Lab: BLE

Introduction

The purpose of this lab is to get you some hands-on experience with Bluetooth Low Energy. This will come in a couple of different forms:

- Scanning BLE traffic with Wireshark
- Write code for BLE peripherals
- Write code for BLE centrals

To get this working, we'll have to install some tools for interacting with the nRF52840DK hardware. This stuff tends to be pretty finicky. It's really easy to mess it up for some reason or another. Since everyone will be working in small groups, hopefully at least one of you can get stuff working for integrating with wireshark and for programming boards.

Let me know if you run into problems and I will help you debug!

Goals

- Enable BLE scanning with the nRF52840DK and Wireshark
- Write embedded applications capable of performing as BLE peripherals and centrals
- Better understand how BLE communication works
    - Peripheral advertisements, Central scanning, and connections with services

Equipment

- Computer
- nRF52840DK + USB cable
- Smartphone (optional)

Github Classroom

- https://classroom.github.com/a/0Z3krAZ7

Partners

- This lab should be done with **your group of three**

Submission

- Write your answers up for each task and submit a PDF to Gradescope.

  Remember: I'm not looking for a formal lab report. Just your answers in any format that makes sense. The goal is to prove that you did the lab and spent some time thinking about it.

Table of Contents

List of Tasks

- Section 4.1: Determine transmissions per second
- Section 4.2: Entirely explain a BLE packet

- Section 8: Commit your test application modifications

- Section 9.1: Prove that you got advertisements working
- Section 9.2: Commit your BLE advertisement application modifications

- Section 10.1: Prove you got scanning working
- Section 10.2: Wireshark capture of a Scan Request and Scan Response
- Section 10.3: Commit your BLE scanning application modifications

- Section 11.1: Prove you got connections working
- Section 11.2: Commit your peripheral and central connection code

1. Optional: nRF Connect Smartphone App

You can optionally install the nRF Connect app on your phone (it's just called 'nrf Connect for Mobile' probably easier to search, but here are links nonetheless):

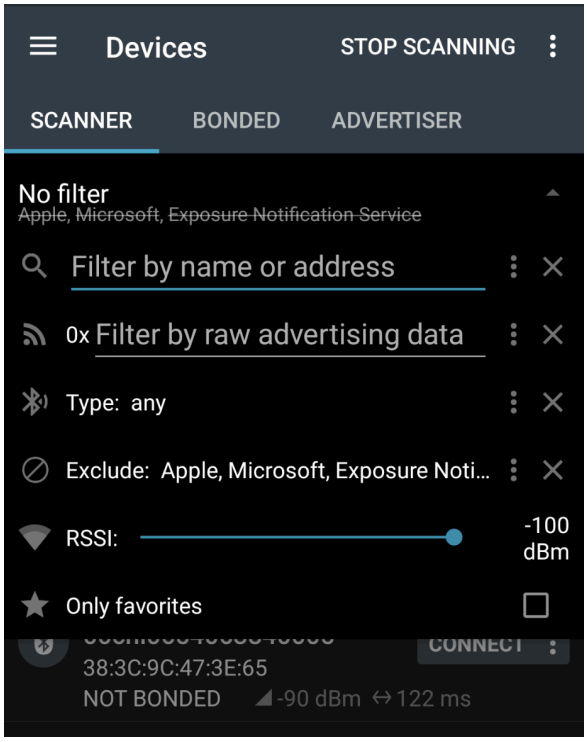
- https://apps.apple.com/us/app/nrf-connect-for-mobile/id1054362403
- https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp

You'll find this app generally useful for understanding what's going on in this lab and interacting with devices around you. I personally used it while developing all of the applications.

The application can allow your phone to scan for devices and to advertise. Clicking an individual device will show more data, possibly including raw advertisement data. You can also connect to devices, look at their services, and read/write characteristics.

Android allows you to do everything, while Apple allows some subset of this. For example on Apple you cannot see the addresses of BLE devices. You may also not be able to see the device at all if its advertisement is malformed.

In the app, you'll find that you're overwhelmed with how many devices there are around. I strongly recommend you filter the devices. You could set an RSSI limit of -70 to only see relatively nearby devices. You should also exclude Apple, Microsoft, and Exposure Notifications so they don't overload your feed.



2. Install nRF Connect for Desktop

Nordic has a suite of *really* nice software tools that help support experimentation with their hardware platforms.The app will work on Windows, MacOS, and Linux. and it uses your nRF52840DK hardware to actually interact with devices.

Not everything in the nRF Connect panel is supported by the nrf52840DK (and some things that look like they wouldn't be supported, are; e.g. the "RSSI Viewer" works fine, despite saying it's for the nRF52832).

Download and install the nRF Connect for Desktop tools:
https://www.nordicsemi.com/Products/Development-tools/nRF-Connect-for-desktop

By default, the desktop app is just an empty shell that can install sub-apps. Go ahead and install the *Bluetooth Low Energy* app, the *Programmer*, and the *RSSI Viewer*.

The *Bluetooth Low Energy* app functions very similarly to the nRF Connect app on your phone. Both allow you to scan for nearby devices, connect to them, and investigate services they provide. Play around for a bit and see what's nearby. You might be surprised by what you find.

When you finish using an app, be sure to disconnect from it:

TASK: None. Continue to the next section.

3. Integrate BLE Scanning into Wireshark

Next, we're going to add an external capture source to Wireshark that allows it to sniff BLE communication by using the nRF52840DK. The full guide that we're following is here: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_sniffer_ble%2FUG%2Fsniffer_ble%2Finstalling_sniffer.html

1. Get a copy for the sniffer ZIP: https://www.nordicsemi.com/Products/Development-tools/nrf-sniffer-for-bluetooth-le/download

2. Open the *Programmer* app, and drag the `/hex/sniffer_nrf52840dk_nrf52840_4.1.1.hex` precompiled firmware over for programming. Then write that firmware to your nRF52840DK.

3. The sniffer receiver is written in Python. You'll need Python3 and `pyserial >= 3.5`. If you don't have Python3, follow the python install guide. For pyserial, you can run `python3 -m pip install pyserial` once Python is installed. This *does* work on Windows with a little bit of effort.

4. We need to copy over the “extcap” stuff to the correct folder so Wireshark can find it. I can’t write better instructions than Nordic already did: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_sniffer_ble%2FUG%2Fsniffer_ble%2Finstalling_sniffer_plugin.html
   (Note: we’ve already handled the python requirements from step 1 by installing `pyserial`)

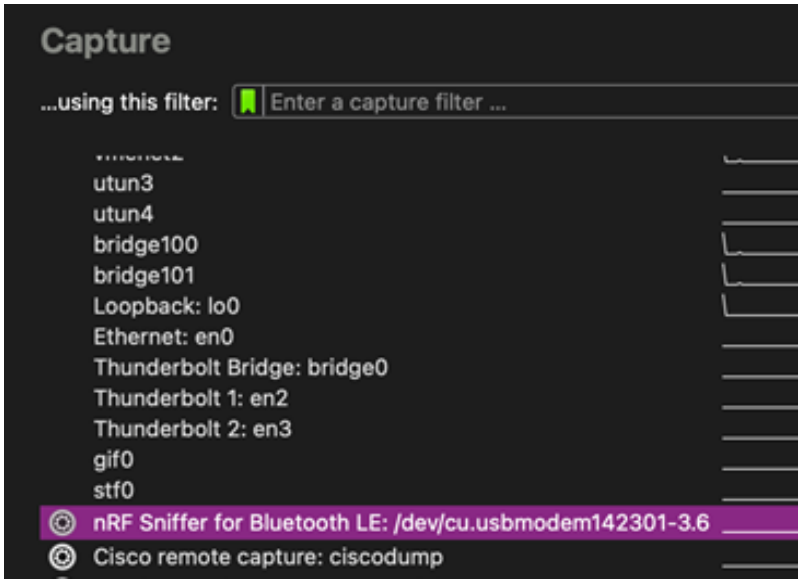
What's `extcap`?

We are setting up wireshark to use an **ext**ernal **cap**ture device (your dev kit). That requires a few pieces, which those instructions walk you through.

- First, you need a physical radio which is configured to sniff packets.
- Then, you need some interface software that runs on your computer and talks to the radio (this is the `nrf_sniffer_ble` program – it doesn't actually sniff, it just sets up a serial tunnel to record packets being streamed off by the firmware loaded on the dongle).
- Finally, wireshark needs to know what kind of packets are being sniffed and how to decode them. That's what the 'profile' is.

Heads Up (for Windows folks): The default extcap folder on windows is a temporary folder. If you suddenly can't find the capture interface and it used to be there, check if you need to re-copy the `extcap` files and set it up again.

5. Finally, make sure you have your nRF52840DK reprogrammed and connected over USB, then either restart Wireshark or go to “Capture Menu -> Refresh Interfaces”. You should now see a new capture interface: “nRF Sniffer for Bluetooth LE”.



Double-click it to start capturing!

Lots of things can go wrong here! Be sure that you’re following all the steps and didn’t skip anything. Also check the troubleshooting steps here: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_sniffer_ble%2FUG%2Fsniffer_ble%2Ftroubleshooting.html&cp=10_5_6

If you’re still having problems, definitely reach out and I’m happy to help!

TASK: None. Continue to the next section.

4. Investigating BLE Advertisements

Now that you've (hopefully) got the Wireshark external capture working, let's investigate some BLE packets! Run wireshark and collect packets for a few seconds. Then take a look at the packets you received and answer a few questions. Include screenshots as makes sense.

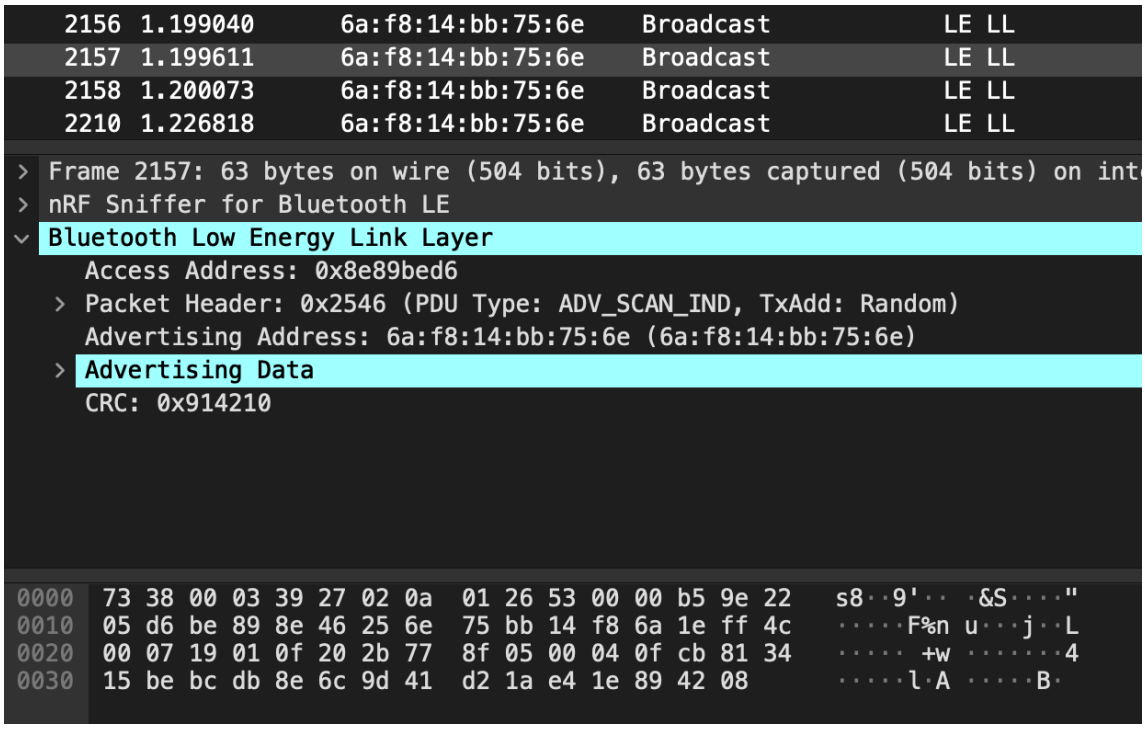
1. **TASK:** How many transmissions do you see in one second?

   Note: if you don't see many devices around first HOW?! and secondly, try again on campus. I was literally collecting *thousands* of packets from my office.

2. **TASK:** Pick a received packet and explain the meaning of all of the bytes of it.

   Note: you can ignore the bytes that are part of the "nRF Sniffer for Bluetooth LE". That appends extra bytes to the start with metadata.

   The real data should be 47 bytes or less and will be highlighted when you select "Bluetooth Low Energy Link Layer" in Wireshark. Clicking different parts within this will highlight the bytes that correspond to different fields.



- I recommend you keep Wireshark up on one of your computers as you do the next steps. You'll need to do some scanning again to check that stuff is working.

5. Create Your Lab Git Repo

I'll want to look at the code you wrote, so I need to give you somewhere to put it. Github classroom makes private repos for each student team so you can get the starter code and upload your own modifications. I can access all student repos, but you can only access your own.

- There is a github classroom link on the first page of this document. Click it!
- Pick a team name
  - Unless someone else already started it, in which case, join their team name
- Generally, do what github classroom says
- At the end, it should create a new private repo that you have access to for your code
  - Be sure to commit your code to this repo often during class!
- The repo link might 404. If so, you first have to go to https://github.com/nu-ce346-student and join the organization
  - I'm reusing the CE346 org for student repos for this class
- Clone the repo locally on your computer
  - If you're on Windows: git BASH does a good job https://gitforwindows.org/
  - If you're on MacOS or Linux, you can clone the repo from command line
  - We'll be using VSCode for everything, and it has a mechanism for working with git too, so you could use that: https://code.visualstudio.com/docs/sourcecontrol/overview#_cloning-a-repository

TASK: None. Continue to the next section.
However, make sure to commit your code as you go, as I'll want to see the final results.

6. Setting up PlatformIO

To program our boards, we're going to use PlatformIO. It's a relatively new embedded toolchain management system, which I'm honestly pretty excited about. One of the really hard parts about embedded systems development is making sure you have the right tools installed on your computer: compiler, programmer, debugger, etc. And every hardware board and software framework you work with has slightly different tools that are required. PlatformIO automatically determines what stuff you need to program a board, and so far in my experience it does a pretty good job. It has worked on Windows, Linux, and MacOS (both Intel and ARM).

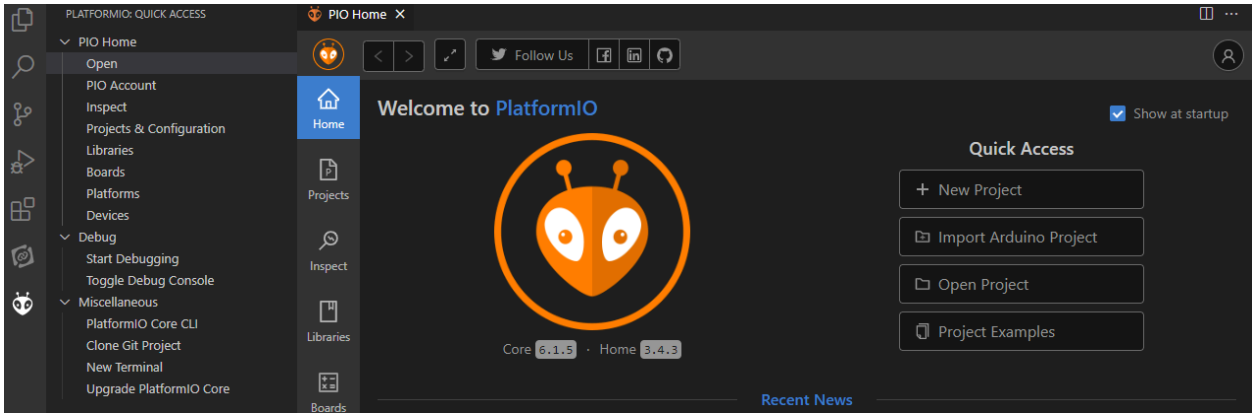
For better and for worse, PlatformIO is almost entirely used as an extension for VSCode, so that means we'll be using VSCode to write our software and PlatformIO to manage our toolchains, compile code, upload it to boards, and display output from boards.

1. Install VSCode and Platform IO

   Follow the steps here: https://platformio.org/install/ide?install=vscode

   It'll take a minute or two to actually install PlatformIO. When it's done, you'll have to reload VSCode and the little Ant icon will appear on the left side of it.

2. Get yourself to the PlatformIO homepage

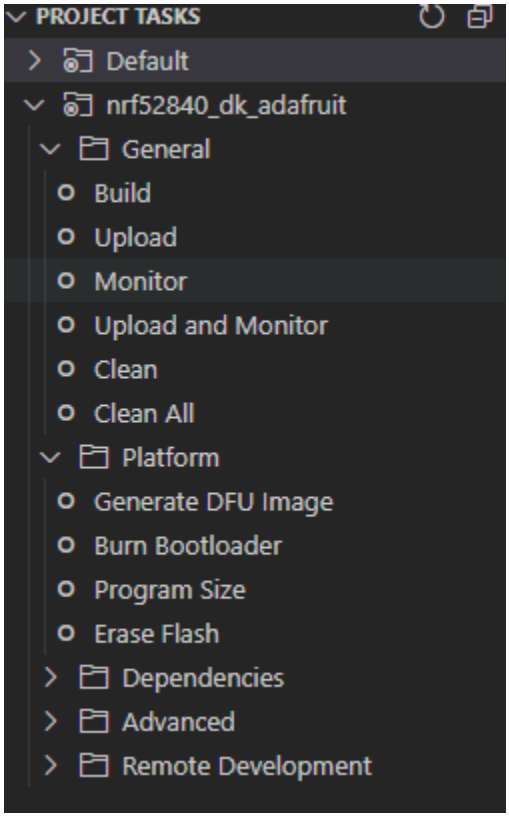


3. Use the "Open Project" button on the PlatformIO homepage to open one of the folders within your group's Github starter repo. The best one to start with is "blink-and-print" since we'll use that in the next section.

4. You should now have two useful toolbars to interact with code.

   The first is the VSCode explorer tab on the left, which will show you the file structure of the application and allow you to open the code in src/main.cpp.

   The second is the PlatformIO tab on the left, which will show you “Project Tasks” like Build (compile code), Upload (load code onto the nRF52840DK), and Monitor (open a serial console to see print output from the nRF52840DK). I mostly use “Upload and Monitor” which recompiles, uploads, and opens a serial terminal all at once. If you later end up opening multiple projects at once, the blue bar at the bottom has both shortcuts to these actions and tells you which project the actions are applying to.



TASK: None. Continue to the next section.

7. Loading the Bootloader

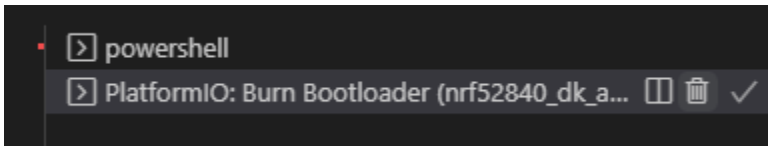
This step is **VERY IMPORTANT**. If you skip this, you will hypothetically spend an entire darn Thursday trying to determine why no application code works on your boards at all.

Good news, you only have to run it once per dev kit. After that you can load applications as many times as you want, and as long as you never entirely “Erase Flash” on the board, they’ll all work.

Bad news, this step is very buggy. It didn’t work well at all on MacOS for me for some reason, and it totally fails if *anywhere* in your file path you have a folder with a space in its name. For example, if your Windows username is your first and last name. I have a PR out to fix this, but we’ll have to deal with it ourselves.

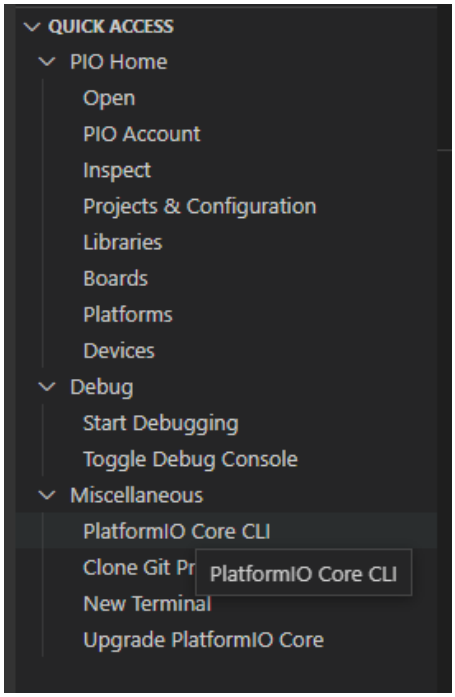
- Try the “Burn Bootloader” option under “Project Tasks” with a board connected to your computer.
    - This should and *could* just work. You’ll have to look at the terminal output to see if everything seems valid or not. If you think it worked, move on to the next section. You can always come back later if applications don’t work.

- If it just hangs forever, you’ll need to stop it. You can exit a script by clicking the “trash can” in the process list on the right.



- If it either hung, said something dumb about invalid files, or otherwise failed, we’ll need to do it manually.

- In the PlatformIO bar, choose “PlatformIO Core CLI” on the left.



- Type in the command: **`pio run -v -t bootloader`**
    - Again, if everything just works at this step, you can be done. If not, we’ll keep going.

- In the text output from running that command, look for a line starting with “`nrfjprog -program`”. That’s the new command we’re going to need to run. Except that we should put double quotes around the path argument to fix space issues.

- Type in the command: `nrfjprog --program “filepath from that line” -f nrf52 --chiperase --verify`
    - This should actually work this time.

- If things still don’t work for you, you should first have a teammate try to burn the bootloader for you. If no one on your team can burn a bootloader, it’s time to talk to the professor for help.

8. Programming a Test Application

We're going to (*finally*) load some code on the dev kit and start playing around with it! We'll start with the "hello world" of embedded systems: blinking some LEDs.

- Open the "blink-and-print" project.

- Take a look at the code and understand what's going on with it. Some functions are taking from the Arduino API: https://www.arduino.cc/reference/en/

- Upload the code to the board and Monitor the board. You should see that LED1 on the board toggles once per second. You should also see print statements appear on the serial console.
    - If nothing at all happens, you should double-check that you uploaded code. Then double-check the bootloader step from before.
    - If just the LED blinks, you'll have to debug your serial connection to the board. I don't expect students to have issues here though.

- You can click the Reset button on the board to restart the code without re-uploading it. It's labeled "IF BOOT/RESET".

- Play around with the code here and make sure you can modify it successfully. Change the rate, or which LED is toggling, or the print output, or something like that.

TASK: Commit and Push your modified code to your Github classroom repo.

9. Programming a BLE Advertiser

Now that some application is working, let's move on to more complicated stuff. We'll send BLE advertisements from a board that's been programmed as a BLE peripheral.

For this lab, we'll be using an Adafruit BLE library called: Bluefruit52. It wraps up the default Nordic SDK calls in a few simple-to-use C++ classes. It's not exactly the most documented thing around, but honestly I think it's really well designed.

Bluefruit API: (occasionally out of-date, but mostly good)
https://learn.adafruit.com/introducing-the-adafruit-nrf52840-feather/bluefruit-nrf52-api

Bluefruit Source Code: (I end up looking here if the API isn't clear)
https://github.com/adafruit/Adafruit_nRF52_Arduino/tree/master/libraries/Bluefruit52Lib/src

Bluefruit example code: (Useful for examples of how to get stuff working)
https://github.com/adafruit/Adafruit_nRF52_Arduino/tree/master/libraries/Bluefruit52Lib/examples

- Open the "ble-peripheral-advertise" project. Build the code, Upload it, and Monitor the board output.
    - A new BLE device should begin advertising with the name "CS397 BLE Device".
    - You can use Wireshark or a phone with the nRF Connect app to see that the device exists.
    - When the device starts, it prints out some information about its BLE configuration including its BLE address. You might have to hit the Reset button to see the message (as it likely printed before the Monitor task had started).

- Play around with this code:
    - Change the device's name to reflect your team in some way. The goal here is to know that you're working with your own device, not someone else's.
    - Change the advertising interval so that packets are sent every 333 ms.

- Add appearance to the advertising payload. The value 0x0040 should make the device claim to be a "Generic Phone" per the BLE specification:
  https://specificationrefs.bluetooth.com/assigned-values/Appearance%20Values.pdf
    - You'll need to look through the API/source/examples to figure out how to do this

1. **TASK:** prove that you got advertisements working
    - A screenshot from Wireshark or even a phone would be fine here

2. **TASK:** Commit and Push your modified code to your Github classroom repo.
    - Should include new device name, change advertisement interval, and the addition of appearance to the advertisement.

10. Programming a BLE Scanner

Bluefruit52 and the nRF52840DK also support the Central role, so let's try that out too! Particularly, scanning for other BLE devices is a very important and useful functionality.

- Open the "ble-central-scan" project. Build the code, Upload it, and Monitor the board output.
    - Your device should begin printing information about the BLE devices around it.
    - If you make one board the scanner, and one the peripheral, you should see the peripheral's advertisements appearing in the scanner's output. (Leave your third device as Wireshark so you can debug!)

If your space is anything like mine, there should be a LOT of data printed. Let's reduce that.

- Add RSSI information to the printed output for each scanned device
    - You'll need to grab it from the advertisement report. Here's the layout of that struct: https://infocenter.nordicsemi.com/topic/com.nordic.infocenter.s132.api.v6.0.0/structble__gap__evt__adv__report__t.html?cp=4_7_3_6_2_1_4_44

- Filter which device information is shown based on RSSI
    - Pick whatever RSSI value you think makes sense, and only print data from devices with an RSSI value greater than that (RSSI is negative, so smaller magnitude is greater signal strength received).
    - You can filter either manually in the "scan_callback()" function, or by applying a configuration to your Bluefruit.Scanner at setup time. Either way works.

One more thing to try here is the addition of Scan Requests and Scan Responses.

- Enable Scan Requests for your scanner. In BLE terms, this is known as "active scanning" and is a configuration you can apply to the Bluefruit.Scanner at setup time. Go check the API for the function.

- Use your Wireshark setup to capture a Scan Request and Scan Response occurring.
    - If there are no devices responding to Scan Requests nearby, you could program your peripheral to have Scan Response data! (but I won't require you to)

Tasks are on the next page!

1. **TASK:** prove that you got scanning working.
    - A screenshot of terminal output works here.

2. **TASK:** demonstrate a scan request and scan response pair for a single device
    - A screenshot from Wireshark is great

3. **TASK:** Commit and Push your modified code to your Github classroom repo.
    - Should include the RSSI modifications you made and active scanning.

11. BLE Services

This is the big finale for this lab. We're going to program BOTH sides of a BLE connection. The end result is that one of your nRF52840DKs will control the LED on the other nRF52840DK, which is incredibly satisfying when you get it working.

The "ble-peripheral-led-service" app is mostly written for you. It creates a service with two characteristics. The "data" characteristic just has some unique, static data in it so you can know stuff is working. The "control" characteristic should be used to control the LED in some way.

- "start_adv()" is left mostly empty for you. You can decide what information you want to advertise, but you MUST include the service UUID
    - Warning: since it's a custom 128-bit service, that UUID is going to take up a lot of space (18 bytes out of 31), so make sure whatever else you include in the advertisement is succinct.

- "control_char_write_callback()" is called whenever a Central writes to the control characteristic. You should write code that modifies the LED state based on the value written.
    - Note that the LED is active low, so writing "low" turns it on and writing "high" turns it off

- If you have the nRF Connect app on your phone, you can test the peripheral implementation on its own by connecting to the device. The Up/Down arrows by characteristics let you Read/Write them.

The "ble-central-connect" app is mostly empty. You should write the rest of the application so that it scans, discovers your peripheral device, connects to it, discovers the service/characteristics, and writes to the control characteristic to change the LED state.

- Useful here will be some example code with a custom service. It's not exactly the same as what we're doing, but it'll help you be aware of what things to do and what order to do them in.

  Example peripheral service implementation:
  https://github.com/adafruit/Adafruit_nRF52_Arduino/blob/master/libraries/Bluefruit52Lib/examples/Peripheral/custom_hrm/custom_hrm.ino

  Example central service implementation:
  https://github.com/adafruit/Adafruit_nRF52_Arduino/blob/master/libraries/Bluefruit52Lib/examples/Central/central_custom_hrm/central_custom_hrm.ino

Tasks are on the next page!

1. **TASK:** prove to me that this works
    - I'll take your word for it. Write a couple of sentences on what you did and how it worked.
    - You could alternatively include a link to a short video of it working.

2. **TASK:** Commit and Push your modified code to your Github classroom repo.
    - Include both the peripheral and central applications!