Documentación – Control de LEDs por BLE con ESP32 y MIT App Inventor

Fecha: 19/02/2026

Integrantes:

• Eduardo Cadengo López

• Itzel Citlalli Martell De La Cruz

• Damian Alexander Diaz Piña

Investigación BLE vs Bluetooth

Diferencias principales

- Bluetooth Clásico se usa para enviar audio o datos continuos (audífonos, bocinas, manos libres).
- BLE (Bluetooth Low Energy) se usa para sensores, IoT y dispositivos que envían pocos datos ocasionalmente (smartbands, beacons, ESP32).
- Bluetooth Clásico transmite más datos y más rápido.
  BLE consume muchísimo menos energía.

Ventajas de BLE

- Muy bajo consumo de energía (ideal para baterías pequeñas).
- Conexión rápida y ligera.
- Perfecto para IoT, sensores y dispositivos simples como el ESP32.
- Compatible con Android y iOS.

Desventajas de BLE

- No sirve para audio, no tiene suficiente ancho de banda.
- Menor velocidad para transferir archivos o datos pesados.Compatible con Android y iOS.
- Está pensado solo para mensajes pequeños.

Placa utilizada y Componentes utilizados:

- ESP32 (compatible con cualquier ESP32 estándar con Bluetooth BLE)
- ESP32 Placa de expansion
- 3 LEDs (Verde, Azul, Rojo, o cualquier color)
- Protoboard
- Jumpers
- Teléfono con Android (para MIT App Inventor; iOS también soportado)

Descripción del proyecto:

Este proyecto implementa un control inalámbrico vía Bluetooth (BLE) para manejar 3 LEDs conectados a un ESP32 mediante una App móvil creada en MIT App Inventor.

La aplicación móvil es capaz de:

- Escanear dispositivos BLE cercanos.
- Mostrar la lista de dispositivos encontrados.
- Conectarse al ESP32.
- Enviar caracteres vía BLE para encender/apagar LEDs.

La ESP32 funciona como servidor BLE (GATT Server), recibe comandos en una característica BLE y ejecuta acciones sobre los LEDs.

Objetivo del código:

- Crear un servidor BLE en la ESP32 para recibir comandos desde la App móvil.
- Enviar mensajes desde MIT App Inventor a la ESP32 mediante una característica BLE.
- Controlar LEDs externos con dichos comandos (‘A’, ‘a’, ‘B’, ‘b’, ‘C’, ‘c’).
- Permitir conexión desde Android y iOS utilizando la extensión BluetoothLE.

Conexiones (LEDs de estado/control):

LED	GPIO ESP32	Descripción
Verde	16	Listo para conectar / Control LED 1
Azul	17	Bluetooth conectado / Control LED 2
Rojo	18	Bluetooth desconectado / Control LED 3

Lógica del programa

1. Inicialización

- Se configura BLE con:
- Service UUID: 12345678-1234-1234-1234-1234567890ab
- Characteristic UUID: abcd1234-5678-1234-5678-abcdef123456

2. Conexión desde App

- La app escanea y muestra el dispositivo ESP32_BLE_LEDS.
- Al conectarse, se habilitan botones de control en la app y el monitor serial indica "Conectado".

3. Recepción de comandos

- La app envía un carácter a la característica BLE:

Comandos (carácter → acción)

Comando	Acción	GPIO
A	Enciende LED Verde	16
a	Apaga LED Verde	16
B	Enciende LED Azul	17
b	Apaga LED Azul	17
C	Enciende LED Rojo	18
c	Apaga LED Rojo	18

4. Desconexión

- El monitor serial muestra "Desconectado".
- El ESP32 vuelve a anunciarse para permitir que otro teléfono se conecte.

Funciones clave

- MyServerCallbacks: Maneja conexión/desconexión.
- MyCallbacks: Recibe caracteres de la app y ejecuta acciones.
- setEstado(): Controla LEDs de estado (verde/azul/rojo).
- setLedByCommand(): Ejecuta comando recibido ("A", "a", etc.).

Cómo usar la App en MIT App Inventor

- Instalar extensión BluetoothLE.
- Agregar componentes: BluetoothLE1, ListView para dispositivos, botones A/a, B/b, C/c, X.
- Escanear, después seleccionar ESP32 y luego a Conectar.
- Cada botón envía un carácter a la ESP32.

Cómo ejecutar

- Conectar ESP32 a la PC.
- Cargar el programa proporcionado.
- Abrir monitor serial (115200).
- En MIT App Inventor: abrir App luego, Escanear y después Conectar.
- Usar los botones para encender/apagar LEDs en tiempo real.

Características especiales que se usaron:

- Comunicación BLE compatible con Android y iOS.
- App cambia automáticamente de interfaz al conectarse.
- Comandos simples (1 carácter) con baja latencia.
- Estados visuales con LEDs físicos.

Código:

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>


// ID para identificar el dispsitivo y mandar las acciones desde mit app
inventor //
#define SERVICE_UUID        "12345678-1234-1234-1234-1234567890ab"
#define CHARACTERISTIC_UUID "abcd1234-5678-1234-5678-abcdef123456"

// numero de pines de los leds //
#define LED1_PIN 16
#define LED2_PIN 17
#define LED3_PIN 18

bool dispositivoConectado = false;

void setAll(bool on) {
```

```
  digitalWrite(LED1_PIN, on ? HIGH : LOW);
  digitalWrite(LED2_PIN, on ? HIGH : LOW);
  digitalWrite(LED3_PIN, on ? HIGH : LOW);
}

class MyServerCallbacks : public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) override {
    dispositivoConectado = true;
    Serial.println("Conectado");
  }
  void onDisconnect(BLEServer* pServer) override {
    dispositivoConectado = false;
    Serial.println("Desconectado");
    pServer->getAdvertising()->start(); // desconcetarse de el esp32
  }
};

class MyCallbacks : public BLECharacteristicCallbacks {

void onWrite(BLECharacteristic* pCharacteristic) override {
  String rx = pCharacteristic->getValue();   // ahora lo recibe como texto
para que cuando en la aplicacion se ejecute un boton, el esp32 lo tome y
pueda hacer la accion segun correponda con el boton que se presiono
  if (rx.length() == 0) return;

  char c = rx.charAt(0);                     // tomamos el primer caracter
o letra que da la aplicacion o el boton presionado
  Serial.print("Recibido: ");
  Serial.println(c);

  switch (c) {
    case 'A': digitalWrite(LED1_PIN, HIGH); break;
    case 'a': digitalWrite(LED1_PIN, LOW);  break;
    case 'B': digitalWrite(LED2_PIN, HIGH); break;
    case 'b': digitalWrite(LED2_PIN, LOW);  break;
    case 'C': digitalWrite(LED3_PIN, HIGH); break;
    case 'c': digitalWrite(LED3_PIN, LOW);  break;
    default:
      Serial.println("Comando no reconocido");
      break;
  }
}

};
```

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

  pinMode(LED1_PIN, OUTPUT);
  pinMode(LED2_PIN, OUTPUT);
  pinMode(LED3_PIN, OUTPUT);
  setAll(false);

  BLEDevice::init("ESP32_EquipoSicsSeven");
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  BLEService *pService = pServer->createService(SERVICE_UUID);

  BLECharacteristic *pCharacteristic = pService->createCharacteristic(
    CHARACTERISTIC_UUID,
    BLECharacteristic::PROPERTY_WRITE     |
    BLECharacteristic::PROPERTY_WRITE_NR  | // por si la app escribe sin
respuesta
    BLECharacteristic::PROPERTY_READ
  );

  pCharacteristic->setCallbacks(new MyCallbacks());

  pService->start();
  pServer->getAdvertising()->start();

  Serial.println("ESP32 listo (BLE)");
}

void loop() {
  // no es necesario hacer nada aqui ya que el funcionamiento se hace
afuera del loop porque desde ahi se puede hacer
}
```

Evidencias y video de demostración:

Video demostrativo:

https://youtube.com/shorts/lvOKkD2mKzA?feature=share

Conclusiones:

Damian: Este proyecto permitió al equipo consolidar conocimientos sobre la arquitectura de microcontroladores modernos como el ESP32 y su interacción con dispositivos móviles. Se demostró que, mediante herramientas de programación por bloques como MIT App Inventor y el uso de librerías estándar de BLE, es posible prototipar soluciones de Internet de las Cosas (IoT) funcionales en corto tiempo. El sistema final cumple con todos los requisitos planteados, ofreciendo una respuesta inmediata (tiempo real) al encender y apagar los indicadores LED, validando así la lógica de programación empleada para la gestión de puertos GPIO y la transmisión de datos seriales inalámbricos.

Eduardo: En general, la práctica nos permitió entender de manera sencilla cómo usar el ESP32 con BLE para controlar LEDs desde una app del celular. Vimos que la comunicación es rápida, funciona bien y es fácil de implementar. Además, la integración con MIT App Inventor nos mostró que se pueden crear controles inalámbricos sin mucha complicación, aunque si batallamos un poco para entenderlo inicialmente.

Itzel: En conclusión, en este proyecto logramos implementar correctamente el control inalámbrico de tres LEDs utilizando tecnología BLE con la ESP32 y una aplicación creada en MIT App Inventor. Pudimos comprobar en la práctica las diferencias entre Bluetooth clásico y BLE, entendiendo por qué BLE es más adecuado para proyectos IoT debido a su bajo consumo de energía y su eficiencia para enviar datos pequeños.

Además, integramos conocimientos de programación, electrónica y comunicación inalámbrica, logrando que la app se conectara al ESP32 y enviara comandos para encender y apagar los LEDs en tiempo real. Considero que el objetivo se cumplió, ya que el sistema funcionó de manera estable y demostró cómo se puede aplicar BLE en proyectos prácticos y funcionales.