Proyecto: Sistema de Monitoreo de Inclinación y Temperatura con ESP32 y Comunicación Bluetooth BLE

Objetivo

El objetivo de esta práctica es desarrollar un sistema de monitoreo que permita medir la inclinación y la temperatura de un dispositivo mediante un ESP32, usando sensores MPU6050 y BMP180, y enviar los datos recolectados a una aplicación en un dispositivo Android a través de Bluetooth de Baja Energía (BLE). Adicionalmente, el sistema implementa un modo de bajo consumo de energía (modo "deep sleep") que puede ser activado mediante un botón.

Descripción General del Proyecto

El sistema está compuesto por un ESP32 conectado a dos sensores: el MPU6050, que mide la inclinación a través de su giroscopio, y el BMP180, que mide la temperatura. El ESP32 recolecta los datos de estos sensores cada minuto y los almacena en un arreglo. Cuando el dispositivo Android se conecta al ESP32 vía BLE, recibe el conjunto de datos almacenados, que luego son graficados en tiempo real en la aplicación.

El ESP32 también cuenta con un botón que permite ponerlo en modo "deep sleep" para reducir el consumo energético cuando no se necesita el monitoreo, ideal para extender la duración de la batería. Para reactivar el ESP32, basta con volver a presionar el botón.

Componentes

1. **ESP32:** Microcontrolador con capacidad de comunicación BLE y modo de bajo consumo energético.
2. **MPU6050:** Sensor de movimiento que proporciona información de la inclinación en tres ejes (X, Y y Z).
3. **BMP180:** Sensor de temperatura que permite medir la temperatura ambiente.
4. **Aplicación Android:** App que se conecta al ESP32 vía BLE, recibe los datos de inclinación y temperatura, y los grafica en tiempo real.
5. **Botón de encendido/apagado:** Configurado para activar el modo "deep sleep" del ESP32.

Funcionalidad

1. **Medición de Inclinación:** El ESP32 lee los valores de inclinación en los ejes X, Y y Z a partir del sensor MPU6050, calculando los ángulos en tiempo real.
2. **Medición de Temperatura:** El BMP180 proporciona la lectura de la temperatura, que el ESP32 registra y almacena.
3. **Almacenamiento de Datos:** Los valores de inclinación y temperatura se almacenan en un arreglo en el ESP32, con una frecuencia de muestreo de un minuto.

4. **Envío de Datos a la App:** Cuando el dispositivo Android se conecta, el ESP32 envía el conjunto de datos almacenados en el arreglo mediante BLE, permitiendo que la app grafique las lecturas.
5. **Modo de Bajo Consumo:** Al presionar el botón, el ESP32 entra en modo "deep sleep". Esto apaga la mayoría de los circuitos del ESP32 para reducir el consumo energético. Al presionar el botón nuevamente, el ESP32 se reactiva y reanuda el monitoreo.

Objetivos Específicos de la Práctica

1. Implementar y configurar el protocolo BLE en el ESP32 para la transmisión de datos de sensores.
2. Diseñar e integrar una aplicación Android capaz de conectarse al ESP32 y recibir datos en tiempo real.
3. Programar el ESP32 para leer, calcular y almacenar valores de inclinación y temperatura.
4. Implementar el modo de bajo consumo mediante el "deep sleep" del ESP32 y configurar un botón para activarlo.
5. Visualizar y graficar los datos de inclinación y temperatura en la app de Android.

Procedimiento

1. **Configuración de Hardware:**
    - Conectar el MPU6050 y el BMP180 al ESP32 a través del bus I2C.
    - Configurar el pin del botón en el ESP32 como entrada con "pull-up" interno.
2. **Configuración de Software:**
    - Programar el ESP32 para recolectar datos de inclinación y temperatura, almacenarlos en un arreglo y enviarlos mediante BLE.
    - Programar la aplicación Android para conectarse al ESP32 vía BLE, recibir los datos y graficarlos en tiempo real.
    - Implementar la función de modo "deep sleep" para reducir el consumo energético, con el botón de activación.
3. **Pruebas:**
    - Verificar la correcta adquisición y almacenamiento de datos de los sensores.
    - Comprobar la transmisión de datos al dispositivo Android y la visualización en la app.
    - Testear el botón de activación para que el ESP32 entre y salga del modo "deep sleep".

Resultados Esperados

- El ESP32 debe ser capaz de leer la inclinación y la temperatura cada minuto, almacenarlos y enviarlos al dispositivo Android cuando se conecte.
- La aplicación Android debe graficar los datos de inclinación y temperatura en tiempo real, mostrando un historial de las últimas lecturas.

- Al presionar el botón, el ESP32 debe entrar en modo "deep sleep" para reducir el consumo, y debe reactivarse al presionar el botón nuevamente.

Conclusión

Esta práctica permite familiarizarse con el uso de sensores de movimiento y temperatura, el protocolo BLE para transmisión de datos y el modo de bajo consumo de un microcontrolador. Además, se desarrolla una app Android para graficar los datos en tiempo real, completando así un sistema de monitoreo de inclinación y temperatura eficiente y con funcionalidad de ahorro energético.

Aquí tienes los códigos completos para el proyecto, incluyendo tanto el código del ESP32 como el código básico de la app de Android.

1. Código para el ESP32

Este código utiliza el sensor MPU6050 para medir la inclinación y el sensor BMP180 para medir la temperatura. Los datos se almacenan en un arreglo y se envían al dispositivo Android a través de Bluetooth Low Energy (BLE). Además, el ESP32 puede entrar en modo de bajo consumo (deep sleep) al presionar un botón.

Código ESP32:
cpp
Copiar código

```
#include <Wire.h>
#include <MPU6050.h>
#include <Adafruit_BMP085.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <esp_sleep.h>

MPU6050 mpu;
Adafruit_BMP085 bmp;

#define SERVICE_UUID        "12345678-1234-1234-1234-1234567890ab"
#define CHARACTERISTIC_UUID "87654321-4321-4321-4321-1234567890ab"
#define BUTTON_PIN 15  // Pin del botón

BLECharacteristic *pCharacteristic;
bool deviceConnected = false;
```

```
unsigned long lastTime = 0;
const int dataInterval = 60000; // Intervalo de almacenamiento (1
minuto)
int currentIndex = 0;
const int dataBufferSize = 10; // Tamaño del buffer para guardar 10
muestras

struct SensorData {
  float gyroX, gyroY, gyroZ;    // Inclinación en cada eje
  float temp;                   // Temperatura del BMP180
};

SensorData dataBuffer[dataBufferSize];

class MyCallbacks : public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
  };

  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
  }
};

void enterDeepSleep() {
  Serial.println("Entrando en modo de sueño profundo...");
  esp_sleep_enable_ext0_wakeup(GPIO_NUM_15, 0);  // Configura para
despertar cuando el botón se presione
  esp_deep_sleep_start();  // Entra en modo de sueño profundo
}

void setup() {
  Serial.begin(115200);
  Wire.begin();
  mpu.begin();
  mpu.calcGyroOffsets(true);
```

```
  if (!bmp.begin()) {
    Serial.println("Error al iniciar BMP180");
    while (1);
  }

  pinMode(BUTTON_PIN, INPUT_PULLUP);  // Configura el pin del botón
con pull-up interno

  // Configuración BLE
  BLEDevice::init("ESP32_Sensor");
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyCallbacks());

  BLEService *pService = pServer->createService(SERVICE_UUID);
  pCharacteristic = pService->createCharacteristic(
                      CHARACTERISTIC_UUID,
                      BLECharacteristic::PROPERTY_NOTIFY
                    );
  pCharacteristic->addDescriptor(new BLE2902());
  pService->start();

  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(false);
  pAdvertising->setMinPreferred(0x06);
  BLEDevice::startAdvertising();

  Serial.println("Esperando conexión BLE...");
}

void loop() {
  // Revisa si el botón ha sido presionado para entrar en modo de
sueño
  if (digitalRead(BUTTON_PIN) == LOW) {
    delay(50);  // Debounce
    if (digitalRead(BUTTON_PIN) == LOW) {
      enterDeepSleep();
    }
```

```
  }

  unsigned long currentTime = millis();

  if (currentTime - lastTime > dataInterval) {
    // Lee y calcula inclinación del giroscopio
    mpu.update();
    float gyroX = atan2(mpu.getAccY(), mpu.getAccZ()) * 180 / PI;
    float gyroY = atan2(-mpu.getAccX(), sqrt(mpu.getAccY() *
mpu.getAccY() + mpu.getAccZ() * mpu.getAccZ())) * 180 / PI;
    float gyroZ = mpu.getGyroZ();

    // Lee la temperatura del BMP180
    float temp = bmp.readTemperature();

    // Almacena datos en el buffer
    dataBuffer[currentIndex] = { gyroX, gyroY, gyroZ, temp };
    currentIndex = (currentIndex + 1) % dataBufferSize;

    lastTime = currentTime;
  }

  // Si está conectado, envía el buffer completo
  if (deviceConnected) {
    String jsonData = "[";
    for (int i = 0; i < dataBufferSize; i++) {
      jsonData += "{\"gyroX\":" + String(dataBuffer[i].gyroX) +
",\"gyroY\":" + String(dataBuffer[i].gyroY) +
                  ",\"gyroZ\":" + String(dataBuffer[i].gyroZ) +
",\"temp\":" + String(dataBuffer[i].temp) + "}";
      if (i < dataBufferSize - 1) jsonData += ",";
    }
    jsonData += "]";
    pCharacteristic->setValue(jsonData.c_str());
    pCharacteristic->notify();

    delay(2000); // Pausa para evitar saturar la conexión BLE
  }
```

```
}
```

2. Código de la Aplicación Android (Java)

La app Android establece la conexión con el ESP32 a través de BLE, recibe los datos de inclinación y temperatura y los grafica en tiempo real. A continuación, se muestra un código básico para la actividad principal (`MainActivity.java`), suponiendo que se usa una biblioteca de gráficos como **MPAndroidChart** para graficar los datos.

`MainActivity.java`

java
Copiar código

```
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattService;
import android.bluetooth.BluetoothManager;
import android.os.Bundle;
import android.util.Log;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
import com.github.mikephil.charting.data.LineDataSet;
import org.json.JSONArray;
import org.json.JSONObject;

public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MainActivity";
    private static final String DEVICE_NAME = "ESP32_Sensor";
    private static final String SERVICE_UUID =
"12345678-1234-1234-1234-1234567890ab";
    private static final String CHARACTERISTIC_UUID =
"87654321-4321-4321-4321-1234567890ab";

    private BluetoothAdapter bluetoothAdapter;
```

```
    private BluetoothGatt bluetoothGatt;
    private LineChart chart;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        chart = findViewById(R.id.chart);
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

        BluetoothDevice device = findDeviceByName(DEVICE_NAME);
        if (device != null) {
            connectToDevice(device);
        } else {
            Toast.makeText(this, "Dispositivo no encontrado",
Toast.LENGTH_SHORT).show();
        }
    }

    private BluetoothDevice findDeviceByName(String name) {
        for (BluetoothDevice device :
bluetoothAdapter.getBondedDevices()) {
            if (device.getName().equals(name)) {
                return device;
            }
        }
        return null;
    }

    private void connectToDevice(BluetoothDevice device) {
        bluetoothGatt = device.connectGatt(this, false, new
BluetoothGattCallback() {
            @Override
            public void onConnectionStateChange(BluetoothGatt gatt,
int status, int newState) {
                if (newState == BluetoothGatt.STATE_CONNECTED) {
                    gatt.discoverServices();
```

```
                }
            }

            @Override
            public void onServicesDiscovered(BluetoothGatt gatt, int
status) {
                BluetoothGattService service =
gatt.getService(java.util.UUID.fromString(SERVICE_UUID));
                if (service != null) {
                    BluetoothGattCharacteristic characteristic =
service.getCharacteristic(java.util.UUID.fromString(CHARACTERISTIC_UUI
D));
                    gatt.setCharacteristicNotification(characteristic,
true);
                }
            }

            @Override
            public void onCharacteristicChanged(BluetoothGatt gatt,
BluetoothGattCharacteristic characteristic) {
                String json = characteristic.getStringValue(0);
                updateChart(json);
            }
        });
    }

    private void updateChart(String jsonData) {
        try {
            JSONArray jsonArray = new JSONArray(jsonData);
            LineData lineData = new LineData();
            LineDataSet dataSet = new LineDataSet(null, "Datos de
Sensor");

            for (int i = 0; i < jsonArray.length(); i++) {
                JSONObject obj = jsonArray.getJSONObject(i);
                float temp = (float) obj.getDouble("temp");
                dataSet.addEntry(new Entry(i, temp));
            }
```

```
            lineData.addDataSet(dataSet);
            runOnUiThread(() -> chart.setData(lineData));
        } catch (Exception e) {
            Log.e(TAG, "Error parsing JSON", e);
        }
    }
}
```

Nota:

Este código es solo un ejemplo básico y podría necesitar ajustes, especialmente en la configuración de la interfaz de usuario (como `activity_main.xml` con el `LineChart`). Para un gráfico más sofisticado, puedes agregar configuraciones de estilo a `LineDataSet`.