Application Report
SLOA208B–November 2014–Revised March 2019

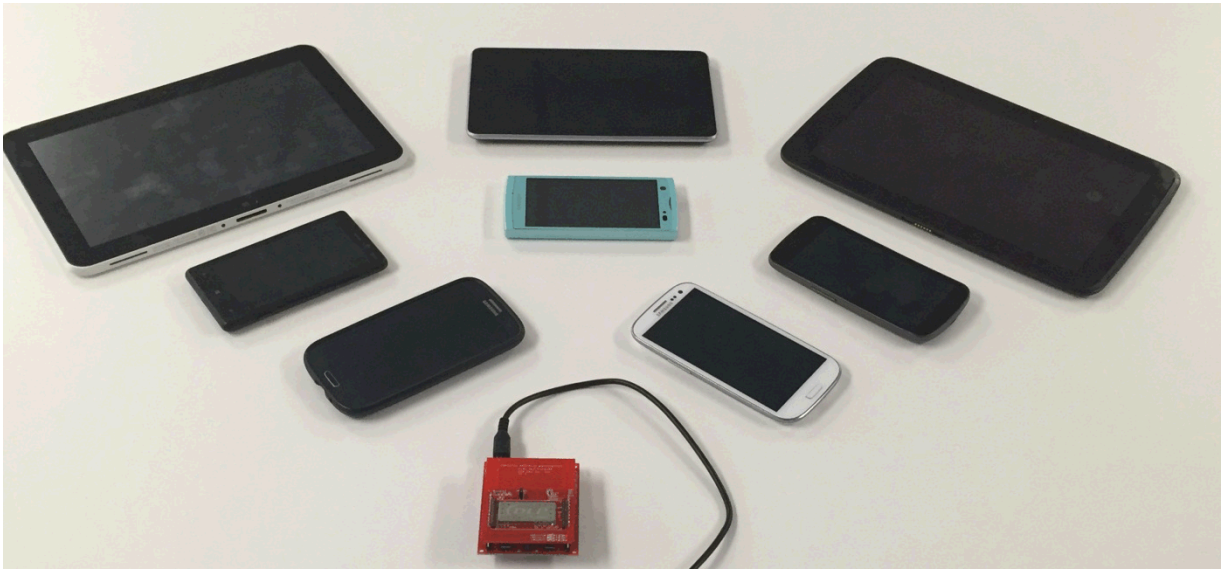
NFC card emulation using the TRF7970A

Erick Macias, Ralph Jacobi, Josh Wyatt *Safety and Security (S2) NFC/RFID Applications*

ABSTRACT

The Near Field Communication (NFC) market is emerging into multiple fields including medical, consumer, retail, industrial, automotive, and smart grid. Card emulation is one of the three operational modes supported by the TRF7970A. When using card emulation, the user can configure the TRF7970A to emulate a Type 4A or Type 4B tag platform. When emulating either of the tag platforms, record type definitions (RTD) can be used such as text, URI, smart poster, or V-Card. It should also be noted here that card emulation on the TRF7970A is not limited to NFC applications. Other applications could include payment systems, access control, proprietary customer systems, and other ISO 14443-4 applications. This application report describes the fundamental concepts of how card emulation is to be implemented and also how to successfully accomplish the mastery of the concept while using the TRF7970A.

The sample code described in this document can be downloaded from www.ti.com/lit/zip/sloa208.



Contents

List of Figures

List of Tables

Trademarks

BoosterPack, LaunchPad are trademarks of Texas Instruments.
Arm, Cortex are registered trademarks of Arm Limited.
Bluetooth is a registered trademark of Bluetooth SIG.
Wi-Fi is a registered trademark of Wi-Fi Alliance.
All other trademarks are the property of their respective owners.

Terms, Definitions, and Symbols

ALL_REQ

NFC-A polling command, equivalent to ISO 14443-3 short frame command WupA (0x52)

ALLB_REQ

NFC-B polling command, equivalent to ISO 14443-3 command WupB

APDU

Application **P**rotocol **D**ata **U**nit, used in command-response pairs to exchange I (information), R (receive ready) or S (supervisory) blocks.

ATTRIB

PICC selection command, Type B

ATQA

Answer **T**o Re**Q**uest **A,** ISO 14443-3 term, equivalent to NFC term SENS_RES

ATQB

Answer **T**o Re**Q**uest **B,** ISO 14443-3 term, equivalent to NFC term SENSB_RES

CC

Capability **C**ontainer, contains management data and is stored inside a read-only EF file. This EF is located inside the NDEF tag application. Default (or basic) file ID for this file (for NFC) is 0xE103. See NFC Type 4 Tag Operation Specification and ISO/IEC 7816-4 for more information.

CE

Card **E**mulation, one of the three modes offered by NFC devices. In this optional mode of NFC operation, an NFC Forum device is considered to be a card emulation platform only when it is emulating a Type 3, Type 4A or Type 4B tag platform. Emulation of any other cards or tags is outside the scope of the NFC Forum Brand Promise.

DEP

Data **E**xchange **P**rotocol, an abstracted operational layer that uses either ISO or NFC protocols to exchange data. Thus, two terms can be created with this acronym: ISO-DEP and NFC-DEP. In the context of this document, after activation and selection of the emulated card and before deactivation, ISO-DEP is used exclusively to exchange data between the reader/writer (PCD) and the tag platform (PICC).

EF

Elementary **F**ile, a set of data objects, records or units sharing the same file identifier and the same security attributes

File Identifier

2-byte data element used to address a file

fc

Carrier frequency, in the context of NFC/HF RFID, is 13.56 MHz ±7 kHz. This is the fundamental transmit frequency of the reader/writer (also called PCD).

Initiator

Generator of the RF field and source of the beginning of the NFCIP-1 communication

MIME

Multipurpose **I**nternet **M**ail **E**xtensions

Modulation Index, *m*

Signal amplitude ratio of [(peak – minimum) / (peak + minimum)] or [(1 – b) / (1 + b)], where ***b*** is the ratio between the modulated amplitude and the initial signal amplitude. The index, ***m***, is defined per protocol type for both downlink and uplink and is generally expressed as a percentage (for example, Type A uses ***m*** = 100%, Type B uses ***m*** = 8 to 14%)

NDEF

NFC **D**ata **E**xchange **F**ormat

NFC

Near **F**ield **C**ommunication

PCB

Protocol **C**ontrol **B**yte, used for conveying information required to control data transmission of blocks during the exchange of command-response APDU pairs. Bit coding of this byte and usage rules are found in ISO/IEC FDIS 14443-4.

PCD

Proximity **C**oupling **D**evice (also commonly referred to as reader/writer)

PICC

Proximity **I**ntegrated **C**ircuit **C**ard (also commonly referred to as tag, tag platform, or transponder)

PUPI

Pseudo **U**nique **P**ICC **I**dentifier (randomly generated or static number returned by PICC as part of the response to REQB, WupB, ALLB_REQ or SENSB_REQ)

RTD

Record **T**ype **D**efinition

SAK

Select **A**c**K**nowledge (from ISO 14443-3), in NFC terms this is also called SEL_RES

SDD_RES

Equivalent to ISO 14443-3 response to SDD_REQ, and is complete NFCID1 CLn + BCC (if cascade level 1 (single size UID) or indicates NFCID1 is incomplete in the response and further cascade levels must be completed to obtain complete NFCID1+BCC.

SDD_REQ

Equivalent to ISO 14443-3 Type A anticollision sequence. Comprised of SEL_CMD, SEL_PAR and n data bits coded based on cascade level specified by SEL_CMD and calculated by value of SEL_PAR

SEL_RES

Equivalent to ISO 14443-3 SAK response

SENS_REQ

NFC-A polling command, equivalent to ISO 14443-3 short frame command REQ_A (0x26)

SENS_RES

NFC-A polling response, equivalent to ISO 14443-3 ATQA

SENSB_RES

NFC polling command response, equivalent to ISO 14443-3 ATQB

SENSB_REQ

Polling command, equivalent to ISO 14443-3 command REQ_B

Single Device Detection (SDD)

Algorithm used by the reader/writer to detect one out of several targets in its RF field (Type A anticollision, from ISO/IEC 14443-3)

T4T

Type **4** **T**ag

Tag Platform

Responds to reader/writer (PCD) commands by using load modulation scheme (passive operation)

TLV

Type **L**ength **V**alue

UID

Unique **ID**entifier, a randomly generated or static number returned by Type A PICC as part of the response to REQA, WupA, ALL_REQ or SENS_REQ. It must be 4, 7, or 10 bytes long.

1 Introduction

The TRF7970A supports three operational modes: reader/writer, card emulation, and peer-to-peer. This document describes on how to use the TRF7970A in card emulation mode. Card emulation allows an NFC enabled system to act as or 'emulate' a tag platform. In the case of the TRF7970A, it is possible to emulate both Type 4A and Type 4B tags concurrently. This feature is a differentiator when compared to static tags that typically offer a single tag type platform.

Card emulation for Type 4A uses ISO/IEC 14443A technology at a baud rate of 106 kbps. Card emulation for Type 4B uses ISO/IEC 14443B technology at a baud rate of 106 kbps. After technology selection for either mode has been completed, the higher layers are the same (as shown in Figure 1).

When configured for the aforementioned mode, the TRF7970A transceiver behaves as an emulated card, so it does not produce its own RF field. When a RF field is presented from a reader, and properly formatted commands are issued, the transceiver load modulates the reader's field to communicate with it.

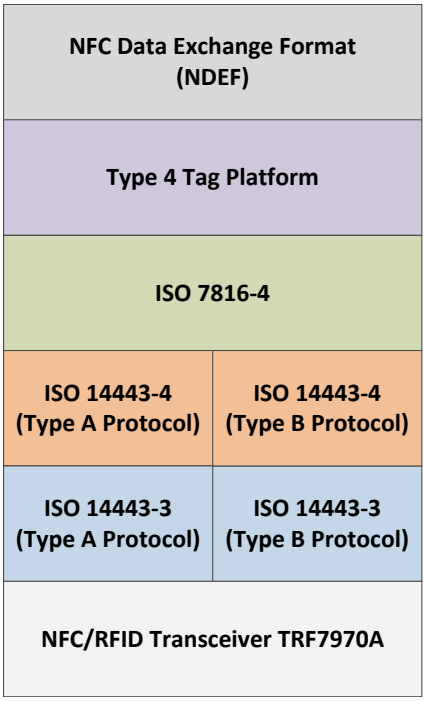


Figure 1. Card Emulation Layers Including the PHY (TRF7970A)

A 16-bit and a 32-bit microcontroller are used to interface with the TRF7970A to demonstrate a reference example of the card emulation mode. The firmware supports flexible functions that allow the user to enable or disable the supported card emulation modes. Additionally, the firmware demonstrates how to format various NDEF message types for card emulation applications, and allows NFC-enabled smart phones to write custom NDEF content to the device.

Table 1 lists the NFC-enabled devices that were used to validate the firmware.

Table 1. NFC Enabled Devices Used to Test Card Emulation

Smartphone Model (Release Date)	Operating System	Kernel Version
Samsung Galaxy Nexus (Nov 2011)	Android 4.3	3.0.72 Jun 7 2013
Samsung Galaxy S3 (AT&T) (June 2012)	Android 4.0.4	3.0.8 Aug 29 2012
Samsung Galaxy S3 (T-Mobile) (June 2012)	Android 4.3	3.0.31 Mar 8 2014
Asus Nexus 7 (July 2012)	Android 4.4.2	3.1.10 Nov 20 2013
Samsung Galaxy Note 2 (Sept 2012)	Android 4.4.2	3.0.31 May 23 2014
AU Arrows Fujitsu FJL21 (Oct 2012)	Android 4.0.4	3.0.21 Oct 16 2012
Samsung S3 Mini (Oct 2012)	Android 4.4.2	3.4.0 Jun 9 2014
Nokia Lumia 820 (Nov 2012)	Windows Phone 8	8.0.10328.78
HP Elite Tablet (Nov 2012)	Windows 8	Windows 8 Pro
Samsung Nexus 10 (Nov 2012)	Android 4.4.2	3.4.39 Nov 20 2013
Google Nexus 4 (Nov 2012)	Android 4.4.4	3.4.0 Apr 16 2014
Samsung Galaxy S4 (April 2013)	Android 4.4.4	3.4.0 Aug 27 2014
Hisense Sero 7 Pro (June 2013)	Android 4.4.1	3.1.0
Asus Nexus 7 (July 2013)	Android 4.4.3	3.4.0 Mar 18 2014
Google Nexus 5 (Oct 2013)	Android 4.4.4	3.4.0 Mar 17 2014
Samsung Galaxy S5 (April 2014)	Android 4.4.2	3.4.0 Jul 22 2014
Sony Xperia Z3 (September 2014)	Android 4.4.4	3.4.0 Aug 19 2014

2 Card Emulation

The TRF7970A supports card emulation for both Type A and Type B at 106 kbps (*fc*/128). When the transceiver is in default mode [ISO mode (See TRF7970A data sheet Section 5.9.6 Direct Mode for more information)] only the decoded data is available to the MCU through the FIFO.

This section covers the program flow and register settings for the anticollision sequence of each card emulation mode, and then provides a brief overview of how the data exchange sequence functions for Type 4A and Type 4B tag platforms. Figure 2 shows a high-level flow diagram.

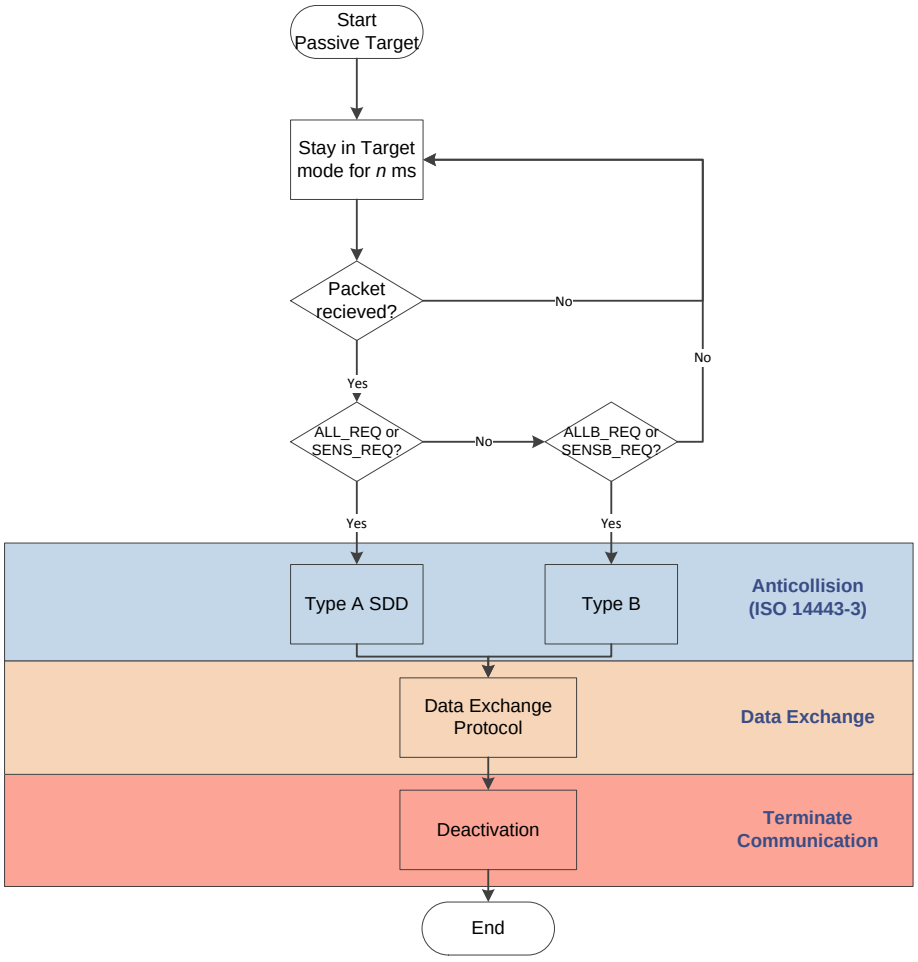


Figure 2. Card Emulation Flow Diagram

2.1 Anticollision

2.1.1 Card Emulation Type A

The frame format for Data Exchange Protocol (DEP) packets at 106 kbps during anticollision (see Figure 3) is based on NFC-A technology specifications in NFCForum-TS-DigitalProtocol-1.0 or the ISO 14443-3 specifications.

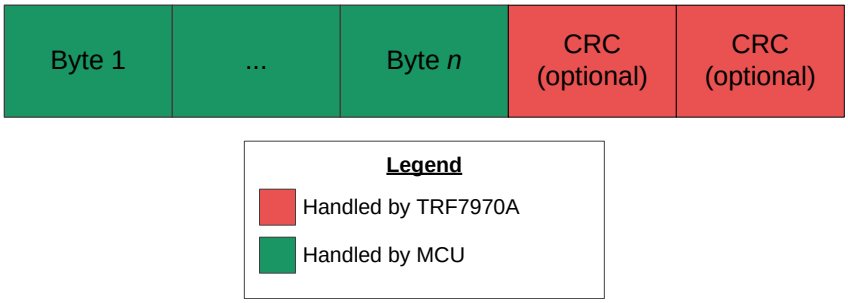


Figure 3. Frame Format for Card Emulation Type A Commands During Anticollision

Initially, when configured for Type 4A tag platform emulation, the TRF7970A must be receiving without CRC, because of the ISO 14443A protocol being used. After receiving commands from the reader/writer, the following registers must be modified before and after the anticollision is completed:

1. ISO Control Register (0x01) → 0xA4 (ISO 14443A 106 kbps, receive without CRC during anticollision, before Select command)

   or

2. ISO Control Register (0x01) → 0x24 (ISO 14443A 106 kbps, receive with CRC, after anticollision is completed).
3. Send packet:
   1. Reset FIFO (0x0F) direct command.
   2. Transmission without (0x10, anticollision before Select command) or with (0x11, after anticollision is completed) CRC direct command.
   3. TX Length Byte 1 and 2 (0x1D and 0x1E) registers
   4. Write the response to the FIFO.

The ISO control register needs to be modified for the anticollision state to receive without CRC for the required commands (See the TRF7970A data sheet and ISO/IEC 14443-3 specification for more information). When the anticollision is completed, the ISO Control register needs to be modified to receive with CRC. Step 2 must be used to send commands to the reader/writer.

Figure 4 shows the Type A anticollision flowchart. The TRF7970A supports only 106 kbps for card emulation of Type A.

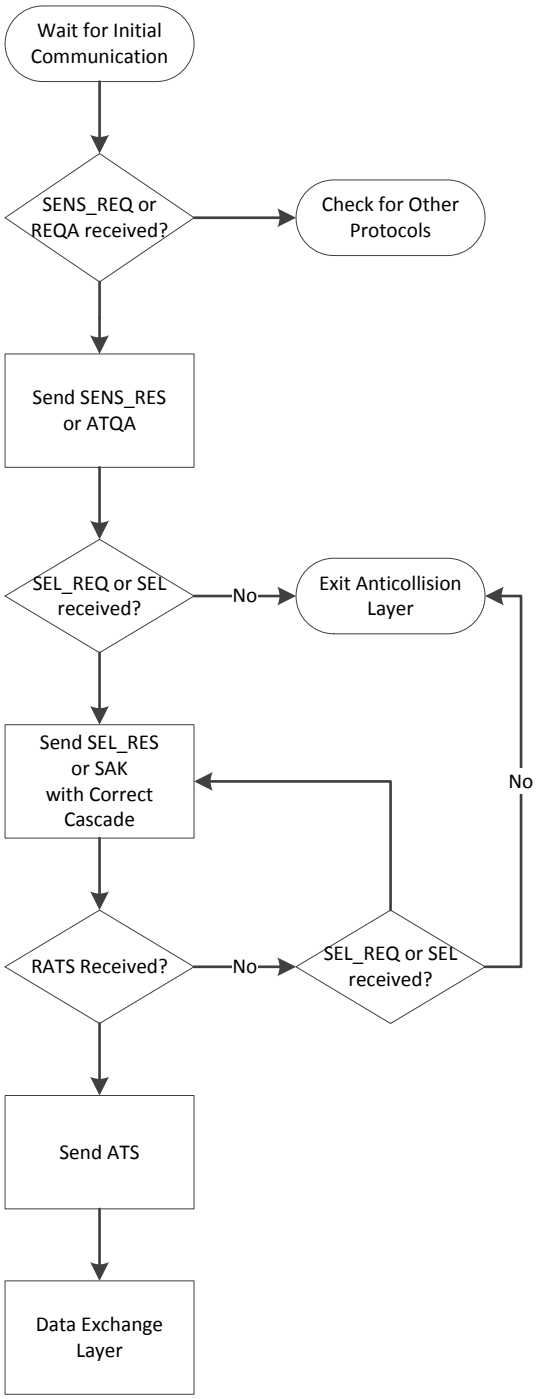


Figure 4. Card Emulation Type A Anticollision

2.1.2 Card Emulation Type B

The frame format for Data Exchange Protocol (DEP) packets at 106 kbps during anticollision (see Figure 5) is based on the NFC-B technology specifications in NFCForum-TS-DigitalProtocol-1.0 or the ISO 14443-3 specifications. The Type B frame format must always include the two CRC bytes.

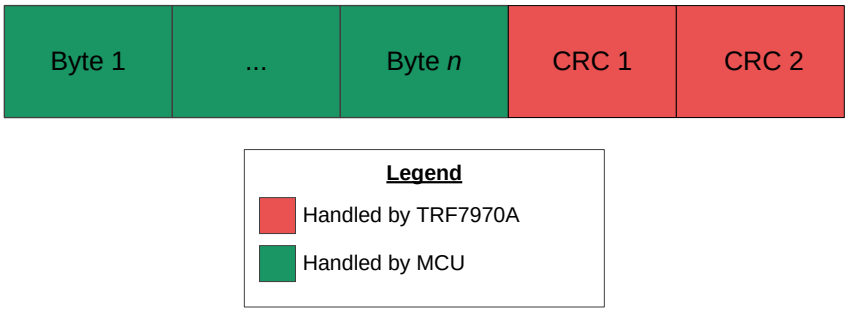


Figure 5. Frame Format for Card Emulation Type B Commands During Anticollision

Initially, when configured for Type 4B tag platform emulation, the TRF7970A must receive with CRC.

After receiving commands from the reader/writer, the following registers must be modified before and after the anticollision is completed:

1. ISO Control register (0x01) → 0x25 (ISO 14443B 106 kbps, receive with CRC, after anticollision is completed).
2. Send packet:
    1. Reset FIFO (0x0F) direct command
    2. Transmission (0x11) with CRC direct command
    3. TX Length Byte 1 and 2 (0x1D and 0x1E) registers
    4. Write the response to the FIFO

Figure 6 shows the Type B anticollision flowchart. The TRF7970A supports only 106 kbps for card emulation of Type B.

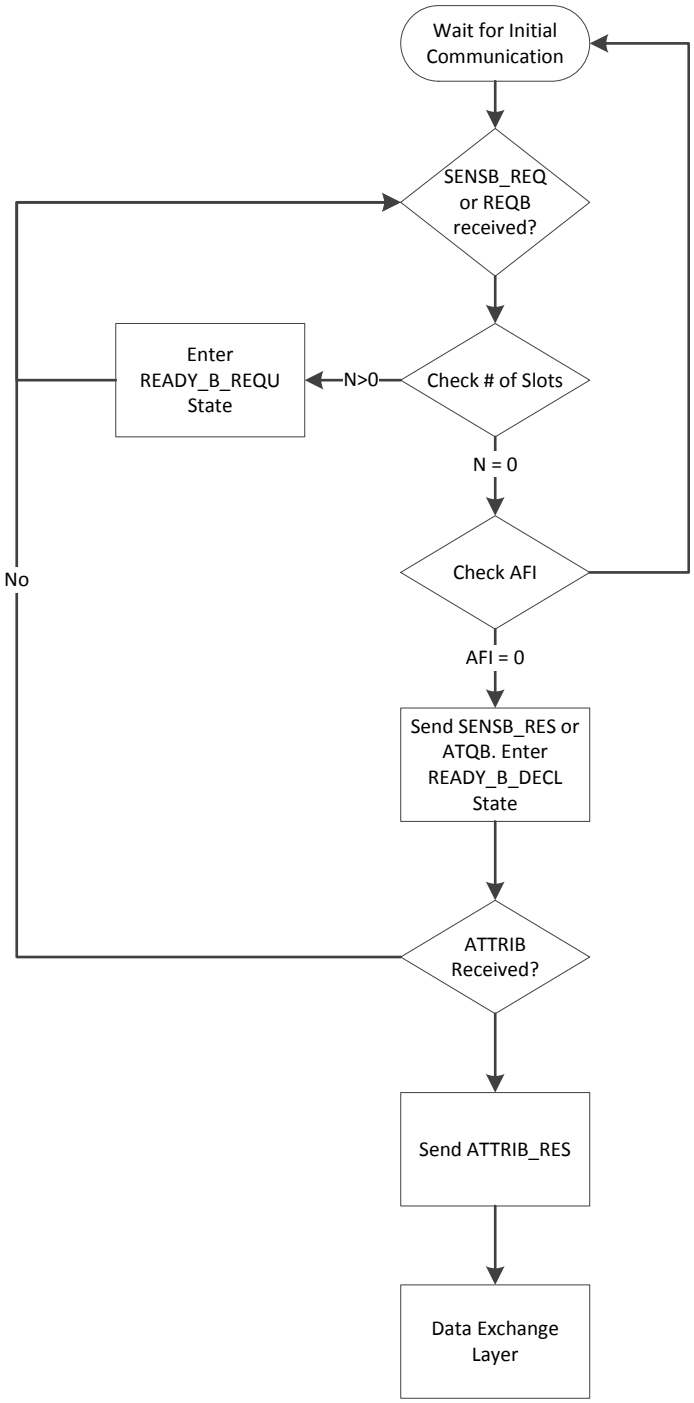


Figure 6. Card Emulation Type B Anticollision

2.2 Data Exchange

After the anticollision procedures for Type A or Type B are completed, the data exchange layer is entered. Within the data exchange layer, the commands for Type 4A and Type 4B tag platforms are identical. Both Platforms also use the same frame formats.

The frame format for DEP packets at 106 kbps during data exchange (see Figure 7) is different than the frame format during anticollision. Depending on the command that is issued by the device only certain optional bytes are included.

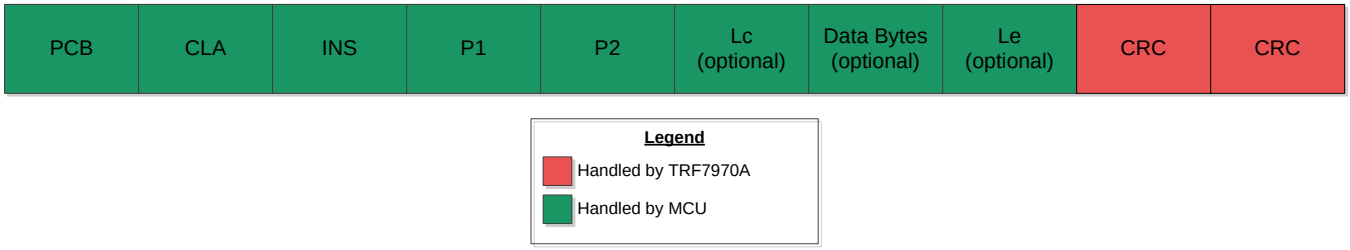


Figure 7. Frame Format for ISO 7816-4 Data Exchange Commands

The commands used by both tag platforms use the ISO 7816-4 specifications and are further defined in the NFC Forum T4TOP specifications. When an NFC enabled reader/writer is presented, the firmware allows for the selection of either supported tag type (Type 4A/B) during the anticollision process and then enters the data exchange layer.

Because the data exchange layer supports DEP commands for both tag platforms, the firmware allows has been designed to minimize size by using the same layer for both tag types. This is accomplished by only having a distinction between Type 4A and Type 4B tag platforms in the technology detection and anticollision sequences. This modularity gives the TRF7970A a more robust experience than basic static tags that are constrained to only working with one reader/writer type.

3 Configuration and Commands for Type 4 Tag Platforms

Type 4 tag platforms are structured in a specific format that must be used to have a functional tag. Only a properly structured tag file, such as the one provided with the firmware, shall be able to reply to all Type 4 tag commands from NFC enabled reader/writer devices. This section provides an overview of how to properly structure an emulated Type 4 tag platform, some common data types stored within tags, and how the Type 4 tag commands function.

3.1 Overview of Type 4 Tag Configuration

To take advantage of the TRF7970 compatibility with both Type A and Type B reader/writers, only a single Type 4 tag is emulated within the example firmware. The tag is structured to support multiple applications, and multiple files within each application, which provides a flexible platform to emulate both tag types. Figure 8 shows a structural overview of the emulated tag that is in the firmware.

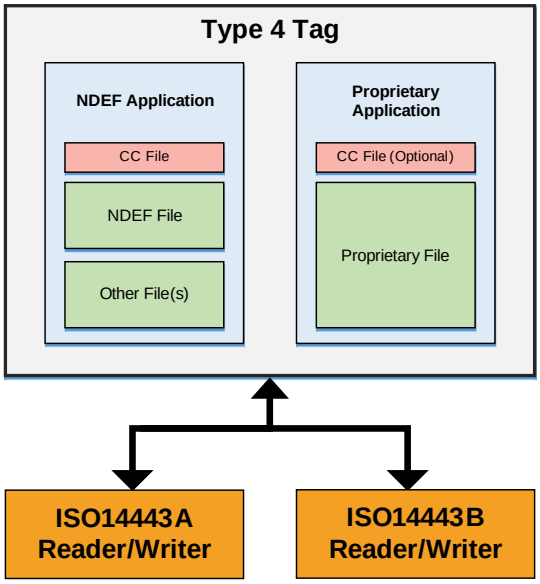


Figure 8. Type 4 Tag Structure

3.1.1 Tag

A Type 4 tag must be compliant with the ISO 14443 specifications for communication, and it receives APDU commands based on the ISO 7816-4 specifications. The ISO 7816-4 commands that have been implemented within the example firmware are Select, Read Binary, and Update Binary. These are the only three required by the NFC Forum specifications for Type 4 tag platforms. The firmware could also be modified to add additional commands that may be needed for other applications, such as contactless payments.

3.1.2 Applications

Each Type 4 tag can have different applications. There are two commons applications that are used: NDEF and Proprietary. It is possible to have multiple applications within a single tag; however for most use cases only one application is used inside each tag. In this firmware a single NDEF application is implemented.

3.1.3 Files

Within NDEF applications, three different file types exist:

1. **Capability Container (CC) File**

   The CC File is required within each NDEF application. The CC file includes the file IDs for the other files within the application as well as information such as the read/write privileges and maximum file size.

2. **NDEF File**

   The NDEF file is required for all NDEF applications. There can only be one NDEF file, and it must have a specific file ID that is in accordance with NFC Forum specifications. The NDEF file is optional for proprietary applications, but any proprietary application that does not use an NDEF file does not conform to the NFC Forum specifications.

3. **Proprietary Files**

   The proprietary files are optional, and they are used in proprietary applications to do specific functions that do not meet the normal NDEF specifications.

The amount of data contained within the files cannot exceed 64kB, and typically the limit is lower than that due to the memory constraints of the hardware (the MCU memory).

3.2 Firmware Structure

Within the example firmware there are pre-defined structures for the tag, applications, and files. The structures (see Example 1) are designed to be easy to use and simplify the process of setting up a tag. This section provides an overview of the structures and how they correspond to the structural overview of the Type 4 tag.

Example 1. Type 4 Tag Structures Within Example Firmware

```
typedef struct
{
    uint16_t ui16Type4FileId;
    uint8_t * pui8Type4File;
    uint16_t ui16Type4FileLen;
    bool bReadOnly;
}tType4File;

typedef struct
{
    uint8_t * pui8AppId;
    uint8_t ui8AppIdLen;
    tType4File * pui8Type4FileArray;
    uint8_t ui8Type4FileLen;
}tType4App;

typedef struct
{
    tType4App * sType4AppArray;
    uint8_t ui8AppArrayLen;
}tType4AppDS;
```

3.2.1 tType4AppDS

This structure represents the entire Type 4 tag (see Figure 8), and it contains the following two parameters:

- **sType4AppArray** – An array of Type 4 applications, corresponding to the N different applications contained within the Type 4 tag.

  Typically this should be set as an array of length N with the correct structure type, and each element of the array should point to one of the applications.
- **ui8AppArrayLen** – The total number of different applications contained within the Type 4 tag. If an array is used as described above, this value is equivalent to the length N of that array.

3.2.2 tType4App

This structure defines the parameters for each different application that is contained within the tag (see Figure 8). It contains the following four parameters:

- **pui8AppId** – A pointer to the array that contains the application ID for the specific application

  The application ID for a NDEF application must be set to 0xD2760000850101.

  The application ID for other applications may be up to 16 bytes long.
- **ui8AppIdLen** – The length of the application ID.
- **pui8Type4FileArray** – A pointer to the array of different files contained within the specific application.

  This should be set as an array that contains at least two elements because the CC file is mandatory for all Type 4 tags. The total number of elements should be equal to the number of different files contained within the specific application. Each array element should point to a different file, and the order of the files does not matter.
- **ui8Type4FileLen** – The total number of different files contained within the specific application.

3.2.3 tType4File

This structure defines the parameters for each individual file within the specified application (see Figure 8). It contains the following four parameters:

- **ui16Type4FileId** – This is a pointer to the array that contains the file ID number for the specific file. The file ID number is always 2 bytes long.

  The file ID for the CC file must be set to 0xE103 to be in compliance with the NFC Forum specifications. All other files must use values that are not reserved for use by the NFC Forum. See the NFC Forum Type 4 Tag Operation Specification 3.0 for details on the available file ID numbers.
- **pui8Type4File** – This is a pointer to the array that contains the file data.
- **ui16Type4FileLen** – This value is the length of the file data array for the specific file.
- **bReadOnly** – This flag determines whether or not the file is set as a read only file. When bReadOnly is set to true, the tag file cannot be written to by any device and the tag is locked in read only mode until the chip is programmed.

  The read/write privileges of the file must match the privileges set inside of the capability container file. The write privilege values between the CC and the tType4File structure are independent of each other. A mismatch between the values could lead to unintentional file updates.

  It is important to ensure that the write privileges are correctly set for code that is loaded into flash memory. If the write privileges are not disabled then writers such as handsets may attempt to overwrite the tag data.

  The bReadOnly flag is also useful to ensure that tag data can be modified only by the application. For example, if an application is measuring a value such as temperature and updating the tag with that information then using the bReadOnly would prevent a handset from accidently overwriting the measurement.

3.2.4 Example Setup

To have a clear picture of how these structures should be used to set up a Type 4 tag, refer to Example 2 for an example setup of a properly implemented Type 4 tag structure. This structure has been implemented in the ce_t4t_config.c file.

Example 2. Example Setup of a Type 4 Tag Structure

```
// NDEF Application ID
uint8_t pui8NdefAppId[7] = {0xD2, 0x76, 0x00, 0x00, 0x85, 0x01, 0x01};

// Test Application # 1 - Example
uint8_t pui8TestAppId[9] = {0xA0, 0x00, 0x00, 0x03, 0x08, 0x00, 0x00, 0x10, 0x00};

// Buffer including all the Type 4 Files
tType4File psType4Buffer[3];

// Capability Container File (0xE103)
tType4File sCCFile = {0xE103, pui8CCBuffer, 23,false};

// NDEF File (0xE104)
tType4File sNdefFile = {0xE104, (uint8_t *) pui8NDefBuffer, 500, false};

// Proprietary File # 1 (0xE105)
tType4File sProp1File = {0xE105, pui8PropBuffer, 82,false};

//NDEF application
tType4App sNdefApp = {pui8NdefAppId, 7, psType4Buffer, 3};

// Test Application # 1 - Example
tType4App sTestApp = {pui8TestAppId, 9, 0, 0};

// Array of length N=2 of type 'tType4App'
tType4App psAppBuffer[2];
```

Example 2. Example Setup of a Type 4 Tag Structure (continued)

```
// Type 4 Application Data Structure including all the applications
tType4AppDS sType4AppDS = {psAppBuffer, 2};
```

3.3 File Structure

3.3.1 Capability Container

In order for an NFC device to read or write the data of the emulated tag it needs to fetch the Capability Container (CC) file. The CC contains information such as the file IDs of all the other files included within the application and the read/write privileges of each file. It also contains the maximum data size for each file to determine how many bytes of space are available for writing. To get that information, the reader/writer needs to access the NDEF or Proprietary File Control Type Length Value (TLV). The File Control TLV must be included for every file within the CC to be accessed by an NFC compliant device.

The format for the Capability Container is specified in the NFC Forum Type 4 Tag Operation Specification 3.0.

1. **CC Length** – This 2 byte value is determined by the number of different files contained within the tag. The formula to calculate the CC Length is: [7 + (#Files * 8)].
2. **Mapping Version** – This value tells the reader the mapping specification used by the tag. The firmware provided supports Version 2.0.
3. **MLe** – These 2 bytes indicate the maximum number of bytes that can be read from the tag by a single Read Binary command. For the provided firmware, this value should be set to 0xFB to maximize the amount of data that can be read. That value accounts for the first three data bytes sent in a Read Binary command, including the optional CID.
4. **MLc** – These 2 bytes indicate the maximum number of bytes that can be written to the tag by a single Update Binary command. For the provided firmware, this value should be set to 0xF9 to maximize the amount of data that can be written. That value accounts for the first six data bytes sent in an Update Binary command, including the optional CID.
5. **File Control TLV** – The TLV blocks provide the reader with information about the tag data.
    1. **Type Field (T)** – This field contains the file type information of the TLV block. The following values may be used:
        1. 0x04 – NDEF File
        2. 0x05 – Proprietary File
        3. 0x06 – Extended NDEF file
    2. **Length Field (L)** – This field contains the length of the value field of the TLV block
    3. **Value Field (V)** – This field contains the file ID, the maximum file size, and the read/write access conditions of the file.
        1. File ID – Each file within the tag should have its own unique file ID number.
        2. Max File Size – This value is not the size of the data itself, but the size of the maximum possible data that can be stored within the file.
        3. Read Access (R) – Determines the read access properties of the file. This should be set to 0x00 to ensure the file has full read access. All other values are for limited read access.
        4. Write Access (W) – Determines the write access properties of the file. This value must be set so that the write privileges of the CC matches the write privileges specified by bReadOnly flag of the file. The value 0x00 allows for full write privileges. The value 0xFF disables all write privileges and makes the file read-only. All other values are for limited write access.

There must be a File Control TLV for each different file contained within the tag. Therefore, if there are three different files, there must be three TLVs that correspond to the settings of each file.

Example 3. Format of the Capability Container for a Tag With Two Files

```
uint8_t pui8CCBuffer[23] = {
    0x00, 0x17,     // CC Len - 7 (fixed) + #Files * 8 (i.e. 2 files, 7+2*8=23)
                    // This CC is set up for 2 Files (1 NDEF 1 Proprietary)
    0x20,           // MAP Ver 2.0
    0x00, 0xFB,     // MLe
    0x00, 0xF9,     // MLc

    0x04,           // T (NDEF File)          TLV for NDEF File
    0x06,           // L
    0xE1, 0x04,     // File ID
    0x01, 0xF4,     // Max NDEF - 500 bytes
    0x00,           // R
    0x00,           // W - 0x00 (write capability available), 0xFF (read-only)

    0x05,           // T (Proprietary File)   TLV for Proprietary File
    0x06,           // L
    0xE1, 0x05,     // File ID
    0x00, 0xFF,     // Max NDEF
    0x00,           // R
    0x00            // W - 0x00 (write capability available), 0xFF (read-only)
};
```

3.3.2 Text RTD

The Text RTD type is a simple format that can be used for many purposes. Some examples include transmitting the results from a sensor measurement or sending a promotional message about a product.

The format for the Text RTD type is discussed in the NFC Text Record Type Definition (RTD) Technical Specification. See Example 4 for an example of a properly formatted Text RTD.

Example 4. Example of a Text RTD Within an NDEF File

```
// NDEF File (read-only) NFC Powered by Texas Instruments
uint8_t pui8NDefBuffer[500] =
{
    0x00, 0x2E,     // These two bytes are excluded from the File Length count
    // File Header
    0xD1,           // NDEF Header
    0x01,           // Length of the record name
    0x2A,           // Length of the payload data
    0x54,           // Binary Encoding of record name - 0x54 (Text RTD)
    //Payload
    0x02,           // Status Byte - UTF-8, two byte language code
    0x65, 0x6E,     // Language Code - English
    0x4E, 0x46, 0x43, 0x20, 0x2D, 0x20, 0x50, 0x6F, 0x77, 0x65,
    0x72, 0x65, 0x64, 0x20, 0x62, 0x79, 0x20, 0x54, 0x65, 0x78,
    0x61, 0x73, 0x20, 0x49, 0x6E, 0x73, 0x74, 0x72, 0x75, 0x6D,
    0x65, 0x6E, 0x74, 0x73, 0x20, 0x49, 0x6E, 0x63, 0x2E
};
```

3.3.3 URI RTD

The Uniform Resource Identifier (URI) RTD type is used to send information such as URLs or phone numbers. It also is used by other NFC RTD types such as Smart Posters.

The format for the URI RTD type is discussed in the NFC URI Record Type Definition (RTD) Technical Specification. See Example 5 for an example of a properly formatted URI RTD.

Example 5. Example of a URI RTD Within a Proprietary File

```
// Proprietary File # 1 - http://www.ti.com/tool/DLP-7970ABP
uint8_t pui8PropBuffer[43] = {
       // File Length
       0x00, 0x29,    // These two bytes are excluded from the File Length count.
       // File Header
       0xD1,         // NDEF Header
       0x01,         // Length of record name
       0x25,         // Length of the payload data
       0x55,         // Binary encoding of record name - 0x55 (URI RTD)
       // Payload
       0x01,         // URI Identifier code - 0x01 = http://www.
       0x74, 0x69, 0x2e, 0x63, 0x6f, 0x6d, 0x2f, 0x74, 0x6f, 0x6f,
       0x6c, 0x2f, 0x44, 0x4c, 0x50, 0x2d, 0x37, 0x39, 0x37, 0x30,
       0x41, 0x42, 0x50
};
```

3.3.4 Smart Poster

The Smart Poster RTD type is one of the more commonly used formats for card emulation. Smart Posters can contain actions within them that would cause a device to react in certain ways when the emulated tag is presented. For example, a Smart Poster that contains a URI could open an application and go to the website when it is read by an NFC enabled device.

The format for the Smart Poster RTD type is discussed in the NFC Smart Poster Record Type Definition (RTD) Technical Specification.

3.3.5 V-Card

The V-Card RTD type is used as an effective method to transfer contact information between multiple parties. A V-Card is typically contains basic information such as the name, e-mail address, and contact phone number of an individual. V-Cards can also provide more detailed information including job titles, websites, and street addresses.

The format for a V-Card is discussed in the RFC6350 specifications.

3.3.6 MIME

The MIME (Multipurpose Internet Mail Extensions) RTD type is a used to represent data that is based on content types. Some of the content types for MIME are: application, image, message, text, and video. For tag emulation, the most common use cases are MIME image types to display an image or logo, and MIME application types to handle *Bluetooth*® or Wi-Fi® handovers.

The format for a MIME is discussed in the RFC2045 specifications.

3.4 Available Type 4 Tag Commands

The example firmware that is provided with this application note supports three Type 4 tag commands: Select, Read Binary, and Write Binary. These commands are all that is required to read tags and to modify the content contained within the tags. This section describes the implementation of three commands in the example firmware.

If the implementation of other commands is required, then they would need to be added to the ISO7816_4_processReceivedRequest function that is contained within the iso_7816_4.c file. Any additional commands that are implemented that way should follow the ISO 7816-4 specifications.

3.4.1 Frame Format

This section describes the frame format bytes for the Type 4 tag platform commands that are sent within the data exchange layer. Understanding what each byte represents is important to conceptually understand how a reader/writer communicates with a Type 4 tag platform.

PCB	CLA	INS	P1	P2	Lc (optional)	Data Bytes (optional)	Le (optional)

Figure 9. Frame Format For Type 4 Tag Commands in Data Exchange Layer

1. **PCB** – Protocol Control Byte, this byte is used to transfer format information about each PDU block. For further details see the ISO 14443-4 specifications.
2. **CLA** – Class Byte, this byte indicates the class of the command. For NFC Compliment tag platforms, this value is always set to 0x00. For further details see the ISO 7816-4 specifications.
3. **INS** – Instruction Byte, this byte specifies the command being sent.
4. **P1, P2** – Parameter Bytes 1 and 2, these bytes are used to send additional information about the command that is being sent.
5. **Lc** – Data Field Length, this byte signifies how many Data Bytes will be sent out by the command.
6. **Data Bytes** – This field contains all of the Data Bytes for the command.
7. **Le** – Expected Response Length, this byte signifies the maximum number of Data Bytes should be sent back in the response. The tag cannot send back more bytes than Le specifies, but it is permitted to send back less bytes.

3.4.2 Select

To read the data from an emulated Type 4 tag an NFC reader/writer must first select an application within the tag and then select the files within the application by using the Select command.

To select an application the ID number of application must be sent by the NFC reader/writer. If the ID number matches an application contained within the tag then a R-PDU is sent back in reply to confirm the selection of the file. If the ID number does not match an error code 0x6A82 is sent back instead to inform the device the application it requested was not found.

A file can only be selected by an NFC reader/writer after an application has been selected. The process to select a file is the same as selecting an application. The file ID number must be sent, and if file with that ID is in the application, a reply is sent to confirm the selection. Otherwise, the error code 0x6A82 is sent back. All NFC devices should search for the CC file first (file ID = 0xE103) to get the information about all of the other files available within the application.

The firmware only supports two Select commands; by name and by file identifier.

When using the Select command, the Lc byte shall be set to the length of the application or file ID number, and the Data Bytes shall contain the ID number itself. The Le byte shall not be included.

3.4.3 Read Binary

After a file has been selected by an NFC device, the device can issue a Read Binary command to read the data contained within the file. If a file has not been selected when a Read Binary is issued, then the error code 0x6982 is sent back to the NFC device.

The Read Binary shall contain the Le byte to tell the tag how many bytes of data are to be read. If the Le byte of the Read Binary exceeds the length of the file data, then the error code 0x6A86 is sent back to the NFC device. A Read Binary command shall not have the Lc byte, or any data bytes.

3.4.4 Update Binary

The Update Binary command can only be issued when a file has been selected by an NFC device. If a file has not been selected when an Update Binary is issued, then the error code 0x6982 is sent back to the NFC device. In order for a file to be modified by the Update Binary command the write privileges of the file need to be checked. If the NFC device does not have the privileges to write the file, the error code 0x6A86 is returned.

The Update Binary contains the Lc byte, which is equal to the length of the Data Bytes being sent. The Data Byte fields contain the data to be written into the tag. The Update Binary command shall not have the Le byte.

If the NFC device has the proper write privileges, the Lc byte is then checked with the maximum file size of the selected file. If the length exceeds the maximum file size, then the error code 0x6A86 is returned. Otherwise, the data within the file is overwritten with the Data Field bytes sent by the NFC device.

A special case of the Update Binary commands occurs when an NFC device attempts to overwrite the CC file. The only byte that is allowed to be updated in the CC file is the write privileges of the file. To update the write privilege, the Lc byte should be set to 0x01 and the only Data Field byte should be the new write privilege for the CC file.

3.5 Modifying Stored Tag Information

The easiest way to modify the contents of a tag with the provided example firmware is to use an NFC enabled handset to re-write the tag with a new message. Many NFC handsets have applications that can be used to write custom messages and information to be written into a tag. Because the example firmware can work with both Type 4A and 4B writers, it does not matter which type is used to write the tag and which type is used to read it.

If it is necessary to modify the contents of a tag that will be placed in flash memory or that must have the correct information upon power-up, then the ce_t4t_config.c file needs to be modified.

If modifications must be made to the file for such reasons, then the CC file, the file length and format information of the modified files, and payload contents all need to be changed. It may be necessary to re-size the buffers as well, and that would require additional changes to the defined file structures in regards to the maximum size of each buffer.

4 Hardware Description

4.1 LaunchPad™ Development Kit and BoosterPack™ Plug-in Module Setup

4.1.1 BoosterPack Plug-in Module: DLP-7970ABP

The third party provider DLP Design NFC/RFID BoosterPack™ plug-in module (DLP-7970ABP) is an add-on board designed to fit all of TI's MCU LaunchPad™ development kits. This BoosterPack plug-in module allows the software application developer to get familiar with the functionalities of TRF7970A multi-protocol fully integrated 13.56-MHz NFC/HF RFID IC on their TI embedded microcontroller platform of choice without having to worry about designing the RF section (see Figure 10 and Figure 11).

The TRF7970A device is an integrated analog front end and data-framing device for a 13.56-MHz NFC/HF RFID system. Built-in programming options make the device suitable for a wide range of applications for proximity and vicinity identification systems. The device can perform in one of three modes: reader/writer, peer-to-peer, or card emulation mode. Built-in user-configurable programming registers allows fine tuning of various reader parameters as needed.

Link for purchase: https://store.ti.com/dlp-7970abp.aspx

4.1.2 LaunchPad Development Kit: MSP-EXP430F5529LP

The MSP-EXP430F5529LP LaunchPad development kit is an easy-to-use evaluation module for the MSP430F5529 USB microcontroller. It contains everything needed to start developing, including on-board emulation for programming and debugging, as well as on-board buttons and LEDs for quickly adding a simple user interface. Rapid prototyping is a snap, thanks to 40-pin access headers and a wide range of BoosterPack plug-in modules. This enables technologies such as wireless, display drivers, temperature sensing, and much more (see Figure 10).

Link for purchase: https://store.ti.com/msp-exp430f5529lp.aspx

Figure 10. MSP430F5529 LaunchPad Development Kit and DLP-7970ABP BoosterPack Plug-in Module

4.1.3 LaunchPad Development Kit: MSP-EXP432P401R

The MSP432P401R LaunchPad development kit enables you to develop high-performance applications that benefit from low-power operation. It features the MSP432P401R – which includes a 48-MHz Arm® Cortex®-M4F, 95-µA/MHz active power, and 850-nA RTC operation, a 14-bit 1-MSPS differential SAR ADC, and an AES256 accelerator.

This LaunchPad development kit includes an on-board emulator with EnergyTrace+ technology, which means you can program and debug your projects without the need for additional tools, while also measuring total system energy consumption (see Figure 11).

Link for purchase: https://store.ti.com/msp-exp432p401r.aspx

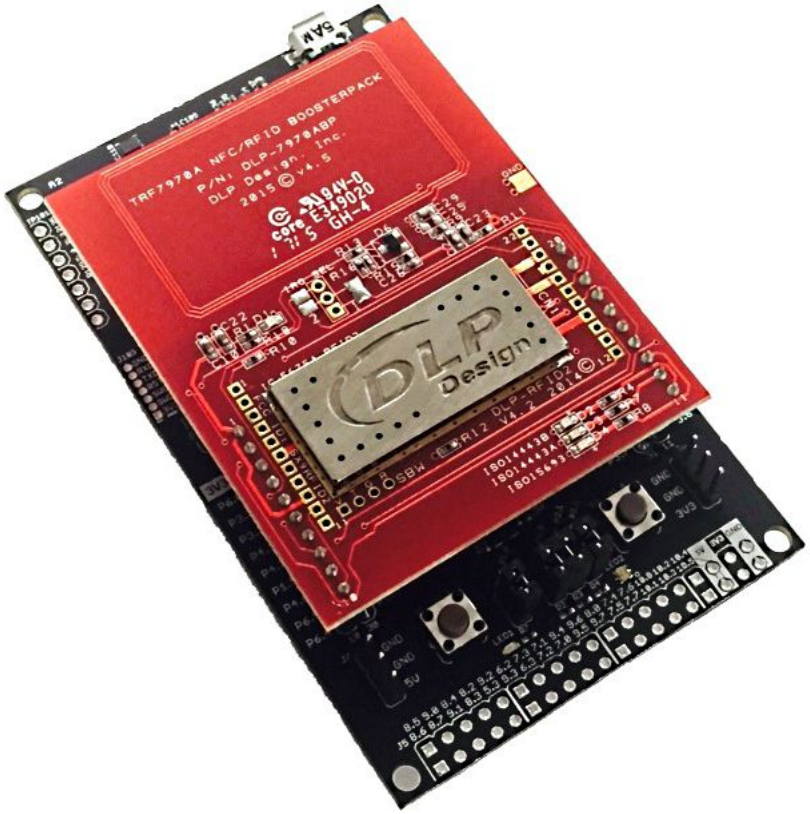


Figure 11. MSP432P401R LaunchPad Development Kit and DLP-7970ABP BoosterPack Plug-in Module

4.2 Bundle Available for Purchase

The TI store offers this bundle:

MSP-EXP430F5529LP and DLP-7970ABP

5 Card Emulation Firmware Example

This section explains which APIs are used by the NFC/RFID layer (see Figure 12) to initialize and handle the card emulation modes. Furthermore, it explains how to setup a basic card emulation application that enables both Type A and Type B modes at the same time.

The firmware example that contains the reader/writer APIs discussed in this document can be downloaded from www.ti.com/lit/zip/sloa208.

As downloaded, the firmware example includes the full TI NFC stack, which supports peer-to-peer, card emulation, and reader/writer modes. For applications that do not require all NFC operating modes, there are configuration options available to reduce the NFC stack memory footprint (only compiling required operating modes). These configurations can be made by modifying the #define statements within the *nfc_config.h* file, located at [Install Path]\nfclink\Source\headers.

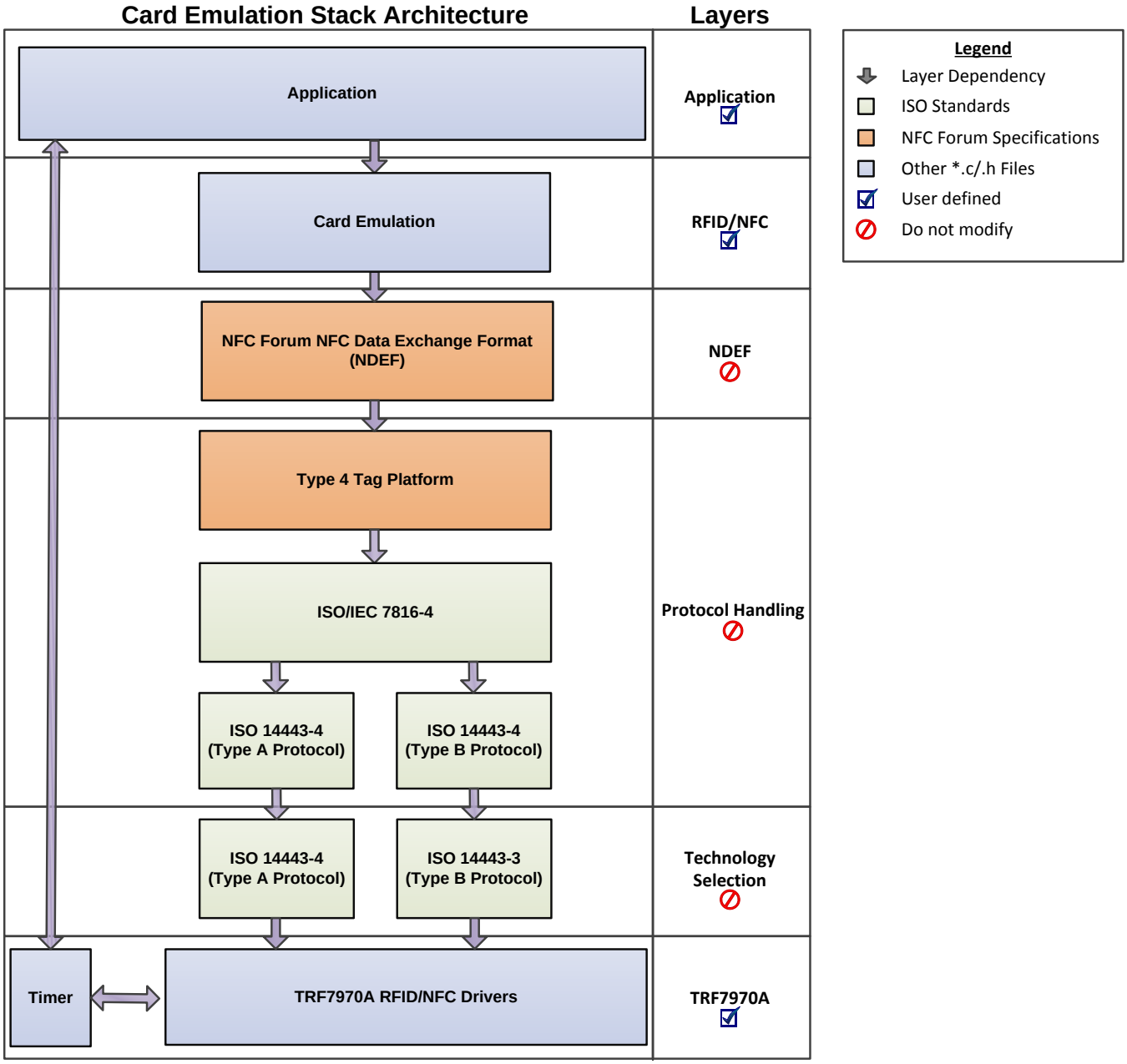


Figure 12. Card Emulation NFC Stack Architecture

5.1 Card Emulation APIs

For details on all available APIs used in the provided example firmware for NFC card emulation mode, see the *NFCLink Standalone Software Library API Guide* included in the install package. The guide is located in [Install Path]\doc.

The *NFCLink Standalone Software Library API Guide* describes the flow of the software stack, all APIs that are available for NFC card emulation functionality, and each function to help users with developing custom NFC card emulation applications.

5.2 Implementing a Card Emulation Sample Application

This section explains how to implement a Card Emulation sample application that uses buttons S1 and S2 on the MSP430F5529 LaunchPad development kit to select different RTD messages for the firmware to emulate. Table 2 and Table 3 lists the connections between the MSP430F5529 and the TRF7970A for the different MSP430F5529 evaluation platforms. Table 4 shows the connections between the MSP432P401R and the TRF7970A for the MSP432P401R LaunchPad development kit.

Table 2. DLP-7970ABP BoosterPack Plug-in Module and MSP-EXP430F5529LP Hardware Connections

DLP-7970ABP Pins	MSP430F5529 LaunchPad Development Kit Pins
TRF7970A EN1	P4.1
TRF7970A IRQ	P2.2(1)
MOSI	P3.0
MISO	P3.1
CLK	P3.2
Slave Select	P4.2
I/O_2	P6.6(2)
I/O_3	P2.0(2)
I/O_5	P1.6(2)

(1) IRQ is defaulted to P2.2 for DLP-7970ABP v4.5 and newer (see the *DLP-7970ABP hardware update overview*).

(2) Pin is only needed for using Special Direct mode

Table 3. TRF7970ATB and MSP-EXP430F5529 Experimenter Board Hardware Connections

TRF7970ATB Pins	MSP430F5529 Experimenter Board Pins
TRF7970A EN1	P2.3
TRF7970A IRQ	P2.0(1)
MOSI	P3.0
MISO	P3.1
CLK	P3.2
Slave Select	P2.6
MOD	P2.1
ASK/OOK	P4.7

(1) Requires a jumper to be placed between P2.0 and P4.0 on the Experimenter Board.

Table 4. DLP-7970ABP and MSP-EXP432P401R LaunchPad Development Kit Hardware Connections

DLP-7970ABP Pins	MSP432P401R LaunchPad Development Kit Pins
TRF7970A EN1	P6.2
TRF7970A IRQ	P3.0[(1)]
MOSI	P1.6
MISO	P1.7
CLK	P1.5
Slave Select	P6.5
I/O_2	P4.3[(2)]
I/O_3	P2.5[(2)]
I/O_5	P4.1[(2)]

(1) IRQ defaults to P3.0 for DLP-7970ABP v4.5 and newer (see the *DLP-7970ABP hardware update overview*).

(2) Pin is needed only for using Special Direct mode.

5.2.1 Low-Level Initialization

For the low level initialization the MCU is initialized in MCU_init() by setting the MSP430F5529 main clock frequency to 25 MHz. The TRF7970A hardware connections and the MSP430F5529 SPI module (SPI clock running at 4 MHz; minimum recommended is 2 MHz) is initialized in TRF79x0_init(). The variables in Example 6 are needed for the COM port transmit and receive and the card emulation stack initialization. The Buttons_init function set the GPIO direction to inputs and the Buttons_interruptEnable() function enables the interrupt for buttons S1 and S2.

Example 6. MCU and TRF7970A Initialization Code Snippet

```
#include "msp430.h"
#include "nfc_controller.h"
#include "ndef_image.h"
#include "lp_buttons.h"

// Card Emulation Current Mode
t_sNfcCEMode sCESupportedModes;

void main(void)
{
    tNfcState eTempNFCState;
    tNfcState eCurrentNFCState;
    char pcBytesReceivedString[5];

    // CE Variables
    t_sNfcCEMode sCEMode;

    // Initialize MCU
    MCU_init();

    //Enable interrupts globally
    __enable_interrupt();

    // Initialize USB serial port
    Serial_init();

    // Initialize TRF7970
    TRF79x0_init();

    // Initialize external push buttons
    Buttons_init(BUTTON_ALL);
```

Example 6. MCU and TRF7970A Initialization Code Snippet (continued)

```
    Buttons_interruptEnable(BUTTON_ALL);

    // Initialize TRF7970A idle mode
    TRF79x0_idleMode();

    // Initialize the NFC controller
    NFC_init();

    // This function will configure all the settings for each protocol
    NFC_configuration();

    // Initialize Type 4 Tag RTD Message
    T4T_CE_initNDEF();

    // Initialize IDs for NFC-A, NFC-B and NFC-F
    NFC_initIDs();
```

5.2.2 Card Emulation NFC Stack Setup

The card emulation NFC stack is initialized by setting the bT4TAEnabled or bT4TBEnabled bits inside the g_sCESupportedModes variable (see Example 7). If both are enabled then the firmware can emulate both Type 4 tag platforms. For this demo, both tag platforms are enabled.

Example 7. Card Emulation Initialization Code

```
    // Enable Card Emulation Supported Modes
    g_sCESupportedModes.bits.bT4TAEnabled = 1;
    g_sCESupportedModes.bits.bT4TBEnabled = 1;

    NFC_CE_configure(g_sCESupportedModes);
```

5.2.3 Emulation of Different RTDs

The NFC_run function sets the transceiver to listening mode, and it waits for a polling command. When a reader/writer is presented, it goes through the protocol activation and anticollision procedures to establish a connection. After the tag is selected, the state machine is in the NFC_DATA_EXCHANGE_PROTOCOL state, and the firmware emulates a tag based on the selected RTD type (default is Text RTD).

When no NFC device is present, the firmware checks if button S1 or button S2 has been pressed. When button S1 is pressed, the firmware cycles between the emulation of five RTD types: Text, URI, Smart Poster, V-Card, and MIME. The MIME RTD is an example of a *Bluetooth* handover application.. When button S2 is pressed, the emulated RTD becomes a stored MIME image. The MIME image is 3597 bytes, and it can be overwritten by NFC enabled reader/writers to hold any NDEF-compatible RTD that fits within the 3597-byte size constraint. The RTD that is emulated on button S2 is restored to the original MIME image when the board is reset or power is cycled. Figure 13 shows the process.

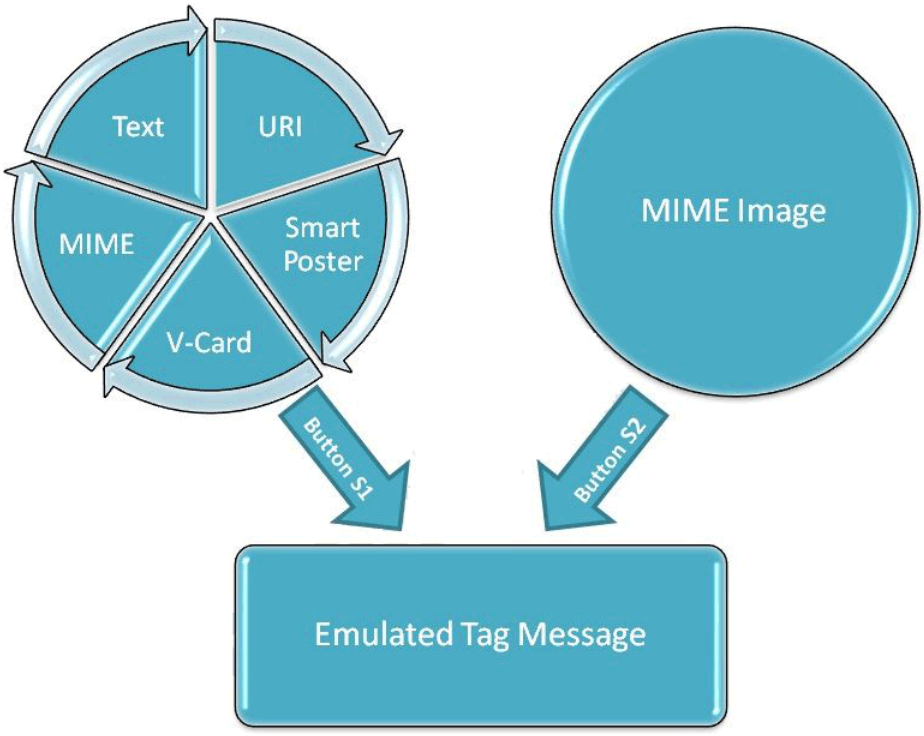


Figure 13. Toggling Between RTD Types

6 Quick Start Guide

The *NFCLink Standalone getting started guide* provides complete details of how to get started with the provided example firmware and TI hardware.

This guide describes how to load the example firmware to TI evaluation boards and explains the features of the TI NFC Tool GUI (see Figure 14), which is installed with the firmware package.

The TI NFC Tool allows for quick configuration of the different NFC modes and provides an interface to send and receive data with NFC-enabled devices.

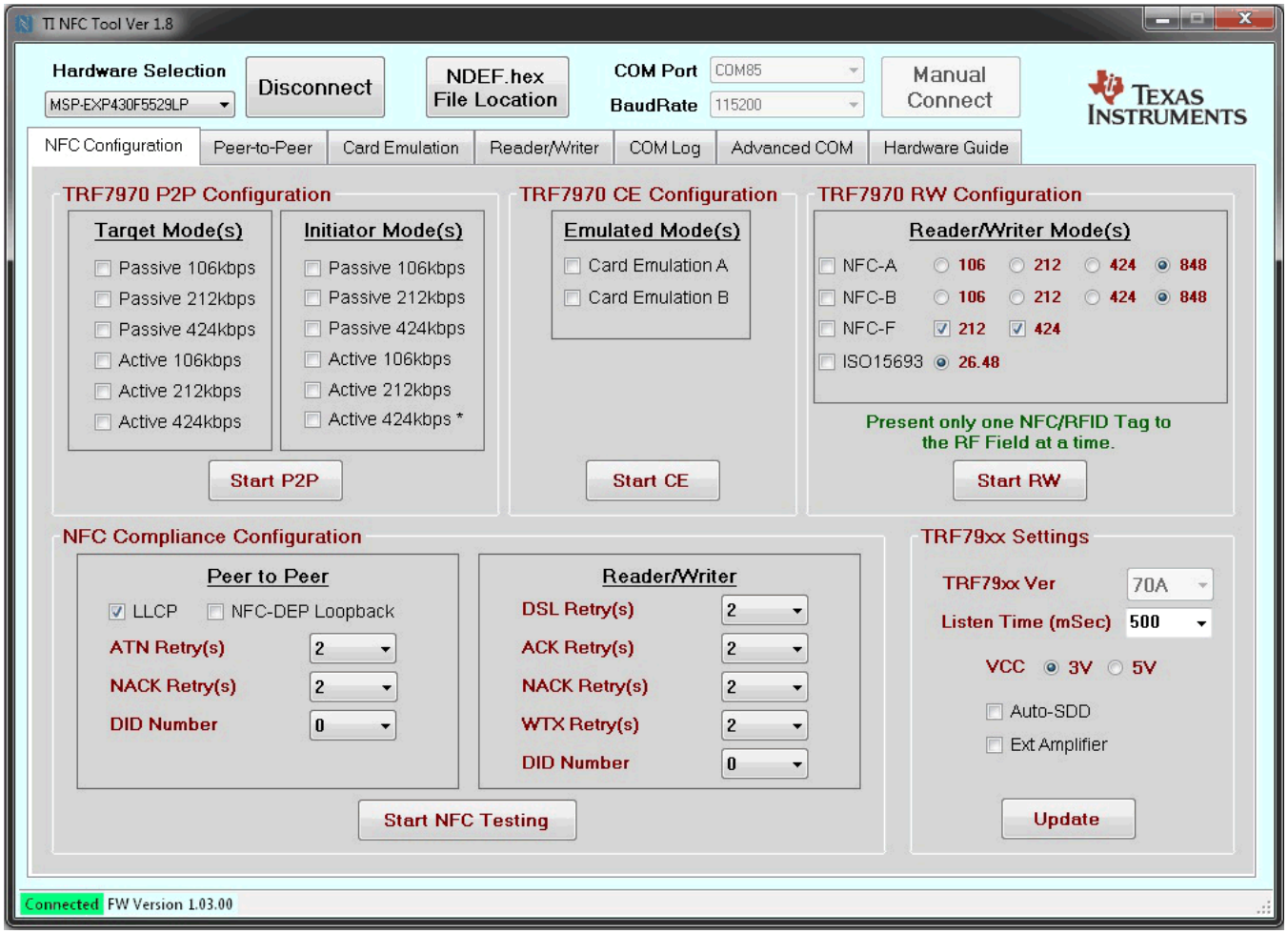


Figure 14. TI NFC Tool GUI

7 Operational Overview

The card emulation example on the MSP430F5529 supports the emulation of Type 4A and Type 4B tag platforms. When emulating a tag, the TRF7970A transceiver is a listening device. It waits for a RF field to be presented and for the correct over-the-air commands to be transmitted. If additional NFC modes are enabled that allow the TRF7970A to act as a polling device, then the transceiver needs to toggle between polling and listening states (see Figure 15).

A function is used to run the NFC stack that handles all states of communication from technology detection to the data exchange protocol. When the NFC stack function is called it evaluates the enabled modes and calls the appropriate state machines. For card emulation, the NFC stack uses the target state machine, enters a listening mode, and then the firmware waits for a technology to be activated for 500 ms. If no technology is activated during that time, the NFC stack returns back to the main code. Note that this behavior would differ if a polling mode was enabled. In that case, the NFC stack would either switch to a polling mode instead before returning to the main code, or start by polling and then listen for the 500 ms. The interval of 500 ms is used to provide a fluid user experience and can be increased or decreased as needed on an application basis.

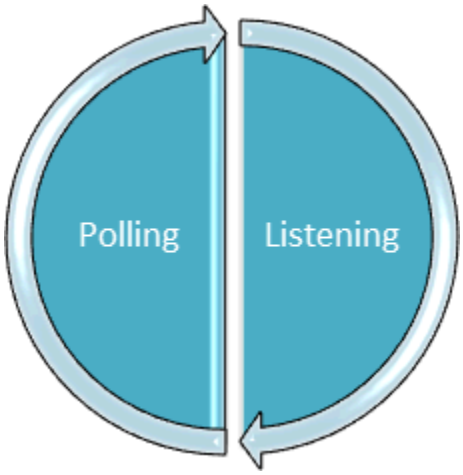


Figure 15. NFC Initiator and Target Switching Mechanism

The second mode requires a host (PC) to run the Embedded RF Tool GUI and connect to the MSP430F5529 LaunchPad development kit through the USB CDC (Figure 16 shows the system block diagram). The GUI lets the user select the card emulation modes to enable or disable.

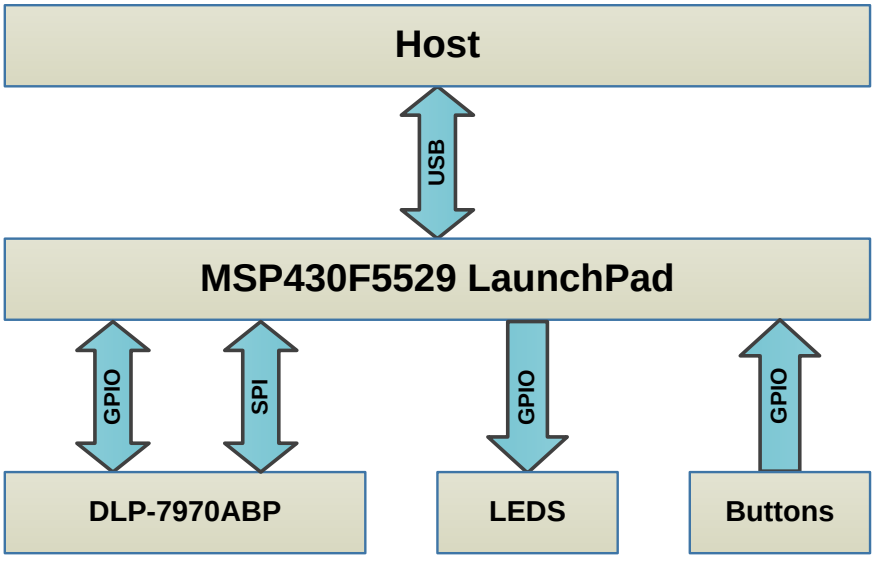


Figure 16. Card Emulation Demo System Block Diagram

8 Card Emulation Interoperability Results

This section describes the results of the interoperability between the existing TRF7970A card emulation stack and the list of NFC enabled devices previously mentioned (see Table 1). Use the legend in Table 5 to understand the results in Table 6.

Table 5. Legend For The Result of the NFC Enabled Devices Tests

Legend	
✓	Success
✓1	The NDEF contents of the emulated tag can only be read if a single file is contained within the NDEF application.
✗	The device does not read the emulated tag

Table 6 includes the interoperability results for test cases where the TRF7970A emulates a Type 4A or Type 4B tag platform. The smartphone devices are sorted from oldest to the latest, and the interoperability for the card emulation modes is better for newer NFC enabled devices.

Table 6. TRF7970A Card Emulation and Smart Phone Interoperability Results

Test With Firmware 1.00.14	TRF7970A Modes	
Smartphone Model (Release Date)	**Card Emulation Type A**	**Card Emulation Type B**
Samsung Galaxy Nexus (Nov 2011)	✓1	✓1
Samsung Galaxy S3 (AT&T) (June 2012)	✓1	✓1
Samsung Galaxy S3 (T-Mobile) (June 2012)	✓	✓
Asus Nexus 7 (July 2012)	✓	✓
Samsung Galaxy Note 2 (Sept 2012)	✓	✓
AU Arrows Fujitsu FJL21 (Oct 2012)	✓	✓
Samsung S3 Mini (Oct 2012)	✓	✓
Nokia Lumia 820 (Oct 2012)	✓1	✓1
HP Elite Tablet (Nov 2012)	✓1	✓1
Samsung Nexus 10 (Nov 2012)	✓	✓
Google Nexus 4 (Nov 2012)	✓	✓
Samsung Galaxy S4 (April 2013)	✓	✓
Hisense Sero 7 Pro (June 2013)	✓	✓
Asus Nexus 7 (July 2013)	✓	✓
Google Nexus 5 (Oct 2013)	✓	✓
Samsung Galaxy S5 (April 2014)	✓	✓

Table 7 shows the results for the time it took to send a 20.317kB file from the TRF7970A to the NFC enabled reader/writer devices listed. The start edge was measured from the Transmit Received interrupt of the first READ BINARY command of the NDEF file (sent from the NFC enabled device to the TRF7970A). The end edge was measured at the TX complete interrupt of the last READ BINARY response.

The results show that the throughput is highly related to the performance of the processor in the NFC enabled device. The highest throughput was measured with the Nexus 10, which has the fastest processor. While the results show that Type B is slightly slower than Type A across all devices, it was observed that the RF performance for Type B is more robust and provides a more consistent user experience in comparison to Type A.

Table 7. Data Throughput at 106 kbps for a 20.317kB NDEF

Test With Firmware 1.00.14	Card Emulation Type A		Card Emulation Type B	
Smartphone Model (Release Date)	Send 20.317kB File (seconds)	Throughput at 106 kbps (kBps)	Send 20.317kB File (seconds)	Throughput at 106 kbps (kBps)
Samsung Galaxy Nexus (November 2011)	4.8412	4.208	5.0803	4.010
Samsung Galaxy S3 (AT&T) (June 2012)	7.2580	2.807	7.7903	2.615
Samsung Galaxy S3 (T-Mobile) (June 2012)	8.4155	2.421	8.5383	2.386
Asus Nexus 7 (July 2012)	7.7125	2.641	7.9498	2.562
Samsung Galaxy Note 2 (Sept 2012)	4.4066	4.623	4.8398	4.209
AU Arrows Fujitsu FJL21 (Oct 2012)	2.6749	7.616	2.9011	7.022
Samsung S3 Mini (Oct 2012)	6.3088	3.220	6.6212	3.068
Nokia Lumia 820 (Oct 2012)	3.7173	5.480	4.0130	5.076
HP Elite Tablet (Nov 2012)	3.6028	5.639	3.9071	5.214
Samsung Nexus 10 (Nov 2012)	2.1884	9.309	2.2714	8.968
Google Nexus 4 (Nov 2012)	2.7696	7.355	2.8361	7.186
Samsung Galaxy S4 (April 2013)	6.2131	3.270	6.72564	3.021
Hisense Sero 7 Pro (June 2013)	3.0598	6.640	3.6128	5.624
Asus Nexus 7 (July 2013)	2.6499	7.687	2.8898	7.049
Google Nexus 5 (Oct 2013)	4.2604	4.781	4.5450	4.470
Samsung Galaxy S5 (April 2014)	2.6891	7.555	2.9499	6.887
Sony Xperia Z3 (September 2014)	2.8512	7.126	3.1773	6.394

9 Conclusion

Card emulation is one of the three modes supported by the TRF7970A transceiver. The existing card emulation NFC stack supports the emulation of Type 4A and Type 4B tag platforms at 106 kbps. The TRF7970A only supports 106 kbps for Type 4A/B card emulation. The transceiver is in listening mode when it is emulating a tag. It does not generate its own RF field, but when an NFC-enabled reader/writer is presented, the TRF7970A load modulates the RF field of the NFC device to respond.

The card emulation demo has two modes, a standalone mode and a GUI mode. The standalone mode can emulate both Type 4A and Type 4B tag platforms for six RTD messages: text, URI, smart poster, V-Card, and two MIME types. The GUI mode enables the user to choose which card emulation modes are enabled and customize a text or URL message to emulate (246 bytes maximum).

Based on the tests executed with NFC enabled devices in Table 7, the newer NFC enabled devices have improved throughput.

As downloaded, the firmware example includes the full TI NFC stack, which supports peer-to-peer, card emulation, and reader/writer modes. For applications that do not require all NFC operating modes, there are configuration options available to reduce the NFC stack memory footprint (only compiling required operating modes). These configurations can be made by modifying the #define statements within the *nfc_config.h* file, located at [Install Path]\nfclink\Source\headers.

For more information about NFC peer-to-peer operation, see *NFC active and passive peer-to-peer communication using the TRF7970A*.

For more information about NFC reader/writer, see *NFC/HF RFID reader/writer using the TRF7970A*.

10 References

1. *TRF7970A multiprotocol fully integrated 13.56-MHz RFID and NFC transceiver IC*
2. ISO/IEC 7816-4:2005(E) (http://www.ansi.org)
3. ISO/IEC 14443-3:2009(E) (http://www.ansi.org)
4. ISO/IEC 14443-4:2008(E) (http://www.ansi.org)
5. NFCForum-TS-DigitalProtocol-1.0 (Digital Protocol) (http://www.nfc-forum.org)
6. NFCForum-TS-Activity-1.0 (Activity Protocol) (http://www.nfc-forum.org)
7. NFC Forum T4TOP (Type 4 Tag Operation) (http://www.nfc-forum.org)
8. NFCForum-TS-RTD_Text_1.0 (Text Record Type Definition) (http://www.nfc-forum.org)
9. NFCForum-TS-RTD_URI_1.0 (URI Record Type Definition) (http://www.nfc-forum.org)
10. NFCForum-SmartPoster_RTD_1.0 (Smart Poster Record Type Definition) (http://www.nfc-forum.org/)
11. NFCForum-TS-RTD_1.0 (NFC Record Type Definition (RTD)) (http://www.nfc-forum.org/)
12. NFC Research Lab Hagenberg Android Application (http://www.nfc-research.at/)
13. NFC Forum Logo (http://nfc-forum.org/our-work/nfc-branding/n-mark/the-n-mark-license/)

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from December 10, 2016 to March 13, 2019 **Page**

- Removed former Section 4.2 *Experimenter Board Setup* and Section 4.3 *TRF7970ATB Module* 23
- Updated available bundle and link in Section 4.2, *Bundle Available for Purchase* 23

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265