ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y COMPUTACIÓN

Sistemas Embebidos

TAREA #3

Docente: Ronald Solis

Jean Carlos Aguilar 202108643
Cristina Albán 202200606
Jaime Escala 202107678
Yancrist Pazmiño 202108635

Paralelo: 2

19 – Junio – 2025

TÉRMINO I PAO 2025 – 2026

Índice

Objetivos:

Objetivo general

Diseñar un sistema de juego interactivo basado en el microcontrolador ATMEGA328P que permita ejecutar, en una matriz LED 8x8, el juego “Snake”. La cual será controlada mediante pulsadores y tendrá implementada una lógica de niveles, generación de “frutas” y detección de choques, además de emitir señales auditivas de victoria o derrota. Todo esto con el fin de reforzar los conocimientos en lenguaje C, el manejo de periféricos digitales y el diseño de sistemas embebidos.

Objetivos específicos

1. Diseñar e implementar la lógica digital del juego “Snake” en el lenguaje C para poder adaptar su funcionalidad al microcontrolador del ATMEGA328P, sin dejar perder su estructura original.
2. Controlar dinámicamente la matriz LED 8x8 mediante el multiplexado de los puertos digitales del microcontrolador para emitir las señales visuales.
3. Gestionar la detección de choques y victorias para el correcto avance, finalización o aumento de dificultad del juego.
4. Emitir señales auditivas mediante el microcontrolador PIC16F887 que permitan la indicación de eventos clases como inicio, victoria y derrota.

Descripción técnica:

Funciones:

enviar_senal_ganar(): Envia una señal por los pines PC4 del puerto C para indicar que el jugador ha ganado. Esto se realiza configurando los pines correspondientes, activando el PC4 y luego esperando 100 ms antes de apagar la señal.

enviar_senal_perder(): Envia una señal por los pines PC5 del puerto C para indicar que el jugador ha perdido. Similar a la función anterior, se configura el PC5, se espera 100 ms y luego se apagan ambas señales.

enviar_senal_inicio(): Envia una señal inicial de inicio, donde los pines PC4 y PC5 se activan por un breve periodo (300 ms) al inicio del juego para indicar que se está comenzando. Después de 50 ms, se apagan.

limpiarPantalla(): Limpia el arreglo pantalla de la matriz 8x8, poniendo todos sus valores a 0. Esto se realiza inicializando todos los valores de la pantalla a 0.

generar_frutas(uint8_t cantidad): Genera frutas en ubicaciones aleatorias dentro de la matriz. La cantidad de frutas generadas depende del parámetro cantidad, y la función asigna valores aleatorios a las coordenadas x y y de las frutas activas dentro de los límites definidos de la matriz.

iniciar(int* tamanio): Inicializa el estado del juego. Configura la posición inicial de la serpiente, asigna valores iniciales a las propiedades de la serpiente, y genera las frutas para el nivel actual.

actualizarDatos(int tamanio): Actualiza los datos de la serpiente en la pantalla. Mueve cada segmento de la serpiente y actualiza las posiciones de las frutas en la pantalla, colocando cada uno en las filas y columnas correctas.

direcciones(bool* fin): Lee las entradas del jugador a través de los pines PC0 a PC3, que controlan las direcciones de la serpiente (arriba, abajo, izquierda, derecha). Si el jugador presiona un botón, la dirección de la serpiente cambia. Se asegura que la serpiente no pueda ir en la dirección opuesta a la actual.

moverSerpiente(int* tamanio, bool* fin): Mueve la serpiente de acuerdo con su dirección actual. También verifica si la serpiente ha colisionado con las frutas o las paredes del borde del área de juego. Si la serpiente come una fruta, aumenta de tamaño.

update(int tamanio, bool* fin): Actualiza la posición de la serpiente, mueve los segmentos y coloca la serpiente en la pantalla. Después de mover la serpiente, actualiza el estado de la pantalla con las nuevas posiciones.

pantallaWin(): Muestra la pantalla de "victoria" cuando el jugador gana el juego. Envía la señal de ganar y luego hace que se encienda la matriz para mostrar el mensaje de victoria, en un bucle infinito.

pantallaPerder(): Muestra la pantalla de "perder" cuando el jugador pierde. Envía la señal de perder y luego hace que se encienda la matriz con el mensaje de derrota en un bucle infinito.

Definiciones:

#define F_CPU 8000000UL: Define la frecuencia del reloj del microcontrolador (en este caso, 8 MHz). Esto es necesario para que las funciones de retraso (delay) funcionen correctamente.

#include <avr/io.h>: Incluye la librería estándar para acceder a los registros de entrada y salida del microcontrolador.

#include <util/delay.h>: Incluye la librería que proporciona funciones de retraso (_delay_ms()), que se utilizan en varias funciones para crear retrasos en la ejecución del código.

#include <stdlib.h>: Incluye la librería estándar de C para operaciones como la generación de números aleatorios (rand()).

#include <time.h>: Incluye la librería estándar para manejar funciones de tiempo. Se usa para inicializar el generador de números aleatorios mediante srand(time(NULL)).

#define DELAY 1: Define el valor del retraso (en milisegundos) entre los ciclos de actualización de la pantalla del juego.

#define Y_MAX 8 y #define X_MAX 8: Define las dimensiones máximas de la matriz 8x8 del juego, que tiene un tamaño de 8 filas (Y) y 8 columnas (X).

#define LARGO 64: Define la longitud máxima de la serpiente, que es 64 segmentos.

uint8_t PORT[8] = {1,2,4,8,16,32,64,128};: Define un arreglo de valores que se utilizan para controlar las filas de la pantalla 8x8.

uint8_t pantalla[8] = {0};: Define un arreglo que representa la matriz de la pantalla del juego. Inicialmente está vacío (todos los valores a 0).

uint8_t ganar[8] = {...}; y uint8_t perder[8] = {...};: Define los patrones de píxeles que representan los mensajes de "ganar" y "perder" que se mostrarán en la pantalla.

typedef struct {...} snake;: Define la estructura snake que contiene la información de cada segmento de la serpiente: las coordenadas x e y, los desplazamientos modEnX y modEnY para

determinar la dirección de la serpiente, y un valor booleano on para saber si el segmento está encendido.

typedef struct {...} fruta;: Define la estructura fruta que contiene la información de cada fruta en el juego: las coordenadas x e y y un valor booleano activa que indica si la fruta está disponible.

snake snk[LARGO]; y fruta frt[5];: Declara un arreglo de serpientes snk de tamaño LARGO y un arreglo de frutas frt con un máximo de 5 frutas.

uint8_t puntaje = 0;: Define la variable puntaje para llevar la cuenta de la puntuación del jugador.

bool direccionCambiada = false;: Indica si la dirección de la serpiente ha cambiado. Se usa para evitar cambios de dirección demasiado rápidos.

uint8_t nivel = 1;: Define el nivel actual del juego. El nivel aumenta conforme el jugador avanza en el juego.

const uint8_t frutas_por_nivel[4] = {0, 3, 4, 5};: Define cuántas frutas aparecen en cada nivel del juego.

Simulación:

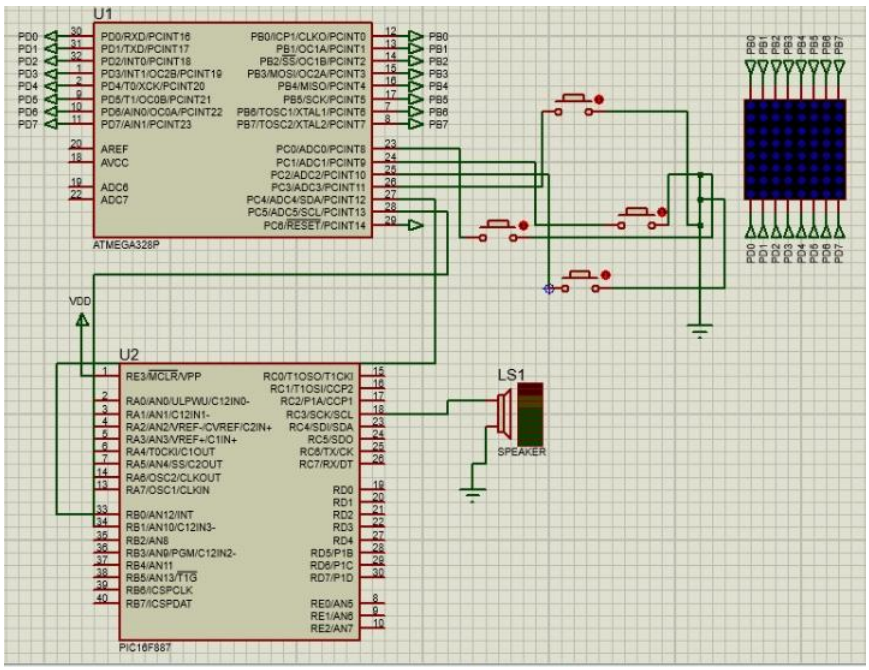

Imagen 1. Simulación en proteus de juego

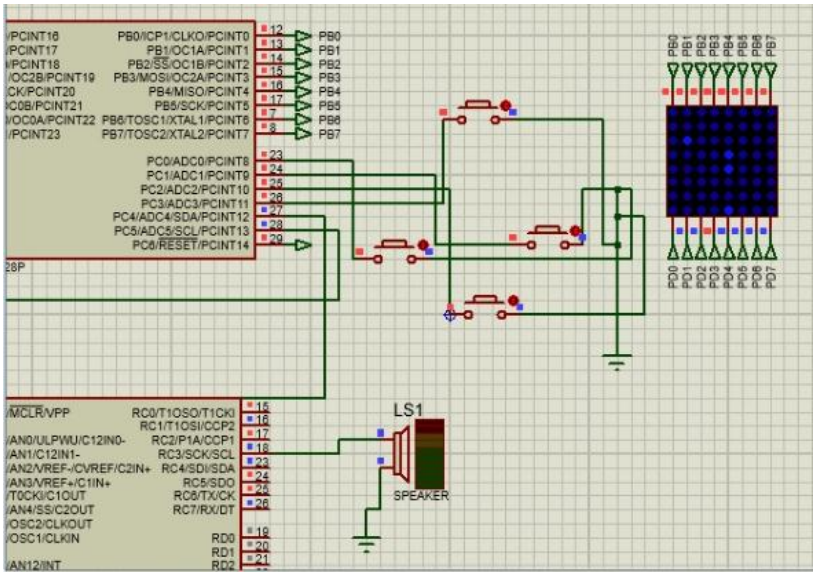

Imagen 2. Simulacion de partida en proceso en Proteus

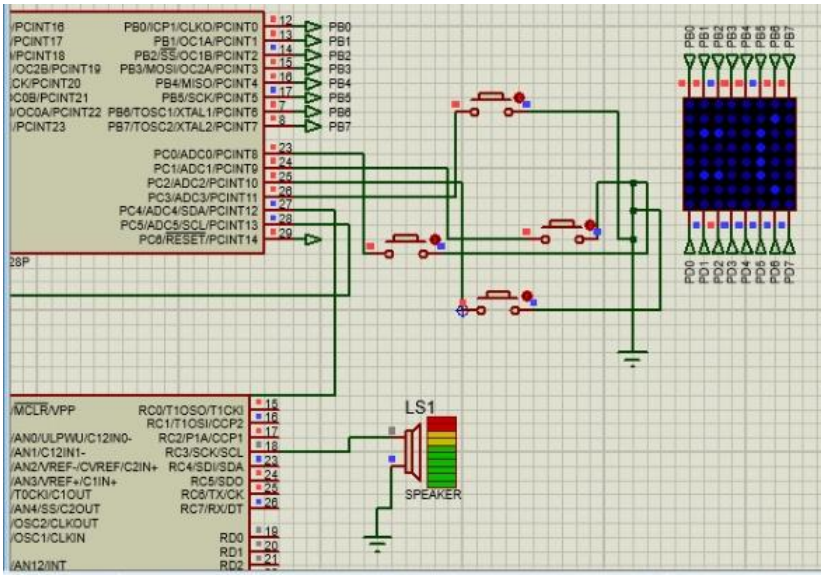

Imagen 3. Dibujo es matriz LED al perder con su canción.

Lógica del juego:

1. Inicialización del nivel
   - Se llama a iniciar(): configura la serpiente con 2 segmentos, dirección inicial a la derecha, y genera frutas.
2. Bucle principal del nivel

- Se hace un refresco constante de la matriz LED mostrando:
- La serpiente.
- Las frutas activas.
- Se lee la dirección desde botones con direcciones().
- Se actualiza la posición con update().
- Se verifica colisiones con moverSerpiente():
- Si choca con paredes: se pierde.
- Si come fruta: crece y suma puntos.

3. Fin del nivel
   - Si el jugador come todas las frutas, se envía señal de ganar y se pasa al siguiente nivel.
   - Si el jugador pierde: se muestra la pantalla de perder en bucle infinito.
   - Victoria total
   - Si se completan los 3 niveles, se muestra un patrón de victoria en bucle infinito.

Código de proyecto:

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

#define DELAY 1
#define Y_MAX 8
#define X_MAX 8
#define LARGO 64

uint8_t PORT[8] = {1,2,4,8,16,32,64,128};
uint8_t pantalla[8] = {0};
uint8_t ganar[8] = {0x00, 0x24, 0x24, 0x00, 0x42, 0x3C, 0x00,
0x00};
uint8_t perder[8] = {0x00, 0x24, 0x24, 0x00, 0x00, 0x3C, 0x42,
0x00};

typedef struct {
 int x, y;
 int modEnX, modEnY;
 bool on;
} snake;

typedef struct {
 int x, y;
 bool activa;
} fruta;

snake snk[LARGO];
fruta frt[5];
uint8_t puntaje = 0;
bool direccionCambiada = false;
uint8_t nivel = 1;
const uint8_t frutas_por_nivel[4] = {0, 3, 4, 5}; // índice 0 no
usado

// Señales por PC4 y PC5
void enviar_senal_ganar() {
 PORTC = (PORTC & ~((1 << PC4) | (1 << PC5))) | (1 << PC4);
 _delay_ms(100);
 PORTC &= ~((1 << PC4) | (1 << PC5));
}

void enviar_senal_perder() {
 PORTC = (PORTC & ~((1 << PC4) | (1 << PC5))) | (1 << PC5);
 _delay_ms(100);
 PORTC &= ~((1 << PC4) | (1 << PC5));
}

// Función corregida para la señal de inicio
void enviar_senal_inicio() {
 // Limpiar PC4 y PC5 primero
 PORTC &= ~((1 << PC4) | (1 << PC5));
 _delay_ms(50);

 // Encender PC4 y PC5 (señal de inicio)
 PORTC |= (1 << PC4) | (1 << PC5);
 _delay_ms(300);

 // Apagar PC4 y PC5
```

```
  PORTC &= ~((1 << PC4) | (1 << PC5));
}

void limpiarPantalla() {
  for (int i = 0; i < Y_MAX; i++) pantalla[i] = 0;
}

void generar_frutas(uint8_t cantidad) {
  for (uint8_t i = 0; i < cantidad; i++) {
    frt[i].x = rand() % (X_MAX - 1);
    frt[i].y = rand() % (Y_MAX - 1);
    frt[i].activa = true;
  }
}

void iniciar(int* tamanio) {
  snk[0].x = 5;
  snk[0].y = 4;
  *tamanio = 2;
  srand(time(NULL));
  for (int i = 0; i < *tamanio; i++) {
    snk[i].modEnX = 1;
    snk[i].modEnY = 0;
    snk[i].on = 1;
  }
  generar_frutas(frutas_por_nivel[nivel]);
}

void actualizarDatos(int tamanio) {
  for (int i = 1; i < tamanio; i++) {
    snk[i].x = snk[i - 1].x - 1;
    snk[i].y = snk[i - 1].y;
    snk[i].on = 1;
  }
  for (int i = 0; i < tamanio; i++) {
    if (snk[i].x - 1 >= 0 && snk[i].x - 1 < X_MAX)
      pantalla[snk[i].y] |= snk[i].on << (7 - (snk[i].x - 1));
  }
  for (int i = 0; i < frutas_por_nivel[nivel]; i++) {
    if (frt[i].activa)
      pantalla[frt[i].y] |= 1 << (7 - frt[i].x);
  }
}

void direcciones(bool* fin) {
  if (direccionCambiada) return;
  if (!(PINC & (1 << PC0))) {
    _delay_ms(20);
    if (!(PINC & (1 << PC0)) && snk[0].modEnY != 1) {
      snk[0].modEnX = 0; snk[0].modEnY = -1;
      direccionCambiada = true;
    }
  } else if (!(PINC & (1 << PC1))) {
    _delay_ms(20);
    if (!(PINC & (1 << PC1)) && snk[0].modEnY != -1) {
      snk[0].modEnX = 0; snk[0].modEnY = 1;
      direccionCambiada = true;
    }
  } else if (!(PINC & (1 << PC2))) {
    _delay_ms(20);
    if (!(PINC & (1 << PC2)) && snk[0].modEnX != 1) {
      snk[0].modEnX = -1; snk[0].modEnY = 0;
      direccionCambiada = true;
    }
  } else if (!(PINC & (1 << PC3))) {
    _delay_ms(20);
    if (!(PINC & (1 << PC3)) && snk[0].modEnX != -1) {
      snk[0].modEnX = 1; snk[0].modEnY = 0;
      direccionCambiada = true;
    }
  }
}

void moverSerpiente(int* tamanio, bool* fin) {
  if (snk[0].x <= 0 || snk[0].x > X_MAX || snk[0].y < 0 || snk[0].y >=
Y_MAX) {
    *fin = 1;
    return;
  }

  for (uint8_t i = 0; i < frutas_por_nivel[nivel]; i++) {
    if (frt[i].activa && (snk[0].x - 1 == frt[i].x) && (snk[0].y ==
frt[i].y)) {
      frt[i].activa = false;
      (*tamanio)++;
      snk[*tamanio - 1].on = 1;
      puntaje++;
    }
  }
}

void update(int tamanio, bool* fin) {
  limpiarPantalla();
  for (int i = tamanio - 1; i > 0; i--) {
    snk[i].x = snk[i - 1].x;
    snk[i].y = snk[i - 1].y;
  }
  snk[0].x += snk[0].modEnX;
  snk[0].y += snk[0].modEnY;

  for (int i = 0; i < tamanio; i++) {
    if (snk[i].x - 1 >= 0 && snk[i].x - 1 < X_MAX)
      pantalla[snk[i].y] |= snk[i].on << (7 - (snk[i].x - 1));
  }

  for (int i = 0; i < frutas_por_nivel[nivel]; i++) {
    if (frt[i].activa)
      pantalla[frt[i].y] |= 1 << (7 - frt[i].x);
  }
}

void pantallaWin() {
  enviar_senal_ganar();
  while (1) {
```

```
  for (int i = 0; i < 8; i++) {
    PORTD = PORT[i];
    PORTB = ~(ganar[i]);
    _delay_ms(DELAY);
  }
 }
}

void pantallaPerder() {
 enviar_senal_perder();
 while (1) {
  for (int i = 0; i < 8; i++) {
    PORTD = PORT[i];
    PORTB = ~(perder[i]);
    _delay_ms(DELAY);
  }
 }
}

int main() {
 DDRB = 0xFF;
 DDRD = 0xFF;
 DDRC = 0b0110000;  // PC4 y PC5 como salida, PC0-PC3
como entrada
 PORTC |= 0x0F;    // Pull-ups en PC0-PC3

 // Señal de inicio solo al arrancar el juego
 enviar_senal_inicio();

 while (nivel <= 3) {
  int tamanio;
  bool fin = 0;
  puntaje = 0;
  direccionCambiada = false;

  iniciar(&tamanio);
  actualizarDatos(tamanio);

  uint8_t contador = 0;

  while (!fin && puntaje < frutas_por_nivel[nivel]) {
   for (int i = 0; i < 8; i++) {
    PORTD = PORT[i];
    PORTB = ~(pantalla[i]);
    _delay_ms(DELAY);
   }

   contador = (contador + 1) % 255;
   direcciones(&fin);

   if (contador % (25 - nivel * 5) == 0) {
    update(tamanio, &fin);
    moverSerpiente(&tamanio, &fin);
    direccionCambiada = false;
   }

   if (fin) {
    limpiarPantalla();
    pantallaPerder();  // Señal + pantalla perder
   }
  }

  // Se comieron todas las manzanas de nivel, enviamos señal
ganar
  enviar_senal_ganar();

  nivel++;

  if (nivel > 3) {
   limpiarPantalla();
   pantallaWin();
  }
 }

 return 0;
}
```

Enlace a repositorio de GitHub:

https://github.com/JaimeEscala/TAREA3EMBEBIDOS.git