UM11038

OM27642 Smart Lock KIT

Rev. 1.1 — 11 October 2018
407611

User manual
COMPANY PUBLIC

Document information

Info	Content
Keywords	OM27462NBR Smart Lock KIT User manual
Abstract	This document describes the content, the hardware implementation and the usage of the Android application as well as the access token structure and processing of the NFC Bluetooth® Low Energy Smart Lock KIT - OM27462NBR. It also describes how to update the QN9021 and PN7462 firmware.

Revision history

Rev	Date	Description
1.1	20181011	• Fig. 14 updated • Editorial updates
1.0	20161116	First release

Contact information

For more information, please visit: http://www.nxp.com

1. Introduction

This document describes the implementation of a battery-operated Bluetooth® Low Energy NFC Smart Lock for the hospitality market. This document does not replace the PN7462AU or QN9021 datasheets and respective application notes. Please refer to the datasheets for detailed information of the ICs. This document also describes the process of firmware upgrade for both the PN7462 and QN9021 (QS9322 module) as well as for the implemented demonstration functionality.

The Smart Lock KIT hardware design consists of two PCBs and incorporate the following main IC components:

- **PN7462AU** – all-in-one full NFC application controller with freely programmable µController
(*PN from now on*)
- **QN9020** – for Bluetooth Low Energy communication (*QN from now on*)
- **PCF8883T** – for capacitive wake up of the application processor circuitry (PN7x62)
- **NX5P2553 -** precision adjustable current-limited power switch

The PN and the QN ICs communicate with each other by means of serial communication over a UART interface.

Each of the micro-controllers contains its own firmware which performs the required actions depending on the origin of the unlock token. The token can be delivered to the lock in two ways:

- **over RFID** – with the help of an MIFARE DESFire ISO14443 Type A card or via an Android based phone using HCE.
- **over Bluetooth Low Energy** – with help of any Bluetooth Low Energy compliant device

In this document the term „MIFARE DESFire card“ refers to a MIFARE DESFire IC-based contactless card.

2. Smart Lock KIT Package Content

Fig 1. OM27462NBR Smart Lock module, power cable, programming cable, touch sensor plate, set of demonstration cards

3. Smart Lock Module Hardware

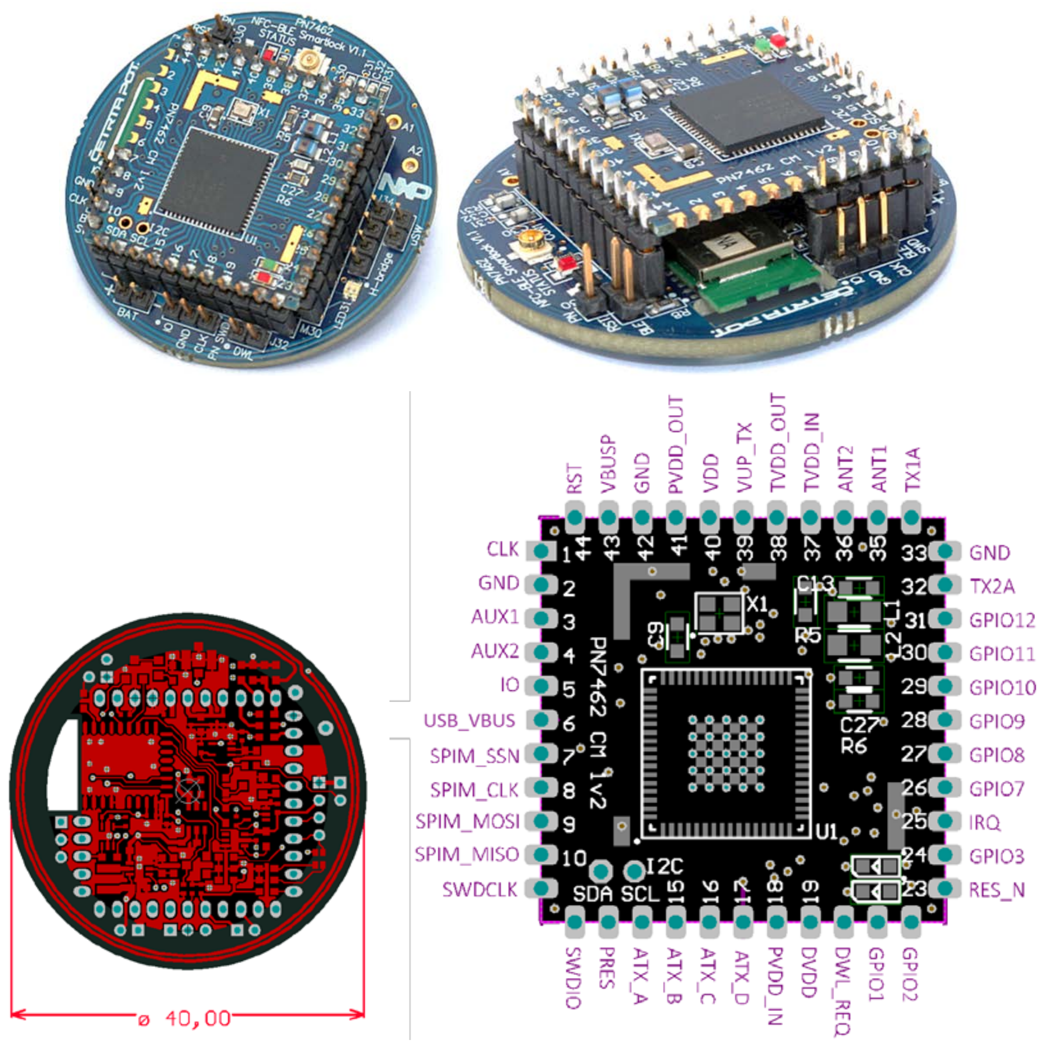

Fig 2. **OM27462NBR Smart Lock module consisting of PN7462AU core module V1.2 and the 40mm Ø base board NFC- Bluetooth Low Energy Smart Lock V1.1**

The core module – stack on board contains all necessary blocking capacitors, XTAL, 2 status LEDs and the antenna EMC filter.
The Core module is designed for general purpose use therefore all power supply options (e.g. internal/external LDO etc.) are configured on the base board. The core Module / stack on approach was taken in order to create a high density low PCB area implementation.

3.1 Smart lock system block diagram

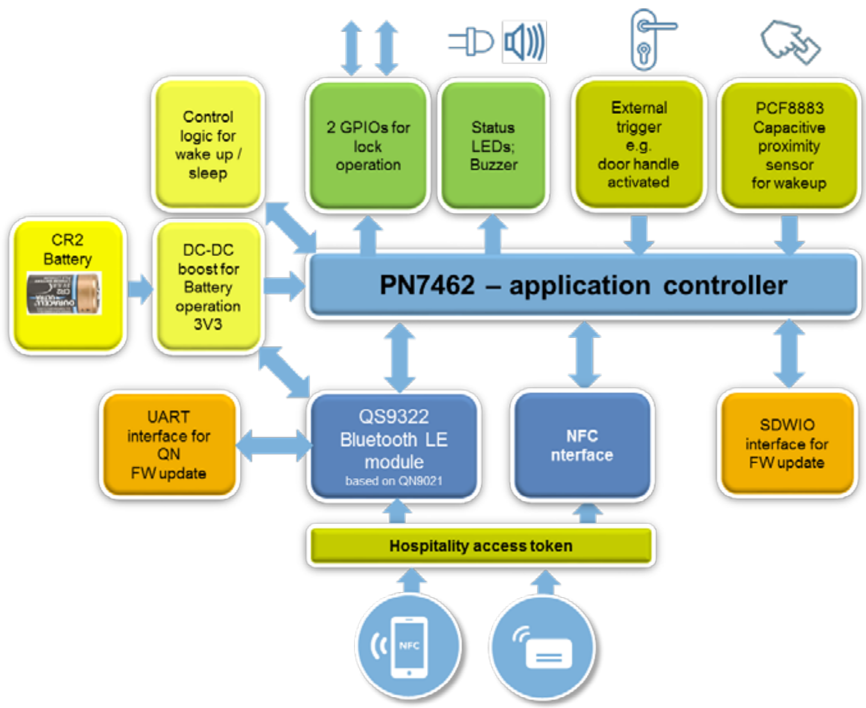

Fig 3. Main building blocks

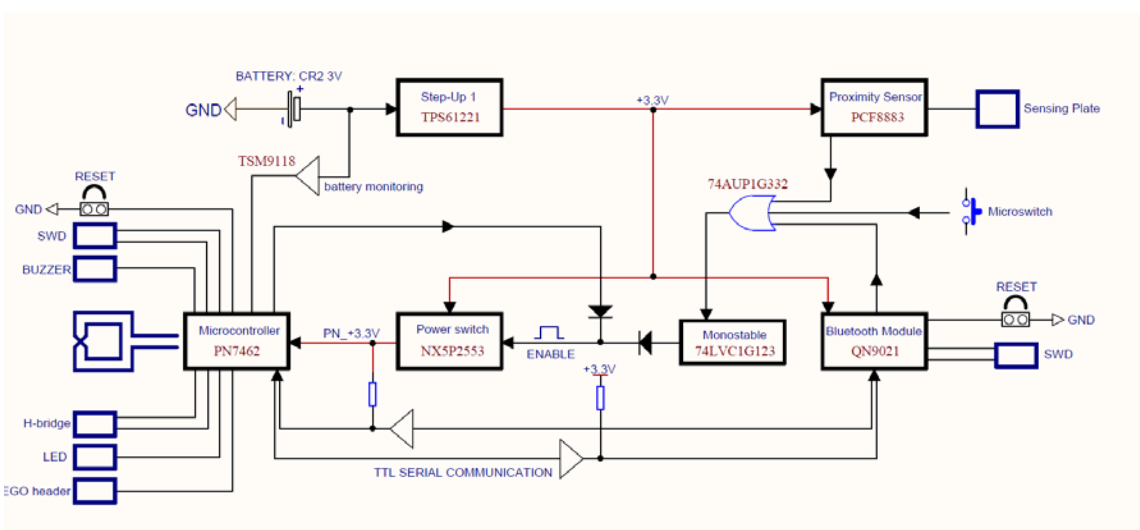

Fig 4. OM27462NBR NFC- Bluetooth Low Energy Module with PN7462 stack on core module

Figure 4 shows the functional block diagram of the Smart Lock module including the main components used.

4. Schematics

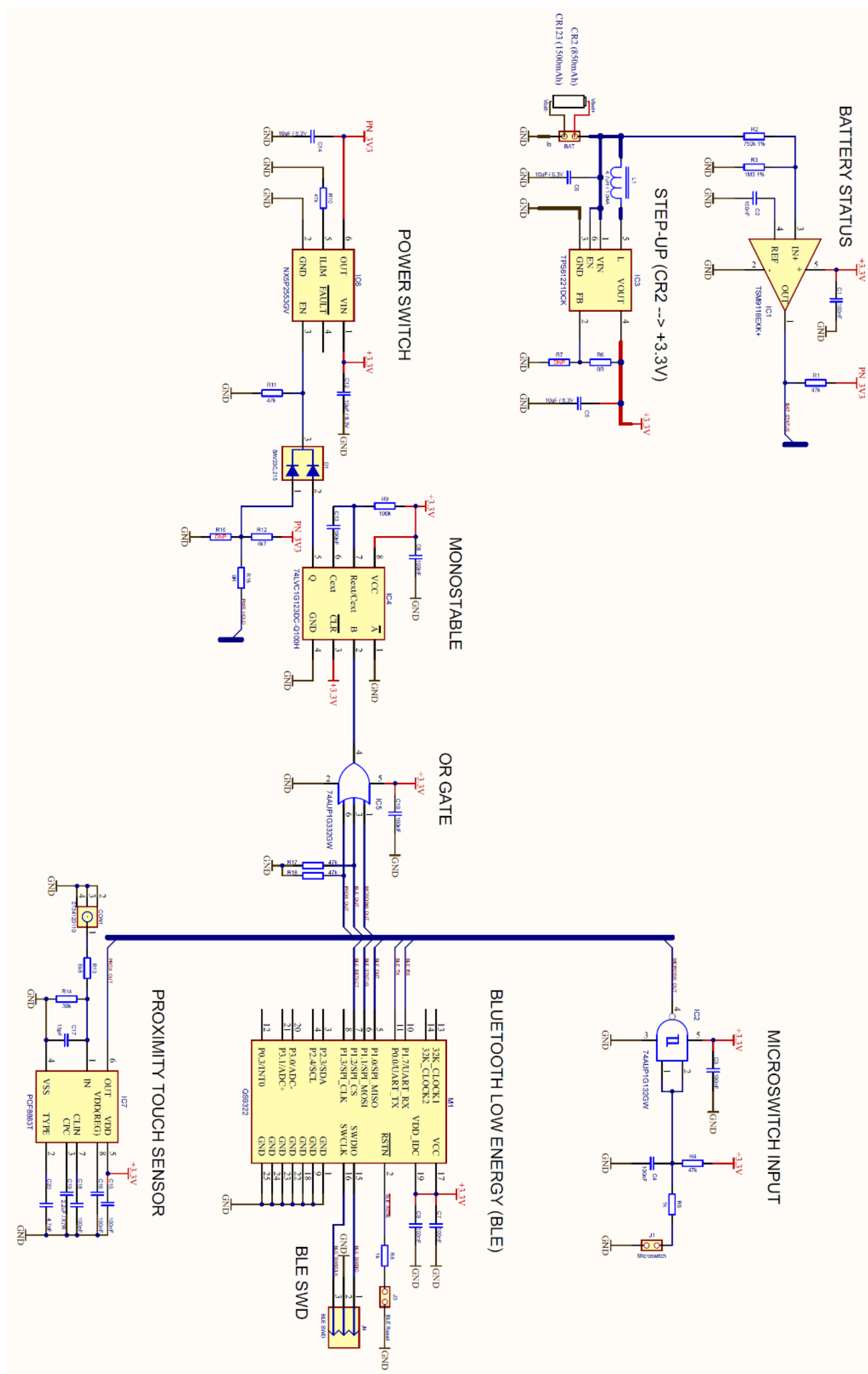

Fig 5. Base board schematic part 1

4.1 Base board schematics continued

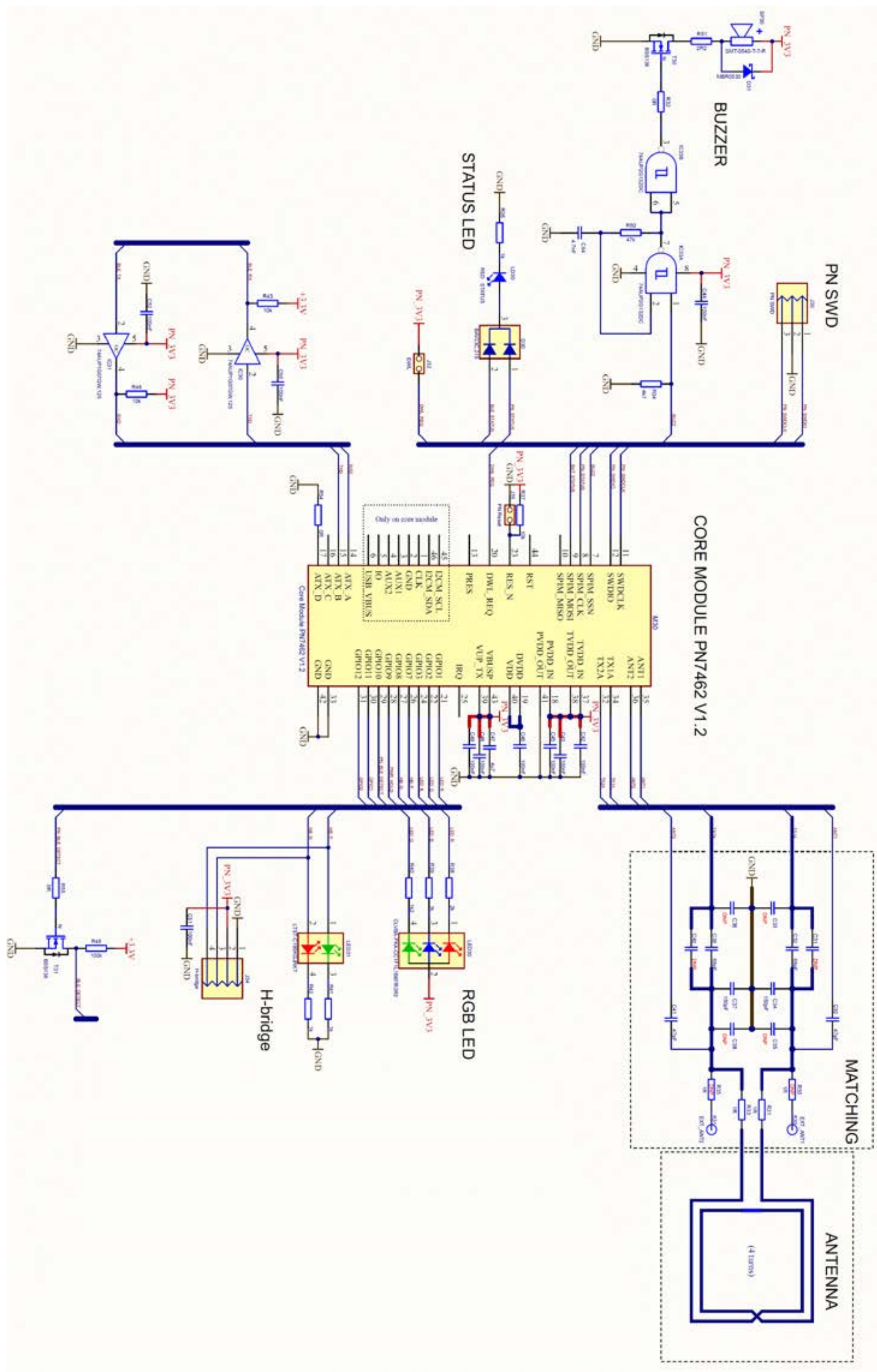

Fig 6. Base board schematic part 2

4.2 Core Module schematics

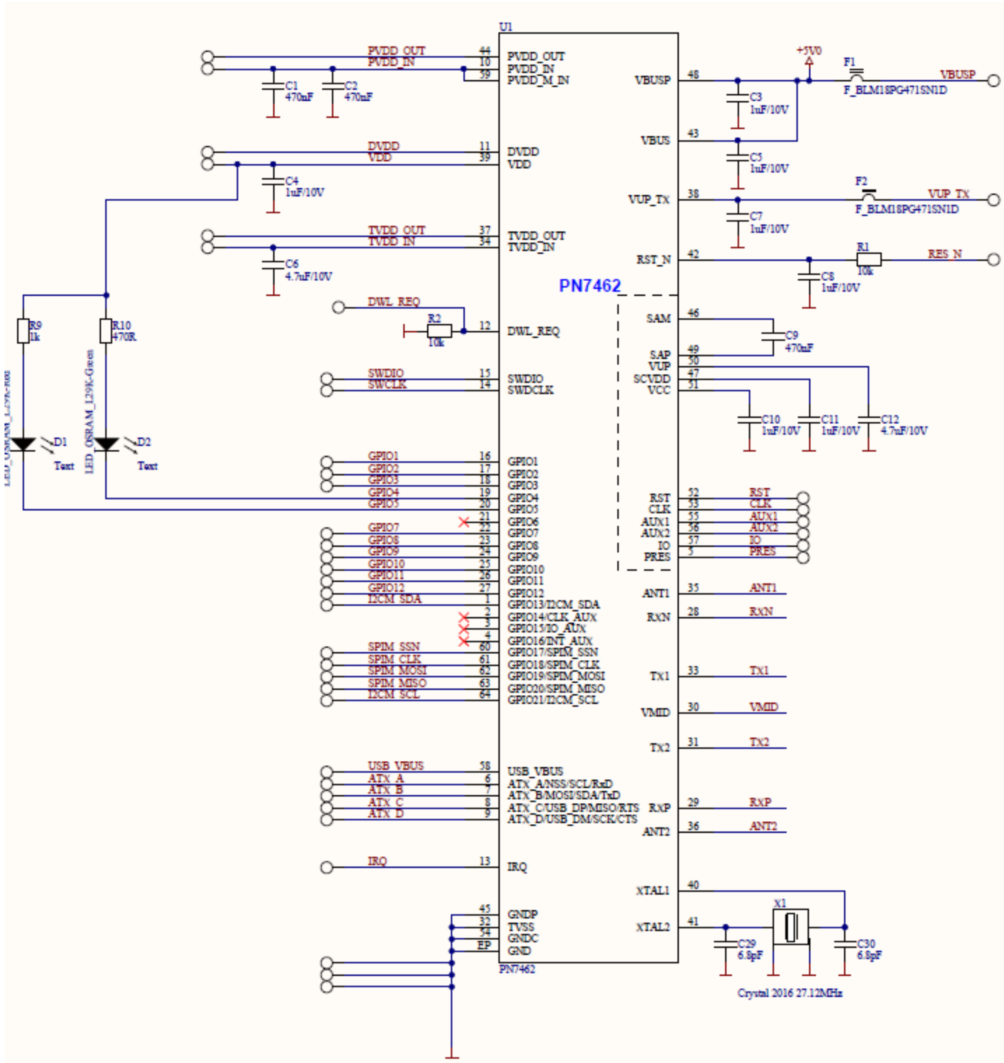

Fig 7. Core Module pin out vs. PN7462AU pin out

4.3 Antenna matching

The antenna matching of the Smart Lock main PCB is shown below. The EMC filter is part of the stack-on core Module. The base board consists of the matching circuit with the damping resistors and the serial and parallel matching capacitors.

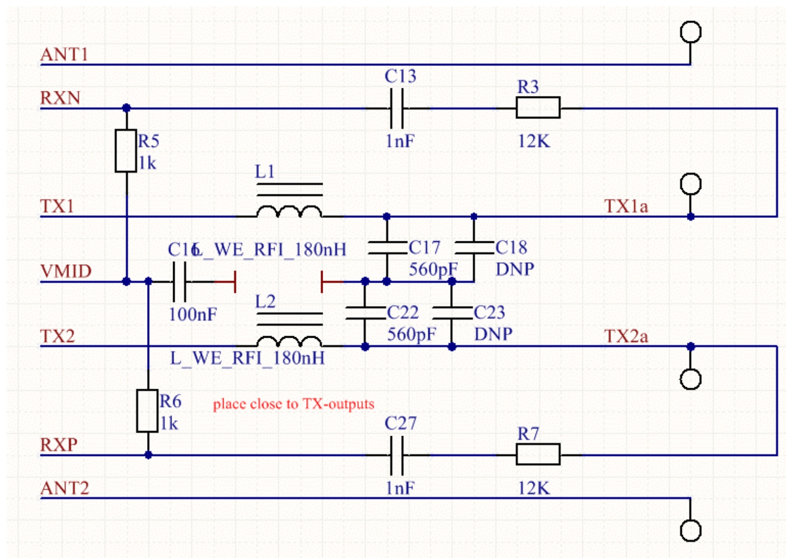

Fig 8. Antenna matching on stack on Core Module

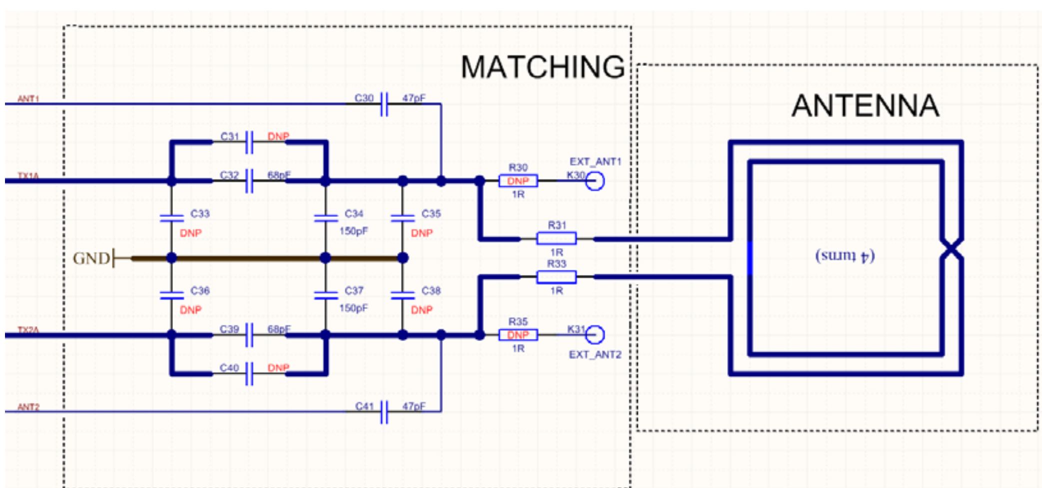

Fig 9. Antenna matching components on the base board

Please Note: high resolution schematics are available as PDF files in the download section under manufacturing data here: www.nxp.com/demoboard/OM27462NBR

5. Base board connectors

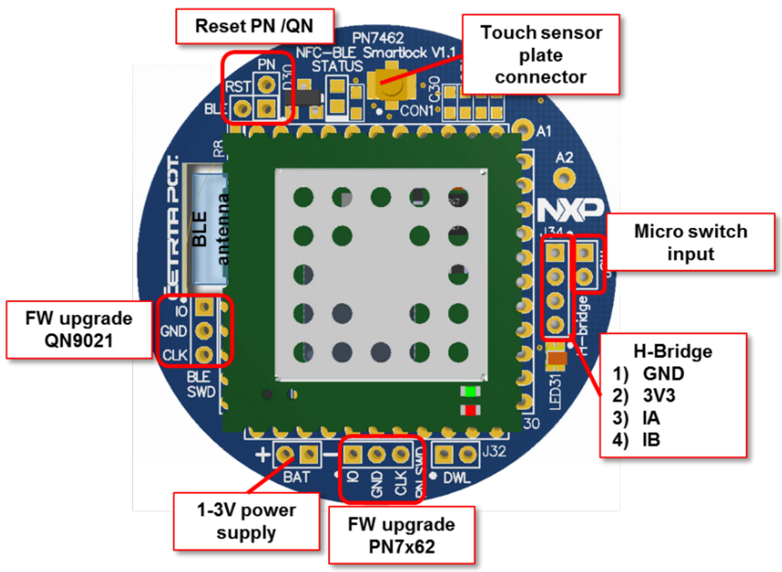

Fig 10. Base board connection description

6. Getting started

6.1 Setting up the hardware module

1) Connect the Touch sensor plate as shown in fig. 1.
2) Connect a DC power source (2V-3.3V) to the pins labeled "BAT" (see fig. 1)

6.1.1 Smart Lock power connection

As the Smart Lock is intended to be operated by a single cell battery (e.g. CR2 ~780 mAh or CR123 ~1,450 mAh) with a nominal voltage of 3V (typically ~2.5 V depending on load current). An ultra-low-power step up converter which operates down to below 1 V input voltage is used to generate 3.3 V circuitry supply.

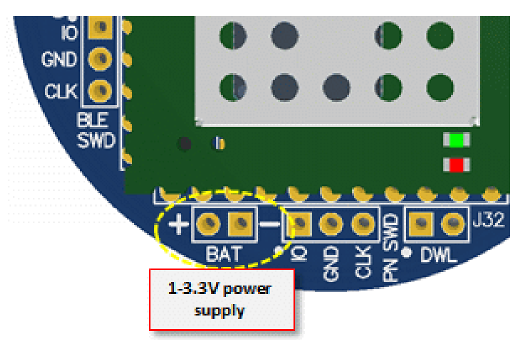

Fig 11. Power Supply Connector location on Smart Lock base board

7. Smart Lock operation

The Smart Lock KIT comes with pre-configured functionality.

7.1 Operating the Smart Lock using the demonstration cards

1) Configure Room 201 by bringing the “Lock Configuration Room 201” card in close proximity of the module. Wait for lock configuration sequence completion which is indicated by the RGB LED and on-board buzzer while active.

2) Present “Room 201 ACCESS Guest 1” card or “Room 201 ACCESS Guest 1 re-issued” card in close proximity of the module. If the card is valid and accepted, LED will light up green and the buzzer will give a brief beep indicating acceptance. If an invalid card is used, the LED will light up in red and the buzzer will beep 3 times.
Note: By presenting the “Room 201 Guest 1 re-issued” card, the “Guest 1” card will be invalidated and not accepted further until you re-configure Room 201 with the respective configuration card.

3) In order to set the Smart Lock to an alternative room / guest, re-configure the lock to room 202 / guest 3 with the respective “Lock Configuration Room 202” card. Guest 3 will be accepted whereas Guest 1 – Room 201 cards are declined.

Fig 12. Set of demonstration cards

7.2 Operating the Smart Lock using the mobile Smart Lock App

Download the “NXP SMARTLOCK” app from Google Play Store and install it on your mobile NFC enabled Android device.

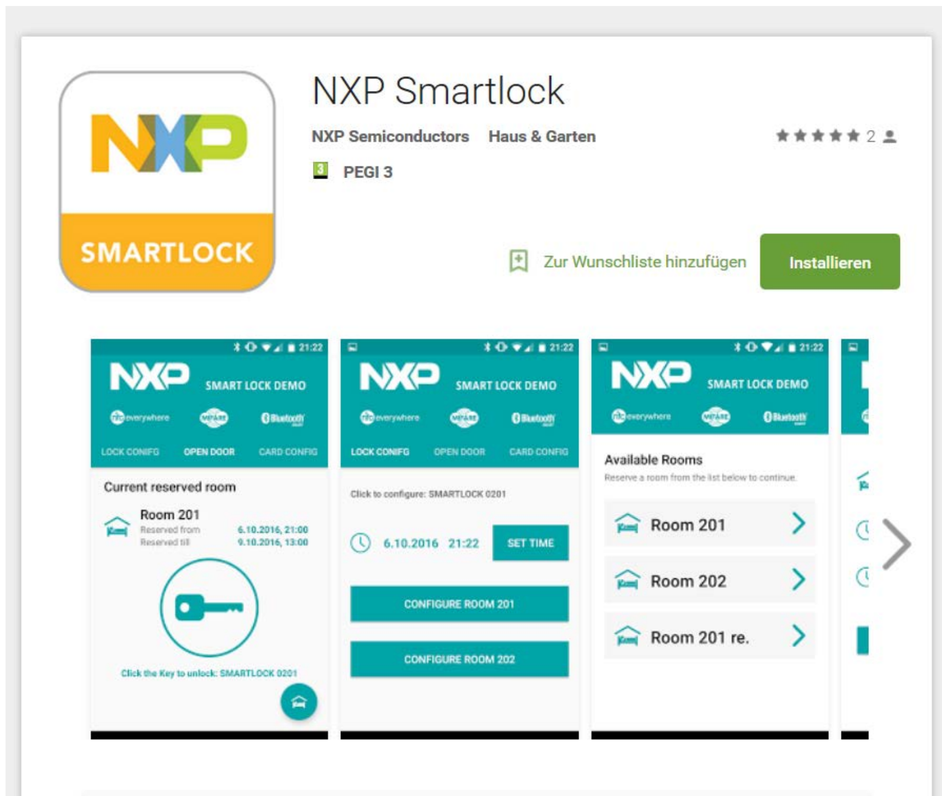

Fig 13. [Download link Google Playstore]

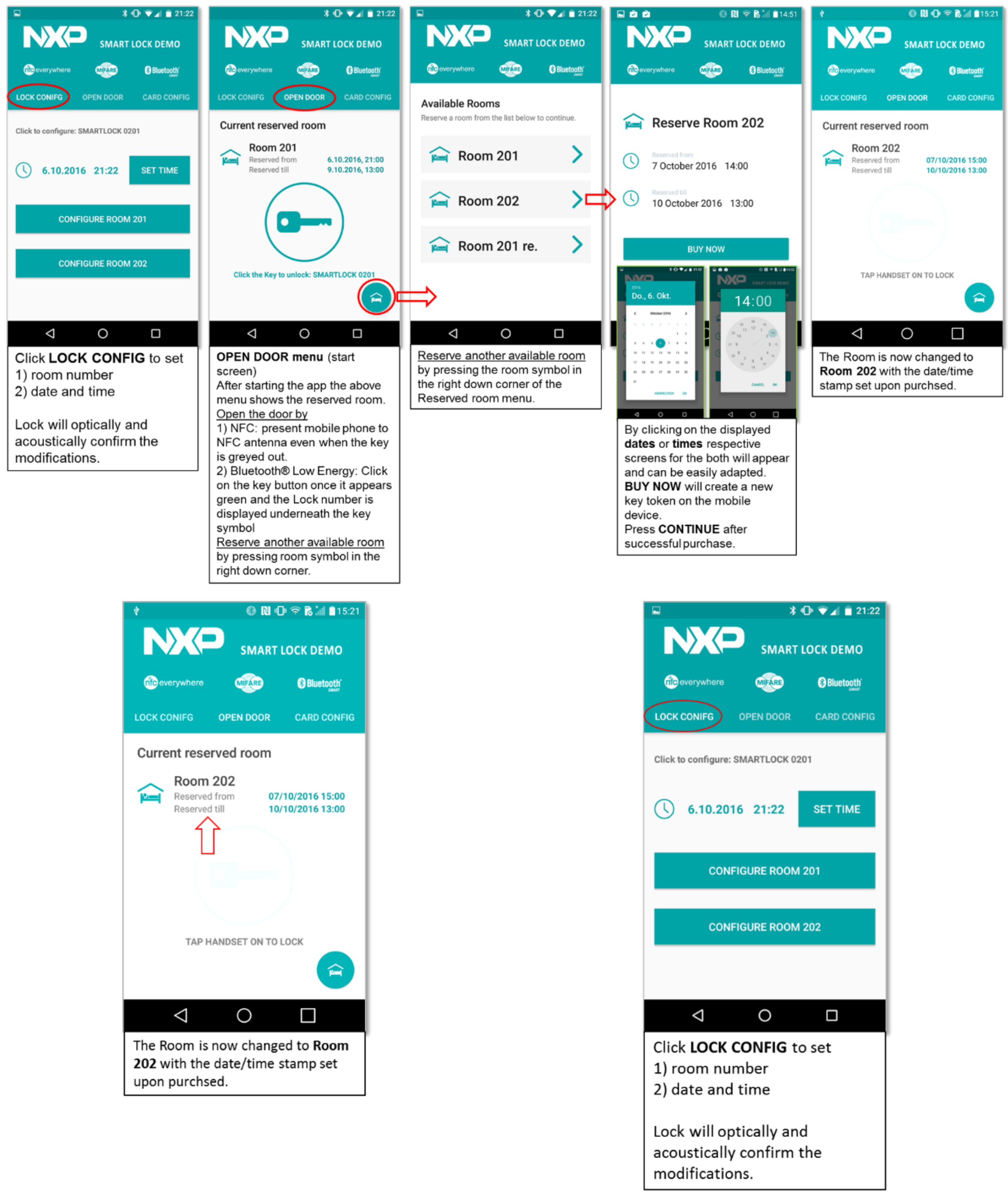

Fig 14. Quick reference for Smart Lock App

8. Firmware design description

8.1 Power saving

As RFID and Bluetooth Low Energy Smart Lock Reference Design is powered by a battery it is important that we consume as little energy as possible. The PN spends most of its time shut down and can be woken in multiple ways:

- over proximity touch sensor switch
- over micro-switch – external input
- by QN (Bluetooth Low Energy module)

QN is powered all the time and sends out Bluetooth Low Energy *advertisements* periodically so that Bluetooth Low Energy devices can discover the lock. Between Bluetooth Low Energy advertisements broadcasts it is running in low power mode. In this mode most peripherals are switched off and QN is running on 32 kHz clock to support RTC. PN can wake up QN from low power mode any time by interrupt on GPIO when PN requires the current time from QNs RTC to check the token.

8.2 Interaction between PN and QN when token is delivered over RFID

When the token is delivered over RFID the PN gets powered up by means of proximity sensor or switch and starts polling for RFID cards or NFC phones with HCE. If a device is found it tries to retrieve the access token. After successful retrieval of the token, the validity of the token must be checked. As time and date is required to do so, the PN must wake up the QN to retrieve the current date and time from QNs RTC. After date/time is retrieved from the QN and the token is deemed valid the process of unlocking the door is started.

8.3 Interaction between PN and QN when token is delivered over Bluetooth Low Energy

When the token is delivered over Bluetooth Low Energy, the QN powers up the PN after a connection was established successfully between the QN and the Bluetooth Low Energy client. Commands to retrieve the token are issued by the PN. After successful retrieval of the token, validity of the token must be verified. If the token is deemed valid the process of unlocking the door proceeds.

9. Communication protocol between QN and PN

Communication between QN and PN is done using two GPIO lines and UART communication (115200 bps and data format 8N1).

The GPIO lines are used to power the PN (in case a Bluetooth Low Energy client connects) or to wake-up QN (in case a card or phone is presented to the PN via NFC).

The UART is used to transfer data for

- Handshake
- Data exchange

Data is transferred using a single character command, a 2-byte length indicator and the data bytes. All data is sent using little endian format.
Only during the handshake phase the PN sends repeatedly a single character 'P' without any additional bytes until QN responds.

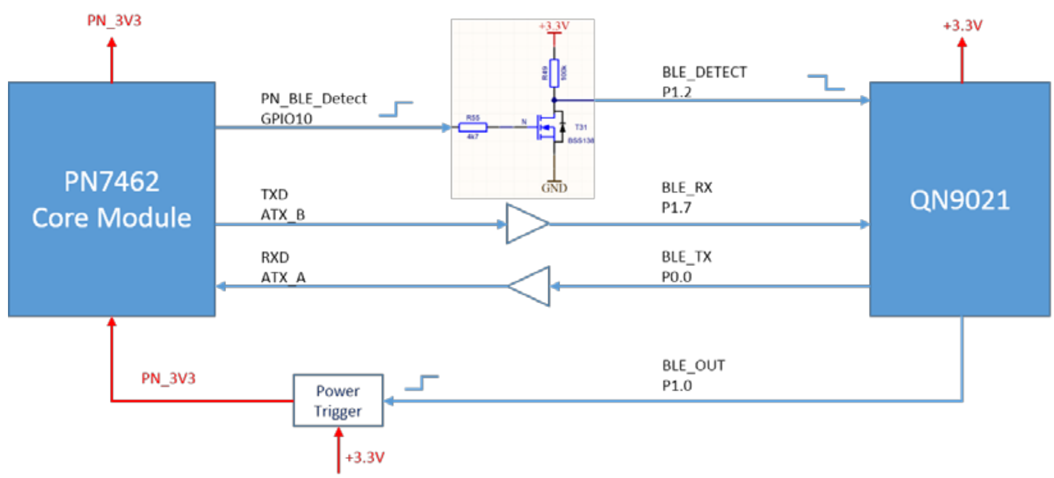

Fig 15. Communication interface between QN and PN

10. Handshake between the QN and PN

According to the use cases NFC and Bluetooth Low Energy communication, two scenarios exist to start the communication between QN and PN

10.1.1 NFC Transaction

In this use case the PN is powered up either by the proximity switch or the micro switch. In both cases the PN has to wake-up the QN to leave low power mode and activate its UART peripheral. Therefore, the PN triggers a GPIO interrupt using the BLE_DETECT line. At the same time the PN sends ‘P’.

This procedure is repeated until QN responds with ‘V’ and a 4-byte UNIX timestamp (little endian).

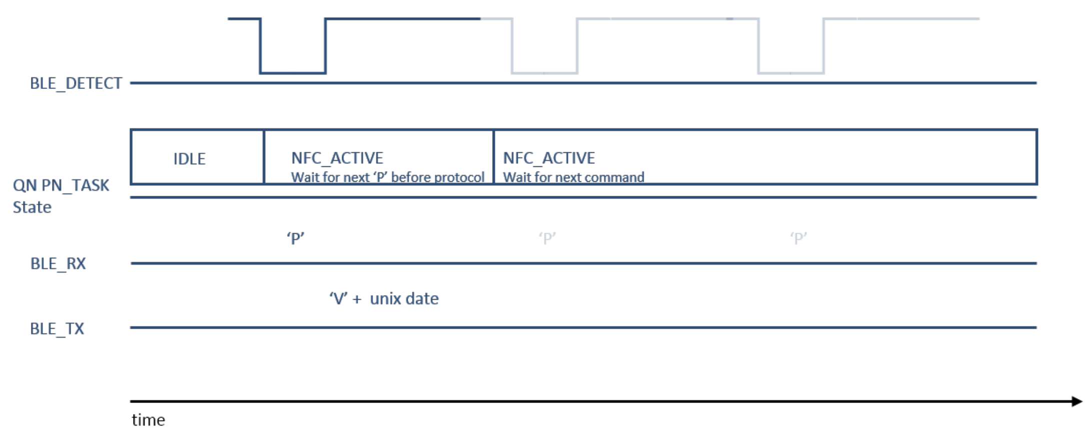

Fig 16. Communication procedure between QN and PN - details

Example:

PN		QN
0x50	→	
	←	0x560400FB435E54

10.1.2 Bluetooth Low Energy Transaction

In this use case QN will power PN using the BLE_OUT line after establishing the Bluetooth Low Energy connection. After power on the PN will start the communication by triggering the QN GPIO interrupt using the BLE_DETECT line. At the same time PN will send ‘P’.

This procedure is repeated until QN responds with ‘O’ and a 4-byte UNIX timestamp (little endian).

The following figure shows the whole procedure:

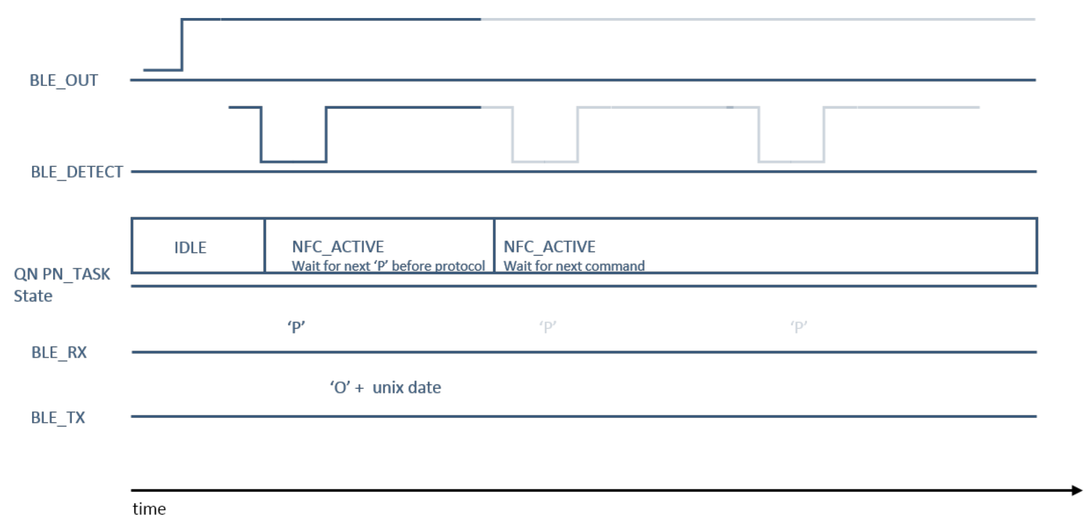

Fig 17. Bluetooth Low Energy Transaction - details

Example:

PN		QN
0x50	→	
	←	0x4F0400FB435E54

10.2 Data exchange

After the initial handshake data exchange may start. The following commands are available:

- ‘C’: PN sends APDU command to QN (QN forwards data to the Bluetooth Low Energy client)
- ‘A’: QN responds with response APDU
- ‘T’: PN sets RTC in QN
- ‘N’: PN sets a new Bluetooth Low Energy device name in QN.
- ‘D’: PN indicates QN that the transaction is finished and communication shall be disconnected

All commands are followed by two bytes for length and then data. In case of *'T'* command the answer is the same as at power up (*'O'<len><time>*), but for *'C'* command we return *'A'<len><data>*:

PN		QN
'C'<len><data>	→	
	←	'A'<len><data>

Example:

PN		QN
0x4308000102030405060708	→	
	←	0x4102009000

11. Structure of the QN Firmware

The QN firmware is built on NXP Bluetooth Low Energy library. This library implements a proprietary profile which is optimized sending and receiving APDU commands. It automatically splits data onto several Bluetooth Low Energy packets if necessary. In addition, the library uses the Bluetooth Low Energy feature to send several Bluetooth Low Energy packets in one communication interval to speed up communication.
All data is transferred using unacknowledged Bluetooth Low Energy packets:

- Write Commands (Bluetooth Low Energy client to QN)
- Notifications (QN to Bluetooth Low Energy client)

Data sheet QN902x: http://www.nxp.com/documents/data_sheet/QN902X.pdf
Application note: http://www.nxp.com/documents/application_note/AN11664.pdf

The communication with PN is basically done by two modules:

- app_pn_task.c: The PN TASK handles the whole communication with the PN. It controls the state, interprets the commands and starts/stops the communication
- pn_nfc.c: driver to communicate with PN. This module handles the low-level functions to communicate with PN. It contains functions which run the context of the GPIO interrupt as well as UART interrupt. In case of an event this module will send a message to the PN TASK which will handle the event in a synchronous way.

Note: In the download area only the binary files are provided.

11.1 src\app\pn\app_pn_task.c

The PN TASK has the overall responsibility to take actions and control the state of the QN. This means that all input from PN (UART data but also GPIO wake-up events) is converted to messages and sent via messages queues to the PN TASK.

To send messages to the PN TASK this module provides global functions which build and send the appropriate messages.

From the QN point of view two events may start a transaction and thus the communication with the PN:

- PN is woken up by proximity sensor or micro switch and wants to communicate with QN. The handshake sequence may start immediately.
- A Bluetooth Low Energy client connects to QN. QN has to power on PN before starting with the handshake sequence.

The PN_TASK basically uses three app states:

- PN_STATE_IDLE: QN advertises and uses low power mode. Sleep mode is not possible since RTC must be powered. The task is waiting either for a Bluetooth Low Energy connection or a wake-up event from PN via GPIO line.
- PN_STATE_BLE_ACTIVE: A Bluetooth Low Energy Client has connected and subscribed to receive notifications from QN. In this state the PN_TASK acts as a kind of proxy between the Bluetooth Low Energy client and the PN, meaning it transfers APDU commands from PN to the Bluetooth Low Energy client and APDU responses in the other direction.
- PN_STATE_NFC_ACTIVE: PN is powered up (e.g. proximity sensor) and wants to read the current RTC value from QN. In this state no APDUs are exchanged since those are handled locally between the PN and a NFC medium directly. Still

the QN is ready to accept commands from PN like reconfiguring the device name (e.g. room number is changed).

The main handlers in active state are the functions

- `pn_uart_rx_ind_handler()` It processes messages sent from the pn driver and thus evaluates the commands received from PN.
- `pn_ble_rx_ind_handler()` It forwards response APDUs from the Bluetooth Low Energy client to the PN

11.2 src\driver\pn_nfc.c

This driver is responsible for:

- wake-up QN in case PN wants to communicate
- power on PN
- receive and evaluate data from PN, build messages from incoming data and forward these messages to the PN TASK
- build and send messages to the PN

When PN starts a transaction (e.g. is powered due to proximity sensor) it will trigger a GPIO interrupt on line BLE_DETECT. The GPIO interrupt handler calls `pn_nfc_gpio_interrupt_cb()`. This callback function processes the event and sends the message PN_NFC_ACTIVATE_REQ to the PN TASK. Before it will block additional GPIO events to avoid sending multiple messages (`allow_wakeup=0`).

ATTENTION: To be sure that the QN OS Kernel is active, the library function `sw_wakeup_ble_hw()` must be called. Otherwise the sent message could be processed only after a considerable delay.

The PN_TASK will then activate the UART by calling the function `pn_nfc_activate()`

The original uart driver from the QN SDK is modified to work with pn_nfc.c. The define `UART0_PN_EN` activates a different receive mode in uart.c (in contrast to the original QN SDK procedure). All characters being received via UART are forwarded via callback function to pn_nfc.c: `static void pn_uart_rx_data(uint8_t rx)`

During initial handshake phase all characters different from ‘P’ are being ignored (static variable `ignore`). This shall filter possible spurious signals on the serial line when PN is powered on.

`pn_uart_rx_data` processes all incoming data and assembles it to messages. After having received a full message from PN it is forwarded to the PN TASK. It is then the responsibility of the PN TASK to take appropriate actions.

If PN TASK wants to send messages to PN it uses the function `pn_nfc_send_response(uint8_t cmd, const uint8_t *data, uint16_t len)`

After the transaction is finished PN TASK it calls pn_nfc_power_off(). This will also deactivate the UART to save power. In fact, the PN has to power off itself. The BLE_OUT line is set to low, so it can retrigger power for the PN later on.

11.3 Interaction with Bluetooth Low Energy Stack

NXP Bluetooth Low Energy Lib only needs few interfaces to exchange data with the Bluetooth Low Energy Client. When the Bluetooth Low Energy Client sends a BLE write command the function `gatt_write_cmd_ind_handler()` in file smx_task.c gets called. This file is part of the proprietary profile.

In this function the PN_TASK is called in two places:

- When the Bluetooth Low Energy Client connects and registers to receive Bluetooth Low Energy notifications the PN_TASK is activated by calling `pn_send_ble_activate_req(smx_env.con_info.conhdl)`
- During normal operation the NXP Bluetooth Low Energy Lib collects data from 1 or more Bluetooth Low Energy frames. After the payload frame has been received completely, the function `pn_send_ble_rx_ind(smx_env.in_buf,smx_env.nb_received_data_bytes)` gets called to forward incoming payload from the Bluetooth Low Energy client to the PN_TASK.

In the opposite direction the PN_TASK calls function `smx_send_apdu_to_phone_req(param->data,param->len)` in file smx.c. If necessary, the data will be split automatically into several Bluetooth Low Energy frames and sent as Bluetooth Low Energy notifications to the Bluetooth Low Energy client.

After a Bluetooth Low Energy disconnect sequence is completed (trigger via Bluetooth Low Energy client or smartlock) the PN_TASK receives a PN_BLE_DISC_IND message and the handler `pn_ble_disc_ind_handler()` gets called. This handler checks (`reset_after_ble_disc_cmpl`) whether a system reset is necessary (e.g. new device name).

Since there are situations where the Bluetooth Low Energy connection is closed (eg Bluetooth Low Energy Client disconnects) but PN activity is still not finished (eg lock open/close ongoing), Bluetooth Low Energy advertising may not automatically be started after a Bluetooth Low Energy disconnect complete event. Therefore the event handler for `GAP_DISCON_CMP_EVT` in file usr_design.c and function `app_task_msg_hdl()` will check whether advertising is allowed by invoking `pn_adv_allowed()`.

In this situation advertising has to be started manually after PN has finished its operation. For the manual advertising start PN_TASK will call `app_start_adv()` in file usr_design.c.

11.4 Bluetooth Low Energy Service definition

The file smx.h contains the definition for the UUIDs used in the Smart Lock application:

`SMX_SVC_UUID` defines the service UUID. This UUID is also advertised. The smartphone app is looking for devices advertising this service.

`SMX_WR_CHAR_UUID` defines the characteristic to which the Bluetooth Low Energy client will write to
`SMX_RD_CHAR_UUID` defines the characteristic which will transfer data from QN to the Bluetooth Low Energy client using Bluetooth Low Energy notifications. Therefore, the Bluetooth Low Energy client has to subscribe at first to receive notifications.

11.5 Smart Lock\Smart Lock .c

The file contains the main state machine for UART-to- Bluetooth Low Energy communication. As described there are two main states:

1) power-on-handshake
2) data exchange

11.6 Power-On-Handshake

The sequence begins after Bluetooth Low Energy receives new connection in the *app_qpps_cfg_indntf_ind_handler* function which calls *Smart Lock _power_on_pn* to set up the UART callback to wait for ***'P'*** command. When something is received on the UART the *Smart Lock _recived_pn_power_on* is called to check if the command is correct and in this case the current time is send back (*Smart Lock _send_rtc_time*) and the Smart Lock enters state 2) – waiting for command - by setting up the correct callback to receive data on UART
– *Smart Lock _uart_wait_for_cmd.*

If ***'P'*** command is not received and a time-out happens the QN enters the *sleep/advertise* cycle again.

11.7 Data exchange

11.7.1.1 Send data

Data exchange state is entered by calling *Smart Lock _uart_wait_for_cmd* to set up the correct callbacks on UART to wait for command from PN. When the command header (first 3 bytes) is received by the QN the function *Smart Lock _uart_recive_cmd_header* is called and the command is checked if it is valid.

Valid commands are:

- '**C**' – send data to the Bluetooth Low Energy client
- '**T**' – set time of RTC

If a valid command is received the QN proceeds to read the *<len>* bytes of data from the *UART* by registering a UART receive *callback Smart Lock _uart_recive_cmd*.

When the whole command is received the *Smart Lock _uart_recive_cmd* function sends the data to the Bluetooth Low Energy client and sets up a time-out on Bluetooth Low Energy if the client does not respond in case of a ***'C'*** command.

The ***'T'*** command sets the RTC to the time sent by PN – no Bluetooth Low Energy communication is performed at this time.

11.7.1.2 Receive data

Data is received from the Bluetooth Low Energy client in the *app_qpps_data_ind_handler* function. Which uses *Smart Lock _uart_transmit_add_data* to store the fragmented pieces of data to the UART RX buffer and *Smart Lock _uart_transmit_reply* to send the data to PN after the entire package was received.

12. Structure of the PN Firmware

PN firmware is split into two tasks:

1) CLIF – RFID task (implemented in smart_lock_clif.c)
2) HIF – UART task (implemented in smart_lock_hif.c)

The tasks are created in the **appMain** function in src/smart_lock.c.
Common code for retrieving the token and door opening is *smart_lock_token.c*.

12.1 Token retrieval

Token retrieval is done by *smart_lock_retrive_token* function from *smart_lock_token.c*. The function uses *phalMfdf* functions of the *NxpRdLib* to retrieve the token and is independent of the underlying implementation of *PAL* and *HAL* layers.

Before *phalMfdf* functions can be used one must instantiate and configure the *NxpRdLib* stack. This action is performed in *smart_lock_init_mfdf* function, which takes pointer to *PAL* and *HAL* layers.

In case you would like to change the card type from *MIFARE DESFire* to some other card from *MIFARE family* these two functions (*smart_lock_init_mfdf* and *smart_lock_retrive_token*) should be replaced.

After the token is retrieved it is passed to *smart_lock_process_token* function which in turn validates its content and perform the correct action (error/door opening).

12.2 Task CLIF

Task CLIF is implemented in the *smart_lock_task_clif.c.* It uses standard *NxpRdLib* to initialize the RFID reader chip and set up the correct *NxpRdLib* stack.

The task uses a timer to schedule itself. When timer lapses it initializes the discovery loop to search and activate *ISO14443 Type A* cards (with *smart_lock_clif_init_discovery_loop*) and run the discovery loop. If valid target gets activated the correct *PAL* needs to be initialize and then we can perform the token retrieval.

The poll loop is performed a finite number of times. Number of tries and time in-between can be set in *smart_lock_cfg.h* by setting:

- *SMART_LOCK_CFG_TIMER_WAKE_UP_COUNT – how many times we should poll*
- *SMART_LOCK_CFG_TIMER_WAKE_UP –time in-between polls*

4. Task CLIF NxpRdLib component stack for activation:

<table>
<tr><td colspan="2">phhalHw_Nfc_Ic_DataParams_t</td></tr>
<tr><td rowspan="2">phacDiscLoop_Sw_DataParams_t</td><td>phpalI14443p3a_Sw_DataParams_t</td></tr>
<tr><td>phpalI14443p4a_Sw_DataParams_t
phpalI14443p4_Sw_DataParams_t</td></tr>
</table>

5. Task CLIF NxpRdLib component stack for communication:

phhalHw_Nfc_Ic_DataParams_t	
phpalI14443p4_Sw_DataParams_t	
phpalMifare_Sw_DataParams_t	
phalMfdf_Sw_DataParams_t	phKeyStore_Sw_DataParams_t
	phCryptoSym_Sw_DataParams_t
	phCryptoRng_Sw_DataParams_t

12.3 Task HIF

Task HIF is implemented in *smart_lock_task_hif.c*. After the task starts it initializes custom BAL and HAL layers that implement *QN* ↔ *PN* communication protocol. Next it tries to perform the handshake between the QN and PN. When the handshake is successfully performed it initialize the custom *PAL* layer, which is used to initialize *NxpRdLib* software stack and then perform the token retrieval.

12.3.1 Task HIF NxpRdLib component stack:

phbalReg_Smart Lock _DataParams_t	
phhalHw_Smart Lock _DataParams_t	
phpalMifare_Smart Lock _DataParams_t	
phalMfdf_Sw_DataParams_t	phKeyStore_Sw_DataParams_t
	phCryptoSym_Sw_DataParams_t
	phCryptoRng_Sw_DataParams_t

12.3.2 phbalReg_Smart Lock _DataParams_t

phbalReg_Smart Lock _DataParams_t component can be found at *comps/phbalReg/src/Smart Lock Serial/phbalReg_Smart Lock Serial.c*. The file contains the low level implementation of the communication protocol.

12.3.3 phhalHw_Smart Lock _DataParams_t

phhalHw_Smart Lock _DataParams_t component can be found at *comps/phhalHw/src/Smart Lock /phhalHw_Smart Lock .c* and implements the phhalHw_Smart Lock _Exchange function with buffering and manipulation of the data for upper layers.

12.3.4 phpalMifare_Smart Lock _DataParams_t

phpalMifare_Smart Lock _DataParams_t component can be found at comps/phpalMifare/src/Smart Lock /phpalMifare_Smart Lock .c and is a wrapper used by the upper layers.

12.4 Configuration options for PN firmware

PN firmware can be configured in different ways by means of defines in *smart_lock_cfg.h* in the following ways:

- Cryptographic algorithm configuration options:
  - SMART_LOCK_CFG_AUTH_2KTDES
  - SMART_LOCK_CFG_AUTH_3KTDES
  - SMART_LOCK_CFG_AUTH_AES
- Key diversification: SMART_LOCK_CFG_AUTH_DIVERSIFY

13. Firmware flashing

13.1 Requirements

For developing PN7462AU and QN9021 firmware and customer applications all components listed in the below table are required.

Table 1. Development environment

Item	Version	Purpose
OM27462NBR Smart Lock KIT	1.2 or higher	Demonstrator board
OM13054 LPC Link 2	1.0	Standalone debug adaptor
LPCXpresso IDE	8.0.0 or higher	Development IDE for PN
LPCXpresso PN7462AU plugin	com.nxp.pn7xxxxx.update -8.0.0-SNAPSHOT-150	Add PN7462AU reader to the LPCXpresso
ARM Keil MDK	5.21 or higher	Development IDE for QN
QN9020QBlue Software tools	1.3.9 or higher	Plugins for Keil and optional for flashing via serial port

For an example of how to use LPCXpresso to update the PN7462 firmware refer to the following document: UM10883 – PN7462 family Quick Start Guide Development Kit.

13.2 LPC-Link2 connection

On the SmartLock board there are two SWD connections for upgrading both the PN7462 and the QN9021 firmware. The SWD/JTag connectors are marked with PN SWD and Bluetooth Low Energy SWD. The LPC-Link2 board can be used to flash the PN7462AU and QN9021. In Fig 18 the pins of the flashing cable are marked according to the pins on the SmartLock.

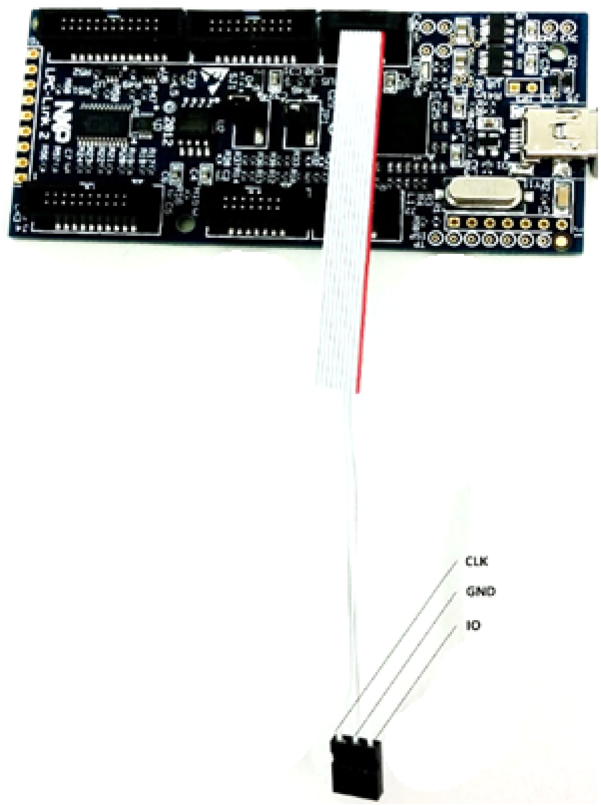

Fig 18. LPC Link2 SWD connection cable

13.3 Flash new firmware on PN

- Install LPCXpresso IDE and activate the free edition under Help→Activate→Activate (Free Edition)
- Install LPCXpresso PN7462AU plugin
- Import the Smartlock PN Project via File→Import→Existing Project into Workspace
- Build the Project via Project→Build All. Additional compiler may need to be installed based on your environment.
- Connect the LPC Link2 and the Smartlock **PN** SWD Pins with the flashing cable to the correct pins as shown in Fig 18.
- Make sure the JP2 jumper on the LPC-Link 2 is set and the Battery is connected to the Smartlock.
- Click the Program Flash icon to flash the PN software. If there is a CoreException shown when connecting with the LPC-Ling 2 then ignore it.
- For further questions have a look into UM10883 – PN7462 family Quick Start Guide Development Kit..

13.4 Flash new firmware on QN

13.4.1 Flash with LPC Link 2

- Install ARM Keil MDK
- Install the QN9020QBlue Software tools
- Run the QN9020DevDBforIDE tool to install the Keil QN Device Database
- Open the QN Project in Keil via Project->Open Project
- Build the Project via Project→Build Target
- Connect the LPC Link2 and the Smartlock **QN** SWD Pins with the flashing cable to the correct pins as shown in Fig 18.
- Make sure the JP2 jumper on the LPC-Link 2 is set and the Battery is connected to the Smartlock.
- Open LPCXpresse IDE and click the flash icon to ensure that the LPC-Link2 is operating in the correct mode.
- Now switch back to Keil and click Project->Options for Target ‘smartlock’. A new Window should open.
- Under Device the previous installed NXP Bluetooth Low Energy CLU Database as shown in with the flashing cable to the correct pins as shown in Fig 19 has to be selected.

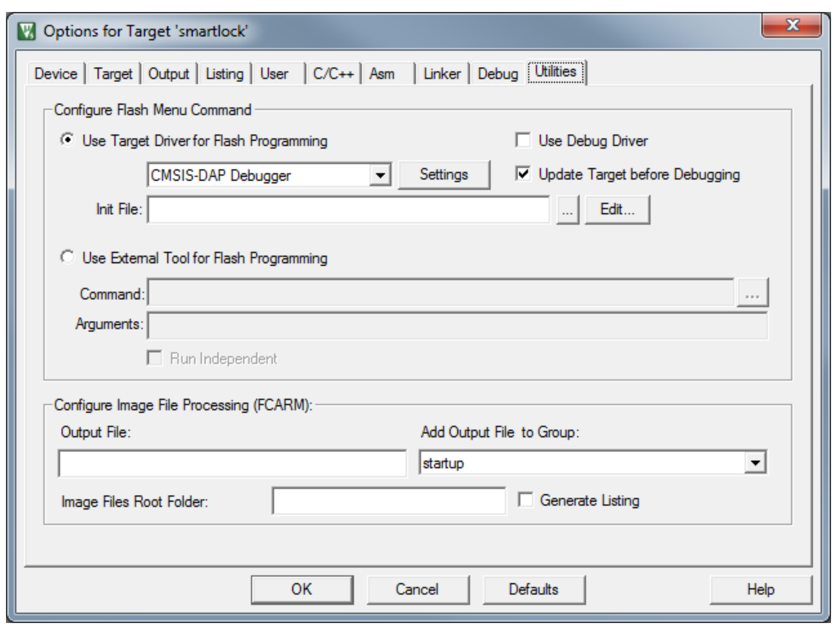

Fig 19. Option for Target in Keil

In the Debug settings under “Use” select CMSIS-DAP Debugger and click on Settings.

A new Window should open and show the SW Device in the List. Device Name should be “ARM CoreSight SW-DP” as in Fig 20.

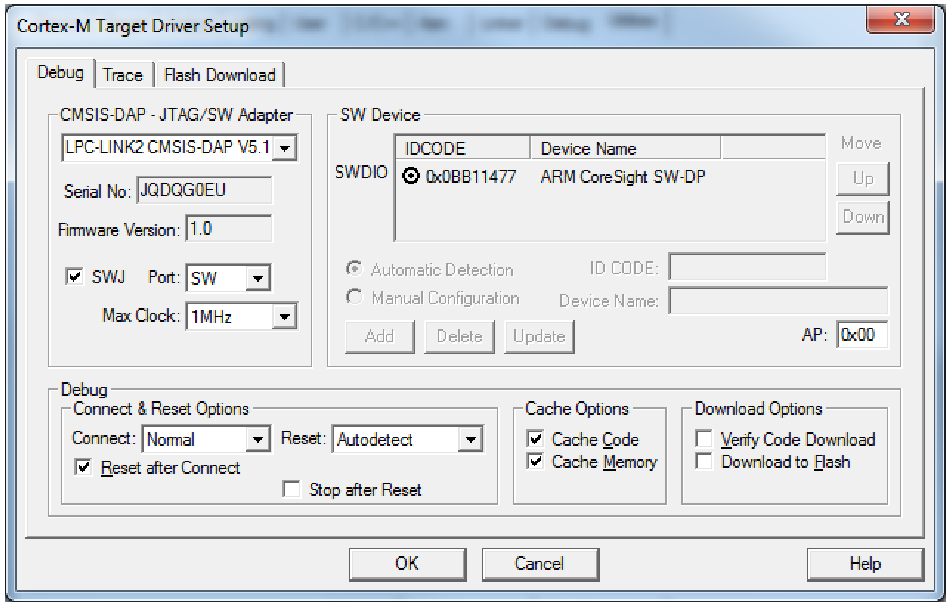

Fig 20. Configure Debug Adapter in Keil

Also ensure that CMSIS-DAP Debugger is selected as Target Driver in the Utilities Settings and the Settings are matching with the Fig 20.

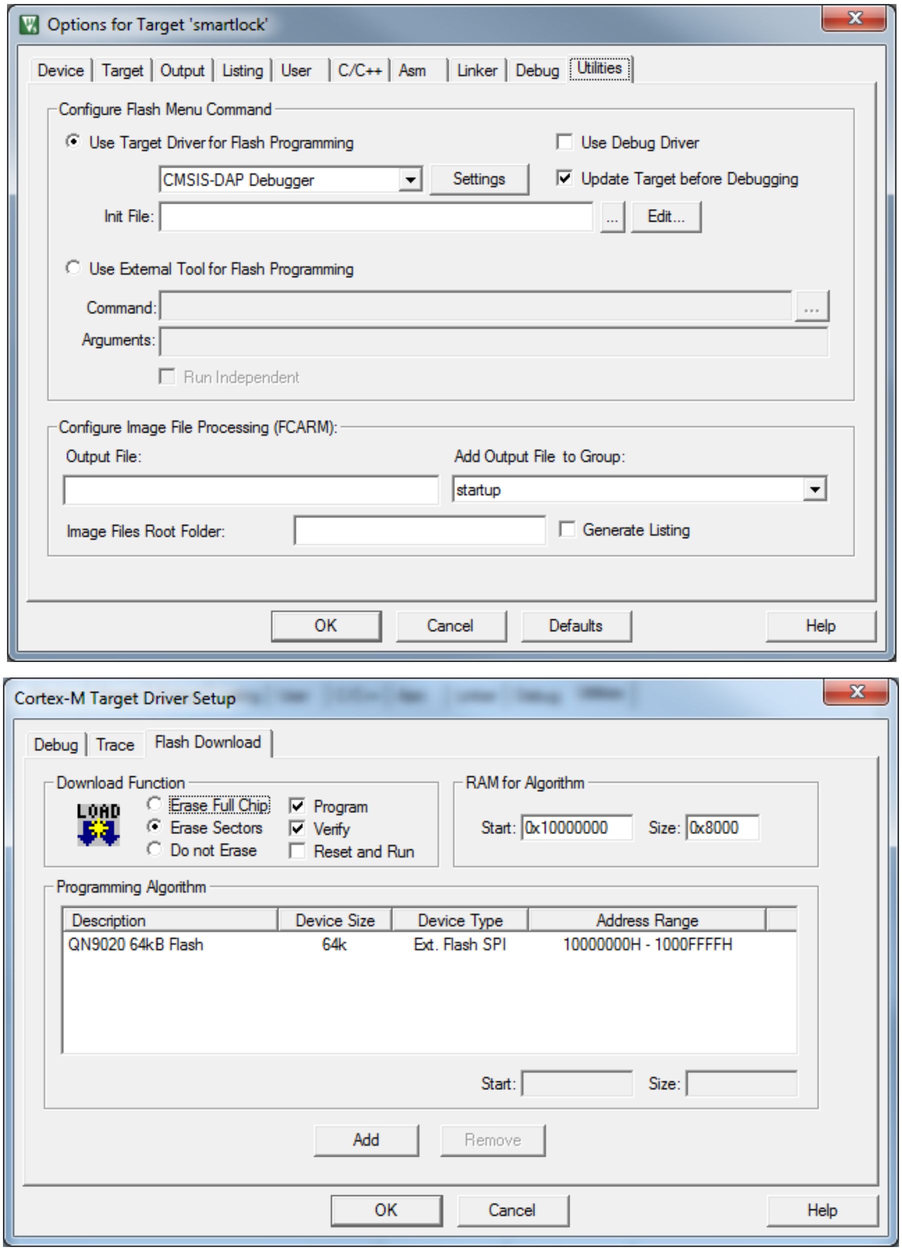

Fig 21. Configure Utilities in Keil

Save the settings with OK. Now you should be able to flash the QN via Flash→Download

13.4.2 Flash with serial

Optional it is also possible to flash the QN via a serial port. Remove the PN Core module and connect your serial port with the QN as shown in the Fig 22.

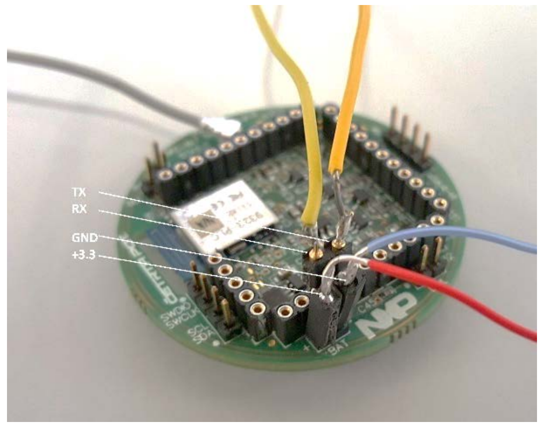

Fig 22. Connections for serial flashing

Open the QBlue ISP Studio and check that the settings matching the Fig 23.

Click the Start button to start the flashing procedure. The tool waits until the QN goes in a reset to allow flashing. Simply remove and connect the supply to trigger a reset.

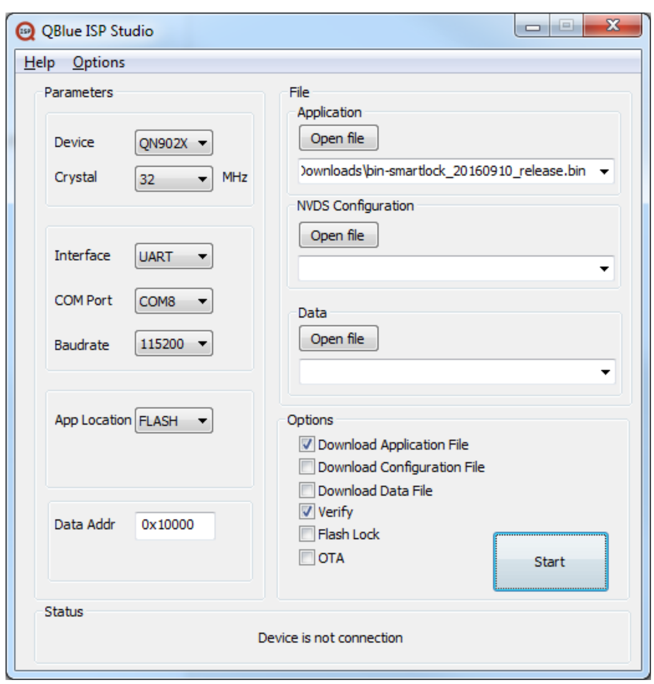

Fig 23. Options for serial flashing

14. Hospitality Access Token

14.1 Overview

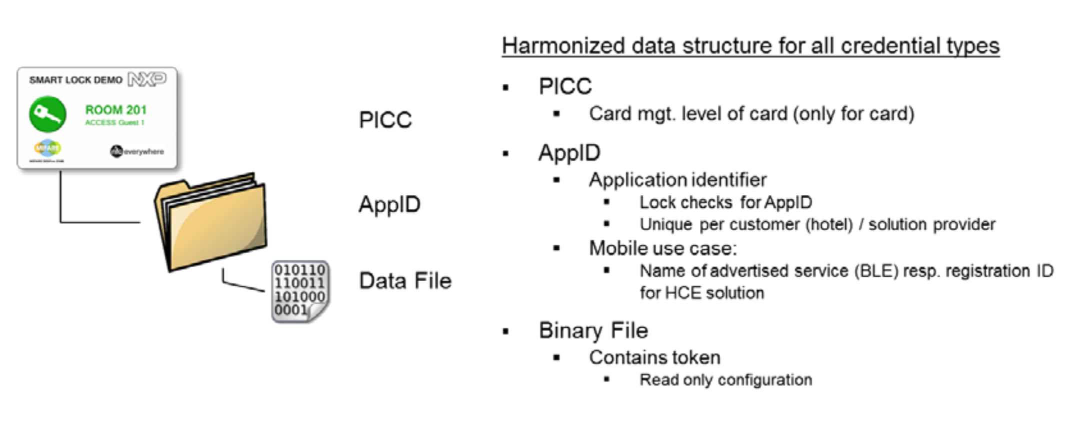

Fig 24. Harmonized data structure

14.2 MIFARE DESFire card configuration

14.2.1 Application configuration:

- Application Master Key Setting:
  0x0B (Change App Key, GetFile enabled, master key changeable)

14.2.2 Application Key settings:

- APP ID: 010203
- Crypto Mode: AES
- DF-Name: 07060504030201 (HEX)
- 3 keys
  - Key 0 app Master key
    - **Diversified Key needed**
    - Non-diversified Key: 00000000000000000000000000000001
  - Key 1 Read key for File id 00
    - **Diversified key needed**
    - Non-diversified Key: 00000000000000000000000000000000
  - Key 2 No rights only used for GetCardUID command
    - **No diversification**
    - Key: 00000000000000000000000000000000

14.2.3 Symmetric key diversification details see application note AN10922

14.2.3.1 Input for diversified Key

- 0x01 // *fixed start constant not needed in reader library*
- 0xXX 0xXX 0xXX 0xXX 0xXX 0xXX 0xXX // XX = 7 byte Card UID
- 0x01, 0x02, 0x03, *// 3 byte AID fixed*
- 'S', 'M', 'A', 'R', 'T', 'L', 'O', 'C', 'K' *// 9 byte ASCII to HEX fixed*
- Together fixed start constant + 19 byte diversification input

14.2.4 Standard Data File settings

- File ID: 0x00
- Supported Communication Modes
  - Fully Encrypted
- Access rights: 0x1000
  - Key #0 for write, R&W, Change Access rights
  - Key #1 for read
- File size: 0x000080 (Size: 128 bytes)

14.3 Standard Data file content

14.3.1 Token format

Tag	Length	Value
0xAB	**0x68**	**Total data length**
0xD0	0x10	2-byte label **[==0x00 00]** 1-byte token version **[0x01]** 4-byte issuer ID **[0xCA FF EE 00 ==AID]** 4-byte time stamp **from** - Unix timestamp in seconds 4-byte Unique sequence number **[Random No.]** 1-byte ECC-P26 key version **[0x01]**
0xD1	0x04	4-byte time stamp **until** - Unix timestamp in seconds
0xD3	0x0C	4-byte Hotel ID **[0x47 52 41 5A]** 4-byte Room ID **[0x30 32 30 31 “0201” or 0x30 32 30 32 “202”]** 4-byte RevToken for revoking of previous tokens **[0x00 00 00 01]**
0xD5	0x40	ECC-P256 signature

14.3.2 Token verification process

1. Validate Rev_{Token} → chose appropriate processing
   - If revision is below current Rev in lock → Token isn´t valid anymore
   - If revision is equal to current Rev in lock → no actions
   - If revision is higher than current Rev in lock → increase Lock Rev
2. Validate Time stamps
3. Validate HotelID/RoomID → correct lock number?
4. Validate Rev_{Revoke} → can be used for offline blocking of “lost card”
5. Validate Rev_{ECCkey} → chose appropriate ECC public key
6. Verify ECC signature
7. Read Unique sequence number [Random No.] and store in log-file
   → which token has been used

15. Power consumption figures

The above table shows different power consumption figures based on Bluetooth Low

Low Power Mode	Adv Interval	Transmit Power	Current Consumption
No (Sleep possible)	no advertising	n/a	19 µA
Yes	no advertising	n/a	37 µA
Yes	600 ms	-14 dBm	~ 93 µA
Yes	600 ms	0 dBm	~ 107 µA
Yes	300 ms	-14 dBm	~ 155 µA
Yes	300 ms	0 dBM	~ 170 µA

Energy advertising intervals. The current QN firmware implementation is configured for 600ms -14 dBm.

at around 93 µA while advertising.

16. Power optimization considerations

At present the Bluetooth Low Energy is operating in polling mode and advertises every 600ms. In order to further reduce the average power consumption, one might consider changing the hardware and software design in a way that the Bluetooth Low Energy circuitry is also woken up by an external trigger such as the capacitive wakeup sensor. Care must be taken to ensure proper RTC (real time clock) operation of the QN9021 in this case.

Note: the KIT´s hardware and software are delivered as is. Any modifications on this hardware and/or software are not in the responsibility of NXP.

17. Download available Documents and firmware

www.nxp.com/demoboard/OM27462NBR

18. References

[1] LPCXpresso webpage

19. Legal information

19.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

19.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

19.3 Licenses

Purchase of NXP ICs with NFC technology
Purchase of an NXP Semiconductors IC that complies with one of the Near Field Communication (NFC) standards ISO/IEC 18092 and ISO/IEC 21481 does not convey an implied license under any patent right infringed by implementation of any of those standards. Purchase of NXP Semiconductors IC does not include a license to any NXP patent (or other IP right) covering combinations of those products with other products, whether hardware or software.

19.4 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

MIFARE — is a trademark of NXP B.V.

DESFire — is a trademark of NXP B.V.

Bluetooth — The Bluetooth word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

20. Contents

Date of release: 11 October 2018

407611

Document identifier: UM11038

Date of release: 11 October 2018

407611

Document identifier: UM11038