



**FIEC**

**TAREA 2**

**COMUNICACIÓN ENTRE  
MICROCONTROLADORES**

**PROFESOR:**

**Msc. RONALD SOLIS**

**ESTUDIANTES**

**VICTOR BARRAGAN**

**RICARDO VELASTEGUI**

**PAOII – 2025**

## **Objetivos.**

### **1. Objetivo General**

· Desarrollar un sistema de juego interactivo de 3 niveles de dificultad que combine salidas visuales mediante una matriz LED 8x8 controlada por el microcontrolador ATmega328P, y salidas auditivas mediante un sistema de reproducción de melodías con el microcontrolador PIC16F887, y entradas mediante pulsadores, empleando comunicación entre ambos dispositivos y simulando el funcionamiento completo en Proteus, con el propósito de reforzar habilidades en programación y control de sistemas embebidos. El sistema deberá ser programado en C para ambos microcontroladores.

### **2. Objetivos Específicos**

- Diseñar e implementar un sistema de control de una matriz LED 8x8 con el microcontrolador ATmega328P para desplegar caracteres, símbolos y animaciones del juego.
- Programar un sistema de reproducción de melodías utilizando el PIC16F887, capaz de interpretar comandos recibidos desde el ATmega328P.
- Integrar botones físicos como entradas para interactuar con el juego (selección, acción, reinicio, etc.).
- Implementar 3 niveles de dificultad ya sea modificando la velocidad, lógica o complejidad del juego.
- Establecer un canal de comunicación entre los microcontroladores que permita la sincronización entre efectos visuales y sonoros.
- Simular el juego de manera completa en el entorno Proteus, integrando los dos microcontroladores, la matriz LED y el sistema de audio.
- Documentar todo el proceso de diseño, desarrollo y simulación en un informe técnico, incluyendo repositorio con evidencias y código

## Descripción del juego.

### 1. Concepto General

El proyecto es un juego interactivo de tipo *runner* (corredor) donde el objetivo es guiar a un personaje a través de un recorrido de obstáculos. La acción se visualiza en una matriz LED de 8x8, y el jugador debe usar sus reflejos para esquivar los obstáculos que avanzan por la pantalla. El juego concluye cuando el jugador choca (derrota) o cuando completa exitosamente el mapa (victoria).

### 2. Arquitectura del Sistema

El diseño se basa en una arquitectura de doble microcontrolador, separando las tareas de lógica y audio para un funcionamiento más eficiente.

- Controlador Principal (ATmega328P): Es el cerebro del sistema. Se encarga de toda la operación del juego:
  - Gestión Visual: Controla la matriz LED 8x8, dibujando al jugador y a los obstáculos, y creando la animación de desplazamiento.
  - Lógica de Juego: Lee las entradas de los pulsadores (salto, salto doble, bajar) y actualiza la posición del jugador.
  - Detección de Colisiones: Comprueba constantemente si el personaje ha chocado contra algún obstáculo del mapa.
  - Señalización: Al terminar el juego, envía una señal de "victoria" o "derrota" al otro microcontrolador.
- Controlador de Audio (PIC16F887): Actúa como un procesador de sonido dedicado. Permanece en espera hasta que recibe una señal de estado desde el

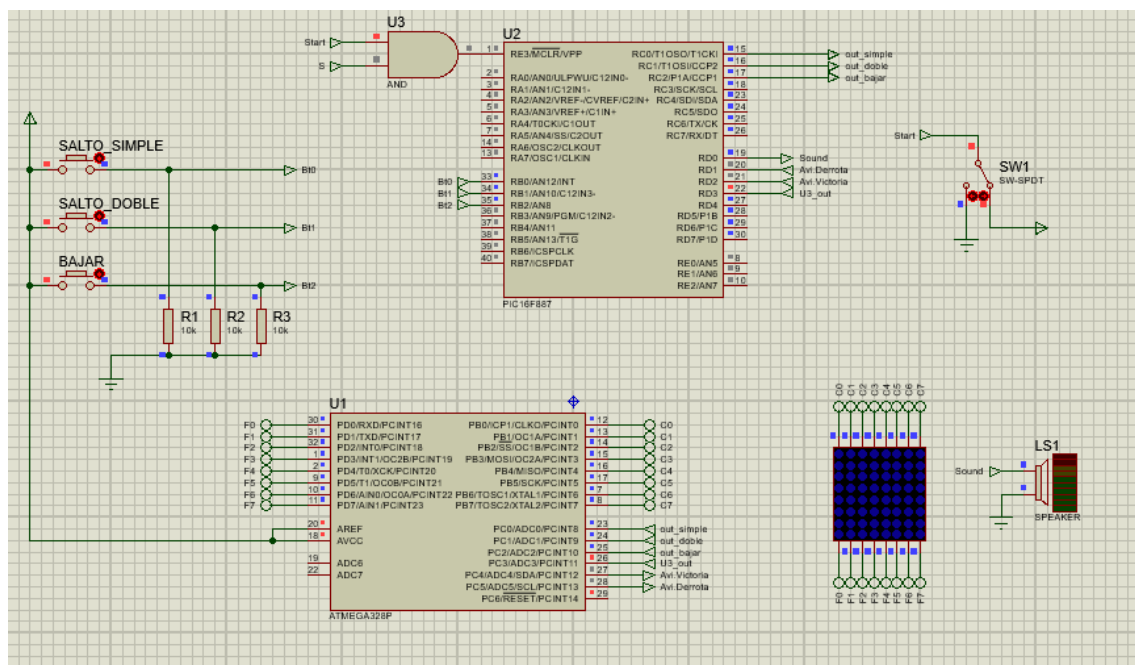
ATmega328P. Al detectarla, reproduce una melodía de victoria o una melodía de derrota a través de un altavoz.

### 3. Funcionamiento y Jugabilidad

El ATmega328P ejecuta el bucle principal del juego, actualizando la pantalla y leyendo los botones de forma continua. El jugador utiliza los botones para hacer que el personaje salte y esquive el mapa de obstáculos que avanza. La detección de colisiones se realiza comparando la posición del jugador con la del mapa en cada instante.

La comunicación entre los dos microcontroladores es fundamental: el ATmega se enfoca al 100% en la jugabilidad y, solo al final, delega la tarea de audio al PIC, permitiendo que cada procesador se especialice en su función.

### Capturas de la simulación en Proteus.



### Explicación del código.

## Bloque 1: Configuración y Variables

- Pines: Se configuran los PORTB y PORTD como salidas completas (8 pines cada uno) para manejar el multiplexado de la matriz LED.
- Controles: Se configuran los pines PC0 y PC1 como entradas (DDRC |= 0xF0;) y se activan sus resistencias *pull-up* (PORTC |= 0x0F;). El código está diseñado para leer los botones directamente en estos pines.
- Variables: Se inicializan variables clave como mapa\_index (para el desplazamiento), salto (para la física) y vivo (para el estado del juego).

## Bloque 2: Lógica Visual (Multiplexado y Sprites)

- Sprites: Se definen en arreglos (const unsigned char[]) los patrones de bits para el jugador (JUGADOR\_SUELO, JUGADOR\_SALTO, etc.), el mapa de obstáculos (MAPA) y las caras de victoria/derrota.
- Función dibujar\_frame(): Es el núcleo del motor gráfico. Realiza el multiplexado (barrido de filas) combinando el *sprite* del jugador y la porción actual del MAPA. Se repite 15 veces por fotograma (repet < 15) para reducir el parpadeo.
- Función mostrar\_cara(): Es una función de estado final. Toma el control total de la matriz para mostrar una cara estática (ganar o perder) durante un tiempo prolongado.

## Bloque 3: Lógica del Juego (Física y Colisiones)

- Bucle Principal: El juego corre dentro de un while(vivo).
- Física: El código gestiona un sistema de salto simple. La variable salto (0, 1 o 2) actúa como un estado. Cuando está activa, la variable altura incrementa, y la posición jugador\_y se modifica para simular la subida y bajada del personaje.
- Colisión: Se implementa una detección por máscara de bits. En cada fotograma, se comprueba (if (MAPA[...] & sprite[...])) si algún bit encendido del jugador se superpone con algún bit encendido del mapa. Si ocurre, la bandera vivo se pone a 0.

- Avance: La variable `mapa_index` se incrementa en cada ciclo, "moviendo" el mapa hacia el jugador.
- Condición de Victoria/Derrota: El juego termina si vivo se vuelve 0 (colisión) o si `mapa_index` supera el tamaño del mapa (victoria).

## 2. Explicación del Código: PIC16F887 (Coprocesador de Audio y Controles)

Este microcontrolador actúa como un dispositivo de entrada/salida y un procesador de audio dedicado.

### Bloque 1: Configuración de Pines (PIC)

- Pines de Entrada (Botones): Se configura `TRISB` como entrada (`0xFF`) para leer los 3 botones físicos. Se activan las resistencias *pull-up* internas (`WPUB`) para `RB0`, `RB1` y `RB2`, por lo que los botones deben conectarse a tierra (`GND`).
- Pines de Salida (hacia ATmega): Se configura `TRISC` como salida (`0x00`) para enviar las señales de los botones al otro microcontrolador.
- Pines de Comunicación y Audio: Se configura `TRISD` para tener una salida de audio (`RD0`) y tres entradas (`RD1`, `RD2`, `RD3`) que esperan señales (Gana, Pierde, Start) desde el ATmega.

### Bloque 2: Lógica de Controles (Lectura y Envío)

- Lectura: En el bucle principal, el PIC lee constantemente el estado de `RB0`, `RB1` y `RB2` (`BTN_SALTO`, etc.).
- Envío: El valor leído (0 lógico si está presionado, 1 si está suelto) se transfiere directamente a los pines de salida de `PORTC` (`OUT_SALTO = BTN_SALTO;`). Este PIC actúa como un "puente" para los botones.

### Bloque 3: Lógica de Audio (Reproductor)

- Melodías: Las notas y duraciones de las 3 melodías (fondo, ganar, perder) se almacenan en arreglos `const` en la memoria de programa.

- Función `melodias_update()`: Esta función gestiona la reproducción. Aunque usa `Sound_Play()` (que es bloqueante), la lógica del tick intenta que la función se llame periódicamente para reproducir la *siguiente* nota de la secuencia.
- Iniciadores: Las funciones `iniciar_...()` actúan como disparadores que configuran las banderas (`melodia_activa`, `melodia_ganar`, etc.) para indicarle al `melodias_update()` qué secuencia debe reproducir.

#### Bloque 4: Lógica de Estado (Máquina de Estados Simple)

- Bucle Principal: El `main()` del PIC funciona como una máquina de estados simple.
- Estado 1: En Espera/Detenido (`START_FLAG == 1`): Si la señal `START` está alta, el juego se considera detenido. La música se apaga (`melodia_activa = 0`).
- Estado 2: Jugando (`START_FLAG == 0`): Cuando la señal `START` baja, el PIC asume que el juego ha comenzado. Llama una sola vez a `iniciar_melodia_fondo()` y entra en el modo de juego.
- Modo Juego: Mientras juega, monitorea las señales `WIN_FLAG` y `LOSE_FLAG`. Si alguna se activa, llama al "iniciador" correspondiente (`iniciar_ganar()` o `iniciar_perder()`) para cambiar la melodía.

### **Descripción del esquema de comunicación.**



**Enlace al repositorio de GitHub.**

**<https://github.com/ricardovelastegui/TAREA-2-GRUPO-8.git>**

## Conclusiones y recomendaciones

### Conclusiones

1. Se implementó exitosamente un sistema de juego interactivo en una matriz LED 8x8, separando las responsabilidades del sistema en una arquitectura de doble microcontrolador.
2. El ATmega328P demostró ser eficaz como unidad de procesamiento principal, manejando la lógica del juego, la física del salto y el multiplexado de la matriz LED en tiempo real.

3. El PIC16F887 funcionó de manera efectiva como un coprocesador de entrada/salida y audio. Esta delegación de tareas permitió al ATmega328P dedicar todos sus recursos al motor gráfico y la lógica del juego, evitando interrupciones o retrasos que habrían sido causados por la lectura de botones o la generación de melodías.
4. Se estableció un esquema de comunicación unidireccional para los controles (PIC ATmega) y otro para los estados del juego (ATmega PIC), validando el objetivo de la tarea de comunicación entre microcontroladores.
5. La lógica de reproducción de música en el PIC, aunque funcional, demostró que las funciones de sonido bloqueantes (Sound\_Play) pueden ser un desafío, requiriendo una máquina de estados simple para gestionarlas sin detener por completo otras tareas del coprocesador.

### **Recomendaciones**

1. Para una futura versión, se recomienda implementar una comunicación serial (como I2C o UART) en lugar de un bus paralelo para las señales. Esto reduciría drásticamente el número de pines utilizados y simplificaría el cableado entre los dispositivos.
2. Se sugiere refinar el reproductor de audio del PIC utilizando interrupciones (como el Timer0) para generar las frecuencias. Esto crearía un sistema de audio "no bloqueante" real, permitiendo que el PIC realice otras tareas (como leer botones) de forma más fluida y sin latencia.
3. El código del ATmega podría optimizarse para implementar la lógica de niveles de dificultad (como se pedía en el PDF) usando el pin START\_FLAG del PIC. El ATmega podría esperar esta señal para iniciar el juego, permitiendo al PIC manejar un menú de selección de nivel antes de que comience la acción.
4. Se recomienda explorar el uso de *sprites* almacenados en la memoria de programa (PROGMEM) en el ATmega para optimizar el uso de la memoria RAM, lo que permitiría mapas de juego más largos y complejos.