Universidad de Guadalajara

Piensa y trabaja

REPORTE DE PRÁCTICA

Práctica extra: Sistema de alarma con ESP32, buzzer y displays

Participantes:

Lorena Villalobos Carrillo

Ximena Lucía Rodríguez Oliva

Brayam De Jesus De La Cerda Valdivia

Seminario de problemas de programación embebidos

(D01 –I9893)

Universidad de Guadalajara

Piensa y trabaja

1. Introducción

En este proyecto se diseñó un sistema de alarma simulada con ESP32, que detecta la activación de sensores representados por botones para indicar una posible intrusión por puerta (PU), ventana 1 (V1) o ventana 2 (V2). El sistema responde a la detección mostrando el evento en dos displays alfanuméricos de 16 segmentos y emitiendo un sonido de alerta con un buzzer.

La activación del sistema se controla por medio de un botón maestro. Cuando la alarma está activada y uno de los botones de "sensores" es presionado, se genera un evento visual y sonoro, simulando una intrusión real. Se incluye además un LED de alerta que parpadea cuando se detecta un evento.

Esta práctica combina conceptos clave como multiplexación de displays, manejo de eventos, control de salida digital y programación estructurada en sistemas embebidos, todos aplicados de forma integrada en un mismo sistema.

2. Desarrollo

2.1 Materiales y herramientas

- 1 microcontrolador ESP32
- 2 displays alfanuméricos de 16 segmentos
- 4 botones (uno para activar alarma, tres para simular eventos)
- 1 buzzer piezoeléctrico
- 1 LED de alerta
- Resistencias de 10kΩ
- Protoboard y cables Dupont
- Computadora con Arduino IDE
- Fuente de alimentación USB o batería externa

2.2 Configuración del sistema

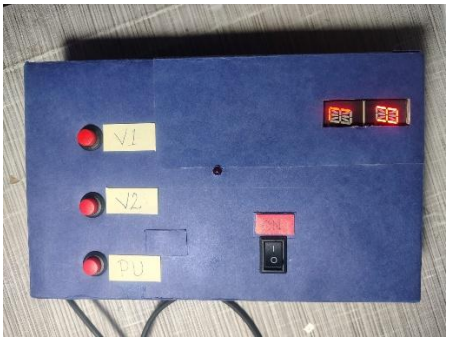
• Displays: Se utilizan dos displays multiplexados de 16 segmentos. Los segmentos A–I son compartidos, y se activan por turnos mediante transistores conectados a los pines 22 y 23.

- Entradas: Cuatro botones con resistencias pull-up internas:

1. pinAL (21): Activa/desactiva la alarma.
2. pinPU (15): Simula evento en la puerta.
3. pinV1 (18): Simula evento en la ventana 1.
4. pinV2 (4): Simula evento en la ventana 2.

- Salidas:

1. LED (19): Parpadea al detectar eventos.
2. Buzzer (16): Emite tono intermitente mientras el evento esté activo.



2.3 Lógica de funcionamiento

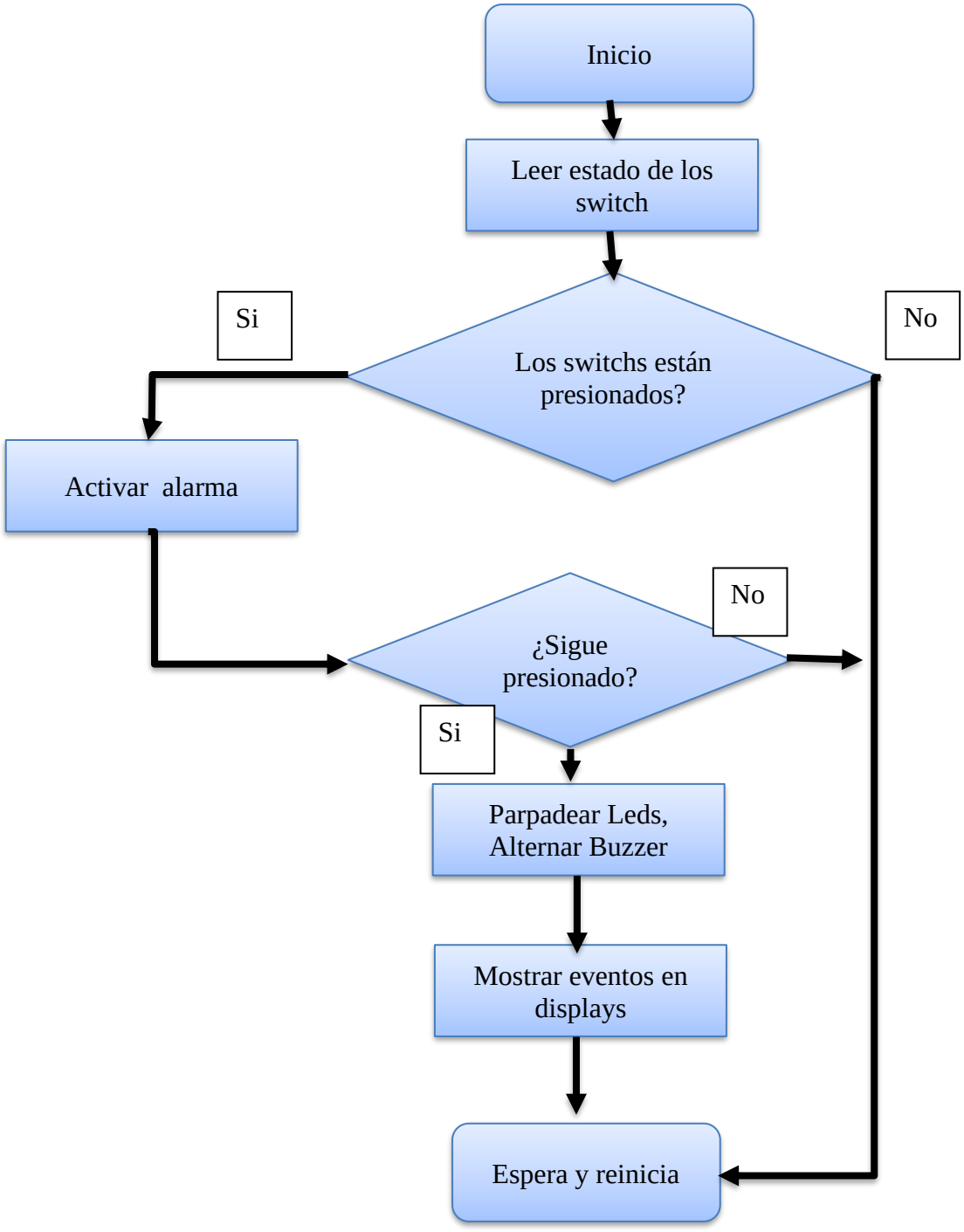
La lógica del sistema se puede describir como una máquina de estados:

1. Estado de reposo: Alarma desactivada. No hay sonidos ni visualización activa.
2. Activación: Al presionar el botón AL, la alarma se activa y entra en modo vigilancia.
3. Detección de evento: Si alguno de los botones PU, V1 o V2 se presiona:

- Se registra un nuevo evento.
- Se muestra el evento en los displays con letras codificadas (por ejemplo, "PU", "V1", "V2").
- Se activa el buzzer intermitente.
- Se activa el parpadeo del LED.

4. Multiplexación: Se alterna cada 5 ms el display para mostrar ambas letras del evento.
5. Cambio de evento: Si hay múltiples eventos simultáneos, se alternan cada 2 segundos.
6. Desactivación: Al presionar nuevamente el botón AL, se apaga la alarma, el buzzer, el LED y se limpian los displays.

2.4 Diagrama de flujo



3. Resultados y observaciones

Logros

- Se logró una **multiplexación funcional** entre dos displays alfanuméricos con un solo conjunto de pines.
- El sistema **reacciona inmediatamente** ante un botón presionado y cambia el mensaje mostrado.
- El **buzzer intermitente** y el **LED parpadeante** funcionan de manera sincronizada con los eventos.
- El sistema puede registrar **hasta tres eventos simultáneos** y alternar su visualización automáticamente.

Observaciones

- Si los eventos ocurren muy rápido, pueden superponerse en la lógica. Podría añadirse una cola o buffer de eventos.
- El sistema podría beneficiarse de **debounce por software** para evitar lecturas falsas por rebotes en los botones.
- La visualización con letras requiere codificación precisa para evitar mostrar segmentos erróneos.

4. Conclusión

Este proyecto permitió integrar múltiples elementos del desarrollo embebido: entradas digitales, salidas visuales y auditivas, multiplexación, estructuras condicionales y control de eventos.

El resultado es un sistema de alarma funcional que, aunque simulado con botones, puede extenderse fácilmente a sensores reales (magnéticos, PIR, etc.) para detectar presencia o aperturas reales.

Gracias al uso de ESP32, se logró mantener el control con alta velocidad y flexibilidad, y el proyecto sienta las bases para versiones más avanzadas con conexión Wi-Fi o app móvil.

5. Código

```
COIDGO ALARMA TERMINADO                              // Pines del display
(segmentos compartidos)
const int displayPins[9] = {12, 33, 32, 25, 26, 14, 27, 2, 13};  //
Segmentos A-I

// Pines de transistores para displays
const int displayEnablePins[2] = {23, 22};

// Pines de entrada
const int pinPU = 15;
const int pinV1 = 18;
const int pinV2 = 4;
const int pinAL = 21;

// LED de alerta
const int pinLED = 19;

// Buzzer
const int pinBuzzer = 16;  // GPIO16 (RX2)

// Letras codificadas
const byte letras[][9] = {
  {0, 0, 1, 1, 1, 1, 1, 0, 0},  // P
  {1, 1, 1, 0, 1, 1, 0, 0, 0},  // U
  {0, 0, 0, 0, 1, 1, 0, 1, 1},  // V
  {0, 1, 1, 1, 1, 1, 1, 0, 0},  // A
  {1, 0, 0, 0, 1, 1, 0, 0, 0},  // L
  {0, 1, 1, 0, 0, 0, 0, 1, 0},  // 1
  {1, 0, 1, 1, 0, 1, 1, 0, 0}   // 2
};

enum {IDX_P, IDX_U, IDX_V, IDX_A, IDX_L, IDX_1, IDX_2};

// Eventos
int eventos[5][2];
int numEventos = 0;
int eventoActual = 0;

bool alarmaActiva = false;
bool mostrarAL = false;
unsigned long tiempoInicioAL = 0;

bool estadoAnteriorSwitchAL = HIGH;

unsigned long lastBlink = 0;
bool ledOn = false;

unsigned long lastDisplaySwap = 0;
```

```
int currentDisplay = 0;

unsigned long lastEventoCambio = 0;

// Variables para el buzzer
unsigned long lastBuzzerToggle = 0;
bool buzzerOn = false;
const int buzzerInterval = 500; // 500ms para tono intermitente

void setup() {
  for (int i = 0; i < 9; i++) pinMode(displayPins[i], OUTPUT);
  for (int i = 0; i < 2; i++) {
    pinMode(displayEnablePins[i], OUTPUT);
    digitalWrite(displayEnablePins[i], LOW);
  }

  pinMode(pinPU, INPUT_PULLUP);
  pinMode(pinV1, INPUT_PULLUP);
  pinMode(pinV2, INPUT_PULLUP);
  pinMode(pinAL, INPUT_PULLUP);
  pinMode(pinLED, OUTPUT);

  // Configurar pin del buzzer
  pinMode(pinBuzzer, OUTPUT);
  digitalWrite(pinBuzzer, LOW); // Asegurar que el buzzer está apagado al
inicio
}

void loop() {
  bool estadoActualSwitchAL = digitalRead(pinAL);

  // Detectar flanco de bajada del switch AL
  if (estadoAnteriorSwitchAL != estadoActualSwitchAL) {
    estadoAnteriorSwitchAL = estadoActualSwitchAL;

    if (estadoActualSwitchAL == LOW) {
      alarmaActiva = true;
      mostrarAL = true;
      tiempoInicioAL = millis();
    } else {
      alarmaActiva = false;
      mostrarAL = false;
      apagarDisplays();
      digitalWrite(pinLED, LOW);
      digitalWrite(pinBuzzer, LOW); // Apagar buzzer al desactivar alarma
      return;
    }
  }

  if (!alarmaActiva) {
    apagarDisplays();
```

```
  digitalWrite(pinLED, LOW);
  digitalWrite(pinBuzzer, LOW); // Asegurar que el buzzer está apagado
  return;
}

// Leer botones
bool estadoPU = digitalRead(pinPU) == LOW;
bool estadoV1 = digitalRead(pinV1) == LOW;
bool estadoV2 = digitalRead(pinV2) == LOW;

// Registrar eventos
numEventos = 0;

if (estadoPU && numEventos < 5) {
  eventos[numEventos][0] = IDX_P;
  eventos[numEventos][1] = IDX_U;
  numEventos++;
}
if (estadoV1 && numEventos < 5) {
  eventos[numEventos][0] = IDX_V;
  eventos[numEventos][1] = IDX_1;
  numEventos++;
}
if (estadoV2 && numEventos < 5) {
  eventos[numEventos][0] = IDX_V;
  eventos[numEventos][1] = IDX_2;
  numEventos++;
}

// Control del LED y buzzer
if (numEventos > 0) {
  // LED parpadeante
  if (millis() - lastBlink >= 200) {
    ledOn = !ledOn;
    digitalWrite(pinLED, ledOn);
    lastBlink = millis();
  }

  // Control del buzzer (tono intermitente)
  if (millis() - lastBuzzerToggle >= buzzerInterval) {
    buzzerOn = !buzzerOn;
    digitalWrite(pinBuzzer, buzzerOn ? HIGH : LOW);
    lastBuzzerToggle = millis();
  }

  mostrarAL = false; // Desactivar mostrarAL cuando hay eventos
} else {
  digitalWrite(pinLED, LOW);
  digitalWrite(pinBuzzer, LOW); // Apagar buzzer cuando no hay eventos
  mostrarAL = true; // Mostrar AL cuando no hay eventos
}
```

```
  // Mostrar "AL" o los eventos
  if (mostrarAL) {
    if (millis() - lastDisplaySwap >= 5) {
      currentDisplay = !currentDisplay;
      mostrarLetraEnDisplay(currentDisplay, currentDisplay ? IDX_L : IDX_A);
      lastDisplaySwap = millis();
    }
  }
  else if (numEventos > 0) {
    // Alternar evento cada 2 segundos
    if (millis() - lastEventoCambio >= 2000) {
      eventoActual = (eventoActual + 1) % numEventos;
      lastEventoCambio = millis();
    }

    // Multiplexar displays
    if (millis() - lastDisplaySwap >= 5) {
      currentDisplay = !currentDisplay;
      mostrarLetraEnDisplay(currentDisplay,
eventos[eventoActual][currentDisplay]);
      lastDisplaySwap = millis();
    }
  }
}

void mostrarLetraEnDisplay(int displayNum, int letraIdx) {
  if (letraIdx < 0) {
    apagarDisplays();
    return;
  }

  digitalWrite(displayEnablePins[0], LOW);
  digitalWrite(displayEnablePins[1], LOW);

  for (int i = 0; i < 9; i++) {
    digitalWrite(displayPins[i], letras[letraIdx][i]);
  }

  digitalWrite(displayEnablePins[displayNum], HIGH);
}

void apagarDisplays() {
  for (int i = 0; i < 9; i++) digitalWrite(displayPins[i], LOW);
  digitalWrite(displayEnablePins[0], LOW);
  digitalWrite(displayEnablePins[1], LOW);
}
```