32Gamepad).
1473.             // 2º: si falla, usar el nivel aproximado que ofrece SDL.
1474.             char bat_text[128];
1475.             if (gamepad)
1476.             {
1477.                 int real_percent = -1;
1478.                 if (read_esp32_battery_percent(&real_percent) == 0)
1479.                 {
1480.                     snprintf(bat_text, sizeof(bat_text), "Batería: %d%%", real_percent);
1481.                 }
1482.                 else
1483.                 {
1484.                     SDL_Joystick *joy = SDL_GameControllerGetJoystick(gamepad);
1485.                     SDL_JoystickPowerLevel lvl = joy ? SDL_JoystickCurrentPowerLevel(joy) :
SDL_JOYSTICK_POWER_UNKNOWN;
1486.                     const char *lvl_str = "Desconocida";
1487.                     int approx = -1;
1488.                     switch (lvl)
1489.                     {
1490.                         case SDL_JOYSTICK_POWER_EMPTY:    lvl_str = "Vacía"; approx = 0;
break;
1491.                         case SDL_JOYSTICK_POWER_LOW:      lvl_str = "Baja"; approx = 25;
break;
1492.                         case SDL_JOYSTICK_POWER_MEDIUM:   lvl_str = "Media"; approx = 50;
break;
1493.                         case SDL_JOYSTICK_POWER_FULL:     lvl_str = "Alta"; approx = 100;
break;
1494.                         case SDL_JOYSTICK_POWER_WIRED:    lvl_str = "Cable"; approx = 100;
break;
1495.                         case SDL_JOYSTICK_POWER_MAX:      lvl_str = "Máx"; approx = 100;
break;
1496.                         case SDL_JOYSTICK_POWER_UNKNOWN:  lvl_str = "Desconocida"; approx =
-1; break;
1497.                     }
1498.                     if (approx >= 0)
1499.                         snprintf(bat_text, sizeof(bat_text), "Batería: ~%d%% (%s)", approx,
lvl_str);
1500.                     else
1501.                         snprintf(bat_text, sizeof(bat_text), "Batería: %s", lvl_str);
1502.                 }
1503.             }

```

```
1504.         }
1505.         else
1506.         {
1507.             snprintf(bat_text, sizeof(bat_text), "Batería: sin gamepad");
1508.         }
1509.
1510.         render_text_center(r, fontSmall, bat_text, SCREEN_H / 2 + 20);
1511.
1512.         // Línea de versión / créditos
1513.         render_text_center(r, fontSmall, "Firmware v1.0 Feb 2026 - BSdevs0", SCREEN_H - 40);
1514.
1515.         SDL_RenderPresent(r);
1516.         SDL_Delay(80);
1517.     }
1518.
1519.     }
1520.
1521.     // cleanup
1522.     for(int i=0;i<total;i++) safe_DestroyTexture(&items[i].logo);
1523.     for(int i=0;i<rom_count;i++) safe_DestroyTexture(&roms[i].boxart);
1524.     if(fontSmall) TTF_CloseFont(fontSmall);
1525.     if(fontSplash) TTF_CloseFont(fontSplash);
1526.     free_playlist();
1527.     Mix_CloseAudio();
1528.     SDL_DestroyRenderer(r);
1529.     SDL_DestroyWindow(win);
1530.     TTF_Quit();
1531.     IMG_Quit();
1532.     SDL_Quit();
1533.
1534.     // Asegurar terminar RetroArch si sigue corriendo
1535.     if(retro_pid != 0)
1536.     {
1537.         kill(retro_pid, SIGTERM);
1538.         waitpid(retro_pid, NULL, 0);
1539.     }
1540.
1541.     // Asegurar terminar servidor de importación (python3 main.py) si sigue corriendo
1542.     if(import_pid != 0)
1543.     {
1544.         kill(import_pid, SIGTERM);
1545.         waitpid(import_pid, NULL, 0);
1546.     }
1547.
1548.     return 0;
1549. }
1550.
1551.
```

- **Código del Servidor - *main.py***

```
1. import os
2. import socket
3. import requests
4. import re
5. import unicodedata
6. from flask import Flask, request, jsonify, Response
7. from werkzeug.utils import secure_filename
8. from urllib.parse import unquote
9.
10. # ===== #
11. # ----- CONFIGURACIÓN GENERAL ----- #
12. # ===== #
13.
14. current_dir = os.path.dirname(os.path.abspath(_file_))
15.
16. # --- Configuración del servidor ---
17. PORT = 5000
18. THUMB_NAIL_BASE = "https://thumbnails.libretro.com"
19. THUMB_CATEGORIES = ["Named_Boxarts", "Named_Logos", "Named_Snaps", "Named_Titles"]
20. SONGS_DIR = os.path.join(current_dir, "songs")
21. PLAYLIST_FILE = os.path.join(current_dir, "songs_playlist.txt")
22.
23. # ===== #
24. # ----- SERVIDOR FLASK ----- #
25. # ===== #
26.
27. app = Flask(_name_)
28.
29. # IMPORTANTE: Desactivar el límite de tamaño para permitir Streaming manual
30. app.config['MAX_CONTENT_LENGTH'] = None
31.
32.
33. def get_local_ip():
34.     """Obtiene la dirección IP local de la máquina."""
35.     s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
36.     try:
37.         s.connect(("8.8.8.8", 80))
38.         ip = s.getsockname()[0]
39.     except Exception:
40.         ip = "127.0.0.1"
41.     finally:
42.         s.close()
43.     return ip
44.
45.
46. def get_consoles_for_server():
47.     """Detecta consolas disponibles en el directorio del proyecto."""
48.     consoles = set()
49.     # 1. Por archivos de base de datos .txt
50.     for f in os.listdir(current_dir):
51.         if f.lower().endswith('.txt') and f != "songs_playlist.txt":
52.             consoles.add(os.path.splitext(f)[0])
53.
54.     # 2. Por carpetas existentes con 'roms'
55.     for name in os.listdir(current_dir):
56.         p = os.path.join(current_dir, name)
57.         if os.path.isdir(p) and os.path.exists(os.path.join(p, 'roms')):
58.             consoles.add(name)
59.
60.     return sorted(list(consoles))
61.
62.
63. def normalize(text):
```

```
64.     text = (text or '').lower()
65.     text = ''.join(c for c in unicodedata.normalize('NFD', text) if unicodedata.category(c) != 'Mn')
66.     text = re.sub(r'[^a-z0-9]+', ' ', text)
67.     return text.strip()
68.
69.
70. def get_games_from_db(console, query=""):
71.     """Busca en el archivo de texto de la base de datos (para autocompletar nombre)."""
72.     file_path = os.path.join(current_dir, f"{console}.txt")
73.     if not os.path.exists(file_path):
74.         return []
75.
76.     with open(file_path, "r", encoding="utf-8") as f:
77.         games = [line.strip() for line in f if line.strip()]
78.
79.     if not query:
80.         return games
81.
82.     q_norm = normalize(query)
83.     return [g for g in games if q_norm in normalize(g)]
84.
85.
86. def get_installed_roms(console):
87.     """Lista los archivos fisicos que existen en la carpeta roms."""
88.     if "game boy" in console.lower():
89.         console = "Nintendo - Game Boy"
90.
91.     roms_path = os.path.join(current_dir, console, "roms")
92.     if not os.path.exists(roms_path):
93.         return []
94.
95.     files = []
96.     for f in os.listdir(roms_path):
97.         if not f.startswith('.'): # Ignorar ocultos
98.             files.append(f)
99.     return sorted(files)
100.
101.
102. HTML_PAGE = """
103. <!DOCTYPE html>
104. <html lang="es">
105. <head>
106. <meta charset="UTF-8">
107. <meta name="viewport" content="width=device-width, initial-scale=1.0">
108. <title>Cargador Pi Zero 2W</title>
109. <link rel="stylesheet" href="https://bootswatch.com/5/cerulean/bootstrap.min.css">
110. <style>
111.     body { padding-top: 70px; padding-bottom: 50px; }
112.     #drop_zone {
113.         margin-top: 20px; padding: 40px; border: 2px dashed #adb5bd; border-radius: 8px;
114.         background-color: #343a40; color: #ced4da; text-align: center;
115.         font-size: 1.1em; cursor: pointer; transition: all 0.3s;
116.     }
117.     #drop_zone.dragover { background-color: #495057; border-color: #0d6efd; }
118.     #suggestions-list {
119.         position: absolute; z-index: 1000; width: 100%;
120.         max-height: 300px; overflow-y: auto;
121.     }
122.     .list-group-item { background-color: #495057; color: #f8f9fa; border-color: #6c757d; }
123.     .list-group-item-action:hover { background-color: #6c757d; }
124.     .progress { height: 30px; font-weight: bold; font-size: 1.1em; }
125.
126.     /* Estilos para la lista de borrar */
127.     .delete-item { background-color: #fff3cd; color: #856404; border-color: #ffeeba; }
128.     .delete-item:hover { background-color: #ffeeba; }
```

```

129. </style>
130. </head>
131. <body>
132. <nav class="navbar navbar-dark bg-primary fixed-top">
133.   <div class="container-fluid justify-content-center">
134.     <span class="navbar-brand mb-0 h1">Gestor Raspberry Pi Zero</span>
135.   </div>
136. </nav>
137.
138. <div class="container">
139.
140.   <div class="card shadow-sm mb-4">
141.     <div class="card-header bg-dark text-white">Subir Juegos (Instalar)</div>
142.     <div class="card-body">
143.       <div class="mb-3">
144.         <label for="console_select" class="form-label">1. Selecciona la Consola</label>
145.         <select id="console_select" class="form-select"></select>
146.       </div>
147.
148.       <div class="mb-3">
149.         <label class="form-label">2. Indica el Nombre del Juego</label>
150.         <div class="card p-3 bg-light text-dark">
151.           <div class="form-check mb-2">
152.             <input class="form-check-input" type="radio" name="gameOption" id="gameSearchOpt" value="search"
checked>
153.               <label class="form-check-label" for="gameSearchOpt">
154.                 Buscar en BD (Descarga carátulas)
155.               </label>
156.             </div>
157.             <div id="suggestions-container" class="mb-3 position-relative">
158.               <input type="text" id="game_input" class="form-control" autocomplete="off" placeholder="Escribe
para buscar...">
159.               <div id="suggestions-list" class="list-group mt-1"></div>
160.             </div>
161.             <div class="form-check">
162.               <input class="form-check-input" type="radio" name="gameOption" id="gameOtherOpt" value="other">
163.               <label class="form-check-label" for="gameOtherOpt">
164.                 Usar nombre del archivo (Sin carátulas)
165.               </label>
166.             </div>
167.           </div>
168.         </div>
169.
170.         <div class="mb-3">
171.           <label class="form-label">3. Arrastra el Archivo</label>
172.           <div id="drop_zone">Arrastra y suelta el ROM aquí</div>
173.         </div>
174.
175.         <div class="progress mt-3" id="progressContainer" style="display:none;">
176.           <div id="progressBar" class="progress-bar progress-bar-striped progress-bar-animated bg-info"
role="progressbar" style="width: 0%;>0%</div>
177.         </div>
178.
179.         <div id="status" class="mt-3 text-center"></div>
180.       </div>
181.     </div>
182.
183.   <div class="card shadow-sm mb-4 border-warning">
184.     <div class="card-header bg-warning text-dark">Gestionar y Eliminar Juegos Instalados</div>
185.     <div class="card-body">
186.       <div class="mb-3">
187.         <label for="manage_console_select" class="form-label">Selecciona Consola para ver juegos:</label>
188.         <select id="manage_console_select" class="form-select"></select>
189.       </div>
190.       <p class="text-muted small">Haz clic en "Eliminar" para borrar el ROM y todas sus carátulas asociadas.</p>

```

```

191.         <div id="installed_games_list" class="list-group">
192.             <div class="text-center text-muted p-3">Selecciona una consola para cargar la lista...</div>
193.         </div>
194.     </div>
195. </div>
196.
197. <div class="card shadow-sm mt-4">
198.     <div class="card-header bg-success text-white">Música de Fondo</div>
199.     <div class="card-body">
200.         <div class="mb-3">
201.             <div id="songs_list" class="list-group"></div>
202.         </div>
203.         <button id="save_playlist" class="btn btn-primary btn-sm">Guardar Playlist</button>
204.         <div id="songs_status" class="mt-2 text-center"></div>
205.         <hr>
206.         <h6>Subir nueva canción (.mp3, .ogg)</h6>
207.         <div class="input-group">
208.             <input type="file" id="song_file" class="form-control" accept="audio/*">
209.             <button id="upload_song_btn" class="btn btn-outline-success">Subir</button>
210.         </div>
211.         <div id="upload_song_status" class="mt-2 text-center"></div>
212.     </div>
213. </div>
214. </div>
215.
216. <script>
217. // --- VARIABLES GLOBALES ---
218. let selectedConsole = "";
219. let selectedGame = "";
220. const consoleSelect = document.getElementById("console_select");
221. const manageConsoleSelect = document.getElementById("manage_console_select");
222. const gameInput = document.getElementById("game_input");
223. const suggestionsList = document.getElementById("suggestions-list");
224.
225. // --- INICIALIZACIÓN ---
226. document.addEventListener("DOMContentLoaded", async () => {
227.     // Cargar consolas
228.     const res = await fetch("/get_consoles");
229.     const consoles = await res.json();
230.
231.     // Llenar ambos selectores de consola
232.     [consoleSelect, manageConsoleSelect].forEach(sel => {
233.         sel.innerHTML = '<option value="" disabled selected>Selecciona...</option>';
234.         consoles.forEach(c => sel.add(new Option(c, c)));
235.     });
236.
237.     // Cargar canciones
238.     loadSongsAndPlaylist();
239. });
240.
241. // --- LÓGICA DE AUTOCOMPLETADO Y SELECCIÓN (SUBIDA) ---
242. consoleSelect.addEventListener("change", () => {
243.     selectedConsole = consoleSelect.value;
244.     gameInput.value = "";
245.     suggestionsList.innerHTML = "";
246. });
247.
248. gameInput.addEventListener("input", async () => {
249.     if (document.querySelector('input[name="gameOption"]:checked').value !== "search") return;
250.     const query = gameInput.value;
251.     suggestionsList.innerHTML = "";
252.     if (!selectedConsole || query.length < 2) return;
253.
254.     const res = await
fetch(/search_games?console=${encodeURIComponent(selectedConsole)}&query=${encodeURIComponent(query)});

```

```
255.     const games = await res.json();
256.
257.     if (games.length > 0) {
258.         games.slice(0, 10).forEach(g => {
259.             const item = document.createElement("button");
260.             item.type = "button";
261.             item.className = "list-group-item list-group-item-action";
262.             item.textContent = g;
263.             item.onclick = () => {
264.                 gameInput.value = g;
265.                 selectedGame = g;
266.                 suggestionsList.innerHTML = "";
267.             };
268.             suggestionsList.appendChild(item);
269.         });
270.     }
271. });
272. document.body.addEventListener('click', () => { suggestionsList.innerHTML = ''; });
273.
274. // --- LÓGICA DE SUBIDA (STREAMING) ---
275. const dropZone = document.getElementById("drop_zone");
276. const progressContainer = document.getElementById("progressContainer");
277. const progressBar = document.getElementById("progressBar");
278. const statusDiv = document.getElementById("status");
279.
280. dropZone.addEventListener("dragover", e => { e.preventDefault(); dropZone.classList.add("dragover"); });
281. dropZone.addEventListener("dragleave", e => { e.preventDefault(); dropZone.classList.remove("dragover"); });
282. dropZone.addEventListener("drop", e => {
283.     e.preventDefault();
284.     dropZone.classList.remove("dragover");
285.     if (e.dataTransfer.files.length === 0) return;
286.
287.     const file = e.dataTransfer.files[0];
288.     if (!selectedConsole) { alert("Selecciona una consola arriba."); return; }
289.
290.     const option = document.querySelector('input[name="gameOption"]:checked').value;
291.     if (option === "search" && !gameInput.value) { alert("Busca un juego."); return; }
292.
293.     selectedGame = (option === "other") ? file.name.split('.').slice(0, -1).join('.') : gameInput.value;
294.
295.     uploadFileStream(file, selectedConsole, selectedGame);
296. });
297.
298. function uploadFileStream(file, consoleName, gameName) {
299.     const xhr = new XMLHttpRequest();
300.     xhr.open("POST", "/upload_stream", true);
301.     const option = document.querySelector('input[name="gameOption"]:checked').value;
302.
303.     xhr.setRequestHeader("X-Console", encodeURIComponent(consoleName));
304.     xhr.setRequestHeader("X-Game-Name", encodeURIComponent(gameName));
305.     xhr.setRequestHeader("X-Filename", encodeURIComponent(file.name));
306.     xhr.setRequestHeader("X-Game-Option", encodeURIComponent(option));
307.     xhr.setRequestHeader("Content-Type", "application/octet-stream");
308.
309.     progressContainer.style.display = "block";
310.     progressBar.style.width = "0%";
311.     progressBar.textContent = "0%";
312.     progressBar.className = "progress-bar progress-bar-striped progress-bar-animated bg-info";
313.     statusDiv.innerHTML = '<div class="alert alert-light">Iniciando...</div>';
314.
315.     xhr.upload.onprogress = (e) => {
316.         if (e.lengthComputable) {
317.             const percent = Math.round((e.loaded / e.total) * 100);
318.             progressBar.style.width = percent + "%";
319.             progressBar.textContent = percent + "%";
```

```
320.         if (percent === 100) {
321.             statusDiv.innerHTML = '<div class="alert alert-warning">Escribiendo en disco y descargando
carátulas...</div>';
322.         }
323.     }
324. });
325.
326. xhr.onload = () => {
327.     if (xhr.status === 200) {
328.         progressBar.className = "progress-bar bg-success";
329.         statusDiv.innerHTML = '<div class="alert alert-success mt-2">¡Instalado!</div>';
330.         setTimeout(() => {
331.             progressBar.style.display = "none";
332.             statusDiv.innerHTML = "";
333.             gameInput.value = "";
334.             // Si la consola de gestión es la misma, recargar lista
335.             if (manageConsoleSelect.value === consoleName) {
336.                 loadInstalledGames(consoleName);
337.             }
338.         }, 3000);
339.     } else {
340.         progressBar.className = "progress-bar bg-danger";
341.         statusDiv.innerHTML = '<div class="alert alert-danger mt-2">Error: ${xhr.responseText}</div>';
342.     }
343. });
344.
345. xhr.onerror = () => { statusDiv.innerHTML = '<div class="alert alert-danger">Error de red.</div>'; };
346. xhr.send(file);
347. }
348.
349. // --- LÓGICA DE GESTIÓN Y ELIMINADO ---
350. const installedGamesList = document.getElementById("installed_games_list");
351.
352. manageConsoleSelect.addEventListener("change", () => {
353.     loadInstalledGames(manageConsoleSelect.value);
354. });
355.
356. async function loadInstalledGames(consoleName) {
357.     installedGamesList.innerHTML = '<div class="text-center p-3">Cargando...</div>';
358.     const res = await fetch(`/get_installed_games?console=${encodeURIComponent(consoleName)}`);
359.     const files = await res.json();
360.
361.     installedGamesList.innerHTML = "";
362.     if (files.length === 0) {
363.         installedGamesList.innerHTML = '<div class="alert alert-secondary">No hay juegos instalados en esta
consola.</div>';
364.         return;
365.     }
366.
367.     files.forEach(filename => {
368.         const div = document.createElement("div");
369.         div.className = "list-group-item d-flex justify-content-between align-items-center";
370.         div.innerHTML = `
371.             <span>${filename}</span>
372.             <button class="btn btn-danger btn-sm" onclick="deleteGame('${consoleName}', '${filename}')">
373.                 🗑 Eliminar
374.             </button>
375.         `;
376.         installedGamesList.appendChild(div);
377.     });
378. }
379.
380. async function deleteGame(consoleName, filename) {
381.     if (!confirm(`¿Estás seguro de ELIMINAR "${filename}"?\n\nSe borrará el juego y todas sus imágenes asociadas de
forma permanente.)) return;
```

```
382.
383.     try {
384.         const res = await fetch('/delete_game', {
385.             method: 'POST',
386.             headers: { 'Content-Type': 'application/json' },
387.             body: JSON.stringify({ console: consoleName, filename: filename })
388.         });
389.
390.         if (res.ok) {
391.             alert("Juego eliminado correctamente.");
392.             loadInstalledGames(consoleName); // Recargar lista
393.         } else {
394.             const txt = await res.text();
395.             alert("Error al eliminar: " + txt);
396.         }
397.     } catch (e) {
398.         alert("Error de red al intentar eliminar.");
399.     }
400. }
401.
402. // --- LÓGICA CANCIONES (SIN CAMBIOS) ---
403. const songsListDiv = document.getElementById("songs_list");
404. const savePlaylistBtn = document.getElementById("save_playlist");
405. const songsStatusDiv = document.getElementById("songs_status");
406. const songFileInput = document.getElementById("song_file");
407. const uploadSongBtn = document.getElementById("upload_song_btn");
408. const uploadSongStatusDiv = document.getElementById("upload_song_status");
409.
410. async function loadSongsAndPlaylist() {
411.     try {
412.         const [songsRes, playlistRes] = await Promise.all([fetch('/get_songs'), fetch('/get_playlist')]);
413.         const songs = await songsRes.json();
414.         const playlist = await playlistRes.json();
415.         const selectedSet = new Set(playlist);
416.
417.         songsListDiv.innerHTML = '';
418.         if (songs.length === 0) {
419.             songsListDiv.innerHTML = '<div class="alert alert-warning">Sin canciones.</div>';
420.             return;
421.         }
422.
423.         songs.forEach(name => {
424.             const item = document.createElement('label');
425.             item.className = 'list-group-item d-flex align-items-center py-1';
426.             const cb = document.createElement('input');
427.             cb.type = 'checkbox';
428.             cb.className = 'form-check-input me-2';
429.             cb.checked = selectedSet.has(name);
430.             cb.dataset.song = name;
431.
432.             const span = document.createElement('span');
433.             span.textContent = name;
434.
435.             const delBtn = document.createElement('button');
436.             delBtn.className = 'btn btn-sm btn-outline-danger ms-auto';
437.             delBtn.innerHTML = 'X';
438.             delBtn.onclick = (e) => { e.preventDefault(); deleteSong(name); };
439.
440.             item.append(cb, span, delBtn);
441.             songsListDiv.appendChild(item);
442.         });
443.     } catch (e) { console.error(e); }
444. }
445.
446. savePlaylistBtn.addEventListener('click', async () => {
```

```
447.         const selected = Array.from(songsListDiv.querySelectorAll('input[type="checkbox"]:checked')).map(cb =>
cb.dataset.song);
448.         await fetch('/update_playlist', {
449.             method: 'POST',
450.             headers: { 'Content-Type': 'application/json' },
451.             body: JSON.stringify({ songs: selected })
452.         });
453.         songsStatusDiv.innerHTML = '<span class="text-success fw-bold">Guardado OK</span>';
454.         setTimeout(() => songsStatusDiv.innerHTML='', 2000);
455.     });
456.
457. uploadSongBtn.addEventListener('click', async () => {
458.     const file = songFileInput.files[0];
459.     if (!file) return;
460.     const formData = new FormData();
461.     formData.append('file', file);
462.
463.     uploadSongStatusDiv.innerHTML = "Subiendo...";
464.     const res = await fetch('/upload_song', { method: 'POST', body: formData });
465.
466.     if (res.ok) {
467.         uploadSongStatusDiv.innerHTML = '<span class="text-success">OK</span>';
468.         songFileInput.value = '';
469.         loadSongsAndPlaylist();
470.     } else {
471.         uploadSongStatusDiv.innerHTML = '<span class="text-danger">Error</span>';
472.     }
473.     setTimeout(() => uploadSongStatusDiv.innerHTML='', 2000);
474. });
475.
476. async function deleteSong(name) {
477.     if(!confirm('¿Borrar canción?')) return;
478.     await fetch('/delete_song', {
479.         method: 'POST',
480.         headers: { 'Content-Type': 'application/json' },
481.         body: JSON.stringify({ name })
482.     });
483.     loadSongsAndPlaylist();
484. }
485. </script>
486. </body>
487. </html>
488. """
489.
490. @app.route("/")
491. def index():
492.     return HTML_PAGE
493.
494. @app.route("/get_consoles")
495. def get_consoles_route():
496.     return jsonify(get_consoles_for_server())
497.
498. @app.route("/search_games")
499. def search_games_route():
500.     console = request.args.get("console")
501.     query = request.args.get("query", "")
502.     if not console: return jsonify([])
503.     return jsonify(get_games_from_db(console, query))
504.
505. # ===== #
506. # RUTAS DE GESTIÓN (LISTAR Y ELIMINAR)
507. # ===== #
508.
509. @app.route("/get_installed_games")
510. def get_installed_games_route():
```

```
511.     console = request.args.get("console")
512.     if not console: return jsonify([])
513.     return jsonify(get_installed_roms(console))
514.
515. @app.route("/delete_game", methods=['POST'])
516. def delete_game_route():
517.     data = request.get_json()
518.     console = data.get("console")
519.     if "game boy" in console.lower():
520.         console = "Nintendo - Game Boy"
521.     filename = data.get("filename")
522.
523.     if not console or not filename:
524.         return "Faltan datos", 400
525.
526.     # 1. Eliminar el ROM
527.     # Usamos secure_filename pero en este caso necesitamos el nombre exacto del archivo,
528.     # solo aseguramos que no tenga '..'
529.     safe_filename = os.path.basename(filename)
530.     rom_path = os.path.join(current_dir, console, "roms", safe_filename)
531.
532.     deleted_any = False
533.
534.     if os.path.exists(rom_path):
535.         try:
536.             os.remove(rom_path)
537.             deleted_any = True
538.         except Exception as e:
539.             return f"Error borrando ROM: {e}", 500
540.
541.     # 2. Eliminar carátulas asociadas
542.     # Las carátulas usan el nombre base del archivo ROM (sin extensión) y extensión .png
543.     base_name = os.path.splitext(safe_filename)[0]
544.
545.     for category in THUMB_CATEGORIES:
546.         # Nota: aquí hay un detalle sutil. Si renombraste el juego al subirlo,
547.         # la carátula tiene el nombre "bonito" y el ROM también.
548.         # Por lo tanto, buscar imagen con el mismo nombre base que el ROM es correcto.
549.         img_path = os.path.join(current_dir, console, category, f"{base_name}.png")
550.         if os.path.exists(img_path):
551.             try:
552.                 os.remove(img_path)
553.             except:
554.                 pass # Si no se puede borrar la imagen, no es crítico
555.
556.     if not deleted_any and not os.path.exists(rom_path):
557.         return "El archivo no existía (¿ya fue borrado?)", 404
558.
559.     return "OK", 200
560.
561.
562. # ===== #
563. # RUTA DE STREAMING (SOLUCIÓN MEMORIA) - SIN CAMBIOS
564. # ===== #
565. @app.route("/upload_stream", methods=["POST"])
566. def upload_stream():
567.     try:
568.         console = unquote(request.headers.get("X-Console", ""))
569.         game_name = unquote(request.headers.get("X-Game-Name", ""))
570.         raw_filename = unquote(request.headers.get("X-Filename", ""))
571.         game_option = unquote(request.headers.get("X-Game-Option", ""))
572.     except Exception:
573.         return "Error decodificando encabezados", 400
574.
575.     if not console or not game_name or not raw_filename:
```

```

576.         return "Faltan datos en headers", 400
577.
578.     if "game boy" in console.lower():
579.         console_aux = "Nintendo - Game Boy"
580.     else:
581.         console_aux = console
582.
583.     console_dir = os.path.join(current_dir, console_aux)
584.     roms_dir = os.path.join(console_dir, "roms")
585.     os.makedirs(roms_dir, exist_ok=True)
586.
587.     _, extension = os.path.splitext(raw_filename)
588.     new_rom_filename = f"{game_name}{extension}"
589.     save_path = os.path.join(roms_dir, new_rom_filename)
590.
591.     CHUNK_SIZE = 8192
592.     try:
593.         with open(save_path, "wb") as f:
594.             while True:
595.                 chunk = request.stream.read(CHUNK_SIZE)
596.                 if not chunk: break
597.                 f.write(chunk)
598.     except Exception as e:
599.         return f"Error escritura: {e}", 500
600.
601.     print(game_name, console, game_option)
602.     if game_option == "search":
603.         download_thumbnails(console, game_name)
604.
605.     return "OK", 200
606.
607. def download_thumbnails(console, game_name):
608.     if "game boy" in console.lower():
609.         console_aux = "Nintendo - Game Boy"
610.     else:
611.         console_aux = console
612.
613.     for category in THUMB_CATEGORIES:
614.         category_dir = os.path.join(current_dir, console_aux, category)
615.         os.makedirs(category_dir, exist_ok=True)
616.         safe_url_name = game_name.replace(" & ", "").replace(":", "").replace("'", "_")
617.
618.         thumb_url = f"{THUMB_NAIL_BASE}/{requests.utils.quote(console)}/{requests.utils.quote(category)}/{requests.utils.quote(safe_url_name)}.png"
619.         print(f"Intentando descargar carátula desde: {thumb_url}")
620.         try:
621.             r = requests.get(thumb_url, timeout=5)
622.             if r.status_code == 200:
623.                 with open(os.path.join(category_dir, f"{game_name}.png"), "wb") as t:
624.                     t.write(r.content)
625.         except: pass
626.
627. # ===== #
628. # RUTAS DE CANCIONES
629. # ===== #
630.
631. ALLOWED_SONG_EXTS = {'.mp3', '.ogg', '.wav'}
632.
633. def list_songs():
634.     if not os.path.isdir(SONGS_DIR): return []
635.     names = []
636.     for n in os.listdir(SONGS_DIR):
637.         if not n.startswith('.') and os.path.splitext(n)[1].lower() in ALLOWED_SONG_EXTS:
638.             names.append(n)
639.     return sorted(names)
640.
641. def read_playlist():

```

```
640.     if not os.path.exists(PLAYLIST_FILE): return []
641.     with open(PLAYLIST_FILE, 'r', encoding='utf-8') as f:
642.         existing = set(list_songs())
643.         return [ln.strip() for ln in f if ln.strip() in existing]
644.
645. def write_playlist(selected):
646.     with open(PLAYLIST_FILE, 'w', encoding='utf-8') as f:
647.         for s in selected: f.write(f"{s}\n")
648.
649. @app.route('/get_songs')
650. def get_songs_route(): return jsonify(list_songs())
651.
652. @app.route('/get_playlist')
653. def get_playlist_route(): return jsonify(read_playlist())
654.
655. @app.route('/update_playlist', methods=['POST'])
656. def update_playlist_route():
657.     data = request.get_json(silent=True) or {}
658.     try:
659.         write_playlist(data.get('songs', []))
660.         return "OK", 200
661.     except Exception as e: return str(e), 500
662.
663. @app.route('/upload_song', methods=['POST'])
664. def upload_song_route():
665.     if 'file' not in request.files: return "Error", 400
666.     f = request.files['file']
667.     if not f.filename: return "Error", 400
668.     os.makedirs(SONGS_DIR, exist_ok=True)
669.     f.save(os.path.join(SONGS_DIR, secure_filename(f.filename)))
670.     return "OK", 200
671.
672. @app.route('/delete_song', methods=['POST'])
673. def delete_song_route():
674.     data = request.get_json() or {}
675.     name = os.path.basename(data.get('name', ''))
676.     path = os.path.join(SONGS_DIR, name)
677.     if os.path.isfile(path):
678.         os.remove(path)
679.         write_playlist([s for s in read_playlist() if s != name])
680.         return "OK", 200
681.     return "No encontrado", 404
682.
683. if __name__ == "__main__":
684.     ip = get_local_ip()
685.     print(f"Servidor iniciado en http://{ip}:{PORT}")
686.     app.run(host="0.0.0.0", port=PORT, debug=False)
687.
```

ANEXO B — *Gamepad*

Descripción general

El *gamepad* constituye el dispositivo principal de entrada de la consola y permite al usuario interactuar con la interfaz gráfica y los sistemas de emulación. El mismo se implementa mediante un microcontrolador encargado de leer las entradas digitales correspondientes a los botones y las señales analógicas asociadas a los ejes del *joystick*. La información adquirida es procesada localmente y transmitida hacia la unidad principal, donde es interpretada por el *frontend* como eventos de navegación y control.

Arquitectura del sistema

El *firmware* se organiza en los siguientes módulos lógicos:

- **Inicialización del sistema (*setup*):** Configuración de pines digitales con resistencias internas *pull-up*, calibración del punto central del *joystick* y establecimiento de la conexión *BLE-HID*.
- **Gestión de botones digitales:** Implementación de un mecanismo de anti-rebote por *software* (*debounce*) basado en temporización, que evita múltiples detecciones ante una sola pulsación.
- **Procesamiento del *stick* analógico:**
 - Promediado múltiple de muestras *ADC*.
 - Filtro exponencial para suavizado.
 - Zona muerta (*deadzone*).
 - Umbrales diferenciados de entrada y salida (histéresis).
 - Bloqueo dinámico de eje para evitar activaciones diagonales no deseadas.
- **Monitoreo de batería:** Lectura periódica del voltaje mediante *ADC*, conversión a porcentaje de carga, filtrado progresivo y envío del nivel de batería al host mediante el perfil *BLE*.

Código fuente

En las secciones siguientes se presenta el código correspondiente al *firmware* del *gamepad*, incluyendo el módulo principal de lectura de entradas y la implementación del protocolo de comunicación con la unidad principal.

```

1. #include <BleGamepad.h>
2.
3. /* ===== CONFIGURACIÓN GENERAL ===== */
4. #define DEBOUNCE_MS 120
5. #define LOOP_DELAY 2
6.
7. /* ===== PINES ===== */
8. // Botones
9. #define BTN_TRIANGLE 25
10. #define BTN_CIRCLE 27
11. #define BTN_CROSS 26
12. #define BTN_SQUARE 21
13. #define BTN_L 32
14. #define BTN_R 33
15. #define BTN_SELECT 18
16. #define BTN_START 22
17. #define BTN_STICK 16
18.
19. // ADC
20. #define BATTERY_PIN 35
21. #define VRX_PIN 34
22. #define VRY_PIN 39
23.
24. /* ===== JOYSTICK ===== */
25. #define DEADZONE 260
26. #define EDGE_ENTER 520
27. #define EDGE_EXIT 380
28. #define ADC_SAMPLES 8
29.
30. /* ===== BLE GAMEPAD ===== */
31. BleGamepad bleGamepad("ESP32Gamepad", "Ismael11", 100);
32.
33. /* ===== BOTONES ===== */
34. struct Button {
35.     uint8_t pin;
36.     uint8_t id;
37.     const char *name;
38.     bool lastState;
39.     unsigned long lastTime;
40. };
41.
42. Button buttons[] = {
43.     {BTN_TRIANGLE, BUTTON_5, "TRIANGLE", HIGH, 0},
44.     {BTN_CIRCLE, BUTTON_6, "CIRCLE", HIGH, 0},
45.     {BTN_CROSS, BUTTON_7, "CROSS", HIGH, 0},
46.     {BTN_SQUARE, BUTTON_8, "SQUARE", HIGH, 0},
47.     {BTN_SELECT, BUTTON_2, "SELECT", HIGH, 0},
48.     {BTN_START, BUTTON_9, "START", HIGH, 0},
49.     {BTN_L, BUTTON_14, "L", HIGH, 0},
50.     {BTN_R, BUTTON_15, "R", HIGH, 0},
51.     {BTN_STICK, BUTTON_13, "STICK", HIGH, 0},
52. };
53.
54. const uint8_t BUTTON_COUNT = sizeof(buttons) / sizeof(buttons[0]);
55.
56. /* ===== FILTROS ===== */
57. float filtX = 0, filtY = 0;

```

```

58. int centerX = 0, centerY = 0;
59. uint8_t lastHat = HAT_CENTERED;
60.
61. /* ===== BLOQUEO DE EJE ===== */
62. enum StickAxis { AXIS_NONE, AXIS_X, AXIS_Y };
63. StickAxis activeAxis = AXIS_NONE;
64.
65. /* ===== BATERÍA ===== */
66. float filtBattery = 0.0;
67. int lastPercent = -1;
68. unsigned long lastBatteryUpdate = 0;
69.
70. /* ===== UTILIDADES ===== */
71. int readADCStable(uint8_t pin) {
72.     long sum = 0;
73.     for (int i = 0; i < ADC_SAMPLES; i++) {
74.         sum += analogRead(pin);
75.         delayMicroseconds(150);
76.     }
77.     return sum / ADC_SAMPLES;
78. }
79.
80. /* ===== SETUP ===== */
81. void setup() {
82.     Serial.begin(115200);
83.     delay(400);
84.
85.     for (uint8_t i = 0; i < BUTTON_COUNT; i++) {
86.         pinMode(buttons[i].pin, INPUT_PULLUP);
87.
88.         pinMode(BATTERY_PIN, INPUT);
89.         analogSetAttenuation(ADC_11db);
90.
91.         delay(600);
92.         centerX = readADCStable(VRX_PIN);
93.         centerY = readADCStable(VRY_PIN);
94.
95.         filtX = centerX;
96.         filtY = centerY;
97.
98.         bleGamepad.begin();
99.     }
100.
101. /* ===== BOTONES ===== */
102. void updateButtons() {
103.     for (uint8_t i = 0; i < BUTTON_COUNT; i++) {
104.         Button &b = buttons[i];
105.         bool reading = digitalRead(b.pin);
106.
107.         if (reading != b.lastState && millis() - b.lastTime > DEBOUNCE_MS) {
108.             b.lastTime = millis();
109.             b.lastState = reading;
110.
111.             if (reading == LOW) bleGamepad.press(b.id);
112.             else bleGamepad.release(b.id);
113.         }
114.     }
115. }
116.
117. /* ===== JOYSTICK → HAT (CORREGIDO) ===== */
118. void updateJoystick() {
119.     const float alpha = 0.15;
120.
121.     int rawX = readADCStable(VRX_PIN);
122.     int rawY = readADCStable(VRY_PIN);

```

```

123.
124.   filtX += alpha * (rawX - filtX);
125.   filtY += alpha * (rawY - filtY);
126.
127.   int dx = filtX - centerX;
128.   int dy = filtY - centerY;
129.
130.   uint8_t hat = lastHat;
131.
132.   /* ----- Volver al centro ----- */
133.   if (abs(dx) < DEADZONE && abs(dy) < DEADZONE) {
134.       hat = HAT_CENTERED;
135.       activeAxis = AXIS_NONE;
136.   }
137.
138.   /* ----- Elegir eje solo una vez ----- */
139.   if (activeAxis == AXIS_NONE) {
140.       if (abs(dx) > EDGE_ENTER) activeAxis = AXIS_X;
141.       else if (abs(dy) > EDGE_ENTER) activeAxis = AXIS_Y;
142.   }
143.
144.   /* ----- EJE X ----- */
145.   if (activeAxis == AXIS_X) {
146.       if (dx > EDGE_ENTER) hat = HAT_RIGHT;
147.       else if (dx < -EDGE_ENTER) hat = HAT_LEFT;
148.       else if (abs(dx) < EDGE_EXIT) {
149.           hat = HAT_CENTERED;
150.           activeAxis = AXIS_NONE;
151.       }
152.   }
153.
154.   /* ----- EJE Y ----- */
155.   if (activeAxis == AXIS_Y) {
156.       if (dy > EDGE_ENTER) hat = HAT_UP;
157.       else if (dy < -EDGE_ENTER) hat = HAT_DOWN;
158.       else if (abs(dy) < EDGE_EXIT) {
159.           hat = HAT_CENTERED;
160.           activeAxis = AXIS_NONE;
161.       }
162.   }
163.
164.   if (hat != lastHat) {
165.       lastHat = hat;
166.       bleGamepad.setHat(hat);
167.   }
168. }
169.
170. /* ===== BATERÍA ===== */
171. void updateBattery() {
172.     if (millis() - lastBatteryUpdate < 500) return;
173.
174.     long sum = 0;
175.     for (int i = 0; i < 8; i++) {
176.         sum += analogRead(BATTERY_PIN);
177.         delayMicroseconds(200);
178.     }
179.
180.     float raw = sum / 8.0;
181.     float v_adc = raw * (3.3 / 4095.0);
182.     float v_bat = (v_adc * 2.0) + 0.4;
183.
184.     filtBattery = (filtBattery == 0.0) ? v_bat : filtBattery * 0.92 + v_bat * 0.08;
185.
186.     float v = constrain(filtBattery, 3.30, 4.20);
187.     int percent = map(v * 100, 330, 420, 0, 100);

```

```
188. percent = constrain(percent, 0, 100);
189.
190. if (lastPercent >= 0 && abs(percent - lastPercent) < 2)
191.     percent = lastPercent;
192.
193. lastPercent = percent;
194. bleGamepad.setBatteryLevel(percent);
195.
196. lastBatteryUpdate = millis();
197. }
198.
199. /* ===== LOOP ===== */
200. void loop() {
201.     if (bleGamepad.isConnected()) {
202.         updateButtons();
203.         updateJoystick();
204.         updateBattery();
205.     }
206.     delay(LOOP_DELAY);
```

ANEXO C — Planos del encapsulado del dispositivo

Descripción general

En el presente anexo se expone la documentación mecánica correspondiente al encapsulado del dispositivo desarrollado, generada mediante modelado tridimensional asistido por computadora (*CAD*) utilizando el *software FreeCAD*.

Estos planos definen la geometría, dimensiones nominales y características constructivas de los distintos elementos que conforman el gabinete, permitiendo su fabricación mediante procesos de manufactura aditiva y asegurando la correcta integración de los subsistemas electrónicos.

Plano superior

En la Imagen N° 48 se presenta el plano mecánico correspondiente al elemento estructural superior del encapsulado, el cual constituye la cubierta externa del dispositivo y cumple funciones de protección mecánica y cierre del conjunto.

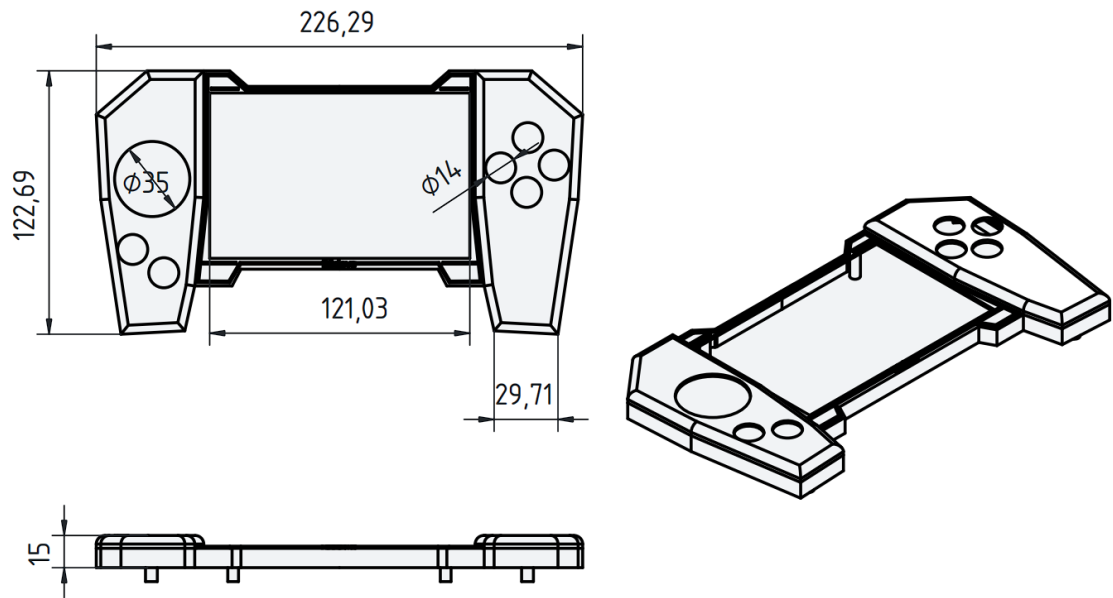


Imagen N° 48: Plano del modelo 3D de la pieza superior de la consola.

Fuente: Elaboración Propia

Plano inferior

En la Imagen N° 49 se muestra el plano mecánico correspondiente al elemento estructural inferior del encapsulado, el cual actúa como soporte primario del sistema, proporcionando la interfaz de montaje para los distintos módulos electrónicos y contribuyendo a la rigidez estructural del conjunto.

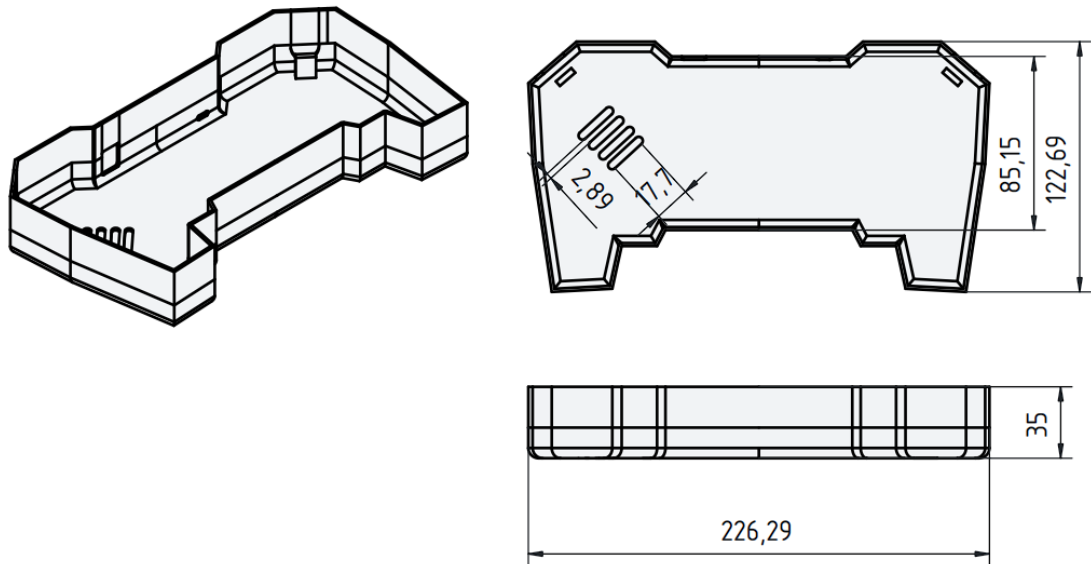


Imagen N° 49: Plano del modelo 3D de la pieza inferior de la consola.

Fuente: Elaboración Propia.

ANEXO D — Documentación de la placa de circuito impreso

Descripción general

En el presente anexo se presenta la máscara de grabado utilizada durante el proceso de fabricación de la placa de circuito impreso mediante el método de ataque químico con cloruro férrico. Esta máscara define el patrón de las pistas conductoras del circuito y fue generada a partir del diseño del *PCB* mediante el *software* de diseño electrónico *KiCAD*. Posteriormente, fue transferida al sustrato de cobre para proteger las áreas conductoras durante el proceso de corrosión química. Este procedimiento permitió eliminar el material excedente y obtener la geometría final de las interconexiones eléctricas del circuito.

Máscara de grabado utilizada

En la Imagen N° 50 se exhibe la máscara de grabado empleada.

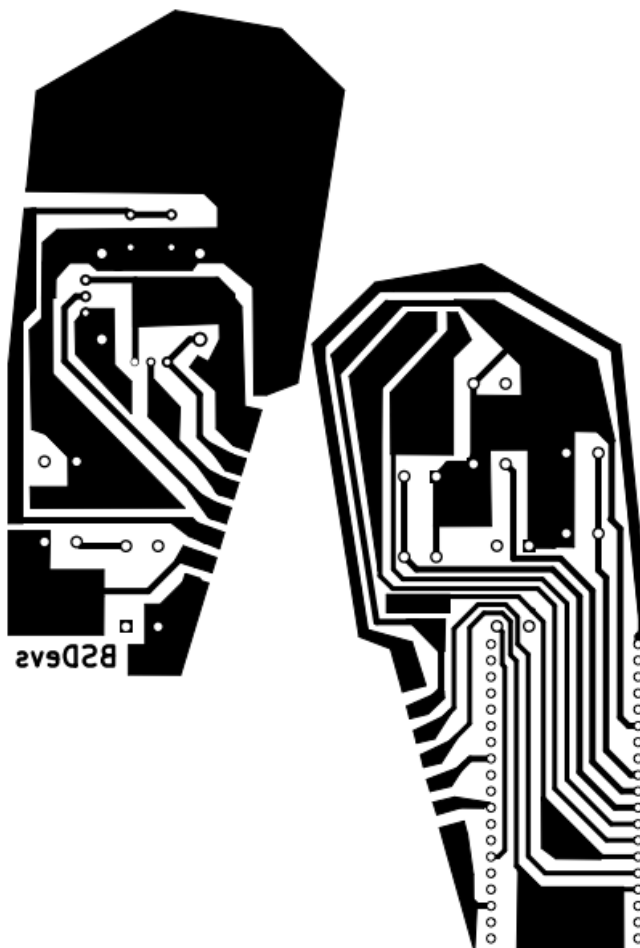


Imagen N° 50: Diseño de la placa de circuito impreso.

Fuente: Elaboración Propia.

Referencias Bibliográficas

- [1] “Raspberry Pi Zero 2 W – Raspberry Pi.” Accessed: Nov. 14, 2025. [Online]. Available: <https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/>
- [2] “RetroArch - Wikipedia, la enciclopedia libre.” Accessed: Jul. 13, 2025. [Online]. Available: <https://es.wikipedia.org/wiki/RetroArch>
- [3] “ESP32 - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: <https://es.wikipedia.org/wiki/ESP32>
- [4] “Bluetooth - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: <https://es.wikipedia.org/wiki/Bluetooth>
- [5] “Wifi - Wikipedia, la enciclopedia libre.” Accessed: Feb. 16, 2026. [Online]. Available: <https://es.wikipedia.org/wiki/Wifi>
- [6] “GameMaker | Make 2D Games With The Free Engine.” Accessed: Feb. 18, 2026. [Online]. Available: <https://gamemaker.io/en>
- [7] “GNU/Linux - Wikipedia, la enciclopedia libre.” Accessed: Feb. 18, 2026. [Online]. Available: <https://es.wikipedia.org/wiki/GNU/Linux>
- [8] “Nintendo Entertainment System - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: https://es.wikipedia.org/wiki/Nintendo_Entertainment_System
- [9] “Super Nintendo - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: https://es.wikipedia.org/wiki/Super_Nintendo
- [10] “Sega Genesis - Wikipedia.” Accessed: Nov. 14, 2025. [Online]. Available: https://en-wikipedia-org.translate.goog/wiki/Sega_Genesis?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc
- [11] “PlayStation (consola) - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: [https://es.wikipedia.org/wiki/PlayStation_\(consola\)](https://es.wikipedia.org/wiki/PlayStation_(consola))
- [12] “Emulador de videoconsola - Wikipedia, la enciclopedia libre.” Accessed: Feb. 22, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Emulador_de_videoconsola
- [13] “Historia de las computadoras personales - Wikipedia, la enciclopedia libre.” Accessed: Feb. 22, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Historia_de_las_computadoras_personales

- [14] “Era de la información - Wikipedia, la enciclopedia libre.” Accessed: Feb. 22, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Era_de_la_informaci%C3%B3n
- [15] “Ingeniería inversa - Wikipedia, la enciclopedia libre.” Accessed: Feb. 22, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Ingenier%C3%ADa_inversa
- [16] “Intérprete (informática) - Wikipedia, la enciclopedia libre.” Accessed: Feb. 22, 2026. [Online]. Available: [https://es.wikipedia.org/wiki/Int%C3%A9rprete_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Int%C3%A9rprete_(inform%C3%A1tica))
- [17] “Dynamic recompilation - Wikipedia.” Accessed: Feb. 22, 2026. [Online]. Available: https://en-wikipedia-org.translate.goog/wiki/Dynamic_recompilation?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc
- [18] A. Bender and J. Cusidó Titulació, “Desarrollo de una consola de videojuegos”.
- [19] “Raspberry Pi computer hardware - Raspberry Pi Documentation.” Accessed: Feb. 16, 2026. [Online]. Available: <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>
- [20] D. Fernández Arlandiz, “Diseño y desarrollo de una consola retro,” May 16, 2022, *Universitat Politècnica de Catalunya*. Accessed: Feb. 16, 2026. [Online]. Available: <https://hdl.handle.net/2117/369659>
- [21] “Game Boy - Wikipedia, la enciclopedia libre.” Accessed: Feb. 16, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Game_Boy
- [22] C. Ramirez Muñoz, “DESARROLLO DE UNA CONSOLA DE VIDEOJUEGOS RETRO Memoria y Anexos Autor”.
- [23] “Matriz de puerta programable en campo - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: https://es.wikipedia.org/wiki/Matriz_de_puerta_programable_en_campo
- [24] “Magnavox Odyssey - Wikipedia, la enciclopedia libre.” Accessed: Feb. 16, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Magnavox_Odyssey
- [25] “Preservación digital - Wikipedia, la enciclopedia libre.” Accessed: Feb. 22, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Preservaci%C3%B3n_digital
- [26] “Video game culture - Wikipedia.” Accessed: Feb. 22, 2026. [Online]. Available: https://en.wikipedia.org/wiki/Video_game_culture

- [27] “Historia de los videojuegos - Wikipedia, la enciclopedia libre.” Accessed: Feb. 22, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Historia_de_los_videojuegos
- [28] “High-Definition Multimedia Interface - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: https://es.wikipedia.org/wiki/High-Definition_Multimedia_Interface
- [29] “C (lenguaje de programación) - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: [https://es.wikipedia.org/wiki/C_\(lenguaje_de_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n))
- [30] “Imagen ROM - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: https://es.wikipedia.org/wiki/Imagen_ROM
- [31] “Consola Portátil Retro R36s 2025 64gb Violeta | Mismo precio en cuotas.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.mercadolibre.com.ar/consola-portatil-retro-r36s-2025-64gb/up/MLAU3756881419#polycard_client=search-desktop&search_layout=grid&position=7&type=product&tracking_id=28919c81-a588-4866-bc82-db589da43be6&wid=MLA2817174092&sid=search
- [32] “Consola De Juegos Portátil Trimui Smart Pro De 5 Pulgadas | Envío gratis.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.mercadolibre.com.ar/trimui-smart-pro-console-de-jogo-portatil-de-5-polegadas/p/MLA2037223999#polycard_client=search-desktop&search_layout=grid&position=5&type=product&tracking_id=0818ae84-564e-40ee-ae0d-c83193614c4a&wid=MLA1614626665&sid=search
- [33] “Nintendo DS - Wikipedia, la enciclopedia libre.” Accessed: Feb. 18, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Nintendo_DS
- [34] “Consola De Juegos Retro Portátil Hd Con Gran Espacio De Alma | Envío gratis.” Accessed: Feb. 16, 2026. [Online]. Available: https://articulo.mercadolibre.com.ar/MLA-2636123678-consola-de-juegos-retro-portatil-hd-con-gran-espacio-de-alma-_JM?searchVariation=193794725573#polycard_client=search-desktop&searchVariation=193794725573&search_layout=grid&position=9&type=item&tracking_id=4c264a36-5c81-4779-9d52-5eeb3ca79e3b

- [35] “Consola Portatil Pocket 6 Pro 12gb Ram 256gb Violeta | Envío gratis.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.mercadolibre.com.ar/consola-portatil-pocket-6-pro--12gb-ram-256gb/up/MLAU3782591187#polycard_client=search-desktop&search_layout=grid&position=5&type=product&tracking_id=1e521865-3d51-4ad0-aef0-4b4814691e7c&wid=MLA2856113180&sid=search
- [36] “Nintendo Switch - Wikipedia, la enciclopedia libre.” Accessed: Feb. 18, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Nintendo_Switch
- [37] “Consola Portátil M17 - Color Gris Gris | Envío gratis.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.mercadolibre.com.ar/consola-portatil-m17--color-gris/up/MLAU3521140155#polycard_client=search-desktop&search_layout=grid&position=9&type=product&tracking_id=a288191a-4bbd-4f99-aa52-c689e74eff47&wid=MLA1569932403&sid=search
- [38] “Simulación (informática) - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: [https://es.wikipedia.org/wiki/Simulaci%C3%B3n_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Simulaci%C3%B3n_(inform%C3%A1tica))
- [39] “Virtualización - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: <https://es.wikipedia.org/wiki/Virtualizaci%C3%B3n>
- [40] “MIPS (procesador) - Wikipedia, la enciclopedia libre.” Accessed: Feb. 18, 2026. [Online]. Available: [https://es.wikipedia.org/wiki/MIPS_\(procesador\)](https://es.wikipedia.org/wiki/MIPS_(procesador))
- [41] “Arquitectura ARM - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: https://es.wikipedia.org/wiki/Arquitectura_ARM
- [42] “High and low-level emulation - Emulation General Wiki.” Accessed: Nov. 14, 2025. [Online]. Available: https://emulation.gametechniki.com/index.php/High/Low_level_emulation
- [43] “CPU - NESdev Wiki.” Accessed: Nov. 14, 2025. [Online]. Available: <https://www.nesdev.org/wiki/CPU>
- [44] “PPU - NESdev Wiki.” Accessed: Nov. 14, 2025. [Online]. Available: <https://www.nesdev.org/wiki/PPU>
- [45] “APU - NESdev Wiki.” Accessed: Nov. 14, 2025. [Online]. Available: <https://www.nesdev.org/wiki/APU>
- [46] “BIOS - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: <https://es.wikipedia.org/wiki/BIOS>

- [47] “Microcontrolador - Wikipedia, la enciclopedia libre.” Accessed: Nov. 14, 2025. [Online]. Available: <https://es.wikipedia.org/wiki/Microcontrolador>
- [48] “Libretro – A crossplatform application API, powering the crossplatform gaming platform RetroArch.” Accessed: Nov. 14, 2025. [Online]. Available: <https://www.libretro.com/>
- [49] “Flask - Wikipedia, la enciclopedia libre.” Accessed: Feb. 22, 2026. [Online]. Available: <https://es.wikipedia.org/wiki/Flask>
- [50] “ARM Cortex-A53 - Wikipedia, la enciclopedia libre.” Accessed: Feb. 22, 2026. [Online]. Available: https://es.wikipedia.org/wiki/ARM_Cortex-A53
- [51] “VideoCore - Wikipedia.” Accessed: Feb. 22, 2026. [Online]. Available: <https://en.wikipedia.org/wiki/VideoCore>
- [52] “System in a package - Wikipedia.” Accessed: Feb. 22, 2026. [Online]. Available: https://en.wikipedia.org/wiki/System_in_a_package
- [53] “Sistema en un chip - Wikipedia, la enciclopedia libre.” Accessed: Feb. 22, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Sistema_en_un_chip
- [54] “Linux embebido - Wikipedia, la enciclopedia libre.” Accessed: Feb. 22, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Linux_embebido
- [55] “Raspberry pi zero w headless setup wifi - jasmis.” Accessed: Feb. 16, 2026. [Online]. Available: <https://jasmis.weebly.com/blog/raspberry-pi-zero-w-headless-setup-wifi>
- [56] “NodeMCU-32s – ESP32 ESP-WROOM-32 Development Board – CircuitSetup.” Accessed: Feb. 16, 2026. [Online]. Available: <https://circuitsetup.us/product/nodemcu-32s-esp32-esp-wroom-32-development-board/?v=c582dec943ff>
- [57] “5 inch HDMI Display for Raspberry Pi – Digitelectronics.lk.” Accessed: Feb. 16, 2026. [Online]. Available: <https://digitelectronics.lk/product/5-inch-hdmi-display-for-raspberry-pi/>
- [58] “Raspberry Pi OS (64-bit) - Raspberry Pi.” Accessed: Feb. 16, 2026. [Online]. Available: <https://www.raspberrypi.com/news/raspberry-pi-os-64-bit/>
- [59] “Raspberry Pi Zero 2w - Made In Uk | MercadoLibre.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.mercadolibre.com.ar/raspberry-pi-zero-2w-made-in-uk/p/MLA35340704#polycard_client=search-

- desktop&search_layout=grid&position=1&type=product&tracking_id=c787566e-cc5c-44c7-a38a-ee3a51afcd09&wid=MLA1476733635&sid=search
- [60] “(3) 5 X Cargador Bateria Litio Tp4056 1a Fuente Boost Step Up 2a | MercadoLibre.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.mercadolibre.com.ar/5-x-cargador-bateria-litio-tp4056-1a-fuente-boost-step-up-2a/up/MLAU2907964464#polycard_client=search-desktop&search_layout=grid&position=10&type=product&tracking_id=0e95b1c1-5e7c-4369-9161-34001341f7d9&wid=MLA1981112606&sid=search
- [61] “(3) Modulo Amplificador Audio Pam8403 2x3w Pam 8403 - Pack X 3 | MercadoLibre.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.mercadolibre.com.ar/modulo-amplificador-audio-pam8403-2x3w-pam-8403--pack-x-3/up/MLAU255218286#polycard_client=search-desktop&search_layout=grid&position=2&type=product&tracking_id=85e1cea5-4882-447b-96b2-4becbc3dbb71&wid=MLA1156552792&sid=search
- [62] “Lcd Touch 5” Hdmi Raspberry Pi Pantalla Tactil Beaglebone | MercadoLibre.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.mercadolibre.com.ar/pantalla-tactil-raspberry-pi-lcd-touch-5-800-x-480-hdmi/p/MLA2045601485#polycard_client=search-desktop&search_layout=grid&position=5&type=product&tracking_id=bc05997a-2a57-4c05-81e6-b54e5dc4010f&wid=MLA864457610&sid=search
- [63] “Parlantes Para Auriculares Akg K55 K420 K430 K450 K55 32ohm Negro | MercadoLibre.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.mercadolibre.com.ar/parlantes-para-auriculares-akg-k55-k420-k430-k450-k55-32ohm/up/MLAU182323398#polycard_client=search-desktop&search_layout=grid&position=1&type=product&tracking_id=5dba7e54-eb56-4f72-b5e5-3a8f38d59cae&wid=MLA1391423889&sid=search
- [64] “Bateria Litio Polimero 3.7v 4000mah 606090 | MercadoLibre.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.mercadolibre.com.ar/bateria-litio-polimero-37v-4000mah-606090/p/MLA2063688621#polycard_client=search-desktop&search_layout=grid&position=11&type=product&tracking_id=4a056815-cc80-4063-840c-754b8b5c31b8&wid=MLA880316987&sid=search
- [65] “(3) Fuente Mt3608 Step Up Buck Boost Elevador Dc Dc 28v 2a X 2u | MercadoLibre.” Accessed: Feb. 16, 2026. [Online]. Available:

- https://www.mercadolibre.com.ar/fuente-mt3608-step-up-buck-boost-elevador-dc-dc-28v-2a-x-2u/up/MLAU3508747894#polycard_client=search-desktop&search_layout=grid&position=1&type=product&tracking_id=b46954c4-9aca-4891-81ce-579c4bbd509d&wid=MLA2472856810&sid=search
- [66] “Cooler Ventilador 30 X 30mm 5v Para Raspberry / Overclock | MercadoLibre.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.mercadolibre.com.ar/cooler-ventilador-30-x-30mm-5v--para-raspberry--overclock/up/MLAU2998726599#polycard_client=search-desktop&search_layout=grid&position=11&type=product&tracking_id=6c3f5866-5aa6-4d89-84c0-02f15afa0121&wid=MLA1475088699&sid=search
- [67] “Nodemcu Esp32 38 Pines Micro USB Wifi Bluetooth 4.2 Compatible Arduino | MercadoLibre.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.mercadolibre.com.ar/nodemcu-esp32-38-pines-micro-usb-wifi-bluetooth-42-compatible-arduino/p/MLA34891403#polycard_client=search-desktop&search_layout=grid&position=5&type=product&tracking_id=ae460e7c-7b45-44df-a143-ad4b03d75a2a&wid=MLA1418532703&sid=search
- [68] “FreeCAD: Su modelador paramétrico en 3D.” Accessed: Feb. 16, 2026. [Online]. Available: https://www.freecad.org/index.php?lang=es_ES
- [69] “ender-3-pro-3d-printer.” Accessed: Feb. 16, 2026. [Online]. Available: <https://www.creality.com/es/download/ender-3-pro-3d-printer>
- [70] “Ácido poliláctico - Wikipedia, la enciclopedia libre.” Accessed: Feb. 16, 2026. [Online]. Available: https://es.wikipedia.org/wiki/%C3%81cido_polil%C3%A1ctico
- [71] “Fork (llamada al sistema) - Wikipedia.” Accessed: Feb. 16, 2026. [Online]. Available: [https://en-wikipedia-org.translate.goog/wiki/Fork_\(system_call\)?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc](https://en-wikipedia-org.translate.goog/wiki/Fork_(system_call)?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc)
- [72] “Pruebas de duración de batería - YouTube.” Accessed: Feb. 18, 2026. [Online]. Available: <https://www.youtube.com/watch?v=fd3GeheQrO8>
- [73] “ESP32-C3 Wi-Fi & BLE 5 SoC | Espressif Systems.” Accessed: Feb. 17, 2026. [Online]. Available: <https://www.espressif.com/en/products/socs/esp32-c3>

- [74] “Universal Asynchronous Receiver-Transmitter - Wikipedia, la enciclopedia libre.” Accessed: Feb. 17, 2026. [Online]. Available: https://es.wikipedia.org/wiki/Universal_Asynchronous_Receiver-Transmitter
- [75] “NodeMCU ESP32 Wifi + Bluetooth 4.2 IOT Wroom ESP32S - Placa de Desar.” Accessed: Feb. 17, 2026. [Online]. Available: <https://arsumo.com.ar/productos/nodemcu-esp32-wifi-bluetooth-4-2-iot-wroom-esp32s-arduino/>
- [76] “Videojuego desarrollado con GameMaker Studio corriendo en consola portátil Raspberry Pi Zero 2 W - YouTube.” Accessed: Feb. 18, 2026. [Online]. Available: <https://www.youtube.com/watch?v=CXbRh4dSm3g>
- [77] “RetroArch-Frontend/project at c_final_version · IsmaelBecker13/RetroArch-Frontend.” Accessed: Feb. 24, 2026. [Online]. Available: https://github.com/IsmaelBecker13/RetroArch-Frontend/tree/c_final_version/project
- [78] “christianhaitian/retroarch-cores: Various libretro cores built for arm64 (aarch64) and armhf (arm32) architecture. Primarily used by ArkOS and TheRA and welcomed to be used for other projects.” Accessed: Feb. 17, 2026. [Online]. Available: <https://github.com/christianhaitian/retroarch-cores/tree/master>

Formulario de autorización de depósito de tesis/trabajo final integrador en la Comunidad Ingenierías y Tecnologías del RIDUNaM (Repositorio Institucional Digital de la UNaM)

Por intermedio de la presente, el abajo firmante, AUTOR del TFI (Grado) titulada Diseño e implementación de una consola portátil de emulación de videojuegos retro basada en Raspberry Pi Zero 2 W

Da FE de la autoría y originalidad de la obra mencionada, que fue dirigida por el Dr. Ing. CARBALLO, Roberto Esteban, presentada y defendida en la Facultad de Ingeniería de la Universidad Nacional de Misiones (FI-UNaM), el (fecha) ...13.../...03.... /...2026....., Acta/Expte. N° 205422 con el fin de obtener el título de Ingeniero en computación.

Tildar según corresponda

- ☐ Tesis de Posgrado
☐ Doctorado ☐ Maestría ☒ **Trabajo Final Integrador**
☐ Tesis de Grado

Derechos patrimoniales

Como autor, expreso mi conformidad en cuanto a la cesión gratuita de los derechos de reproducción y circulación de esta obra, en forma NO EXCLUSIVA, a la Facultad de **Ingeniería**-UNaM. Dicha reproducción y circulación se podrá realizar, una o varias veces, en cualquier soporte, para todo el mundo, con fines sociales, educativos y científicos.

En virtud del carácter no exclusivo de esta cesión, el autor podrá reproducir y comunicar libremente la tesis o trabajo final integrador, a través de los medios que estime oportunos.

Condiciones de acceso en línea

- ☒ Autorizo el depósito del trabajo final integrador en forma inmediata
☐ Autorizo el depósito del documento con embargo por el plazo de _____ meses a partir de la defensa de la misma.

Condiciones de uso TFI

Será puesta a disposición pública bajo las siguientes condiciones de uso:

	(BY) Atribución — Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciante (pero no de una manera que sugiera que tiene su apoyo o que apoyan el uso que hace de su obra).
	(NC) No Comercial — No puede utilizar esta obra para fines comerciales.
	(SA) Permite trabajos derivados — Siempre que se mantenga la misma licencia.
	Reconocimiento – NoComercial – CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Referencias:

CC (Licencias Creative Commons).

BY (Atribución).

NC (No comercial).

SA (Compartir igual).

Dados personales (llenar un cuadro por cada autor)

Apellido y Nombres	BECKER, Ismael Lucas
Teléfono/Celular	+54 9 3757 511177
Correo electrónico	ismaelbecker90@gmail.com

Apellido y Nombres	STEHR, Mateo Nicolás
Teléfono/Celular	+54 9 3751 616248
Correo electrónico	matistehr@gmail.com

Se firma la presente en la Ciudad de Oberá, Misiones a los.....05..... días del mes
 de..... mayo..... de..... 2026.....-

BECKER Ismael

Firma

Becker

Firma

DNI - 44987752

Tipo y N° Documento

DNI - 43702536

Tipo y N° Documento

BECKER ISMAEL

Aclaración

Stehr Mateo Nicolás

Aclaración