Integración con ATmega328P de cuatro sistemas: control de plotter, selector de color (LDR–WS2812–SG90), piano electrónico y cerradura con teclado–LCD

1° Lucas Elizalde
Ingeniería en Mecatrónica
UTEC
Fray Bentos, Río Negro
lucas.elizalde@estudiantes.utec.edu.uy

2° Juan Manuel Ferreira
Ingeniería en Mecatrónica
UTEC
Fray Bentos, Río Negro
juan.ferreira.m@estudiantes.utec.edu.uy

3° Felipe Morrudo
Ingeniería en Mecatrónica
UTEC
Fray Bentos, Río Negro
felipe.morrudo@estudiantes.utec.edu.uy

***Resumen*—En este informe se presenta el diseño e implementación de cuatro sistemas mecatrónicos basados en ATmega328P: un control de plotter integrado a PLC con selección de figuras por UART y temporización ajustable, un selector de color con fotocelda, servomotor SG90 y matriz de LEDs WS2812, un piano electrónico con ocho pulsadores, doble buzzer y reproducción de melodías almacenadas, y una cerradura electrónica con teclado 4×4, pantalla LCD I2C y memoria EEPROM para contraseñas. Las pruebas en simulación y montaje físico demostraron un funcionamiento estable y validaron la capacidad del ATmega328P para integrar control, comunicación y generación de señales en distintos sistemas embebidos.**

***Index Terms*—ATmega328P, C, Piano Electrónico, Cerradura electrónica, EEPROM, Tira de LEDs, Fotocelda, Plotter.**

I. Introducción

El presente trabajo tiene como finalidad desarrollar y analizar distintos sistemas mecatrónicos basados en el microcontrolador ATmega328P, integrando conceptos de control digital, comunicación serial y memoria EEPROM. El estudio se centra en la construcción y programación de prototipos que permiten vincular los contenidos teóricos de la unidad curricular con aplicaciones prácticas en laboratorio, asegurando una visión integral del diseño, simulación y validación de sistemas embebidos

En una primera etapa, se abordará el control de un plotter monocromático mediante la integración del ATmega328P con un PLC. El sistema contempla funciones de seguridad (E-STOP, retorno a posición inicial y límites de área de trazado) y una interfaz por relés hacia las entradas del PLC para accionar los ejes de movimiento y el solenoide de marcado. La operación se realizará mediante un menú por UART que habilita la ejecución de figuras predeterminadas, respetando los tiempos de conmutación y las restricciones mecánicas.

Posteriormente, se implementará un sistema de percepción y visualización cromática compuesto por una fotocelda, una etapa de lectura por ADC, un servomotor y una tira WS2812. A partir de umbrales preestablecidos, el dispositivo identificará colores de referencia, posicionará el servomotor en función del color detectado y reproducirá el tono cromático en la tira LED, informando por comunicación serial el valor leído, el color identificado, el valor objetivo y la diferencia de lectura.

Finalmente, se desarrollarán aplicaciones embebidas orientadas a la interacción humano–máquina: un piano electrónico con ocho pulsadores y un buzzer controlado por PWM para la generación de notas y la reproducción de melodías seleccionables por UART; y una cerradura electrónica con teclado 4×4, pantalla LCD 16×2, indicadores luminosos y almacenamiento persistente en EEPROM, contemplando validación de contraseña, límite de intentos con señalización de alarma y funciones de cambio de clave desde el menú. Este conjunto de prácticas permitirá comprender la integración de sensores, actuadores y rutinas de control en tiempo real, así como los mecanismos de comunicación y seguridad propios de sistemas embebidos aplicados.

II. Objetivos

II-A. Objetivo General

- Desarrollar sistemas mecatrónicos basados en el microcontrolador ATmega328P que integren control digital, comunicación serial y procesamiento analógico–digital para la automatización de tareas, detección de colores, generación de sonidos y control de acceso.

II-B. Objetivos Específicos

- Diseñar un sistema de control que permita la ejecución automatizada de movimientos y trazados mediante la interacción entre el microcontrolador y un PLC, considerando la seguridad operativa y los límites mecánicos.
- Implementar un sistema de detección cromática que identifique colores mediante una fotocelda, posicione un servomotor y reproduzca el color detectado en una tira de LEDs WS2812, informando los valores por comunicación serial.
- Programar un instrumento musical electrónico capaz de generar notas mediante pulsadores y reproducir melodías

predefinidas utilizando un buzzer controlado por PWM y comandos UART.
- Configurar un sistema de acceso electrónico que gestione contraseñas a través de un teclado matricial, muestre información en una pantalla LCD y utilice memoria EEPROM para el almacenamiento persistente de datos.

III. Materiales

- ATmega328P
- Fuente DC.
- Pantalla LCD 16x2
- LEDs
- Buzzer activo
- Buzzer pasivo
- Multímetro digital.
- FotoCelda
- Servomotor
- Protoboard.
- Matriz de LEDs RGB 8x8
- Pulsadores
- Resistencias

IV. Marco Teórico

IV-A. Lenguaje C

El lenguaje C es un lenguaje de programación de propósito general, estructurado y de nivel medio, ampliamente utilizado en el desarrollo de sistemas embebidos. Su sintaxis combina la eficiencia del lenguaje ensamblador con la legibilidad y modularidad de los lenguajes de alto nivel, lo que lo convierte en una herramienta ideal para programar microcontroladores como el ATmega328P. A través de C, es posible manipular directamente los registros y puertos del hardware, definir rutinas de interrupción y controlar periféricos, manteniendo al mismo tiempo una organización clara del código (MSMK, 2025).

En el contexto de los sistemas mecatrónicos, el uso del lenguaje C permite desarrollar programas que gestionan sensores, actuadores y comunicaciones seriales con mayor rapidez y portabilidad. Además, su integración con compiladores como *avr-gcc* posibilita traducir el código fuente a instrucciones máquina optimizadas, aprovechando al máximo los recursos del microcontrolador. Este enfoque facilita la implementación de sistemas de control, monitoreo y automatización en tiempo real, asegurando precisión, eficiencia y mantenimiento sencillo (MSMK, 2025).

IV-B. Pantalla LCD

La pantalla LCD (Liquid Crystal Display) es un dispositivo electrónico que utiliza cristales líquidos para representar información visual mediante el control de la luz polarizada. Su funcionamiento se basa en la orientación de las moléculas de cristal líquido entre dos filtros polarizadores, lo que permite bloquear o permitir el paso de la luz según el voltaje aplicado. Este principio hace posible la formación de caracteres y símbolos en la pantalla, con bajo consumo energético y buena visibilidad. En aplicaciones con microcontroladores como el ATmega328P, los módulos LCD, especialmente los de formato 16×2, se emplean para mostrar datos, mensajes de estado y resultados de procesos, facilitando la interacción entre el sistema y el usuario mediante una interfaz visual simple y eficiente (Del Valle Hernández, 2021). A continuación, en Fig. 1 se muestra una figura del mismo.

Fig. 1: Pantalla LCD

IV-C. Comunicación I2C

El protocolo I2C (Inter-Integrated Circuit) es un estándar de comunicación serial síncrono que permite el intercambio de datos entre múltiples dispositivos digitales utilizando únicamente dos líneas: SDA (Serial Data) y SCL (Serial Clock). Su arquitectura maestro–esclavo posibilita conectar hasta 127 dispositivos en un mismo bus, manteniendo una comunicación eficiente con velocidades de hasta 1 Mbit/s. Este protocolo resulta especialmente útil en sistemas embebidos, como los basados en el microcontrolador ATmega328P, debido a su simplicidad de conexión, bajo número de pines y capacidad para gestionar múltiples periféricos simultáneamente. Además, I2C incorpora mecanismos de confirmación de datos y direccionamiento interno, garantizando transmisiones seguras y organizadas, lo que lo convierte en una opción ideal para la comunicación con sensores, memorias y módulos de visualización LCD con adaptador I2C (Administrador, 2025).

IV-D. Teclado Matricial

El teclado matricial es un dispositivo de entrada que permite detectar múltiples pulsaciones utilizando un número reducido de pines del microcontrolador mediante una disposición en filas y columnas como semuestra en Fig. 2. Cada tecla del teclado conecta un punto único entre una fila y una columna, cerrando el circuito solo cuando se presiona. Este principio permite escanear el estado de las teclas activando secuencialmente las filas y leyendo las columnas para identificar qué tecla ha sido presionada. En un teclado 4×4, por ejemplo, se requieren únicamente 8 pines en lugar de 16, lo que optimiza el uso de recursos del microcontrolador. Este tipo de interfaz es ampliamente utilizado en sistemas embebidos como los basados en el ATmega328P, en aplicaciones que requieren ingreso de datos o contraseñas, como cerraduras electrónicas o sistemas de control, brindando una solución práctica, económica y escalable (Teclados Matriciales – Prometec, s.f.).

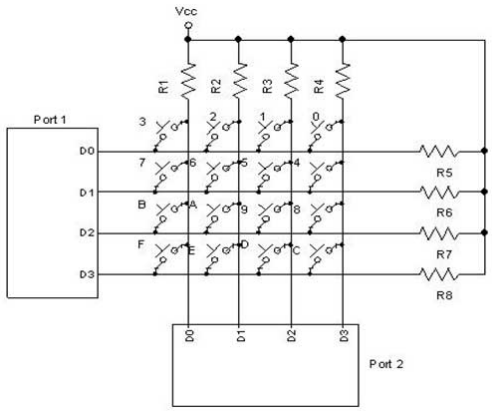

Fig. 2: Teclado Matricial

IV-E. Buzzer

El zumbador, también conocido como buzzer, es un dispositivo electrónico que convierte una señal eléctrica en sonido mediante el efecto piezoeléctrico. Este fenómeno se produce cuando ciertos materiales, al ser sometidos a una diferencia de potencial, se deforman y generan vibraciones mecánicas capaces de producir ondas sonoras. En el ámbito de la electrónica y los sistemas embebidos, los buzzers se utilizan frecuentemente como elementos de aviso o señalización acústica. Existen dos tipos principales: los zumbadores activos, que incluyen un oscilador interno y emiten sonido al ser alimentados directamente, y los pasivos, que requieren una señal externa de tipo cuadrada, generalmente generada mediante modulación por ancho de pulso (PWM), en Fig.3 se observa cada uno. En aplicaciones con el microcontrolador ATmega328P, los buzzers pasivos permiten generar diferentes tonos controlando la frecuencia de la señal, lo que los hace ideales para proyectos educativos, alarmas, sistemas de alerta o instrumentos musicales electrónicos (Del Valle Hernández, 2022).

Fig. 3: Buzzers

IV-F. Memoria EEPROM

La memoria EEPROM (Electrically Erasable Programmable Read-Only Memory) es un tipo de memoria no volátil incorporada en los microcontroladores, que permite almacenar datos de forma permanente incluso cuando el sistema se apaga o se reinicia. A diferencia de la memoria RAM, cuyo contenido se borra al interrumpirse la alimentación, la EEPROM conserva los valores escritos hasta que son modificados o eliminados manualmente. En el caso del microcontrolador ATmega328P, utilizado en placas como el Arduino UNO, esta memoria posee una capacidad de 1024 bytes, donde cada posición puede almacenar un byte (un valor entre 0 y 255). Gracias a la librería *EEPROM.h*, es posible leer, escribir y borrar información fácilmente, lo que la convierte en una herramienta esencial para guardar configuraciones, contraseñas o estados del sistema que deben mantenerse tras un reinicio. Aunque su vida útil es limitada a aproximadamente 100.000 ciclos de escritura por posición, esta cifra resulta suficiente para la mayoría de aplicaciones en sistemas embebidos, como cerraduras electrónicas, controladores de iluminación o dispositivos configurables (Del Valle, 2022).

IV-G. Algoritmo de Bresenham

El algoritmo de Bresenham es una técnica de dibujo digital que permite representar figuras geométricas en una pantalla o matriz de puntos utilizando únicamente operaciones con enteros, lo que lo hace muy eficiente en sistemas de recursos limitados como los microcontroladores. Su principio consiste en calcular, de manera incremental, qué puntos de la retícula deben encenderse para aproximar la forma ideal. Aunque fue desarrollado inicialmente para el trazado de líneas rectas, también se adapta al dibujo de circunferencias y elipses, generando sus puntos a partir de la simetría de la figura y evitando cálculos complejos de raíz cuadrada o números en coma flotante (Hurtado, s.f.).

IV-H. Matriz de LEDs RGB 8x8

Este módulo es un panel cuadrado con una interfaz XH2.54 de 3 pines. Este módulo de matriz LED a todo color RGB de 8x8 se basa en LED de control inteligentes WS2812. Cada LED se puede direccionar de forma independiente con píxeles RGB que pueden alcanzar 256 niveles de brillo DFRobot (s. f.).

IV-I. Fotocelda

Una LDR o resistencia dependiente de la luz también conocida como fotoresistencia, fotocélula, o fotoconductor, es un tipo de resistencia cuya resistencia varía dependiendo de la cantidad de luz que cae sobre su superficie. Cuando la luz cae sobre la resistencia, entonces la resistencia cambia. Por lo tanto, son dispositivos sensibles a la luz (TecnoSalva, 2024)

IV-J. Microservo SG90

Pequeño y ligero, con alta potencia de salida. El servo puede girar aproximadamente 180 grados (90° en cada dirección) y funciona igual que los servos estándar, pero más pequeño. Puedes usar cualquier código, hardware o biblioteca para controlarlo Servo Motor SG90 Data Sheet. (s. f.).

IV-K. MIDI (Interfaz Digital de Instrumentos Musicales)

El protocolo **MIDI** (Interfaz Digital de Instrumentos Musicales) es un estándar utilizado para la transmisión de información musical entre dispositivos electrónicos, como teclados, computadoras y otros instrumentos musicales. A diferencia de los formatos de audio tradicionales, el MIDI no transmite sonidos,

sino que envía datos relacionados con las acciones musicales, como qué notas se tocan, su duración, la velocidad con la que se tocan, y otros parámetros de interpretación.

Este protocolo permite una representación más compacta y flexible de la música, ya que se enfoca en los eventos musicales y no en las grabaciones de audio. MIDI también facilita la edición y modificación de la música, lo que lo convierte en una herramienta poderosa en aplicaciones musicales y en la industria de la música digital. Los dispositivos que soportan MIDI pueden comunicarse entre sí, lo que abre un amplio rango de posibilidades para el control y la producción musical en tiempo real.

En términos de implementación en sistemas embebidos, como en el caso de los microcontroladores, MIDI es utilizado para controlar instrumentos y otros dispositivos mediante comandos de bajo nivel que pueden ser procesados por el hardware. Gracias a su eficiencia y versatilidad, MIDI se emplea comúnmente en dispositivos musicales, sintetizadores y secuenciadores, así como en sistemas que requieren la integración de múltiples fuentes musicales o de audio.(Joan, 2024)

V. Procedimiento

V-A. *Control del Plotter con ATmega328P*

En este laboratorio se controló el trazador gráfico con un microcontrolador ATmega328P. Como punto de partida, se retomaron *exclusivamente* las tres figuras básicas del trabajo anterior (triángulo, cruz y círculo), originalmente programadas en ensamblador, y se tradujeron a `C` para mejorar legibilidad, facilitar ajustes de temporización y unificar el flujo de compilación sobre el ATmega328P.

La interacción con la máquina se realizó gobernando el PLC del plotter mediante etapas de potencia (relés/MOSFETs) que adaptan niveles y corriente, evitando cargar directamente el microcontrolador. El PLC acciona los desplazamientos en los ejes y la válvula neumática del útil de trazo (subida/bajada del solenoide). La asignación de señales de salida del ATmega328P hacia las entradas del PLC se resume en el Cuadro I.

Cuadro I: Asignación de pines del ATmega328P hacia el PLC del plotter

Pin digital	Conexión PLC
D2	Bajar solenoide → X0
D3	Subir solenoide → X1
D4	Movimiento hacia abajo → X5
D5	Movimiento hacia arriba → X6
D6	Movimiento hacia la izquierda → X7
D7	Movimiento hacia la derecha → X10

La traducción de las figuras básicas a `C` se materializó como secuencias de escritura sobre `PORTD` del tipo $PORTD = (1'PDx)$ para provocar cada paso en la dirección deseada y conmutar la herramienta. La temporización se normalizó con la macro `AUTOGEN_DELAY()` (escalada por `AUTOGEN_SCALE_MS_PER_S`) para garantizar un tiempo mínimo reproducible por el conjunto PLC–actuadores sin pérdida de pasos. Para minimizar uso de RAM, las tablas de movimiento se almacenaron en `PROGMEM` mediante un *struct* compacto `Patron{portd_value, delay_units}` que, además, permite codificar repeticiones (*run-length*) de un mismo pin con N *ticks*. Durante el trazado se deshabilitó la UART (`ejecutar_con_uart_desactivado()`) para evitar *jitter* en los tiempos y se reactivó al finalizar cada figura.

Para las figuras complejas (por ejemplo, *Rana* y *Manzana*) se desarrolló adicionalmente una herramienta propia con interfaz gráfica empaquetada como ejecutable, `PlotterSuite.exe`, que integra el flujo **PNG → patrón "curly" → simulación**. El módulo de conversión toma una imagen binaria (contorno negro sobre fondo blanco), aplica preprocesado (suavizado, cierre morfológico, simplificación y remuestreo de contornos) y genera una LUT compacta de pares {`PORTD`, `ticks`} lista para `PROGMEM`. El módulo de simulación previsualiza el trazado (viajes con lapicera arriba, segmentos con lapicera abajo) y permite ajustar parámetros antes de llevarlo al equipo real. Las siluetas binarias utilizadas fueron elaboradas por los autores en Adobe Photoshop (contorno negro sobre fondo blanco).

La selección de figuras se realizó por UART mediante un menú simple, mapeando cada comando a su rutina de dibujo. El Cuadro II resume los comandos implementados.

Cuadro II: Comandos UART para selección de figura

Comando	Acción
1	Triángulo
2	Círculo
3	Cruz
4	Manzana
5	Rana

Con fines ilustrativos, la Fig. 4 muestra una silueta binaria típica utilizada como entrada del conversor, y la Fig. 5 la correspondiente previsualización generada por el ejecutable a partir de la LUT obtenida. El resto de las siluetas y sus previsualizaciones se incluyen en Anexos para no sobrecargar esta sección.

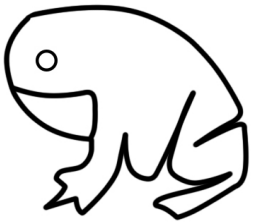

Fig. 4: Silueta binaria (negro sobre blanco) creada por los autores en Photoshop; entrada del conversor en `PlotterSuite.exe`.

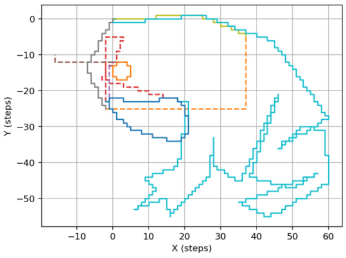

Fig. 5: Previsualización del trazado generada por `PlotterSuite.exe` a partir de la LUT.

Finalmente, el procedimiento de validación consistió en: (i) generar la LUT desde la imagen objetivo en el ejecutable, (ii) verificar el trazado con la previsualización para ajustar `AUTOGEN_SCALE_MS_PER_S` y descartar saltos no deseados, y (iii) ejecutar la figura en el equipo real con la UART deshabilitada durante el trazo. Este flujo aseguró que el dibujo físico coincidiera con la previsualización dentro de las tolerancias mecánicas del sistema.

En el enlace a Google Drive que se encuentra en el Anexo, hay un video que muestran el uso del ejecutable: (i) conversión *PNG* $\rightarrow$ *patrón* y exportación de la LUT a `PROGMEM`, y (ii) previsualización/simulación del trazado (viajes con lapicera arriba y segmentos con lapicera abajo), útil para ajustar `AUTOGEN_SCALE_MS_PER_S` antes de dibujar en el plotter real.

V-B. Sistema de Selección de Colores con ATmega328P, Fotocelda, Matriz de LEDs y Servomotor

En este apartado, se debe detectar el color de una hoja con un circulo de colores, y el color detectado debe ser reflejado con un microservo SG90 colocado en el centro de una estrella de colores, apuntando hacia el color detectado. También debe ser reflejada encendiendo el color correspondiente en una tira LED WS2812b.

Para detectar el color, se va a utilizar una fotocelda, dicha fotocelda va a ser encerrada junto con un LED blanco, esto con la finalidad de mantener una iluminación constante y uniforme, para no depender de la iluminación del entorno donde se utiliza el detector.

Se implementa un divisor de voltaje después de la fotocelda, utilizando una resistencia de 10 k, con el fin de generar una señal de voltaje que dependa de la intensidad de luz que la fotocelda recibe. En este divisor de voltaje, el valor de la resistencia cambia proporcionalmente según la cantidad de luz incidente sobre la fotocelda. Esto significa que la fotocelda cambia su resistencia dependiendo de la luz que recibe, generando un voltaje variable en el punto de unión entre la fotocelda y la resistencia de 10 k.

Estos niveles de voltaje se leen a través de un pin ADC (convertidor analógico a digital) en el microcontrolador. El ADC convierte este voltaje analógico en valores digitales que representan distintos niveles de luz. Dependiendo del valor leído por el ADC, el microcontrolador va a asignarle a la tira LED el color correspondiente al detectado, y va a orientar el servomotor hacia el color.

Antes de desarrollar el código principal que implementa la lógica para resolver la problemática, se implementó un código preliminar para determinar el rango de valores del ADC correspondientes a los diferentes colores seleccionados. Este proceso de promediado consiste en acumular una serie de valores de ADC para cada color y luego dividir la suma total por el número de lecturas realizadas. De esta manera, se obtiene un rango más preciso y consistente para cada color, lo que mejora la exactitud del sistema al identificar colores.

Debido al que el microservo SG90 solo tiene un rango de 180°, se deben seleccionar 4 de los 6 colores pertenecientes a la estrella, de los cuales se opto rosa, rojo, verde y amarillo. A continuación, en la Fig.6 se observa el circulo de colores junto a la estrella de colores.

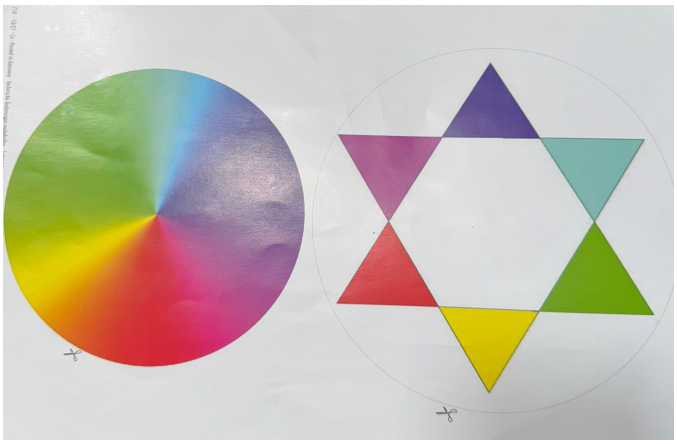

Fig. 6: Hoja con circulo y estrella de colores

El selector también debe contar con comunicación serial, donde el usuario puede observar el color detectado, el valor de la fotocelda, valor del color establecido y el valor de la diferencia entre el valor establecido y el de lectura.

Una vez se comprendió lo necesario para abordar la problematica, se comenzo a desarrollar el código principal y posteriormente simular en el software PROTEUS. Este software no cuenta con una tira de LEDs WS2812b, por lo que en su defecto se uso un LED RGB. En el CuadroIII se observa las conexiones de los componentes en la simulación.

Cuadro III: Conexión del Selector de colores en simulación

Selector de color	Pin ATmega328P
Fotocelda	PC0
Microservo SG90	PB1
Pin R del RGB	PD6
Pin G del RGB	PD5
Pin B del RGB	PB3
Terminal Virtual	PD1 (TXD)

Después de finalizar la simulación en PROTEUS, se procedió a montar el circuito en físico, con la diferencia de que se intercambió el LED RGB por una tira LED WS2812b. Modificando también los valores ADC, ya que en simulación el valor de la fotocelda se modifica manualmente, y en físico este valor depende de factores externos. A continuación, se observa

V-C. Piano Electrónico con ATmega328P

El piano electrónico fue implementado utilizando el microcontrolador ATmega328P. El sistema cuenta con 8 pulsadores, cada uno asociado a una nota musical, y un buzzer para emitir las notas correspondientes. Para mejorar la calidad del sonido y permitir la reproducción de dos pistas simultáneamente, se optó por añadir un segundo buzzer. De este modo, cada buzzer

puede emitir una pista diferente, logrando un sonido más claro y con mayor calidad, lo que hace que la experiencia musical sea mucho más agradable.

Además de las 8 notas originales, se agregaron notas adicionales a las canciones **C1: "Lose Yourself"de Eminem** y **C2: "Spyder Dance"de Undertale** para enriquecer la composición y mejorar la calidad del sonido de las canciones, dando más posibilidades de variación tonal en la interpretación.

El primer paso consistió en el diseño del circuito en el software de simulación Proteus. En este diseño, se conectaron los pulsadores a los pines de entrada del microcontrolador ATmega328P (BTN0 a BTN7), mientras que los buzzers se conectaron a dos pines de salida diferentes (PB1 y PB3) para cada uno. Al presionar un pulsador, el sistema debería emitir la frecuencia correspondiente a la nota musical asociada.

La asignación de las notas musicales a los pulsadores se muestra en el Cuadro IV.

Cuadro IV: Correspondencia entre los pulsadores y las frecuencias de las notas musicales

Pulsador	**Frecuencia (Hz)**
BTN0 (Do)	262
BTN1 (Re)	294
BTN2 (Mi)	330
BTN3 (Fa)	349
BTN4 (Sol)	392
BTN5 (La)	440
BTN6 (Si)	494
BTN7 (Do Agudo)	523

Cada pulsador está asignado a una nota musical específica, con las frecuencias correspondientes a cada una. Además, se configuró el microcontrolador para que las notas fueran emitidas utilizando una señal PWM controlada por los timers disponibles, en particular, el Timer1 para el primer buzzer (B1) y el Timer2 para el segundo buzzer (B2).

El diseño del sistema también incluye la capacidad de seleccionar entre dos canciones predefinidas, a saber: **C1: "Lose Yourself"de Eminem** y **C2: "Spyder Dance"de Undertale**. Estas canciones fueron extraídas en formato MIDI y luego convertidas a un formato compatible con el microcontrolador. Para esto, se utilizó la página web *MIDI to Arduino*, que permite convertir archivos MIDI en código Arduino, dicha página se encuentra en la bibliografía.

A partir de este código, se adaptó el código básico para el lenguaje C utilizado en Microchip Studio, almacenando las canciones en una especie de Look-Up Table (LUT). Cada pista fue almacenada y se activaba mediante la selección de la canción a través de UART.

Además de las notas originales de las canciones, se agregaron notas extras a las pistas de **C1** y **C2** para enriquecer la música, creando un sonido más complejo y agradable. Estas notas adicionales fueron cuidadosamente seleccionadas y agregadas para mejorar la calidad sonora general.

Una vez diseñado el circuito y validado su funcionamiento en la simulación, se pasó a la implementación física del sistema en protoboard. Se utilizaron los mismos componentes de la simulación, conectando los pulsadores, los buzzers y el microcontrolador ATmega328P de acuerdo al diseño establecido.

El siguiente paso fue la integración de las funciones de UART para permitir la selección de las canciones predefinidas, que pueden ser activadas mediante los comandos 'C1' y 'C2'. Esta comunicación serial también permite detener la reproducción de las canciones mediante el comando 'S' y volver al modo de piano.

Finalmente, se verificó que el sistema estuviera funcionando correctamente tanto en la simulación como en el montaje físico.

V-D. Cerradura Electrónica con Teclado y LCD

El objetivo de este apartado es programar el microcontrolador ATmega328P para implementar una cerradura electrónica que permita el ingreso y verificación de una contraseña numérica por medio de un teclado matricial 4×4, mostrando mensajes y estados en una pantalla LCD 16×2 con conexión I2C. La contraseña deberá tener una longitud configurable (entre 4 y 6 dígitos) y mantenerse persistida en la EEPROM del microcontrolador, de modo que se conserve tras reinicios o cortes de alimentación.

La interfaz de usuario deberá incluir un mensaje de bienvenida al encender el sistema, seguido de un menú interactivo que permita seleccionar entre las siguientes opciones: (i) ingresar la contraseña para desbloquear, (ii) cambiar la contraseña almacenada. Durante el ingreso de la clave, la LCD deberá proporcionar retroalimentación visual para cada dígito ingresado y mensajes de confirmación o error según corresponda.

El sistema debe contemplar mecanismos de seguridad: se establecerá un límite de intentos fallidos (tres intentos consecutivos). Al superarse dicho límite, la cerradura activará una alarma (mediante buzzer) de estado de error, manteniendo un bloqueo para siempre. Asimismo, se deberá señalizar el estado de éxito con un LED de confirmación y mensajes en la LCD que informen claramente la condición del sistema (acceso concedido, acceso denegado, bloqueo activo, etc.).

Para la gestión de la contraseña, el sistema deberá incluir un menú que solicite en primer lugar la validación de la clave actual, garantizando que solo el usuario autorizado pueda realizar cambios. Una vez verificada, se permitirá el ingreso de una nueva contraseña, cuya longitud deberá encontrarse dentro del rango establecido (entre 4 y 6 dígitos). Posteriormente, se pedirá una confirmación de la nueva clave antes de almacenarla de forma permanente en la memoria EEPROM del microcontrolador.

En caso de que la EEPROM no contenga una contraseña guarda por el usuario al inicio del sistema, este se inicializa automáticamente con una clave predeterminada, configurada por defecto como "0000".

Una vez comprendido el funcionamiento del sistema, se procedió a realizar la programación mientras, de manera simul-

tánea, se efectuaba la simulación del circuito en el software PICSimLab. Para organizar el desarrollo, la implementación se dividió en tres etapas principales:

En primer lugar, se realizó la inicialización de la pantalla LCD mediante la comunicación I2C, asegurando su correcto funcionamiento y la posibilidad de mostrar texto y permitir la escritura en pantalla mediante el teclado matricial.

En segundo lugar, se implementó el menú principal, junto con las rutinas para la lectura e ingreso de la contraseña desde el teclado, permitiendo la interacción del usuario con el sistema.

Finalmente, se llevó a cabo la integración de la memoria EEPROM, con el propósito de almacenar y recuperar la contraseña, además de la programación de los LEDs y del buzzer activo, utilizados para señalar los distintos estados del sistema, como acceso correcto, error o activación de la alarma.

Para la programación y simulación se utilizó la siguiente distribución de pines entre el circuito y el microcontrolador ATmega328P. En el Cuadro V se muestra la conexión del teclado matricial, mientras que en el Cuadro VI se detallan las conexiones correspondientes a la pantalla LCD, los LEDs indicadores y el buzzer activo.

Cuadro V: Conexión de Teclado Matricial al ATmega328P

Teclado Matricial	Pin ATmega328P
Fila 1 (R1)	D0 (PD0)
Fila 2 (R2)	D1 (PD1)
Fila 3 (R3)	D2 (PD2)
Fila 4 (R4)	D3 (PD3)
Columna 1 (C1)	D8 (PB0)
Columna 2 (C2)	D9 (PB1)
Columna 3 (C3)	D10 (PB2)
Columna 4 (C4)	D11 (PB3)

Cuadro VI: Conexión de otros componentes al ATmega328P

Componente	Pin ATmega328P
Pantalla LCD (SDA)	A4 (PC4)
Pantalla LCD (SCL)	A5 (PC5)
LED Verde	A0 (PC0)
LED Rojo	A1 (PC1)
Buzzer Activo	A2 (PC2)

Una vez concluida la simulación en el software, se implementó el circuito en físico utilizando la misma distribución de pines definida en la etapa de prueba.

VI. Resultados

VI-A. Control del Plotter con ATmega328P

El sistema fue validado primero en simulación y luego en el equipo real. En la simulación, el ejecutable con GUI `PlotterSuite.exe` cargó las LUT generadas por su módulo de conversión (a partir de siluetas negro sobre blanco preparadas en Photoshop), permitiendo verificar continuidad de trazos, distinguir viajes con lapicera arriba (PD3) de trazos con lapicera abajo (PD2) y ajustar la constante `AUTOGEN_SCALE_MS_PER_S` hasta lograr un avance estable sin pérdida de pasos ni sobretrazos en esquinas. En la Fig. 7 se muestra un ejemplo de la previsualización.

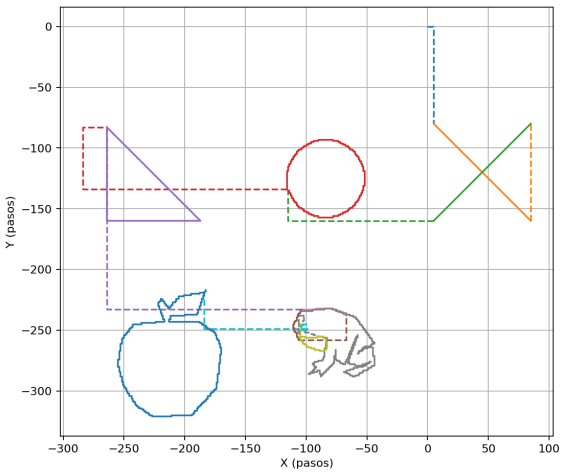

Fig. 7: Previsualización del trazado generada por `PlotterSuite.exe`.

En el montaje físico, la selección por UART ejecutó las figuras sin *jitter* gracias a la desactivación temporal del puerto serie durante el dibujo. Las figuras básicas (cruz, triángulo y círculo) reprodujeron la previsualización: la cruz y el triángulo presentaron vértices definidos, y el círculo mostró una aproximación uniforme con pasos cardinales coherentes. Para las figuras complejas (*Manzana* y *Rana*) las LUT en `PROGMEM` mantuvieron el binario dentro de los límites de `FLASH` y se observó un contorno continuo y fiel a la silueta origen. En la Fig. 8 se aprecia el trazado físico.

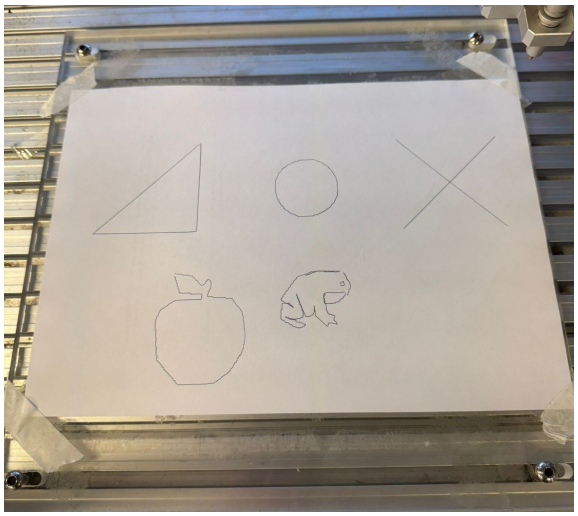

Fig. 8: Resultado físico del trazado en el plotter.

Durante las pruebas repetidas, la superposición de pasadas fue consistente y no se evidenciaron pérdidas de paso. Ajustes menores en los tiempos asociados a PD2/PD3 (subir/bajar

útil) mejoraron la nitidez en cambios de dirección muy cortos. Asimismo, siluetas con bordes limpios (negro pleno sobre blanco) y parámetros de conversión moderados (suavizado y simplificación) redujeron segmentos redundantes y suavizaron "esquinas duras".

VI-A1. Estructura de la LUT y mapeo de pines.: Cada paso del plotter se codifica como un par {`PORTD`, `delay`} almacenado en la estructura:

Listing 1: Estructura de la LUT y mapeo de pines.

```
typedef struct { uint8_t portd_value; uint8_t
    delay_units; } Patron;
```

Un elemento típico `{ (1<<PD4), 2 }` significa "activar PD4 (mover en Y) durante 2 unidades de tiempo". El mapeo usado es: PD7=+X (derecha), PD6=X (izquierda), PD5=+Y (arriba), PD4=Y (abajo), PD2=*pen down* (baja útil), PD3=*pen up* (sube útil). Las diagonales se logran *oreando* bits (p.ej., `(1<<PD4)`(1PD7)| para abajo–derecha).

VI-A2. Temporización con única constante de escala.: La duración efectiva de `delay_units` se obtiene con:

Listing 2: Temporización con única constante de escala.

```
defineAUTOGEN_SCALE_MS_PER_S 50
defineAUTOGEN_DELAY(s) delay_ms_u16((uint16_t)((s) *
     AUTOGEN_SCALE_MS_PER_S))
```

De este modo, el mismo patrón puede acelerarse o desacelerarse ajustando sólo `AUTOGEN_SCALE_MS_PER_S` sin regenerar la LUT.

VI-A3. LUTs por figura y ejecución en `PROGMEM`.: Las trayectorias se almacenan en `PROGMEM` para ahorrar `SRAM`. El procedimiento genérico de ejecución recorre la LUT y aplica cada par:

Listing 3: LUTs por figura y ejecución en `PROGMEM`.

```
static void ejecutar_patron_autogen(const Patron *
    seq, uint16_t len){
  for (uint16_t i=0;i<len;i++){
    uint8_t p  = pgm_read_byte(&seq[i].portd_value);
    uint8_t du = pgm_read_byte(&seq[i].delay_units);
    PORTD = p; AUTOGEN_DELAY(du);
  }
}
```

Se prepararon LUTs específicas: `cruz_seq[]` (posicionamiento + dos diagonales), `triangulo_seq[]` (catetos y cierre), `circulo_seq[]` (aproximación por pasos cardinales cortos) y patrones extensos derivados por conversión automática para *Manzana* y *Rana*. Estas últimas reutilizan el mismo ejecutor, garantizando que lo simulado sea lo ejecutado en el hardware.

VI-A4. Menú UART y ejecución sin interferencias.: El interfaz serie a 9600baud presenta un menú (1–5) y ejecuta la figura elegida. Para evitar *jitter* o interrupciones durante el trazo, la rutina de dibujo se encapsula con la UART deshabilitada temporalmente:

Listing 4: Menú UART y ejecución sin interferencias.

```
void ejecutar_con_uart_desactivado(FuncionDibujo fn)
    {
  _delay_ms(200);    // drenar TX
  UCSR0B = 0; UCSR0A = 0;  // UART off
  fn();              // trazo
  uart_init();       // UART on
}
```

En `main()`, tras leer el carácter con `uart_getc()`, un `switch` invoca `Triangulo()`, `Circulo()`, `Cruz()`, `Manzana()` o `Rana()` usando el envoltorio anterior.

Evidencia adicional (comparativas simulación/físico de todas las figuras) y videos de uso del ejecutable (*conversión y previsualización*) se incluyen en el Anexo (carpeta Drive).

VI-B. Sistema de Selección de Colores con ATmega328P, Fotocelda, Matriz de LEDs y Servomotor

Se implemento correctamente la solución a la problemática, tanto como de manera simulada y física. En la simulación mediante el software PROTEUS, se valido que el servomotor y el LED RGB respondan correctamente ante los valores ADC recibidos por las diferentes intensidades lumínicas de la foto celda, correspondientes al grupo de colores. A continuación, en el Cuadro VII se observan el rango de valores ADC resultante para cada color.

Cuadro VII: Rango ADC y valor RGB correspondiente a cada color

Color	Rango ADC	Valor RGB
Rosa	128 - 172	255, 0, 80
Rojo	210 - 244	255, 0, 0
Amarillo	274 - 301	255, 255,0
Verde	326 - 349	0, 255, 0

En cuanto al microservo, se ajustaron ángulos para cada color. A continuación, en el CuadroVIII se observan los valores con sus correspondientes colores.

Cuadro VIII: Ángulos del microservo según color establecido

Selector de color	Ángulo del microservo
Rosa	0°
Rojo	60°
Amarillo	120°
Verde	180°

También se implemento la visibilidad de los colores detectados vía monitor serial, donde ademas se pueden apreciar datos mas específicos, como el valor de la fotocelda, el punto medio del rango ADC del color detectado, y la diferencia del valor ADC exacto que esta siendo detectado respecto al punto medio. A continuación, en la Fig.9 se observa la simulación del circuito completo.

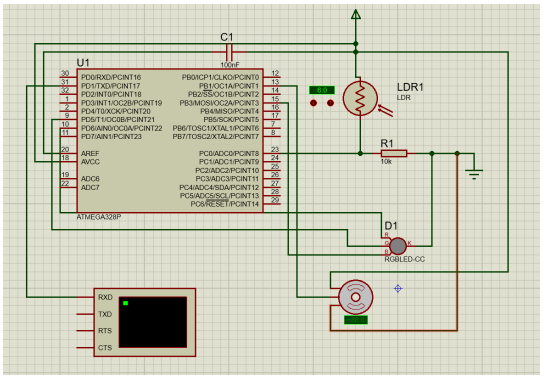

Fig. 9: Simulación del Selector de Colores en software PROTEUS

Una vez concluida la simulación, se paso a montar el circuito en físico, donde se debió modificar los componentes a utilizar, precisamente el LED RGB por una matriz 8x8 de LEDS RGB, esta tuvo que ser alimentada con una fuente a 5V ya que no bastaba la alimentación del Arduino. Como cobertura de la fotocelda para protegerla de la luz del entorno, se utilizo un recorte del interior de un rollo de papel higiénico, también se colocó junto a ella dos LEDs blancos, con el fin de una iluminación precisa y constante en todo momento. A continuación, en la Fig.10 se observa el circuito montado en físico.

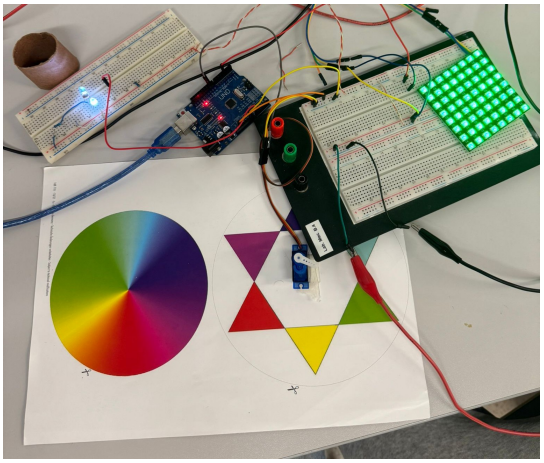

Fig. 10: Montaje en físico del Selector de Colores

Al no ser constantes los valores ADC como en la simulación, debido que no es un entorno ideal, se tuvo que hacer un promedio para el rango ADC de cada color, tomando 20 mediciones en cada caso, esto con el fin de tener valores mas precisos. A continuación, en el CuadroIX se observa los rangos ADC resultantes para cada color junto con el valor RGB para la matriz de LEDs en la implementación física.

Cuadro IX: Rango ADC y valor RGB correspondiente a cada color

Color	Rango ADC	Valor RGB
Rosa	580 - 620	255, 0, 80
Rojo	635 - 650	255, 0, 0
Amarillo	715 - 745	255, 190, 0
Verde	680 - 700	0, 255, 0

A continuación, se citaran diversos fragmentos del código esenciales para el correcto funcionamiento del Selector de Colores.

VI-B1. UART: Se inicializa el UART para el funcionamiento de la comunicación entre el microprocesador y el sistema.

Listing 5: UART

```
defineVELOCIDAD_BAUDIO    9600
defineVALOR_UBRR          ((F_CPU/16/VELOCIDAD_BAUDIO
    )-1)

static int uart_enviar_caracter(char c, FILE *s){
        if(c=='\n') uart_enviar_caracter('\r', s);
        while(!(UCSR0A & (1<<UDRE0)));
        UDR0 = c;
        return 0;
}
FILE uart_salida = FDEV_SETUP_STREAM(
    uart_enviar_caracter, NULL, _FDEV_SETUP_WRITE);

static inline void uart_iniciar(void){
        UBRR0H = (uint8_t)(VALOR_UBRR>>8);
        UBRR0L = (uint8_t)VALOR_UBRR;
        UCSR0B = (1<<TXEN0);
        UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
        stdout  = &uart_salida;
}
```

VI-B2. ADC: Se inicializa y configura el ADC para poder convertir las diferencias de voltaje provocados por la exposición de la fotocelda ante diferentes exposiciones a la luz.

Listing 6: ADC

```
defineCANAL_ADC_LDR 0

static void adc_iniciar(void){
        ADMUX  = (1<<REFS0) | (CANAL_ADC_LDR & 0x0F)
    ;
        ADCSRA = (1<<ADEN) | (1<<ADPS2)|(1<<ADPS1)|(
    1<<ADPS0);
        _delay_ms(2);
}
static uint16_t adc_leer(uint8_t canal){
        ADMUX = (ADMUX & 0xF0) | (canal & 0x0F);
        ADCSRA |= (1<<ADSC);
        while(ADCSRA & (1<<ADSC));
        return ADC;
}
static uint16_t adc_promedio(uint8_t canal, uint8_t
    n){
        uint32_t suma = 0;
        for(uint8_t i=0;i<n;i++) suma += adc_leer(
    canal);
        return (uint16_t)(suma/n);
}
```

VI-B3. Definición de Matriz de LEDs y Servomotor: La matriz de LEDs y el servomotor cumplen con la señalización

del color detectado, en el código se definen de la siguiente manera

Listing 7: Definición de Matriz y Servo

```
// servo
#define SERVO_OC1A_PIN PB1
#define SERVO_LIMITE_TIMER     40000
#define SERVO_MIN_TICKS      1000
#define SERVO_MAX_TICKS      5000

static void servo_iniciar(void){
        DDRB  |= (1<<SERVO_OC1A_PIN);
        TCCR1A = (1<<COM1A1)|(1<<WGM11);
        TCCR1B = (1<<WGM13)|(1<<WGM12)|(1<<CS11);
        ICR1   = SERVO_LIMITE_TIMER;
        OCR1A  = SERVO_MIN_TICKS;
}
static void servo_angulo(uint8_t a){
        if(a>180) a=180;
        uint16_t pulso = SERVO_MIN_TICKS +
        (uint32_t)(SERVO_MAX_TICKS - SERVO_MIN_TICKS
    )*a/180UL;
        OCR1A = pulso;
}

typedef enum { NINGUN_COLOR=0, ROSA, ROJO, AMARILLO,
     VERDE } color_t;

// matriz LED 8x8
#define PIN_LED 3
#define ANCHO_MATRIZ 8
#define ALTO_MATRIZ 8
#define NUM_LEDS_MATRIZ (ANCHO_MATRIZ * ALTO_MATRIZ)

extern uint8_t leds_matriz[NUM_LEDS_MATRIZ][3];

static void matriz_enviar_bit(uint8_t valor_bit);
static void matriz_enviar_byte(uint8_t byte);
static void matriz_mostrar(void);
static void matriz_poner_led(uint8_t indice_led,
    uint8_t r, uint8_t g, uint8_t b);
static void matriz_llenar_todo(uint8_t r, uint8_t g,
     uint8_t b);
static void matriz_iniciar(void);
static void matriz_aplicar_color(color_t color);
```

VI-B4. Rango ADC: En esta sección del código se establecen los valores ADC para cada color y se le asignan las caracteristicas del servo y la matriz correspondientes a este.

Listing 8: Rango ADC

```
typedef struct {
        const char *nombre;
        uint16_t bajo, alto; // rango ADC
        uint8_t  angulo;
        uint8_t  rgb[3];
} rango_t;

// rango de colores
static rango_t R[] = {
        [ROSA]     = {"ROSA",      580, 620,   0,  {
    255,  0,  80}},
        [ROJO]     = {"ROJO",      635, 650,  60,  {
    255,  0,   0}},
        [AMARILLO] = {"AMARILLO",  715, 745, 120,  {
    230,190,   0}},
        [VERDE]    = {"VERDE",     680, 700, 180,  {
      0,255,   0}},
};

static inline color_t detectar(uint16_t v){
        for(color_t c=ROSA; c<=VERDE; c++)
        if(v>=R[c].bajo && v<=R[c].alto) return c;
        return NINGUN_COLOR;
}
static inline uint16_t punto_medio(color_t c){
        return (uint16_t)((R[c].bajo + R[c].alto)/2)
    ;
}
```

VI-B5. Funciones para la matriz de LEDs: Para el manejo de la matriz de LEDs, se declaran las siguientes funciones

Listing 9: Funciones para la matriz

```
int main(void){
        cli();
        uart_iniciar();
        adc_iniciar();
        servo_iniciar();
        matriz_iniciar();
        sei();

        color_t actual = NINGUN_COLOR;
        uint8_t estable = 0, LEC_CONSEC = 3;

        printf("Esperando a detectar un color\n");

        while(1){
                uint16_t v = adc_promedio(
    CANAL_ADC_LDR, 32);
                color_t c = detectar(v);

                if(c == actual){
                        estable = 0;
                        } else if(c != NINGUN_COLOR)
    {
                        if(++estable >= LEC_CONSEC){
                                actual = c; estable
    = 0;
                                servo_angulo(R[c].
    angulo);
                                matriz_aplicar_color
    (actual);
                                uint16_t sp  =
    punto_medio(c);
                                int16_t  dif = (int1
    6_t)sp - (int16_t)v;
                                printf("LDR=%u |
    Color=%s | PM=%u | Dif=%d\n", v, R[c].nombre,
    sp, dif);
                        }
                        } else {
                        actual = NINGUN_COLOR;
                        estable = 0;
                }

                _delay_ms(10);
        }
}
```

VI-B6. Main: Por último, en este apartado se muestra el main del código.

Listing 10: Main

```
int main(void){
        cli();
        uart_iniciar();
        adc_iniciar();
        servo_iniciar();
        matriz_iniciar();
        sei();

        color_t actual = NINGUN_COLOR;
        uint8_t estable = 0, LEC_CONSEC = 3;
```

```
        printf("Esperando a detectar un color\n");

        while(1){
                uint16_t v = adc_promedio(
    CANAL_ADC_LDR, 32);
                color_t c = detectar(v);

                if(c == actual){
                        estable = 0;
                        } else if(c != NINGUN_COLOR)
    {
                        if(++estable >= LEC_CONSEC){
                                actual = c; estable
    = 0;
                                servo_angulo(R[c].
    angulo);
                                matriz_aplicar_color
    (actual);
                                uint16_t sp  =
    punto_medio(c);
                                int16_t  dif = (int1
    6_t)sp - (int16_t)v;
                                printf("LDR=%u |
    Color=%s | PM=%u | Dif=%d\n", v, R[c].nombre,
    sp, dif);
                        }
                        } else {
                        actual = NINGUN_COLOR;
                        estable = 0;
                }

                _delay_ms(10);
        }
}
```

VI-C. Piano Electrónico con ATmega328P

El sistema fue validado tanto en simulación como en la implementación física. Durante la simulación en el software Proteus, se verificó que el funcionamiento del piano electrónico y la reproducción de las canciones predefinidas respondía correctamente a los comandos UART y que los pulsadores generaban las notas correctas en los buzzers.

En la Figura 11, se observa la simulación del circuito completo, donde se pueden identificar las conexiones del ATmega328P, los pulsadores, los dos buzzers y la red de conexión a los pines del microcontrolador. Además, la señal PWM generada en los pines de los buzzers fue analizada, confirmando que las frecuencias generadas correspondían a las notas musicales esperadas.

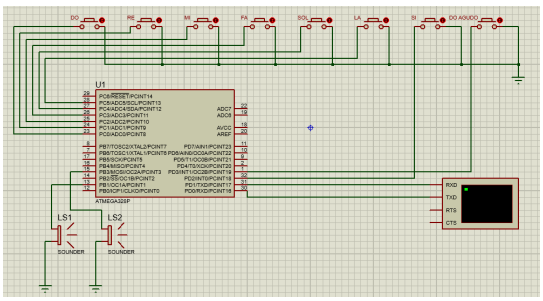

Fig. 11: Simulación del Piano Electrónico en Proteus con el ATmega328P.

Una vez realizado el montaje físico del circuito en protoboard, se procedió a probar cada funcionalidad del sistema. Los pulsadores, al ser presionados, generaron la nota correspondiente en los buzzers. Se verificó que las frecuencias de las notas eran correctas y que el sistema respondía sin fallos a los comandos UART enviados desde una computadora. En la Figura 12, se muestra la implementación física del sistema.

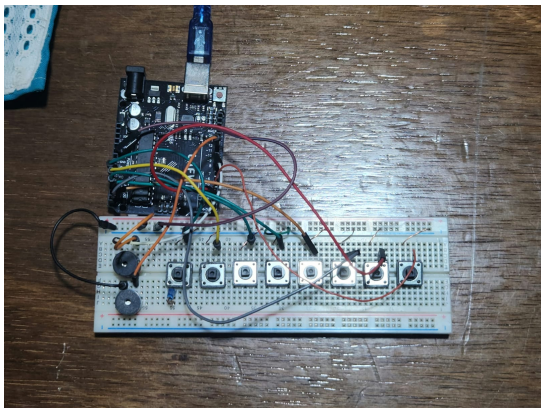

Fig. 12: Montaje físico del Piano Electrónico en protoboard con el ATmega328P.

En cuanto a la reproducción de las canciones predefinidas, la comunicación UART permitió seleccionar entre las dos canciones almacenadas en la LUT. La canción **C1: "Lose Yourself"de Eminem** y **C2: "Spyder Dance"de Undertale** se reprodujeron correctamente en ambos buzzers, cada uno con una pista diferente, lo que permitió escuchar las canciones en su totalidad y con la calidad de sonido esperada. Las funciones de parada de canción y cambio entre canciones también fueron validadas correctamente, asegurando la versatilidad del sistema.

Se comprobó que la selección de canciones mediante los comandos 'C1' y 'C2' a través de UART funcionaba sin inconvenientes. En la siguiente figura 13, se muestra la interfaz UART utilizada para interactuar con el sistema.

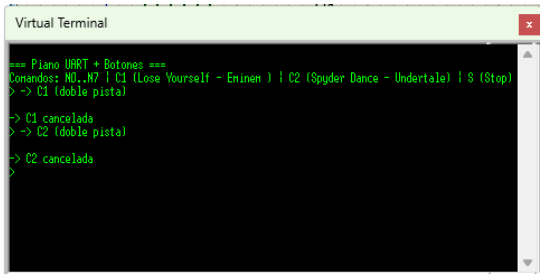

Fig. 13: Interfaz UART utilizada para seleccionar las canciones.

En resumen, el sistema funciona correctamente tanto en el modo piano, donde los 8 pulsadores emiten las notas correspondientes, como en el modo reproducción de canciones predefinidas (C1 y C2). La adición de un segundo buzzer permitió la reproducción de dos pistas simultáneas, mejorando la calidad del sonido.

Se verificó que las canciones **C1** y **C2** incluían notas adicionales a las originales, lo que enriqueció la experiencia

sonora y dio como resultado una reproducción más compleja y agradable. El sistema respondió correctamente tanto en la simulación como en el montaje físico, y las funciones de selección y parada de canciones a través de UART se ejecutaron sin errores.

El siguiente cuadro X muestra todas las notas que se utilizaron en las canciones, tanto las proporcionadas inicialmente como las notas adicionales que fueron agregadas para mejorar la experiencia sonora.

Cuadro X: Notas adicionales utilizadas en las canciones C1 y C2

Nota	Frecuencia (Hz)	Nota	Frecuencia (Hz)
A5	880	G3	196
F5	698	F3	175
D5	587	A3	220
G5	784	E3	165
E5	659	A2	110
C5	523	Cb3	139
C6	1047	Cb4	277
Ab5	932	Cb6	1109
Ab4	466	Gb4	415
G4	392	Gb5	831
A4	440	F6	1397
Cb5	554	E6	1319
E4	330	Db6	1245
B4	494	Db5	622
F4	349	B5	988
D4	294	Gb2	104
C4	262	Gb3	208
Ab3	233	Cb2	69
G3	196	F2	87
F3	175	E2	82
A3	220	Fb2	92
E3	165	Fb3	185
A2	110	Ab2	117
Cb3	139	C3	131
Db3	156	D3	147
B3	247	Db4	311

A continuación, se destacan algunos fragmentos importantes del código que permiten entender cómo se gestionan las notas, la selección de canciones y la comunicación UART.

VI-C1. Configuración UART: La configuración de UART permite la comunicación entre el microcontrolador y el sistema, y se inicializa de la siguiente forma:

Listing 11: Configuración UART

```
static void uart_iniciar(void){
    uint16_t ubrr = 103;  // 16MHz, 9600, U2X=0
    UCSR0A &= ~(1<<U2X0);
    UBRR0H = (uint8_t)(ubrr >> 8);
    UBRR0L = (uint8_t)(ubrr & 0xFF);
    UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
    UCSR0B = (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0);
}
```

VI-C2. Definición de los pulsadores y buzzers: Los pulsadores están conectados a los pines del microcontrolador, y el código que gestiona su lectura es el siguiente:

Listing 12: Definición de los pulsadores y buzzers

```
// Botones
defineBTN0 PC0
defineBTN1 PC1
defineBTN2 PC2
defineBTN3 PC3
defineBTN4 PC4
defineBTN5 PC5
defineBTN6 PD2
defineBTN7 PD3

static inline void botones_iniciar(void){
        DDRC  &= ~((1<<BTN0)|(1<<BTN1)|(1<<BTN2)|(1
    <<BTN3)|(1<<BTN4)|(1<<BTN5));
        PORTC |=  ((1<<BTN0)|(1<<BTN1)|(1<<BTN2)|(1
    <<BTN3)|(1<<BTN4)|(1<<BTN5));
        DDRD  &= ~((1<<BTN6)|(1<<BTN7));
        PORTD |=  ((1<<BTN6)|(1<<BTN7));
}

static inline uint8_t leer_boton(void){
        if(!(PINC&(1<<BTN0))) return 0;
        if(!(PINC&(1<<BTN1))) return 1;
        if(!(PINC&(1<<BTN2))) return 2;
        if(!(PINC&(1<<BTN3))) return 3;
        if(!(PINC&(1<<BTN4))) return 4;
        if(!(PINC&(1<<BTN5))) return 5;
        if(!(PIND&(1<<BTN6))) return 6;
        if(!(PIND&(1<<BTN7))) return 7;
        return 255;
}
```

VI-C3. Menú de selección: El menú para la selección de canciones y notas se gestiona mediante UART, con el siguiente código:

Listing 13: Menú de selección

```
static void imprimir_banner(void){
        uart_imprimir("\r\n=== Piano UART + Botones
    ===\r\n");
        uart_imprimir("Comandos: N0..N7 | C1 (Lose
    Yourself - Eminem ) | C2 (Spyder Dance -
    Undertale) | S (Stop) | H\r\n> ");
}
```

VI-C4. LUT para C1 y C2: Las canciones C1 y C2 se almacenan en Look-Up Tables (LUT) y se gestionan de la siguiente forma:

Listing 14: LUT para C1 y C2

```
// TABLAS para C1 y C2
// C1 PISTA A Lose Yourself - Eminem
const uint16_t midiC1a[][3] PROGMEM = {
        {A5,474,26},{F5,474,26},{D5,474,26},{A5,474,
    26},{G5,474,26},...
};

// C2 PISTA A Spyder Dance - Undertale
const uint16_t midiC2a[][3] PROGMEM = {
        {F5,247,14},{C5,247,14},{Gb4,247,14},{F4,247
    ,145},{B4,123,8},...
}
```

VI-D. Cerradura Electrónica con Teclado y LCD

El programa, desarrollado en lenguaje C de bajo nivel para el microcontrolador ATmega328P, implementa una cerradura electrónica que gestiona la lectura del teclado matricial 4×4, la interacción con la pantalla LCD 16×2 y el almacenamiento

persistente de la contraseña en la EEPROM. El sistema permite el ingreso y verificación de claves (longitud configurable entre 4 y 6 dígitos), incluye un menú de opciones para operar (ingresar clave y cambiar clave) y Señaliza sus estados mediante LEDs y un buzzer activo (acceso concedido, error, bloqueo por intentos fallidos). Además, se integraron rutinas de antirrebote en la lectura del teclado.

El programa se organizo en funciones pequeñas y con propósito único. Así es más fácil de leer, probar y documentar: cada tarea (inicializar periféricos, operar la LCD, gestionar el teclado, leer/escribir la clave, validar y mostrar menús) queda encapsulada en una *función*, y el flujo general se entiende llamando a esas piezas bien nombradas desde *main*.

En cuanto al codigo, este incluye las librerías de AVR para registros (<avr/io.h>), retardos (<util/delay.h>), enteros (<stdint.h>), acceso a memoria de programa/flash (<avr/pgmspace.h>) y EEPROM (<avr/eeprom.h>), además de booleanos (<stdbool.h>). En particular, <avr/eeprom.h> habilita eeprom_read_∗, eeprom_update_∗ y **eeprom_update_block**, y <avr/pgmspace.h> aporta **PSTR**() y pgm_read_byte() para manejar cadenas almacenadas en la memoria flash (PROGMEM) sin consumir RAM.

Sobre las funciones de la LCD: se inicializa el bus I2C con i2c_init , se envían datos a la pantalla mediante el expansor con **lcd_i2c_write**, y se ejecuta la secuencia estándar de arranque en 4 bits con lcd_i2c_init . Luego se ofrecen utilitarias para limpiar pantalla (**lcd_clear**), mover el cursor a la segunda línea (**lcd_linea2**) y mostrar cadenas directamente desde flash (**lcd_i2c_write_string_P**).

Este primer bloque presenta el envio de cadenas almacenadas en la memoria flash (PROGMEM) hacia la LCD. A continuacion, byte a byte se extrae con pgm_read_byte y se vuelca mediante **lcd_i2c_write**; seguidamente se ilustra el uso tipico con **PSTR**().

Listing 15: PROGMEM a LCD: lectura con pgm_read_byte y envio con lcd_i2c_write

```
/* Escribe un string almacenado en FLASH (PROGMEM)
    caracter por caracter */
void lcd_i2c_write_string_P(PGM_P p){
    char c;
    while ((c = pgm_read_byte(p++))) {
        lcd_i2c_write((uint8_t)c, LCD_DATA);
    }
}

/* Ejemplos de uso con PSTR(...) (cadenas en flash)
    */
lcd_i2c_write_string_P(PSTR("Ingrese Clave"));
lcd_i2c_write_string_P(PSTR("Clave Correcta"));
lcd_i2c_write_string_P(PSTR("4 o 6 digitos"));
```

Seguidamente, se aborda el acceso basico a la EEPROM asociado a la clave. Primero se define la obtencion/actualizacion del largo; luego, de manera coherente, se leen y escriben los digitos individuales evitando desgaste innecesario con **eeprom_update_byte**.

Listing 16: EEPROM: largo de clave y acceso a digitos

```
/* Lee el largo de la clave desde EEPROM (4 o 6) */
static inline uint8_t ee_get_largo(void) {
    return eeprom_read_byte(EE_DIRECCION_LARGO);
}

/* Escribe/actualiza el largo de la clave en EEPROM
    */
static inline void ee_set_largo(uint8_t largo) {
    eeprom_update_byte(EE_DIRECCION_LARGO, largo);
}

/* Lee un digito i de la clave desde EEPROM */
static inline uint8_t ee_get_digit(uint8_t i) {
    return eeprom_read_byte(EE_BASE + 1 + i);
}

/* Escribe/actualiza un digito i en EEPROM */
static inline void ee_set_digit(uint8_t i, uint8_t v
    ) {
    eeprom_update_byte(EE_BASE + 1 + i, v);
}
```

A continuacion, se captura una clave desde el keypad mostrando cada ingreso en la LCD; posteriormente, se guarda de forma persistente en EEPROM. Nótese que, cuando el largo es 4, se limpian los sobrantes para mantener consistencia.

Listing 17: Keypad: lectura de clave y persistencia en EEPROM

```
/* Lee 'largo' teclas del keypad y las muestra en
    LCD */
static void leer_clave(uint8_t largo, char *buf) {
    for (uint8_t i = 0; i < largo; i++) {
        char x;
        do { x = Leer_keypad(); } while (x == 0);
        buf[i] = x;
        lcd_i2c_write((uint8_t)x, LCD_DATA);
    }
}

/* Pide nueva clave por keypad y la guarda en EEPROM
    ; limpia sobrantes si len=4 */
static void escribir_clave(uint8_t largo) {
    ee_set_largo(largo);
    for (uint8_t i = 0; i < largo; i++) {
        char x;
        do { x = Leer_keypad(); } while (x == 0);
        lcd_i2c_write((uint8_t)x, LCD_DATA);
        ee_set_digit(i, (uint8_t)x);
    }
    if (largo == 4) { ee_set_digit(4, 0xFF);
    ee_set_digit(5, 0xFF); }
}
```

Acto seguido, se verifica una clave ingresada contra la almacenada en EEPROM. Para completar el flujo de interaccion, se define una lectura bloqueante que actua como tecla Enter, util en secuencias que esperan confirmacion.

Listing 18: Validacion de clave y lectura bloqueante de Enter

```
/* Compara un buffer en RAM contra la clave guardada
    en EEPROM */
static bool clave_coincide(uint8_t len, const char *
    buf) {
    for (uint8_t i = 0; i < len; i++) {
        if ((uint8_t)buf[i] != ee_get_digit(i))
    return false;
    }
    return true;
}

/* Lee una tecla de forma bloqueante (Enter) */
```

```
static char Enter(void){
    char x = 0;
    do {
        x = Leer_keypad();
    } while (x == 0);
    return x;
}
```

Posteriormente, se implementa el flujo de cambio de clave: se valida la clave anterior con hasta tres intentos; seguidamente, el usuario elige el nuevo largo (4 o 6) y se actualiza la EEPROM, brindando ademas retroalimentacion visual y sonora.

Listing 19: Flujo de cambio de clave: seleccion 4/6 y actualizacion

```
static void cambiar_clave(void){
    uint8_t l = ee_get_largo(); // 4 o 6
    uint8_t c = 0; // intentos
    uint8_t x = 0;
    char ingresada[6];

    do{
        lcd_clear();
        lcd_i2c_write_string_P(PSTR("Clave Vieja?"))
    ;
        lcd_linea2();
        leer_clave(l, ingresada);

        if (clave_coincide(l, ingresada)) {
            lcd_clear();
            lcd_i2c_write_string_P(PSTR("Clave
    Correcta"));
            PORTC &= ~(1<<LED_ROJO);
            PORTC |=  (1<<LED_VERDE);
            _delay_ms(1000);
            PORTC &= ~(1<<LED_VERDE);

            /* Elegir largo de la nueva clave */
            do {
                lcd_clear();
                lcd_i2c_write_string_P(PSTR("Clave
    nueva"));
                lcd_linea2();
                lcd_i2c_write_string_P(PSTR("4 o 6
    digitos"));
                x = Enter();
            } while (x != '4' && x != '6');

            lcd_clear();
            lcd_i2c_write_string_P(PSTR("Clave nueva
    :"));
            lcd_linea2();
            if (x == '4') { escribir_clave(4); }
            else          { escribir_clave(6); }

            /* Feedback final */
            lcd_clear();
            lcd_i2c_write_string_P(PSTR("Clave"));
            lcd_linea2();
            lcd_i2c_write_string_P(PSTR("Actualizada
    "));
            _delay_ms(1000);
            return;
        }
        else { /* Error de clave vieja */
            lcd_clear();
            lcd_i2c_write_string_P(PSTR("Clave Vieja
    "));
            lcd_linea2();
            lcd_i2c_write_string_P(PSTR("Incorrecta"
    ));
            PORTC &= ~(1<<LED_VERDE);
            PORTC |=  (1<<LED_ROJO);
            _delay_ms(1000);
            c++;
        }
    } while (c < 3);

    /* 3 fallos activan alarma/bloqueo con buzzer */
    if (c >= 3) {
        PORTC |= (1<<BUZZER);
        _delay_ms(2500);
        PORTC &= ~(1<<LED_VERDE);
        PORTC &= ~(1<<BUZZER);
        PORTC &= ~(1<<LED_ROJO);
    }
}
```

Finalmente, se incluye el flujo de “poner clave”, donde se solicita la clave actual y se valida con un máximo de tres intentos. En caso de acierto, se enciende el LED verde y se notifica al usuario; en caso contrario, se enciende el LED rojo y se incrementa el contador de fallos. Tras tres intentos fallidos, suena el buzzer como indicación de bloqueo/alarma.

Listing 20: Flujo de puesta de clave: validacion con 3 intentos

```
static void poner_clave(void) {
    uint8_t l = ee_get_largo(); // 4 o 6
    uint8_t c = 0;
    char ingresada[6];

    do {
        lcd_clear();
        lcd_i2c_write_string_P(PSTR("Ingrese Clave")
    );
        lcd_linea2();

        leer_clave(l, ingresada);

        if (clave_coincide(l, ingresada)) {
            lcd_clear();
            lcd_i2c_write_string_P(PSTR("Clave
    Correcta"));
            PORTC &= ~(1<<LED_ROJO);
            PORTC |=  (1<<LED_VERDE);
            _delay_ms(2500);
            PORTC &= ~(1<<LED_VERDE);
            PORTC &= ~(1<<BUZZER);
            PORTC &= ~(1<<LED_ROJO);
            return;   /* fin del flujo */
        } else {
            lcd_clear();
            lcd_i2c_write_string_P(PSTR("Clave
    Incorrecta"));
            PORTC &= ~(1<<LED_VERDE);
            PORTC |=  (1<<LED_ROJO);
            _delay_ms(1000);
            c++;
        }
    } while (c < 3);

    /* 3 intentos fallidos: buzzer */
    if (c >= 3) {
        PORTC |= (1<<BUZZER);
        _delay_ms(2500);
        PORTC &= ~(1<<LED_VERDE);
        PORTC &= ~(1<<BUZZER);
        PORTC &= ~(1<<LED_ROJO);
    }
}
```

En este aparado, se contempla la inicializacion de la EEPROM cuando esta vacia: si el largo almacenado no es

valido, se establece una clave por defecto y se marcan los restantes como no usados.

Listing 21: Inicializacion: clave por defecto si EEPROM vacia

```
static void clave_si_eeprom_vacia(void){
    uint8_t len = ee_get_largo(); /* lee 0x000 */
    if (len != 4 && len != 6) {   /* EEPROM no
    inicializada */
        /* LEN=4, digitos '0''0''0''0', resto 0xFF
    */
        eeprom_update_block(
            (const void*)(uint8_t[]){ 4,'0','0','0',
    '0',0xFF,0xFF },
            EE_BASE, 7
        );
    }
}
```

Por ultimo, se inicializan los periféricos (keypad, salidas de buzzer/LEDs, I2C y la LCD) y se asegura un estado valido de EEPROM con una clave por defecto si fuera necesario. Luego, el programa entra en un bucle que repite el menu únicamente cuando la ruta “cambiar clave” concluye con exito (es decir, Menu() devuelve 1). En cambio, si el usuario elige “poner clave” y finaliza el flujo, Menu() retorna 0, se sale del bucle y el firmware permanece en un idle infinito (while(1)), a la espera de un reinicio o nueva interaccion.

Listing 22: Funcion principal: inicializacion, menu condicional e idle

```
int main(void)
{
    iniciar_keypad();
    init_buzzer_led();
    i2c_init();
    lcd_i2c_init();
    clave_si_eeprom_vacia();

    /* Repetir el menu solo cuando cambiar_clave
    termina con exito (devuelve 1) */
    while ( Menu() ) {
        /* loop de menu */
    }

    /* Si se eligio "Poner Clave" y termino,
    permanecer en idle */
    while (1) {
        /* idle */
    }

    return 0; /* opcional en entornos bare-metal */
}
```

Luego de terminado el código y documentado, se pasó a la etapa de simulación para comprobar que todo funcionara como estaba pensado. Seguidamente, en la Fig. 14 se muestra el circuito simulado.

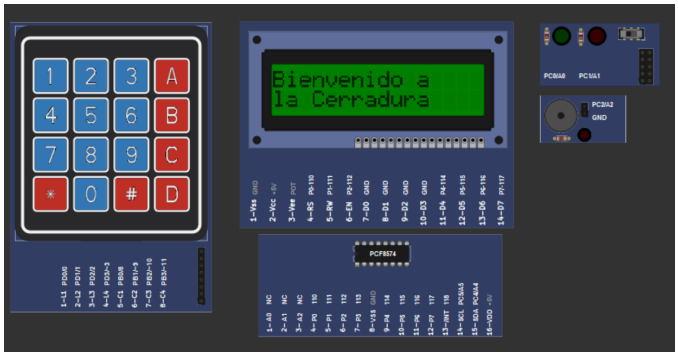

Fig. 14: Circuito cerradura electrónica simulado

En cuanto a la simulación, el funcionamiento general fue correcto. Al reiniciar la ATmega328p, el contenido de la EEPROM se conserva como es esperado, y la simulación responde adecuadamente tanto con la contraseña por defecto (“0000”) como luego de cambiarla: el valor queda almacenado en EEPROM y el sistema opera de forma correcta con la nueva clave. En el Drive adjunto se incluye un video que muestra el funcionamiento esperado en la simulación.

Una vez realizada la simulación, se procedió con el montaje físico, como se muestra en la Figura 15.

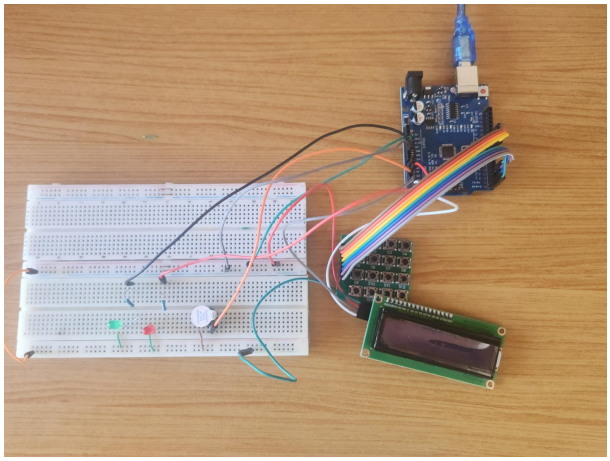

Fig. 15: Circuito de la cerradura electrónica en su implementación física

En el montaje físico, el sistema funciona correctamente en todos sus aspectos, incluyendo la gestión de la EEPROM. En el Drive se adjunta un video que muestra el funcionamiento del circuito real.

VII. Conclusiones

Para el apartado de cerradura electrónica, se concluye que la cerradura electrónica con teclado 4×4 y LCD sobre el microcontrolador ATmega328P cumple los objetivos funcionales y de diseño planteados: lectura confiable con antirrebote, interfaz clara en la pantalla LCD (incluyendo cadenas en PROGMEM para optimizar el uso de RAM), persistencia de la clave en la memoria EEPROM con escritura cuidadosa para minimizar el desgaste, y un flujo de uso completo con menú (poner/cambiar clave), retroalimentación mediante LEDs y buzzer, y bloqueo

tras intentos fallidos. La organización modular del código, con funciones pequeñas y de propósito único, favoreció la legibilidad, las pruebas y la documentación, reflejándose en una integración estable.

Respecto a la simulación, no se observaron desvíos: la persistencia del contenido de la `EEPROM` se mantuvo tras un reinicio de la ATmega328P en PICSimLab, reproduciendo el comportamiento del montaje físico. Esto refuerza que el diseño y el código implementado son correctos.

En síntesis, el prototipo valida la solución propuesta en condiciones reales. Como posibles mejoras a nivel de código, se podría enmascarar la contraseña en el LCD con "*" durante la introducción, ya que actualmente se muestran los dígitos en claro. Esto mejoraría la privacidad del usuario sin afectar la usabilidad.

En el apartado de control de plotter, la estrategia de codificar cada paso como pares {`PORTD`, `delay`} almacenados en `PROGMEM`, junto con una temporización unificada mediante `AUTOGEN_SCALE_MS_PER_S`, permitió reproducir en el equipo real lo verificado en la previsualización. La desactivación temporal de la UART durante el trazo evitó *jitter* y mejoró la definición de vértices y curvas, sin evidenciar pérdidas de paso en las pruebas repetidas. La herramienta de conversión (imagen binaria $\rightarrow$ LUT) y la simulación previa resultaron clave para reducir iteraciones de ajuste y asegurar correspondencia entre simulación y trazado físico.

Siguiendo con el apartado de piano electrónico, la generación de tonos por PWM y la incorporación de dos buzzers en paralelo funcional (una pista por canal) mejoraron la claridad y la riqueza tímbrica frente a una sola salida. La selección y control por UART se integraron sin interferir con el modo "teclado", y la organización de las melodías en LUTs facilitó su edición y el escalado temporal. En simulación y en el montaje físico, las frecuencias de las notas y la sincronización entre pistas se mantuvieron dentro de lo esperado, validando la arquitectura de temporizadores y el mapeo de pines empleados

Por último, para el apartado del selector de colores, se concluye un funcionamiento eficaz de este, siendo capaz de identificar todos los colores del grupo de colores seleccionado, y siendo cada uno correctamente señalizado mediante monitor serial, matriz de LEDs y el servomotor en la estrella de colores. A modo de mejora, se recomienda cambiar la cobertura de la fotocelda, ya que el rollo recortado cumple su función, pero podría colocarse algo más eficiente, lo mismo para la iluminación. Lo más optimo es colocar una pared dentro de la cubierta que separe el LED de la fotocelda para que no la ilumine directamente, este LED debe ser un LED RGB. Para el ajuste de los valores del rango ADC, se debe posicionar sobre el color deseado y variar el LED RGB entre rojo, azul y verde, esto con el fin de que la luz salga de la cubierta y se refleje contra el color, volviendo a la fotocelda, conociendo así los componentes RGB del color donde se posiciono.

VIII. Anexos

1. Link al Repositorio de GITHUB: https://github.com/juanferreiram-cell/Tec.Microprocesamiento
2. Link a la Carpeta de Drive con los videos: https://drive.google.com/drive/folders/1dfyBWJg07bqZ1Sa87PI9gOdNfCK04ClB?usp=sharing

IX. Bibliografía

1. Administrador. (2025, May 6). I2C - Puerto, Introducción, trama y protocolo. HeTPro-Tutoriales. https://hetpro-store.com/TUTORIALES/i2c/
2. Del Valle Hernández, L. (2021, March 23). LCD con Arduino texto en movimiento paso a paso. Programarfacil Arduino y Home Assistant. https://programarfacil.com/blog/arduino-blog/texto-en-movimiento-en-un-lcd-con-arduino/
3. Del Valle Hernández, L. (2022, January 13). Zumbador o buzzer con Arduino y librería EasyBuzzer. Programarfacil Arduino y Home Assistant. https://programarfacil.com/blog/arduino-blog/buzzer-con-arduino-zumbador/
4. Del Valle, L. (2022, January 13). Memoria EEPROM de Arduino. Programarfacil Arduino y Home Assistant. https://programarfacil.com/blog/arduino-blog/eeprom-arduino/
5. DFRobot. (s. f.). 8×8 RGB LED Matrix – SKU DFR0459. https://wiki.dfrobot.com/8x8_RGB_LED_Matrix_SKU__DFR0459
6. Joan. (2024, May 2). ¿Qué es el MIDI?: La guía del principiante para la herramienta musical más poderosa. LANDR Blog. https://blog.landr.com/es/que-es-el-midi-la-guia-del-principiante-para-la-herramienta-musical-mas-poderosa/
7. MIDI to arduino tone. (n.d.). https://arduinomidi.netlify.app
8. MIT Illuminations Seminar. (s. f.). https://learn.illuminations.mit.edu/chapter/ws2812b
9. MSMK. (2025, Mayo 19). ¿Qué es el lenguaje de programación C? https://msmk.university/que-es-el-lenguaje-de-programacion-c/
10. Servo Motor SG90 Data Sheet. (s. f.). SG90 9 g Micro Servo – Data sheet. http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf
11. TecnoSalva. (2024, 24 abril). Qué es y como funciona una LDR (resistencia dependiente de la luz) - Tecnosalva. Tecnosalva. https://www.tecnosalva.com/que-es-y-como-funciona-una-ldr/