Masterarbeit

# Design and Implementation of a non-mobile Embedded NFC Interface

Thomas Spiss

Institut für Technische Informatik
Technische Universität Graz
Vorstand: Univ.-Prof. Dipl.-Inform. Dr. sc.ETH Kay Uwe Römer


Begutachter: Ass.-Prof. Dipl.-Ing. Dr. techn. Christian Steger

Betreuer: Ass.-Prof. Dipl.-Ing. Dr. techn. Christian Steger
Dipl.-Ing. (FH) Christian Eisendle

Graz, im April 2014

## STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

..... .....
date (signature)

## Danksagung

Diese Diplomarbeit wurde im Studienjahr 2014 am Institut für Technische Informatik an der Technischen Universität Graz durchgeführt.

An dieser Stelle möchte ich mich bei allen die diese Masterarbeit ermöglicht haben bedanken. Insbesondere bei Prof. Christian Steger (Institut für Technische Informatik, TU Graz) sowie bei Alexander Maili und Christian Eisendle (NXP).

Weiters danke ich Robin Martin, nicht nur für die unzähligen Stunden des Korrekturlesens sondern auch für die humorvollen und dennoch produktiven Kommentare und Anmerkungen. Meinen Studienkollegen Manuel, Michael und Hannes für unsere Lerngruppen und Freundschaft.

Mein ganz besonderer Dank geht an meine Freundin und Lebensmensch Utta, für ihre jahrelange moralische Unterstützung und Geduld.

Graz, im April 2014 Thomas Spiss

## Kurzfassung

Das Ziel dieser Masterarbeit ist der Entwurf und die Implementierung eines Near Field Communication (NFC) basierten Interface für stationäre Geräte. Durch die kurze Kommunikationsdistanz und der somit intuitiven Bedienung eignet sich solch ein Interface sehr gut zur Steuerung von Geräten wie Druckern, Heizungsanlagen oder Haushaltsgeräten mit Hilfe von NFC fähigen Mobiltelefonen. Dazu wird ein direkter Kommunikationskanal zwischen den beiden Geräten benötigt. Um die Integration in bestehende Systeme möglichst einfach zu gestalten, bietet das Interface die Möglichkeit weit verbreitete Kommunikationsprotokolle, wie zum Beispiel TCP/IP, zu verwenden.

Es wurden mehrere Möglichkeiten evaluiert und diese anhand der Kriterien Datenrate, Stromverbrauch, Hardwarekosten, Integrationsaufwand sowie der für den Verbindungsaufbau benötigten Zeit, bewertet. Dabei stellte sich die Integration des zum bidirektionalen Datenaustausch über NFC verwendeten Logical Link Control Protocol (LLCP) in die Firmware eines bestehenden NFC Controller als die erfolgversprechendste Variante heraus.

Dieser Ansatz wurde mit PN547 NFC Controllern von NXP Semiconductors (NXP) umgesetzt und evaluiert. Da das Interface häufig in Linux basierenden Systemen eingesetzt werden könnte, wurden Raspberry Pi Linux Einplatinen-Computer als Hostsysteme verwendet. Um das Ziel der einfachen Integration des NFC Controllers in diese Systeme zu erreichen, wurde ein Linux Kernel Modul, welches das Interface als Netzwerkkarte in den Linux Netzwerkstack integriert, entwickelt.

Zur Evaluierung des entwickelten Interface wurde ein Prototypensystem bestehend aus zwei Raspberry Pi Computern und zwei NFC Interface Controllern realisiert. Durch die Integration des Kommunikationsinterface als Netzwerkkarte konnten die weit verbreiteten Linux Tools ping und iperf zur Bestimmung von Latenz und Datenübertragungsrate verwendet werden. Die bei einer NFC Datenrate von 424 kBit/s ermittelten Ergebnisse zeigten, dass sowohl die durchschnittliche Datenübertragungsrate von bis zu 73,8 kBit/s, als auch die Latenzzeiten von durchschnittlich 14,5 Millisekunden, für die Verwendung des Interface zur Bedienung von Geräten geeignet sind. Dabei können einfache Benutzeroberflächen ohne aufwändige Grafiken und Animationen direkt per NFC übertragen werden. Bei aufwändigen Benutzeroberflächen empfiehlt es sich die Animationen und Grafiken am zur Bedienung genutzten Gerät zu speichern und nur Steuerbefehle und Daten über das NFC Interface zu übertragen.

## Abstract

Near Field Communication (NFC) Technology is known to be very intuitive and user friendly because of its short communication distance. NFC based user interfaces for controlling devices like printers, heaters or white goods with NFC enabled handsets could therefore take advantage of NFC's "touch to interact" concept and improve their usability. Therefore within this thesis a non-mobile embedded NFC interface for peer to peer connections is designed, implemented and evaluated.

The thesis starts by evaluating the design-space for an NFC interface for integration in non-mobile devices, with regard to performance, power, costs and functionality. To ensure easy integration and reliable communication the solutions should support the use of high level communication protocols like TCP/IP. The evaluation has shown that the integration of the LLCP protocol, used to enable bidirectional data exchange over NFC, into an existing NFC controller is the most promising approach. This concept was realized by adapting NXP's PN547 NFC controller.

As many of today's embedded systems are running with Linux based operating systems, Raspberry Pi single board computers, running Linux, have been used as host systems. To provide an easy integration of the NFC controllers into existing appliances a Linux driver, mapping the NFC interface as a generic network card, was developed.

To evaluate the solution a prototype system consisting of two adapted NFC controllers as well as two Raspberry Pi boards was developed. As the NFC controllers have been integrated into the used Linux operating system as network card, well known tools like ping and iperf could be used to measure latency and data throughput of the system. The best results have been achieved using 424 kBit/s as Radio-Frequency (RF) data rate. With this RF data rate average values for latency and data throughput of 14.5 milliseconds and 73,8 kBit/s have been measured. This shows that the concept can be used as an interface for providing lightweight user interfaces over the NFC link. For more advanced user interfaces with large graphics or animations, these should be preloaded to the device used for operating and only control commands and data shall be transferred over the NFC link.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The number of electronic devices surrounding us in our everyday life is huge and still growing. We need to control electronic devices in many situations at home or at work. Some appliances such as heating controls, printers or even white goods offer limited user interfaces. Limited in terms of input and feedback possibilities. Manufacturers try to reduce production costs by keeping the number of input devices, like buttons and switches, low. It is also common to keep the costs for feedback units as low as possible. They favor low-cost small displays or Light-emitting Diodes (LEDs) over easy to read displays of sufficient size. That is why these user interfaces are often not very intuitive and users have to spend a lot of time to learn how to use different user interfaces for a large number of devices.

Therefore controlling these appliances through a common, portable interaction device would be a good approach. The interaction device should provide an intuitive, well known input and feedback mechanism and in best case is always carried by the user. A ideal solution is to use smartphones or tablet computers.

While most smartphones and tablets already provide Bluetooth and Wireless Local Area Network (WLAN) connectivity, the number of NFC enabled devices is still growing. Many smartphone and tablet user interface applications for appliances use WLAN or Bluetooth for communication. These communication techniques both suffer from their need to be paired, in the case of Bluetooth, or connected properly, when using WLAN, by the user.

NFC technology provides a "touch" experience to consumers. The "touch" allows the exchange of a small amount of data that allows a more complex connection like Bluetooth or WLAN to be setup automatically, without any further user interaction. With the ongoing deployment of NFC technology by handset makers, the touch experience can be extended to interface between phones and other consumer devices like printers, heating systems or other NFC enabled sensors. In case of NFC enabled Sensors the sensors could be powered from the NFC field and therefore be fully passive. Such sensors could for example be placed in walls as no maintenance (battery change) is needed.

This master thesis aims to evaluate the design-space for an NFC interface integration within non-mobile devices with regard to performance, power, cost and functionality. Furthermore a prototyping system based on NXP-Chipsets is implemented to evaluate different design options.


Figure 1.1: An Example of a simplified Service Advertisement. [RSP12]

## 1.1 Motivation

In today's consumer market, many companies whose products are equipped with WLAN will also provide smartphone Applications to control them. Often these are only available for the top price models. For most low cost consumer products wireless LAN communication is not an option. An other problem is that connecting multiple devices with WLAN can be a issue for inexperienced users.

Using near field communication can be a solution for these problems. The short communication distance nature of near field communication helps the user to connect to the right device in a very intuitive way. Riekki et al. [RSP12] describes a way to mark appliances with NFC support. As shown in Figure 1.1 a pictogram indicates the spot to touch to initiate an NFC communication. They also investigated the usage of NFC to interact with smart environments with the following result: "NFC excels in interacting with the local environment. There are no other input methods with an equal combination of user control, easiness to use, robustness, and price." [RSP12]

Major factors for the success of NFC as a user interface technique are the low costs, power consumption and development effort to integrate NFC communication into devices.

Several low cost and low power NFC controllers, like the PN547 from NXP Semiconductors (NXP), are available. Reducing the functionality of the NFC controller to fulfill the needs for non-mobile devices can only lead to even lower costs and power consumption.

To keep the development effort to integrate NFC communication into devices low, its important to provide the possibility for end to end communication using well known protocols. One of the most used protocols is TCP/IP [CDS74] [oSC81] but also other protocols such as OBject EXchange (OBEX) [Ass13] shall be supported.

## 1.2 Goals

In this thesis the possibilities for integrating NFC interfaces into non-mobile devices are evaluated. Therefore the following goals are defined:

- Investigate existing solutions for non-mobile NFC interfaces and classify them by:
  - Power consumption,
  - Costs and
  - Development effort to integrate.
- Choose the concept best fitting the criteria mentioned above.
- Evaluate the possibilities to realize the concept with NXP's PN547 NFC controller.
- Adapt the NFC controller's firmware to enable data exchange using high level protocols.
- Build a prototype system, based on NXP's PN547 NFC controller, showing communication over NFC using well known network protocols like TCP/IP.
- Test function and performance of the prototype system.
- Discuss the results and define possibilities for further research.

## 1.3 Structure

Including this introduction the thesis consists of six more chapters. The results of the related work research is summarized in chapter 2. In chapter 3 a comparison of High Level Protocol NFC Communication Possibilities is set out and the best solution for the following implementation phase is chosen. Based on this decision the Systems Architecture is defined. The hardware and software components as well as the concept and details used for the implementation are described in chapter 4. In chapter 5 tests to evaluate the implemented solution are defined and the results are documented. Following this the results are analyzed and discussed. Chapter 6 contains a summary of the thesis and some final thoughts about the results, as well as some for further research.

# Chapter 2

# Related Work

In this chapter a brief first introduction into NFC is given, and then some research projects regarding non-mobile NFC interfaces are summarized. These are projects focusing on the excellent user experience provided by the "touch to interact" concept of NFC, some research about using NFC to simplify the connection setup of other wireless communication channels and projects related to peer to peer communication over NFC.

## 2.1 Near Field Communication

The German book “Anwendungen und Technik von Near Field Communication (NFC)” by Josef Langer and Michael Roland [LR10] describes the technology of NFC. On the following pages the chapters containing information related to this thesis are summarized.

### 2.1.1 General

NFC was developed in 2002 by NXP and Sony. It combines the two vendor specific Radio-Frequency Identification (RFID) systems MIFARE and FeliCa and is fully compatible to the existing 13,56Mhz RFID standards. NFC is designed to work on distances up to 10cm.

Figure 2.1 shows the existing RFID and the new NFC features into the NFC standard. This thesis focuses on the communication between two NFC-Devices, so the grey parts

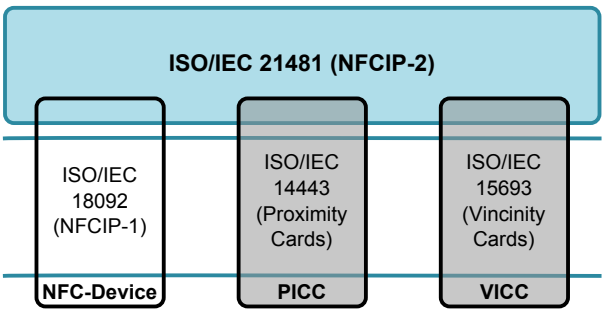


Figure 2.1: Communication modes of the NFC Standard. (Based on [LR10])

(Proximity Cards and Vicinity Cards) are not used and therefore not described further. The NFCIP-1 standard combines MIFARE and FeliCa communication protocols with three

available transfer rates: 106 kbit/s, 212 kbit/s and 424 kbit/s. The 106 kbit/s mode is based on MIFARE, the others on FeliCa.

NFC defines an active and a passive communication mode. In the active mode the initiator as well the target use their own Radio-Frequency (RF) field to communicate. The target responds to initiators commands by modulating the self generated RF field. In passive mode the target responds by using load modulation on the initiators RF field[ECM].

The NFC architecture provides three operating modes. These three modes are shown in figure 2.2. The grayed out modes, Reader / Writer and Card Emulation mode, are

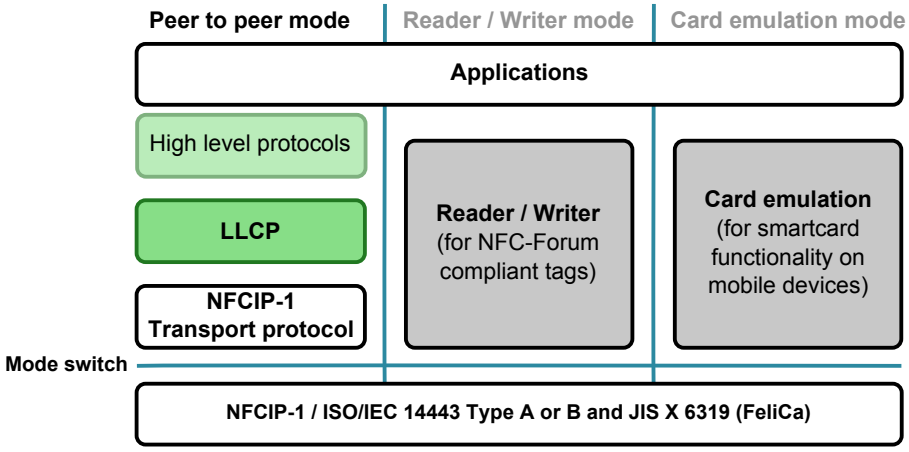


Figure 2.2: NFC operating Modes. (Based on [LR10])

not needed for communication between two NFC devices. The peer to peer mode allows direct communication between two devices. Mainly the green parts, Logical Link Control Protocol (LLCP) and High Level Protocols, will be addressed within this thesis.

### 2.1.2 Peer to Peer Mode

Figure 2.3 shows the protocol stack used in peer to peer mode. The physical layer is

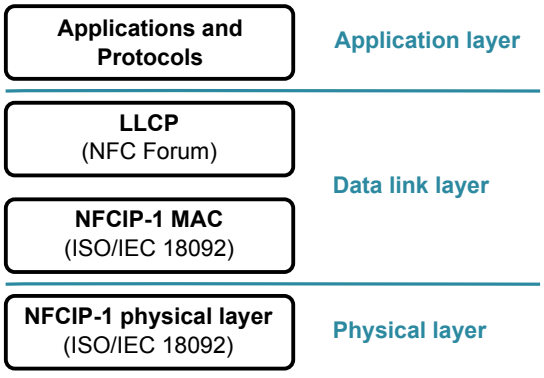


Figure 2.3: NFC Peer to Peer Protocol Stack. (Based on [LR10])

specified in ISO/IEC 18092(NFCIP-1)[ECM] hence an active and a passive communication mode is available. The data link layer is realized by a Media Access Control (MAC) layer

and the Logical Link Control (LLC) layer. The MAC layer is also specified in ISO/IEC 18092(NFCIP-1) and is used for connection establishment and data exchange. The LLC layer is specified by the NFC Forum in LLCP [For11].

### Passive Communication Mode

In passive communication mode the initiator generates the RF field during the whole communication. The target sends the response by modulating the RF field with an Amplitude Shift Keying (ASK) modulation scheme. Therefore the power consumption of the initiator is higher than the targets.

The basic protocol flow for the passive communication mode is shown in figure 2.4. After start up devices are in target mode and waiting for an external RF field. To start

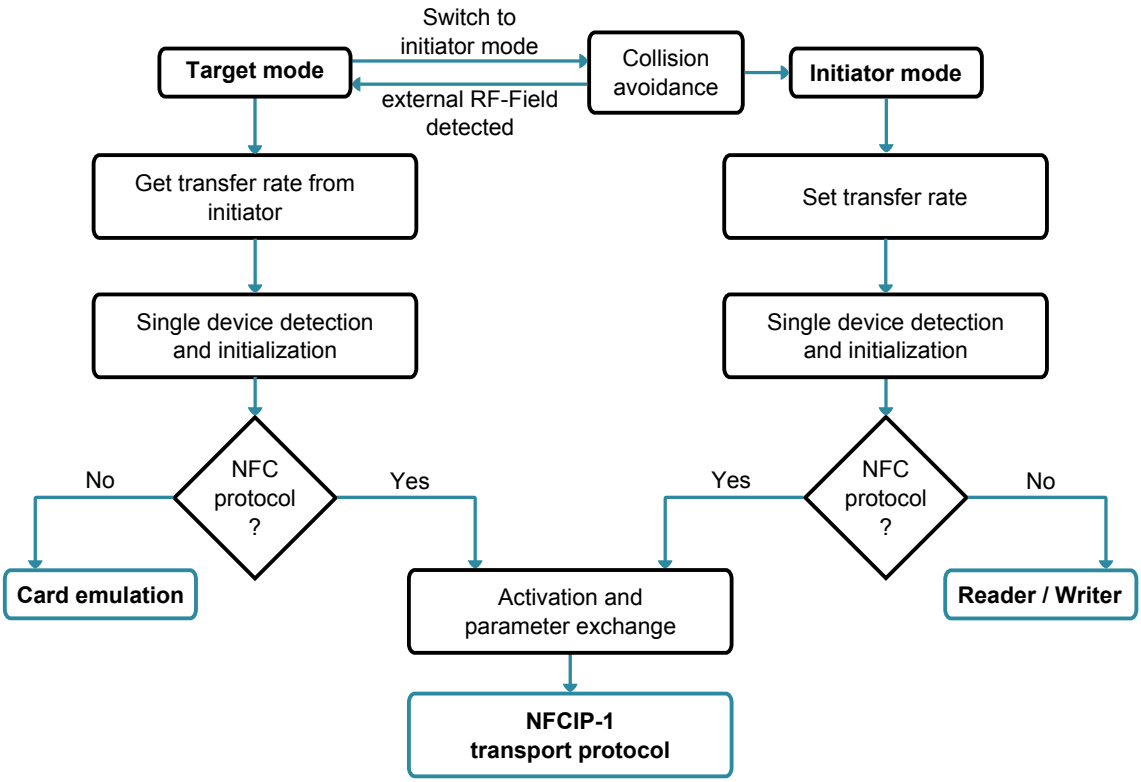


Figure 2.4: NFC Peer to Peer Protocol Flow. (Based on [LR10])

communication as initiator the device tests if an external RF field is present, if yes it stays in target mode. If no external RF field is detected, the device switches to initiator mode and activates the RF field. This process is called Collision Avoidance (CA). After CA the initialization procedure is started. Depending on the desired transfer rate, either by a Polling Request, for 106kbit/s, or a Sense Request, for 212kbit/s and 424kbit/s. The physical and MAC layers, and therefore the anti collision and Single Device Selection (SDD) methods, for the 106kbit/s transfer rate are defined by MIFARE (ISO/IEC 14443) [ISO01] and for the two higher transfer rates in FeliCa (JIS X 6319) [JIS05]. At the end of the SDD the target tells the initiator if it supports data exchange based on NFCIP-1

or a proprietary command set. If NFCIP-1 is supported the data exchange is started by an Attribute Request. Otherwise the Reader/writer mode for the initiator or the Card Emulation mode for the target is activated.

The active mode communication is not described, as it is not used in this thesis.

### Data Exchange

Starting with the Attribute Request command a block oriented Data Exchange Protocol (DEP) is used. The instructions of the DEP always consist of a command/response pair. The initiator always starts with a command and gets the response from the target.

The used frame format is shown in figure 2.5. The frame format differs depending on

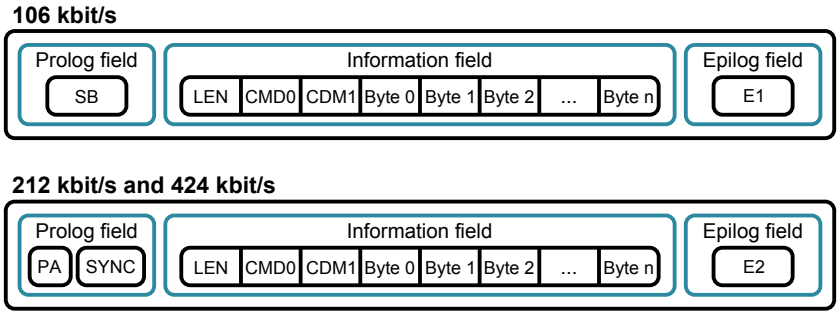


Figure 2.5: NFCIP-1 DEP Frame Format. (Based on [LR10])

the chosen transfer rate. The frames consist of three main fields. Prolog, Information and a Epilog. The information field is the same for all three transfer rates. It starts with one byte containing the length of the information field(LEN), followed by two bytes specifying the command(CMD0 and CMD1) and finally the information bytes. For the 106kbit/s transfer rate the epilog and prolog fields differ from the other two transfer rates. For 106kbit/s the prolog field is a start byte(SB) with the hexadecimal value 'F0' and the epilog field contains a Cyclic Redundancy Check (CRC) checksum. For 212kbit/s and 424kbit/s the prolog field consists of a preamble (PA) containing six bytes with the hexadecimal value '00' and the two byte synchronization sequence(SYNC) with a hexadecimal value of 'B2D4'. For these bit rates the epilog E2 contains a 16 Byte CRC checksum.

The following six command/response pairs are specified in NFCIP-1:

ATR_REQ/ATR_RES: Attribute request/response. This command starts the connection setup. Initiator ant target exchange attributes such as their identification number (NFCID3), supported transfer rates and the maximum frame size. Once a target has sent a response to an attribute request it will not respond to other attribute requests. Hence its possible for the initiator to define a device ID and address and activate up to 14 devices.

PSL_REQ/PSL_RES: Parameter selection request/response. With this command the values for the parameters exchanged during the attribute request/response command can be changed.

DSL_REQ/DSL_RES: Deselect request/response. This command deactivates the target. The target stays in the initialized mode but ignores further requests. In active mode a target is reactivated by a wake up request containing its NFCID3. In passive mode commands from the anti-collision sequence are used.

WUP_REQ/WUP_RES: Wake up request/response. Wakes up a deselected target in active mode using its NFCID3.

RLS_REQ/RLS_RES: Release request/response. This command closes the connection to a target. The target switches back to its initial state.

DEP_REQ/DEP_RES: Data exchange protocol request/response. This command is used to exchange data using Protocol Data Units (PDUs)

The format of PDUs is shown in figure 2.6. The two bytes CMD1 and CMD2 define the operation, DEP request or DEP response. The Protocol Function Byte (PFB) indicates the Protocol Data Unit (PDU) type and the presence of optional parameters. The Device ID (DID) is used to address a specific target and with the Node Address (NAD) logical connections for peer to peer connections can be realized.


Figure 2.6: Protocol Data Unit Format. (Based on [LR10])

### Logical Link Control Protocol (LLCP)

One of the main drawbacks of the request/response nature of NFCIP-1 DEP communication is the lack of a possibility for a target device to initiate a data exchange. A data exchange always starts by a DEP request by the initiator. This mode is called normal response mode. LLCP is used to enable both the initiator and also the target device to start data exchange. This is called asynchronous balanced mode. This capability is required to enable data exchange between devices using higher level protocols like TCP/IP. The NFC-Forum describes the features of LLCP as:

- Link Activation, Supervision and Deactivation
  "The LLCP specifies how two NFC Forum Devices within communication range recognize compatible LLCP implementations, establish an LLCP Link, supervise the connection to the remote peer device, and deactivate the link if requested." [For11]

- Asynchronous Balanced Communication
  "Typical NFC MACs operate in Normal Response Mode where only a master, called the Initiator, is allowed to send data to and request data from the slave, called the Target. The LLCP enables Asynchronous Balanced Mode (ABM) between service endpoints in the two peer devices by use of a symmetry mechanism. Using ABM, service endpoints may initialize, supervise, recover from errors and send information at any time." [For11]

- Protocol Multiplexing
  "The LLCP is able to accommodate several instances of higher level protocols at the same time." [For11]

- Connectionless Transport
  "With minimum protocol overhead, connectionless transport provides a service user an unacknowledged data transmission facility that does not require preparative steps to actually send service data units. This transport mode can be used if upper protocol layers implement their own flow control and so need not rely on the link layer flow control mechanism. It can also be used by applications that operate in a command/response model wherein a command is always followed by a response returned before the next command is sent." [For11]

- Connection-oriented Transport
  "This transport mode provides a data transmission service with sequenced and guaranteed delivery of service data units. Traffic is controlled by a numbering scheme known as the sliding window protocol. Connection-oriented transport requires the preliminary setup of a data link connection and the assignment of resources for as long as the connection persists." [For11]

## 2.2 Non-mobile NFC Applications

### 2.2.1 Smart NFC Interface

In 2007 Ailisto et al. developed a device called Smart NFC Interface[AMH+07]. The Smart NFC Interface offers wired and wireless communication possibilities with a C programmable micro controller as main component. The modular structure allows the use of the Smart NFC Interface for many different applications.

#### Hardware Description

Figure 2.7 shows the block diagram. The Smart NFC Interface consists of two boards, the basic board and the communication board. The basic board contains the power supply (charging connector, charging circuit, Lithium-ion (Li-Ion) battery and power regulation), components for data logging (real-time clock, flash memory), wired communications (JTAG, RS-232, SPI and analog/digital General Purpose Input/Output (GPIO)), a temperature sensor and the micro controller. Components for wireless communication (NFC, Bluetooth, Infrared Data Association (IrDA) and a remote control receiver) and an optional Secure Element (SE)) are on the communication board.

The micro controller is a 8-bit Atmel AVR series ATmega128L with 128 kilobytes flash program memory, 4 kilobytes Random-access memory (RAM) and 4 kilobytes Electrically Erasable Programmable Read-Only Memory (EEPROM). NFC communication is driven by NXP's PN531 NFC controller connected to the micro controller via Serial Peripheral Interface Bus (SPI). Bluetooth is based on a Bluegiga WT12-A-AO module connected to the micro controllers Universal Asynchronous Receiver/Transmitter (UART). IrDA is realized with hardware based on Michrochips MCP2150-I/SO controller and Ziglogs ZHX1403 IrDA transceiver.

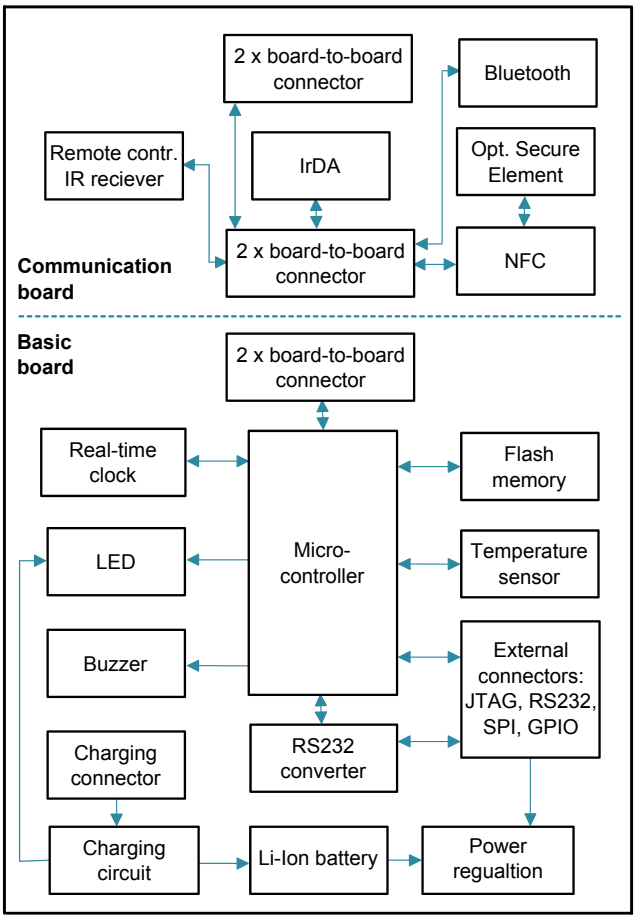


Figure 2.7: Smart NFC Interface Block Diagram. (Based on [AMH+07])

### Application Scenarios

For the Smart NFC Interface two main application scenarios are defined. It can either act as an NFC-Bluetooth Gateway or an NFC server.

The NFC-Bluetooth Gateway mode is used to add NFC functions to devices with Bluetooth but no NFC support. Such devices can be mobile phones or laptops. As this thesis targets on adding NFC communication possibility to non-mobile devices, with minimum costs and development effort, the application described next seems to be more interesting.

In the NFC server scenario NFC communication can be added to devices by embedding the Smart NFC Interface into them. The device exchanges data with the Smart NFC Interface via a wired connection, for example a UART interface. If a NFC-Initiator comes in range of the Smart NFC Interface an action can be triggered. The required information is transferred from the device to the Smart NFC Interface and the response is sent via the Smart NFC Interface. Another option is to constantly transfer data from the device to the Smart NFC Interface, if now a NFC-Initiator enters the NFC operating range the data can be exchanged between the Smart NFC Interface and the NFC-Initiator.

For both operations the peer to peer mode of NFC is used. The Smart NFC Interface works in NFC passive target mode, therefore the power consumption can be minimized.

The Smart NFC Interface is often used as an NFC interface to sensors, but also controlling a device without a User Interface (UI) with a NFC enabled handset can be realized. As an example the interaction with a thermostat is described. The actual value is read by touching the thermostat with a NFC enabled mobile phone. Then the desired Value can be set on the phones display and by touching the thermostat again the old value can be replaced with the chosen value. Triggering UI actions by touching the corresponding objects enhances the user experience, as no connection setup or device selection is needed, such as necessary when using other short range radios like Bluetooth.

There are various papers describing applications using the Smart NFC Interface to add NFC connectivity to non-mobile devices. Some of them are summarized on the next pages.

### 2.2.2 Sensors with NFC Communication

In 2007 Strömmer et al. [SHYo07] evaluated the possibilities of building passive or semi-passive NFC equipped sensors. They defined three application scenarios:

- Passive sensors
- User-controlled semi-passive sensors
- Stand-alone semi-passive sensors for long term monitoring

A passive sensor takes all the energy it needs from the RF files generated by the NFC initiator. If an RF field is present the sensor is powered up with energy taken from the field, measures and sends a response to the commands sent by the initiator. If more energy is needed a power capacitor in the sensor module can be charged by the field before the measurement and provide the energy for the measurement.

Semi-passive sensors have power supplies integrated in the sensor board. So the sensor functions are powered from this internal source, but the energy for NFC communication is taken from the RF field. In case of user controlled semi-passive sensors the sensor is activated by an NFC transceiver operating in NFC's passive target mode. For stand-alone semi-passive sensors the sensor gets periodically activated to perform measurements and transfers the data over NFC when a initiator activates the sensors NFC controller. For the two semi passive sensors prototypes have been build and evaluated using the Smart NFC Interface like shown in figure 2.8.

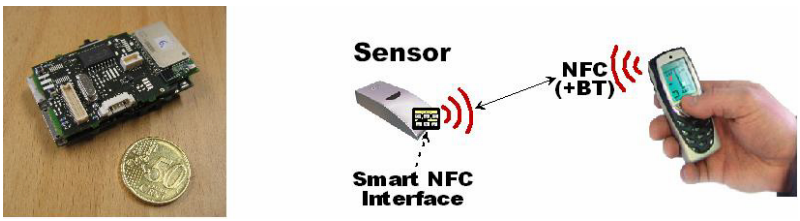


Figure 2.8: Smart NFC Interface Module (Communication Board on top, NFC antenna not installed) and its basic application scenario as a sensor interfacing module. [SHYo07]

The results of their prototype tests are summarized in this paragraph. When using NFC's 106kbit/s communication mode the time for channel setup and sending of a few bytes of data is within a few tens of milliseconds. Compared to the delays on their mobile phone software this is quite fast. The NFC controller used (NXP's PN531) supports passive communication mode (load modulating the initiators RF field) but does not support taking the energy for the sensor application from the RF field. Therefore no fully passive sensors are possible with this NFC controller. A other drawback of this controller was the lowest achievable power consumption of 30$\mu$A from 2.5 volts. With this power consumption a several years lifetime of coin-cell batteries in semi-passive sensors is not possible. Furthermore they found that "Good power management support is essential for semi passive sensors. NFC chips with reduced functionality (e.g. without active modes) could be a reasonable approach to ultra low power sensors, since this would cut down their power consumption and price further." [SHYo07].

2009 they also published a paper on practical implementations of passive and semi passive sensors with an NFC interface [MMA09]. There they presented implementations of NFC sensor prototypes using commercial NFC chips (NXP's PN531 and the Insight Contactless Microread).

The main conclusion was that NFC chips are generally designed for the mobile phone market, and chips for NFC sensor interfaces do not need to support the full NFC functionality. As already mentioned in [SHYo07] the PN531 consumes too much power in power down mode, so it is not suitable for battery powered semi-passive sensors. The Microread NFC chip was used to implement a passive sensor as it has the possibility to supply a passive application with power from the RF field. This worked with NFC readers continuously providing an RF field. But as NFC mobile phones, for power saving reasons, only activate their RF field in bursts the host controller was not able to reply fast enough.

## 2.3 NFC Peer to Peer Communication

### 2.3.1 OPEN-NPP, an open Source Library to enable P2P over NFC

In 2012 Lotito and Mazzocchi published a paper called "OPEN-NPP: An Open Source Library to Enable P2P over NFC" [LM12]. They developed a library to enable bi-directional NFC communication link between a NFC enabled Android device and a NFC reader. They used a Samsung Nexus S phone with Android 2.3 and an ACR122 NFC terminal from Advanced Card Systems (ACS). NDEF Push Protocol (NPP) is a protocol to push NFC Data Exchange Format (NDEF) [For06] Messages between devices.

Although the NPP procedure is a one way communication from a client to a server, it is possible to realize bidirectional NDEF message exchange by setting up both device as a server and run the client procedure when it has messages to push. An NPP server is registered at the LLCP with the service name "com.android.npp" and processes NDEF messages pushed by an NPP client.

"An NPP server must accept all connections with the major protocol version 0x0 and must ignore all NDEF entries with unknown action code." [npp06] If an NPP client has messages to send they are sent immediately after a DEP connection is established. Therefore the following procedure is used:

- "Connect to LLCP socket with service name "com.android.npp".
- Send the NPP Header following by NDEF Entries as defined in the Data Format section.
- Disconnect the LLCP socket." [For06]

The Library is realized in Java and uses the PC/SC interface [Wor05] to communicate with the NFC reader. OPEN-NPP provides a byte array communication between a computer and an android phone over NFC.

The communication flow to set up an NPP communication link is shown in figure 2.9.

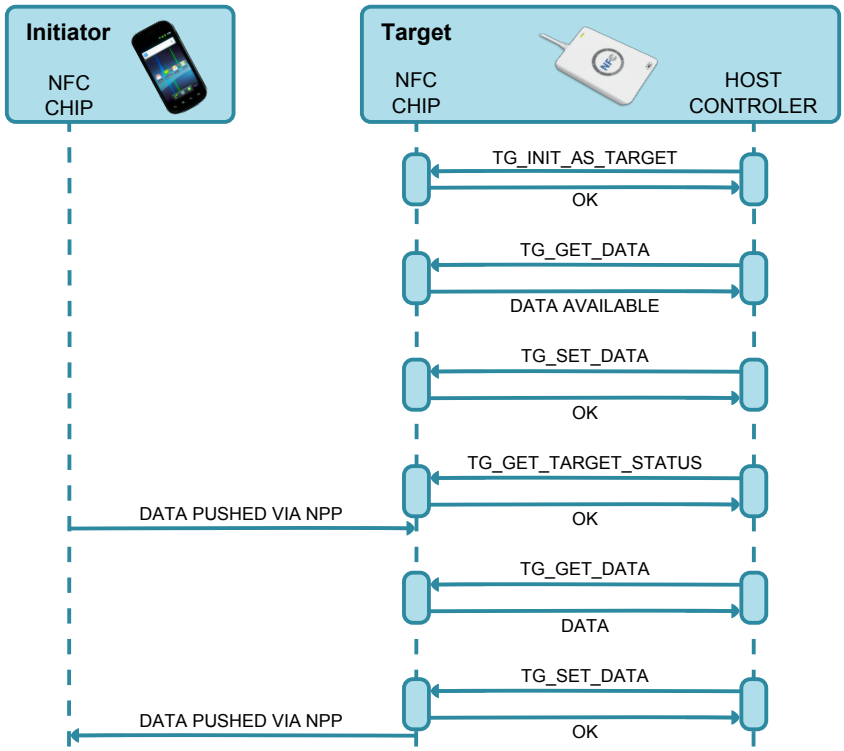


Figure 2.9: OPEN-NPP from Smartphone to PC flow. (Based on [LM12])

These are the steps in detail:

1. The reader is configured to be the communication target with a command called TG_INIT_AS_TARGET.
2. With the TG_GET_DATA command the NFC Controller is configured as the target for the DEP if there is available data. The response tells that there is available data and contains a LLCP PDU indicating the request for a data link connection.

3. Now an LLCP PDU telling the initiator that the connection request is accepted is sent with the TG_SET_DATA command.

4. With the TG_GET_TARGET_STATUS command the actual status of the NFC controller is retrieved. The response confirms that the target is activated and both target and initiator use the same data rate.

5. The response to the TG_GET_DATA command contains an LLCP PDU containing the transferred NDEF message.

The library is used in the NaviNFC project [BFLS11]. NaviNFC is an indoor navigation system for Android phones based on NFC. With NaviNFC the user can touch NFC tags at strategic positions to show his position and the shortest way to a chosen destination on a map. Therefore the application and the maps have to be loaded on the phone. This is realized with an NFC peer to peer connection. So the application and maps are transferred over NFC without the need of an active internet connection or any other data link.

As a next step the library will be extended to work with any NFC reader and NFC devices supporting LLCP. They also want to turn the library into a embedded solution to build intelligent readers with reduced costs and size.

### 2.3.2 TCP/IP over NFCIP-1

Stefan Grünberger wrote his thesis titled "Analyse und Implementierung des IP-Protokolls über NFCIP-1" [Grü07] in 2007. For his thesis he designed and implemented a system for transmitting data using the TCP/IP protocol over NFC. Two NFC-Boxes designed by FH Hagenberg have been connected to two Personal Computers (PCs) using Bluetooth. This is shown in Figure 2.10.

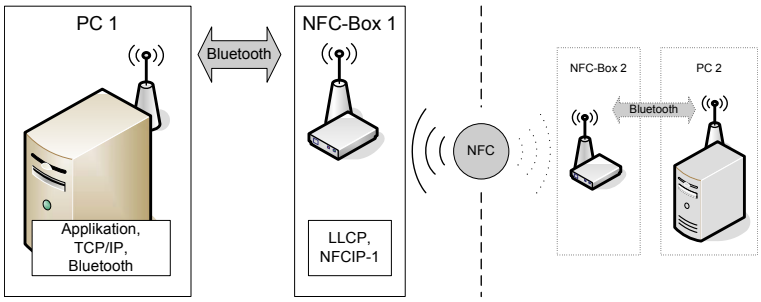


Figure 2.10: System Overview TCP/IP over NFCIP-1 by Stefan Grünberger. [Grü07]

The so called NFC-Box is a combination of a Atmel ATmega128L microcontroller, NXP's PN512 NFC-Controller with Antenna and a LMX9820A Bluetooth module. This can be seen in Figure 2.11. The two PCs are running Gentoo Linux with a 2.6.20-gentoo-r8 Linux Kernel.

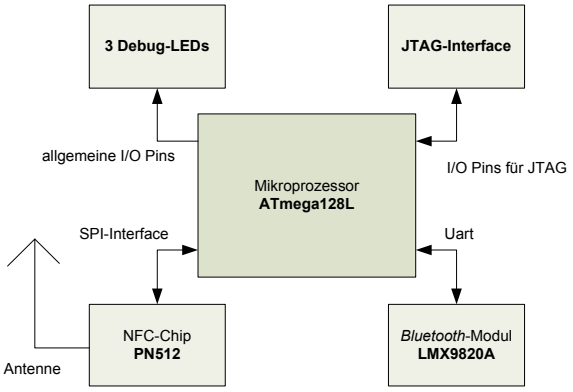


Figure 2.11: FH Hagenberg NFC-Box. [Grü07]

### Protocol Stack

The following Protocol stack is used for communication:

| | |
|---|---|
| NFCIP-1: | The NFC-Communication is made in peer to peer mode using NFCIP-1. The Protocol is handled by the PN512 NFC-Controller in the NFC-Box. |
| LLCP: | To enable intentional message transfer from the initiator as well as the target side LLCP is used. The LLCP Stack is implemented in the ATmega128L microcontroller in the NFC-Box. |
| Radio Frequency Communication (RFCOMM): | RFCOMM is part of the Linux Bluetooth Stack and is used to establish a serial connection between two Bluetooth endpoints. After connection establishment a new device file is created, which can be used like the device file of a hardware serial port. In this case the port is used to communicate with the microcontroller in the NFC-Box. |
| Point-to-point protocol (PPP): | PPP is a data link protocol used to establish a direct network connection between two network nodes. As PPP can also establish network connections over serial ports and is already included in the Linux network Stack, PPP is used to establish the network connection over the serial port provided by RFCOMM. |
| TCP/IP: | TCP/IP is used on top of the previous described protocols and also included in the Linux Network Stack. |

The IP-Packets sent from the Host PC to the NFC-Box are buffered in the microcontroller RAM. The Maximum Transmission Unit (MTU) for IP Packets is set to 512 Bytes, therefore the Buffer in the microcontroller RAM is defined with a size of 520Bytes, as space for the LLCP Header is also required.

### Tests and Results

In the thesis most of the testing only concerned data transfer rates. Two Test cases were defined. In the first scenario data transfer rate for sending packets between the two microcontrollers using NFCIP-1 were investigated. The second scenario covers data transfer rates for communication between the two Host PCs tunneled through Bluetooth to the NFC-Box and from there via NFCIP-1 to the counterpart.

**Test Case 1:** In this test case the data transfer rate between the two microcontrollers was evaluated. Therefore 8 measurements of the usable data rate between the two endpoints connected via NFCIP-1 were made. Four measurements with NFCIP-1 in 104 kBit/s and four measurements with NFCIP-1 in 424 kBit/s mode. Figure 2.12 shows the results of the first test scenario with disabled LLCP stack in the microcontrollers.

The average end to end transfer rate with NFCIP-1 in 106 kBit/s mode was 6.184 kByte/s. There was almost no difference between the 4 measurements as no transfer errors could be seen. This was because of the close distance between the NFC Antennas. With increasing antenna distance the transfer rates would be significantly lower because lost or corrupt data frames would be retransmitted.

The second data set in the diagram shows the end to end transfer rates between the microcontrollers using NFCIP-1 in 424 kBit/s mode. A confusing fact is the lower end to end data rate of about 4.6 kByte/s although the four times faster NFCIP-1 data rate was used. In the thesis this is explained by showing that setting some Registers in the NFC controller for 424 kBit/s mode needs 2.8ms more than in 106kBit/s mode.

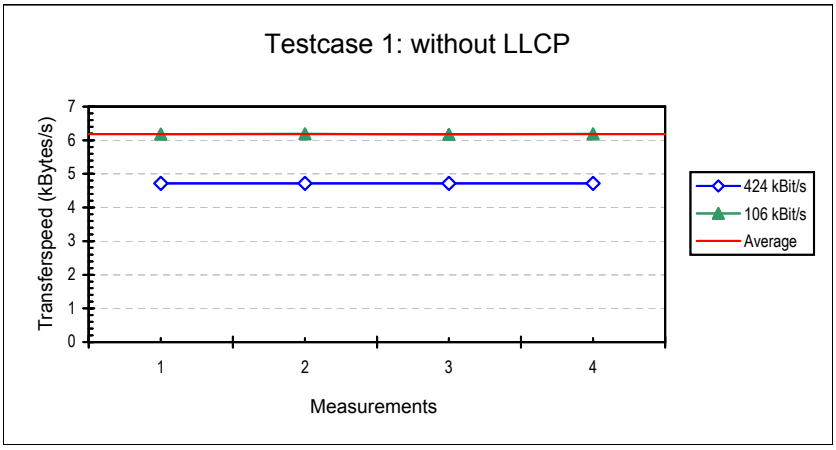


Figure 2.12: TCP/IP over NFCIP-1 Results without LLCP. (Based on [Grü07])

Figure 2.13 shows the same measurement with the LLCP stack enabled. This results are almost the same as without the LLCP stack enabled. The small speed drop is explained with the higher Central Processing Unit (CPU) load of the microcontrollers with the LLCP stack enabled.

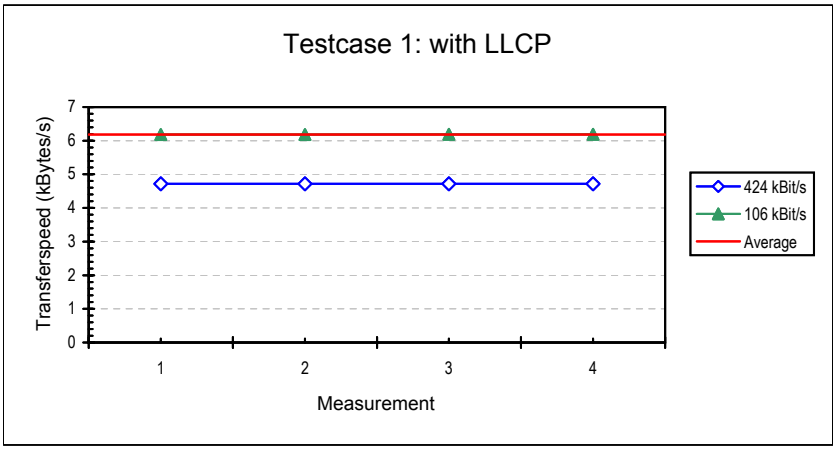


Figure 2.13: TCP/IP over NFCIP-1 Results with LLCP. (Based on [Grü07])

**Test Case 2:** This test case covers the TCP/IP communication between the two host PCs. Therefore the TCP/IP tunnel is established like described before. Then one of the host PCs is configured as File Transfer Protocol (FTP) server and the other one as FTP client. FTP server software ProFTPd [Pro13b] is used for the server and the Graphical User Interface (GUI) application gFTP [Mas13] is used as client on the other PC. During the measurements a binary file of 20 kByte is transferred and the average transfer rate calculated by gFTP is taken into account. The Linux command line tool ifconfig was used to get the number of transmission errors occurred during the measurement.

To investigate the influence of the MTU used by PPP three values for the MTU have tested. Therefore 128, 400 and 512 Bytes have been used as value for the MTU. 128 Bytes is the smallest MTU accepted by PPP so 128 Bytes was chosen as the smallest MTU for the test. All data packets are buffered within the microcontroller, the data buffer in the microcontroller is able to buffer up to 512 Bytes. Therefore 512 Bytes is used as the highest MTU. As a Value in between the maximum and minimum, 400 Bytes is used as the third tested MTU.

With an MTU of 128 Bytes an average data rate of 1.49 kByte/s with 6.5 errors per measurement is reached. The measurement number 7 is the only measurement without transfer errors. For this measurement the data rate was 2.55 kByte/s. This is shown in Figure 2.14.

Figure 2.15 shows the results with the MTU set to 400 Bytes. During this test only one complete data packet could be stored in the microcontroller's buffer. During this test the average data rate dropped to 1.17 kByte/s and the average number of transmission errors increased to 15.5 errors per measurement.

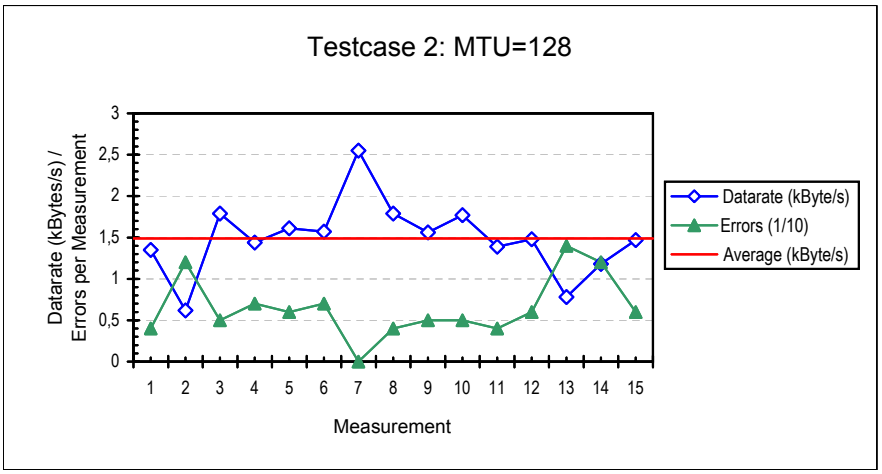


Figure 2.14: TCP/IP over NFCIP-1 Results for FTP MTU=128 Bytes. (Based on [Grü07])

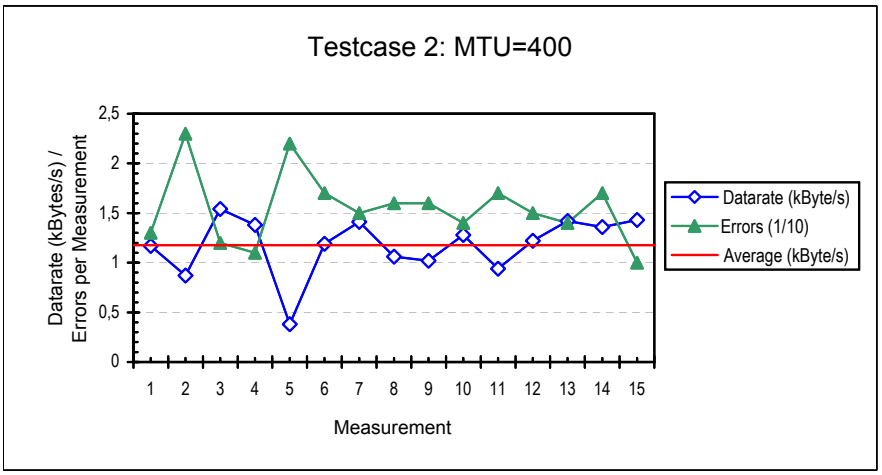


Figure 2.15: TCP/IP over NFCIP-1 Results for FTP MTU=400 Bytes. (Based on [Grü07])

With 512 Bytes as MTU the average data rate was 1.24 kBytes/s and the average number of transmission errors further increased to 16.7 errors per measurement. This is shown in Figure 2.14.

These results are quite confusing, as with increasing MTU the data rate is assumed to increase, but the results are showing a drop from 1.49 to 1.17 kByte/s when increasing the MTU from 128 to 400 Bytes. In the original thesis this is explained with the increasing number of transmission errors and therefore retransmission of packets. The results of the measurements from microcontroller to microcontroller without application showed no transmission errors on NFCIP-1 level. Hence it seems data packets got lost between the host PCs and the microcontrollers or data packets received by the microcontrollers are not transferred to the host PCs or NFC-Controllers correctly.

The highest measured data rate is about half of the NFCIP-1 data rate measured during test case 1. This is explained by the Protocol overhead consisting of the LLCP header, the Internet Protocol (IP) header and the Transmission Control Protocol (TCP)

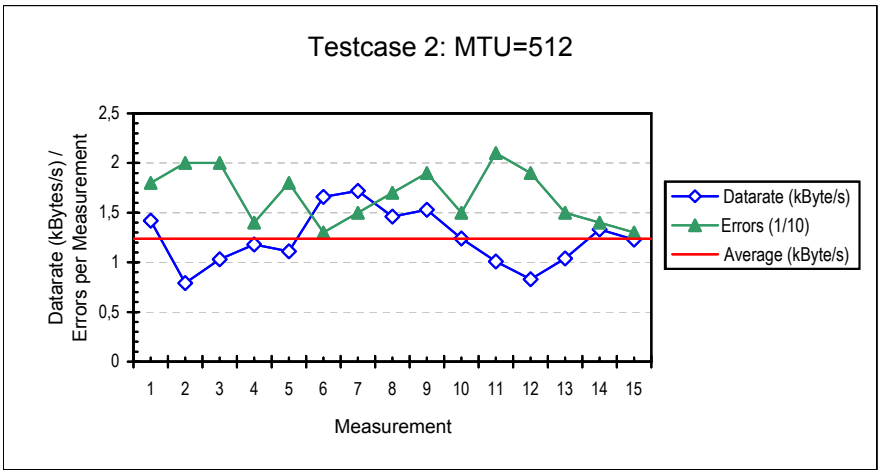


Figure 2.16: TCP/IP over NFCIP-1 Results for FTP MTU=512 Bytes. (Based on [Grü07])

header. This are at least 45 Bytes per data packet. Considering this overhead about 34 percent of the data rate gap is assumed. The rest of the drop is assumed to occur because of slow SPI and Bluetooth connections between the host PCs and the microcontrollers or between the microcontrollers and the NFC-Controllers. The cause for the transmission errors is not explained in the thesis.

## 2.4 TCP/IP over lossy Channel Enhancements

The goal of this thesis is to provide a communication channel for TCP/IP [oSC81] [CDS74] communication. TCP/IP was designed for wired channels with minimal losses except congestion and works well on such channels. Wireless channels not only provide lower bandwidth, longer propagation delays and reduced channel reliability they also suffer from bursty error losses. TCP/IP interprets these bursty losses as a sign of congestion and reduces the congestion window, therefore also the throughput of the connection is drastically reduced. [PGLA00]

In this chapter a link layer protocol, trying to reduce this problem, is reviewed.

### 2.4.1 TULIP

In 2000 Christina Parsa and J. J. Garcia-Luna-Aceves published the paper “Improving TCP performance over wireless networks at the link layer” [PGLA00]. In this paper they described a approach to optimize TCP/IP over wireless channels. The solution is called Transport Unaware Link Improvement Protocol (TULIP). TULIP is designed to be used on half-duplex radio links and provides a link-layer to efficiently use the wireless channel’s bandwidth. “TULIP causes no modification of the network or transport layer software, and the link layer is not required to know any details regarding TCP or the algorithms it uses. TULIP maintains no TCP state whatsoever, and makes no decisions on a TCP-session basis.” [PGLA00] Hence the maintenance overhead for multiple TCP sessions for a single destination is reduced. With TULIP a lossy connection over a wireless channel

seems to be a slow and error free connection for the transport layer. So TCP can maximize the throughput on a link delivering packets in sequence as long there is no congestion.

As there are long timeouts in TCP, TULIP is able to recover from losses within this time without TCP noticing it. A simple repeat retransmission strategy is used to realize this behavior.

To optimize throughput TULIP provides not only a reliable service but also an unreliable service. TCP data packets are sent via the reliable service, TCP acknowledge packets with no data and UDP packets and also link level ACK packages are sent via the unreliable service. For details about the implementation and function principle of TULIP refer to [PGLA00].

Simulations have shown, that TULIP minimizes the TCP timeouts for exponentially distributed channel errors. Also the throughput is higher than when using the Snoop protocol, which is one of the best common solutions for TCP over wireless channels. When burst losses or channel fading occurs, TULIP retransmits the lost packets and provides a reduced but consistent throughput. An advantage over other known solutions is that TULIP works without inspecting the TCP headers. Hence TULIP is compatible with any TCP version and also with encrypted TCP headers.

# Chapter 3

# System Design

In this chapter first some possibilities to enable high level NFC communication are reviewed and compared. Next the architecture of the chosen approach is described. This includes the basic components needed for NFC communication, the communication interfaces to be used and especially the NFC controller as it is one of the most important components for this thesis.

## 3.1 Comparison of High Level Protocol NFC Communication Possibilities

Generally high level protocol NFC communication for non-mobile devices, can be enabled using either off the shelf NFC Controllers (NFCCs) or by developing a new type of NFC controller specially adapted and with reduced feature-set, to fulfill given requirements.

On the following pages some concepts for high level protocol NFC communication are evaluated and compared. Therefore these criteria are used:

| | |
|---|---|
| Hardware costs: | Costs of the NFCC and additional communication hardware |
| Power consumption: | Consumption of the NFCC and additional communication hardware |
| Connection setup time: | Time until the communication channel is ready to send or receive data |
| Development effort on host: | Amount of additional software components to be implemented in the host controller firmware for communication |
| Data rate: | Data Throughput once the communication channel is set up |

### 3.1.1 Solutions using off the Shelf NFC Controllers

#### Bluetooth or Wireless LAN with NFC Pairing

This concept uses NFC only for setting up a Bluetooth or WLAN connection. This combines the ease of use of the NFC touch to interact paradigm with the speed and well proven high level protocol communication capabilities of Bluetooth or WLAN communication. Salminen et al. evaluated Bluetooth with NFC pairing in [SHR06]. The outcome is that this solution provides good usability and takes about four seconds until the connection is established. For WLAN pairing usability and connection setup time is assumed to be the same as for Bluetooth with NFC pairing. Both solutions, Bluetooth and WLAN with NFC pairing offer high data rates. The need for NFC and also Bluetooth or WLAN hardware is a major drawback of these solutions. The hardware costs, as well as the power consumption, is significantly increased by the additional hardware. Also, additional software components to support the communication hardware, have to be implemented in the host controller software.

#### TCP/IP over NFC with Software LLCP Stack

Another possibility is to use an off the shelf NFC controller with the LLCP stack on the host controller. As LLCP offers Asynchronous Balanced Mode (ABM) it is possible to use the TCP/IP protocol over NFC. This approach offers several advantages over the Bluetooth or WLAN with NFC pairing concepts. No further hardware is needed in addition to the NFC hardware, therefore hardware costs and power consumption are significantly lower. Also the connection setup time is slightly shorter, than for Bluetooth or WLAN connections. An LLCP stack has to be implemented in the host controller so there is additional implementation effort. In this approach the whole data communication is done over NFC in peer to peer mode. Therefore the data rates are lower than using Bluetooth or WLAN.

### 3.1.2 Solution with special adapted NFC Controllers

#### TCP/IP over NFC with Software LLCP Stack

Although NFC controllers available today are already low cost devices, the costs can be reduced further by adapting the NFC controller hardware. NFC controllers normally support reader an card emulation mode as well as the peer to peer mode used for LLCP. They also often include additional features and interfaces. By developing an NFC controller that only supports peer to peer mode and the required contact less and host interfaces the costs and power consumption could be minimized. Connection setup time and data rates are the same as with off the shelf NFC controllers. This solution also requires an LLCP stack in the host controller software, and therefore adapting the host controllers software causes additional effort.

#### TCP/IP over NFC with LLCP enabled NFC-Controller

When developing a special NFC controller just for peer to peer communication the integration of the LLCP stack into the NFCC avoids the need for an LLCP software stack in the host controller software. Although the chip size, and as a consequence thereof the hardware costs, and will be slightly larger than without the LLCP stack, the overall costs could be further reduced as the host controller firmware does not have to be adapted. The connection setup time and data rates are the same as with a lot of off the shelf NFC controllers.

### 3.1.3 Comparison

The following Table (Table 3.1) contains an overview of the discussed solutions. The criteria defined in Section 3.1 are used and the concepts are rated using a scale from one to five. Where five stands for the worst and one for the best fitting the criteria.

| Hardware Type | Standard NFC Hardware | | | Adapted NFC Controller | |
|---|---|---|---|---|---|
| Concept | Bluetooth w. NFC pairing | WLAN w. NFC pairing | LLCP handled by Host | LLCP handled by Host | LLCP handled by NFCC |
| Hardware Costs | 4 | 5 | 3 | 1 | 2 |
| Power Consumption | 4 | 5 | 3 | 1 | 2 |
| Connection setup Time | 5 | 4 | 1 | 1 | 1 |
| Development effort on Host | 4 | 4 | 4 | 4 | 1 |
| Data rate | 2 | 1 | 4 | 4 | 4 |
| Overall rating | 3.6 | 3.8 | 3.0 | 2.2 | 2.0 |

Table 3.1: Comparison of High Level Protocol NFC Communication Possibilities

The comparison in Table 3.1 shows that the approach with an adapted NFC controller with LLCP handled by the NFC controller would seem to best fit the defined criteria. This approach is therefore chosen to be realized within this thesis.

## 3.2 Architecture

The chosen approach is an adapted NFC controller and LLCP handled by the NFC controller. For the embedded system the system architecture mainly consists of three parts:

RF hardware: The analog frontend required for NFC RF communication.

NFC controller: The NFC controller handles the various levels of NFC communication protocols and provides interfaces to the host controller via NFC Controller Interface (NCI). Optional Interfaces to secure elements or Subscriber Identity Module (SIM) cards can be provided.

Host controller: The main controller of the appliance. Handles the high level communication Protocol (for example TCP/IP) sends and receives data.

For development and testing in this thesis a combination of two of this subsystems is used to enable TCP/IP communication between two host controllers. This is shown in Figure 3.1.

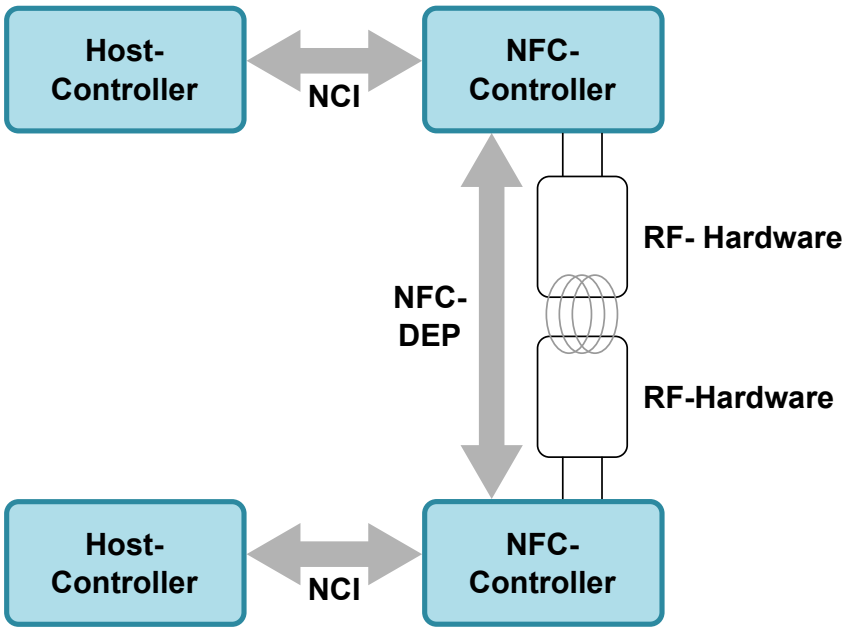


Figure 3.1: Communication System Architecture

### 3.2.1 Radio Frequency Hardware

Modern NFC controllers, like for example the NXP PN533 [Sem12a], provide a so called Contactless Interface Unit (CIU). In this unit almost the complete RF hardware is implemented. This means the modulation, demodulation, framing and error detection for the supported communication protocols is handled by the NFC controller. Hence the required RF hardware is reduced to the antenna and some passive components for the matching network.

There are many commercially distributed NFC-Antennas available. Planar ferrite sheet antennas for mobiles, for example the NFC antennas from Pulse Electronics, are small in size ($15cm^2$). They can also be used in non-mobile devices and because their thinness allows them to be easily integrated into the device case.

In 2011 Li Li et al. published a paper about NFC antenna matching. In this paper they explained why a matching network is needed. "The Near Field Communication (NFC) operates in 13.56MHz frequency band, with low cost and easy to use features. In the NFC communication process, information transfers through the electromagnetic induction as the high frequency RFID. And the antenna performance is one of the core problems. Because of the volume restrictions, coil antennas are currently used. And the matching network [1] is to make sure that the antenna works in the 13.56MHz frequency band."[LGW11]

As an NFC controller like those used in mobile phones is generally also used for non-mobile NFC interfaces, the RF hardware used in mobiles can also be used for non-mobile devices.

### 3.2.2 NFC Controller

An NFC controller is used to abstract the complexity of NFC communication for the host controller. Therefore NFC connection setup, data exchange and disconnection is handled by the NFC controller. An NFC controller also provides the possibility to route the data traffic to an embedded secure element or a Universal Integrated Circuit Card (UICC). After configuring the routing functionalities in a routing table an embedded secure element or a UICC can handle modes like card emulation without any interaction with the host controller.

To be compliant with the NFC forum rules an NFC controller has to support several operating modes [LR10]:

- ISO/IEC 14443A Reader/Writer
- FeliCa Reader/Writer
- ISO/IEC 14443B Reader/Writer
- ISO/IEC 14443A/Card emulation
- FeliCa Card emulation
- ISO/IEC 18092, ECMA 340 Peer-to-Peer

The communication data can be routed either to the host, or a UICC, or a secure element. For example in peer to peer, reader or writer modes the communication can be routed to the host interface and for card emulation mode to a UICC or an embedded secure element.

Figure 3.2 shows a possible NFC controller interface structure.

Communication with an embedded secure element or a UICC is realized via Single Wire Protocol (SWP) or NFC Wired Interface (NFC-WI).

In order to be flexible enough to react to changes in NFC forum specifications or to support new NFC functions, most NFC controllers are built around a micro controller core.

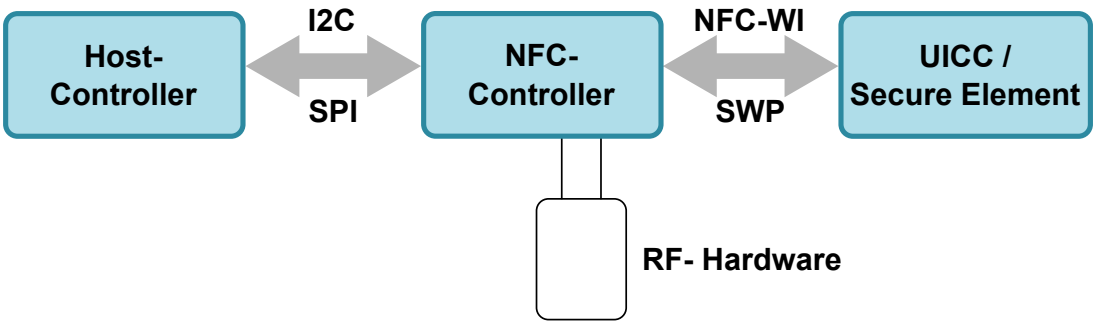


Figure 3.2: NFC Controller Interface Structure

Besides the micro controller core NFC controllers are equipped with Read Only Memory (ROM), EEPROM or FLASH memory and additional peripherals for NFC communication and various other communication interfaces. All these components are integrated into a System On Chip (SOC).

A possible Hardware Structure is shown in Figure 3.3.

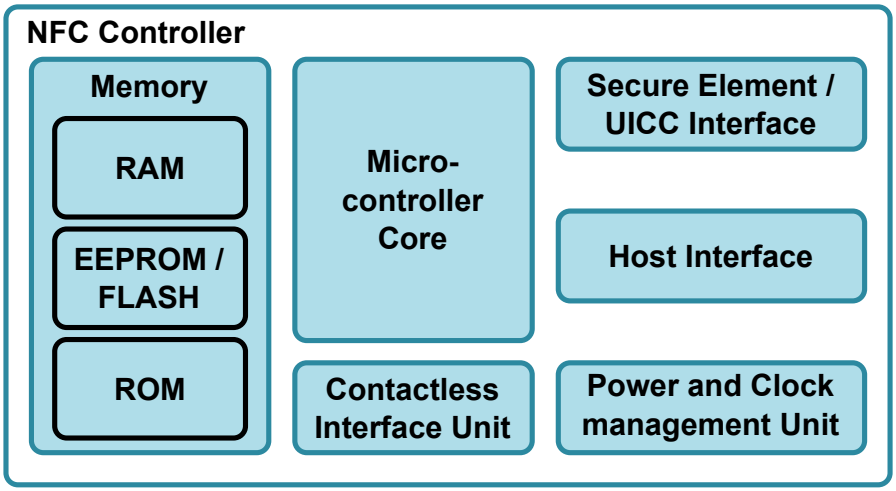


Figure 3.3: NFC Controller Hardware Structure

As NFC controllers are mostly used in mobile phones they have to fulfill strict requirements in terms of power consumption as well as chip size and price.

### Micro Controller Core

Like the whole NFC controller SOC the micro controller core has to fulfill the previously explained requirements. So a low power, low chip size micro controller core with adequate processing power should be used. In current SOC design flow not all components are designed from scratch. Often pre-designed components are used. Such components are called Intellectual Property Cores (IP-Cores).

Depending on the requirement for the micro controller the used core can vary from low end 8 bit cores (for example the 8051 processor family) up to state of the art 32 bit Cores like the ARM Cortex processor family.

### Memory

"ROM is used for highly area optimized instruction memory, although this comes at a price of lengthy integration time due to its need to be correct before the chip is sent for fabrication. Flash is an alternative instruction memory that can significantly reduce the time to market by allowing embedded software to be upgraded after fabrication, meaning that software test and fabrication can be overlapped." [SC01]

Because ROM memory is more area efficient, and therefore cheap, than FLASH or EEPROM memory, the non-volatile memory can be split up into ROM and FLASH or EEPROM. The software parts that are not likely to change over time are stored in ROM and cannot be updated after production. Software stored in FLASH or EEPROM is much more flexible and can be updated after production of the controller or even in the field. Bugs in ROM code can be fixed using software patches. A possible approach for ROM patching is described in the paper "Patchable Instruction ROM Architecture" by Timothy Sherwood and Brad Calder. [SC01]

### Contactless Interface Unit

The CIU acts as a modem for NFC communication. As described earlier, up to date NFC controllers include nearly all the parts needed for 13.56MHz (the Frequency NFC operates at) RF communication. Therefore the CIU has to provide the following functions for all the previously mentioned operating modes of a NFC controller:

- Detection of RF level
- Data mode detection
- RF demodulation
- RF load modulation
- RF data encoding and decoding
- Data framing and error detection
- Transmitter drivers
- Optionally encryption and decryption of MIFARE [Gmb13] data

Because various antenna designs can be used, the parameters of the antenna may vary with the usage of the controller. Therefore the transceiver unit can be adapted by using adjustable parameters. For communication with the micro controller core the CIU needs a suitable interface. Figure 3.4 shows a possible hardware structure of a contactless interface unit.

The analog interface covers modulation and demodulation and transmitter drivers. The presence of an external field is detected by the RF level detector. The data mode

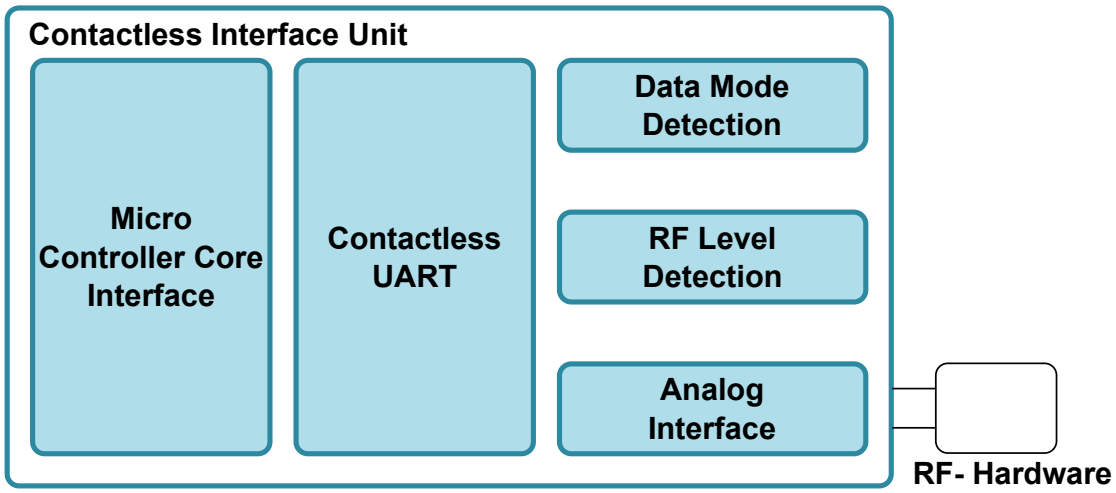


Figure 3.4: Contactless Interface Unit Hardware Structure

detector is used to determine the current communication and provides this information to the demodulator. The contactless UART handles parts of the communication protocols. Some parts of the communication protocols may be handled by the firmware. The micro controller core interface has to ensure communication with the micro controller core without data loss.

### Power and Clock management Unit

In this unit the power management of the NFC controller is handled. Usually one or more standby and sleep modes are realized.

The system clock generation is often also realized in this unit. The clock generation is needed to provide the, often different, clocks for the micro controller core as well as for various peripherals or communication interfaces. Therefore it is common to use a crystal oscillator to generate a main clock, and generate the other clocks by dividing the main clock or multiplying them by a Phase Locked Loop (PLL) clock multiplier.

### Host Interface

The task of the host interface is to ensure reliable data communication between the host controller and the NFC controller. To be more flexible most NFC controllers support multiple types of host interfaces. These interfaces can either be serial or parallel. The usage of parallel interfaces leads to increased chip size because more Input/Output (I/O) pins are needed. Because most NFC controllers are designed for small chip size serial interfaces are used normally. Two possible interface options are described in the following paragraph:

Serial Peripheral Interface Bus (SPI): The SPI is a standard for a synchronous serial port, developed by Motorola. SPI defines a master-slave concept, with two data signals (MOSI and MISO) a serial clock (SCLK) and

| | |
|---|---|
| | one or more chip-select signals. A detailed description of SPI is given in "Spi Block Guide v03.06" [Inc03]. |
| Inter-Integrated Circuit ($I^2C$): | Inter-Integrated Circuit ($I^2C$) is also a master-slave bus system, sometimes it is also called Two-Wire-Interface (TWI). $I^2C$ uses two signals, a clock signal (SCL) and a data signal (SDA). To identify multiple devices an $I^2C$ address is assigned to each device. In $I^2C$ also multiple masters on one bus are allowed. This is called multi master mode. Details about $I^2C$ can be found in "The $I^2C$-Bus Specification and User Manual" [Sem12b] published by NXP. |

#### Secure Element / UICC Interface

In this thesis only the peer to peer functions of NFC controllers are used. Peer to peer mode does not use a secure element or UICC so those interfaces will not be described here.

#### Adaptions

To reach lower chip size and power consumption the NFC controller hardware and firmware can be adapted or optimized. The goal of the firmware optimization is to lower the memory requirement of the program code. This allows to reduce the size of the ROM and EEPROM and therefore the chip size can be reduced. The optimization of the firmware also helps to identify areas of potential hardware optimizations.

The adaption of the NFC controller hardware starts with obvious elements like the hardware for the unused interfaces for secure element and UICC communication. Other unused components, like for unused communication modes in the CIU, are not so obvious but can also help to decrease the chipsize and power consumption of the NFC controller.

A detailed description of the software modifications made within this thesis are described in the implementation chapter. Hardware modification are out of scope of this thesis.

### 3.2.3 Host System

The host system can be any computer system supporting one of the host interfaces the NFC controller offers. Currently most NFC controllers are used in mobile phones. Therefore the host system often contains a processor based on the ARM Architecture[Ltd13]. As the goal of this thesis is to design a NFC controller for non-mobile devices various other host architectures will be used.

The Host systems used for tests and measurements in this thesis are of two Raspberry Pi single board computers. The Raspberry Pi boards handle the TCP/IP stack and communication with the NFC controllers via their onboard SPI interface. This structure is described more detailed in the implementation chapter.

### 3.2.4 Used Communication Protocols

#### Logical Link Control Protocol (LLCP)

The basics of the Logical Link Control Protocol have been described in chapter 2, Related Work. The components to realize LLCP functionality in an NFC controller are described in the following pages.

LLCP provides two connection types, connection-oriented and connection-less transport. As logical connections are handled by the high level protocols, for example TCP/IP, only the connection-less mode is implemented.

The main parts to be handled by the LLCP implementation are:

- LLCP connection setup
  - LLCP parameter exchange
  - LLCP Version number agreement procedure
  - Link Maximum Information Unit (MIU) determination procedure
- LLCP symmetry procedure
  - Send an LLCP after the local timeout, if no data to send
  - End the LLCP connection after the remote timeout if no data or symmetry frame is received
- Link deactivation procedure
  - Intentional link deactivation
  - Deactivation if a LLCP DISC Frame is received

#### LLCP Connection Setup

The connection setup procedure is required to ensure that both communication partners run compatible LLCP versions and optionally negotiate the MIU length. Therefore some LLCP parameters have to be exchanged. As LLCP can be mapped to various MAC layers, the LLCP parameters can either be exchanged with Parameter Exchange (PAX) PDUs or the data gets exchanged during the MAC layer activation. As in this thesis the NFC-DEP is used the parameter exchange happens via the general bytes during NFC-DEP activation.

The LLCP parameters specified in the LLCP specification document [For11] are shown in Table 3.2. As just the connection-less mode is used most of these parameters are not used. The used Parameters (VERSION and LTO) are highlighted green in Table 3.2. The link timeout parameter (LTO) is used by the symmetry procedure.

The Parameters are encoded in Type-Length-Value (TLV) format. The data format of the version number (VERSION) and Link Timeout (LTO) parameters are shown in Table 3.3 and Table 3.4. The VERSION parameter value consists of four bits for the Major version number and four bits for the minor version number. The value of the link timeout parameter is given in multiples of 10 milliseconds.

| Parameter | Included | In PDU Type | Used in Thesis |
|---|---|---|---|
| Maximum Information Unit Extension<br>MIUX | MAY | PAX | NO |
| Well-Known Service List<br>WKS | SHOULD | PAX | NO |
| Version Number<br>VERSION | SHALL | PAX | YES |
| Link Timeout<br>LTO | MAY | PAX | YES |
| Receive Window Size<br>RW | MAY | PAX | NO |
| Service Name<br>SN | MAY | PAX | NO |
| Option<br>OPT | MAY | PAX | NO |
| Service Discovery Request<br>SDREQ | MAY | PAX | NO |
| Service Discovery Response<br>SDRES | MAY | PAX | NO |

Table 3.2: LLCP Parameters. (Based on [For11])

| Version Number (VERSION) | | | |
|---|---|---|---|
| Type | Length | Value | |
| 0x01 | 0x01 | Major | Minor |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte 0 | | | | | | | | Byte 1 | | | | | | | | Byte 2 | | | | | | | |

Table 3.3: Format of the VERSION Parameter TLV. (Based on [For11])

| Link Timeout (LTO) | | |
|---|---|---|
| Type | Length | Value |
| 0x04 | 0x01 | LTO |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte 0 | | | | | | | | Byte 1 | | | | | | | | Byte 2 | | | | | | | |

Table 3.4: Format of the LTO Parameter TLV. (Based on [For11])

**LLCP Version Number agreement Procedure**
The version agreement is done by both communication partners. The local version number is compared with the version number received from the remote partner. To find the correct LLCP version to use the following rules have to be followed:

- "In the case where the major release values are identical and the minor release values are identical then version agreement SHALL be achieved." [For11]
- "In the case where the major release values are identical but the minor release values are not identical then version agreement SHALL be achieved. The agreed minor release value SHALL then be the lower of the two exchanged minor release values." [For11]
- "In the case where the major release values are not identical then the LLC with the higher major release number SHALL decide if version agreement is possible. If version agreement is possible, the agreed LLCP version SHALL be the lower of the two version values exchanged (e.g., LLCs with versions 2.3 and 1.7 agree on version 1.7)." [For11]

After the LLCP version number agreement both communication partners shall behave like specified in the agreed LLCP version.

**Link Maximum Information Unit (MIU) determination procedure**
The MIU describes the maximum length of the information field in an LLCP PDU. The default MIU value is 128 bytes. To operate with higher MIU values the Maximum Information Unit Extension (MIUX) can be exchanged during the parameter exchange procedure. If an endpoint supports information unit fields larger than the default value, the MIUX parameter can be sent by this endpoint. The MIU is then extended by the value of MIUX. If no MIUX is sent the MIU is assumed to be the default value. [For11]

**LLCP PDU Format**
In this paragraph the used LLCP PDUs are described. A full description of all LLCP PDUs can be found in "Logical Link Control Protocol, Technical Specification" [For11].

Table 3.5 shows the LLCP frame format. The Length of the LLCP header depends on the PDU type. Only PDUs for connection oriented communication need the sequence field. Hence all the PDUs used within this thesis do not need this field and therefore have a header length of two bytes.

The fields Destination Service Access Point Address Field (DSAP) and Source Service Access Point Address Field (SSAP) are the so called address fields. Each LLCP PDU contains these fields. The address zero (0x000000) must not be used to identify a service address point as it is used to identify the LLCP link management component. The PDU Type (PTYPE) field contains 4 bit to identify the PDU type. A List of all PTYPE values can be found in [For11]. The PTYPE values used within this thesis are described in the format information of the commands described later. The information field contains the LLCP payload. The length of the information field can be zero up to the length specified by the MIU

| LLCP PDU Format | | | | |
|---|---|---|---|---|
| LLCP Header | | | | LLCP Payload |
| DSAP | PTYPE | SSAP | Sequence | Information |
| 6 Bits | 4 Bits | 6 Bits | 0 or 8 Bits | m x 8 Bits |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ... | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte 0 | | | | | | | | Byte 1 | | | | | | | | Byte 2 | | | | | | | | Byte 3 ... Byte n | | | | | | | | | | | | | | | | |

Table 3.5: LLCP PDU Format. (Based on [For11])

### Unnumbered Information (UI) PDU

Table 3.6 shows the frame format of this PDU.

| LLCP Unnumbered Information (UI) PDU | | | |
|---|---|---|---|
| DSAP | PTYPE | SSAP | Information |
| DDDDDD | 0011 | SSSSSS | m x 8 Bits |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ... | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte 0 | | | | | | | | Byte 1 | | | | | | | | Byte 2 ... Byte n | | | | | | | | | | | | | | | | |

Table 3.6: LLCP Unnumbered Information (UI) PDU. (Based on [For11])

The unnumbered information PDU is used for all LLCP data transfers in the implementation described in this thesis. The information field of this PDU type can also be empty.

### Symmetry (SYMM) PDU

If there is no data to be sent, the LLCP symmetry procedure uses this PDU type to ensure asynchronous balanced mode. Like shown in Table 3.7 the PDU is a minimal LLCP header with all bits set to zero.

| LLCP Symmetry (SYMM) PDU | | |
|---|---|---|
| DSAP | PTYPE | SSAP |
| 000000 | 0000 | 000000 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte 0 | | | | | | | | Byte 1 | | | | | | | |

Table 3.7: LLCP Symmetry (SYMM) PDU. (Based on [For11])

### Disconnect (DISC) PDU

"The DISC PDU is an unnumbered PDU which is used to terminate a data link connection or is used to deactivate the LLCP Link." [For11] This PDU is described in Table 3.8.

| LLCP Disconnect (DISC) PDU | | |
|---|---|---|
| DSAP | PTYPE | SSAP |
| DDDDDD | 0101 | SSSSSS |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte 0 | | | | | | | | Byte 1 | | | | | | | |

Table 3.8: LLCP Disconnect (DISC) PDU. (Based on [For11])

### NCI

The communication interface between an NFC controller and the host controller is specified in "NFC Controller Interface (NCI) Specification"[For12] by the NFC forum. NCI is independent of a specific transport layer (a physical connection and any associated link protocol), as shown in Figure 3.5. Details on how to run NCI using various transport layers are defined via NCI transport mappings. The parts handled by NCI are highlighted green.

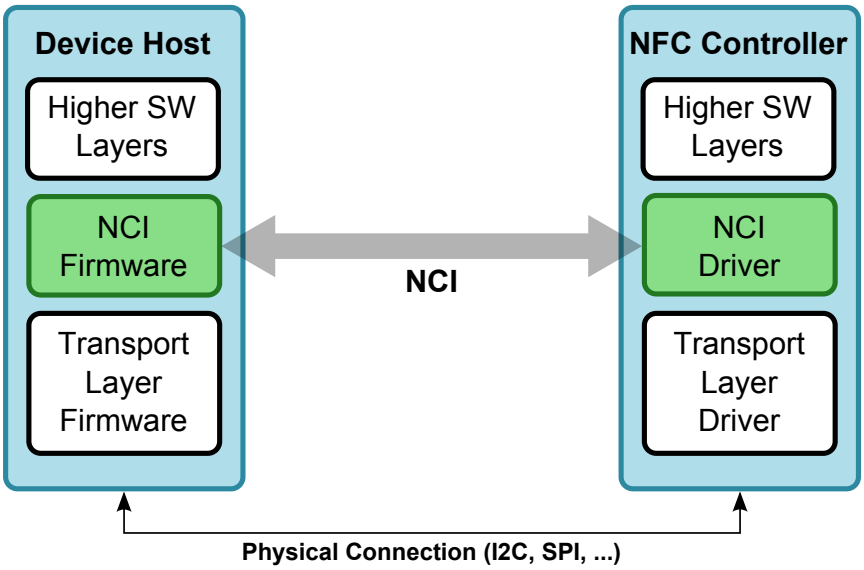


Figure 3.5: NCI Interface Structure. (Based on [For12])

The parts of NCI used within this thesis are described in the following paragraphs. For more detailed information refer to the NCI specification.

### NCI Components

NCI consists of three main logical components:

NCI Core: The NCI core handles the communication basics between the host controller and the NFC controller. Therefore control and data messages are supported. Control messages are command, response and notification messages.

Transport Mappings: As described earlier NCI can work on top of various transport layers. Therefore Transport mappings are used to link NCI to the used transport layer (physical connection and associated protocol).

NCI modules: The basic functionality provided by the NCI core can be extended with NCI modules. For example such modules are used for NFCC configuration and NFC endpoint communication.

### NCI Messages

In NCI two message types are defined. Messages can be control or data messages. Control messages can be commands, responses or notifications. Commands are used by the device host to configure the NFC controller. Responses and notifications are used by the NFC controller to answer received commands or inform the device host on events. The format of control packets is shown in Table 3.9.

| NCI Control Packet | | | | | | |
|---|---|---|---|---|---|---|
| MT | PBF | GID | RFU | OID | Payload Length (L) | Payload |
| 3 Bits | 1 Bit | 4 Bits | 2 Bits | 6 Bits | 8 Bits | m x 8 Bit |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ... | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte 0 | | | | | | | | Byte 1 | | | | | | | | Byte 2 | | | | | | | | Byte 3 ... Byte 2 + L | | | | | | | | | | | | | | | | |

Table 3.9: NCI Control Packet

Data Messages are used for data transport. Long data messages can be segmented into multiple data packets. Before data packets can be sent a logical connection between two endpoints has to be established. A logical connection for RF communication is established by default during NCI initialization. The packet format of data messages is described in Table 3.10.

| NCI Data Packet | | | | | |
|---|---|---|---|---|---|
| MT | PBF | Conn ID | RFU | Payload Length (L) | Payload |
| 3 Bits | 1 Bit | 4 Bits | 8 Bits | 8 Bits | m x 8 Bit |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ... | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte 0 | | | | | | | | Byte 1 | | | | | | | | Byte 2 | | | | | | | | Byte 3 ... Byte 2 + L | | | | | | | | | | | | | | | | |

Table 3.10: NCI Data Packet

The shortcuts used in the packet format tables are:

| | |
|---|---|
| Message Type (MT): | This value indicates the type of the message. The value is coded in three bits (0b000 ... data packet, 0b001 control packet with a command message, 0b010 control packet with a response message, 0b011 control packet with a notification message, other values are Reserved for further Usage (RFU)) |
| Packet Boundary Flag (PBF): | This Flag is used for segmentation and reassembly. A value of 0b0 indicates a package containing a complete message or the last segment of a message, 0b1 indicates segments of a not complete message. |
| Group Identifier (GID): | NCI commands, responses and notification are categorized into groups. The group identifier is used to indicate the messages group. |
| Opcode Identifier (OID): | The opcode identifier identifies the control messages within their groups. |
| Connection Identifier (Conn ID): | This is used to identify the logical connection that the data message belongs to. |
| Payload Length (L): | The payload length value indicates the number of payload octets including the payload length octet. So the value has to be the number of payload octets plus one octet for the payload length value. |

### NCI Interfaces

One of the main concepts in NCI is the binding of RF protocols to RF interfaces. Interfaces describe the communication between the device host and a remote endpoint. Interfaces can either support communication with an RF endpoint or with an NFC Execution Environment (NFCEE).

As showed in figure 3.6 there are two interfaces foreseen for LLCP. These are the LLCP-Low and LLCP-High interfaces.

At the moment neither of these two interfaces is fully specified by the NFC forum. Therefore within this thesis an own specification for the two interfaces will be created and used. For the LLCP-Low interface a draft version exists. As this draft is not complete it is used as a starting point for the own specification of the interface.

### LLCP Low Interface

The draft specification for the LLCP-Low interface only takes care of the LLCP symmetry mechanism. The LLCP link activation and connection setup has to be done by the host controller. Hence an activation of this interface is the same as activating the NFC-DEP RF Interface. After the interface is active the host controller is able to send NCI Data packets to the NFCC. The payload of these packets is sent to the remote NFCC

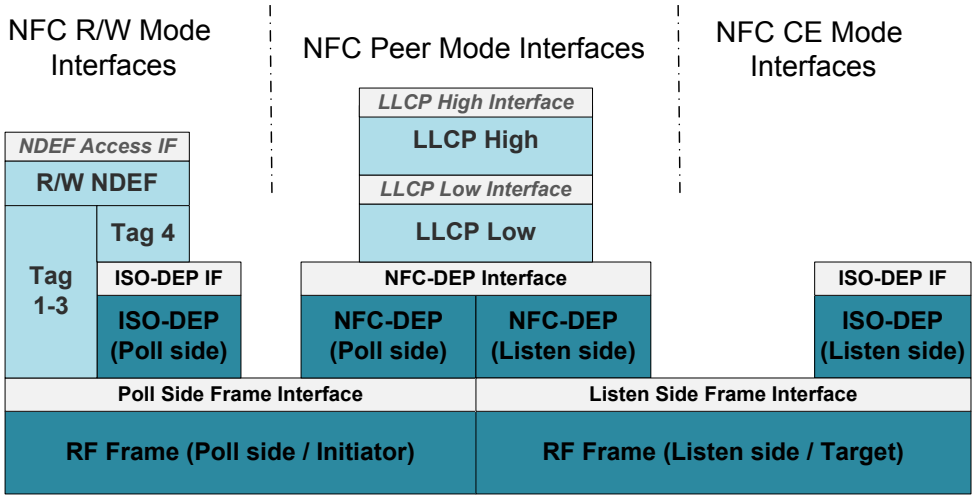


Figure 3.6: RF Interface Architecture. [For12]

wrapped in an NFC-DEP frame and therefore the information is provided to the remote host controller as an NCI data message. Due to the command response principle of NFC-DEP communication the target host can only send data as a response to a message from the initiator. So if the initiator host has no data to send the target host is not able to send data.

To avoid this the LLCP-Low interface takes care of the LLCP symmetry procedure. This means that if the initiator has no data to send, after a specified idle time an LLCP SYM-PDU is sent to the target by the NFCC. The target NFCC can send data to the initiator as response to this LLCP SYM-PDU or another LLCP SYM-PDU if no data is to be sent.

When using this interface the LLCP protocol handling, except the symmetry procedure, has to be done by the host controller. Hence the LLCP version of the symmetry procedure has to be sufficient for the used version of LLCP on the host controller. To check this the host is able to retrieve the supported LLCP version from the NFCC by reading the read-only configuration parameter LLCP_VERSION. The format of this parameter is shown in table 3.11.

| LLCP_VERSION | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit Mask | | | | | | | | Description |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | |
| X | X | X | X | | | | | LLCP Major Version |
| | | | | X | X | X | X | LLCP Minor Version |

Table 3.11: LLCP Version Parameter

To control the symmetry procedure the interface provides two NCI control messages RF_LLCP_SYMMETRY_START_CMD and RF_LLCP_SYMMETRY_STOP_CMD. Tables 3.12 and 3.13 show the commands and the corresponding responses.

| RF_LLCP_SYMMETRY_START_CMD | | |
|---|---|---|
| Payload Field(s) | Length | Description |
| Remote Link Timeout | 1 Octet | An 8-bit unsigned integer that specifies the value of the Remote NFC Endpoint's link timeout. In order to align with [For11], the value is expressed in multiples of 10 ms. |
| Local Symmetry Timeout | 1 Octet | An 8-bit unsigned integer that specifies the value of symmetry timeout. In order to align with [For11], the value is expressed in multiples of 10 ms. |

| RF_LLCP_SYMMETRY_START_RSP | | |
|---|---|---|
| Payload Field(s) | Length | Description |
| Status | 1 Octet | STATUS_OK (0x00) if successful or STATUS_FAILED (0x03) if failed |

Table 3.12: LLCP Symmetry Start NCI Control Messages

| RF_LLCP_SYMMETRY_STOP_CMD | | |
|---|---|---|
| Payload Field(s) | Length | Description |
| No Payload | | |

| RF_LLCP_SYMMETRY_STOP_RSP | | |
|---|---|---|
| Payload Field(s) | Length | Description |
| Status | 1 Octet | STATUS_OK (0x00) if successful or STATUS_FAILED (0x03) if failed |

| RF_LLCP_SYMMETRY_STOP_NTF | | |
|---|---|---|
| Payload Field(s) | Length | Description |
| Status | 1 Octet | LLCP_SYMM_STOP (0x0D) |

Table 3.13: LLCP Symmetry Stop NCI Control Messages

An end to end LLCP-Low connection is established by the following steps:

Target LLCP-Low activation: On the target side the host controller sets up the NFCC to listen mode by sending NCI configuration, map and discover commands. The target NFCC now waits for an incoming connection.

Initiator LLCP-Low activation: On the initiator side the host controller sets up the NFCC to polling mode by sending NCI configuration, map and discover commands. The initiator NFCC now polls for targets in its RF range.

Connection established: After the initiator NFCC finds a target in range the connection is established and both the initiator and the target host are informed with an NCI interface activated notification.

| | |
|---|---|
| LLCP Activation: | If the interface activation is successful, the DH uses the general bytes received in ATR_RES to determine whether LLCP link activation is possible. If yes, the target host controller now starts the LLCP Activation by performing the MIU requirements and link activation procedures described in "Logical Link Control Protocol, Technical Specification" [For11]. |
| Start Symmetry procedures: | Now both the target and the initiator host controller can start the symmetry mechanism in the NFCC by sending the LLCP symmetry start command over NCI, shown in table 3.12. The NFCC replies with a STATUS_OK or STATUS_FAILED response message. If a host controller does not start the symmetry procedure in the NFCC the host controller has to generate and handle LLCP SYM frames. |
| LLCP Communication: | After starting the symmetry procedure on both sides asynchronous balanced mode is active. Therefore both the target and the initiator host controllers are able to send LLCP packages at any time. If no LLCP SYM or data packet is received within the specified timeout, the host controller is informed by the NFCC via a LLCP symmetry stop notification (Table 3.13). |
| LLCP Stop Symmetry procedure: | A host controller can end the connection by sending an LLCP symmetry stop command to the NFCC via NCI. The NFCC then sends an LLCP disconnect message to the remote NFCC and informs the host controller by sending an LLCP symmetry stop notification. The remote NFCC also informs it's host controller with an LLCP symmetry stop notification. |

### LLCP High Interface

The LLCP High interface is also not yet specified by the NFC forum. As there is even no draft version of the specification available from NFC forum, an own specification for this interface is made in this paragraph. With the LLCP-Low interface, the NFCC handles the LLCP symmetry procedure but the host controller has to take care of the LLCP link activation and the LLCP frame format. To enable peer to peer communication for hosts without any knowledge of the LLCP protocol, the NFCC shall handle the complete LLCP logic including connection setup, the symmetry procedure and the link deactivation procedure. This means, once the LLCP-High interface is activated by the hosts, they can send and receive data in ABM mode. Like the LLCP-Low interface the LLCP High interface is activated in a similar manner to the NFC-DEP RF Interface.

As described in the LLCP link activation procedure the LLCP magic number, the LLCP version, and the value for the link timeout are used for the LLCP connection setup and symmetry procedures. Therefore this information has to be provided to the NFCC before activating the interface. This is done via a CORE_SET_CONFIG_CMD NCI command. Within this command a parameter called PN_ATR_REQ_GEN_BYTES

is used to set the data to be used as general bytes during the NFC-DEP RF Interface activation.

To set up an end to end connection using the LLCP-High interface the following steps have to be done:

| | |
|---|---|
| Setting the general bytes: | Set the general bytes including the LLCP magic number, the LLCP version and the link timeout via an NCI command, called CORE_SET_CONFIG_CMD, on target and initiator side. |
| Target LLCP-High activation: | On the target side the host controller sets up the NFCC to listen mode by sending configuration, map and discover NCI commands. The target NFCC now waits for an incoming connection. |
| Initiator LLCP-High activation: | On the initiator side the host controller sets up the NFCC to polling mode by sending NCI configuration, map and discover commands. The initiator NFCC now polls for targets in its RF range. |
| Connection established: | After the initiator NFCC finds a target in range the NFC-DEP is established. The LLCP link setup is done by the NFCCs using the information exchanged with the general bytes. If the link setup is successful the symmetry procedure is started. After receiving the first LLCP symmetry frame both the initiator and the target host are informed with an NCI interface activated notification. |
| LLCP Communication: | After receiving the interface activated notification both sides, the target and the initiator, are able to send LLCP packages at any time. Similar to the LLCP-Low interface an LLCP symmetry stop notification (Table 3.13) is sent to the host if no LLCP SYM or data packet is received within the specified timeout. |
| Closing the Connection: | The connection can either be closed intentionally by either the target or initiator host using an NCI deactivate command or by a loss of the RF link (for example by bringing the antennas out of range). In both situations the LLCP symmetry procedure is stopped by the NFCCs and the hosts are informed via an NCI interface deactivated notification. |

# Chapter 4

# Implementation

Like shown in Figure 3.1 the implemented system consists of two NFC-Controllers connected to Host-Controllers. To provide the peripherals and RF Hardware needed by the NFC-Controller matching Evaluation Boards are used. As mentioned earlier the goal of this thesis is to use high level communication Protocols over NFC. Therefore the Host-Controllers will run a Linux operating system and include the NFC-Controller into the Network-Stack via a Network Card Device driver.

## 4.1 Hardware

For the implementation two PN547 NFC Controllers from NXP are used and therefore NXP development tools are also used. NXP also provided two evaluation boards for the PN547 called PN547 EvalBoard Light 2.0. Two Raspberry Pi Boards [Fou13] were used as Host-Controllers, because of their ARM Architecture, low price and Linux Operating System. The hardware used is described in more detail in the next paragraphs.

### 4.1.1 NFC-Controller PN547

Figure 4.1 shows a basic overview of NXP's PN547 NFC Controller. The device's Microcontroller Core, Memory and Host interface are described more detailed later. Other components like the Secure Element / UICC Interface, the Contactless Interface Unit or the Power and Clock Management Unit are not described as they are either not relevant for this implementation or describing them will be too extensive for the scope of this thesis.

#### Microcontroller Core

For the Microcontroller core an ARM Cortex-M0 core is used. This Core is based on the ARMv6-M architecture. It is a 32 Bit Processor supporting most of the instructions defined in ARM's Thumb[1] and a subset of the Thumb2[2] Instruction-Sets. The Core is optimized for low chip size and power consumption [Lim12]. The included SWD Interface enables on chip debugging.

[1] CBZ, CBNZ, IT and the divide instruction.
[2] BL, DMB, DSB, ISB, MRS and MSR.

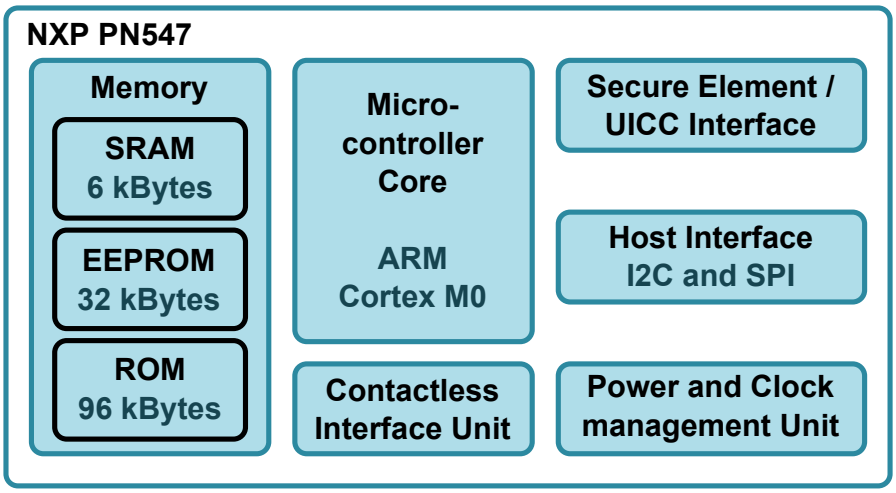


Figure 4.1: PN547 Hardware Architecture

### Memory

The PN547 contains 6 kBytes of Static Random-Access Memory (SRAM), 32 kBytes of Electrically Erasable Programmable Read-Only Memory (EEPROM) and 96kBytes of Read Only Memory (ROM). 4 kBytes of the EEPROM are used as DATA EEPROM used for persistent storage of data and 28 kBytes are used as program code storage. The 96 kBytes ROM are also used for program code storage. As EEPROM needs about five times more chip size per kByte than ROM, and code execution speed is higher for ROM code, most of the firmware program code is allocated in ROM. This saves costs but the code in ROM can only be changed with a new tape out, this code has to be stable and well tested.

### Host Interface

The PN547 provides $I^2C$ as well as SPI interfaces for communication with the host. As these two interfaces share the same interface buffers and also some I/O Pins, can not be used at the same time. The interface to be used and settings such as communication modes and speed are set via an entry in the DATA EEPROM.

For this implementation the SPI interface is used as it is also available and well documented on the used host controller.

Although SPI is able to operate in full duplex mode, it is implemented in a half-duplex mode on the PN547. Therefore the first byte sent from the host to the NFC-Controller indicates if the following bytes shall be sent from the host to the NFC -Controller, or if the hosts wants to read from the NFC-Controller. The PN547 indicates data to be read by setting the Interrupt Request (IRQ) pin. This mechanism if described in the chapter Implementation Details later in this thesis.

### System Clock

The PN547 can either generate the system clock via an external crystal or use a clock signal provided by the host system. The system clock source is also set by an entry in the DATA EEPROM.

## 4.1.2 Evaluation Board

As the PN547 NFC-Controller needs some external components to work, it is used as part of an Evaluation Board. This Board is shown in Figure 4.2.

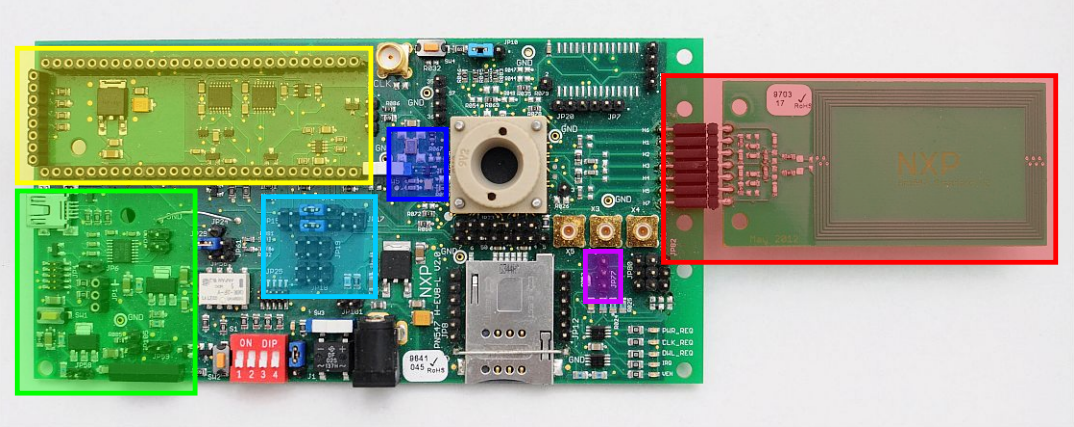

Figure 4.2: PN547 EvalBoard Light 2.0

These external components are the Antenna Matching Circuit, Antenna, Power-Supply and System-Clock generation as well as connectors for the Host Interface and the IRQ Pin. These components are highlighted in Figure 4.2 and further described later in this text.

Within NXP the Evaluation Board is used for testing during firmware development. Therefore a Microcontroller board [3] can be mounted on top of the Evaluation board. The Microcontroller board is connected to a Personal Computer via Universal Serial Bus (USB) and is used as an interface between a Firmware Testbench, operating on a PC and the PN547, as well as a platform for directly running time critical or timing tests.

Antenna and Matching Circuit: Marked with red. The used antenna is a 6 Loop antenna with dimensions of 50mm x 35mm. As the NFC-Controller can be used with antennas in various form factors the Antenna Impedance has to be matched with the Impedance needed by the NFCC for best performance. This is done via the Antenna matching circuit. This circuit also contains an Electromagnetic Compatibility (EMC)-Filter.

Power-Supply: Marked with green. The PN547 evaluation board can operate with a broad Range of Supply Voltages provided either via a micro-USB connector or a dedicated

[3] A standard NXP LPCxpresso Microcontroller board.

power connector. From this Supply either 3.3 Volts or 1.8 Volts are generated for the PN547. For the tests done during this thesis, the Evaluation Board is powered by 5 Volts from the mini-USB port and an operating voltage of 3.3 Volts is used for the PN547.

System-Clock generation: Marked with blue. The Evaluation Board provides two possibilities for clock generation. The System cock is either generated by the NFC- Controller using a crystal or by the Evaluation Board using its built in oscillator. For this implementation the Evaluation Boards oscillator is used.

Host Interface Connectors: When using the Evaluation Board with the LPCxpresso Microcontroller board like mentioned before, the Host Interface I/Os of the NFC-Controller are shifted to the power supply voltage of the Microcontroller board via level shifters. The connector for the LPCxpresso Microcontroller board is marked with yellow. As in this implementation the Host Controller as well as the NFC-Controller are running at the same supply voltage these level shifters are not used and the Host Interface signals are directly connected to the NFC-Controller using the connectors marked in the image. These connectors are marked with cyan.

IRQ Pin Connector: Marked with magenta. The IRQ Signal is set by the NFC-Controller to inform the Host Controller about data to be fetched from the Host Interface. Once all available data is read the signal is reset by the NFC-Controller. Like the Host Interface signals also the IRQ signal can be level shifted, but like the Host Interface signals this signal is used without level shifting in this Implementation.

### 4.1.3 Host Controller

As already stated, two Raspberry Pi Boards are used as Host Controllers. As they were already available two of the more powerful (in terms of RAM and I/O possibilities) Raspberry Pi Model B Revision 1.0 boards are used. The board is shown in Figure 4.3

#### Hardware Specifications

The Raspberry Pi board is based on a Broadcom BCM2835 SOC [Bro13]. This SOC contains a CPU-Core, a Graphics Processing Unit (GPU), a Digital Signal Processor (DSP) as well as the Synchronous Dynamic Random Access Memory (SDRAM) and a single port USB-Controller.

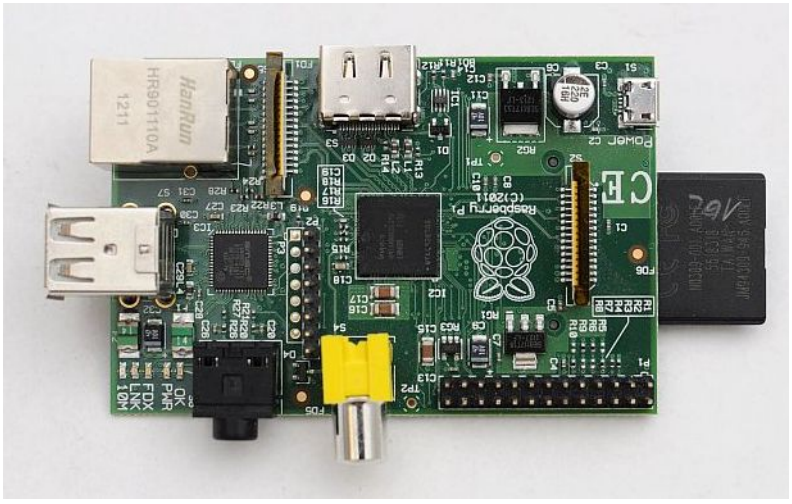

Figure 4.3: Raspberry Pi Model B Revision 1.0 Board

| | |
|---|---|
| CPU: | ARM1176JZF-S CPU-Core running at 700 MHz. |
| GPU: | Broadcom VideoCore IV running at 250 MHz. |
| SDRAM: | 512 MB SDRAM shared with the GPU. |
| USB-Ports: | Onboard 3 Port USB-Hub, of which one Port is used by the onboard Network Card, two further ports are available for peripherals. |

For network connectivity a 10/100 MBit Ethernet USB adapter is mounted on the board and connected to a port of the 3 port USB- Hub. Beside the various Audio and Video In- and Outputs also several low level peripherals are available. 17 GPIOs are accessible on a 2 x 13 Pin Pinheader. Two of them are also used for a UART and two other GPIOs are used for a $I^2C$ interface. There is also a SPI interface available. The interface contains two data signals (MOSI and MISO) a serial clock (SCLK) and two chip-select signals. These pins are also shared with GPIOs. Figure 4.4 shows the GPIO pinout.

### Mass Storage

The Raspberry Pi is designed to boot the operating system from a Secure Digital (SD)-Card, therefore it provides an SD-Card Slot. Also user data can be stored on this SD-Card. Additional data can also be stored on hard disks or flash drives connected to one of the USB ports. For this implementation an 8 GB SD-Card is used as operating system and user data storage for each of the two raspberry pi boards.

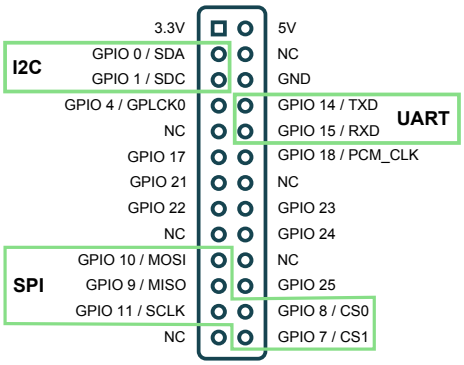


Figure 4.4: Raspberry Pi Model GPIO Pinout

### 4.1.4 Hardware Setup

As previously stated, the Raspberry Pi boards are connected to the PN547 NFC-Controllers via SPI. This is done using 6 wires. 4 Wires are for the SPI Signals MOSI(brown wire), MISO(red wire), SCLK(orange wire), and CS0(yellow wire). CS0 is the Chipselect Signal used for addressing the PN547 NFC-Controller. Of the additional two wires, the green one is connected to the IRQ-Pin of the NFC-Controller to inform the host controller about data to be read and the blue wire is used to connect the ground of the host controller and the NFC-Controller. This is shown in Figure 4.5.

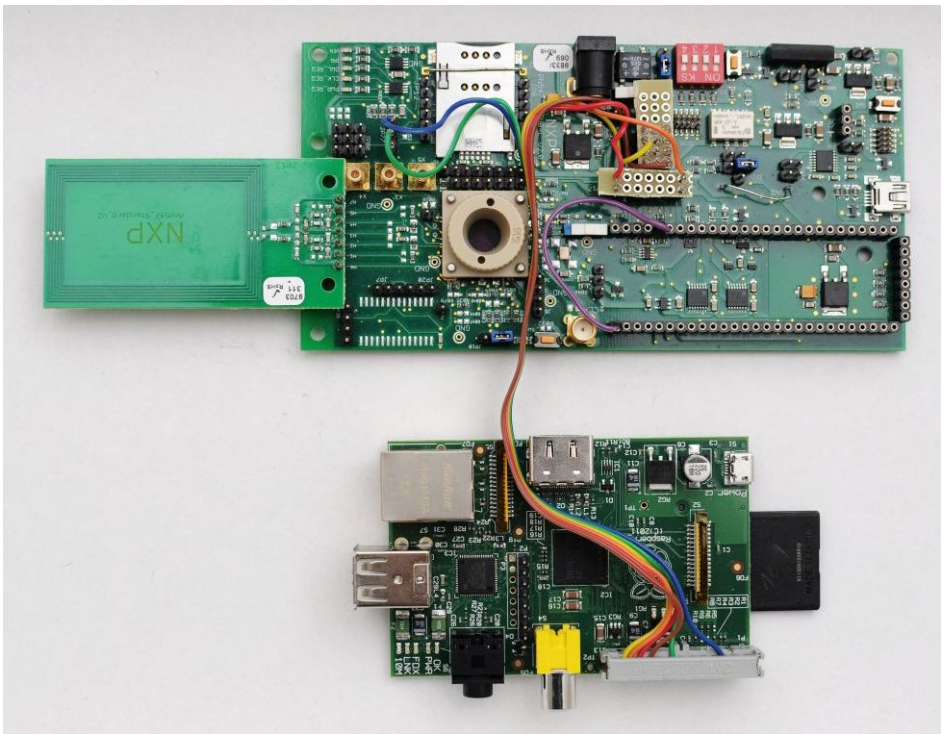

Figure 4.5: Hardware Setup Overview

Although the Raspberry Pi boards could be controlled using a PC-Monitor and a keyboard, for this implementation a Secure Shell (SSH)-Connection is used. Therefore the Raspberry Pi Boards are set up to use static IP-Addresses for the build in Network interface and connected to a laptop over an Ethernet switch.

The power supply of the PN547 Evaluation Boards as well as the Raspberry Pi Boards can be done using their build in USB-Ports. On the Raspberry Pi Boards a micro-USB and on the PN547 Evaluation Board a mini-USB connector is used. The power for all the four boards is provided by a external powered USB-Hub. As this USB hub is only used for power supply it is not connected to another PC.

## 4.2 Software

### 4.2.1 SW-Architecture of the PN547

As the NXP PN547 NFC-Controller is a feature rich NFC controller the Firmware architecture is quite complex and describing the whole architecture would be beyond the scope of this thesis. Therefore only the parts directly related to the implementation are described. The main programming language used is ANSI C, but some parts are also written in Assembler.

#### Operating System

To handle the various tasks from the different interfaces that the NFCC has to take care of, the Firmware is split into several parallel running processes. As the NFCC CPU-Core is a single core CPU the computing time has to be shared by the different processes. Also the system resources, for example memory or GPIOs, have to be managed by an operating system. To fulfill timing constraints for card emulation or some RF protocols a Real Time Operating System (RTOS) is used. The RTOS itself is not developed by NXP rather an already existing RTOS is linked into the Firmware as an external component.

The used RTOS provides the following Features:

- Events enabling synchronization between two processes with time-out functionality in order to avoid dead-locks.
- Messages for synchronization and message exchange between two processes.
- Message queues (mailboxes) with implemented conflict management.
- Semaphores for access control.
- Unlimited number of software timers.
- Preemptive and cooperative scheduling.

### Software Processes

In its normal operation mode the PN547 firmware consists of four processes. The processes are initialized and started each time the Firmware enters this operation mode. Three of the four processes are serving the physical interfaces, the SWP-Interface process, Host-Interface process and RF-Interface process. The fourth process is needed to manage the data flow between the interface processes and is called the Kernel process. The Kernel process handles all data flow between the interfaces except for the card emulation use case. In this case the data flow is directly from the RF process to the SWP process in order to meet timing requirements. Figure 4.6 shows the processes. The arrows are indicating data exchange via RTOS messages.

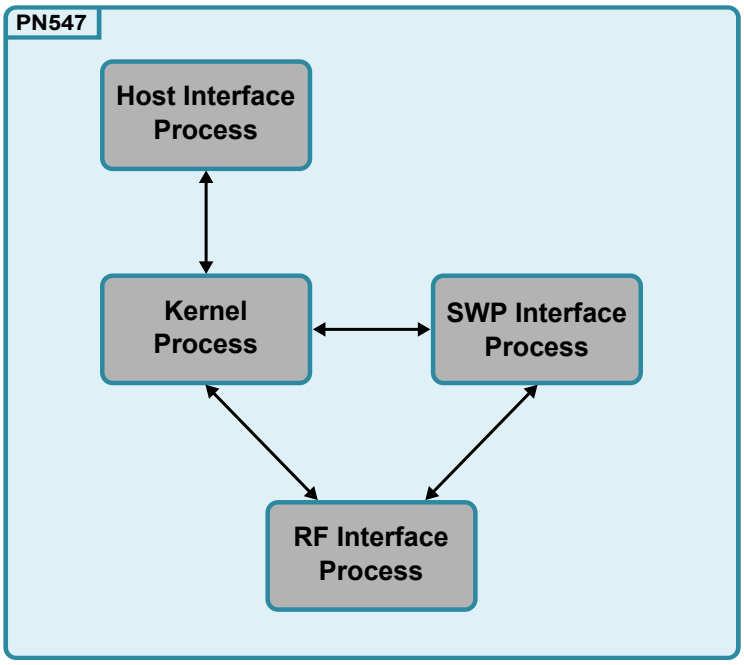


Figure 4.6: NXP PN547 Software Processes

The Host Interface process and the SWP Interface process are both triggered by incoming data either from the corresponding external interface or the Kernel process. All received frames are decoded / checked according to the data link protocol. Valid frames are either forwarded to the Kernel process or the corresponding interface. The System Kernel Process executes all received commands from the Host Interface and the SWP Interface. Data exchange commands are forwarded to the process which handles the corresponding external interface. The data which is transferred over RF field is handled by the RF Interface process. [Sem11]

Each process has its own mailbox. In idle mode the processes check if there are messages in their mailbox. If yes the message is dispatched and a firmware module to handle the message is called. The module handling the message can then either post a message to another process or to the process the module is called from.

#### Inter Process Communication

As previously mentioned the Inter Process Communication (IPC) is made with messages and mailboxes for each process. For data exchange between the processes shared data structures are used. The firmware works with a fixed number and structure of these buffers. One of the main goals of this approach is to avoid copying data from one memory area to another. For example data received by the Host-Interface can be sent by the RF-Interface without copying the data by sharing the data structure allocated by the Host-Interface. To ensure that the buffer is not reused by the Host-Interface before the data is sent by the RF-Interface a Buffer Management Module is implemented in the Kernel process for managing allocation and release of buffers. With this module, for example, the Host-Interface process requests the Kernel process to lock a Buffer via an RTOS message, fills the buffer with data, and shares the buffer with the Kernel process. The Kernel process identifies the data as data to be sent by the RF-Process and passes the buffer to the RF-Process. After the sending of the frame is completed the RF-Process posts a release buffer message to the Kernel process and the buffer is marked as free by the Buffer Management.

### 4.2.2 SW-Architecture of the Host PC

Like for the NFC-Controller it would be beyond the scope of this thesis to describe the complete software architecture of the Raspberry Pi boards used as Host PCs. Hence only the parts used in this thesis are described. These are mainly the Kernelspace / Userspace separation, handling of Kernel modules and the Network driver structure.

The operating system used for the Raspberry Pi boards is Linux based and called Raspian. “Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make your Raspberry Pi run. However, Raspbian provides more than a pure OS: it comes with over 35,000 packages, pre-compiled software bundled in a nice format for easy installation on your Raspberry Pi.” [TG13]

The core of an operating system is called the Kernel. The kernel is responsible for managing the computer’s hardware like CPU, memory and I/O-Devices. The Kernel performs operations considered critical for the Systems stability, therefore it is loaded into a protected memory area and is protected from being overwritten by applications. The operations performed by the Kernel are executed in this memory area, called “Kernel Space”. User applications are executed in a different Memory area, called “User Space”. [Pro13a] “The Kernel provides basic services for all other parts of the operating system, typically including memory management, process management, file management and I/O (input/output) management (i.e., accessing the peripheral devices). These services are requested by other parts of the operating system or by application programs through a specified set of program interfaces referred to as system calls.” [Pro13a]

The handling of various hardware devices within the Kernel is done by device drivers. The job of device drivers is to hide the complexity of the hardware to the Kernel. Therefore a device driver implements a set of standardized calls and maps them to the target hardware. The interface for device drivers is designed in a way that they can be swapped in or out during runtime. This makes the Kernel modular and therefore these swap able parts are called Kernel Modules. [CRKH05]

## 4.3 Implementation Details

### 4.3.1 NFC-Controller Firmware Adaptions

As described earlier the Firmware of the NXP PN547 NFC-Controller is divided into four processes. The main task for LLCP is inspecting and manipulating data packets received either from host or the contactless interface and sending symmetry data frames. The PN547 Kernel process manages all data exchange between the contactless and the host interface, therefore the LLCP-Implementation is done in the Kernel process. For the LLCP functionality a software module is implemented. Figure 4.7 shows the new module and the processes involved to provide the LLCP functionality.

As all LLCP communication is based on the NFC-DEP protocol the RF-Interface is set up the same way as when activating the NCI NFC-DEP Interface.

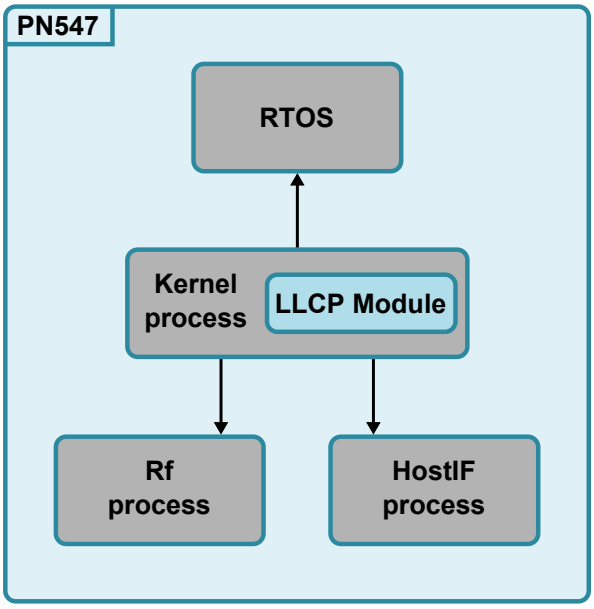


Figure 4.7: PN547 Firmware LLCP Module and Processes

#### Major user-visible Functions

The LLCP functionality is provided as NCI interfaces, therefore external communication is handled over NCI. The two new NCI Interfaces are implemented, like described in the Chapter System Design. The interfaces are the LLCP-High and LLCP-Low interfaces. The use case diagram Figure B.1 is attached in the appendix to illustrate the typical use cases for the newly introduced LLCP interfaces. The new user-visible software states are shown in Figure 4.8.

With these two new interfaces new user-visible functions are introduced. The new implemented functions are listed below:

- LLCP High Interface
  - Interface Start
  - Data exchange
  - Interface deactivation
- LLCP Low Interface
  - Interface Start
  - LLCP Symmetry Start
  - Data exchange
  - LLCP Symmetry Stop
  - Interface deactivation

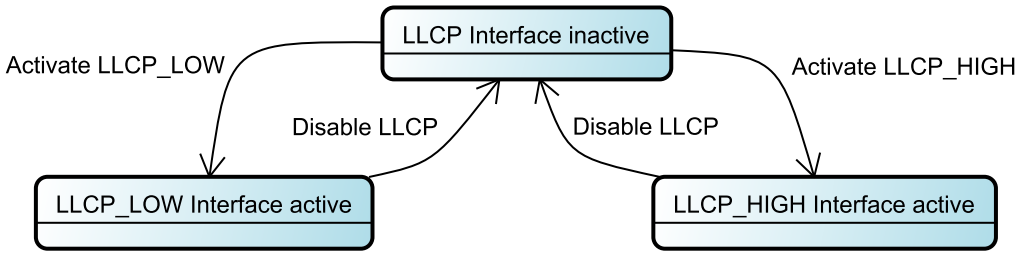


Figure 4.8: PN547 Firmware LLCP User visible Software States

To use one of the LLCP interfaces the host controller has to set the configuration parameters as for activating the NFC-DEP interface. The desired LLCP Interface is configured with the RF_DISCOVER_MAP command. Therefore two new values for RF-Interfaces (LLCP_LOW and LLCP_HIGH) are defined and shown in Table 4.1.

To control the LLCP Symmetry procedure with the LLCP_LOW interface activated several new NCI messages are added. They are described in Table 4.2.

| NCI RF-Interfaces | |
|---|---|
| Value | Definition |
| 0x04 | LLCP_LOW Interface |
| 0x05 | LLCP_HIGH Interface |

Table 4.1: New defined NCI Interfaces for LLCP

| NCI Messages | | |
|---|---|---|
| GID | OID | Message Name |
| RF Management 0b0001 | 0b1100 | RF_LLCP_SYMMETRY_START_CMD<br>RF_LLCP_SYMMETRY_START_RSP |
| | 0b1101 | RF_LLCP_SYMMETRY_STOP_CMD<br>RF_LLCP_SYMMETRY_STOP_RSP<br>RF_LLCP_SYMMETRY_STOP_NTF |

Table 4.2: Additional NCI Messages for LLCP

### LLCP Firmware Module

In the PN547 Firmware a software module is defined as one or more components used to provide a defined functionality. A component is one or more C functions in one C-Source file.

The LLCP firmware module consists of the following three components:

LLCP Symmetry (phLlcp_LlcpSymm): This component handles the sending of LLCP Symmetry Messages. Therefore a software timer is used to toggle the sending of a Symmetry Frame after the specified timeout. The timer can be reset if a data message has been sent.

LLCP Management (phLlcp_LlcpMgt): The LLCP Management processes data packages from host to RF and from RF to host. It also controls the symmetry timer and other LLCP components

LLCP Setup Manager (phLlcp_LlcpLinkSetupMgt): This component handles the LLCP Link Activation after an NFC-DEP connection is established, if the LLCP-High Interface is used.

Although the programming language C does not offer object orientated concepts the PN547 Firmware code is structured by using Structs and encapsulating functionality in Modules and Components. Figure B.2 in Appendix B shows an Object based decomposition of the LLCP Module.

To describe the dynamic behavior of the new software module several sequence diagrams are attached in Appendix B. The activation sequence for the LLCP-High as well as for the LLCP-Low interface are shown in Figure B.3 and Figure B.4. The sequences for data exchange are depicted in Figure B.5 and Figure B.6.

### Code Size and RAM Consumption

NFC-Controllers are sold in high quantities, therefore even small changes in the chip size resulting in higher or lower hardware costs can lead to a massive increase or decrease of the profit generated by the product. Therefore the available program code memory as well as the available RAM is limited by design and all firmware modules have to be designed to minimize these two values and still fulfill all functional and timing requirements. Table 4.3 shows the code size of the three components and the complete LLCP Module. The RAM consumption is shown in 4.4.

| PN547 LLCP Module Code Size | |
|---|---|
| Component | Code Size |
| phLlcp_LlcpMgt | 1020 Bytes |
| phLlcp_LlcpSymm | 174 Bytes |
| phLlcp_LlcpLinkSetupMgt | 348 Bytes |
| Total | 1542 Bytes |

Table 4.3: PN547 LLCP Module Code Size

| PN547 LLCP RAM Consumption | |
|---|---|
| Component | RAM Consumption |
| phLlcp_LlcpMgt | 20 Bytes |
| phLlcp_LlcpSymm | 28 Bytes |
| phLlcp_LlcpLinkSetupMgt | 5 Bytes |
| Total | 53 Bytes |

Table 4.4: PN547 LLCP Module RAM Consumption

### Integration into the existing Firmware

For the Interface Activation only a small change in the existing NCI Module is needed. As mentioned before both LLCP Interfaces are based on the NFC-DEP Interface and therefore the RF process shall not notice that LLCP is active in the Kernel and still setup NFC-DEP communication. The RF_DISCOVER_MAP command is analyzed in the Kernel process, if a mapping to a LLCP interface is found the corresponding flag in the kernel context is set, the LLCP interface is exchanged with NFC-DEP interface in the mapping table and the table is written to EEPROM. Therefore the existing Firmware sets up the RF system for NFC-DEP communication but the kernel process is aware that an LLCP interface is activated.

If an LLCP interface is activated the LLCP timer has to be reset every time data is sent by the host. If the LLCP-High interface is active also the LLCP-Header has to be added before passing the data to the RF-Process. Therefore all the data traffic from the host interface is routed to the LLCP firmware module.

To manage the timer for the LLCP activation and symmetry procedure also all incoming data from the RF interface is processed by the LLCP firmware module. If the incoming frame is an LLCP data frame the frame is forwarded to the host interface. If the frame is an LLCP symmetry frame the used buffer is released and the LLCP symmetry timer is started. Also LLCP control frames, like a disconnect frame, are processed by the module.

### 4.3.2 Linux Device Driver

To provide a straight forward possibility to include the PN547 NFC-Controller including the LLCP stack into a Linux operating system a Network Device Driver is implemented. This driver module is called PN547Net.

#### Mapping of the SPI Devices

The PN547 evaluation boards are connected to the Raspberry Pi boards via SPI. Raspian, the Linux distribution used on the Raspberry Pi boards, comes with a driver for the SPI interface. As the Raspberry Pi offers two chip select signals two SPI Interfaces are mapped to the device files /dev/spidev0.0 and /dev/spidev0.1. This is done via the Code shown in Listing 4.1 in the Linux Kernel Source File bcm2708.c located in /arm/mach-bcm2708 within the Source Code of the Kernel used by Raspian.

```
1  static struct spi_board_info bcm2708_spi_devices[] = {
2          {
3                  .modalias = "spidev",
4                  .max_speed_hz = 500000,
5                  .bus_num = 0,
6                  .chip_select = 0,
7                  .mode = SPI_MODE_0,
8          }, {
9                  .modalias = "spidev",
10                 .max_speed_hz = 500000,
11                 .bus_num = 0,
12                 .chip_select = 1,
13                 .mode = SPI_MODE_0,
14         }
15 };
```

Listing 4.1: Original SPI Device Mapping in bcm2708.c

As the SPI interface is used to communicate with the PN547 NFC-Controller from the PN547 Network Device Driver the mapping of the two SPI devices is changed to only assign a device file to one of them and map the other directly to the PN547 Network Device Driver Module. This is shown in Listing 4.2.

```
static struct spi_board_info bcm2708_spi_devices[] = {
  {
    .modalias = "pn547Net",
    .max_speed_hz = 500000,
    .bus_num = 0,
    .chip_select = 0,
    .mode = SPI_MODE_0,
  }, {
    .modalias = "spidev",
    .max_speed_hz = 500000,
    .bus_num = 0,
    .chip_select = 1,
    .mode = SPI_MODE_0,
  }
};
```

Listing 4.2: Adapted SPI Device Mapping in bcm2708.c

### Interrupt Assignment

The PN547 informs the device host about data to be fetched from the host interface by a interrupt signal. This line is connected to a GPIO of the Raspberry Pi boards. To avoid checking this input by continuous polling the pin is assigned as an interrupt. This is also done in the bcm2708.c Kernel source code File and shown in Listing 4.3.

```
static void __init pn547Net_init(void){
  bcm2708_spi_devices[0].irq = gpio_to_irq(PN547NET_INT_GPIO_PIN);
  irq_set_irq_type(bcm2708_spi_devices[0].irq, IRQ_TYPE_EDGE_RISIN);
  printk(KERN_DEBUG "BCM 2708 pn547Net_init: got IRQ %d for PN547Net\n",
      bcm2708_spi_devices[0].irq);
}
```

Listing 4.3: Adapted SPI Device Mapping in bcm2708.c

### Network Device Driver Kernel Module

The PN547 Network Device Driver can be abstracted as a black box interfacing the PN547 via the SPI interface on one side and the Linux Network Stack on the other side. The SPI interface is provided by another device driver provided by Raspian. The interface to the Linux Network stack is reflected by the Linux Network Driver API.

### SPI Interface

SPI offers full duplex communication but the PN547 uses a half-duplex implementation, and therefore additional framing is used to reduce SPI to half-duplex. This is done by inserting a transfer direction byte at the beginning of each SPI transmission. To read data from the PN547 the transfer direction byte is set to 0xFF. With the next SPI clock cycle the PN547 starts to provide the data on the MISO line. The data provided by the host controller on the MOSI line is ignored. This is illustrated in Figure 4.9.

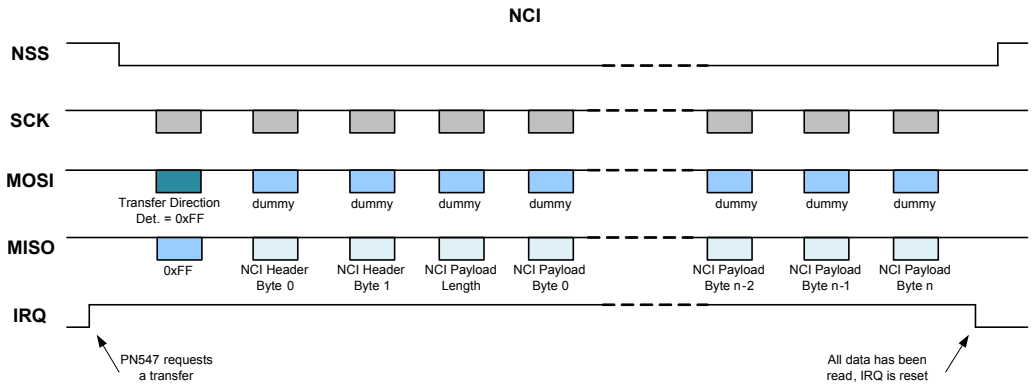


Figure 4.9: SPI Framing for reading from the PN547

If the first bit of the transfer direction byte is zero the PN547 reads the data from the MOSI line with the next SPI clock cycles. The PN547 puts 0xFF on the MISO line during the complete read cycle. This is shown in Figure 4.10.

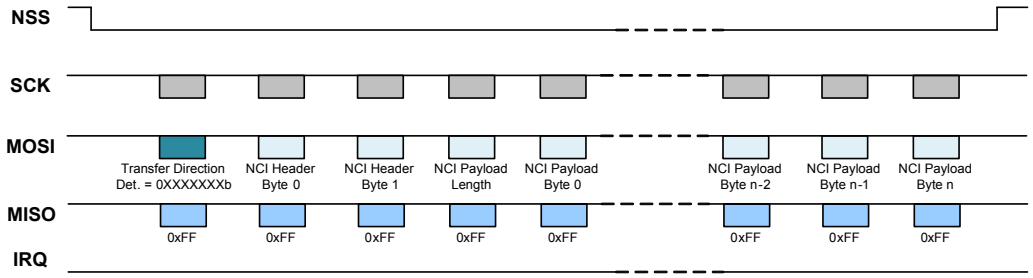


Figure 4.10: SPI Framing for writing to the PN547

The SPI driver provides a function called spi_sync. By calling this function a given number of bytes is synchronized. This means the bytes to transfer are pushed to the MOSI Line and the bytes on the MISO line are written into a receive buffer.

### Interface to the Network Stack

Each interface is described by a struct net_device item, which is defined in netdevice.h. This struct is allocated within the Kernelmodule and is shared with the Kernel. The structure contains pointers to the various functions implementing the Network Driver API as well as information about the driver and a pointer to a private structure used by the Kernel Module. This private Structure is called pn547NetData and is shown in Listing 4.4.

```
/* Driver local data */
struct pn547NetData {
  struct net_device *netdev;
  struct spi_device *spi;
  struct mutex lock;
  struct sk_buff *tx_skb;
  struct work_struct tx_work;
  struct work_struct irq_work;
  struct work_struct restart_work;
  u32 msg_enable;
  u8 spi_tx_buf[SPI_TRANSFER_BUF_LEN];
  u8 spi_rx_header_buf[NCI_HEADER_LEN + 1];
  u8 spi_rx_data_buf[MAX_FRAMELEN];
  struct semaphore nci_sem;
  u8 nci_rx_buf[MAX_FRAMELEN + 3];
  int nci_rx_buf_len;
  bool linkActive;
};
```

Listing 4.4: PN547Net private Data Structure

The private Data Structure contains pointers to the net_device struct and to the spi_device struct provided by the SPI driver. As the spi_sync function is a blocking function all SPI communication is done in own threads to avoid blocking the driver by waiting for the SPI device. The Mutex called lock is used for synchronization of these threads. The threads are handled by work queues. The structs tx_work, irq_work, and restart_work are handlers for them. The 32 bit integer msg_enable is a bitmap used for conditional logging of debug information. This can be controlled via the Linux command-line tool eth_tool while the driver is running.

The struct net_device_ops is used to map the calls of the Linux Network Stack to the corresponding functions within the driver kernel module. The struct and its members are shown in Listing 4.5.

```
static const struct net_device_ops pn547Net_netdev_ops = {
  .ndo_open  = pn547Net_net_open,
  .ndo_stop  = pn547Net_net_close,
  .ndo_start_xmit  = pn547Net_net_send_packet,
  .ndo_set_rx_mode = 0,
  .ndo_set_mac_address = 0,
  .ndo_tx_timeout  = pn547Net_tx_timeout,
  .ndo_change_mtu  = pn547Net_change_mtu,
  .ndo_validate_addr = eth_validate_addr,
};
```

Listing 4.5: PN547Net net_ops Structure

The struct contains function pointers to the functions containing the implementation in the Kernel module. Not implemented functions are indicated by setting the pointer to null. The following description contains further information about the functions.

ndo_open: This function is called when the network interface is started by the Network Stack. Within this function the driver starts discovery on the PN547 by sending the discover NCI command. Afterwards the Link status is set to inactive and the send queue of the network interface is started. As the link status is inactive no packets can be sent via the network interface at this time.

ndo_stop: This function stops the interface send queue.

ndo_start_xmit: If the Network Stack has data packets to transmit this function is called. The data to send is passed to the function in a buffer allocated by the network stack. The device driver first stops the send queue of the interface to avoid accepting new data packets before the current one is sent. Afterwards the data packet is sent to the PN547 via SPI in a new thread.

ndo_set_rx_mode: This function is not required because for the peer to peer connection between the two NFC-Controllers all data packets are received. Therefore the function is not implemented in the driver.

ndo_set_mac_address: Also this function is not required because the IP packets are not filtered within the NFC-Controller and therefore random MAC addresses are used.

ndo_tx_timeout: If a data packet sent by the network stack is not processed within a predefined timeout, then the current implementation only logs the event and does not take care of resolving the problem.

ndo_change_mtu: To avoid accepting data packets that are to large, the value for the MTU set by the user is checked within this function. If the desired MTU is not larger than the maximum length of an NCI message accepted by the PN547, the set MTU function of the network stack (eth_change_mtu) is called.

ndo_validate_addr: This function is directly mapped to the corresponding function (eth_validate_addr) of the network stack.

### Initialization

When the driver Kernel Module is loaded the probe function (pn547Net_probe) is called by the Kernel. Within this function the memory for the net_device, including the drivers private data structure, is allocated and initialized. Also the function pointers in the net_ops Structure are set. To process incoming data, indicated by NFC-Controller via the IRQ line, the interrupt handler is registered with the code shown in Listing 4.6.

```
1  /* Board setup must set the relevant edge trigger type;
2   * level triggers won't currently work.
3   */
4  ret = request_irq(spi->irq, pn547Net_irq, IRQF_TRIGGER_RISING, DRV_NAME, priv);
5  if (ret < 0) {
6    if (netif_msg_probe(priv))
7      dev_err(&spi->dev, DRV_NAME ": request irq %d failed "
8        "(ret = %d)\n", spi->irq, ret);
9    goto error_irq;
10 }
```

Listing 4.6: PN547Net Interrupt Registration

Line 4 of the Listing shows the actual function called to register the interrupt handler. The assignment of the GPIO to be used as Interrupt was done before in the File bcm2708.c and therefore this information is available in the SPI driver structure. The next argument (pn547Net_irq) is the function to be called for each IRQ. The constant IRQF_TRIGGER_RISING is passed to the function to trigger the IRQ on every rising edge of the NFC-Controllers IRQ line. Finally the driver's name and private data structure are passed to the function.

After this registration the interrupt handler function is called on each IRQ. Fetching messages from the NFC-Controller is done via the blocking SPI sync function, and therefore this is done in an own thread.

### Interrupt Handler

The first action taken by the interrupt handler is to fetch the NCI message from the NFC-Controller. As the length of the message and therefore the amount of bytes to be fetched is not known this has to be done in two steps. First the three byte NCI header is read. The third byte of the header indicates the length of the NCI messages payload, with this information the rest of the message is read. The header also contains information about the NCI message type. The driver handles three types of messages:

Notifications: Four types of notifications are handled. If a Reset or an RF Interface Deactivated Notification is received from the NFC-Controller the links status is set to inactive. In case of an Error Notification, this can be either a generic or an interface error notification, the NFC-Controller is reconfigured and the net_open function is called to bring the interface up again. After the NFC-Link between the two Controllers is established an Interface Activated Notification is received. In this case the link status is set to active and the send queue of the Network Interface is started. The last

type of notification handled is the Credit Notification. If a data packet is passed to the NFC-Controller and successfully sent, this notification is used to inform the driver that the NFC-Controller is able to receive a new frame. After receiving this notification the driver releases the buffer used to pass the data from the Network Stack to the driver and wakes the send queue of the network interface.

Response Messages: As NCI commands are acknowledged by the NFC-Controller with response messages the driver has to wait for the corresponding response after each command message sent. Therefore a Semaphore is used to avoid busy waiting. Once a response is received the next thread sleeping on the Semaphore wakes up and processes the response.

Data Messages: If a data packet is received by the NFC-Controller it is forwarded to the host via an NCI data message. After removing the NCI-Header the data is passed to the network stack by allocating a buffer, copying the data and calling the function netif_rx_ni with the buffer as argument.

### Establishing a Network Connection

To establish a network connection the PN547Net driver module has to be loaded. After this the Network Interface has to be started with the desired IP-Address and MTU. This is done by the command sequence shown in Listing 4.7 from a Linux Terminal as root user. In line one the driver module is loaded, line two sets the MTU of the Interface created by the driver to 250 bytes and finally line three starts the interface with the IP-Address set to 192.168.200.1.

```
1 insmod pn547Net.ko
2 ifconfig eth1 mtu 250
3 ifconfig eth1 up 192.168.200.1
```

Listing 4.7: Command Sequence to set up the Network Link

# Chapter 5

# Experimental Evaluation

## 5.1 Tests and Results

The tests used to evaluate the implemented solutions are described on the following pages. The tests can be divided into two groups, Latency tests and Throughput tests. The Latency tests are used to analyze the link delay and the Throughput tests to show the link speed.

As described in the implementation chapter the NFC controllers are integrated into the Linux System on the Hosts via a network driver. Therefore tools that come with the Raspian Linux distribution can be used to run the tests. For the tests the MTU for the Linux Network stack is set to 200. The Throughput tests measuring Throughput with various MTUs are an exception as different sizes for MTU are used there.

### 5.1.1 Latency Tests

To test the link delay the Linux Ping tool is used. To collect a sufficient amount of data to eliminate testing outliers 200 Pings are made for each Test run. This is done by running the command shown in Listing 5.1.

```
1 ping -c 200 192.168.200.1
```

Listing 5.1: Ping command used for Latency Tests

The option -c 200 tells the Ping command to send 200 Pings in a row. As no further options are passed to the command the default packet size of 56 Bytes, which translates into 64 ICMP Bytes with 8 Bytes ICMP header, and a timeout of one second between the pings is used.

The first test is used to analyze the influence of the Antenna distance on the Link delay. This is done using a connection with 424 kBit/s RF-Speed and LLCP timeouts of 10, 30 and 50 milliseconds on the target as well as on the initiator side. The results of these Tests are shown in Table 5.1 and Figure 5.1.

| 424 kBit/s RF Speed, 10 ms LTO | | | | | | |
|---|---|---|---|---|---|---|
| Antenna distance | 5 mm | | 25 mm | | 45 mm | |
| min | 14.20 | 14.30 | 14.30 | 14.20 | 14.30 | 14.30 |
| lower quartile | 14.40 | 14.40 | 14.40 | 14.40 | 14.40 | 14.40 |
| median | 14.40 | 14.50 | 14.60 | 14.40 | 14.50 | 16.70 |
| upper quartile | 25.90 | 24.50 | 25.35 | 24.55 | 24.25 | 25.60 |
| max | 38.20 | 33.10 | 40.50 | 35.20 | 52.10 | 47.50 |

Table 5.1: Latency of the Network Link at 424 kBit/s RF Speed and varying Antenna Distance

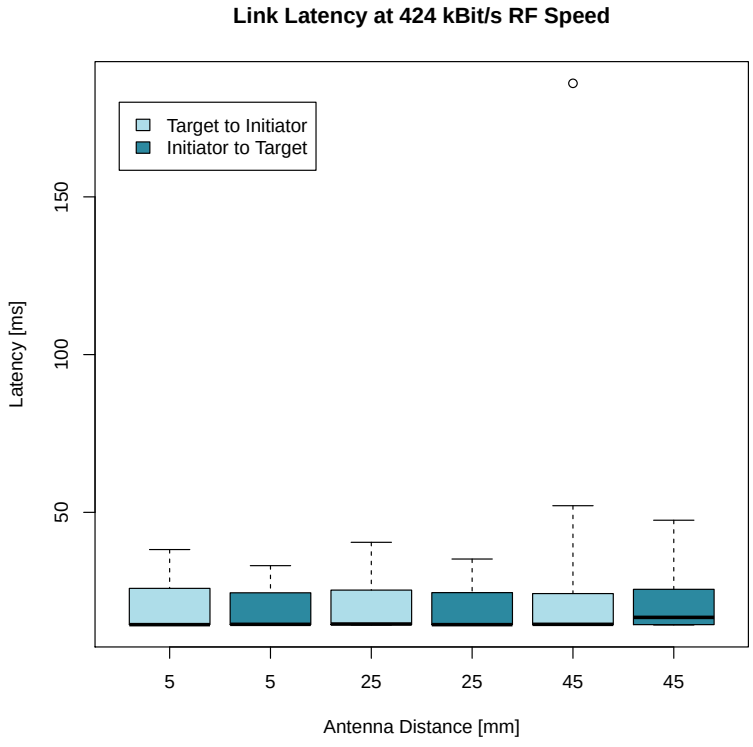


Figure 5.1: Latency of the Network Link at 424 kBit/s RF Speed and varying Antenna Distance using 10 ms as LLCP Timeout

The small circle in the Boxplot Figure 5.1 marks an outlier in the measured latency data. The value of this outlier is not taken into account for the calculation of the statistics. The tests are showing that there is almost no difference in the results because of the antenna distance, as long as the maximum communication distance before the connection aborts is not reached, all further tests are done with an antenna distance of 25 millimeters.

To show the influence of the RF-Speed and the LLCP timeout on the link delay further tests with 106 kBit/s, 424 kBit/s and timeouts of 10, 20 and 50 milliseconds are done. The results for 106 kBit/s are shown in Table 5.2. To allow an easy comparison of the results the results are also displayed as Boxplots in Figure 5.2. For the RF Speed of 424 kBit/s the results are displayed in the Boxplot Figure 5.3 and the corresponding Table 5.3.

| 106 kBit/s RF Speed | | | | | | |
|---|---|---|---|---|---|---|
| LLCP Link Timeout | 10 ms | | 30 ms | | 50 ms | |
| min | 28.2 | 28.2 | 28.20 | 28.2 | 28.2 | 28.2 |
| lower quartile | 28.3 | 28.3 | 28.30 | 28.3 | 28.3 | 28.3 |
| median | 28.3 | 29.7 | 30.05 | 28.8 | 28.3 | 28.3 |
| upper quartile | 31.9 | 39.0 | 39.30 | 37.3 | 41.9 | 37.8 |
| max | 49.0 | 49.7 | 66.10 | 58.5 | 98.1 | 90.8 |

Table 5.2: Latency of the Network Link at 106 kBit/s RF Speed in Milliseconds

| 424 kBit/s RF Speed | | | | | | |
|---|---|---|---|---|---|---|
| LLCP Link Timeout | 10 ms | | 30 ms | | 50 ms | |
| min | 14.30 | 14.20 | 14.30 | 14.30 | 14.30 | 14.30 |
| lower quartile | 14.40 | 14.40 | 14.40 | 14.40 | 14.40 | 14.40 |
| median | 14.60 | 14.40 | 14.40 | 14.40 | 18.55 | 17.35 |
| upper quartile | 25.35 | 24.55 | 21.70 | 26.45 | 58.85 | 59.45 |
| max | 40.50 | 35.20 | 55.00 | 58.00 | 79.90 | 70.10 |

Table 5.3: Latency of the Network Link at 424 kBit/s RF Speed in Milliseconds

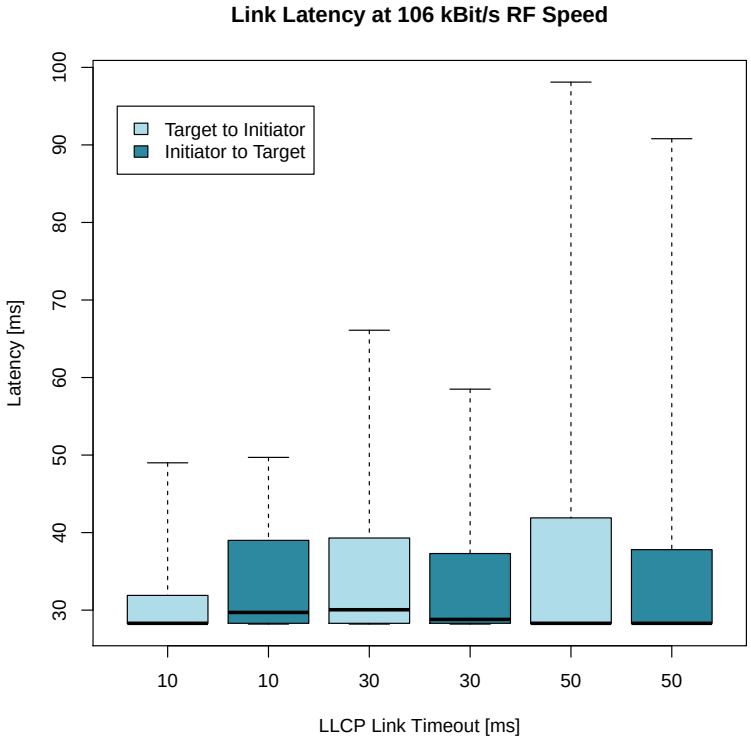


Figure 5.2: Latency of the Network Link at 106 kBit/s RF Speed

### 5.1.2 Throughput Tests

To measure the end to end payload throughput of the TCP/IP connection the Linux tool iperf is used. Iperf is based on a client/server principle therefore two commands have to be called. Listing 5.2 shows the command to start the server and Listing 5.3 the command used to connect the client and start the test.

```
1 iperf -s
```

Listing 5.2: Iperf server side command used for Throughput Tests

```
1 iperf -c 192.168.200.1
```

Listing 5.3: Iperf client side command used for Throughput Tests

Like the Ping command iperf is used with no additional options and therefore the default size of the data packet used for the tests is 256 Kilobytes. Like the Latency tests also the Throughput tests are made using RF-Speeds of 106 kBit/s and 424 kBit/s with LLCP timeouts of 10, 20 and 50 milliseconds. Unlike the Latency tests the Throughput tests are already reporting an average result. Nevertheless to avoid measurement errors five iperf measurements are made for each evaluated RF Speed and LLCP Link Timeout combination. If a measurement result is judged to be an outlier, then it is repeated.

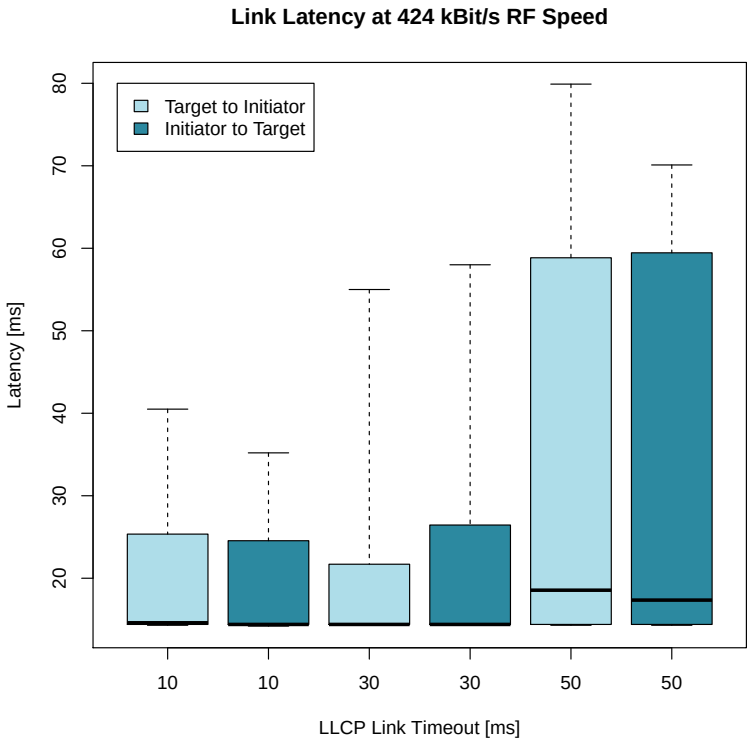


Figure 5.3: Latency of the Network Link at 424 kBit/s RF Speed

The Results are shown in Table 5.4 and 5.5. A graphical representation of the Results is displayed in Figures 5.4 and 5.5.

| 106 kBit/s RF Speed | | |
|---|---|---|
| LLCP Link Timeout | Target to Initiator | Initiator to Target |
| 10 ms | 32.12 kBit/s | 31.58 kBit/s |
| 30 ms | 25.84 kBit/s | 29.76 kBit/s |
| 50 ms | 21.30 kBit/s | 23.50 kBit/s |

Table 5.4: Throughput of the Network Link at 106 kBit/s RF Speed

The PN547 NFC Controller accepts up to 255 Bytes in one SPI message. This has to be taken in account when choosing the MTU for the network stack. In addition to the payload several protocol headers, e.g. the TCP and IP Headers, the NCI Header or the Direction indicator for SPI, have to be transferred with each packet, and therefore the MTU has to be chosen as high as possible, without exceeding the maximum SPI message length accepted by the NFC Controller. To analyze the influence of the MTU on the end to end payload throughput iperf measurements with various values for the MTU are used. The tests are made at an RF Speed of 424 kBit/s and an LLCP Link Timeout of 10ms. Iperf is used with the same settings as for the previous Throughput tests. Table

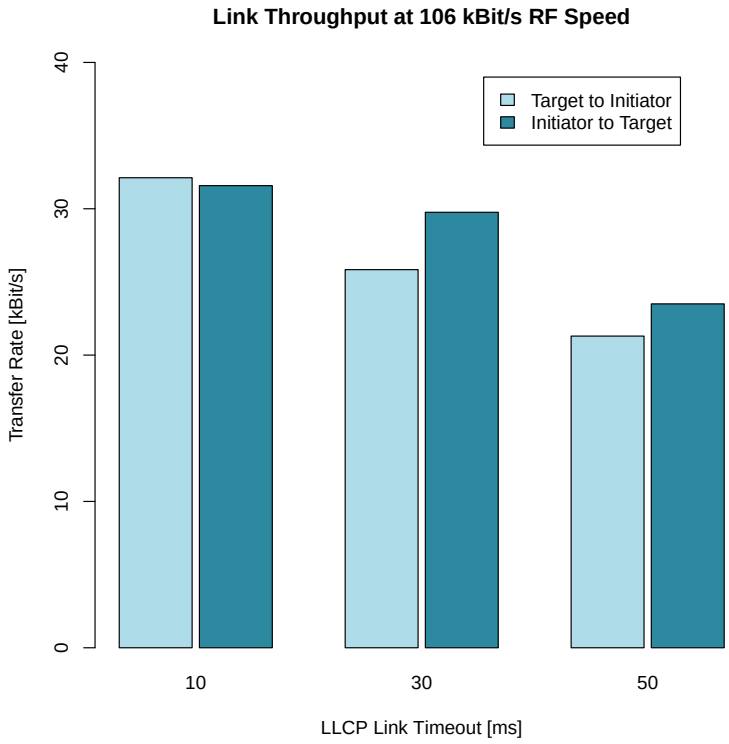


Figure 5.4: Throughput of the Network Link at 106 kBit/s RF Speed

5.6 contains the average over five measurements per MTU value and Figure 5.6 displays the results as a Bar plot.

| 424 kBit/s RF Speed | | |
|---|---|---|
| LLCP Link Timeout | Target to Initiator | Initiator to Target |
| 10 ms | 63.06 kBit/s | 61.36 kBit/s |
| 30 ms | 46.44 kBit/s | 53.32 kBit/s |
| 50 ms | 39.48 kBit/s | 50.44 kBit/s |

Table 5.5: Throughput of the Network Link at 424 kBit/s RF Speed

| 424 kBit/s RF Speed | | |
|---|---|---|
| MTU | Target to Initiator | Initiator to Target |
| 230 | 71.72 kBit/s | 73,80 kBit/s |
| 200 | 63.06 kBit/s | 61.36 kBit/s |
| 170 | 44.30 kBit/s | 45.52 kBit/s |
| 140 | 33.98 kBit/s | 33.98 kBit/s |

Table 5.6: Throughput of the Network Link at 424 kBit/s RF Speed and varying MTU

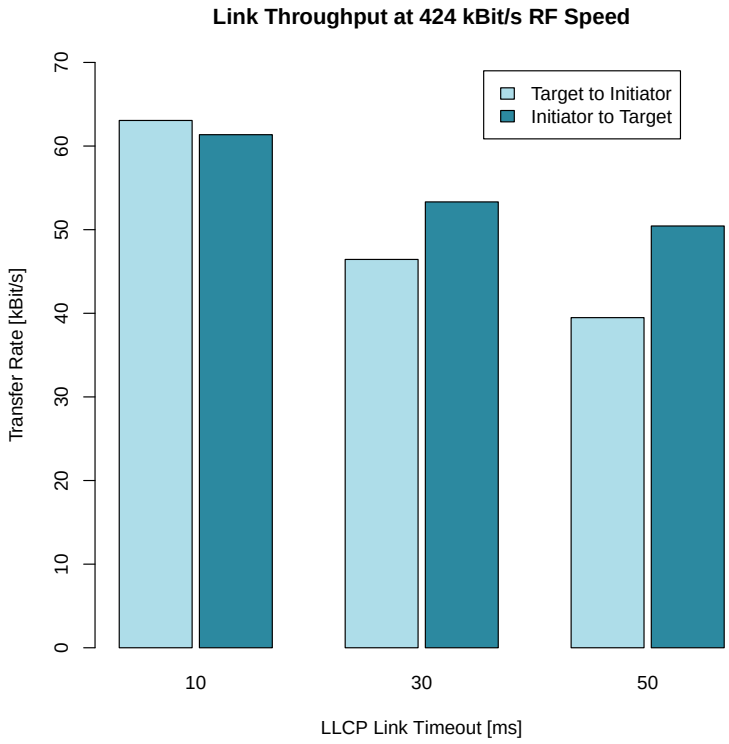


Figure 5.5: Throughput of the Network Link at 424 kBit/s RF Speed

## 5.2 Interpretation of the Results

Like the description of the Tests and their results also the evaluation of the results is done in two parts. One part for the results of the latency tests and one for the results of the throughput tests.

### 5.2.1 Latency Tests

The first Latency test shows the link latency at 424 kBit/s RF Speed for different antenna distances. The distances used are five millimeters, 25 millimeters and at last 45 millimeters. 45 millimeters represents the distance where the connection begins to be unstable and reactivations of the RF link occurs. The LLCP link timeout is chosen as 10 milliseconds for both the target and the initiator NFC controller. As displayed in Figure 5.1 the antenna distance has almost no impact on the link latency. It was expected that the difference in the RF propagation time has no impact. At the highest used antenna distance the RF link was lost from time to time. The PN547 network device driver reports this to the upper layers as network cable unplugged therefore no pings are sent until the link is reestablished. Therefore the outlier, marked in the boxplot by a circle, can be explained by a corruption of the data packet while transmitting over the RF link. If for example the target NFC controller receives a corrupt RF package, no response is sent. After a timeout the initiator recognizes the lack of an answer and initiates the retransmission of

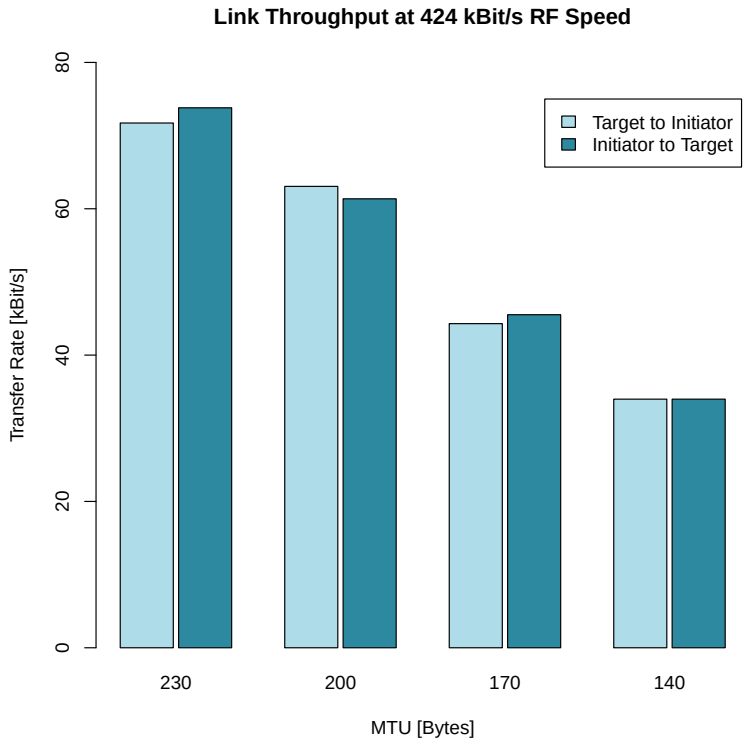


Figure 5.6: Throughput of the Network Link at 424 kBit/s RF Speed and varying MTU

the package. The time needed for this procedure can explain the outliers difference to the average value of almost 170 milliseconds.

The Latency tests at 106 kBit/s RF speed show median values of 28.3 to 30.05 milliseconds for the three measured LLCP link timeouts. The minimum value for all three link timeouts is 28.20 milliseconds. As expected the value used for the link timeout has no influence on the minimum value, as these values represent the best case where no timeout has to be waited for. The small differences in these values are expected, because of the low possibility of the worst case scenario where a complete timeout has to be waited before the ping can be sent. The influence of the link timeout can be seen in the maximum values. For example the difference between the maximum and the minimum values at 10 milliseconds link timeout is about 20 milliseconds. The software timer used for the LLCP timeout in the NFC controller firmware is based on a system tick occurring every 10 milliseconds. The timer is designed to ensure that at least the given time is waited. This means for 10 milliseconds waiting time two system ticks have to occur before the timer triggers. Therefore the time until the timer triggers is at least, and not more than, the given waiting time plus 10 milliseconds. This explains the 20 milliseconds difference at 10 milliseconds link timeout. Also the differences for 30 and 50 milliseconds are plausible.

At the RF speed of 424 kBit/s lower minimum and median values can be seen. This is explained by the lower time needed to transfer the ping request and ping reply data packages at the higher RF transfer rate. As for 106 kBit/s the maximum values show the

influence of the link timeout. The maximum values are about the link timeout plus 10 milliseconds higher than the minimum values.

The difference between the minimum values for 106 kBit/s and 424 kBit/s are corresponding to the difference of the time needed to transfer the ping request and reply data packets over the RF link.

The data packages to be transferred for 106 kBit/s consist of the ping packet itself (64 Bytes), the IP header (20 Bytes), the LLCP header (2 Bytes) plus the NFC-DEP header and preamble for 106 kBit/s (3 Bytes) as well as the CRC-Checksum (2 Bytes). In addition at 106 kBit/s a parity bit is added to each byte.

The complete length in Bit is calculated in the following equations.

$$\begin{aligned} len_{req106} &= (len_{ping} + len_{IP} + len_{LLCP} + len_{NFC106}) * 9 \\ &= (64 + 20 + 2 + 5) * 9 = 819Bit \end{aligned} \tag{5.1}$$

With a transfer rate of 106 kBit/s the time to send the package is:

$$time_{req106} = \frac{len_{req106}}{106 * 10^3} = \frac{819}{106 * 10^3} = 7.73ms \tag{5.2}$$

The ping reply has the same length as the ping request, therefore complete RF transmission time for a ping request and the reply can be calculated as:

$$time_{106} = time_{req106} * 2 = 7.73 * 2 = 15.46ms \tag{5.3}$$

The SPI interface used for the transmission of the data packet from the host controller to the NFC controller is operated with a clock frequency of 500 kHz. The data packet transferred over SPI consists of the 64 Byte ping packet, 20 Bytes IP header, 3 Bytes NCI header and an SPI direction Byte:

$$len_{spidata} = len_{ping} + len_{IP} + len_{NCI} + len_{spidirection} = 64 + 20 + 3 + 1 = 88Byte \tag{5.4}$$

With 500 kHz SPI clock frequency the time required for the SPI transmission of the ping request and response is calculated as:

$$time_{spi} = 4 * \frac{len_{spidata}}{500 * 10^3} = 4 * \frac{88}{500 * 10^3} = 0.7ms \tag{5.5}$$

With these values the processing time in the NFC controller and the host controller can be calculated.

$$\begin{aligned} time_{hostNFCC106} &= time_{minPing106} - time_{106} - time_{spi} \\ &= 28.2 - 15.46 - 0.7 = 12.04ms \end{aligned} \tag{5.6}$$

At 424 kBit/s the datapacket to be transferred over the RF link consists of of the ping packet itself (64 Bytes), the IP header (20 Bytes), the LLCP header (2 Bytes) plus the NFC-DEP header and preamble for 424 kBit/s(8 Bytes) as well as the CRC-Checksum (2 Bytes).

$$\begin{aligned} len_{req424} &= len_{ping} + len_{IP} + len_{LLCP} + len_{NFC424} * 8 \\ &= (64 + 20 + 2 + 10) * 8 = 768Bit \end{aligned} \tag{5.7}$$

At a transfer rate of 424 kBit/s the time needed for transferring the ping request and reply over the RF link is:

$$time_{424} = 2 * \frac{len_{req424}}{424 * 10^3} = 2 * \frac{768}{424 * 10^3} = 3.62ms \tag{5.8}$$

As the same data has to be transferred via SPI as for 106kBit/s the host and NFCC processing time is calculated as:

$$\begin{aligned} time_{hostNFCC424} &= time_{minPing424} - time_{424} - time_{spi} \\ &= 14.2 - 3.62 - 0.7 = 9.88ms \end{aligned} \tag{5.9}$$

The results of these calculations show that there is a difference in the time needed for processing within the NFC controller and the host controller of 2.16 milliseconds. As this result is not expected it has to be investigated if this gap occurs within the NFC controller or the host controller and how it can be eliminated. To send a ping request and receive the response, the ping data packets are passing the host controllers and NFC controllers, on both the target and the initiator side, two times. The processing time of a data packet in the NFC controller can be assumed to be at maximum one millisecond. For a ping request and the response two NFC controllers have to be passed two times, so four milliseconds are estimated for processing in the NFC controllers. With this result the remaining time for processing on the host controllers is at least 5.88 milliseconds. The processing on the host controller also has to be done four times for one complete ping request/response, so 1.47 milliseconds are estimated for the processing of one data packet. This time is consumed by the network stack, the PN547Net driver and the driver of the used SPI interface. This delay may cause serious impact on the maximum achievable throughput, an should therefore be investigated in future.

### 5.2.2 Throughput Tests

The first two throughput tests were made to show the influence of the LLCP link timeout. The results for 106 kBit/s as well as the results for 424 kBit/s show a clear trend of decreasing throughput when the link timeout is increased. The difference in the maximum values is about 30 kBit/s. This means the throughput at 424 kBit/s is about twice as high as for 106 kBit/s. The gaps between the two directions, target to initiator and initiator to target, at link timeouts of 30 and 50 milliseconds are not expected and should be investigated later.

The third throughput test is done to show the influence of the MTU. The PN547 NFC controller only accepts payload packages up to 255 Bytes via NCI. Therefore the maximum data size can be calculated as the difference of the maximum accepted packet size and the various protocol headers. The complete headers consist of 20 Bytes IP header, 40 Bytes TCP header, 3 Bytes NCI header and the SPI direction Byte:

$$len_{headers} = len_{TCP} + len_{IP} + len_{NCI} + len_{spidirection} = 20 + 20 + 3 + 1 = 44Byte \tag{5.10}$$

As the header has to be transmitted for every data packet the used MTU has serious impact on the throughput. The values for the MTU represents the length of the payload and the 20 Bytes TCP header. Hence a maximum of 235 Bytes can be used. The results for the MTU tests are as expected.

In general the interpretation of the throughput tests is difficult because of the complex combination of parameters influencing the TCP/IP communication. Even for a one way file transfer the data packets have to be acknowledged by the receiver. The amount and point in time of acknowledge frames depend on multiple parameters of the TCP stack, such as the TCP congestion control or sliding window. These acknowledge frames as well as the data frames cannot always be sent immediately. The time to be waited before sending can vary between no delay and the LLCP link delay. A detailed analysis of these behavior would require some time and is beyond the scope of this thesis. Therefore a detailed analysis and optimization of the throughput should be made in a further thesis or paper.

# Chapter 6

# Conclusion and Further Work

## 6.1 Conclusion

The research presented in the chapter Related Work shows that NFC is often used for one way data transfers or to ease the connection setup of wireless LAN or Bluetooth connections. NFC enabled smartphones with Google's Android operating system also include a software LLCP stack and can use NFC for transferring small amounts of data between two phones. To promote NFC as an interface technology for non-mobile devices a cheap and easy to use solution is required. The evaluation of the design space showed that the best solution is an NFC controller supporting the LLCP High interface, as defined in this thesis. The NFC controller should provide a reduced feature set and only support peer to peer communication. With this reduced feature set the chip size as well as the power consumption of the NFC controller could be reduced. Since many consumer devices, such as printers or routers, use Linux based operating systems and often also have a TCP/IP stack implemented, the solution developed during this thesis can be used for integrating NFC as an interface with minimal further effort.

At the start of this thesis the NCI interfaces LLCP Low and LLCP High had not been standardized by the NFC Forum. The NFC forum is currently preparing version 1.1 of NCI. This version includes the LLCP Low interface as described in this thesis. For the LLCP High interface there is not jet either a draft or a proposal. Therefore the solution described in this thesis could be used for this interface.

The adaptation of the NFC controllers firmware took quite some time as the time for understanding the existing firmware and the removal of unused features to gain some Flash space for the implementation of the new features consumed several weeks. The software design was considered in detail before coding began, and therefore the implementation of the LLCP High and Low interfaces was straightforward, and the first simulations quickly showed results. After the completion of the Linux device drivers the first trials with TCP/IP data revealed some bugs regarding the data flow and required some debugging sessions. After solving these bugs the TCP/IP communication was stable. Besides the tests described earlier this work also allowed the investigation of some other use cases such a remote Desktop session via VNC. The latency and transfer rates of the TCP/IP connection showed that the current solution could be used for user interfaces as long as the transfer of large data sizes, e.g. high resolution graphics or animations are not required.

For the solution two types of user interfaces could be implemented. Either a lightweight user interface without the need for preloaded data on the user's device, or a graphically advanced user interface with high resolution graphics and animations with preloaded data. The graphics and animations could for example be part of a smartphone Application. The fast connection setup, which is below one second, is also more than sufficient for a fluent user interface experience.

Overall the results of the evaluation of the prototype showed that the concept is promising and further research on this topic should be done. Some possible further research topics are described in the final chapter, Further Work.

## 6.2 Further Work

Some topics for possible further research have been mentioned earlier in this thesis. In this chapter these topics are collected and described in more detail. The main topics, hardware adaptions to reduce the NFC controllers chip size and Improvement of the systems transfer rate are discussed below.

Chip size reduction: In the system design chapter it was stated that the best solution is an NFC controller with built in LLCP stack and adapted hardware. These Hardware adaptions could reduce the costs and power consumption of the controller. If the controller's feature set is reduced to cover just peer to peer communication, the code size and therefore the chip area required for ROM and Flash memory will decrease. Also hardware components, such as components for the SWP interface, that are not required for peer to peer communication, could be removed. Most NFC controllers are able to act as NFC target as well as NFC initiator. Therefore it should be considered if the reduction of the controller to only support target mode can lead to further improvements. The required resources for such a product would need to be balanced against the possible market for such a specialized device.

Transfer rate improvement: As described in the evaluation chapter the maximum transfer rate reached by the prototype system does not reach the maximum design transfer rate for NFCIP-1 connections. This is linked to the TCP data flow control mechanisms as well as to the link latency. Therefore a complete analysis of the TCP data flow and researching alternatives could be done in a further thesis. Topics for further research on the link latency are the processing times of data packets in the host and the NFC controller. Decreasing the minimum processing time of 1.5 milliseconds could improve the throughput. Also the different processing times for 424kBit/s and 106kBit/s communications, which is about 1.5 milliseconds, should be researched further. A first step would be to determine if the difference occurs in the host or the NFC controller and then research the causes and methods of reducing the data.

# Appendix A

# Shortcuts and Symbols

## A.1 Glossary

| | |
|---|---|
| ABM | Asynchronous Balanced Mode. 31, 48 |
| ACS | Advanced Card Systems. 21 |
| ASK | Amplitude Shift Keying. 15 |
| CA | Collision Avoidance. 15 |
| CIU | Contactless Interface Unit. 33, 36, 38 |
| CPU | Central Processing Unit. 26, 53, 54, 56, 58 |
| CRC | Cyclic Redundancy Check. 16, 78 |
| DEP | Data Exchange Protocol. 16, 17, 22, 39, 45, 46, 48, 49, 59, 62, 78 |
| DID | Device ID. 17 |
| DSAP | Destination Service Access Point Address Field. 41 |
| DSP | Digital Signal Processor. 53 |
| EEPROM | Electrically Erasable Programmable Read-Only Memory. 18, 35, 36, 38, 51, 52 |
| EMC | Electromagnetic Compatibility. 52 |
| FeliCa | Stands for Felicity Card, a contact less RFID smart card system from Sony. 13, 14 |
| FTP | File Transfer Protocol. 26 |
| GPIO | General Purpose Input/Output. 18, 54, 56, 68 |

GPU Graphics Processing Unit. 53, 54
GUI Graphical User Interface. 26

I/O Input/Output. 37, 58
$I^2C$ Inter-Integrated Circuit. 38, 51, 54
IP Internet Protocol. 27, 56, 67, 69, 73, 78–81
IP-Core Intellectual Property Core. 35
IPC Inter Process Communication. 58
IrDA Infrared Data Association. 18
IRQ Interrupt Request. 51–53, 55, 68

LED Light-emitting Diode. 10
Li-Ion Lithium-ion. 18
LLC Logical Link Control. 15
LLCP Logical Link Control Protocol. 14, 15, 17, 21–27, 31–33, 39, 41, 42, 45–49, 59, 61–63, 70, 72–74, 76–82

MAC Media Access Control. 14, 15, 39, 67
MIFARE MIFARE recovers technologies based on the ISO/IEC 14443 Type A 13.56 MHz standard by NXP Semiconductors. 13, 14
MIU Maximum Information Unit. 39, 41, 48
MIUX Maximum Information Unit Extension. 41
MTU Maximum Transmission Unit. 25–27, 67, 69, 70, 74, 75, 79

NAD Node Address. 17
NCI NFC Controller Interface. 33, 43–49, 59, 62, 67–69, 78, 79, 81
NDEF NFC Data Exchange Format. 21, 23
NFC Near Field Communication. 7, 10–15, 18–25, 27, 28, 30–39, 43–46, 48–53, 55, 56, 58, 59, 62, 63, 67–70, 74, 76–79, 81, 82
NFC-WI NFC Wired Interface. 34
NFCC NFC Controller. 30, 32, 44–49, 52, 56, 79
NFCEE NFC Execution Environment. 45
NFCIP-1 NFC Interface and Protocol 1. 7, 13–17, 24, 25, 27, 82
NPP NDEF Push Protocol. 21, 22
NXP NXP Semiconductors. 11–13, 18, 21, 23, 33, 38, 50, 52, 56, 59

| | |
|---|---|
| OBEX | OBject EXchange. 11 |
| PAX | Parameter Exchange. 39 |
| PC | Personal Computer. 23, 25–28, 56, 58 |
| PDU | Protocol Data Unit. 17, 22, 23, 39, 41, 42, 46 |
| PDUs | Protocol Data Units. 17 |
| PFB | Protocol Function Byte. 17 |
| PLL | Phase Locked Loop. 37 |
| PPP | Point-to-point protocol. 24, 26 |
| PTYPE | PDU Type. 41 |
| RAM | Random-access memory. 18, 25, 53, 62 |
| RF | Radio-Frequency. 14, 15, 20, 21, 33, 34, 36, 44, 45, 47, 49, 50, 56–59, 62, 63, 68, 70, 72–74, 76–79 |
| RFCOMM | Radio Frequency Communication. 24 |
| RFID | Radio-Frequency Identification. 13 |
| RFU | Reserved for further Usage. 45 |
| ROM | Read Only Memory. 35, 36, 38, 51, 82 |
| RTOS | Real Time Operating System. 56–58 |
| SD | Secure Digital. 54 |
| SDD | Single Device Selection. 15 |
| SDRAM | Synchronous Dynamic Random Access Memory. 53, 54 |
| SE | Secure Element. 18 |
| SIM | Subscriber Identity Module. 33 |
| Smart NFC Interface | Smart NFC Interface is a multi-purpose platform for developing and demonstrating physical browsing applications. 18–20 |
| SOC | System On Chip. 35, 53 |
| SPI | Serial Peripheral Interface Bus. 18, 28, 37, 38, 51, 54, 63–68, 74, 78, 79 |
| SRAM | Static Random-Access Memory. 51 |
| SSAP | Source Service Access Point Address Field. 41 |
| SSH | Secure Shell. 56 |
| SWP | Single Wire Protocol. 34, 57, 82 |
| TCP | Transmission Control Protocol. 27, 73, 79–82 |
| TLV | Type-Length-Value. 39 |

TULIP Transport Unaware Link Improvement Protocol. 28, 29
TWI Two-Wire-Interface. 38

UART Universal Asynchronous Receiver/Transmitter. 18, 19, 37, 54
UI User Interface. 20
UICC Universal Integrated Circuit Card. 34, 38
USB Universal Serial Bus. 52–54, 56

WLAN Wireless Local Area Network. 10, 11, 31

# Appendix B

# Software Design Documents

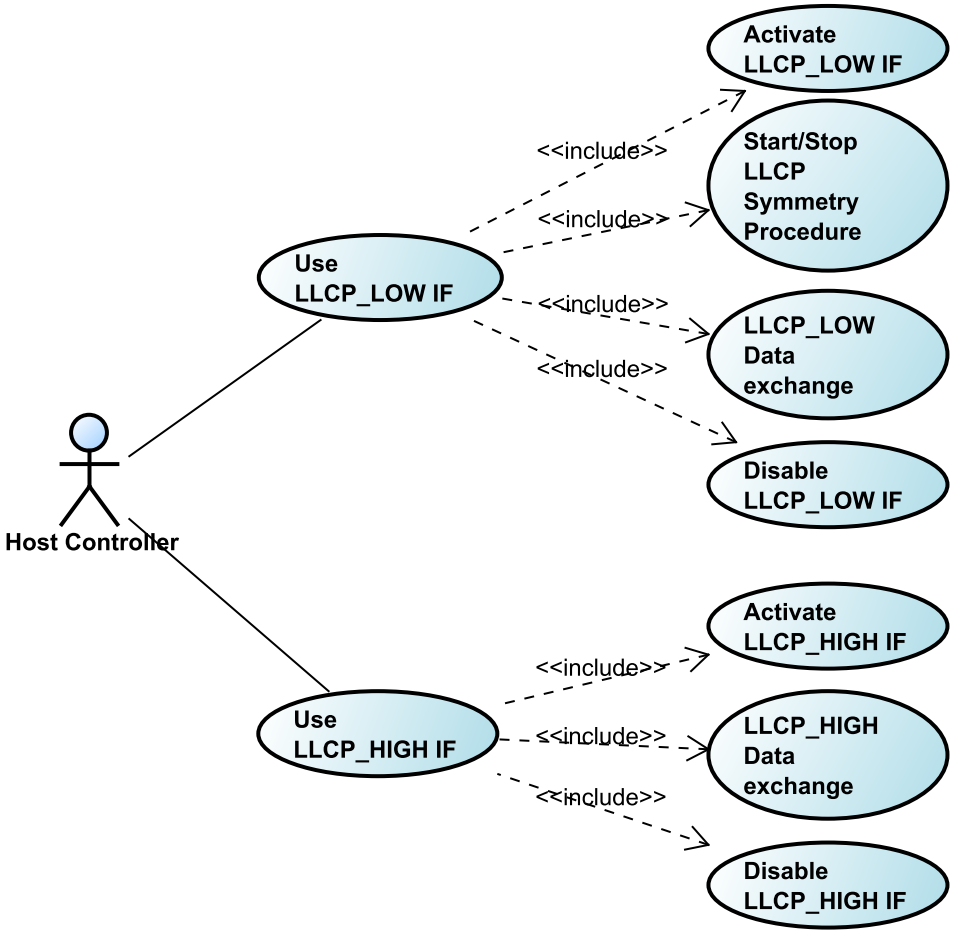


Figure B.1: PN547 Firmware LLCP Use Cases

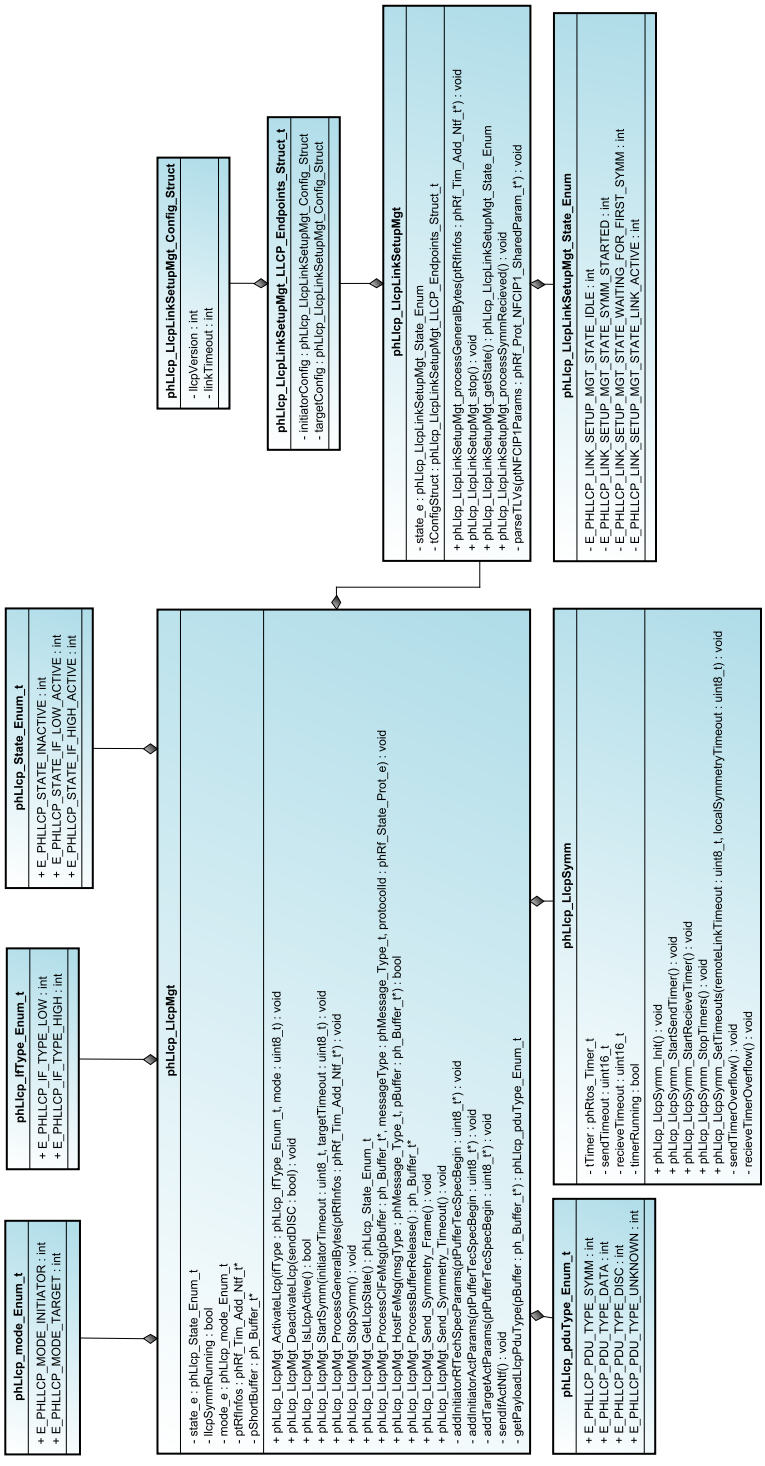


Figure B.2: PN547 Firmware LLCP Module Object based Decomposition

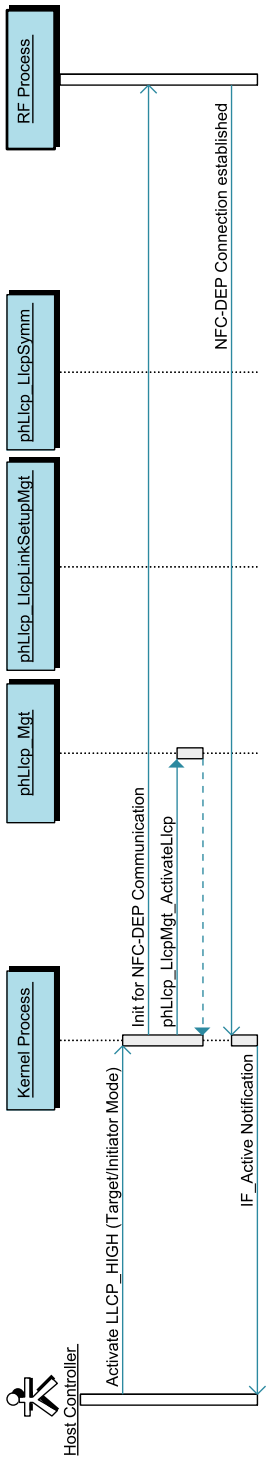


Figure B.3: PN547 Firmware LLCP Low Activation Sequence Diagram

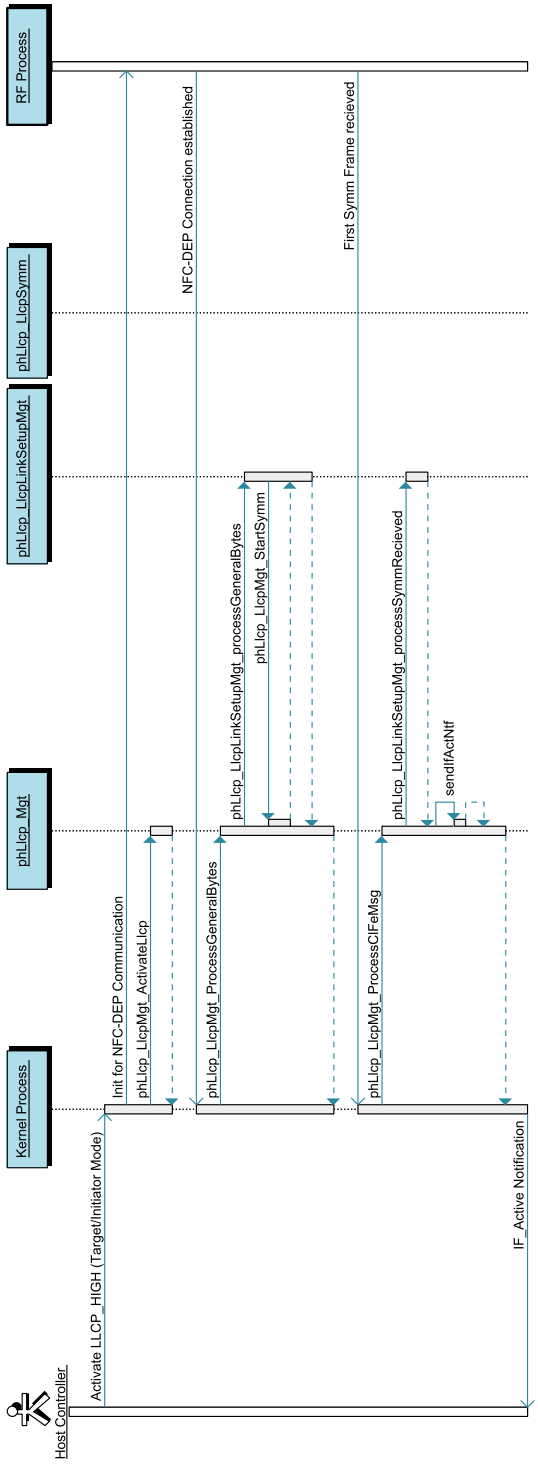


Figure B.4: PN547 Firmware LLCP High Activation Sequence Diagram

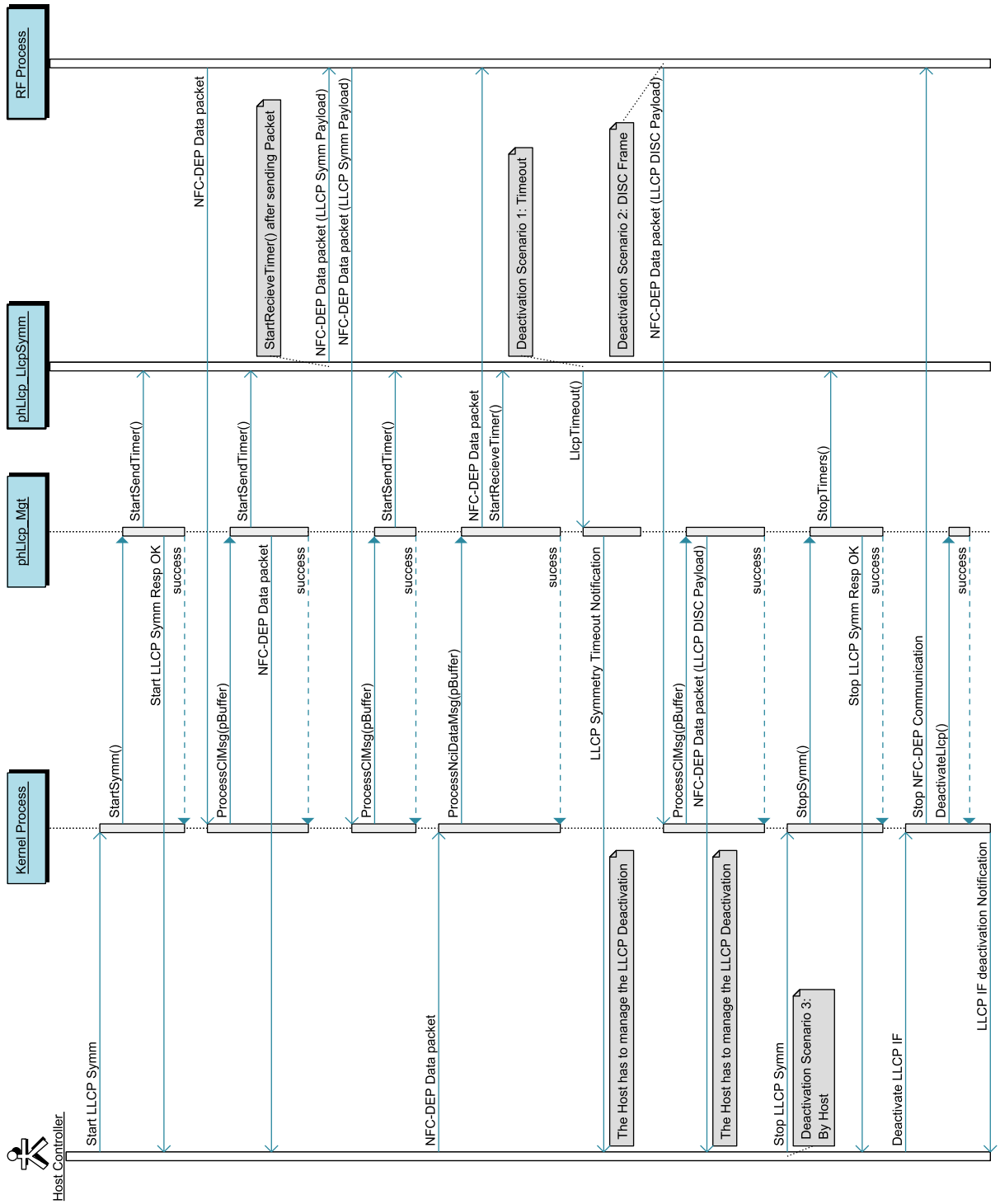


Figure B.5: PN547 Firmware LLCP Low Communication Sequence Diagram

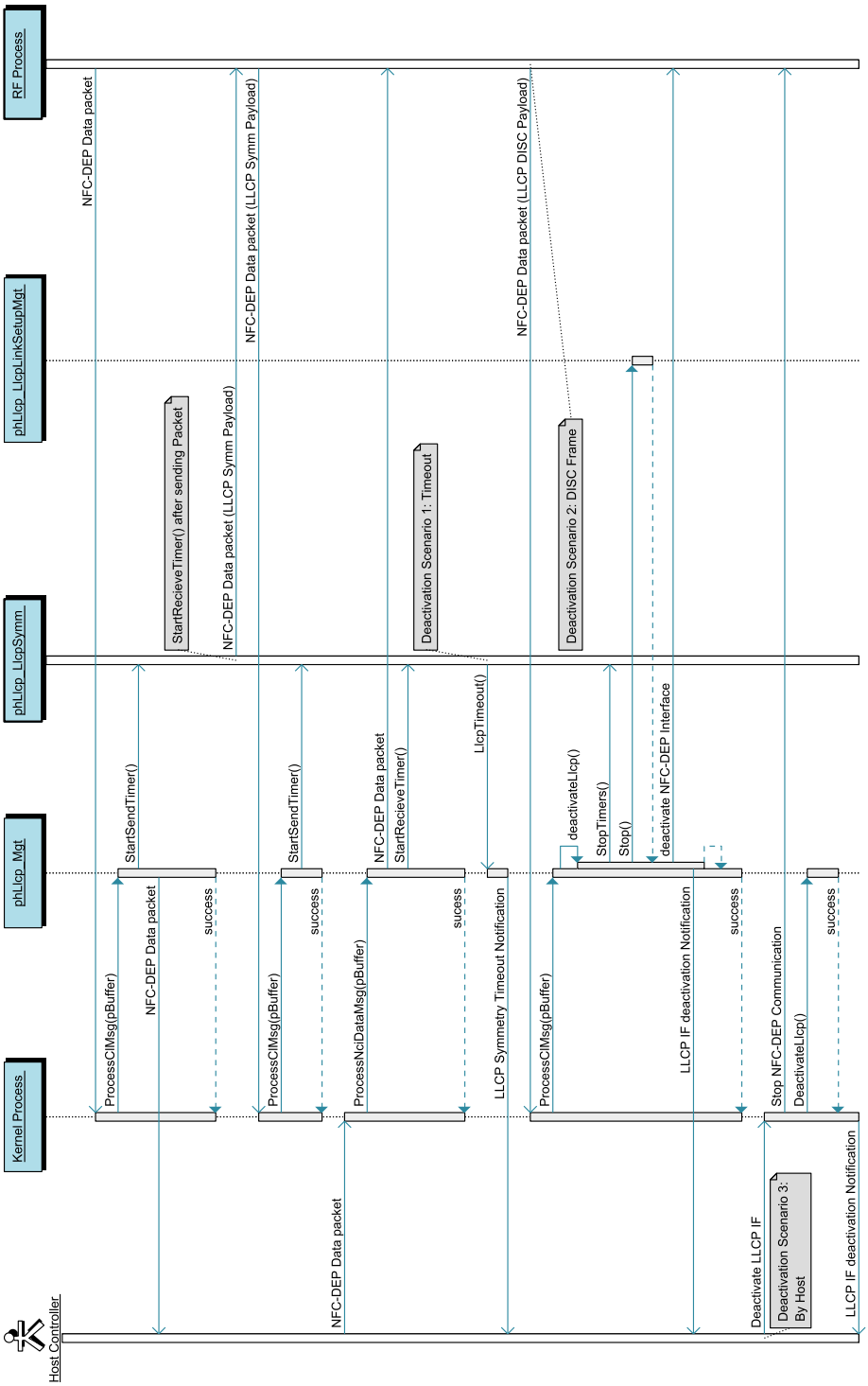


Figure B.6: PN547 Firmware LLCP High Communication Sequence Diagram

# Bibliography


[AMH+07] Heikki Ailisto, Tapio Matinmikko, Juha Häikiö, Arto Ylisaukko-oja, Esko Strömmer, Mika Hillukkala, Arto Wallin, Erkki Siira, Aki Pöyry, Vili Törmänen, Tua Huomo, Tuomo Tuikka, Sonja Leskinen, and Jarno Salonen. Physical browsing with NFC technology. Technical report, VTT Publications Register (Finland), 2007. 2.2.1, 2.7

[Ass13] Infrared Data Association. OBEX specification at IrDA.org. `http://www.irda.org/`, November 2013. 1.1

[BFLS11] T. Bolognesi, A. Frisiello, A. Lotito, and G. Spoto. NaviNFC: a solution for indoor navigation fully based on NFC. In *Proceedings of ASITA Conference*, pages 391–402, 2011. 2.3.1

[Bro13] Broadcom. Broadcom BCM2835, High Definition 1080p Embedded Multimedia Applications Processor. `http://www.broadcom.com/products/BCM2835`, October 2013. 4.1.3

[CDS74] Vinton Cerf, Yogen Dalal, and Carl Sunshine. Specification of Internet Transmission Control Program, 1974. 1.1, 2.4

[CRKH05] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *LINUX DEVICE DRIVERS*. O'Reilly, 3rd edition, 2005. 4.2.2

[ECM] ECMA. ECMA-340: Near Field Communication Interface and Protocol (NFCIP-1). 2.1.1, 2.1.2

[For06] NFC Forum. NFC Data Exchange Format (NDEF) Technical Specification, 2006. 2.3.1

[For11] NFC Forum. LLCP 1.1: Logical Link Control Protocol, Technical Specification, 2011. 2.1.2, 2.1.2, 3.2.4, 3.2, 3.3, 3.4, 3.2.4, 3.2.4, 3.2.4, 3.5, 3.6, 3.7, 3.2.4, 3.8, 3.2.4, 3.2.4

[For12] NFC Forum. NFC Controller Interface (NCI) Technical Specification, 2012. 3.2.4, 3.5, 3.6

[Fou13] The Raspberry Pi Foundation. Raspberry Pi. `http://www.raspberrypi.org`, October 2013. 4.1

[Gmb13] NXP Semiconductors Austria GmbH. MIFARE.net contactless Smartcard Technology. `http://www.mifare.net`, February 2013. 3.2.2

[Grü07] Stefan Grünberger. Analyse und Implementierung des IP-Protokolls über NFCIP-1. Master's thesis, Upper Austria University of Applied Sciences School of Informatics/Communications/Media, 09 2007. 2.3.2, 2.10, 2.11, 2.12, 2.13, 2.14, 2.15, 2.16

[Inc03] Motorola Inc. SPI Block Guide V03.06, 2003. 3.2.2

[ISO01] ISO/IEC. ISO/IEC 14443: Identification Cards — Contactless Integrated Circuit(s) Cards — Proximity Card, 2001. 2.1.2

[JIS05] JIS. JIS X 6319: Specification of Implementation for Integrated Circuit Cards, 2005. 2.1.2

[LGW11] Li Li, ZeHua Gao, and Yazhou Wang. NFC antenna research and a simple impedance matching method. In *Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on*, volume 8, pages 3968 –3972, aug. 2011. 3.2.1

[Lim12] ARM Limited. Cortex-M0 Technical Reference Manual Revision: r0p0, 2012. 4.1.1

[LM12] A. Lotito and D. Mazzocchi. OPEN-NPP: An Open Source Library to Enable P2P over NFC. In *Near Field Communication (NFC), 2012 4th International Workshop on*, pages 57 –62, march 2012. 2.3.1, 2.9

[LR10] Josef Langer and Michael Roland. *Anwendungen und Technik von Near Field Communication (NFC)*. Springer Berlin Heidelberg, 2010. 2.1, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 3.2.2

[Ltd13] ARM Ltd. ARM - The Architecture For The Digital World. `http://www.arm.com`, March 2013. 3.2.3

[Mas13] Brian Masney. gFTP Official Homepage. `http://gftp.seul.org`, October 2013. 2.3.2

[MMA09] H. Mika, H. Mikko, and Y.-o. Arto. Practical Implementations of Passive and Semi-passive NFC Enabled Sensors. In *Near Field Communication, 2009. NFC '09. First International Workshop on*, pages 69 –74, feb. 2009. 2.2.2

[npp06] NFC Data Exchange Format (NDEF) Technical Specification, 2006. 2.3.1

[oSC81] Information Sciences Institute University of Southern California. Internet Protocol. Technical report, The Internet Engineering Task Force, 1981. 1.1, 2.4

[PGLA00] Christina Parsa and J. J. Garcia-Luna-Aceves. Improving TCP performance over wireless networks at the link layer. *Mob. Netw. Appl.*, 5(1):57–71, March 2000. 2.4, 2.4.1

[Pro13a] The Linux Information Project. The Linux Information Project. `http://www.linfo.org/`, October 2013. 4.2.2

[Pro13b] The ProFTPD Project. ProFTPD, Highly configurable GPL-licensed FTP server software. `http://www.proftpd.org`, October 2013. 2.3.2

[RSP12] Jukka Riekki, Ivan Sanchez, and Mikko Pyykkönen. NFC-Based User Interfaces. Technical report, Dept. Computer Science and Engineering and Infotech Oulu, 2012. 1.1, 1.1

[SC01] Timothy Sherwood and Brad Calder. Patchable instruction ROM architecture. In *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*, CASES '01, pages 24–33, New York, NY, USA, 2001. ACM. 3.2.2

[Sem11] NXP Semiconductors. PN547 Firmware Architecture Design, 2011. 4.2.1

[Sem12a] NXP Semiconductors. PN5331B3HN Near Field Communication (NFC) controller. Datasheet, 2012. 3.2.1

[Sem12b] NXP Semiconductors. The I2C-Bus Specification and User Manual, 2012. 3.2.2

[SHR06] Timo Salminen, Simo Hosio, and Jukka Riekki. Enhancing Bluetooth connectivity with RFID. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications*, pages 36–41, 2006. 3.1.1

[SHYo07] Esko Strömmer, Mika Hillukkala, and Arto Ylisaukko-oja. Ultra-low Power Sensors with Near Field Communication for Mobile Applications. In Luis Orozco-Barbosa, Teresa Olivares, Rafael Casado, and Aurelio Bermúdez, editors, *Wireless Sensor and Actor Networks*, volume 248 of *IFIP International Federation for Information Processing*, pages 131–142. Springer US, 2007. 2.2.2, 2.8, 2.2.2

[TG13] Mike Thompson and Peter Green. raspbian.org. `http://www.raspbian.org`, October 2013. 4.2.2

[Wor05] PC/SC Workgroup. Interoperability Specification for ICCs and Personal Computer Systems, 2005. 2.3.1