Wireless & The IoT

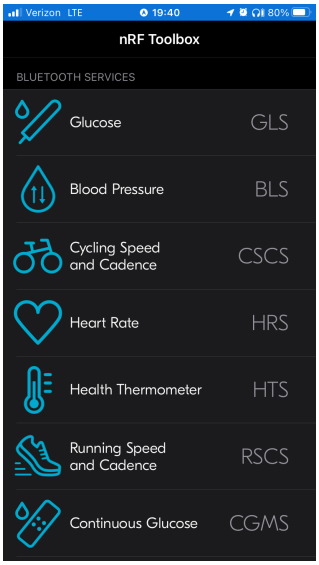
Lab 6: Anyone can be a Heart Rate Monitor, even you!

Introduction

The purpose of today's lab is to play around some with BLE connections and BLE's profiles to understand what really defines devices.

The Assignment

We're going to turn our BLE dongles into various BLE peripherals. On your phone, download and install the nRF Toolbox app. This app expects to connect to various types of peripheral devices. You can see examples in the screenshot below. Pick one that looks interesting, and program your dongle to "be" a glucose monitor, blood pressure monitor, etc.



Giant Caveat to Today's Lab

I had a bunch of difficulties with setting up the firmware correctly and getting the DFU and everything to work reliably at first. I eventually got to a semi-reliable state, but it's pretty easy to accidentally overwrite the DFU and then not be able to re-program the device (without a JTAG programmer). It's a quick fix to recover, but I don't have a ton of JTAGs able to interface with the dongle :/. I'll be on hand during lab to un-brick dongles today, and I'll look into a better solution for next week (quite probably just giving in [$$] and buying a bunch of 52840dk's instead).

What to submit?

Please use this document as a template, add your responses directly, and export it as a PDF to Gradescope. Folks are encouraged to collaborate as much as you like with others. If you work with others, please put everyone's name who worked together below. I believe I have also configured Gradescope to allow "group submission," so please submit to Gradescope as a group.

_____

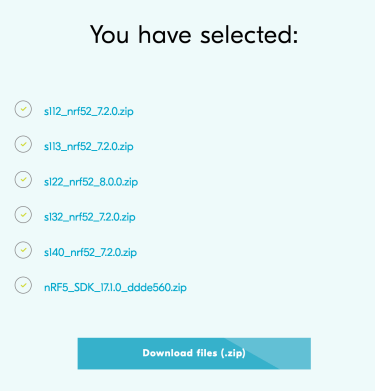
(your name**(S)** here)

Part 0: Setup & Pre-Lab (if you can)

We need to set up our software environment. If you're running into issues, generally we are following the Nordic Getting Started Guide, which may have more details or help for debugging.

First, download the Segger Embedded Studio for ARM (for your platform).

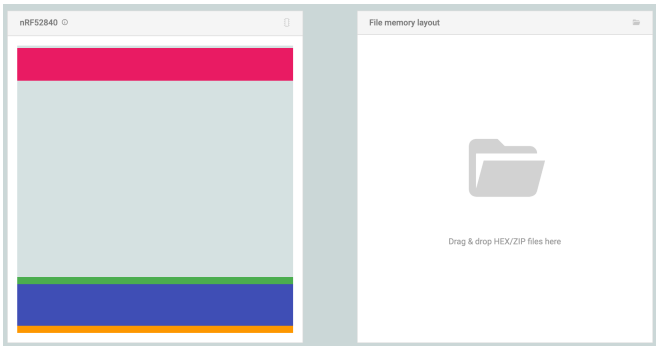
We will also need the Nordic SDK, the default bundle is fine for our purposes. (You can also uncheck older softdevices (<140) for a smaller download).



Finally, we'll be using the Programmer that you used last week.

Creating fused firmware images with the nRF Connect Programmer

With the SDK, we'll be creating applications. However, we will need to stitch together a complete image to program on to the device. When you fire up the programmer, you can see several regions loaded on the existing dongle image, and a blank canvas on your side:



Generally, there a 4 major regions that we care about:

- The gold region at the bottom of memory is the Master Boot Record (MBR) bootloader; it contains the code that will run immediately on reset (i.e. when the device turns on).
- The next interesting region is the red region at the top, this where the DFU (Device Firmware Updater) lives; this is a Nordic-specific implementation of a feature that lets a device rewrite itself without requiring an external debugger (i.e. in our case, it receives new firmware images over the USB <> serial link and writes them to flash).
- The large blue region is the *SoftDevice*, this is another Nordic-specific element, and it's basically their closed-source BLE stack.
- Finally, the green region is application code.

During boot, the MBR bootloader will jump into the SoftDevice, which will set up the radio, interrupts, and other core timing elements, and then will jump to your application. Applications must be at a known, fixed address (0x27000 for SoftDevice v1.4.0) as that's where the SoftDevice expects to find them.

As a special-case, if the device was already powered on and you hit the reset button, the MBR bootloader will jump to the DFU instead of to the SoftDevice. It is *only* when this happens that the dongle will appear as an "Open DFU Bootloader" device accessible to the programmer.

You know the dongle is in DFU mode if the red LED is glowing gently.

After a DFU programming event, the programmer will be default reset the board and jump back to run the newly programmed application (the Auto reset checkbox in the programmer).

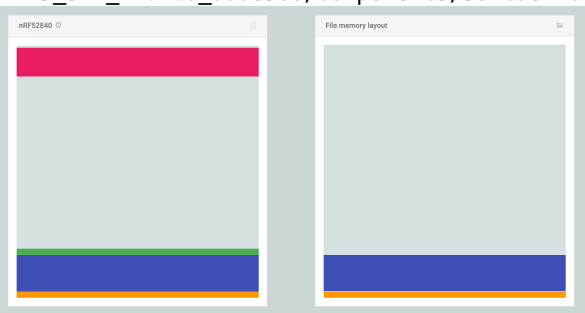
To summarize, the easiest path to reliable states:

- Unplug and replug board → application
- Hit reset button (this is the littler one, with the side press, not the big grey one) → DFU

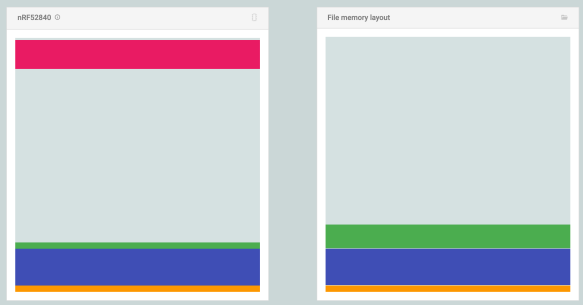
Piecing together firmware images

To program the board, then, we will need to piece together firmware images. The order of operations here can be significant, as the DFU programmer will write out images in the order you added them. ***For this reason, it's important to always add the DFU hex file last.*** This will make sure that if all else fails, you can program something else ☺.

First, if your application does not already include the softdevice in its build process (some of the examples do, some do not), you'll want to add a copy of the softdevice:
`nRF5_SDK_17.1.0_ddde560/components/softdevice/s140/hex/s140_nrf52_7.2.0_softdevice.hex`



Next, you will add your application; the hex image will generally be found here:
`Output/Release/Exe/<yourappname>_s140.hex`



Finally, we need to add the DFU, which is found here:
`nRF5_SDK_17.1.0_ddde560/examples/dfu/open_bootloader/pca10059_usb_debug/hex/open_bootloader_usb_mbr_pca10059_debug.hex`

When you load this image, you'll get some complaints from the tool, these are okay. In particular, notice that this *overwrites* the MBR that came with the SoftDevice; that's because this is what adds the "press reset to enter DFU" feature to the MBR bootloader.

Now you can hit "Write" to upload your program to the device.

```
15:04:05.913
All dfu images have been written to the target device
15:04:08.922
Failed to write: Timeout while waiting for device D379257BC418 to be attached and enumerated
```

When all is done, the programmer will error out, because your application is now running (instead of the DFU) and so there is nothing for it to connect to. This is normal and expected.

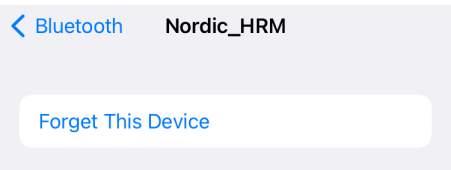
Applications with Segger Embedded Studio

I recommend starting with one of the established projects; it's easier than trying to configure the toolchain and paths and such correctly. Look through the `examples/` folder and find things that have an example for the `pca10059` (this is the nrf52840 dongle). The `.emProject` file in the `ses/` folder of an example will open the example project ready-to-compile.

Once you build (Build menu -> Build Solution), the output application image should be placed in an `Output/Release/Exe/<yourappname>_s140.hex` folder inside the project working directory.

Warning: Cached Services

One issue I ran into was my phone caching the service list. Generally, devices don't change their services, so the spec dictates that centrals can (and should) cache discovered services. You can clear this by forgetting the device, power cycling Bluetooth, and/or power cycling your phone.



Part 1: No really, I'm a bike, I swear.

Drop some screenshots demoing your forged peripheral.

Optimist: If this was super fast / too easy / etc, for some stretch goals:

- Add a timer that changes peripheral values and 'BLE notifies' the Toolbox appropriately.
- Make your device into two different peripherals; can the tool box on one phone see both profiles; can two phone both connect, one for each profile (if not, why not)?

Possible Reality: If you couldn't get this working, look through code in the Nordic examples for how peripherals are configured; can you find the software elements that map to the profiles, services, and characteristics we described in lecture?

Part 2: [hard maybe] Sniffing BLE Connections?

Don't stress too much if this part doesn't work great, give a try for a few minutes, but then bail if needed. Partner up with someone if you haven't already, and try to capture some of the BLE traffic between their phone and peripheral. You should probably be able to capture some of the peripheral advertisements, what data do you recognize here? Can you capture any of the connection handshake or data during the connection (or is this a fool's errand, and if so, why)?

Backup: If Part 1 didn't work out, just answer the question of which parts (advertisements; connection initiation; data during connection) you expect to be able to have seen with your scanner.

