# nRF53 comprehensive example

This is a comprehensive example. It contains the following child examples.

1. BLE example. Both Bluetooth LE host and controller run on network core. In this example, NUS(Nordic UART Service) service and SMP(Simple Management Protocol) service are created and run on network core instead of application core(By default, BLE host and services run on application core). The key kconfig or macro of this example is CONFIG_BLE_NETWORK_CORE.
2. Dual core interaction example. Application core and network core can communicate with each other. There are 2 ways to achieve the interactions: calling nrf_rpc APIs directly to communicate like a 'dual-core UART' or encapsulating nrf_rpc with cbor coding to communicate like remote procedure call(RPC). In this example, the first method is controlled by macro: CONFIG_RPC_SIMULATE_UART. The later one is identified by marco: CONFIG_RPC_REMOTE_API. These 2 kconfig are not compatible. You can only select one of them. Please note that DFU example below uses dual core communication too. To make DFU work, you must select CONFIG_RPC_REMOTE_API.
3. DFU example. This example supports updating both application core and network core images. The new images can be uploaded to the device by means of BLE or serial protocol. Both the new application core image or new network core images can be stored on internal Flash or external QSPI Flash during upgrading process. By default, we use SMP(Simple Management Protocol) protocol to handle the image management.  CONFIG_EXAMPLE_DFU_OTA is used to control the DFU example. To store the new images on external Flash, the following configurations are required:
    a. CONFIG_DFU_EXTERNAL_FLASH=y , CONFIG_PM_EXTERNAL_FLASH=y, etc
    b. CONFIG_NORDIC_QSPI_NOR=y
    c. pm_static_external_flash.yml
4. High speed UART example. By this example, you can achieve 1Mbps baud rate. UART has 3 working mode: poll, interrupt and asynchronize. To achieve the high speed UART, asyn mode must be used.
5. SPI master example. This example shows how to call Zephyr-related SPI APIs to communicate with a SPI slave. The SPI slave image can be directly obtained from nRF5_SDK\examples\peripheral\spis.
6. I2C master example. This example shows how to use Zephyr-related I2C APIs to communicate with a I2C slave. The I2C slave image can be directly obtained from nRF5_SDK\examples\peripheral\twi_master_with_twis_slave.
7. ADC example. ADC has 2 working modes: sync and async mode. And it can sample many channels simultaneously. This example samples 2 channels together, and work in both sync and async mode.
8. Flash access example. There are 3 layers(sets) of Flash access APIs in NCS: Flash area API, NVS API and Settings API. The bottom layer is Flash area API which access Flash directly without additional headers or tails. NVS API invokes Flash area API to achieve the Flash access purpose. To have a better reliability and readability, NVS would add some additional info at the end of a

page. Settings API calls NVS API to access Flash memory. Thus, Settings module has a further encapsulation of raw serialized data. All data is managed by key/value pair in Settings module.

9. Raw nrfx driver example. Many users want to invoke nrfx drivers API directly so that they can skip Zephyr layers to speed up the access or not to use kconfig or deviceTree to have a back compatibility of his old projects. This example shows how to call SPI and RTC bottom layer driver API directly without the awareness of Zephyr RTOS.
10. Kconfig example. In this example, we have prj.conf or similar to configure the parent image(app) and the child images(MCUBoot and B0n).
11. DeviceTree example. In nrf5340dk_nrf5340_cpuapp.overlay, we demonstrate how to delete a Zephyr node, how to delete a property of a node, how to adjust RAM partitions in app image.
12. Memory Partition example. In pm_static_external_flash.yml, we show how to adjust Flash partitions for each image, how to allocate RAM usage for a shared area between 2 images. To make pm_static_external_flash.yml to be effective, we use build system variable in CMakeLists.txt.
13. IO assignments between 2 cores. By default Button4 is assigned to application core. By combination of overlay file and C code, we assign Button4 to network core.
14. External interrupt example. We have 2 external interrupt examples. One is on application core. The other is on network core. By reading the code, you would find API usage on network core is just the same as that of application core.
15. Device power management (PM) example. We can use PM to turn on/off peripherals dynamically to save power consumption.

**Apart from dual core interaction, the rest examples can apply to nRF52 series and nRF91 series too.**

# Contents

nRF53 comprehensive example ........ 1

1. Preparations ........ 4

2. Build ........ 5

3. Testing ........ 7

3.1 BLE NUS example ........ 8

3.2 DFU example ........ 8

3.3 Dual core interaction example ........ 14

3.4 High speed UART example ........ 20

3.5 SPI example ........ 21

3.6 I2C example ........ 21

3.7 ADC example ........ 22

3.8 External Interrupt example ........ 23

3.9 Flash access example ........ 24

3.10 Raw nrfx driver example ........ 27

3.11 Device power management (PM) example ........ 28

3.12 kconfig example ........ 28

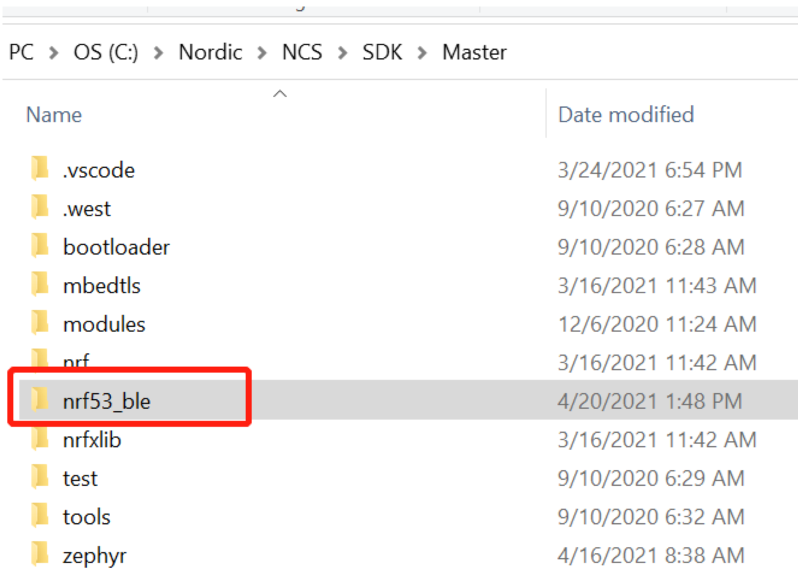
3.13 DeviceTree example ........ 30

3.14 Static memory partition example ........ 32

3.15 IO assignment between 2 cores ........ 32

## 1. Preparations

Unzip the package to the same folder as Zephyr repo like below:




Change nrf\samples\CMakeLists.txt like below. you can refer to nrf53_ble/resources/1CMakeLists.txt for the full file and nrf53_ble/resources/1cmake.diff for the diff file.

```
-if (CONFIG_BT_RPMSG_NRF53)
+if (CONFIG_BT_RPMSG_NRF53 OR CONFIG_BLE_NETWORK_CORE)
  if (CONFIG_SOC_NRF5340_CPUAPP)

    if (CONFIG_NRF_802154_SER_HOST)
@@ -78,9 +78,15 @@ if (CONFIG_BT_RPMSG_NRF53)
      "CONFIG_BT_RPMSG_NRF53 and CONFIG_NRF_802154_SER_HOST are set to 'y'")
    else()
      set(NETCORE_IMAGE "hci_rpmsg")
-     set(NETCORE_IMAGE_PATH "${ZEPHYR_BASE}/samples/bluetooth/${NETCORE_IMAGE}")
-     message("Adding 'hci_rpmsg' firmware as child image since "
-     "CONFIG_BT_RPMSG_NRF53 is set to 'y'")
+     if (CONFIG_BLE_NETWORK_CORE)
+       set(NETCORE_IMAGE_PATH "${ZEPHYR_BASE}/../nrf53_ble/ble_netcore")
+       message("Adding 'ble_netcore' firmware as child image since "
+       "CONFIG_BLE_NETWORK_CORE is set to 'y'")
+     else()
+       set(NETCORE_IMAGE_PATH "${ZEPHYR_BASE}/samples/bluetooth/${NETCORE_IMAGE}")
+       message("Adding 'hci_rpmsg' firmware as child image since "
+       "CONFIG_BT_RPMSG_NRF53 is set to 'y'")
+     endif()
    endif()
```

Change nrf/modules/mcuboot/CMakeLists.txt like below. you can refer to nrf53_ble/resources/2CMakeLists.txt for the full file and nrf53_ble/resources/2cmake.diff for the diff file.

```
@@ -236,7 +236,7 @@ if(CONFIG_BOOTLOADER_MCUBOOT)
      )

    if (CONFIG_NRF53_UPGRADE_NETWORK_CORE
-         AND CONFIG_HCI_RPMSG_BUILD_STRATEGY_FROM_SOURCE)
+         AND (CONFIG_HCI_RPMSG_BUILD_STRATEGY_FROM_SOURCE OR CONFIG_BLE_NETWORK_CORE))
      # Network core application updates are enabled.
```

Change nrf\cmake\partition_manager.cmake like below. you can refer to nrf53_ble/resources/3partition_manager.cmake for the full file and nrf53_ble/resources/3pm.diff for the diff file.

```
     endforeach()

     if (CONFIG_NRF53_UPGRADE_NETWORK_CORE
-          AND CONFIG_HCI_RPMSG_BUILD_STRATEGY_FROM_SOURCE)
+          AND (CONFIG_HCI_RPMSG_BUILD_STRATEGY_FROM_SOURCE OR CONFIG_BLE_NETWORK_CORE))
          # Create symbols for the offset reqired for moving the signed network
```

## 2. Build

The example supports SES, CMD or other terminals. Here we use CMD for the building process.

Type in the following command to enter nrf53_ble\appcore

***cd C:\Nordic\NCS\SDK\Master\nrf53_ble\appcore***

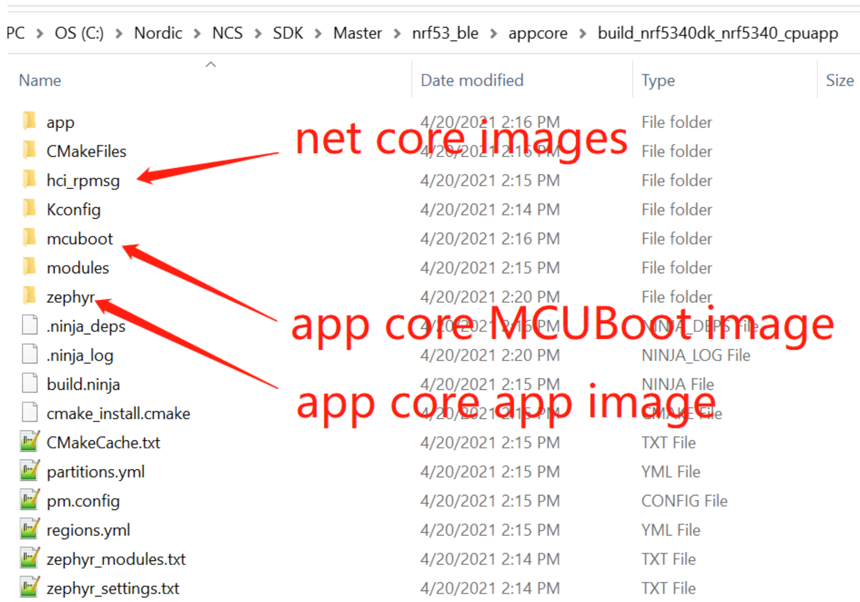
Type in the following command to build the project. After this command, we would get 4 images: app image and MCUBoot image on application core, app image and B0n image on network core.

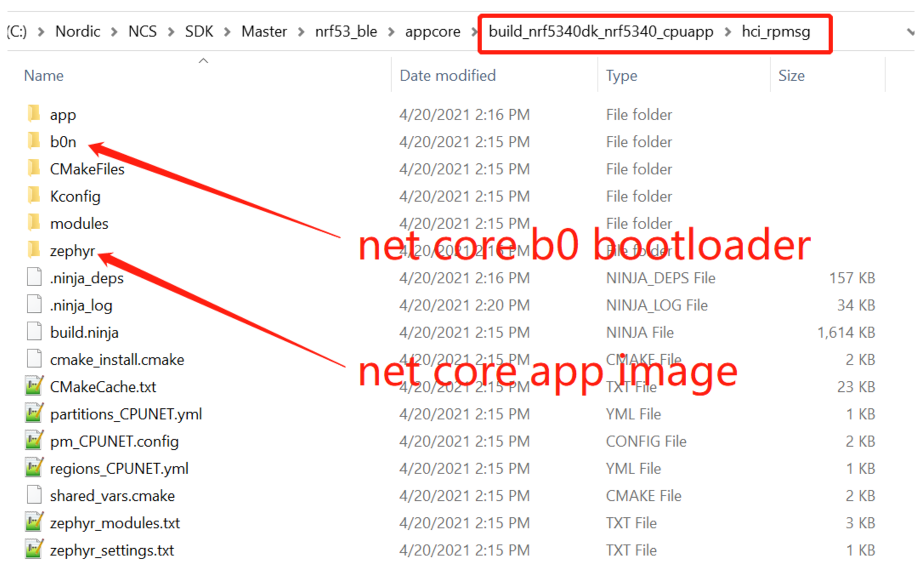
***west build -b nrf5340dk_nrf5340_cpuapp -d build_nrf5340dk_nrf5340_cpuapp -p***

we get the following output after the previous two commands:

```
C:\Nordic\NCS\SDK\Master\nrf\samples>cd C:\Nordic\NCS\SDK\Master\nrf53_ble\appcore

C:\Nordic\NCS\SDK\Master\nrf53_ble\appcore>west build -b nrf5340dk_nrf5340_cpuapp -d build_nrf5340dk_nrf5340_cpuapp
-- west build: generating a build system
Including boilerplate (Zephyr base): C:/Nordic/NCS/SDK/Master/zephyr/cmake/app/boilerplate.cmake
-- Application: C:/Nordic/NCS/SDK/Master/nrf53_ble/appcore
-- Zephyr version: 2.4.99 (C:/Nordic/NCS/SDK/Master/zephyr)
-- Found Python3: C:/Python38/python.exe (found suitable exact version "3.8.5") found components: Interpreter
-- Found west (found suitable version "0.9.0", minimum required is "0.7.1")
-- Board: nrf5340dk_nrf5340_cpuapp
-- Cache files will be written to: C:/Nordic/NCS/SDK/Master/zephyr/.cache
-- Found dtc: C:/ProgramData/chocolatey/bin/dtc.exe (found suitable version "1.5.0", minimum required is "1.4.6")
```

As mentioned before, we will have 4 build directories for each image like below:




Type in the following command to download the 4 images to nRF53 by one go.

***west flash -d build_nrf5340dk_nrf5340_cpuapp***

we get the following output:

```
C:\Nordic\NCS\SDK\Master\nrf53_ble\appcore>west flash -d build_nrf5340dk_nrf5340_cpuapp
-- west flash: rebuilding
[0/9] Performing build step for 'mcuboot_subimage'
ninja: no work to do.
[1/6] Performing build step for 'hci_rpmsg_subimage'
[0/5] Performing build step for 'b0n_subimage'
ninja: no work to do.
[2/3] cmd.exe /C "cd /D C:\Nordic\NCS\SDK\Master\nrf53_ble\appcore\build_nrf5340dk_nrf5340_cpuapp\zeph

-- west flash: using runner nrfjprog
Using board 960193652
-- runners.nrfjprog: Flashing file: C:\Nordic\NCS\SDK\Master\nrf53_ble\appcore\build_nrf5340dk_nrf534
-- runners.nrfjprog: Generating CP_NETWORK hex file C:\Nordic\NCS\SDK\Master\nrf53_ble\appcore\build_r
-- runners.nrfjprog: Generating CP_APPLICATION hex file C:\Nordic\NCS\SDK\Master\nrf53_ble\appcore\bu
ains.hex
Parsing image file.
Erasing page at address 0x1000000.
Erasing page at address 0x1000800.
Erasing page at address 0x1001000.
Erasing page at address 0x1001800.
Erasing page at address 0x1002000.
Erasing page at address 0x1002800.
```

## 3. Testing

After downloading, the board would advertise “nus_netcore”. Netcore would output the following logging:

```
J-Link RTT Viewer V6.88a
File  Terminals  Input  Logging  Help
All Terminals  Terminal 0  Terminal 1
00> [00000026] <inf> rp_ll_api: Handshake done
00> [00000034] <inf> peripheral_uart: ### netcore firmware compiled at 17:39:43 May  3 2021
00>
00> [00000142] <inf> peripheral_uart: Bluetooth initialized
00> [00000175] <inf> peripheral_uart: Started NUS example on netcore
```

Regarding app core, we support both UART and RTT backend for the logging. By default, we use UART0 for the logging. After reset, we would get the following output:

```
*** Booting Zephyr OS build v2.4.99-ncs2  ***
I: Starting bootloader
I: Primary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Secondary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Boot source: none
= secondary slot in External Flash 0 = Ext vtable 10000200, reset addr 100ffa9 I: Swap type: none
I: Bootloader chainload address offset: 0xc000
I: Jumping to the first image slot
I: pcd_lock_ram
[00:00:00.261,077] <inf> rp_ll_api: Handshake done
*** Booting Zephyr OS build v2.4.99-ncs2  ***

[00:00:00.261,627] <inf> main: ### comprensive examples @ appcore version v0.1 compiled at 17:39:40 May  3 2021

[00:00:00.261,779] <inf> main: ## OTA/Serial DFU example ##
[00:00:00.261,810] <wrn> main: exit main thread

[00:00:00.261,840] <inf> rpc_thread: **RPC usage example

[00:00:00.261,871] <inf> i2c_thread: **I2C master example working with nRF5_SDK\examples\peripheral\twi_master_with_twis_slave directly

[00:00:00.261,901] <inf> i2c_thread: **External interrupt example at P0.6

[00:00:00.261,901] <inf> spi_thread: **SPI master example working with nRF5_SDK\examples\peripheral\spis directly
[00:00:00.261,932] <inf> uart_thread: **high speed UART example
[00:00:00.262,023] <inf> adc_thread: **ADC sampling example
[00:00:00.262,023] <inf> adc_thread: ADC thread
[00:00:00.262,054] <inf> flash_thread: **Flash access example using both NVS and settings API
[00:00:00.262,176] <inf> adc_thread: Voltage0: 2994 mV / 3407
[00:00:00.262,176] <inf> adc_thread: Voltage1: 256 mV / 292
[00:00:00.264,068] <inf> fs_nvs: 3 Sectors of 4096 bytes
[00:00:00.264,068] <inf> fs_nvs: alloc wra: 0, f70
[00:00:00.264,068] <inf> fs_nvs: data wra: 0, 5c
[00:00:00.264,160] <inf> flash_thread: Key value in NVS:
                                 ff fe fd fc fb fa f9 f8                  |........
[00:00:00.264,160] <inf> flash_thread: ***Reboot_counter in NVS: 5 ***
[00:00:00.264,160] <inf> flash_thread: save new reboot counter by NVS API
```

## 3.1 BLE NUS example

When powering on the kit programmed with this example, the kit would advertise ‘nus_netcore’. After enabling CONFIG_RPC_SIMULATE_UART(**Refer to Section3.3 for more information**), you can test the example following the instructions in nrf\samples\bluetooth\peripheral_uart\ README.rst. The only difference between them is that NUS runs on netcore in this example while NUS runs on application core in nrf\samples\bluetooth\peripheral_uart. The network core project locates at nrf53_ble\ble_netcore. To build nrf53_ble\ble_netcore automatically, The kconfig or macro CONFIG_BLE_NETWORK_CORE must be defined. Once CONFIG_BLE_NETWORK_CORE=y, network core image would be built automatically when you build the application core image.

## 3.2 DFU example

This example supports updating both application core and network core images. The new images can be uploaded to the device by means of BLE or serial protocol. Both the new application core image or new network core images can be stored on internal Flash or external QSPI Flash during upgrading process. By default, we use SMP(Simple Management Protocol) protocol to handle the image management. CONFIG_EXAMPLE_DFU_OTA is used to control the DFU example.

NCS v1.5.1 or earlier cannot support updating network core if update image is stored on external Flash. To support this feature, the following files should be changed.

1. bootloader\mcuboot\boot\bootutil\src\loader.c. See nrf53_ble/resources/9loader.c for the full file and nrf53_ble/resources/9sdk-mcuboot_dfu_netcore_qspi.diff for the diff file

2. nrf\subsys\pcd\src\pcd.c. See nrf53_ble/resources/9pcd.c for the full file and nrf53_ble/resources/9sdk-nrf_dfu_netcore_qspi.diff for the diff file.
3. nrf\samples\nrf5340\netboot\src\main.c. See nrf53_ble/resources/9main.c for the full file and nrf53_ble/resources/9sdk-nrf_dfu_netcore_qspi.diff for the diff file.

### *3.2.1 BLE OTA DFU*

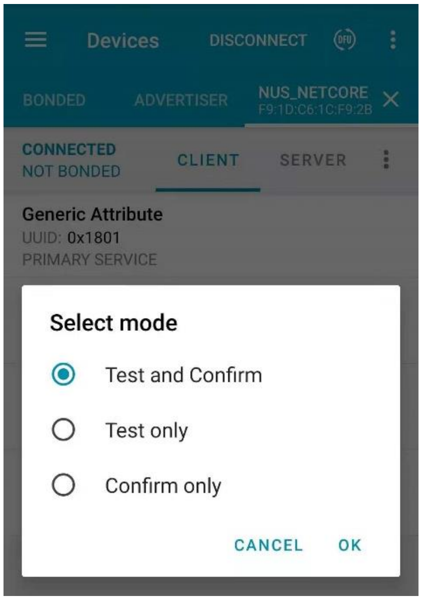
#### 3.2.1.1 Storing new images on external QSPI Flash

By default, the example uses external QSPI Flash to hold the new images. Both application core and network core new images can be stored on external Flash.

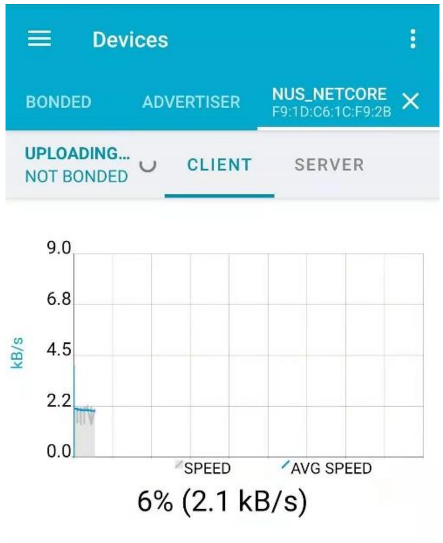
**Network core OTA procedures**

To OTA network core application, follow the steps below.

1. Copy *build_nrf5340dk_nrf5340_cpuapp\zephyr\net_core_app_update.bin* to your mobile phone(Make some changes on your project so you can get a different net_core_app_update.bin).
2. Open nRF connect for Mobile on your phone.
3. Connect the board.
4. Tap "DFU" button on the right top corner.
5. Select net_core_app_update.bin in your phone. The following UI would pop up:




**Be aware you may need to change your file explorer to re-select the image file if the previous one fails.** After selection is done, OTA DFU would start automatically. You would get the following UI.




After the new image is uploaded to the device, a reset would occur automatically. After the reset, MCUBoot would detect a new image ready for updating and perform the swap operations required to finish the DFU process. The logging on app core is shown below (Note: the following screenshot has some extra logging).

```
[00:01:40.272,552] <inf> adc_thread: Voltage1: 247 mV / 282 async
[00:02:00.263,061] <inf> adc_thread: ADC thread
[00:02:00.263,153] <inf> adc_thread: Voltage0: 3005 mV / 3420
[00:02:00.263,153] <inf> adc_thread: Voltage1: 253 mV / 288
*** Booting Zephyr OS build v2.4.99-ncs2 ***
I: Starting bootloader
I: Primary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Secondary image: magic=good, swap_type=0x2, copy_done=0x3, image_ok=0x3
I: Boot source: none
= secondary slot in External Flash 0 = Ext vtable 10000200, reset addr 100ffa9I: Swap type: test
I: Starting network core update
new image addr 10000200 len 31b58 offset 8800Turned on network coreI: Turned off network core
I: Done updating network core
*sz=1000 off=ef000, areaSize=f0000****nor erase addr=ef000, size=4096 **I: Bootloader chainload address offset: 0xc000
I: Jumping to the first image slot
I: pcd_lock_ram
[00:00:00.009,033] <inf> rp_ll_api: Handshake done
*** Booting Zephyr OS build v2.4.99-ncs2 ***

[00:00:00.009,582] <inf> main: ### comprehensive examples @ appcore version v0.1 compiled at 17:39:40 May  3 2021

[00:00:00.009,735] <inf> main: ## OTA/Serial DFU example ##
[00:00:00.009,765] <wrn> main: exit main thread

[00:00:00.009,796] <inf> rpc_thread: **RPC usage example

[00:00:00.009,826] <inf> i2c_thread: **I2C master example working with nRF5_SDK\examples\peripheral\twi_master_with_twis_slave directly

[00:00:00.009,857] <inf> i2c_thread: **External interrupt example at P0.6

[00:00:00.009,857] <inf> spi_thread: **SPI master example working with nRF5_SDK\examples\peripheral\spis directly
[00:00:00.009,887] <inf> uart_thread: **high speed UART example
[00:00:00.009,979] <inf> adc_thread: **ADC sampling example
```

### Application core OTA

The OTA procedure is just the same as that of network core. The only difference is the image file. Use build_nrf5340dk_nrf5340_cpuapp\zephyr\ app_update.bin for the application core updating. The logging on **app core** when MCUBoot performs the swap operation is shown below (Note: the following screenshot has some extra logging).

```
[00:01:00.020,263] <inf> adc_thread: Voltage0: 2999 mV / 3413 async
[00:01:00.020,294] <inf> adc_thread: Voltage1: 239 mV / 272 async
[00:01:20.010,681] <inf> adc_thread: ADC thread
[00:01:20.010,772] <inf> adc_thread: Voltage0: 3000 mV / 3414
[00:01:20.010,772] <inf> adc_thread: Voltage1: 239 mV / 272
*** Booting Zephyr OS build v2.4.99-ncs2 ***
I: Starting bootloader
I: Primary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Secondary image: magic=good, swap_type=0x2, copy_done=0x3, image_ok=0x3
I: Boot source: none
= secondary slot in External Flash 0 = Ext vtable 10000200, reset addr 10481I: Swap type: test
I: erasing trailer; fa_id=3 dev_name=NRF_FLASH_DRV_NAME
fa size=f0000, offset=c000 fap id =3 fa_id_primary =3 fa_id_secondary=1sz=4096 total_sz=4096 trailer_sz=2928 off=ef000I: erasing trailer; fa_id=1
dev_name=MX25R64
fa size=f0000, offset=0 fap id =1 fa_id_primary =3 fa_id_secondary=1**nor erase addr=ef000, size=4096 **sz=4096 total_sz=4096 trailer_sz=2928 off=ef000
**nor erase addr=0, size=4096 ****nor erase addr=1000, size=4096 ****nor erase addr=2000, size=4096 ****nor erase addr=3000, size=4096 ****nor erase
addr=4000, size=4096 ****nor erase addr=5000, size=4096 ****nor erase addr=6000, size=4096 ****nor erase addr=7000, size=4096 ****nor erase addr=8000,
size=4096 ****nor erase addr=9000, size=4096 ****nor erase addr=a000, size=4096 ****nor erase addr=b000, size=4096 ****nor erase addr=c000, size=4096
****nor erase addr=d000, size=4096 ****nor erase addr=e000, size=4096 ****nor erase addr=f000, size=4096 ****nor erase addr=10000, size=4096 ****nor
erase addr=11000, size=4096 ****nor erase addr=12000, size=4096 ****nor erase addr=13000, size=4096 ****nor erase addr=14000, size=4096 ****nor erase
addr=15000, size=4096 ****nor erase addr=16000, size=4096 ****nor erase addr=17000, size=4096 ****nor erase addr=18000, size=4096 ****nor erase addr=
19000, size=4096 ****nor erase addr=1a000, size=4096 ****nor erase addr=1b000, size=4096 **I: Bootloader chainload address offset: 0xc000
I: Jumping to the first image slot
I: pcd_lock_ram
[00:00:00.009,033] <inf> rp_ll_api: Handshake done
*** Booting Zephyr OS build v2.4.99-ncs2 ***

[00:00:00.009,552] <inf> main: ###New comprensive examples @ appcore version v0.1 compiled at 20:45:06 May  1 2021

[00:00:00.009,704] <inf> main: ## OTA/Serial DFU example ##
[00:00:00.009,704] <wrn> main: exit main thread

**RPC usage example

**I2C master example working with nRF5_SDK\examples\peripheral\twi_master_with_twis_slave directly

**External interrupt example at P0.6
```

### 3.2.1.2 Storing new images on internal Flash

To store update images onto internal Flash, we need to do following modifications.

1. Remove pm_static_external_flash.yml
2. Change nrf53_ble/appcore/prj.conf like below. You must define CONFIG_PM_PARTITION_SIZE_SETTINGS_STORAGE=0x4000 to make NVS work.

```
## open the following kconfig to enable the feature of external flash served as
CONFIG_DFU_EXTERNAL_FLASH=y
CONFIG_PM_EXTERNAL_FLASH=y
CONFIG_PM_EXTERNAL_FLASH_DEV_NAME="MX25R64"
CONFIG_PM_EXTERNAL_FLASH_BASE=0x0
CONFIG_PM_EXTERNAL_FLASH_SIZE=0x800000

CONFIG_NORDIC_QSPI_NOR=y
CONFIG_NORDIC_QSPI_NOR_FLASH_LAYOUT_PAGE_SIZE=4096
CONFIG_NORDIC_QSPI_NOR_STACK_WRITE_BUFFER_SIZE=16
```

```
## open the following kconfig to enable the feature of external flash
# CONFIG_DFU_EXTERNAL_FLASH=y
# CONFIG_PM_EXTERNAL_FLASH=y
# CONFIG_PM_EXTERNAL_FLASH_DEV_NAME="MX25R64"
# CONFIG_PM_EXTERNAL_FLASH_BASE=0x0
# CONFIG_PM_EXTERNAL_FLASH_SIZE=0x800000

# CONFIG_NORDIC_QSPI_NOR=y
# CONFIG_NORDIC_QSPI_NOR_FLASH_LAYOUT_PAGE_SIZE=4096
# CONFIG_NORDIC_QSPI_NOR_STACK_WRITE_BUFFER_SIZE=16
```

   You can replace the content of appcore/prj.conf with that of nrf53_ble/resources/5appcore_prj_ble_dfu_internal_flash.conf. Be aware to keep the name: prj.conf

3. Modify nrf53_ble/appcore/child_image/mcuboot.conf like below.

```
## the following kconfig is used to make external flash as the secondary slot
CONFIG_FLASH=y

# Required by QSPI
CONFIG_MULTITHREADING=y

CONFIG_NORDIC_QSPI_NOR=y
CONFIG_NORDIC_QSPI_NOR_FLASH_LAYOUT_PAGE_SIZE=4096
CONFIG_PM_EXTERNAL_FLASH=y
CONFIG_PM_EXTERNAL_FLASH_DEV_NAME="MX25R64"
CONFIG_PM_EXTERNAL_FLASH_BASE=0x0
CONFIG_PM_EXTERNAL_FLASH_SIZE=0x800000
CONFIG_BOOT_MAX_IMG_SECTORS=240
```

```
## the following kconfig is used to make external flash as the
# CONFIG_FLASH=y

# Required by QSPI
# CONFIG_MULTITHREADING=y

# CONFIG_NORDIC_QSPI_NOR=y
# CONFIG_NORDIC_QSPI_NOR_FLASH_LAYOUT_PAGE_SIZE=4096
# CONFIG_PM_EXTERNAL_FLASH=y
# CONFIG_PM_EXTERNAL_FLASH_DEV_NAME="MX25R64"
# CONFIG_PM_EXTERNAL_FLASH_BASE=0x0
# CONFIG_PM_EXTERNAL_FLASH_SIZE=0x800000
# CONFIG_BOOT_MAX_IMG_SECTORS=240
```

You can replace the content of appcore/child_image/mcuboot.conf with that of nrf53_ble/resources/ 5appcore_mcuboot_ble_dfu_internal_flash.conf. Remember to keep the name: mcuboot.conf.

4. #define CONFIG_NORDIC_QSPI_NOR_FLASH_LAYOUT_PAGE_SIZE 4096 in pcd.c
5. Rebuild the project like Section2
6. Perform the DFU procedures shown in 3.2.1.1. They are just the same procedures.

### 3.2.2 UART DFU

To enable UART DFU, open the following macros.

```
## UART DFU example ##
CONFIG_MCUMGR_SMP_UART=y
CONFIG_UART_MCUMGR_RX_BUF_SIZE=256
CONFIG_UART_MCUMGR_RX_BUF_COUNT=2

CONFIG_UART_0_INTERRUPT_DRIVEN=y
```

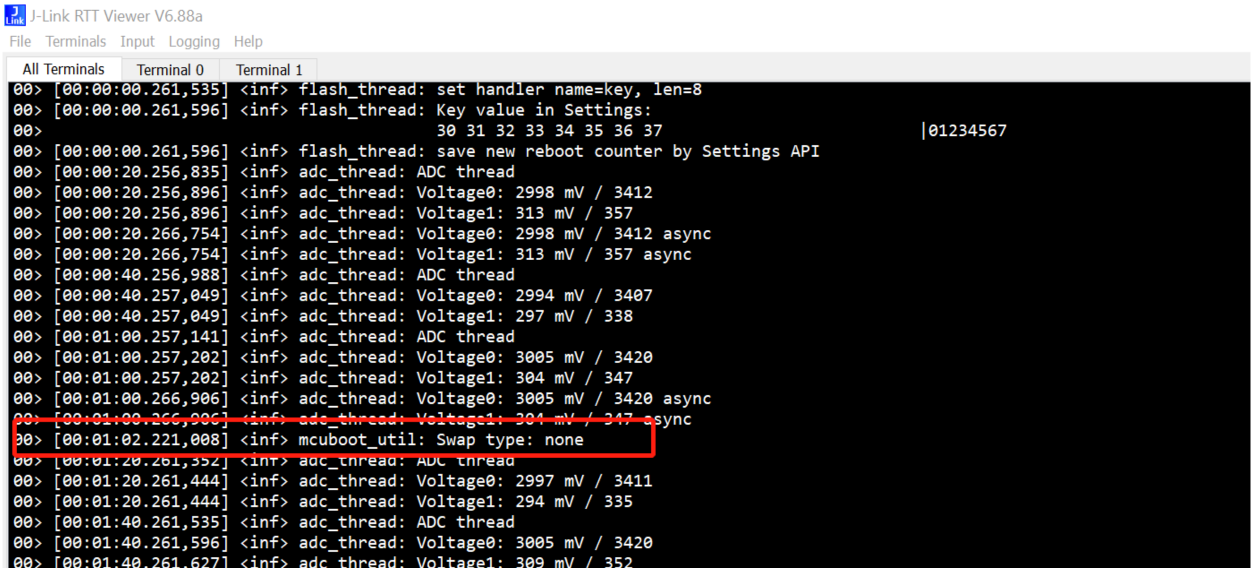
```
## UART DFU example ##
# CONFIG_MCUMGR_SMP_UART=y
# CONFIG_UART_MCUMGR_RX_BUF_SIZE=256
# CONFIG_UART_MCUMGR_RX_BUF_COUNT=2
# CONFIG_MCUMGR_SMP_UART_MTU=512
# CONFIG_UART_0_INTERRUPT_DRIVEN=y
```

To avoid UART resources conflict, you can use RTT logging instead of UART logging and turn off the high speed UART example. Refer to nrf53_ble/resources/8appcore_prj_uart_dfu_internal_flash.conf for the details.

The screenshot of DFU process on PC end is shown below.

```
C:\Nordic\NCS\SDK\Master\nrf53_ble\appcore>mcumgr conn add myCOM type="serial" connstring="dev=COM40,baud=115200,mtu=256"
Connection profile myCOM successfully added

C:\Nordic\NCS\SDK\Master\nrf53_ble\appcore>mcumgr -c myCOM image upload build_nrf5340dk_nrf5340_cpuapp/zephyr/net_core_app_update.bin
 51.84 KiB / 199.66 KiB [============================>-------------------------------------------------------------------------------------]  25.96% 2.78 KiB/s 00m53s
```

App core RTT logging is shown below.




#### 3.2.2.1 Storing new images on internal Flash

Replace the content of appcore/prj.conf with that of nrf53_ble/resources/8appcore_prj_uart_dfu_internal_flash.conf and replace the content of appcore/child_image/mcuboot.conf with that of nrf53_ble/resources/ 5appcore_mcuboot_ble_dfu_internal_flash.conf. Refer to https://docs.zephyrproject.org/latest/guides/device_mgmt/index.html#mcumgr-cli for the DFU procedures. And see nrf53_ble/resources/8mcumgr_cmd.txt for the commands tested in Windows(mcumgr cli is not so stable on Windows).

#### 3.2.2.2 Storing new images on external QSPI Flash

Replace the content of appcore/prj.conf with that of nrf53_ble/resources/ 8appcore_prj_uart_dfu_external_flash.conf and replace the content of appcore/child_image/mcuboot.conf with that of nrf53_ble/resources/ 9appcore_mcuboot_ble_dfu_ext_flash.conf. Ensure pm_static_external_flash.yml is present in nrf53_ble/appcore. Refer to https://docs.zephyrproject.org/latest/guides/device_mgmt/index.html#mcumgr-cli for the DFU procedures. And see nrf53_ble/resources/8mcumgr_cmd.txt for the commands tested in Windows(mcumgr cli is not so stable on Windows).

### *3.2.3 custom DFU*

We use SMP protocol to upload the update images to the secondary slot. However, you can use your own protocol to transfer the new images to the secondary slot. To have a custom DFU, what you should do is to transfer app_update.bin(application core new image) or net_core_app_to_sign.bin(network core new image) to the secondary slot. You can find the starting address and size of secondary slot in partitions.yml(secondary slot on internal Flash) or pm_static_external_flash.yml(secondary slot on external Flash). Once you upload the new image, you should call boot_request_upgrade(true). This API will write a magic number into the last sector of secondary slot. MCUboot would do a swap operation once it detects this magic number after a reboot.

To know more about MCUboot working mechanism, see this link: [https://github.com/mcu-tools/mcuboot/blob/master/docs/design.md](https://github.com/mcu-tools/mcuboot/blob/master/docs/design.md).

Below is an example to use J-Link to transfer the new image to the secondary slot(Internal Flash). Please note that the last sector would be updated too after the J-Link programming. Thus we don't need to call boot_request_upgrade() any more in this case.

**Network core custom DFU**

***nrfjprog --program build_nrf5340dk_nrf5340_cpuapp/zephyr/net_core_app_moved_test_update.hex --sectorerase -r***

Once the new image is programed to the secondary slot and reset occurs, MCUBoot would perform updating operations since the last sector has the magic number. See below for the logging.

```
*** Booting Zephyr OS build v2.4.99-ncs2 ***
I: Starting bootloader
I: Primary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Secondary image: magic=good, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Boot source: none
=secondary slot in internal Flash= In vtable 84200, reset addr 100ffa9 : Swap type: test
I: Starting network core update
new image addr 84200 len 31b54 offset 8800Turned on network coreI: Turned off network core
: Done updating network core
I: erasing trailer; fa_id=5 dev_name=NRF_FLASH_DRV_NAME
fa size=78000, offset=84000 fap id =5 fa_id_primary =2 fa_id_secondary=5sz=4096 total_sz=4096 trailer_sz=1584 off=77000I: Bootloader chainload address
offset: 0xc000
I: Jumping to the first image slot
I: pcd_lock_ram
```

**App core custom DFU**

***nrfjprog --program build_nrf5340dk_nrf5340_cpuapp/zephyr/app_moved_test_update.hex --sectorerase -r***

Once the new image is programed to the secondary slot and reset occurs, MCUBoot would perform updating operations since the last sector has the magic number. See below for the logging.

```
*** Booting Zephyr OS build v2.4.99-ncs2 ***
I: Starting bootloader
I: Primary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Secondary image: magic=good, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Boot source: none
=secondary slot in internal Flash= In vtable 84200, reset addr 11929 : Swap type: test
I: erasing trailer; fa_id=2 dev_name=NRF_FLASH_DRV_NAME
fa size=78000, offset=c000 fap id =2 fa_id_primary =2 fa_id_secondary=5sz=4096 total_sz=4096 trailer_sz=1584 off=77000I: erasing trailer; fa_id=5
dev_name=NRF_FLASH_DRV_NAME
fa size=78000, offset=84000 fap id =5 fa_id_primary =2 fa_id_secondary=5sz=4096 total_sz=4096 trailer_sz=1584 off=77000I: Bootloader chainload address
offset: 0xc000
I: Jumping to the first image slot
I: pcd_lock_ram
```

## 3.3 Dual core interaction example

Application core and network core can communicate with each other. There are 2 ways to achieve the interactions: calling nrf_rpc APIs directly to communicate like a 'dual-core UART' or encapsulating nrf_rpc with cbor coding to communicate like remote procedure call(RPC). In this example, the first method is controlled by macro: CONFIG_RPC_SIMULATE_UART. The later one is identified by marco: CONFIG_RPC_REMOTE_API. These 2 kconfig are not compatible. You can only select one of them. Please note that DFU example above uses dual core communication too. To make DFU work, you must select CONFIG_RPC_REMOTE_API.

Please note that the shared memory configurations of application core and network core must be the same. Otherwise, the dual core would fail to communicate with each other. See below for the application core shared memory definitions.

C:\Nordic\NCS\SDK\Master\nrf53_ble\appcore\nrf5340dk_nrf5340_cpuapp.overlay - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

nrf5340dk_nrf5340_cpuapp.overlay

```
&sram0_s{
    reg = < 0x20000000 0x7A000 >;
};

/delete-node/ &sram0_shared;

/ {
    chosen {
        /* shared memory reserved for the inter-processor communication */
        zephyr,ipc_shm = &sram0_shared;
    };

    reserved-memory {
        #address-cells = <1>;
        #size-cells = <1>;
        ranges;
        sram0_shared: memory@0x2007A000 {
            /* SRAM allocated to shared memory */
            reg = <0x2007A000 0x4000>;
        };
    };
};
```

See below for the network core shared memory definitions.

C:\Nordic\NCS\SDK\Master\nrf53_ble\ble_netcore\nrf5340dk_nrf5340_cpunet.overlay - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

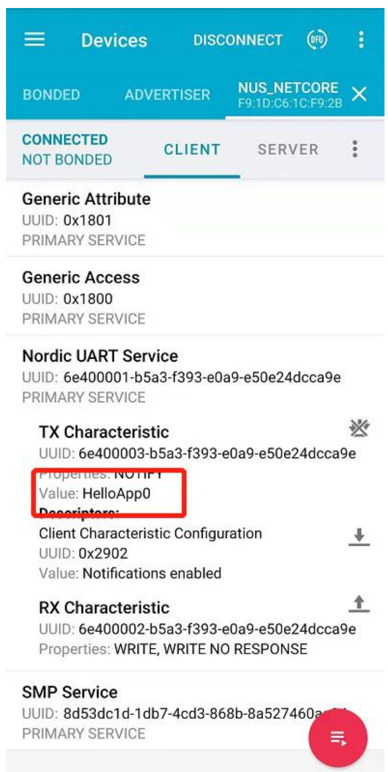
nrf5340dk_nrf5340_cpunet.overlay

```
/delete-node/ &sram0_shared;

/ {
    chosen {
        /* shared memory reserved for the inter-processor communication */
        zephyr,ipc_shm = &sram0_shared;
    };

    reserved-memory {
        #address-cells = <1>;
        #size-cells = <1>;
        ranges;
        sram0_shared: memory@0x2007A000 {
            /* SRAM allocated to shared memory */
            reg = <0x2007A000 0x4000>;
        };
    };
};
```

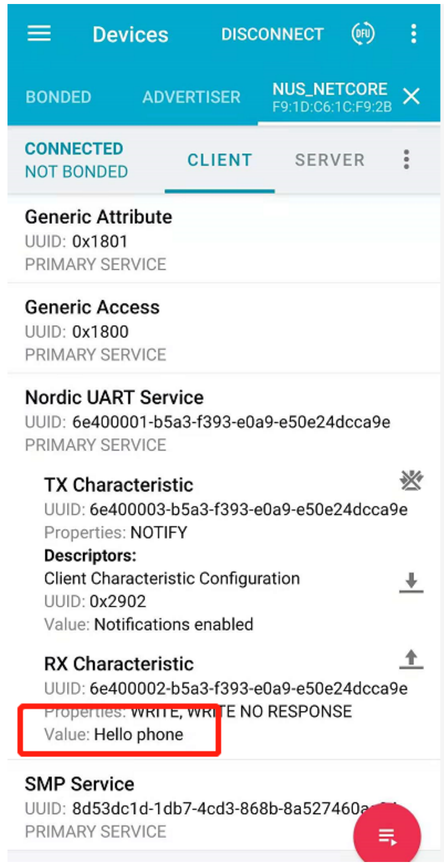
## Button trigger

rpc_thread.c, rpc_app_nus.c and rpc_net_nus.c are used to do the dual core communication. We can test this feature by simply pressing **Button1**, which will let app core send a message to mobile app. See below for the logging.

```
[00:00:00.261,047] <inf> flash_thread: settings subsys initialization: OK.
[00:00:00.261,077] <inf> flash_thread: Load all key-value pairs using registered handlers
[00:00:00.261,138] <inf> flash_thread: set handler name=boot_cnt, len=4
[00:00:00.261,169] <inf> flash_thread: *** Reboot counter in Settings: 1 ****
[00:00:00.261,230] <inf> flash_thread: set handler name=key, len=8
[00:00:00.261,291] <inf> flash_thread: Key value in Settings:
                    30 31 32 33 34 35 36 37                  |01234567
[00:00:00.261,291] <inf> flash_thread: save new reboot counter by Settings API
[00:00:17.661,132] <inf> main: button1 isr

[00:00:17.661,224] <inf> rpc_thread: RPC thread

[00:00:17.661,834] <inf> rpc_thread: sent by app core:
                   48 65 6c 6c 6f 41 70 70  30 00 00 00         |HelloApp 0...
[00:00:20.256,866] <inf> adc_thread: ADC thread
[00:00:20.256,958] <inf> adc_thread: Voltage0: 3000 mV / 3414
[00:00:20.256,958] <inf> adc_thread: Voltage1: 241 mV / 275
[00:00:20.266,754] <inf> adc_thread: Voltage0: 3000 mV / 3414 async
[00:00:20.266,784] <inf> adc_thread: Voltage1: 241 mV / 275 async
[00:00:40.257,019] <inf> adc_thread: ADC thread
[00:00:40.257,110] <inf> adc_thread: Voltage0: 3009 mV / 3424
```




You can also use mobile app to send data to app core. Enter “Hello phone” under RX Characteristic, and you would have.




```
00:00:20.256,866] <inf> adc_thread: ADC thread
00:00:20.256,927] <inf> adc_thread: Voltage0: 2998 mV / 3412
00:00:20.256,927] <inf> adc_thread: Voltage1: 243 mV / 277
00:00:20.266,754] <inf> adc_thread: Voltage0: 2998 mV / 3412 async
00:00:20.266,754] <inf> adc_thread: Voltage1: 243 mV / 277 async
00:00:21.648,803] <inf> rpc_thread: received ble data from netcore
00:00:21.648,834] <inf> rpc_thread: data:
                 48 65 6c 6c 6f 20 70 68  6f 6e 65            |Hello ph one
00:00:40.257,019] <inf> adc_thread: ADC thread
00:00:40.257,080] <inf> adc_thread: Voltage0: 3004 mV / 3419
00:00:40.257,080] <inf> adc_thread: Voltage1: 399 mV / 454
```

**PC terminal trigger**

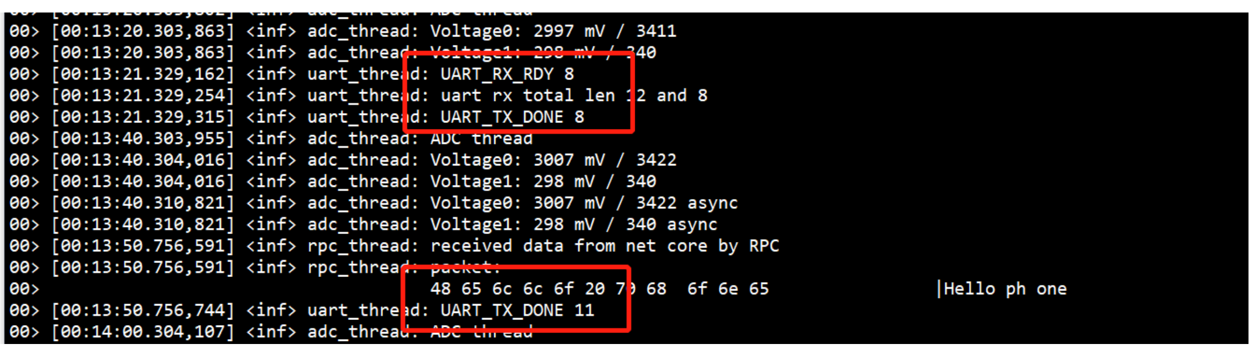
Network core can receive BLE data from the mobile app and forward it to app core. After receiving the data, app core can send it to the PC terminal by UART interface. Both CONFIG_RPC_SIMULATE_UART and CONFIG_RPC_REMOTE_API support this direction of communication in this example. The logging is shown below:

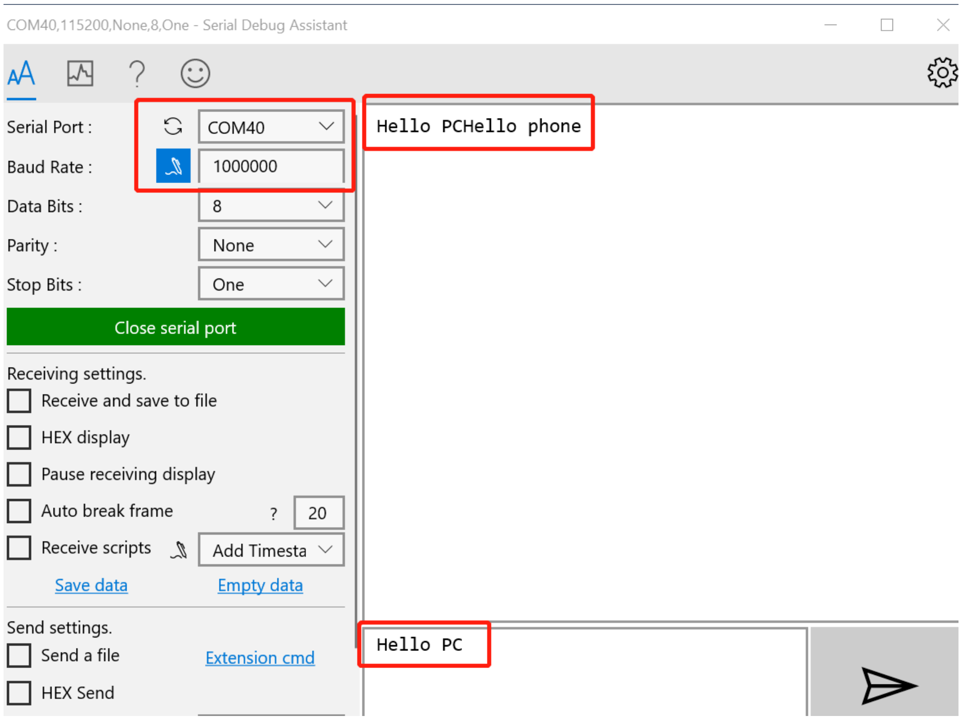
To send data from PC terminal to mobile app, you can only select CONFIG_RPC_SIMULATE_UART in this example. And UART1 is used instead of console UART0. To switch to CONFIG_RPC_SIMULATE_UART mode, we need to disable OTA DFU and build app core and net core images separately. See below for the details.

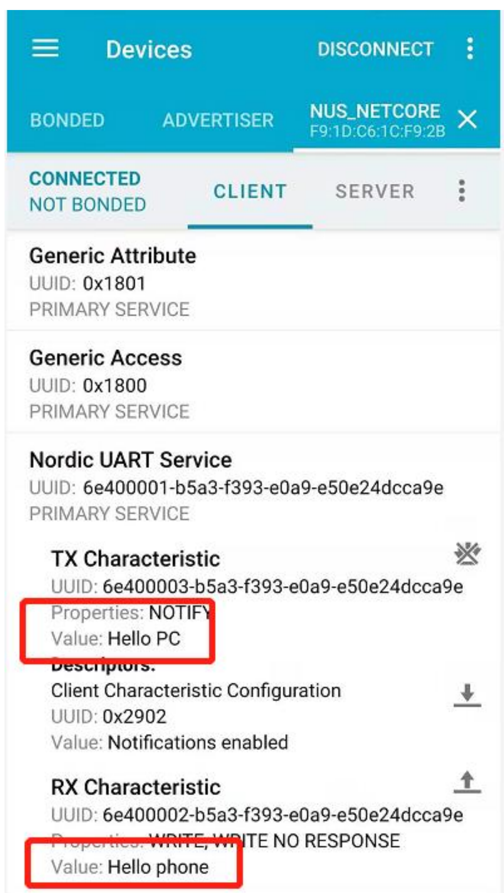
### *3.3.1 dual-core UART communication (CONFIG_RPC_SIMULATE_UART)*

Follow the steps below to build app core and network core images separately, and switch to the CONFIG_RPC_SIMULATE_UART mode.

- Replace the content of nrf53_ble/appcore/prj.conf with that of nrf53_ble/resources/4appcore_prj_rpc_simulate_uart.conf. be aware to keep the name: prj.conf
- Replace the content of nrf53_ble/ble_netcore/prj.conf with that of nrf53_ble/resources/4netcore_prj_rpc_simulate_uart.conf. be aware to keep the name: prj.conf
- Replace the content of nrf53_ble/appcore/nrf5340dk_nrf5340_cpuapp.overlay with that of nrf53_ble/resources/4nrf5340dk_nrf5340_cpuapp.overlay. be aware to keep the name: nrf5340dk_nrf5340_cpuapp.overlay. This step is not required. We do it since we want to use DK board J-Link CDC COM directly without any wirings.
- Application core image build and download. Type in following commands to complete the build process and flash process.
    - **cd C:\Nordic\NCS\SDK\Master\nrf53_ble\appcore**
    - **west build -b nrf5340dk_nrf5340_cpuapp -d build_nrf5340dk_nrf5340_cpuapp -p**
    - **west flash -d build_nrf5340dk_nrf5340_cpuapp**
    - **Note: the above steps only build and download application core image. Network core image is not built and flashed automatically like before.**
- Network core image build and download. Enter following commands to finish the build process and download process of network core image.
    - **cd C:\Nordic\NCS\SDK\Master\nrf53_ble\ble_netcore**
    - **west build -b nrf5340dk_nrf5340_cpunet -d build_nrf5340dk_nrf5340_cpunet -p**
    - **west flash -d build_nrf5340dk_nrf5340_cpunet**
- Download "Serial Debug Assistant" from Microsoft Store.
- Connect your board with nRF connect mobile app. And enable CCCD.
- Send some data via mobile app to PC terminal and vice versa. You would get the following outputs.




COM40,115200,None,8,One - Serial Debug Assistant
Serial Port :
COM40
Baud Rate :
1000000
Data Bits :
8
Parity :
None
Stop Bits :
One
Close serial port
Receiving settings.
Receive and save to file
HEX display
Pause receiving display
Auto break frame
20
Receive scripts
Add Timesta
Save data
Empty data
Send settings.
Send a file
Extension cmd
HEX Send
Hello PCHello phone
Hello PC


Devices
DISCONNECT
BONDED
ADVERTISER
NUS_NETCORE
F9:1D:C6:1C:F9:2B
CONNECTED
NOT BONDED
CLIENT
SERVER
Generic Attribute
UUID: 0x1801
PRIMARY SERVICE
Generic Access
UUID: 0x1800
PRIMARY SERVICE
Nordic UART Service
UUID: 6e400001-b5a3-f393-e0a9-e50e24dcca9e
PRIMARY SERVICE
TX Characteristic
UUID: 6e400003-b5a3-f393-e0a9-e50e24dcca9e
Properties: NOTIFY
Value: Hello PC
Client Characteristic Configuration
UUID: 0x2902
Value: Notifications enabled
RX Characteristic
UUID: 6e400002-b5a3-f393-e0a9-e50e24dcca9e
TE NO RESPONSE
Value: Hello phone

## 3.4 High speed UART example

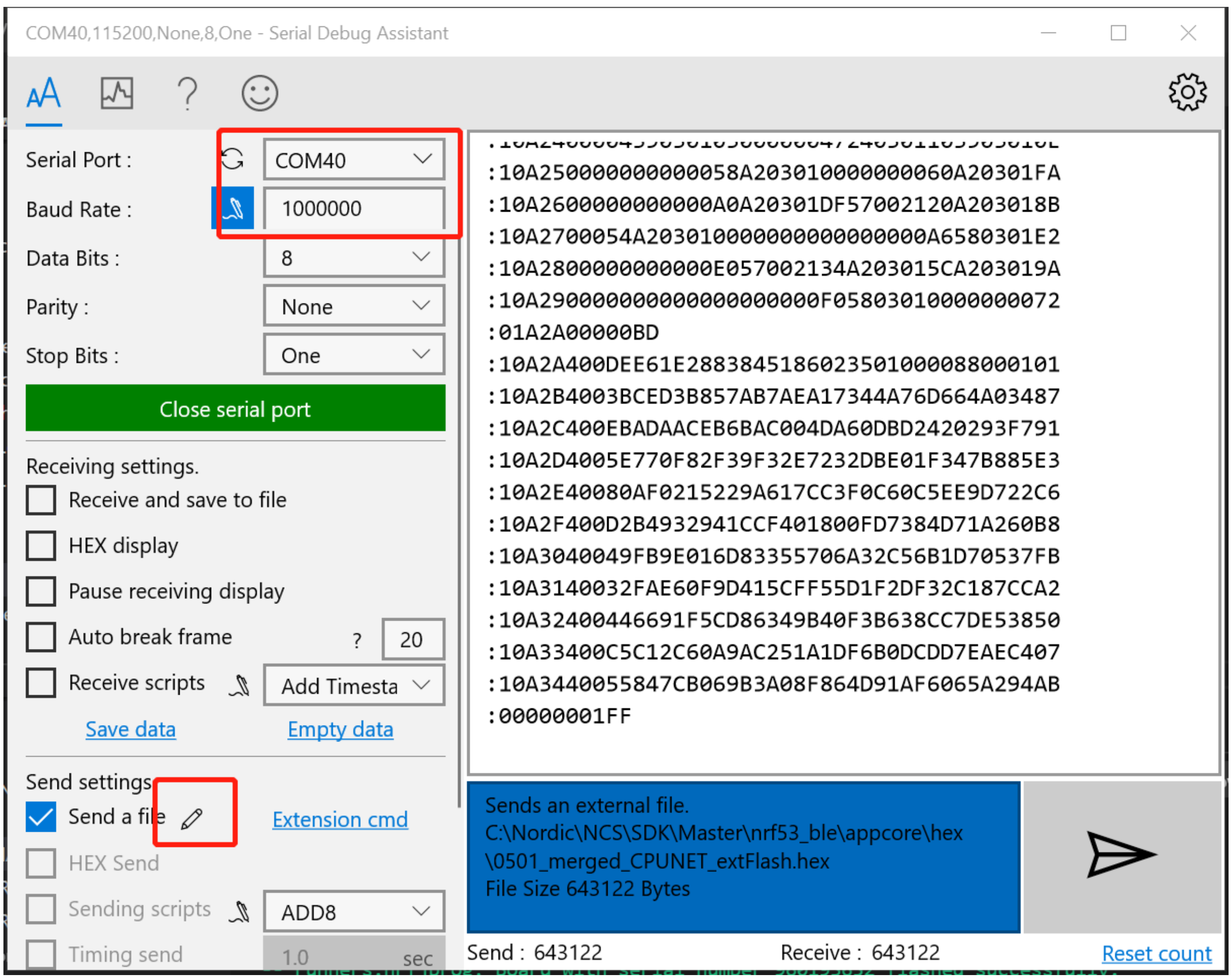
By this example, you can achieve 1Mbps baud rate. UART has 3 working mode: poll, interrupt and asynchronize. To achieve the high speed UART, asyn mode must be used.

In fact, we already tested 1Mbps in section3.3.1. To test the reliability of 1Mbps UART, you can transfer a file from PC end to the device end. In this example, when PC sends some data to the device, the device would send the same data back to the PC. In this way, you can verify the reliability of 1Mbps UART. You can use the same configuration files as section3.3.1, or you can also use the default configuration files. When doing the loopback test of 1Mbps UART, make sure BLE connection is disconnected and logging terminal is closed since they would have a great impact on the UART communication.

As mentioned earlier, please use “Serial Debug Assistant” from Microsoft Store for the test. We recommend using UART0 for the test. Thus, you need to change the following line.

```
#define UART_DEVICE_NAME          DT_LABEL(DT_NODELABEL(uart0))
```

And you would get the following testing result.



## 3.5 SPI example

This example shows how to call Zephyr-related SPI APIs to communicate with a SPI slave. The SPI slave image can be directly obtained from nRF5_SDK\examples\peripheral\spis. To facilitate the test, we put the spis images at nrf53_ble/resources/hex. The spis pin definitions are shown below.

| | |
|---|---|
| APP_SPIS_SCK_PIN - Pin number | 26 (P0.26) |
| APP_SPIS_MISO_PIN - Pin number | 30 (P0.30) |
| APP_SPIS_MOSI_PIN - Pin number | 29 (P0.29) |
| APP_SPIS_CS_PIN - Pin number | 31 (P0.31) |

And SPI master pin definitions are shown below.

```
&spi3 {
    status = "okay";
    compatible = "nordic,nrf-spim";
    sck-pin = < 47 >;
    miso-pin = < 46 >;
    mosi-pin = < 45 >;
```

#define PIN_SPIM_CS    12  //P1.12 //44

Program the spis hex file to a 832DK or 840DK. Connect the spis related pins to their counterparts in our nRF53 board.

After pressing **Button2**, this example can start to communicate with spis. The logging is shown below.

```
[00:00:05.266,876] <inf> adc_thread: Voltage1: 298 mV / 340 async
[00:00:05.325,592] <inf> spi_thread: spi master thread
[00:00:05.325,653] <inf> spi_thread: Received SPI data:
                         4e 6f 72 64 69 63 00              |Nordic.
[00:00:05.525,756] <inf> spi_thread: spi master thread
[00:00:05.525,817] <inf> spi_thread: Received SPI data:
                         4e 6f 72 64 69 63 00              |Nordic.
[00:00:05.725,921] <inf> spi_thread: spi master thread
[00:00:05.725,982] <inf> spi_thread: Received SPI data:
                         4e 6f 72 64 69 63 00              |Nordic.
[00:00:05.926,055] <inf> spi_thread: spi master thread
[00:00:05.926,116] <inf> spi_thread: Received SPI data:
                         4e 6f 72 64 69 63 00              |Nordic.
[00:00:06.126,190] <inf> spi_thread: spi master thread
[00:00:06.126,251] <inf> spi_thread: Received SPI data:
                         4e 6f 72 64 69 63 00              |Nordic.
[00:00:06.326,324] <inf> spi_thread: spi master thread
[00:00:06.326,385] <inf> spi_thread: Received SPI data:
```

## 3.6 I2C example

This example shows how to use Zephyr-related I2C APIs to communicate with a I2C slave. The I2C slave image can be directly obtained from nRF5_SDK\examples\peripheral\twi_master_with_twis_slave.

To facilitate the test, we put the twis images at nrf53_ble/resources/hex. The twis pin definitions are shown below.

```
#define EEPROM_SIM_SCL_S         31    //!< Slave SCL pin.
#define EEPROM_SIM_SDA_S         30    //!< Slave SDA pin.
```

And I2C master pin definitions are shown below.

```
&i2c2 {
    status = "okay";
    compatible = "nordic,nrf-twim";
    sda-pin = <4>;
    scl-pin = <27>;
};
```

Program the twis hex file to a 832DK or 840DK. Connect the twis related pins to their counterparts in our nRF53 board.

After **P0.06 is pulled down**, this example can start to communicate with twis. The logging is shown below.

```
[00:00:00.261,260] <inf> flash_thread: set handler name=key, len=8
[00:00:00.261,291] <inf> flash_thread: Key value in Settings:
                    30 31 32 33 34 35 36 37                  |01234567
[00:00:00.261,322] <inf> flash_thread: save new reboot counter by Settings API
[00:00:04.379,455] <inf> i2c_thread: external interrupt occurs at 143506

[00:00:04.379,516] <inf> i2c_thread: i2c master thread
[00:00:04.380,126] <inf> i2c_thread: EEPROM:
                   f8 f7 66 ff 1e b9 25 a1  f4 20 f8 f7 61 ff 28 46 |..f...%. . ..a.(F
[00:00:04.380,737] <inf> i2c_thread: EEPROM:
                   00 f0 60 f8 10 b1 11 20  bd e8 f0 9f 66 61 4f f0 |..`....  ....faO.
[00:00:04.381,317] <inf> i2c_thread: EEPROM:
                   00 09 c4 f8 20 90 a7 60  84 f8 28 90 4f f4 8e 78 |.... ..` ..(.O..x
[00:00:04.381,927] <inf> i2c_thread: EEPROM:
                   4e 46 41 46 28 68 ff f7  09 f9 28 68 4f f0 01 0a |NFAF(h.. ..(hO...
[00:00:04.382,537] <inf> i2c_thread: EEPROM:
                   c0 f8 08 a0 21 46 28 68  01 f0 b6 fa 60 68 00 bb |....!F(h ....`h..
[00:00:04.383,148] <inf> i2c_thread: EEPROM:
                   d5 f8 00 b0 67 69 0e e0  94 f8 28 00 08 b1 0f 26 |....gi.. ..(....&
[00:00:04.383,758] <inf> i2c_thread: EEPROM:
                   15 e0 41 46 58 46 ff f7  ec f8 00 28 f4 d0 21 46 |..AFXF.. ...(..!F
```

### 3.7 ADC example

ADC has 2 working modes: sync and async mode. And it can sample many channels simultaneously. This example samples 2 channels (VDD and P0.05) together, and work in both sync and async mode.

Change the voltage on P0.05, we would see a changing ADC value from the log.

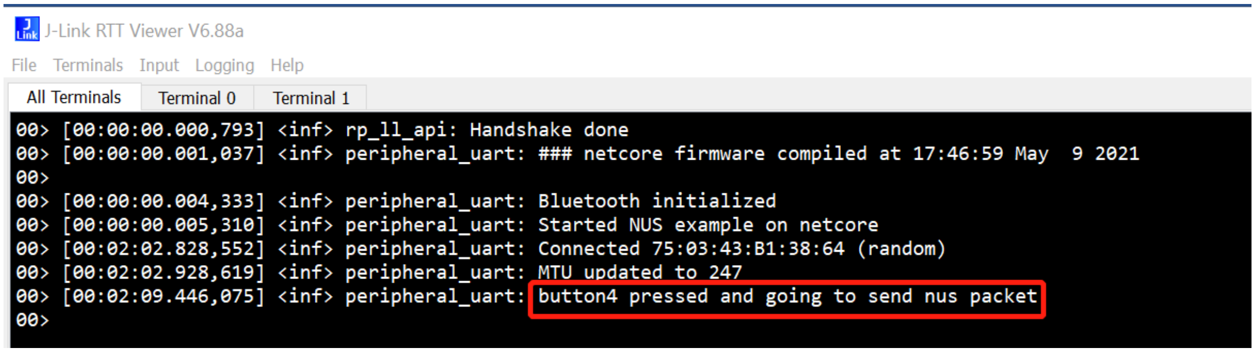
```
[00:01:00.257,232] <inf> adc_thread: ADC thread
[00:01:00.257,293] <inf> adc_thread: Voltage0: 3002 mV / 3416
[00:01:00.257,293] <inf> adc_thread: Voltage1: 222 mV / 253
[00:01:00.266,998] <inf> adc_thread: Voltage0: 3002 mV / 3416 async
[00:01:00.267,028] <inf> adc_thread: Voltage1: 222 mV / 253 async
[00:01:20.257,385] <inf> adc_thread: ADC thread
[00:01:20.257,446] <inf> adc_thread: Voltage0: 2990 mV / 3402
[00:01:20.257,446] <inf> adc_thread: Voltage1: 186 mV / 212
[00:01:40.257,537] <inf> adc_thread: ADC thread
[00:01:40.257,598] <inf> adc_thread: Voltage0: 2997 mV / 3410
[00:01:40.257,598] <inf> adc_thread: Voltage1: 2868 mV / 3264
[00:01:40.267,181] <inf> adc_thread: Voltage0: 2997 mV / 3410 async
[00:01:40.267,181] <inf> adc_thread: Voltage1: 2868 mV / 3264 async
```

## 3.8 External Interrupt example

We have 2 external interrupt examples. One is on application core. The other is on network core. By reading the code, you would find API usage on network core is just the same as that of application core. Regarding application core external interrupt example, see section3.6, where we use an external interrupt to trigger I2C communication. In terms of network core external interrupt example, you just press **Button4** which would trigger an external interrupt on network core. After pressing **Button4**, network would send a message to the mobile app directly without appcore's awareness. The logging of application core external interrupt example is shown below.

```
00:00:00.256,561] <inf> i2c_thread: **I2C master example working with nRF5_SDK\examples\peripheral\twi_master_with_twis_slave
00:00:00.256,591] <inf> i2c_thread: **External interrupt example at P0.6

00:00:00.256,622] <inf> spi_thread: **SPI master example working with nRF5_SDK\examples\peripheral\spis directly
00:00:00.256,652] <inf> adc_thread: **ADC sampling example
00:00:00.256,683] <inf> adc_thread: ADC thread
00:00:00.256,713] <inf> flash_thread: **Flash access example using both NVS and settings API
00:00:00.256,713] <inf> flash_thread: NVS flash offset f8000
00:00:00.256,835] <inf> adc_thread: Voltage0: 2992 mV / 3405
00:00:00.256,835] <inf> adc_thread: Voltage1: 258 mV / 294
00:00:00.258,697] <inf> fs_nvs: 3 Sectors of 4096 bytes
00:00:00.258,697] <inf> fs_nvs: alloc wra: 0, fb0
00:00:00.258,697] <inf> fs_nvs: data wra: 0, 3c
00:00:00.258,758] <inf> flash_thread: Key value in NVS:
                                 ff fe fd fc fb fa f9 f8                          |........
00:00:00.258,789] <inf> flash_thread: *** Reboot counter in NVS: 1 ***
00:00:00.258,789] <inf> flash_thread: save new reboot counter by NVS API
00:00:00.261,077] <inf> fs_nvs: 4 Sectors of 4096 bytes
00:00:00.261,077] <inf> fs_nvs: alloc wra: 0, fa8
00:00:00.261,077] <inf> fs_nvs: data wra: 0, 40
00:00:00.261,108] <inf> flash_thread: settings subsys initialization: OK.
00:00:00.261,138] <inf> flash_thread: Load all key-value pairs using registered handlers
00:00:00.261,199] <inf> flash_thread: set handler name=boot_cnt, len=4
00:00:00.261,230] <inf> flash_thread: *** Reboot counter in Settings: 1 ****
00:00:00.261,291] <inf> flash_thread: set handler name=key, len=8
00:00:00.261,322] <inf> flash_thread: Key value in Settings:
                                 30 31 32 33 34 35 36 37                          |01234567
00:00:00.261,352] <inf> flash_thread: save new reboot counter by Settings API
00:00:20.256,896] <inf> adc_thread: ADC thread
00:00:20.256,958] <inf> adc_thread: Voltage0: 2997 mV / 3411
00:00:20.256,958] <inf> adc_thread: Voltage1: 386 mV / 440
00:00:20.266,815] <inf> adc_thread: Voltage0: 2997 mV / 3411 async
00:00:20.266,815] <inf> adc_thread: Voltage1: 386 mV / 440 async
00:00:20.649,414] <inf> i2c_thread: external interrupt occurs at 676640

00:00:20.649,444] <inf> i2c_thread: i2c master thread
```

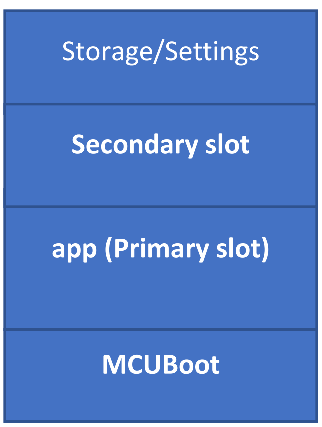
And see below for the logging of network core external interrupt usage example.



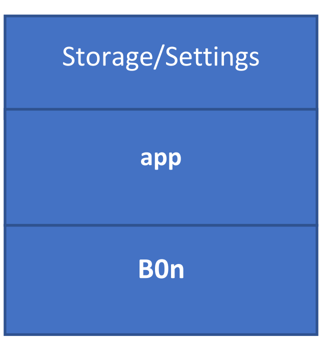

## 3.9 Flash access example

Each SoC has his own Flash memory. And we can partition the whole memory into different areas. The partition can be done by DeviceTree or Nordic-developed PM(partition manager). To get the layout or partitions of your device, you can use Flash map API. Generally, nRF53 has the following memory layout.

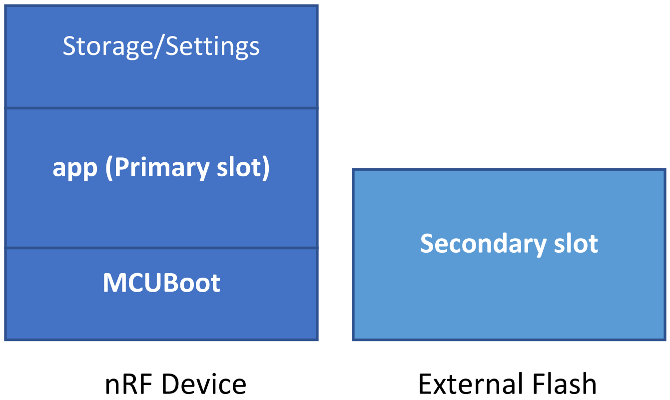
**Application core memory layout**:



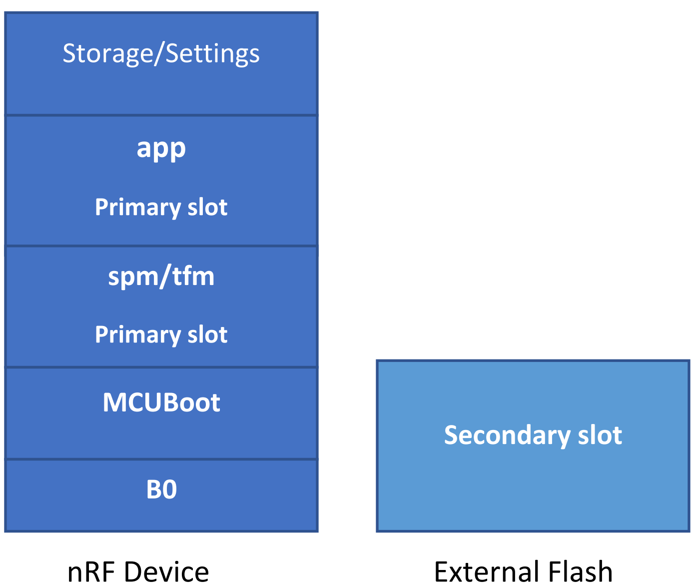

**Network core memory layout**



If secondary slot is located on external Flash, the memory layout of application core would become:

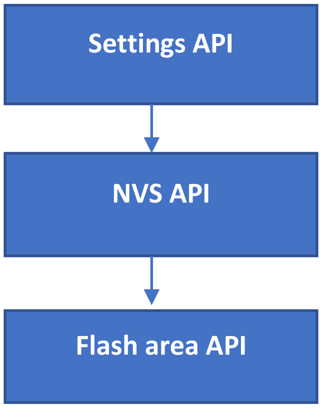



If app runs on non-secure region, secondary slot on external Flash, and MCUBoot is upgradable, the memory layout would become:




After we get the partitions of Flash memory, we can use different Flash APIs to access each partition. There are 3 layers(sets) of Flash access APIs in NCS: Flash area API, NVS API and Settings API. The bottom layer is Flash area API which access Flash directly without additional headers or tails. NVS API invokes Flash area API to achieve the Flash access purpose. To have a better reliability and readability, NVS would add some additional info at the end of a page.  Settings API calls NVS API to access Flash memory.

Thus, Settings module has a further encapsulation of raw serialized data. All data is managed by key/value pair in Settings module. The relations are shown below.




Apart from Storage/Settings area, we generally use Flash area API to operate other areas. See *Section3.2 DFU example* for Flash area usage example.

Regarding Storage/Settings area, we can use NVS API or Settings API to access it. Storage/Settings area is used to store user persistent data. Bluetooth bonding is dependent on Settings API. Thus, if your application uses Bluetooth bonding, Settings module would be turned on automatically. In this example, we use both NVS API and Settings API to do the same thing: store a secret and reboot counter onto the internal Flash. See below for the related logging:

The first-time boot:

```
00:00:00.256,652] <inf> adc_thread: **ADC sampling example
00:00:00.256,683] <inf> adc_thread: ADC thread
00:00:00.256,713] <inf> flash_thread: **Flash access example using both NVS and settings API
00:00:00.256,713] <inf> flash_thread: NVS flash offset f8000
00:00:00.256,835] <inf> adc_thread: Voltage0: 2997 mV / 3410
00:00:00.256,835] <inf> adc_thread: Voltage1: 581 mV00:00.258,666] <inf> fs_nvs: 3 Sectors of 4096 bytes
00:00:00.258,666] <inf> fs_nvs: alloc wra: 0, ff0
00:00:00.258,697] <inf> fs_nvs: data wra: 0, 0
00:00:00.258,697] <inf> flash_thread: No key found, adding it at id 1 by NVS API
00:00:00.258,972] <inf> flash_thread: save new reboot counter by NVS API
00:00:00.261,199] <inf> fs_nvs: 4 Sectors of 4096 bytes
00:00:00.261,199] <inf> fs_nvs: alloc wra: 0, fe0
00:00:00.261,199] <inf> fs_nvs: data wra: 0, c
00:00:00.261,230] <inf> flash_thread: settings subsys initialization: OK.
00:00:00.261,260] <inf> flash_thread: Load all key-value pairs using registered handlers
00:00:00.261,291] <inf> flash_thread: save key_s By Settings API
00:00:00.262,054] <inf> flash_thread: save new reboot counter by Settings API
```

The second time boot:

```
[00:00:00.256,652] <inf> adc_thread: **ADC sampling example
[00:00:00.256,683] <inf> adc_thread: ADC thread
[00:00:00.256,713] <inf> flash_thread: **Flash access example using both NVS and settings API
[00:00:00.256,713] <inf> flash_thread: NVS flash offset f8000
[00:00:00.256,835] <inf> adc_thread: Voltage0: 2992 mV / 3405
[00:00:00.256,835] <inf> adc_thread: Voltage1: 258 mV / 294
[00:00:00.258,697] <inf> fs_nvs: 3 Sectors of 4096 bytes
[00:00:00.258,697] <inf> fs_nvs: alloc wra: 0, fb0
[00:00:00.258,697] <inf> fs_nvs: data wra: 0, 3e
[00:00:00.258,758] <inf> flash_thread: Key value in NVS:
                          ff fe fd fc fb fa f9 f8                          |........
[00:00:00.258,789] <inf> flash_thread: *** Reboot counter in NVS: 1 ***
[00:00:00.258,789] <inf> flash_thread: save new reboot counter by NVS API
[00:00:00.261,077] <inf> fs_nvs: 4 Sectors of 4096 bytes
[00:00:00.261,077] <inf> fs_nvs: alloc wra: 0, fa8
[00:00:00.261,077] <inf> fs_nvs: data wra: 0, 40
[00:00:00.261,108] <inf> flash_thread: settings subsys initialization: OK.
[00:00:00.261,138] <inf> flash_thread: Load all key-value pairs using registered handlers
[00:00:00.261,199] <inf> flash_thread: set handler name=boot_cnt, len=4
[00:00:00.261,230] <inf> flash_thread: *** Reboot counter in Settings: 1 ****
[00:00:00.261,291] <inf> flash_thread: set handler name=key, len=8
[00:00:00.261,322] <inf> flash_thread: Key value in Settings:
                          30 31 32 33 34 35 36 37                          |01234567
[00:00:00.261,352] <inf> flash_thread: save new reboot counter by Settings API
[00:00:20.256,896] <inf> adc_thread: ADC thread
[00:00:20.256,958] <inf> adc_thread: Voltage0: 2997 mV / 3411
```

## 3.10 Raw nrfx driver example

Many users want to invoke nrfx drivers API directly so that they can skip Zephyr layers to speed up the access or not to use kconfig or deviceTree to have a back compatibility of his old projects. This example shows how to call SPI and RTC bottom layer driver API directly without the awareness of Zephyr RTOS.

Follow the steps below to build this example.

- Replace the content of nrf53_ble/appcore/prj.conf with that of nrf53_ble/resources/10appcore_prj_raw_nrfx.conf. be aware to keep the name: prj.conf
- Replace the content of nrf53_ble/appcore/nrf5340dk_nrf5340_cpuapp.overlay with that of nrf53_ble/resources/10nrf5340dk_nrf5340_cpuapp.overlay. be aware to keep the name: nrf5340dk_nrf5340_cpuapp.overlay.

Regarding SPI example, it serves the same function as Section3.5. Please refer to Section3.5 for the testing steps.

Regarding RTC example, it's just the same function as nRF5_SDK\examples\peripheral\rtc: after 5 seconds, LED2 is turned on by RTC ISR.

The logging is shown below.

```
30 31 32 33 34 35 36 37                    |01234567
[00:00:00.261,444] <inf> flash_thread: save new reboot counter by Settings API
[00:00:05.256,713] <inf> raw_nrfx_thread: raw RTC cc0 evt
[00:00:08.256,683] <inf> main: button2 isr

[00:00:08.256,774] <inf> raw_nrfx_thread: raw spi master thread
[00:00:08.256,835] <inf> raw_nrfx_thread: Transfer completed.
[00:00:08.256,835] <inf> raw_nrfx_thread: Received:
                         4e 6f 72 64 69 63                    |Nordic
[00:00:08.456,909] <inf> raw_nrfx_thread: raw spi master thread
[00:00:08.456,970] <inf> raw_nrfx_thread: Transfer completed.
[00:00:08.456,970] <inf> raw_nrfx_thread: Received:
                         4e 6f 72 64 69 63                    |Nordic
[00:00:08.657,043] <inf> raw_nrfx_thread: raw spi master thread
[00:00:08.657,073] <inf> raw_nrfx_thread: Transfer completed.
[00:00:08.657,104] <inf> raw_nrfx_thread: Received:
                         4e 6f 72 64 69 63                    |Nordic
```

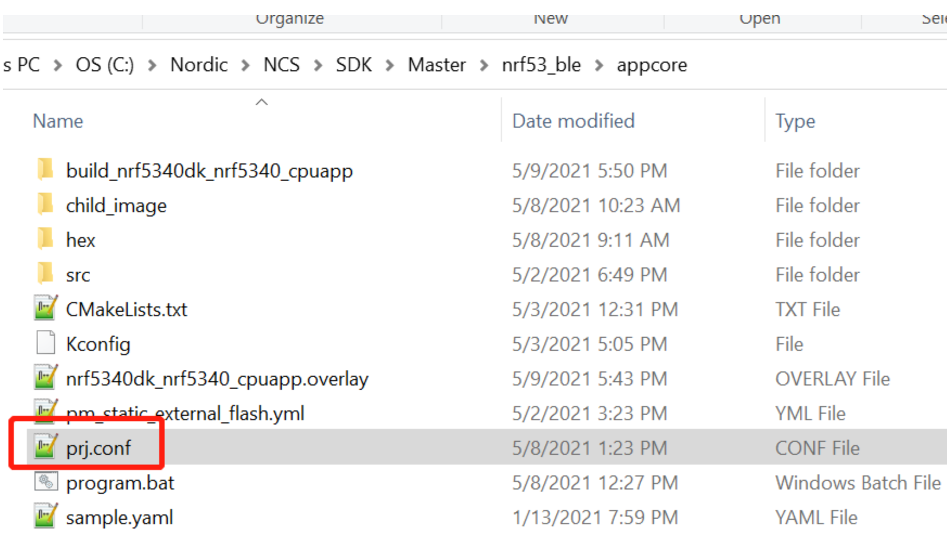
### 3.11 Device power management (PM) example

We can use PM to turn on/off peripherals dynamically to save power consumption. This example is controlled by the macro: CONFIG_PM_DEVICE. device_set_power_state is used to turn on or turn off the peripherals. The related logging is shown below.

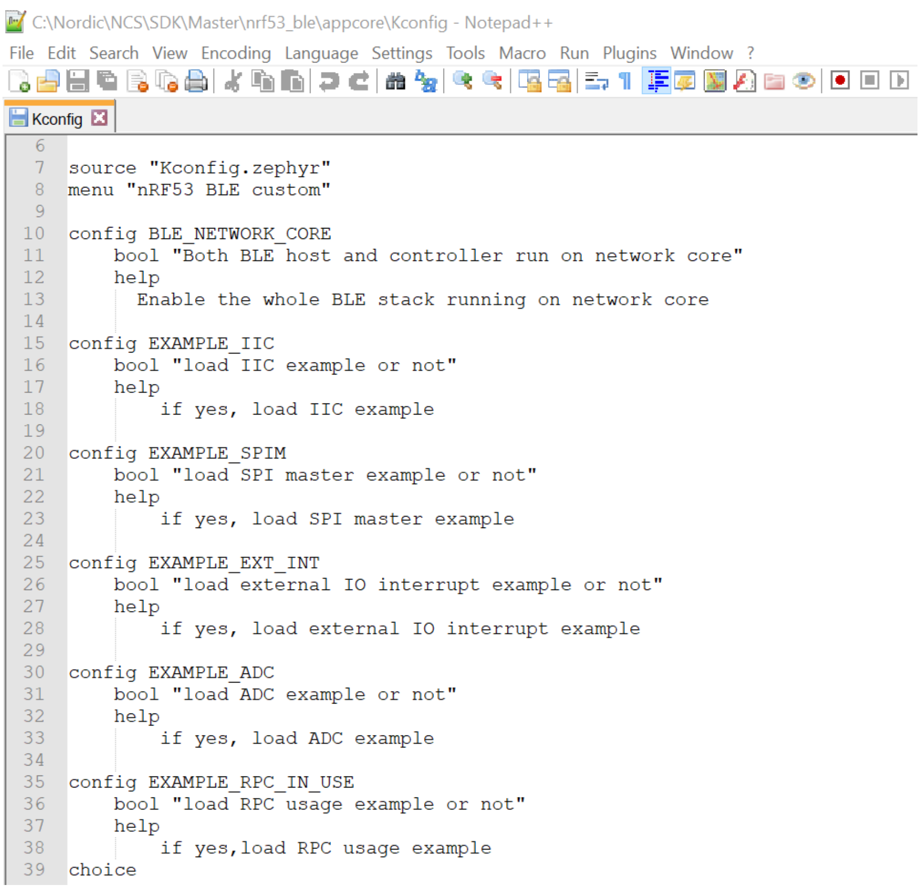
```
[00:00:03.968,231] <inf> main: button3 isr

[00:00:03.968,261] <inf> main: UART1 is in active state. We suspend it
[00:00:03.968,261] <inf> main: ## UART1 is suspended now ##
[00:00:03.968,292] <inf> main: UART0 is in active state. We suspend it
[00:00:11.754,455] <inf> main: button3 isr

[00:00:11.754,486] <inf> main: UART1 is in suspend state. We activate it
[00:00:11.754,486] <inf> main: ## UART1 is actvie now ##
[00:00:11.754,516] <inf> main: UART0 is in suspend state. We activate it
[00:00:11.754,516] <inf> main: ## UART0 is actvie now ##
[00:00:15.257,202] <inf> adc_thread: ADC thread
[00:00:15.257,263] <inf> adc_thread: Voltage0: 2998 mV / 3412
[00:00:15.257,263] <inf> adc_thread: Voltage1: 302 mV / 344
[00:00:15.266,967] <inf> adc_thread: Voltage0: 2998 mV / 3412 async
[00:00:15.266,998] <inf> adc_thread: Voltage1: 302 mV / 344 async
[00:00:20.257,354] <inf> adc_thread: ADC thread
```

### 3.12 kconfig example

The prj.conf below applies to the parent image: app.




In Kconfig we created some new kconfig definitions for your reference. See below.

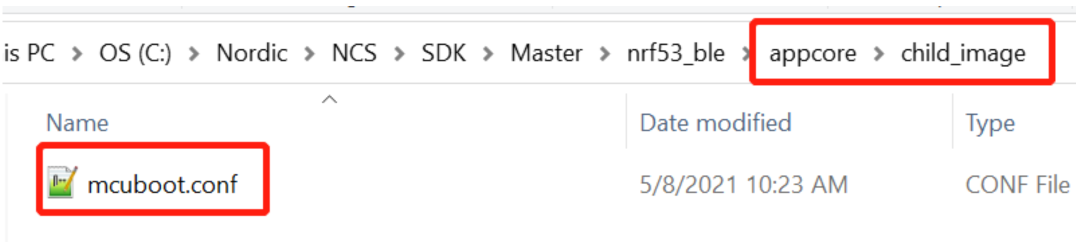



In NCS, we have many child images. Each child image has his own name. See below for the details.

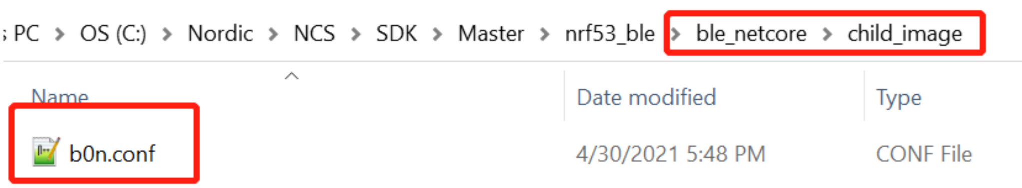
- mcuboot. A third-party bootloader which may be upgradable.
- b0. Nordic in house immutable bootloader.
- spm. Nordic in house Cortex-M33 boot program for non-secure application.
- tfm (trusted-firmware-m). A third-party Cortex-M33 boot program for non-secure application.

- b0n. Nordic in house immutable bootloader running on network core.
- hci_rpmsg. Bluetooth LE controller running on network core.
- 802154_rpmsg. 802.15.4 controller running on network core.

To apply kconfig to each child image, you need to create a child_image folder and a conf file naming after the child image name. Below mcuboot.conf would apply to the MCUboot child image.




And below b0n.conf would apply to the B0n child image.




## 3.13 DeviceTree example

In nrf5340dk_nrf5340_cpuapp.overlay, we demonstrate how to enable a peripheral, how to disable a peripheral, how to delete a Zephyr node, how to delete a property of a node, how to adjust RAM partitions in app image.

By default, nrf5340DK turns on SPI2. We disable it by:

```
&spi2 {
     status = "disabled";
};
```

To turn on I2C2, we do it by:

```
&i2c2 {
    status = "okay";
    compatible = "nordic,nrf-twim";
    sda-pin = <4>;
    scl-pin = <3>;
};
```

By default, Button4(sw3) is defined by application core. We would like to use this button in network core. Thus, we need to delete its definition first, which is done by:

```
/* delete button4 to initialize the related pin for network core use */
/ {
 aliases {
  /delete-property/ sw3;
 };
};

/delete-node/ &button3;
```

By default, RTS and CTS are defined in UART0. We can remove these 2 properties by:

```
&uart0 {
    compatible = "nordic,nrf-uarte";
    status = "okay";
    current-speed = < 115200 >;
    tx-pin = < 0x14 >;
    rx-pin = < 0x16 >;
    /* delete rts-pin&cts-pin to release the related 2 pins for other uses */
    /delete-property/ rts-pin;
    /delete-property/ cts-pin;
};
```

By default, RAM size 0x70000 is reserved for application usage. We increase it to 0x7A000 as below.

```
/* adjust the RAM size of application image */
&sram0_image{
     reg = < 0x20000000 0x7A000 >;
};

&sram0_s{
     reg = < 0x20000000 0x7A000 >;
};
```

By default, shared RAM memory size is 0x10000. We adjust it to 0x4000 like below.

```
/delete-node/ &sram0_shared;

/ {
    chosen {
        /* shared memory reserved for the inter-processor communication */
        zephyr,ipc_shm = &sram0_shared;
    };

    reserved-memory {
        #address-cells = <1>;
        #size-cells = <1>;
        ranges;
        sram0_shared: memory@0x2007A000 {
            /* SRAM allocated to shared memory */
            reg = <0x2007A000 0x4000>;
        };
    };
};
```

By default, the size of Storage area is 0x8000, we adjust it to 0x4000. If PM(partition manager) is enabled, we just need to define the following kconfig to achieve it.

*CONFIG_PM_PARTITION_SIZE_SETTINGS_STORAGE=0x4000*

If PM is not enabled, we do the following define.

```
/delete-node/ &storage_partition;
&flash0{
        partitions {
            compatible = "fixed-partitions";
            #address-cells = < 0x1 >;
            #size-cells = < 0x1 >;

            storage_partition: partition@fc000 {
                label = "storage";
                reg = < 0xfc000 0x4000 >;
            };
        };
};
```

## 3.14 Static memory partition example

If there are many separate images on SoC, we need partition Flash memory for each image. In NCS, PM(partition manager) is used to partition the memory. By default, PM would partition the memory automatically without any user input. However, you can designate a static yml file to instruct PM how to partition the memory. In this example, we use pm_static_external_flash.yml to do the user-assigned memory allocation. To edit this file, be aware of the following notes.

1. mcuboot size must be a multiple of 16kB.
2. The size of mcuboot_secondary must equal to the size of mcuboot_primary.
3. If external Flash is enabled, the sum of all partitions on external Flash must equal to the physical size of the external Flash. In this example, the sum of mcuboot_secondary size and external_flash size must be the same as CONFIG_PM_EXTERNAL_FLASH_SIZE (Note: the external Flash has 2 partitions: mcuboot_secondary and external_flash in this example).
4. sram_primary (0x7e000) is the maximum RAM size of application image, where 0x4000 is reserved for shared memory. Thus, 0x7A000 becomes the maximum available memory for application image.
5. pcd_sram is used to DFU network core image. In fact, we do not use 8kB at all. You can adjust it if necessary.

## 3.15 IO assignment between 2 cores

IO assignment can only be done in secure image of application core. P0.09 (Button4 pin) is re-assigned to network core by following code lines.

```
static void assign_io_to_netcore(void)
{
    NRF_P0_S->PIN_CNF[9] = (GPIO_PIN_CNF_MCUSEL_NetworkMCU <<
                            GPIO_PIN_CNF_MCUSEL_Pos);
}
```

Since dk_buttons_init is used in application core, we must delete button4 deviceTree definitions to make P0.09 work on network core. See below for the details.

```
/* delete button4 to initialize the related pin for network core use */
/ {
 aliases {
  /delete-property/ sw3;
 };
};

/delete-node/ &button3;
```

Besides IO assignment, please note that shared memory definitions of application core and network core must be identical, which is shown below.

```
/delete-node/ &sram0_shared;

/ {
    chosen {
        /* shared memory reserved for the inter-processor communication */
        zephyr,ipc_shm = &sram0_shared;
    };

    reserved-memory {
        #address-cells = <1>;
        #size-cells = <1>;
        ranges;
        sram0_shared: memory@0x2007A000 {
            /* SRAM allocated to shared memory */
            reg = <0x2007A000 0x4000>;
        };
    };
};
```