AN13925

PN76 EEPROM configuration handling

Rev. 1.0 — 18 August 2023 | Application note

Document information

Information	Content
Keywords	PN76, open controller, EEPROM, configuration handling, settings, PN76 family
Abstract	This document will show how to dump, load and handle EEPROM settings of the PN76 family. By using the NXP NFC Cockpit and PN76 SDK examples.

Revision history

Revision history

Rev	Date	Description
v.1.0	20230818	Initial version

1 Introduction

This document explains various ways how the EEPROM values can be modified, stored, and copied from and to the PN7642. What settings are correct, how to trim and correctly configure the PN76 is not part of this document. There are various dedicated application notes available how to find the best settings for your individual setup.

Even though it is referred to as EEPROM, it is located in the flash memory of the PN7642 controller. Only accessible via the EEPROM HAL (Hardware Abstraction Layer) APIs from the user.

The settings stored in this configuration EEPROM are used for a basic configuration of the controller, which does not change frequently. Typically it is performed once during the configuration of a product. For frequently changing configurations, the registers have to be used, which do not keep their value during power off or reset.

Note: Writing to the EEPROM has to be performed with Read-Modify-Write for all memory addresses that contain RFU bits. While RFU bits are not to be modified!

The detailed description of the registers and EEPROM values is available in the [1] Datasheet and [2] PN76 API Documentation.

1.1 Environment

The environment used and referred to within this application note:

- MCUXpresso v11.7.1 (or later)
- MCUXpresso PN7642 SDK 2.12.3
- PNEV7642 Rev.B FW v1.0
- NFC Cockpit v7.2.0
- SEGGER J-Link

2 Use cases

At one point in developing a product the PN76 configuration must be adapted to its environment (PCB, antenna, power supply, enclosure, …). How to handle the EEPROM configuration the easiest way depends on the users system architecture, environment and goal.

This document explains the most common use-cases and approaches.

2.1 Developing phase

In the developing phase of a product, the configuration of the PN76 must be adapted to its unique environment. Such as housing, PCB, surrounding components, antenna, power budget and so on. This adaptation of settings is crucial for a good performance and operating within the PN76 limits.

This document does not describe the process of finding the correct configuration of the PN76. There are dedicated Application Notes, User Manuals, and Guides for different functions (DPC, ULPCD, Antenna setup, …) existing.

The goal in this phase is to find the best EEPROM configurations for your unique environment. In this process, the EEPROM is read/written multiple times and also shared with colleagues and other parties.

The easiest and fastest way is using the *NNC* (NXP NFC Cockpit). With the *NNC*, writing and reading single values and loading or dumping the whole EEPROM is easily done.

See Section 4 "NXP NFC Cockpit" for further information on *NNC*.

2.2 Production phase

A EEPROM configuration for the PN76 has been found and this configuration must be replicated on every PN76 chip.

The EEPROM is not accessible from the outside via a debugger or programming interface. Only via the system service APIs, called from the user application space. The EEPROM must be updated from an application.

Two options to do so are:

Table 1. EEPROM update options

	Option	Pro	Con
1	Integration of the EEPROM update into the user application.	• Only a single application to be developed. • Only one programming cycle in production.	• Uses flash space to hold the EEPROM configuration in the user application. • Checking every time if already updated.
2	Dedicated application for configuration	• The user application can be developed independent. • No additional flash space occupied with EEPROM configuration.	• A dedicated application for EEPROM update must be developed *(1)*. • Two programming cycles in production.

(1): NXP provides an example how to update the EEPROM from an application as part of its PN76 SDK. The example is described in more detail in Section 5 "MCUXpresso Example".

Those two options are described in more detail in the next chapters. How the final implementation can look like is user individual and depends on each unique system design and software architecture. Following options are only two of many and can be used as inspiration.

2.2.1 Integration in user application

The biggest benefit of this approach would be that save programming cycles in production. As a dedicated update application (Section 2.2.2 "Dedicated application") has to be programmed first, executed and after the successful execution overwritten (reprogrammed) with the user application.

On the other hand is the integration of the EEPROM update into the users application making the app itself more complex. At one point, there must be a check if the chip has been already updated or not. This logic has to be solid, as a EEPROM update (write) at every boot-up means more write-cycles and a premature flash wear-out.

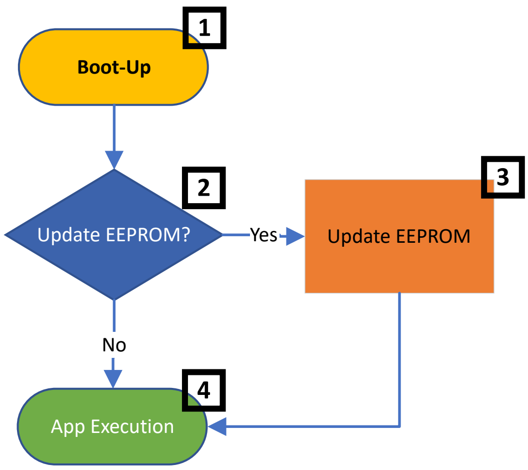


Figure 1. Integration in user application flowchart

[1] Boot-Up

The controller boots up and basic initializations are done.

[2] Update EEPROM?

Before starting the actual business logic of the user application, there must be a check if the EEPROM configuration has been updated already. This can be done, for example, by checking a value stored in the EEPROM_USER_AREA or at an empty flash page.

If an EEPROM update is necessary progress to step [3].

The EEPROM update only must be done once, as the EEPROM is persistent memory and will not be overwritten by the firmware.

[3] Update EEPROM

The EEPROM is updated with the provided values. This can be the whole area or only a small portion. NXP provides an example how it is done. The final integration of this code is up to the user.

See Section 5 "MCUXpresso Example" for more details about the example.

[4] App Execution

The users application is executed. At this point, the EEPROM must have its final content.

2.2.2 Dedicated application

Building a dedicated application for the EEPROM update, and potentially firmware update as well, has the benefit that the user application is independent. In the user application, there is also no check update necessity needed.

The drawback is that one extra programming cycle is necessary. To flash first the dedicated update application, execute it, and after successful execution, flash the chip again with the final user application.

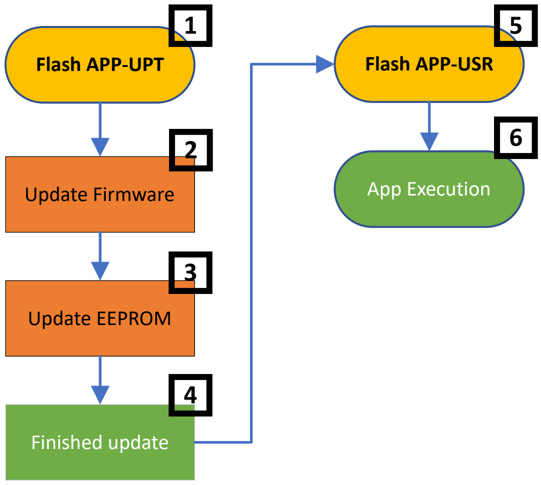


Figure 2. Dedicated application flow

[1] Flash APP-UPT

Program the PN76 with the application dedicated to configure and update the PN76.

[2] Update firmware

If the firmware must be updated, it updates the PN76s firmware.

[3] Update EEPROM

Updates the EEPROM of the PN76 with the provided values.

[4] Finish update

As the firmware and/or EEPROM update is done in application, an indicator to signal that the chip is finished with the update process should be considered. To avoid flashing the user application before the update is finished.

This can be a signal on a GPIO, an output on the HIF (Host Interface), or any other signal. This depends on the customers programming environment. If nothing is feasible a timeout might be used. But there is no indication of success or fail.

[5] Flash APP-USR

The second programming cycle. Here the users final application is programmed.

[6] App Execution

A sanity check if the user application is running properly, is always recommended.

3 EEPROM

The PN76 has three EEPROM locations accessible by the user. The EEPROM is only accessible via the dedicated Eeprom APIs. Offset based addressing is used instead of the real physical address. For every area, the offset starts at 0.

For a detailed description of the EEPROM values, see the PN76 product data sheet.

Table 2. EEPROM areas

Area	Symbol	Size	Start - End	Usage
User Area	EEPROM_USER_AREA	128 bytes	0x000 - 0x080	The first 128 bytes are reserved for NXP use and its content is described in the data sheet.
		894 bytes	0x081 - 0x3FF	The remaining 894 bytes are open for the customer to use for data storage.
Secure Lib	EEPROM_SECURE_LIB	2048 bytes	0x000 - 0x7FF	Chip settings accessible and configurable by the user. Described in data sheet.
IC Config	EEPROM_IC_CONFIG	1024 bytes	0x000 - 0x3FF	IC configurations, not accessible for the user.

3.1 EEPROM APIs

The EEPROM of the PN76 can be accessed via the system services layer of the PN76.

Functions

PN76_Status_t **PN76_WriteEeprom** (**uint8_t** *pDataToWrite, **uint16_t** wE2PromAddress, **uint16_t** wDataLength, **PN76_EEPROM_Config_t** eConfig)
Write values sequentially to E2PROM. More...

PN76_Status_t **PN76_WriteEeprom_ProtocolUserArea** (**uint8_t** *pDataToWrite, **uint16_t** wE2PromAddress, **uint16_t** wDataLength)
Write values sequentially to E2PROM UserArea and Protocol Area. More...

PN76_Status_t **PN76_ReadEeprom** (**uint8_t** *pReadData, **uint16_t** wE2PromAddress, **uint16_t** wDataLength, **PN76_EEPROM_Config_t** eConfig)
Read values sequentially to E2PROM. More...

PN76_Status_t **PN76_ReadEeprom_ProtocolUserArea** (**uint8_t** *pReadData, **uint16_t** wE2PromAddress, **uint16_t** wDataLength)
Read values sequentially from E2PROM Protocol and User Area. More...

PN76_Status_t **PN76_Eeprom_CheckIntegrity** (**PN76_EEPROM_Config_t** eRegion)
Write values sequentially to E2PROM. More...

Figure 3. EEPROM functions

For a detailed description of the functions, consolidate the PN7642 NFC Controller User API Documentation.

4 NXP NFC Cockpit

The NXP NFC Cockpit (NNC) is a powerful GUI tool to configure and adapt IC settings. It is recommended to use the *NNC* to adjust your antenna settings or other EEPROM values.

This chapter explains what EEPROM operations you can do with *NNC*. It is assumed the reader of this document is familiar with the *NNC* and how to start, and connect, it to the PN76.

4.1 Read/write EEPROM

The NFC Cockpit allows direct access to the EEPROM. When a reader is connected the area "EEPROM Single-Byte Access" comes available. As mentioned in Table 2 "EEPROM areas" the user can access the 'SECURE_LIB_CONFIG' or 'USER_AREA'.

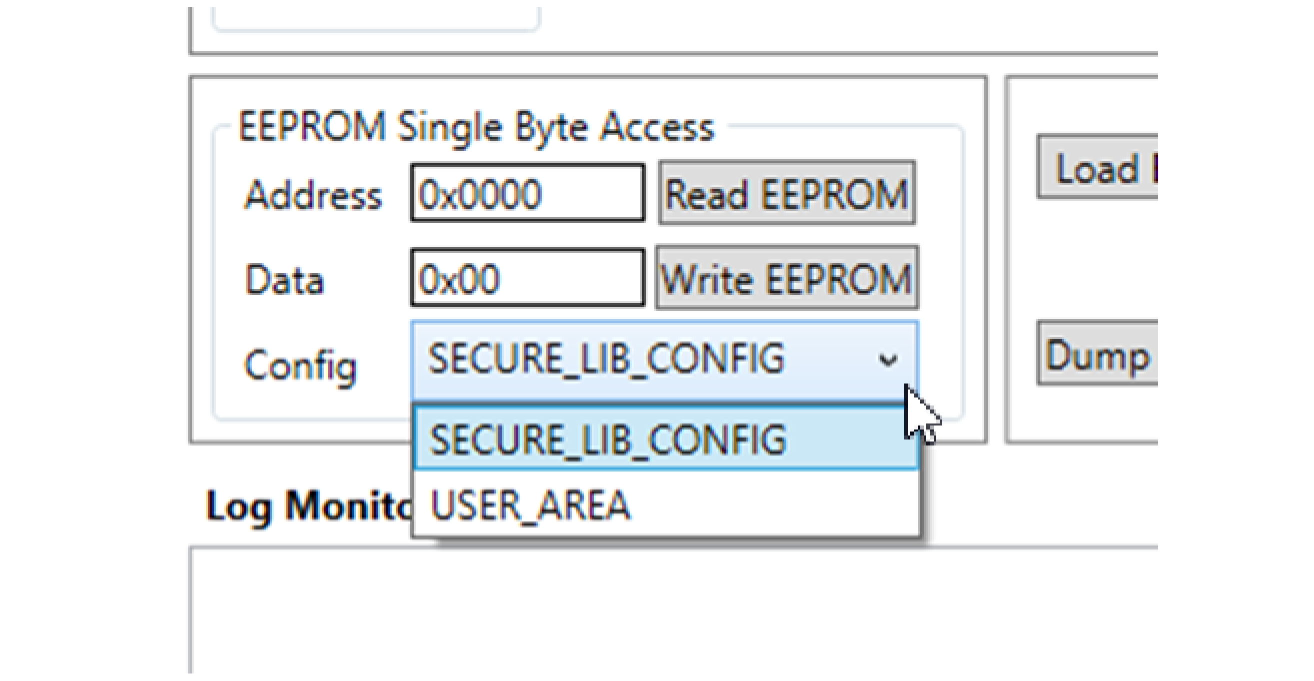


Figure 4. NNC EEPROM Single-Byte Access

Explanation:

- *Address*: The address (offset) where do read/write.
- *Data*: Value that is either read or value to be written.
- *Config*: The EEPROM map from, or where to, read and write.
- '*Read EEPROM*' button: Reads from the address in the *Address* field and EEPROM map chose in *Config*. The read value is shown in the *Data* field.
- '*Write EEPROM*' button: Writes the value from the *Data* field to the address in the *Address* field.

Read EEPROM Example:

Reading from address '0x01' of config 'SECURE_LIB_CONFIG', this represents the 'DCDC_CONFIG' value.

The read value is 0x31, as shown in the *Data* field and also in the Log Monitor.

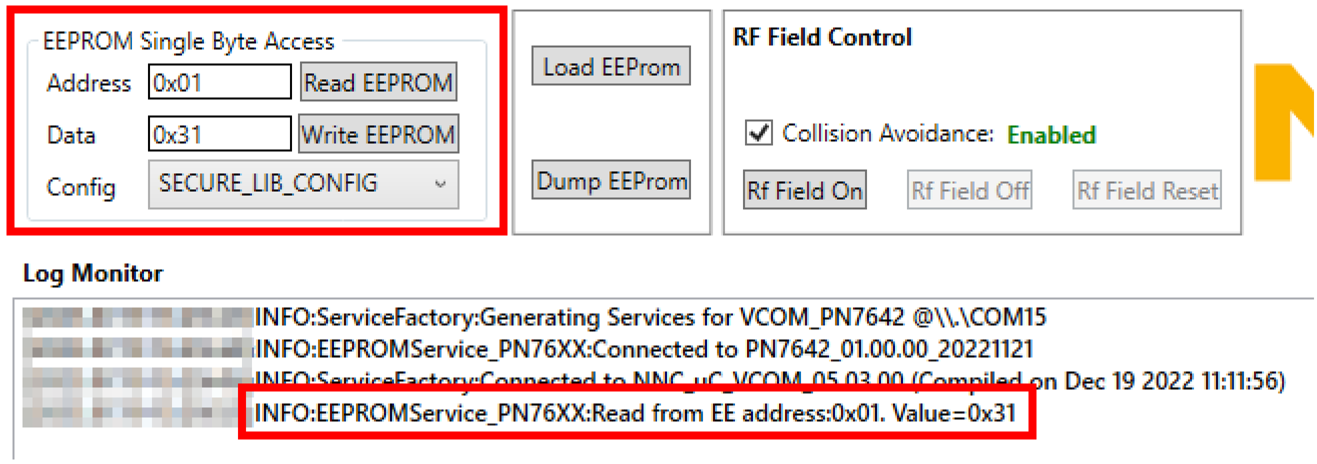


Figure 5. NNC Read EEPROM

4.2 Load/Dump EEPROM

With the NNC, it is easy to load EEPROM files or Dump them.

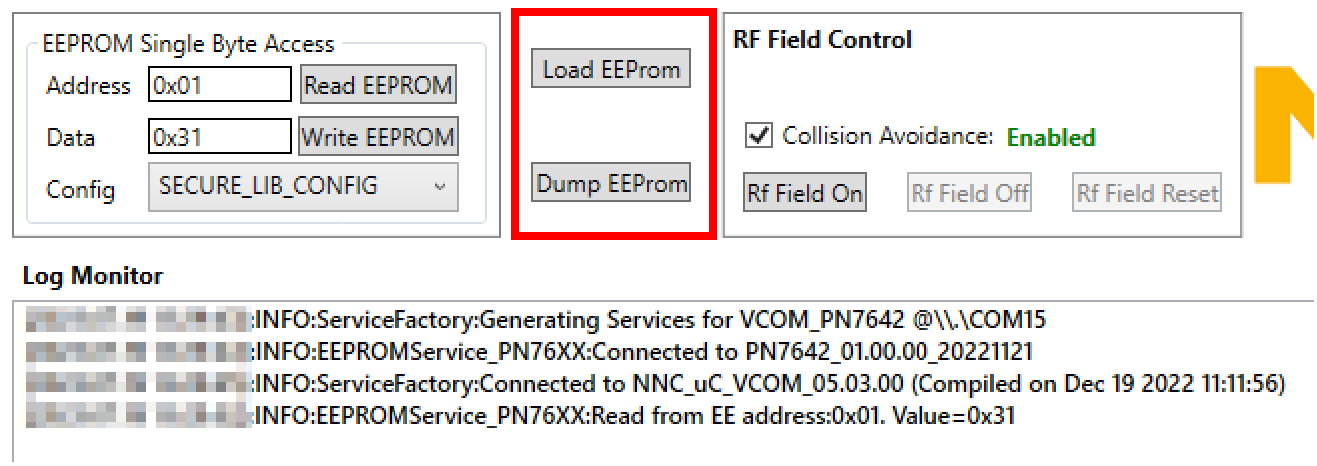


Figure 6. NNC Load/Dump EEPROM

Load EEPRrom

By clicking *Load EEProm*, a navigation window opens and a EEPROM file (*.xml or *.c) can be chosen. This EEPROM file and all its value are written from NFC Cockpit to the PN76 EEPROM regions.

This is helpful to replicate one configuration to another chip. You can share the EEPROM file and easily load the same file on multiple chips.

Use-cases:

- Replicate the same configuration on multiple chips while developing to have the same base configuration.
- Share EEPROM files with other users (for example, NXP for debugging).
- Switch between different EEPROM configurations to evaluate changes of behavior.

A default EEPROM file can be found in the NFC Cockpit installation folder → cfg → PN7642 → Default.

Dump EEProm

The *Dump EEProm* button dumps the whole EEPROM configuration of the chip to a file (*.xml, *.h or *.c). This dump can further be used in the *Load EEProm* of the NFC Cockpit or to prepare your *Chip Configuration Application*.

4.3 EEPROM files

The *NNC* can load EEPROM files with the file ending *.xml and *.c. While it can dump the EEPROM configuration to files with the ending *.xml, *.h and *.c.

While the *.xml file is for humans the easiest to read and manipulate, for the PN76 use case the *.h or *.c is more suitable.

4.3.1 EEPROM file *.xml

The *.xml file and style are used for other NXP NFC chips as well. It is the easiest human readable file. But cannot only be used with the NFC Cockpit and not integrated into a user application.

Each parameter is easy to be found with its *Name* and *Offset*. And the present *Value* of this EEPROM register.

PN7642_EEPROM_Dump.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<EEPROM>
    <Region RegionName="USER_PMU" RegionAccess="RW" RegionType="DATA" Config="EEPROM_SECURE_LIB_CONFIG">
    <Parameter Name="PwrConfig" Offset="0x00" Value="0xE4" />
    <Parameter Name="DcdcConfig" Offset="0x01" Value="0x31" />
    <Parameter Name="TxldoConfig" Offset="0x02" Value="0xFFFFAEA7" />
    <Parameter Name="TxldoStartVddpa" Offset="0x06" Value="0x00" />
    <Parameter Name="TxLdoVddpaMaxRdr" Offset="0x07" Value="0x2A" />
    <Parameter Name="TxLdoVddpaMaxCard" Offset="0x08" Value="0x2A" />
    <Parameter Name="BoostDefaultVoltage" Offset="0x09" Value="0x1D" />
    </Region>

    <Region RegionName="CLKGEN" RegionAccess="RW" RegionType="DATA" Config="EEPROM_SECURE_LIB_CONFIG">
        <Parameter Name="XtalConfig" Offset="0x0F" Value="0x00" />
        <Parameter Name="XtalTimeOut" Offset="0x10" Value="0xFF" />
    </Region>

    <Region RegionName="RF_CLOCK_CFG" RegionAccess="RW" RegionType="DATA" Config="EEPROM_SECURE_LIB_CONFIG">
        <Parameter Name="PLLClkInputFrq" Offset="0x11" Value="0x08" />
        <Parameter Name="XtalCheckDelay" Offset="0x12" Value="0xF6" />
    </Region>

    <Region RegionName="RM_TECHNO_TX_SHAPING" RegionAccess="RW" RegionType="DATA" Config="EEPROM_SECURE_LIB_CONFIG">
        <Parameter Name="ResidualAmplitudeLevel_A106" Offset="0x14" Value="0x00" />
        <Parameter Name="EdgeType_A106" Offset="0x15" Value="0x33" />
        <Parameter Name="EdgeStyleConfiguration_A106" Offset="0x16" Value="0x64" />
        <Parameter Name="EdgeLength_A106" Offset="0x17" Value="0x10" />
        <Parameter Name="ResidualAmplitudeLevel_A212" Offset="0x18" Value="0x00" />
        <Parameter Name="EdgeType_A212" Offset="0x19" Value="0x33" />
        <Parameter Name="EdgeStyleConfiguration_A212" Offset="0x1A" Value="0x44" />
        <Parameter Name="EdgeLength_A212" Offset="0x1B" Value="0x10" />
        <Parameter Name="ResidualAmplitudeLevel_A424" Offset="0x1C" Value="0x00" />
        <Parameter Name="EdgeType_A424" Offset="0x1D" Value="0x33" />
        <Parameter Name="EdgeStyleConfiguration_A424" Offset="0x1E" Value="0x24" />
        <Parameter Name="EdgeLength_A424" Offset="0x1F" Value="0x10" />
        <Parameter Name="ResidualAmplitudeLevel_A848" Offset="0x20" Value="0x00" />
        <Parameter Name="EdgeType_A848" Offset="0x21" Value="0x11" />
        <Parameter Name="EdgeStyleConfiguration_A848" Offset="0x22" Value="0x18" />
        <Parameter Name="EdgeLength_A848" Offset="0x23" Value="0x10" />
        <Parameter Name="ResidualAmplitudeLevel_B106" Offset="0x24" Value="0xCA" />
        <Parameter Name="EdgeType_B106" Offset="0x25" Value="0x44" />
```

Figure 7. NNC EEPROM dump as *.xml

4.3.2 EEPROM file *.h

For the PN76, the NFC Cockpit can dump the EEPROM also in an C/C++ compliant header file. Each EEPROM value is represented as a macro with its Name, offset, length, and value.

PN7642_EEPROM_Dump.h

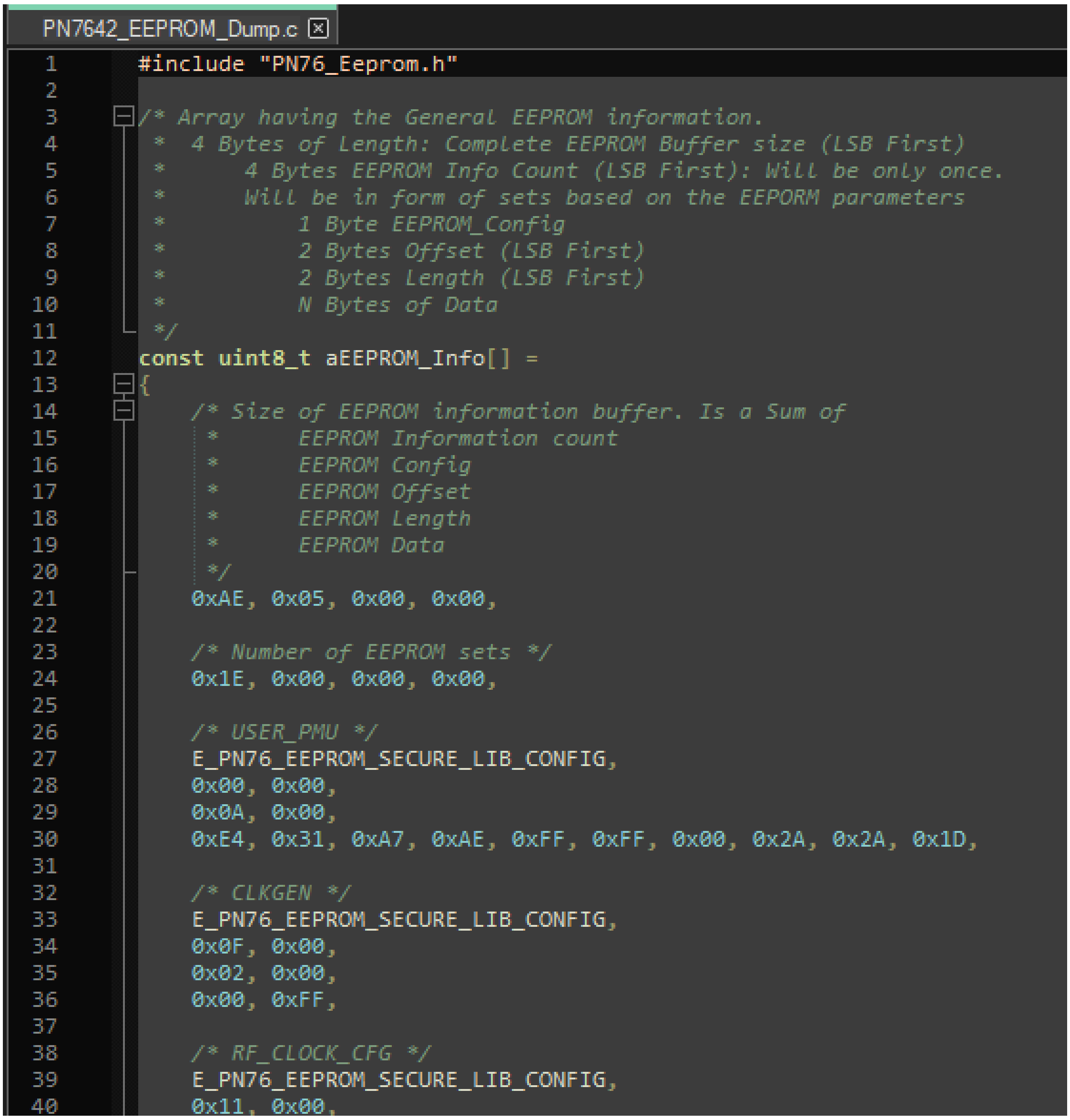
```
/* PN7642 EEPROM Dump */
#ifndef EE_DUMP_PN7642_H_INCLUDE
#define EE_DUMP_PN7642_H_INCLUDE


/* :: Region: USER_PMU, CONFIG: EEPROM_SECURE_LIB_CONFIG */
#define EE_USER_PMU_REGION_CONFIG 1
#define EE_USER_PMU_REGION_OFFSET 0x0000
#define EE_USER_PMU_REGION_LENGTH 10
#define EE_USER_PMU_VALUES    {\
    0xE4, 0x31, 0xA7, 0xAE,     0xFF, 0xFF, 0x00, 0x2A,  \
    0x2A, 0x1D, \
    }

/* :: Region: CLKGEN, CONFIG: EEPROM_SECURE_LIB_CONFIG */
#define EE_CLKGEN_REGION_CONFIG 1
#define EE_CLKGEN_REGION_OFFSET 0x000F
#define EE_CLKGEN_REGION_LENGTH 2
#define EE_CLKGEN_VALUES    {\
    0x00, 0xFF, \
    }

/* :: Region: RF_CLOCK_CFG, CONFIG: EEPROM_SECURE_LIB_CONFIG */
#define EE_RF_CLOCK_CFG_REGION_CONFIG 1
#define EE_RF_CLOCK_CFG_REGION_OFFSET 0x0011
#define EE_RF_CLOCK_CFG_REGION_LENGTH 2
#define EE_RF_CLOCK_CFG_VALUES    {\
    0x08, 0xF6, \
    }

/* :: Region: RM_TECHNO_TX_SHAPING, CONFIG: EEPROM_SECURE_LIB_CONFIG */
#define EE_RM_TECHNO_TX_SHAPING_REGION_CONFIG 1
#define EE_RM_TECHNO_TX_SHAPING_REGION_OFFSET 0x0014
#define EE_RM_TECHNO_TX_SHAPING_REGION_LENGTH 84
#define EE_RM_TECHNO_TX_SHAPING_VALUES    {\
    0x00, 0x33, 0x64, 0x10,     0x00, 0x33, 0x44, 0x10,  \
    0x00, 0x33, 0x24, 0x10,     0x00, 0x11, 0x18, 0x10,  \
    0xCA, 0x44, 0x00, 0x10,     0xCF, 0x22, 0x66, 0x10,  \
    0xCF, 0x22, 0x55, 0x10,     0xCE, 0x22, 0x34, 0x10,  \
    0xCF, 0x22, 0x65, 0x10,     0xCE, 0x22, 0x55, 0x10,  \
```

Figure 8. NNC EEPROM dump as *.h

All information provided in this document is subject to legal disclaimers.

4.3.3 EEPROM file *.c

For the PN76, the NNC is able to dump the whole EEPROM configuration into a C/C++ compliant array within a *.c file. The PN76s MCUXpresso SDKs example "*pnev7642fama_userdata_update*" shows, how such a file can be incorporated in an application. To update the EEPROM from user application space.



PN7642_EEPROM_Dump.c

```
#include "PN76_Eeprom.h"

/* Array having the General EEPROM information.
 *  4 Bytes of Length: Complete EEPROM Buffer size (LSB First)
 *      4 Bytes EEPROM Info Count (LSB First): Will be only once.
 *      Will be in form of sets based on the EEPORM parameters
 *          1 Byte EEPROM_Config
 *          2 Bytes Offset (LSB First)
 *          2 Bytes Length (LSB First)
 *          N Bytes of Data
 */
const uint8_t aEEPROM_Info[] =
{
    /* Size of EEPROM information buffer. Is a Sum of
     *      EEPROM Information count
     *      EEPROM Config
     *      EEPROM Offset
     *      EEPROM Length
     *      EEPROM Data
     */
    0xAE, 0x05, 0x00, 0x00,

    /* Number of EEPROM sets */
    0x1E, 0x00, 0x00, 0x00,

    /* USER_PMU */
    E_PN76_EEPROM_SECURE_LIB_CONFIG,
    0x00, 0x00,
    0x0A, 0x00,
    0xE4, 0x31, 0xA7, 0xAE, 0xFF, 0xFF, 0x00, 0x2A, 0x2A, 0x1D,

    /* CLKGEN */
    E_PN76_EEPROM_SECURE_LIB_CONFIG,
    0x0F, 0x00,
    0x02, 0x00,
    0x00, 0xFF,

    /* RF_CLOCK_CFG */
    E_PN76_EEPROM_SECURE_LIB_CONFIG,
    0x11, 0x00,
```

Figure 9. NNC EEPROM dump as *.c

In the array *aEEPROM_Info[]* all dumped EEPROM settings can be found. How to read them is explained below.

```
/* ULPCD_CONFIG */
1 E_PN76_EEPROM_SECURE_LIB_CONFIG,
2 0x3A, 0x06,
3 0x06, 0x00,
4 0x97, 0x3D, 0x3F, 0x6A, 0x00, 0x10,
```

Figure 10. EEPROM array assembly explained

The setting in Figure 10 is to be interpreted as follows:

Table 3. EEPROM array assembly explained

Nr.	Length	Description
1	1 byte	Specifies in which EEPROM area this configuration is written. • EEPROM_SECURE_LIB • EEPROM_USER_AREA • EEPROM_IC_CONFIG
2	2 bytes	Offset (address) of the configuration parameter. LSB first. See data sheet for all EEPROM configuration addresses. In this example, the address is set as '0x3A, 0x06' with LSB first → 0x063A offset
3	2 bytes	Length of the data field. LSB first. '0x06, 0x00' → 0x0006 → 6 bytes of data length
4	n byte	The first byte is written to the initial offset. Counting upward. • 0x97 → Offset + 0 → 0x063A • 0x3D → Offset + 1 → 0x063B • 0x3F → Offset + 2 → 0x063C • …

5 MCUXpresso Example

The PN76 MCUXpresso SDK (PN76 SDK) example "*userdata_update*" shows, besides other functions, how a EEPROM configuration file (*.c) can be included in a user application.

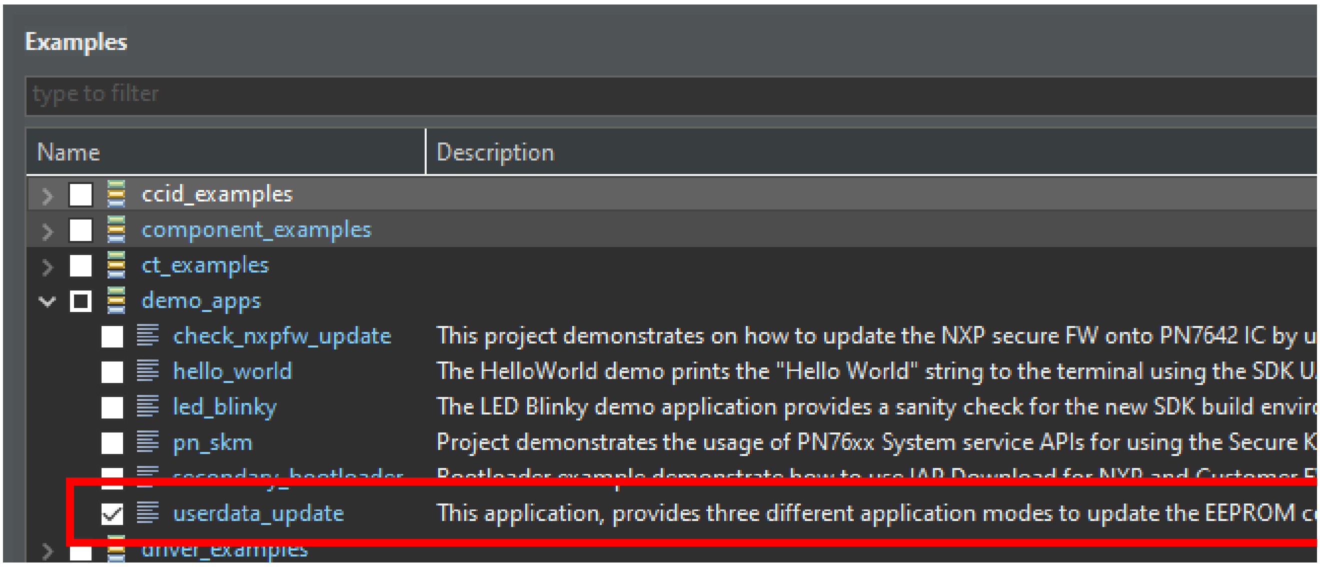


Figure 11. PN76 SDK example userdata_update

5.1 Example flow

The high-level example flow is as shown in the below flowchart.

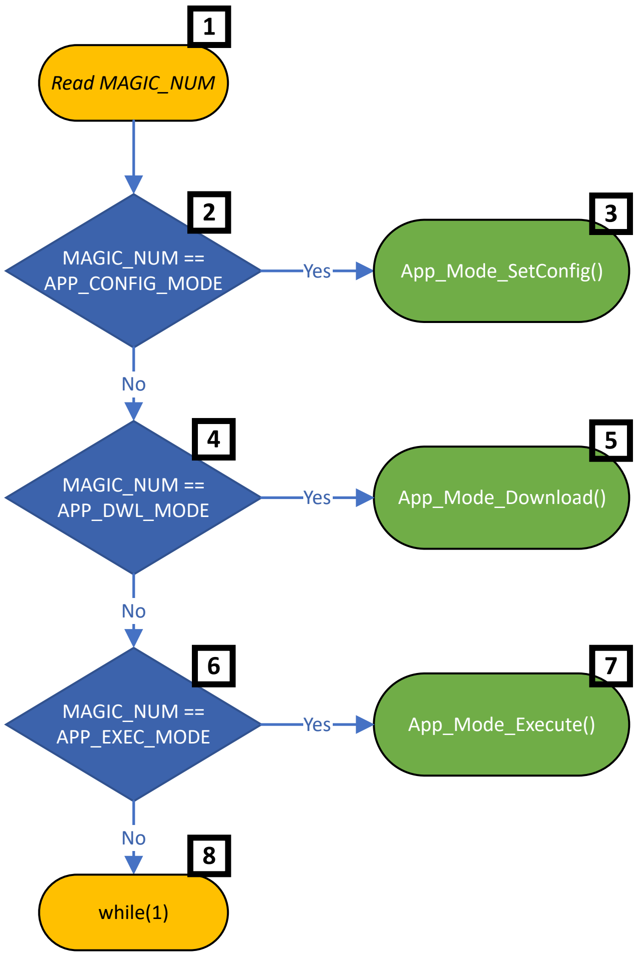


Figure 12. Example 'userdata_update' flowchart

[1] Read Magic Number

The *magic number* determines which mode is executed. The *magic number* is stored at a flash address, specified at the beginning of the file *userdata_update.c* with the macro *ADDR_NSFLASH_APP_MODE.*

```
    /** check which mode the application has to go into
     * 1. APP_CONFIG_MODE --> Magic Number: 0x11335577 Application is in configuration mode, where the configuration
     * settings are applied..
     * 2. APP_DWL_MODE --> Magic Number: 0x22446688 Application download mode.
     * 3. APP_EXEC_MODE --> Magic Number: 0x99AABBCC Application is in execution mode.
     *
     * For the same, a fixed location in Non-secure flash will be assigned to indicate, which mode of application to
     * boot-in.. Fixed location in Non-Secure flash would be say: 0x00234F00
     *
     */
    // Here read the application magic number...
    GET_NSFLASH_APP_MODE(&dwValue);
    dwValue = MAGIC_NUM_APP_CONFIG_MODE;
    switch (dwValue)
```

Figure 13. Example code - Read magic number

In Figure 13 at line 458, the *magic number* is read from the flash space. As the example is only a showcase how it could be done, the next line - 459, is making the value "*dwValue*" fixed. Per default, there is no valid *magic number* written at the specific flash address.

Note: To execute the wanted mode, modify either the value at the flash address or the variable *dwValue* directly in the source code.

[2] Config Mode

If the '*dwValue'* is equal to *MAGIC_NUM_APP_CONFIG_MODE* the yellow LED is turned on, and the method **[3]** '***App_Mode_SetConfig(…)***' is called. This method is responsible to read, interpret, and write the EEPROM values represented in the dumped *.c array.

In the example 'userdata_update', the file 'DumpEEPROM.c' is holding the EEPROM dump from a default PN7642 FW v1.0 chip.

Following while loop is iterating through the Eeprom array and writing the values with the system service API '*PN76_WriteEeprom(…)*'.

```
    dwSize        = *((uint32_t *)ADDR_EEPROM_CFG_DATA_SIZE);
    dwElementsCnt = *((uint32_t *)ADDR_EEPROM_CFG_DATA_CNT);
    ptr           = (uint8_t *)ADDR_EEPROM_CFG_DATA;

    do
    {
        bType = *ptr++;
        if (bType > E_PN76_EEPROM_IC_CONFIG)
        {
            PRINTF("\nType is not correct");
            break;
        }
        wOffset = *ptr++;
        wOffset |= *ptr++ << 8;
        bLen = *ptr++;
        bLen |= *ptr++ << 8;

        // ptr is pointing to the data now.. so call the NSC APIs..
        eStatus = PN76_WriteEeprom((uint8_t *)ptr, wOffset, bLen, (PN76_EEPROM_Config_t)bType);
        if (PN76_STATUS_SUCCESS != eStatus)
        {
            /** An error is occurred. Need to exit out of this.. */
            break;
        }

        ptr += bLen;
        dwCtr++;
        dwTrSize += bLen + 2 + 2 + 1;
    } while ((dwTrSize < dwSize) && (dwCtr < dwElementsCnt));
```

Figure 14. Write EEPROM loop

The next part is iterating through the RF Protocol settings and writing them into the RF Configuration area. Using the system service API '*PN76_Sys_UpdateRfConfiguration*'.

```
    dwSize   = *((uint32_t *)ADDR_PROTO_CFG_DATA_SIZE);
    ptr      = (uint8_t *)ADDR_PROTO_CFG_DATA;
    dwTrSize = 0;

    do
    {
        bLen = *ptr++;
        // Protocol register update structure is present at ptr now. So, directly call the API...
        eStatus = PN76_Sys_UpdateRfConfiguration((PN76_RfConfig_t *)ptr, bLen);
        if (PN76_STATUS_SUCCESS != eStatus)
        {
            /** An error is occurred. Need to exit out of this.. */
            break;
        }
        ptr += bLen;
        dwTrSize += bLen + 1;
    } while (dwTrSize < dwSize);
```

Figure 15. Write RF Configuration loop

If both have been successfully executed, the EEPROM of the PN76 is updated with the values of the provided EEPROM array.

6 References

[1] PN7642 Product data sheet
[2] PN7642 NFC controller User API Documentation

7 Abbreviations

Table 4. Abbreviations

Acronym	Description
NNC	NXP NFC Cockpit
SDK	Software Development Kit
NFC	Near Field Communication
FW	Firmware

8 Legal information

8.1 Definitions

Draft — A draft status on a document indicates that 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 in a draft version of a document and shall have no liability for the consequences of use of such information.

8.2 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.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

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.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

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

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. - NXP B.V. is not an operating company and it does not distribute or sell products.

8.3 Trademarks

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

NXP — wordmark and logo are trademarks of NXP B.V.

SEGGER Embedded Studio — is a trademark of SEGGER Microcontroller GmbH.

Tables

Figures

Contents

Date of release: 18 August 2023
Document identifier: AN13925