ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

SNAKE GAME

Sistemas Embebidos

Tarea: 03

Docente: Ing. Ronald Solís

Integrantes:

David Acosta Lara

Christopher Tomalá

Franklin Pelaez Montero

PAO I - 2025

Contenido

1. Objetivo General

Desarrollar e implementar Snake 8 × 8, un sistema de juego interactivo de tres niveles de dificultad que combine salidas visuales mediante una matriz LED 8 × 8 controlada por el microcontrolador ATmega328P, salidas auditivas mediante un sistema de reproducción de melodías con el microcontrolador PIC16F887 y entradas mediante pulsadores, empleando comunicación entre ambos dispositivos, con el fin de reforzar habilidades en programación y control de sistemas embebidos.

2. Objetivos Específicos

Diseñar y programar, en lenguaje C, el multiplexado de la matriz LED 8 × 8 con el ATmega328P para representar la serpiente, las manzanas y la indicación breve del nivel (L1, L2, L3).

Implementar la lógica de movimiento, la detección de colisiones con manzanas y el cambio automático de nivel, ajustando la velocidad y la cantidad de manzanas para los modos fácil, medio y difícil.

Programar, en MikroC, el PIC16F887 para reproducir melodías diferenciadas a partir de eventos recibidos desde el ATmega328P.

3. Descripción Técnica Completa Del Juego

Visión General

El proyecto Snake 8 × 8 implementa el clásico juego de la serpiente en una matriz LED de 8 × 8, combinando salidas visuales y auditivas en tiempo real. El microcontrolador ATmega328P gobierna la lógica del juego y el refresco de la matriz, mientras que un PIC16F887 reproduce melodías asociadas a los eventos clave. Ambos dispositivos intercambian información mediante un bus paralelo de ocho bits, lo que garantiza una latencia imperceptible entre la acción visual y la respuesta sonora. El

usuario interactúa a través de cinco pulsadores: cuatro controlan la dirección de la serpiente y uno actúa como botón de inicio / reinicio.

Hardware

Matriz LED y Multiplexado

La matriz LED está cableada de modo que las columnas comparten cátodo y las filas comparten ánodo. El ATmega328P activa una única columna a la vez y, simultáneamente, selecciona las filas que deben iluminarse, creando cada fotograma de forma secuencial. Con un tiempo de activación de microsegundos por fila, la pantalla se refresca a más de 100 Hz, eliminando cualquier parpadeo visible.

Pulsadores y Anti-rebote

Cinco pulsadores se conectan en configuración pull-down externa. Un filtro de software permite que la señal sea considerada válida solo si permanece estable durante varios milisegundos consecutivos, lo que impide que los rebotes eléctricos afecten la jugabilidad.

Comunicación entre Microcontroladores

El sistema de audio del juego se basa en la comunicación entre el ATmega y el PIC. El ATmega está configurado para enviar pulsos a través de un pin de salida, que se conecta al pin de entrada RD5 del PIC. Este enlace permite que el PIC interprete señales de eventos importantes como comer una manzana o el fin del juego. Al recibir estas señales, el PIC inicia la reproducción de la melodía correspondiente, asegurando una respuesta inmediata sin interrumpir la lógica visual gestionada por el ATmega.

Subsistema de Audio

El subsistema de audio utiliza el pin RC3 del PIC como salida conectada a un altavoz para la reproducción de sonido. La música de fondo se reproduce continuamente, configurada a través de la función musica_fondo(). Cuando el PIC detecta un pulso que indica un evento, como comer una manzana, interrumpe la música para tocar una nota "Re" durante medio segundo. Del mismo modo, al finalizar el juego, se reproduce una nota "Sol", proporcionando retroalimentación acústica sobre el estado del juego.

Software — ATmega328P

Estructura General

El programa se organiza en tres fases: inicialización, espera de la orden de inicio y ciclo principal. Durante la inicialización se configuran los puertos, se encienden los pull-ups y se definen los valores iniciales de la serpiente y las manzanas. A continuación, el sistema permanece inactivo hasta que el usuario pulse el botón Start.

Ciclo Principal

Cada iteración del juego sigue la secuencia:

Refresco visual: La serpiente y las manzanas se dibujan en la matriz LED mediante multiplexado. Lectura de entradas. Los pulsadores se monitorizan para ajustar la dirección.

Actualización de estado: Transcurrido el intervalo programado, la serpiente avanza una casilla, se verifica la captura de manzanas y se evalúa el avance de nivel o el fin de juego.

Comunicación de eventos: Al producirse un cambio de nivel o el final, se envía el código correspondiente al PIC16F887, que reproduce la melodía asociada.

Software — PIC16F887

Estructura General

El programa desactiva las entradas y comparadores no necesarios para centrarse en señales digitales. Configura el puerto D para detectar si se presiona algún botón y el puerto C para enviar sonidos desde el pin 3. De este modo, está preparado para manejar las entradas y producir sonidos según las acciones del jugador.

Ciclo Principal

Revisión Continua:

- Verifica si el botón (D5) está presionado.
- Si está presionado: Reproduce tocar_notas_manzana.
- Si no está presionado: Ejecuta continuamente musica_fondo.

Detalle de los Niveles

Nivel 1 — Fácil

Dos manzanas aparecen en posiciones predeterminadas y la serpiente se desplaza a velocidad baja. Este escenario permite al jugador familiarizarse con la mecánica de control. Al iniciar, el ATmega328P envía el código correspondiente; el PIC responde con la melodía de bienvenida.

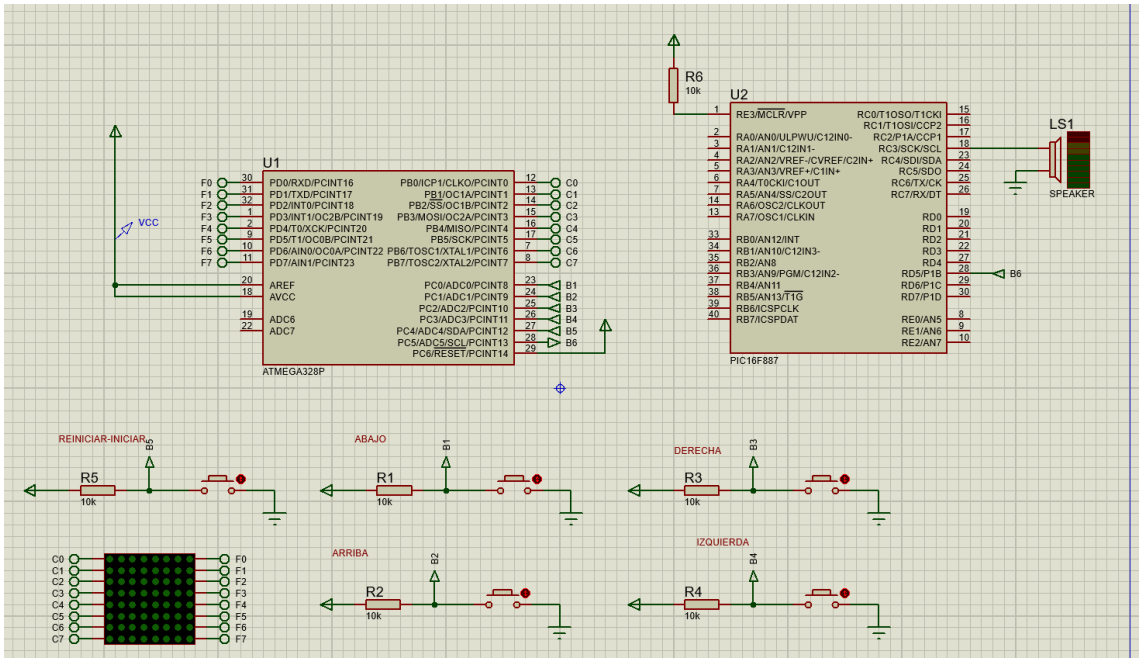
Nivel 2 — Medio

Al consumir las dos manzanas iniciales, el juego incrementa la velocidad al doble y añade una tercera manzana. Un breve indicativo visual (“2”) se muestra en la matriz para informar el cambio; simultáneamente, el PIC reproduce una melodía ascendente que refuerza la sensación de progreso.

Nivel 3 — Difícil

El nivel final introduce una cuarta manzana y cuadruplica la velocidad inicial. El reto se centra en la capacidad del jugador para anticipar movimientos en un entorno acelerado. Al terminar de comer las cuatro manzanas, la matriz enciende los 64 LEDs y el PIC ejecuta la melodía de victoria, concluyendo la partida.

4. Simulación Proteus



5. Explicación del Código

ATMEGA328P

```
/*
 * Deber3.c
 *
 * Created: 09/06/2025 20:45:15
 * Author : aleja - Franklin - Cristopher
 */

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

// Constantes de dirección
#define DIR_DERECHA 0
#define DIR_ABAJO 1
#define DIR_IZQUIERDA 2
#define DIR_ARRIBA 3
```

El #define F_CPU 8000000UL fija la frecuencia del oscilador interno a 8 MHz; las rutinas de delay*() usan este valor para calcular cuántos ciclos NOP ejecutar. Los encabezados <avr/io.h> y <util/delay.h> habilitan el acceso a los registros de E/S de bajo nivel y a los retardos por software. Finalmente se crean cuatro macros con los valores 0-3, suficientes para indexar un switch de movimiento sin usar comparaciones de texto ni enum complejos.

```
18    // Posición inicial serpiente
19    uint8_t cabeza_x = 2, cabeza_y = 2;
20    uint8_t cuerpo_x = 1, cuerpo_y = 2;
21    uint8_t cola_x = 0, cola_y = 2;
22    uint8_t direccion = DIR_DERECHA;
23
24    // Posición manzanas
25    uint8_t manzana1_x = 5, manzana1_y = 5;
26    uint8_t manzana2_x = 7, manzana2_y = 7;
27    uint8_t manzana3_x = 0, manzana3_y = 0;
28    uint8_t manzana4_x = 0, manzana4_y = 0;  // Nueva manzana para nivel 3
29    uint8_t manzana1_estado = 1;
30    uint8_t manzana2_estado = 1;
31    uint8_t manzana3_estado = 0;
32    uint8_t manzana4_estado = 0;  // Estado inicial inactivo
33
34    // Velocidad del juego
35    uint8_t tiempo = 25;
36
37    // Variables control niveles
38    uint8_t nivel_actual = 1;
39
```

Las coordenadas de cabeza, cuerpo y cola se almacenan en registros de 8 bits: solo se necesitan 3 bits para el rango 0-7, pero usar uint8_t evita operaciones de máscara. Cada manzana dispone de su par (x, y) más un flag de validez (1 = visible). En la práctica solo dos manzanas están activas en el nivel 1; la tercera y cuarta se habilitan en los niveles 2 y 3. tiempo actúa como pre-escalador de fotogramas (25 → 12 → 6), regulando la velocidad de avance sin modificar los retardos fijos internos. nivel_actual mantiene el contexto para decidir qué transición ejecutar.

```
// ========================
// Mostrar número de nivel
// ========================
void mostrar_numero_nivel(uint8_t nivel) {
        // '1'
        const uint8_t digito1[8] = {0x10, 0x18, 0x10, 0x10, 0x10, 0x10, 0x7C, 0x00};
        // '2'
        const uint8_t digito2[8] = {0x38, 0x44, 0x40, 0x20, 0x18, 0x04, 0x7C, 0x00};
        // '3'
        const uint8_t digito3[8] = {0x38, 0x44, 0x40, 0x38, 0x40, 0x44, 0x38, 0x00};
        const uint8_t *digito;

        switch (nivel) {
                case 1: digito = digito1; break;
                case 2: digito = digito2; break;
                case 3: digito = digito3; break;
                default: digito = digito1; break;
        }

        for (uint16_t t = 0; t < 50; t++) {
                for (uint8_t fila = 0; fila < 8; fila++) {
                        PORTD = (1 << fila);
                        PORTB = ~digito[fila];
                        _delay_us(300);
                }
        }
        PORTB = 0xFF;
        PORTD = 0x00;
}
```

Tres bit-maps de 8 bytes representan los dígitos 1-3. El doble bucle escanea las filas 50 veces; cada fila permanece 300 µs encendida, resultando en ≈ 417 Hz de refresco por fila y 0,5 s totales en pantalla. La lógica activo-bajo obliga a invertir () el byte antes de enviarlo a PORTB. Al finalizar se ponen todos los cátodos a ‘1’ y todos los ánodos a ‘0’ para borrar la matriz.

```
// ========================
// Reiniciar juego (reset a valores iniciales)
// ========================
void reiniciar_juego(void) {
    cabeza_x = 2; cabeza_y = 2;
    cuerpo_x = 1; cuerpo_y = 2;
    cola_x = 0; cola_y = 2;
    direccion = DIR_DERECHA;
    manzana1_x = 5; manzana1_y = 5; manzana1_estado = 1;
    manzana2_x = 7; manzana2_y = 7; manzana2_estado = 1;
    manzana3_x = 0; manzana3_y = 0; manzana3_estado = 0;
    manzana4_x = 0; manzana4_y = 0; manzana4_estado = 0;
    tiempo = 25;
    nivel_actual = 1;
}

void nivel_2(void) {
    mostrar_numero_nivel(2);
    tiempo = tiempo / 2;
    manzana1_x = 2; manzana1_y = 1; manzana1_estado = 1;
    manzana2_x = 6; manzana2_y = 4; manzana2_estado = 1;
    manzana3_x = 3; manzana3_y = 7; manzana3_estado = 1;
    nivel_actual = 2;
}

void nivel_3(void) {
    mostrar_numero_nivel(3);
    tiempo = tiempo / 2;
    manzana1_x = 1; manzana1_y = 3; manzana1_estado = 1;
    manzana2_x = 4; manzana2_y = 6; manzana2_estado = 1;
    manzana3_x = 7; manzana3_y = 1; manzana3_estado = 1;
    manzana4_x = 5; manzana4_y = 2; manzana4_estado = 1;
    nivel_actual = 3;
}
```

reiniciar_juego() restaura el escenario base (dos manzanas activas, velocidad lenta). Las rutinas de nivel 2 y nivel 3 muestran el dígito informativo, dividen tiempo con un shift lógico (>>=1) —más eficiente que la división— y activan nuevas manzanas. De este modo la velocidad se multiplica por dos en cada transición sin alterar la constante de retardo de la matriz.

```
105     void mostrar_serpiente(void) {
106         for (uint8_t i = 0; i < 3; i++) {
107             PORTD = (1 << cabeza_y);
108             PORTB = ~(1 << cabeza_x);
109             _delay_us(300);
110             PORTD = (1 << cuerpo_y);
111             PORTB = ~(1 << cuerpo_x);
112             _delay_us(300);
113             PORTD = (1 << cola_y);
114             PORTB = ~(1 << cola_x);
115             _delay_us(300);
116             if (manzana1_estado) {
117                 PORTD = (1 << manzana1_y);
118                 PORTB = ~(1 << manzana1_x);
119                 _delay_us(300);
120             }
121             if (manzana2_estado) {
122                 PORTD = (1 << manzana2_y);
123                 PORTB = ~(1 << manzana2_x);
124                 _delay_us(300);
125             }
126             if (manzana3_estado) {
127                 PORTD = (1 << manzana3_y);
128                 PORTB = ~(1 << manzana3_x);
129                 _delay_us(300);
130             }
131             if (manzana4_estado) {
132                 PORTD = (1 << manzana4_y);
133                 PORTB = ~(1 << manzana4_x);
134                 _delay_us(300);
135             }
136         }
137     }
138
```

La función multiplexa la matriz columna-por-columna durante tres ciclos para equilibrar el brillo de todos los LED. Cada segmento visible se enciende 300 µs por pasada, resultando en un refresco global > 100 Hz.

```
160     uint8_t leer_botones(void) {
161         static uint8_t estado_anterior = 0x0F;
162         static uint8_t contador_debounce = 0;
163         uint8_t estado_actual = PINC & 0x0F;
164         uint8_t botones_presionados = (~estado_actual) & estado_anterior;
165         if (botones_presionados) {
166             contador_debounce++;
167             if (contador_debounce > 2) {
168                 contador_debounce = 0;
169                 estado_anterior = estado_actual;
170                 return botones_presionados;
171             }
172         } else {
173             contador_debounce = 0;
174             estado_anterior = estado_actual;
175         }
176         return 0;
177     }
178 
179     void cambiar_direccion(uint8_t nueva_direccion) {
180         switch (nueva_direccion) {
181             case DIR_DERECHA: if (direccion != DIR_IZQUIERDA) direccion = DIR_DERECHA; break;
182             case DIR_IZQUIERDA: if (direccion != DIR_DERECHA) direccion = DIR_IZQUIERDA; break;
183             case DIR_ARRIBA: if (direccion != DIR_ABAJO) direccion = DIR_ARRIBA; break;
184             case DIR_ABAJO: if (direccion != DIR_ARRIBA) direccion = DIR_ABAJO; break;
185         }
186     }
187 
```

Lee PC0-PC3, detecta flancos de bajada (botón presionado) y exige tres lecturas consecutivas para aceptación, bloqueando rebotes de ≈ 15 ms sin hardware externo. Retorna un bitmask con los botones recién pulsados.

En cambiar_direccion solo aplica la nueva dirección si no es la opuesta inmediata, evitando giros de 180° que autocolisionarían la serpiente y simplificando la lógica de juego.

```
void todos_los_leds_on(void) {
    // Prender todos los LEDs de la matriz para siempre (hasta reinicio)
    while (1) {
        PORTB = 0x00; // Todas las columnas activas (cátodo común)
        PORTD = 0xFF; // Todas las filas activas
        // Si presionan el botón de reinicio (PC4), sale del ciclo y reinicia el juego
        if ((PINC & (1 << 4)) == 0) {
            _delay_ms(80);
            while ((PINC & (1 << 4)) == 0) {
                // Mantener encendido hasta soltar
            }
            _delay_ms(80);
            break;
        }
    }
    PORTB = 0xFF;
    PORTD = 0x00;
}
```

Enciende permanentemente los 64 LED para indicar victoria y permanece en bucle hasta que el jugador mantiene presionado Start/Reset.

```
207  int main(void) {
208      DDRB = 0xFF;
209      DDRD = 0xFF;
210      DDRC = 0x00;
211      PORTC = 0x1F; // PC0-PC4 pull-up
212
213      // ---- Esperar inicio (PC4 presionado), mantén la matriz apagada ----
214      PORTB = 0xFF;
215      PORTD = 0x00;
216      while ((PINC & (1 << 4)) != 0) {
217          // No hagas nada, todo apagado hasta presionar iniciar
218      }
219      _delay_ms(80); // Antirebote
220      while ((PINC & (1 << 4)) == 0); // Esperar que lo suelten
221      _delay_ms(80);
222
223      reiniciar_juego();
224      mostrar_numero_nivel(1);
225
226      uint16_t contador = 0;
227      uint8_t cambiar_nivel = 0;
228
229      while (1) {
230          // --- Botón PC4 para reiniciar ---
231          if ((PINC & (1 << 4)) == 0) {
232              _delay_ms(80);
233              while ((PINC & (1 << 4)) == 0) {
234                  // No muestres nada mientras está presionado para reinicio
235                  PORTB = 0xFF;
236                  PORTD = 0x00;
237              }
238              _delay_ms(80);
239              reiniciar_juego();
240              mostrar_numero_nivel(1);
241              contador = 0;
242              cambiar_nivel = 0;
243              continue;
244          }
```

Al inicio se configura la matriz como salida, y los pulsadores como entradas. El bucle While gestiona:

1. Reinicio si Start es presionado.
2. Transición de nivel cuando cambiar nivel tiene la flag = 1
3. Framew: refresca, lee botones con anti-rebote, cambia dirección, incrementa el contador y mueve la serpiente cuando contador == tiempo. Comprueba si todas las manzanas del nivel están comidas para pasar al siguiente nivel o llamar a la función todos_los_leds_on () al finalizar el nivel 3

PIC16F887

```
void playNote(unsigned int frequency, unsigned int duration) {
    Sound_Play(frequency, duration / 10);
    Delay_ms(25); // Pausa reducida
}

void musica_fondo() {
    // Escala: Do Re Mi Fa Sol
    playNote(523, 50);  // Do
    playNote(587, 50);  // Re
    playNote(659, 50);  // Mi
    playNote(698, 50);  // Fa
    playNote(784, 50);  // Sol
}

void tocar_notas_manzana() {

    Sound_Play(587, 100);
}

void musica_fin_juego() {

    Sound_Play(784, 1000);
}

void main() {
```

Se define varias funciones para reproducir sonidos. La función principal, playNote, toma una frecuencia y duración para generar un sonido breve. musica_fondo reproduce una serie de notas para crear una pequeña melodía. tocar_notas_manzana y musica_fin_juego reproducen notas específicas, probablemente para eventos particulares en un juego, como recolectar un objeto o terminar una partida.

```
void main() {
    ANSEL = 0;
    ANSELH = 0;
    C1ON_bit = 0;
    C2ON_bit = 0;

    TRISD = 0xFF;
    TRISC = 0xFB;

    Sound_Init(&PORTC, 3);

    while (1) {
        if (PORTD.F5 == 1) {
            Delay_ms(50);
            while (PORTD.F5 == 1);
            tocar_notas_manzana();
            Delay_ms(100);
        } else {
            musica_fondo();
        }
    }
}
```

Se configura el microcontrolador apagando las entradas analógicas y comparadores, y estableciendo el puerto D como entrada y el puerto C como salida para el sonido. Se inicializa el sonido en el pin 3 del puerto C. En el bucle principal, se comprueba continuamente si un botón conectado al puerto D está presionado. Si es así, se reproduce una nota específica mientras el botón esté presionado. De lo contrario, se reproduce una melodía de fondo. Esto permite responder a las interacciones del usuario de manera dinámica.

6. Enlace al Repositorio de GitHub

https://github.com/ChristopherTomala02/Comunicacion-entre-Microcontroladores

7. Conclusiones

El proyecto cumple el objetivo al combinar una matriz LED 8 × 8 controlada por el ATmega328P, la generación de melodías en el PIC16F887 y la lectura de pulsadores. La comunicación paralela permite que cada evento visual (inicio, cambio de nivel, fin de juego) dispare la melodía correspondiente, demostrando la sincronización entre el control de matriz, la reproducción de audio y enlace entre microcontroladores.

La reducción programada de la variable tiempo (25 → 12 → 6) y la activación progresiva de manzanas logran tres niveles diferenciados que incrementan la complejidad de forma lineal, satisfaciendo el objetivo de implementar modos fácil, medio y difícil. Las pruebas en Proteus confirman que la lógica de movimiento, la gestión de niveles y la pantalla de victoria funcionan con estabilidad, lo que valida todos los objetivos específicos de diseño, programación y simulación.

8. Recomendaciones

Se aconseja implementar una matriz de ocupación de 8 × 8 (bit-array) o una cola circular de segmentos para registrar las casillas ocupadas por la serpiente. En cada avance bastaría comparar la nueva posición de la cabeza con el registro correspondiente; si coincide con un segmento activo, se dispara un estado de “derrota”, añadiendo profundidad al reto sin alterar la rutina de multiplexado existente.

Reemplazar el contador por software con un temporizador basado en interrupciones (por ejemplo, TIMER1 overflow cada X ms) permitiría que el refresco de la matriz y la

lógica de juego se ejecuten en tareas separadas y no dependan del mismo lazo ocupado, mejorando la precisión.