espol Facultad de Ingeniería en Electricidad y Computación
Guayaquil – Ecuador
Campus Gustavo Galindo Velasco - Km. 30.5 Vía Perimetral - Pbx: (593-4) 2269 269

UNIVERSIDAD POLITECNICA DEL LITORAL (ESPOL)

TAREA DE SISTEMAS EMBEBIDOS

JUEGO BASADO EN EL CHIP ATMEGA 328 P Y PIC16F887

INTEGRANTES

- **ISAAC VASQUEZ SOTO**
- **ANGELO CHALEN PIMENTEL**

PROFESOR

ING. RONALD SOLIS MESA

OBJETIVO GENERAL

Diseñar e implementar un juego interactivo en una matriz LED 8x8 controlado por microcontroladores ATmega328P y PIC16F887, donde el jugador debe alinear LEDs mediante botones, integrando visualización y sonido para mejorar la experiencia.

Objetivos Específicos:

- Programar el ATmega328P para controlar la matriz LED y gestionar la lógica del juego de esquivar obstáculos.
- Programar el PIC16F887 para generar efectos sonoros sincronizados con eventos del juego.
- Implementar comunicación efectiva entre ATmega328P y PIC16F887 para coordinar visualización y sonido.
- Simular y verificar el circuito completo en Proteus.
- Documentar detalladamente el diseño, desarrollo y resultados del proyecto.

DESCRIPCION DEL JUEGO

El juego es una experiencia interactiva implementada en una matriz LED de 8x8, donde el jugador controla un punto luminoso que se mueve horizontalmente en la fila inferior (última fila) con la finalidad de esquivar obstáculos que descienden desde las filas superiores.

Mecánica principal:

- Movimiento del jugador: Un LED representa al jugador y se puede mover hacia la izquierda o derecha mediante botones.
- Obstáculos: Varios LEDs que simulan obstáculos se desplazan verticalmente desde la parte superior hacia la parte inferior de la matriz.
- Esquivar: El jugador debe posicionar su LED de forma que no coincida con la columna de un obstáculo al llegar a la última fila.
- Colisión: Si un obstáculo alcanza la última fila y coincide en columna con el jugador, el juego termina.
- Interacción: Un botón permite controlar el movimiento lateral del jugador.
- Retroalimentación: El juego incluye mensajes iniciales (“ARE YOU READY?”, “GO”) y sonidos para mejorar la experiencia, generados por un microcontrolador adicional que recibe órdenes por comunicación paralela.
- Objetivo: Mantenerse vivo esquivando todos los obstáculos que bajan, logrando la mayor puntuación o duración posible.

CAPTURAS DE SIMULACION

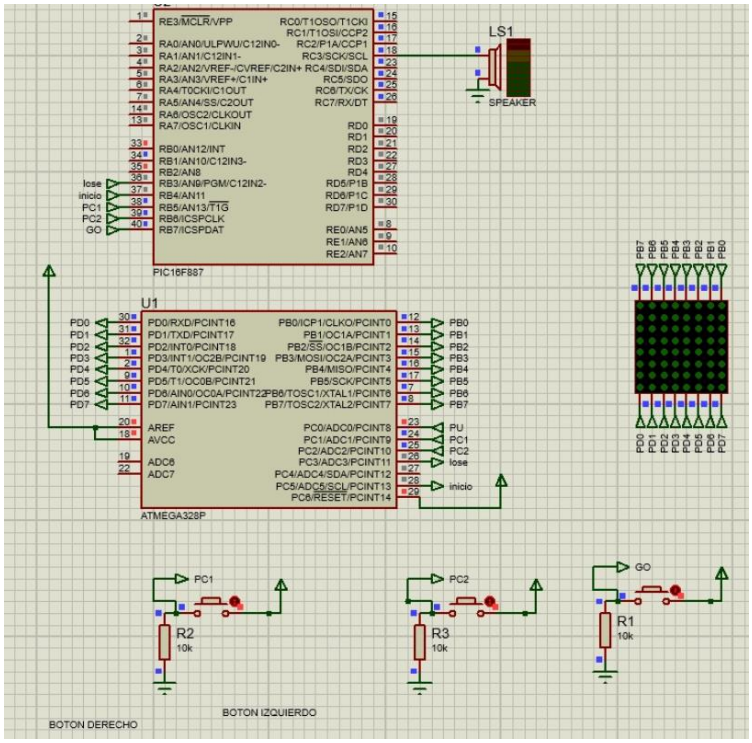

Ilustración 1.Simulacion 1.

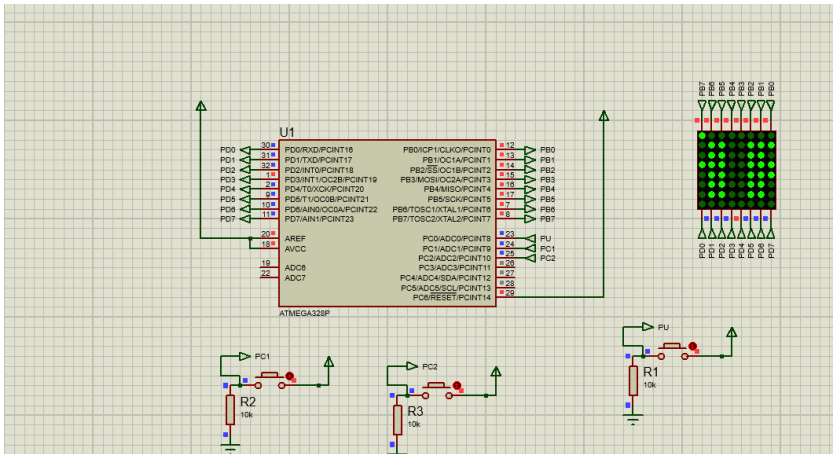

Ilustración 2.Simulacion 2

DESCIPCION DEL CODIGO

ATMEGA 328P

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

int main(void) {
    DDRB = 0xFF;
    DDRD = 0xFF;
    DDRC &= ~((1 << PC0) | (1 << PC1)); // Botones en PC0 y PC1 como entrada
    PORTC |= (1 << PC0) | (1 << PC1);   // Pull-up activado

    unsigned char columna[8] = {1, 2, 4, 8, 16, 32, 64, 128};

    int AYR[] = {
        // Mensaje: ARE YOU READY?
        0x00, 0x7E, 0x7E, 0x90, 0x90, 0x7E, 0x7E, 0x00,
        0x00, 0x7E, 0x7E, 0x58, 0x5E, 0x56, 0x72, 0x00,
        0x00, 0x7E, 0x7E, 0x5A, 0x5A, 0x5A, 0x5A, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x7C, 0x7E, 0x02, 0x02, 0x7E, 0x7C, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x7E, 0x7E, 0x58, 0x5E, 0x56, 0x72, 0x00,
        0X00, 0X7E, 0X7E, 0X66, 0X66, 0X7E, 0X7E, 0X00,
        0x00, 0x70, 0x10, 0x1F, 0x10, 0x70, 0x00, 0x00,
        0x00, 0xE0, 0x8D, 0x88, 0x88, 0xF8, 0x00, 0x00,
    };

    int go[] = {0x7E, 0x4A, 0x6E, 0x00, 0X7E, 0X42, 0X7E, 0X00};
```

```
uint8_t jugador_pos = 3;  // Jugador empieza en la columna 3 (centro)
int juego = 0;

// MOSTRAR "ARE YOU READY?" UNA SOLA VEZ
 while (!(PINC & (1 << PC0))) {
for (int i = 0; i < 73; i++) {
    for (int k = 0; k < 50; k++) {
        for (int j = 0; j < 8; j++) {
            PORTD = columna[j];
            PORTB = ~AYR[i + j];
            _delay_us(40);
        }
    }
}}

// ESPERAR A QUE SUELTE EL BOTÓN (por si ya estaba presionado)



// ESPERAR A QUE PRESIONE EL BOTÓN
while ((PINC & (1 << PC0))) { }

// MOSTRAR GO
for (int r = 0; r < 200; r++) {
    for (int j = 0; j < 8; j++) {
        PORTD = columna[j];
        PORTB = ~go[j];
        _delay_us(100);
    }
}
```

```
    // INICIAR EL JUEGO
    juego = 1;

    while (juego == 1) {
        // Movimiento izquierda
        if (!(PINC & (1 << PC2))) {
            if (jugador_pos > 0) jugador_pos--;
            _delay_ms(10);  // antirrebote
        }

        // Movimiento derecha
        if (!(PINC & (1 << PC1))) {
            if (jugador_pos < 7) jugador_pos++;
            _delay_ms(10);  // antirrebote
        }

        // Mostrar jugador en la fila 7 (última)
        for (int j = 0; j < 8; j++) {
    PORTB= ~(1<<0);  // Activar la columna: poner en bajo (GND) para la columna
deseada

    if (j == jugador_pos) {
        PORTD = (1 << j);  //
    }

    _delay_us(200);  // Retardo para visualización
}

    }
}
```

EXPLICACION

Este código en lenguaje C está diseñado para ejecutarse en un microcontrolador AVR y controlar una matriz de LEDs 8x8 junto con botones físicos conectados al puerto C. Su propósito es crear la base visual y de interacción de un videojuego simple. El programa está dividido en varias fases: primero muestra un mensaje de bienvenida (“ARE YOU READY?”), luego una señal de inicio (“GO”), y finalmente permite al usuario controlar un punto de luz que representa al jugador, el cual puede moverse horizontalmente en la parte inferior de la matriz utilizando botones. Todo el control visual se hace mediante multiplexado, activando una columna a la vez y mostrando los datos correspondientes en las filas, lo cual es típico para este tipo de visualizaciones.

Inicialmente, el código configura los puertos B y D como salidas para manejar la matriz de LEDs, y los pines PC0 y PC1 como entradas con resistencias pull-up activadas para leer el estado de los botones. Se definen dos matrices de datos: una con los patrones necesarios para desplazar el mensaje "ARE YOU READY?" por la matriz, y otra con la palabra "GO", que se muestra estáticamente antes de que comience la interacción con el jugador.

El programa comienza mostrando la animación del mensaje de bienvenida mientras el botón conectado a PC0 esté presionado. Esta animación se logra desplazando una ventana de 8 columnas sobre el arreglo de datos, con un retardo visual suficiente para que el texto se perciba claramente. Una vez que el usuario suelta el botón y vuelve a presionarlo, se muestra el mensaje "GO" por un momento, lo cual sirve como señal para iniciar el juego.

En la fase principal del juego, se activa el control del jugador. Este está representado por un solo LED encendido en la última fila de la matriz, y su posición horizontal se almacena en una variable (jugador_pos). El jugador puede moverse hacia la izquierda o la derecha usando botones conectados a los pines PC2 y PC1, respectivamente, con un retardo pequeño para evitar el rebote mecánico de los botones.

Sin embargo, aunque el diseño original del proyecto sí contemplaba incluir obstáculos que descendieran desde la parte superior de la matriz hacia el jugador, esta parte no se logró implementar correctamente. Se hicieron intentos para programar la caída de obstáculos y su desplazamiento vertical dentro del mismo bucle del juego, pero los patrones no se mostraban correctamente o el sistema dejaba de funcionar como se esperaba. Esto puede deberse a limitaciones en la sincronización del multiplexado, errores en la lógica de desplazamiento vertical o problemas al combinar la visualización del jugador con los obstáculos en el mismo espacio de matriz.

Debido a estas dificultades técnicas, se decidió omitir temporalmente la parte de los obstáculos y dejar únicamente la funcionalidad que sí operaba correctamente: el mensaje de bienvenida animado, el mensaje de inicio y el movimiento del jugador. Por tanto, el código actual representa una versión funcional pero parcial del juego, lista para ser extendida con los obstáculos una vez se resuelvan los errores que impedían su visualización o movimiento adecuado. La estructura general del programa, con su control de botones, multiplexado y visualización dinámica, queda como una base sólida sobre la cual continuar desarrollando la lógica del juego completo.

PIC16F887

```
};

// === Función central para reproducir una nota ===
void TocarNota(enum Notas nota, enum Duracion duracion) {
  unsigned int frecuencia = 0;
  unsigned int tiempo_ms = 0;

  // Asignar frecuencia según la nota
  switch (nota) {
    case NOTA_A5: frecuencia = 880; break;
    case NOTA_B5: frecuencia = 987 ; break;
    case NOTA_C6: frecuencia = 1046; break;
    case NOTA_D6: frecuencia =1175 ; break;
    case NOTA_E6: frecuencia = 1318; break;
    case NOTA_A2: frecuencia = 220; break;
    case NOTA_C3: frecuencia = 261; break;
    case NOTA_B2: frecuencia = 247; break;
    case NOTA_G2: frecuencia = 207; break;
    case NOTA_SILENCIO: frecuencia = 0; break;



// === Enumeraciones de notas y duraciones ===
enum Notas {
  NOTA_SILENCIO,
  NOTA_A5,
  NOTA_B5,
  NOTA_C6,
  NOTA_D6,
  NOTA_E6,
  NOTA_A2,
  NOTA_C3,
  NOTA_B2,
  NOTA_G2
};

enum Duracion {
  SEMICORCHEA,
  CORCHEA,
  NEGRA,
  BLANCA,
  REDONDA
```

```
}

// Asignar duración según figura
switch (duracion) {
  case SEMICORCHEA: tiempo_ms = 125; break;
  case CORCHEA:     tiempo_ms = 50; break;
  case NEGRA:       tiempo_ms = 250; break;
  case BLANCA:      tiempo_ms = 500; break;
}

// Ejecutar sonido o silencio
if (frecuencia == 0){
  switch (duracion) {
     case SEMICORCHEA: Delay_ms(125); break;
     case CORCHEA:     Delay_ms(250); break;
     case NEGRA:       Delay_ms(250); break;
     case BLANCA:      Delay_ms(500); break;
  }
  return;
```

```
}
 else
   Sound_Play(frecuencia, tiempo_ms);
}

// ==== Melodía básica como ejemplo ===
void MelodiaEjemplo() {
 TocarNota(NOTA_A5, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_A5, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_B5, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_B5, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_C6, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_C6, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_D6, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_D6, NEGRA);
 TocarNota(NOTA_E6, NEGRA);
 TocarNota(NOTA_B5, NEGRA);
 TocarNota(NOTA_A5, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_A5, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_B5, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_B5, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_C6, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_C6, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_D6, NEGRA);
 TocarNota(NOTA_E6, BLANCA);
 TocarNota(NOTA_D6, NEGRA);
 TocarNota(NOTA_E6, NEGRA);
 TocarNota(NOTA_B5, NEGRA);

}

void lose(){
 TocarNota(NOTA_A2, NEGRA);
```

```
  TocarNota(NOTA_A2, NEGRA);
  TocarNota(NOTA_A2, SEMICORCHEA);
  TocarNota(NOTA_A2, NEGRA);
  TocarNota(NOTA_C3, NEGRA);
  TocarNota(NOTA_B2, SEMICORCHEA);
  TocarNota(NOTA_B2, NEGRA);
  TocarNota(NOTA_A2, SEMICORCHEA);
  TocarNota(NOTA_A2, NEGRA);
  TocarNota(NOTA_G2, SEMICORCHEA);
  TocarNota(NOTA_A2, NEGRA);

}

// === Configuración principal del sistema ===
void main() {
  ANSEL  = 0;          // Pines AN como digitales
  ANSELH = 0;
  C1ON_bit = 0;        // Comparadores desactivados
  C2ON_bit = 0;

  TRISB = 0xF8;        // RB7-RB3 como entradas (botones)
  TRISC = 0;           // Puerto C como salida
  PORTC = 0;

  Sound_Init(&PORTC, 3);   // Inicializa RC3 para salida de sonido
  //Sound_Play(880, 1000);   // Sonido de bienvenida

  while (1) {
    if (Button(&PORTB, 7, 1, 1)) TocarNota(NOTA_A5, NEGRA);
    while (RB7_bit);

    if (Button(&PORTB, 6, 1, 1)) TocarNota(NOTA_B5, CORCHEA);
    while (RB6_bit);

    if (Button(&PORTB, 5, 1, 1)) TocarNota(NOTA_D6, CORCHEA);
    while (RB5_bit);

    if (Button(&PORTB, 4, 1, 1)) MelodiaEjemplo();  // Ejecuta pequeña melodía
    while (RB4_bit);

    if (Button(&PORTB, 3, 1, 1)) Lose();  // Perdio
    while (RB3_bit);
  }
}
```

EXPLICACION

El código implementado en el microcontrolador **PIC16F887** tiene como función principal recibir señales del ATmega328P a través de una comunicación paralela para reproducir sonidos específicos asociados a diferentes eventos del juego, como inicio, movimiento, colisión y victoria. Para ello, se configuran varios pines del PIC como entradas digitales para leer los datos de 4 bits enviados por el ATmega328P, así como una línea adicional que funciona como señal de habilitación o “strobe” para indicar cuándo los datos están disponibles y deben ser leídos.

El programa utiliza un ciclo de espera o una interrupción para monitorear constantemente el estado de la línea de strobe. Cuando detecta un flanco activo en esta línea, lee los 4 bits de datos que representan un código de comando específico. Según el código recibido, el PIC ejecuta una función que genera el sonido correspondiente mediante un buzzer conectado a uno de sus pines de salida. La generación de sonido se puede realizar mediante técnicas de modulación por ancho de pulso (PWM) o mediante el uso de temporizadores internos que generan frecuencias específicas.

El diseño modular del código permite asociar cada comando a una rutina distinta de reproducción sonora, facilitando la inclusión de nuevas melodías o efectos en futuras versiones. Además, el uso de la señal de strobe asegura que el PIC sincronice correctamente la lectura de comandos con el envío desde el ATmega328P, evitando errores de lectura o sonidos no deseados. En conjunto, este código permite que el sistema completo del juego combine la parte visual controlada por el ATmega con la parte sonora controlada por el PIC, mejorando la experiencia del usuario y aportando dinamismo al juego.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

- El proyecto logró avanzar en la integración del microcontrolador ATmega328P con la matriz LED 8x8 y la comunicación paralela con el PIC16F887, manteniendo una base funcional para el manejo de sonidos.
- Sin embargo, el desarrollo del juego en sí, especialmente la lógica relacionada con el movimiento y la gestión de los obstáculos, no pudo ser culminado completamente debido a dificultades técnicas.
- La simulación en Proteus permitió validar parcialmente algunos componentes y la comunicación entre microcontroladores, pero no fue posible completar ni probar a fondo la mecánica del juego en su totalidad.

RECOMENDACIONES

- Se recomienda optimizar los retardos utilizando temporizadores para mejorar la respuesta y evitar bloqueos.
- Es conveniente modularizar el código para facilitar su mantenimiento y futuras mejoras.
- Se aconseja realizar pruebas en hardware real para ajustar y validar el comportamiento bajo condiciones reales.

ENLACE REPOSITORIO:

lsavsoto/EMBEBIDOS: TAREAS EMBEBIDOS