TAREA 3 - SISTEMAS EMBEBIDOS

Nombre del Proyecto: Juego de Construcción de Torre 8x8 con Matriz LED y Sonido

Integrantes: Gonzabay Rojas Walter Alexander

Paralelo: 2

1. Objetivo General

Desarrollar un videojuego interactivo basado en la construcción de una torre en una matriz LED 8x8 controlada por el microcontrolador ATmega328P, con efectos de sonido gestionados mediante un PIC16F887. El sistema debe incluir niveles de dificultad, pulsadores como entradas y comunicación I2C entre ambos microcontroladores, simulando completamente su funcionamiento en Proteus.

2. Objetivos Específicos

- Implementar el control visual del juego mediante una matriz LED 8x8 usando el ATmega328P.
- Programar efectos sonoros asociados a eventos del juego en el PIC16F887.
- Incorporar pulsadores para interactuar con el sistema (iniciar juego, detener bloque, reiniciar).
- Establecer comunicación I2C entre los microcontroladores.
- Diseñar tres niveles de dificultad con variación en velocidad y complejidad.
- Simular todo el sistema en Proteus incluyendo microcontroladores, matriz LED, pulsadores y sonido.

3. Descripción Técnica del Juego

El videojuego consiste en construir una torre bloque por bloque. Un bloque (grupo de LEDs encendidos en una fila) se mueve horizontalmente sobre la matriz LED 8x8. El jugador debe

presionar un botón para detener el bloque en el momento justo, intentando que coincida al menos parcialmente con el bloque inferior. La lógica del juego está programada desde cero, utilizando registros directamente para controlar los pines del ATmega328P y gestionar el encendido de columnas y filas de la matriz mediante multiplexado.

Si no hay solapamiento entre bloques, el jugador pierde. Si logra completar una torre hasta la parte superior de la matriz, gana. El sistema reconoce la victoria o derrota y emite una melodía acorde mediante el PIC16F887, que está a la escucha por I2C.

Una parte esencial del desarrollo fue la implementación de un archivo de funciones separado donde se definieron subrutinas para mostrar letras, animaciones, mensajes y efectos visuales. Esto nos permitió mantener el código principal del juego más organizado, y facilitar la reutilización de código. Entre estas funciones se encuentran:

- Mostrar mensajes como "Bievenido", "WIN".
- Visualizar el nivel actual (1, 2, 3)
- Evaluar la superposición de bloques anteriores con el actual.

Niveles del Juego

- **Nivel 1 (Fácil)**: Velocidad baja. Cantidad de bloques por secciones: 5, 4, 3. Es ideal para familiarizarse con el ritmo del juego.
- **Nivel 2 (Medio)**: Velocidad media. Cantidad de secciones: 5, 3, 2. Requiere mayor precisión y reacción.
- **Nivel 3 (Difícil)**: Alta velocidad. Cantidad de secciones: 4, 2, 1. Máxima dificultad. Un margen de error muy limitado.

4. Capturas de Simulación en Proteus

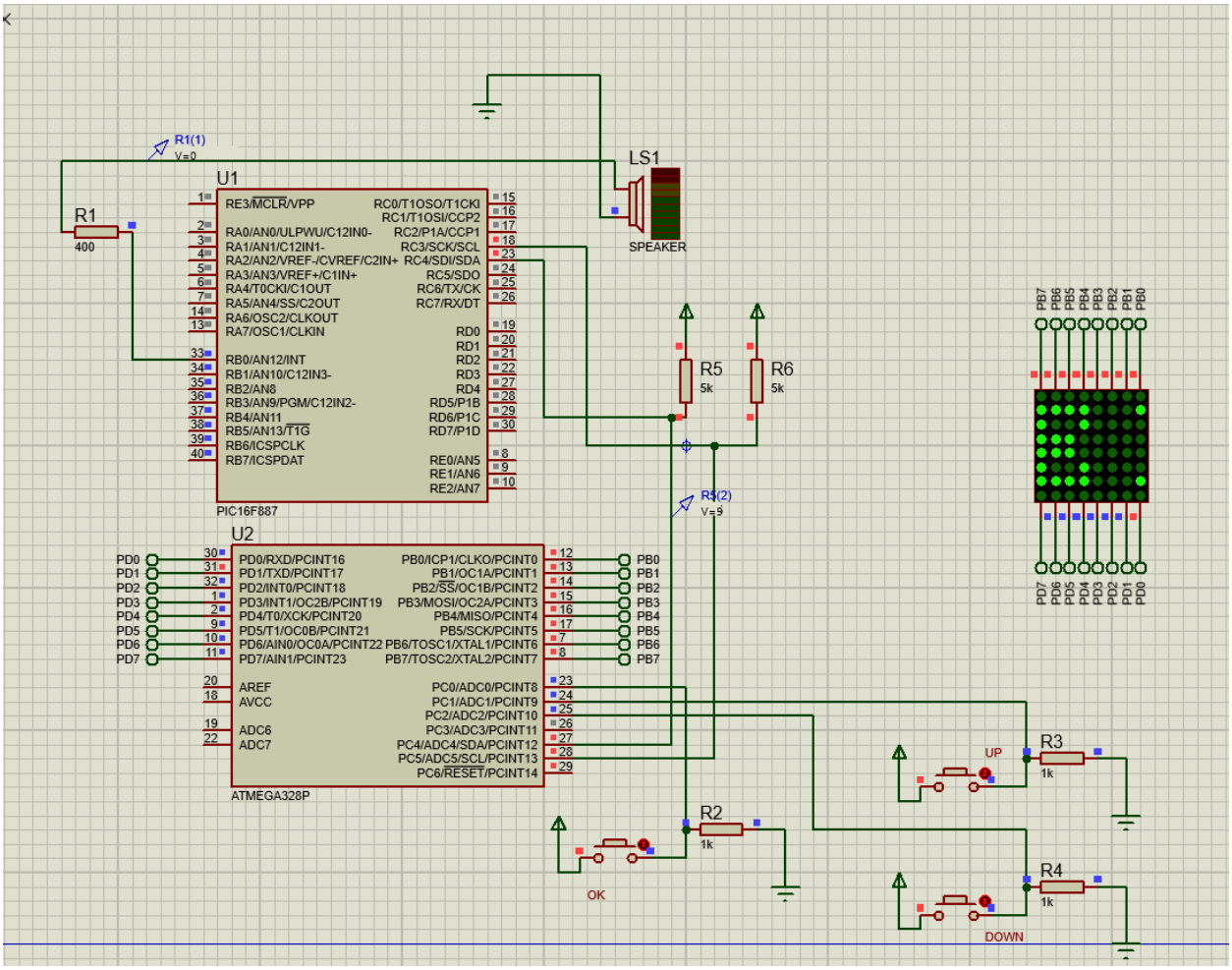

Ilustración 1. Pantalla de bienvenida.

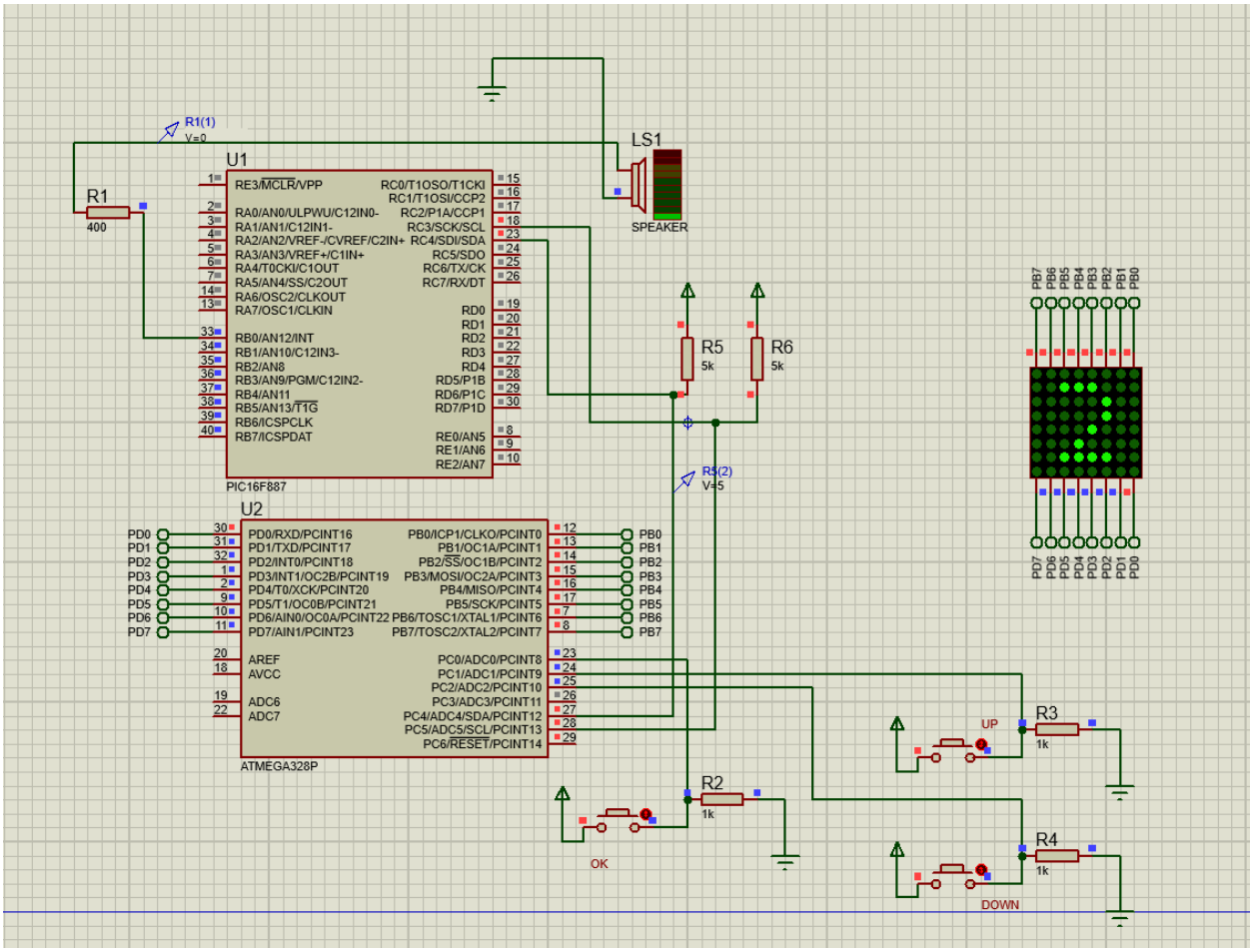

Ilustración 2. Selección de niveles

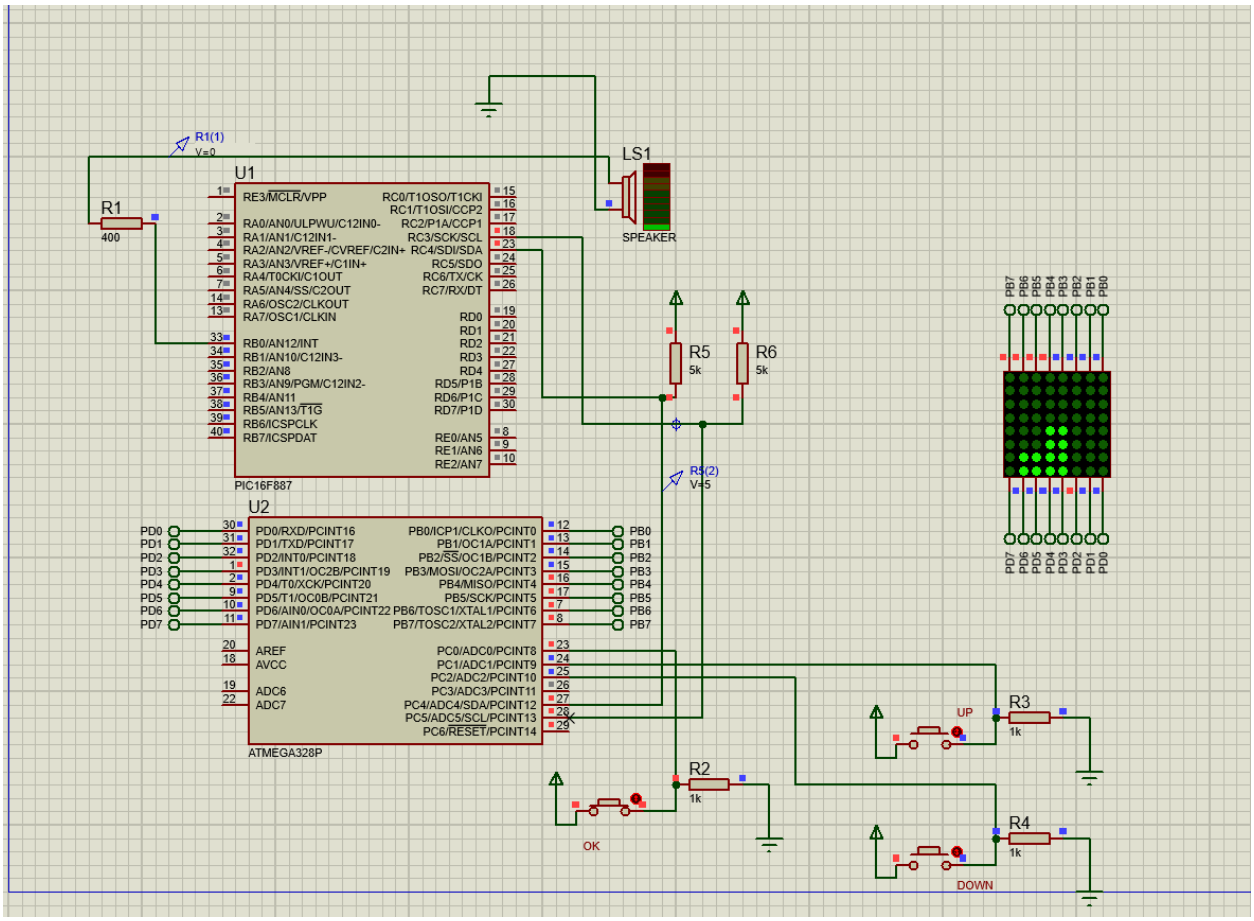

Ilustración 3. Durante el juego.

5. Explicación del Código de forma geeneral

ATmega328P

El código se estructuró en dos archivos: el archivo principal donde se encuentra la lógica del juego, y un archivo de funciones donde se manejan las tareas de visualización.

- Se inicializaron los pines de salida para el manejo manual de columnas y filas.
- Se implementó multiplexado por software, sincronizado con retardos calculados para lograr fluidez.
- Se leyó el estado del pulsador para determinar el momento de detener el bloque.

- Una función compara los bits de la fila actual con la anterior para validar solapamiento.
- Al detectarse un evento de juego (inicio, acierto, victoria o error), se envió un código por I2C al PIC.
- Se utilizo programación por estados para mejor organización en la etapa de la construcción del videojuego.

PIC16F887

- Se configuró el PIC como esclavo I2C.
- No se utiliza interrupciones por lo que el PIC espera la instrucción (No recomendable).
- Dependiendo del valor recibido, se reproduce una melodía generando una señal cuadrada con una frecuencia dependiendo del tono que se requiere.
- Las melodías están definidas por secuencias de frecuencias y tiempos adaptadas al buzzer conectado.

6. Esquema de Comunicación entre Microcontroladores

- **Tipo**: I2C
- **Master**: ATmega328P
- **Slave**: PIC16F887
- **Protocolo**: El ATmega328P envía un byte de comando al PIC cuando ocurre un evento clave:
    - 0x01: Inicio de juego
    - 0x02: Nivel 1

- 0x03: Nivel 2
- 0x04: Nivel3
- 0x05: Victoria
- 0x06: Derrota

El PIC permanece en espera continua por interrupciones de recepción, reaccionando en tiempo real para sincronizar la experiencia audiovisual del juego.

```
void TWI_init(void)
{
    // Fórmula: SCL freq = F_CPU / (16 + 2*TWBR*Prescaler)
    TWSR = 0x00;                          // Prescaler = 1
    TWBR = ((F_CPU / SCL_CLOCK) - 16) / 2; // TWBR para 100kHz con F_CPU = 16MHz
    TWCR = (1 << TWEN);                   // Habilitar TWI
}
```

Ilustración 4. Inicialización de la comunicación I2C atmega328p

```
//Configurar I2C
TRISC3_bit = 1;
TRISA4_bit = 1;
//Direccion 0x50

SSPADD = 0xA0;  // Dirección del esclavo
SSPSTAT = 0x80; // Slew rate off (para 100 kHz)
SSPCON = 0x36;  // Modo esclavo 7-bit + habilita SSP + CKP
SSPCON2 = 0x01; // Habilita esclavo
```

Ilustración 5. Inicialización del I2C en el PIC16F887.

7. Repositorio GitHub

https://github.com/Walalex04/VideoGame.git

8. Conclusiones

1. Se comprobó la posibilidad de implementar videojuegos simples en sistemas embebidos mediante programación de bajo nivel, sin uso de módulos externos.
2. Se demostró la comunicación efectiva I2C entre dos microcontroladores de diferente familia para sincronizar efectos visuales y auditivos.
3. Se evidenció que es posible integrar entradas, salidas visuales y sonoras en un sistema embebido para lograr una experiencia de usuario interactiva.
4. La separación del código en módulos (principal y funciones) mejoró la organización, el mantenimiento y la reutilización del código.

9. Recomendaciones

1. Optimizar el consumo de energía al apagar LEDs inactivos o usar modos de bajo consumo en el microcontrolador.
2. Asegurar el debounce de botones en sistemas físicos reales mediante software o componentes RC.
3. Probar la implementación en hardware real para validar aspectos como latencias y posibles interferencias en la comunicación I2C.
4. Considerar el uso de timers por hardware para lograr mayor precisión y eficiencia en los efectos visuales.
5. Ampliar la funcionalidad del juego agregando niveles aleatorios o tablas de puntaje con EEPROM.