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 advertisers/peripherals
- Write code for BLE scanners/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.

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)

Documentation:

- https://docs.zephyrproject.org/apidoc/latest/index.html

Github Classroom

- https://classroom.github.com/a/lc_NXTxr

Partners

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

Table of Contents

List of Checkoffs

- Section 5.1: Show that the LED is now blinking

- Section 7.2: Explain a BLE packet captured with Wireshark

- Section 9.1: Show that you got advertisements working

- Section 10.1: Show that you got scanning working
- Section 10.2: Receive an advertisement from your own advertiser
- Section 10.3: Wireshark capture of a Scan Request and Scan Response

- Section 12.1: Show the count value read multiple times

- Section 13.1: Demonstrate the working application

Lab Setup

1. Get hardware!

For this lab and the next you'll need an nRF52840DK board (blue rectangle) and a USB cable. Staff can provide them to you.

Warning: make sure to always unplug the USB before putting the board in your backpack. They're generally pretty robust, but the USB port can rip right off if the USB is left plugged in.

2. 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):

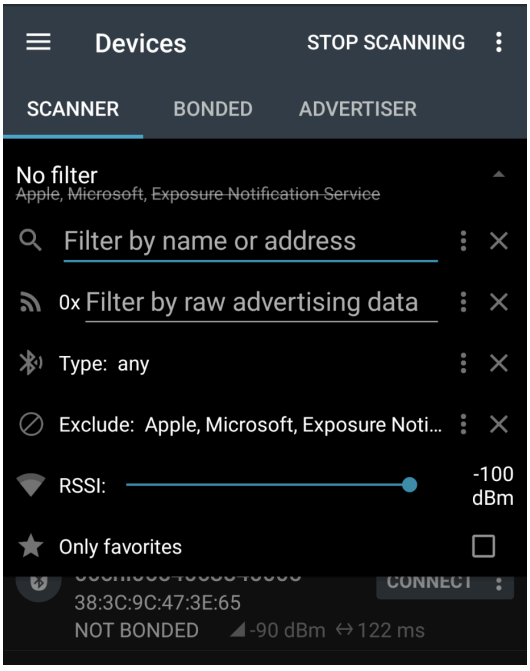
- https://apps.apple.com/us/app/nrf-connect-for-mobile/id1054362403
- https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp

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.



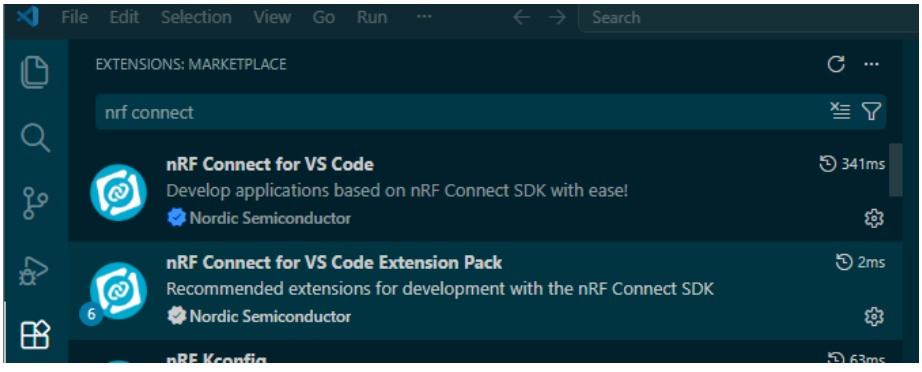
3. Install nRF Connect SDK for VS Code

To program our boards, we're going to use Nordic's nRF Connect extension for VS Code.
https://docs.nordicsemi.com/bundle/nrf-connect-vscode/page/get_started/install.html

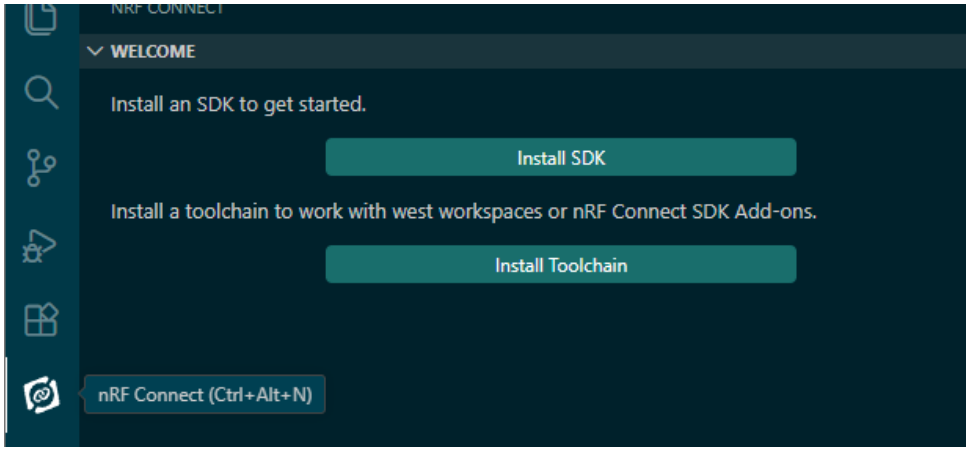
1. Install nRF Command Line Tools
   https://www.nordicsemi.com/Products/Development-tools/nRF-Command-Line-Tools
   - Yes, I know it says it's deprecated.
   - Scroll down to Downloads and select the newest version for your OS.
   - **Linux:** Pick "Linux x86 64" from the dropdown (unless you're a different arch, but I'd be surprised), and then if you're not sure which file, download the DEB file.
     - After it's downloaded, you can install it with "sudo apt install ./name.deb" (where name is its name and you'd better be in the right directory already)
   - **MacOS:** Make sure you pick the "Universal" option if you get one. The Apple Silicon option doesn't seem to actually work for some reason…

2. Make sure you've got VS Code installed. If you're one of the few students who isn't a fan of it, apologies. It happens to be a great tool for nRF Programming since nRF wrote extensions for it to manage software and hardware.

3. In Extensions, install the "nRF Connect for VS Code Extension Pack".
   - I don't know who "Trond Snekvik" is, but I trust them with my life.



4. Open the nRF Connect extension on the left sidebar and click “Install SDK”



   - Options you should pick: “nRF Connect SDK” and version v2.9.2
   - This will take quite a few minutes to download and install.
     - Took me like 10 minutes when testing.
     - While you’re waiting, you can go to the next section to play around with the nRF Connect for Desktop app.

CHECKOFF: None. Continue to the next section.

4. 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

Note: If you're on MacOS or Linux, you will need to install SEGGER J-Link separately. You can install "J-Link Software and Documentation Pack" from here (For M1 Macs, you might have to pick the Universal Installer). If you don't have J-Link, you can get error logs like these:

```
2023-01-31T21:03:20.324Z INFO Installed JLink version does not match the provided version (V7.66a)
2023-01-31T21:03:22.747Z ERROR Unsupported device.
            The detected device could not be recognized as
            neither JLink device nor Nordic USB device.
2023-01-31T21:03:22.747Z ERROR Please make sure J-Link Software is installed. See https://github.com
/NordicSemiconductor/pc-nrfconnect-launcher/#macos-and-linux
```

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*.

1. To use the nRF52840DK, first connect the board to your computer using the USB port on the narrow side of the board (that is, NOT the port labeled "nRF USB").
2. Turn on the board. Make sure the Power Switch in the corner is set to "ON".
    a. The other switches should be on "VDD" and "Default" respectively
3. Choose the Programmer app from the nRF Connect tools.
4. Select the nRF52840DK by clicking "Select Device" in the top left.
5. Then choose "Erase all" to reset the board.
6. After that, you should be ready to use other applications in nRF Connect.

- Windows: If you can't get the board to Erase and it's throwing up red messages about not finding your debugger or running an emulator, then you need to install the USB driver for JLink devices. Go to Device Manager, find your device, which is probably listed as "Bulk interface" with a little yellow exclamation point, mostly the follow this to install the driver: https://wiki.segger.com/Incorrect_J-Link_USB_driver_installed
    - The folder is something like: "C:\Program Files\SEGGER\JLink_V794e"

- MacOS: sometimes the system starts continuously popping up messages about not ejecting the board safely.
    - The solution generally seems to be to reinstall JLink and maybe to update the JLink firmware on your board
    - You can try installing those tools directly with brew:

        - `brew install --cask nordic-nrf-command-line-tools`
        - Then run `jlinkexe` which should hopefully trigger a firmware udpate
    - An extreme, but not unreasonable, option here is to disable those notifications permanently: https://www.reddit.com/r/MacOS/comments/10xpjk9/comment/la7ar8k/?utm_source=share&utm_medium=web3x&utm_name=web3xcss&utm_term=1&utm_content=share_button

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

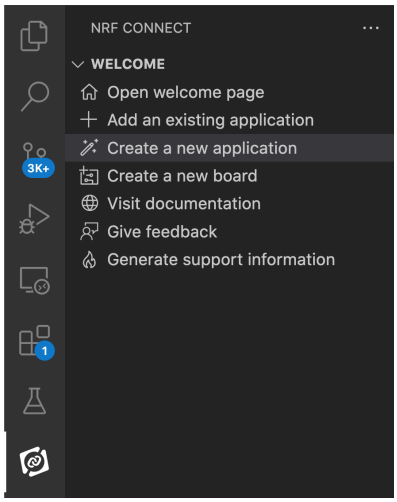
Try out some of those apps you installed. The *RSSI Viewer* should show you signal strength measurements for all 40 BLE channels. The *Bluetooth Low Energy* app functions very similarly to the nRF Connect app on your phone. Both allow you to scan for nearby devices, connect to them, and investigate services they provide. Play around for a bit and see what's nearby. You might be surprised by what you find.

CHECKOFF: None. Continue to the next section.

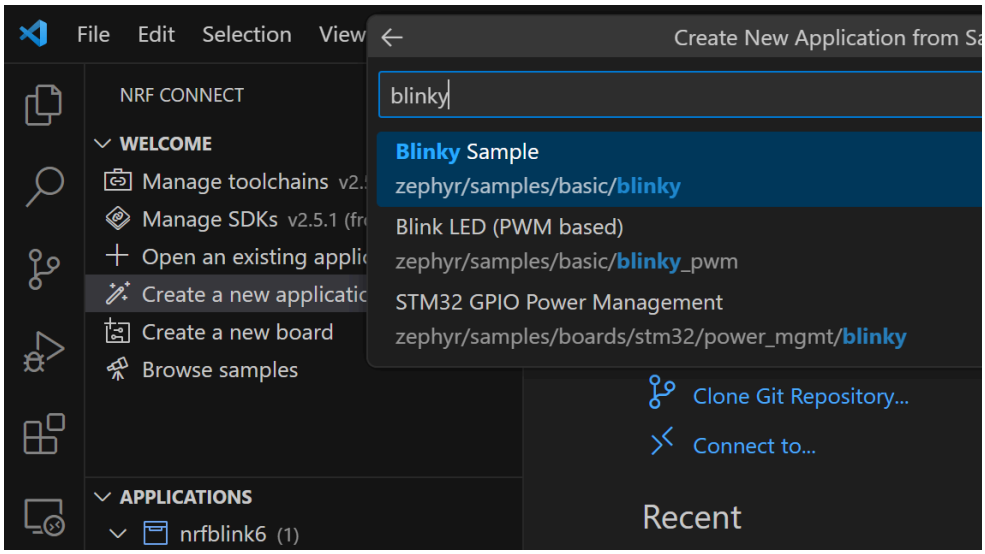
5. Run a Sample Application

Once you're done installing the nRF Connect SDK in VS Code, do this part. We're going to 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 nRF Connect extension on VS Code.
- Choose Create a new application from the side bar.

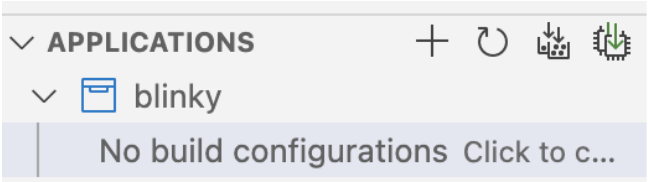


- In the VSCode dropdown, choose to "Copy a sample." Choose the Application template as zephyr/samples/basic/blinky.

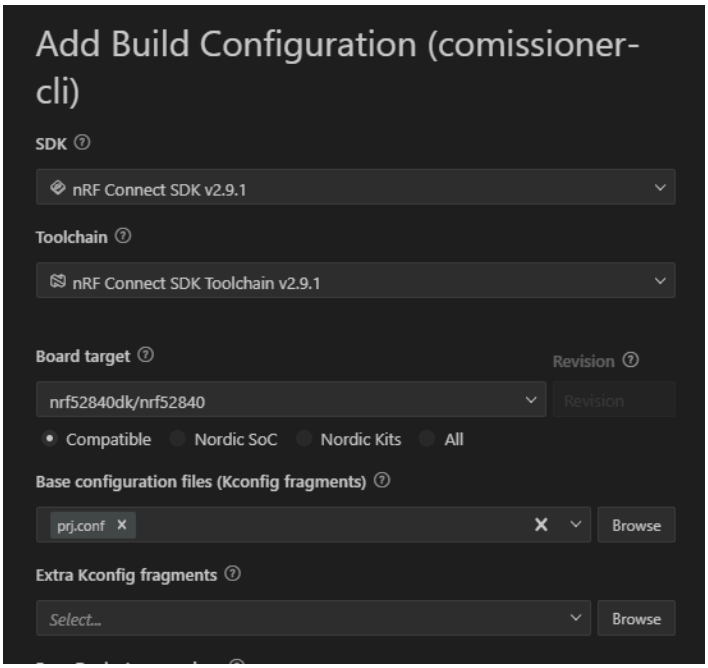


- Click on Create Application.
  - **Windows:** if your username has a space in it, you likely have to put the application in a different location. I made "C:/nrf_apps/" and put my stuff in there.
- You will now see a "blinky" application appear under the Applications tab in the nRF Connect extension

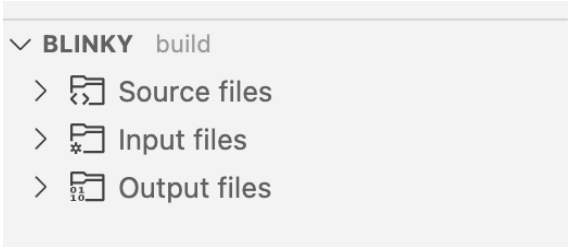
- Before we can run this application, we need to create a Build Configuration for this application. Click on where it says “Click to create one” or “Add Build Configuration”.



- In the Add Build Configuration page, choose:
  - Board as **nrf52840dk/nrf52840**
  - Base Configuration as **prj.conf**
  - Click on “Generate and Build” (previously “Build Configuration”). This should start building the app. It’ll take a minute, let it run until the pop-up in the bottom right finishes.

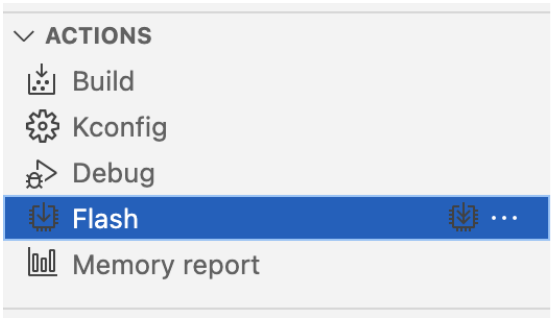


- You will also notice a few new tabs appear below the Application tab on the side bar.
  - In the “BLINKY” tab, you can see the source code for the application. This will change based on the current active application (the one you’ve picked in the Applications tab).



- We will now flash the application to your nrf52840DK board.
  - Connect your board over USB (the one on the narrow end, NOT labeled “nRF USB”)

- Choose Flash from the Actions tab. The Actions tab gives you options to Build, Debug, and Flash your application.



- Note: On MacOS if you're given a choice, pick the lower numbered JLink serial port.

- Once it has flashed, you should see the LED on your board flashing.
  - You may have to power cycle your board for the app to start (either flip the on/off switch or unplug/replug the board)
  - Getting this working is honestly a pretty good accomplishment. It means you managed to successfully set up the toolchain for programming the board. And if you can do it once, you can definitely do it again.

1. **CHECKOFF:** Show that the LED is now blinking

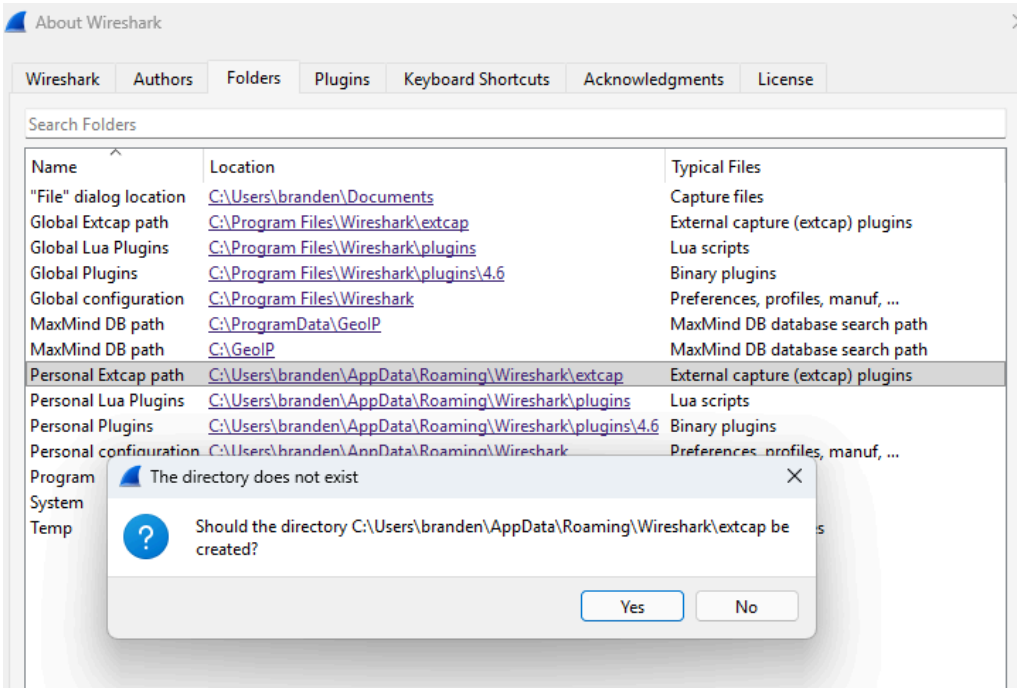
Wireshark Scanning

6. Integrate BLE Scanning into Wireshark

We can use our hardware board to enable Wireshark to scan for BLE devices. There are a few steps here, but it's pretty neat to see when it works, and quite powerful.

1. Install nRF Util. https://docs.nordicsemi.com/bundle/nrfutil/page/guides/installing.html#installing-nrf-util-from-the-web-default
    - Once you've got the executable, you need to run `nrfutil install completion` to finish installation

2. Follow the instructions here for Wireshark setup: https://docs.nordicsemi.com/bundle/nrfutil/page/guides/installing_wireshark.html
    - I had to open Wireshark, go to Help > About Wireshark (or Wireshark > About Wireshark) and then the "Folders" tab, and then double click on the "Personal Extcap path" to create it.



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.

3. Follow the instructions here to upload the sniffer firmware. https://docs.nordicsemi.com/bundle/nrfutil/page/nrfutil-ble-sniffer/guides/requirements.html#programming-the-nrf-sniffer-firmware
    - I had to manually specify the path to the firmware file, which is in `~/.nrfutil/share/nrfutil-ble-sniffer/firmware/`
    - Then I had to power cycle my board
4. Now you can run Wireshark using the new BLE interface your board provides. Instructions here: https://docs.nordicsemi.com/bundle/nrfutil/page/nrfutil-ble-sniffer/guides/running_sniffer.html

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://docs.nordicsemi.com/bundle/nrfutil/page/nrfutil-ble-sniffer/guides/sniffer_troubleshooting.html
- Generally, each group only needs one or maybe two students to get Wireshark working. So as long as it's good for most of you, you're okay.

If you're still having problems, ask for help!

CHECKOFF: None. Continue to the next section.

7. 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..

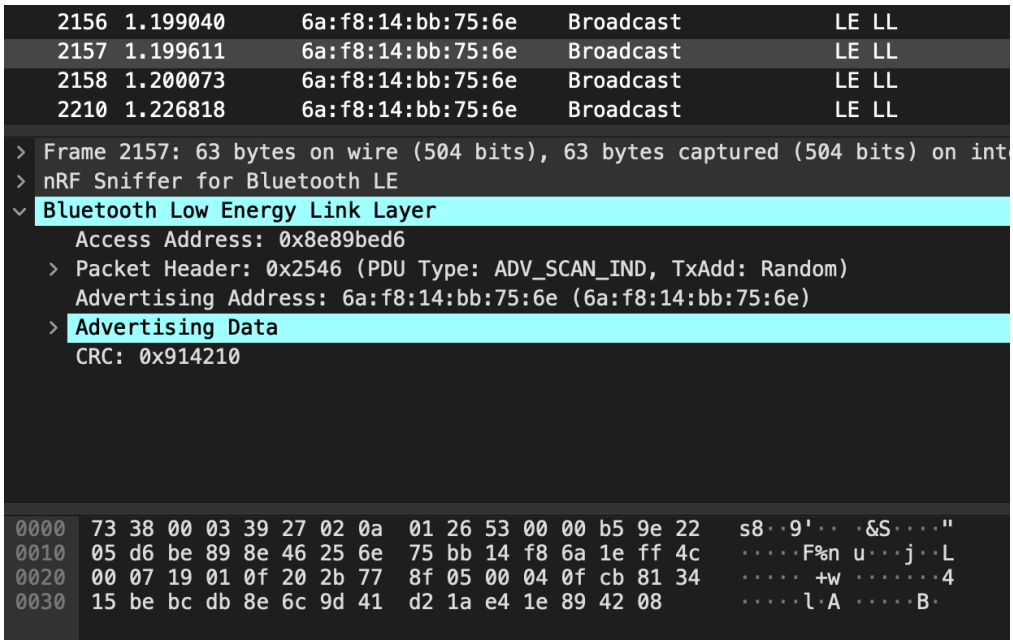
1. How many transmissions do you see in one second?

   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. **CHECKOFF:** Pick a received advertisement, show me the packet data, 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.

BLE Advertising and Scanning

8. Create Your Lab Git Repo

We want to share code between everyone in the group, so we're going to use Github. 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.

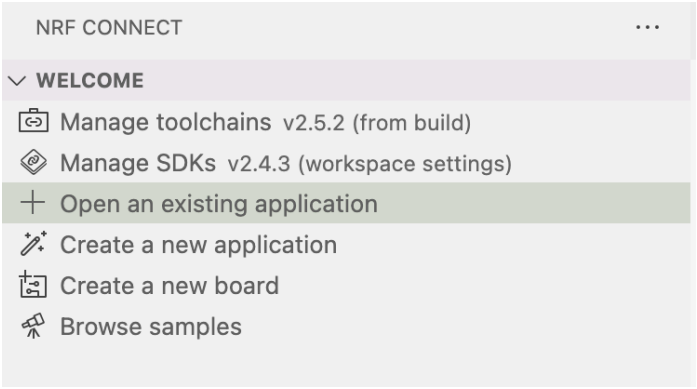
- There is a github classroom link on the first page of this document. Click it!
- Pick a team name
  - Unless someone else already started it, in which case, join their team name
- Generally, do what github classroom says
- At the end, it should create a new private repo that you have access to for 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-cs433-student and join the organization
  - S26: we've had all kinds of problems with Github Classroom lately. It probably sends an invite to the email that you used to create your Github account. But also sometimes it just doesn't. Let us know if you have problems.
- Clone the repo locally on your computer
  - If you're on Windows: git BASH does a good job https://gitforwindows.org/
    - Make sure there are no space characters in the entire path to the repo. Probably put it somewhere like "C:/nrf_apps/REPONAME" if you have a space in your username.
  - 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

CHECKOFF: None. Continue to the next section.
However, make sure to commit your code as you go!

9. Programming a BLE Advertiser

We'll start by sending BLE advertisements from a board that's been programmed as a BLE peripheral. We're using Zephyr, an operating system for the nRF52 devices that provides support for BLE and many other libraries and tools.

- Open the "ble-beacon" application in VSCode using "Open an existing application"



- Add a build configuration for it, same as before:
  - Board as **nrf52840dk_nrf52840**
  - Configuration as **prj.conf** (important, don't skip this. It's not done by default)
- Build the code, Flash it to the nRF52840DK
  - 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).
  - You may see an error, which you can ignore as it doesn't seem to affect the operation of the device.
- Monitor board output
  - To view board print statements, you'll need to open a serial terminal. In the application panel on the left, under "CONNECTED DEVICES", you should see a serial number for your board, then in a dropdown from that, one or more serial devices. The little "plug" icon when you hover over one of the serial devices should open up a serial port to it.
  - If it asks you for settings: 115200, 8n1, and rtscts:off are correct (115200 baudrate, 8 data bits with no parity bits and one stop bit, and no request-to-send/clear-to-send)
  - Hit the reset button to see print output.

- 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.
        - You'll need to change the first argument to `bt_le_adv_start()`
        - Look up the `BT_LE_ADV_PARAM` macro by searching the Zephyr docs
        - Note, the advertising intervals are specified in multiples of 0.625 ms. So, an interval of 0xf0 corresponds to 150 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://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Assigned_Numbers/out/en/Assigned_Numbers.pdf?v=1707335302187 (Section 2.6.3)
    - You might want to look through the SDK (zephyr/samples/bluetooth) to figure out how to do this. In the nRF Connect sidebar of VSCode, you can "Browse Samples", or you could look through the files where they're installed on your computer
    - You will likely find the `BT_DATA_BYTES` macro helpful. Note: `BT_DATA_BYTES` takes bytes one after the other in little endian format.
    - Also, the `BT_DATA_*` defines are helpful as well. (`BT_DATA_GAP_APPEARANCE`)
    - iOS: if you're using the nRF Connect app on iOS, you probably can't see this appearance even if you get it right. Thanks Apple. Use the nRF Connect desktop app instead?

1. **CHECKOFF:** show that you got advertisements working
    - Information on Wireshark or a phone would be fine here
    - Include the Appearance you set

10. Programming a BLE Scanner

Advertisements are only useful if something listens for them. In this portion, we will program the nRF52840DK to support the Central role so it can receive BLE advertisements. Scanning for other BLE devices is a very important and useful functionality.

- Open the “ble-scanner” project. Create a configuration with the right parameters. Build the code, Upload it to your other nRF52840DK, 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.

- Print something special when you receive an advertisement from your own advertiser.

- Filter which device information prints 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).

Next try to use 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 at setup time. Go check the API for the parameter.
    - To get the scan requests to use your actual BLE address, you will also need to add `CONFIG_BT_SCAN_WITH_IDENTITY=y` to the zephyr/prj.conf file.
    - You can edit the file by choosing in the NRF CONNECT side panel: “Config files->Kconfig->prj.conf”

- Use your Wireshark setup with the dongle 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 we won’t require you to)

1. **CHECKOFF:** show that you got scanning working.
    - Data in the terminal output works here.

2. **CHECKOFF:** show that you were able to receive an advertisement from your own advertiser.
    - Again, in terminal is fine.

3. **CHECKOFF:** demonstrate a scan request and scan response pair for ANY device
    - Wireshark is the best for this.
    - Doesn't have to be your device (as you probably didn't add scan response data to it)

(You can do all of these checkoffs at one time, that's fine)

BLE Peripheral and Central

11. Programming a BLE Peripheral

The peripheral device acts as a server. It uses the GATT to host services and characteristics which other BLE devices can interact with. First we will create a simple service that exposes a counter from the nRF52840DK.

- Open the “ble-peripheral” app in VSCode. Configure the build and do all the other normal stuff.
- Change the device name to something unique you can identify.
- Change the UUID very slightly so it doesn’t match any other group in the class.
- Build and flash the app to an nRF52840DK.
    - If you get linker errors during the build, make sure CONFIG_BT=y and `CONFIG_BT_PERIPHERAL=y` are in the prj.conf file. (“Config files->Kconfig->prj.conf”)
- Connect to your board using the nRF connect app (or any other tool).
- You should see a custom service with UUID similar to “5253FF4B-E47C-4EC8-9792-69FDF4923B4A”. Select that service. It should have one characteristic which supports reading. Read the characteristic. You should receive a fixed value (it will not change if you read multiple times).
- Update the code to change from a fixed value to a 32 bit counter. The counter should increment every time the characteristic is read. It should maintain its count as clients connect and disconnect (that is the count should only reset on a power cycle or hard reset).

Tip: You can use the nRF Connect for Desktop BLE Standalone tool to help inspect your peripheral’s GATT server. There is also a `ble-central-explorer` app in the github repo.

CHECKOFF: None. Continue to the next section.

12. Programming a BLE Central

The central device will connect to your peripheral and display the current count.

- Open the “ble-central” app from the repo in VSCode.
- It is helpful to read the example code “bottom up”. That is, scroll to the bottom and start with the main function. You should then be able to follow the functions up through the file to get a sense of what is going on.
- The key function for your central device is the library function `bt_gatt_discover()`. This function is used to discover services and attributes.
    - The discover mechanism is configured using the discover_params struct.
    - You will notice that the `bt_gatt_discover()` function is called multiple times. This allows finding the service first, and then finding the contained characteristic.
- Update the code to discover your counter service and attribute and UUID value.
- Once you find the correct attribute, read it to get the current count. Then print the count to the terminal.
- Update the code to only connect to your group’s peripheral.
- In summary, your code should:
    - Scan for advertisements.
    - Find an advertisement from your device.
    - Connect to your device.
    - Read the characteristic.
    - Print the count value.

1. **CHECKOFF:** Show the count from the terminal and demonstrate that it changes.
    - Connect multiple times to see the count increase.

BLE Service

13. LED Control Application

This is the big finale for this lab. We're going to program one board as a peripheral with controllable LEDs and one board as a Central which controls LEDs based on button presses. You'll have to make new applications for both of these so you don't overwrite other applications.

Peripheral Implementation

- You must implement a new GATT service with a UUID similar to `BDFC9792-8234-405E-AE02-35EF4174B299`.
    - But you should change it slightly to differentiate your group from other groups.
- It must contain four characteristics. The characteristics **must** use 16 bit UUIDs: `0x0001`, `0x0002`, `0x0003`, and `0x0004`. The characteristics correspond to the LEDs on the nRF52840DK:

Characteristic UUID	LED	Length (Bytes)
`0x0001`	LED1	1
`0x0002`	LED2	1
`0x0003`	LED3	1
`0x0004`	LED4	1

- Each characteristic must be writable. Writing any non-zero value to the characteristic should turn on the corresponding LED, while writing a value of zero should turn off the corresponding LED. By default, all LEDs should be off.
- For an example of controlling the LEDs, see the "blink" app in the repo

Central Implementation

- Scan for advertisements with the `BDFC9792-8234-405E-AE02-35EF4174B299` service (well, whatever you changed it to anyways). It should connect if it finds a matching device and then it should discover services/characteristics.
- The `bt_gatt_write()` function can be called any time while a connection exists, it doesn't have to be called during discovery. Your discovery function could save the uint16_t values returned from `bt_gatt_attr_value_handle()` to read from the characteristic at a later time.
- Button presses should result in `bt_gatt_write()` calls. For an example of reading button presses, see the "button" app in the repo.

- You can decide how exactly you want this to work: either pressing a button should toggle the state of the corresponding LED, or pressing a button lights the LED and releasing a button unlights it.

1. **CHECKOFF:** Demonstrate the working LED control application.
    - Explain the code you wrote to make this work as well for both peripheral and central.
    - Was anything particularly challenging to get working?