Informe del Prototipo V2.0: Alarma IoT con Detección Dual

Fecha de Elaboración: 16 de octubre de 2025

1. Síntesis del Prototipo

La Versión 2.0 del prototipo de Alarma IoT representa una evolución significativa respecto a la versión inicial, incorporando un **sistema de detección dual** para incrementar la fiabilidad y reducir la incidencia de falsos positivos. Adicionalmente al sensor de movimiento Infrarrojo Pasivo (PIR), se ha integrado una **Unidad de Medición Inercial (IMU) MPU-6050** para la detección de vibraciones.

El sistema ahora activa una alarma audiovisual centralizada (buzzer) si se cumple cualquiera de las dos condiciones de disparo: un movimiento confirmado o una vibración súbita. Se han implementado indicadores LED independientes para cada sensor, permitiendo una identificación visual inmediata de la causa de la alerta. La arquitectura del software mantiene su modelo de ejecución no bloqueante y la conectividad Wi-Fi para la sincronización horaria vía NTP, registrando todos los eventos con marcas de tiempo precisas.

2. Componentes de Hardware Empleados

- **Unidad de Microcontrolador:** ESP32 DevKit
- **Sensores:**
  - Sensor Primario: Sensor de Movimiento Infrarrojo Pasivo (PIR) HC-SR501.
  - Sensor Secundario: Módulo de Acelerómetro/Giroscopio (IMU) MPU-6050.
- **Actuadores de Alerta:**
  - LED Rojo: Indicador visual para alarma por movimiento.
  - LED Amarillo/Naranja: Indicador visual para alarma por vibración.
  - Buzzer Pasivo con terminal de señal (S).
- **Componentes Pasivos:**
  - 2 Resistencias de 220Ω (una para cada LED).

3. Esquema de Conexiones (Pinout)

Para la integración del MPU-6050, fue necesario liberar los pines I²C por defecto del ESP32, resultando en la siguiente configuración de hardware:

- **Sensor de Vibración (MPU-6050):**
  - VCC → **3V3** del ESP32 (Alimentación 3.3V)
  - GND → **GND** del ESP32 (Tierra)
  - SCL → **GPIO 22** (Línea de Reloj I²C)
  - SDA → **GPIO 21** (Línea de Datos I²C)

- **Sensor de Movimiento (PIR):**
  - VCC → **VIN** del ESP32 (Alimentación 5V)
  - GND → **GND** del ESP32 (Tierra)
  - OUT → **GPIO 23** (Señal de entrada)
- **LED Rojo (Movimiento):**
  - Pata Larga (+) → Resistencia → **GPIO 19** (Pin de salida seguro)
  - Pata Corta (-) → **GND** del ESP32
- **LED Amarillo/Naranja (Vibración):**
  - Pata Larga (+) → Resistencia → **GPIO 18** (Pin de salida seguro)
  - Pata Corta (-) → **GND** del ESP32
- **Buzzer Pasivo:**
  - + → **VIN** del ESP32 (Alimentación 5V)
  - - → **GND** del ESP32
  - S → **GPIO 13** (Señal de control de tono)

4. Calibración y Parámetros de Sensores

- **Sensor PIR:**
  - **Modo de Disparo:** Configurado en **'H' (Repetible)** para detección continua.
  - **Sensibilidad (Sx):** Ajustada a un nivel medio para equilibrar rango y fiabilidad.
  - **Tiempo de Retardo (Tx):** Ajustado al mínimo; la temporización se gestiona por software.
- **Sensor MPU-6050:**
  - **Umbral de Vibración (VIBRATION_THRESHOLD):** Calibrado por software a un valor de **12**. Este umbral define la magnitud de aceleración mínima necesaria para ser considerada una vibración relevante, permitiendo ajustar la sensibilidad a golpes o movimientos bruscos.

5. Lógica del Software y Parámetros Clave

La arquitectura del software ha sido modificada para soportar la lógica de detección dual.

- **Alarma Unificada con Indicadores Independientes:** La alarma principal (buzzer) se activa si se detecta movimiento confirmado **O** una vibración. Los LEDs rojo y amarillo/naranja se encienden de forma independiente para "memorizar" y mostrar la causa del disparo durante todo el ciclo de la alarma.
- **Duración Mínima de Alarma (alarmDuration):** Se ha implementado un temporizador que mantiene la alarma activa por un mínimo de **3 segundos**, incluso si el estímulo que la causó es breve. Esto asegura que la alerta sea siempre notoria.
- **Registro de Eventos con Timestamp:** Gracias a la sincronización NTP, todos los eventos relevantes (detección inicial, confirmación de alarma, causa y finalización) se registran en el Monitor Serie con una marca de tiempo precisa (YYYY-MM-DD HH:MM:SS), sentando las bases para el futuro almacenamiento en una base de datos.

6. Código Final de la Versión 2.0

```
#include <WiFi.h>
#include "time.h"
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

// --- Instancia del sensor MPU6050 ---
Adafruit_MPU6050 mpu;

// --- CONFIGURACIÓN DE RED ---
const char* ssid = "NOMBRE_DE_TU_WIFI";
const char* password = "CONTRASENA_DE_TU_WIFI";

// --- CONFIGURACIÓN DE HORA (NTP) ---
const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = -10800; // UTC-3 para Chile
const int daylightOffset_sec = 0;

// --- Pines de los componentes ---
const int ledRojoPin = 19;      // LED para Movimiento (PIR)
const int ledVibracionPin = 18; // LED para Vibración (MPU-6050)
const int sensorPirPin = 23;
const int buzzerPin = 13;

// --- Parámetros de Sensores y Alarma ---
#define VIBRATION_THRESHOLD 12
const long alarmDuration = 3000; // Duración mínima de la alarma en ms (3 segundos)

// --- Variables para el control de tiempo ---
unsigned long previousCheckTime = 0;
const long checkInterval = 100;

// --- Variables para la lógica de confirmación (PIR) ---
bool motionDetected = false;
unsigned long firstMotionTime = 0;
const long confirmationDelay = 750;

// --- Estados de los componentes ---
bool isAlarmActive = false;
unsigned long alarmStartTime = 0;
bool alarmTriggeredByMotion = false;
```

```
bool alarmTriggeredByVibration = false;

// --- Función para obtener la fecha y hora formateada ---
String getFormattedTime() {
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    return "Hora no sincronizada";
  }
  char timeStringBuff[50];
  strftime(timeStringBuff, sizeof(timeStringBuff), "%Y-%m-%d %H:%M:%S", &timeinfo);
  return String(timeStringBuff);
}

void setup() {
  Serial.begin(115200);

  pinMode(ledRojoPin, OUTPUT);
  pinMode(ledVibracionPin, OUTPUT);
  pinMode(sensorPirPin, INPUT);
  pinMode(buzzerPin, OUTPUT);

  // --- Inicialización del MPU6050 ---
  if (!mpu.begin()) {
    Serial.println("Fallo al encontrar el sensor MPU-6050. Revisa las conexiones.");
    while (1) { delay(10); }
  }
  Serial.println("Sensor MPU-6050 inicializado.");
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

  // --- Conexión a Wi-Fi y Sincronización de Hora ---
  Serial.printf("Conectando a %s ", ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println(" CONECTADO");
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
  Serial.println("Hora sincronizada.");

  Serial.println("Sistema de alarma V2.3 iniciado. Esperando eventos...");
}
```

```
void loop() {
  unsigned long currentTime = millis();

  if (currentTime - previousCheckTime >= checkInterval) {
    previousCheckTime = currentTime;

    // --- LECTURA DE SENSORES ---
    int sensorStatePIR = digitalRead(sensorPirPin);

    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    float totalAccel = sqrt(pow(a.acceleration.x, 2) + pow(a.acceleration.y, 2) +
pow(a.acceleration.z, 2));

    // --- DETERMINAR LAS CONDICIONES DE DISPARO ---
    if (sensorStatePIR == HIGH) {
      if (!motionDetected) {
        motionDetected = true;
        firstMotionTime = currentTime;
      }
    } else {
      motionDetected = false;
    }

    bool triggerByMotionNow = motionDetected && (currentTime - firstMotionTime >=
confirmationDelay);
    bool triggerByVibrationNow = (totalAccel > VIBRATION_THRESHOLD);

    // --- LÓGICA DE LA ALARMA ---

    // 1. CONDICIÓN PARA ACTIVAR LA ALARMA
    if (triggerByMotionNow || triggerByVibrationNow) {
      if (!isAlarmActive) { // Si la alarma está apagada, la enciende
        isAlarmActive = true;
        alarmStartTime = currentTime;
        tone(buzzerPin, 1000);

        Serial.print("[" + getFormattedTime() + "] ");
        Serial.print("¡ALARMA ACTIVADA! Motivo: ");
      }

      // "Memoriza" la causa del disparo.
```

```
      if (triggerByMotionNow) {
        if (!alarmTriggeredByMotion) { Serial.print("Movimiento. "); }
        alarmTriggeredByMotion = true;
      }
      if (triggerByVibrationNow) {
        if (!alarmTriggeredByVibration) { Serial.print("Vibración."); }
        alarmTriggeredByVibration = true;
      }
    }

    // 2. LÓGICA MIENTRAS LA ALARMA ESTÁ ACTIVA
    if (isAlarmActive) {
      // Enciende los LEDs correspondientes a la causa "memorizada"
      digitalWrite(ledRojoPin, alarmTriggeredByMotion);
      digitalWrite(ledVibracionPin, alarmTriggeredByVibration);

      // CONDICIÓN PARA DESACTIVAR LA ALARMA
      if (!triggerByMotionNow && !triggerByVibrationNow && (currentTime - alarmStartTime >=
alarmDuration)) {
        isAlarmActive = false;
        noTone(buzzerPin);
        digitalWrite(ledRojoPin, LOW);
        digitalWrite(ledVibracionPin, LOW);

        // Resetea la memoria de las causas
        alarmTriggeredByMotion = false;
        alarmTriggeredByVibration = false;

        Serial.println("\n[" + getFormattedTime() + "] Condiciones de alarma finalizadas. Sistema
en reposo.");
      }
    }
  }
}
```