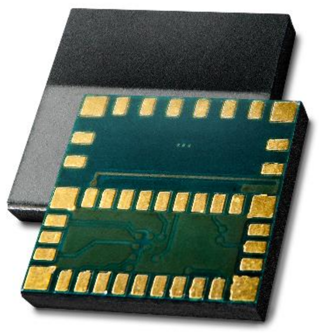
Application Note AN171102

ISP1302-BN Serialization User Guide



Introduction

Scope

This application note describes how to set up BLE serialization between an ISP1302-BN module and a main application CPU.

Contents

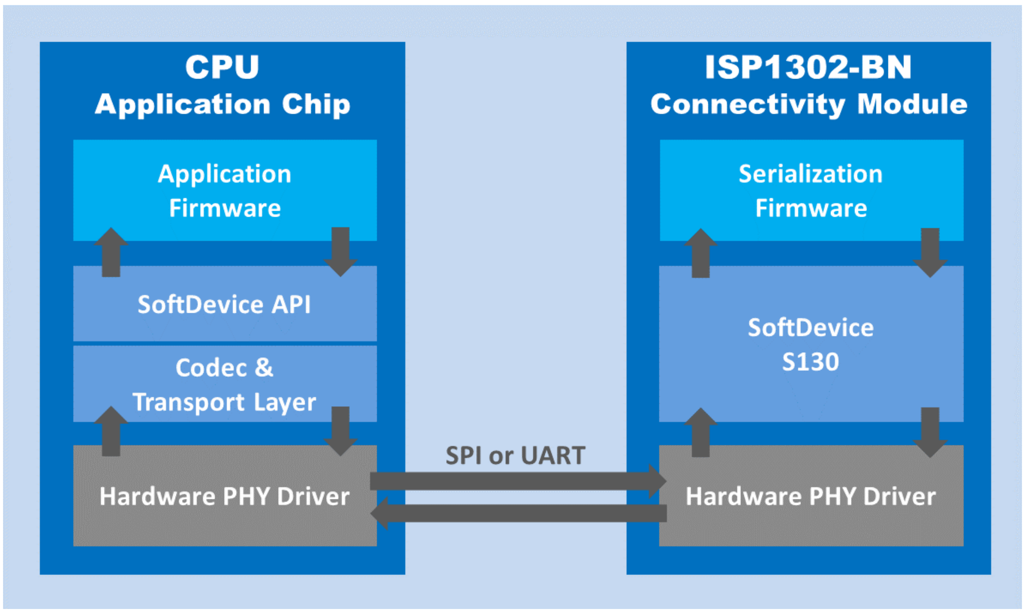
1. Serialization Definition

1.1. Generalities

Some applications need the addition of a BLE connectivity while developers want to keep their system architecture based on a specific CPU. Other applications are using BLE SoftDevice that cannot be ported to the ISP1302, for example because they use specific peripherals or need more resources like RAM, flash memory, or CPU speed.

In this case using the ISP1302-BN module preprogrammed with the Nordic BLE Serialization API may be the solution. Serialization makes it possible to place a Bluetooth application on an application module and connect it to a connectivity module that runs the SoftDevice.

The serialization libraries and the connectivity example simplify the serialization of an existing application, because only limited modifications are needed in the application itself.



1.2. Application module

The application module runs a serialized application, where the SoftDevice is replaced by a commands encoder and events decoder.

In this Application Note, a standard ISP1302-BM is used here merely as a demonstration device. After porting the hardware driver to the selected PHY layer, one can use a different microcontroller.

To port serialization libraries to another microcontroller, refer to the Nordic Infocenter:
http://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v11.0.0%2Fserialization_porting_guide.html

The application chip does not need a SoftDevice. It is replaced by a Codec that implements the SoftDevice API.

All function calls to the Codec are serialized and transmitted to the connectivity chip using the transport layer drivers (UART or SPI).

1.3. Connectivity module

The connectivity module is an ISP1302-BN that is pre-programmed with a SoftDevice. It decodes serialized SoftDevice commands from the application chip and issues the corresponding call to the SoftDevice.

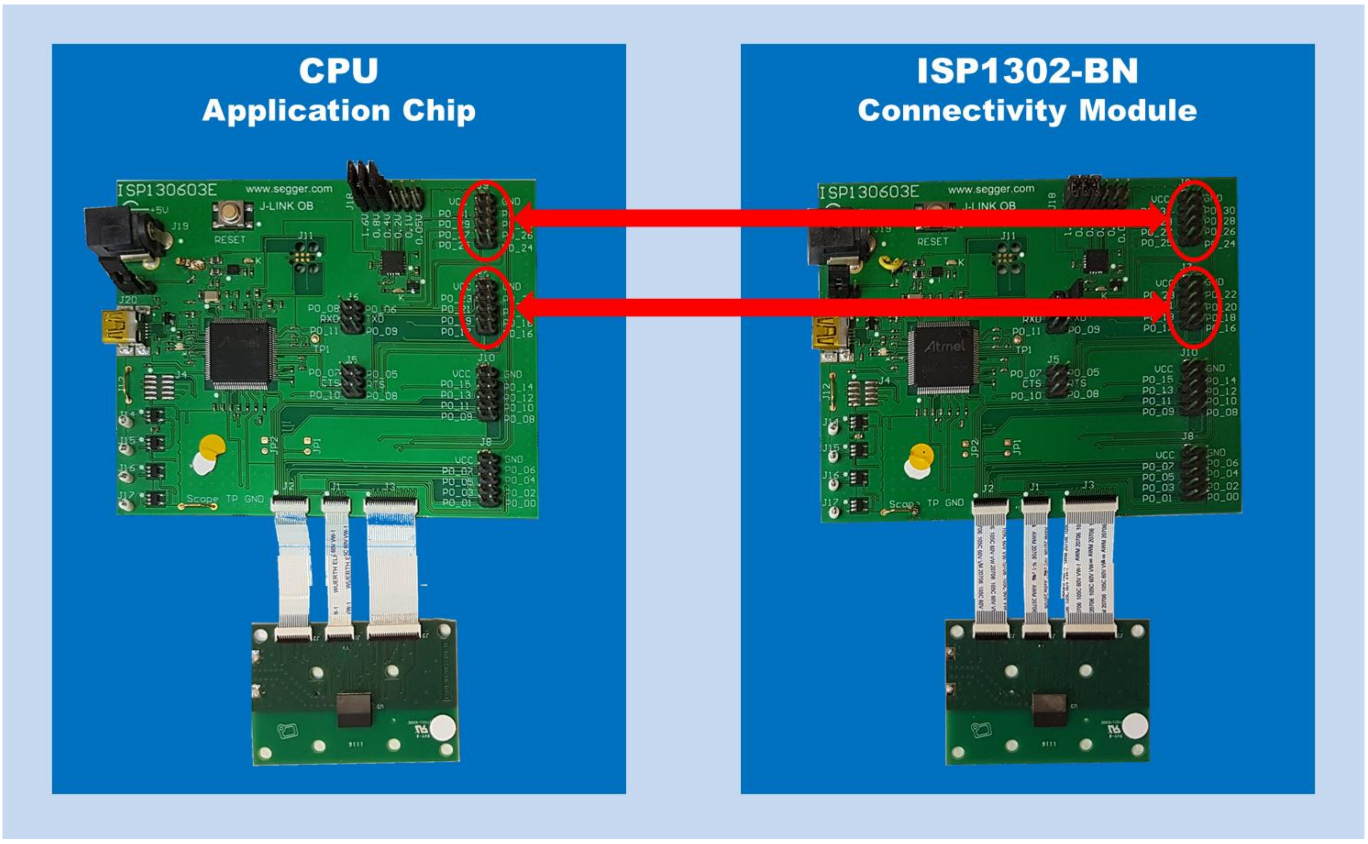
Any event from the SoftDevice is encoded by the Codec that implements the SoftDevice API. Through the transport layer, it is then transmitted to the application chip, where it is decoded and passed to the application.

In addition to the SoftDevice, the connectivity chip must be programmed with the connectivity software.

2. Hardware setup

The serialization setup supports two physical transport interfaces for BLE: UART and SPI. We will see an example with SPI.

The following figure illustrates how to connect two test boards (with their interface board) as an application board and a connectivity board supplying an SPI connection:



Connect all the following pins on the 2 highlighted connectors:
GND, P0_19, P0_20, P0_21, P0_22, P0_24 and P0_25

3. Software setup

3.1. Software Compliance

The following example uses the nRF5 SDK v11.0.0 and the SoftDevice S130 v2.0.0.

3.2. Connectivity Module Side

Prepare the connectivity module by performing the following steps.

1. Connect the interface board (connectivity side) to the computer.
2. Program the Softdevice on the connectivity module.
3. In Keil, open the Connectivity Example for the SPI physical transport layer.

```
<InstallFolder>\examples\ble_central_and_peripheral\ble_connectivity
\ble_connectivity\pca10028\ser_s13x_spi
```

4. Modify the board definition file (pca10028.h or a custom file you have created) and change the pinout definition (This is an example of pinout, others pins can also be used, bear in mind than not all nRF51 pins are available on the ISP1302).

```
#define SER_CON_SPIS_SCK_PIN      19    // SPI SCK signal.
#define SER_CON_SPIS_MOSI_PIN     20    // SPI MOSI signal.
#define SER_CON_SPIS_MISO_PIN     21    // SPI MISO signal.
#define SER_CON_SPIS_CSN_PIN      22    // SPI CSN signal.
#define SER_CON_SPIS_RDY_PIN      24    // SPI READY GPIO pin number.
#define SER_CON_SPIS_REQ_PIN      25    // SPI REQUEST GPIO pin number.
```

5. Modify the board definition file (pca10028.h or a custom file you have created) and change the low frequency clock source to RC. This is because the ISP1302 does not embed a 32 kHz crystal).

```
#define NRF_CLOCK_LFCLKSRC     {.source       = NRF_CLOCK_LF_SRC_RC,      \
                                .rc_ctiv      = 1,                        \
                                .rc_temp_ctiv  = 0,                       \
                                .xtal_accuracy = NRF_CLOCK_LF_XTAL_ACCURACY_20_PPM}
```

6. Compile the application and download the created.hex file to the connectivity module.

3.3. Application Module Side

The application module does not need a SoftDevice. Prepare the application module by performing the following steps:

1. Connect application module by performing the following steps.

2. In Keil, open one of the serialized example projects. The serialized version is located in the ser_s130_spi folder. Choose the example project for the same physical transport layer as on the connectivity board.

Example	Physical transport layers
Alert Notification Application	UART, SPI, HCI
Beacon Transmitter Sample Application	UART, SPI, HCI
Blood Pressure Application	UART, SPI, HCI
Cycling Speed and Cadence Application	UART, SPI, HCI
Glucose Application	SPI
HID Keyboard Application	SPI
Heart Rate Application	SPI
Health Thermometer Application	UART, SPI, HCI
Power Profiling Application	UART, SPI, HCI
Running Speed and Cadence Application	UART, SPI, HCI
Apple Notification Center Service (ANCS) Client Application	SPI
Direct Test Mode	UART, SPI, HCI
BLE Heart Rate Collector Example	SPI
BLE Multi-link Example	UART, SPI, HCI

In this example, we choose the Heart Rate Application with SPI.

3. In this example the application board is an ISP1302 test board so we need to modify the board definition file to change the pinout.

```
#define SER_APP_SPIM0_SCK_PIN     19    // SPI clock GPIO pin number.
#define SER_APP_SPIM0_MOSI_PIN    20    // SPI Master Out Slave In GPIO pin number
#define SER_APP_SPIM0_MISO_PIN    21    // SPI Master In Slave Out GPIO pin number
#define SER_APP_SPIM0_SS_PIN      22    // SPI Slave Select GPIO pin number
#define SER_APP_SPIM0_RDY_PIN     24    // SPI READY GPIO pin number
#define SER_APP_SPIM0_REQ_PIN     25    // SPI REQUEST GPIO pin number
```

Note: Normally the reset pin should also be configured (SER_CONN_CHIP_RESET_PIN). This pin should be connected to the SWDIO-nRESET pin of the connectivity module. We don't use it in this example.

4. In this example the application module is an ISP1302 so we need to modify the serialization library in order to set low frequency source to RC (it uses by default the XTAL clock source). In the file ser_app_hal_nrf51.c, modify the function ser_app_hal_hw_init in order to set the clock source (see below).

```
uint32_t ser_app_hal_hw_init(ser_app_hal_flash_op_done_handler_t handler)
{
    nrf_gpio_cfg_output(CONN_CHIP_RESET_PIN_NO);

    NRF_CLOCK->LFCLKSRC = (CLOCK_LFCLKSRC_SRC_RC << CLOCK_LFCLKSRC_SRC_Pos);
    NRF_CLOCK->EVENTS_LFCLKSTARTED = 0;
    NRF_CLOCK->TASKS_LFCLKSTART    = 1;

    while (NRF_CLOCK->EVENTS_LFCLKSTARTED == 0)
    {
        //No implementation needed.
    }


    NRF_CLOCK->EVENTS_LFCLKSTARTED = 0;
    m_flash_op_handler = handler;
    return NRF_SUCCESS;
}
```

5. Compile the application and download the created.hex file to the application module.

3.4. Verification

Power-up both modules at the same time. Check if you can connect with your smartphone using nRF Toolbox or nRF Connect) to the device called Nordic_HRM.