Comunicación entre Microcontroladores

Grupo #3

Paralelo: 2

Docente:

Ronald David Solis Mesa

Nombre:

Newton Toledo

Gabriel Reyes

Ronald Cordero

Eliezer Acebo

Objetivo general

Desarrollar un videojuego interactivo en una matriz LED 8x8 utilizando dos microcontroladores (ATmega328P y PIC16F887), que permita al usuario superar niveles con obstáculos ocultos (minas), integrando señales visuales y auditivas mediante comunicación entre dispositivos.

Objetivos específicos

- Implementar la lógica del juego y el control de movimiento del cursor utilizando el microcontrolador ATmega328P y botones físicos.
- Diseñar la detección de eventos clave (nivel superado, derrota y victoria) e integrarla con salidas visuales en la matriz LED.
- Establecer comunicación digital entre el ATmega328P y el PIC16F887 para activar la reproducción de sonidos personalizados en cada evento del juego.
- Programar melodías musicales representativas para cada estado del juego, mejorando la experiencia del usuario.
- Simular y validar el funcionamiento completo del sistema en Proteus para comprobar la integración entre hardware y software.

Descripción técnica completa del juego

El juego presentado en este informe consiste en una versión simplificada del buscaminas, en el cual se tiene dos microcontroladores (ATMEGA328P y PIC16F887), una matriz LED 8x8 y cuatro pulsadores. Se empieza el juego en la parte inferior de la matriz con nuestro cursor representado por un punto, y mediante los botones tenemos que mover el cursor para llegar a la meta que está arriba en el centro.

Su dificultad radica en que hay minas colocadas en cada nivel, por lo que, si alcanzamos alguna, perdemos el nivel y hay que realizarlo nuevamente; cabe mencionar que existen tres niveles dentro del juego, mientras más alto sea el nivel tendremos más minas ocultas en el campo de la matriz.

Al pasar cada nivel tendremos dibujos y audios que representan el completado del nivel con éxito, así como para completar el juego y cada derrota registrada.

Capturas de la simulación en Proteus

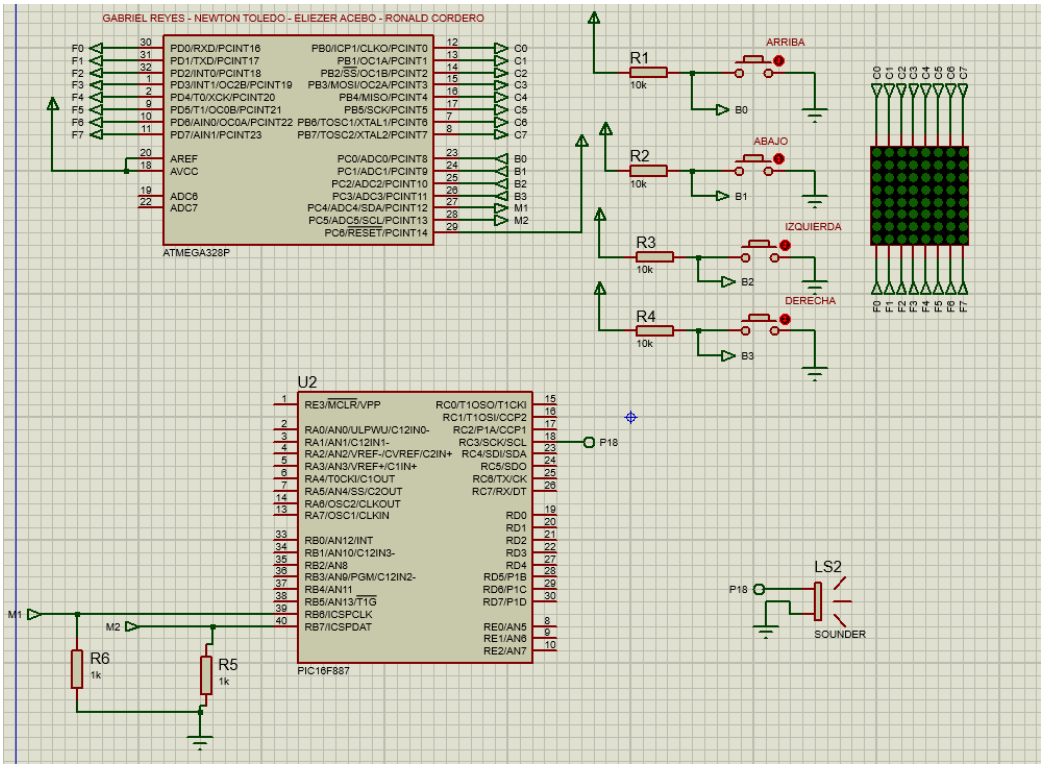

Ilustración 1. Circuito simulado en Proteus

Explicación del código para cada microcontrolador

ATMEGA328P (Estructura del juego):

En este microcontrolador se llevó a cabo todo el código relacionado con la estructura del juego y contenido visual que se deseaba mostrar. Por ello, se empezó definiendo las entradas de los botones y los bits correspondientes a filas y columnas de la matriz led (se usó multiplexación en lugar del driver) junto con las ilustraciones que se deseaba mostrar, como el indicador de nivel, la X de derrota y la cara feliz de juego completado.

Adicionalmente, se genera un punto en el centro inferior de la matriz, el cual funcionará como nuestro cursor en el juego. Por ello, en el código se inicializan funciones con bucles for para poder mover dicho cursor en cualquier dirección de acuerdo a los botones que se presionen. Además, se crea un pequeño bucle para tener una imagen continua del cursor en el tiempo y se escriben aquellas funciones que llaman a los arreglos de las ilustraciones antes mencionadas.

En la rama principal (main), se genera el orden de aparición de los elementos del juego y se inicializan dos pines adicionales que servirán para la comunicación con el PIC16F887. En cuanto al bucle principal, se establecen las condiciones de cumplimiento de cada nivel, las cuales contienen el arreglo de las minas que se generarán en cada dificultad; recordemos que cada condición plantea que si se llega a una mina se muestra la imagen

de derrota y se vuelve a cargar el mismo nivel, caso contrario se avanza al siguiente hasta terminar el nivel 3 con la ilustración de victoria y reiniciando el juego.

Es importante recalcar que los pines de comunicación enviarás una combinación de bits específica que el otro microcontrolador interpretará como avance de nivel, derrota o victoria; con la finalidad de reproducir las canciones deseadas.

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

#define F_CPU 8000000UL  // 8 MHz

// Botones en PC0–PC3
#define BUTTON0 (1 << PC0)  // Arriba
#define BUTTON1 (1 << PC1)  // Abajo
#define BUTTON2 (1 << PC2)  // Izquierda
#define BUTTON3 (1 << PC3)  // Derecha

// Máscaras para filas (PORTB) y columnas (PORTD)
static const uint8_t mascaraFilas[8]    = {1,2,4,8,16,32,64,128};
static const uint8_t mascaraColumnas[8] = {1,2,4,8,16,32,64,128};
static uint8_t nivel = 1;

// Buffer que contiene el punto: un solo bit alto en cada fila
static uint8_t frame[8];

// Patrón de “PERDER” (dos diagonales)
static const uint8_t patrX[8] = {
   0x00, 0x62, 0x14, 0x08, 0x08, 0x14, 0x62, 0x00 //X PERDI
};

// Patrón de “NIVEL 1”
static const uint8_t patrL1[8] = {
   0x00, 0x3E, 0x20, 0x00, 0x24, 0x3E, 0x20, 0x00 //NIVEL 1
};

// Patrón de “NIVEL 2”
static const uint8_t patrL2[8] = {
   0x00, 0x3E, 0x20, 0x00, 0x3A, 0x2A, 0x2E, 0x00 //NIVEL 1
};
```

```
// Patrón de “NIVEL 3”
static const uint8_t patrL3[8] = {
   0x00, 0x3E, 0x20, 0x00, 0x2A, 0x2A, 0x3E, 0x00 //NIVEL 1
};

// Patrón de “GANAR”
static const uint8_t patrG[8] = {
   0x00, 0x20, 0x46, 0x50, 0x50, 0x46, 0x20, 0x00 //NIVEL 1
};

// Lectura simple (sin debounce) de botones
int8_t leerBotones() {
   if (!(PINC & BUTTON0)) return 0;
   if (!(PINC & BUTTON1)) return 1;
   if (!(PINC & BUTTON2)) return 2;
   if (!(PINC & BUTTON3)) return 3;
   return -1;
}

// Funciones de movimiento sobre el buffer `frame[]`
void mover_derecha(uint8_t buf[8]) {
   for (int i = 7; i > 0; i--) buf[i] = buf[i-1];
   buf[0] = 0;
}
void mover_izquierda(uint8_t buf[8]) {
   for (int i = 0; i < 7; i++) buf[i] = buf[i+1];
   buf[7] = 0;
}
void mover_arriba(uint8_t buf[8]) {
   for (int i = 0; i < 8; i++) buf[i] = buf[i] >> 1;
}
void mover_abajo(uint8_t buf[8]) {
   for (int i = 0; i < 8; i++) buf[i] = buf[i] << 1;
}

// Refresca el display mostrando `frame[]` permanentemente
```

```
void refrescar(uint8_t buf[8]) {
  for (int col = 0; col < 8; col++) {
    PORTD = mascaraColumnas[col];
    PORTB = ~buf[col];
    _delay_us(100);
  }
}

// Muestra la X gigante durante 1 segundo
void mostrarX() {
  uint16_t elapsed = 0;
  while (elapsed < 100) {        // 1000 ms = 1 segundo
    refrescar((uint8_t*)patrX);
    _delay_ms(10);
    elapsed += 10;
  }
}

void mostrarL1() {
  uint16_t elapsed = 0;
  while (elapsed < 100) {        // 1000 ms = 1 segundo
    refrescar((uint8_t*)patrL1);
    _delay_ms(10);
    elapsed += 10;
  }
}

void mostrarL2() {
  uint16_t elapsed = 0;
  while (elapsed < 100) {        // 1000 ms = 1 segundo
    refrescar((uint8_t*)patrL2);
    _delay_ms(10);
    elapsed += 10;
  }
}

void mostrarL3() {
```

```
    uint16_t elapsed = 0;
    while (elapsed < 100) {         // 1000 ms = 1 segundo
        refrescar((uint8_t*)patrL3);
        _delay_ms(10);
        elapsed += 10;
    }
}

void mostrarG() {
    uint16_t elapsed = 0;
    while (elapsed < 100) {         // 1000 ms = 1 segundo
        refrescar((uint8_t*)patrG);
        _delay_ms(45);
        elapsed += 10;
    }
}

int main(void) {
    DDRB = 0xFF;  // filas por PORTB
    DDRD = 0xFF;  // columnas por PORTD
    DDRC &= ~(BUTTON0|BUTTON1|BUTTON2|BUTTON3);
    PORTC |=  (BUTTON0|BUTTON1|BUTTON2|BUTTON3);
    DDRC |= (1 << PC4);     // Configura PC4 como salida
    PORTC &= ~(1 << PC4);
    DDRC |= (1 << PC5);     // Configura PC4 como salida
    PORTC &= ~(1 << PC5);

    mostrarL1();

    // Posición inicial: fila 7 (bajo), bit 3 (centro horizontal)
    for (int i = 0; i < 8; i++) frame[i] = 0;
    frame[3] = (1 << 7);

    while (1) {
        int8_t dir = leerBotones();
        if (dir != -1) {
            // detectar posición actual
```

```
int colPos=-1, rowPos=-1;
for(int c=0;c<8;c++){
  if(frame[c]){
    colPos=c;
    for(int b=0;b<8;b++){
      if(frame[c] & (1<<b)){
        rowPos=b;
        break;
      }
    }
    break;
  }
}

// mover solo un paso si no en borde
switch (dir) {
  case 0: if(rowPos>0) mover_arriba(frame);   break;
  case 1: if(rowPos<7) mover_abajo(frame);    break;
  case 2: if(colPos>0) mover_izquierda(frame);break;
  case 3: if(colPos<7) mover_derecha(frame);  break;
}

// comprobar minas en esquinas
colPos=-1; rowPos=-1;
for(int c=0;c<8;c++){
  if(frame[c]){
    colPos=c;
    for(int b=0;b<8;b++){
      if(frame[c] & (1<<b)){
        rowPos=b;
        break;
      }
    }
    break;
  }
}
```

```
if (rowPos == 0 && colPos == 3) {
    nivel = nivel +1;

    if(nivel==2){
        PORTC |= (1 << PC4);    // Enviar señal HIGH al PIC
        _delay_ms(100);         // Espera 100 ms
        PORTC &= ~(1 << PC4);   // Volver a LOW
        mostrarL2();        // muestra “NIVEL 1” como ejemplo de victoria
        _delay_ms(50);      // déjalo visible medio segundo
        // reiniciar posición al inicio
        for (int i = 0; i < 8; i++) frame[i] = 0;
        frame[3] = (1 << 7);
    }
    if(nivel==3){
        PORTC |= (1 << PC4);    // Enviar señal HIGH al PIC
        _delay_ms(100);         // Espera 100 ms
        PORTC &= ~(1 << PC4);   // Volver a LOW
        mostrarL3();        // muestra “NIVEL 1” como ejemplo de victoria
        _delay_ms(50);      // déjalo visible medio segundo
        // reiniciar posición al inicio
        for (int i = 0; i < 8; i++) frame[i] = 0;
        frame[3] = (1 << 7);
    }
    if(nivel==4){
        PORTC |= (1 << PC4);    // Enviar señal HIGH al PIC
        PORTC |= (1 << PC5);
        _delay_ms(100);         // Espera 100 ms
        PORTC &= ~(1 << PC4);   // Volver a LOW
        PORTC &= ~(1 << PC5);
        mostrarG();        // muestra “NIVEL 1” como ejemplo de victoria
        _delay_ms(50);      // déjalo visible medio segundo
        // reiniciar posición al inicio
        mostrarL1();
        for (int i = 0; i < 8; i++) frame[i] = 0;
        frame[3] = (1 << 7);
        nivel = 1;
    }
```

```
}

if (nivel==2){
   if ((rowPos==0 && colPos==0) ||
      (rowPos==0 && colPos==7) ||
      (rowPos==7 && colPos==0) ||
      (rowPos==7 && colPos==7) ||
      (rowPos==0 && colPos==1) ||
      (rowPos==1 && colPos==6) ||
      (rowPos==5 && colPos==1) ||
      (rowPos==0 && colPos==4) ||
      (rowPos==0 && colPos==7) ||
      (rowPos==7 && colPos==0) ||
      (rowPos==7 && colPos==7))
      {
      PORTC |= (1 << PC5);
      _delay_ms(100);         // Espera 100 ms
      PORTC &= ~(1 << PC5);
         nivel = 1;
         mostrarX();
         _delay_ms(50);
         mostrarL1();
         for (int i = 0; i < 8; i++) frame[i] = 0;
         frame[3] = (1 << 7);
      }

}

if (nivel==1){
   if ((rowPos==0 && colPos==0) ||
      (rowPos==0 && colPos==7) ||
      (rowPos==7 && colPos==0) ||
      (rowPos==7 && colPos==7) ||
      (rowPos==0 && colPos==1) ||
      (rowPos==1 && colPos==6) ||
      (rowPos==5 && colPos==1))
```

```
        {
        PORTC |= (1 << PC5);
        _delay_ms(100);         // Espera 100 ms
        PORTC &= ~(1 << PC5);
           nivel = 1;
           mostrarX();
           _delay_ms(50);
           mostrarL1();
           for (int i = 0; i < 8; i++) frame[i] = 0;
           frame[3] = (1 << 7);
        }

}

if (nivel==3){
   if ((rowPos==0 && colPos==0) ||
      (rowPos==0 && colPos==7) ||
      (rowPos==7 && colPos==0) ||
      (rowPos==7 && colPos==7) ||
      (rowPos==0 && colPos==5) ||
      (rowPos==1 && colPos==6) ||
      (rowPos==5 && colPos==1))
      {
         PORTC |= (1 << PC5);
         _delay_ms(100);         // Espera 100 ms
         PORTC &= ~(1 << PC5);
         nivel = 1;
         mostrarX();
         _delay_ms(50);
         mostrarL1();
         for (int i = 0; i < 8; i++) frame[i] = 0;
         frame[3] = (1 << 7);
      }

}

// Esperar a soltar
```

```
        _delay_ms(0);
        while (leerBotones() != -1) _delay_ms(10);
      }
      // refresco continuo
      refrescar(frame);
    }
}
```

PIC16F887 (Reproducción de audio y comunicación):

En este microcontrolador se construyeron los audios a reproducir durante el juego, teniendo sonido de nivel, derrota y victoria dependiendo del caso. En primera instancia, se definió una gama de octavas musicales con sus respectivas frecuencias (incluyendo sostenidos y bemoles) para un mayor rango de reproducción y calidad de cada audio.

Posteriormente, se creó la función Tone para poder llamar a cada nota musical y asignarle un tiempo en ms; es importante que la función tenga estos dos argumentos y cada que llamemos una nota definamos el tiempo, ya que en melodías complejas es poco práctico manejar un mismo tiempo para toda frecuencia auditiva. Con ello, logramos crear las funciones de cada canción requerida tratando de conservar la mayor simetría y fidelidad posible a nivel musical para que se obtenga un sonido armonioso y ordenado.

Finalmente, en el main se inicializan los puertos a donde se conectará la bocina y los dos pines de comunicación, que recibirán la combinación de bits que produce el ATMEGA328P y con ello se escriben condiciones para que en cada posible combinación se ejecute una canción distinta dependiendo del escenario. Cabe mencionar que los delays tuvieron un papel importante en el código, ya que en ocasiones la longitud de la canción superaba el tiempo en pantalla de la ilustración deseada, por lo que se realizaron ajustes continuamente hasta lograr el efecto deseado.

```
#define _XTAL_FREQ 8000000


/*******************************
 * Definici�n de notas musicales *
 *******************************/


// Octava 3
#define DO3  131
#define DOs3 139
#define RE3  147
#define REs3 156
#define MI3  165
#define FA3  175
```

```
#define FAs3 185
#define SOL3 196
#define SOLs3 208
#define LA3  220
#define LAs3 233
#define SI3  247

// Octava 4
#define DO4  262
#define DOs4 277
#define RE4  294
#define REs4 311
#define MI4  330
#define FA4  349
#define FAs4 370
#define SOL4 392
#define SOLs4 415
#define LA4  440
#define LAs4 466
#define SI4  494

// Octava 5
#define DO5  523
#define DOs5 554
#define RE5  587
#define REs5 622
#define MI5  659
#define FA5  698
#define FAs5 740
#define SOL5 784
#define SOLs5 831
#define LA5  880
#define LAs5 932
#define SI5  988

// Octava 6
#define DO6  1047
```

```
#define DOs6 1109
#define RE6  1175
#define REs6 1245
#define MI6  1319
#define FA6  1397
#define FAs6 1480
#define SOL6 1568
#define SOLs6 1661
#define LA6  1760
#define LAs6 1865
#define SI6  1976

// Octava 7
#define DO7  2093
#define DOs7 2217
#define RE7  2349
#define REs7 2489
#define MI7  2637
#define FA7  2794
#define FAs7 2960
#define SOL7 3136
#define SOLs7 3322
#define LA7  3520
#define LAs7 3729
#define SI7  3951

// Octava 8
#define DO8  4186

// Funcion para reproducir una nota
void PlayNote(unsigned int frequency, unsigned int time) {
   Sound_Play(frequency, time);
   delay_ms(5);
}

// Cancion 1: Nivel
void PlayCancion1() {
```

```
  PlayNote(MI5, 150); PlayNote(SOL5, 150); PlayNote(MI6, 150);
  PlayNote(DO6, 150); PlayNote(RE6, 150); PlayNote(SOL6, 150);
  delay_ms(500);
}

// Cancion 2: Derrota
void PlayCancion2() {
  PlayNote(DO5, 200); delay_ms(250); PlayNote(SOL4, 200); delay_ms(250); PlayNote(MI4, 200); delay_ms(150);
  PlayNote(LA4, 300); PlayNote(SI4, 300); PlayNote(LA4, 300);
  PlayNote(SOLs4, 300); PlayNote(LAs4, 300); PlayNote(SOLs4, 300);
  PlayNote(SOL4, 150); PlayNote(MI4, 150); PlayNote(SOL4, 800);
  delay_ms(500);
}

// Cancion 3: Victoria
void PlayCancion3() {
  PlayNote(SOL3, 150); PlayNote(DO4, 150); PlayNote(MI4, 150); PlayNote(SOL4, 150); PlayNote(DO5, 150);  PlayNote(MI5, 150);  PlayNote(SOL5, 450); PlayNote(MI5, 450);
  PlayNote(SOLs3, 150); PlayNote(DO4, 150); PlayNote(REs4, 150); PlayNote(SOLs4, 150); PlayNote(DO5, 150); PlayNote(REs5, 150); PlayNote(SOLs5, 450); PlayNote(REs5, 450);
  PlayNote(LAs3, 150); PlayNote(RE4, 150); PlayNote(FA4, 150); PlayNote(LAs4, 150); PlayNote(RE5, 150); PlayNote(FA5, 150); PlayNote(LAs5, 450); PlayNote(LAs5, 300); PlayNote(LAs5, 150); PlayNote(DO6, 1050);
  delay_ms(500);
}

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

  TRISB7_bit = 1; // Entrada desde ATmega PC5
  TRISB6_bit = 1; // Entrada desde ATmega PC4
  TRISC3_bit = 0;      // Salida para buzzer

  Sound_Init(&PORTC, 3);

  Sound_Play(880, 300);
  delay_ms(300);
  Sound_Play(880, 300);
  delay_ms(1000);
```

```
  while (1) {
    unsigned char bit1 = RB7_bit; // PC5
    unsigned char bit0 =  RB6_bit; // PC4

    if (bit0 && !bit1) {
      PlayCancion1();
    } else if (!bit0 && bit1) {
      PlayCancion2();
    } else if (bit0 && bit1) {
      PlayCancion3();
    }

    delay_ms(300);
  }
}
```

Enlace a un repositorio de GitHub:

https://github.com/EliezerAcebo/Tarea3_SE.git

Conclusiones y recomendaciones.

Conclusiones:

- La integración entre dos microcontroladores permitió una división efectiva de tareas: uno se encargó de la lógica visual y el otro del audio, demostrando una correcta implementación de comunicación entre dispositivos heterogéneos.
- El uso de una matriz LED 8x8 sin controlador externo, a través de multiplexación manual, fue suficiente para representar de forma clara el estado del juego y las ilustraciones de nivel, derrota o victoria.
- La personalización de las melodías para cada evento del juego añadió un valor interactivo significativo, enriqueciendo la experiencia del jugador y haciendo el sistema más dinámico.

Recomendaciones:

- Para mejorar la jugabilidad, se recomienda incluir una rutina de debounce para los botones, ya que podrían registrarse múltiples pulsaciones no deseadas.
- Se sugiere agregar niveles con patrones de minas aleatorios en futuras versiones, para aumentar la jugabilidad y desafío del juego.
- Implementar un sistema de puntuación o cronómetro podría hacer el juego más competitivo y permitir comparar resultados entre jugadores.