Introduction

This application note describes the Bluetooth low energy (BLE) embedded software (I-CUBE-nRF51DRV) implementation on the STM32L0 Series and nRF51822, and explains how to interface with the user application and create the BLE services.

The I-CUBE-nRF51DRV main features are the following:

- Compatibility with BLE profiles provided by Nordic
- Application integration ready
- Easy add-on of the low-power BLE solution on the STM32L0 Series
- Extremely low STM32L0 CPU load (HRS 1s update rate 0.127%)
- No latency requirements on the STM32L0 Series
- Small STM32L0 memory footprint

The I-CUBE-nRF51DRV software is based on STM32CubeL0 HAL drivers (see Section 2). Customer application examples on STM32L0 using the Nordic BLE services are provided.

The reference hardware platform is the STM32-nRF51822 based on the STM32Nucleo/64 and the Wavetek Bluetooth LE shield with the Nordic BLE module nRF51822.

Note: I-CUBE-nRF51DRV can be ported to other STM32 Series.
Contents

1 References .................................................. 7

2 STM32Cube overview ..................................... 8

3 Embedded software definition ......................... 9

4 Getting started ............................................. 11
   4.1 BLE system description .............................. 11
   4.2 Features ............................................. 11
   4.3 Hardware/software quick setup .................... 12

5 Reference platform ....................................... 13
   5.1 Interface description ................................ 13
   5.2 UART 4-wire interface .............................. 14
   5.3 Reset system ......................................... 14
      5.3.1 Power on reset ................................. 15
      5.3.2 System reset ..................................... 15
   5.4 Device programming ................................ 16

6 Embedded software description ....................... 18
   6.1 UART Interface ....................................... 18
      6.1.1 BLE SD FW module ............................. 19
      6.1.2 Low power manager ............................ 20
      6.1.3 Interrupt ....................................... 20
      6.1.4 Context manager ............................... 20
      6.1.5 Configuration ................................. 21
      6.1.6 STM32 resource requirements when BLE feature is used ................................. 21
      6.1.7 Integration ..................................... 22
      6.1.8 Portability ..................................... 22
   6.2 Timer interface ....................................... 23
      6.2.1 BLE SD FW module ............................. 25
      6.2.2 Initialization .................................. 25
      6.2.3 User module ................................... 26
      6.2.4 Interrupt ...................................... 26
6.2.5 Context manager ........................................... 26
6.2.6 Configuration .............................................. 27
6.2.7 Integration .................................................. 27
6.2.8 Portability .................................................... 28

6.3 NVM interface ................................................ 28
6.3.1 BLE SD FW module ........................................ 31
6.3.2 Initialization ............................................... 31
6.3.3 Operations ................................................ 31
6.3.4 Low power manager ....................................... 32
6.3.5 Interrupt .................................................. 32
6.3.6 Configuration ............................................. 32
6.3.7 Integration ................................................ 32
6.3.8 Portability ................................................ 32

6.4 Low power manager interface ................................. 33
6.4.1 API ......................................................... 35
6.4.2 Configuration ............................................. 35
6.4.3 Integration ................................................ 35
6.4.4 Portability ................................................ 35

6.5 BLE SD FW module firmware integration in application 36
6.5.1 BLE startup ............................................... 36
6.5.2 Communication with the BLE SD FW module ......... 36

6.6 Portability ..................................................... 37

7 BLE application configuration ................................. 38
7.1 Architecture .................................................. 39
7.2 Direct test mode application ................................. 39
7.3 Heart rate monitoring / health thermometer application 40
7.3.1 Advertising application .................................. 40
7.3.2 Heart rate application .................................... 40
7.3.3 Health thermometer application ....................... 42

8 Performances results ........................................... 43
8.1 Static power consumption ................................... 43
8.2 SD command exchange power consumption profile ....... 43
8.3 Connection average power consumption .................... 44
8.4 Data throughput ............................................. 46
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>System limitations</td>
<td>47</td>
</tr>
<tr>
<td>Revision history</td>
<td>48</td>
</tr>
</tbody>
</table>
## List of tables

| Table 1. | STM32 memory footprint (HRS application) | 9 |
| Table 2. | Interface between the STM32 and nRF51822 | 13 |
| Table 3. | Interface between the STM32 and Wavetek board | 13 |
| Table 4. | configuration of the BLE application | 40 |
| Table 5. | HRS profile average power consumption | 46 |
| Table 6. | Maximum data throughput L2CAP average power consumption | 46 |
| Table 7. | Document revision history | 48 |
# List of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STM32-nRF51822 system architecture</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>STM32-nRF51822 HRM on Android</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>HW platform stacking (Arduino signals)</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>nRF51822 - STM32 communication interface UART 4-wire</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>STM32 programming with ST-LINK utility</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>nRF51822 device programming</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>Embedded software overview API</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>UART interface overview</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>Timer interfaces overview</td>
<td>24</td>
</tr>
<tr>
<td>10</td>
<td>Timer design overview</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>NVM interface overview</td>
<td>29</td>
</tr>
<tr>
<td>12</td>
<td>NVM design overview</td>
<td>30</td>
</tr>
<tr>
<td>13</td>
<td>Low power manager interface overview</td>
<td>33</td>
</tr>
<tr>
<td>14</td>
<td>Low power manager design overview</td>
<td>34</td>
</tr>
<tr>
<td>15</td>
<td>Scheduler implementation</td>
<td>37</td>
</tr>
<tr>
<td>16</td>
<td>Application overview</td>
<td>38</td>
</tr>
<tr>
<td>17</td>
<td>Application DTM access</td>
<td>39</td>
</tr>
<tr>
<td>18</td>
<td>RxSD command exchange power consumption</td>
<td>43</td>
</tr>
<tr>
<td>19</td>
<td>TxSD command exchange power consumption</td>
<td>44</td>
</tr>
<tr>
<td>20</td>
<td>STM32 heart rate monitoring mobile notification</td>
<td>45</td>
</tr>
</tbody>
</table>
1 References

[1] nRF51822 soft device specification, Nordic:
http://www.nordicsemi.com/

[2] nRF51822 reference manual, Nordic:
http://www.nordicsemi.com/

[3] nRF51822 product specification, Nordic:
http://www.nordicsemi.com/

[4] Nordic serializer project APIs
http://www.nordicsemi.com/

www.st.com

[6] STM32 datasheet – production data, ST:
www.st.com

www.st.com

http://www.wavetek.com.hk
STM32Cube overview

2 STM32Cube overview

STMCube™ initiative was originated by STMicroelectronics to ease developers’ life by reducing the development efforts, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL0 for STM32L0 Series)
  - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, TCP/IP and graphics
  - All embedded software utilities coming with a full set of examples.
3 Embedded software definition

The following software components are used:

- I-CUBE-nRF51DRV: STM BLE application solution including the BLE SD FW module and based on STM32CubeL0 (see Section 2: STM32Cube overview).
- nRF51822 S110 SoftDevice binary (see reference to Nordic documentation)
- nRF51822 connectivity serialized solution from Nordic (see reference to Nordic documentation)

The system solution (STM32-nRF51822) provides a 2 chip BLE solution, where the STM32 operates as a host device and nRF51822 is the connectivity part. The whole BLE stack is located on the nRF51822 device, which interfaces with the STM32 via a UART.

The BLE SD FW module, UART, timer, NVM and the customer application are built on the STM32 platform. The customer application may be either a dedicated simple BLE application or a wider application on which BLE connectivity is added:

The following modules are required to add a BLE connectivity to the STM32:

- **Profile service management**
  The BLE services are running on the nRF51822 system and this module is providing an implementation proposal on the STM32 side to manage each supported BLE services. Although the implementation proposal is ready to be used, the user may implement its own services management.

- **BLE SD FW module**
  This module is not expected to be modified by the user. It provides an interface to the application to send and receive messages with the nRF51822 system. It requires from the STM32 platform a timer, a UART and a NVM.

- **Timer server**
  The Timer server is an implementation proposal of a module that can provide multiple virtual timers all sharing the RTC Wakeup. The BLE module gets a timer from that module to operate but the user may implement its own timer mechanism to be connected to the BLE module.

- **NVM driver**
  The Non Volatile Memory driver is a very specific implementation proposal, using the EEPROM, suitable to be used by the BLE SD FW module. It provides an efficient mechanism to minimize the performance impact of the high latency required to write in the EEPROM. The user may implement its own NVM driver to be connected to the BLE SD FW module.

<table>
<thead>
<tr>
<th></th>
<th>ROM (Kbytes)</th>
<th>RAM (Kbytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLE SD FW module</td>
<td>25.7</td>
<td>2.6</td>
</tr>
<tr>
<td>NVM driver, timer server, UART driver, low power manager</td>
<td>3.1</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 1. STM32 memory footprint (HRS application)
• **UART driver:**
  The UART driver implements a proprietary low power protocol communication over UART between the STM32 and the nRF51822 system. Although the user may implement its own driver, it is strongly recommended to use the current implementation that has been pre-integrated so that it is ready to be used.

• **Low power manager:**
  The UART and NVM drivers are making dynamically requests on the availability of the HS clock. On top of this, the application may have some modules that could require different low-power mode over the time. The low power manager provides an implementation proposal of an easy interface to receive all requests regarding power management and compute the lowest power mode the system may enter. The user may implement its own low power manager.

A detailed description on the nRF51822 system is available on the Nordic's website.

The STM32-nRF51822 BLE system architecture is given in the figure below.

*Figure 1. STM32-nRF51822 system architecture*
4 Getting started

4.1 BLE system description

By default the UART communication between the target MCU and ST-LINK MCU is enabled in order to support virtual com port for SB13 and SB14 ON, SB62 and SB63 OFF.

As the communication between the target MCU and the shield or the extension board is required, SB62 and SB63 should be ON, SB13 and SB14 should be OFF.

The solution is composed of the following embedded software items:

- STMicroelectronics BLE FW package: three pre-compiled applications
  - stm_dtm_app.hex
  - stm_hrs_app.hex
  - stm_hts_app.hex

- nRF51822 SoftDevice binary (www.nordicsemi.com)

- s110_nrf51822_6.0.0_softdevice.hex

- nRF51822 serialized solution from Nordic (www.nordicsemi.com)
  - ble_app_connectivity.hex

As the reference platform to run the previous software the following hardware items are used:

- 1 x STM32Nucleo/64 board revC with STM32L053R8
  see Nucleo rev C hardware user manual [7] for more details

- 1 x Wavetek Arduino shield with Nordic BLE module nRF51822
  see Wavetek shield user manual [8] for more details

Android phone running STM32 BLE toolbox can be used as demonstration tool.
- STM32_BLE_Toolbox.apk delivered with the solution

4.2 Features

The followings BLE services are supported with the system solution:

- Heart rate service with battery monitoring
- Health thermometer service
- Direct test mode entry
- Advertiser only
4.3 **Hardware/software quick setup**

- Plug the Wavetek Bluetooth LE shield on STM32Nucleo/64 board via the Arduino connectors.
- Connect the STM32 Nucleo/64 board to the PC with an USB cable (USB cable not provided)
- The PWD LED2 shall light up
- As pre-condition, the nRF51822 S110 soft device binary must be programmed. Refer to Nordic quick starter guide documentation. (This part can’t be programmed with the ST-LINK)
- The nRF51822 needs to be programmed with the “connectivity serialized” application.
- To program the “connectivity serialized” via ST-LINK
  – ST-LINK CN2 jumpers are OFF
  – Wavetek Bluetooth LE shield RST J5&J6 jumpers OFF
  – Connect SWDCLK interface pin
    STM32Nucleo/64 ST-LINK SWD CN4 pin 2 to Wavetek Bluetooth LE shield P1/JLINK SWDCLK pin4
  – Connect SWDIO interface pin
    STM32Nucleo/64 ST-LINK SWD CN4 pin4 to Wavetek Bluetooth LE shield J6.1
  – Load “connectivity” binary on nRF51822 with Keil™ or other tool
- Select BLE feature for STM_ble_app” application
  – Open “STM_ble_app” project with MDK-ARM microcontroller development kit by Keil™ or other tool
  – Open app_conf.h to configure the user project and compile.
- Program the STM32 with “STM_ble_app” application
  – Remove SWD interface connections between STM32Nucleo/64 ST-LINK and Wavetek Bluetooth LE shield
  – ST-LINK CN2 jumpers are On
  – Load “STM_ble_app” binary on STM32 with Keil™ or other tool (STM32 ST-LINK Utility)
- Reset the system with the button reset B2 on Nucleo board.
- BLE state is advertising
- Use the Android or iPhone application to verify the advertiser packets

*Figure 2. STM32-nRF51822 HRM on Android*
5 Reference platform

5.1 Interface description

The STM32-nRF51822 system HW platform consists of a STM32Nucleo/64 board from STMicroelectronics with a plugged nRF51822 Wavetek Arduino shield with the BLE module nRF51822 from Nordic.

In addition there is possibility to add an application module on top of the system via an extended Arduino shield.

![Figure 3. HW platform stacking (Arduino signals)](image)

The interface between the STM32 and nRF51822 consists of the following Arduino signals:

**Table 2. Interface between the STM32 and nRF51822**

<table>
<thead>
<tr>
<th>nRF51822 Signal</th>
<th>IO</th>
<th>STM32 Signal</th>
<th>IO</th>
<th>Arduino</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>RXD</td>
<td>I</td>
<td>USART2_TX, PA2</td>
<td>O</td>
<td>D1</td>
<td>UART data STM32 to nRF51822</td>
</tr>
<tr>
<td>TXD</td>
<td>O</td>
<td>USART2_RX, PA3</td>
<td>I</td>
<td>D0</td>
<td>UART data nRF51822 to STM32</td>
</tr>
<tr>
<td>CTS</td>
<td>I</td>
<td>USART2_RTS, PA1</td>
<td>O</td>
<td>A1</td>
<td>UART STM32 RX Flow Control</td>
</tr>
<tr>
<td>RTS</td>
<td>O</td>
<td>USART2_CTS, PA0</td>
<td>I</td>
<td>A0</td>
<td>UART nRF51822 RX Flow Control</td>
</tr>
<tr>
<td>nRESET</td>
<td>I</td>
<td>PC7, GPIO</td>
<td>O</td>
<td>D9</td>
<td>BLE nReset nRF51822 device</td>
</tr>
<tr>
<td>VCC</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3V3</td>
<td>Power provided by STM32Nucleo/64</td>
</tr>
<tr>
<td>GND</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>GND</td>
<td>Ground</td>
</tr>
</tbody>
</table>

The additional application interface between STM32 and Wavetek board consists of the following Arduino signals:

**Table 3. Interface between the STM32 and Wavetek board**

<table>
<thead>
<tr>
<th>Application Signal</th>
<th>IO</th>
<th>Arduino</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA10, GPIO</td>
<td>IO</td>
<td>D2</td>
<td>GPIO (LED3 - D3) - Wavetek SF3</td>
</tr>
<tr>
<td>PB3, GPIO</td>
<td>IO</td>
<td>D3</td>
<td>GPIO (LED2 - D2) - Wavetek SF4</td>
</tr>
<tr>
<td>PB5, GPIO</td>
<td>IO</td>
<td>D4</td>
<td>GPIO (LED1 - D1) - Wavetek SF5</td>
</tr>
<tr>
<td>PB4, GPIO</td>
<td>IO</td>
<td>D5</td>
<td>GPIO (Button3 - SW3) - Wavetek SF6</td>
</tr>
</tbody>
</table>
By default, the Wavetek Bluetooth LE shield is not configured to use the buttons and LEDs on the board.

The following solder fields have to be close:
- SF8: SW1-Button1
- SF7: SW2-Button2
- SF6: SW3-Button3
- SF5: LED1
- SF4: LED2
- SF3: LED3

### 5.2 UART 4-wire interface

The UART 4-wire interface is used to communicate between the 2 devices. The HW CTS/RTS flow control signals are also used for the low power control.

### 5.3 Reset system

The reset system can be divided into the following functions:
- Power on reset
- System reset

nRF51822 shield (Wavetek) jumper reset J1 shall be OFF.

#### Figure 4. nRF51822 - STM32 communication interface UART 4-wire
5.3.1 Power on reset

Both the STM32 and the nRF51822 have their own integrated power on reset detector these will guarantee that both the STM32 and the nRF51822 devices are reset correctly on application power on.

5.3.2 System reset

A system reset button is connected to the STM32. A system resets the STM32 which subsequently will generate a reset pulse on the GPIO to reset the nRF51822. This will guarantee that both the STM32 and the nRF51822 devices are reset on a system reset.

To allow the STM32 to reset nRF51822 by GPIO, Wavetek Bluetooth LE shield jumpers reset J5 and J6 shall be OFF if D9 and nReset (J6.1) are connected by a cable.

Another option is to reset the system by STM32Nucleo/64 system reset, the jumper J5 shall be OFF and the jumper J6 shall be ON if the cable is not connected.
5.4 Device programming

The system solution provides a FW download possibility through ST-LINK for both the STM32 and the nRF51822 serialized application. (The nRF51822 “Soft Device” cannot be loaded with the ST-LINK.)

See the STM32Nucleo/64 HW user manual on how to program the embedded software.

To program the STM32 solution with the ST-LINK/V2, the following procedure has to be respected:

- Remove SWD cable connection
- STM32Nucleo/64 CN2 jumpers ON
- See the example done with the STM32 ST-LINK utility to program the STM32 on STM32Nucleo/64 board
  - Connect ST-LINK utility tool with option: connect under reset
  - Open “STM_ble_app.hex”
  - Program

![Figure 5. STM32 programming with ST-LINK utility](image)
To program the nRF51822 solution with the ST-LINK, the following procedure has to be respected:

- STM32Nucleo/64 CN2 jumpers OFF
- Waveletk Bluetooth LE shield RST J5 and J6 Jumpers OFF. nReset between STM32 and nRF51822 not connected
- Connect **JTCK**
  - STM32Nucleo/64 ST-link SWD CN4 pin2 to Waveletk Bluetooth LE shield JLINK SWDCLK pin4
- Connect **SWDIO**
  - STM32Nucleo/64 ST-link SWD CN4 pin4 to Waveletk Bluetooth LE shield J6 pin1 or JLINK SWDIO pin2
- See the example done with Keil to program the nRF51822 on Waveletk Bluetooth LE shield
  - Use ST-Link debugger
  - Select debug port SW
  - Select programming algorithm **nRF51xxx**

![nRF51822 device programming](image-url)
6 Embedded software description

This section provides a description of the interfaces of the different blocks shown in Figure 7:

- The interfaces between the BLE SD FW module and the STM32 peripherals (UART, Timer, NVM)
- The interfaces between the BLE SD FW module and the application
- The interface with the low power manager
- The project structure folder. It describes the split between the BLE SD FW module as such that it does not require any modification by the application and the files that need to be adapted by the application.

Figure 7. Embedded software overview API

6.1 UART Interface

The BLE module sends and receives data over the UART interface. The UART driver provides a simple interface to send and receive messages and implements a proprietary protocol to notify the application when the UART may be disabled to enter low-power mode.
6.1.1 BLE SD FW module

The following interfaces shall be mapped to the BLE module. They are not used by the application:

- LPPUART_uart_open()
- LPPUART_send_data()
- LPPUART_receive_data()
As the mapping is not depending on the application but only on the BLE SD FW module interface, a default implementation is done in the file `ble_uart.c`.

There is no need for the application to make any modification.

### 6.1.2 Low power manager

- **LPPUART_HSclkRequest()**
  
  This API is notifying the application when the HS clock may be stopped and when it shall be kept enabled. In addition, the application may switch ON the clock of the DMA used with the UART when the HS clock is requested and switch OFF the clock of the DMA when the HS clocked is not required. This would save additional power when entering SLEEP mode. The default implementation in the examples provided keeps the DMA clock enabled all the time as this has marginal impact on system power consumption.

### 6.1.3 Interrupt

- **LPPUART_wcts_handler()**
  
  This interrupt handler shall be called by the application in the EXTI interrupt handler related to the IO pin where CTS has been defined in the file `stm32yyxx_lppuart_configuration.h`. This handler takes care of clearing all status flag required in the UART peripheral.

- **LPPUART_uart_handler()**
  
  This interrupt handler shall be called by the application in the UART interrupt handler used for the communication with the nRF device and defined in the file `stm32yyxx_lppuart_configuration.h`. This handler takes care of clearing all status flag required in the UART peripheral.

- **LPPUART_dma_rx_handler()**
  
  This interrupt handler shall be called by the application in the DMA interrupt handler used with the UART peripheral (as defined in the reference manual [5]) connected to the nRF device. This handler takes care of clearing all the status flags required in the DMA peripheral.

### 6.1.4 Context manager

- **LPPUART_Msg_Handler()**
  
  This API is notifying the BLE SD FW module that an event occurred on the UART interface. By default, all the messages are handled in the UART interrupt context when sent from the STM32 and in the DMA interrupt context when received by the STM32. The application may implement an Operating System to change the context priority where the BLE messages may be handled. A signal to the BLE task may be sent from this API to notify a full message has been received or transmitted over UART.
6.1.5 Configuration

The UART driver is fully configured in the file `stm32yyxx_lppuart_configuration.h`.

- The application may select the UART peripheral to connect to the nRF device and the UART IO mapping on the pins.
- The application may change the UART baudrate. However, it is recommended to keep the default setting 1Mbits for power consumption optimization. When the UART baudrate is changed, this shall be done on both the STM32 and nRF devices.
- The UART driver uses DMA, UART and EXTI interrupts. The priority set in the NVIC may be changed to allow the application to select the best priority with regard to already existing interrupts and timing requirement. The UART driver is able to run fine whatever priority is set. It is recommended to keep the same order as the default configuration which provides the best latency over UART. Default order is DMA (the highest), UART and EXTI (the lowest).
- The UART driver uses critical sections to avoid context corruption. This is achieved with the macro `LPPUART_ENTER_CRITICAL_SECTION` and `LPPUART_EXIT_CRITICAL_SECTION`. The default implementation masks all the STM32 interrupts with the PRIMASK bit of the CortexM CPU. It is possible to use the BASEPRI register of the CortexM CPU to keep allowed some interrupts with high priority. In that case, the user shall make sure when BLE is enabled and the UART driver critical section is entered that:
  - The GPIO and EXTI peripherals are not accessed.
  - The UART interrupt handler is masked
  - The application does not call API from the BLE SD FW module (to send a command, etc...)

6.1.6 STM32 resource requirements when BLE feature is used

The UART driver requires some hardware resources that shall be made available before enabling the BLE feature. These hardware resources shall be kept enabled as long as the BLE feature is enabled. The application should switch OFF all these hardware resources when BLE is disabled to save power.

The hardware resources concerned are:

- The DMA clock used with the UART connected to the nRF device shall be available in RUN and SLEEP mode.
- The SRAM clock shall be available in SLEEP Mode
- The SYSCFG peripheral clock shall be available in RUN mode
- The PWR peripheral clock shall be available is RUN mode
- The clock of the UART connected to the nRF device shall be available in RUN and SLEEP mode
- The GPIO clock on which the UART is mapped shall be available in RUN mode
6.1.7 Integration

To include the UART low power driver in an existing application using BLE SD FW, the application shall:

1. Implement the file `stm32xx_lppuart_conf.h`  
   It shall include `stm32xx_lppuart_interfaces.h` and all header files providing the required API from the dedicated HAL (Refer to the STM32L0xx implementation example provided)

2. Set the configuration required (USART, IOs, baudrate, wakeup pin and interrupt priority) in the file `stm32yyxx_lppuart_configuration.h`

3. Include the file `stm32xx_lppuart.c` in the project. (Depending on the platform and the STM32Cube version used, some other files may be required as described in the next chapter)

4. Map the UART BLE interface on the HAL UART driver (This step is already done in the wrapper implemented in the file `ble_uart.c`. The application may just include that file in the project)

5. Implement the API `LPPUART_HSclkRequest()` to handle the HS clock requirement and DMA clock according to the low power UART driver activity.

6. Update the EXTI interrupt handler to call the API `LPPUART_wcts_handler()` when the CTS line has triggered the interrupt. Te API `LPPUART_wcts_handler()` shall not be called if the CTS line is not the source that triggers EXTI interrupt handler when this one is shared with other IOs interrupt capability set by the application.

7. Update the UART interrupt handler to call the API `LPPUART_uart_handler()`

8. Update the DMA interrupt handler to call the API `LPPUART_dma_rx_handler()`

9. When an Operating System is used, the `LPPUART_Msg_Handler()` may be used to send a signal with the associated parameter to handle the message in a different context than the UART interrupt. This API is executed within the UART interrupt handler context.

6.1.8 Portability

The current delivery supports the STM32L0 family but the full implementation is STM32Cube compliant which means it can easily be ported on any other STM32 families.

When the STM32 family required is not supported in the delivery, the user shall implement the required API as this has been done for the STM32L0 family.

All the required API specific to a family to be implemented are located in the same folder (STM32L0xx for the STM32L0 family). The user should implement the configuration files (in the config folder), the MSP files (in the msp folder) and the HAL specific API (in the `STM32yyxx_HAL_LPPUART_Drivers` folder - `STM32L0xx_HAL_LPPUART_Drivers` for the STM32L0xx family).

- **configuration files**

  The `stm32yyxx_lppuart_driver_definition.h` contains the mapping between the macro definition used by the software and the platform definition described in the reference manual.

  The `stm32yyxx_lppuart_configuration.h` contains the user configuration of the UART driver. It is based on the `stm32yyxx_lppuart_driver_definition.h` file.
• **MSP files**
  The `stm32yyxx_lppuart_msp.c` contains the DMA initialization that could be different between STM32 families.
  This file shall be included in the project during the integration

• **HAL specific API**
  The `STM32yyxx_HAL_LPPUART_Drivers` folder contains all HAL API that are not part of the STM32Cube delivery. The list of API required to be implemented may change according to the version of STM32Cube. Any files that required to be implemented with regard to the STM32Cube shall be included in the project during the integration.

### 6.2 Timer interface

The BLE SD FW module uses a timer to timeout the connection parameter negotiations with the remote device. In order to save power consumption, this timer should be running on either the LSI or LSE LPO clock.

The timer server provides the following features:

- Up to 255 virtual timers (or less due to RAM limitation)
- Single shot and Repeated mode
- Stop a virtual timer and restart it with a different timeout value
- Delete a timer
- Timeout from 1 to 65535 ticks

The timer server provides multiple virtual timers sharing the wakeup TIMER. Each virtual timer may be defined to be a single shot timer or a repeated timer. When a repeated timer elapses, the user is notified and the virtual timer is restarted automatically with the same timeout. When a single shot timer elapses, the user is notified and the virtual timer is set to the pending state. The user may stop a virtual timer and restart it with a different timeout value. When a virtual timer is not needed anymore, the user should delete it to free the slot in the timer server.
Figure 9. Timer interfaces overview

- BLE SD FW Module
  - ble_timer_create
    (p_timer_id, mode, timeout_handler)
  - ble_timer_start
    (timer_id, timeout_ms, p_context)
  - ble_timer_stop
    (timer_id)

- Application

- Context manager

- Interrupt

- TIMER Interfaces
  - TIMER_Create
    (oTimerModuleID, pTimerId, oTimerMode, pTimerCallBack)
  - TIMER_Start
    (ubTimerID, ubTimeoutTriggers)
  - TIMER_Delete
    (ubTimerID)
  - TIMER_Init
    ()
  - TIMER_Stop
    (ubTimerID)
  - TIMER_Notification
    (oTimerModuleID, pTimerCallBack)
  - TIMER_RTC_WakeUp_Handler
    ()

- TIMER Server

MSv.35955V1
6.2.1 BLE SD FW module

The following interfaces from the BLE SD FW module shall be mapped to the timer server:

- ble_timer_create()
- ble_timer_start()
- ble_timer_stop()

A default mapping to the timer server is done in the file ble_timer.c.

There is no need for the application to make any modification.

6.2.2 Initialization

- TIMER_Init()

This API shall be called by the application before any timer is requested to the timer server. It configures the RTC module to be connected to the LSI input clock.
6.2.3 User module

- **TIMER_Create()**
  The user shall call this API to create a timer. Once created, the timer is reserved to the module until it has been deleted. When creating a timer, the user shall specify the mode (single shot or repeated), the callback to be notified when the timer expires and a module ID to identify in the timer interrupt handler which module is concerned.
  In return, the User gets a timer ID to handle it.

- **TIMER_Start()**
  This API shall be used to start a timer. The timeout value is specified and may be different each time.
  When the timer is in the single shot mode, it will move to the pending state when it expires. The user may restart it at any time with a different timeout value.
  When the timer is in the repeated mode, it always stay in the running state. When the timer expires, it will be restarted with the same timeout value.
  This API shall not be called on a running timer.

- **TIMER_Stop()**
  This API may be used to stop a running timer. A timer which is stopped is move to the pending state.
  A pending timer may be restarted at any time with a different timeout value but the mode cannot be changed.

- **TIMER_Delete()**
  This API should be used when a timer is not needed anymore by the user. A deleted timer is removed from the timer list managed by the timer server. It cannot be restarted again. The user has to go with the creation of a new timer if required and may get a different timer id.

6.2.4 Interrupt

- **TIMER_RTC_Wakeup_Handler()**
  This interrupt handler shall be called by the application in the RTC interrupt handler.
  This handler takes care of clearing all status flag required in the RTC and EXTI peripherals.

6.2.5 Context manager

- **TIMER_Notification()**
  This API notifies the application that a timer expires. This API is running in the **RTC Wakeup interrupt context**. The application may implement an Operating System to change the context priority where the timer callback may be handled. This API provides the module ID to identify which module is concerned and to allow sending the information to the correct task.
6.2.6 Configuration

Some parameters of the timer server may be configured in the file `stm32xx_timerserver.h`.

- **MAX_NBR_CONCURRENT_TIMER**: the user may define the maximum number of virtual timer supported. It shall not exceed 255. Default configuration is 8.
- **NVIC_UART_RTC_WAKEUP_IT_PRIORITY**: the user may define the priority in the NVIC of the RTC_WKUP interrupt handler that is used to manage the wakeup timer. Default configuration is 3.
- **eHAL_TIMER_ModuleID_t**: the user shall define the list of Module Id that the timer server shall support.
- The TimerServer driver uses critical sections to avoid context corruption. This is achieved with the macro `TIMER_ENTER_CRITICAL_SECTION` and `TIMER_EXIT_CRITICAL_SECTION`. The default implementation masks all STM32 interrupts with the PRIMASK bit of the CortexM CPU. It is possible to use the BASEPRI register of the CortexM CPU to keep allowed some interrupts with high priority. In that case, the user shall make sure when the TIMER critical section is entered that no TimerServer API are called.

Before running the timer server, the application shall:

- Configure the input clock of the RTC and the clock prescaler. This is defining the timer tick. In the current implementation, the timer tick is equal to the LSI input clock divided by 16 (432 us). It may provide a timeout value up to 28 s.
- Keep the access always available in the RTC register while the timer server is running
- Keep disable the write protection in the RTC block while the timer server is running

6.2.7 Integration

To include the timer server in an existing application, the application shall:

1. Implement the file `stm32xx_timerserver_conf.h`
   It shall include `stm32xx_timerserver.h` and all header files providing the required API from the dedicated HAL (Refer to the STM32L0xx implementation example provided)
2. Include the file `stm32xx_timerserver.c` in the project
3. Map the timer BLE interface on the timer server (This step is already done in the wrapper implemented in the file `ble_timer.c`. The application may just include that file in the project)
4. When an Operating System is used, the `TIMER_Notification()` may be used to send a signal with the associated parameter to handle the Callback in a different context than the RTC wakeup interrupt. This API is executed within the RTC wakeup interrupt handler context. This API shall be implemented by the application.
5. Update the RTC interrupt handler to call the API `TIMER_RTC_Wakeup_Handler()`

Note: The timer created by the BLE SD FW module is deleted on a disconnect event. The application shall first disconnect the link before switching OFF the nRF device to make sure the reserved timer for BLE is deleted.
6.2.8 Portability

The current delivery supports the STM32L0 family but the full implementation is STM32Cube compliant which means it can easily be ported on any other STM32 families.

When the STM32 family required is not supported in the delivery, the user shall implement the required API as this has been done for the STM32L0 family.

All the required API specific to a family to be implemented are located in the same folder (STM32L0xx for the STM32L0 family). The user should implement the HAL specific API (in the STM32yyxx_HAL_TimerServer_Drivers folder - STM32L0xx_HAL_TimerServer_Drivers for the STM32L0 family).

- HAL specific API

The STM32yyxx_HAL_TimerServer_Drivers folder contains all HAL API that are not part of the STM32Cube delivery. The list of API required to be implemented may change according to the version of STM32Cube. Any files that required to be implemented with regard to the STM32Cube shall be included in the project during the integration.

6.3 NVM interface

The BLE SD FW module requires storing some parameters in a non-volatile memory. While the embedded software is running, the BLE SD FW module keeps a copy in the RAM memory. This copy in RAM is uploaded at BLE initialization with the data from the NVM. When a parameter is updated, a store to NVM is requested. There is no read access to the NVM from the BLE SD FW module except at initialization. The BLE SD FW module always reads the parameters from the RAM copy.

The NVM driver has been designed to especially fit the BLE behavior. Due to the high latency on writing data in the NVM, the driver stores in a queue the operation (write or erase) requested and releases the CPU to keep on running code. The NVM driver writes the data in the NVM and waits for the completion interrupt. On each interrupt, the NVM driver writes new data until there is no more data to be written.

As the NVM drivers stores only the operation to execute and the reference of the data, these data shall be kept available in the RAM memory.

There is no feedback to the application of the completion of an operation.

During the programming of the NVM, the HS clock shall be kept available. The NVM driver notifies the application at the start and the end of the programming the lowest power mode supported.
Figure 11. NVM interface overview

BLE SD FW Module

- pstorage_register
  - (*p_module_param, *p_block_id)
- pstorage_block_identifier_get
  - (*p_base_id, block_num, *p_block_id)
- pstorage_store
  - (*p_dest, *p_src, size, offset)
- pstorage_load
  - (*p_dest, *p_src, size, offset)
- pstorage_clear
  - (*p_dest, size)
- pstorage_init
  - ()

Low Power Manager

- LPM_Mode_Request
  - (eid, eMode)

Wrapper

NVM Interfaces

- NVM_Register
  - (*NVMAddressBlock, BlockSize)
- NVM_Read
  - (*UserAddress, *NVMAddress, Size)
- NVM_Operation
  - (eOperation, *UserAddress, *NVMAddress, Size)
- NVM_Init
  - ()
- NVM_HSclkRequest
  - (eHSclkMode)
- NVM_Flash_Interrupt_handler
  - ()

NVM Driver

Application

MSv35951V1
Figure 12. NVM design overview

NVM Driver
- The DATA shall always been kept available in SRAM on the requester side
- The queue is storing the reference of the DATA to copy in NVM
- There is no local copy of the DATA to be written in NVM in the driver

Register an operation (write/erase) to execute

Queue of operations
- Operation
- Source address
- Dest address
- Size

- Asynchronous data read from NVM
- Does not check if the Id to be read is in the queue

- As soon as an operation is queued, the driver is requesting the HSclock to be kept running
- When the last data has been written in the NVM, the driver sends a notification that the HSclock is not required anymore

NVM_Init()

NVM_Operation (eOperation, *UserAddress, *NVMAddress, Size)

NVM_Register (*NVMAddressBlock, BlockSize)

NVM_Read (*UserAddress, *NVMAddress, Size)

NVM_HScbRequest (isHScbMode)

NVM_Flash_Interrupt_handler ()

Context

operation

Free
Free

Operation1
Operation2

Return the Id of the reserved block
Manage the NVM segmentation

Application

BLE SD FW Module

Low Power Manager

Interrupt

EEPROM

Embedded software description

AN4605

30/49

DocID027053 Rev 2

30/49
6.3.1 BLE SD FW module

The following interfaces from the BLE SD FW module shall be mapped to the NVM driver:

- pstorage_register()
- pstorage_block_identifier_get()
- pstorage_store()
- pstorage_load()
- pstorage_clear()
- pstorage_init()

A default mapping to the timer server is done in the file `ble_nvm.c`.

There is no need for the application to make any modification.

6.3.2 Initialization

- NVM_Init()
  
  This API shall be called before any request is done to the NVM driver.

6.3.3 Operations

- NVM_Register()
  
  This API reserves a block of the required size in the NVM and returns the base address to the user.

- NVM_Operation()
  
  This API is used to either write or erase data in the NVM.

  When a “write” operation is requested, the user shall provide the address of the data to be copied, the NVM destination address and the amount of data to be written. The data shall be kept available at any time from the source address as the NVM driver does not make temporary copy.

  When an “erase” operation is requested, the user shall provide the base address in NVM of the data to be erased and the amount of data to erase. The source address parameter when an “erase” operation is requested is not considered by the NVM driver and can be set to any value.

- NVM_Read()
  
  This API is used to read an amount of data from the NVM.

  The user shall provide the NVM start address, the destination address where to copy the data and the amount of data to copy.

  The NVM driver does not check whether the section to be read from the NVM may be in the operation queue to be written or not. It is the responsibility of the application to not read a section if it may be on the way to be written.
6.3.4 Low power manager

- NVM_HSclkRequest()
  This API notifies the application when the HS clock may be stopped and when it shall be kept enabled. The NVM driver sends the notification when the operation queue gets a command and when the operation queue is empty.
  There is no notification to the application for each command completed but this API may be used as well to make sure all commands (write/erase) that have been sent to the NVM driver have been completed.

6.3.5 Interrupt

- NVM_Flash_Interrupt_handler()
  This interrupt handler shall be called by the application in the FLASH interrupt handler. This handler takes care of clearing all status flag required in the FLASH peripheral.

6.3.6 Configuration

Some parameters of the NVM driver may be configured in the file stm32xx_nvm.h.

- NVM_OPERATION_QUEUE_SIZE: the user may define the maximum number of operations to be queued. It shall not exceed 255. Default configuration is 30
- NVM_BASE_ADDRESS: the user may define the base address in the NVM from where the NVM driver may allocate blocks to the user. There is no block size limitation checking in the NVM driver. Default configuration is 0x08080460 (The first 1120 bytes are reserved for BLE operation. 0x08080000 is the DATA EEPROM base address for STM32L0x3 device).
- NVM_IT_PRIORITY: the user may define the priority in the NVIC of the FLASH interrupt handler that is used to notify the NVM driver the end of programming. Default configuration is 3.

6.3.7 Integration

To include the NVM driver, the application shall:
1. Implement the file stm32xx_nvm_conf.h
   It shall include stm32xx_nvm.h and all header files providing the required API from the dedicated HAL (Refer to the STM32L0xx implementation example provided)
2. Set the configuration required in the file stm32xx_nvm.h
3. Include the file stm32xx_nvm.c in the project
4. Map the pstorage BLE interface on the HAL NVM driver (This step is already done in the wrapper implemented in the file ble_nvm.c. The application may just include that file in the project)
5. Implement the API NVM_HSclkRequest() to handle the HS clock requirement according to the NVM driver activity.
6. Update the FLASH interrupt handler to call the API NVM_Flash_Interrupt_handler()

6.3.8 Portability

The NVM driver is fully implemented on top of STM32Cube.
There is no update required in the NVM driver when moving to any STM32 device.
6.4 Low power manager interface

When the BLE feature is used, it requires the UART and NVM driver to be supported. Both drivers have some requirements on the availability of the HS clock. The request comes asynchronously to the application. In addition, the application may have some requirements on the availability of the HS clock that would come as well asynchronously to both the UART and NVM driver requests. There is a need for the application to implement a module that would take care of all these requirements and to compute the lowest power mode the system may enter at any time.

The low power manager provides a simple interface to receive the input of different users (up to 32) and computes the lowest power mode the system may enter. It provides as well hooks to the application before entering or on exit of low-power mode.

Figure 13. Low power manager interface overview
Figure 14. Low power manager design overview

Low Power Manager
- Update each shadow register depending on the requested mode
- Code running in critical section
- Evaluate each shadow register to define which mode to enter
- Update the hardware register to set the correct mode
- Code running in critical section
- Hooks for the application to execute code before or on return from low power mode
- Hooks are executed from critical section

STM32
(Deepsleep, LPSDR, PPDS)
6.4.1 **API**

- **LPM_Mode_Request()**
  This API shall be used by all the users to specify the lowest low-power mode it supports.

- **LPM_Enter_Mode()**
  This API shall be used by the application when there is no more code to execute so that the system may enter low-power mode.

- **LPM_ExitStopMode()**
  This API is called by the low power manager in a critical section (PRIMASK bit set) to allow the application to implement dedicated code before getting out from the critical section.
  This is where the application should reconfigure the clock tree if needed when getting out of Stop mode.

- **LPM_EnterStandbyMode()**
  This API is called by the low power manager in a critical section (PRIMASK bit set) to allow the application to implement dedicated code before entering standby mode.
  This is the location where the application could save data in the retention memory as the RAM memory content will be lost.

- **LPM_ExitStandbyMode()**
  This API is called by the low power manager in a critical section (PRIMASK bit set) to allow the application to implement dedicated code before getting out from Standby mode.
  This can only happen when the Standby mode is finally not entered. In that case, the application may reverse some configurations done before entering Standby mode.
  When Standby mode is successful, the system is reset when getting out from this low-power mode.

6.4.2 **Configuration**

The application shall configure in the file `stm32xx_lpm.h` the Id for each user supported and the list of low-power mode supported.

The default lowest power mode supported for all users is **Standby mode**.

6.4.3 **Integration**

To include the low power manager driver, the application shall:

1. Implement the file `stm32xx_lpm_conf.h`
   It shall include `stm32xx_lpm.h` and all header files providing the required API from the dedicated HAL (Refer to the STM32L0xx implementation example provided)

2. Set the configuration required in the file `stm32xx_lpm.h`

3. Include the file `stm32xx_lpm.c` in the project.

6.4.4 **Portability**

The Low Power Manager is fully implemented on top of STM32Cube.

There is no update required in the Low Power Manager when moving to any STM32 device.
6.5 BLE SD FW module firmware integration in application

The STM_ble_app project provides an application using the BLE SD FW module based on a scheduler. It comes with a Timer server, NVM driver and low power protocol UART driver to interface with the STM32 platform. Whenever the BLE function is enabled, the STM32 shall not enter standby mode, the application shall take care that STOP mode is the lowest low-power mode used.

The user may either implement its own services or integrate those which are provided in the project.

6.5.1 BLE startup

When the nRF device is in OFF mode, the application shall execute following steps to get BLE ready to be used:

1. Enable the TIMER and NVM services.
   - To enable the timer server, the application shall called the API TIMER_Init().
   - To enable the NVM driver, the application shall called the API NVM_Init().

2. Enable hardware resources required for the BLE low power protocol UART driver
   - All the resources as described in “5.1.6 STM32 resource requirements when BLE feature is used” are enabled in the API UART_BLE_Init(). The user may update the content of this API depending on its own implementation.

3. Reset and initialize the nRF device
   - The nRF device shall be first reset before running any initialization. The reset is applied when the API connectivity_chip_reset() is called. The embedded software drives low the IO connected to the nRF nRESET pin for a minimum amount of time. After that time, the software configures the IO to analog mode to release the nRF nRESET pin. The nRF nRESET pin should not be driven high to release the reset. The reset timing requirement may be found on the Nordic website.
   - After the nRF device has been reset, the embedded software runs the nRF initialization with the API ble_app_main_init_Reset(). All details on the nRF initialization may be found on the Nordic website.

6.5.2 Communication with the BLE SD FW module

The application sends requests or data to the BLE SD FW module with a SD command.

When a SD command is sent, the application shall wait for the response before sending another SD command. On each SD command, the BLE SD FW module calls the API blocking_resp_wait() to pause the BLE processing until the response is received. The application may enter low-power mode until the response is received. In addition, all the events that do not require sending a SD command may be processed as well over that time.

As described below, the current project provides a scheduler where the API blocking_resp_wait() is implemented in the file blocking.c to enter the background task when waiting for a response. When the embedded software enters the background task from the BLE SD FW module, all the events that do not require sending a SD command are processed.
6.6 Portability

The current delivery supports the STM32L0 family but the full implementation of the application is STM32Cube compliant which means it can easily be ported on any other STM32 families.

When porting to another STM32 family device, the user shall update the file `msp_main.c` that implements the API from the application that are specific to the STM32 device and the interrupt handlers in the file `stm32yyxx_it.c`.
7 BLE application configuration

The Nordic nRF51822 SDK provides examples how to manage the different profiles and services.

To allow customers to verify BLE profiles and services, two applications are available in the STM32 “Project” directory:

- Direct test mode application
  - “STM_dtm_app” project
- Heart rate monitoring/health thermometer application
  - “STM_ble_app” project

Figure 16. Application overview
7.1 Architecture

The application part contains the code source to manage the different BLE services. To use a BLE profile, several services are initialized. Next, the service manages the values characteristics, the notifications and BLE events by using the Soft Device functions and events.

The files “ble_services”.c are reused from the Nordic SDK.

7.2 Direct test mode application

The Direct test mode application is a embedded software example that manages to initialize the nRF51822 to enter direct test mode and redirect the nRF51822 UART to the tester.

- Project: STM_dtm_app
  - To initialize the DTM on nRF51822 side, the following function sends information to the nRF51822:
    - ble_dtm_init (&uart_params)
  - The DTM uart parameters are configured by the following defines:
    - #define DTM_RX_PIN_NUMBER                10
    - #define DTM_TX_PIN_NUMBER                 11
  - 57600 is the default Uart Baud rate

Next, the STM32 after having sent the DTM command goes to Standby.

When in DTM test mode the UART between the STM32 and nRF51822 does no longer operate. The nRF51822 communicates directly to the BLE tester via the UART on the configured set of IO’s.

If needed the conversion from 3.3V to RS232 is done outside the Wavetek Bluetooth LE shield by a RS232 to 3.3V TTL converter. (i.e. SerialComm TTL-232-33P).

![Figure 17. Application DTM access](image)
7.3  Heart rate monitoring / health thermometer application

The “STM_ble_app” project allows the customers to enable both the heart rate service or/and the health thermometer service.

In case both “compilation switches” are not used, the embedded software just initializes and starts the advertiser parameters.

The configuration is done with app_conf.h file.
- #define APP_HRS
- #define APP_HTS

The application supports the bond manager. The identification information exchanged between the central device and the peripheral device are stored in the memory to be used to identity verification when they reconnect in the future.

After the startup of the application, pushing the button “B1” erases all the stored bonding information. It is also recommended to delete the bonding information on the remote device.

Once a BLE connection is done, the button “B1” is used to update and provide the heart value to the connectivity device.

### Table 4. configuration of the BLE application

<table>
<thead>
<tr>
<th></th>
<th>ADV only</th>
<th>HRS Only</th>
<th>HTS only</th>
<th>HRS &amp; HTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Device name</strong></td>
<td>STM32_ADV</td>
<td>STM32_HRM</td>
<td>STM32_HRS</td>
<td>STM32_HRM_HTS</td>
</tr>
<tr>
<td><strong>Manufacturer name</strong></td>
<td>STM32</td>
<td>STM32</td>
<td>STM32</td>
<td>STM32</td>
</tr>
<tr>
<td><strong>Advertising interval</strong></td>
<td>40ms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Connection interval(1)</strong></td>
<td>NA</td>
<td>1s</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Conn sup timeout</strong></td>
<td>NA</td>
<td></td>
<td>4s</td>
<td></td>
</tr>
<tr>
<td><strong>BLE services</strong></td>
<td>NA</td>
<td>Heart rate battery device information</td>
<td>Health thermometer</td>
<td>Heart rate battery device information health thermometer</td>
</tr>
</tbody>
</table>

1. The connection interval is proposed by the server once the service notification is requested by the client.

### 7.3.1 Advertising application

In case, both “APP_HRS” and “APP_HTS” are not defined, the application manages the advertising interval, the device name, the GAP configuration, and connection establishment according to the following parameters. The application also manages the connection, disconnection and timeout events.

### 7.3.2 Heart rate application

Once the “APP_HRS” is defined, the application includes:
- Heart rate service
- Device information service
- Battery service
The application reuses the Nordic services code example.

- ble_hrs.c
- ble_dis.c
- ble_bas.c

The module ble_hrs.c implements the heart rate service with the heart rate measurement, the body sensor location and the heart rate control point characteristics. During initialization it adds the heart rate service and heart rate measurement characteristic to the BLE stack database. It also adds the body sensor location and the heart rate control point characteristics.

The module ble_dis.c implements the device information service. During initialization it adds the device information service to the BLE stack database. Then it encodes the supplied information, and adds the corresponding characteristics.

The module ble_bas.c implements the battery service with the battery level characteristic. During initialization it adds the battery service and the battery level characteristic to the BLE stack database. It also adds a report reference descriptor to the battery level characteristic (used when including the battery service in the HID service).

By calling the previous modules, the application initializes the services.

- Initializes the heart rate service
  - Sensor location: Finger (can be changed easily by a define)
  - Security level for the heart rate service measurement attribute
  - Security level for the body sensor location attribute
- Initializes the battery service
  - Security level for the battery characteristics attribute
  - Security level for the battery report read attribute
- Initializes the device information service
  - Manufacturer name: “STM32”

The application manages the GAP configuration, and the connection establishment according to the Bluetooth specification heart rate profile.

The heart rate values are provided by an application sensor module (ble_sensorsim.c”), there is no external sensor.

Different timers are started to simulate:

- Battery measurement
- Heart rate measurement
- RR interval
- Sensor contact detection

At the first connection, the parameters are provided by the client side.

The connection interval is requested by the application once the heart rate notification is enabled by the server. (ble_conn_param.c)
7.3.3 Health thermomether application

Once the “APP_HTS” is defined, the application includes:

- Health thermomether service
  The application reuses the Nordic service code example.
- ble_h.ts.c
  The module ble_h.ts.c implements the health thermometer service to provide to the client the temperature in degree celsius, the temperature type and the time Stamp.
  The temperature value is measured by the internal STM32 temperature sensor. There is no external sensor.
  At each indication enabled by the remote, the temperature is measured, provided to the nRF51822 and finally transferred to the remote.
  The connection parameters are provided by the client side and may be changed by the application.
8 Performances results

This chapter relates the system performance measured for power consumption and data throughput. It describes efficiency of the overall system in terms of low power consumption along with maximum data throughput achievable.

8.1 Static power consumption

The referenced STM32 static power consumption numbers such as Standby mode, Low-power, Stop mode, have to be taken from the STM32 datasheet reference [6].

8.2 SD command exchange power consumption profile

The next figures show power value diagrams of the power consumption on STM32 devices for UART message processing through the BLE SD FW module:

In UART receiving mode, the current vs time using #APP_HRS only:

As example: start advertising command, with Tactive - 800 us (typ).
- RUN mode: 40% (typ)

![Figure 18. RxSD command exchange power consumption](image-url)
As example: start advertising command, with Tactive - 458 us (typ).
- RUN mode: 48% (typ)

### 8.3 Connection average power consumption

To predict power consumption in a dynamic approach, typical profile scenarios are selected, on which the device is connected to the mobile phone running Android or iOS BLE application (as shown in Figure 20).

The heart rate monitoring application is used to measure the system power consumption.
The Table 5 indicates the average power consumption for HRM/HTS profile providing datas to the mobile phone at different connection interval.

The system configuration is set as below:

- 1 x STM32Nucleo/64 board revC with STM32L053R8
- 1 x Wavetek Arduino shield with BLE module nRF51822v1.0
- Voltage: 3.3 V
- STM32 CPU HCLK Clock: 8 MHz
- UART: 1 MegaBaud
- Ambient temperature: 25°C
- Firmware: STM_ble_app (#define APP_HRS)
Performances results

8.4 Data throughput

The following data throughput is measured using the same system configuration as described previously.

To reach maximum data throughput over the air between The STM32-nRF51822 BLE system and BLE remote device, the connection interval is fixed to 7.5ms and data packet are sent via L2CAP Layer.

<table>
<thead>
<tr>
<th>Table 5. HRS profile average power consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STM32L053R8#APP_HRS</strong></td>
</tr>
<tr>
<td>(Connection interval &amp; heart rate update)</td>
</tr>
<tr>
<td>nRF51822 power consumption (µA)</td>
</tr>
<tr>
<td>LP_STOP mode</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Note: The numbers are typical, measured on a single STM32 + nRF51822 solution for reference only.

<table>
<thead>
<tr>
<th>Table 6. Maximum data throughput L2CAP average power consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STM32L053R8 #APP_L2CAP_TX_TEST</strong></td>
</tr>
<tr>
<td>(L2CAP throughput Test)</td>
</tr>
<tr>
<td>Connection interval</td>
</tr>
<tr>
<td>Average Payload Throughput</td>
</tr>
<tr>
<td>nRF51822 power consumption(µA)</td>
</tr>
<tr>
<td>LP STOP Mode</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

The frontline BLE air sniffer is used to capture in real time all data transmit over the air.
9 System limitations

- The STM32 shall not enter standby mode when BLE is enabled.
## 10 Revision history

Table 7. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-Nov-2014</td>
<td>1</td>
<td>initial release.</td>
</tr>
<tr>
<td>19-Feb-2016</td>
<td>2</td>
<td>Updated the whole document replacing X-CUBE-nRF51DRV by I-CUBE-nRF51DRV.</td>
</tr>
</tbody>
</table>