NTNU
Norwegian University of
Science and Technology

Department of Engineering Cybernetics

TTK4155 - Embedded and Industrial Computer Systems

Getting started with the BT840

Version 0.0.1 - Aug 2020

1 Introduction

Wireless protocols are becoming increasingly popular in many areas due to its increased ease of use, simple installation and decreased cost. The design project in TTK4155 is no exception; the wireless chip provided give many opportunities for adding cool extras to your project. What about controlling the game with your phone, a Playstation controller or even using the accelerometer and buttons of a *micro::bit*?

To make it simple to connect a wireless interface, the Arduino Due shield has been supplied with a BT840 chip armed with a nRF52840 SoC. Many of the students taking this course is already familiar with the nRF5 series through the *micro::bit* in TTK4235 "Embedded Systems". This miniguide will get you started with programming the BT840 with examples from the Nordic nRF5 SDK, as a first step to programming the nRF52840 SoC yourself.

2 Background information

2.1 Hardware

2.1.1 The nRF52 series

The nRF52 series is a family of ultra low-power wireless Systems on Chips (SoCs) from Nordic Semiconductor. The SoCs are based on ARM Cortex-M4 CPUs. The different SoCs in the nRF52 series have different memory configurations and features, and support different (wireless) protocols. Common for all are SPI, UART and TWI as well as the 2Mbps Bluetooth 5 protocol.

2.1.2 The BT840

BT840 is a powerful low power BLE chip using Nordic nRF52840 SoC, the most powerful in the nRF52 series. With an embedded 2.4GHz multiprotocol transceiver and an integrated PCB trace antenna, there are little extra peripherals needed to get this chip up and running. A SWD-connector is added on the shield for programming, as well as a 32 kHz crystal for more stable low frequency operations and lower average current consumption. The nRF52840 support protocols as Bluetooth Mesh, ANT, Thread, ZigBee, 802.15.4 and more. Some protocols, as NFC, require an extra specialized antenna which is not implemented on the shield. For more information see the BT840 and nRF52840 datasheets. Also, have a look at the shield schematic to see how the BT840 is connected to the Arduino.

2.2 Software

2.2.1 Nordic SoftDevices

A SoftDevice is a wireless protocol stack provided by Nordic that complements an nRF5 Series SoC. Nordic provides them as tested and precompiled binary (hex) files. It is possible to build applications without using a SoftDevice, however it requires more detailed knowledge of the chip being programmed. Different nRF SoCs require different SoftDevices. For the BT840 (and other nRF52840 designs) you can use SoftDevice S140 and S340. For more information on SoftDevices, see Nordic Semiconductors Infocenter (`https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_gsg_ses%2FUG%2Fgsg%2Fsoftdevices.html`)

2.2.2 nRF5 SDK

The nRF5 SDK (Software Development Kit) from Nordic Semiconductor is a development environment with drivers, libraries, peripheral examples, SoftDevices, and proprietary radio protocols for use with the nRF5 SoCs. The provided examples are designed to be run on the nRF5 development kits (nRF5-DK), but can easily be adapted for custom designed boards

Note: All nRF5 SDK example applications with Bluetooth® Low Energy require a SoftDevice. This can be flashed to the SoC by itself or merged with the program hex using `mergehex` *before flashing.*

3 Getting Started

3.1 Initial setup

3.1.1 Toolchain

To compile and flash the code for the BT840, we need to install a toolchain. To compile the code, GCC (`arm-none-eabi-gcc`) is a good choice. For flashing the device we will be using `openocd` which enable us to use the already available Atmel ICE. For those of you who have a j-link from Segger (or a nRF5-DK) you can use `nrfjprog`.

Toolchain for linux

1. Call `sudo apt install gcc-arm-none-eabi` to install the compiler
2. Install prerequisites for a newer version of openocd with `sudo apt install libusb-dev libusb-1.0-0-dev libusb-1.0-0`
3. Open-jtag also requires `sudo apt install libhidapi-dev libftdi-dev libftdi1-dev`
4. Download openocd `git clone https://git.code.sf.net/p/openocd/code openocd`
5. Go into the folder `cd openocd`
6. Create the configure script with `./bootstrap`
7. Create and go into a directory for the build `mkdir build; cd build`
8. Configure the build `../configure --enable-cmsis-dap --enable-openjtag --prefix=/opt/openocd`
9. Make and install with `make`, then `make install`
10. Install nrfutil with `pip install nrfutil` and check that it works with `nrfutil --help`. If not you might add it to PATH. Run `export PATH=$PATH:/home/student/.local/bin/` if nrfutil is located there.

Toolchain for windows

1. Download `arm-none-eabi-gcc` from `https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads` and `Make for windows` from `http://gnuwin32.sourceforge.net/packages/make.htm`. Alternatively, you can use Segger Embedded Studio for ARM `https://www.nordicsemi.com/Software-and-tools/Development-Tools/Segger-Embedded-Studio/Download`.
2. Download the latest openOCD version from `https://gnutoolchains.com/arm-eabi/openocd/` and add it to your PATH.
3. Install nrfutil with `pip install nrfutil` and check that it works with `nrfutil --help`. If not you might add it to PATH.

3.1.2 Software

We will get started with the BT840 by using one of the examples from the nRF5 SDK. Download the SDK v.17.0.0 (other versions may also work) from `https://www.nordicsemi.com/Software-and-tools/Software/nRF5-SDK` and extract the zip at your desired location.

3.2 Code adaption

As mentioned earlier, the nRF5 SDKs are tailored for use with the nRF5 development kits. To make an example work with our hardware setup of the BT840 there might be necessary to make changes in the code. As an start we will have a look at the ble_app_uart example which forward characters between the uart and bluetooth interface. *Note that you also have to setup the correct USART channel on the Arduino Due in order to send and receive messages.* In the SDK go to examples → ble_peripheral → ble_app_uart. Open the main.c file in your favourite text-editor and change the configuration in `uart_init()` to fit the Arduino connections. You can disable the rts and cts pins by setting them to `0xffffffff`. Remember to also set the correct baudrate.

3.3 Compiling the code

You are now ready to compile the code. You can do this by going to `pca10056`→ `s140`→ `armgcc` in a terminal. Use `make` to compile the code. You may have to define the location where `arm-none-eabi-gcc` is located on the PC in the file Makefile.posix/Makefile.windows. This file can be found in `SDK`→ `components`→ `toolchain`→ `gcc`
For Segger Studio, go to `pca10056`→ `s140`→ `SeS` and open the Segger Embedded Studio Project. Select `build`→ `build solution`. The compiled code will appear in the `_build` directory.

3.4 Bootloder

The bootloader is responsible for booting into the application, activating new firmware, feeding the watchdog timer and optionally, entering DFU mode. There should already be a working bootloader on the chip. If for some reason this bootloader is not serving your purpose you can use nrfutil to compile another. A bootloader can also be compiled thorough GCC.

Make the bootloader by running
`nrfutil settings generate --family NRF52840 --application _bulid/ble_app_uart_pca10056_s140.hex --application-version 3 --bootloader-version 2 --bl-settings-version 2 _bulid/bootloader.hex`. This will create the bootloader in the `_build` folder.

3.5 Flashing the code to BT840

The premade Makefile will work if you have a j-link or nRF5-DK to flash the BT840. If not, you will need to change some of its content to use Atmel-ICE. Replace the `nrfjprog` commands with the appropriate `openocd` commands. Se Figure 1. Be aware that Makefiles makes a difference between four spaces and a TAB. When copying or typing the code in Figure 1 make sure the indentations become TABs. In particular, Visual Studio Code has a tendency to convert TABS to spaces, use VIM to be sure TABs have been inserted.
Make sure you have made the required changes in the Makefile and added the openocd.cfg file to your porject. Connect the SWD-pins (*SDWIO,SWDCLK, GND, 3.3V*) to the connector on the shield as described in the Atmel-ICE userguide. You may use the squid-cable from Node 1 or another way you find practical. Flash the code with `make flash`. You might also need to flash the SoftDevice with `make flash_softdevice` and the bootloader with `make flash_bootloader`. For debugging run `make debug`. Be aware that the Bluetooth part will behave strange during debugging. So stepping through the code will work best for the parts not reliant on the Bluetooth.

```
.PHONY: flash flash_softdevice flash_bootloader erase debug

# Flash program
flash:
    @echo Flashing: $(OUTPUT_DIRECTORY)/nrf52840_xxaa.hex
    /opt/openocd/bin/openocd -f openocd.cfg -c "program $(OUTPUT_DIRECTORY)/nrf52840_xxaa.hex verify reset exit"

# Flash softdevice
flash_softdevice:
    @echo Flashing: s140_nrf52_7.0.1_softdevice.hex
    /opt/openocd/bin/openocd -f openocd.cfg -c "program $(SDK_ROOT)/components/softdevice/s140/hex/s140_nrf52_7.0.1_softdevice.hex verify reset exit"

# Flash bootloader
flash_bootloader:
    @echo Flashing: $(OUTPUT_DIRECTORY)/bootloader.hex
    /opt/openocd/bin/openocd -f openocd.cfg -c "program $(OUTPUT_DIRECTORY)/bootloader.hex verify reset exit"

# erase
erase:
    @echo Erasing device
    /opt/openocd/bin/openocd -f openocd.cfg -c init -c halt -c "nrf5 mass_erase"

# Debug device
debug:
    if pgrep /opt/openocd/bin/openocd; then pkill /opt/openocd/bin/openocd; fi
    mergehex -m $(SDK_ROOT)/components/softdevice/s140/hex/s140_nrf52_7.0.1_softdevice.hex _build/nrf52840_xxaa.hex -o out.hex
    x-terminal-emulator -e /opt/openocd/bin/openocd -f openocd.cfg -c "program out.hex verify"
    sleep 10
    arm-eabi-gdb -tui -iex "target extended-remote localhost:3333" _build/nrf52840_xxaa.out
    killall -s 9 /opt/openocd/bin/openocd
```

Figure 1: Makefile changes for Atmel-ICE

```
# Atmel-ICE JTAG/SWD in-circuit debugger.
interface cmsis-dap
#cmsis_dap_vid_pid 0x03eb 0x2141

#Specify Atmel-ICE serial number if using multiple at the same time
#cmsis_dap_serial J41800035666

# Chip info
#set CHIPNAME nrf52
source [find target/nrf52.cfg]
```

Figure 2: openocd.cfg

4 Check that it works

You can check that the code have been transfered to the BT840 and that it works by using Nordics nRF Toolbox app. The app is available on Google Play (`https://play.google.com/store/apps/details?id=no.nordicsemi.android.nrftoolbox`) and App Store (`https://apps.apple.com/us/app/nrf-toolbox/id820906058`).
Open the app and go to UART → connect. The device should appear in the list as *Nordic_Uart* or the `DEVICE_NAME` you have specified. Press connect, scroll left and try writing something to it.

What you write should be sent to the Arduino Due over UART.

5 Moving on

You are free to experiment with the examples and SoftDevice provided by Nordic. If you want to connect the *micro::bit* via the bluetooth interface, you are recommended to check out the ”Introducing Bluetooth Low Energy” exercise by Kolbjørn Austreng. The exercise is available on Blackbord. This also applies if you want to program the BT840 without using the nRF5 SDK examples, but note that it require a different SoftDevice and therefore some adaptions. Remember that when connecting two devices over Bluetooth, one has to function as central, the other as peripheral. Since the nRF52840 is more powerful than the *micro::bit*, it is recommended to use it as central. The micro::bit has an embedded J-link, so you can use `nrfjprog` to program the device. You might have to upload a hex file, see *Blackbord* → *Lab support data* → *Extras* for more information.

Some gotchas for using the SDK:

- The `pca100xx` folder includes a `sdk_config.h` file. Here features like TWI or UART can be enabled or disabled. Strange behaviour will appear if you forget to enable features you are using.
- The *micro::bit* uses a nRF51822. The SDK fitting this chip is SDK 12.3, softdevices S130 or S110. S110 can only function as a peripheral BLE device, while S130 can be both a central and peripheral device. Consider which you need for your application. This might be useful: `https://devzone.nordicsemi.com/nordic/short-range-guides/b/getting-started/posts/adjustment-of-ram-and-flash-memory`.
- Most of the examples in the SDK 12.3 are written for a nRF51422 with 256kB flash and 32kB RAM. The *micro::bit* has 256kB flash and a 16kB RAM. Because of this some things need to be changed in the linker script (the .ld file found in the `armgcc` folder). Have a look at the RAM origin and the length.
- The examples are written for a board with an external 32.768kHz clock for the Nordic chip. The *micro::bit* does not have this, so an adjustment must be made in the `ble_stack_init` function for the examples using the BLE.
- For Nordics UART library to function with `printf`, retargeting of the stdio functions need to be enabled. This can be done by setting the `#define RETARGET_ENABLED` to 1 in the `sdk_config.h` file.
- The compare files function of Visual Studio Code is a handy function for making sure that you remember to move everything you need from one example to your code.
- Remember to add the additional .c files you use in the Makefile, as well as the path to the corresponding header files.
- The cflags given by the line `CFLAGS += -Wall -Werror -O3 -g3` in the Makefile may be giving you trouble if you write sloppy code. Remove at your own risk.

Good luck!