# Entwicklung einer Applikation zum dynamischen Testen von analogen und digitalen Funktionen von RFID Readern

Dorian Achim Prill

26. Februar 2015

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik am Department
Technik und Informatik an der Hochschule für Angewandte
Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Henning Dierks
Zweitprüfer: Prof. Dr.-Ing. Ralf Wendel

Eingereicht am: 27. Februar 2015

**Dorian Achim Prill**

### Thema der Bachelorthesis

Entwicklung einer Applikation zum dynamischen Testen von analogen und digitalen Funktionen von RFID Readern

### Stichworte

NFC, RFID, ISO/IEC 14443, Testautomation, Qt5, NXP Semiconductors

### Zusammenfassung

Diese Arbeit befasst sich mit der Entwicklung einer grafischen PC Software zur Steuerung von NFC Readern für Testabläufe, wie sie zur Verifikation von Funktion und Standardkonformität benötigt werden. Es wird ein Überblick der verwendeten, sowie verwandten Technologien und Standards gegeben. In einer Anforderungsanalyse werden die spezifischen Kriterien erfasst, und ebenfalls bestehende Systeme untersucht. Darauf aufbauend wird das Konzept der Applikation erarbeitet und die Implementierung erklärt. Letztendlich folgt eine Verifikation der Funktionalität.

**Dorian Achim Prill**

### Title of the paper

Development of an application for dynamic testing of analog and digital functions of RFID readers

### Keywords

NFC, RFID, ISO/IEC 14443, Testautomation, Qt5, NXP Semiconductors

### Abstract

This work focuses on the development of a graphical PC software to control RFID readers for testing purposes, as required to verify function and conformity regarding certain standards. An overview of used and related technologies and standards is given. A requirements analysis reveals the specific criteria and investigates related software. Based on this, the concept for the application is being elaborated and the implementation explained. Finally, the functional verification of the software is done.

# Inhaltsverzeichnis

## Danksagung

Ich möchte mich für die Ermöglichung dieser Arbeit in der Firma NXP Semiconductors bei Herrn Renke Bienert bedanken, ebenso wie für die Betreuung während der Durchführung dieser. Gleichermaßen danke ich den Professoren Dr. rer. nat. Henning Dierks, der sich dazu bereit erklärt hat, die Erstprüfung zu übernehmen, und Dr.-Ing. Ralf Wendel, der die Zweitprüfung übernimmt.

# 1 Ziel dieser Arbeit

Um einen RFID Reader erfolgreich in einem kommerziellen Endprodukt einzusetzen, ist neben funktionellen Aspekten ebenfalls Konformität zu gewissen, vom Einsatzbereich abhängigen, Standards erforderlich.
Diese Standards fordern die Einhaltung von Prozeduren und Grenzwerten, sowohl analog als auch digital. Darunter fallen zum Beispiel die Reglementierung der Feldstärke im Raum, die maximale Abweichung vom vorgegebenen Modulationsindex und das Einhalten definierter Time Frames.
Die Abteilung Customer Application Support der Firma NXP Semiconductors sieht sich konfrontiert mit Fragestellungen, welche im Zusammenhang mit der Zertifizierung eines Endprodukts des Kunden stehen. Hierbei ist es wichtig, einen flexiblen und vor allen Dingen definierten Aufbau ohne Seiteneffekte zu haben, der das Ändern und Auswerten von Parametern, sowie das Ausführen definierter Test-Sequenzen ermöglicht.

Zu diesem Zweck wird eine PC-Applikation entwickelt, welche eine komfortable Steuerung eines Readers über eine grafische Oberfläche möglich macht. Die Applikation basiert auf einer bestehenden Bibliothek und ist so konzipiert, dass eine Erweiterung für andere, von dieser Bibliothek unterstützte Reader-Typen möglich ist.

# 2 Einführung

Im vorliegenden Abschnitt werden zunächst die zugrunde liegenden Technologien erläutert, um den recht weit gefassten Begriff RFID im Rahmen dieser Arbeit besser abzugrenzen.

## 2.1 RFID

RFID steht für Radio Frequency Identification, frei übersetzt etwa Funkfrequenz Identifikation oder Radiowellen Identifikation und wird eingesetzt um Objekte oder Personen zu Identifizieren. RFID Technologie umfasst eine Menge von Systemen zur kabellosen bzw. kontaktlosen [1] Übertragung von Daten im Frequenzbereich von 135 kHz bis hin zu 5.8 GHz[siehe Finkenzeller 2006, Abschnitt 2.3]. Die eingesetzten Frequenzen hängen stark von den jeweiligen Regulierungen vor Ort ab. Dabei ist der Begriff nicht auf eine technische Implementierung festgesetzt, da diese durch den konkreten Anwendungsfall diktiert wird. Allgemein wird die Reichweite des Systems maßgeblich von der verwendeten Freuqenz und der Art der verwendeten Transponder bestimmt. Dabei hat sich folgende Klassifizierung etabliert[siehe Finkenzeller 2006, Abschnitt 2.3]:

**Proximity coupling** 0 bis 10 cm

**Vicinity coupling** 10 cm bis 1 m

**Long Range** 1 m bis etwa 30 m

Bei einer Identifikation oder Authentifizierung von Personen beispielsweise, werden vorrangig sogenannte Proximity oder Vicinity (dt. Nähe) Systeme eingesetzt, etwa in Form einer Chipkarte, welche in einem Abstand von etwa 0-1 m vor ein Lesegerät gehalten werden muss. Diese Systeme Arbeiten nach dem Prinzip der induktiven Kopplung. Bei der industriellen Identifikation von Objekten hingegen, kommen vermehrt auch sog. Long Range Technologien zum Einsatz. Dabei werden große Stückzahlen von mit RFID-Tags versehenen Objekten durch Schleusen in der Liefer- oder Produktionskette bewegt, um ihre Daten automatisch und ohne weiteren Eingriff wie etwa dem Öffnen einer Liefereinheit auszulesen. Die Daten können in diesem Fall durch den Produzierenden bereits im Tag hinterlegt, oder aber zusätzlich durch einen Datenlogger während des Lieferungsweges akquiriert worden sein. Diese Systeme arbeiten nach dem Prinzip der Wellenausbreitung.

[1] contactless: pertaining to the achievement of signal exchange with, and supply of power to, the card without the use of galvanic elements (i.e. the absence of an ohmic path from the external interfacing equipment to the integrated circuit(s) contained within the card [siehe ISO/IEC 2008a, Abschnitt 3]

### 2.1.1 Bestandteile eines RFID Systems

Ein RFID System besteht aus einem Lesegerät, dem Reader[2], und einem elektronischen Etikett oder Kennzeichen, Tag genannt. Zur Steuerung des Readers und für die Verarbeitung der auf diesem Wege übermittelten Daten ist weiter ein übergeordnetes System nötig („Backend“). Der generelle Aufbau eines solchen Systems ist schematisch in Abbildung 1 dargestellt.
Der Reader versorgt das Tag über ein wechselndes Magnetfeld mit Energie und einem Takt. Der Takt ist dabei gleich der Träger- beziehungsweise Mittenfrequenz des verwendeten Standards, spricht man zum Beispiel über RFID bei 13.56 MHz, ist damit die Mittenfrequenz gemeint. Die Daten werden dann durch Modulation des Trägers aufgebracht. Das Tag antwortet über eine Lastmodulation des Feldes. Die damit erreichte Übertragungsrate ist allerdings deutlich geringer als die Mittenfrequenz, sie beträgt aktuell für gängige Standards nicht mehr als 848 kbit/s.

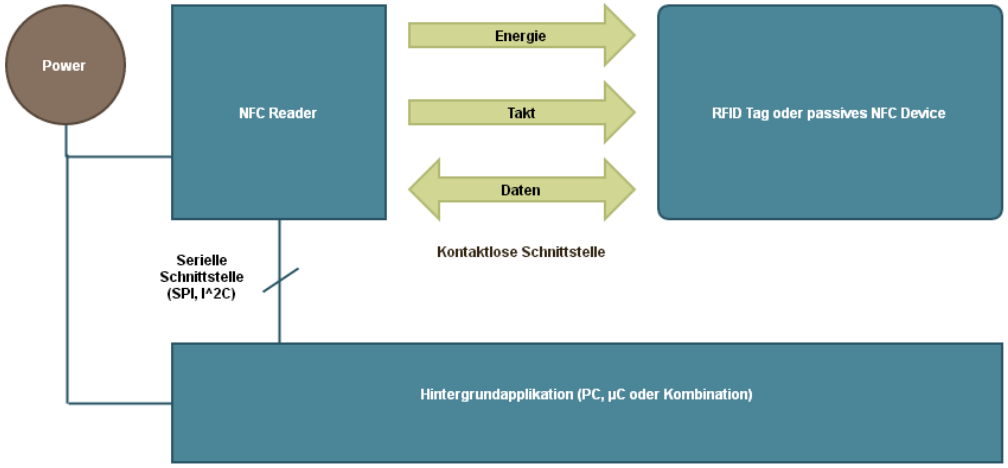


Abbildung 1: Schematische Darstellung eines RFID/NFC Systems mit passivem Tag

## 2.2 Near Field Communication

Da der verwendete Reader als NFC-Reader ausgewiesen ist, folgt hier die Erklärung des Begriffs NFC und die Beziehung zu RFID.

[2] Obwohl sich dieser Begriff aus den Anfängen im Sprachgebrauch durchgesetzt hat, ist es meist ein Lese- und Schreibgerät

Der Begriff NFC steht als Abkürzung für Near Field Communication, zu deutsch Nahfeldkommunikation, und wird üblicherweise verwendet um Kommunikationstechnologien, welche nach dem Prinzip der induktiven Kopplung bei 13.56 MHz arbeiten, zu beschreiben. Obwohl sich dieser Begriff zumeist auf einen speziellen von der ISO/ECMA [3] [4] beschriebenen Standard bezieht, wird er mehr oder weniger fälschlicherweise auch für proprietäre Formate genutzt. Dies ist dem Umstand geschuldet, dass der Name gleichzeitig auch einfach die Abkürzung für das technische Konzept ist.
Wenn im Rahmen dieser Arbeit vom synonym NFC Gebrauch gemacht, und sofern nicht weiter differenziert wird, sind explizit die durch das NFC-Forum erarbeiteten, und von der ISO/ECMA übernommenen Standards gemeint. Diese wiederum basieren dabei auf bereits bestehenden RFID Standards.
Anwendung findet NFC zum Beispiel in Pairing- oder Micropayment-Anwendungen, eine moderne Implementierung ist, auf mobilen Geräten beispielsweise, Android Beam, das über die NFC Schnittstelle ein Bluetooth-Pairing durchführt.

Im folgenden wird nicht mehr strikt zwischen RFID- und NFC Reader unterschieden, da diese Klassifizierung für den hier betrachteten Fokus eine untergeordnete Rolle spielt. Fortan wird der Begriff Reader verwendet.

### 2.2.1 Standardisierung

Um die Standardisierung und Entwicklung von NFC voranzutreiben, schlossen sich die Firmen NXP Semiconductors, Sony und Nokia zusammen und gründeten ein Konsortium genannt, NFC-Forum, welchem sich mittlerweile viele weitere Unternehmen aus Industrie und Finanzwelt angeschlossen haben.
Die Ergebnisse der Arbeit des NFC-Forum führten zur Festlegung zweier ISO/ECMA Standards für NFC Geräte. Das NFC-Forum existiert weiterhin und hat weitere Standards für höhere Protokollfunktionen entwickelt. Ausserdem kümmert sich das NFC-Forum um die Zertifizierung von Geräten und um die Promotion von NFC.
Die Standards sind:

**NFCIP-1** ist definiert in ISO 18092 respektive ECMA 340. Er legt die drei Kommunikationsmodi für NFC Geräte fest. Ferner spezifiziert er Modulationsschemata, Kanalkodierung, Übertragungsraten und die Methoden zur Initialisierung und Antikollision[ECMA 2013a]. Er basiert auf ISO/IEC 14443 und außerdem weiteren Standards, welche jedoch für die Arbeit nicht relevant sind.

**NFCIP-2** ist definiert in ISO 21481 respektive ECMA 352. Er definiert den Mechanismus zur Auswahl des Kommunikationsmodus ohne Störung bestehender Kommunikation[ECMA 2013b].

[3] ISO: International Standardization Organization
[4] ECMA: European Computer Manufacturers Association

## 2.3 Smart Cards

Eine „Smart Card“, manchmal auch „Integrated Circuit Card“(ICC) genannt, ist eine spezielle Form einer kleinen Karte, welche über einen in einer Fräsung eingelassenen Mikrochip verfügt. Dieser Chip kann ein anwendungsspezifischer Schaltkreis oder ein Mikroprozessor sein. Diese Karten nenn man dann auch Speicherkarten und Prozessorkarten. An einer fest definierten Position auf der Karte sitzen Kontaktflächen, über welche ein Reader dann mit dem Chip kommunizieren kann. Für neuere Karten ist ebenfalls ein konktaktloses Interface spezifiziert.
Typische Beispiele für eine Smart Card sind etwa die Kreditkarte oder der neue deutsche Personalausweis, wobei letzterer ausschließlich über ein konktaktloses Interface verfügt. Die Merkmale von Smart Cards sind in der Norm ISO/IEC 7816 zusammengefasst, welche wiederum auf der ISO/IEC 7810 (Formate für Identitätsdokumente) aufbaut. Die kontaktlose Kommunikation wird jedoch in der ISO/IEC 14443 beschrieben.
Moderne Karten können bereits über ein Java-basiertes Betriebssystem verfügen, welches das Erstellen von Dateien und Applikationen ermöglicht.


Abbildung 2: Zwei kontaktlose Chipkarten mit NXP MIFARE Technologie

## 2.4 ISO/IEC 14443

Die ISO/IEC 14443 ist eine übergeordnete Norm, welche die Eigenschaften von kontaktlosen Smart Cards beschreibt. Diese setzt sich zusammen aus folgenden Teilen:

**ISO/IEC 14443-1** Physikalische Eigenschaften der Karte.

**ISO/IEC 14443-2** Modulationsverfahren und Kanalkodierung.

**ISO/IEC 14443-3** Initialisierung der Karte und Behandlung von Kollisionen im Feld.

**ISO/IEC 14443-4** spezifiziert das eigentliche Übertragungsprotokoll. Hierfür hat sich auch der Begriff T=CL (Transmission = Contactless) etabliert.

Außerdem gibt es in der Norm eine weitere Unterscheidung der Karten in Typ A und Typ B, welche sich in den Teilen 14443-2 und 14443-3 und in bestimmten Merkmalen auch in 14443-4 anders verhalten.
Der Standard bezeichnet die Systemkomponenten wie folgt:

**PCD** Proximity Coupling Device

**PICC** Proximity Integrated Circuit Card

Diese Bezeichnungen werden hier ebenfalls verwendet, wenn sich der Text auf den Standard bezieht.

### 2.4.1 Teil 3: Antikollision und Initialisierung

In der ISO/IEC ist eine Initialisierungssequenz für Karten spezifiziert. Sinn dieser ist es, eine Karte für weitere Kommunikation auszuwählen.
Das PCD sendet als initiales Kommando einen sogenannten RequestA-Befehl, und das Tag antwortet auf diesen mit einem speziellen Befehl, der Answer to RequestA (ATQA) . Danach befindet sich die PICC im READY Zustand und kann selektiert werden. Für die möglichen Zustände einer PICC siehe Abbildung 4.
Es können jedoch mehrere PICCs gleichzeitig im Feld sein. Da diese alle gleichzeitig antworten, entstehen sogenannte Kollisionen im Antwortsignal. Entsteht eine Kollision , wird ein Frame für die gesamte Dauer moduliert, was vom Reader dann als solche erkannt werden kann. Um diesen Konflikt zu lösen, gibt es wiederum spezielle Kommandos und Sequenzen.
Bei Typ A PICCs werden Kollisionen durch eine bitweise Vervollständigung der UID aufgelöst. Das genaue Verfahren hierfür wird in Abschnitt 5 behandelt. Die Aktivierungssequenz aus der Norm[ISO/IEC 2011] ist in Abbildung 3 dargestellt.

Die in der Sequenz verwendeten Befehle zeigt die folgende Übersicht. Diese wurde aus dem Originaldokument[siehe ISO/IEC 2011] übersetzt, und stellenweise vereinfacht, um sie im Rahmen der Arbeit zu halten.

**REQA und WUPA** Steht für Request A und Wakeup A. Diese Kommandos werden von dem PCD verwendet, um nach PICCs im Feld zu suchen. Sie bewirken in der PICC einen übergang vom IDLE in den READY bzw. vom HALT in den READY* Zustand . Bitkodierung: REQA = 0x26, WUPA = 0x52. Größe: 1 Byte. Abweichende Kodierungen sind für eine optionale Methode[siehe ISO/IEC 2011, Annex C] sowie proprietäre Formate vorgesehen.
Die PICC antwortet mit ihrer ATQA

**ATQA** Answer to Request A, die Antwort der PICC auf den Request A Befehl.
Nach dem Erhalt eines REQA Kommandos antworten alle PICCs im IDLE Zustand gleichzeitig mit ihrer ATQA .
Nach dem Erhalt eines WUPA Kommandos antworten alle PICCs im IDLE oder HALT Zustand mit ihrer ATQA.
Hauptbestandteil der ATQA sind die Größe der UID (möglich sind single = (3+1) Byte, double = (6+1) Byte, triple = (9+1) Byte) und der Bit-Frame Anticollision Indikator, mit welchem die PICC anzeigt, ob sie die Antikollisionprozedur unterstützt.

**ANTICOLLISION** Dieses Kommando wird während der Antikollisionsschleife verwendet. Es kann je Cascade-Level mehrmals ausgeführt werden.
ANTICOLLISION hat folgende Struktur:

Tabelle 1: Aufbau des Anticollision Kommandos

| Bezeichnung | Beschreibung | Größe |
|---|---|---|
| SEL | Select code, enthält das Cascade-Level | 1 Byte |
| NVB | Anzahl der gültigen Bits einer partiellen UID | 1 Byte |
| UID CLn | Partielle UID für Cascade-Level n | 0-40 bit |

Die PICC antwortet mit Ihrer UID.

Solange NVB keine 40 gültigen Bits angibt (NVB = 0x70), wird der Befehl ANTICOLLISION genannt. Wenn 40 gültige Bits erreicht sind, wird eine Prüfsumme (CRC_A) angehängt. Für diesen Fall wird der Befehl SELECT genannt.

**SELECT** Dieses Kommando wird während der Antikollisionsschleife verwendet, nach den eigentlichen ANTICOLLISION Kommandos auf jedem Cascade-Level.
SELECT hat folgende Struktur:

Tabelle 2: Aufbau des Select Kommandos

| Bezeichnung | Beschreibung | Größe |
|---|---|---|
| SEL | Select code, enthält das Cascade-Level | 1 Byte |
| NVB | 0x70 | 1 Byte |
| UID CLn | Partielle UID für Cascade-Level n | 0-40 bit |
| CRC_A | Typ A Prüfsumme. Nur bei NVB = 0x70 | 2 Byte |

Die PICC antwortet mit ihrer SAK - Select acknowledge.

**SAK** Select acknowledge Kommando. Es besteht aus 3 Bytes, dem eigentlichen SAK-Byte und einer Prüfsumme (CRC_A, 2 Byte). Das SAK Byte zeigt an, ob die UID vollständig ist.

Tabelle 3: Kodierung der SAK

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | Beschreibung |
|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | 1 | x | x | Cascade Bit gesetzt, UID nicht vollständig |
| x | x | 1 | x | x | 0 | x | x | UID vollständig, Konform zu 14443-4 |
| x | x | 0 | x | x | 0 | x | x | UID vollständig, nicht konform zu 14443-4 |

Wenn die UID vollständig ist, und die PICC die SAK gesendet hat, geht sie vom READY/READY* Zustand in den ACTIVE/ACTIVE* Zustand über. Andernfalls verbleibt die PICC im READY/READY* Zustand. Siehe Abbildung 4.

**HLTA** Der Halt A Befehl. Er dient dazu, aktivierte PICCs in einen Ruhezustand (HALT) zu versetzen, so dass sie nur noch auf bestimmte Befehle (z.B WUPA) antworten.

Tabelle 4: Aufbau des Halt A Kommandos

| | | |
|---|---|---|
| 0x50 | 0x00 | CRC_A |

Die Karte sendet keine Antwort auf dieses Kommando. Stattdessen wird folgendermaßen vorgegangen:
Moduliert die PICC das Feld innerhalb von 0,1 ms nach Erhalt von HLTA in irgendeiner Weise, wird der Befehl als nicht bestätigt angesehen. Eine entsprechende Wartezeit muss von dem PCD eingehalten werden.

Hinweis: Die Befehle nutzen teilweise unterschiedliche Bit-Frames , da diese jedoch von der Reader Library verwaltet werden, sind sie hier nicht aufgeführt.

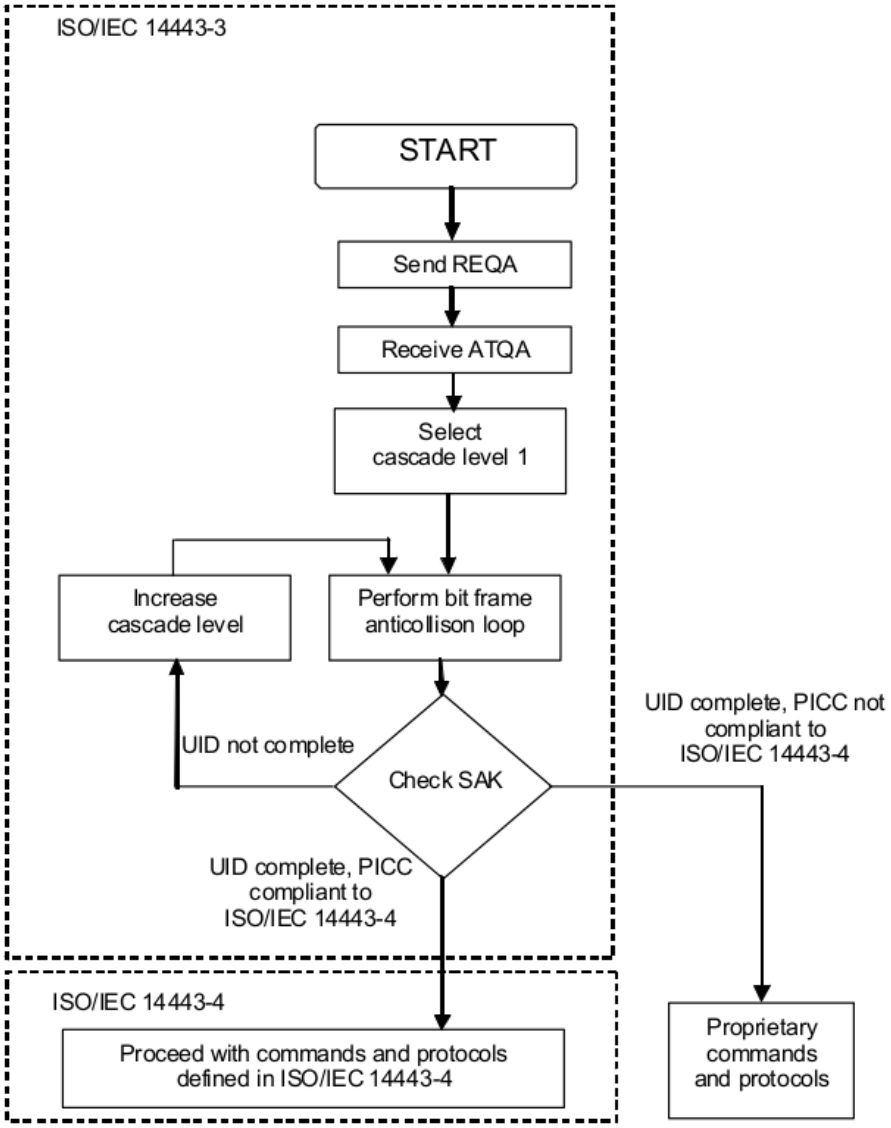


Abbildung 3: Initialisierung und Anticollision aus Sicht des PCD[siehe ISO/IEC 2011, Abschnitt 6.5.1]

Das Verhalten der PICC während der Aktivierungssequenz kann aus dem allgemeinen Zustandsdiagramm der PICC abgeleitet werden. Die Bedeutung der Symbole für Abbildung 4 ist wie folgt:

**AC** Antikollision-Befehl bei zutreffender UID

**nAC** Antikollision-Befehl bei nicht zutreffender UID

**SELECT** Select-Befehl bei zutreffender UID

**nSELECT** Select-Befehl bei nicht zutreffender UID

**RATS** Request-Answer-To-Select Befehl vorereitend für die nächste Protokollebene ISO/IEC 14443-4

**DESELECT** Deselektionsbefehl für höhere Protokollebene ISO/IEC 14443-4

**Error** Übertragungs- oder Framing-Fehler detektiert.

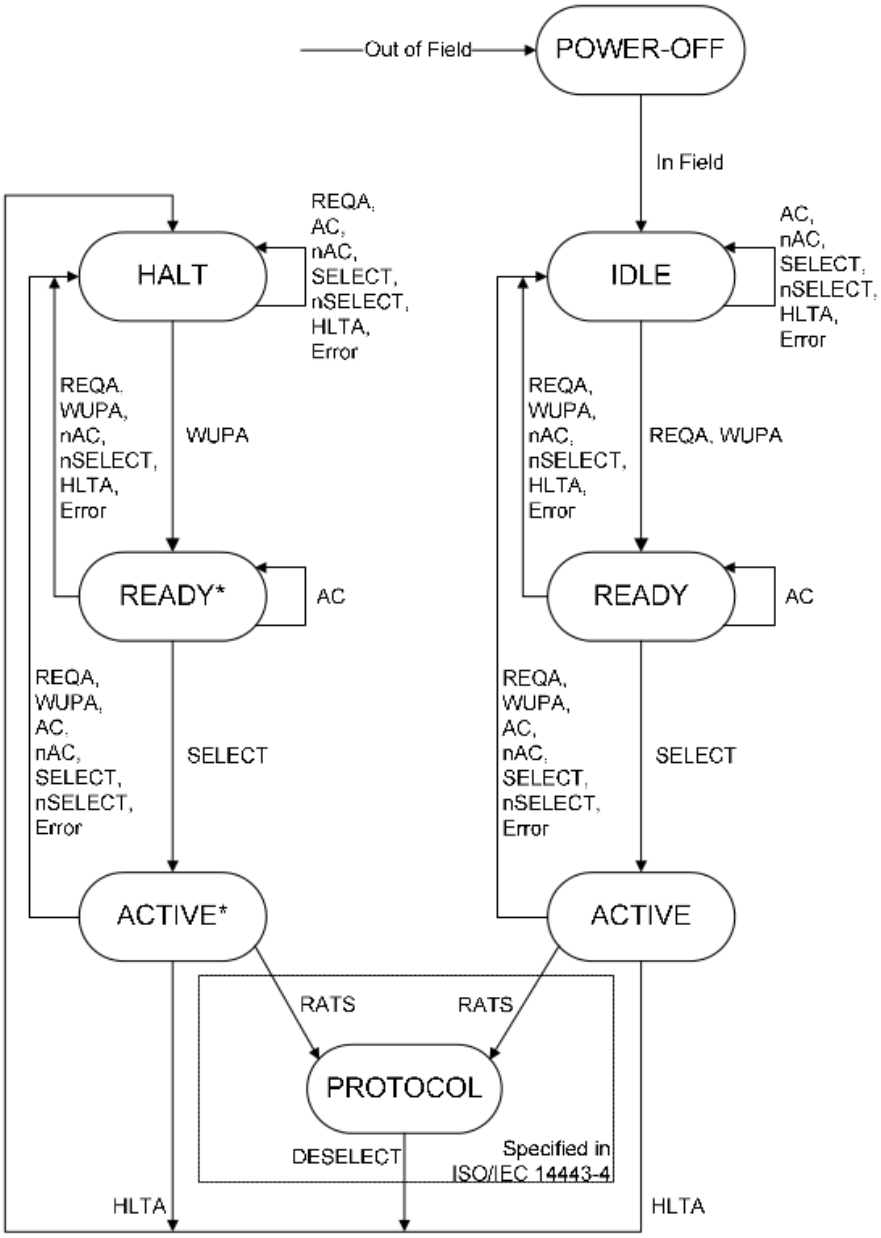


Abbildung 4: Zustandsdiagramm der PICC[siehe ISO/IEC 2011, Abschnitt 6.3]

### 2.4.2 Teil 4: Übertragungsprotokoll

Das Übertragungsprotokoll beschreibt die oberste Schicht der ISO/IEC 14443 Norm. Sie dient dem Austausch von Daten in Paketen, sogenannten Application Protocol

Data Unit(s) (APDU). Mit diesen kann wiederum auf Applikationen und Dateien auf der Karte zugegriffen werden, wie bei kontaktbehafteten Smart Cards auch.

Die hier wichtigen, Part 4 spezifischen Befehle

**RATS** Mit Request Answer To Select fordert das PCD eine Selektion der PICC auf Protokollebene. Die PICC antwortet mit ihrer spezifischen Antwort, der Answer To Select (ATS).
Diese Antwort teilt dem PCD die unterstützten Kommunikationsmöglichkeiten der PICC mit.

Tabelle 5: Aufbau des RATS Kommandos

| Start Byte | Parameter Byte | Integriätsmechanismus |
|---|---|---|
| 0x50 | Oberes Nibble: FSDI, Unteres Nibble: CID | 1 Byte CRC1 + 1 Byte CRC2 |

FSDI: beschreibt die Frame Größe des PCD als kodiertes Integer.
CID: Eine Ganzzahl zur logischen Adressierung der PICC auf Protokollebene. Ersetzt die UID.

**ATS** Answer To Select, die spezifische Antwort der PICC auf die RATS. Sie Teilt dem PCD die unterstützten Parameter der PICC mit. Sie ist folgendermaßen aufgebaut.

Tabelle 6: Aufbau der ATS

| Längen Byte | Fomat Byte | Interface Bytes | Historische Bytes | Integritätsmech. |
|---|---|---|---|---|
| Beschreibt die Länge der ATS. Das Längen-Byte selbst wird eingerechnet, die Prüfsumme nicht | u.A. Frame Größe der PICC | 3 Bytes, kodieren die Bitraten, Timings und Protokolloptionen | 1-N Bytes für generelle Informationen, hier nicht relevant | 1 Byte CRC1 + 1 Byte CRC2 |

**PPS** Steht für Protocol Parameter Selection, also die Auswahl der Protokollparameter. Hier werden die Bitraten in beide Richtungen ausgewählt. Die PICC muss die Anfrage bestätigen. Diese Kodierung der Bitraten erfolgt mit Ganzzahlen: 106 kbit/s = 0, 212 kbit/s = 1, 424 kbit/s = 2, 848 kbit/s = 3.

Abbildung 5 zeigt die gesamte Aktivierungssequenz und die Zugehörigkeit zu den verschiedenen Teilen der Norm.[siehe ISO/IEC 2008b, Abschnitt 5]

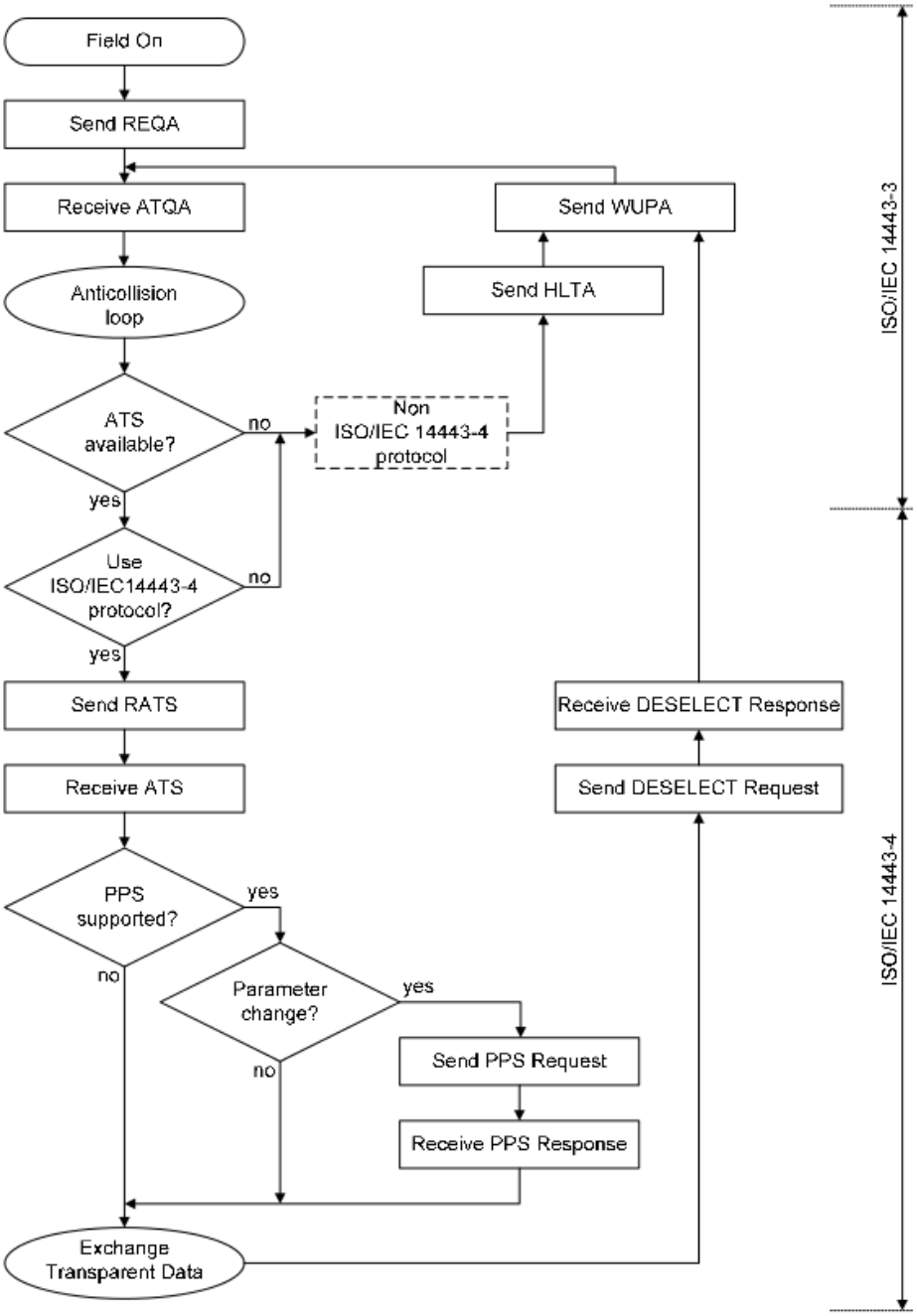


Abbildung 5: Vollständige ISO/IEC 14443 Typ A Aktivierungssequenz [siehe ISO/IEC 2008b, Abschnitt 5]

## 2.5 Verwendete Hardware

Die von der Software direkt zu unterstützende Hardware ist der NXP NFC Reader mit der Bezeichnung CLRC663, auch einfach RC663 genannt. Dieser Reader untersützt eine vielzahl gängiger Standards, darunter eben auch ISO/IEC 14443.
Ein Evaluationsboard steht zur Verfügung und wird zum Testen der Software genutzt. Die Kommunikation mit dem Reader kann über die Serielle Schnittstelle des Rechners erfolgen, oder, unter Verwendung eines NXP LPC1769 ARM Cortex-M3 Mikrocontrollers mit entsprechender Firmware, über USB. Abbildung 6 zeigt die Nutzungsstruktur im Blockdiagramm.

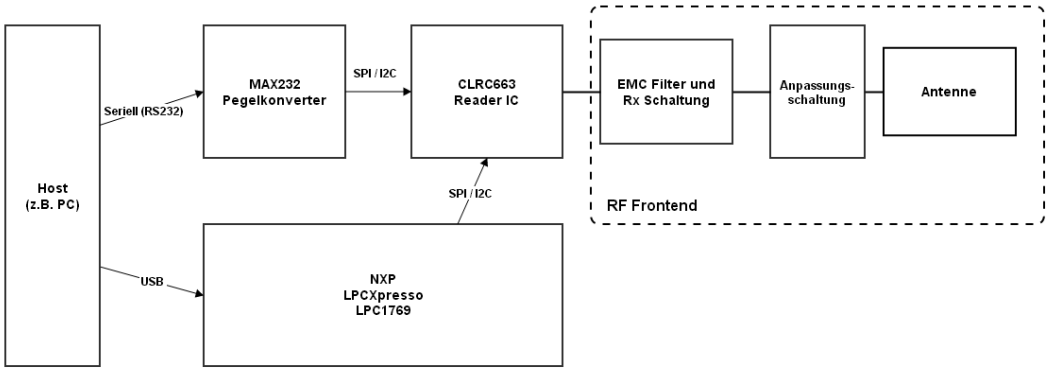


Abbildung 6: Schematisches Blockdiagramm der Systemkomponenten

Abbildung 8 zeigt das Evaluationsboard. Im rechten Teil mit der Aussparung sitzt die Schleifenantenne. Direkt daran auf dem schmaleren Verbindungstück sitzt die Anpassungsschaltung des RF Transceivers vor dem EMC Filter und dem eigentlichen Chip, welcher in dem relativ kleinen, schwarzen rechteckigen Gehäuse in nächster Nähe untergebracht ist.
Restliche Komponenten auf dem Board sind in der Nähe des seriellen Anschlusses ein RS232-Konverter auf der Unterseite, und am oberen Rand der Anschluss für die Spannungsversorgung mit den dazugehörigen Spannungsreglern auf der Unterseite.
Abbildung 7 beschreibt die Anordnung noch ein mal grafisch.

Desweiteren ist das Evaluationsboard für die Nutzung in Verbindung mit einem NXP LPCXpresso Microcontroller Board vorgesehen. Dieser kann über die Pin-Header auf dem Board (im Bild links unten) via Steckverbindung verbunden werden. Abbildung 9 zeigt das Board mit aufgebrachtem LPC1769 Board.

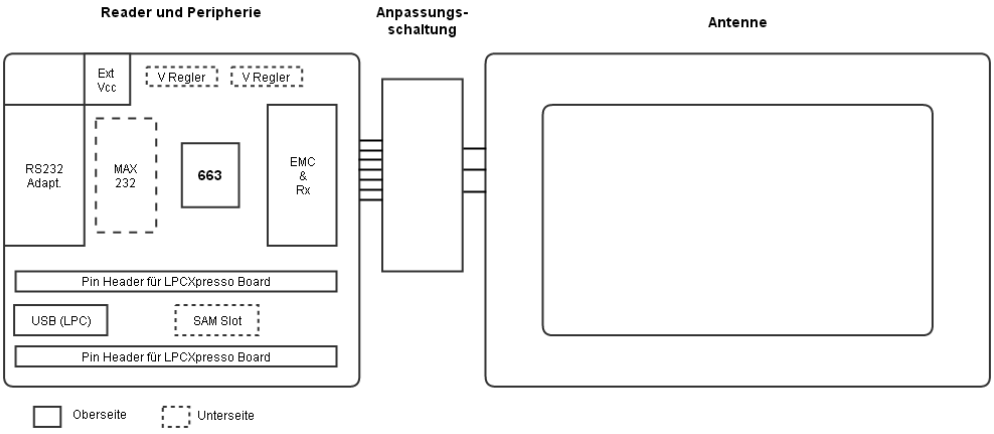


Abbildung 7: Schematische Darstellung der Komponentenanordnung

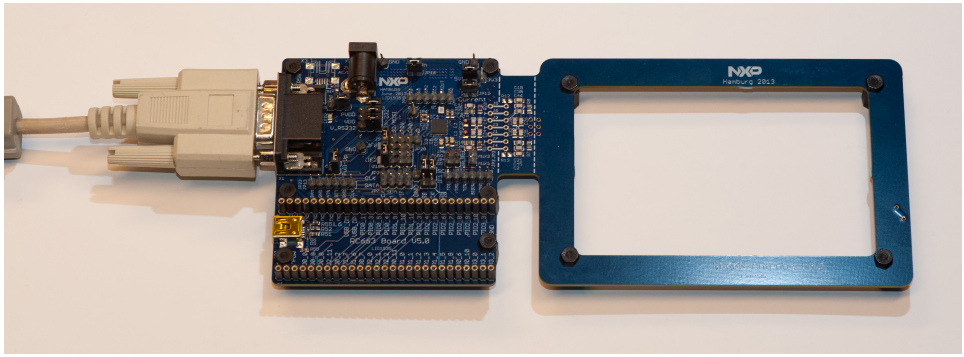

Abbildung 8: Evaluationsboard in Konfiguration für die Nutzung mit serieller Schnittstelle am PC

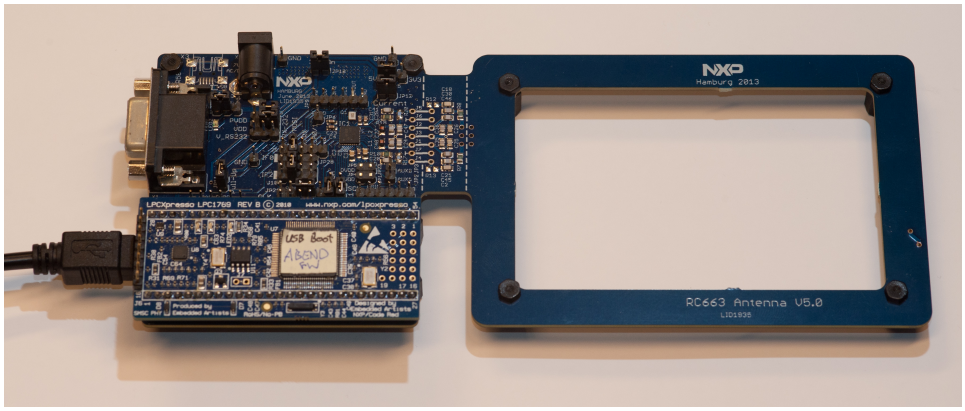


Abbildung 9: Evaluationsboard mit aufgestecktem NXP LPC 1769

## 2.6 NXP NFC Reader Library

NXP Semiconductors bietet eine Bibliothek für einige ihrer Reader Produkte an. Im folgenden Abschnitt wird kurz die Architektur dieser Bibliothek beschrieben.

Die NXP NFC Reader Library ist eine modulare Software-Bibliothek, geschrieben in C. Sie bietet dem Anwender eine API, welche es ermöglicht, Teile für eine eigene Applikation für kontaktlose Reader zu verwenden.
Die Bibliothek ist nach einem Schichtenmodell aufgebaut. Diese modulare Struktur hat den Vorteil, dass einzelne Komponenten ausgetauscht werden können, ohne das sich die Nutzung bzw. API der höheren Schichten ändert.
Diese Schichten sind, bei der untersten beginnend:

**Bus Abstraktionsschicht** Implementiert das Kommunikationsinterface zwischen Reader IC und Host Gerät. Zur Verfügung stehen unter Anderem eine serielle Schnittstelle für Windows (COM), eine serielle Schnittstelle für Linux (Pipe) und, wie hier genutzt, die Schnittstelle für USB über die entsprechende Firmware des LPC1769 (ABEND).
Im folgenden referenziert als BAL (Bus Abstraction Layer).

**Hardware Abstraktionsschicht** Implementiert die spezifischen Datentypen und Funktionen für die verschiedenen Reader ICs. Im folgenden referenziert als HAL (Hardware Abstraction Layer).

**Protokoll Abstraktionsschicht** Implementiert die Funktionen für die Kartenaktivierung und Übertragungsprotokolle. Im folgenden referenziert als PAL (Protocol Abstraktion Layer) .

**Applikationsschicht** Implementiert Befehle für weitere kontaktlose Protokolle und Kommandos wie zum Beispiel NXP MiFare Classic, Ultralight und Derivate, DESFire, Sony FeliCa. Genannt Application Layer, AL.

Abbildung 10 zeigt den Aufbau der Reader Library. Einige Module wurden in der Übersicht zur exemplarischen Darstellung weggelassen.

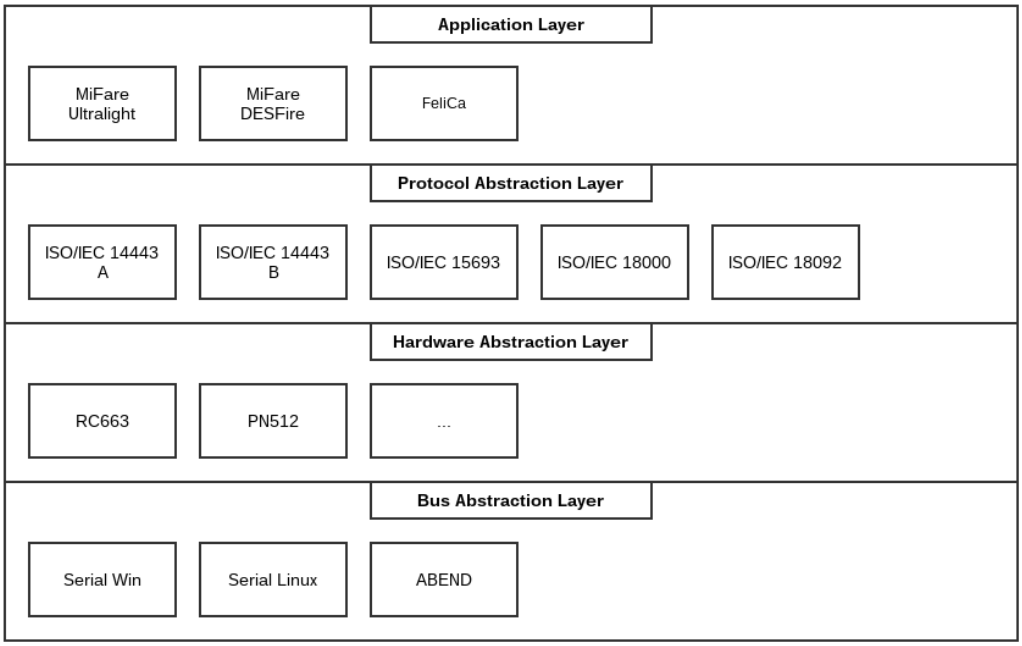


Abbildung 10: Schichtenmodell der Reader Library (reduzierte Darstellung)

# 3 Anforderungsanalyse

Die Anforderungen an die Software werden direkt vom Auftraggeber, in diesem Fall also von der Abteilung Customer Application Support (CAS) definiert.

## 3.1 Use Case: Vorverifikation

Der typische Anwendungsfall der Software liegt im Bereich der Vorverifikation von Standardkonformität einer NFC Reader Implementierung. Vorverifikation heisst in diesem Kontext, dass eine Überprüfung der kritischsten Parameter bereits erfolgen sollte, bevor ein Verifizierungslabor beauftragt wird. Diese Prozedur ist kosten- und zeitintensiv und es soll deswegen so gut als möglich sichergestellt werden, dass die Tests im ersten Durchgang bestanden werden.
In der Regel nähert sich ein Kunde der Abteilung CAS mit der Bitte um Unterstützung bei dem Design seiner Implementierung.
Im Kern dieses Vorgangs steht die Optimierung der RF Schnittstelle. Dabei gilt es unter anderem folgende Fragen zu beantworten:

1. Ist die Feldstärke ausreichend?
2. Ist die Stromaufnahme unter allen Randbedingungen innerhalb der Spezifikation?
3. Werden die Grenzen für den Modulationsindex eingehalten?
4. Werden die spezifizierten Signalformen eingehalten (z.B. Rise- und Fall-Time)?
5. Ist die Signalstärke am Empfänger groß bzw. klein genug?
6. Wieviele Tags können gelesen werden?

Soll ein bestimmter Aufbau nun hinsichtlich seiner RF Leistung optimiert werden, gilt es, die optimalen Einstellungen des Readers bezogen auf oben stehende Anforderungen zu ermitteln. Da das analoge RF Modul eines Readers ein breites Spektrum von Einstellungsmöglichkeiten bietet, sind mitunter einige Iterationen bei der Parameteranpassung erforderlich.
Dies erfordert außerdem ein definiertes Hard- und Software Setup bei dem Nebeneffekte ausgeschlossen oder bekannt sind. Das verwendete Evaluationsboard bietet die Möglichkeit, die Antenne sowie den Anpassungsschaltkreis abzutrennen. So lässt sich das Board mit einer vom Kunden bereitgestellten Antennen-Hardware benutzen.
Dies ist zum Beispiel hilfreich, wenn sich Nebeneffekte durch das Layout des Kunden auf das analoge Interface auswirken, diese lassen sich so ausschließen und gegebenenfalls eingrenzen.
Die Verwendung einer bekannten, bereits getesteten Software ermöglicht eine transparente und nachvollziehbare Funktionsprüfung der Hardware des Kunden. Mit diesem

Aufbau und der Testapplikation ist es nun möglich, den Optimierungsprozess durchzuführen.

## 3.2 Hardware und Systemanforderungen

Das Programm soll auf gängigen PCs mit Windows Betriebsystem lauffähig sein. Eine Portierung ist derzeit nicht vorgesehen, aber denkbar. Das Programm soll binär distributierbar sein. Dabei ist eine 32-Bit Version ausreichend, da diese mit den entsprechenden Bibliotheken auch auf 64-Bit Systemen lauffähig sind

## 3.3 Anforderung an die grafische Benutzerschnittstelle

Es soll eine grafische Benutzeroberfläche bereitgestellt werden, welche es dem Nutzer erlaubt, Operationen durch klicken eines dedizierten Buttons auszuführen. Dabei sind elementare, sowie abstrahierte Operationen bezogen auf die Hardware wie auch die Protokollfunktionen zu implementieren. Hardware Funktionen sind:

**Verbindungsauf- und abbau mit dem Reader**
: Initialisieren der Kommunikationskomponente

**Lesen und Schreiben von Registern**
: Direkter Zugriff auf die 8 bit Register durch Eingabe von Hexadezimalzahlen

**An- und auschalten des Trägersignals**
: Das Trägersignal liefert die Energie für die Tags. Die Feldstärke ist ein wichtiger Parameter und wird extern mit Messgeräten erfasst.

**Reset des Readers**
: Rücksetzen des Readers in einen Grundzustand. Entweder durch einen Soft Reset (Interner Mechanismus ohne Unterbrechung der Spannungsversorgung) oder, wenn nicht vom IC unterstützt, Rücksetzen in den Idle-Zustand.

**Laden der Protokollspezifischen Register**
: Um den Zustandsautomaten und das Analoge Interface des Readers für ein bestimmtes Protokoll und die Übertragungsrate zu konfigurieren ist eine Funktion zu implementieren.

**Low Power Card Detection**
: Implementierung der LPCD, einem Stromsparenden Polling Modus des Readers.

Auf Protokollebene sollen die ISO/IEC 14443-3 und 14443-4 Type A Kommandos unterstützt werden, sowie einige ISO/IEC 14443-4 Kommandos.
Diese sind

**ISO/IEC 14443-3 Type A**

1. Request A

2. Anticollision (für jedes der drei Cascade-Level)
3. Select (für jedes der drei Cascade-Level)
4. Wakeup A
5. Halt A

**ISO/IEC 14443-4 Type A**

1. Request Answer To Select (RATS)
2. Protocol Parameter Selection (PPS) mit Auswahl der Bitraten-Integer

**ISO/IEC 14443-4**

1. Exchange (Datenaustausch auf Protokollebene)
2. Deselect (Analog zu Halt A versetzt es die Karte aus der höheren Protokollebene in den Halt-Zustand)

## 3.4 Analyse bestehender Systeme

Um ein geeignetes Format für die Software festzulegen, werden zuerst vorhandene Programme betrachtet.

### 3.4.1 PC-Serial Skript Tool

Bisher wurde für das Produkt RC663 nur eine simples Skript-Programm geliefert, welches nicht auf der Reader Library basiert. Es bietet ausschliesslich direkten Registerzugriff via Read/Write und einige wenige für den Nutzer zu Verfügung stehende globale Speicherplätze innerhalb der Skriptumgebung. Außerdem gibt es Assembler-ähnliche Befehle wie Jumps/Labels und Move. Das Programm arbeitet mit der seriellen Schnittstelle in Windows. Folgende Tabelle zeigt die Vor- und Nachteile auf einen Blick.

Tabelle 7: Vorteile und Nachteile des PC-Serial Skript Tools

| Vorteile | Nachteile |
|---|---|
| - Direkter Zugriff auf Registerebene | - Keine abstrahierten Funktionen, weder für Hardware noch Protokoll |
| - Möglichkeit Abläufe als Skript festzuhalten (Speichern & Laden) | - Kein Bibliotheksmechanismus, Funktionsblöcke müssen in jedem Skript neu auscodiert werden<br>- Langer und unübersichtlicher Code<br>- Kein intuitiv bedienbares GUI |

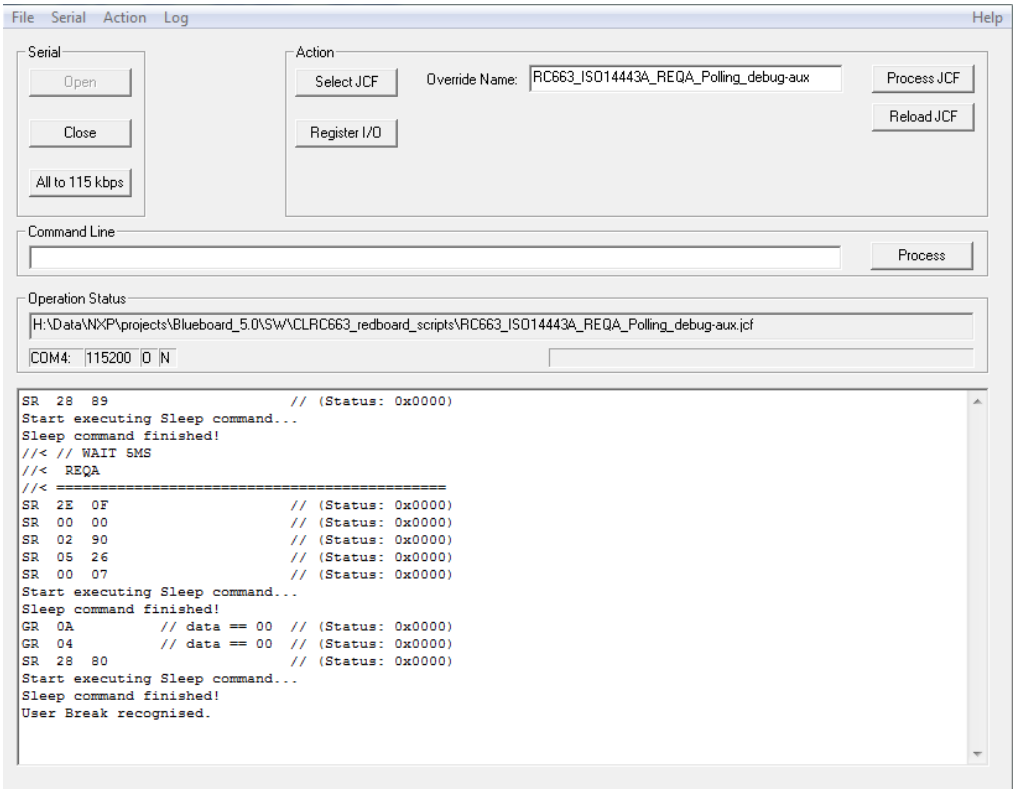


Abbildung 11: PCSerial Skript Tool

### 3.4.2 MifareWnd

MifareWnd ist eine Testsoftware basierend auf einem älteren Reader IC, welches das Nutzen von höheren Protokollfunktionen aus einer grafischen Oberfläche ermöglicht. Es implementiert die Funktionen der ISO/IEC 14443-3 Typ A und des Übertragungsprotokolls nach ISO/IEC 14443-4 sowie proprietäre MIFARE Kommandos. Es ist ausdrücklich gewünscht, dass sich die Software an dem Aufbau dieses Programms orientiert. Da dieses Programm nicht auf der Reader Library basiert, ist eine Wiederverwendung von Ressourcen nicht möglich.
Die in Abbildung 12 sichtbare Gruppierung von Interface, Reader und Card Kommandos macht in einer ähnlichen Form hier sicher auch Sinn. Die Rückverfolgung von Abläufen mittels einer Historie ist ebenfalls wünschenswert.
Durch die Aufteilung in verschiedene Fenster ist hier eine Duplizierung mancher Funktionen zu erkennen, die möglichst vermieden werden soll.

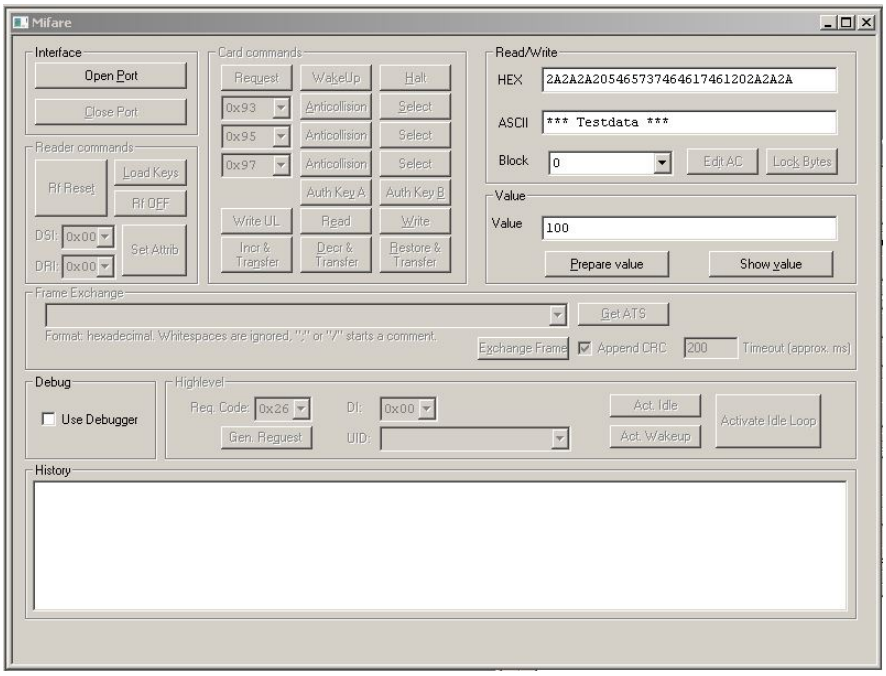


Abbildung 12: MifareWnd Hauptfenster

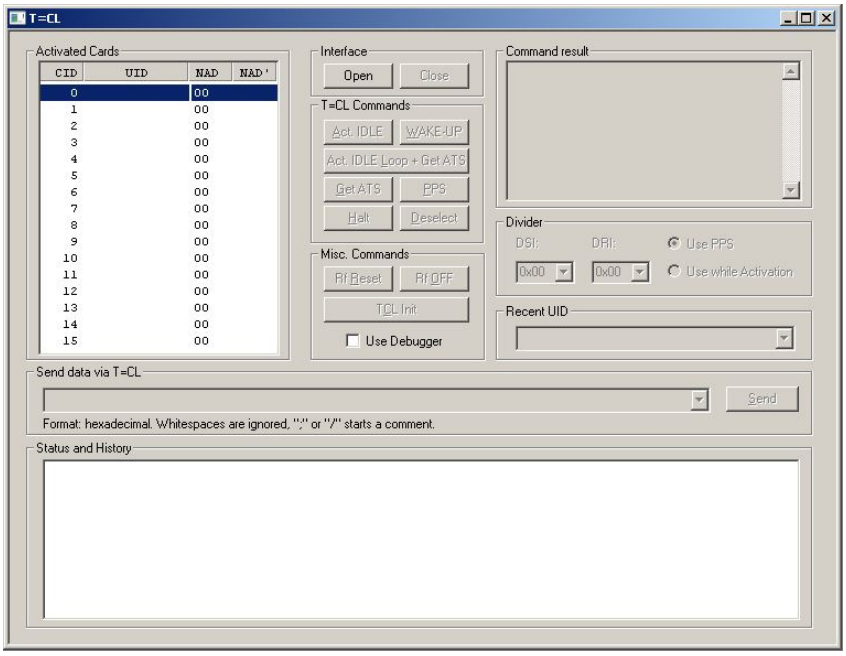


Abbildung 13: MifareWnd Eingabemaske für Part 4 spezifische Funktionen

# 4 Konzeption

## 4.1 Build System

Um ein gutes Maß an Pflegbarkeit der Software zu erreichen, wird ein Build System verwendet, welches unabhängig von der Entwicklungsumgebung ist. Ein mittlerweile auch in der Industrie weit verbreitetes und freies Werkzeug ist CMake[5] (Cross Make). Es bietet eine einheitliche Konfigurationsebene basierend auf einer Scriptsprache, ähnlich GNU Make[6]. Es bietet ausserdem die Möglichkeit, aus der vorhandenen Konfiguration automatisch die Projektdateien für alle gängigen Entwicklungsumgebungen und Plattformen zu erzeugen.

### 4.1.1 Konfiguration und Vorteile

CMake wird für einen Out-of-Source-Build konfiguriert. Das heißt, alle während des Build-Vorgangs generierten Dateien werden in einem gesonderten Verzeichnis untergebracht, welches völlig unabhängig von den Quelldateien ist.
Dies hat mehrere Vorteile:

1. Es ist einfacher, ein Verzeichnis, in welchem bereits kompiliert wurde, zu verschieben oder zu kopieren (z.B. für Backups oder Check-In in einem Versionskontrollsystem), da lediglich das Build-Verzeichnis als Ganzes bewegt werden muss. Es müssen so keine einzelnen Dateien verwaltet oder exkludiert werden.
2. Um die Datenmenge zu verringern, wird für ein Backup das Build-Verzeichnis gelöscht. So ist direkt sichergestellt, dass keine Änderungen an den Projektdateien der Entwicklungsumgebung undokumentiert weiterverwendet werden, da die Projektdateien ebenfalls im Build-Verzeichnis liegen und neu generiert werden müssen. Dies gilt natürlich auch für andere übriggebliebene generierte Dateien.
3. Als Entwickler kann man so mehrere Build-Versionen nebeneinander betreiben. Beispielsweise zum Testen von verschiedenen Konfigurationen der Software, oder sogar verschiedenen Compilern.

## 4.2 Grafische Oberfläche

Obwohl durch die firmeninterne Verwendung auf Windows Systemen eine Implementierung als Microsoft .NET-GUI-Applikation eine Option darstellt, wurde im Hinblick auf Portierbarkeit und Distributierbarkeit das Qt Framework gewählt.
So ist es prinzipiell nicht notwendig, zusätzliche Software Pakete zu installieren, da der GUI-Quellcode direkt zu nativem Code kompiliert wird. Die Laufzeitbibliotheken

[5] http://www.cmake.org/
[6] http://www.gnu.org/software/make/

können direkt mit der ausführbaren Datei distributiert werden.
Die Verwendung eines C++-Frameworks vereinfacht ausserdem die Entwicklung, da das gesamte Projekt mit einer einzigen Compiler-Toolchain verwaltet werden kann.
Für die Gestaltung der grafischen Oberfläche wird der vom Qt-Framework mitgelieferte QtDesigner verwendet. Der Qt-Designer ist ein grafischer Editor, der das Erstellen und Bearbeiten vo grafischen Qt-Elementen ermöglicht.
Das Qt Framework bietet weiter umfangreiche Möglichkeiten zur Anwendungsprogrammierung. Eigene Klassen für nicht GUI-spezifische Objekte wie Strings (*QString*), Threads (*QThread*) und Mutexe (*QMutex*, *QMutexLocker*) erleichtern die Entwicklung und Integration der Applikation.

Die Analyse der bestehenden Werkzeuge, speziell MifareWnd, gab bereits einen Eindruck einer sinnvollen Anordnung der grafischen Elemente. Da jedoch nicht der volle Funktionsumfang gefordert ist, wird die Darstellung vereinfacht.

Bestimmte Funktionen innerhalb der GUI sollen gleichzeitig verwendbar, also auf der selben Ebene angeordnet sein. Dazu gehören die Konnektivitätselemente, die Reader-Hardware spezifischen Elemente sowie der Zugriff auf Registerebene. Diese drei Gruppen werden im oberen Teil des Fensters angeordnet.
Die protokollspezifischen Funktionen werden hierarchisch darunter platziert. Diese werden in Tabulatorseiten aufgeteilt. Jeder dieser „Tabs“soll ausserdem eine eigene Liste für Tags erhalten, da sich die verschiedenen Standards doch recht stark unterscheiden, und es dadurch nicht unbedingt praktikabel ist, sie in einer Liste zusammenzufassen.
Darunter folgt das History-Feld mit einer Chronik der ausgeführten Befehle. Informationen wie Erfolg/Misserfolg einer Aktion und ausgewählte erhaltene Daten einer Kartenantwort sollen hier dargestellt werden.
Darunter liegt der Status-Balken, welcher genauere Informationen über die aufgetretenen Fehler gibt, wie im Abschnitt *Fehlerbehandlung* beschrieben wird. Die beschriebene Anordnung zeigt Abbildung 14. Die einzelnen Elemente werden im Abschnitt Implementierung konkretisiert.

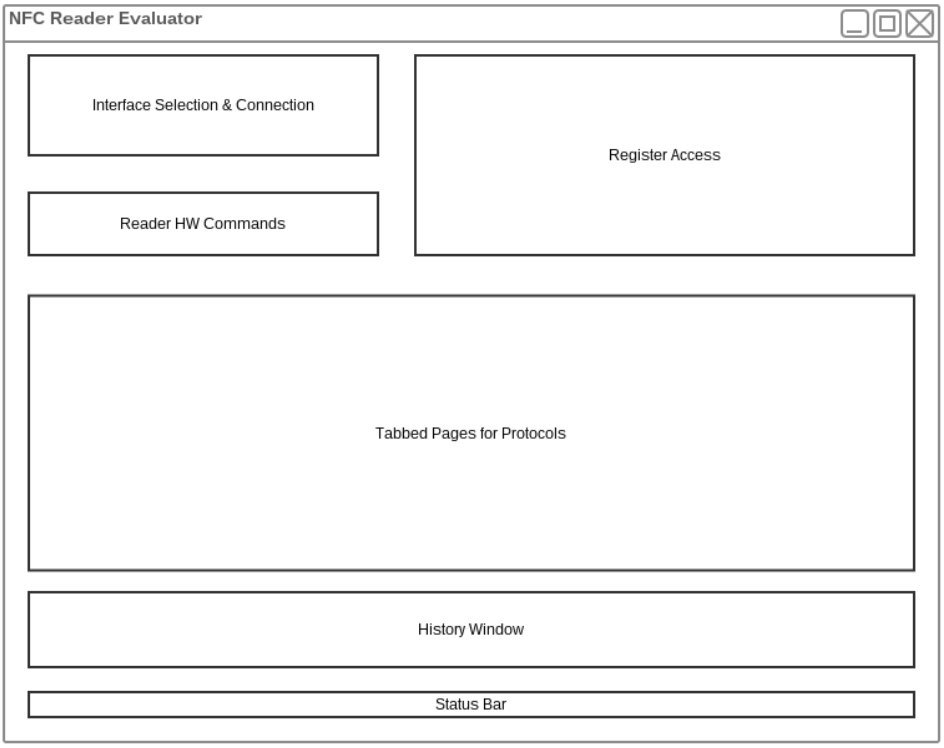


Abbildung 14: Konzeptionelle Darstellung des GUI

## 4.3 C++-Abstraktionsschicht auf Basis der NXP Reader Library

Wie bereits in Abschnitt 2.6 erklärt, bietet die NXP Reader Library ein umfangreiches Interface zur Programmierung einer Reader Applikation. Da im Rahmen des Verwendungzwecks als Testsystem vordefinierte Funktionsblöcke vorgesehen sind, bietet es sich an, die Funktionen der Reader Library in einer Zwischenschicht zusammenzufassen, um so unnötige Komplexität zu verbergen.
Ein solches Reader Framework soll sich intern um die Ressourcenverteilung kümmern, besonders die Initialisierungsbedingungen der Reader Library sollen vor dem Anwender versteckt werden. Die Reader Library arbeitet mit C-Datenstrukturen für jede der Softwarekomponenten, erfordert aber noch eine Menge zusätzlicher Variablen, welche von verschiedenen Funktionen für die Kommunikation benötigt werden.
Besonders die Protokoll-Abstraktionsschicht erfordert einige dieser Struktur-Variablen. Allein die Initialisierungen mittels bereitgestellter Funktionen für den hier benötigten Umfang belaufen sich auf knapp unter 100 Zeilen kommentierten und formatierten Quellcode. Diese Prozedur wird idealerweise direkt im Konstruktor der neuen Klasse untergebracht, so dass ein Nutzer sich darum nicht mehr kümmern muss.

### 4.3.1 Klassenstruktur

Eine naheliegende klassenbasierte Abstrahierung eines Readers ergibt sich hier durch die Schichtstruktur der Reader Library. Die Protokoll, Bus- und Applikationsschicht sind bereits unabhängig vom verwendeten IC-Typ. Somit werden diese Komponenten in einer abstrakten Reader-Klasse zusammengefasst. Diese soll durch eine Hardware-spezifische Klasse erweitert werden können. Diese Klasse soll die spezifischen Funktionen des IC implementieren, die Komponenten der Hardwareschicht beinhalten und außerdem eine Liste mit Registernamen und Adressen liefern.

Die abstrakte Reader-Klasse wird durch eine Klasse mit privaten, rein virtuellen Methoden für die hardwarespezifischen Funktionen repräsentiert. Dies ist in C++ die Definition eines Interfaces. Die Hardware-spezifische Klasse wird dann von der abstrakten Klasse abgeleitet.
In der Implementierung kann das Objekt dann über einen Zeiger auf die Basisklasse referenziert werden, somit ist egal, welche Klasse abgeleitet wurde, da sich alle zugängigen Methoden im public-Interface befinden. In Abbildung 15 ist in abstrakter Weise zu sehen, wie die Funktionalität aus der Reader Library verteilt ist.

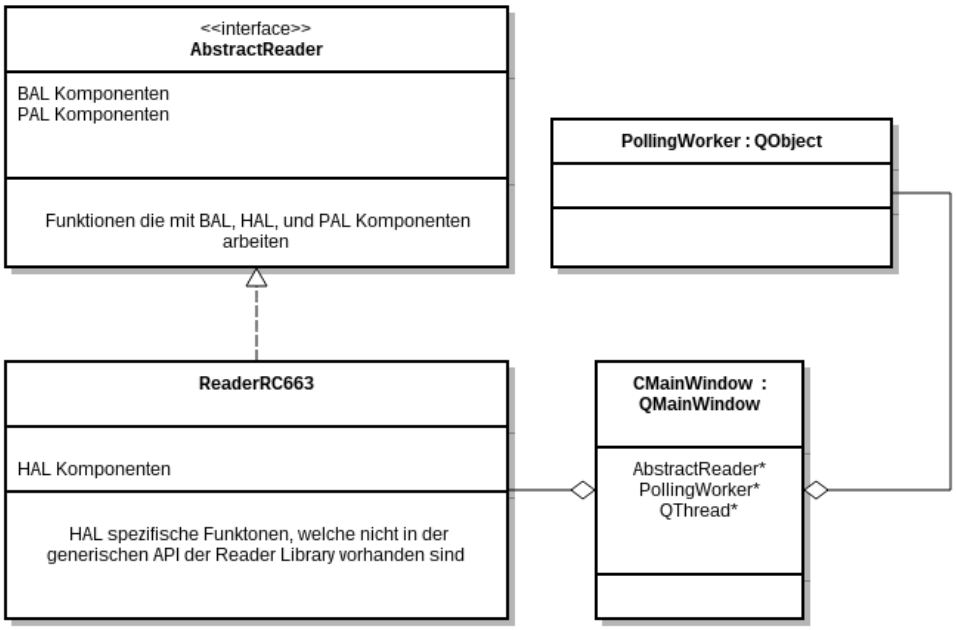


Abbildung 15: Abstraktes Klassendiagramm zur Konzeption

## 4.4 Fehlerbehandlung

Um dem Nutzer eine detaillierte Rückmeldung über den Status des Programms zu geben, wird eine Fehlerbehandlung benötigt. Die Reader Library besitzt bereits ein Error-Handling und Mitteilungsystem, welches kritische Informationen über die untersten Ebenen enthält.
Für den Nutzer der Zwischenschicht muss es allerdings ersichtlich sein, wo genau der Fehler im Reader Framework aufgetreten ist, was der Grund dafür ist, und wie er ihn beheben kann. Dafür wird eine Reihe von eigenen Fehlercodes definiert, und für jeden Code wird ein entsprechender Informations-String vorbereitet, der dem Nutzer mitteilt, bei welcher Aktion der Fehler aufgetreten ist, was die Ursache ist, und wie er gegebenenfalls zu beheben ist.
Da es der Software nicht möglich ist, die internen Zustände der Karte nachzuverfolgen und es ebenfalls nicht praktikabel ist, die Initialisierungsvorraussetzungen der Library zu verfolgen, sind etwaige Fehlerbehebungsvorschläge meist nicht genau zu bestimmen. Deshalb wird auf Exception Handling verzichtet, und ein eigenes Error Makro verwendet.
Eine Fehlermeldung soll folgendermaßen aufgebaut sein:

(Interner Fehlercode)...(Quelldatei)...(Zeile)...(Grund)...(Vorschlag)...(RdLibStatus)

Die Punkte kennzeichnen hierbei Bindewörter um den Text für den Nutzer direkt lesbar zu machen. RdLibStatus ist der interne Status Code der Reader Library, um anhand der Dokumentation die genaue Ursache und die betreffende Schicht der Library ermitteln zu können.

In der grafischen Benutzeroberfläche sollen die aufgetretenen Fehler durch ein, den Programmfluss, nicht unterbrechendes Mitteilungssystem dargestellt werden. Dazu wird der Status-Balken am unteren Rand der GUI dienen. Der vom Reader Framework gelieferte Fehlerstring wird dann direkt in diesem ausgegeben.

# 5 Implementierung

Dieser Abschnitt befasst sich mit der konkreten Beschreibung der implementierten Features im Reader Framework und in der GUI-Applikation.

## 5.1 Grafisches User-Interface

Das grafische Interface wird um eine eigene Klasse CMainWindow für das Hauptfenster herum entwickelt. Diese Klasse leitet sich direkt von der, von Qt5.4 bereitgestellten, QMainWindow-Klasse ab.
In dieser werden alle grafischen Elemente über den QtDesigner hinzugefügt. Ausserdem müssen für jedes der Elemente, die irgendeine Aktion ermöglichen sollen, die entsprechenden Funktionen implementiert werden.
In Qt gibt es dafür das Prinzip von Signals und Slots. Jedes Objekt in Qt besitzt vordefinierte Signale und Slots. Signale werden von Objekten ausgesendet und von Slots angenommen. Signale sind auch in der Lage Parameter zuübermitteln.

Beispiel: Ein einfaches grafisches Objekt, mit welchem man interagieren kann, ist ein Button (*QPushButton*). Wird er angeklickt, emittiert er das Signal clicked(). Dieses Signal muss jetzt von einem anderen Objekt angenommen werden, um einen Effekt zu erzielen. Dazu muss mittels eines connect()-Makros das entsprechende Signal des Objektes mit dem gewünschten Slot verbunden werden. Ein Slot ist dabei eine einfache Funktion, die eine definierte Aktion ausführt.

Abbildung 16 stellt das Konzept laut der offiziellen Qt-Dokumentation dar.[siehe Ltd. 2015b]

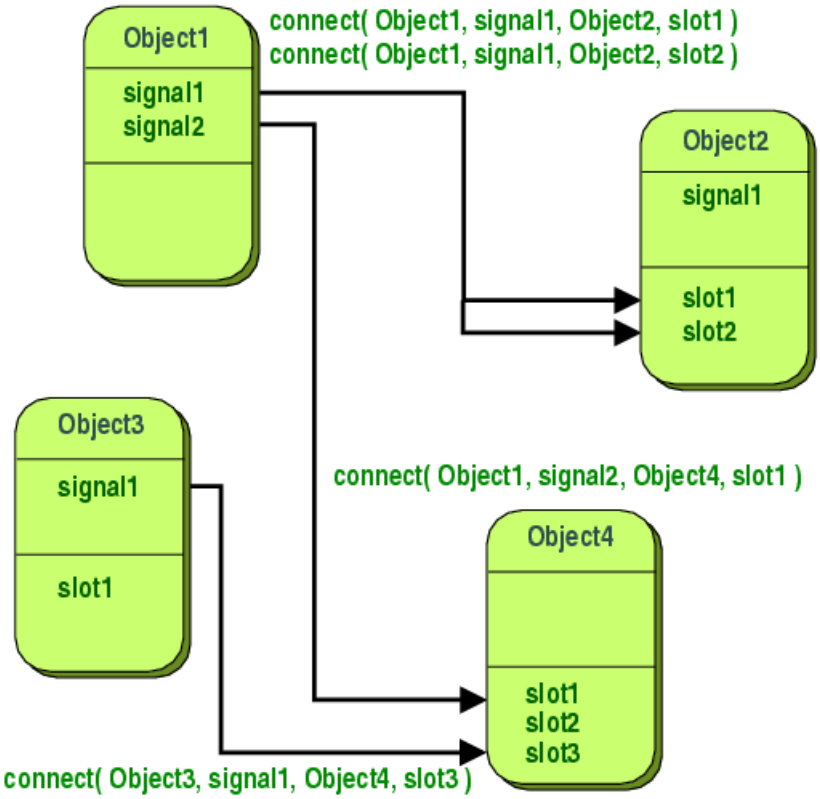


Abbildung 16: Abstrakte Darstellung des Signal- und Slot-Konzeptes in Qt [aus Ltd. 2015b]

In der Test-Applikation stoßen die GUI-Elemente einen zugewiesenen Slot der CMainWindow-Instanz an. Wird beispielsweise der Connect-Button geklickt, wird die Funktion slotButtonConnectClicked() aufgerufen, welche dann die Inhalte der Drop-Down Listen auswertet, und dementsprechend ein Reader-Objekt initialisiert. Daraufhin wird direkt versucht, die Verbindung aufzubauen.

Nachdem bereits im Abschnitt Anforderungsanalyse die geforderte Funktionalität festgehalten, und im Abschnitt Konzeption bereits das grobe Layout beschrieben wurde, wird hier jetzt die konkrete Gestaltung der einzelnen Elemente gezeigt.
Abbildung 17 zeigt den aktuellen Stand der grafischen Oberfläche mit den geforderten ISO/IEC 14443 Funktionen ohne Typ B. Die anderen Tabs besitzen lediglich eine simple Polling-Funktion für ein einziges Tag im Feld, da dies ohne viel Aufwand zu implementieren war. In Abbildung 18 ist dazu der Tab für die Low Power Card Detection zu sehen.

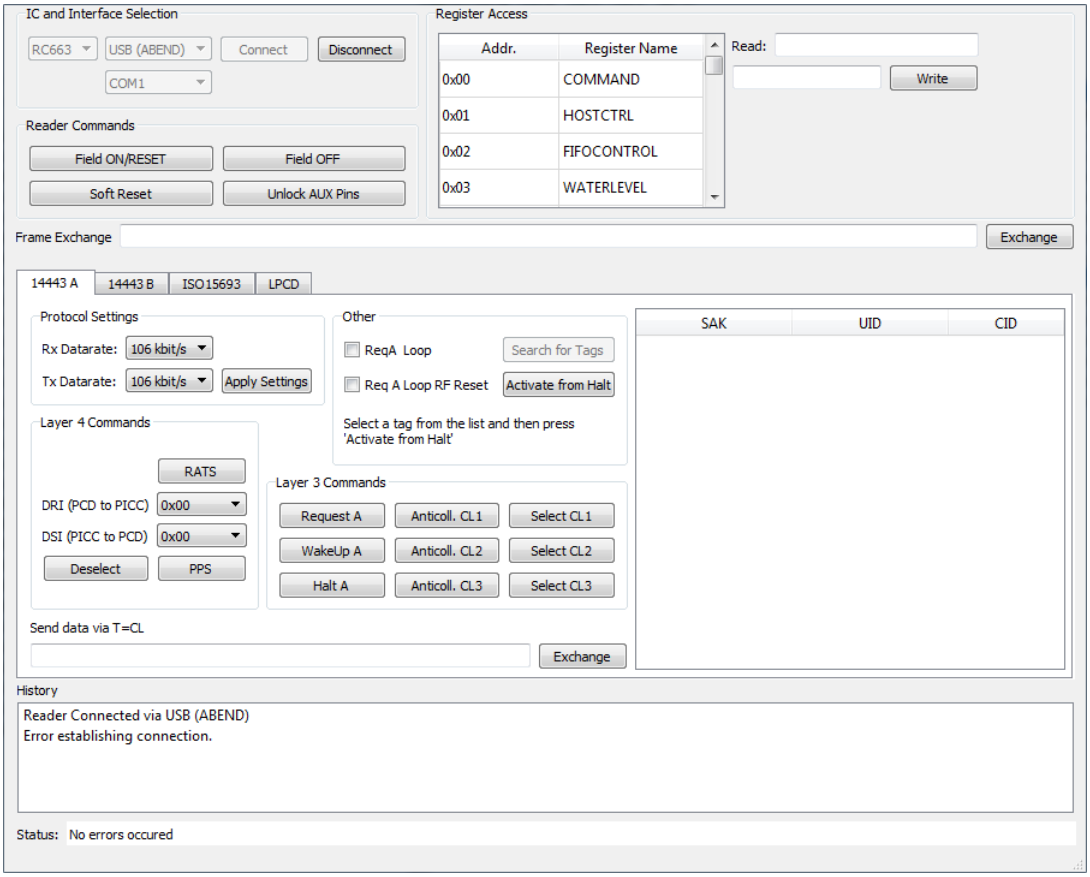


Abbildung 17: Gestaltung des implementierten GUI

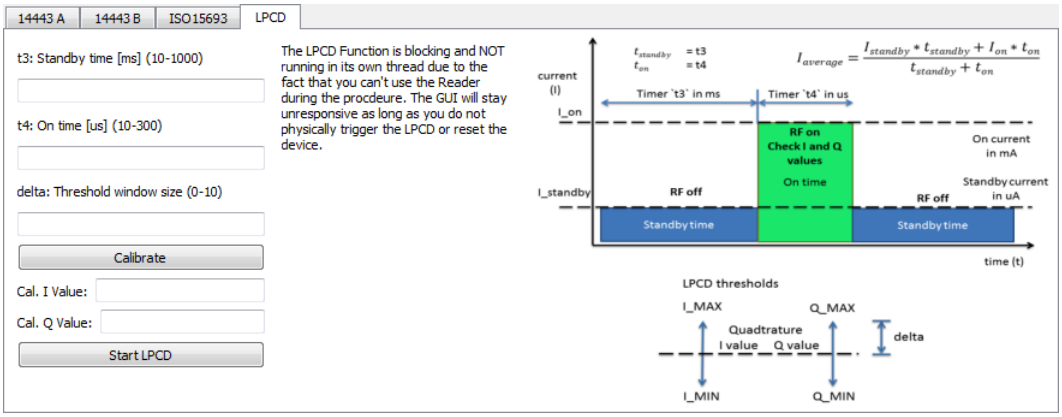


Abbildung 18: Ausschnitt des LPCD Tabs

Abbildung 19 zeigt das Klassendiagramm der Hauptfenster-Klasse. Die benötigten Slots sind namentlich den GUI-Elementen angepasst. Außerdem gibt es noch einige intern genutzte Funktionen, welche in mehreren Slot-Funktionen benötigt werden.

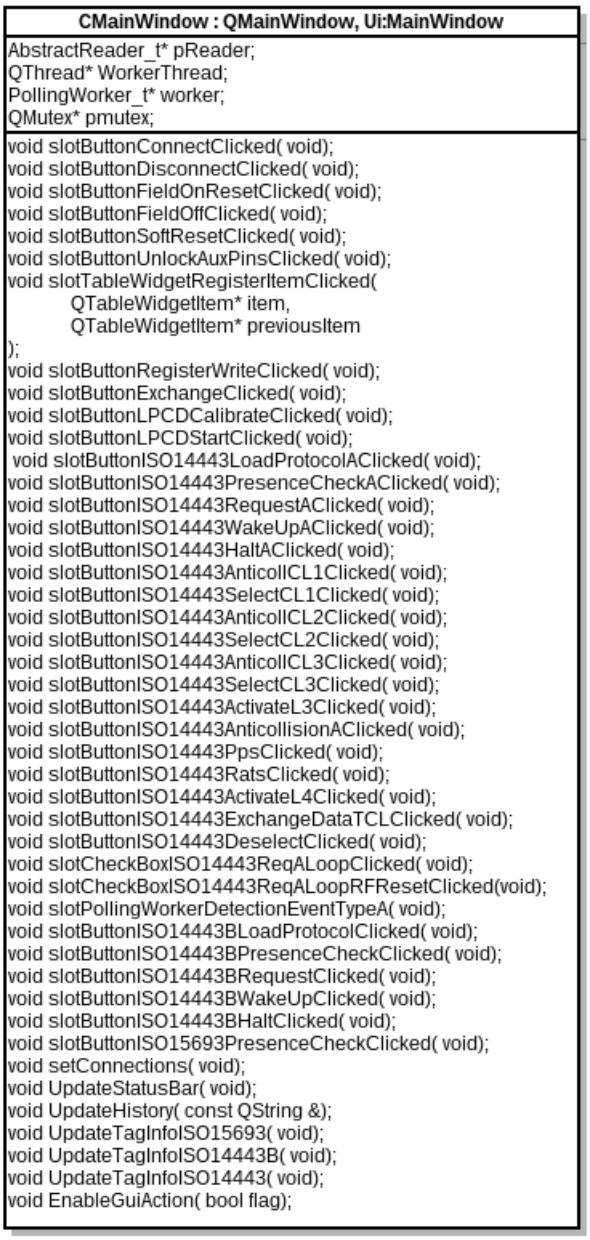


Abbildung 19: Klassendiagramm von CMainWindow

## 5.2 Die Klasse AbstractReader

In dieser Klasse kann, Dank der Struktur der Reader Library, bereits die meiste Funktionalität zusammengefasst werden. Es gibt jedoch auch Hardware-spezifische Funktionen, welche keine gemeinsame API besitzen. Beispiele hierfür sind das Laden bestimmter Protokollfunktionen im Reader IC, und der Low Power Card Detection Modus. Während der RC663 dafür ein Kommando in Hardware implementiert hat, besitzt der Reader IC PN512 dieses nicht.
Im Interface AbstractReader sind diese Funktionen dennoch Teil der öffentlichen Schnittstelle. Diese rufen eine rein virtuelle Funktion auf.
Diese Funktion muss in der ableitenden Klasse als normal-virtuelle Funktion implementiert werden. Unterstützt ein Reader bestimmte Funktionen nicht, so soll ein Unsupported-Error zurückgegeben werden.

Abbildung 20 veranschaulicht die Aufrufhierarchie vom Nutzer über die Reader Framework-Klassen bis zur Reader Library.

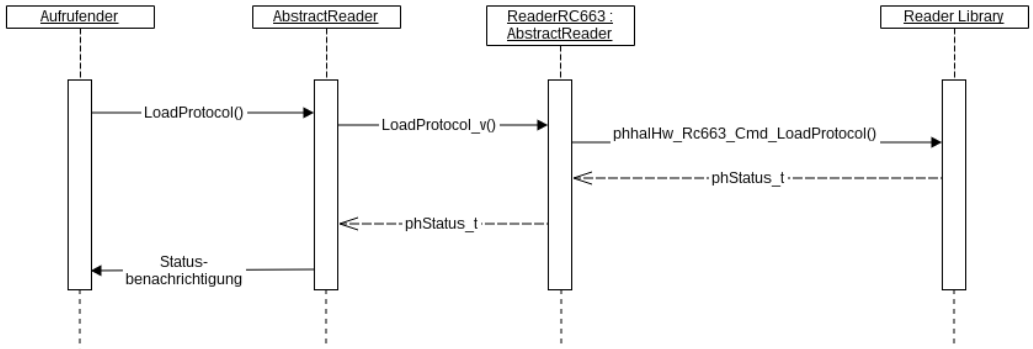


Abbildung 20: Aufrufhierarchie im Reader Framework

Das Klassendiagramm zeigt die für diese Arbeit relevante API:

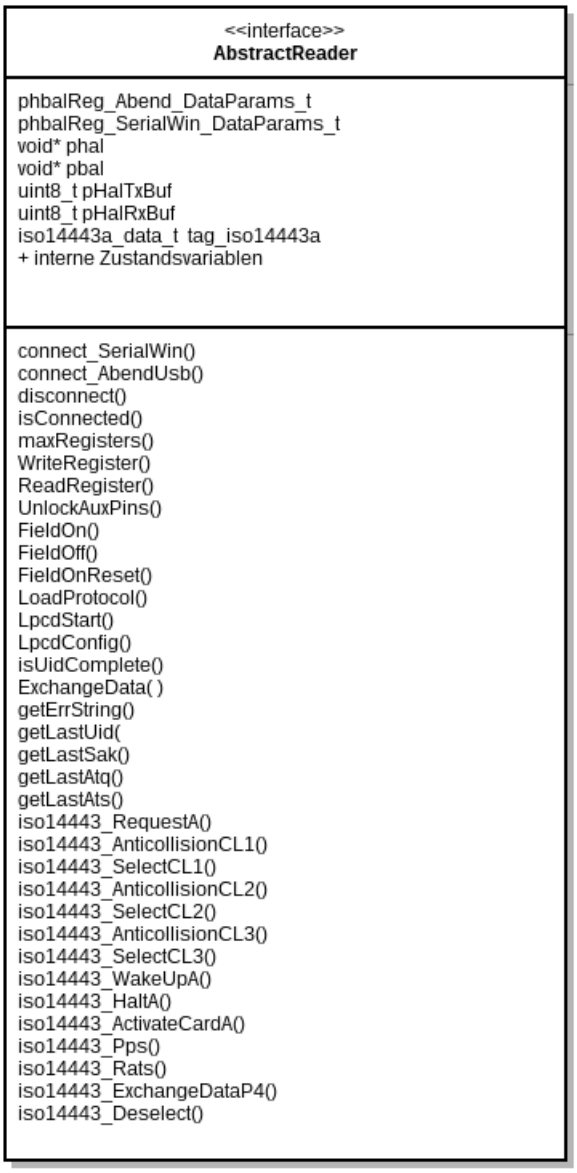


Abbildung 21: Klassendiagramm der Klasse AbstractReader

## 5.3 Die Klasse ReaderRC663

Diese Klasse repräsentiert die spezifische Hardware, den IC Typ. Sie leitet sich von der Klasse AbstractReader ab, und erbt so alle hardware-übergreifenden Methoden dieser. Um die nicht Hardware-spezifischen Methoden abzudecken, muss sie die, von der Vater Klasse vorgegebenen, rein virtuellen Methoden implementieren.

Sie enthält außerdem die hardware-spezifischen Bestandteile der Reader Library, wie die Hardware-Abstraction-Layer-Strukturvariable und die Tx und Rx Buffer. Weiter enthält sie die maximale Anzahl der Register als Konstante, welche in der Basisklasse über die Methode unsigned int maxRegisters(void) abgerufen werden kann. Dies wird für die Initialisierung der Register-Liste benötigt.
Der Basisklasse müssen die Referenzen auf die HAL-Bestandteile mitgeteilt werden, da diese in verschiedenen Initialiserungfunktionen benötigt werden. Dafür ist die Funktion *void setHalRef(void)* vorgesehen, welche direkt im Konstruktor aufgerufen werden kann, und somit vom Nutzer des Frameworks nicht mehr berücksichtigt werden muss. Abbildung 22 zeigt das Klassendiagramm. Es sind noch nicht alle Hardware-spezifischen Funktionen, welche die Reader Library bietet, implementiert. Dies ist in den aktuellen Anforderungen nicht vorgesehen.
Im Fall des RC663 ist die Implementierung der Funktion *LoadProtocol_v()* sehr einfach, da dieser Befehl bereits im IC selbst unterstützt wird. *UnlockAuxPins_v()* erfordert auch lediglich eine definierte Abfolge von Register Reads/Writes.

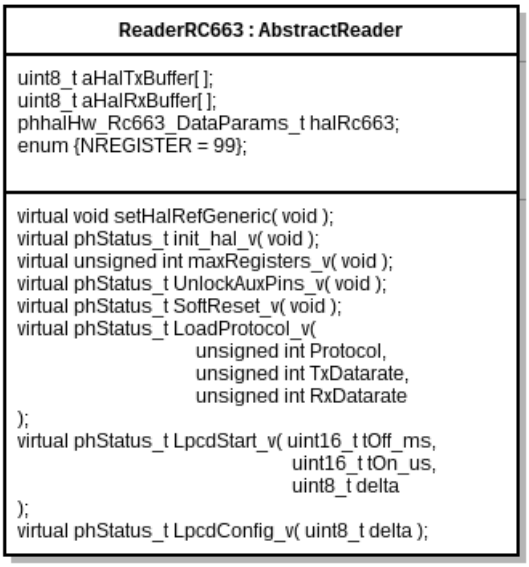


Abbildung 22: Einfaches Klassendiagramm von ReaderRC663

Die Implementierung zusätzlicher Reader IC Klassen wird also denkbar einfach, da die Komplexität dieser konkreten Reader-Klassen relativ gering ist.

## 5.4 Fehlerbehandlung

Die Fehlercodes werden als klasseninterne Konstanten des Typs *enum* implementiert. Sie indizieren ein *std::string* Array mit den Fehlernachrichten und Lösungsvorschlägen an der entsprechenden Position.
Ein Makro wird dazu verwendet, auf Fehler zu prüfen, um dann gegebenenfalls den Fehlerstring zusammenzusetzen und aus der aktuellen Funktion direkt zurückzukehren.
Das Framework bietet die Funktion *std::string getErrString()* um dem Aufrufenden die, im zuvor definierten Muster, zusammengesetzte Fehlerbeschreibung zurückzuliefern.
Abbildung 23 zeigt die generierte Fehlermeldung im Statusbalken(unten) im Falle einer provozierten Kollision beim Ausführen eines REQA-Kommandos.

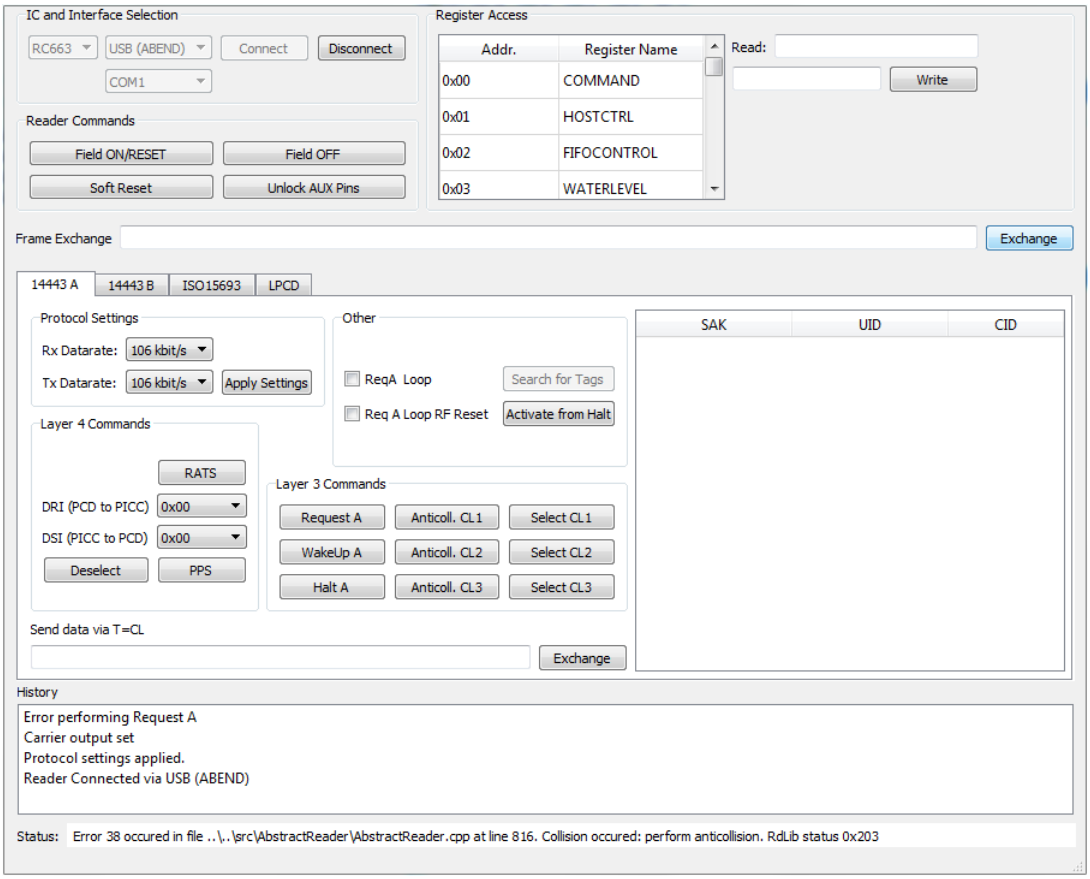


Abbildung 23: Provozierte Fehlerausgabe im Falle einer detektierten Kollision

## 5.5 Polling Loop

Für die Realisierung der Polling Loop ist eine Implementierung als Thread mit Mutex notwendig, da das Ausführen der Request A Methode den internen Zustand des Reader Frameworks ändert. Hierzu werden die Qt-Klassen *QThread*, *QMutex* und *QMutexLocker* verwendet. Die Funktionalität selbst wird in einer eigenen Klasse „PollingWorker“gekapselt um bei Bedarf instanziiert und in einen Thread verschoben zu werden. Dies ist die offiziell empfohlene Methode zur Nutzung von QThread. [siehe Ltd. 2015a]

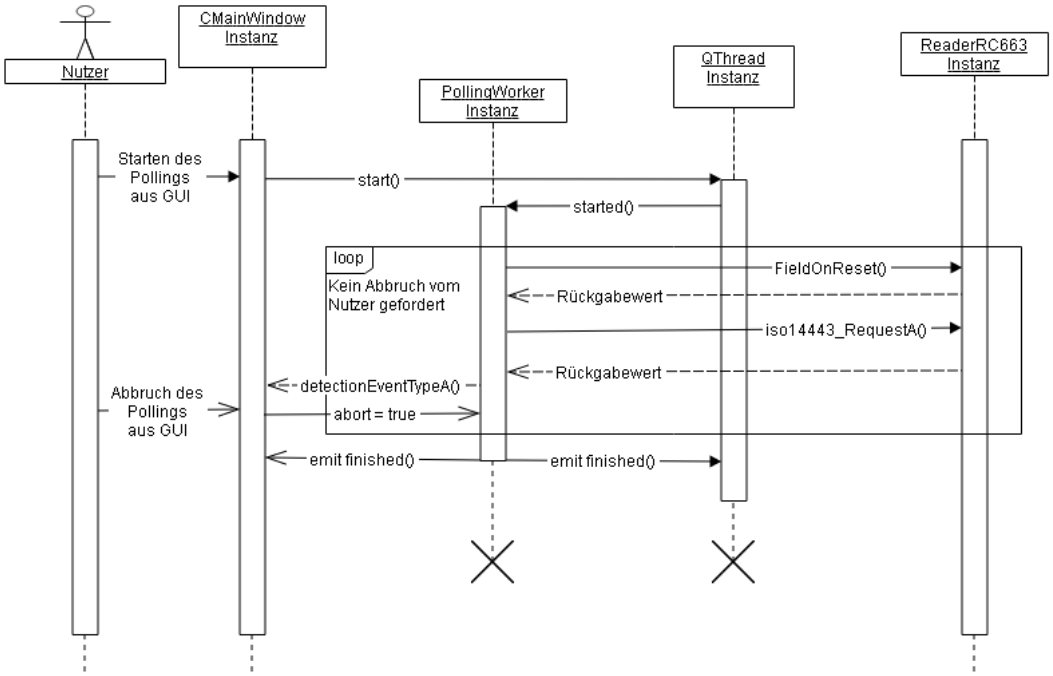


Abbildung 24: Sequenzdiagramm der Polling-Funktionalität

Anmerkung zu Abbildung 24: Die Klasse *PollingWorker* bietet optionale Parameter, um zwischen jedem RequestA-Befehl einen Field-Reset durchzuführen oder bei einer Erfolgreichen Detektion den Thread zu beenden. Ist kein Field-Reset gewünscht, entfällt entsprechend auch diese Aktion aus der Schleife.
In der Implementierung läuft die Worker-Routine in einer Endlosschleife, bis vom Nutzer das *abort*-Flag gesetzt wird. Dies passiert durch Entfernen des Häkchens in der entsprechenden Checkbox.

## 5.6 ISO/IEC 14443 Anticollision

Die Reader Library bietet zwar eine Funktion zur Anticollision, diese setzt jedoch lediglich den Befehl zusammen und übermittelt ihn im korrekten Frame-Format. Die eigentliche Antikollisionsprozedur muss selbst implementiert werden.
Da in der Benutzeroberfläche jeweils ein Button für Anticollision und Select in jedem der drei Cascade-Level zur Verfügung stehen soll, wird die Sequenz dementsprechend in sechs parameterlose Funktionen aufgeteilt. Folgender Algorithmus wurde frei aus der Norm übersetzt[siehe ISO/IEC 2011, Abschnitt 6.5.3.1].

Erklärung der Abkürzungen:

**SEL** Typ A Select Code Byte, zeigt das Cascade-Level an.

**NVB** Number of Valid Bits, kodiert als Byte. Enthält die Anzahl der gültigen Bits der UID die das PCD empfangen hat. Das obere Nibble gibt die Anzahl der gültigen Bytes an, das untere Nibble die Anzahl der gültigen Bits. Die gesamte Bitanzahl ergibt sich also nach $N_{bit}$ = (NVB & 0xF0)*8 + (NVB & 0x0F)

Beschreibung des Algorithmus:

1. Das PCD soll SEL den Wert für das gewählte Cascade-Level zuweisen.
2. Das PCD soll NVB den Wert 0x20 zuweisen. Dieser Wert legt fest, dass das PCD keinen Teil einer UID übermittelt.
3. Das PCD soll SEL und NVB in dieser Reihenfolge übermitteln.
4. Alle PICCs im Feld sollen mit ihrer vollständigen UID antworten.
5. Wenn mehr als eine PICC antwortet, kann eine Kollision auftreten. In diesem Fall werden die Schritte 6 bis 10 übersprungen.
6. Das PCD soll die Position der ersten Kollision erkennen.
7. Das PCD soll NVB als Wert die Anzahl der gültigen empfangenen Bits vor der Kollision zuweisen und entweder eine binäre '0' oder '1' anhängen. Dies ist frei wählbar durch das PCD, '1' ist jedoch typisch. Das hat zur Folge, dass sich die '1' „durchsetzt“.
8. Das PCD soll SEL, NVB und die gültigen Bits übermitteln (in dieser Reihenfolge).
9. Nur PICCs, deren erste Teile der UID den eben empfangenen entspricht, sollen die restlichen Bits ihrer UID senden.
10. Treten weitere Kollisionen auf, sollen die Schritte 6 bis 9 wiederholt werden (bis zu einer maximalen Anzahl von 32 Durchgängen).

11. Treten keine weiteren Kollisionen auf, soll das PCD NVB den Wert 0x70 zuweisen. Dieser Wert legt fest, dass das PCD die gesamte UID übermittelt.
12. Das PCD soll SEL, NVB und alle 40 Bits der UID übermitteln, gefolgt von einer Typ A CRC.
13. Die PICCs, deren UID mit den 40, vom PCD gesendeten, Bits übereinstimmen, sollen mit ihrem SAK antworten. Anm.: Prinzipiell sollte eine UID einzigartig sein, es gibt aber durchaus Hersteller, welche sich nicht an die Vorgabe der Namensräume halten, z.B. bei Kopien.
14. Wenn die UID vollständig ist, soll die PICC ihr SAK mit gelöschtem Cascade-Bit senden und vom READY/READY* in den ACTIVE/ACTIVE* Zustand übergehen.
15. Das PCD soll das Cascade-Bit der SAK auswerten um zu entscheiden, ob weitere Antikollisionsdurchgänge mit erhöhtem Cascade-Level erforderlich sind.

Soweit keine Kollisionen auftreten, können die Schritte 2 bis 10 übersprungen werden.

Abbildung 25 Zeigt den zuvor beschriebenen Vorgang in einem Ablaufdiagramm, wie im Standard festgelegt[siehe ISO/IEC 2011, Abschnitt 6.5.3.1].

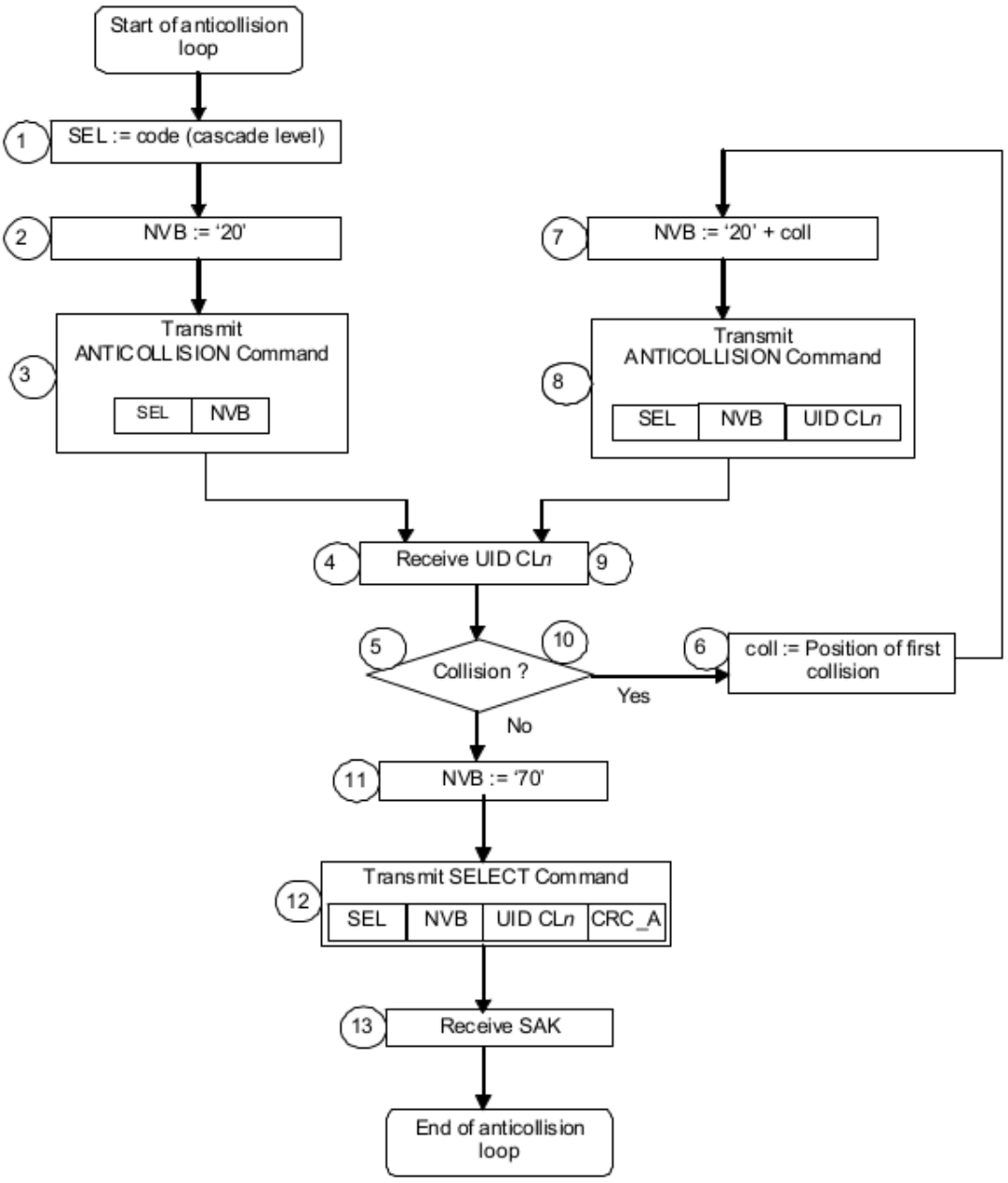


Abbildung 25: Algorithmus zur Auflösung von Kollisionen im Feld für ein Cascade-Level[siehe ISO/IEC 2011, Abschnitt 6.5.3.1]

Folgendes Listing zeigt die Implementierung der Anticollision und Select Befehle mit Cascade-Level 1 exemplarisch. Für Cascade-Level 2 ist wiederum ist eine Anpassung notwendig. Die Implementierung ist dem Quellcode im Anhang zu entnehmen[7].

```
1  phStatus_t AbstractReader::iso14443_AnticollisionCL1(void)
       {
2      this->clearError();
3      tag_iso14443a.bNvbUid = 0x00;
4      tag_iso14443a.bUidIncompleteLen = 0;
5      tag_iso14443a.isUidComplete = false;
6      while (tag_iso14443a.bNvbUid != 0x40) {
7          status = phpalI14443p3a_Anticollision  (
8                   &tag_iso14443a.palI14443p3a,
9                   PHPAL_I14443P3A_CASCADE_LEVEL_1,
10                  &tag_iso14443a.bUidIncomplete[0],
11                  tag_iso14443a.bNvbUid,
12                  &tag_iso14443a.bUidIncomplete[0],
13                  &tag_iso14443a.bNvbUid
14               );
15         // if a collision was detected
16         if ((status & PH_ERR_MASK) == PH_ERR_COLLISION_ERROR)
               {
17             isCollisionDetected = true;
18             tag_iso14443a.bMoreCardsAvailable = 1;
19             // if number of valid bits < 7
20             if ((tag_iso14443a.bNvbUid & 0x07) < 7) {
21                 // increment bitcount (lower nibble)
22                 tag_iso14443a.bNvbUid++;
23             } else {
24                 // increment bytecount (upper nibble)
25                 tag_iso14443a.bNvbUid = (uint8_t)((((
                       tag_iso14443a.bNvbUid & 0xF0) >> 4) + 1) <<
                       4);
26             }
27             // else just copy the partial UID
28         } else if (status == PH_ERR_SUCCESS) {
29             // copy the partial UID
30             if (tag_iso14443a.bNvbUid != 0x40) {
31                 // cascade level 1, last 3 bytes are valid
32                 memcpy(tag_iso14443a.bUid, &tag_iso14443a.
                       bUidIncomplete[1], 3);/
33                 tag_iso14443a.bUidIncompleteLen += 3;
```

[7] Dateipfad: testing/src/AbstractReader/AbstractReader.cpp

```
34          } else {
35             // else 4 bytes are valid
36             memcpy(&tag_iso14443a.bUid[tag_iso14443a.
                  bUidIncompleteLen], tag_iso14443a.
                  bUidIncomplete,4);
37             tag_iso14443a.bUidIncompleteLen += 4;
38          }
39          tag_iso14443a.bLength = tag_iso14443a.
               bUidIncompleteLen;
40       } else {
41          // some error that can't be handled
42          ON_ERROR_RETURN(status, ERR_ANTICOLL_CMD);
43       }
44    }
45    if ( (status & PH_ERR_MASK) == PH_ERR_COLLISION_ERROR) {
46       return PH_ERR_SUCCESS;
47    }
48    return status;
49 }
50
51 phStatus_t AbstractReader::iso14443_SelectCL1(void) {
52    this->clearError();
53    status = phpalI14443p3a_Select(&tag_iso14443a.
         palI14443p3a,
54             PHPAL_I14443P3A_CASCADE_LEVEL_1,  // cascade
                  level 1 code
55             &tag_iso14443a.bUidIncomplete[0],
56             &tag_iso14443a.bSak[0]
57          );
58    ON_ERROR_RETURN(status, ERR_SELECT_CMD);
59    // if a UID has been resolved
60    if (!(tag_iso14443a.bSak[0] & 0x04)) {
61       tag_iso14443a.isUidComplete = true;
62    }
63    //copy it to 'gettable' buffer
64    memcpy(tag_iso14443a.bUid, tag_iso14443a.bUidIncomplete,
          tag_iso14443a.bUidIncompleteLen);
65    tag_iso14443a.bLength = tag_iso14443a.bUidIncompleteLen;
66
67    return status;
68 }
```

## 5.7 Detektieren von PICCs

Das Detektieren von PICCs baut auf der Antikollision auf. Der in Abbildung 26 gezeigte Ablauf muss manuell abgearbeitet werden, damit eine erkannte PICC in der Liste erscheint. Dann kann der Nutzer wählen, ob er die PICC in den HALT -Zustand versetzen, oder mit der Kommunikation fortfahren möchte. Sollen mehrere PICCs erkannt werden, muss die erkannte PICC in den HALT-Zustand versetzt, und die Prozedur mehrmals vollzogen werden.

Die Verwaltung der PICCs muss die GUI übernehmen, da der Reader dies nicht unterstützt. Das Reader Framework bietet nach der selben Logik deswegen ebenfalls keine Speicherung von PICC-Daten (UID mit zugehöriger SAK). Zum Speichern der PICC-Daten wird direkt das QListWidget-Objekt verwendet, da es praktischerweise das Speichermanagement für die übergebenen Einträge übernimmt.
Abbildung 26 zeigt die Sequenz mitsamt der vom Nutzer geforderten Interaktion mit der Software.

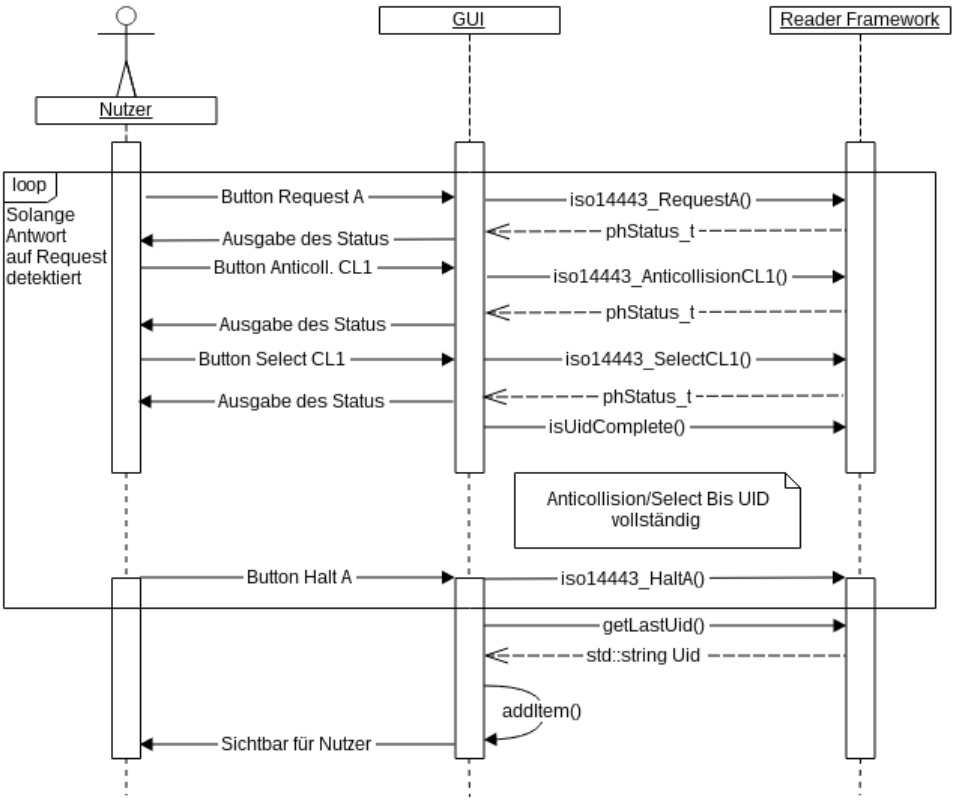


Abbildung 26: Sequenzdiagramm zum Detektieren aller Typ A PICCs im Feld

## 5.8 Initialisieren des Übertragungsprotokolls

Ausgehend von der vorigen Sektion wird der Vorgang zur Initialisierung des Übertragungsprotokolls beschrieben. Zur Aktivierung von ISO/IEC 14443-4 muss zuerst ein RATS and eine, nach ISO/IEC 14443-3 aktivierte, PICC gesendet werden. Aus der Antwort ergibt sich dann, ob der PPS Befehl unterstüzt wird. Die ISO/IEC 14443-4 Typ A Befehle RATS sowie PPS sind in der Reader Library vorhanden. Die RATS Funktion der Reader Library ist wie folgt aufgebaut:

```
1 phStatus_t phpalI14443p4a_Sw_Rats(
2                 phpalI14443p4a_Sw_DataParams_t *
                     pDataParams,
3                 uint8_t bFsdi,
4                 uint8_t bCid,
5                 uint8_t * pAts
6             );
```

Da Parameter wie die Strukturvariable der Schicht und die FSD aber sowieso feststehen, und vom Nutzer nicht angefasst werden, können diese intern im Reader Framework gehandhabt werden. Der Zeiger für den ATS-Buffer wird ebenfalls nicht benötigt, da eine Methode getLastAts() in der Reader Framework API zur Verfügung steht.

Davor ist allerdings noch ein Aufruf von folgender Funktion notwendig, um die vordefinierten Werte für *bFsdi* zu ermitteln. *bFsdi* enthält die FSDI - maximale Frame size des PCD als kodiertes Integer.

```
1 phStatus_t phpalI14443p4a_Sw_GetProtocolParams(
2                 phpalI14443p4a_Sw_DataParams_t *
                     pDataParams,
3                 uint8_t * pCidEnabled,
4                 uint8_t * pCid,
5                 uint8_t * pNadSupported,
6                 uint8_t * pFwi,
7                 uint8_t * pFsdi,
8                 uint8_t * pFsci
9              );
```

Nach senden des RATS müssen die, durch die ATS gewonnenen, restlichen Parameter (z.B. FSCI: maximale Frame Size der PICC) erneut mit *phpalI14443p4a_Sw_GetProtocolParams()* ausgelesen, und in die Struktur der ISO/IEC 14443-4 Schicht übernommen werden:

```
1 phStatus_t phpalI14443p4_Sw_SetProtocol(
```

```
2                   phpalI14443p4_Sw_DataParams_t * pDataParams
                       ,
3                   uint8_t   bCidEnable,
4                   uint8_t   bCid,
5                   uint8_t   bNadEnable,
6                   uint8_t   bNad,
7                   uint8_t   bFwi,
8                   uint8_t   bFsdi,
9                   uint8_t   bFsci
10              );
```

Der zusätzlich notwendige Aufruf dieser Funktionen kann ebenfalls gekapselt werden, da alle Variablen intern gehandhabt werden. Da die Anforderung keine vollständige CID-Unterstützung vorsieht, wird sie in der Implementierung gleich Null gesetzt. Die Wrapper-Methode sieht demnach so aus:

```
1 phStatus_t iso14443_Rats(uint8_t bCid);
```

Beim PPS-Kommando macht sich die Vereinfachung nicht so stark bemerkbar. Hier entfällt lediglich die Strukturvariable, da DRI und DSI parametriert werden sollen.

```
1 phStatus_t phpalI14443p4a_Sw_Pps(
2                   phpalI14443p4a_Sw_DataParams_t *
                       pDataParams,
3                   uint8_t bDri,
4                   uint8_t bDsi
5               );
```

Vereinfacht sich zu:

```
1 phStatus_t iso14443_Pps(uint8_t dri, uint8_t dsi);
```

dri und dsi werden in der GUI ausgewählt. Mit den so definierten Funktionen kann dann mit folgender Sequenz das Übertragungsprotokoll aktiviert werden:

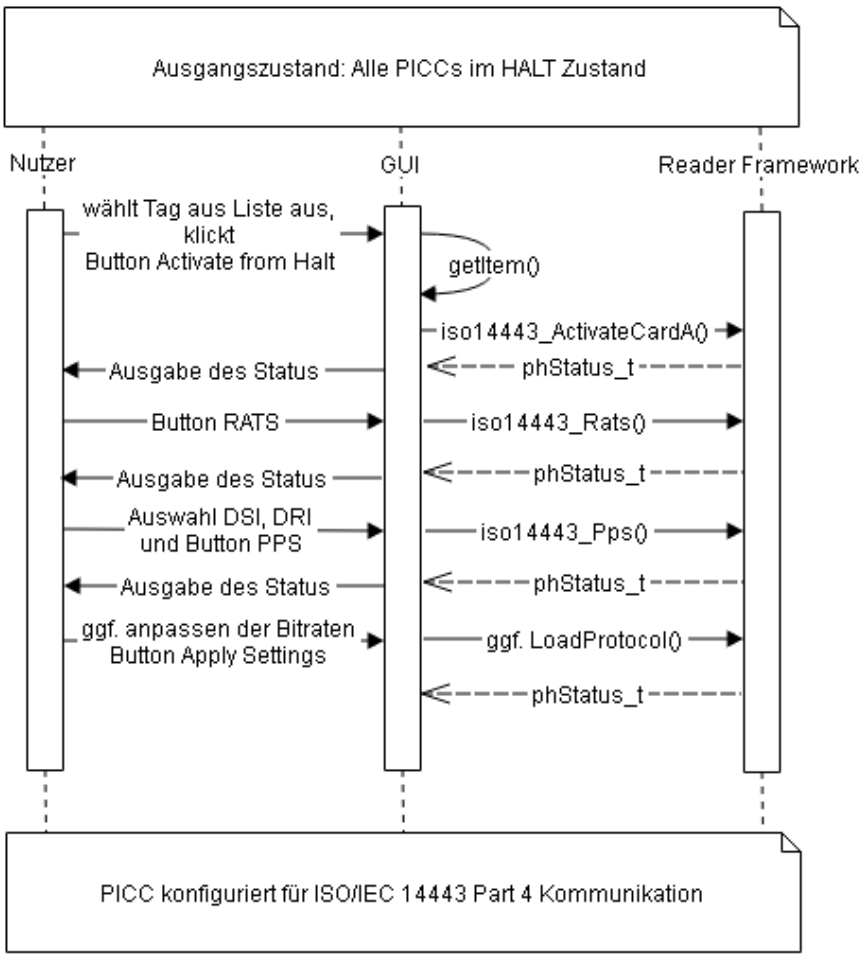


Abbildung 27: Sequenzdiagramm zur Initialisierung des Übertragungsprotkolls

## 5.9 Datenaustausch über das Übertragungsprotokoll

Nachdem das Übertragungsprotokoll initialisiert wurde, können Daten ausgetauscht werden. Auch hier wird wieder das Handling der Strukturvariablen und der Buffer in das Reader Framework verlagert. Durch die Option PH_EXCHANGE_DEFAULT wird zur Zeit kein Paket-Chaining unterstützt. Zur komfortablen benutzung der Funktion werden die zu sendenden Daten als Hex-String übergeben, und die empfangene Antwort wird ebenfalls in einen Hex-String

```
1 std::string AbstractReader::iso14443_ExchangeDataP4(std::
    string TxData) {
2     std::stringstream ss;
```

```
3     unsigned int offset = 0, i = 0;
4     // cancel on odd data length -> leading zeros are
          required
5     if (TxData.length() % 2 != 0) {
6        return "";
7     }
8     // set uid length
9     lenHalTxBuf = TxData.length() / 2;
10    // convert to byte array
11     while (offset < TxData.length()) {
12         unsigned int buffer;
13         ss << std::hex << TxData.substr(offset, 2);
14         ss >> std::hex >> buffer;
15         pHalTxBuf[i] = static_cast<unsigned char>(buffer);
16         offset += 2;
17         i++;
18         // empty the stringstream
19         ss.str(std::string());
20         ss.clear();
21     }
22    // perform data exchange
23    status = phpalI14443p4_Exchange(
24                          &tag_iso14443a.palI14443p4,
25                          PH_EXCHANGE_DEFAULT,
26                          pHalTxBuf,
27                          lenHalTxBuf,
28                          &pHalRxBuf,
29                          &lenHalRxBuf);
30    if (status != PH_ERR_SUCCESS) {
31       ON_ERROR_RETURN_STR(status, ERR_EXCHANGE_TCL);
32       return "";
33    }
34     //forge hex string out of answer
35     ss << std::hex << std::setfill('0') << std::right <<
           std::uppercase;
36    for(int i = 0; i < lenHalRxBuf ; i++) {
37       ss << std::setw(2) << (int) pHalRxBuf[i];
38    }
39     return ss.str();
40 }
```

# 6 Verifikation

Um die Implementierung auf Korrektheit in Funktion und Umsetzung der Vorgaben zu prüfen, werden eine Reihe von Testabläufen definiert.
Alle Testabläufe werden folgendermaßen beschrieben:

- Beschreibung der erwarteten Funktionalität und des Ergebnis
- Ablaufplan und Bedingungen um das gewünschte Ergebnis zu erreichen
- Beschreibung des erhaltenen Ergebnisses

## 6.1 Tests

Folgende Tests werden durchgeführt:

### 6.1.1 Initialisierung des Reader Objekts und Konnektivität via USB

Prüfen des Verbindungsaufbaus. Es kann allerdings nur geprüft werden, ob die Verbindung zur Bus-Schicht der Reader Library besteht, d.h ob die Applikation mit der Firmware des LPC1769 kommunizieren kann. Dies stellt aber noch nicht sicher, dass der Reader korrekt angesteuert werden kann.

Zu Beginn sind alle GUI-Elemente, welche mit dem Reader Objekt arbeiten, deaktiviert um eine Nullzeiger Ausnahme zu verhindern. In vorbereiteten Drop-Down Listen ist der IC-Typ und die Verbindungsart auszuwählen. Diese sind bereits mit den Default werden für den relevanten Anwendungsfall vorbelegt. Andere Optionen sind (noch) nicht implementiert, da sie zum jetzigen Zeitpunkt nicht gefordert sind. Durch klicken des Connect-Buttons wird die, der Auswahl entsprechenden, spezifische Reader-Klasse instanziiert und es wird versucht, die Verbindung aufzubauen. Ist beides erfolgreich, werden die GUI Elemente aktiviert. Jetzt kann ein einfacher Befehl, wie etwa „Field On/Reset“ausgeführt werden, um zu prüfen, ob die Initialisierung korrekt verlaufen ist.
Die History-Box zeigt den Erfolg oder Misserfolg der Aktion an. Damit ist auch gleich bekannt, ob die Funktionen *RegisterRead()* und *RegisterWrite()* ordnungsgemäß arbeiten.

### 6.1.2 Lesen und Schreiben von Registern aus der GUI

Um den Wert eines Registers zu lesen, muss lediglich das entsprechende Register aus der Liste ausgewählt werden. Die demzufolge emittierten Signale des QListWidget triggern den zugewiesenen Slot in CMainWindow, welcher dann ein RegisterRead() mit

der Adresse aus der Liste ausführt. War das Lesen des Registers erfolgreich, erscheint in der Box rechts oben neben der Liste der gelesene Wert in Hex-Darstellung.

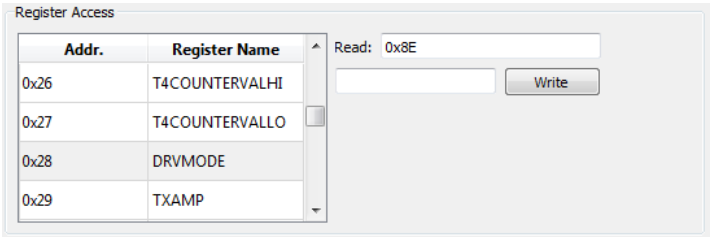


Abbildung 28: Ausgabe bei gelesenem Register

Schreibt man einen Wert in eines der Register, wird das betreffende Register sofort wieder ausgelesen und zeigt den neuen Wert an.

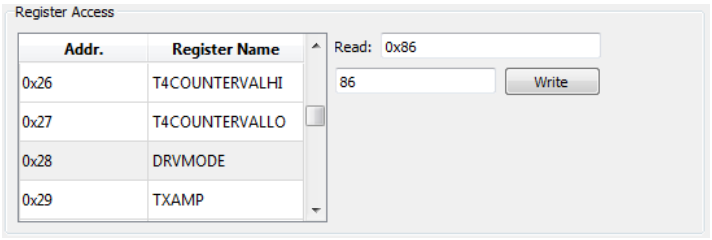


Abbildung 29: Ausgabe bei geschriebenem Register

### 6.1.3 Konfiguration der Bitraten für Typ A und Übertragungsprotokoll

Im Protokoll-Tab werden die Datenraten für Sende- und Empfangsstufe ausgewählt, und dann der Button Apply Settings geklickt. Die Sendeeinstellung kann sehr schnell mit einem Oszilloskop überprüft werden, indem einfach ein beliebiges Bitmuster via Frame Exchange gesendet wird.
Einfacher ist jedoch, für das Übertragungsprotokoll mit der PPS verschiedene Tx und Rx Bitraten auszuwählen, und eine Übertragung zu versuchen. So lassen sich zwei Fliegen mit einer Klappe schlagen, da so die Funktionalität der Teil 4 spezifischen direkt mit verifiziert werden kann. Es werden in den Tests jeweils die Applikations-IDs einer MIFARE DESFire Karte abgefragt.
Die Sequenzen lassen sich anhand der Historie ablesen, daher wird auf eine weitere Erklärung verzichtet.

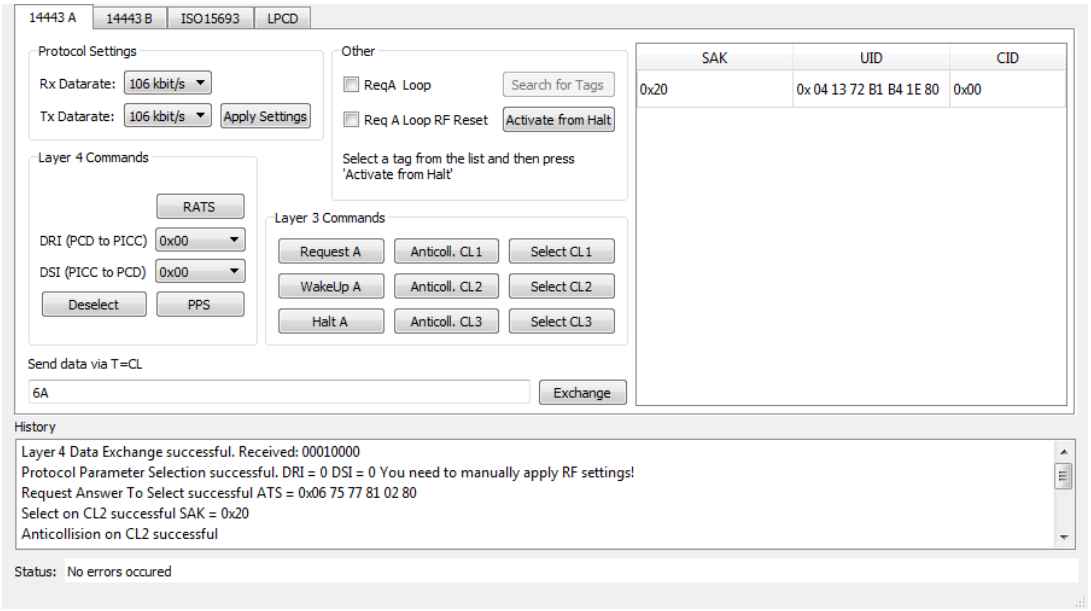


Abbildung 30: Kommunikation bei PCD->PICC = PICC->PCD: 106 kbit/s

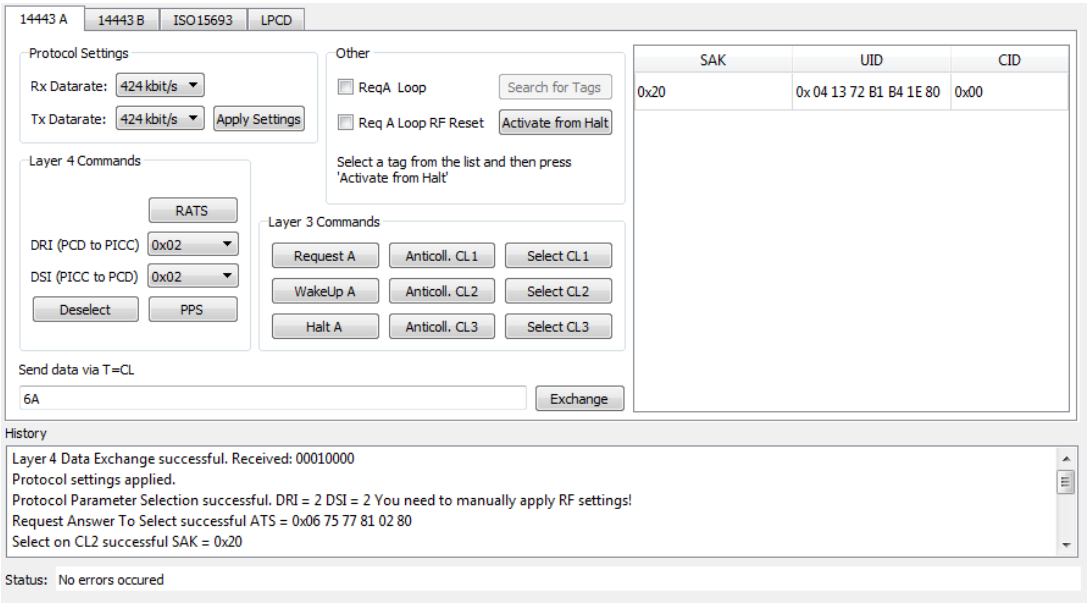


Abbildung 31: Kommunikation bei PCD->PICC = PICC->PCD: 424 kbit/s

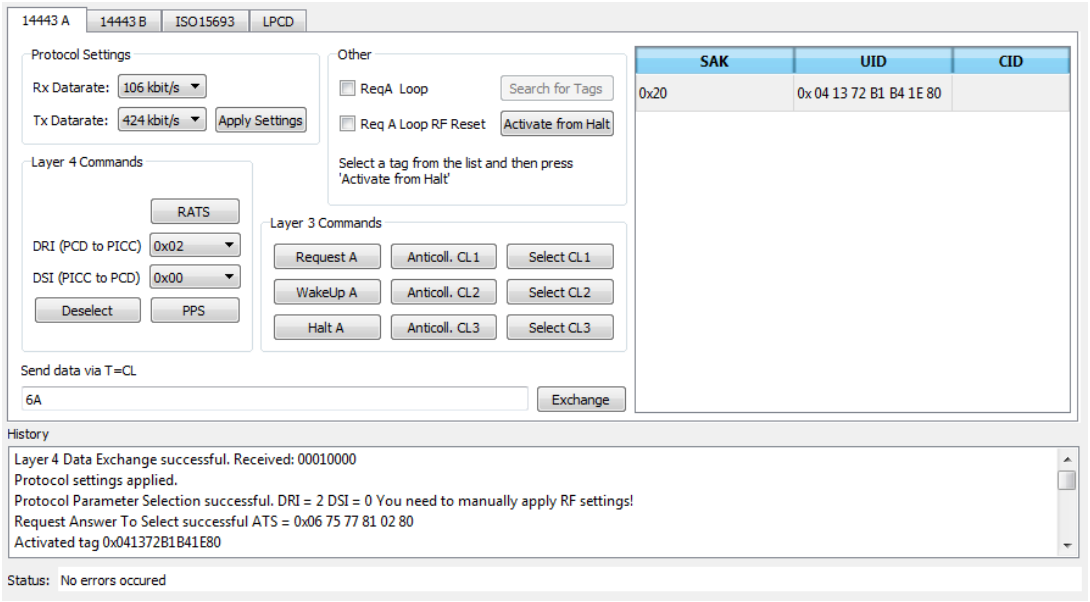


Abbildung 32: Kommunikation bei PCD->PICC: 424 kbit/s und PICC->PCD: 106 kbit/s

### 6.1.4 Lesen aller PICCs im Feld

Die in Abschnitt 5.7 beschriebenen Schritte zur Durchführung der Antikollision sollen die im Protokoll Tab befindliche Liste sukzessive mit der Information über alle PICCs im Feld füllen. Für den Test werden 6 Karten im Feld platziert, und die Sequenz manuell ausgeführt.
Abbildung 33 zeigt das Ergebnis in der grafischen Oberfläche. Wie zu sehen, wurden alle PICCs erkannt.

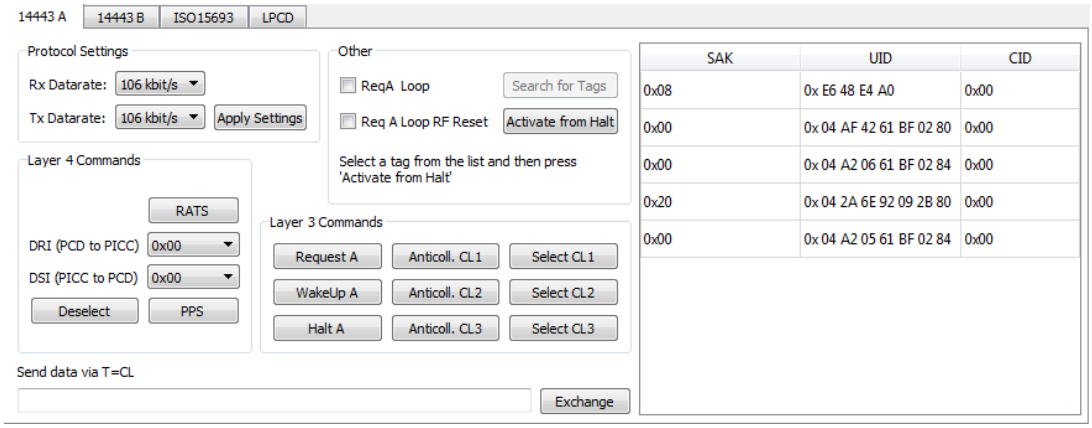


Abbildung 33: Liste mit erkannten Tags

Hinweis: Es tritt hierbei selten ein Fehler auf, bei dem die ATQA einer PICC korrekt gelesen wird, aber in der Reader Library selbst ein Fehler bei der Antikollision auftritt. Dies ist möglicherweise auf einen Bug in der verwendeten Library zurückzuführen. Derzeit laufen Untersuchungen, um das Problem zu lokalisieren.

Sind alle PICCs erkannt und im HALT-Zustand, antwortet keine von hnen mehr auf ein REQA Kommando. Der Fehler wird entsprechend in der GUI angezeigt:

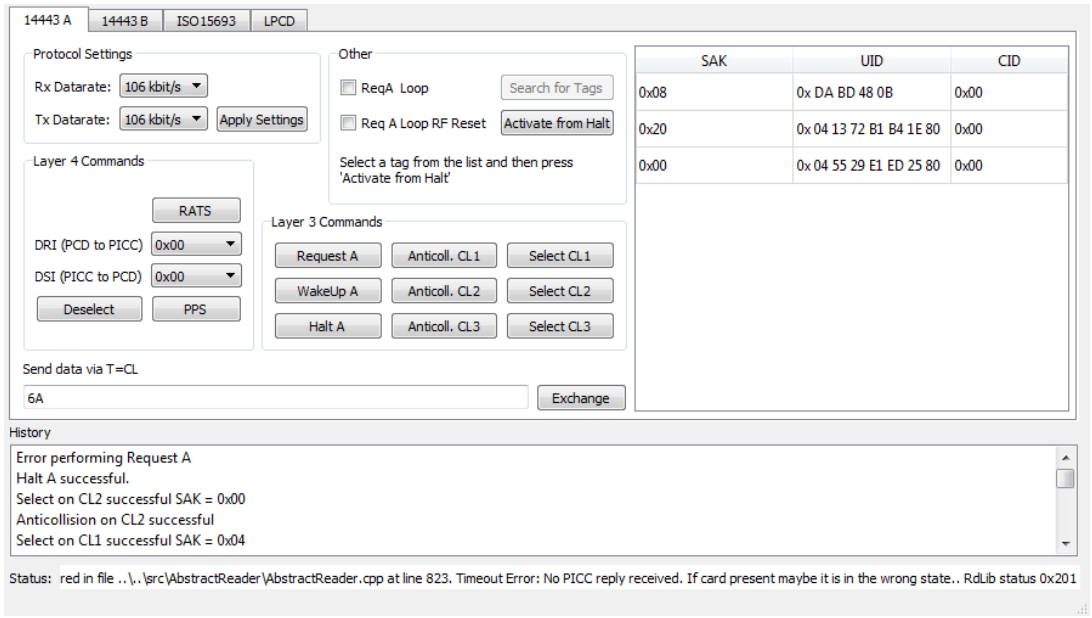


Abbildung 34: Liste mit erkannten Tags

# Abbildungsverzeichnis

# Tabellenverzeichnis

## Literatur

ECMA (2013a). *ECMA-340 Near Field Communication Interface and Protocol -1 (NFCIP-1)*. 3. Aufl. Rue du Rhone 114 CH-1204 Geneva, Switzerland: Ecma International (siehe S. 9).

– (2013b). *ECMA-352 Near Field Communication Interface and Protocol -2 (NFCIP-2)*. 3. Aufl. Rue du Rhone 114 CH-1204 Geneva, Switzerland: Ecma International (siehe S. 9).

Finkenzeller, Klaus (2006). *RFID Handbuch Grundlagen und praktische Anwendungen von Transpondern, kontaktlosen Chipkarten und NFC*. 4. Aufl. Kolbergerstraße 22, D-81679 München: Carl Hanser Verlag (siehe S. 7).

ISO/IEC (2008a). *FDIS 14443-1:2008, Identification cards – Contactless integrated circuit cards – Proximity cards – Part 1: Physical characteristics*. Geneva, Switzerland: International Organization for Standardization (siehe S. 7).

– (2008b). *FDIS 14443-4:2008, Identification cards – Contactless integrated circuit cards – Proximity cards – Part 4: Transmission protocol*. Geneva, Switzerland: International Organization for Standardization (siehe S. 18).

– (2011). *FDIS 14443-3:2011, Identification cards – Contactless integrated circuits cards – Proximity cards – Part 3: Initialization and anticollision*. Geneva, Switzerland: International Organization for Standardization (siehe S. 11, 12, 14, 16, 42–44).

Ltd., The Qt Company (2015a). *Qt 5.4 QThread Class 2015-02-25*. URL: http://doc-snapshot.qt-project.org/qt5-5.4/qthread.html (siehe S. 41).

– (2015b). *Qt 5.4 Signals & Slots 2015-02-25*. URL: http://doc.qt.io/qt-5/signalsandslots.html (siehe S. 33, 34).

## Glossary

**ABEND** ARM Board Engineering and Development. Eine Reihe von Werkzeugen zur Entwicklung mit NXP LPC Mikrocontrollern

**AL** Application Layer, dt. Anwendungsschicht. Vierte Schicht der NXP Reader Library. Setzt auf die PAL auf.

**APDU** Application Protocol Data Unit. Einheit eines Datenpakets auf Protokollebene, ISO/IEC 14443-4.

**ATQA** Answer To Request A. Antwort der PICC auf einen REQA Befehl der PCD

**BAL** Bus Abstraction Layer, dt. Bus Abstraktionsschicht. Unterste Schicht der NXP Reader Library

**DRI** Divisor Receive Integer, PCD -> PICC

**DSI** Divisor Send Integer, PICC -> PCD

**ECMA** Europpean Computer Manufacturers Association

**FSCI** Frame Size Card Integer

**FSDI** Frame Size Device Integer

**HAL** Hardware Abstraction Layer, dt. Hardware Abstraktionsschicht. Zweite Ebene der NXP Reader Library. Setzt auf die BAL auf.

**HALT** HALT Zustand der PICC. In diesem Zustand reagiert sie nur auf bestimmte Befehle.

**IDLE** IDLE Zustand der PICC. Dieser ist der Initialzustand nachdem die Karte mit Energie versorgt wird.

**IEC** International Electrotechnical Comission

**ISO** International Standardization Organization

**MIFARE** MIkron FARE Collection System. Ursprünglich von der Firma Mikron entwickelt für kontaktlose Fahrkartensysteme. Heute im Besitz von NXP Semiconductors. Entspricht den geltenden ISO/IEC Standards 7816 und 14443.

**NFC** Near Field Communication, dt. Nahfeldkommunikation. Funkstandard zum kontaktlosen Datenaustausch über kurze Distanzen

**PAL** Protocol Abstraction Layer, dt. Protokoll Abstraktionsschicht. Dritte Ebene der NXP reader Library. Setzt auf die HAL auf.

**PCD** Proximity Coupling Device, der Reader.

**PICC** Proximity Integrated Circuit Card, die Karte bzw. das Tag.

**PPS** Protocol Parameter Selection

**RATS** Request Answer To Select

**READY** READY Zustand der PICC. Dieser wird von der PICC nach senden der ATQA eingenommen.

**REQA** ISO/IEC 14443-3 Request A Befehl. Wird verwendet, um im Feld nach PICCs zu suchen.

**RFID** Radio Frequency IDentification, dt. etwa Funkfrequenz Identifikation oder Radiowellen Identifkation

**UID** Unique IDentifier, einzigartige Indentifikationsnummer der PICC.

**WUPA** ISO/IEC 14443-3 Wake Up A Befehl. Wird verwendet um im Feld nach PICCs zu suchen, auch wenn sie sich im HALT Zustand befinden.

# Index

# Anhang

Der Anhang befindet sich auf der CD und kann außerdem bei Prof. Dr. rer. nat. Henning Dierks und Prof. Dr. Ralf Wendel eingesehen werden.

## Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

| Hamburg, 26. Februar 2015 | |
|---|---|
| Ort, Datum | Unterschrift |