Lab 3 - BLE Connections

Goals

- Set up services and characteristics for a connection
-

Equipment

- Computer with lab toolchain setup
- Three nRF52840DK and cables
- Smartphone iOS or Android (optional but highly useful)

Simple BLE API for Services and Characteristics

Overview: https://github.com/lab11/nrf52x-base/tree/master/lib/simple_ble#connections
Header: https://github.com/lab11/nrf52x-base/blob/master/lib/simple_ble/simple_ble.h#L146

```
// Standard service and characteristic creation
void simple_ble_add_service(simple_ble_service_t* service_char);

/* Add a discoverable characteristic
 *  read: Set to 1 if characteristic is readable, 0 otherwise
 *  write: Set to 1 if characteristic is writeable, 0 otherwise
 *  notify: Set to 1 if characteristic notifies, 0 otherwise
 *  vlen: Set to 1 if characteristic has a variable length, 0 otherwise
 *  len: length of characteristic data buffer
 *  buf: array data buffer
 *  service_handle: service handle this characteristic should be registered under
 *  char_handle: char handle that should be associated with this registration
 */
void simple_ble_add_characteristic(uint8_t read, uint8_t write, uint8_t notify, uint8_t vlen,
                                   uint16_t len, uint8_t* buf,
                                   simple_ble_service_t* service_handle,
                                   simple_ble_char_t* char_handle);

/* Notify a characteristic
 *  char_handle: char to notify
 */
uint32_t simple_ble_notify_char(simple_ble_char_t* char_handle);

/* Returns true if a ble event corresponds to a specific characteristic,
 *  Useful for application logic on characteristic write/read/notify
 *  p_ble_evt: event sent to callback (i.e. on_ble_evt)
 *  char_handle: char handle to check if p_ble_evt is related to
 */
bool simple_ble_is_char_event(ble_evt_t const* p_ble_evt, simple_ble_char_t* char_handle);
```

Additional code is capable of: updating the length of variable-length characteristics and creating characteristics with read/write authorization (which generates the ble_evt_rw_auth() callback, example). You won't need them for this lab though.

Lab Steps

1. Update your repository

- Pull in updates from https://github.com/brghena/nu-wirelessiot-base
    - cd into the base of your forked repo
    - `git pull https://github.com/brghena/nu-wirelessiot-base.git main`
    - Then merge and push to your own forked repo.

2. Service and characteristic example application

You'll need at least two nrf52840DK boards connected today, so might as well start off with both connected now. If you ever have them both advertise, make sure to modify their BLE addresses.

- cd software/apps/ble_service_example/
- `make flash` to build and `make rtt` to view output
    - Note: You will have to choose a board by JTAG serial number

- Now you need a way to connect to the device:
    - The nRF Connect app we used in the last lab has the ability to do so: https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Connect-for-mobile
        - This may or may not work on iOS. It worked for at least two people

    - There is also a version that should work on desktop (using one of your nRF52840DK boards): https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Connect-for-desktop
        - Installation
            - Use wget (or curl) to download the Appimage file
            - chmod +x nrfconnect361x8664.AppImage
            - ./nrfconnect361x8664.AppImage
        - The "Bluetooth Low Energy" app is the one you want.
        - It's a bit finicky from time-to-time, but overall works great.

3. Wireshark (semi-optional)

Wireshark is an open-source tool for analyzing packets. It can be hooked up to various input sources from which it retrieves packets. It then decodes the packets and gives users options to introspect them and filter input data based on them.

Upsides: super valuable in understanding network communication. Downsides: super super touchy and needs constant attention and restarts in order to work successfully. I think Wireshark will be really valuable for some of you as part of your projects, but I'm also a little worried that some people just straight-up won't be able to get this working.

My take is: **spend thirty minutes on this**. Follow the directions for installation. Try it once or twice. If nothing works, you can move on and we'll debug stuff later if you need it for a project.

- sudo apt install wireshark
    - MacOS: brew install --cask wireshark
    - Hit enter to choose defaults (<No>) if prompted

- Program an nRF board with the sniffer app
    - The README explains the JLinkExe command to run. It's the same as for uploading the secret message from the previous lab.
    - After you've programmed it, unplug/replug and reconnect it to your VM
        - Wait about 30 seconds after connecting it to the VM before opening Wireshark. It should appear as a USB device on the left in Ubuntu

- Follow remaining instructions here: https://infocenter.nordicsemi.com/pdf/nRF_Sniffer_BLE_UG_v3.2.pdf
    - All the files you need are in: software/apps/wireshark/
    - Start from "2.2 Installing the nRF Sniffer capture tool"
        - Don't worry that the interface doesn't show up in wireshark yet
    - Also do "2.3 Adding a Wireshark profile for nRF Sniffer"

- Close Wireshark when you are finished
    - Also unplug/replug your device right now to get yourself into a clean state

- Open wireshark
    - You should see "nRF Sniffer for Bluetooth LE: /dev/ttyACM0" in the middle screen (or something like that)
        - If it's not there, close wireshark,try unplugging/replugging your nRF52840DK, and then try again
    - Double click it to begin scanning
    - Click the Red Square on the top left to stop scanning
        - And the Blue Shark Fin to start scanning again
    - If things stop working:
        - Try closing and reopening wireshark

        - Then try closing wireshark, unplug/replug board, opening wireshark
        - Repeat until it works
        - You can check the Log on the top right of the screen. If it at least says something about packets, you're close to it working. If it says something about no device, try reprogramming the nRF.
    - Section "4 nRF Sniffer Usage" explains some details on how to use and understand Wireshark

4. Advanced service example

Lab Requirements:

- At least one service
- At least three characteristics
    - One characteristic must be read and notify (no write). It should notify whenever a button is pressed and change the value based on which button
    - The other two characteristics should control something. LED blinking and printf() statements seem reasonable here.

Tips

- The UUID16 handle of the second characteristic can just be the first handle incremented by one.

- The infinite loop at the end of main doesn't have to call power_manage(). It could instead call nrf_delay and read buttons and write LEDs, like in the first lab. And a global variable could control those LEDs and be written to during button presses.

- You have to manually notify the connected device each time the value of your notifiable characteristic changes. Write the value to the buffer before calling simple_ble_notify_char() to notify. You should be writing its value and calling notify in main(), not in the ble_evt_write() callback.

- Do NOT call nrf_delay_ms() in the ble_evt_write() callback! It's being run in an interrupt context and will not be happy about it.

- iOS may not show you multiple custom characteristics within a custom service. The nRF Connect app for Desktop should be able to though.

5. Environmental sensing service

Let's also make a service and characteristics using a well-known standardized profile: environmental sensing. It must contain the Environmental Sensing Service, and one or more characteristics from a list the standard specifies.

Lab Requirements:

- Create an environmental sensing service with at least five characteristics
    - Starter code in software/apps/ble_ess_example/
- Each characteristic need only be initialized (never updated)
    - But should be given a value that's valid for the environment where you live
- You'll have to connect to it with a nRF Connect app, smartphone or desktop, and display the values of those characteristics (they might be hex instead of meaningful on iOS).
    - Actually, they're going to be hex instead of meaningful even with the nRF Connect for Desktop app. 😡 That's fine.

Environmental Sensing Service requirements:

- Specification (taken from List of all the BLE Specs)
- Most of that document isn't necessary. The important takeaways are:
    - Must use one or more characteristics from table below
    - Optional use of descriptors as metadata to explain measurement conditions

ESS Characteristic requirements:

- Mandatory: Read, Optional: Notify, Disallowed: Write, Vlen
- You can ignore the descriptors
- Possible characteristics: Listed here
  Values from 16-bit UUID Numbers Document

GATT Characteristic and Object Type	0x2A6C	Elevation
GATT Characteristic and Object Type	0x2A6D	Pressure
GATT Characteristic and Object Type	0x2A6E	Temperature
GATT Characteristic and Object Type	0x2A6F	Humidity
GATT Characteristic and Object Type	0x2A70	True Wind Speed
GATT Characteristic and Object Type	0x2A71	True Wind Direction
GATT Characteristic and Object Type	0x2A72	Apparent Wind Speed
GATT Characteristic and Object Type	0x2A73	Apparent Wind Direction
GATT Characteristic and Object Type	0x2A74	Gust Factor
GATT Characteristic and Object Type	0x2A75	Pollen Concentration
GATT Characteristic and Object Type	0x2A76	UV Index
GATT Characteristic and Object Type	0x2A77	Irradiance
GATT Characteristic and Object Type	0x2A78	Rainfall
GATT Characteristic and Object Type	0x2A79	Wind Chill
GATT Characteristic and Object Type	0x2A7A	Heat Index
GATT Characteristic and Object Type	0x2A7B	Dew Point
GATT Characteristic and Object Type	0x2AA3	Barometric Pressure Trend

GATT Characteristic and Object Type	0x2AA0	Magnetic Flux Density - 2D
GATT Characteristic and Object Type	0x2AA1	Magnetic Flux Density - 3D

ESS Characteristic values:

- Defined in the GATT Specification Supplement
    - Be sure to match the name with the table above *exactly.* There are multiple temperature-like characteristics, but only one is named “Temperature”.

- Some characteristics have weird sizes that don’t align to C types (e.g. 24-bits). The simplest method is to use a C type that is larger than it (e.g. 32-bits), and use a bitmask to ensure the value is always using the correct number of bits (e.g. value & 0xFFFFFF).

- Each value will also specify its “Represented Values”. These are explained in the supplement: “Chapter 2 Values and represented values”. Essentially, it’s the multiplier to the raw two’s complement value recorded in the characteristic.

Tips

- You can copy a lot of characteristic setup starter code from the service example app.

- If the characteristic isn’t automatically named in the nRF Connect app, you probably got the two bytes flipped in the characteristic.

- When you read the characteristic value in the app, it should automatically append units and translate into the correctly represented value. For example, it should say "27.00°C” rather than “2700”, although it’s definitely not 27°C in Chicago right now…

- Stupid iOS doesn’t seem to automatically translate characteristic values into proper units and I don’t know why. As long as you get the services and characteristics to appear, you’re fine. Still put a valid value in there, but it’s okay that it’ll just be hex.

Lab Submission

1. If you got wireshark working, show me. If not, give me an explanation of where you’re at in the process and what might be stopping it from working.

2. Explain your advanced service. (Explanation, pointer to code, and screenshot are sufficient)

3. Demonstrate that your environmental sensing app works. A screenshot of nRF Connect reading and interpreting values correctly would be good. If it can't interpret values, that's fine.

4. Did you run into any particular issues?