Module 6 : Bluetooth

Module 6: Bluetooth

Introduction to Bluetooth Technology

What is Bluetooth?

Bluetooth is a wireless technology that enables devices to communicate with each other over short distances, typically up to 10 meters. Bluetooth uses radio waves in the 2.4 GHz frequency band to transmit data between devices, such as smartphones, laptops, speakers, headphones, keyboards, mice, printers, and more. Bluetooth can also be used for low-power applications, such as health and fitness sensors, smart home devices, and wearable gadgets.

Bluetooth is a standard that defines how devices can discover, connect, and exchange data with each other. There are various versions of Bluetooth, each with different features and capabilities, such as Classic Bluetooth, Bluetooth Low Energy (BLE), and Bluetooth Mesh. Each version of Bluetooth has its own specifications and protocols that determine how devices interact with each other.

History and Evolution of Bluetooth

The name "Bluetooth" is inspired by a 10th-century Danish king named Harald Bluetooth, who unified the tribes of Denmark and Norway. The Bluetooth logo is a combination of runes for his initials. The idea of Bluetooth was conceived in 1989 by Nils Rydbeck, a chief technology officer at Ericsson Mobile in Sweden. He wanted to create a wireless headset that could connect to a mobile phone. He assigned Tord Wingren, Jaap Haartsen, and Sven Mattisson to work on the project.

In 1994, they developed a prototype of short-range radio technology that could connect a phone and headset. They called it the "Multi-Communicator Link." In 1997, they joined forces with other companies like Intel, Nokia, IBM, and Toshiba to form the Bluetooth Special Interest Group (SIG), a consortium that would develop and promote the technology. In 1998, they officially named the technology "Bluetooth" and released the first specifications for it.

Since then, Bluetooth has evolved through several versions, enhancing performance, security, reliability, and functionality. Major versions include:

1. **Bluetooth 1.0 (1999)**: The first version of Bluetooth supporting data speeds up to 1 Mbps and voice communication.
2. **Bluetooth 2.0 + EDR (2004)**: Enhanced Data Rate (EDR) increased data speeds up to 3 Mbps and reduced power consumption.
3. **Bluetooth 3.0 + HS (2009)**: High Speed (HS) added an optional feature using Wi-Fi for faster data transfers up to 24 Mbps.
4. **Bluetooth 4.0 (2010)**: Introduced Bluetooth Low Energy (BLE), a new mode that allows low-power devices to operate with very low energy consumption.
5. **Bluetooth 5.0 (2016)**: Increased range up to four times and speed up to twice that of Bluetooth 4.2, along with new features for broadcasting and location services.
6. **Bluetooth 5.1 (2019)**: Added direction-finding capabilities to allow devices to determine the angle of arrival or departure of a Bluetooth signal.
7. **Bluetooth 5.2 (2020)**: Improved audio transmission quality and efficiency and added support for LE Audio, a new standard for wireless audio.

Advantages & Applications of Bluetooth

Bluetooth is a vital technology with numerous benefits and applications across various domains. Some advantages of Bluetooth include:

1. **Wireless**: Eliminates the need for cumbersome cables.
2. **Universal**: Can work with various types of devices from different manufacturers and platforms.
3. **Easy to use**: Does not require complex setups or configurations.
4. **Secure**: Uses encryption and authentication mechanisms to protect data from unauthorized access or interference.
5. **Low-cost**: Does not require expensive hardware or infrastructure to operate.

Applications of Bluetooth:

1. **Wireless Audio**: Bluetooth enables high-quality wireless audio streaming between devices like speakers, headphones, microphones, car stereos, etc.
2. **Wireless Data**: Bluetooth enables wireless data transfer between computers, smartphones, tablets, printers, scanners, cameras, etc.
3. **Wireless Control**: Bluetooth allows devices to control other devices wirelessly using keyboards, mice, game controllers, remote controls, etc.
4. **Wireless Networking**: Bluetooth enables devices to form networks wirelessly using piconets and scatternets.
5. **Wireless Sensors**: Bluetooth enables devices to collect and transmit sensor data wirelessly using health monitors, fitness trackers, smartwatches, etc.
6. **Wireless Location**: Bluetooth allows devices to determine their location and direction wirelessly using beacons and direction-finding.

Key Characteristics of Bluetooth Technology

1. **Frequency Band**: Bluetooth operates in the ISM (Industrial, Scientific, and Medical) 2.4 GHz frequency band, which is globally available and unlicensed. Bluetooth uses 79 channels with 1 MHz spacing in this band and uses frequency hopping spread spectrum (FHSS) to avoid interference and improve security.
2. **Modulation**: Bluetooth uses various modulation schemes to encode data into radio signals, such as Gaussian frequency shift keying (GFSK), phase shift keying (PSK), and quadrature amplitude modulation (QAM). The choice of modulation depends on the Bluetooth version and mode.
3. **Data Rate**: Bluetooth supports different data rates depending on the Bluetooth version and mode. The maximum data rate for Classic Bluetooth is 3 Mbps, while the maximum data rate for Bluetooth Low Energy is 2 Mbps. The data rate also depends on the modulation scheme and channel conditions.
4. **Range**: Bluetooth supports different ranges depending on the Bluetooth version and mode. The typical range for Classic Bluetooth is up to 10 meters, while the typical range for Bluetooth Low Energy is up to 100 meters. The range also depends on transmission power, receiver sensitivity, and environmental factors.
5. **Power Consumption**: Bluetooth supports different power consumption levels depending on the Bluetooth version and mode. The typical power consumption for Classic Bluetooth is about 100 mW, while the typical power consumption for Bluetooth Low Energy is about 1 mW. Power consumption also depends on data rate, duty cycle, and sleep mode.
6. **Topology**: Bluetooth supports different topologies depending on the Bluetooth version and mode. The basic topology for Classic Bluetooth is point-to-point connections between two devices, called a piconet. A piconet can have up to eight active devices, where one acts as a master and the others as slaves. Multiple piconets can connect to form a larger network, called a scatternet. The basic topology for Bluetooth Low Energy is a star network, where one device acts as a central hub and connects to multiple peripheral devices. Bluetooth Low Energy also supports mesh networking, where devices can relay messages to each other without a central hub.

Bluetooth Classic

Bluetooth Classic vs. Bluetooth Low Energy (BLE): Key Differences

Bluetooth Classic and BLE are two distinct technologies that use the same 2.4 GHz ISM band but have different characteristics and purposes. Key differences include:

1. **Power Consumption**: BLE devices consume significantly less power than Classic Bluetooth devices, making them ideal for battery-powered applications that require only periodic data transfer. Classic Bluetooth devices consume more power but can stream data continuously and support higher data rates.
2. **Data Rate**: Classic Bluetooth devices can achieve a maximum data rate of 3 Mbps, while BLE devices can reach a maximum data rate of 1 Mbps.
3. **Range**: Classic Bluetooth devices can have a range of up to 100 meters, while BLE devices can have a range of up to 50 meters, although actual range depends on various factors such as environment, antenna design, and interference levels.
4. **Compatibility**: Classic Bluetooth devices support backward compatibility with previous Bluetooth versions, allowing them to connect with older devices that support the same profiles. BLE devices do not support backward compatibility with Classic Bluetooth devices, meaning they can only connect with other BLE devices or dual-mode devices that support both technologies.
5. **Profiles**: Classic Bluetooth devices support a variety of profiles that define device functions and use cases. For example, A2DP (Advanced Audio Distribution Profile) allows stereo audio streaming, HFP (Hands-Free Profile) allows wireless speakerphone functionality, and AVRCP (Audio/Video Remote Control Profile) allows remote control of audio and video devices. BLE devices support a set of profiles based on the GATT (Generic Attribute Profile) protocol, enabling flexible and customizable data exchange between devices. For example, HRP (Heart Rate Profile) allows heart rate measurement, FMP (Find Me Profile) allows device tracking, and PXP (Proximity Profile) allows device proximity detection.

Bluetooth Classic Architecture

The architecture of Classic Bluetooth consists of two main components: Host and Controller. The Host is the part of the device that runs application layers and upper layers of the Bluetooth protocol stack, such as L2CAP (Logical Link Control and Adaptation Protocol), RFCOMM (Radio Frequency Communication), SDP (Service Discovery Protocol), and various profiles. The Controller is the part of the device that runs the lower layers of the Bluetooth protocol stack, such as HCI (Host Controller Interface), Baseband, Link Manager, and Radio. The Host and Controller communicate with each other through the HCI layer, which provides a standard interface for sending commands and events between them.

Pairing and Connection in Bluetooth Classic

Pairing is the process of establishing a trusted relationship between two Bluetooth devices by exchanging security information such as encryption keys and PIN codes. Pairing allows devices to authenticate each other and prevent unauthorized access to their data and services.

There are various pairing methods depending on the Bluetooth version and type of devices involved. The most common method is **Secure Simple Pairing (SSP)**, introduced in Bluetooth 2.1 + EDR, which uses four association models: **Numeric Comparison**, **Just Works**, **Passkey Entry**, and **Out Of Band (OOB)**. These models differ in how they display or exchange confirmation values or PIN codes between devices to verify their identity.

Connection is the process of establishing a logical link between two paired Bluetooth devices to transfer data. The connection can be initiated by either device, depending on the device's role and status. A device can act as either **Master** or **Slave**, depending on whether it initiates or accepts the connection. A device can also be in **Page** or **Inquiry** mode, depending on whether it sends or receives a connection request.

Bluetooth Profiles and Protocols

Bluetooth Profiles are specifications that define the functions and use cases of Bluetooth devices. Profiles determine which protocols and parameters a device uses to communicate with other devices that support the same profile. A device may support multiple profiles depending on its capabilities and features.

Bluetooth Protocols are sets of rules and procedures that regulate how data is exchanged between Bluetooth devices. A protocol operates on a specific layer of the Bluetooth protocol stack and provides services to the upper or lower layers. A protocol may be used by one or more profiles, depending on their requirements.

The following table lists some common Bluetooth profiles and protocols along with their descriptions:

Profile	Description
A2DP	Advanced Audio Distribution Profile. Enables high-quality audio streaming between devices.
AVRCP	Audio/Video Remote Control Profile. Allows remote control of audio and video devices.
HFP	Hands-Free Profile. Enables wireless speakerphone functionality between devices.
HSP	Headset Profile. Enables basic audio communication between headsets and mobile phones.
PBAP	Phone Book Access Profile. Allows access to phone book entries and call history on mobile phones.

Profile	Description
SPP	Serial Port Profile. Enables serial data communication between devices.

Protocol	Description
L2CAP	Logical Link Control and Adaptation Protocol. Provides packet segmentation and reassembly, logical channel multiplexing, and quality of service (QoS) management.
RFCOMM	Radio Frequency Communication. Provides serial port emulation over L2CAP channels.
SDP	Service Discovery Protocol. Provides service discovery and attribute information on Bluetooth devices.
HCI	Host Controller Interface. Provides a standard interface for communication between the Host and Controller.
Baseband	Defines the physical layer of the Bluetooth protocol stack, including frequency hopping, modulation, encryption, and error correction.
Link Manager	Manages the link layer of the Bluetooth protocol stack, including link setup, authentication, encryption, and power control.

Bluetooth Low Energy (BLE)

Introduction to BLE

Bluetooth Low Energy (BLE), also known as Bluetooth Smart, is a wireless communication technology designed for short-range data exchange between electronic devices. It emerged in response to the need for energy-efficient wireless communication in various applications, especially where power consumption is a critical concern.

BLE is a subset of the Bluetooth 4.0 specification, which also includes the classic Bluetooth protocol. BLE is not compatible with classic Bluetooth, but both can coexist on the same device and share the same radio frequency. BLE has a simpler modulation system and a lower data rate compared to classic Bluetooth, enabling it to achieve lower power consumption and cost.

BLE is suitable for applications requiring sporadic or periodic data transfer, such as sensors, beacons, fitness trackers, smartwatches, and remote controls. BLE also supports mesh networking, where multiple devices can relay data to each other and form large networks.

BLE Architecture and Terms

BLE has a layered architecture consisting of several components, such as the physical layer, link layer, host, and controller. The host and controller are logical entities that can be implemented on the same or separate chips. The host contains high-level protocols and profiles, such as the Generic Attribute Profile (GATT) and Generic Access Profile (GAP). The controller contains low-level protocols and functions, such as the physical layer and link layer.

Some key terms and concepts in BLE include:

- **GATT**: The Generic Attribute Profile defines how devices can exchange data using a common format and structure. GATT is based on the concept of attributes, which are data elements with a unique identifier (UUID), type, value, and permissions. Attributes are organized into services and characteristics, which represent logically related sets of attributes. For example, a service may represent a device function, like a heart rate monitor, and characteristics may represent features of that function, like heart rate measurements.
- **GAP**: The Generic Access Profile defines how devices can discover, connect, and link to each other. GAP also defines the roles and modes that devices can take in BLE communication. The primary roles are:
  - **Peripheral**: A device that advertises its presence and offers services to other devices. A peripheral can accept only one connection at a time.
  - **Central**: A device that scans advertisements and initiates connections with peripheral devices. A central device can connect to multiple peripherals simultaneously.
  - **Broadcaster**: A device that only advertises its presence and does not accept connections. A broadcaster can send data to multiple devices without establishing a connection.
  - **Observer**: A device that only scans advertisements and does not initiate connections. An observer can receive data from multiple broadcasters without forming a connection.

  The primary modes are:
  - **Advertising**: A mode where a device transmits packets containing information about itself and its services.
  - **Scanning**: A mode where a device listens for advertisement packets from other devices.
  - **Connecting**: A mode where two devices establish a bidirectional link and exchange data using GATT.
  - **Bonding**: A mode where two devices form a trusted relationship by exchanging security keys and storing them for future use.

BLE Advertising & Connection Procedures

Here is an overview of these procedures:

- **Advertising**: The peripheral device enters advertising mode and sends packets containing the device address, device name, service UUID, and other information. Advertising packets can be of three types: connectable directed, connectable undirected, and non-connectable. Connectable undirected packets indicate that the peripheral is open to connections from any central device. Connectable directed packets indicate that the peripheral is open to connections from a specific central device only. Non-connectable packets indicate that the peripheral is not open for connection at all.
- **Scanning**: The central device enters scanning mode and listens for advertising packets from other devices. Scanning mode can be passive or active. In passive scanning, the central device only receives advertising packets without sending a response. In active scanning, the central device sends a scan request packet to the peripheral after receiving an advertising packet, and the peripheral responds with a scan response packet containing additional information.
- **Connecting**: The central device initiates a connection with the peripheral device by sending a connection request packet after receiving an advertising packet from the peripheral. The connection request packet contains parameters for the connection, such as connection interval, slave latency, and supervision timeout. The peripheral accepts the connection request and sends an acknowledgment packet to the central device. Both devices then enter connected mode and exchange data using GATT.
- **Bonding**: Two connected devices can optionally enter bonding mode to form a trusted relationship by exchanging security keys and storing them for future use. Bonding mode can be either Just Works or Passkey Entry. In Just Works bonding, devices use a fixed key of 000000 to encrypt their communication. In Passkey Entry bonding, devices use a randomly generated six-digit key to encrypt their communication. The key can be entered manually by the user or displayed on the device screen.

ESP32 & Bluetooth Integration

Bluetooth Classic in ESP32

To use Bluetooth Classic on the ESP32, include the `BluetoothSerial` library in your code:

```
#include "BluetoothSerial.h"
```

Then, create an instance of the BluetoothSerial class:

```
BluetoothSerial SerialBT;
```

In the setup() function, initialize serial communication and start the Bluetooth serial device with a name:

```
void setup() {
  Serial.begin(115200);               // start serial communication
  SerialBT.begin("ESP32test");        // start Bluetooth serial device with the name "ESP32test"
  Serial.println("Device started, now you can pair it with Bluetooth!");
}
```

In the loop() function, you can send and receive data through Bluetooth serial as you would with normal serial communication:

```
void loop() {
  if (Serial.available()) {                 // if data is available in the serial monitor
    SerialBT.write(Serial.read());          // send data to the Bluetooth device
  }
  if (SerialBT.available()) {               // if data is available from the Bluetooth device
    Serial.write(SerialBT.read());          // print data in the serial monitor
  }
  delay(20);
}
```

To test this code, upload it to your ESP32 board and open the serial monitor. Then, pair your smartphone or computer with the ESP32 via Bluetooth under the name "ESP32test." Afterward, open a Bluetooth terminal app and connect to the ESP32. You should be able to send and receive messages between the serial monitor and the app.

Bluetooth Low Energy in ESP32

To use BLE on the ESP32, include several libraries that are part of the ESP32 add-on:

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

Define some variables and constants used for BLE communication:

```
// UUID for service and characteristic
#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"

// BLE server and characteristic objects
BLEServer* pServer = NULL;
```

```
BLECharacteristic* pCharacteristic = NULL;

// flag to notify client
bool deviceConnected = false;
```

Define a class to handle server events, such as client connections and disconnections:

```
// class to handle server events
class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;  // set flag when client is connected
  };
  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false; // clear flag when client is disconnected
  }
};
```

Create a characteristic for the service and set properties and an initial value for it:

```
pCharacteristic = pService->createCharacteristic(
  CHARACTERISTIC_UUID,
  BLECharacteristic::PROPERTY_READ |   // set read, write, and notify properties
  BLECharacteristic::PROPERTY_WRITE |
  BLECharacteristic::PROPERTY_NOTIFY
);
pCharacteristic->setValue("Hello World");  // set initial value for the characteristic
```

Define a class to handle characteristic events, such as reading and writing data:

```
// class to handle characteristic events
class MyCallbacks: public BLECharacteristicCallbacks {
  void onRead(BLECharacteristic *pCharacteristic) {
    Serial.println("Read request received");  // print message when read request is received
  }
  void onWrite(BLECharacteristic *pCharacteristic) {
    std::string value = pCharacteristic->getValue();  // get the characteristic value
    if (value.length() > 0) {  // if value is not empty
      Serial.print("New value: ");  // print message and value
      for (int i = 0; i < value.length(); i++) {
        Serial.print(value[i]);
      }
```

```
      Serial.println();
    }
  }
};
```

Set the callback for characteristics and start the service:

```
pCharacteristic->setCallbacks(new MyCallbacks());  // set callback for characteristic events
pService->start();  // start the service
```

Start advertising the service so other devices can discover it:

```
BLEAdvertising *pAdvertising = pServer->getAdvertising();  // get advertising object
pAdvertising->start();  // start advertising
Serial.println("Waiting for client connection...");
```

In the loop() function, send notifications to the connected client with the current value of the characteristic:

```
void loop() {
  if (deviceConnected) {  // if a client is connected
    pCharacteristic->setValue("Hello from ESP32");  // set new value for the characteristic
    pCharacteristic->notify();                      // send notification to the client
    Serial.println("Notification sent");
    delay(1000);  // wait one second
  }
}
```

To test this code, upload it to your ESP32 board and open the serial monitor. Then, use a BLE scanner app on your smartphone or computer to search for nearby BLE devices. You should see the ESP32 with the name "ESP32test" and the service UUID. Connect to the ESP32 and find its characteristic. You should see the characteristic UUID and its value. You can also read and write data to the characteristic using the app. Messages should appear in the serial monitor.

Another Example: BLE Server

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
```

```
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"

class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    Serial.println("Device connected");
  };
  void onDisconnect(BLEServer* pServer) {
    Serial.println("Device disconnected");
  }
};

class MyCharacteristicCallbacks: public BLECharacteristicCallbacks {
  void onRead(BLECharacteristic *pCharacteristic) {
    Serial.println("Characteristic read");
  };
  void onWrite(BLECharacteristic *pCharacteristic) {
    Serial.println("Characteristic written");
  }
};

void setup() {
  Serial.begin(115200);
  BLEDevice::init("ESP32 Device");
  BLEServer *pServer = BLEDevice::createServer();
  BLEService *pService = pServer->createService(SERVICE_UUID);
  BLECharacteristic *pCharacteristic = pService->createCharacteristic(
    CHARACTERISTIC_UUID,
    BLECharacteristic::PROPERTY_READ |
    BLECharacteristic::PROPERTY_WRITE |
    BLECharacteristic::PROPERTY_NOTIFY |
    BLECharacteristic::PROPERTY_INDICATE
  );
  pCharacteristic->addDescriptor(new BLE2902());
  pCharacteristic->setCallbacks(new MyCharacteristicCallbacks());
  pService->start();
  pServer->getAdvertising()->start();
  pServer->setCallbacks(new MyServerCallbacks());
}

void loop() {
```

```
}
```

BLE Client

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEScan.h>
#include <BLEAdvertisedDevice.h>

#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"

class MyClientCallback : public BLEClientCallbacks {
  void onConnect(BLEClient* pClient) {
    Serial.println("Connected to server");
  }
  void onDisconnect(BLEClient* pClient) {
    Serial.println("Disconnected from server");
  }
};

void setup() {
  Serial.begin(115200);
  BLEDevice::init("ESP32 Client");
  BLEClient* pClient = BLEDevice::createClient();
  pClient->connect("ESP32 Device");
  BLEService* pService = pClient->getService(SERVICE_UUID);
  BLECharacteristic* pCharacteristic =
      pService->getCharacteristic(CHARACTERISTIC_UUID);
  pClient->setCallbacks(new MyClientCallback());
}

void loop() {
}
```

External Reference

Bluetooth Classic

Bluetooth Low Energy (BLE)

BLE Server

BLE Client

BLE Characteristics & Callbacks