CSE 122 / CSE 222C / WES 269 Wireless Networks
Winter 2025

[WI25 WxIoT] 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/rN38XDxS

Partners

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

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.

CSE 122 / CSE 222C / WES 269 Wireless Networks
Winter 2025

Table of Contents

- Introduction
- Table of Contents
- List of Tasks
- Pre-Lab
  - A. Optional: nRF Connect Smartphone App
  - B. Install nRF Connect for Desktop
  - C. Install nRF Connect SDK for VS Code
  - D. Run a Sample Application
- Lab
  - Wireshark Scanning (Week 1)
    - E. Integrate BLE Scanning into Wireshark
    - F. Investigating BLE Advertisements
  - BLE Advertising and Scanning (Week 1)
    - G. Create Your Lab Git Repo
    - H. Programming a BLE Advertiser
    - I. Programming a BLE Scanner
  - BLE Peripheral and Central (Week 2)
    - J. Programming a BLE Peripheral
    - K. Programming a BLE Central
  - BLE Service (Week 2)
    - L. LED Control Application

CSE 122 / CSE 222C / WES 269 Wireless Networks
Winter 2025

List of Tasks

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

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

- Section I:
    - Prove you got scanning working
    - Receive an advertisement from your own advertiser
    - Wireshark capture of a Scan Request and Scan Response
    - Commit your BLE scanning application modifications

- Section J:
    - Document the initial characteristic value
    - Commit your peripheral code

- Section K:
    - Show the count value read multiple times
    - Commit your central code

- Section L:
    - Write up what you did to make this work
    - Commit your service application code, peripheral AND central

CSE 122 / CSE 222C / WES 269 Wireless Networks
Winter 2025

Pre-Lab

A. 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):

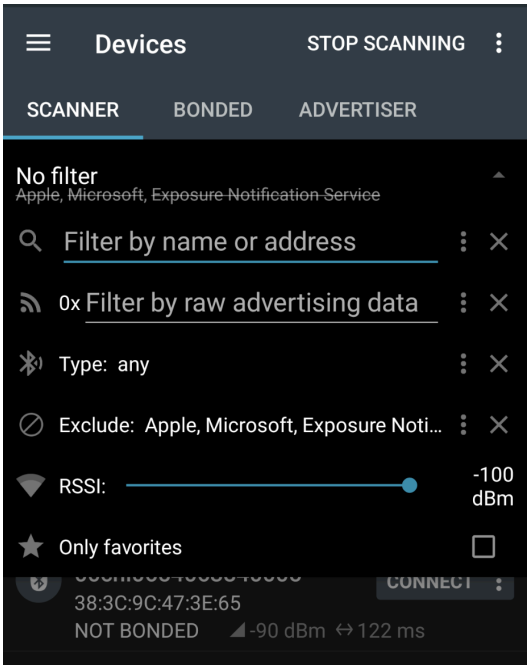
- https://apps.apple.com/us/app/nrf-connect-for-mobile/id1054362403
- https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp

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.



B. 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"

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.

TASK: Provide a screenshot of the RSSI viewer app.

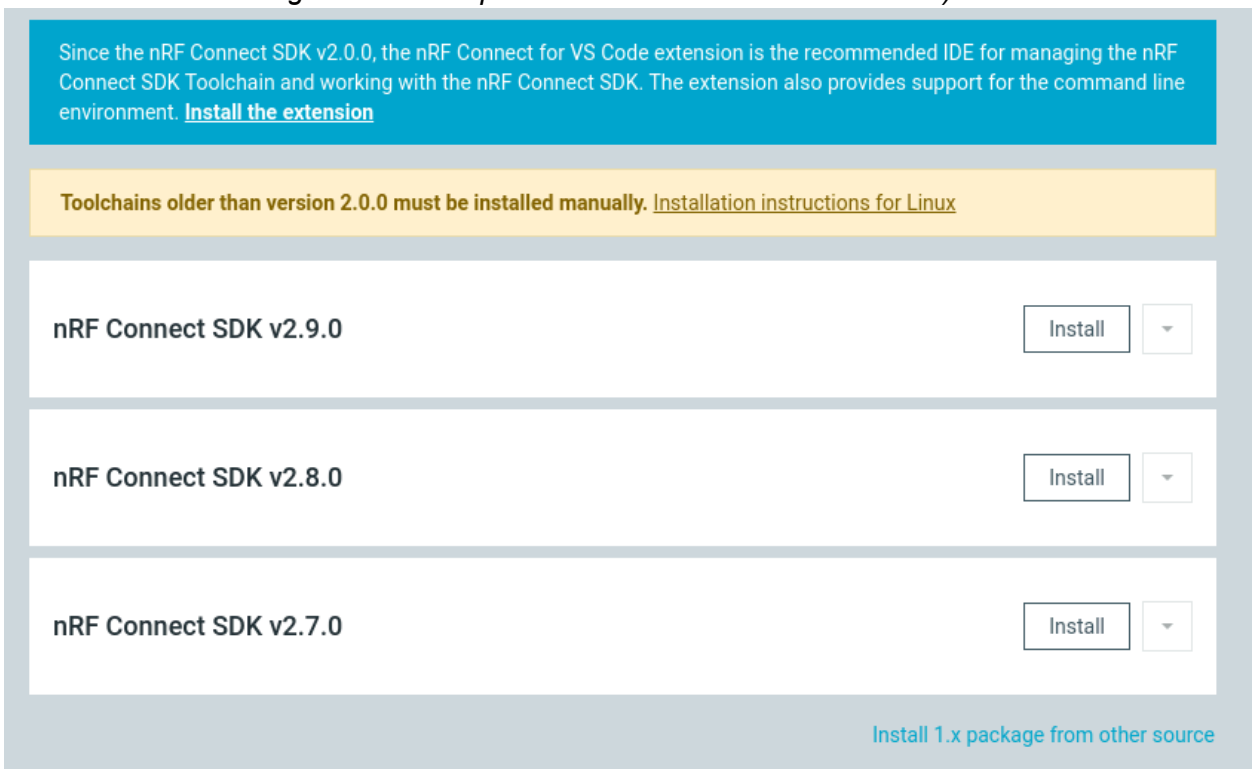
C. Install nRF Connect SDK for VS Code

To program our boards, we're going to use Nordic's nRF Connect extension for VS Code. Using this installation guide: https://nrfconnect.github.io/vscode-nrf-connect/get_started/install.html

1. Install the nRF Connect SDK using the Toolchain Manager on the nRF Connect for Desktop.

    a. Open the nRF Connect for Desktop. Install the Toolchain Manager tool.

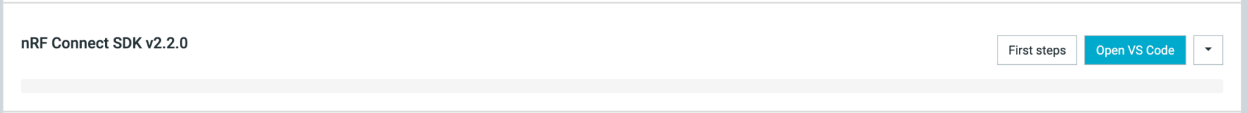
    b. Open the Toolchain Manager and install nRF Connect SDK v2.9.0. *(It seems like nRF now recommends installing the toolchain from the VS Code extension. You can install the extension from the extension marketplace and install the toolchain with the "Manage toolchain" option in the sidebar of the extension)*



c. Once the nRF Connect SDK is installed, we will use it on Visual Studio Code using the **nRF Connect for VS Code** extension.

2. Install nRF Connect for VS Code Extension

   a. Click on the Open VS Code button.



   b. A notification appears with a list of missing extensions that you need to install, including those from the nRF Connect for Visual Studio Code extension pack. Click “Install missing extensions”.

   c. Once the extensions are installed, click the Open VS Code button again. You should see the nRF Connect extension installed in your VS Code.

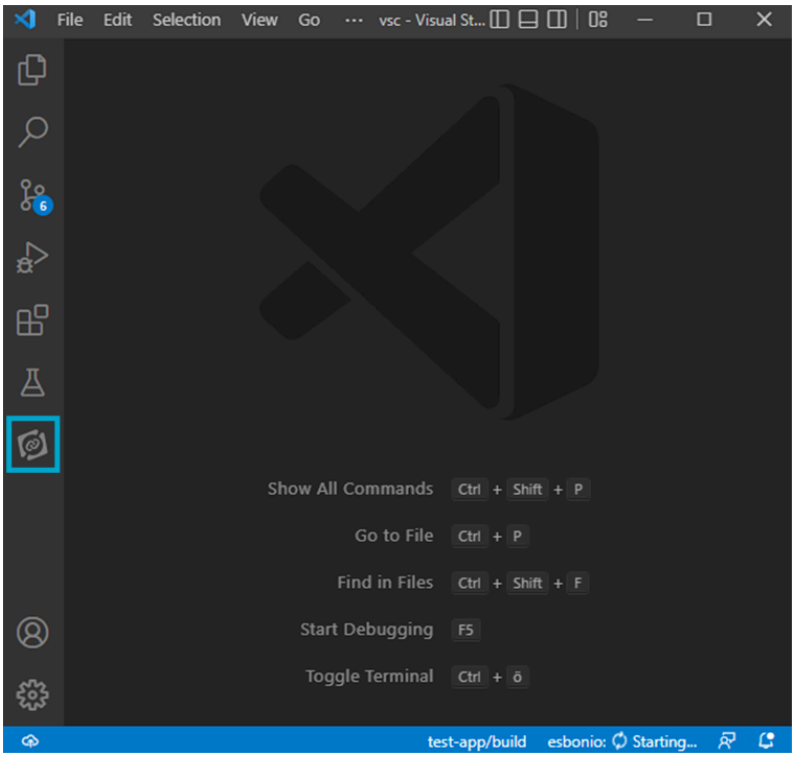


Image source: https://nrfconnect.github.io/vscode-nrf-connect/get_started/install.html

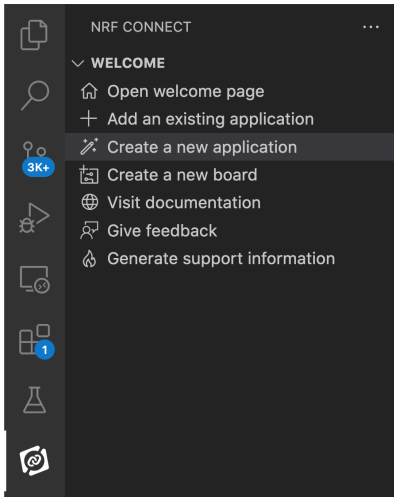
- **Linux:** you need to install nRF Command Line tools manually. Pick “Linux x86 64” from the dropdown (unless you’re a different arch, but I’d be surprised), and then if you’re not sure which file, download the DEB file.
  - After it’s downloaded, you can install it with “sudo apt install ./name.deb” (where name is it’s name and you’d better be in the right directory already)

TASK: None. Continue to the next section.

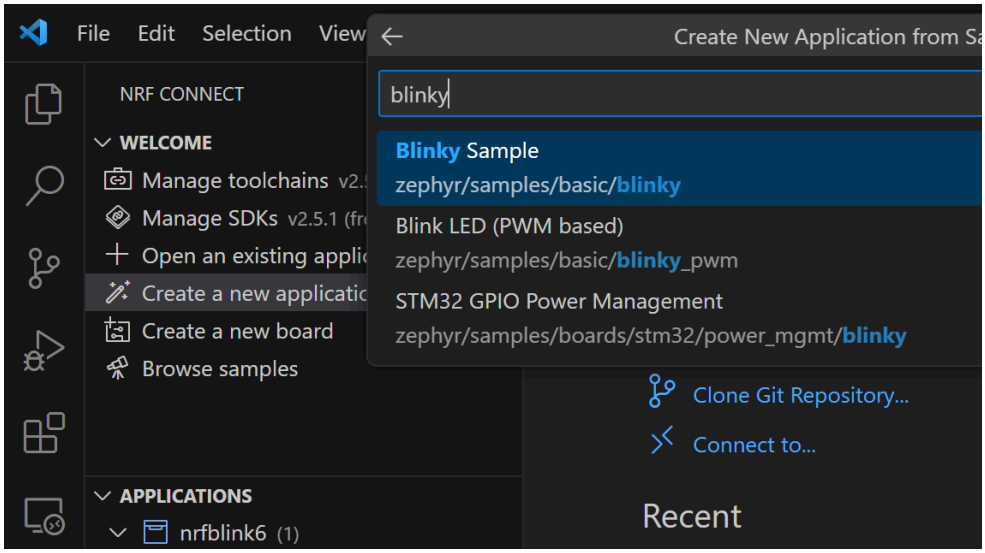
D. Run a Sample Application

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.

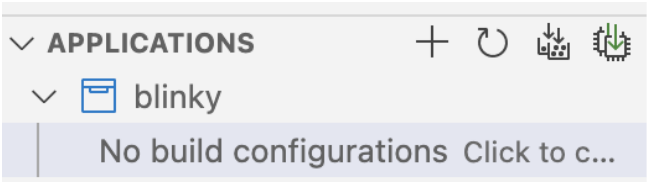
- Open the nRF Connect extension on VS Code.
- Choose Create a new application from the side bar.



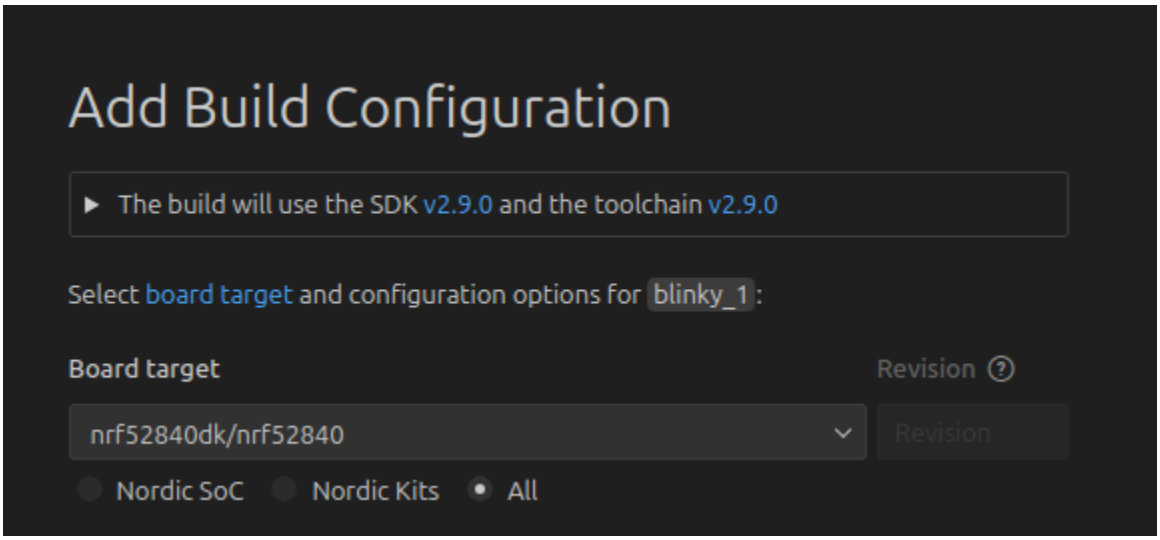
- In the VSCode dropdown, choose to "Copy a sample." Choose the Application template as zephyr/samples/basic/blinky.



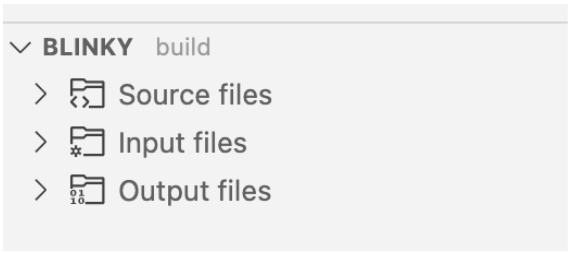
- Click on Create Application.
  - **Windows:** if your username has a space in it, you likely have to put the application in a different location. I made "C:/nrf_apps/" and put my stuff in there.
- You will now see a "blinky" application appear under the Applications tab in nRF Connect.
- Before we 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**. If using the nrf52840 Dongle, select **nrf52840dongle/nrf52840**.
    - Click on 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.

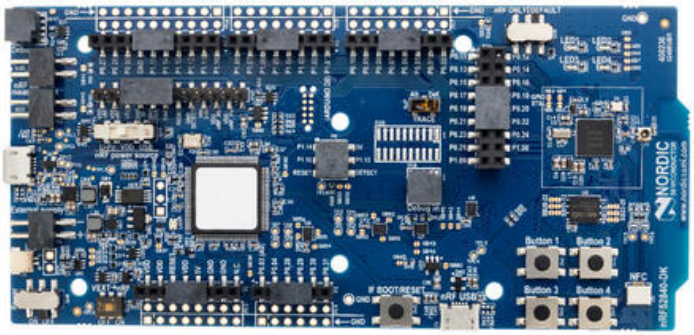


- You will also notice a few new tabs appear below the Application tab on the side bar.
    - In the "BLINKY" tab, you can see the source code for the application. This will change based on the current active application (the one you've picked in the Applications tab).



- We will now flash the application to your board. Be sure to follow the instructions for the board you're trying to flash.

- Flashing **nrf52840DK**
    - Make sure it's the right board. Should look like this:

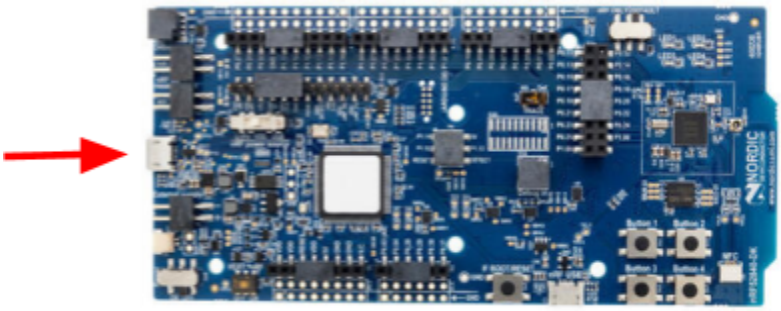


There are a couple of configurations on the nRF52840DK board that you should double-check:

1. The "Power" switch on the top left should be set to "On".
2. The "nRF power source" switch in the top middle of the board should be set to "VDD".
3. The "nRF ONLY | DEFAULT" switch on the bottom right should be set to "DEFAULT".

For now, you should plug one USB cable into the top of the board for programming (NOT into the "nRF USB" port on the side). We'll attach the other USB cable later.

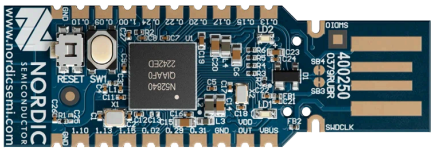
- Connect your board over USB (the one on the narrow end, NOT labeled "nRF USB")



- 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)

- Flashing **nrf52840 Dongle**
    - Make sure it's the right board. Should look like this.

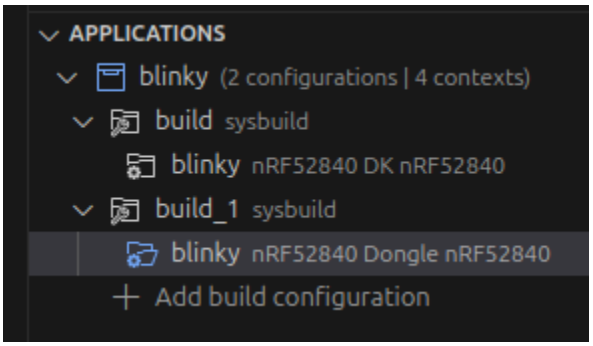


    - Before flashing, make sure you've built the application using the "Build" button in the nRF Connect extension sidebar. Also make sure that you've created a build configuration for "**nrf52840dongle/nrf52840".** See earlier steps in this section if you haven't set up your build configuration yet.
    - Plug it into a USB port.
    - Put the board into DFU mode by clicking the reset button. Note that the reset button doesn't face straight up, it's angled by 90 degrees away from the USB end of the board. **It's not the big white button labeled SW1**. The reset button is the one right next to it.

    - You should now see the board slowly flash with a red LED. If you don't, then you didn't hit the reset button or your board isn't getting any power from your USB port (probably loose).
    - Now we can flash the board. We're going to be following this nRF guide.
    - You'll need **nrfutil**, a new command line tool from Nordic, to flash the board.
    - Download and set it up for your system. Verify you can run it on your system by running something like "nrfutil help".
        - NOTE: If you see an error similar to "command not found" in your shell, you need to add nrfutil to your path so you don't have to specify the global path to use it each time. The error is because Nordic is distributing a binary without an installer, so your shell does not know where to find the command. On Windows, search for "environment variables" in the search menu and there should be a "New" button to add the absolute path to the

file. On Linux/MacOS you can make the download executable with "chmod +x nrfutil" and then move the file to /usr/bin, which should already be on your path. You can double check whether /usr/bin is the correct path for your OS by checking for it in the output of running "echo $PATH" in your shell. There should be a similarly named folder to move nrfutil into.

- You should install the "device", "nrf5sdk-tools", and "toolchain-manager" commands. You can install them by running "nrfutil install <command-name>".
- Once they're installed we need to navigate to the folder with the .hex file we built earlier.
- If we go back to VSCode, we can see the build configurations in our sidebar. We want the one for the dongle, not the DK. In this screenshot, the build files for the dongle are at "builld_1".



- In your terminal, `cd` to your project directory.
- The nRF guide say that the .hex file should be inside the `build/zephyr` directory. (For my example, `build_1/zephyr`). However, on my system I can't find the files there. Instead go to `<build-dir-name>/<project-name>/zephyr`. So for this example where the dongle build directory is `build_1` and the project name is `blinky`, the full path is `build_1/blinky/zephyr`. Once you've found the directory, `cd` into it.
- If you already built the app, you should have a `zephyr.hex` file in this directory.
- Now you can run `nrfutil pkg generate --hw-version 52 --sd-req=0x00 --application zephyr.hex --application-version 1 app.zip`
- And then finally `nrfutil device program --firmware app.zip --traits nordicDfu`

TASK: Make a modification to the blink app and verify you can flash the app and see the change on the board. Copy your modified code or a screenshot. Additionally include a screenshot that flashing the board was successful.

CSE 122 / CSE 222C / WES 269 Wireless Networks
Winter 2025

CSE 122 / CSE 222C / WES 269 Wireless Networks
Winter 2025

Lab

Wireshark Scanning (Week 1)

E. 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://docs-be.nordicsemi.com/bundle/nrfutil_ble_sniffer_pdf/raw/resource/enus/nRF_Sniffer_BLE_UG_v4.0.0.pdf (WARNING: the interface won't appear in Wireshark until you actually program your board, see the instructions below)

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

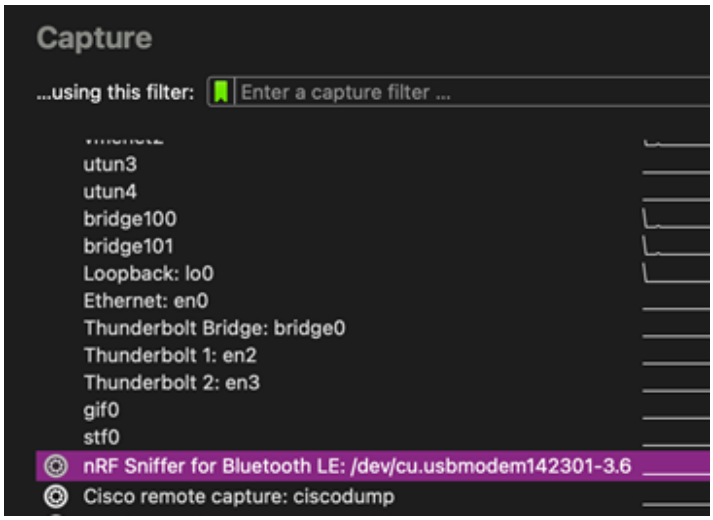
2. The sniffer receiver is written in Python. You'll need Python3 and `pyserial >= 3.5`. If you don't have Python3, follow the python install guide. For pyserial, you can run `python3 -m pip install pyserial` once Python is installed. This *does* work on Windows with a little bit of effort. (Windows users: allowing the installer to add Python to the PATH will make it possible to run Python without typing the full path to the executable.)

3. 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 (Section 2.2 for Revision 4.0.0 of the guide): https://docs-be.nordicsemi.com/bundle/nrfutil_ble_sniffer_pdf/raw/resource/enus/nRF_Sniffer_BLE_UG_v4.0.0.pdf#page=7
   (Note: we've already handled the python requirements from step 1 by installing `pyserial`) *Warning*: The interface still won't appear in Wireshark until you actually program the device in the next step.

4. 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.

5. After reprogramming, re-connect your USB to the *other* USB port (labeled "nRF USB")

6. Either restart Wireshark or go to "Capture Menu -> Refresh Interfaces". You should now (hopefully) see a new capture interface: "nRF Sniffer for Bluetooth LE".



7. Double-click it to start capturing!

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.

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-be.nordicsemi.com/bundle/nrfutil_ble_sniffer_pdf/raw/resource/enus/nRF_Sniffer_BLE_UG_v4.0.0.pdf#page=24

More troubleshooting tips:

1. If Wireshark is unresponsive when you click on the sniffer interface, it is probably due to a permissions issue where your user can't access the board's serial device. On Linux, make sure your user is a part of the "wireshark" and "dialout" groups.

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

TASK: None. Continue to the next section.

F. 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.

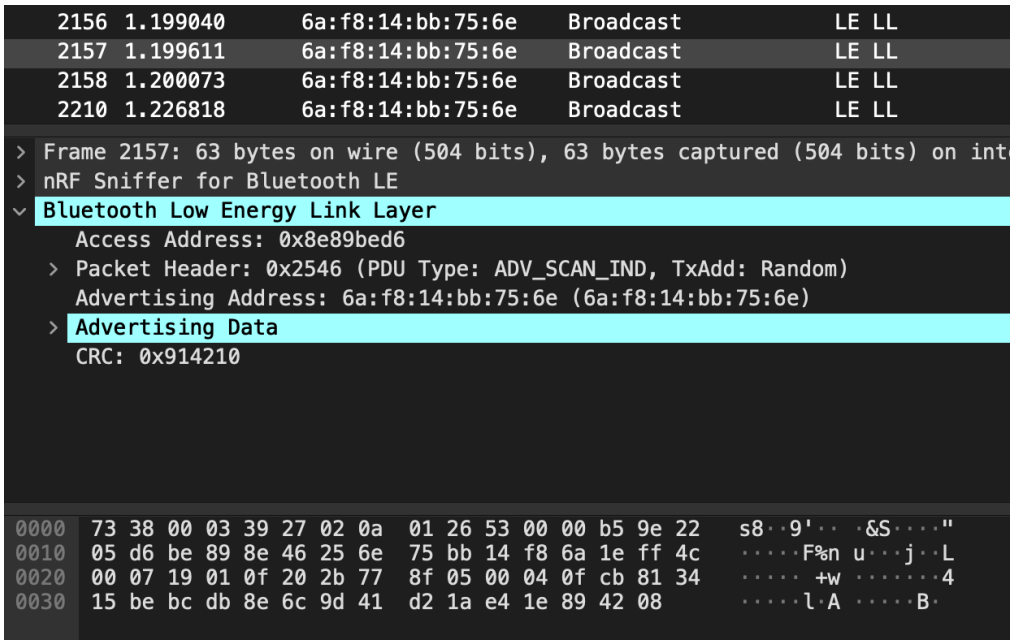
1. **TASK:** How many transmissions do you see in one second?

   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 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 (Week 1)

G. 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.

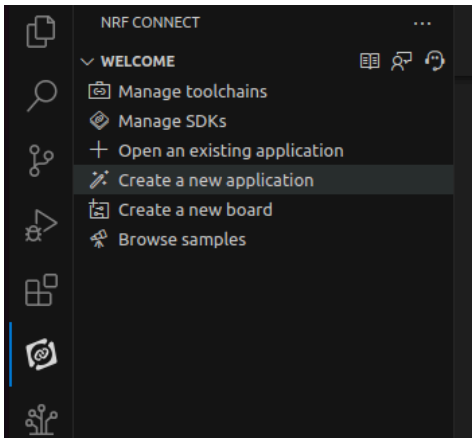
- Click this GitHub Classroom link to create a repo for this lab.
- Pick a team name
  - Unless someone else already started it, in which case, join their team name
- Generally, do what github classroom says
- At the end, it should create a new private repo that you have access to for your code
  - Be sure to commit your code to this repo often during lab!
- 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

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.

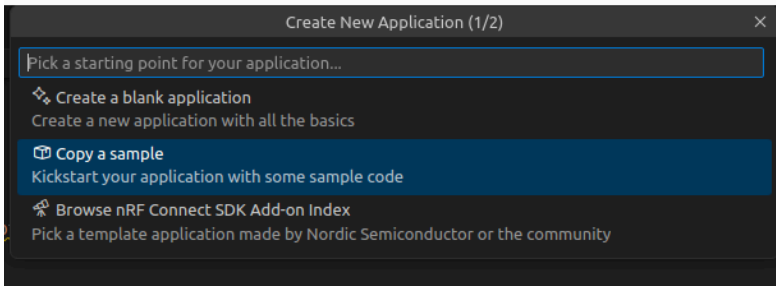
H. Programming a BLE Advertiser

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.

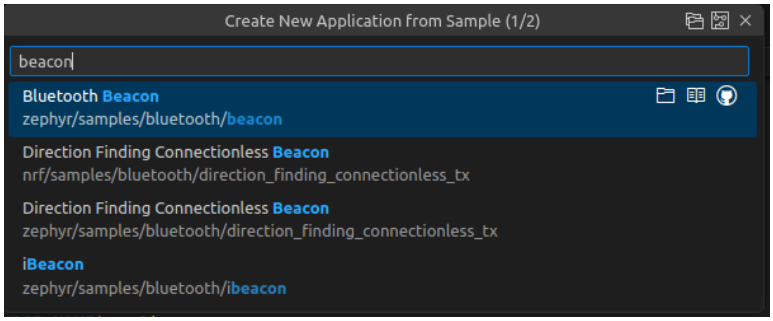
- Let's make a new application from the "Bluetooth Beacon" template. Select "Create a new application" from the sidebar of the NRF Connect extension.



- Select "Copy a sample"



- Pick "Bluetooth Beacon"



- Add a build configuration for it, same as before:
  - Board as **nrf52840dk/nrf52840**
- Build the code, Flash it to the nRF52840DK (make sure that the micro USB is plugged into the port on the shorter side of the board)
  - You can use Wireshark or a phone with the nRF Connect phone app to see that the device exists (it should show up as "Test beacon").

    - 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.
    - **NOTE:** The nrfF52840 dongle can't be viewed in the serial terminal by default. What I recommend doing is getting things working on the nrf52840DK by debugging with print statements. Once you've figured out the kinks, you can deploy the finished version to the dongle. Alternatively, you could debug the dongle by blinking the LEDs instead of with prints. Even more alternatively, you could have the dongle appear as a "virtual USB device" to be able to read/write over serial. See this forum post for more: https://devzone.nordicsemi.com/f/nordic-q-a/59328/using-the-nrf52840-dongle-to-receive-and-display-data-via-the-serial-port-terminal.
    - 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. It should look something like this:

```
*** Booting nRF Connect SDK v2.9.0-7787b2649840 ***
*** Using Zephyr OS v3.7.99-1f8f3dc29142 ***
Starting Beacon Demo
[00:00:00.000,366] <inf> bt_sdc_hci_driver: SoftDevice Controller build revision:
                                            2d 79 a1 c8 6a 40 b7 3c  f6 74 f9 0b 22 d3 c4 80 |-y..j@.< .t.."...
                                            74 72 82 ba                                     |tr..
[00:00:00.002,075] <inf> bt_hci_core: HW Platform: Nordic Semiconductor (0x0002)
[00:00:00.002,105] <inf> bt_hci_core: HW Variant: nRF52x (0x0002)
[00:00:00.002,136] <inf> bt_hci_core: Firmware: Standard Bluetooth controller (0x00) Version 45.41337 Build 3074452168
[00:00:00.002,899] <inf> bt_hci_core: Identity: CD:70:0D:4F:D6:40 (random)
[00:00:00.002,929] <inf> bt_hci_core: HCI: version 6.0 (0x0e) revision 0x106b, manufacturer 0x0059
[00:00:00.002,960] <inf> bt_hci_core: LMP: version 6.0 (0x0e) subver 0x106b
Bluetooth initialized
Beacon started, advertising as CD:70:0D:4F:D6:40 (random)
```

- 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 with one of your other boards acting as the sniffer.

1. **TASK:** prove that you got advertisements working
   - A screenshot from Wireshark or even a phone would be fine here
   - The picture should show the Appearance you set

2. **TASK:** link to your updated and committed code.

CSE 122 / CSE 222C / WES 269 Wireless Networks
Winter 2025

I. 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.

- Create a copy of the “Observer” sample project (similar to how we did for the beacon). Create a build 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 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. **TASK:** prove that you got scanning working.
    - A screenshot of terminal output works here.

2. **TASK:** show that you were able to receive an advertisement from your own advertiser.
    - A screenshot of terminal output works here.

3. **TASK:** demonstrate a scan request and scan response pair for a single device
    - A screenshot from Wireshark is great.
    - Tasks 1-3 could be a single picture if you're skilled enough

4. **TASK:** Provide a link to your committed code on Github

CSE 122 / CSE 222C / WES 269 Wireless Networks
Winter 2025

BLE Peripheral and Central (Week 2)

J. 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.

- We've provided starter code in the "ble-peripheral" folder of this repo: https://github.com/ucsd-wxiot-wi25/ble-week2-starter. Open the "ble-peripheral" folder in VS Code (might be a good idea to copy it over to the git repo you've been using for the previous sections). Configure the build and do all the other normal stuff.
- Change the device name to something unique you can identify (try to keep the name under 10 characters).
- 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 phone app (or any other tool).
- You should see a custom service with UUID "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.

1. **TASK:** document the initial fixed characteristic value.
2. **TASK:** link to your updated and committed code which increments the 32 bit counter on every characteristic read

K. Programming a BLE Central

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

- Create a copy of the “ble-central” sample app in VSCode. Configure the build and do all the other normal stuff.
- 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 discovery 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 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. **TASK:** Show the count from the terminal. Connect multiple times to see the count increase.
2. **TASK:** link to your updated and committed code.

CSE 122 / CSE 222C / WES 269 Wireless Networks
Winter 2025

BLE Service (Week 2)

L. 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 LED(s) 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 the UUID `BDFC9792-8234-405E-AE02-35EF4174B299`.
- It must contain at least one characteristic. The characteristic **must** use 16 bit UUID: `0x0001`. Since we only have one DK, one of your boards (either peripheral or central) must be the dongle. With the dongle as the central, it makes it difficult to control multiple LEDs with one button. With the dongle as the peripheral, we can't control multiple LEDs if there's only one LED.

Characteristic UUID	LED	Length (Bytes)
`0x0001`	LED1	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 Zephyr samples

Central Implementation

- Scan for advertisements with the `BDFC9792-8234-405E-AE02-35EF4174B299` service. 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 Zephyr samples.
- 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. **TASK:** Write a few sentences on what you had to do to make this work. Particularly note anything that was especially challenging to get working.

2. **TASK:** Include your new code for BOTH peripheral and central in your git repo
3. **CHECKOFF:** Showcase the central board controlling the LEDs of the peripheral board to a TA during lab or office hours