ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Sistemas Embebidos

TAREA #2

Comunicación entre microncontroladores - Desarrollo de videojuego

Estudiantes:
José Aguirre Vacas
Kevin Andy Morán
Helen Gualoto

Profesor:
Ing. Ronald Solis

Objetivos

Objetivo General

- Implementar y validar un sistema de juego interactivo mediante el microcontrolador Atmega328p para la lógica del sistema y el PIC16F887 para el sonido del mismo demostrando comunicación efectiva entre dispositivos para realizar operaciones complejas.

Objetivos específicos

- Definir un protocolo de comunicación que permita al microcontrolador principal (ATmega328P) al módulo de sonido, asegurando la robustez y la correcta interpretación de los comandos entre sistemas.
- Programar el microcontrolador PIC16F887 para operar como un dispositivo esclavo y asíncrono que implemente un mapeo de comandos rígido. Este controlador debe ser capaz de recibir el comando del ATmega mientras el sistema de sonido ejecuta la melodía completa, asegurando que la reproducción de audio no bloquee el juego principal.

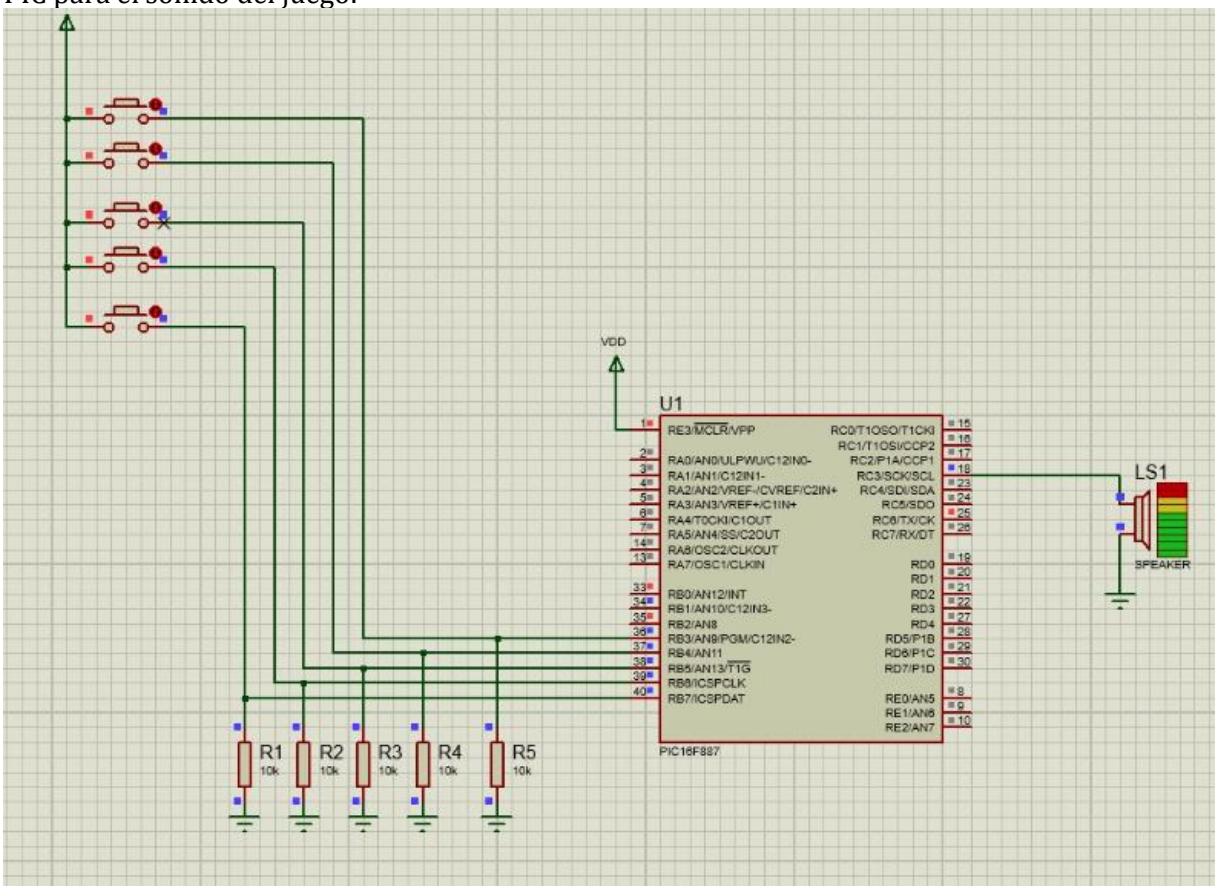
Descripción Técnica del juego

El juego desarrollado consiste en una repetición de una secuencia específica de botones en base a la observación y memorización de esta secuencia en una matriz de leds. Este proyecto considera el sonido del videojuego según las interacciones del usuario: derrota, victoria, selección, etc. La lógica del juego se implementa mediante un microcontrolador Atmega328P y el sonido a través de un PIC16F887 simulados en Proteus. El Atmega 328p enciende las filas y columnas de la matriz de leds 8x8 de forma que produzcan una secuencia aleatoria, constando de cuatro secciones o bloques de leds que se encienden una a la vez.

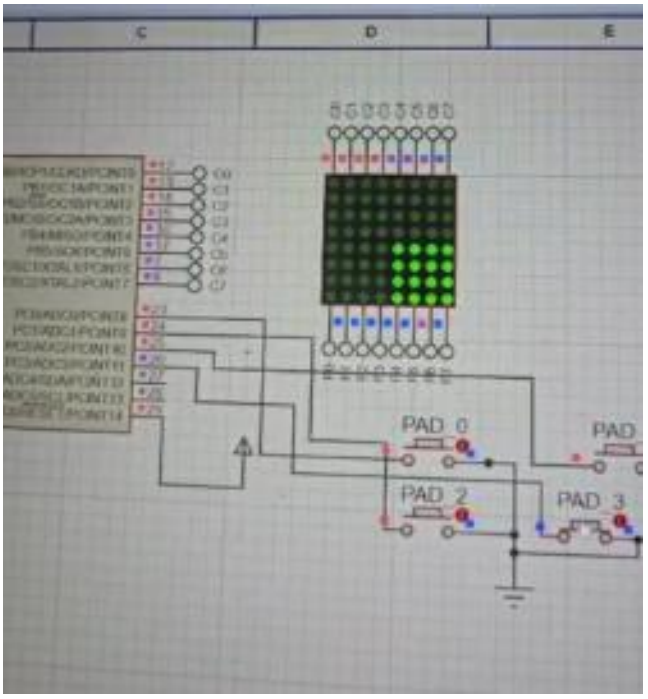
También se puede realizar un cambio de dificultad del juego, el cual tiene tres niveles y estos aumentan en la velocidad de la secuencia y el número de encendidos de esta. El usuario reproduce la secuencia vista utilizando botones cuyas señales ingresan al Atmega328P. Al perder se muestra en la matriz la letra "L" y se envía la señal al PIC16F887que libera el sonido de Game Over. Al terminar cada nivel, también se liberan sonidos respectivos para indicar la victoria del usuario. Estos cambian en cada victoria de nivel.

Simulación en Proteus

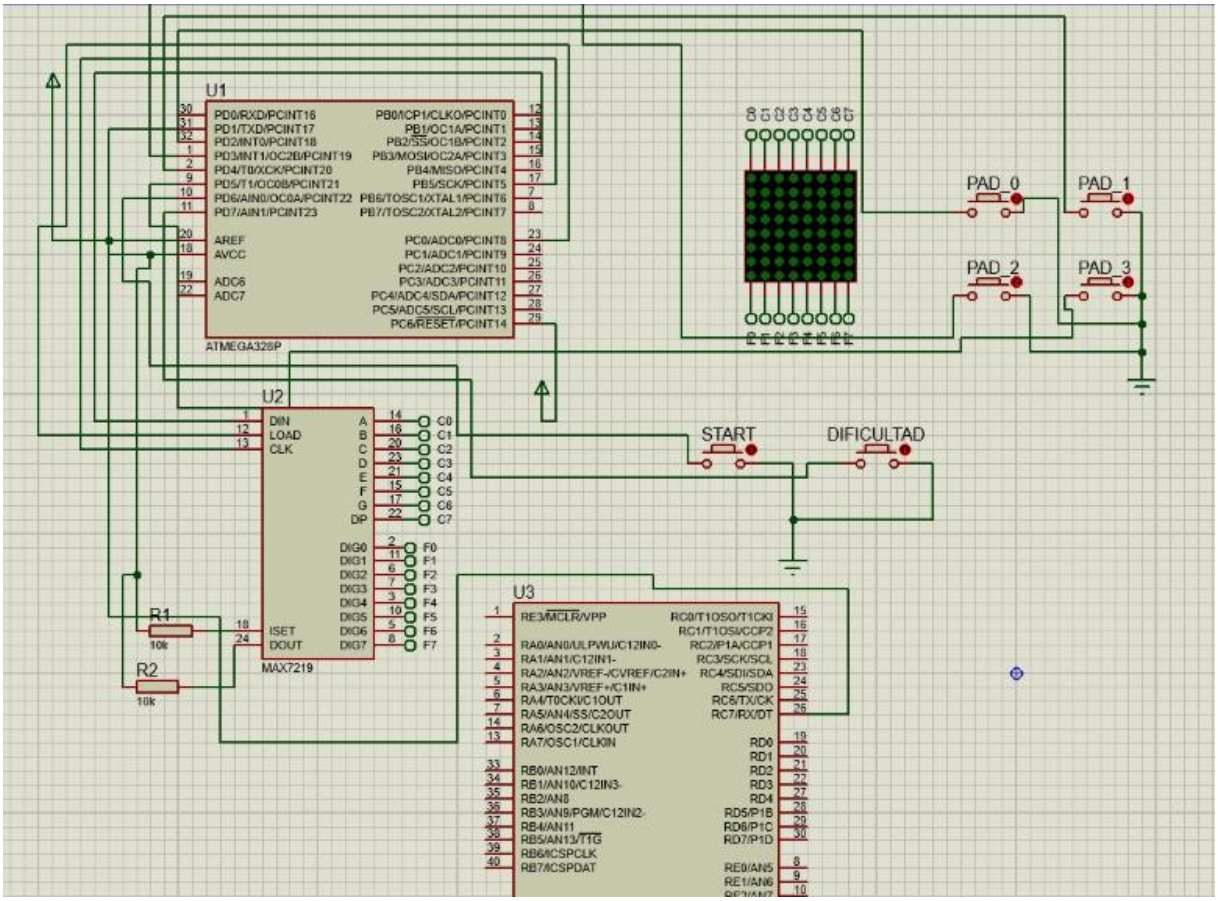
PIC para el sonido del juego:



Atmega para el control de la matriz



Sistemas comunicados



Código

Atmega328p:

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>
#include <stdint.h>

#define MAX_STEPS 6
#define NUM_PADS  4

#define BTN0       PC0
#define BTN1       PC1
#define BTN2       PC2
#define BTN3       PC3
#define BTN_MASK  ((1<<BTN0)|(1<<BTN1)|(1<<BTN2)|(1<<BTN3))

#define BTN_START  PC4
#define BTN_LEVEL  PC5

uint8_t secuencia[MAX_STEPS];
uint8_t longitud = 0;
```

```
uint8_t dificultad = 1;

void clearMatrix(void)
{
    PORTD = 0x00;
    PORTB = 0xFF;
}

const uint8_t PORTFILAS[8] = {1,2,4,8,16,32,64,128};

const uint8_t LETRA_S[8] = {
    0x00,0x7E,0x7E,0x5A,0x5A,0x5A,0x5A,0x00
};

const uint8_t LETRA_L[8] = {
    0x00,0x7E,0x7E,0X60,0x60,0x60,0x60,0x00
};

// NUEVA LETRA G (GANASTE)
const uint8_t LETRA_G[8] = {
    0x00,0x7E,0x7E,0x42,0x52,0x72,0x72,0x00
};

const uint8_t NUM_1[8] = {
    0x00,0x40,0x44,0X7E,0x7E,0x40,0x40,0x00
};

const uint8_t NUM_2[8] = {
    0x00,0x66,0x76,0X7E,0x5E,0x4C,0x00,0x00
};

const uint8_t NUM_3[8] = {
    0x00,0x5A,0x5A,0X5A,0x5A,0x5E,0x7E,0x00
};

void waitForStart(void)
{
    while (1)
    {
        for (uint8_t r = 0; r < 8; r++)
        {
            PORTD = PORTFILAS[r];
            PORTB = ~LETRA_S[r];
            _delay_us(300);
        }

        if (!(PINC & (1 << BTN_START)))
        {
            _delay_ms(80);
            while (!(PINC & (1 << BTN_START)));
            _delay_ms(40);
```

```
            clearMatrix();
            return;
        }
    }
}

void showLose(void)
{
    for (uint16_t t = 0; t < 300; t++)
    {
        for (uint8_t r = 0; r < 8; r++)
        {
            PORTD = PORTFILAS[r];
            PORTB = ~LETRA_L[r];
            _delay_us(100);
        }
    }
    clearMatrix();
}

void showWin(void)
{
    for (uint16_t t = 0; t < 300; t++)
    {
        for (uint8_t r = 0; r < 8; r++)
        {
            PORTD = PORTFILAS[r];
            PORTB = ~LETRA_G[r];
            _delay_us(100);
        }
    }
    clearMatrix();
}

void mostrarNumero(uint8_t nivel)
{
    const uint8_t *bitmap;

    if (nivel == 1)      bitmap = NUM_1;
    else if (nivel == 2) bitmap = NUM_2;
    else                 bitmap = NUM_3;

    for (uint8_t r = 0; r < 8; r++)
    {
        PORTD = PORTFILAS[r];
        PORTB = ~bitmap[r];
        _delay_us(100);
    }
}

uint8_t seleccionarNivel(void)
```

```
{
    uint8_t nivel = 1;
    uint8_t prev = (PINC & (1 << BTN_LEVEL));

    while (1)
    {
        mostrarNumero(nivel);

        uint8_t p = PINC;
        uint8_t curr = (p & (1 << BTN_LEVEL));

        if (prev && !curr)
        {
            nivel++;
            if (nivel > 3) nivel = 1;
            _delay_ms(60);
        }

        prev = curr;

        if (!(p & (1 << BTN_START)))
        {
            _delay_ms(80);
            while (!(PINC & (1 << BTN_START)));
            _delay_ms(40);
            clearMatrix();
            return nivel;
        }
    }
}

uint8_t randomPad(void)
{
    if (longitud == 0) return rand() % NUM_PADS;

    uint8_t nuevo;
    do {
        nuevo = rand() % NUM_PADS;
    } while (nuevo == secuencia[longitud-1]);

    return nuevo;
}

void agregarPaso(void)
{
    if (longitud < MAX_STEPS)
        secuencia[longitud++] = randomPad();
}

void showPad(uint8_t pad)
{
```

```
    uint8_t rowStart = (pad < 2) ? 0 : 4;
    uint8_t colStart = (pad % 2 == 0) ? 0 : 4;

    for (uint16_t t = 0; t < 40; t++)
    {
        for (uint8_t r = rowStart; r < rowStart + 4; r++)
        {
            clearMatrix();
            PORTD |= (1 << r);

            for (uint8_t c = colStart; c < colStart + 4; c++)
                PORTB &= ~(1 << c);

            _delay_us(300);
        }
    }
    clearMatrix();
}

void mostrarSecuencia(void)
{
    for (uint8_t i = 0; i < longitud; i++)
    {
        showPad(secuencia[i]);

        if (dificultad == 1)
            _delay_ms(90);
        else if (dificultad == 2)
            _delay_ms(45);
        else
            _delay_ms(1);
    }
    clearMatrix();
}

void esperarTodosSueltos(void)
{
    while ((PINC & BTN_MASK) != BTN_MASK);
    _delay_ms(20);
}

uint8_t leerBoton(void)
{
    esperarTodosSueltos();

    while (1)
    {
        uint8_t p = PINC;

        if (!(p & (1<<BTN0))) { _delay_ms(15); if (!(PINC &
(1<<BTN0))) return 0; }
```

```
        if (!(p & (1<<BTN1))) { _delay_ms(15); if (!(PINC &
(1<<BTN1))) return 1; }
        if (!(p & (1<<BTN2))) { _delay_ms(15); if (!(PINC &
(1<<BTN2))) return 2; }
        if (!(p & (1<<BTN3))) { _delay_ms(15); if (!(PINC &
(1<<BTN3))) return 3; }
    }
}

uint8_t usuarioCorrecto(void)
{
    for (uint8_t i = 0; i < longitud; i++)
    {
        uint8_t pad = leerBoton();

        showPad(pad);
        clearMatrix();
        _delay_ms(10);

        if (pad != secuencia[i])
            return 0;
    }
    return 1;
}

int main(void)
{
    DDRD = 0xFF;
    DDRB = 0xFF;

    DDRC &= ~BTN_MASK;
    PORTC |= BTN_MASK;

    DDRC &= ~(1<<BTN_START);
    PORTC |= (1<<BTN_START);

    DDRC &= ~(1<<BTN_LEVEL);
    PORTC |= (1<<BTN_LEVEL);

    clearMatrix();
    srand(1);

    while (1)
    {
        waitForStart();
        dificultad = seleccionarNivel();

        longitud = 0;
        agregarPaso();

        while (1)
```

```
        {
            mostrarSecuencia();

            if (usuarioCorrecto())
            {
                if (longitud >= MAX_STEPS)
                {
                    showWin();
                    break;
                }

                agregarPaso();
            }
            else
            {
                showLose();
                break;
            }
        }
    }
}
```

- #define MAX_STEPS 6: Limita el juego a seis pasos. Este es el máximo que la matriz secuencia puede almacenar.
- #define NUM_PADS 4: Define el juego como de cuatro zonas/botones.
- #define BTN_START PC4, #define BTN_LEVEL PC5: Asignan pines específicos para las funciones de Inicio y Selección de Dificultad.
- uint8_t dificultad = 1;: Inicializa la variable que controla la velocidad de visualización de la secuencia.
- uint8_t longitud = 0;: Mantiene el nivel actual o el número de pasos en la secuencia.

El main contiene el flujo maestro del juego:

1. **Configuración de E/S (DDR*, PORTC |= ...):** Establece los puertos B y D como salidas (matriz) y configura los pines de control como entradas con pull-ups. Esto asegura que los botones funcionen correctamente, detectando una pulsación cuando se cierran a tierra.
2. **Bucle while (1) (Juego Nuevo):** Todo el juego está dentro de un bucle infinito, asegurando que el jugador regrese a la pantalla de inicio al terminar o perder.
3. **waitForStart():** El juego se pausa aquí, mostrando la letra 'S' hasta que el usuario presione el botón de inicio.
4. **dificultad = seleccionarNivel();:** Ejecuta el bucle de selección, permitiendo al usuario elegir la velocidad antes de comenzar.
5. **Inicialización del Nivel:** Se resetea longitud = 0; y se llama a agregarPaso() para empezar con el primer elemento de la secuencia.
6. **Bucle while (1) (Partida Activa):** El bucle interno gestiona los turnos:
    - **if (usuarioCorrecto()):** Verifica la entrada del jugador.

        - Si es correcto y longitud < MAX_STEPS, llama a agregarPaso() y el juego continúa.
        - Si es correcto y longitud >= MAX_STEPS, llama a showWin() y rompe el ciclo interno, volviendo a la pantalla de inicio.
    - **else { showLose(); break; }:** Si es incorrecto, muestra la animación de fallo y rompe el ciclo interno.

- **const uint8_t PORTFILAS[8]:** Este arreglo define los bits selectores de fila (1, 2, 4, ...). Se utiliza en el bucle de multiplexación para activar secuencialmente solo un pin de fila a la vez.
- **Patrones (LETRA_S, LETRA_L, NUM_1, etc.):** Son arreglos de 8 bytes, donde cada byte representa el patrón de columnas (datos) de una fila específica. Un bit en 1 indica que el LED en esa posición debe encenderse.

- **uint8_t rowStart = (pad < 2) ? 0 : 4;**: Determina si el bloque de 4x4 está en las filas superiores (0) o inferiores (4).
- **uint8_t colStart = (pad % 2 == 0) ? 0 : 4;**: Determina si el bloque de 4x4 está en las columnas izquierdas (0) o derechas (4).

Función seleccionarNivel()

- Usa dos botones: **BTN_LEVEL** para ciclar entre los números 1, 2 y 3 (usando detección de flanco: if (prev && !curr)), y **BTN_START** para confirmar la selección y salir de la función.
- Utiliza un puntero constante a un arreglo (const uint8_t *bitmap;) para **seleccionar dinámicamente** el patrón del número (NUM_1, NUM_2, NUM_3) a mostrar en la matriz.

Función leerBoton()

- esperarTodosSueltos(): Función de seguridad. Bloquea la ejecución hasta que todos los botones estén en estado liberado (HIGH).
- Bucle de Detección: Espera indefinidamente hasta que un botón sea presionado (LOW).
- **Debounce por Software:** El patrón if (!(p & (1<<BTN0))) { _delay_ms(15); if (!(PINC & (1<<BTN0))) return 0; } realiza un filtro debounce. Después de la primera detección, espera 15 ms y vuelve a leer el pin. Si el pin sigue siendo LOW, confirma que la pulsación es válida y devuelve el índice del pad.

PIC16F887:

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>
#include <stdint.h>

#ifndef F_CPU
#define F_CPU 8000000UL                  // <<< NUEVO: ajusta a 8 MHz
o 16 MHz según tu Proteus
#endif

#define MAX_STEPS 20
#define NUM_PADS  4

#define BTN0       PC0
#define BTN1       PC1
#define BTN2       PC2
#define BTN3       PC3
#define BTN_MASK  ((1<<BTN0)|(1<<BTN1)|(1<<BTN2)|(1<<BTN3))

#define BTN_START  PC4
#define BTN_LEVEL  PC5

// ---------------- COMANDOS AUDIO → PIC ----------------  // <<<
NUEVO
enum {
  CMD_NAV      = 0x10,  // moverse entre opciones
  CMD_OK       = 0x11,  // confirmar
  CMD_START    = 0x12,  // iniciar juego
  CMD_RESET    = 0x13,  // volver al menú

  CMD_SHOW_L1  = 0x21,  // (opcional) anunciar L1
  CMD_SHOW_L2  = 0x22,  // (opcional) anunciar L2
  CMD_SHOW_L3  = 0x23,  // (opcional) anunciar L3

  CMD_LOSE     = 0x40,  // perder
  CMD_WIN      = 0x42,  // ganar L1/L2
  CMD_WIN_L3   = 0x43   // ganar L3 (Megalovania)
};

// ---------------- TX SOFT UART en PB6 @ 9600 bps --------  // <<<
NUEVO
#define TX_DDR   DDRB
#define TX_PORT  PORTB
#define TX_PIN   PB6         // Cambia a PB1 si PB6/PB7 están en
cristal

#define BAUD     9600UL
#define BIT_US   (1000000UL/BAUD)   // 104 us para 9600 bps
```

```
static inline void tx_delay(void){ _delay_us(BIT_US); }
static inline void tx_init(void){
  TX_DDR  |= (1<<TX_PIN);     // salida
  TX_PORT |= (1<<TX_PIN);     // idle alto
}
static inline void tx_write(uint8_t b){
  uint8_t i;
  // start bit (0)
  TX_PORT &= ~(1<<TX_PIN); tx_delay();
  // 8 bits, LSB primero
  for(i=0;i<8;i++){
    if (b & 0x01) TX_PORT |=  (1<<TX_PIN);
    else          TX_PORT &= ~(1<<TX_PIN);
    tx_delay();
    b >>= 1;
  }
  // stop bit (1)
  TX_PORT |= (1<<TX_PIN); tx_delay();
}
static inline void send_cmd(uint8_t c){ tx_write(c); _delay_ms(1); }

// ---------------- LÓGICA DEL JUEGO ----------------
uint8_t secuencia[MAX_STEPS];
uint8_t longitud = 0;
uint8_t dificultad = 1;   // 1..3

void clearMatrix(void)
{
    PORTD = 0x00;
    PORTB = 0xFF;
}

const uint8_t PORTFILAS[8] = {1,2,4,8,16,32,64,128};

const uint8_t LETRA_S[8] = {
    0x00,0x7E,0x7E,0x5A,0x5A,0x5A,0x5A,0x00
};

const uint8_t LETRA_L[8] = {
    0x00,0x7E,0x7E,0X60,0x60,0x60,0x60,0x00
};

const uint8_t NUM_1[8] = {
    0x00,0x40,0x44,0X7E,0x7E,0x40,0x40,0x00
};

const uint8_t NUM_2[8] = {
    0x00,0x66,0x76,0X7E,0x5E,0x4C,0x00,0x00
};
```

```
const uint8_t NUM_3[8] = {
    0x00,0x5A,0x5A,0X5A,0x5A,0x5E,0x7E,0x00
};

// meta por nivel (para disparar WIN y volver al menú)         //
<<< NUEVO
static inline uint8_t goal_for(uint8_t lvl){
    switch(lvl){
        case 1: return 8;
        case 2: return 10;
        default: return 12; // L3
    }
}

void waitForStart(void)
{
    while (1)
    {
        for (uint8_t r = 0; r < 8; r++)
        {
            PORTD = PORTFILAS[r];
            PORTB = ~LETRA_S[r];
            _delay_us(300);
        }

        if (!(PINC & (1 << BTN_START)))
        {
            _delay_ms(80);
            while (!(PINC & (1 << BTN_START)));
            _delay_ms(40);
            clearMatrix();
            send_cmd(CMD_START);                // <<< NUEVO: sonido
de inicio
            return;
        }
    }
}

void showLose(void)
{
    send_cmd(CMD_LOSE);                         // <<< NUEVO: perder
    for (uint16_t t = 0; t < 300; t++)
    {
        for (uint8_t r = 0; r < 8; r++)
        {
            PORTD = PORTFILAS[r];
            PORTB = ~LETRA_L[r];
            _delay_us(100);
        }
    }
    clearMatrix();
```

```
}

void mostrarNumero(uint8_t nivel)
{
    const uint8_t *bitmap;

    if (nivel == 1)      bitmap = NUM_1;
    else if (nivel == 2) bitmap = NUM_2;
    else                 bitmap = NUM_3;

    for (uint8_t r = 0; r < 8; r++)
    {
        PORTD = PORTFILAS[r];
        PORTB = ~bitmap[r];
        _delay_us(100);
    }
}

uint8_t seleccionarNivel(void)
{
    uint8_t nivel = 1;
    uint8_t prev = (PINC & (1 << BTN_LEVEL));

    while (1)
    {
        mostrarNumero(nivel);

        uint8_t p = PINC;
        uint8_t curr = (p & (1 << BTN_LEVEL));

        if (prev && !curr)
        {
            nivel++;
            if (nivel > 3) nivel = 1;
            send_cmd(CMD_NAV);                  // <<< NUEVO: mover
opción
            // (opcional) anunciar nivel:
            //
send_cmd(nivel==1?CMD_SHOW_L1:(nivel==2?CMD_SHOW_L2:CMD_SHOW_L3));
            _delay_ms(60);
        }

        prev = curr;

        if (!(p & (1 << BTN_START)))
        {
            _delay_ms(80);
            while (!(PINC & (1 << BTN_START)));
            _delay_ms(40);
            clearMatrix();
            send_cmd(CMD_OK);                   // <<< NUEVO:
```

```
confirmar nivel
            return nivel;
        }
    }
}

uint8_t randomPad(void)
{
    if (longitud == 0) return rand() % NUM_PADS;

    uint8_t nuevo;
    do {
        nuevo = rand() % NUM_PADS;
    } while (nuevo == secuencia[longitud-1]);

    return nuevo;
}

void agregarPaso(void)
{
    if (longitud < MAX_STEPS)
        secuencia[longitud++] = randomPad();
}

void showPad(uint8_t pad)
{
    uint8_t rowStart = (pad < 2) ? 0 : 4;
    uint8_t colStart = (pad % 2 == 0) ? 0 : 4;

    for (uint16_t t = 0; t < 40; t++)
    {
        for (uint8_t r = rowStart; r < rowStart + 4; r++)
        {
            clearMatrix();
            PORTD |= (1 << r);

            for (uint8_t c = colStart; c < colStart + 4; c++)
                PORTB &= ~(1 << c);

            _delay_us(300);
        }
    }
    clearMatrix();
}

void mostrarSecuencia(void)
{
    for (uint8_t i = 0; i < longitud; i++)
    {
        showPad(secuencia[i]);
```

```
        if (dificultad == 1)      _delay_ms(90);
        else if (dificultad == 2) _delay_ms(45);
        else                      _delay_ms(1);
    }
    clearMatrix();
}

void esperarTodosSueltos(void)
{
    while ((PINC & BTN_MASK) != BTN_MASK);
    _delay_ms(20);
}

uint8_t leerBoton(void)
{
    esperarTodosSueltos();

    while (1)
    {
        uint8_t p = PINC;

        if (!(p & (1<<BTN0))) { _delay_ms(15); if (!(PINC &
(1<<BTN0))) return 0; }
        if (!(p & (1<<BTN1))) { _delay_ms(15); if (!(PINC &
(1<<BTN1))) return 1; }
        if (!(p & (1<<BTN2))) { _delay_ms(15); if (!(PINC &
(1<<BTN2))) return 2; }
        if (!(p & (1<<BTN3))) { _delay_ms(15); if (!(PINC &
(1<<BTN3))) return 3; }
    }
}

uint8_t usuarioCorrecto(void)
{
    for (uint8_t i = 0; i < longitud; i++)
    {
        uint8_t pad = leerBoton();

        showPad(pad);
        clearMatrix();
        _delay_ms(10);

        if (pad != secuencia[i])
            return 0;
    }
    return 1;
}

int main(void)
{
   DDRD = 0xFF;
```

```
    DDRB = 0xFF;

    // --- inicializa TX soft para audio → PIC ---        // <<<
NUEVO
    tx_init();

    DDRC &= ~BTN_MASK;     PORTC |= BTN_MASK;
    DDRC &= ~(1<<BTN_START); PORTC |= (1<<BTN_START);
    DDRC &= ~(1<<BTN_LEVEL); PORTC |= (1<<BTN_LEVEL);

    clearMatrix();
    srand(1);

    while (1)
    {
        waitForStart();
        dificultad = seleccionarNivel();

        longitud = 0;
        agregarPaso();

        while (1)
        {
            mostrarSecuencia();

            if (usuarioCorrecto())
            {
                agregarPaso();

                // ¿ya alcanzó la meta? envía WIN y regresa a menú
// <<< NUEVO
                if (longitud > goal_for(dificultad)) {
                    if (dificultad == 3) send_cmd(CMD_WIN_L3);
                    else                 send_cmd(CMD_WIN);
                    _delay_ms(50);
                    break; // vuelve a seleccionar nivel
                }
            }
            else
            {
                showLose();
                send_cmd(CMD_RESET);     // <<< NUEVO: sonido de
volver al menú
                break;
            }
        }
    }
}
```

- La enumeración typedef enum { N_C=0, N_CS, ... } NoteName; coloca un índice numérico (0 a 11) a cada not, haciendo que la partitura sea legible.
- El arreglo bidimensional constante const unsigned int NOTE_FREQ[...] es la tabla de frecuencias. Almacena la frecuencia exacta para cada nota en las octavas C2 a B7, definiendo el timbre del sistema. La función unsigned int NoteFreq(NoteName n, unsigned char oct) es la herramienta clave para buscar en esta tabla. Usa la expresión oct - OCT_MIN para calcular el índice correcto de la fila (octava) dentro del arreglo, asegurando que el par (Nota, Octava) se traduzca siempre a la frecuencia correcta
- La estructura typedef struct { signed char note; unsigned char octave; unsigned int dur_ms; } Partitura; define el bloque de datos musical. Es crucial que el campo note sea un signed char porque se utiliza el valor -1 para representar un silencio, diferenciándolo de cualquier nota válida (0 a 11).
- La función void PlayPartitura(const Partitura* p, unsigned char count) es la máquina de reproducción. Itera sobre el arreglo de la partitura y usa la condición if (p[i].note < 0) para decidir si llamar a Rest() (silencio) o a PlayNote() (tono).
- La función void Rest(unsigned int dur_ms) gestiona los silencios. Está optimizada para compensar las limitaciones de algunos compiladores (como MikroC), que prefieren argumentos constantes en Delay_ms, dividiendo el retardo variable en bloques de 10 ms y 1 ms.
- Las secuencias const Partitura SFX_Save[], SFX_Damage_body[], etc., son las melodías pregrabadas para los eventos del juego.

- El sonido de fallo (SFX_Lose_Trombone) utiliza una estructura rítmica alternando notas y silencios para simular el efecto "wa-wa" de trombón. Sin embargo existen limitaciones al ser un instrumento de viento de sonido tan característico.
- void PlayNoiseBurst(unsigned char grains) genera pseudo-ruido de daño. Llama repetidamente a Sound_Play con frecuencias y duraciones generadas aleatoriamente (rand()), que superpuestas simulan un *click.*

La función void main() en el código es como un banco de pruebas que:

- Configura las entradas/salidas (TRISB = 0xF8;).
- Las sentencias if (Button(&PORTB,7,1,1)) prueban si se ha presionado un botón en PORTB.
- Las funciones de melodía (Tone1_Ok(), Melody_Lose(), etc.) están relacionadas con estas pulsaciones de botón.

Descripción del esquema de comunicación

El sistema se compone de dos micros con roles diferenciados: ATmega328P como “motor de juego” (gestiona la matriz LED 8×8, genera la secuencia, lee botones y evalúa aciertos/errores) y PIC16F887 como “motor de audio” (reproduce efectos y melodías). La sincronización entre lo visual y lo sonoro se realiza mediante una comunicación UART unidireccional desde el ATmega hacia el PIC.

Simplicidad y bajo costo en pines: un único hilo UART y GND bastan para sincronizar audio con la lógica del juego; el uso del MAX7219 reduce la matriz a solo 3 pines del ATmega, liberando PORTD para botones y UART. La UART hardware garantiza temporización precisa (9600 bps) independientemente del refresco de la matriz o de la lectura de botones.

Desacoplo claro de funciones: el ATmega opera una máquina de estados del juego; el PIC se concentra en audio (partituras, efectos y duraciones), evitando mezclar tareas.

Conclusiones

- El código utiliza un control de matriz altamente optimizado en `showPad()`, calculando dinámicamente `rowStart` y `colStart`. Esto reduce el ciclo de multiplexación a solo las cuatro filas y cuatro columnas del bloque activo (4x4), en lugar de escanear la matriz completa de 8x8. Esto minimiza la sobrecarga de la CPU y garantiza una respuesta visual rápida para los destellos.

    - La dificultad se administra directamente a través de un retardo variable dentro de `mostrarSecuencia()`. La función `seleccionarNivel()` utiliza un mecanismo de detección de flanco (`prev && !curr`) para cambiar limpiamente la dificultad con cada pulsación del botón `BTN_LEVEL`, controlando así el tiempo disponible para la memorización del jugador (de 90 ms a 1 ms).

- El código maneja la entrada de botones de manera fiable utilizando dos técnicas: Debounce por Software (doble verificación con `_delay_ms(15);` en `leerBoton()`) para filtrar el ruido eléctrico, y Sincronización de Turnos (`esperarTodosSueltos()`) para bloquear la lectura hasta que el jugador haya liberado todos los botones, lo que previene errores de múltiples pulsaciones entre turnos del juego.