Proyecto:

La Gallinita Ciega

Integrantes:

Abraham Jaramillo Quinde
Jeremy Delgado Rodríguez
William Ortiz Vásquez
Armando Moreira Salvatierra

Paralelo:

2

Docente:

SOLIS MESA RONALD DAVID

Contenido

1. Objetivo General

- Desarrollar un videojuego interactivo basado en hardware embebido, que permita al usuario controlar una gallina en una matriz LED 8x8 con obstáculos en movimiento, mediante botones físicos y con retroalimentación sonora a través de la comunicación entre dos microcontroladores.

2. Objetivos Específicos

- Integrar los microcontroladores ATmega328P y PIC16F887 para coordinar el control visual y sonoro del juego.
- Diseñar e implementar los tres niveles de dificultad con condiciones progresivas para la jugabilidad.
- Desarrollar una interfaz de usuario basada en botones físicos que permita un control eficiente y simple de la gallina.

3. Descripción Técnica del Juego

El sistema desarrollado es un juego interactivo de desplazamiento vertical que simula el cruce de una "gallina" a través de una vía con obstáculos. El objetivo del jugador es llevar la gallina desde la parte inferior hasta la parte superior de la matriz LED 8x8 sin colisionar con obstáculos que se mueven horizontalmente. El juego se controla mediante pulsadores físicos y está implementado sobre un sistema embebido que integra dos microcontroladores: un ATmega328P y un PIC16F887, los cuales se comunican entre sí mediante un canal de datos paralelo.

3.1. Componentes del sistema

- ATmega328P: Controla la matriz LED 8x8, gestiona la lógica del juego, lectura de pulsadores y determina los eventos (inicio, colisión, victoria).
- PIC16F887: Reproduce sonidos a través de un buzzer o altavoz (LS1), en respuesta a eventos específicos del juego (inicio, colisión, victoria, cambio de nivel).
- Matriz LED 8x8: Actúa como el escenario del juego, mostrando la posición de la gallina y los obstáculos en movimiento.
- Pulsadores: Cuatro botones físicos conectados al ATmega328P permiten el movimiento de la gallina en las direcciones: arriba, abajo, izquierda y derecha.
- Canal de comunicación paralela: Permite que el ATmega328P informe al PIC16F887 sobre el estado actual del juego para generar efectos sonoros correspondientes.

Al iniciar el sistema, se reproduce una melodía de bienvenida y se presenta el nivel 1 (L1) brevemente en la matriz LED. El jugador usa los pulsadores para mover la gallina en la matriz, evitando los obstáculos que se desplazan horizontalmente. Si la gallina colisiona con un obstáculo, se reproduce un sonido de error y se reinicia al nivel 1. Si el jugador llega con éxito a la fila superior, se reproduce una melodía de victoria y se avanza al siguiente nivel.

El juego consta de 3 niveles progresivos de dificultad, donde se modifican las condiciones del entorno de juego.

3.2. Descripción Detallada de los Niveles

Nivel 1 – Fácil

- **Objetivo:** Familiarizar al jugador con la mecánica básica.
- **Obstáculos:** Pocos y de menor tamaño.
- **Velocidad de desplazamiento: Lenta**.
- **Tiempo límite:** Ilimitado.
- **Sonido:** Se reproduce una melodía al iniciar el juego y otra al ganar.

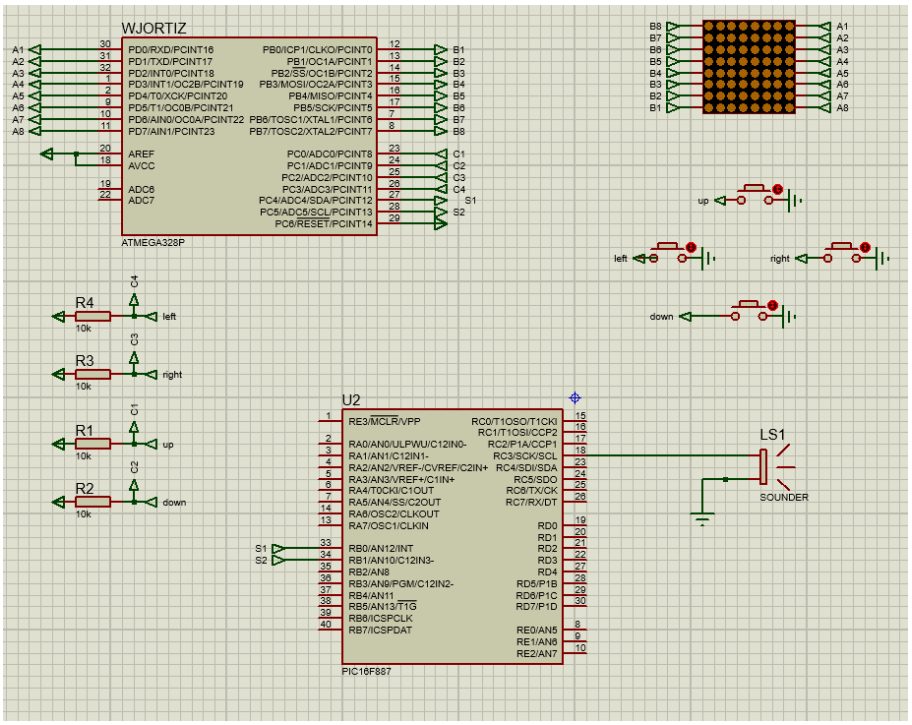
Nivel 2 – Medio

- **Objetivo:** Aumentar el desafío.
- **Obstáculos:** De mayor tamaño (2-3 LEDs por fila), movimiento más rápido.
- **Velocidad de desplazamiento:** Media.
- **Tiempo límite:** Ilimitado.
- **Sonido:** Efectos de cambio de nivel y victoria/derrota.

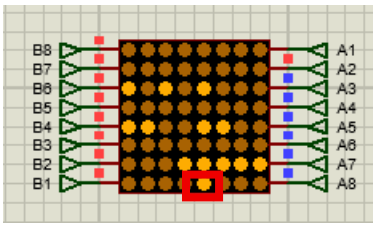
Nivel 3 – Difícil

- **Objetivo:** Evaluar reflejos y velocidad del jugador.
- **Obstáculos:** Grandes y numerosos, distribuidos en varias filas.
- **Velocidad de desplazamiento:** Alta.
- **Tiempo límite:** 15 segundos.
- **Mecánica adicional:** Si el jugador no logra cruzar dentro del tiempo, se reproduce una melodía de derrota y se reinicia al nivel 1.

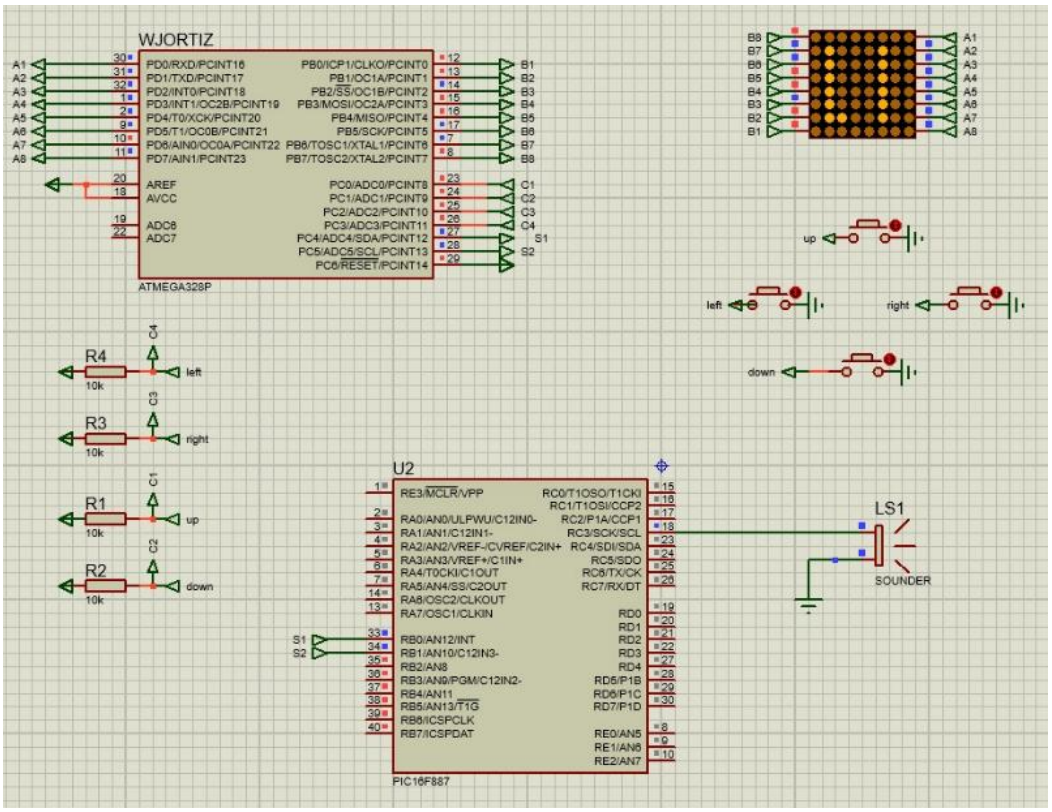
4. Capturas de Simulación en Proteus



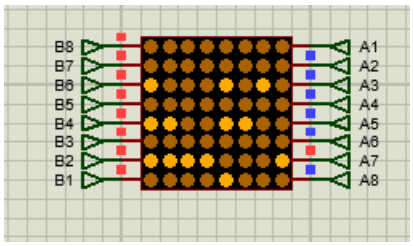
Captura 1 Diagrama esquemático del juego en Proteus.



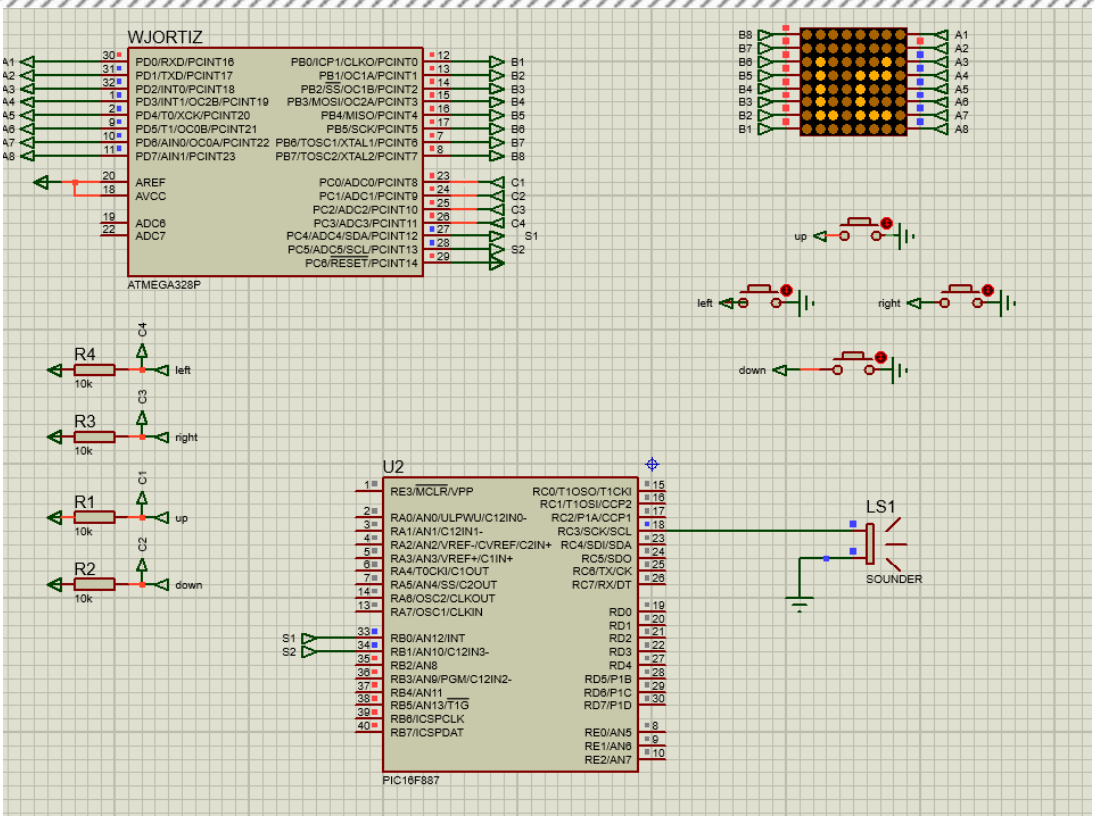
Captura 2 Personaje



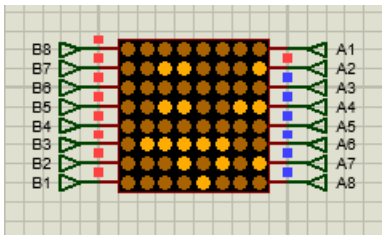
Captura 3 Inicio Nivel 1.



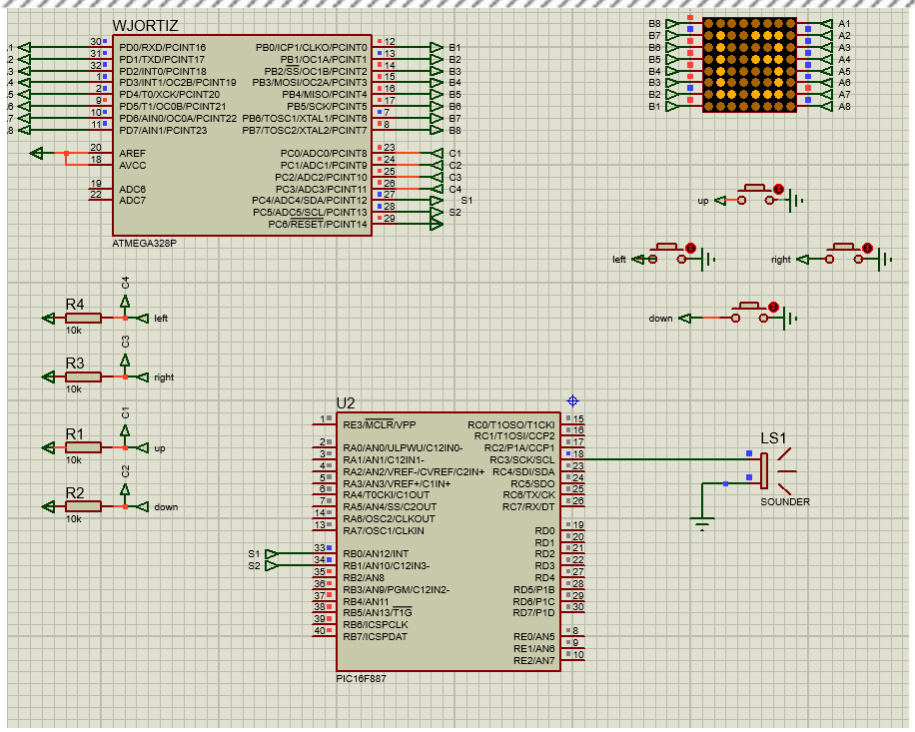
Captura 4 Obstáculos pequeños.



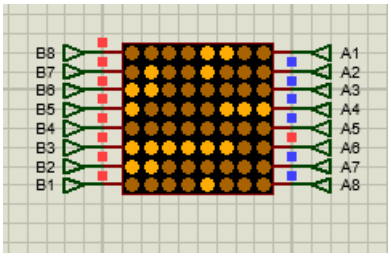
Captura 5 Nivel 2.



Captura 6 Obstáculos medianos.



Captura 7 Nivel 3.



Captura 8 Obstáculos grandes y límite de tiempo.

5. Descripción del Código ATmega328P

El microcontrolador ATmega328P gestiona el juego. Define la matriz LED, las posiciones de la gallina y obstáculos, y contiene funciones para lectura de botones, detección de colisiones y manejo de niveles. Utiliza los puertos D y B para mostrar la matriz y PORTC para la comunicación con el PIC.

1. Inicialización de constantes y variables globales

El sistema establece como frecuencia de operación 8 MHz, y se definen las posiciones iniciales de la gallina, así como las estructuras que contienen los obstáculos y su dirección de movimiento.

Código:

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

//Posicion de la gallina y variable nivel
uint8_t gallina_x = 3;
```

```
uint8_t gallina_y = 7;
uint8_t nivel = 1;

uint8_t obstaculos[8];
int8_t direcciones[8];

// Matriz de L1
uint8_t L1[8] = {
    0x00, 0x44, 0x44, 0x44, 0x44, 0x44, 0x64, 0x00
};

// Matriz de L2
uint8_t L2[8] = {
    0x00, 0x4E, 0x42, 0x4E, 0x48, 0x48, 0x6E, 0x00
};

// Matriz de L3
uint8_t L3[8] = {
    0x00, 0x4E, 0x42, 0x4E, 0x42, 0x42, 0x6E, 0x00
};
```

También se definen tres matrices que representan los patrones gráficos de los niveles L1, L2 y L3 que se visualizan al inicio de cada nivel.

2. Mostrar el número de nivel – mostrar_letra()

Esta función despliega en la matriz LED el número del nivel actual (1, 2 o 3) mediante desplazamientos hacia la izquierda, simulando un efecto visual atractivo.

Código:

```
void mostrar_letra(uint8_t letra[8], uint16_t duracion_ms) {

    uint16_t repeticiones = duracion_ms / 8;  // Duración total /
tiempo por ciclo
    uint8_t desplazamiento_max = 8;            // Mover 8 veces para
desplazar toda la letra fuera

    for (uint8_t d = 0; d < desplazamiento_max; d++) {
        for (uint16_t r = 0; r < repeticiones / desplazamiento_max;
r++) {
            for (uint8_t j = 0; j < 8; j++) {
                PORTD = 1 << j;
                // Desplaza los bits de la fila j hacia la derecha d
veces (efecto derecha a izquierda)
                PORTB = ~(letra[j] >> d);
                _delay_ms(1);
            }
```

```
        }
    }
}
```

3. Lectura de pulsadores – leer_botones()

La función captura las entradas de cuatro pulsadores físicos conectados a PC0–PC3, y permite mover la gallina hacia arriba, abajo, izquierda y derecha. También incluye un retardo de antirrebote.

Código:

```
void leer_botones() {
    static uint8_t last_state = 0x0F;
    uint8_t current = PINC & 0x0F;

    if (current != last_state) {
        _delay_ms(10); // rebote
        current = PINC & 0x0F;

        if (!(current & (1 << PC0)) && gallina_y > 0) gallina_y--;
        if (!(current & (1 << PC1)) && gallina_y < 7) gallina_y++;
        if (!(current & (1 << PC2)) && gallina_x > 0) gallina_x--;
        if (!(current & (1 << PC3)) && gallina_x < 7) gallina_x++;

        last_state = current;
    }
}
```

4. Definición de niveles – cargar_nivel()

Define las posiciones y direcciones de los obstáculos para cada nivel. Cada obstáculo se representa como una secuencia de 8 bits encendidos o apagados, desplazados horizontalmente.

Código:

```
void cargar_nivel(uint8_t n) {

    for (int i = 0; i < 8; i++) {
        obstaculos[i] = 0;
        direcciones[i] = 0;
    }

    if (n == 1) {
        obstaculos[2] = 0b10001010;
        obstaculos[4] = 0b11001100;
        obstaculos[6] = 0b11110001;
        direcciones[2] = 1;
        direcciones[4] = -1;
        direcciones[6] = 1;
    } else if (n == 2) {
```

```
        obstaculos[1] = 0b10001001;
        obstaculos[3] = 0b01100110;
        obstaculos[5] = 0b11100011;
        obstaculos[6] = 0b10100010;
        direcciones[1] = -1;
        direcciones[3] = 1;
        direcciones[5] = -1;
        direcciones[6] = 1;
    } else if (n == 3) {
        obstaculos[0] = 0b10000001;
        obstaculos[1] = 0b01000010;
        obstaculos[2] = 0b00011000;
        obstaculos[3] = 0b00111100;
        obstaculos[4] = 0b00000000;
        obstaculos[5] = 0b11100111;
        obstaculos[6] = 0b00011000;
        direcciones[0] = 1;
        direcciones[1] = -1;
        direcciones[2] = 1;
        direcciones[3] = -1;
        direcciones[4] = 1;
        direcciones[5] = -1;
        direcciones[6] = 1;
    }
}
```

5. Movimiento de obstáculos – mover_obstaculos()

Los obstáculos se desplazan hacia la izquierda o la derecha usando desplazamientos circulares de bits según su dirección asignada:

Código:

```
void mover_obstaculos() {
    for (int i = 0; i < 8; i++) {
        if (direcciones[i] == 1)
            obstaculos[i] = (obstaculos[i] << 1) | (obstaculos[i] >>
7);
        else if (direcciones[i] == -1)
            obstaculos[i] = (obstaculos[i] >> 1) | (obstaculos[i] <<
7);
    }
}
```

6. Visualización en la matriz LED – mostrar()

La función muestra en la matriz la combinación de obstáculos y la gallina, actualizando fila por fila. Si la gallina está en una fila, se añade su posición al patrón de esa fila.

Código:

```
void mostrar() {
    for (uint8_t j = 0; j < 8; j++) {
        PORTD = 1 << j;
        uint8_t fila = obstaculos[j];
        if (j == gallina_y) fila |= (1 << gallina_x);
        PORTB = ~fila;
        _delay_ms(0.5);
    }
}
```

7. Detección de colisión – colision()

Detecta si la gallina ha colisionado con un obstáculo. Devuelve verdadero si ambos coinciden.

Código:

```
uint8_t colision() {
    return (obstaculos[gallina_y] & (1 << gallina_x));
}
```

8. Reinicio de nivel – reiniciar()

Posiciona la gallina en el punto de inicio y presenta el número del nivel en pantalla. También actualiza los obstáculos correspondientes.

Código:

```
void reiniciar() {
    gallina_x = 3;
    gallina_y = 7;
    if (nivel == 1) mostrar_letra(L1, 400);
    if (nivel == 2) mostrar_letra(L2, 400);
    if (nivel == 3) mostrar_letra(L3, 400);
    cargar_nivel(nivel);
}
```

9. Comunicación con el PIC – señales por PORTC

El ATmega328P emite señales lógicas a través de los pines PC4 y PC5, los cuales están conectados a RB1 y RB0 del PIC16F887. Las combinaciones indican:

Inicio: PC4 = 1, PC5 = 1

errota: PC4 = 0, PC5 = 1

Victoria: PC4 = 1, PC5 = 0

10. Victoria del juego – victoria_final()

Si el jugador completa los tres niveles, se encienden todos los LEDs en un patrón intermitente y se reinicia el juego.

Código:

```
void victoria_final() {
    for (int i = 0; i < 5; i++) {
        PORTD = 0xFF;
        PORTB = 0x00;
        _delay_ms(100);
        PORTD = 0x00;
        PORTB = 0xFF;
        _delay_ms(200);
    }
    nivel = 1;
    PORTC |= (1 << PC4);     // PC4 = 1
    PORTC &= ~(1 << PC5);    // PC5 = 0
    _delay_ms(100);
    PORTC &= ~((1 << PC4) | (1 << PC5));

    reiniciar();
}
```

11. Control de tiempo en nivel 3 – controlar_tiempo_nivel3()

En el nivel 3 se cuenta el tiempo de juego. Si se excede un umbral (aproximadamente 15 segundos), se reinicia al nivel 1.

Código:

```
void controlar_tiempo_nivel3() {
    static uint16_t tiempo_nivel3 = 0;

    if (nivel == 3) {
        tiempo_nivel3++;
        if (tiempo_nivel3 >= 200) {
            nivel = 1;
            reiniciar();              // Reinicia el juego
            tiempo_nivel3 = 0;
        }
    } else {
        tiempo_nivel3 = 0;  // Si sale del nivel 3, reinicia el
contador
    }
}
```

12. Función principal – main()

Se configuran los puertos y se inicia el sistema. En el bucle principal se realizan las siguientes operaciones:

- Leer los pulsadores
- Verificar colisiones
- Detectar victoria
- Mostrar matriz
- Controlar tiempo en nivel 3
- Desplazar obstáculos

Código:

```
int main() {
    DDRD = 0xFF; DDRB = 0xFF;
    DDRC = 0x00; PORTC = 0x0F;

    DDRC |= (1 << PC4) | (1 << PC5); // PC4 y PC5 como salidas
    PORTC &= ~((1 << PC4) | (1 << PC5)); // Inicialmente en 0

    // sonido de inicio
    PORTC |= (1 << PC4) | (1 << PC5);

    _delay_ms(10);            // Pulso

    PORTC &= ~((1 << PC4) | (1 << PC5)); // Apaga la señal

    _delay_ms(200);
    reiniciar();

    uint8_t contador_obs = 0;

    while (1) {
        leer_botones();


        // Detecta la colision, manda senal de sonido de colision y
rinicia nivel.
        if (colision()) {

            PORTC &= ~(1 << PC4);    // PC4 = 0 (RB1)
            PORTC |= (1 << PC5);     // PC5 = 1 (RB0)
            _delay_ms(10);          // Pulso breve
            PORTC &= ~((1 << PC4) | (1 << PC5)); // Apaga señal

            nivel = 1;
            reiniciar();
        }
```

```
        // Victoria por nivel
        if (gallina_y == 0) {

            PORTC |= (1 << PC4);     // PC4 = 1 (RB1)
            PORTC &= ~(1 << PC5);    // PC5 = 0 (RB0)
            _delay_ms(10);
            PORTC &= ~((1 << PC4) | (1 << PC5));

            if (nivel < 3) {
                nivel++;
                reiniciar();
            } else {
                victoria_final();
            }
        }


        for (int i = 0; i < 4; i++) {

            mostrar();

            controlar_tiempo_nivel3();
        }


        // Mueve los obstaculos

        contador_obs++;
        if (contador_obs >= 10) {
            mover_obstaculos();
            contador_obs = 0;
        }
    }
```

6. Descripción del Código PIC16F887

El PIC se encarga de reproducir sonidos a partir de señales enviadas por el ATmega328P. Dependiendo del valor de los bits RB0 y RB1, reproduce melodías para inicio, victoria o derrota. El canal de sonido está en PORTC.F3.

1) Inicialización de configuración digital

Al inicio del programa, se desactivan los módulos de funciones analógicas y comparadores internos, para que los pines se comporten como entradas/salidas digitales.

Código:

```
    ANSEL  = 0;
    ANSELH = 0;
    C1ON_bit = 0;
```

```
    C2ON_bit = 0;

    OPTION_REG.F7 = 0;
    TRISB.F0 = 1;
    TRISB.F1 = 1;
```

Esto garantiza que los pines RB0 y RB1 puedan recibir correctamente las señales lógicas provenientes del ATmega328P.

Los pines RB0 y RB1 se configuran como entradas, ya que serán las líneas receptoras del canal paralelo de 2 bits.

Luego, se inicializa el módulo de sonido, especificando que el buzzer está conectado a RC3:

```
    Sound_Init(&PORTC, 3);
```

2) Funciones de melodías

Se definen tres funciones que reproducen una secuencia de tonos en diferentes frecuencias, asociadas a los eventos clave del juego.

Código:

```
void SonidoInicio() {
    Sound_Play(440, 120); Delay_ms(50);
    Sound_Play(523, 120); Delay_ms(50);
    Sound_Play(659, 120); Delay_ms(50);
    Sound_Play(880, 100); Delay_ms(30);
    Sound_Play(1046, 180); Delay_ms(100);
}

void SonidoVictoria() {
    Sound_Play(784, 150); Delay_ms(50);
    Sound_Play(880, 150); Delay_ms(50);
    Sound_Play(988, 200); Delay_ms(50);
    Sound_Play(1046, 300); Delay_ms(100);
    Sound_Play(1318, 200); Delay_ms(200);
}

void SonidoDerrota() {
    Sound_Play(988, 100); Delay_ms(50);
    Sound_Play(784, 100); Delay_ms(50);
    Sound_Play(523, 100); Delay_ms(50);
    Sound_Play(349, 150); Delay_ms(50);
    Sound_Play(196, 300); Delay_ms(100);
}
```

Cada frecuencia representa una nota musical (en Hz) y el segundo parámetro representa la duración del tono en milisegundos.

3) Lógica de interpretación de señales

Dentro del bucle principal del programa (while(1)), el PIC16F887 evalúa continuamente el estado de los dos bits menos significativos de PORTB. Estas líneas están conectadas a PC5 y PC4 del ATmega328P, formando así un canal de comunicación paralela de 2 bits.

Código:

```
while (1) {
        unsigned char valor = (PORTB & 0b00000011);

        switch (valor) {
            case 0b01: SonidoVictoria(); break;
            case 0b11: SonidoInicio(); break;
            case 0b10: SonidoDerrota(); break;
        }
```

Este sistema de codificación lógica permite al PIC reaccionar con precisión a los eventos del juego:

11 (Inicio): se reproduce una melodía de bienvenida.

10 (Derrota): se emite una melodía que indica colisión o fallo.

01 (Victoria): se toca una melodía alegre por superar un nivel o completar el juego.

7. Esquema de Comunicación entre Microcontroladores

El ATmega328P envía combinaciones lógicas a los pines PC4 y PC5 que son leídas por el PIC16F887 en RB0 y RB1. Según la combinación, el PIC reproduce el sonido adecuado. Esto se realiza mediante señales de pulsos breves tras eventos clave (inicio, colisión, victoria).

8. Repositorio del Proyecto

Link del Repositorio en GitHub https://github.com/TheWillyXD/Tarea-3-Grupo-2.git

9. Conclusiones

- La implementación de sistemas embebidos permite desarrollar videojuegos funcionales e interactivos con componentes sencillos.
- La comunicación entre microcontroladores mediante un canal paralelo resultó efectiva para coordinar acciones visuales y sonoras.
- El juego ofrece una experiencia completa al usuario con niveles progresivos, control físico intuitivo y retroalimentación auditiva.

10. Recomendaciones

- Añadir un sistema de puntuación para mejorar la competitividad del juego.
- Considerar el uso de pantallas OLED para mayor claridad visual.
- Explorar la integración de almacenamiento externo para guardar niveles o puntajes.