Facultad de ingeniería en
Electricidad y computación

CURSO DE SISTEMAS EMBEBIDOS – TEÓRICO

TAREA # 2

COMUNICACIÓN ENTRE MICROCONTROLADORES

ATMEGA328P - PIC16F887

Grupo # 9

INTEGRANTES:

Diego Alberto Casanova Rizzo

Ariel Omar Padilla Posligua

Mauricio Alejandro Torres Benites

PARALELO 2

DOCENTE:

Ronald David Solis Mesa

NOMBRE DEL PROYECTO:

“Colisión de LEDs”

Contenido

1. Introducción

1.1. Objetivo general

Implementar un juego arcade por medio de dos microcontroladores, usando el chip ATmega-328P y el PIC16F887, donde el primer chip realizará el procesamiento del juego y los estados de este mientras que el segundo chip se encargará de acciones de audio e inmersión del juego.

1.2. Objetivos específicos

- Implementar el control visual del juego en una matriz 8x8 mediante multiplexación utilizando los puertos B y D del ATmega328P.
- Diseñar un sistema de interacción mediante botones que permita iniciar el juego y disparar la bala.
- Establecer una comunicación digital sencilla entre el ATmega328P y el PIC16F887 a través de dos líneas de estado.
- Desarrollar un programa capaz de interactuar con el usuario, determinando si salió victorioso o no por medio de condiciones de la partida en tiempo real, mostrando el resultado al mismo tiempo que se genera una melodía para mayor experiencia de juego.

2. Descripción del juego

El proyecto implementa un juego minimalista tipo arcade desarrollado sobre una matriz LED de 8×8 controlada por un microcontrolador ATmega328P. El juego consiste en una barra vertical ubicada en la columna izquierda, la cual se desplaza continuamente hacia arriba y hacia abajo siguiendo un movimiento oscilante. El jugador cuenta con un botón de disparo que permite lanzar una bala desde la columna derecha hacia la izquierda. El objetivo es acertar el disparo justo cuando la bala atraviesa la posición ocupada por la barra.

El juego posee tres pantallas o estados:

Pantalla inicial (Start): se despliega un mensaje desplazante “PRESS START” hasta que el jugador presiona el botón correspondiente.

Juego en progreso: la barra se mueve automáticamente y el jugador puede disparar una sola bala por ronda.

Pantalla de resultado: Si la bala impacta la barra dentro del rango de su altura, aparece la letra “W” indicando victoria.

Si la bala no coincide con la barra o sale del área de juego sin impactar, se despliega la letra “A”, indicando derrota.

3. Explicación de los scripts

El programa está estructurado mediante módulos funcionales que gestionan desde el hardware hasta la lógica del juego.

3.1. Inicialización de hardware

La función `io_init()` configura:

- **PORTB:** como salida para el control de columnas de la matriz LED.
- **PORTD:** como salida para las filas.
- **PORTC:** como entrada con resistencias pull-up para los botones Start y Fire, y como salida para LEDs indicadores de estado.

Esta inicialización permite realizar el multiplexado de la matriz y capturar las entradas del usuario.

3.2. Sistema de renderizado (multiplexado)

El corazón de la visualización es la función `scan_frame()`:

- Recorre secuencialmente las 8 filas.
- Activa una fila a la vez y escribe el patrón de columnas correspondiente.
- Repite este proceso a alta velocidad para generar la persistencia visual necesaria.

La matriz se actualiza a partir de arreglos de 8 bytes donde cada byte representa el estado de las columnas en una fila específica.

3.3. Presentación de imágenes y texto

- `show_glyph()` recibe una imagen estática 8×8 y la muestra cierto tiempo.
- `show_start_initial()` implementa un desplazamiento horizontal del mensaje “PRESS START” utilizando un buffer continuo de columnas.

Estas funciones componen la interfaz gráfica del juego.

3.4. Construcción del frame de juego

La función `build_game_frame()` arma cada fotograma del juego:

- Limpia el arreglo de columnas.
- Dibuja la barra vertical según su posición (`bar_top`) y su longitud (`bar_len`).
- Dibuja la bala si está activa, en su fila y columna correspondiente.

Este buffer resultante es enviado directamente a `scan_frame()` para su representación.

3.5. Lógica del juego

La función play_round() contiene toda la mecánica:

- Inicializa la barra oscilante, la bala y sus posiciones.
- Detecta el disparo mediante lectura del botón Fire.
- Actualiza el movimiento vertical de la barra, invirtiendo su dirección en los límites superior e inferior.
- Mueve la bala hacia la izquierda decrementando su columna.
- Determina condiciones de victoria o derrota:
    - Si la bala alcanza la columna 0 y su fila coincide con la barra → victoria.
    - Si la bala sale de la matriz sin impactar → derrota.

El retorno de true o false es usado para decidir qué símbolo mostrar en pantalla.

3.6. Máquina de estados (main)

El main() gestiona el flujo del juego mediante un switch:

- **Estado 0:** pantalla inicial con el mensaje.
- **Estado 1:** ejecución de una ronda del juego.
- **Estado 2:** despliegue del símbolo de victoria.
- **Estado 3:** despliegue del símbolo de derrota.

4. Descripción del esquema de comunicación

Aunque el sistema no utiliza protocolos avanzados, sí implementa un esquema interno de comunicación entre:

1. El hardware de la matriz LED,
2. Los botones del usuario,
3. El firmware estructurado en funciones.

El esquema puede describirse en tres niveles:

4.1. Comunicación microcontrolador - matriz LED

Se basa en multiplexado dinámico:

- PORTD transmite la fila activa.
- PORTB transmite la información de las columnas.
- Los frames (patrones de 8 filas) se generan en software y se envían en ciclos rápidos para formar imágenes estables.

La comunicación es unidireccional y de muy bajo nivel, directamente mediante escritura de registros.

4.2. Comunicación usuario → microcontrolador

Los botones Start (PC0) y Fire (PC1):

- Usan lógica activa en 0 (pull-up interno).
- El microcontrolador verifica el estado leyendo el registro PINC.
- La interacción se procesa dentro de la lógica del juego:
  - Start cambia el estado del sistema.
  - Fire activa la creación de la bala.

Es un esquema de comunicación digital simple, por consulta (polling).

4.3. Comunicación entre módulos internos del software

La coordinación se realiza via paso de variables y retornos:

- play_round() informa al main() el resultado mediante un valor booleano.
- build_game_frame() comunica la información gráfica hacia scan_frame().
- Las funciones de presentación comunican el contenido visual mediante arreglos de bytes.

Esta estructura modular permite separar:

- Lógica del juego
- Renderizado de la matriz
- Interacción con el usuario

permitiendo escalabilidad y claridad en la implementación.

5. Capturas de la simulación en Proteus

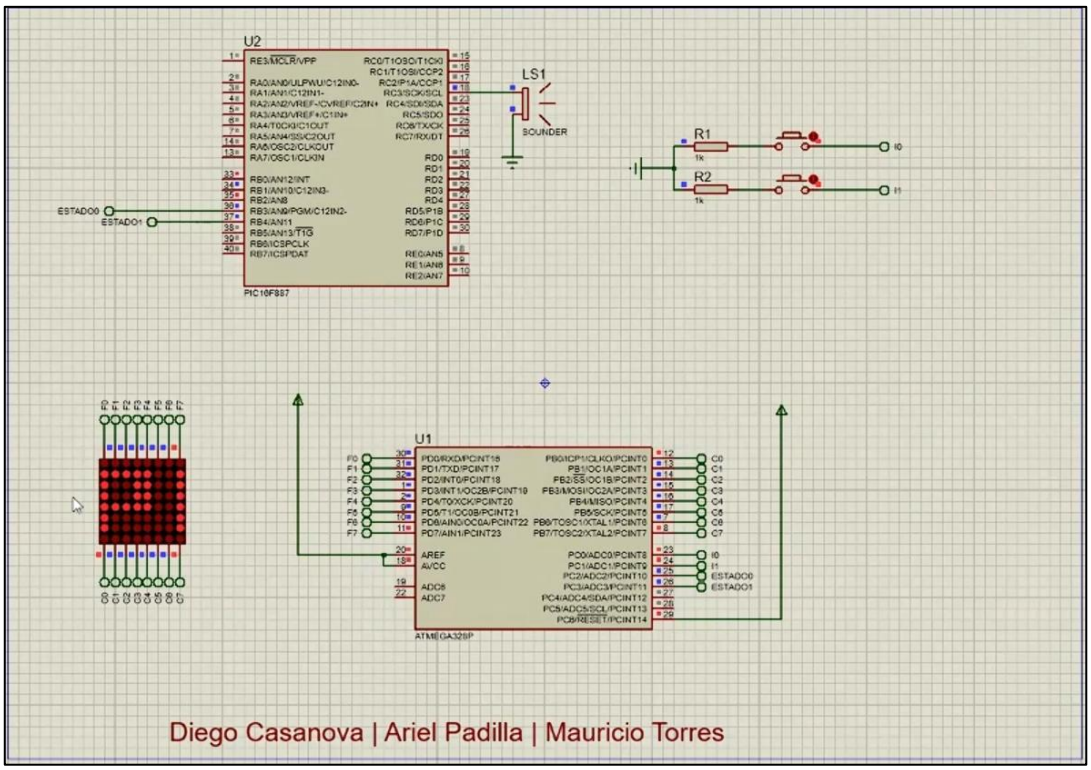

Ilustración 1. Pantalla de inicio del juego (Press Start).

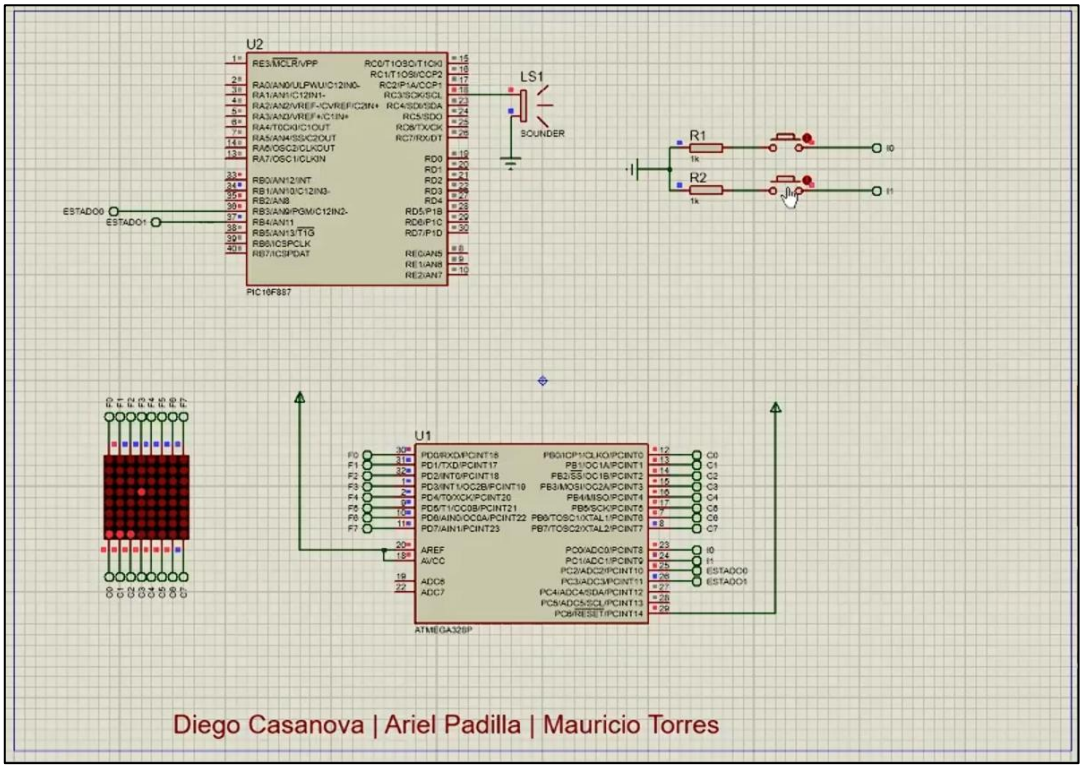

Ilustración 2. Lanzamiento del proyectil durante la sesión del juego.

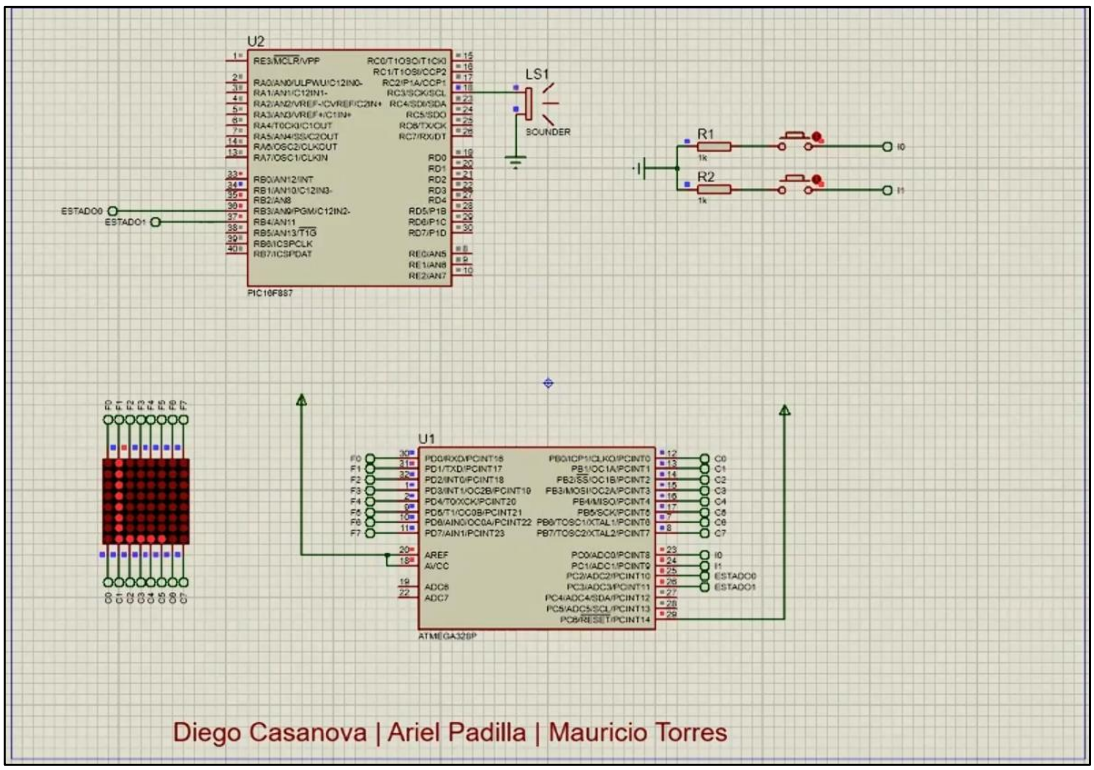

Ilustración 3. Pantalla de derrota de la partida (Lost).

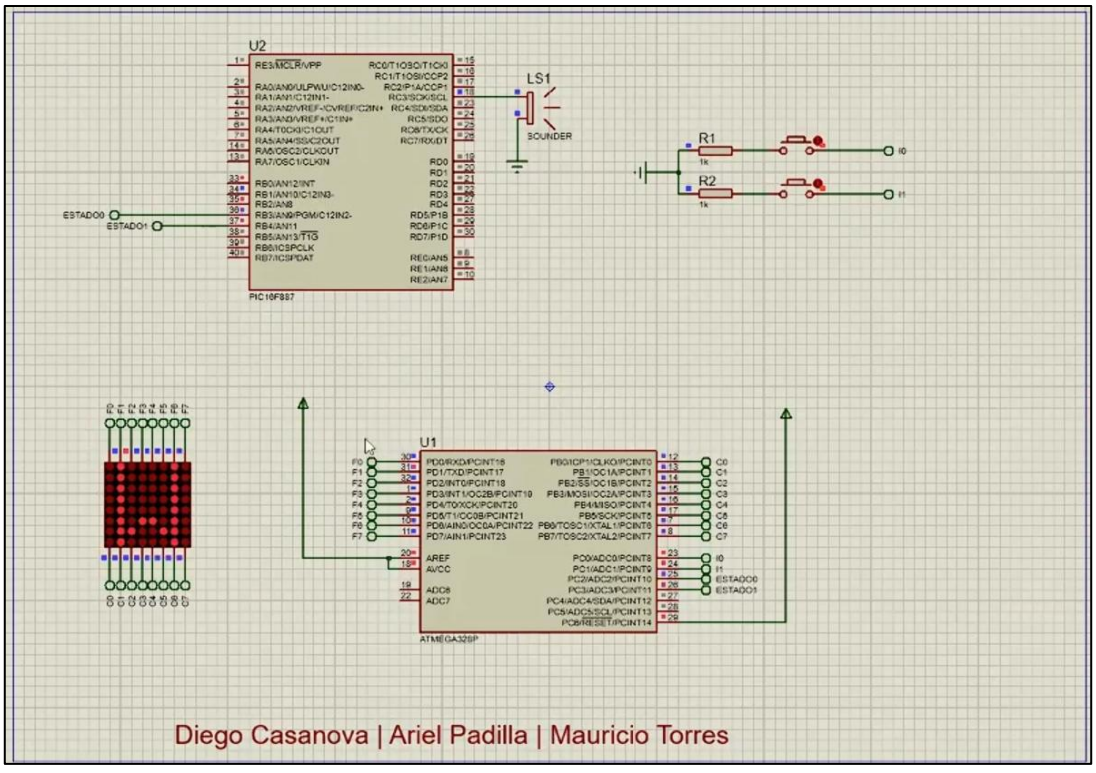

Ilustración 4. Pantalla de victoria de la partida (Win).

6. Enlace al repositorio GitHub del proyecto

- https://github.com/arielomar28/Tarea_2_SE_G9

7. Conclusiones y recomendaciones

Conclusiones

- La implementación del juego “Bullet” demuestra la correcta integración entre dos microcontroladores con funciones distintas: uno dedicado a la lógica y visualización, y otro al audio.
- El uso de multiplexación permitió aprovechar eficientemente la matriz 8×8 sin necesidad de hardware adicional.
- La comunicación mediante 2 bits resultó suficiente para transmitir todos los estados relevantes del sistema.
- La simulación en Proteus permitió validar la interacción entre ambos dispositivos antes de una implementación física.

Recomendaciones

- Añadir limitadores o filtros de ruido en las líneas PC2–PC3 cuando se lleve a hardware real.
- Considerar agregar animaciones adicionales para transiciones entre estados.
- Utilizar interrupciones en el ATmega para una mejor respuesta del disparo.
- Optimizar las melodías del PIC reduciendo llamadas repetidas y creando patrones reutilizables.