TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA ĐIỆN ĐIỆN TỬ

BỘ MÔN KỸ THUẬT MÁY TÍNH - VIỄN THÔNG

ĐỒ ÁN MÔN HỌC 1

**THUẬT TOÁN KHÁM PHÁ THIẾT BỊ BLE TIẾT KIỆM NĂNG LƯỢNG**

**NGÀNH CÔNG NGHỆ KỸ THUẬT MÁY TÍNH**

|  |  |
| --- | --- |
| Sinh viên | : **TRẦN MINH HÀO** |
| MSSV | : 19119173 |

TP. HỒ CHÍ MINH – 06/2022

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA ĐIỆN ĐIỆN TỬ

BỘ MÔN KỸ THUẬT MÁY TÍNH - VIỄN THÔNG

ĐỒ ÁN MÔN HỌC 1

**THUẬT TOÁN KHÁM PHÁ THIẾT BỊ BLE TIẾT KIỆM NĂNG LƯỢNG**

**NGÀNH CÔNG NGHỆ KỸ THUẬT MÁY TÍNH**

|  |  |
| --- | --- |
| Sinh viên | : **TRẦN MINH HÀO** |
| MSSV | : 19119173 |

Hướng dẫn**: PGS TS. VÕ MINH HUÂN**

TP. HỒ CHÍ MINH – 6/2022

**BẢN NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪNLỜI CẢM ƠN**

Tôi xin gửi lời cảm ơn chân thành đến thầy **Võ Minh Huân** – giảng viên khoa Điện – Điện Tử đã trang bị những kiến thức, kỹ năng cần thiết để tôi hoàn thành đồ án này, cũng như quan tâm theo dõi xuyên suốt quá trình thực hiện đồ án.

Trong quá trình nghiên cứu đề tài, hạn chế về mặt thời gian cũng như kiến thức chưa đủ nên sản phẩm sẽ có nhiều thiếu sót về mặt tìm hiểu, đánh giá cũng như trình bày đề tài. Rất mong nhận được sự quan tâm, góp ý của các quí thầy/ cô giảng viên bộ môn để tôi có thể cải thiện đề tài đầy đủ và hoàn chỉnh hơn.

Xin chân thành cảm ơn!

**TÓM TẮT**

Được biết tới là một công nghệ truyền thông không dây tầm ngắn và kết nối ngang hàng peer-to-peer, Bluetooth Low Energy (BLE) nhận được nhiều sự quan tâm trong khoảng thời gian này. Với đặc trưng là có thể khám phá các thiết bị ở vùng lân cận, BLE phù hợp với các ứng dụng Internet of things (IoT) mang tính di động khác nhau. Rõ ràng là, với một hệ thống di động có nguồn năng lượng hạn chế thì việc cân bằng giữa số lượng thiết bị khám phá được và mức tiêu thụ năng lượng cho việc này là hết sức quan trọng. Đồ án này triển khai một cơ chế khám phá thiết bị thông minh, đặt tên là hDiscovery, có khả năng tự điều chỉnh các thông số ScanWindow và ScanInterval theo sự biến đổi của môi trường thiết bị xung quanh, đồng thời thay đổi các chế độ ngủ tương ứng. Kết quả cho thấy hDiscovery tiết kiệm năng lượng hơn là cơ chế khám phá thiết bị thông thường. Tôi đã triển khai mô hình thuật toán này trên bộ phát triển ESP32, một vi điều khiển hỗ trợ BLE, bên cạnh đó tôi cũng sử dụng module nRF51822 để tìm hiểu chức năng của một thiết bị phát bản tin quảng bá trong mô hình BLE.

**MỤC LỤC**

[**DANH MỤC HÌNH ẢNH** vi](#_Toc106906787)

[**CÁC TỪ VIẾT TẮT** vii](#_Toc106906788)

[Chương 1 Giới thiệu 1](#_Toc106906789)

[1.1 GIỚI THIỆU 1](#_Toc106906790)

[1.2 MỤC TIÊU ĐỀ TÀI 2](#_Toc106906791)

[1.3 GIỚI HẠN ĐỀ TÀI 2](#_Toc106906792)

[1.4 PHƯƠNG PHÁP NGHIÊN CỨU 3](#_Toc106906793)

[1.5 ĐỐI TƯỢNG VÀ PHẠM VI NGHIÊN CỨU 3](#_Toc106906794)

[1.6 BỐ CỤC QUYỂN BÁO CÁO 3](#_Toc106906795)

[Chương 2 CƠ SỞ LÝ THUYẾT 4](#_Toc106906796)

[2.1 BLUETOOTH LOW ENERGY (BLE) 4](#_Toc106906797)

[2.1.1 Tổng quan 4](#_Toc106906798)

[2.1.2 Bluetooth Low Energy Stack 5](#_Toc106906799)

[2.1.3 Advertising và Scanning 6](#_Toc106906800)

[2.2 PHẦN CỨNG VÀ KIẾN THỨC LIÊN QUAN 10](#_Toc106906801)

[2.2.1 ESP32 DevKit và các chế độ ngủ (Sleep modes) 10](#_Toc106906802)

[2.2.2 Module BLE nRF51822 và phần mềm KeilC 15](#_Toc106906803)

[Chương 3 thiết kế và triển khai thuật toán 17](#_Toc106906804)

[3.1 SƠ ĐỒ TỔNG QUÁT 17](#_Toc106906805)

[3.2 PHÁT TRIỂN THIẾT BỊ BLE BEACON VỚI NRF51822 18](#_Toc106906806)

[3.2.1 Lưu đồ thuật toán 18](#_Toc106906807)

[3.2.2 Firmware 19](#_Toc106906808)

[3.3 PHÁT TRIỂN THIẾT BỊ BLE SCANNER VỚI ESP32 22](#_Toc106906809)

[3.3.1 Thuật toán hDiscovery 22](#_Toc106906810)

[3.3.2 ESP32 Scanner với hDiscovery 23](#_Toc106906811)

[Chương 4 kết quả 28](#_Toc106906812)

[4.1 HOẠT ĐỘNG CỦA BEACON 28](#_Toc106906813)

[4.1.1 Phần mềm nRF Connect 28](#_Toc106906814)

[4.1.2 Phần tích gói tin Beacon 28](#_Toc106906815)

[4.2 HOẠT ĐỘNG CỦA SCANNER 31](#_Toc106906816)

[Chương 5 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 34](#_Toc106906817)

[5.1 KẾT LUẬN 34](#_Toc106906818)

[5.2 HƯỚNG PHÁT TRIỂN 34](#_Toc106906819)

[PHỤ LỤC 35](#_Toc106906820)

[TÀI LIỆU THAM KHẢO 36](#_Toc106906821)

DANH MỤC HÌNH ẢNH

Hình 2.1 Một số đặc trưng của BLE so với Classic Bluetooth……………………..4

Hình 2.2 Bluetooth Low Energy Stack……………………………………………..6

Hình 2.3 BLE mô hình Connection-oriented…………………………………….....7

Hình 2.4 BLE và mô hình Connectionless………………………………………....7

Hình 2.5 BLE và mô hình Mesh …………………………………………………...8

Hình 2.6 Advertising Data………………………………………………………….8

Hình 2.7 Advertising và Scanning …………………………………………………9

Hình 2.8 ESP32 Development Kit……………………………………………...…10

Hình 2.9 ESP32 pinout……………………………………………...…………….11

Hình 2.10 ESP32 Active mode……………………………………………………12

Hình 2.11 ESP32 Modem Sleep mode……………………………………………13

Hình 2.12 ESP32 Light Sleep mode………………………………………………14

Hình 2.13 ESP32 Deep Sleep mode………………………………………………14

Hình 2.14 Module BLE nRF51822……………………………………………….16

Hình 3.1 Sơ đồ hệ thống…………………………………………………………..17

Hình 3.2 Lưu đồ thuật toán tổng quát của Beacon………………………………..18

Hình 3.3 Lưu đồ thuật toán tổng quát của Scanner………………………………..23

Hình 4.1 Phần mềm nRF Connect cho Android…………………………………..28

Hình 4.2 Quét gói tin bằng phần mềm nRF Connect……………………………..29

Hình 4.3 Phân tích gói tin Advertising với nRF Connect…………………………30

Hình 4.4 Scanner ESP32…………………………………………………………..31

Hình 4.5 Peers and RedundantPeers………………………………………………32

Hình 4.6 Chế độ Light Sleep………………………………………………………32

CÁC TỪ VIẾT TẮT

|  |  |
| --- | --- |
| BLE | Bluetooth Low Energy |
| IoT | Internet of things |
| DevKIT | Development KIT |
| IDE | Integrated Development Environment |
| RF | Radio Frequency |
| ISM | Industrial – Scientific – Medical |
| SoC | System on Chip |
| RTC | Real Time Clock |
| ADC | Analog-to-Digital Converter |
| DAC | Digital-to-Analog Converter |
| PWM | Pulse-Width Modulation |
| GPIO | General Purpose Input Output |
| PPI | Programmable Peripheral Interconnect |
| TI | Texas Instruments |

# Giới thiệu

## GIỚI THIỆU

Bluetooth Low Energy (BLE) đã nhận được rất nhiều sự quan tâm chủ yếu do mức tiêu thụ điện năng thấp, lượng dữ liệu trao đổi nhỏ và kết nối liên lạc có thể không liên tục. BLE hiện đã được áp dụng rộng rãi cho nhiều loại thiết bị, bao gồm điện thoại thông minh hay các thiết bị gia dụng, các máy tính đeo được (đồng hồ thông minh, cảm biến…), thiết bị y tế và sức khỏe. Tính năng device-to-device của BLE là mối quan tâm lớn đối với sự phát triển của Internet of things (IoT) vì có thể kết nối để tương tác trực tiếp giữa các thiết bị. So sánh với các chuẩn bluetooth truyền nhận dữ liệu lớn hay âm thanh thông thường, cần phải duy trì kết nối gần như liên tục và thông lượng dữ liệu lớn theo thời gian, thì BLE phù hợp với các ứng dụng gửi nhận các tín hiệu điều khiển giửa các thiết bị IoT di động.

Trong môi trường có nhiều thiết bị, việc khám phá thiết bị trở thành thiết yếu để tạo kết nối cũng như đạt được sự trao đổi thông tin thành công. Để phát hiện ra các thiết bị cùng loại trong một vùng không gian, một thiết bị quét (Scanner) cần lắng nghe các thiết bị xung quanh trong khoảng thời gian ScanWindow và theo chu kì ScanInterval. Nếu chúng ta dành nhiều tài nguyên cho việc quét, chẳng hạn tăng ScanWindow và giảm chu kỳ quét lại, chúng ta sẽ khám phá ra nhiều thiết bị hơn, mặc nhiên nó sẽ tiêu tốn nhiều năng lượng hơn. Điều trên là không cần thiết với một môi trường ổn định tức các thiết bị xung quanh ít biến động (nảy sinh thiết bị mới hay giảm bớt thiết bị hiện có), hay trong các ứng dụng không cần đáp ứng về thời gian quá cao. Làm thế nào để phát hiện đủ số lượng thiết bị cần thiết trong một thời gian chấp nhận được mà không gây lãng phí quá nhiều năng lượng là một vấn đề quan trọng. Thuật toán hDiscovery được đề ra trong đề tài này dựa vào các thông về số lượng thiết bị được ghi lại để tuy chỉnh ScanWindow và ScanInterval nhằm thích ứng với môi trường biến đổi, bên cạnh đó đưa hệ thống vào chế độ ngủ trong khoảng thời gian trống để tối ưu việc tiết kiệm năng lượng.

## MỤC TIÊU ĐỀ TÀI

Trên cơ sở các thuật toán cũng chủ đề đã được công bố trước đó, Han và nhóm nghiên cứu của mình [1] lần đầu tiên đề xuất giao thức eDiscovery, xem xét số lượng thiết bị để điều chỉnh ScanWindow và ScanInterval, hay vào năm 2017, nhóm nguyên cứu trường đại học Khoa học và Công nghệ Quốc gia Đài Loan giới thiệu về thuật toán sDiscovery [2] tận dụng thêm thông tin số lượng thiết bị dư thừa.

Lấy cảm hứng từ ý tưởng của hai thuật toán trên, đề tài này nghiên cứu việc triển khai thuật toán sDiscovery trên một kit phát triển khá phổ biến hiện tại (Esp32) và đồng thời đưa vào thuật toán khái niệm đa chế độ ngủ. Trong thuật toán đưa ra này (đặt tên là hDiscovery), nếu số lượng thiết bị dư thừa qua mỗi lần quét lớn hơn một ngưỡng đặt trước, tức môi trường xung quanh ít biến động và số lượng thiết bị đã được nắm bắt, trong trường hợp này tôi phóng to ScanInterval và thu nhỏ ScanWindow, đồng thơi đưa thiết bị quét vào trạng thái ngủ trong khoảng thời gian trống sau khi đã hết vùng ScanWindow.

## GIỚI HẠN ĐỀ TÀI

Đề tài chủ yếu dừng lại trong việc đưa ý tưởng của thuật toán triển khai vào ESP32. Chương trình mô tả thuật toán hDiscovery chưa thật sự tối ưu và sẽ có thiếu sót.

Là một đề tài về mảng nghiên cứu, việc triển khai thử nghiệm, kiểm chứng thuật toán là khá phức tạp và tốn kém. Chi phí cho việc xây dựng một môi trường thử nghiệm với số lượng thiết bị BLE khá lớn và phải biến động là rất cao. Việc Tính toán, so sánh năng lượng tiêu tốn trong cũng một môi trường với những thuật toán khác cũng mang lại nhiều khó khăn. Do vậy, hạn chế lớn nhất của đề tài là chưa thật sự kiếm chứng thuật toán trong một môi trường thực tế cụ thể, cũng như đo lường lại cái thông số về năng lượng tối ưu.

## PHƯƠNG PHÁP NGHIÊN CỨU

Kết hợp các phương pháp nghiên cứu như phương pháp tổng hợp tài liệu lý thuyết, phương pháp mô phỏng, phương pháp tham khảo ý kiến chuyên gia…

## ĐỐI TƯỢNG VÀ PHẠM VI NGHIÊN CỨU

* Bảng phát triển ESP32 (Development KIT - DevKIT) với ngoại vi Bluetooth Low Energy (BLE) chức năng khám phá thiết bị đóng vai trò của một máy quét (Scanner). Lập trình trên nền tảng Arduino IDE với ngôn ngữ lập trình C/C++.
* Module thu phát BLE nRF51822 đóng vai trò như một Beacon phát bản tin quảng bá (Advertiser). Lập trình điều khiển trên nền tảng KeilC IDE sử dụng ngôn ngữ lập trình C.

## BỐ CỤC QUYỂN BÁO CÁO

Bố cục của báo cáo bao gồm 4 chương như sau:

- Chương 1: GIỚI THIỆU

Giới thiệu tổng quan về thuật toán hDiscovery, hướng triển khai đề tài, cũng như một số hạn chế và mục tiêu của đề tài .

- Chương 2: CƠ SỞ LÝ THUYẾT

Chương này trình bày về các kiến thức lý thuyết liên quan tới bản tin quảng cáo cũng như cơ chế thu phát bản tin trong BLE, các thông số, khái niệm quan trọng được sử dụng trong thuật toán hDiscovery. Giới thiệu các phần cứng (ESP32, nRF51822) và phần mềm (Arduino và KeiLC IDE) sử dụng để thực hiện.

- Chương 3: THIẾT KẾ HỆ THỐNG

Trọng tâm chính của đề tài, chương 3 thể hiện hướng đi trong việc triển khai thuật toán hDiscovery Scanner trên ESP32 DevKit và Beacon trên nRF51822.

- Chương 4: KẾT QUẢ

Trình bày các kết quả kiểm tra quá trình hoạt động của hệ thống.

- Chương 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Xem xét kết quả đã thực hiện được, định hướng phát triển cho đề tài.

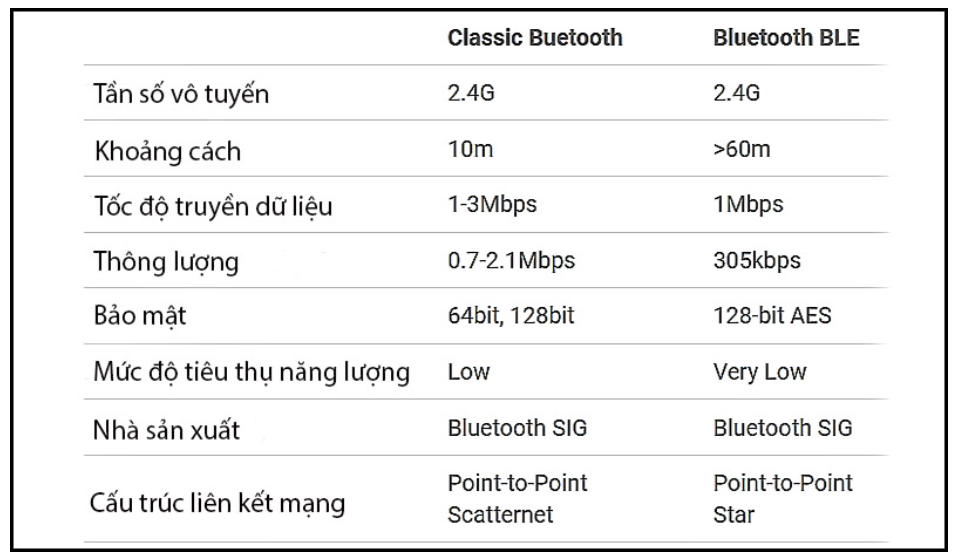
# CƠ SỞ LÝ THUYẾT

## BLUETOOTH LOW ENERGY (BLE)

### Tổng quan

Bluetooth Low Energy (BLE), đôi khi còn được gọi là “Bluetooth Smart”, tạm dịch: Bluetooth năng lượng thấp – là một công nghệ mạng không dây tiêu thụ ít năng lượng, dùng để giao tiếp hoặc kết nối các thiết bị điện tử với nhau.

Tiền thân của BLE ban đầu được Nokia phát minh và nghiên cứu với tên gọi Wibree, sau được tích hợp vào Bluetooth 4.0 và công bố với tên BLE vào năm 2010 bởi tập đoàn **Bluetooth Special Interest Group (Bluetooth SIG)** với mục tiêu là tối ưu năng lượng cho chuẩn truyền thông Bluetooth.



Hình 2.1 Một số đặc trưng của BLE so với Classic Bluetooth

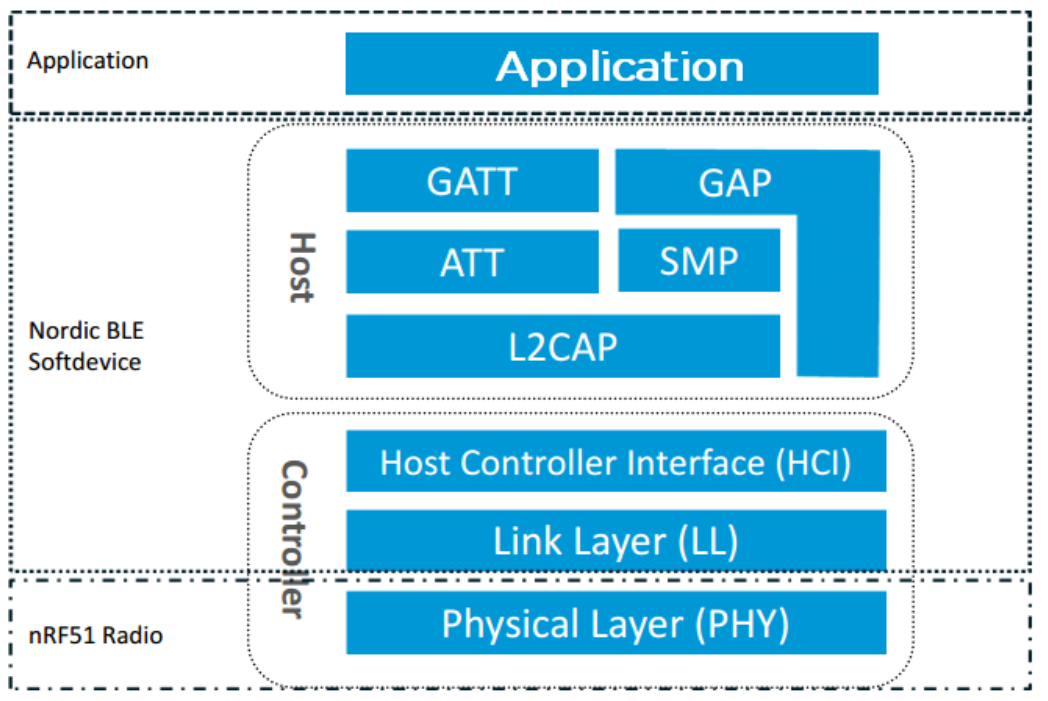
Hình 2.1 trình bày một số đặt điểm của BLE so với Bluetooth tiêu chuẩn (Classic). Có thể thấy, BLE vượt trội trong việc tiết kiệm năng lượng nhưng bù lại thông lượng truyền sẽ không được cao như chuẩn Bluetooth thông thường, một đặc điểm nửa cho thấy phạm vi hoạt động của BLE lớn hơn rất nhiều cho với Bluetooth thông thường (lớn hơn khoảng 6 lần) điều này cũng phần nào ảnh hưởng tới ứng dụng của cả hai là rất khác nhau. Thực chất, hai công nghệ trên được sử dụng cho các mục đích hầu như tách biệt. Trong khi Classic Bluetooth thích hợp để xử lý, truyền, trao đổi một lượng dữ liệu lớn (ví dụ như hình ảnh, âm thanh, hay đoạn video ngắn…) và thường yêu cầu một kết nối được giữ liên tục để đảm bảo tốc độ truyền là nhanh nhất và có thể là đáp ứng thời gian thực (Âm thanh), BLE lại thích hợp hơn trong việc truyền nhận một lượng dữ liệu nhỏ, thông thường là một gói tin chứa một tín hiệu dùng cho việc điều khiển hay báo hiệu… do đó, BLE rất thích hợp trong các ứng dụng IoTs, đặc biệt nó có thể hoạt động trong nhiều năm chỉ với một lượng pin nhỏ, chi phí thấp.

Bluetooth Low Energy ra đời với hy vọng mang lại những hiệu quả tối ưu cho người dùng thông qua các thiết bị IoT, khi mà IoT đang ứng dụng một cách rộng rãi trong các thiết bị công nghệ thông minh với đa dạng lĩnh vực. Chẳng hạn như:

* Thiết bị theo dõi đồ vật: lấy cảm hứng từ việc gắn một thiết bị phát tín hiệu theo chu kì (Beacon) để báo hiệu vị trí tương đối hiện tại của đồ vật.
* Nhà thông minh (Smart Home): Với những đồ vật được gắn thêm module BLE, giúp chúng có thể phối hợp hành động với nhau, hoặc người dùng có thể điều khiển thiết bị bằng Smartphone thông qua BLE.
* Vòng đeo tay thông mình, thiết bị y tế.
* Được nhúng trong các hệ thông IoT, như cảm biến thu thập thông tin (Nhiệt độ, độ ẩm, sức gió…).

### Bluetooth Low Energy Stack

Bluetooth Low Energy hoạt động dựa trên một chồng giao thức được chia thành ba lớp chính là Application, Host và Controller phối hợp với nhau còn được gọi là Bluetooth Low Energy Stack, hình 2.2 cho thấy một cách khái quát về chồng giao thức BLE được cung cấp bởi Nordic (Một công ty bán dẫn lớn chuyên cung cấp các sản phẩm BLE và các thiết bị bán dẫn khác).

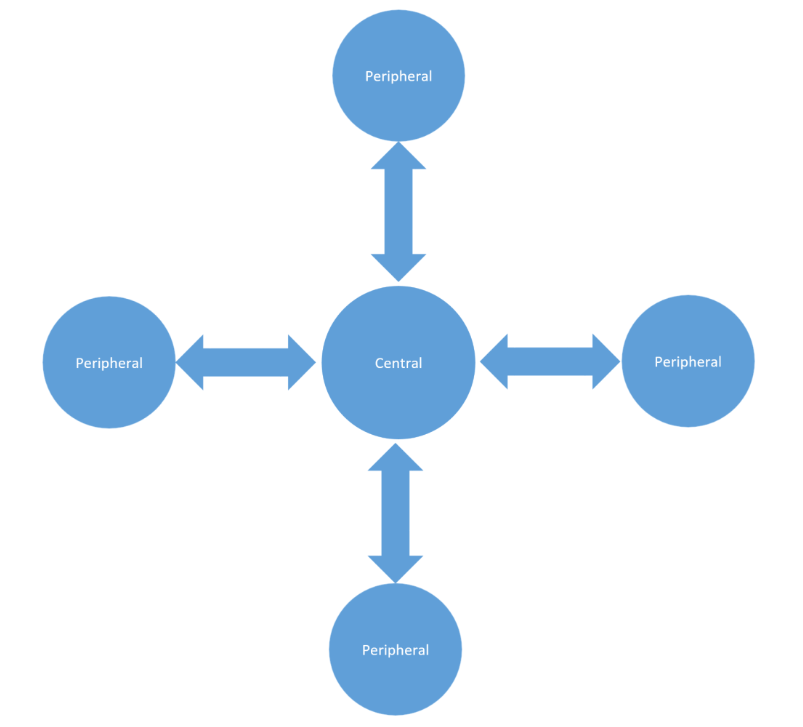


Hình 2.2 Bluetooth Low Energy Stack

Tầng Applicatuon là tầng ứng dụng của người dùng, người dùng có thể viết những chương trình (firmware) cho các ứng dụng có sử dụng BLE. Tầng Host chứa những giao thức thuộc tính và các cấu hình chung thỏa thuận giữa hai phía thiết bị để hiểu lẫn nhau trong BLE. Tầng Controller giữ vai trò liên kết giửa host với tầng vật lý hay Radio giúp biến đổi dữ liệu thành tín hiệu điện để truyền đi và ngược lại khi nhân được dữ liệu. Ở chủ đề lần này, chúng ta không đi sâu vào chồng giao thức này mà chủ yếu phân tích về cơ chế mà hai thiết bị BLE tạo lập kết nối và trao đổi thông tin với nhau dựa trên cơ chế phát và nhận gói tin quảng cáo, tạo kết nối để trao đổi dữ liệu.

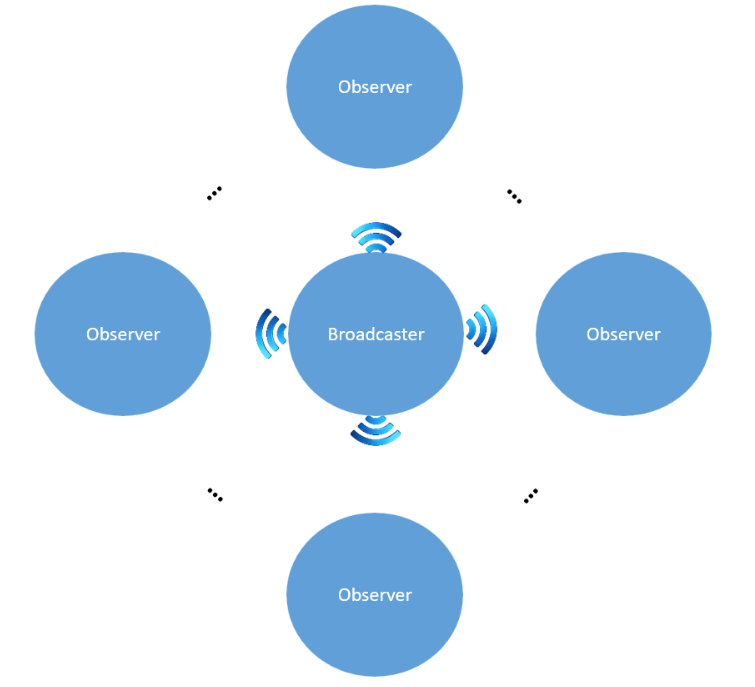
### Advertising và Scanning

Thông thường, việc thiết lập kết nối giửa hai thiết bị sử dụng BLE được coi là một quy trình bất đối xứng, theo đó thiết bị phát quảng cáo gọi là Advertiser sẽ thông báo rằng nó là thiết bị có thể kết nối, trong khi máy quét hay Scanner sẽ lắng nghe các quảng cáo đó. Một mô hình BLE thông thường sẽ có dạng một thiết bị đóng vai trò là Scanner gọi là BLE Central, một hoặc nhiều thiết bị đóng vai trò là Advertiser được gọi là BLE Peripheral, đây là một mô hình có hướng kết nối, thiết bị Central có thể tạo kết nối với thiết bị Peripheral để trao đổi thông tin, còn gọi là mô hình Connection-oriented, thể hiện ở hình 2.3.



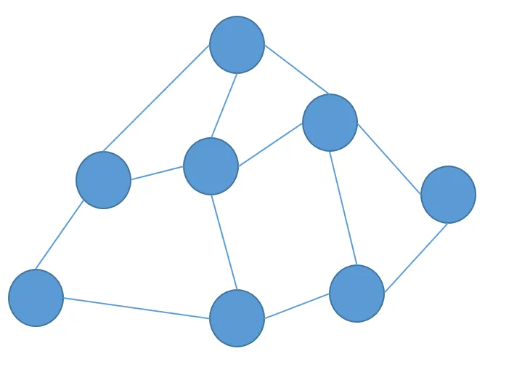
Hình 2.3 BLE mô hình Connection-oriented

Một mô hình thường thấy khác của BLE là dạng Client (Observer) và Sever (Broadcaster). Trong mô hình này, một thiết bị Advertiser đóng vai trò là Sever phát bản tin chứa dữ liệu mà các Client cần, các Client sẽ nhận gói tin từ Sever và thực hiện phân tích xử lý. Ở mô hình này thì không cần thiết cho việc tạo lập kết nối giửa các thiết bị, còn gọi là mô hình Connectionless, hình 2.4.



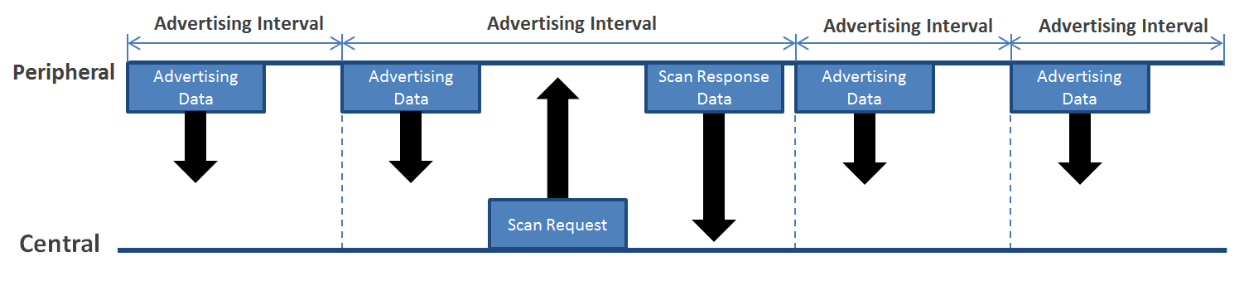
Hình 2.4 BLE và mô hình Connectionless

Một mô hình mới và đang trong quá trình ứng dụng được gọi là mô hình BLE Mesh, đây là mô hình many-to-many tức nhiều thiết bị có thể giao tiếp với nhau tạo thành một mạng lưới thiết bị, hình 2.4.



Hình 2.5 BLE và mô hình Mesh

Dù trong bất kì mô hình nào, thì các Advertiser đều sẽ phát ra các gói tin nhằm mục đích quảng bá (Broadcast) thiết bị đang hoạt động. Có hai loại gói tin có thể được truyền đó là Advertising Data là gói tin quảng cáo cơ bản hoặc Scan Response Data là gói tin phản hồi kết nối với Scanner, mỗi gói có thể chứa tối đa 31 byte dữ liệu. Địa chỉ của thiết bị Advertiser cũng được thông tin trong hai loại gói tin trên. Các Advertiser liên tục phát ra các gói tin quảng cáo sau một khoảng thời gian gọi là Advertising interval, khoảng thời gian này có thể được thay đổi bởi người dùng, tuy nhiên vẫn có ràng buộc bởi khoảng thời gian nhỏ nhất (Min) và lớn nhất (Max).

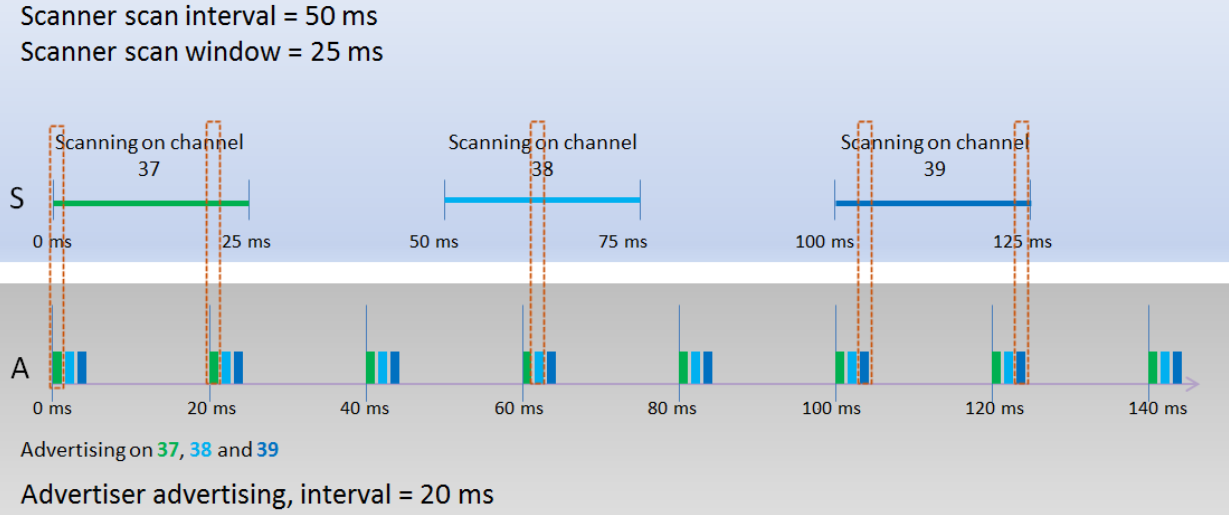


Hình 2.6 Advertising Data

Hình bên trên mô tả cách thức gửi gói quảng cáo và gói phản hồi máy quét. Có thể thấy cứ sau một khoảng thời gian Advertising Interval thì gói tin được phát đi. Trong mô hình có hướng kết nối, bên phát quảng cáo Advertiser sau khi phát gói tin cần giữ tầng radio để đợi nhận gói tin phản hồi, và sau đó nó sẽ gửi lại một gói tin phản hồi Scan Response Data. Có thể thấy, việc bật tầng Radio là rất mất điện năng, do đó trong một số ứng dụng nếu không mong đợi kết nối, thì có thể tắt Radio sau khi phát gói tin quảng cáo để tiết kiệm năng lượng.

Lưu ý rằng, tất cả các gói tin đều có chung một địa chỉ truy cập trực tuyến (0x8E89BED6) điều này thể hiện ở việc tất cả các máy quét đều có thể quét được các gói tin của tất cả các thiết bị mà không phải của riêng một thiết bị nào. Chuẩn BLE sử dụng 40 kênh Radio Frequency (RF) thuộc băng tần ISM 2.4 GHz (viết tắt của Industrial – Scientific – Medical) được đánh số từ 0 – 39, trong đó ba kênh dành riêng cho hoạt động quảng cáo là 37 (2402MHz), 38 (2426MHz) và 39 (2480MHz). Sở dĩ ba kênh này được chọn là vì chúng có tần số thuộc các kênh mà tín hiệu Wifi bận, do đó vấn đề nhiễu sóng được hạn chế.

Chuyển qua với Scanner, thông số Scan Interval chính là khoảng thời gian mà máy quét bắt đầu mở cửa sổ quét ScanWindow để lắng nghe gói tin quảng cáo. Theo dõi hình 2.6 bên dưới để quan sát một ví dụ về hoạt động của Scanner và Advertiser. Trong ví dụ này, Scanner lắng nghe gói tín lần lượt ở ba kênh là lặp lại



Hình 2.7 Advertising và Scanning

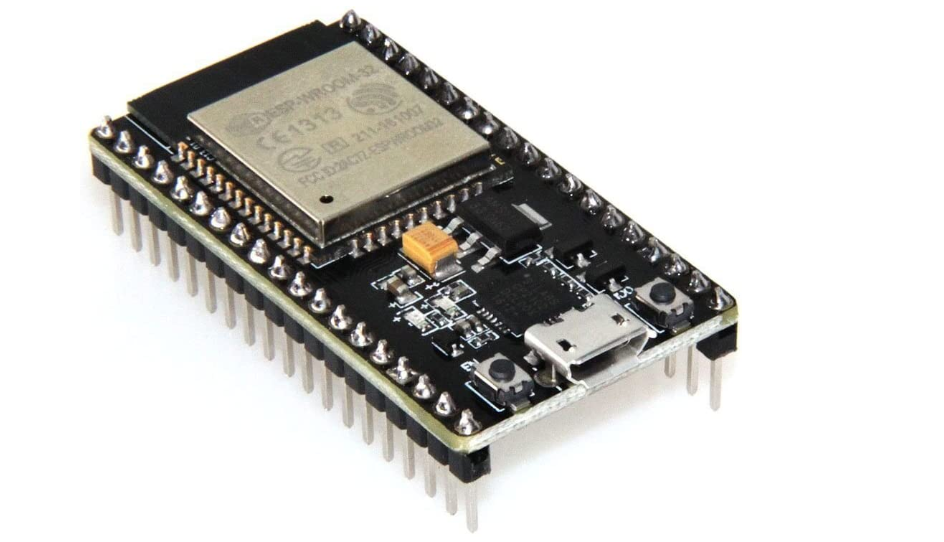
xoay vòng. Khi Scanner bắt được gói tin, nó sẽ đóng vai trò là người khởi tạo kết nối (nếu đây là mô hình có hướng kết nối), Advertiser hi vọng nhận được gói tin mong muốn kết nối ở cùng kênh phát ra gói tin quảng cáo. Có thể thấy hiệu suất của việc kết nối phù thuộc vào các thông số Interval và ScanWindow, với các thông số được thiết lập như trên hình, có thể thấy cùng một bản tin do một Advertiser phát ra, Scanner sau ba chu kỳ quét trên ba kênh đã năm lần bắt được gói tin này. Nếu chỉ cần tính số thiết bị quét được trên duy nhất một bản tín ở bất cứ kênh nào, vậy thì ta thấy số gói tin dư thừa trong trường hợp này là bốn, có thể gọi là Redundant Peers hay số đồng nghiệp dư thừa. Sự dư thừa này là không cần thiết trong hầu hết các ứng dụng không mang tính đáp ứng kết nối nhanh như BLE, do đó, có hai lựa chọn có thể giải quyết trong trường hợp này, thứ nhất là tăng khoảng thời gian Advertising Interval, thứ hai là giảm kích thước ScanWindow hoặc tăng khoảng thời gian ScanInterval… điều này làm giảm thời gian phát Radio, giúp tiết kiệm năng lượng hơn. Rõ ràng, sự cân nhắc giữa việc giảm tiêu thụ năng lượng và khả năng đáp ứng giao tiếp nhanh là việc hết sức quan trọng. Thuật toán hDiscovery dựa trên sự đánh đổi giữa số lượng thiết bị khám phá được và năng lượng tiêu thụ cho việc khám phá.

## PHẦN CỨNG VÀ KIẾN THỨC LIÊN QUAN

### ESP32 DevKit và các chế độ ngủ (Sleep modes)

* **ESP32 Development Kit**

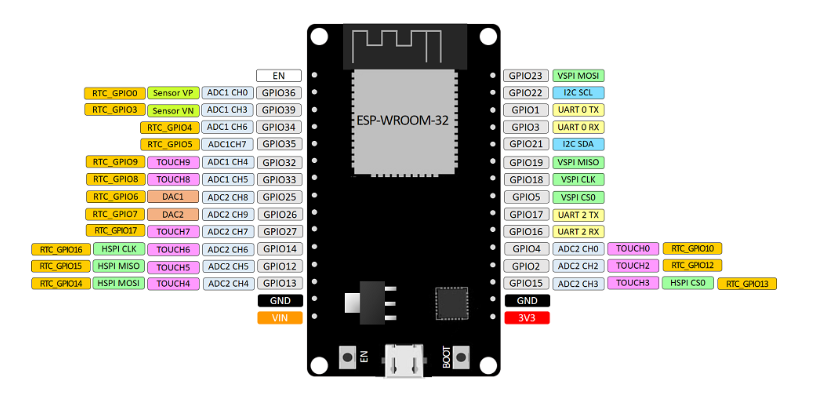
ESP32 là một dòng vi điều tích hợp trên một bảng mạch hay còn được gọi là System on Chip (SoC). Đây là dòng vi điều khiến mạch mẽ được tích hợp Wifi và Bluetooth cho phép ESP32 phù hợp với nhiều ứng dụng khác nhau. ESP32 được



Hình 2.8 ESP32 Development Kit

chế tạo và phát triển bởi Espressif Systems, một công ty Trung Quốc có trụ sở tại Thượng Hải, và được sản xuất bởi TSMC bằng cách sử dụng công nghệ 40 nm. ESP32 là sản phẩm kế thừa từ vi điều khiển ESP8266.

Là một vi điều khiển 32 bit với hiệu suất vợt trội và mức giá thấp, ESP32 dễ dàng đánh bại nhiều vi điều khiển SoC Wifi hiện nay, nó có thể được lập trình bởi nhiều ngôn ngữ khác nhau, điển hình C/C++, Python, NodeJs, Lua… Dưới đây là một số thông số đặc trưng của vi điều khiển này.



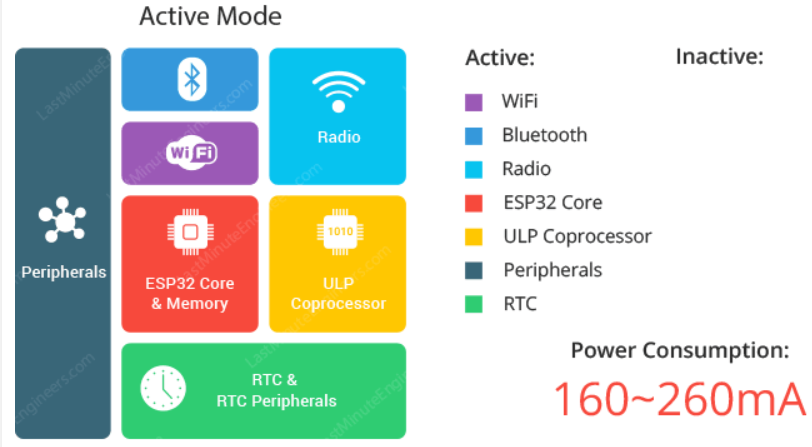
Hình 2.9 ESP32 pinout

* CPU: Bộ xử lý lõi kép (hoặc lõi đơn) Xtensa Dual-core LX6 32 bit mạnh mẽ, hỗ trợ hệ điều hành thời gian thực FreeRTOS với hai core ứng với Protocol cpu và Application cpu.
* Tốc độ xử lý: 160 MHz và có thể lên tới 240 MHz.
* Tốc độ xung nhịp cho chip Flash là 40 – 80 MHz (có thể lập trình).
* Bộ nhớ: ROM 448 Kbyte, External Flash 4 Mbyte, RAM 520 Kbyte (SRAM) trong đó có 16 Kbyte RTC RAM (Real Time Clock) dùng trong chế độ Deep sleep.
* Ngoại vi phong phú với: Wifi, Bluetooth, BLE, 18 kênh ADC, 2 bộ chuyển đổi DAC, 16 kênh PWM, 10 GPIO cảm biến điện dung, các module giao tiếp UART, I2C, SPI…
* **Các chế độ ngủ (Sleep modes)**

Tuy là một SoC WiFi / MCU với hiệu suất cao, giá cả tốt, Tuy nhiên, tùy thuộc vào trạng thái của nó, ESP32 có thể là một thiết bị tương đối ngốn điện. Khi dự án IoT được duy trì cấp nguồn bằng phích cắm, ta có xu hướng không quan tâm quá nhiều đến mức tiêu thụ điện năng. Nhưng đối với một ứng dụng cung cấp năng lượng cho dự án bằng pin, mỗi mA đều có giá trị. Điển hình trong chủ đề lần này sử dụng BLE tối ưu năng lượng, việc nghiên cứu về các chế độ ngủ của ESP32 để tối ưu các khoảng thời gian rảnh là điều tất yếu. Việc đưa vi xử lý vào chế độ ngủ là một chiến lược tuyệt vời để kéo dài tuổi thọ pin.

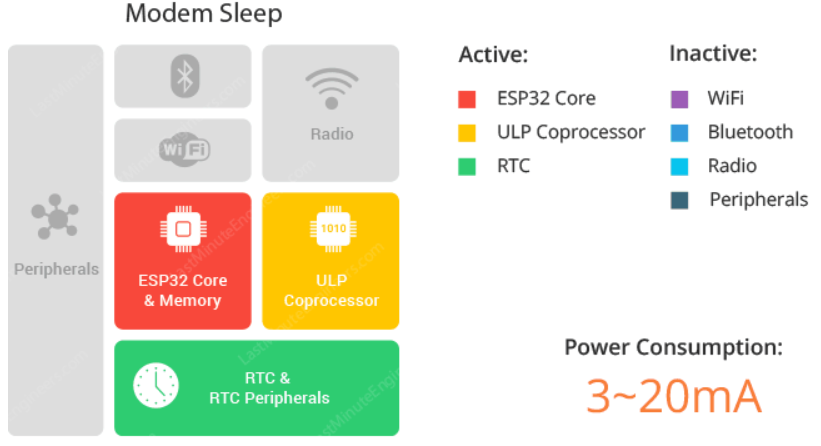
Ứng với mỗi chế độ, lượng điện năng tiêu thụ là khác nhau. ESP32 cũng cấp 5 cấu hình hoạt động khác nhau sau đây:

* Chế độ hoạt động (Active mode): là chế độ mặc định của chip, tất cả mọi thứ trên chip đều sẳn sàng trong trạng thái hoạt động. Ở chế độ này chip yêu cầu dòng điện từ 160 ~ 260mA. Đặt biệt nếu sử dụng song song hai chức năng Wifi và Bluetooth, mức điện năng có thể sẽ cao đột biến (có thể đạt tối đa 790mA). Tất nhiên đây là chế độ kém hiệu quả về năng lượng nhất, nếu muốn tiết kiệm năng lượng thì hấu như chế độ này được bỏ qua, hình 2.8 đề cập các khối chức năng bên trong ESP32, có thể thấy tất cả đều hoạt động bình thường.



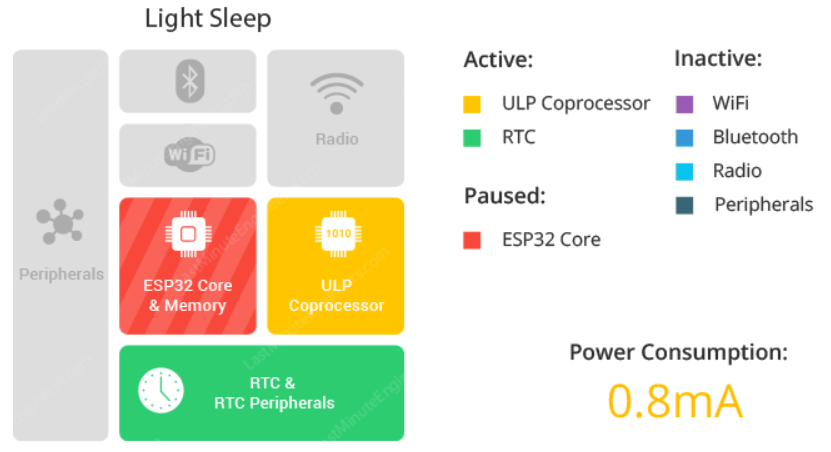
Hình 2.10 ESP32 Active mode

* Chế độ Modem Sleep: Trong chế độ ngủ của modem, mọi thứ hoạt động trong khi các ngoại vi kể cả WiFi, Bluetooth và radio bị tắt. CPU và các bộ đếm, timer đều hoạt động và có thể cấu hình tốc độ, Ở chế độ này, chip tiêu thụ khoảng 3mA ở tốc độ chậm và 20mA ở tốc độ cao. Để duy trì kết nối WiFi / Bluetooth, CPU, Wi-Fi, Bluetooth và radio sẽ được đánh thức theo những khoảng thời gian xác định trước. Hình 2.9 thể hiện chế độ ngủ Modem.



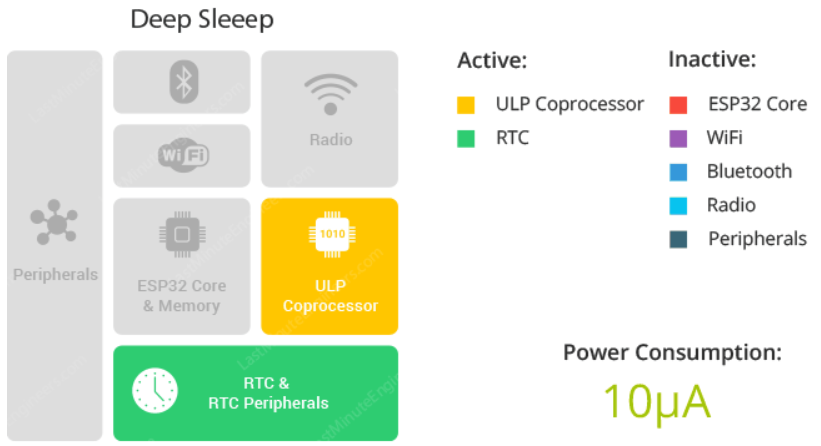
Hình 2.11 ESP32 Modem Sleep mode

* Chế độ ngủ nhẹ (Light Sleep): Chế độ làm việc của chế độ ngủ nhẹ gần như tương tự như chế độ ngủ của modem. Sự khác biệt ở đây là, trong chế độ ngủ nhẹ, hầu hết RAM và CPU đều được điều chỉnh xung nhịp, CPU bị tạm dừng bằng cách tắt nguồn xung đồng hồ của nó, trong khi RTC và ULP-coprocessor vẫn hoạt động. Điều này dẫn đến mức tiêu thụ điện năng ít hơn so với ở chế độ ngủ modem, khoảng 0,8mA. Trước khi vào chế độ ngủ, trạng thái hiện tại được bảo toàn, sau khi rời khỏi trạng thái ngủ, CPU được cấp xung, hệ thống sẽ hoạt động tiếp với trạng thái trước đó.



Hình 2.12 ESP32 Light Sleep mode

* Chế độ ngủ sâu (Deep Sleep): Ở chế độ ngủ sâu, CPU, hầu hết RAM và tất cả các thiết bị ngoại vi kỹ thuật số đều bị tắt nguồn. Các bộ phận của chip vẫn được hoạt động là: RTC controller, RTC peripherals (bao gồm cả bộ đồng xử lý ULP) và bộ nhớ RTC (chậm và nhanh). Vì CPU bị tắt, nên trạng thái hoạt động hiện tại bị mất đi, các biến dữ liệu cần phải được khai báo trong vùng nhớ RTC. Khi CPU được đánh thức, nó sẽ tải lại các dữ liệu trong vùng nhớ RTC và hoạt động lại từ trạng thái đầu tiên. Ở trạng thái deep sleep, dòng điện tiêu thụ là 10µA.



Hình 2.13 ESP32 Deep Sleep mode

* Chế độ ngủ đông (Hibernation Sleep): Ở chế độ ngủ đông, mọi thứ được tắt, kể cả bộ đồng xử lý ULP, chỉ có bộ đếm RTC được hoạt động để tính toán thời gian đánh thức chip, dòng điện tiêu thụ là 2.5µA.

Mỗi chế độ có các đặc điểm hoạt động khác nhau mà mức tiêu thụ điện năng khác nhau, tuy nhiên ở chủ đề này, chế độ Light Sleep và Deep Sleep được tận dụng.

* **Lập trình cho ESP32 với Arduino IDE**

Arduino IDE là một phần mềm mã nguồn mở chủ yếu được sử dụng để viết, biên dịch mã và nạp vào các module Arduino hoặc được hỗ trợ. Phần mềm mang lại sự đơn giản cho quá trình cài đặt và phát triển một firmware cho một ứng dụng IoT. Đây là một phần mềm chính thức được giới thiệu bởi Arduino.cc, trang web chính thức của nhà phát triển.

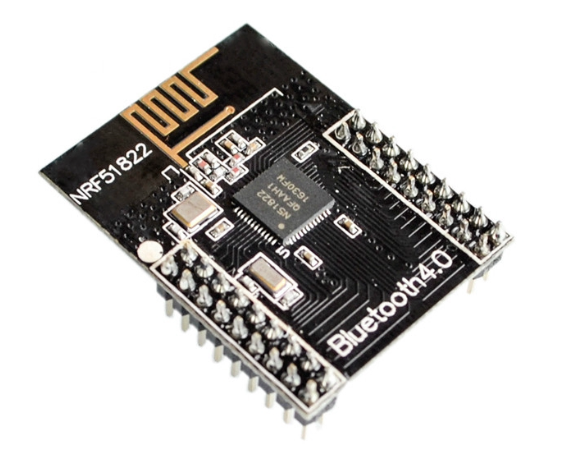
Arduino IDE cung cấp bộ thư viện hỗ trợ lập trình ESP32 bằng ngôn ngữ C/C++, với chuẩn nạp USB giúp việc nạp và chỉnh sửa firmware đơn giản hơn.

### Module BLE nRF51822 và phần mềm KeilC

* **Module BLE nRF51822**

Là một sản phẩm của Nordic Semiconductor, nRF51822 là một SoC đa năng, công suất cực thấp, thiết bị lý tưởng cho các ứng dụng thuần Bluetooth Low Energy hoạt động với 2,4 GHz. Nó được xây dựng trên nền CPU 32-bit ARM Cortex -M0 với 256/128 KB flash và 32/16 KB RAM. Bộ phận Radio 2,4 GHz linh hoạt hỗ trợ Bluetooth Low Energy và các giao thức độc quyền 2,4 GHz, chẳng hạn như Gazell.

nRF51822 kết hợp nhiều lựa chọn thiết bị ngoại vi tương tự và số có thể tương tác trực tiếp mà không cần sự can thiệp của CPU thông qua hệ thống kết nối ngoại vi có thể lập trình - Programmable Peripheral Interconnect (PPI). Các GPIO linh hoạt cho phép kết nối các giao diện kỹ thuật số như SPI Master / slave, TWI Master và UART với bất kỳ GPIO nào trong số 31 GPIO trên thiết bị. Hình 2.13 thể hiện hình dạng của mudule.



Hình 2.14 Module BLE nRF51822

Nordic cung cấp hỗ trợ BLE Stack được thiết lập sẳn dạng file Hex gọi là SoftDevice và một bộ Software Development Kit (SDK) phù hợp với từng module tương ứng. Đối với module nRF51822, phiên bản SoftDevice sử dụng ở đây là S130 hỗ trợ BLE 4.2 và nRF5 SDK.

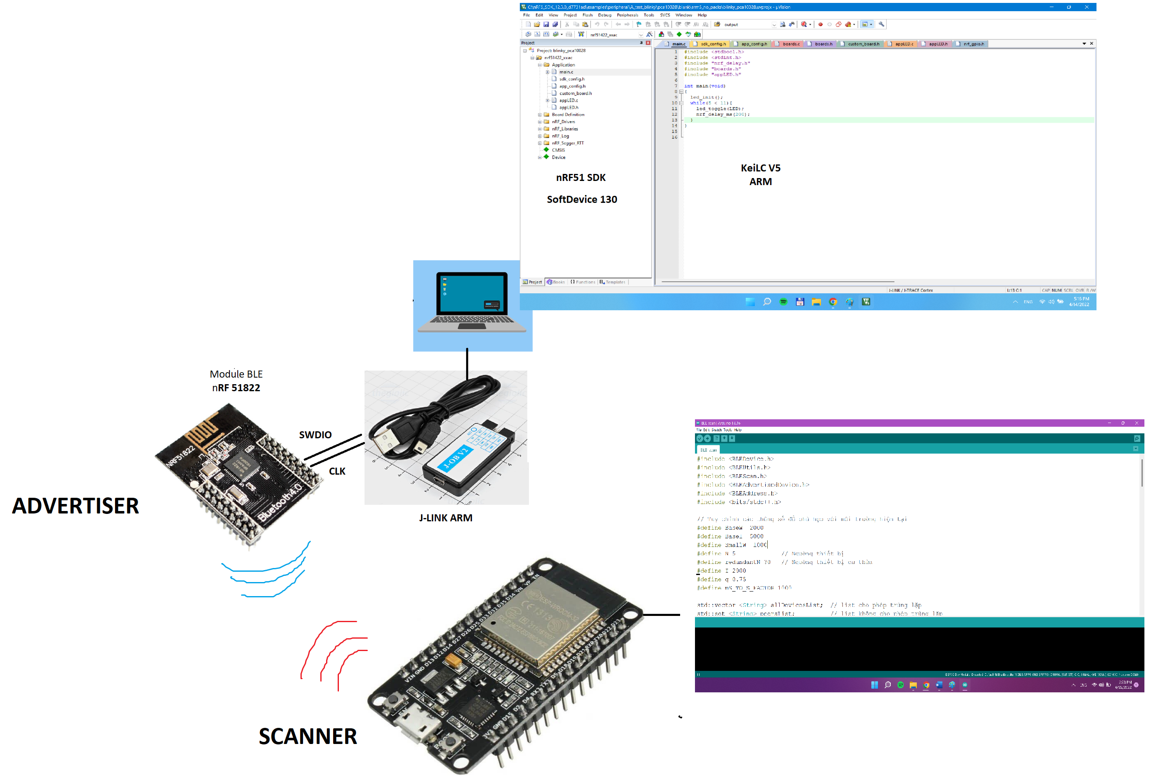
* **KeilC IDE và mạch nạp J-Link**

KeilC hay bản mới nhất là KeilC V5, là một trong những IDE tốt nhất hiện nay, được các lập trình viên sử dụng để phát triển firmware cho các vi điều khiển có lõi ARM. Chúng đóng vai trò như là trình soạn thảo ngôn ngữ (chủ yếu là C và Assembly), cũng như là trình biên dịch, hỗ trợ debug-phát hiện lỗi và qua đó ta có thể sửa các lỗi được phát hiện. Ngoài ra nó cũng hỗ trợ biên dịch firmware thành file hex qua đó nạp vào các dòng vi xử lý.

Mạch nạp mà KeilC hỗ trợ thông thường là J-Link hoặc J-Link OB, một mạch nạp cũng phỗ biến không kém dựa trên chuẩn J-TAG. Mạch cho phép nạp/debug các chip lõi ARM và một số dòng chip khác. J-Link là một lựa chọn tuyệt vời với giá cả thấp, phù hợp cho học tập và nghiên cứu.

# thiết kế và triển khai thuật toán

## SƠ ĐỒ TỔNG QUÁT



Hình 3.1 Sơ đồ hệ thống

Như đã trình bày ở Chương 1, hệ thống gồm hai phần chính là các Advertiser hoạt động theo cơ chế beacon và một Scanner chạy thuật toán hDiscovery, hình 3.1.

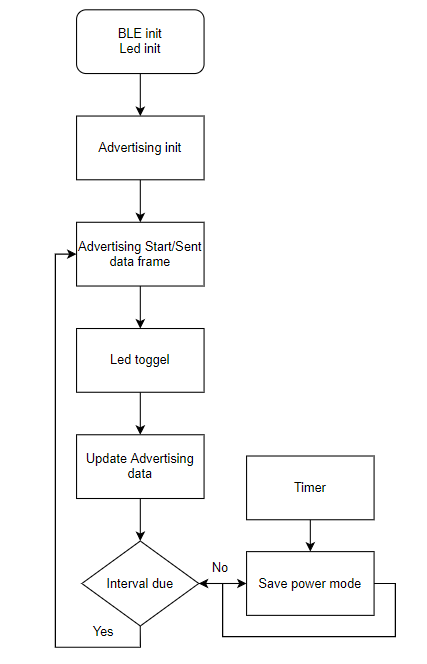
Đối với Advertiser, module BLE được sử dụng là nRF51822 của Nordic. Về qui trình xây dựng, đầu tiên mã nguồn (Firmware) được viết và chỉnh sửa bằng phần mềm KeiLC ( ở đây sử dụng KeiLC V5). Mã nguồn gồm hai phần chính là phần SoftDevice do Nordic cung cấp và phần code Application nơi lập trình ứng dụng (đã được trình bày chi tiết ở Chương 2 cơ sở lý thuyết). Mã nguồn được nạp vào module nRF51822 thông qua mạch nạp J-Link chuẩn J-TAG với hai chân sử dụng là CLK và SWDIO. Lưu ý, khi nạp mã nguồn, cần phân vùng và nạp mã hex của SoftDevice trước và chỉ cần một lần, sau đó mới biên dịch mã code C Application thành mã hex và nạp vào module ở vùng RAM tiếp theo.

Đối với Scanner, phần cứng sử dụng là ESP32 lập trình với Arduino IDE. Thông qua ngoại vi Serial Monitor của Arduino IDE, chúng ta có thể quan sát được những thông báo được in ra khi Scanner hoạt động.

## PHÁT TRIỂN THIẾT BỊ BLE BEACON VỚI NRF51822

### Lưu đồ thuật toán

Hình 3.2 trình bày lưu đồ thuật toán tổng quan của hệ thống beacon, trong đó tín hiệu điều khiển Led nhấp nháy chỉ để kiểm tra trạng thái hoạt động của beacon.



Hình 3.2 Lưu đồ thuật toán tổng quát của Beacon

Sau khi thiết lập các cài đặt để về việc sử dụng BLE và Led, khâu chuẩn bị frame cho gói tin Advertising được triển khai, trong khâu này dữ liệu trước khi gửi đi được mã hóa và chuẩn bị sẳn trong buffer chờ gửi. Tiếp đến, gói tin quảng cáo được gửi đi, nếu gửi đi thành công thì Led sẽ nhấp nháy. Tiếp theo ta sẽ thay đổi dữ liệu cần gửi đi (nếu muốn) thông qua bước update Advertising data. Sau bước cập nhật, hệ thống sẽ được đưa vào trạng thái tiết kiệm năng lượng, chờ tới khi được đánh thức để tiếp tục gửi. Quá trình trên được lặp lại.

### Firmware

Chúng ta hãy xem xét từng phần khác nhau để hiểu rõ hơn về cách thực hiện beacon, ở đây chỉ xem xét những phần trọng tâm của mã, firmware đầy đủ được gửi kèm ở link phần Phụ Lục.

Bắt đầu với **main.c**

Đặt Khoảng thời gian Quảng cáo (Advertising Interval), ở đây là 100 ms:

#define NON\_CONNECTABLE\_ADV\_INTERVAL MSEC\_TO\_UNITS (100, UNIT\_0\_625\_MS)

Đặt tổng độ dài dữ liệu quảng cáo (bao gồm cả giá trị lenght):

#define APP\_BEACON\_INFO\_LENGTH 0x01

Đặt giá trị loại thiết bị (loại thiết bị beacon = 0x02):

#define APP\_DEVICE\_TYPE 0x02

Giá trị RSSI (ở 1 mét và tính bằng dBm):

#define APP\_MEASURED\_RSSI 0xC3

Định danh công ty (ở đây để ngẫu nhiên là 0xFF vì không dùng tới):

#define APP\_COMPANY\_IDENTIFIER 0xFF

Thông tin cần gửi (Beacon), ở đây chỉ sử dụng một byte với mục đích cho nó tăng dần để kiểm tra việc update dữ liệu gửi:

static uint8\_t m\_beacon\_info [APP\_BEACON\_INFO\_LENGTH] = { 0x00 };

Hàm Advertising\_init (), chuẩn bị các trường dữ liệu cho gói tin cần gửi, mã hóa dữ liệu và để ở buffer đợi gửi đi:

static void advertising\_init(void)

{

uint32\_t err\_code;

ble\_advdata\_t advdata;

uint8\_t flags = BLE\_GAP\_ADV\_FLAG\_BR\_EDR\_NOT\_SUPPORTED;

ble\_advdata\_manuf\_data\_t manuf\_specific\_data;

manuf\_specific\_data.company\_identifier = 0xFF;

ble\_gap\_conn\_sec\_mode\_t sec\_mode;

BLE\_GAP\_CONN\_SEC\_MODE\_SET\_OPEN(&sec\_mode);

err\_code = sd\_ble\_gap\_device\_name\_set(&sec\_mode, (const uint8\_t \*) "HaoTr", 5);

manuf\_specific\_data.data.p\_data = (uint8\_t \*) m\_beacon\_info;

manuf\_specific\_data.data.size = 1;

// Build and set advertising data.

memset(&advdata, 0, sizeof(advdata));

advdata.name\_type = BLE\_ADVDATA\_FULL\_NAME;

advdata.flags = flags;

advdata.p\_manuf\_specific\_data = &manuf\_specific\_data;

err\_code = ble\_advdata\_set(&advdata, NULL);

APP\_ERROR\_CHECK(err\_code);

// Initialize advertising parameters (used when starting advertising).

memset(&m\_adv\_params, 0, sizeof(m\_adv\_params));

m\_adv\_params.type = BLE\_GAP\_ADV\_TYPE\_ADV\_NONCONN\_IND;

m\_adv\_params.p\_peer\_addr = NULL; // Undirected advertisement.

m\_adv\_params.fp = BLE\_GAP\_ADV\_FP\_ANY;

m\_adv\_params.interval = NON\_CONNECTABLE\_ADV\_INTERVAL;

m\_adv\_params.timeout = APP\_CFG\_NON\_CONN\_ADV\_TIMEOUT;

}

Hàm Advertising\_start (), bắt đầu phát gói tin:

static void Advertising\_start (void)

{

uint32\_t err\_code;

err\_code = sd\_ble\_gap\_adv\_start(&m\_adv\_params);

APP\_ERROR\_CHECK(err\_code);

err\_code = bsp\_indication\_set(BSP\_INDICATE\_ADVERTISING);

APP\_ERROR\_CHECK(err\_code);

}

Hàm main(), gói các hàm đã thiết lập và chạy như lưu đồ thuật toán đã trình bày :

int main (void)

{

led\_init();

uint32\_t err\_code;

// Initialize.

err\_code = NRF\_LOG\_INIT(NULL);

APP\_ERROR\_CHECK(err\_code);

APP\_TIMER\_INIT(APP\_TIMER\_PRESCALER, APP\_TIMER\_OP\_QUEUE\_SIZE, false);

err\_code = bsp\_init(BSP\_INIT\_LED, APP\_TIMER\_TICKS(100, APP\_TIMER\_PRESCALER), NULL);

APP\_ERROR\_CHECK(err\_code);

ble\_stack\_init();

advertising\_init();

// Start execution.

NRF\_LOG\_INFO("BLE Beacon started\r\n");

advertising\_start();

// Enter main loop.

for (;; )

{

led\_toggle(7);

temp += 1;

advertising\_update();

nrf\_delay\_ms(2000);

if (NRF\_LOG\_PROCESS() == false)

{

power\_manage();

}

}

}

## PHÁT TRIỂN THIẾT BỊ BLE SCANNER VỚI ESP32

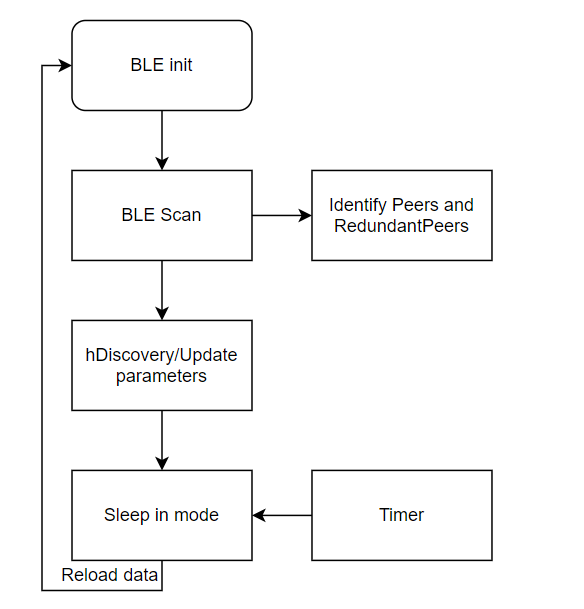
### Thuật toán hDiscovery

|  |  |  |
| --- | --- | --- |
| **Thuật toán 1** | | Giao thức khám phá thiết bị linh động hDiscovery |
|  | 1: ScanWindow = *BaseW* , ScanInterval = *BaseI*  2: **while** (TRUE) **do**  3: StartDiscovery (ScanWindow)  4: identify *peers* and *redundantpeers*  5: **if** (*redundantpeers* > *redundantN*) **then**  6: ScanWindow = *SmallW*  7: ScanInterval = *q × BaseI*  8: SleepMode = *DeepSleep*  9: **else**  10: **if** (*peers > N*) **then**  11: ScanWindow = *BaseW*  12: SleepMode = LightSleep  13: **else**  14: ScanWindow = *SmallW*  15: SleepMode = *DeepSleep*  16: **end if**  17: **if** (*peers == 0 and LastPeers == 0*) **then**  18: ScanInterval = *q × BaseI*  19: **else if** (*peers <> 0 and LastPeers == 0*) **then**  20: ScanInterval = BaseI  21: **else if** (*peers > LastPeers*) **then**  22: ScanInterval − = *I*  23: **else if** (*peers < LastPeers*) **then**  24: ScanInterval + = *I*  25: **end if**  26: **end if**  27: *LastPeers = peers*  28: *sleep* (ScanInterval - ScanWindow)  29: **end while** | |

Thuật toán hDiscovery được xây dựng dựa trên ý tưởng của thuật toán sDiscovery [2] của nhóm nghiên cứu trường đại học Khoa học và Công nghệ Quốc gia Đài Loan. Qua thuật toán, Scanner có thể thay đổi các thông số ScanWindow và ScanInterval để đạt được hiểu quả năng lượng. Với sự đánh đổi (trade-off) giửa số lượng thiết bị quét được và sự hiệu quả năng lượng, thuật toán còn đảm bảo được sự duy trì số lượng thiết bị quét được cần thiết cho việc đáp ứng yêu cầu ứng dụng. Thuật toán tận dụng hai chỉ số là số lượng thiết bị (Peers) và số lượng thiết bị dư thừa (RedundantPeers) để cân nhắc thay đổi các thông số. Thuật toán 1 trình bày hDiscovery.

### ESP32 Scanner với hDiscovery

* **Lưu đồ triển khai thuật toán**



Hình 3.3 Lưu đồ thuật toán tổng quát của Scanner

Quan sát Hình 3.3 Lưu đồ thuật toán tổng quát của Scanner, ý tưởng cơ bản là trước hết cài đặt các thông số ScanWindow và ScanInterval cũng như các cài đặt khác cho việc khởi động BLE Scanner. Tiếp theo, máy quét được khởi động (BLE Scann), trong quá trình quét, địa chỉ MAC của các thiết bị được ghi lại để xác định số lượng thiết bị (Peers) và số lượng dư thừa (RedundantPeers), từ các thông số này, thuật toán hDiscovery được chạy để canh chỉnh ScanWindow và ScanInterval cũng như chọn chế độ ngủ phù hợp. Sau đó hệ thống được đưa vào trạng ngủ tương ứng và chờ đánh thức bằng timer.

* **Firmware**

Đặt các thông số cấu hình cho thuật toán hDiscovery:

#define BaseW 2000 //ScanWindow cơ sở

#define BaseI 5000 //ScanInterval cơ sở

#define SmallW 1000 //ScanWindow thu nhỏ

#define N 5 //Ngưỡng thiết bị

#define redundantN 30 //Ngưỡng thiết bị dư thừa

#define I 2000 //Khoảng thời gian canh chỉnh ScanInterval

#define q 0.75 // Trị số canh chỉnh ScanInterval

Đặt hệ số chuyển từ giây sang mili giấy (dùng cho đặt thời gian sleep):

#define mS\_TO\_S\_FACTOR 1000

Khởi tạo hai mảng co giãn, trong đó mảng allDevicesList kiểu **vector** cho phép lưu địa chỉ tất cả các thiết bị (gói tin Advertising) quét được, trong khi mảng peersList là mảng kiểu **set** chỉ cho phép lưu các địa chỉ không trùng lặp:

std::vector <String> allDevicesList; // list cho phép trùng lặp

std::set <String> peersList; // list không cho phép trùng lặp

Khởi tạo một số biến để sử dụng trong quá trình tính toán, triển khai thuật toán, các biến được khải báo ở vùng nhớ **RTC** để được tải lại khi hệ thống rời khỏi chế độ Deep Sleep:

RTC\_DATA\_ATTR BLEScan\* pBLEScan;

RTC\_DATA\_ATTR int lastPeers = 0;

RTC\_DATA\_ATTR int scanWindow = BaseW;

RTC\_DATA\_ATTR int scanInterval = BaseI;

RTC\_DATA\_ATTR int timeToSleep;

RTC\_DATA\_ATTR int sleepMode = 0; // 0: deepSleep ; 1: lightSleep

Class CallBack, được cung cấp để người lập trình có thể thực hiện những yêu cầu khi một gói tin được bắt được. Ở đây, khi một gói tin được phát hiện, ta sẽ trích xuất địa chỉ trong frame data để lưu vào hai mảng đã khai báo ở trên :

// Class callBack Thêm địa chỉ thiết bị quét được vào danh sách

class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {

void onResult(BLEAdvertisedDevice advertisedDevice) {

Serial.printf("Advertised Device: %s \n", advertisedDevice.toString().c\_str());

allDevicesList.push\_back(advertisedDevice.getAddress().toString().c\_str());

peersList.insert(advertisedDevice.getAddress().toString().c\_str());

}

};

Hàm Scan(), thực hiện khởi tạo các cấu hình hoạt động cho BLE, truyền vào các thông số ScanWindow và ScanInterval và bắt đầu quét:

void Scan(){

Serial.println("Scanning...");

BLEDevice::init("Hao");

pBLEScan = BLEDevice::getScan(); //create new scan

pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());

pBLEScan->setActiveScan(true);

pBLEScan->setWindow(scanWindow/3);

pBLEScan->setInterval(scanWindow/3);

// Đối số truyền vào kiểu integer dạng đơn vị giây

BLEScanResults foundDevices = pBLEScan->start(scanWindow/1000, false);

Serial.println("Scan done!");

pBLEScan->clearResults();

}

Hàm hDiscovery(), chạy thuật toán hDiscovery để thay đổi các thông số, in ra các thông số thay đổi để dễ dàng kiểm tra:

void hDiscovery(){

int peers = peersList.size();

int redundantPeers = allDevicesList.size()- peersList.size();

if(redundantPeers > redundantN){

scanWindow = SmallW;

scanInterval = q \* BaseI;

sleepMode = 0;

}

else {

if(peers > N){

scanWindow = BaseW;

sleepMode = 1;

}

else{

scanWindow = SmallW;

sleepMode = 0;

}

if(peers == 0 && lastPeers == 0)

scanInterval = q \* BaseI;

else if (peers != 0 && lastPeers == 0)

scanInterval = BaseI;

else if (peers > lastPeers)

scanInterval -= I;

else if (peers < lastPeers)

scanInterval += I;

}

lastPeers = peers;

Serial.println("Peers: " + String(peers));

Serial.println("redundantPeers: " + String(redundantPeers));

Serial.println("ScanWindow: " + String(scanWindow));

Serial.println("ScanInterval: " + String(scanInterval));

}

Hàm thực hiện chức năng ngủ theo mode đã chọn từ thuật toán hDiscovery, 0 là deep sleep, 1 là light sleep:

void mulSleepMode(){

timeToSleep = scanInterval - scanWindow;

esp\_sleep\_enable\_timer\_wakeup(timeToSleep \* mS\_TO\_S\_FACTOR);

Serial.println("Setup ESP32 to sleep for every " + String(timeToSleep) + " Seconds");

if (sleepMode == 0){

Serial.println("Going to deep sleep now");

Serial.flush();

esp\_deep\_sleep\_start();

}

else{

Serial.println("Going to light sleep now");

Serial.flush();

esp\_light\_sleep\_start();

}

}

Hàm setup() chạy khi chip khởi động, ở đây chỉ cần dùng cho việc khởi động và chọn tốc độ truyền Serial là 115200 baudrate:

void setup() {

Serial.begin(115200);

}

Hàm loop(), hàm xử lý chính, lặp lại, goi các hàm đã kể ở trên theo thứ tự để chạy thuật toán. Thứ tự là Scan – hDiscovery – Sleep – Scan….:

void loop() {

Scan();

hDiscovery();

mulSleepMode();

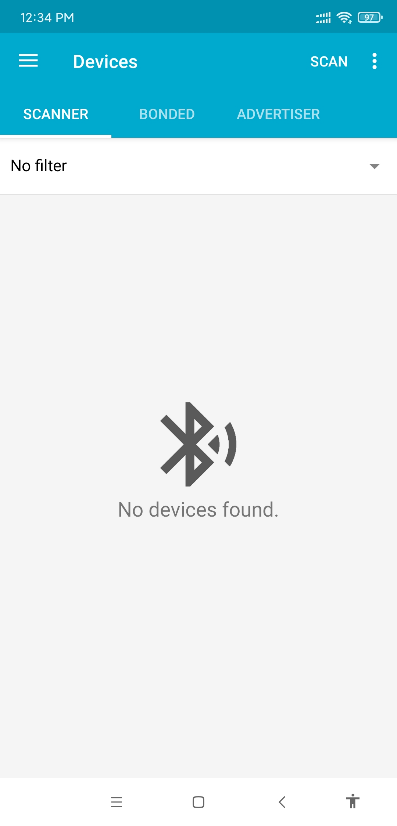
}

# kết quả

## HOẠT ĐỘNG CỦA BEACON

### Phần mềm nRF Connect

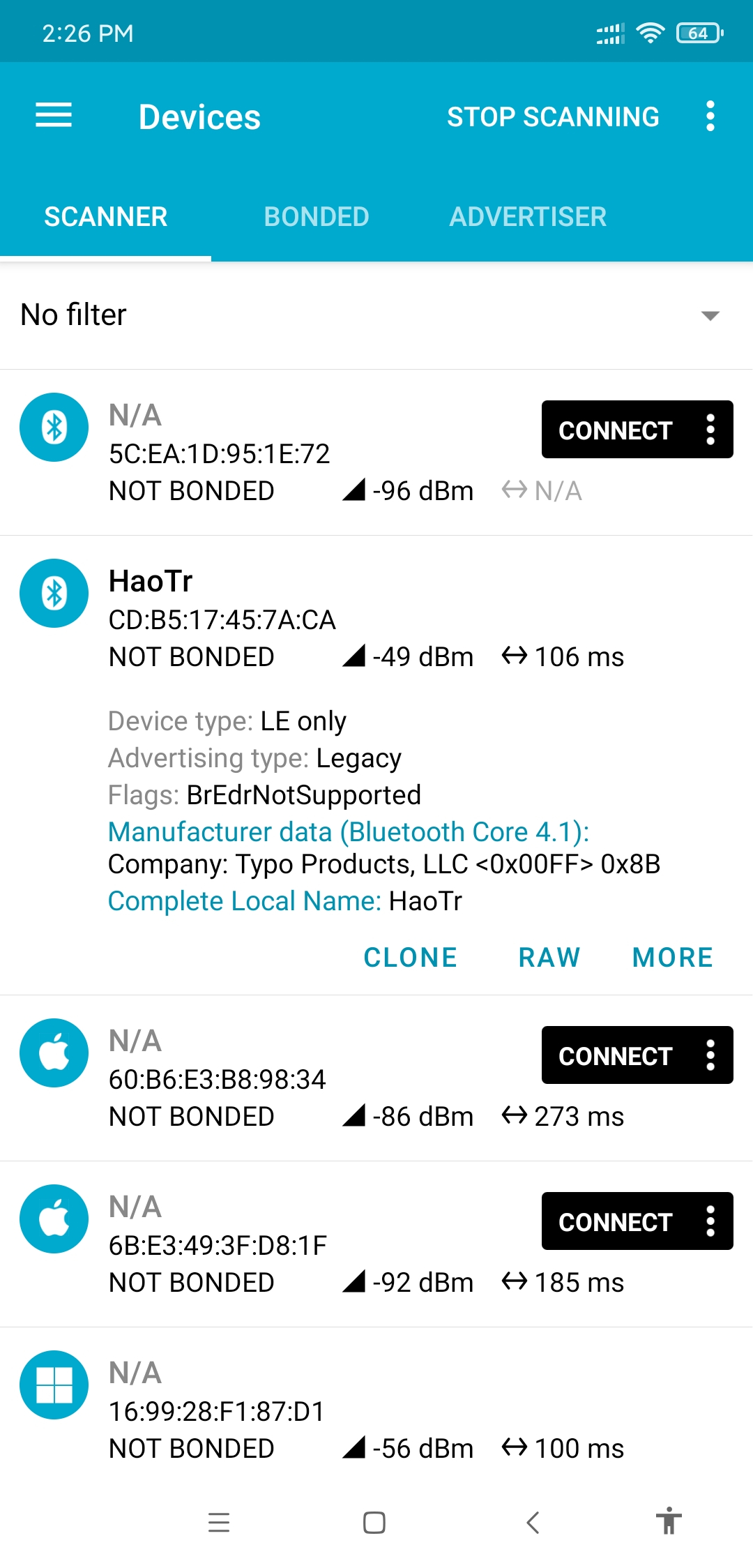
Phần mềm nRF Connect là một khung công cụ đa nền tảng để hỗ trợ phát triển trên các thiết bị nRF. Nó chứa nhiều ứng dụng để kiểm tra, giám sát, đo lường, tối ưu hóa và lập trình các ứng dụng **BLE**.



Hình 4.1 Phần mềm nRF Connect cho Android

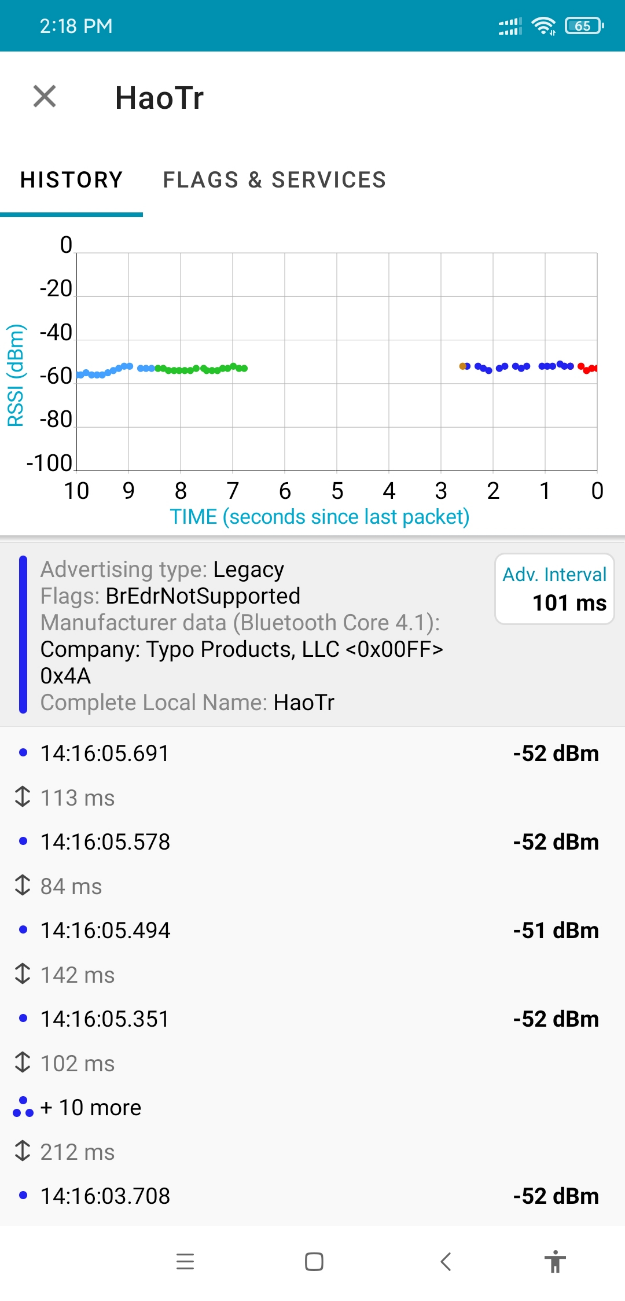
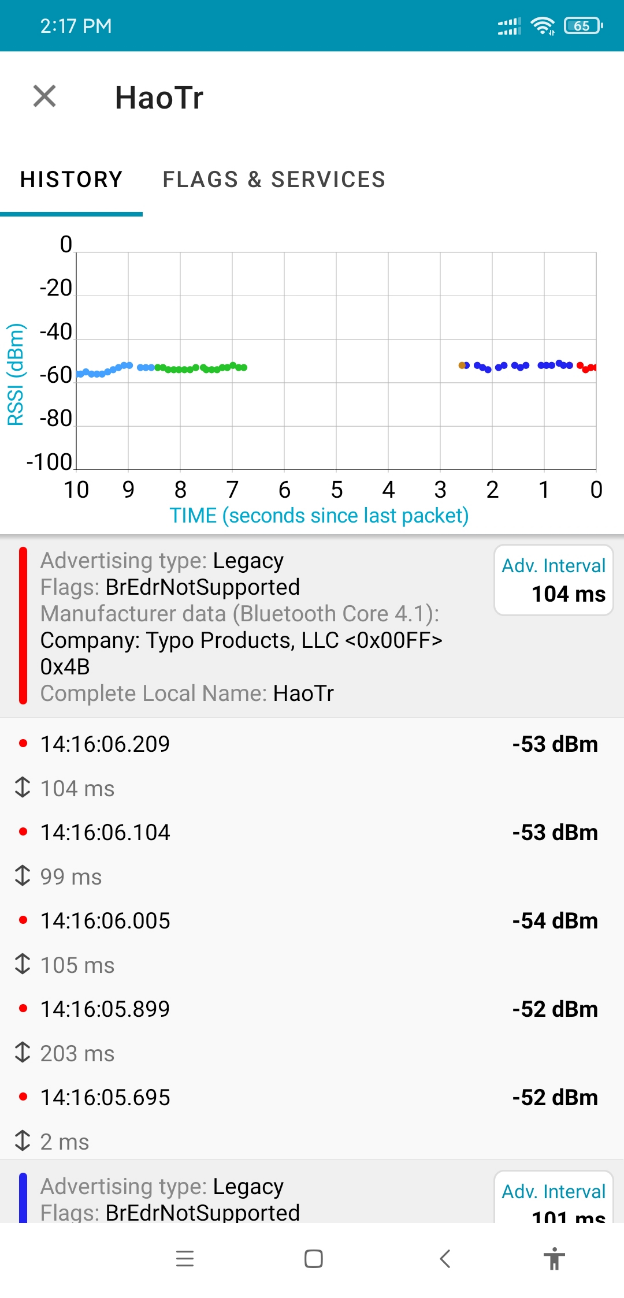
### Phần tích gói tin Beacon

Dùng phần mềm nRF Connect để bắt gói tin Beacon từ module nRF51822, quan sát và so sánh các trường dữ liệu đã định nghĩa ở phần firmware. Hình 4.2 cho thấy giao diện phần mềm lúc quét được gói tin.



Hình 4.2 Quét gói tin bằng phần mềm nRF Connect

Giao diện phần mềm cho thấy rất nhiều thiết bị BLE được phát hiện, trong đó có thiết bị được đề cập tên “HaoTr”. Vì là gói tin dạng Beacon nên có thể thấy nó không có thuộc tính kết nối. Phần mềm cũng hiển thị một số thông tin quan trọng khác trong gói tin, hãy phân tích kỹ hơn ở Hình 4.3.



Hình 4.3 Phân tích gói tin Advertising với nRF Connect

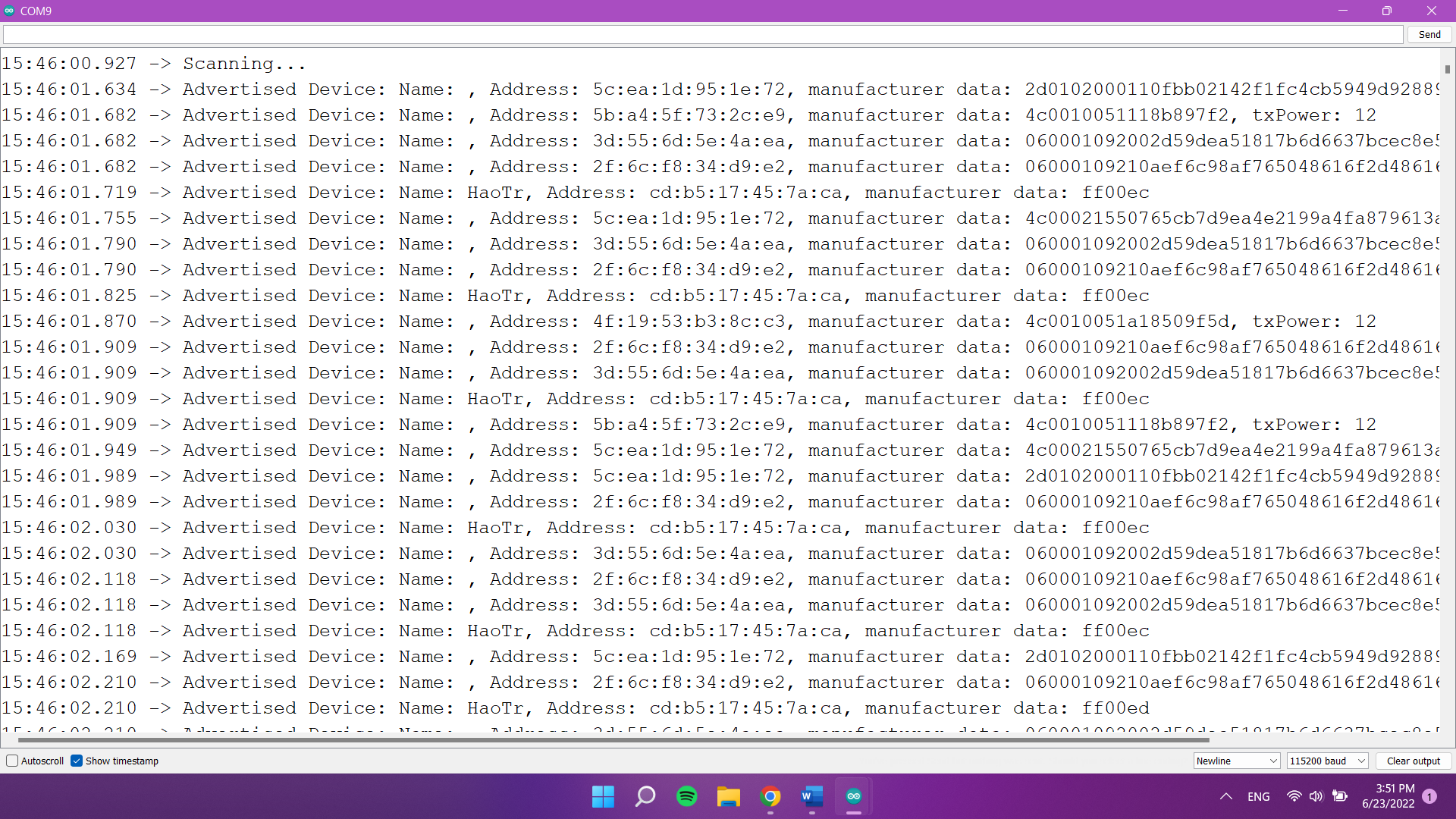
Đối với Hình 4.2 bên trái, thể hiện gói tin mới nhất được nhận bởi thiết bị “HaoTr”, một số thông số đáng chú ý của gói tin:

* Advertising type: loại gói tin là Legacy, dễ hiểu thì đây là một gói tin cơ bản phục vụ cho các mục đích chung của BLE.
* Company: 0x00FF đúng với cài đặt.
* Dữ liệu mang theo: 0x4B, dữ liệu này được cấu hình dưới dạng tăng dần 1 đơn vị sau mỗi gói tin, ta sẽ quan sát hình 4.3 bên phải thể hiện gói tin trước đấy, và trường dữ liệu là 0x4A, vậy nó hoạt động đúng yêu cầu.
* Local Name: tên thiết bị là “HaoTr”.
* Advertising Interval: 104 ms, theo cấu hình là 100 ms, tuy nhiêu sẽ có sự sai lệch nhỏ vì hệ thống cần thực hiện những câu lệnh ngoài lề, và sự sai xót trong tính toán, trễ truyền…
* RSSI: chỉ số RSSI tăng giảm theo khoảng cách của thiết bị phát với thiết bị nhận (Điện thoại chạy app).

Beacon hoạt động đúng với yêu cầu đặt ra.

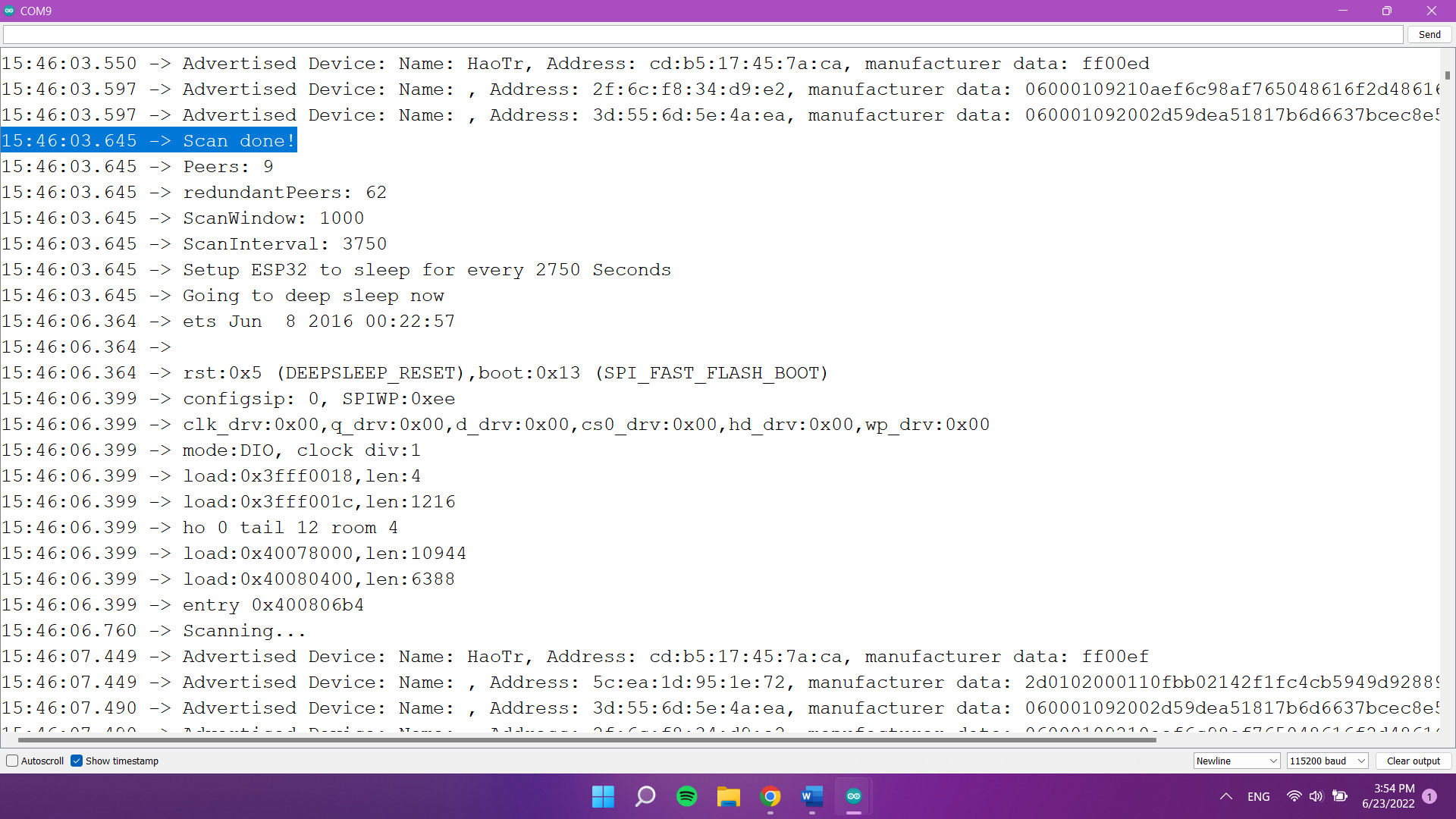
## HOẠT ĐỘNG CỦA SCANNER

Khởi động ESP32 Scanner chạy thuật toán hDiscovery, mở Serial Monitor được cung cấp bởi Arduino IDE, quan sát kết quả thu được ở Hình 4.4.



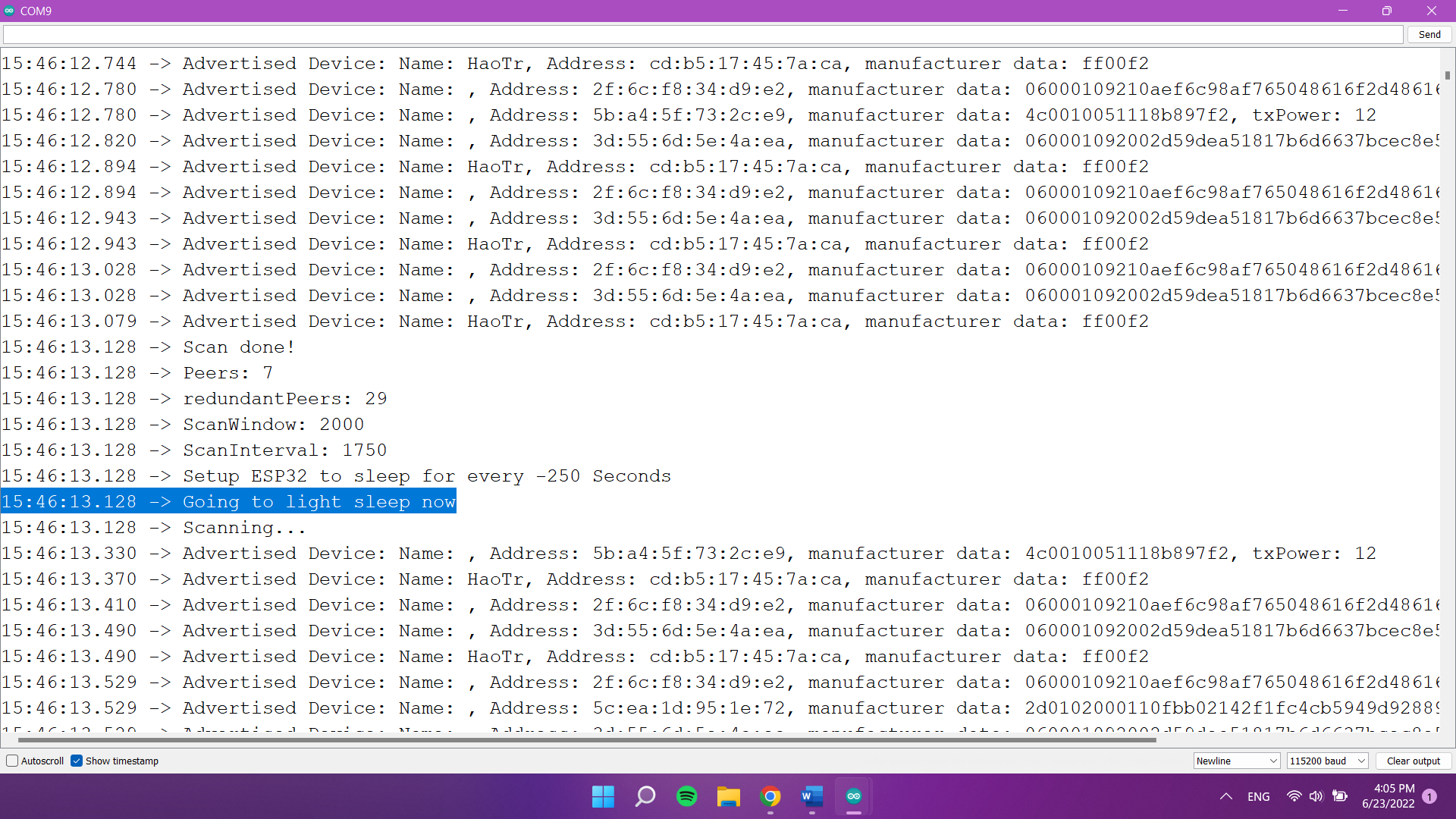
Hình 4.4 Scanner ESP32

Các thiết bị quét được được in ra thông tin, trong đó có thiết bị Beacon “HaoTr” kèm dữ liệu. Có thể thấy số lượng dư thừa rất lớn vì Advertising Interval quá nhỏ. Khi quá trình quét hoàn thành với chỉ số ScanWindow, số lượng thiết bị được in ra màn hình, Hình 4.5. Thông số Peers và RedundantPeers quá cao, thuật toán hDiscovery cập nhật lại thông số ScanWindow là 1000 ( Lúc đầu 2000), ScanInterval giảm xuống 3750 ms (Lúc đầu 4000). Sau đó hệ thống rơi vào trạng thái ngủ được chọn là Deep Sleep với khoảng thời gian ScanInterval – ScanWindow là 2750 ms.



Hình 4.5 Peers and RedundantPeers

Sau khi được đánh thức bởi timer từ Deep Sleep, CPU phải load lại dữ liệu tại các vùng nhớ được thể hiện trên hình 4.5 và tiếp tục quét cho chu kì tiếp theo.



Hình 4.6 Chế độ Light Sleep

Với ngưỡng dư thừa cài đặt cho việc kiểm thử là RedundantPeersN là 30, với trường hợp ở hình 4.6, số lượng thiết bị dư thừa là 29 nhỏ hơn ngưỡng, ScanWindow được tăng lên và ScanInterval được giảm xuống, chế độ ngủ là Light Sleep để CPU tỉnh lại nhanh hơn ít tốn năng lượng hơn trong khoảng thời gian ngắn.

Thuật toán hoạt động và thay đổi được các thông số theo biến đổi môi trường.

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## KẾT LUẬN

Trong chủ đề lần này, một mô hình hoạt động của Bluetooth Low Energy là Advertiser và Scanner được giới thiệu. Với chức năng phát gói tin của Advertiser, ta phần nào thấy được hoạt động của các thiết bị Beacon. Đồng thời thuật toán hDiscovery được đề cập và áp dụng ở Scanner với khả năng thích ứng với môi trường khám phá thiết bị để tối ưu năng lượng. Qua thực nghiệm với vi điều khiển ESP32, có thể thấy thuật toán đã hoạt động tương đối tốt, điều này rất quan trọng đối với các thiết bị IoT hạn chế về năng lượng.

## HƯỚNG PHÁT TRIỂN

Chủ đề lần này có nhiều hạn chế trong việc kiểm chứng năng lượng và hiệu suất của thuật toán hDiscovery. Trong tương lai, những nghiên cứu sâu hơn về mặt tính toán năng lượng của hDiscovery so với các thuật toán khác. Để làm được điều này, việc tạo ra một môi trường với các thiết bị Advertiser được kiểm soát cho quá trình kiểm chứng.

Một hướng phát triển khác là triển khai thuật toán với các vi xử lý khác nhau, điển hình là các thiết bị thuần BLE như của Nordic hay của TI (Texas Instruments) để kiểm chứng xem thiết bị hiệu quả nhất.

PHỤ LỤC

**Phụ lục 1**: Tất cả mã nguồn cho hệ thống.

Link Driver:

<https://drive.google.com/drive/folders/1Qs8wqucOpRPXbcZrJlAcmqUN1Hjto3GI?usp=sharing>

**Phụ lục 2**: Link video giải thích mã nguồn Scanner.

Link Youtube:

<https://www.youtube.com/watch?v=wripgrWSdMw>

TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | B. Han, J. Li, and A. Srinivasan, “On the energy efficiency of device discovery in mobile opportunistic networks: A systematic approach,” IEEE Trans. Mobile Comput., vol. 14, no. 4, pp. 786–799, Apr. 2015. |
| [2] | Bo-Ren Chen, Shin-Ming Cheng, and Jia-Jhun Lin, “Energy-Efficient BLE Device Discovery for Internet of Things.” Fifth International Symposium on Computing and Networking, 2017. |
| [3] | Hung Bui, “Bluetooth Smart and the Nordic's Softdevices - Part 1 GAP Advertising.” Nordic Semiconductor DevZone, [Online]. Available: [*https://devzone.nordicsemi.com/guides/short-range-guides/b/bluetooth-low-energy/posts/bluetooth-smart-and-the-nordics-softdevices-part-1*](https://devzone.nordicsemi.com/guides/short-range-guides/b/bluetooth-low-energy/posts/bluetooth-smart-and-the-nordics-softdevices-part-1) |
| [4] | Sanket Hiremath, “Bluetooth Low Energy(BLE) in a Nutshell.”, Linkedin, [Online]. Available:  [*https://www.linkedin.com/pulse/bluetooth-low-energyble-nutshell-sanket-hiremath*](https://www.linkedin.com/pulse/bluetooth-low-energyble-nutshell-sanket-hiremath) |
| [5] | MartinBL, “Bluetooth low energy Characteristics, a beginner's tutorial.” Nordic Semiconductor DevZone, Mar. 2016, [Online]. Available: [*https://devzone.nordicsemi.com/guides/short-range-guides/b/bluetooth-low-energy/posts/ble-characteristics-a-beginners-tutorial*](https://devzone.nordicsemi.com/guides/short-range-guides/b/bluetooth-low-energy/posts/ble-characteristics-a-beginners-tutorial) |
| [6] | “Insight Into ESP32 Sleep Modes & Their Power Consumption.”, LastMinus ENGINEERS, [Online]. Available:  [*https://lastminuteengineers.com/esp32-sleep-modes-power-consumption*](https://lastminuteengineers.com/esp32-sleep-modes-power-consumption) |