AN14587

PN722x – Direct access to Secure Element

Rev. 1.0 — 13 June 2025 Application note

Document information

Information	Content
Keywords	PN722x, Secure Element
Abstract	This document describes how to use PN722x in combination with Secure Element (SE).

1 Introduction

This document describes how to use PN722x in combination with a Secure Element (SE).

It covers the basic architecture, the required AOSP stack and config file changes, and provides a practical example using the PNEV722xBPx board, i.MX 8M Nano/Mini, and an SE051.

To use this document, users must be capable of building an Android Open Source Project (AOSP) with all necessary changes. For information on building an AOSP, refer to ref.[1].

Direct access to an SE is supported from PN722x FW version 03.02.04 onwards (for PN722x FW, refer to ref.[2]).

2 Hardware architecture

This chapter explains the hardware architecture required for direct access to SE. Figure 1 shows basic architecture. PN722x is connected to Device Host (DH) with I2C for data communication, VEN as reset pin, IRQ as interrupt pin and MODE SWITCH as pin to switch between NFC Forum and EMVCo mode. For more information on those connection, refer to ref.[3]. This document focuses on the connection PN722x and the SE.

The communication between PN722x and the SE is performed with the ISO-7816 protocol. Since PN722x is not capable of directly powering the SE, a power supply needs to be provided as follows to minimize power consumption on the end device:

As shown in Figure 1, a transistor is added to between PN722x and the SE to control the power supply to the SE.The transistor is driven by a PN722x GPIO.

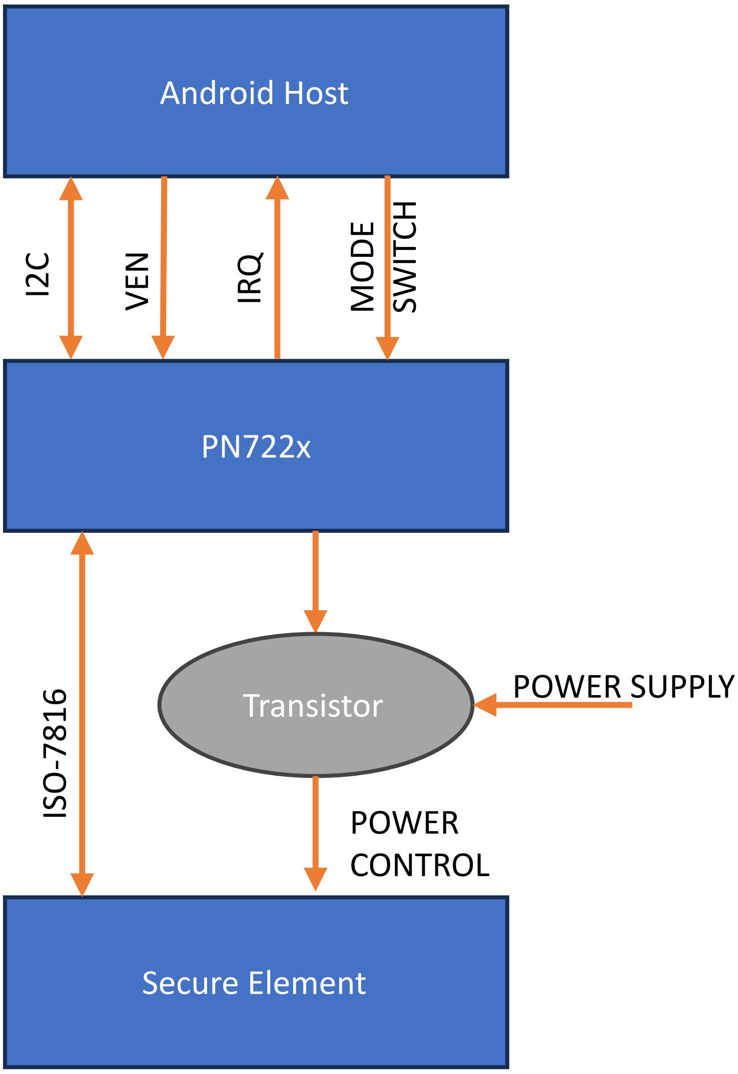

Figure 1. Architecture

Table 1 shows one possible configuration for connecting SE051 with PN722x. GPIO5 functions as the power control pin.

Table 1. SE051 connections to PN722x

SE051 pins	PN722x pins
ENA	GPIO5
ISO_7816_CLK	ISO_CLK_AUX
ISO_7816_IO	ISO_IO_AUX
GND	GND
ISO_7816_RST	GPIO4

For more informations about pins, refer to the respective datasheet (SE051 ref.[4], PN722x ref.[5]).

3 Software architecture

This chapter explains the software architecture required for direct access to SE. The PN722x is controlled via an NCI protocol, which is used to communicate with the SE.

The SE connected to PN722x uses the NCI terminology called NFCEE (NFC Execution Environment). The following actions need to be performed before communication can start. This section covers the theoretical background, Section 6 "Practical approach" presents a practical example.

NFCEE_DISCOVER_CMD needs to be send to check if any NFCEE is connected to the PN722x. On the NXP Android stack, this is done automatically during every NFC Stack bringup. Other commands need to be implemented by the user.

To communicate with the SE:

- Send NFCEE_MODE_SET_CMD to enable the connected NFCEE (SE)
- Send CORE_CONN_CREATE_CMD to create the connection to the NFCEE (SE). This is called Dynamical Logical connection. For all Dynamical Logical connections, the user must also close them.

At this point, the user can send data messages to the NFCEE (SE) using the NCI format. The PN722x FW removes NCI headers and sends only APDU to the NFCEE (SE) via an ISO-7816 interface. When it receives a response, the PN722x FW adds the NCI header to the received APDU and sends it back to the host.

When transactions are done, the user must close the Dynamical Logical connection:

- Send CORE_CONN_CLOSE_CMD to close the Dynamical Logical connection.
- Send NFCEE_MODE_SET_CMD to disable the connected NFCEE (SE)

The explanation mentioned above is theory on how the communication can start with SE. For Android hosts, NXP provides APIs that handle the steps mentioned above. See Section 5.

4 Changes in the Android stack

Note: *The section below shows the minimal changes needed to have enable communication with the SE.* ***It is not production-ready code****. Customers need to check the code, make the necessary changes to their Android build, and perform full testing to ensure proper function.*

The PN722x is mainly used with an Android DH. The main idea was to have up to three TDAs connected to the PN722x, one for EMVCo and two for ISO mode. Each TDA has its own ID (0x20 for EMVCo, 0x21 and 0x22 for ISO mode).

With the PN722x FW release 03.02.04, support for direct access to SE was added. In this case, the only possibility is to have one connected SE and this slot is considered as an ISO slot with the ID 0x20.

In the main Android stack after sending the NFCEE_DISCOVER_CMD, the PN722x returns a response with the number of connected NFCEE devices (three for TDA, one for direct access to SE). After that for each connected NFCEE, the notification with ID and status follows.

Since the ID 0x20 in TDA is used for EMVCo mode, NXP is blocks notifications to the upper layers. The reason for that is that NFC Forum execution must not be aware of any EMVCo related activities. For direct access to the SE, notifications need to be seen by the upper layers, therefore the following changes in the *phNxpNciHal_ext.cc* must be made. Table 2 and Figure 2 show the changes, the left site is original file and right site is updated one.

Table 2. ***phNxpNciHal_ext.cc*** **changes and additions for direct access**

Standard	Code changes and additions

Following the line:
`NFCSTATUS status = NFCSTATUS_SUCCESS;`
	Add:
	`unsigned long isTdapresent = 0;`
	`int isTdaConfFound = 0;`

Before the line:
`if ((p_ntf[0] == 0x62) && (p_ntf[1] == 0x00) &&`
` (p_ntf[3] == 0x20)) {`
	Add:
	`isTdaConfFound =`
	` GetNxpNumValue(NAME_NXP_IS_TDA_CHIP_PRESENT,`
	` &isTdaPresent, sizeof(isTdaPresent));`

`if ((p_ntf[0] == 0x62) && (p_ntf[1] == 0x00) &&`
` (p_ntf[3] == 0x20)) {`
	Change to:
	`if (((p_ntf[0] == 0x62) && (p_ntf[1] == 0x00) &&`
	` (p_ntf[3] == 0x20)) && isTdaPresent == 0x00 {`

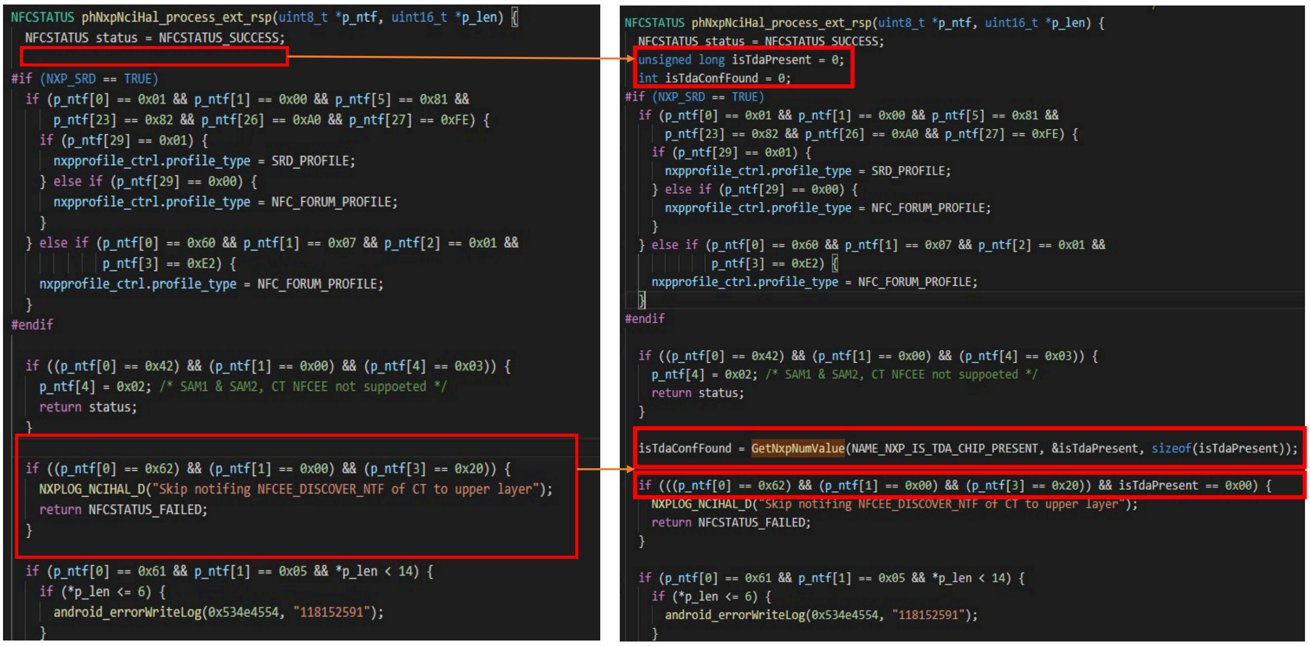

Figure 2. Code location and code difference before and after change

Since in the Android stack, APIs are used to open and close the connection, the following changes need to be madein *NfcService.java*. Table 3, Figure 3 and Figure 4 show the code changes in the functions closeTDA and openTDA in *NfcService.java*.

Note: *(Figure 3 is taken from an Android 14 stack, but the same changes are applicable for other MW versions..*

Table 3. closeTDA and openTDA code changes in *NfcService.java* for direct access

Standard	Code changes and additions

`if ((mState != NfcAdapter.STATE_ON) ||`
`  (tdaID == CT_NFCEE_ID)) { /* CT NFCEE not supported`
` */`
`  Log.d(TAG, "Not in NFC mode. Failed to call the`
` closeTDA API")`
`  tdaResult.setStatus(TdaResult.RESULT_FAILURE);`
`  return;`
`}`
	`if ((mState != NfcAdapter.STATE_ON) {`
	`  Log.d(TAG, "Not in NFC mode. Failed to call the`
	` closeTDA API")`
	`  tdaResult.setStatus(TdaResult.RESULT_FAILURE);`
	`  return;`
	`}`

```java
@Override
public void closeTDA(byte tdaID, boolean standBy, TdaResult tdaResult) {
  NfcPermissions.enforceUserPermissions(mContext);

  try {
    if ((mState != NfcAdapter.STATE_ON) ||
        (tdaID == CT_NFCEE_ID)) { /* CT NFCEE not supported */
      Log.d(TAG, "Not in NFC mode. Failed to call the closeTDA API");
      tdaResult.setStatus(TdaResult.RESULT_FAILURE);
      return;
    }
    Bundle tdaBundle = new Bundle();
    tdaBundle.putByte("tdaID", tdaID);
    tdaBundle.putBoolean("standBy", standBy);
    sendMessage(NfcService.MSG_CLOSE_TDA, tdaBundle);
    synchronized (mCloseTdaObj) { mCloseTdaObj.wait(1000); }
  } catch (Exception e) {
    e.printStackTrace();
  }

  byte st = mCloseTdaBundle.getByte("status");
  if (st == 0x00) {
    tdaResult.setStatus(TdaResult.RESULT_SUCCESS);
  } else {
    tdaResult.setStatus(TdaResult.RESULT_FAILURE);
```

```java
@Override
public void closeTDA(byte tdaID, boolean standBy, TdaResult tdaResult) {
  NfcPermissions.enforceUserPermissions(mContext);

  try {
    if (mState != NfcAdapter.STATE_ON){
      Log.d(TAG, "Not in NFC mode. Failed to call the closeTDA API");
      tdaResult.setStatus(TdaResult.RESULT_FAILURE);
      return;
    }
    Bundle tdaBundle = new Bundle();
    tdaBundle.putByte("tdaID", tdaID);
    tdaBundle.putBoolean("standBy", standBy);
    sendMessage(NfcService.MSG_CLOSE_TDA, tdaBundle);
    synchronized (mCloseTdaObj) { mCloseTdaObj.wait(1000); }
  } catch (Exception e) {
    e.printStackTrace();
  }

  byte st = mCloseTdaBundle.getByte("status");
  if (st == 0x00) {
    tdaResult.setStatus(TdaResult.RESULT_SUCCESS);
  } else {
    tdaResult.setStatus(TdaResult.RESULT_FAILURE);
```

Figure 3. Code location and code difference before and after change for closeTDA

```java
@Override
public byte openTDA(byte tdaID, boolean standBy, TdaResult tdaResult) {
  NfcPermissions.enforceUserPermissions(mContext);

  try {
    if ((mState != NfcAdapter.STATE_ON) ||
        (tdaID == CT_NFCEE_ID)) { /* CT NFCEE not supported */
      Log.d(TAG, "Not in NFC mode. Failed to call the openTDA API");
      tdaResult.setStatus(TdaResult.RESULT_FAILURE);
      return INVALID_CID;
    }
    Bundle tdaBundle = new Bundle();
    tdaBundle.putByte("tdaID", tdaID);
    tdaBundle.putBoolean("standBy", standBy);
    sendMessage(NfcService.MSG_OPEN_TDA, tdaBundle);
    synchronized (mOpenTdaObj) { mOpenTdaObj.wait(1000); }
  } catch (Exception e) {
    e.printStackTrace();
  }

  byte mCID = mOpenTdaBundle.getByte("mCID");
  if ((mCID == SAM1_CID) || (mCID == SAM2_CID)) {
    tdaResult.setStatus(TdaResult.RESULT_SUCCESS);
  } else {
    tdaResult.setStatus(TdaResult.RESULT_FAILURE);
```

```java
@Override
public byte openTDA(byte tdaID, boolean standBy, TdaResult tdaResult) {
  NfcPermissions.enforceUserPermissions(mContext);

  try {
    if (mState != NfcAdapter.STATE_ON){
      Log.d(TAG, "Not in NFC mode. Failed to call the openTDA API");
      tdaResult.setStatus(TdaResult.RESULT_FAILURE);
      return INVALID_CID;
    }
    Bundle tdaBundle = new Bundle();
    tdaBundle.putByte("tdaID", tdaID);
    tdaBundle.putBoolean("standBy", standBy);
    sendMessage(NfcService.MSG_OPEN_TDA, tdaBundle);
    synchronized (mOpenTdaObj) { mOpenTdaObj.wait(1000); }
  } catch (Exception e) {
    e.printStackTrace();
  }

  byte mCID = mOpenTdaBundle.getByte("mCID");
  if ((mCID == SAM1_CID) || (mCID == SAM2_CID)) {
    tdaResult.setStatus(TdaResult.RESULT_SUCCESS);
  } else {
    tdaResult.setStatus(TdaResult.RESULT_FAILURE);
```

Figure 4. Code location and code difference before and after change for openTDA

In the first code, the flag (NXP_IS_TDA_CHIP_PRESENT) that exist in *libnfc-nxp-eeprom.conf* is used. If users want direct connection to SE, this flag need to be set to 0x01 in the configuration file.Figure 5 shows the configuration file flag.

```
################################################################################
# Enabling the bit shows TDA IC is not connected on board and only single slot
# is directly interfaced with PN722x IC.
# 0 : TDA chip present (Default)
# 1 : TDA chip absent
NXP_IS_TDA_CHIP_PRESENT=0x00
################################################################################
```

Figure 5. NXP_IS_TDA_CHIP_PRESENT setting

Following these changes, the user can now directly access the SE while also using the TDA APIs to open/close/communicate with the SE.

5 API explanation

As mentioned in Section 3, if the user is using an Android host with an NXP Android build, the APIs are exposed to the application layer and can be used for the TDA and also for direct access to the SE. Below is a list of APIs that need to be used to create the channel and communicate with the SE. The APIs are located in *com.nxp.nfc*. The APIs need to be called in right following order:

1. ```
   NfcTDAInfo[] discoverTDA(TdaResult var1)
   ```
   This API discovers the SE/TDA.
2. ```
   byte openTDA(byte tdaID, boolean standBy, TdaResult var3)
   ```
   This API opens the connection to SE/TDA.
3. ```
   byte[] transceive(byte[] var1, TdaResult var2)
   ```
   This API sends the APDU to SE/TDA.
   Note: *The Byte array in the transceive API needs to be in the NCI data message format.*
4. ```
   void closeTDA(byte tdaID, boolean standBy, TdaResult var3)
   ```
   This API closes the connection to SE/TDA.

In case of direct access to SE, the tdaID is **0x20**.

Examples of the TDA application can be found in ref.[7]. Users can reuse these examples to develop new ones with communication to the SE.

6 Practical approach

This section provides an example serving as a proof of concept for direct access to the SE using the following boards:

- PNEV722xBP1 (ref.[8])
- SE051 Development Kit (EVK) (ref.[9])
- i.MX 8M Nano/Mini (ref.[10]/ref.[11])

The hardware connections between SE051 and PNEV722xBP1 are the same as shown in Table 1 with one small deviation from the hardware architecture example shown in Section 2 "Hardware architecture" to enable the connection without the use of a transistor. Connect the VIN of the SE051 to 3_3_VDD (VDDIO) of the PN7220. The ENA pin of SE051 is connected directly to GPIO5. To simulate the enabling/disabling of the SE051, the user can connect the ENA pin to GND and then back to GPIO5.

Note: *The setup described above is only valid for testing purposes.*

Table 4 shows the connections between the SE051 EVK and the PNEV722xBP1 board:

Table 4. Pin connections between SE051 EVK and PNEV7220BP1

SE051 EVK	PNEV7220BP1
J11 - 10 (SE_ENA)	J13 - 2 (GPIO5)
J11 - 6 (SE_CLK)	J15 - 5 (ISO_CLK_AUX)
J11 - 9 (SE_IO1)	J15 - 3 (ISO_IO_AUX)
J11 - 7 (GND)	J13 - 4 (GND)
J11 - 2 (VIN)	J12 - 4 (VDDIO)
J11 - 3 (SE_RST)	J14 - 2 (GPIO4)

Table 5 shows the pin configuration on PNEV7220BP1:

Table 5. PNEV7220BP1 pin configuration

Pin name	Status
J4	Shorted
J1	Open
J2	Shorted
J47	Open
J3	Open
J5	3-4 Shorted
J65	Open

Table 6 shows the pin configuration on the SE051 Development Kit (EVK) board:

Table 6. SE051 EVK pin configuration

Pin name	Status
J13	2-3 Shorted
J15	3-4 Shorted
J17	3-4 Shorted
J38	Open

Table 6. SE051 EVK pin configuration*...continued*

Pin name	Status
J37	Open
J10	Open
J9	Open
J7	Open
J6	Open
J14	3-4 Shorted
J24	1-2 Shorted
J18	1-2 Shorted
J12	1-2 Shorted
J16	1-2 Shorted
J19	2-3 Shorted

Users can develop the Android application with the APIs explained in Section 5 "API explanation".

Figure 6 shows the NCI communcation when users call discoverTDA. The same commands need to be sent if the PN7220 is not connected to the Android device host.

```
2201022001   -=(NFCEE_MODE_SET_CMD)=- enabled 20
42010100
62010100   <<NFCEE_MODE_SET_NTF>>
620016200001000101 0F3BD518FF8191FE1FC38073C8211000   <<NFCEE_DISCOVER_NTF>> 20 enabled:APDU
 ATR:3BD518FF8191FE1FC38073C8211000
2004060301010220 00   -=(CORE_CONN_CREATE_CMD)=- NFCEE
 20 APDU
40040400FF010A
```

Figure 6. NCI commands while calling discoverTDA and openTDA

After the connection is opened, the users can start sending APDU with the transceive API (see Section 5). Figure 7 shows the APDU exchange with the SE. CORE_CONN_CREDIT_NTF is received from the controller. Refer to ref.[6] to for information on credit notifications.

```
0A000780CA00FE02DF28
600603010A01   <<CORE_CONN_CREDITS_NTF>>
0A0049FE45DF2842010C0005A4A6145E7CCC169A7BC502080000000000000000103184A3352333531303239423431313130304C0954E73E773C6E05010007010008082E5AD88409C9BADB9000
```

Figure 7. NCI communication when calling transceive API

Figure 8 shows NCI communication while closeTDA API is called.

```
2005010A   -=(CORE_CONN_CLOSE_CMD)=-
40050100
2201022000   -=(NFCEE_MODE_SET_CMD)=- disabled 20
42010100
62010100   <<NFCEE_MODE_SET_NTF>>
```

Figure 8. NCI communication when calling closeTDA API

7 Important considerations and design recommendations

It is recommended to place the SE as close as possible to the PN722x on the PCB to keep connections as short as possible. Cross talk between the data lines must be minimized during the PCB design.

Supply spikes during Activation/Deactivation might be caused by secure element and need to be considered during the power supply design.

The following features provided by TDA8035 are not available in case of direct connection to the secure element:

- Contact Card class selection
- ESD protection offered by TDA
- Card overload protection
- CT EMVCo compliance
- Card detection logic

8 Abbreviations and acronyms

Table 7. Abbreviations

Acronym	Description
AOSP	Android Open Source Project
DH	Device Host
HW	Hardware
I2C	Inter-Integrated Circuit
NCI	NFC Controller Interface
NFCEE	NFC Execution Environment
SE	Secure Element
SW	Software

9 References

[1] Application note – AN14430 – PN7160/PN7220 – Android 14 porting guide (link)
[2] Web page – GitHub PN7220 FW (link)
[3] User manual – UM1180 – PN722X NFC controller (link)
[4] Datasheet – SE051 – Plug & Trust Secure Element (link)
[5] Datasheet – PN7220 – NFC controller with NCI interface supporting EMV and NFC Forum applications (link)
[6] Web page – Controller Interface (NCI) Technical Specification (link)
[7] Web page – NFC TDA Test Application (link)
[8] Web page – PNEV7220BP1 – Development Board for PN7220 NFC Controller for EMVCo and NFC Forum Operation (link)
[9] Web page – OM-SEO51ARD – EdgeLock® SE051 Development Kit (link)
[10] Web page – 8MNANOD4-EVK – Evaluation Kit for the i.MX 8M Nano Applications Processor (link)
[11] Web page – 8MMINILPD4-EVK – Evaluation Kit for the i.MX 8M Mini Applications Processor (link)

10 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

11 Revision history

Table 8. Revision history

Document ID	Release date	Description
AN14587 v.1.0	13 June 2025	• Initial version

Legal information

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.

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 https://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.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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.

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.

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.

Tables

Figures

Contents