nRF8001 Development Kit

User Guide v1.1

December 2011

Liability disclaimer

Life support applications

Contact details

For your nearest dealer, please see http://www.nordicsemi.com

Receive available updates automatically by subscribing to eNews from our homepage or check our website regularly for any available updates.

Main office:

Otto Nielsens veg 12
7004 Trondheim
Norway
Phone: +47 72 89 89 00
Fax: +47 72 89 89 89
www.nordicsemi.com

DNV
NORWEGIAN ACCREDITATION
No. S03
NS-EN ISO 9001 CERTIFICATED FIRM

Revision History

Date	Version	Description
December 2010	1.0	
December 2011	1.1	• Added chapter 9 on page 27 • Removed section 8.1.3 "Porting ACI Transport Layer to another application processor"

Contents

1 Introduction

The nRF8001 Development Kit is your first step in implementing a *Bluetooth*® low energy product. It is designed for use with the nRFgo Starter Kit, and contains all the necessary hardware to begin developing an application. An SDK (Software Development Kit) consisting of firmware libraries, example applications and computer tools also comes with this kit (available for download at www.nordicsemi.com, see the Getting Started Guide for details).

This user guide describes a typical setup, how to use the hardware, and introduces you to the software development tools.

1.1 Who should read this user guide?

This user guide is aimed at anyone who wants to develop a *Bluetooth* low energy product. This requires knowledge of embedded software development and some hardware knowledge.

You may also be interested in this user guide if you are conducting radio peformance tests (see chapter 5 on page 11).

To fully understand this user guide, you will need knowledge of the *Bluetooth* low energy specification and terminology.

Before you begin reading this user guide, you should read the nRF8001 Development Kit's Getting Started Guide.

1.2 Minimum requirements

Below are the minimum requirements for the nRF8001 Development Kit:

- nRFgo Starter Kit
- Computer with 2 USB and 1 RS232 ports
- Windows XP and Windows 7

1.3 Documentation

The following documentation is recommended reading:

Document	Description
nRF8001 Development Kit Getting Started Guide	A step by step guide on how to get started with the Development Kit.
nRF8001 Preliminary Product Specification	Defines the nRF8001 hardware and electrical specifications and describes the ACI (Application Controller Interface).
nRFgo Starter Kit User Guide	Contains information about the nRFgo Motherboard (nRF6310).
nRFgo Studio help	nRFgo Studio help documents the functionality of the nRFgo Studio.

1.4 Writing Conventions

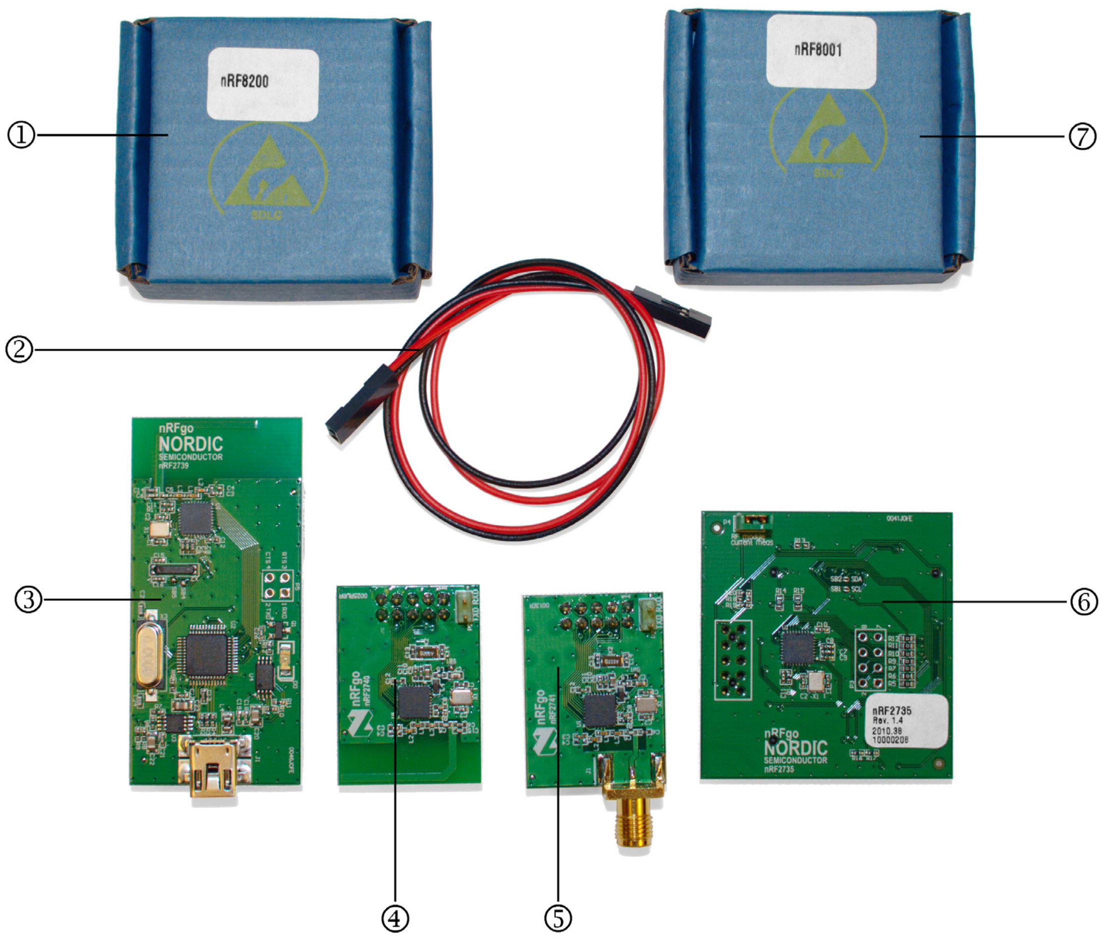
This user guide follows a set of typographic rules that makes the document consistent and easy to read. The following writing conventions are used:

- Commands are written in Courier New.
- File names and User Interface components are written in **bold**.
- Cross references are underlined and highlighted in blue.

2 Kit content

The nRF8001 Development Kit consists of hardware and software components, firmware libraries, and example applications.

2.1 Hardware components



j 5 nRF8200 samples

k 1 patch cable

l **nRF2739:** *Bluetooth* low energy master emulator

m **nRF2740:** nRF8001 module with PCB antenna

n **nRF2741:** nRF8001 module with SMA connector

o **nRF2735:** Carrier board with an nRF8200 application processor

p 5 nRF8001 samples

Figure 1. Hardware components

For more information on the hardware modules see Chapter 7 on page 16.

2.2 Software and documentation components

The SDK (Software Development Kit) can be downloaded from a Nordic MyPage account at www.nordicsemi.com. The SDK is delivered in the form of an installer file containing the following components:

- **nRFgo Studio v1.8:** Computer software for setting up nRF8001. Also controls the nRFgo Motherboard and programs the nRF8200 on the carrier board (nRF2735).
- **SDK:** Source code, compiled firmware, and API documentation
- **Master Control Panel:** Computer software for the master emulator (nRF2739)
- **nRFprobe:** a debugger plugin for Keil C51
- **nRF8001 Development Kit User Guide**

3 Typical setup

The Development Kit is designed for use with the nRFgo Starter Kit. Figure 2. shows the relationship between the Development Kit hardware and software components and the nRFgo Starter Kit.

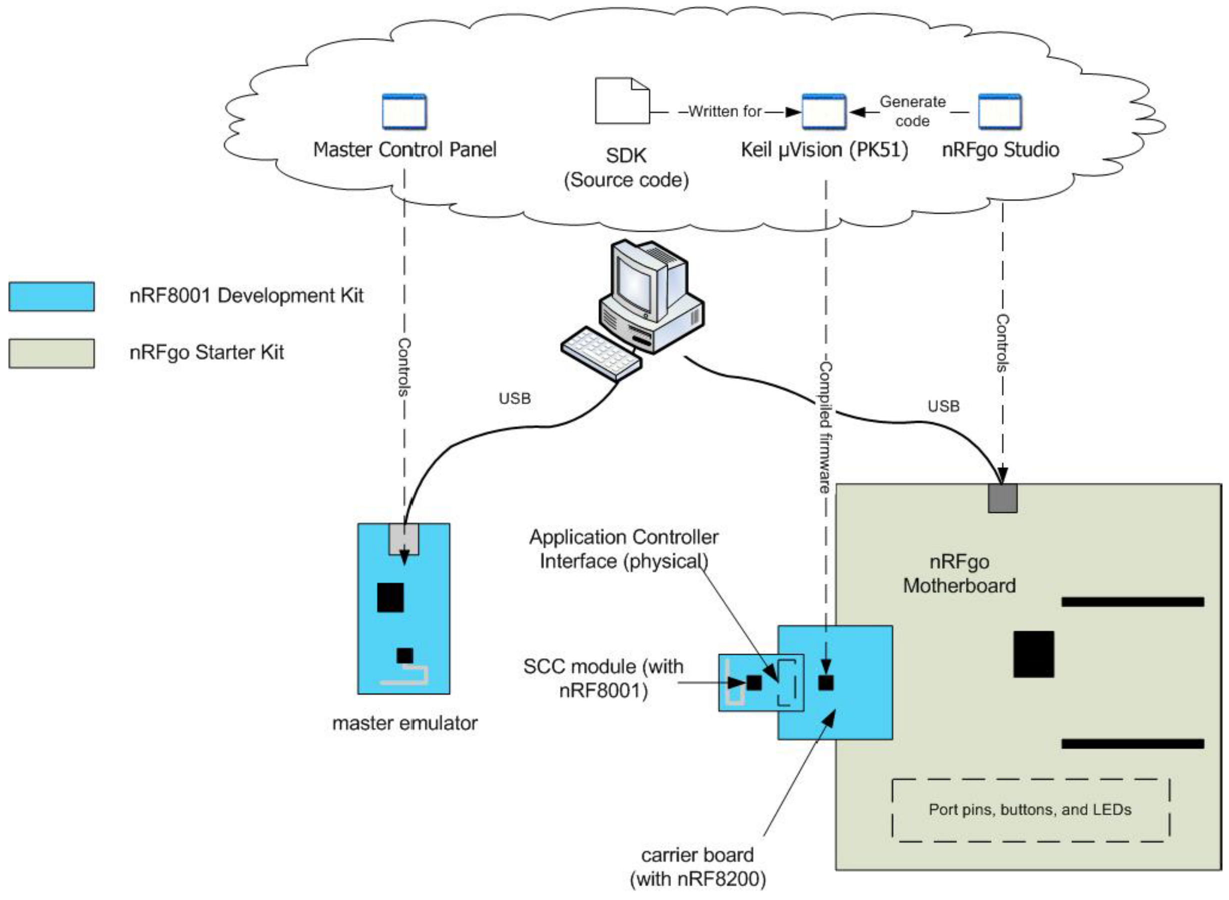


Figure 2. Typical setup

3.1 nRFgo Studio

Before you can start writing any application code for your *Bluetooth* low energy product you need to decide what kind of user data your product needs to transfer. For interoperability with products from other vendors you need to follow set formats specified by *Bluetooth*. These formats are defined in *Bluetooth* low energy services and profiles and decide how, for instance, a temperature sensor reports its data. From the *Bluetooth* menu in nRFgo Studio you can choose and combine the necessary services for your product:

- Setting the static parameters for nRF8001:
  - **Setup of GATT services, profiles:** lets you decide which *Bluetooth* low energy profiles you want to include. The **GATT Services** view allows you to select the services you want to include in your application. You can also define your own services and characteristics.
  - **GAP setup:** lets you tune the parameters of the Generic Access Profile (GAP), for example, timing parameters.
  - **Hardware setup:** lets you select and configure the nRF8001 hardware features you want to use. Make the appropriate choices according to your product design.

When you have the setup you want, you can generate source files containing the setup code. The files generated by nRFgo Studio are used by the ACI library (lib_aci) in the SDK to configure nRF8001 before use. This allows you to focus on the application code rather than spending time ensuring the *Bluetooth* setup details are correct.

You can also conduct *Bluetooth* low energy tests and program the hardware from the *Bluetooth* menu in the nRFgo Studio:

- Testing the RF PHY using Direct Test Mode.
- Programming nRF8200 and controlling the nRFgo Motherboard.

Please see the nRFgo Studio's online help for detailed instructions.

3.2 SDK

The SDK contains source code for developing applications on the application processor (nRF8200). It includes working examples, and contains libraries that are portable to other microcontroller platforms. The examples are written in C, delivered precompiled, and can be programmed on nRF8200 using nRFgo Studio. Project files for Keil µVision (IDE) are also delivered with the SDK.

More details on the SDK code can be found in chapter 8 on page 23 and the SDK's online help file (all source code is documented in this online help file).

3.3 Master Control Panel

The Master Control Panel is used with the master emulator (nRF2739) to create a *Bluetooth* low energy peer device for nRF8001. With the Master Control Panel you can:

- scan for advertisers
- connect with nRF8001
- send/receive data

Two additional components are installed in conjunction with the Master Control Panel:

- Visual C++ 2008 Redistributables: Files needed for the Master Control Panel, unless .Net v3.5 has already been installed
- Drivers for the master emulator (Usb2Spi)

3.4 Keil C51 compiler (not included in this kit)

The source code is written in C and developed using the Keil PK51 Professional Developer's Kit. You can buy this kit from Keil or download the C51 Evaluation Software at www.keil.com.

To debug software for the application processor (nRF8200) you must also install nRFprobe, the debugger plugin for products from Nordic Semiconductor. The debugger integrates with Keil's µVision IDE. Download nRFprobe from www.nordicsemi.com/update.

The SDK installer checks for existing installations of Keil's C51 compiler on the default installation path. If an installation is found, the following files are copied onto the compiler's include path:

- **stdint.h**
- **stdbool.h**
- Header files with register definitions

These files are necessary for compiling the examples in the SDK. If you installed the SDK first, you can run the installer again:

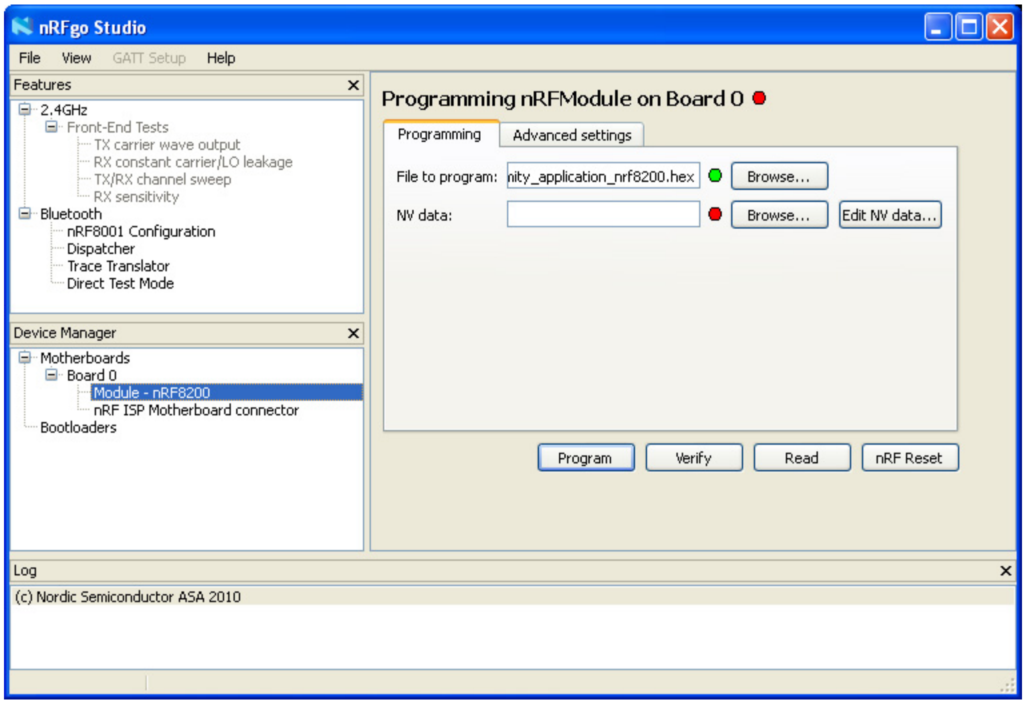
1. On the **Choose components** page, deselect all items except **Software Development Kit.**
2. Complete the installation.

Note: The Keil C51 compiler must be installed prior to installing the nRFprobe program.

4 Programming the carrier board (nRF2735)

The carrier board (nRF2735) features a microcontroller (nRF8200) that is used for the application firmware. In order to test your own applications or examples from the SDK, you need to program the carrier board. This chapter descripes how to program the carrier board from nRFgo Studio.

1. Make sure the ON/OFF switch (**S9**) on the nRFgo Motherboard is turned off.
2. Plug the carrier board (nRF2735) onto the nRFgo module connector on the nRFgo Motherboard.
3. Connect the nRFgo Motherboard to your computer using a USB cable.
4. Turn the nRFgo Motherboard power on by switching **S9** to **ON**.
5. Start the nRFgo Studio from the Windows **Start** menu - **Programs - Nordic Semiconductor - nRFgo Studio**.
6. In nRFgo Studio, select **Motherboards** in the **Device Manager** pane and then select **Module - nRF8200**.
7. Select the file you want to download to nRF8200 for programming.
8. Click the **Program** button.



Programming nRFModule on Board 0

File to program: nity_application_nrf8200.hex

NV data:

(c) Nordic Semiconductor ASA 2010

Figure 3. nRFgo Studio after the carrier board has been successfully programmed

5 Testing the physical layer with Direct Test Mode

To test the RF performance of the physical (PHY) layer of the *Bluetooth* low energy stack in nRF8001 you can use the Direct Test Mode (DTM) interface. This interface is compliant with the description in the *Bluetooth* Specification, Version 4.0, Volume 6, Part F. It can be used for performance testing, tuning your prototypes and compliance testing.

The DTM is accessed through a dedicated UART interface on nRF8001. This interface is only available when the nRF8001 is in test mode. Please see the nRF8001 Preliminary Product Specification for information on the interface and how to enter test mode.

The DTM is designed for use with *Bluetooth* test equipment. If you don't have a *Bluetooth* tester you can access the interface using this Development Kit and using nRFgo Studio to run the tests.

5.1 Setting up the hardware for testing with Direct Test Mode

1. Mount the nRF8001 SCC module (either nRF2740 or nRF2741) and the carrier board (nRF2735) onto the nRFgo Motherboard.
2. Connect the patch cable between **P15** on the nRFgo Motherboard and **P2** on the nRF8001 SCC module (nRF2740 or nRF2741). Make sure the RXD/TXD labels match for each wire.
3. Connect the nRFgo Motherboard to your computer using a USB cable. Make sure switch **S11** is **ON**.
4. Connect a serial cable from **RS232** to the serial port on your computer.
5. You are now ready to begin testing with Direct Test Mode in nRFgo Studio.

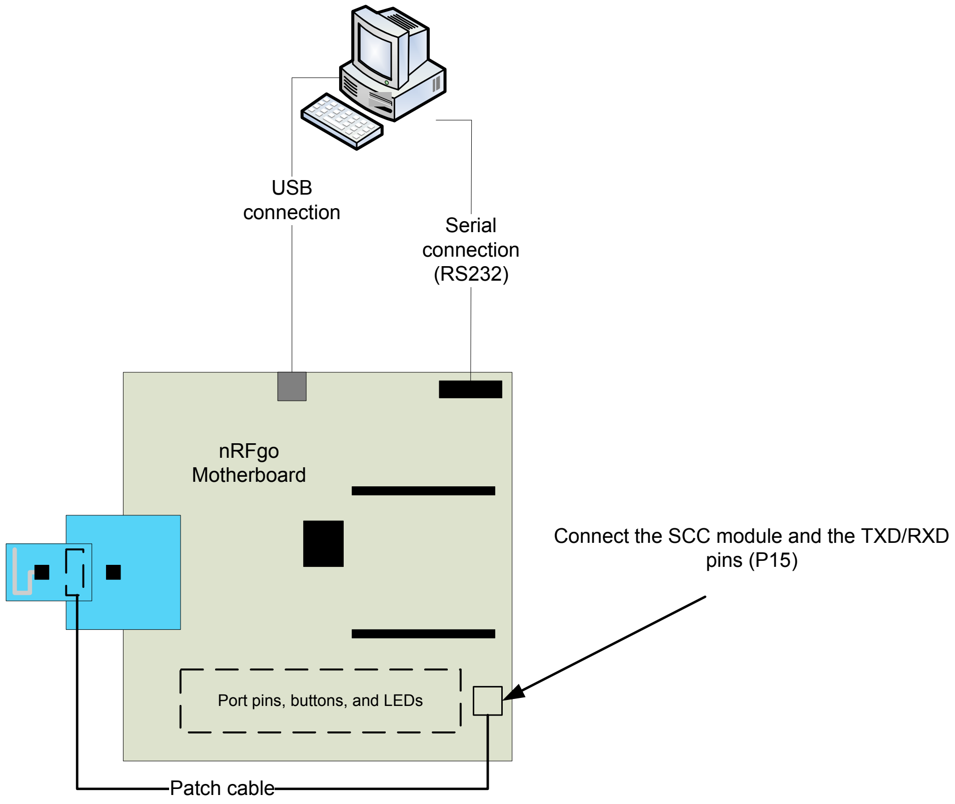

USB connection
Serial connection (RS232)
nRFgo Motherboard
Connect the SCC module and the TXD/RXD pins (P15)
Port pins, buttons, and LEDs
Patch cable

Figure 4. Hardware setup for testing with Direct Test Mode

5.2 Testing the physical layer on your own product

The DTM (Direct Test Mode) must be available on your product so that it can be qualified as a *Bluetooth* low energy product. This means that your application must have a way of setting the nRF8001 in test mode and have connections to the UART interface.

set_in_test_mode is a simple demo example showing how nRF8001 is put into the test state. This is the software that is installed from the Direct Test Mode GUI. From the source code you can see how the ACI command `Test` is used to set the device in test mode.

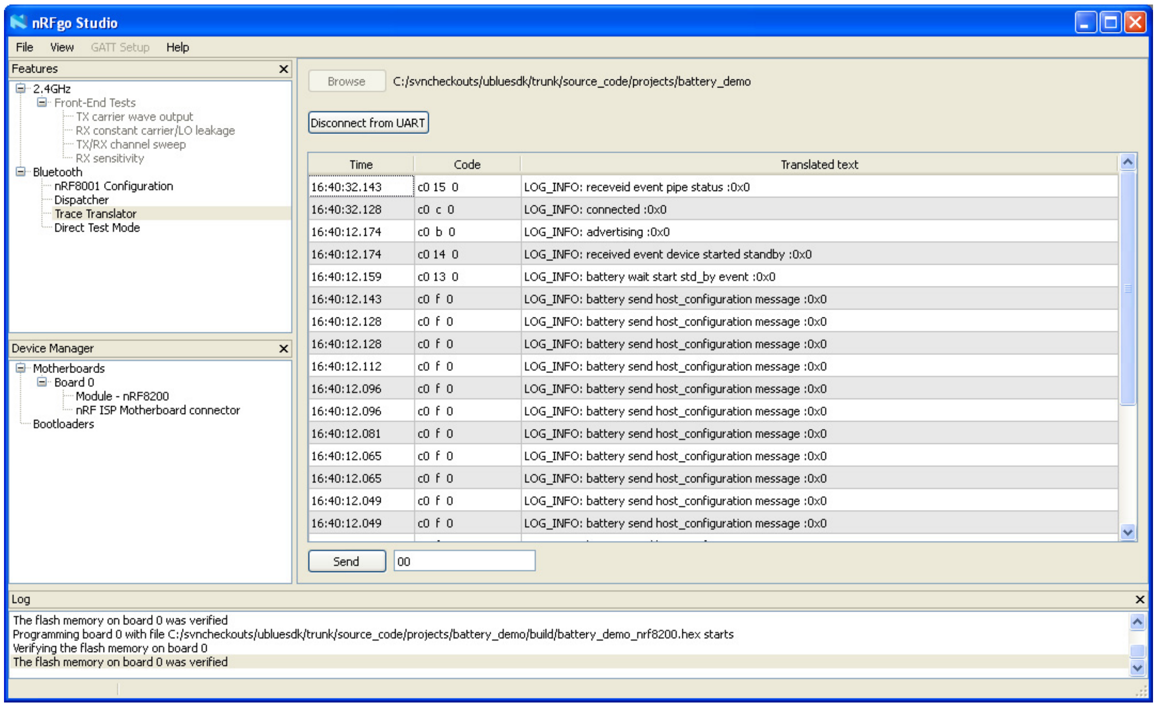
6 Trace library

The nRF8001 SDK has a trace library, called lib_traces, which allows you to trace the execution of your application code. This is useful for debugging, analyzing errors, or to get a better understanding of how the firmware behaves.

Tracing is a common debugging technique, and the trace library implements the functions that allow you to trace. The trace messages are sent through the UART interface, and you can observe them using the computer’s serial port. nRFgo Studio can read the trace messages and translate them to a readable format. The messages are displayed in a list in the order they arrive, giving you a possibilty to observe the execution of the code. In a radio system this is preferrable to the in-circuit debugger, since you only control one end.

The library is designed to send three-byte codes instead of message texts. This is to minimize the transmission time so that the function does not interfere with timing in the application. When the development is finished, it is also possible to disable the functions completely without touching the code.

Using the Trace Translator in nRFgo Studio you can convert these coded messages into a readable format.



C:/svncheckouts/ubluesdk/trunk/source_code/projects/battery_demo

Time	Code	Translated text
16:40:32.143	c0 15 0	LOG_INFO: receveid event pipe status :0x0
16:40:32.128	c0 c 0	LOG_INFO: connected :0x0
16:40:12.174	c0 b 0	LOG_INFO: advertising :0x0
16:40:12.174	c0 14 0	LOG_INFO: received event device started standby :0x0
16:40:12.159	c0 13 0	LOG_INFO: battery wait start std_by event :0x0
16:40:12.143	c0 f 0	LOG_INFO: battery send host_configuration message :0x0
16:40:12.128	c0 f 0	LOG_INFO: battery send host_configuration message :0x0
16:40:12.128	c0 f 0	LOG_INFO: battery send host_configuration message :0x0
16:40:12.112	c0 f 0	LOG_INFO: battery send host_configuration message :0x0
16:40:12.096	c0 f 0	LOG_INFO: battery send host_configuration message :0x0
16:40:12.096	c0 f 0	LOG_INFO: battery send host_configuration message :0x0
16:40:12.081	c0 f 0	LOG_INFO: battery send host_configuration message :0x0
16:40:12.065	c0 f 0	LOG_INFO: battery send host_configuration message :0x0
16:40:12.065	c0 f 0	LOG_INFO: battery send host_configuration message :0x0
16:40:12.049	c0 f 0	LOG_INFO: battery send host_configuration message :0x0
16:40:12.049	c0 f 0	LOG_INFO: battery send host_configuration message :0x0

00

```
The flash memory on board 0 was verified
Programming board 0 with file C:/svncheckouts/ubluesdk/trunk/source_code/projects/battery_demo/build/battery_demo_nrf8200.hex starts
Verifying the flash memory on board 0
The flash memory on board 0 was verified
```

Figure 5. Trace translator in nRFgo Studio

6.1 Using lib_traces in code

The following example shows how to use one of the functions in lib_traces:

```
#include "lib/lib_traces.h"
#include "info_codes.h"
#include "error_codes.h"

void main(void)
{
  LIB_TRACES_LOG_INFO(INFO_CODE_MAIN_STARTED);
  while(1);
}
```

The functions are declared in lib_traces.h, which must be included in your source files. The file is located in the "lib" folder in the SDK. The message codes are defined in the files info_codes.h and error_codes.h. These files are project specific and should be located with the application code.

LIB_TRACES_LOG_INFO() is a macro from the trace library. It takes one parameter: A one-byte information code. INFO_CODE_MAIN_STARTED is defined in info_codes.h as follows:

```
#define INFO_CODE_MAIN_STARTED 0x04 // Main started
```

As result the following three bytes will be sent over the UART: 0xC0 0x04 0x00

- 0xC0: The message type is "Info code"
- 0x04: Value of INFO_CODE_MAIN_STARTED
- 0x00: Optional parameter, not used in this example.

To send the message the compile option DEBUG_ENABLE must be set to CODED_TRACES. Setting DEBUG_ENABLE to NO_TRACES will deactivate the trace functions.

6.2 Hardware setup

The only difference from a normal setup is that the UART interface must be connected to the PC:

1. Connect a serial cable from the RS232 connector on the Motherboard to the PC's serial port.
2. Make a connection from the UART header (P15) to the application processor's UART pins. The UART pins from the nRF8200 can be found on header P8 on the Motherboard; P0.3 (TXD) and P0.4 (RXD).

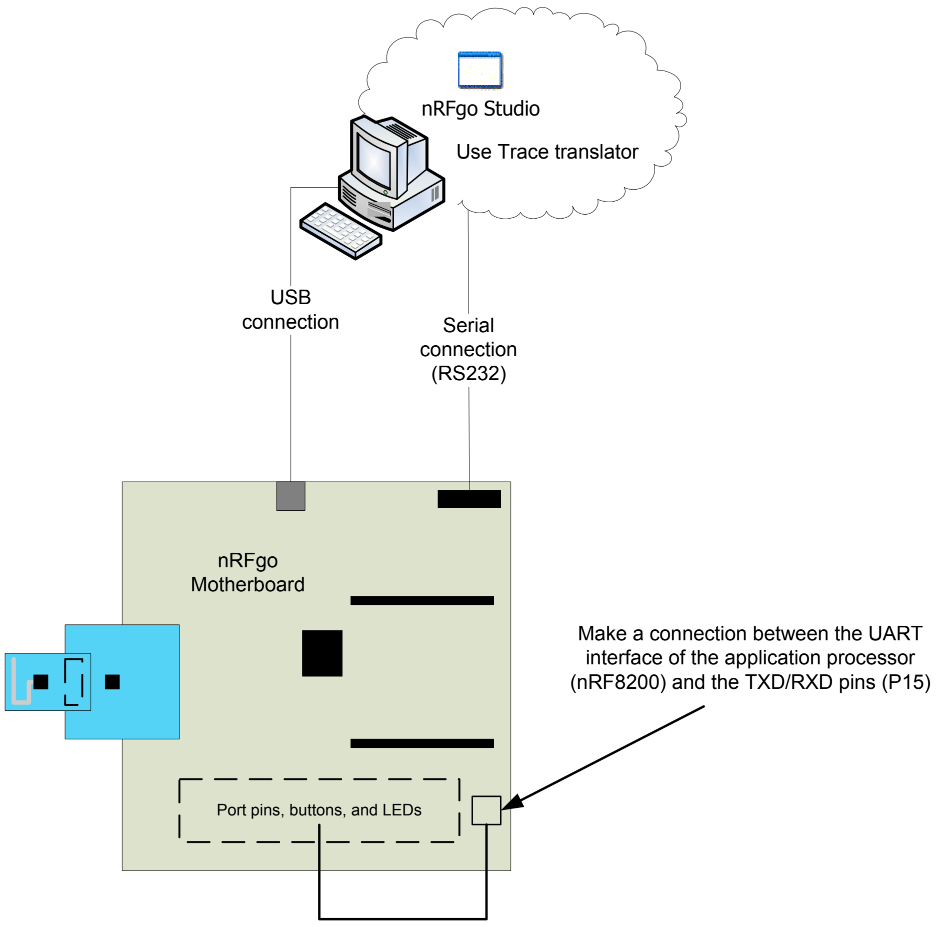

nRFgo Studio
Use Trace translator
USB connection
Serial connection (RS232)
nRFgo Motherboard
Make a connection between the UART interface of the application processor (nRF8200) and the TXD/RXD pins (P15)
Port pins, buttons, and LEDs

Figure 6. Trace library setup

6.3 Interpreting results

An easy way to view the trace is to use the Trace translator in nRFgo Studio. The Trace translator will read the messages from the serial port and interpret them according to the coding definitions. It will also add a timestamp. A timestamp for the example in section 6.1 will be:

Time	Code	Translated text
13:55:04.342	0C 04 00	Main started

The description text is the text in the comment in the info_code.h:

```
#define INFO_CODE_MAIN_STARTED 0x04 // Main started
```

It is therefore important that nRFgo Studio can access the same info_codes.h and error_codes.h that were used when compiling the firmware.

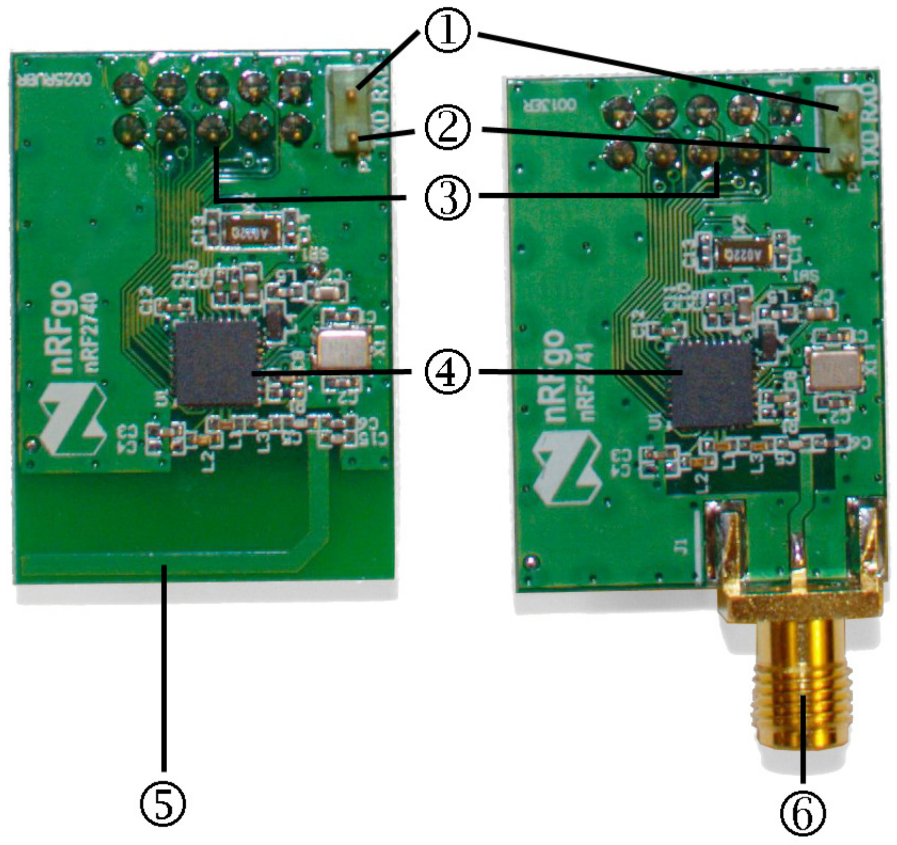
Note: Do not use the timestamp for time measurements if accuracy is required.

7 Hardware description

This appendix describes the SCC (Single Chip Connectivity) modules, carrier board and master emulator.

7.1 SCC modules (nRF2740 and nRF2741)

The two SCC modules (nRF2740 and nRF2741) are identical except nRF2740 has an antenna and nRF2741 has an SMA connector:



①	UART test interface RXD	④	nRF8001
②	UART test interface TXD	⑤	PCB antenna
③	Connector **P1**, ACI interface to the carrier board (nRF2735), located on the bottom side of the SCC modules	⑥	SMA connector

Figure 7. nRF2740 and nRF2741 top sides

Note: Gerber files for the core circuitry PCB layout are available for download from www.nordicsemi.com.

7.1.1 Solder bridge SB1

By default, solder bridge SB1 is shorted. Opening the SB1 solder bridge enables use of the nRF8001 internal step-down DC/DC converter. This feature is enabled from the *Bluetooth* menu in the nRFgo Studio.

7.1.2 Connector P1

For further details about signal description please see the nRF8001 Preliminary Product Specification.

Pin number	Connector P1
1	GND
2	VCC_nRF
3	ACTIVE
4	Not in use
5	SCK
6	MOSI
7	MISO
8	RDYN
9	REQN
10	RESET

Table 1. Pinout for connector P1

7.1.3 Connector P2

This connector is used for the Direct Test Mode interface, which is treated in more detail in chapter 5 on page 11.

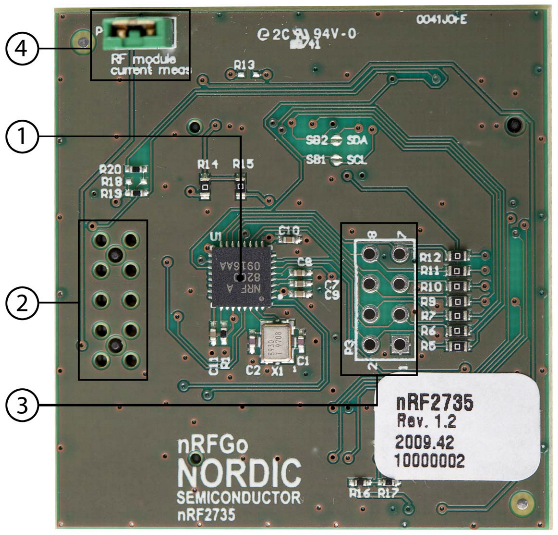
Pin number	Connector P2
1	TXD
2	RXD

Table 2. Pinout for connector P2

7.2 Carrier board module (nRF2735)

The carrier board (nRF2735) contains an application processor (nRF8200) which controls the nRF8001 module and interfaces with the application circuitry.

Note: Connectors **P1** and **P2** interface with the nRFgo Motherboard and are located on the bottom side of the carrier board, see Figure 9. on page 19.



① nRF8200	③ **P3:** Connector for direct access to analog inputs on nRF8200
② **P5:** Connector to SCC modules (nRF2740/nRF2741)	④ **P4:** Jumper for measuring current of SCC modules (nRF2740/nRF2741)

Figure 8. nRF2735, top side

All available nRF8200 I/O pins are routed to the carrier board connectors P1 and P2. Connectors P1 and P2 connects the carrier board to the nRFgo Motherboard. When the carrier board is plugged into the nRF module socket on an nRFgo Motherboard, the nRF8200 I/Os can be accessed from general purpose I/O port connectors P8 and P10 on the nRFgo Motherboard.

	Development Kit Port	
	P0.x	P1.x
nRF8200	P0.x[a]	P1.x[b]
nRFgo Motherboard I/O header	P8	P10

a. Except P0.0, P0.1, P0.2 and P0.6.
b. Except P1.4, P1.5, P1.6 and P1.7.

Table 3. Pinouts for nRF8001 Development Kit

7.2.1 Connectors P1 and P2

Connectors **P1** and **P2** connect the carrier board to the nRFgo Motherboard.

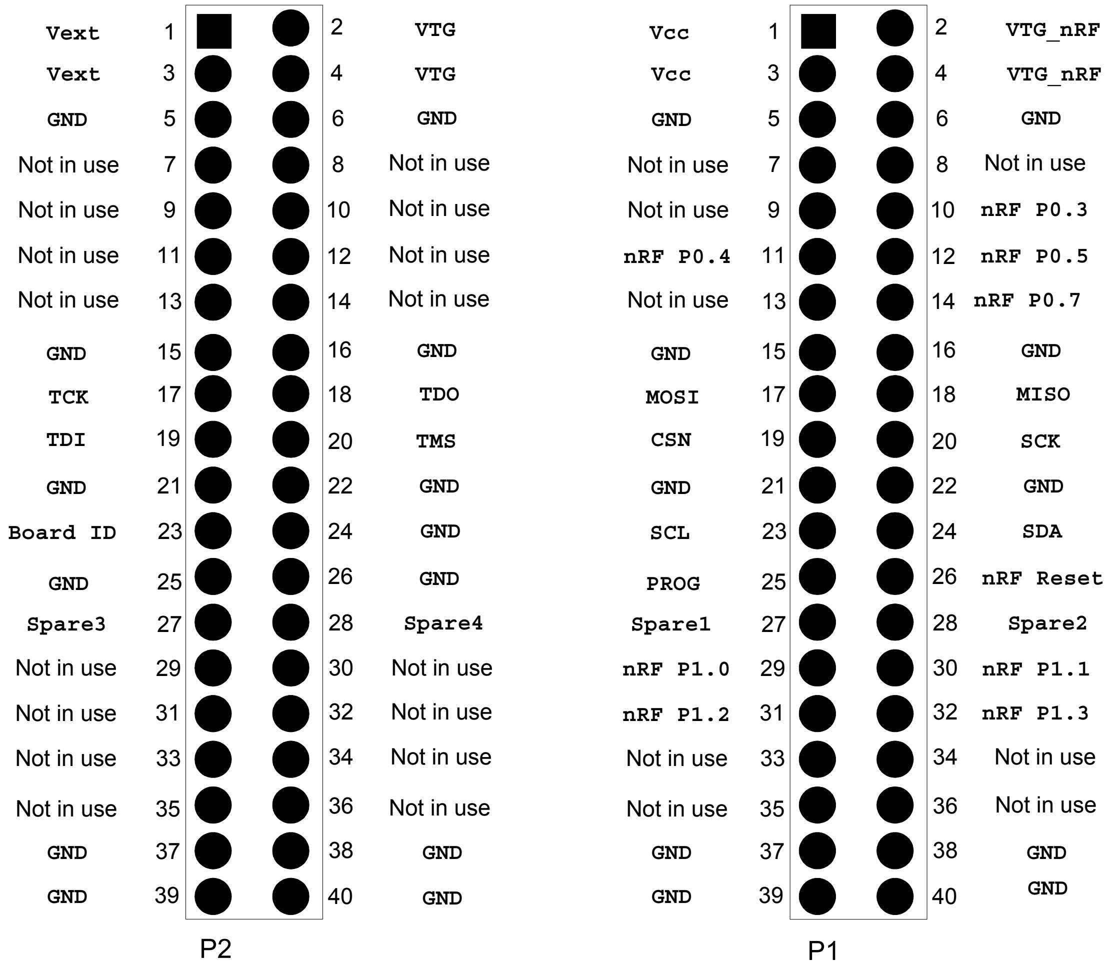


Figure 9. Carrier board module connectors - P2 and P1 as seen through mounted carrier board

Pin numbers	Connector P2		Connector P1	
	Name	Function	Name	Function
1, 3	`Vext`	Power supply output for circuitry on nRFgo Motherboard.	`VCC`	nRFgo Motherboard main power supply
2, 4	`VTG`	Connected to `Vext`.	`VTG_nRF`	Target power supply for nRF8200 on the carrier board module and nRF device on SCC module(s) (nRF2740/nRF2741)
5, 6	`GND`	Ground	`GND`	Ground
7 - 14	Not in use	NC	`P0.x`	nRF8200 device port 0
15 - 16	`GND`	Ground	`GND`	Ground
17 - 20	`TCK,TDI, TDO, TMS`	nRFprobe HW debugger JTAG interface.	`MOSI,MISO, CSN SCK`[a]	nRFgo Motherboard main MCU SPI control interface
21 - 22	`GND`	Ground	`GND`	Ground
23	`Board ID`[b]	Prototype kit ID	`SCL`[a]	2-wire clock from nRFgo Motherboard main MCU
24	`GND`	Ground	`SDA`[a]	2-wire data from nRFgo Motherboard main MCU
25-26	`GND`	Ground	`PROG`[b] `nRF_RST`[b]	nRFgo Motherboard main MCU program enable and reset control of the carrier board module
27 -28	`Spare x`	Reserved	`Spare x`	Reserved
29 - 36	Not in use	NC	`P1.x`	nRF8200 device port 1
37 - 40	`GND`	Ground	`GND`	Ground

a. nRFgo Motherboard main MCU control interfaces only. nRF8200 device SPI and 2-wire interfaces are available in the nRF8200 device ports (pins 7 - 14 or 29 - 36).
b. Used by nRFgo Motherboard only.

Table 4. Description of the carrier board P2 and P1 connector pins

7.2.2 Connector P5

This connector (② in Figure 8. on page 18) contains the ACI interface of nRF8001 and connects the SCC module (either nRF2740 or nRF2741) to the application processor on the carrier board. For detailed signal descriptions please see the nRF8001 Preliminary Product Specification.

Pin number	Connector P5
1	`GND`
2	`VCC_nRF`
3	`ACTIVE`
4	`CSN`
5	`SCK`
6	`MOSI`
7	`MISO`
8	`RDYN`
9	`REQN`
10	`RESET`

Table 5. Pinout for connector P5

7.2.3 Connector P3 - analog inputs

Direct access to the nRF8200 analog inputs is available from connector **P3** (see Figure 8.). To avoid any noise from the nRFgo Motherboard, the 0 Ohm resistors must be removed from the inputs that are directly connected to external analog circuitry.

Pin number	Connector P3
1	`GND`
2	`P0.3`
3	`P0.4`
4	`P0.5`
5	`P1.0`
6	`P0.7`
7	`P1.2`
8	`P1.1`

Table 6. Pinout for connector P3

Note: It is important that you only remove the 0 Ohm resistors on the pins that you use for high quality analog input. Pay special attention to pins that are also used for the nRF8200 programming and HW debug interfaces.

7.2.4 Jumper P4

This jumper supplies voltage to a connected SCC module (nRF2740/nRF2741). By replacing this jumper with an ampere meter it is possible to measure the current drawn by the nRF8001 device on the SCC module (nRF2740/nRF2741) in any operating mode.

Details on the nRF8001 static and dynamic current consumption can be found in the nRF8001 Product Specification.

For details on how to perform dynamic current consumption measurements, please read the white paper 'RF Performance Test Guidelines', available from www.nordicsemi.com.

7.2.5 Solder bridges SB1 and SB2

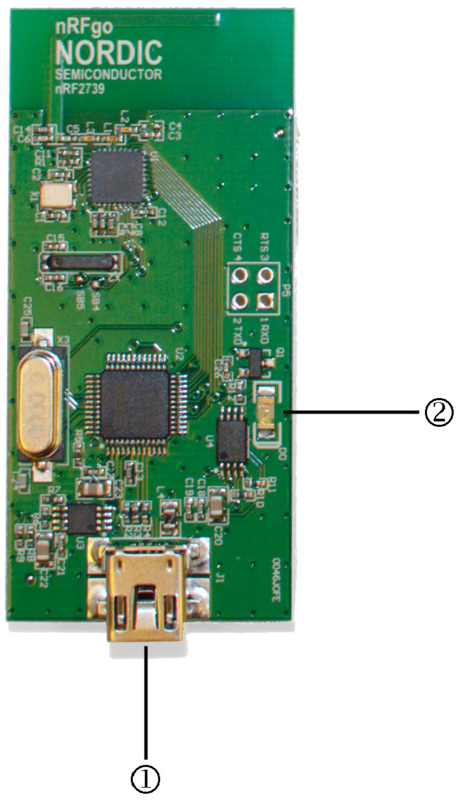
By default all available I/O pins of nRF8200 are routed to general purpose I/O port connectors (P8 and P10) on the nRFgo Motherboard. Alternatively, you can connect the 2-wire interface of nRF8200 to a separate 2-wire bus on the nRFgo Motherboard, giving you access to the nRFgo display module fitted in the extension port of the nRFgo Motherboard.

To make this connection you need to short the solder bridges **SB1** and **SB2** (also marked **SDA** and **SCL**) on the carrier board.

Note: Shorting the solder bridges does not remove the connection to the general purpose port connectors. Make sure you don't have any conflicts between this bus connection and the circuitry attached to the relevant port connector.

7.3 Master emulator (nRF2739)

The master emulator (nRF2739) is a dongle that enables you to control and monitor the traffic with nRF8001. When the master emulator is combined with the Master Control Panel it gives you a peer device for nRF8001 that you can use to test the wireless connection.



①	USB connector	②	LED

Figure 10. Master emulator (nRF2739)

7.4 Module schematics and PCB layouts

You can download nRF8001 Development Kit hardware schematics and PCB layout files from www.nordicsemi.com.

8 SDK Software architecture

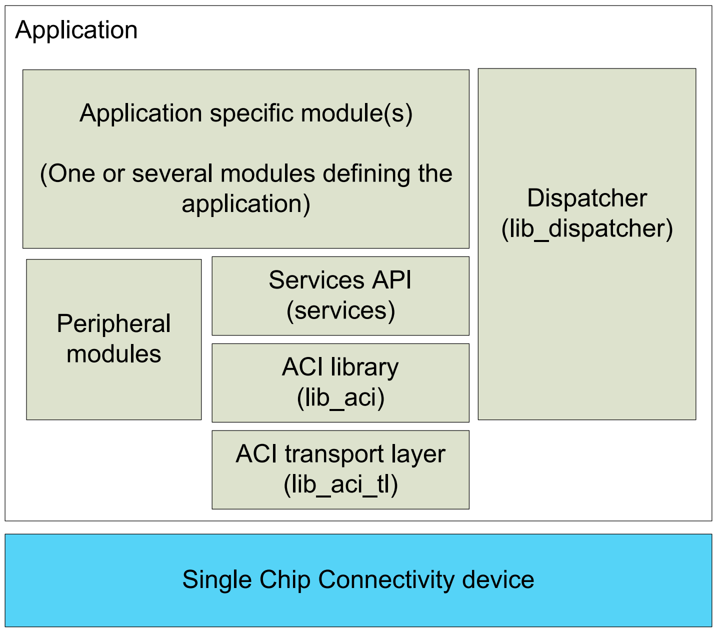

Application
Application specific module(s)
(One or several modules defining the application)
Dispatcher (lib_dispatcher)
Peripheral modules
Services API (services)
ACI library (lib_aci)
ACI transport layer (lib_aci_tl)
Single Chip Connectivity device

Figure 11. Block diagram of the SDK source code

Figure 11. illustrates the main components of the SDK source code. Understanding these main components is important for using the source code correctly and efficiently:

- **ACI library**: Provides an API for the ACI protocol which lets you send ACI commands, and decode ACI events.
- **ACI transport layer**: Handles the physical transport and hardware dependencies. The transport layer must be rewritten for the microcontroller that is used with nRF8001. Provided the interface of the transport layer remains unchanged the ACI library can be used as is.
- **Services**: A module that contains the server setup and access to the ACI channels. Each application has a services module which depends on the ACI library.
- **Dispatcher**: This is the base module for the message driven architecture used in the SDK's examples. The Dispatcher code is generated automatically from a tool integrated in nRFgo Studio.
- **Application**: This block represents the files that implement the application; the code that defines the behavior. The SDK contains examples. Your development should mainly occur within this block.
- **Peripherals**: Hardware modules written for demonstration purposes. If you are using a third party microcontroller then use source code from the MCU vendor.

8.1 ACI modules

ACI is the interface for nRF8001. The SDK has two modules that handle the ACI communication. One handles the physical transport and the other implements the protocol.

8.1.1 ACI library (lib_aci)

This library implements the ACI protocol and lets you send commands, and receive events. Please refer to the nRF8001 Preliminary Product Specification for protocol details. The ACI library gives you:

- An API for ACI commands
- Decoding of ACI events
- Flow control implementation

The ACI library does not have hardware dependencies, and can be used by any microcontroller with a C compiler. The ACI library requires the ACI Transport Layer, described in the following section.

8.1.2 ACI transport layer (hal_aci_tl)

The ACI Transport Layer module is a Hardware Abstraction Layer (HAL) for the ACI physical transport. It handles the SPI communication, and the hand-shake signals. Because of the hardware dependencies this module must be ported to your target microcontroller.

8.2 Dispatcher

All the examples are built around a message dispatcher. Modules that must communicate with each other post messages to the dispatcher. The dispatcher then forwards the message to the correct receiver.

8.2.1 Using the dispatcher

The following is the essential code that sets up and starts the dispatcher:

```
#include "lib/lib_dispatcher" // Includes dispatcher_config.h as well

void main(void)
{
  lib_dispatcher_init();
  EA = 1;

  for (;;)
  {
    lib_dispatcher_dispatch();
  }
}
```

The lib_dispatcher_dispatch() function will go through a message queue and call subscriber functions. Each message has a message handle that defines which subscriber function should be called from the message.

Subscriber functions are defined as the main part of the application and must be declared in the following form:

```
void function_handle (void);
```

Messages are added to the queue by calling lib_dispatcher_post_msg():

```
void lib_dispatcher_post_msg(
  uint8_t msg_handle,
  lib_dispatcher_priority_t msg_priority)
reentrant;
```

There is also an alternative declaration which allows arguments to be passed to the subscriber functions. Use the compile option –D POST_WITH_DATA to use this alternative. In this case there the subscriber functions must be declared as follows:

```
void function_handle (uint8_t size, uint8_t *buffer);
```

And messages are added to the queue by calling lib_dispatcher_post_msg() with the following parameters:

```
void lib_dispatcher_post_msg(
  uint8_t msg_handle,
  uint8_t *buffer,
  uint8_t size,
  lib_dispatcher_priority_t msg_priority)
reentrant;
```

Using parameters requires more code space than not using them.

The lib_dispatcher_dispatch() will handle messages as long as there are messages in the queue. When the queue is empty it will enter power down mode.

The SDK example applications are designed for low power consumption and will mostly be in low power mode. On wake-up the ISR of the wake-up source will perform the critical processing and post a message to the dispatcher using lib_dispatcher_post_msg(). When the ISR is exited the lib_dispatcher_dispatch() will handle the message and call the subscriber functions. When there are no more messages the dispatcher goes back to power down mode.

8.2.2 Configuring the Dispatcher

For setting up the dispatcher we recommend using the **Dispatcher Tool** in nRFgo Studio. The tool lets you define the relationship between handles and subscribers through a graphical user interface.

The Dispatcher Tool generates a configuration file that the dispatcher will include at compile time. See the nRFgo Studio help file for complete reference of the dispatcher library, and the Dispatcher Tool chapter to see how the configuration file is generated.

8.2.3 Creating stack space for reentrant functions

The functions used by the dispatcher are called from interrupt and must, therefore, be declared as reentrant. When using the dispatcher you must enable stack space. This is done by editing the **startup.a51** file.

When creating a new project in Keil's µVision you will be asked if you want to include startup.a51 in your project. Answer **Yes** to that question. You can also include the file later by manually copying from C:\Keil\C51\LIB to your source directory.

The code sample below shows the modified lines in startup.a51: XBPSTACK is set to 1 to enable stack space in XDATA. XBPSTACKTOP defines the size of the stack.

```
; <h> Stack Space for reentrant functions in the LARGE model.
; <q> XBPSTACK: Enable LARGE model reentrant stack
; <i> Stack space for reentrant functions in the LARGE model.
XBPSTACK EQU 1 ; set to 1 if large reentrant is used.
; <o> XBPSTACKTOP: End address of LARGE model stack <0x0-0xFFFF>
; <i> Set the top of the stack to the highest location.
XBPSTACKTOP EQU 0x03FF +1 ; default 0FFFFH+1
; </h>
```

8.2.4 Hardware dependencies

The dependencies are related to power down mode. The code for entering power down mode is found in **hal_power.c**. This file must be ported when using a third-party application processor.

9 Porting the SDK

The SDK is designed for easy porting to any application controller. This chapter gives more information about how to do the porting and the minimum necessary steps.

9.1 Hardware Abstraction Layer (HAL) Modules

To port the SDK to an application controller, you must re-implement the HAL modules you need for your application. However, we recommend that you do not modify the API itself, but only re-implement the functions that are already defined. This will avoid modifications to the application code.

The functionalities you need to port are:

- I/O configurations, in particular for the SPI lines, RDYN line, REQN line and lines connected to the LEDs.
- SPI master module: you need to configure this according to the Application Controller Interface (ACI) specification, see the nRF8001 Product Specification, chapter 7.1. for a description of this interface.

  Note: Make sure that the connections are good if you are using the SCC module (nRF2740 and nRF2741) with a third party application processor development kit.

9.1.1 hal_platform.h

This file gathers all includes and register access that are specific to nRF8200. It also contains a macro to declare variable in specific area of the memory (for data retention during deep sleep modes) and to configure interrupt handlers for nRF8200. This file has to be re-implemented for a new application controller using equivalent functionality on the new application controller.

9.1.2 hal_aci_tl/hal_aci_tl_bb

To communicate with nRF8001 you need to re-implement at least the hal_aci_tl (ACI transport Layer). This module implements the communication interface between the application processor and the nRF8001, the ACI transport layer. To verify your implementation, you can use the aci_tl_demo project which checks bi-directional data transfer and integrity of the data. See the aci_tl_demo project for more information.

The hal_aci_tl_bb file is the same behavior without using the SPI drivers (‘bit banging’ implementation).

9.1.3 hal_io.h

This file contains macros to configure and access Input/Output lines. It has to be re-implemented on a new application controller.

9.1.4 hal_power

This module contains functions to handle the different “sleep modes” of the nRF8200. You will also find in this file functions to handle one timer. This module has to be re-implemented according to the new application controller’s “sleep modes” features and timer capabilities.

9.1.5 hal_uart

This module offers functionalities to interface the serial port (UART) of the nRF8200. In the SDK, this is used only to allow sending debug information through the UART (see paragraph 6 Trace library). This module needs to be re-implemented to work on a new application controller.

9.2 Applications

Each application has parts which need to be ported or partially re-implemented since they are dependent on the application behavior and its interaction with the user (LEDs, Buttons, …). This depends on which pin is connected to which type of interface. These definitions and configurations are in the ‘system.c’ and ‘system.h’ files.

9.3 Other modules

The ‘LIB’ modules are designed to be as portable as possible, so you should be able to use them as is. The same is true for the ‘services’ modules.

10 Troubleshooting

The nRF8001 on the SCC module (nRF2740 and nRF2741) does not respond when I try to contact it. What has happened?

- Verify that the jumper **P4** on the nRF2735 is connected.

The drop-down menu in the Master Control Panel displays no serial number. What has happened?

- Verify that the Master Control Panel software and the driver for USB2SPI have been installed and that the master emulator (nRF2739) has been plugged into a USB port on your computer.

LED D0 on the master emulator (nRF2739) does not light up when I plug in the USB cable. What has happened?

- Verify that the Master Control Panel software and the driver for USB2SPI have been installed and that the master emulator (nRF2739) has been plugged into a USB port on your computer.