Tarea /Taller Grupal

Integrantes:

- Juan Amoretti
- Nicole Azú
- Óscar Muñoz
- José Vera

1. **Objetivo General**
   - Desarrollar un juego de adivinanza interactivo utilizando una matriz LED 8x8 y un sistema de reproducción de melodías mediante la comunicación entre los microcontroladores ATmega328P y PIC16F887, con el fin de integrar experiencias visuales y auditivas que refuercen el aprendizaje en programación y control de sistemas embebidos.

2. **Objetivos Específicos**
   - Configurar la matriz LED 8x8 para mostrar caracteres y símbolos que guíen al usuario en el juego de adivinanza, asegurando la claridad en la presentación de números y símbolos en el sistema.

   - Programar melodías en el PIC16F887 para señalizar aciertos o errores en el juego, brindando retroalimentación auditiva que complemente la experiencia visual.

   - Implementar la comunicación entre el ATmega328P y el PIC16F887 para la transmisión y verificación de los datos de la adivinanza, integrando la selección numérica del usuario.

3. **Recursos**
   - Proteus
   - Visual Studio Code
   - MirkoC for PIC
   - ATmega328P
   - PIC16F887
   - Matriz LED 8x8
   - Botones de entrada
   - Bocina

4. Descripción

El proyecto consiste en implementar un juego de adivinanza utilizando una matriz LED 8x8 controlada por un microcontrolador ATmega328P y un PIC16F887. El objetivo del juego es adivinar un número aleatorio generado por el PIC16F887, que se muestra en la matriz LED del ATmega328P. El usuario selecciona un número a través de botones, y el número elegido es comparado con el número aleatorio generado. Si el usuario acierta, se reproduce una melodía de victoria; en caso contrario, se emite una melodía de derrota. La comunicación entre el ATmega328P y el PIC16F887 se realiza mediante pines específicos, y se utiliza una librería de sonido para la generación de las melodías.

5. Código utilizado

ATmega328P

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

// Pines de comunicación con el PIC16F887
#define RD5_PIN   PC5
#define START_PIC PC0

// Arreglos para mensajes
unsigned char signo[8] = {0x0, 0x04, 0x02, 0x01, 0xB1, 0x0A, 0x04, 0x0}; //
Mensaje "?"
unsigned char PERDER[8] = {0x81, 0xC3, 0x66, 0x18, 0x18, 0x66, 0xC3, 0x81}; //
"X"
unsigned char EMPECEMOS[80] = { // Mensaje "EMPECEMOS"
    0x0, 0x7E, 0x7E, 0x5A, 0x5A, 0x5A, 0x5A, 0x0, // E
    0x0, 0x7E, 0x04, 0x08, 0x08, 0x04, 0x7E, 0x0, // M
    0x0, 0x7E, 0x7E, 0x12, 0x12, 0x1E, 0x1E, 0x0, // P
    0x0, 0x7E, 0x7E, 0x5A, 0x5A, 0x5A, 0x5A, 0x0, // E
    0x0, 0x7E, 0x7E, 0x42, 0x42, 0x42, 0x42, 0x0, // C
    0x0, 0x7E, 0x7E, 0x5A, 0x5A, 0x5A, 0x5A, 0x0, // E
    0x0, 0x7E, 0x04, 0x08, 0x08, 0x04, 0x7E, 0x0, // M
    0x0, 0x7E, 0x7E, 0x66, 0x66, 0x7E, 0x7E, 0x0, // O
    0x0, 0x4E, 0x4E, 0x5A, 0x5A, 0x72, 0x72, 0x0, // s
    0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0        // ESPACIO
};

// Arreglo para fijar filas
unsigned char PORT[8] = {1, 2, 4, 8, 16, 32, 64, 128};

// Arreglos para numéros
unsigned char uno[8] = {0x0, 0x0, 0x81, 0xFF, 0xFF, 0x80, 0x0, 0x0};
unsigned char dos[8] = {0x0, 0xC2, 0xE3, 0xB1, 0x99, 0x8F, 0x86, 0x0};
```

```
unsigned char tres[8] = {0x0, 0xC3, 0x99, 0x99, 0x99, 0xFF, 0xFF, 0x0};
unsigned char cuatro[8] = {0x0, 0x1F, 0x1F, 0x18, 0x18, 0xFF, 0xFF, 0x0};
unsigned char cinco[8] = {0x0, 0x8F, 0x8F, 0x89, 0x89, 0xF9, 0xF9, 0x0};
unsigned char seis[8] = {0x0, 0xFF, 0xFF, 0x91, 0x91, 0xF1, 0xF3, 0x0};
unsigned char siete[8] = {0x0, 0x03, 0xE1, 0xF1, 0x19, 0x0D, 0x07, 0x0};
unsigned char ocho[8] = {0x0, 0xF7, 0xFF, 0x99, 0x99, 0xFF, 0xF7, 0x0};
unsigned char nueve[8] = {0x0, 0x0F, 0x0F, 0x09, 0x09, 0xFF, 0xFF, 0x0};

// Función para mostrar una imagen en la matriz de LEDs
void mostrarMatriz(int side, const unsigned char* imagen) {
    if (side > 8) {
        for (int i = 0; i < side - 8; i++) {
            for (int k = 0; k < 50; k++) {
                for (int j = 0; j < 8; j++) {
                    PORTD = PORT[j]; // Fijar fila
                    PORTB = ~imagen[i + j]; // Mostrar columna
                    _delay_ms(0.05);
                }
            }
        }
    }
    if (side == 8) {
        for (int k = 0; k < 50; k++) {
            for (int j = 0; j < 8; j++) {
                PORTD = PORT[j]; // Fijar fila
                PORTB = ~imagen[j]; // Mostrar columna
                _delay_ms(0.05);
            }
        }
    }
}

// Función para convertir de binario a decimal (pines PC1 a PC4)
unsigned char binarioADecimal() {
    return ((PINC & 0x1E) >> 1);  // Convertir PC1-PC4 a decimal
}

int main(void) {
    // Configuración de pines
    DDRB = 0xFF;  // Configurar PORTB como salida (columnas de la matriz LED)
    DDRD = 0xFF;  // Configurar PORTD como salida (filas de la matriz LED)
    DDRC = 0x01;  // Configurar PORTC como entrada menos PC0 (comunicación con
el PIC16F887)
    PORTC = 0x00; // Desactivar pull-ups en PORTC

    // Mostrar mensaje "EMPECEMOS" y "?"
    mostrarMatriz(80, EMPECEMOS);
    mostrarMatriz(8, signo);
```

```
// Inicializamos el PIC con START_PIC (PC0) con el valor de 1
PORTC |= (1 << START_PIC);

while (1) {
    // Si RD5_PIN es 1, mostrar número y cara feliz
    unsigned char numero = binarioADecimal();

    // Leer el valor de RD5_PIN (PC5) del PIC
    if (PINC & (1 << RD5_PIN)) {
        switch (numero) {
            case 1:
                mostrarMatriz(8, uno); // Mostrar el 1
                break;

            case 2:
                mostrarMatriz(8, dos); // Mostrar el 2
                break;

            case 3:
                mostrarMatriz(8, tres); // Mostrar el 3
                break;

            case 4:
                mostrarMatriz(8, cuatro); // Mostrar el 4
                break;

            case 5:
                mostrarMatriz(8, cinco); // Mostrar el 5
                break;

            case 6:
                mostrarMatriz(8, seis); // Mostrar el 6
                break;

            case 7:
                mostrarMatriz(8, siete); // Mostrar el 7
                break;

            case 8:
                mostrarMatriz(8, ocho); // Mostrar el 8
                break;

            case 9:
                mostrarMatriz(8, nueve); // Mostrar el 9
                break;

            default:
```

```
                    PORTB = 0xFF;  // Apagar todas las columnas
                    PORTD = 0x00;  // Apagar todas las filas
                    break;
                }
            }
            else if (!(PINC & (1 << RD5_PIN)) && numero != 0) mostrarMatriz(8,
PERDER);
            else mostrarMatriz(8, signo); // Si RD5_PIN es 0, mantener el signo
"?"
        }
        return 0;
}
```

PIC16F887

```
/* Juego de adivinanza para el PIC16F887 en MikroC for PIC */

unsigned short numSelec = 0;
unsigned short randomNumber = 1;

// Funciones para cada nota
void sol(int duration) {
    Sound_Play(392.00, duration);
}

void la_sharp(int duration) {
    Sound_Play(466.16, duration);
}

void do_prime(int duration) {
    Sound_Play(523.25, duration);
}

void re_prime(int duration) {
    Sound_Play(587.33, duration);
}

void re_sharp_prime(int duration) {
    Sound_Play(622.25, duration);
}

void fa_prime(int duration) {
    Sound_Play(698.46, duration);
}

void sol_prime(int duration) {
    Sound_Play(783.99, duration);
}
```

```
void la_prime(int duration) {
    Sound_Play(880.00, duration);
}

// Función para reproducir la melodía completa
void Melody_Win() {
    int duration_short = 200;
    int duration_standard = 400;
    int duration_long = 600;

    do_prime(duration_standard);  // C
    la_prime(duration_short);      // A
    sol(duration_short);           // G
    la_prime(duration_standard);    // A
    do_prime(duration_long);       // C
    sol(duration_short);           // G
    la_prime(duration_short);      // A
    do_prime(duration_standard);    // C
    re_prime(duration_short);      // D
    re_prime(duration_standard);    // D
    la_prime(duration_short);       // A
    sol(duration_short);           // G
    fa_prime(duration_long);       // F
    sol(duration_short);           // G
    do_prime(duration_standard);    // C
    la_prime(duration_short);      // A
    sol(duration_short);           // G
    la_prime(duration_standard);    // A
    do_prime(duration_long);       // C
    sol(duration_short);           // G
    la_prime(duration_short);      // A
    do_prime(duration_standard);    // C
    re_prime(duration_short);      // D
    re_prime(duration_standard);    // D
    la_prime(duration_short);       // A
    sol(duration_short);           // G
    fa_prime(duration_long);       // F
    sol(duration_short);           // G
    do_prime(duration_long);       // C
}

void ToneA() {
  Sound_Play(880, 50);
}

void ToneC() {
  Sound_Play(1046, 50);
```

```
}

void ToneE() {
  Sound_Play(1318, 50);
}

void Melody_Lose() {
  unsigned short i;
  for (i = 9; i > 0; i--) {
    ToneA();
    ToneC();
    ToneE();
  }
}

void main() {
  // Configuraciones iniciales
  ANSEL  = 0;
  ANSELH = 0;
  C1ON_bit = 0;
  C2ON_bit = 0;

  TRISB  = 0xFF;
  TRISA  = 0xFF;
  TRISC  = 0x00;
  TRISD  = 0x00;

  Sound_Init(&PORTC, 3);
  srand(1);

  // Enviar 0 por RD1-RD4
  PORTD = (PORTD & 0x01) | (numSelec << 1);

  while (1) {
    // Detectar botón de selección de número
    if (Button(&PORTA, 0, 1, 1)) numSelec = 1;
    while(RA0_bit);
    if (Button(&PORTA, 1, 1, 1)) numSelec = 2;
    while(RA1_bit);
    if (Button(&PORTB, 0, 1, 1)) numSelec = 3;
    while(RB0_bit);
    if (Button(&PORTB, 1, 1, 1)) numSelec = 4;
    while(RB1_bit);
    if (Button(&PORTB, 2, 1, 1)) numSelec = 5;
    while(RB2_bit);
    if (Button(&PORTB, 3, 1, 1)) numSelec = 6;
    while(RB3_bit);
    if (Button(&PORTB, 4, 1, 1)) numSelec = 7;
```

```
    while(RB4_bit);
    if (Button(&PORTB, 5, 1, 1)) numSelec = 8;
    while(RB5_bit);
    if (Button(&PORTB, 6, 1, 1)) numSelec = 9;
    while(RB6_bit);

    // Enviar el número seleccionado por RD1-RD4
    PORTD = (PORTD & 0x01) | (numSelec << 1);

    // Generar el número aleatorio después de soltar el botón
    randomNumber = 1 + rand() % 9;

    // Mostrar el número aleatorio en binario en RC4-RC7
    PORTC = (PORTC & 0x0F) | (randomNumber << 4);

    // Comparación y reacción
    if (numSelec == randomNumber) {
       PORTD |= (1 << 5);
       Melody_Win(); // Reproducir melodía de victoria
    }
    if (!(numSelec == randomNumber) && numSelec != 0) {
       PORTD &= ~(1 << 5);
       Melody_Lose(); // Reproducir melodía de derrota
    }
    numSelec = 0;
    Delay_ms(1000); // Esperar 1 segundo antes de la próxima ronda
  }
}
```

6. Evidencias de la simulación

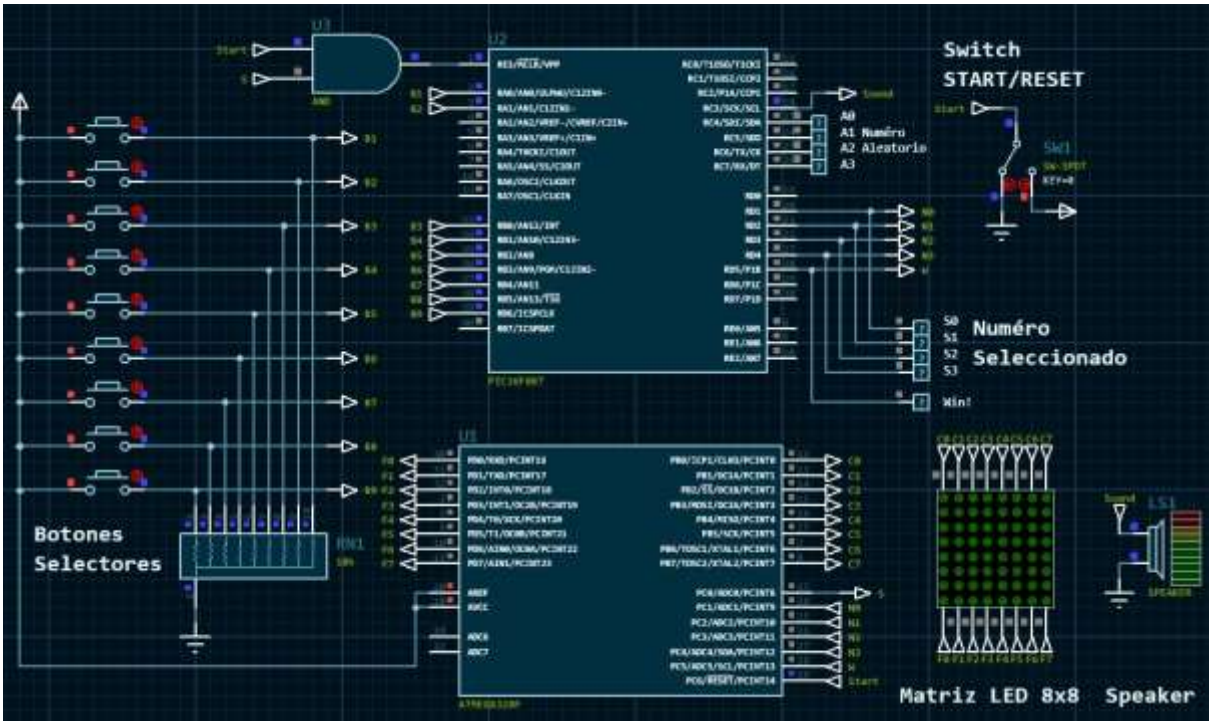

Figura 1. Esquemático del programa

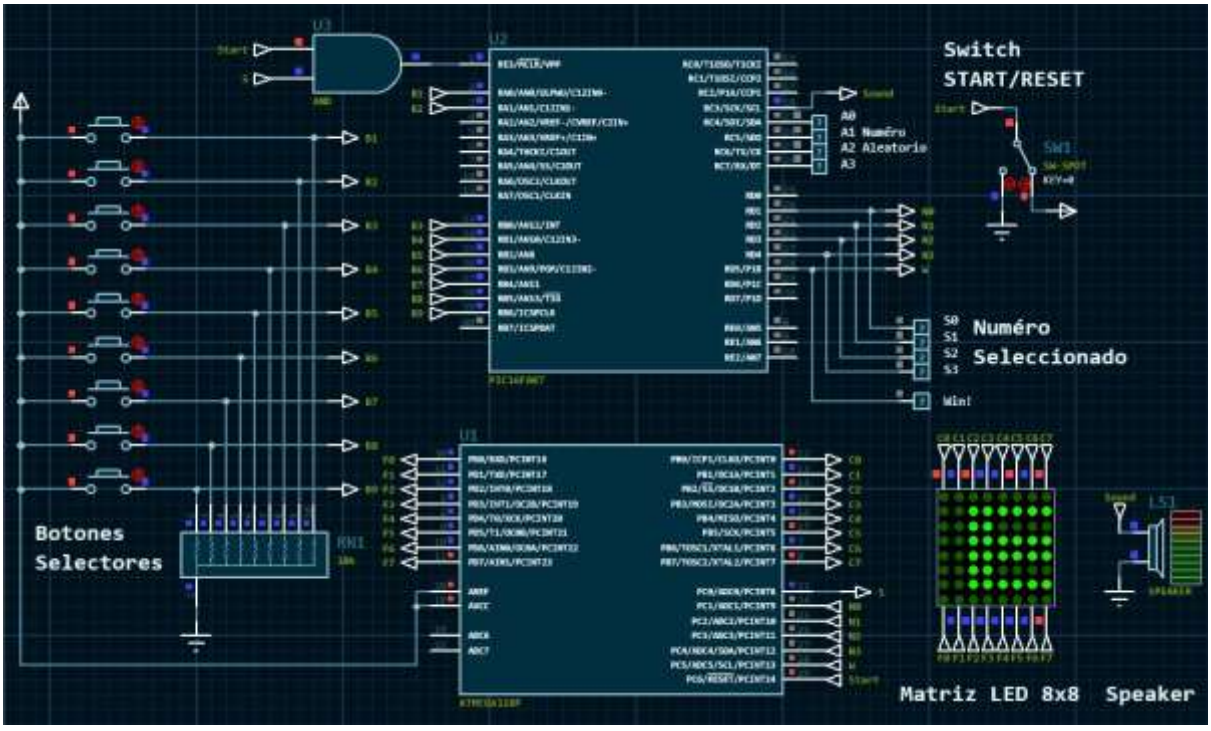

Figura 2. Esquemático del programa mostrando menaje "EMPECEMOS ".

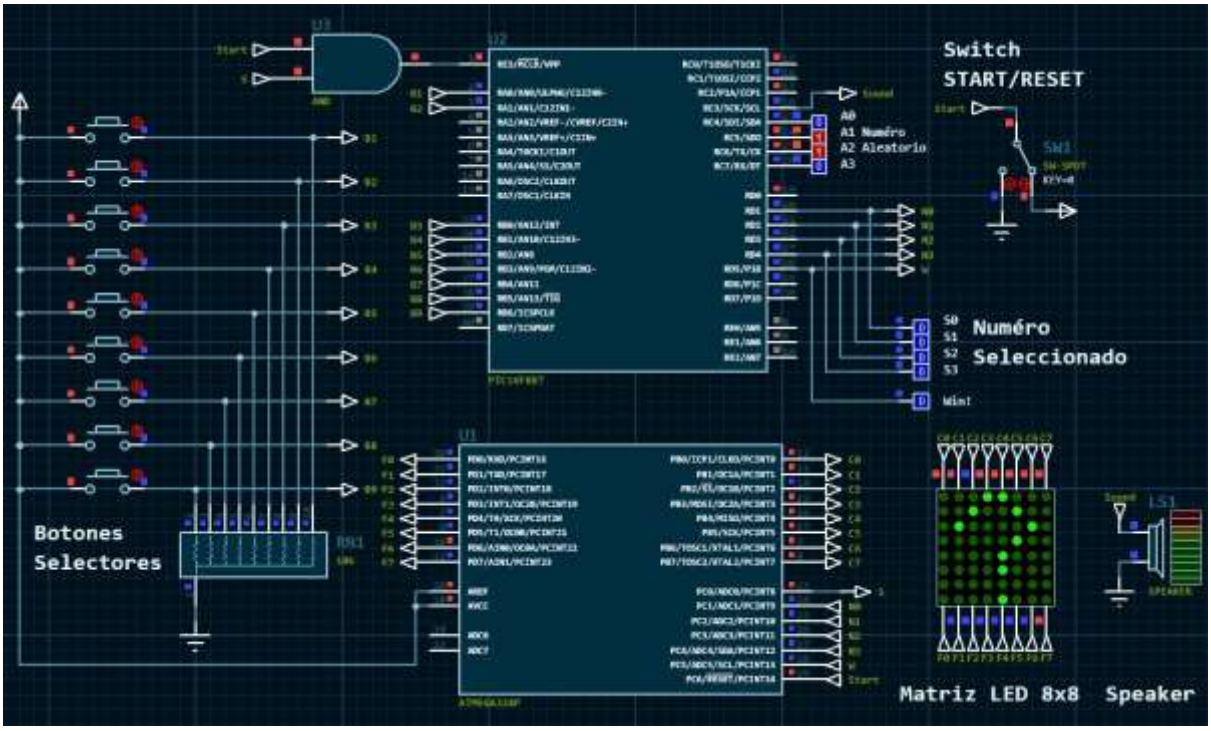

Figura 3. Esquemático del programa cuando el usuario esta adivinando el número.

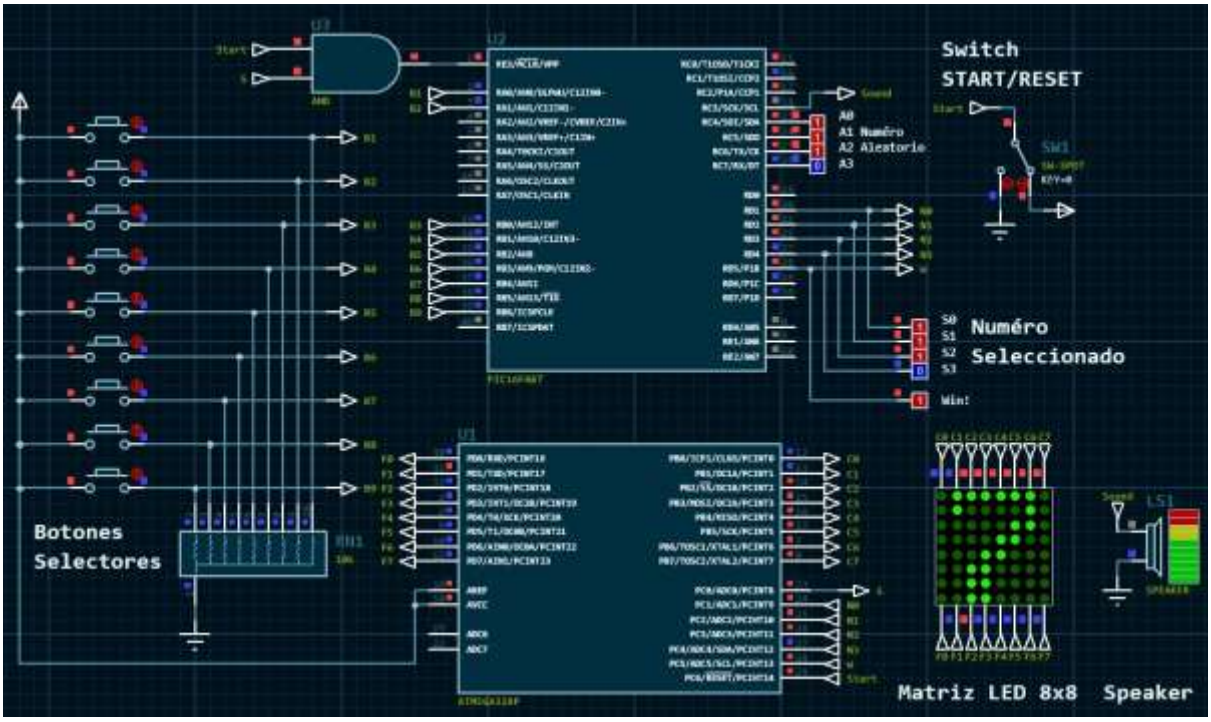

Figura 4. Esquemático del programa si el usuario GANA.

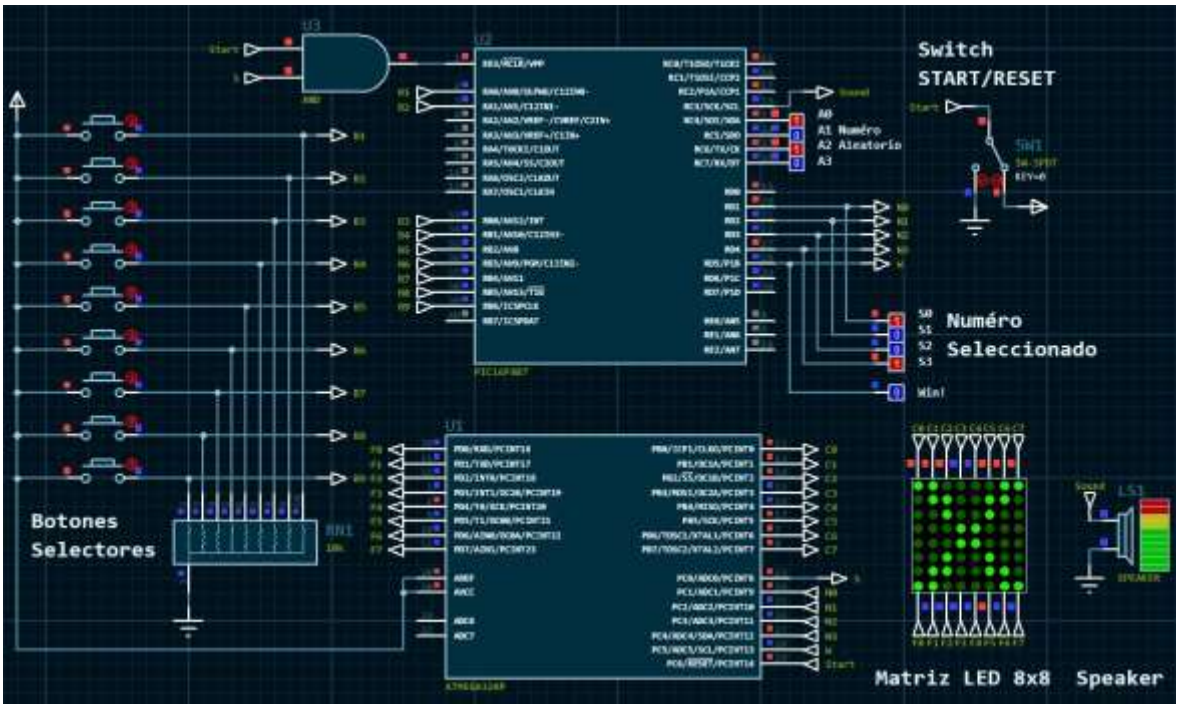

Figura 5. Esquemático del programa si el usuario PIERDE.

7. Enlace de GitHub

https://github.com/osmuburg/Taller-1-Grupo-2.git

8. Conclusiones

- El programa demuestra la capacidad de integrar múltiples microcontroladores en una misma aplicación, logrando una comunicación efectiva entre el ATmega328P y el PIC16F887. Esta comunicación permite una sincronización eficiente entre la visualización en la matriz LED y la generación de melodías.
- La implementación de botones para la selección de números y las melodías para indicar el resultado de la adivinanza hacen que el juego sea interactivo y divertido. Sin embargo, el juego podría mejorarse aún más añadiendo un sistema de puntuación o niveles de dificultad, lo que aumentaría su re-jugabilidad.
- El uso de puertos digitales para la entrada (botones) y la salida (sonido y visualización del número en binario) está bien estructurado. Sin embargo, se debe considerar la posibilidad de que el usuario pueda seleccionar un número rápidamente y que el sistema maneje debidamente las entradas múltiples o rápidas para evitar lecturas incorrectas o duplicadas. Esto se podría lograr implementando un pequeño retardo o un debounce para los botones.

9. Recomendaciones

- Se recomienda optar por el uso de protocolos de comunicación seriales, como I^2C (Inter-Integrated Circuit), para garantizar una transferencia de datos más robusta y sincronizada entre los microcontroladores. La implementación de I^2C permitiría que ambos dispositivos manipulen y compartan datos de forma sincrónica, asegurando que operen de manera coordinada y simultánea. Esto ayudaría a minimizar los posibles retrasos en la transmisión de datos entre ambos microcontroladores, permitiendo así un flujo continuo y estable de información sin interferencias.

- Se recomienda utilizar módulos adicionales que mejoren la presentación y funcionalidad del juego, tales como módulos amplificadores de audio para mejorar la calidad y el volumen del sonido emitido. Además, el uso de matrices LED 8x8 con controladores integrados permitiría reducir la cantidad de pines necesarios para la conexión con el microcontrolador, facilitando el diseño del sistema y optimizando el uso de recursos. Estas mejoras no solo optimizarían la calidad de la experiencia del usuario, sino que también simplificarían el control del hardware involucrado en el proyecto.