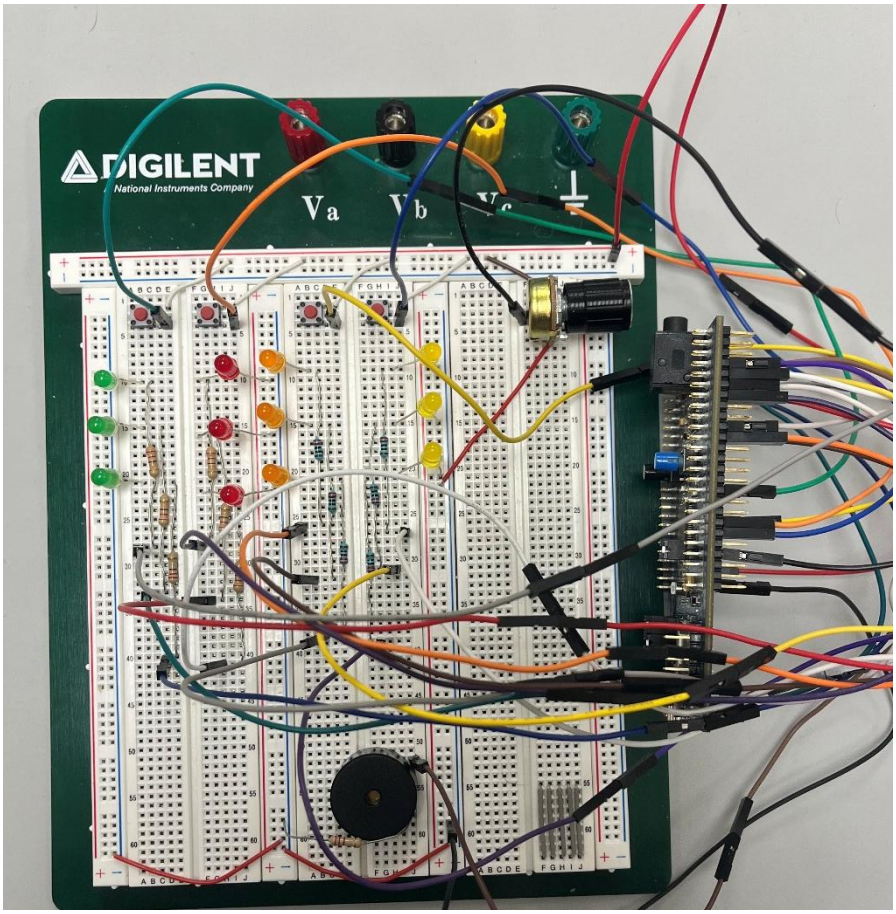
TRABAJO MICROS



GRUPO 24

Mario Fernández Liétor, 55839

Sergio Ballesteros Palomo, 55736

Alejandro Bonilla Hervás, 55756

ÍNDICE

1. INTRODUCCIÓN Y OBJETIVOS

1.1. Enunciado

El objetivo de este proyecto es la realización de una aplicación basada en microcontroladores utilizando la tarjeta de desarrollo STM32F407G-DISC1. El diseño consiste en un juego de habilidad musical y coordinación (estilo "Guitar Hero" o "Piano Tiles").

El sistema debe cumplir requisitos técnicos específicos como la modularización del código, el uso de máquinas de estados finitos (FSM), la gestión de entradas mediante interrupciones hardware (EXTI) y la generación de señales mediante temporizadores (Timers/PWM).

2. DESCRIPCIÓN DEL PROYECTO

2.1. Funcionamiento General

El sistema implementa una consola de juego portátil donde el usuario debe pulsar botones sincronizados con una secuencia de luces (notas musicales). El proyecto se gestiona mediante una máquina de estados con los siguientes estados.

- Estado de ESPERA (Menú): Los LEDs de juego parpadean en secuencia visual. El sistema espera a que el usuario seleccione una de las 4 canciones disponibles pulsando uno de los 4 botones. Durante esta fase, el usuario puede ajustar la dificultad (velocidad) usando el potenciómetro.

- Estado de CONFIGURACIÓN: Una vez seleccionada la canción, el sistema carga los punteros de memoria correspondientes a la melodía y las notas elegidas (separadas en archivos externos) y prepara el inicio del juego.

- Estado de JUEGO: Es el núcleo del programa. Las notas (LEDs) caen por 4 carriles visuales. El usuario debe pulsar el botón correspondiente justo cuando la nota llega a la zona de "meta". Una vez pulsado tenemos tres posibles opciones:

  - Acierto: Suena la nota musical correcta generada por PWM y el juego continúa.
  - Fallo: Si el usuario no pulsa, o pulsa el botón incorrecto (sistema anti-trampas), pierde una vida.
  - Game Over: Si se pierden las 2 vidas iniciales, el juego termina y suena un tono de derrota.

- Estado FIN: Pausa final antes de reiniciar el sistema al menú principal.

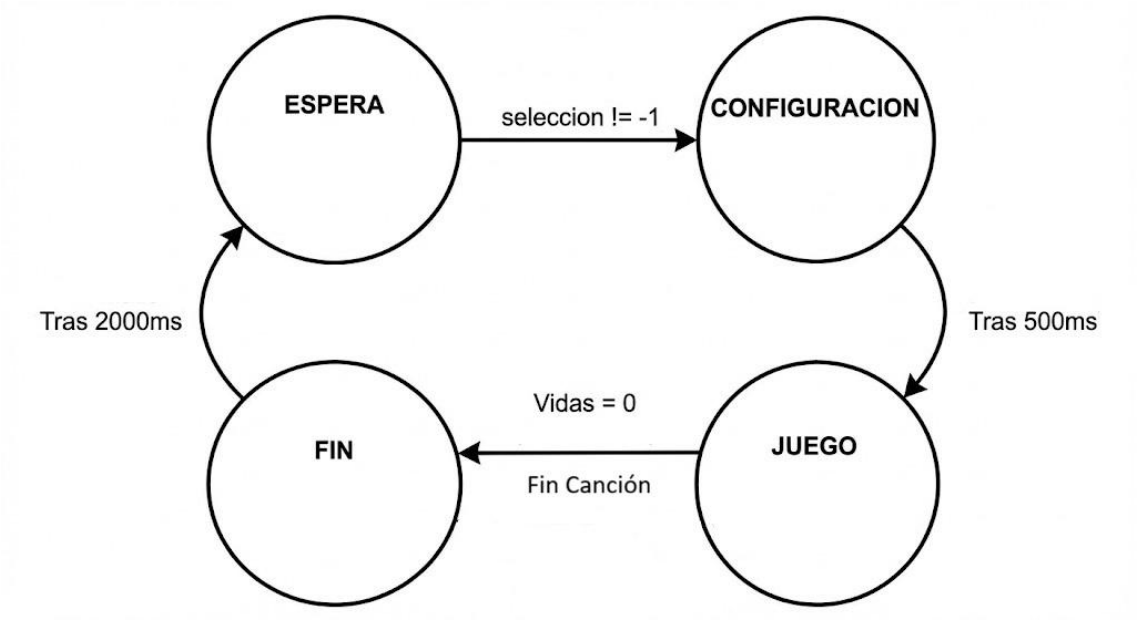


Imagen 1. Diagrama de estados.

2.2. Componentes Empleados

Para la implementación física se han utilizado los siguientes elementos conectados al microcontrolador STM32F407:

- Microcontrolador STM32F407VGTx: Cerebro del sistema encargado de la lógica, temporización y gestión de E/S.
- Interfaz de Entrada (Botones): 4 pulsadores conectados a los pines PA1, PA2, PA3 y PC5, configurados con resistencias Pull-Down internas para evitar estados flotantes.
- Interfaz de Salida (LEDs):
  -Matriz de juego: 12 LEDs divididos en 3 filas (Inicio, Medio, Meta) y 4 columnas (Carriles).
  -Indicadores de Vidas: 2 LEDs adicionales (PD0, PD1) para mostrar la salud del jugador.
  -Actuador Acústico (Buzzer): Zumbador pasivo conectado al pin PD12, gestionado por el Timer 4 para generar frecuencias musicales.
  -Sensor Analógico (Potenciómetro): Conectado al ADC1 para regular la velocidad de caída de las notas (dificultad) en tiempo real.

2.3. Manual de Usuario

Para garantizar una correcta interacción con el sistema, se han definido una serie de pasos secuenciales que permiten al usuario configurar la partida y operar el dispositivo de forma intuitiva.

1. Identificación de Elementos Antes de iniciar el sistema.

- Es necesario identificar los controles físicos disponibles en la interfaz de hardware:

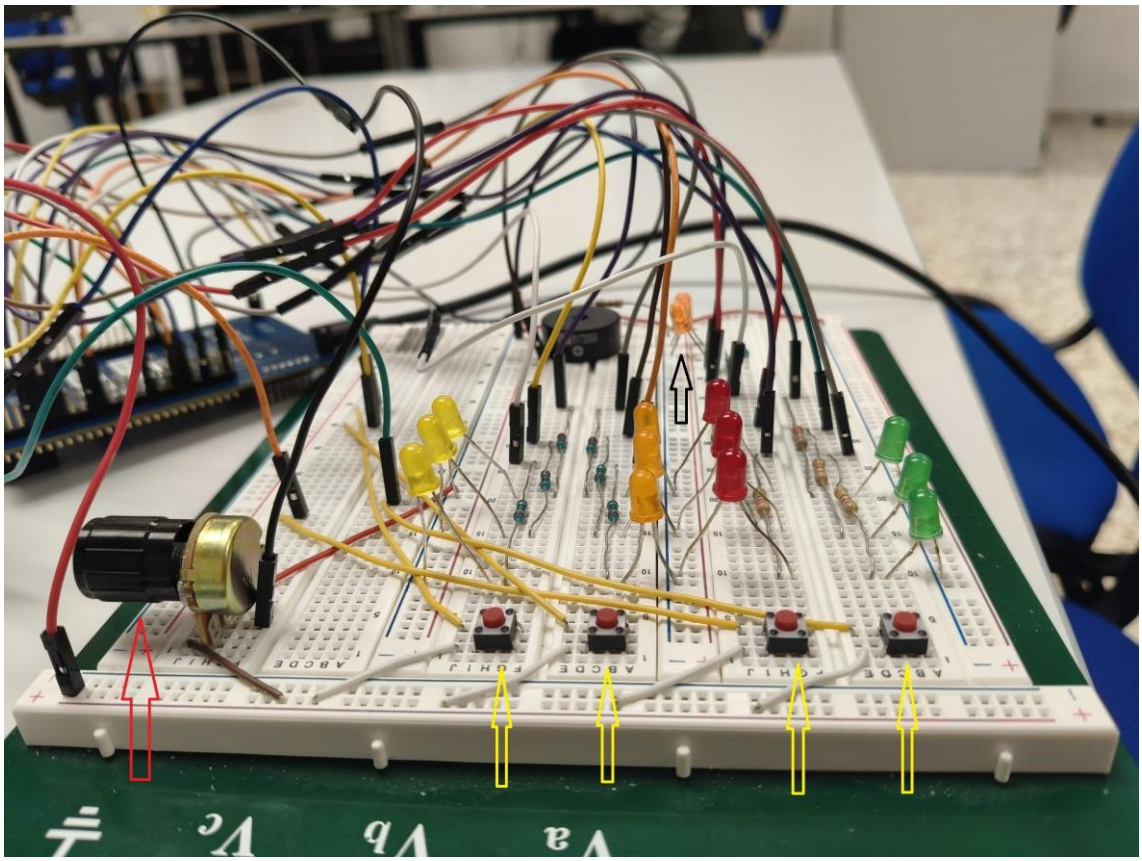


Imagen 2. Vista general de la interfaz de usuario con los controles identificados.

- Flecha Roja (Potenciómetro): Control analógico giratorio. Permite ajustar la velocidad de caída de las notas (dificultad) en tiempo real.
- Flechas Amarillas (Pulsadores): Botones de interacción principal. Cada uno corresponde a un carril de juego. Se utilizan tanto para seleccionar la canción en el menú como para "atrapar" las notas durante la partida.
- Flecha Negra (LED Indicador): Indicador visual de estado/vidas. Permite conocer la salud del jugador o confirmar acciones del menú.
- Zona Central (Matriz de LEDs): Conjunto de diodos luminosos organizados en filas y columnas que representan la caída de las notas musicales.

2. Puesta en Marcha.

- Conectar la tarjeta STM32F407 al ordenador mediante el cable USB.
- Al recibir alimentación, el sistema entrará automáticamente en el Estado de ESPERA.
- Indicador Visual: Los LEDs de la matriz realizarán un parpadeo secuencial y los indicadores de vida se encenderán fijos.

3. Configuración de la Partida.

- Ajuste de Dificultad: Gire el potenciómetro (Flecha Roja). Hacia la izquierda disminuirá la velocidad (menor dificultad) y hacia la derecha aumentará.
- Selección de Canción: Pulse cualquiera de los 4 botones (Flechas Amarillas) para elegir la melodía. Cada botón carga una pista distinta pregrabada en memoria.

4. Mecánica de Juego.

- El objetivo es pulsar el botón correspondiente al carril por el que desciende la luz.
- Momento Exacto: La pulsación debe realizarse cuando la luz llegue a la última fila de LEDs (justo encima de los botones).
- Feedback Sonoro:
    - Si acierta: Sonará la nota musical correspondiente a la melodía a través del Buzzer (situado tras los cables).
    - Si falla: No sonará la nota y se penalizará restando una vida.

5. Fin de Partida.

- El juego termina si la melodía finaliza con éxito o si el contador de vidas llega a cero. Tras una pausa de 2 segundos, el sistema se reinicia automáticamente.

3. CÓDIGO

3.1. Estructura

Para cumplir con los requisitos de modularidad, el software se ha dividido en archivos de cabecera (.h) y archivos fuente (.c).

- main.c: Contiene la máquina de estados principal, la lógica del juego, la configuración de periféricos y la función de retrollamada (callback) de las interrupciones.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
      {
          /* Filtro anti-rebotes por software (200ms) */
          static uint32_t ultimo_tiempo = 0;
          uint32_t tiempo_actual = HAL_GetTick();

          if (tiempo_actual - ultimo_tiempo < 200)
          {
              return; /* Ignorar pulsacion (rebote) */
          }

          ultimo_tiempo = tiempo_actual;

          /* Actualizamos variable global segun el boton pulsado */
          if(GPIO_Pin == GPIO_PIN_1) boton_pulsado_irq = 0;      /* Boton Amarillo */
          else if(GPIO_Pin == GPIO_PIN_2) boton_pulsado_irq = 1; /* Boton Naranja */
          else if(GPIO_Pin == GPIO_PIN_3) boton_pulsado_irq = 2; /* Boton Rojo */
          else if(GPIO_Pin == GPIO_PIN_5) boton_pulsado_irq = 3; /* Boton Blanco */
      }
```

Imagen 3. Callback de los botones.

```
if(ha_acertado_en_meta == 1)
{
    /* Acierto */
    int frecuencia = 300000 / pNotas[i];
    __HAL_TIM_SET_AUTORELOAD(&htim4, frecuencia);
    __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, frecuencia / 4);
    __HAL_TIM_SET_COUNTER(&htim4, 0);
    HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
    HAL_Delay(120);
    HAL_TIM_PWM_Stop(&htim4, TIM_CHANNEL_1);

    /* Seguimos al siguiente bucle (siguiente nota) */
    continue;
}
```

Imagen 4. Reproducción de la nota mediante PWM.

```c
for(int t = 0; t < 50; t++)
{
  if(boton_pulsado_irq != -1)
  {
      /* Si pulsa, comprobamos si es el correcto */
      if(carril == boton_pulsado_irq)
      {
          ha_acertado_en_meta = 1;
          break;
      }
      else
      {
          /* Fallo si pulso el boton incorrecto*/
          fallo_cometido = 1;
          break;
      }
  }
```

Imagen 5. Comprobación de la pulsación al llegar al final.

- canciones.c / canciones.h: Archivos dedicados exclusivamente a almacenar los datos de las melodías (arrays con el orden de los carriles y las frecuencias de las notas). Esto permite añadir nuevas canciones sin modificar la lógica principal del juego.

```c
/* Contiene los arrays de datos (carriles y notas) para las melodias del juego.*/
#include "canciones.h"
/* --- CUMPLEANOS FELIZ (Version extendida) --- */
int c_cumple[] = {0, 0, 1, 0, 3, 2, 0, 0, 1, 0, 2, 3, 0, 0, 3, 1, 3, 2, 1, 1, 1, 2, 3, 2, 3};
int n_cumple[] = {55, 55, 57, 55, 60, 59, 55, 55, 57, 55, 62, 60, 55, 55, 67, 64, 60, 59, 57, 65, 65, 64, 60, 62, 60};
int t_cumple = 25;

/* --- ESTRELLITA DONDE ESTAS --- */
int c_estrella[] = {0, 0, 3, 3, 1, 1, 0, 2, 2, 1, 1, 3, 3, 2, 3, 3, 1, 1, 2, 2, 1, 3, 3, 1, 1, 2, 2, 1, 0, 0, 3, 3, 1, 1, 0};
int n_estrella[] = {60, 60, 67, 67, 69, 69, 67, 65, 65, 64, 64, 62, 62, 60, 67, 67, 65, 65, 64, 64, 62, 67, 67, 65, 65, 64, 64, 62, 60, 60, 67, 67, 69, 69, 60};
int t_estrella = 35;

/* --- STAR WARS (Marcha Imperial) --- */
int c_starwars[] = {1, 1, 1, 0, 2, 1, 0, 2, 1, 3, 3, 3, 0, 2, 1, 0, 2, 1};
int n_starwars[] = {55, 55, 55, 51, 58, 55, 51, 58, 55, 62, 62, 62, 63, 58, 55, 51, 58, 55};
int t_starwars = 18;

/* --- HIMNO DE LA ALEGRIA --- */
int c_alegria[] = {2, 2, 3, 0, 0, 3, 2, 1, 0, 0, 1, 2, 2, 1, 1, 2, 2, 3, 0, 0, 3, 2, 1, 0, 0, 1, 2, 1, 0, 0};
int n_alegria[] = {64, 64, 65, 67, 67, 65, 64, 62, 60, 60, 62, 64, 64, 62, 62, 64, 64, 65, 67, 67, 65, 64, 62, 60, 60, 62, 64, 62, 60, 60};
int t_alegria = 30;
```

Imagen 6. Canciones implementadas en el juego.

3.2. Configuración de los Pines y Periféricos

El proyecto utiliza una configuración avanzada de periféricos definidos en el archivo.ioc y generados por el HAL:

- GPIO Output: Pines configurados como salida digital para controlar la matriz de LEDs y los indicadores de vida.

- GPIO EXTI (Interrupciones): Los botones (PA1, PA2, PA3, PC5) no se leen por sondeo (polling) continuo, sino que están configurados en modo interrupción externa. Al pulsar un botón, el hardware detiene momentáneamente la CPU para ejecutar la función HAL_GPIO_EXTI_Callback, garantizando una respuesta inmediata.

- TIM4 (PWM): El Temporizador 4 (Canal 1 en PD12) se ha configurado en modo PWM Generation. Esto permite generar ondas cuadradas de frecuencia variable para sintetizar las notas musicales sin bloquear el procesador.

- ADC1 (Conversor Analógico-Digital): Configurado en modo conversión simple para leer el valor de tensión del potenciómetro y transformarlo en una variable de velocidad (retardo en milisegundos).

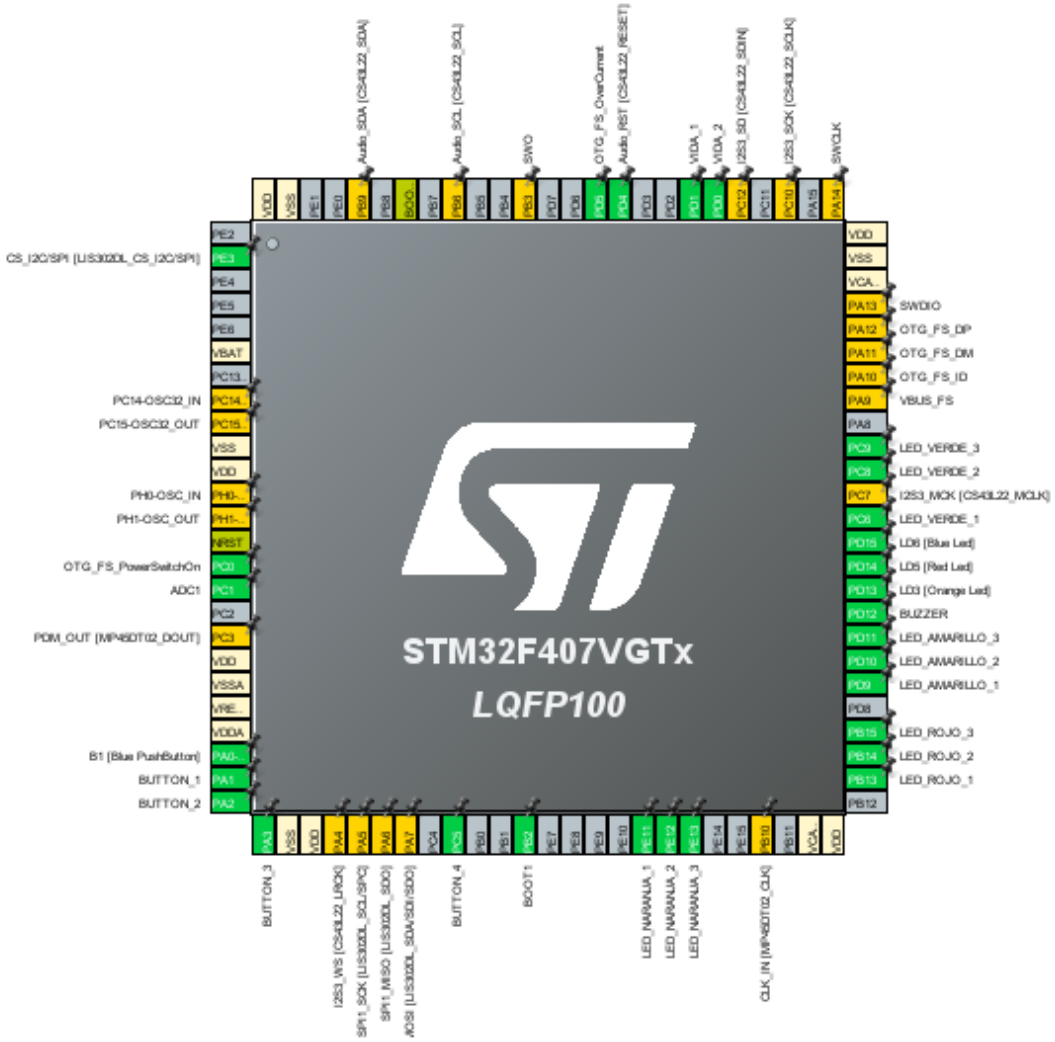


Imagen 7. Pinout view del proyecto.

3.3. Descripción del Algoritmo de Control

3.3.1. Síntesis de Audio Digital mediante Modulación (PWM)

Para la generación de las señales acústicas no se ha recurrido a métodos bloqueantes (como bucles de retardo delay), los cuales inhabilitarían la CPU. En su lugar, se ha configurado el Temporizador 4 (TIM4) en modo de Generación PWM (Pulse Width Modulation).

La frecuencia de la nota musical se determina modificando dinámicamente el registro de Auto-Recarga (ARR) del temporizador. Dado que el periodo de una señal es inversamente proporcional a su frecuencia (T = 1/f), hemos implementado un algoritmo de escalado inverso mediante la relación:

$$ARR = K\_esc / Nota\_val.$$

Donde:
ARR: Valor a escribir en el registro AutoReload Register (define el periodo de la onda).
K_esc: Constante de escalado empírica (300.000), calculada para desplazar el espectro de frecuencias hacia rangos graves, compensando la respuesta en frecuencia del buzzer piezoeléctrico pasivo.
Nota_val: Valor entero recuperado del array de la canción.

Adicionalmente, para mejorar la claridad del sonido y evitar la distorsión armónica típica de las ondas cuadradas puras, se ha ajustado el Ciclo de Trabajo (Duty Cycle) al 25%. Esto se logra configurando el registro de Comparación (CCR) como:

$$CCR = ARR / 4.$$

Este ajuste reduce la potencia eficaz de la señal, resultando en un timbre más nítido y menos estridente.

3.3.2. Algoritmo de Validación de Entrada e Integridad (Sistema Anti-Trampas)

Para garantizar la integridad de la partida y penalizar la pulsación indiscriminada de botones (técnica conocida como button mashing), se ha diseñado un algoritmo de validación por ventana de tiempo activa.

A diferencia de un sistema de interrupción simple que, solo valida aciertos, este algoritmo introduce una lógica de vigilancia continua dentro del bucle de juego:

- Monitorización en Tiempo Real: Durante la caída de la nota (fases de espera), el sistema verifica cada 10 ms si la variable global de interrupción (boton_pulsado_irq) ha cambiado de estado.
- Penalización Inmediata: Si el sistema detecta una pulsación fuera de la ventana de validación (cuando la nota aún no ha llegado a la meta) o en un carril incorrecto, se activa inmediatamente la subrutina de fallo.

Esto obliga al usuario a mantener una precisión temporal estricta. Cualquier interacción fuera de sincronía resulta en la pérdida instantánea de una vida, eliminando la posibilidad de "adivinar" la nota mediante pulsaciones aleatorias.

4. PROBLEMAS ENCONTRADOS Y SOLUCIONES ADOPTADAS

Durante el desarrollo del proyecto, nos enfrentamos a varios desafíos técnicos que requirieron soluciones específicas tanto de software como de adaptación al hardware disponible.

4.1. Limitaciones en la Calidad de Audio

Uno de los inconvenientes encontrados reside en la fidelidad del sonido. Debido a la falta de disponibilidad de un módulo de audio avanzado o un buzzer activo con etapa de amplificación, se optó por utilizar un buzzer pasivo piezoeléctrico estándar.

Este componente, al ser excitado directamente por una onda cuadrada (PWM) del microcontrolador, genera un sonido con muchos armónicos impares, resultando en un timbre metálico o "tipo 8-bits" y un volumen moderado.

Para mitigar esto y mejorar la experiencia de usuario, ajustamos vía software el ciclo de trabajo (duty cycle) del PWM al 25% (en lugar del 50% habitual) y aplicamos una fórmula matemática de división inversa en los temporizadores. Esto permitió suavizar ligeramente el tono y asegurar que las frecuencias musicales fueran matemáticamente precisas, logrando una melodía reconocible y funcional a pesar de las limitaciones físicas del componente.

4.2. Rebotes en los Pulsadores (Efecto Bouncing)

Al utilizar interrupciones para la lectura de los botones, detectamos que una única pulsación física generaba múltiples señales eléctricas espurias, provocando que el sistema interpretara varios "clics" seguidos.

Se implementó un filtro de desborde de software (Debouncing) dentro de la rutina de interrupción (Callback). El sistema ignora cualquier señal que ocurra en un intervalo menor a 200 ms tras la primera detección, garantizando así una lectura limpia y única por cada interacción del usuario.

4.3. Latencia en la Detección de Entrada (Bloqueo por Software)

Inicialmente, la gestión de la velocidad de caída de las notas se realizaba mediante la función HAL_Delay(). Detectamos que, durante ese tiempo de espera, el procesador quedaba "ciego" ante las entradas del usuario, provocando que pulsaciones rápidas no fueran registradas si ocurrían justo durante la transición de los LEDs.

Como solución se sustituyeron las esperas bloqueantes monolíticas por bucles fraccionados de "vigilancia activa". En lugar de detener el procesador durante 500ms seguidos, el sistema realiza iteraciones de 10ms comprobando en cada ciclo el estado de las banderas (flags) levantadas por las interrupciones. Esto permite mantener la temporización visual sin sacrificar la reactividad del sistema ante los eventos de los botones.

4.4. Fluctuaciones en la Lectura Analógica (Ruido ADC)

Al implementar el control de velocidad mediante el potenciómetro, observamos que el valor leído por el Conversor Analógico-Digital (ADC) presentaba pequeñas oscilaciones o "jitter" incluso cuando el componente estaba estático. Esto provocaba que la velocidad de las notas variase erráticamente en momentos puntuales.

Para solucionarlo se implementó un algoritmo de suavizado y escalado en el código. En lugar de usar el valor crudo del ADC (0-4095), se aplicó una división entera (lectura / 6) que actúa como un filtro de cuantización, eliminando el ruido de baja magnitud y estabilizando la variable de velocidad, garantizando una experiencia de juego fluida.

5. POSIBLES MEJORAS Y LÍNEAS FUTURAS

Aunque el sistema actual cumple con todos los requisitos funcionales establecidos, se han identificado varias líneas de desarrollo que permitirían escalar el proyecto hacia un producto más completo y comercial.

5.1. Implementación de Interfaz Visual Avanzada (LCD/OLED)

Actualmente, el feedback visual se limita a los LEDs de la placa. Una mejora significativa sería la integración de una pantalla LCD 16x2 o un display OLED controlado mediante protocolo I2C o SPI.

Esto permitiría mostrar un menú de selección de canciones con nombres reales, visualizar la puntuación en tiempo real y guardar un registro de "High Scores" (Récords) con las iniciales del jugador.

5.2. Mejora de la Calidad de Audio (DAC Y Amplificación)

Para superar las limitaciones del sonido PWM (onda cuadrada), se propone el uso del DAC (Conversor Digital-Analógico) interno del STM32 o un códec de audio externo vía protocolo I2S.

Esto permitiría reproducir muestras de audio real (archivos .wav o sintetizados) en lugar de simples tonos, logrando una experiencia musical polifónica y de alta fidelidad.

5.3. Persistencia de Datos (Memoria No Volátil)

En la versión actual, los récords se pierden al desconectar la alimentación.

Implementar la escritura en la memoria Flash interna del microcontrolador o añadir una memoria EEPROM externa. Esto permitiría guardar las mejores puntuaciones y la configuración de volumen/brillo entre sesiones.

5.4. Conectividad y Carga Dinámica (UART/Bluetooth)

Para evitar tener que reprogramar el microcontrolador cada vez que se quiera añadir una canción nueva.

Añadir un módulo Bluetooth (como el HC-05) conectado al puerto UART. De esta forma, se podría desarrollar una aplicación móvil sencilla que envíe los arrays de las nuevas canciones al STM32 de forma inalámbrica, almacenándolas dinámicamente en la RAM o Flash.

6. CONCLUSIONES

El desarrollo y finalización de este proyecto ha permitido consolidar de manera práctica los conocimientos teóricos adquiridos sobre la arquitectura de microcontroladores, específicamente sobre la familia STM32F4. Más allá de la simple implementación de un juego, el trabajo ha supuesto un ejercicio completo de diseño de sistemas embebidos.

Uno de los logros más significativos ha sido la transición de un paradigma de programación secuencial básica a uno orientado a eventos e interrupciones. La gestión de los pulsadores mediante EXTI (Interrupciones Externas) y el uso de Temporizadores (TIM4) para la síntesis de audio ha demostrado la importancia de liberar a la CPU de tareas bloqueantes, permitiendo que el sistema sea reactivo y eficiente en tiempo real.

Asimismo, el proyecto ha servido para comprender las limitaciones del mundo físico y cómo compensarlas mediante software. Desafíos como el ruido en la conversión analógica-digital (ADC) o los rebotes mecánicos en los pulsadores nos han obligado a implementar filtros digitales y algoritmos de validación robustos. Esto ha resultado en

un dispositivo fiable que no solo ejecuta instrucciones, sino que interpreta correctamente la intención del usuario.

Finalmente, cabe destacar la importancia de la arquitectura modular y el uso de Máquinas de Estados Finitos (FSM). Separar la lógica de control (main.c) de los datos (canciones.c) no solo ha facilitado la depuración del código durante el desarrollo, sino que deja el proyecto preparado para futuras ampliaciones sin necesidad de reestructurar el núcleo del programa. En definitiva, se ha conseguido un prototipo funcional, escalable y que cumple con rigor los estándares de ingeniería exigidos.

7. ENLACE AL REPOSITORIO GIT

Todo el código fuente, incluyendo el historial de versiones y la configuración del dispositivo, se encuentra disponible en el siguiente repositorio.

https://github.com/Sergius842/Trabajo-SED.git