Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Solar Team Subsystem Manual

MANUAL FOR EXTERNAL COMMUNICATION

Erik Kamph
ekh17001@student.mdh.se

16th February 2021

Contents

1. Summary

This section summarizes the system, by everything from its hardware components to its flow-chars and eventually what security and configuration it has to be able to run.

1.1. System Hardware breakdown structure

1.1.1. Hardware option one

Follow-car	nrf52840 Development Kit
Solar car	nrf52840 Development kit

Table 1: First option is to use two of the same device, as seen above both the solar car and the follow-car uses the nrf52840 development kits. In this case the only thing that can differ might be that the follow-car uses antennas to communicate since the card might be inside the car and not mounted on the outside. The nrf52840 development kit can be seen in figure 1[1]

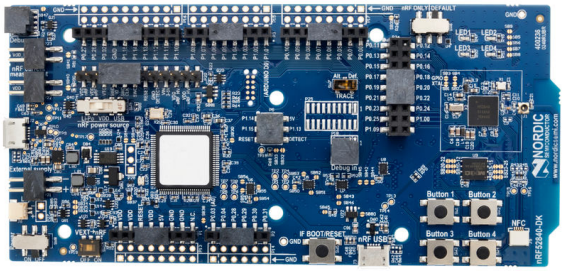

Figure 1: The nrf52840 development kit that utilizes the nrf52840 radio chip and a cortex m4 mcu that programs it.

1.1.2. Hardware option two

Follow-car	USB dongle from nucleo-wb55
Solar car	Nucleo68 from nucleo-wb55

Table 2: Second option is the P-NUCLEO-WB55 kit from STMicroelectronics. The kit has a usb dongle and a expansion card for a nucleo device. In the case of solar team, the expansion board is meant to be placed in the solar car and the usb dongle in the computer that receives the data. The P-NUCLEO-WB55 is seen in figure 2[2]

Figure 2: The nucleo-wb55 kit from stmicroelectronics that has an expansion board and a usb dongle.

1.2. System configuration

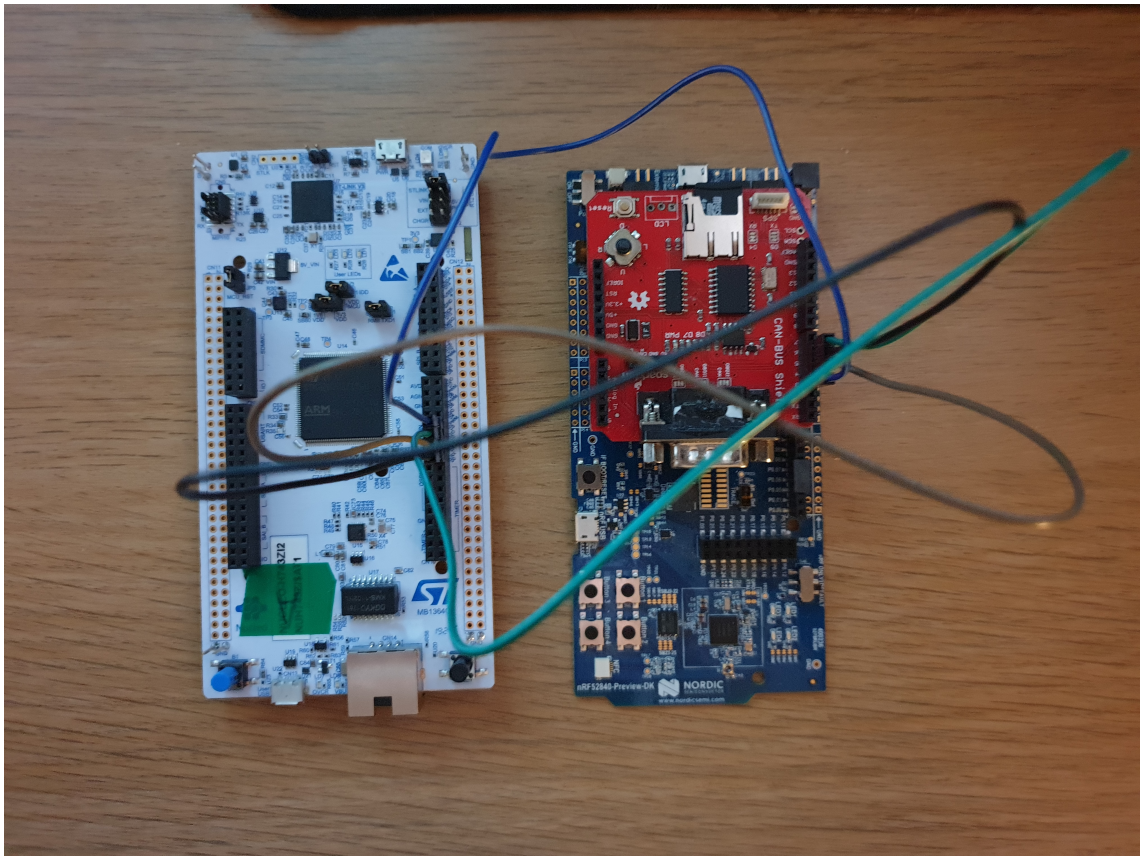

Figure 3: The system configuration above requires one nucleo as master for SPI and one nRF52840 as SPI Slave which receives information from the master and writes it to SD card, as well as wirelessly via bluetooth transmitting it to another nRF52840. The second nRF52840 receives the message and presents it to the user.

1.2.1. Nucleo

Type	Pin
SPI_MOSI	D0
SPI_MISO	D1
SPI_SCK	D2
SPI_CS	D3

Table 3: Pin mapping for SPI communication on the Nucleo seen in figure 3.

1.2.2. nRF52840

The nRF52840 in figure 3 has a CAN-BUS Shield on the same pins that are for an Arduino UNO. On top of that it has pinmapping for SPI communication with the nucleo and they can be seen in table 4

Type	PIN
SPI_MOSI	D1
SPI_MISO	D0
SPI_SCK	D2
SPI_CS	D3

Table 4: Pin mapping for SPI communication on the nRF52840 as seen in figure 3.

1.3. LED Configuration

The current configuration uses the first option that contains two nrf52840 development kits as it has 4 LEDs that we can toggle on or off, while the kit in figure 2 only has 3 LEDs. If using the wb55 kit as seen in figure 2 then the LEDs can be found below under section 1.3.2. As it is right now, the code does not use any of the LEDs, but are supposed to use them for simplicity if looking at the device when testing. These definitions can be found in a file called ”user_config.h” inside a folder called ”lib”.

1.3.1. nRF52840

LED #	State	Rate
1	<<blinking>> Scanning	500ms
2	<<blinking>> Advertising	250ms
3	<<blinking>> Connecting	300ms
	<<on>> Conntected	
4	<<on>> Sending/Receiving	
	<<blinking>> Waiting	400ms
	<<off>> Transmission Done	

Table 5: The led configuration for the nrf52840 development kit

1.3.2. WB55

LED #	State	Rate
1	<<blinking>> Scanning	500ms
	<<on>> Connected	
2	<<blinking>> Advertising	250ms
	<<on>> Sending/Receiving	
	<<off>> Waiting	
3	<<blinking>> Coded PHY	600ms
	<<blinking>> 1M	400ms

Table 6: LED configuration for the WB55 expansion board that only has 3 LEDs.

1.4. Security

Future work includes introducing security to the packages sent over the BLE communication.

1.5. Design Overview

The different diagrams describes the setup more in detail from the system perspective to the code perspecive in form of class diagrams and flow-charts.

1.5.1. Flow-chart for the solar car communication device

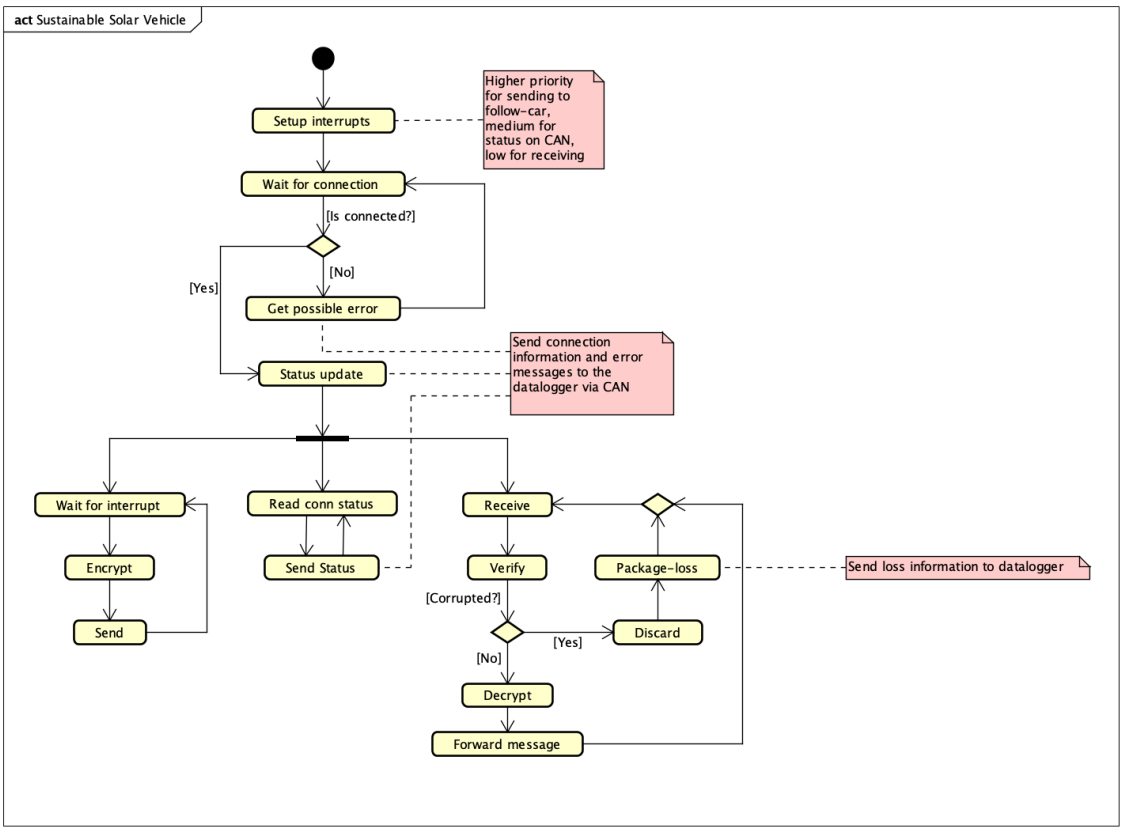

Figure 4: The start is the black dot up top, and then it follows through with setting up the BLE device and connecting it to the other device. In the end it will run an infinite loop that sends status, receives data from other parts of the system and sends it to the other BLE device.

1.5.2. Flow-chart for the follow-car communication device

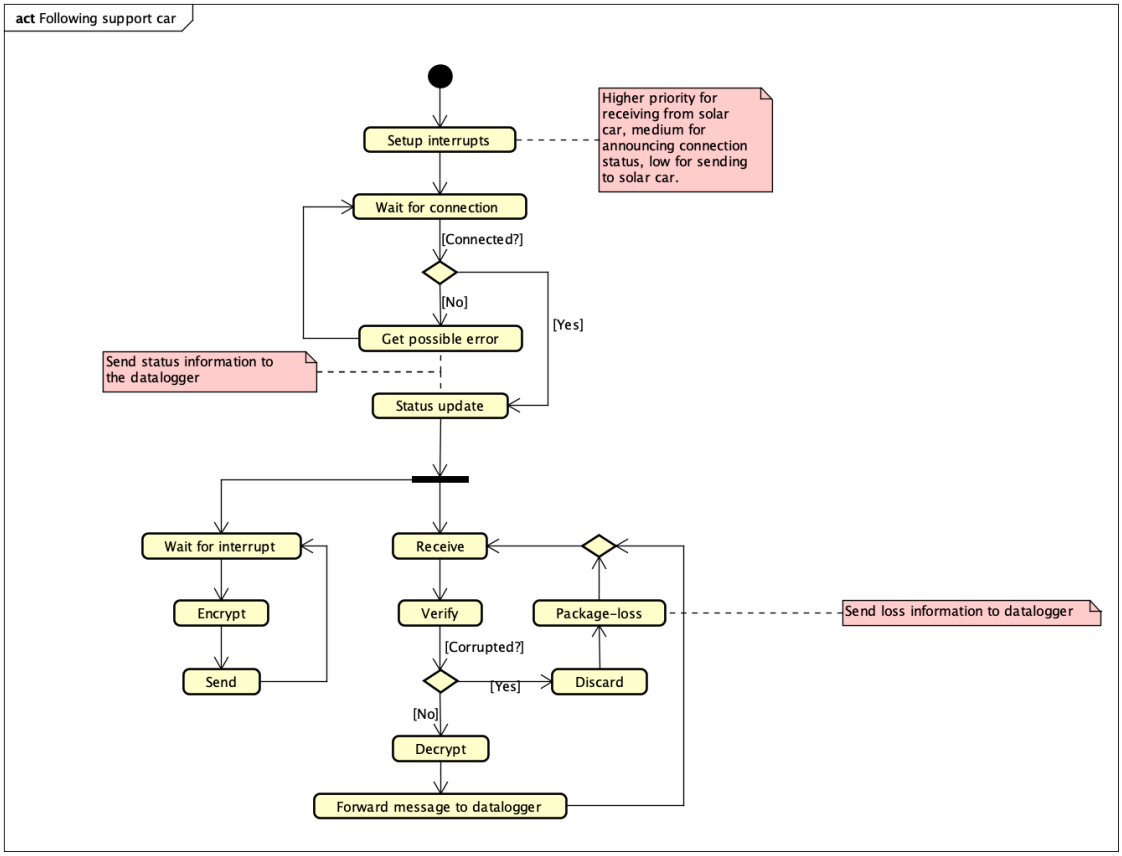

Figure 5: The receiver in the follow-car also sets up the BLE device with the same settings as the sender in the solar car. Moreover, we receive instead of send data in this case and check for corruption. However, corruption, decryption and verification is a part of BLE so that does not need to be re-implemented. Those parts are here because we need to have them to describe the flow of the communication.

1.5.3. Block diagram for external communication

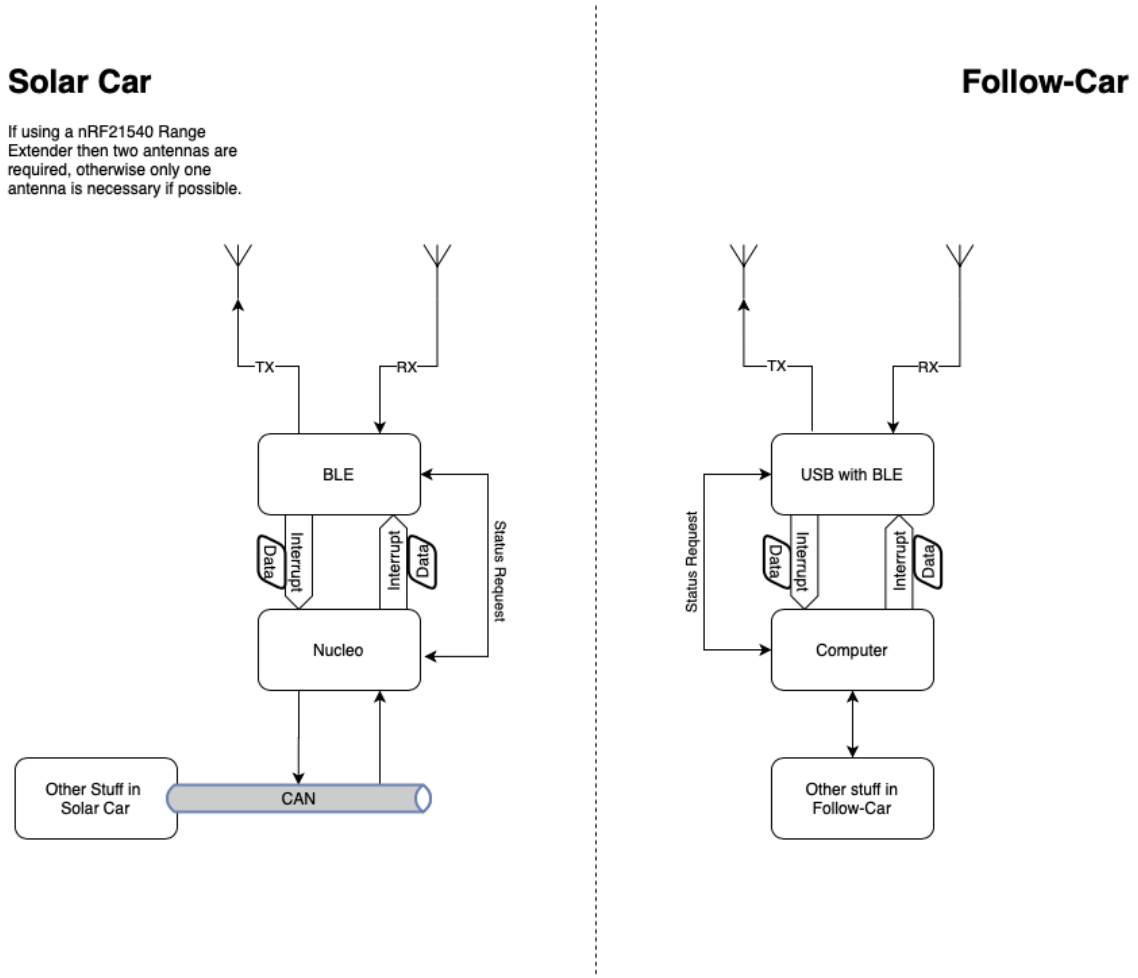

Figure 6: On the left, is how the system is communicating or connected with the other parts of the solar car. On the right is the system for the follow-car, in which case we use a computer instead of a nucleo. The computer is connected to a BLE device, in this case a development kit or a usb dongle that has the same chip for maximum range capabilities. In the follow car there will have to be a card that sends data separate from the nucleo or an expansion card on the nucleo device.

2. Documentation

This serves as a helper to understand how to log, view data, verify that data is being transmitted and how to change baudrate etc. when in field or testing the system over all.

2.1. Changing baudrate

In the file ”user_config.h” found in ”lib”, there is a definition for what the baudrate shall be on. If the code in ”main.cpp” found in ”src” sets the baudrate you will have to change it in ”user_config.h” to some value you’d like. The default value for it is set to 115200 on the line in ”user_config.h” which name is ”BAUDRATE_USER_SETTING”. Check ”main.cpp” in ”src” before changing baudrate, because it might not use Serial, USBSerial or BufferedSerial. If the case is that it does not use any of that, then the second option is to change baudrate in ”platformio.ini” where it says ”monitor_speed”. If it does not work to change baudrate, mbed will default to 9600 as baudrate.

2.2. Logging application

With this development, there is an application or script that logs the data from the nrf52840 development kits or any serial device. To start the script you will first have to go to a folder called ”logger_nrf52840”, the program can be seen in figure 7.

```
Usage: python main.py [optional arguments] [commands] [optional commands]
Version: 1.000000

Optional arguments:
-h, --help                          Display this help dialog
-br in, --baudrate in               Change the default baudrate for information flow on UART bus.
                                    Default: 115200, Accepted values: [300, 600, 1200, 2400, 4800, 9600, 14400,
                                                                       19200, 28800, 31250, 38400, 57600, 115200]
-d device, --device device          Set the path to the device to use
                                    Default: autodetect, Accepts: COM[0-255], /dev/ACM[0-9] or /dev/tty.usbmodem<numbers>
                                    Depending on the OS it might be different
-v, --version                       Display the version of this program
-V, --verbose                       Increase output verbosity, one V or verbose per level.
                                    Highest possible verbose level is 3, which is equal to -VVV

Optional Commands:
scatterplot                         Print a scatter plot with default name: scatter.x.png
                                    Where x is a number from 0 to 9223372036854775807
save path                           Saves the file at the specified path with a specified name
                                    path is the path including the filename

Authors:
Erik Kamph, MDH Solar Team 2020

[V2V][~/GitHub/Team-Embedded/V2V/Software/logger_nRF52840]$
```

Figure 7: Help that is displayed when no argument has been passed onto the logging program

When starting on windows, you will have to find a device that uses the COM-port which is usually found in ”Device Manager” under ”Comports”. There you will see one device which is Bluetooth(which is the computers built-in Bluetooth) and the other is the device you might be searching for which has COM followed by a number from 1 to 255. However if you are on a Linux computer the search path might say ”/dev/ACM” followed by a number. Further, if using macOS it will say ”/dev/tty.usbmodem” followed by either the number that is on the MCU on the board or a generic number such as 141302. Note the path as the logger needs it unless it finds the device automatically.

When choosing baudrate you will have to supply a number after the ”-br” or ”–baudrate” flag, the values that are accepted are those that are in figure 7.

Finally, starting the script on macOS, Linux or Windows can be done with ”python3 main.py -d <device_path> -br <baudrate>”. The script will automatically default to 115200 for baudrate when its not supplied, as well as automatically search for the device on the given OS that the script is executed on.

2.3. Verifying that data is transmitted

To verify that data is being transmitted, open up a terminal and type ”screen”. It is worth noting that the screen command is available on macOS and Linux only. For Windows you will have to download and install a helper program called ”PuTTY”. Once you have typed ”screen” type the device path after a space and follow up with a space. Last type the baudrate that you want to use when connecting using ”screen”-command. See images below. For windows, follow instructions at https://store.chipkin.com/articles/using-putty-for-serial-com-connections-hyperterminal-replacement but skip the introductory text and jump directly to ”To use PuTTY for your serial COM connections, follow these steps”. If everything is done correctly, you should see text as in figure 10.

2.3.1. Terminal with screen command fully typed

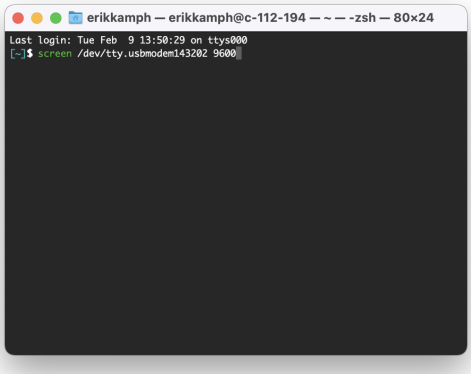

Figure 8: Step one typing the screen command along with device path and possible baudrate. Take a look at figure 9 when choosing baudrate to connect with.

2.3.2. Baudrates for screen

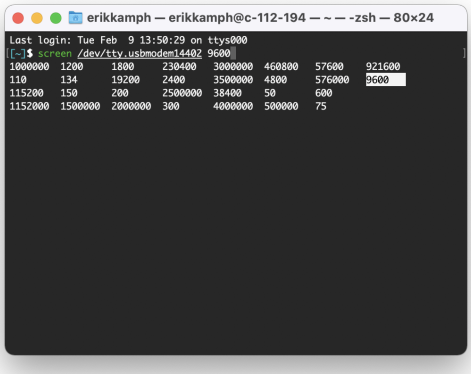

Figure 9: Possible baudrate values for the screen command.

2.3.3. Screen started

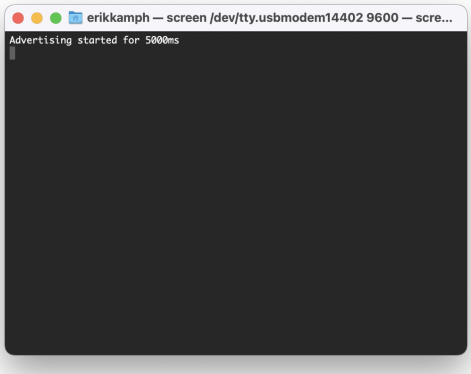

Figure 10: When screen has been started it should look something like this.

2.4. Changing who is sender and receiver

When you want someone to be a scanner (or ”receiver”) then the definition ”DEVICE_SCANNER” in ”user_config.h” in ”lib” needs to be changed to false. However, when you want it to be an advertiser (or ”sender”) it needs to have the definition set to true. The reason for it being reversed logic is that it sets for the one that has true, the value will be changed to false in somewhere in the code and same goes for the receiver. Namely, it reverses the ”is_scanner” variable in the function ”start_role” in ”main.cpp” in ”src”.

2.5. Adding data that needs to be sent

In the code in ”main.cpp” in ”src”-folder, there is a function called ”update_sensor_value” which updates the data that is sent each second. In that function you will have to add the UUID that defines the data as well as the data itself. However when doing so you will have to use the same UUID in a function called ”onPeriodicAdvertisingReport” later on to receive the data on the receiving device. The functions can be seen in the figures below.

```
/* also updates periodic advertising payload */
void update_sensor_value()
{
    /* simulate battery level */
    _battery_level--;
    if (_battery_level < 1) {
        _battery_level = 100;
    }

    /* update the level in the payload */
    ble_error_t error = _adv_data_builder.setServiceData(
        GattService::UUID_BATTERY_SERVICE,
        mbed::make_Span(&_battery_level, 1)
    );

    if (error) {
        return;
    }

    /* the data in the local host buffer has been updated but now
     * we have to update the data in the controller */
    error = _ble.gap().setPeriodicAdvertisingPayload(
        _adv_handle,
        _adv_data_builder.getAdvertisingData()
    );
}
```

Figure 11: update_sensor_value function

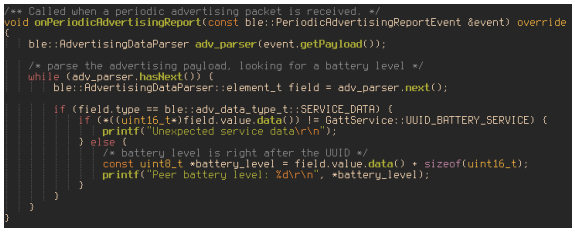

Figure 12: onPeriodicAdvertisingReport function

Be sure not to go over the limit set in ”user_config.h” as ”MAX_ADVERTISING_PAYLOAD_SIZE” is set to 72 for now, with the reason being that by the time of writing we only advertised a faked battery level. However, the maximum advertising size can be set to 255 per payload if necessary which is also maximum of what Bluetooth low energy can handle.

3. Troubleshooting

Known things that can happen when setting up the communication between the solar car and follow-car and its solutions can be found in subsections below. They have the most known problems that is easy to troubleshoot. Bugs come later, after the test results have been presented as how well the device suits solar team.

3.1. No connection between devices

When starting, the devices might not find each other after a few tries. Sometimes it takes a little while before they find each other. But in general, if one of the devices change to advertising and they then find each other, it will be reversed so that the one that’s supposed to receive is advertising.

3.1.1. Solution

There are multiple solutions to this problem and those are:

1. Press the ”IF BOOT/RESET” button that’s next to the ”J3” connector with name ”nRF USB” on both boards to reset the application.
2. Separate the code into a sender and a receiver, like a server/client problem, so that you always know that one will be a sender and the other always a receiver.
3. Make sure to upload the code for the sender onto the board that is advertising data, and not the other way around.

3.2. The data is partial

When sending and you have a look at the receiver, you may notice that the data is partial from what you are sending. The problem here is that you have not changed the size of the package.

3.2.1. Solution

Change the size of the message from it’s current to something bigger at most to 255. The second option is that if you have too much data for one package that you need to split it into equal parts and send each part by itself. If changing the size, then you need to do it in the ”user_config.h” file in ”lib”-folder on the line that says ”MAX_ADVERTISING_PAYLOAD_SIZE”.

3.3. Garbage data is reported when using the logger

If you see garbage text, or text that is not readable from the board. The problem here might be that you selected the wrong baudrate.

3.3.1. Solution

Choose another lower or higher baudrate and keep doing it until the text is readable.

3.4. The logger script can't find the device

If the logger script can't find the device, which happens if you're running on windows, since then it can select the com port named COM1 which is the computers internal Bluetooth and wrong. On Linux, however the problem is less reachable. However on macOS the problem does not exist either. To conclude, this problem might only exist for Windows users.

3.4.1. Solution

Provide a path or com-port like "-d COMXXX" or "-d /dev/tty.XXX" and change the XXX in either case to the real device path provided by the OS. If on Linux or macOS you will have to look in the dev folder under the root of the file-system. While on Windows you will have to look in the device manager after COM-ports and note the port number that is not the internal Bluetooth COM-port

4. Test Results

5. Known Bugs

References

[1] N. Semi, *Nrf52840 development kit*, Accessed: 2021-02-09. [Online]. Available: `https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF52840-DK`.

[2] STMicroelectronics, *P-nucleo-wb55 - stmicroelectronics*, Accessed: 2021-02-09. [Online]. Available: `https://www.st.com/content/st%5C_com/en/products/evaluation-tools/product-evaluation-tools/stm32-nucleo-expansion-boards/p-nucleo-wb55.html%5C#overview`.