Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Oliver Kühn

Portierung und Erweiterung der Software eines Referenzdesigns einer Wegfahrsperre

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Oliver Kühn

Portierung und Erweiterung der Software eines Referenzdesigns einer Wegfahrsperre

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

Betreuender Prüfer : Prof. Dr. Heike Neumann
Zweitgutachter : Dirk Besenbruch (extern)

Abgegeben am 8. Oktober 2018

Oliver Kühn

Thema der Bachelorthesis

Portierung und Erweiterung der Software eines Referenzdesigns einer Wegfahrsperre

Stichworte

Automobil, Wegfahrsperre, MVC-Konzept, V-Modell, GUI

Kurzzusammenfassung

Diese Thesis beschreibt die Erstellung eines Referenzdesigns einer Wegfahrsperre auf Basis von NXP's „Advanced Base Station IC 2“, dem *ABIC2*. Ein vorhandenes Softwareprojekt wird auf den Mikrocontroller NXP *S32K144* portiert und funktional erweitert. Unter Nutzung des V-Modells wird zudem ein Übertragungsprotokoll und eine grafische Benutzerschnittstelle entwickelt, welche eine grundlegende Konfiguration des Referenzdesigns ermöglicht. Die Entwicklung der GUI erfolgt unter Anwendung des MVC-Konzeptes.

Title of the paper

Porting and Expansion of an Immobilizer Reference Design Software

Keywords

Automotive, Immobilizer, MVC-Concept, V-Model, GUI

Abstract

This thesis documents the development of an immobilizer reference design using the NXP Advanced Base Station IC 2, the *ABIC2*. An existing software project is ported to the microcontroller NXP *S32K144* and extended in functionality. Furthermore a communication protocol and a graphical user interface are developed making use of the V-model and the MVC-concept. The GUI enables a general configuration of the reference design.

Danksagung

An dieser Stelle möchte ich mein Wort an all diejenigen richten, die mich bei der Erstellung dieser Arbeit begleitet haben.

Ich bedanke mich bei meinen Kolleginnen und Kollegen von NXP, insbesondere bei Dirk und Sören für die Betreuung dieser Arbeit sowie für die Unterstützung bei Fragen und Problemen.

Bei Frau Neumann möchte ich mich für die umfangreiche Betreuung von Seiten der Hochschule bedanken.

Zu guter Letzt möchte ich mich besonders bei meiner Familie und meinen Freunden für die Unterstützung während des gesamten Studiums und insbesondere für die Hilfe rund um die Knie-Operation im Juli diesen Jahres bedanken.

Hamburg, September 2018

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ACK Acknowledgement

AEC Automotive Electronics Council

AES Advanced Encryption Standard

ASCII American Standard Code for Information Interchange

BPLM Binary Pulse Length Modulation

CAN Controller Area Network

CAS Customer Application Support

COTS Commercial Off-The-Shelf

CRC Cyclic Redundancy Check

FIFO First In - First Out

GNU LGPL GNU Lesser General Public License

GPIO General Purpose Input/Output

GUI Graphical User Interface

HAL Hardware Abstraction Layer

IC Integrated Circuit

IDE Integrated Development Environment

ISO International Organization for Standardization

JAR Java Archive

JRE Java Runtime Environment

LED Light Emitting Diode

LF Low Frequency

LIN Local Interconnected Network

LM Load Modulation

MVC-Concept Model-View-Controller-Concept

NAK Negative Acknowledgement

OEM Original Equipment Manufacturer

OpenSDA Open-Standard Serial and Debug Adapter

RS232 Recommended Standard 232

RFID Radio-Frequency Identification

SDK Software Development Kit

SPI Serial Peripheral Interface

SW Single Wire

ULP Ultra Low Power

UML Unified Modeling Language

USB Universal Serial Bus

1 Einleitung

Eine Person verschafft sich Zugang zum eigenen Fahrzeug und startet den Motor. Ein ursprünglich mechanischer Vorgang, welcher aus dem Entriegeln der Türen, dem Einführen des Schlüssels in das Zündschloss und dem typischen Dreh nach rechts bestand, ist gegenwärtig ein komplexer elektrotechnischer Prozess. Zur Verbesserung des Schutzes vor Fahrzeugdiebstahl entwickelten sich auf Basis einer leitungsungebundenen Kommunikation zwischen Schlüssel und Bordnetzsteuergerät des Fahrzeugs diverse Applikationen. Aktuelle Innovationen bieten neben der Zugangskontrolle zunehmend ein breites Spektrum an Möglichkeiten im Sinne des Komforts bis hin zum „Smart Car Management“. Schlüsselloser Zugang oder ferngesteuertes Einparken sind Beispiele dafür.

Den Startschuss dieser Entwicklung bildete 1993 sinngemäß nach [35] das Anforderungsprofil für die elektronische Wegfahrsperre des Allianz-Zentrums für Technik in Folge stark gestiegener Autodiebstähle nach 1989. Mitsamt der mechanischen Identifikation erfolgt vor Freigabe der Bordelektronik eine Authentifikation zwischen einem im Schlüssel verbauten Transponder und einer Basisstation im Bordnetzsteuergerät. In aktuellen Fahrzeugmodellen dient die Implementierung der elektronischen Wegfahrsperre meist als Ergänzung oder Backup für oben genannte komplexere Applikationen.

Diese Bachelorthesis beschäftigt sich mit dem „**A**dvanced **B**asestation **I**ntegrated **C**ircuit **2**“, dem *ABIC2*, einem Transceiver-Baustein, welcher als Kommunikationsschnittstelle zwischen Fahrzeug und Schlüssel dient. 2006 als Nachfolger des *ABIC* von Philips Semiconductors entwickelt, wechselte der *ABIC2* beim Übergang zu NXP Semiconductors im selben Jahre in deren Zuständigkeit. NXP Semiconductors ist Anbieter für sichere Verbindungen und entwickelt Lösungen im industriellen Bereich. In Hamburg Lokstedt befindet sich der Hauptsitz der NXP Semiconductors Germany GmbH sowie die Abteilung CAS[1] der Produktlinie „Secure Car Access“, welcher die Verantwortlichkeit über den *ABIC2* obliegt.

[1] Kundenbetreuung, engl. **C**ustomer **A**pplication **S**upport

In der Zulieferpyramide der Automobilindustrie befindet sich NXP an Position des Komponentenlieferanten und liefert ICs[2] an den Modullieferanten, welcher für die bekannten Automarken beispielsweise das Bordnetzsteuergerät entwickelt. Folgende Abbildung zeigt konkrete Beispiele der Versorgungskette mit den Fachbegriffen der Zulieferpyramide. Die deutschen Bezeichnungen sind gelb und die englischen grün unterlegt.

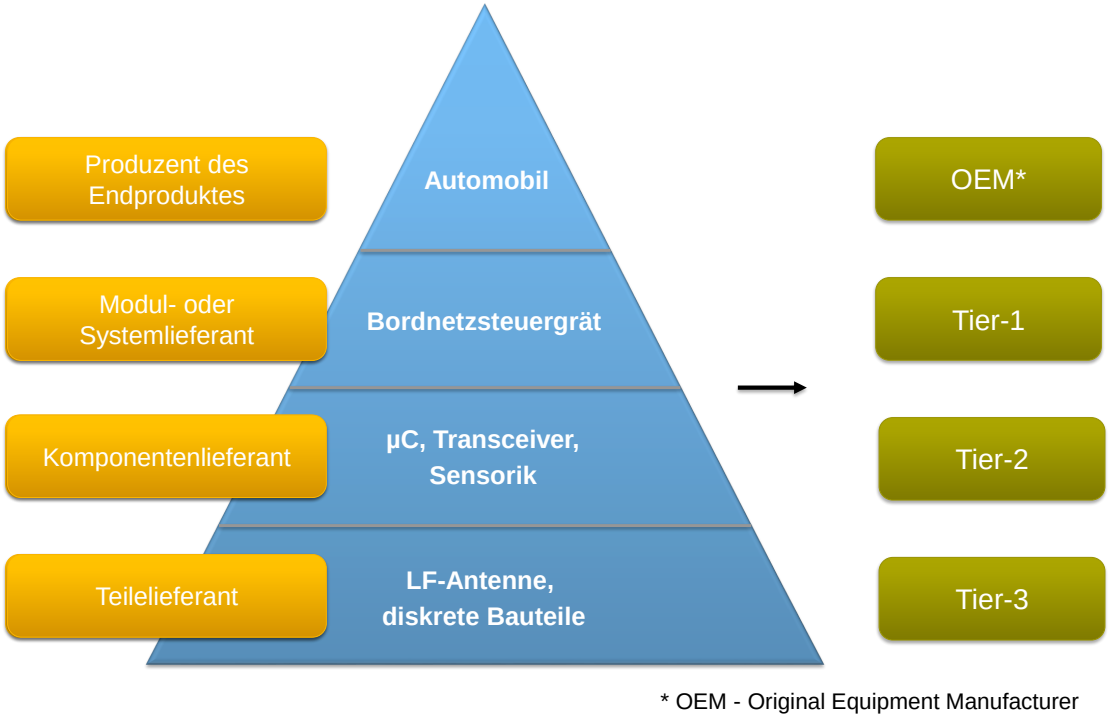

Abbildung 1.1: Zulieferpyramide der Automobilbranche, sinngemäß nach [37]

Vor dem Verkauf des *ABIC2* erfolgt zunächst die Übergabe in Form eines Referenzdesigns, welches die Systemkette innerhalb des Fahrzeugs nachbildet. Während dem Modullieferanten so ein Kennenlernen und Testen des Produktes ermöglicht wird, bildet es intern die Basis für Validierungen und Verifizierungen.

Ziel dieses Projektes ist die Erstellung eines solchen Referenzdesigns unter Anwendung des aktuellen NXP Automobil-Mikrocontrollers *S32K144*. Die vorhandene Funktionalität wird auf das neue Design portiert und durch Entwicklung einer grafischen Benutzerschnittstelle erweitert. Eine detailliertere Beschreibung sowie eine Analyse der Aufgabenstellung finden sich in Kapitel 3.

[2]Integrierter Schaltkreis, engl. **I**ntegrated **C**ircuit

2 Technische Grundlagen

2.1 Von Referenzdesigns und Kundenevaluierungsboards

Die Notwendigkeit eines Referenzdesigns für eine technische Applikation begründet sich über deren Komplexität. Ohne externe Beschaltung ist bereits die Inbetriebnahme eines einzelnen ICs mit großem Aufwand verbunden. Für diesen Zweck existiert gewöhnlich ein Kundenevaluierungsboard, welches den optimalen Betrieb des ICs aufzeigt. Verkauft wird letztendlich aber nur die integrierte Schaltung.

Vor der Entscheidung eines Kunden für den *ABIC2* steht ein zeitintensiver Validierungsprozess für die Implementierung im eigenen System. Dieser resultiert zwangsläufig aus der hohen Lebensdauer eines Automobils, aber auch aus der sehr niedrigen Fehlertoleranz beim Öffnen und Starten des Fahrzeuges. Maßgebend ist die ISO 26262 Norm[3] sowie das AEC Q100 Qualifikationsverfahren[4]. Daraus ergibt sich eine intensive Kundenbetreuung innerhalb des Design-Ins. Eine direkte Einflussnahme des Kunden während der Entwicklungsphase oder eine nachträgliche Anpassung eines Produktes sind übliche Prozesse.

Die Basis der Kundenbetreuung bildet das Referenzdesign. Anhand dessen können Produkteigenschaften vermarktet und kundenseitige Fragen beantwortet werden. Gleichzeitig dient es zur internen Reproduktion bestimmter Verhaltensmuster. Je nach Applikation setzt sich das Referenzdesign aus einer oder mehreren Komponenten zusammen. Im Falle des *ABIC2* und der elektronischen Wegfahrsperre besteht es aus einem steuernden Mikrocontroller, dem *ABIC2*-Kundenevaluierungsboard inklusive Antenne sowie einem schlüsselseitigen Transponder.

Zusammenfassend lässt sich festhalten, dass ein Kundenevaluierungsboard den ordnungsgemäßen Betrieb eines integrierten Schaltkreises gewährleistet, während ein Referenzdesign eine realitätsnahe Nachbildung der Applikation sein soll. Durch den Zugang zu Debug-Ports werden dabei erweiterte Möglichkeiten zum Testen und zur Fehlersuche geschaffen.

[3] Sinngemäß nach [9] Norm für 'Funktionale Sicherheit elektrischer und elektronischer Systeme in Kraftfahrzeugen'

[4] Sinngemäß nach [1] Stresstest-Qualifizierung für integrierte Schaltungen

2.2 Die elektronische Wegfahrsperre

Der Aufbau eines Bordnetzsteuergerätes sowie die Kommunikation innerhalb der Fahrzeugelektronik ist nicht Fokus dieser Arbeit und wird im Folgenden vereinfacht im Sinne der Funktionalität des *ABIC2*-Referenzdesigns beschrieben.

2.2.1 Übersicht

Auf dem Bordnetzsteuergerät steuert ein übergeordneter Mikrocontroller verschiedene Teilmodule der Fahrzeugelektronik. Dazu gehören unter anderem Fensterheber, Sitzheizung, Tachometer, aber auch der LF-Transceiver *ABIC2* der Wegfahrsperre. Die Interaktion vom Controller zum *ABIC2* erfolgt entweder über das Bussystem SPI[5] oder mit einer an den LIN-Bus[6] angelehnten SW[7]-Lösung. Eine Kommunikation des *ABIC2* mit dem Transponder wird über magnetisch gekoppelte Spulen realisiert. Über ein LF[8]-Feld auf 125 kHz können so Reichweiten von mehreren Zentimetern erreicht werden. Während die Spule der Transceiverantenne in das Zündschloss integriert ist, ist der Transponder mit dessen Spule im Fahrzeugschlüssel verbaut. Auf einem höheren Abstraktionslevel setzen die *HITAG*-Transponderprotokolle von NXP die Rahmenbedingungen und Regeln für den Ablauf einer Authentifikation.

Abbildung 2.1 zeigt den konzeptionellen Aufbau der Applikation der Wegfahrsperre. Inklusive der Hostverbindung mittels USB bzw. RS232 sowie die damit verbundenen Möglichkeiten zur Konfiguration und zum Debugging wird ebenso die Struktur des Referenzdesigns dargestellt.

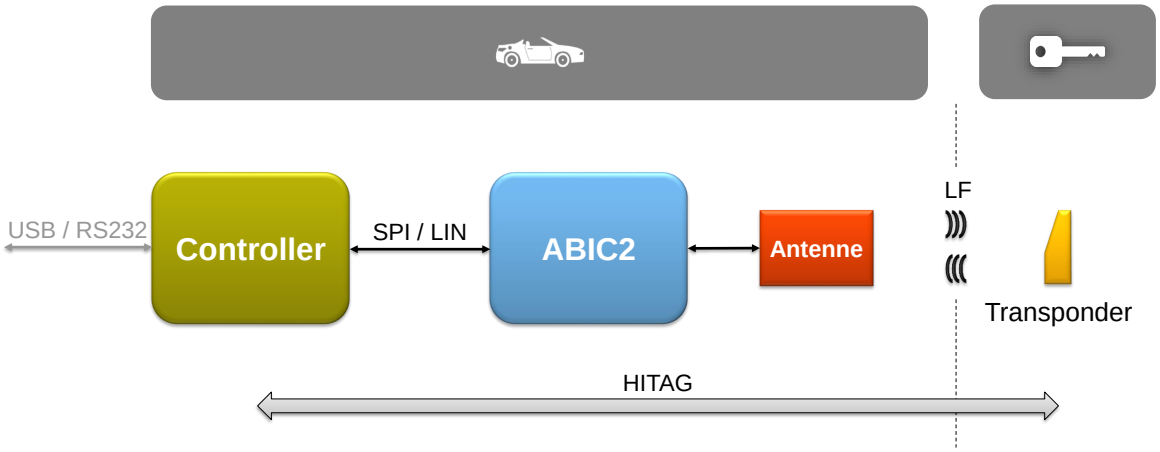

Abbildung 2.1: Konzept *ABIC2*-Referenzdesign, Abbildung vom Autor erstellt

[5] **S**erial **P**eripheral **I**nterface, weiterführende Erklärung in 2.2.4
[6] **L**ocal **I**nterconnected **N**etwork, weiterführende Erklärung in 2.2.4
[7] Halbduplexe Übertragung über eine Datenleitung, engl. **S**ingle **W**ire
[8] Niederfrequentes Band zwischen 30 kHz und 300 kHz, engl. **L**ow **F**requency

Folgende Abschnitte geben einen Einblick in die Komponenten und Schnittstellen des Referenzdesigns. Nach kurzer Vorstellung des *ABIC2* und der *HITAG*-Transponder wird zunächst die Kommunikation zwischen Controller und *ABIC2* beleuchtet. Danach erfolgt eine Betrachtung der LF-Kommunikation zwischen Transponder und *ABIC2*. Abschließend wird eine Authentifikation nach dem Challenge-Response-Prinzip beschrieben.

2.2.2 NXP ABIC2

Folgende Beschreibung ist eine kurze Zusammenfassung auf Basis der Dokumentation in [13] und [14]. Für weiterführende Informationen sei darauf verwiesen.

Der *ABIC2* ist ein LF-Transceiver, welcher für eine einfache Integration in Applikationen der elektronischen Wegfahrsperre entwickelt wurde. Die digitale Ansteuerung erfolgt kommandobasiert und wahlweise über die Bussysteme SPI oder LIN. Eine integrierte digitale Logik erlaubt eine Konfiguration und bietet auslesbare Status- und Parameterregister.

Die leitungsungebundene Kommunikation mit dem Transponder wird durch Nutzung eines magnetischen Feldes auf 125 kHz realisiert. Bestimmte Kommandos erlauben eine direkte Anregung des LF-Feldes. Der *ABIC2* übernimmt Modulation und Demodulation sowie Dekodierung der empfangenen Daten. Die Kommunikation zwischen *ABIC2* und Transponder wird im Abschnitt 2.2.5 noch einmal ausführlicher betrachtet.

Die direkte Spannungsversorgung von der Fahrzeugbatterie gewährleistet die notwendige Energie für den Aufbau des Magnetfeldes. Mit einem Treiberstrom von maximal 400 mA ist eine Kommunikation bei gegebener Kopplung über bis zu 4 cm möglich. Als Antenne wird ein Reihenschwingkreis verwendet.

2.2.3 NXP HITAG

HITAG bezeichnet ein Portfolio an RFID[9]-Transponder-ICs, welche im LF-Segment von 100 kHz bis 150 kHz angesiedelt sind, [18]. Gleichzeitig geben die Transponder mit ihrem internen Aufbau, der Speicherorganisation, einem definierten Kommandoset inklusive Zeitverhalten sowie der Verschlüsselung die High-Level-Kommunikation zwischen Transponder und Basisstation[10] vor. *HITAG*, abgekürzt mit *HT*, ist demnach auch eine Protokollfamilie.

[9] Technologie zum kontaktlosen Identifizieren von Objekten oder Lebewesen, engl. **R**adio-**F**requency **Id**entification

[10] Bezeichnung der fahrzeugseitigen Komponenten einer Wegfahrsperre

Für den *ABIC2* kommen *HT-2*, *HT-3*, *HT-AES* und *HT-Pro* zum Einsatz. Alle in dieser Thesis konkret gemachten Angaben sind dem proprietären *HT-2* entnommen. Weiterführende Informationen sind in [19] zu finden.

2.2.4 Kommunikation zwischen Controller und ABIC2

Die digitale Ansteuerung des *ABIC2* kann entweder unter Verwendung der SPI- oder der LIN-Schnittstelle erfolgen.

Serial Peripheral Interface - SPI

SPI ist sinngemäß nach [34] ein Standard für einen seriellen, vollduplexen, synchronen Datenbus mit einer Master-Slave-Architektur. Für weiterführende Informationen sei auf das in der Quelle angebotene Tutorial verwiesen.

Ein Betrieb des *ABIC2* mit SPI zielt auf Implementierungen in naher Entfernung zum steuernden Mikrocontroller, gewöhnlicherweise auf derselben Platine, [13, Seite 7].

Local Interconnected Network - LIN

Folgende allgemeine Beschreibung der LIN-Schnittstelle basiert sinngemäß auf [36]. Für weiterführende Informationen sei auf das in dieser Quelle angebotene Lernmodul verwiesen. Die Implementierung im *ABIC2* ist sinngemäß aus [13, Seite 5] entnommen.

LIN ist ein Standard für ein serielles Bussystem, welches als kostengünstige Alternative zum CAN[11]-Bus entwickelt wurde. Zum Einsatz kommt LIN besonders in Komfortanwendungen wie zum Beispiel für die Steuerung der Klimaanlage, des Sitzes oder der Türen. Die Kommunikation erfolgt seriell, halbduplex und in einer Master-Slave-Architektur. Die Spannungslevel orientieren sich an der Batteriespannung des Fahrzeuges. Das macht den Einsatz eines LIN-Transceivers wie dem *TJA1027* auf Controllerseite notwendig. Eine detallierte Beschreibung kann aus dessen Datenblatt in [21] entnommen werden.

Die Implementierung der LIN-Schnittstelle im *ABIC2* ist proprietär und unterstützt aufgrund der, durch die *HITAG*-Protokolle vorgegebenen, Transponderkommunikation nicht alle Merkmale des Standards. Die implementierte „Single-Wire-Lösung“ erlaubt aber ein Betrieb als Slave innerhalb eines LIN-Netzwerkes ohne Beeinflussung der Kommunikation des Masters mit anderen Slaves. Ein Betrieb unter Verwendung von LIN zielt auf Implementierungen, bei denen der *ABIC2* auf einer anderen Platine als der steuernde Mikrocontroller betrieben wird. Gegebenenfalls bedient der Mikrocontroller weitere Knoten für andere Funktionen.

[11] Serielles Bussystem im Fahrzeug, engl. **C**ontroller **A**rea **N**etwork

2.2.5 LF-Kommunikation zwischen ABIC2 und Transponder

Folgende Beschreibung basiert sinngemäß auf dem Datenblatt des *ABIC2* in [14].

Die Kopplung zweier Spulen mit einem magnetischen Feld einer Frequenz von 125 kHz erlaubt eine Kommunikation über kurze Distanzen von wenigen Zentimetern. Der Aufbau kann als lose gekoppelter Transformator betrachtet werden. Ein Stromfluss durch die Antennenspule baut ein Feld auf, welches bei Durchdringung der Transponderspule eine Spannung induziert.

Der *ABIC2* erfüllt im Wesentlichen drei Kernaufgaben. Neben der Übertragung eines Befehls und dem Empfangen einer Antwort muss der Transponder über das Magnetfeld auch mit Energie versorgt werden.

Die Energieversorgung wird über einen Ladekondensator bereitgestellt. Durch Speichern der, aus der induzierten Spannung resultierenden, Energie kann der Transponder auch für eine gewisse Zeit ohne die direkt bereitgestellte Energie des aktiven Feldes operieren. Das erlaubt eine Kommunikation von der Basisstation zum Transponder, indem die logische Null und Eins mit verschiedenen Pulslängen moduliert wird. Diese digitale Modulationsart wird BPLM[12] genannt und ist in Abbildung 2.2 visualisiert. Das Antennensignal beschreibt den Spannungsverlauf zwischen Kondensator und Spule des Reihenschwingkreises.

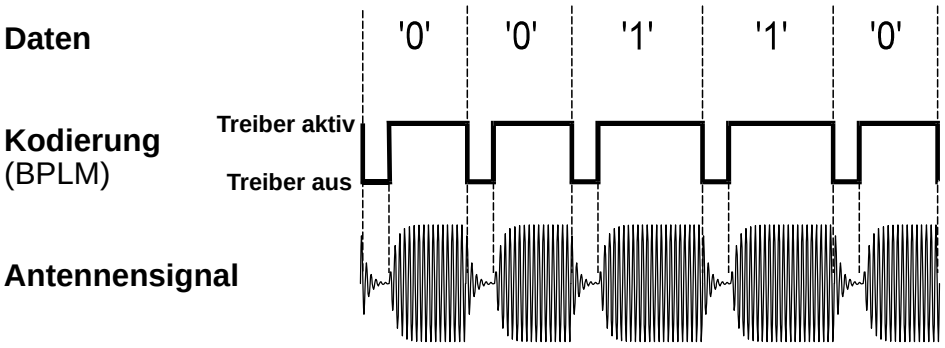

Abbildung 2.2: Funktionsweise BPLM im *ABIC2*, sinngemäß nach [14, Seite 17]

Der Transponder erkennt den Anfang eines Bits anhand einer definierten Zeit, in der das Feld deaktiviert ist. Logisch Null und Eins werden anhand der Zeit unterschieden, welches das Feld danach aktiv ist. Eine zusätzliche Stoppbedingung ist notwendig, um das Ende der Übertragung zu kennzeichnen. Das genaue Zeitverhalten wird durch das jeweilige Transponderprotokoll festgelegt.

Die Kommunikation vom Transponder zur Basisstation wird mit einer Lastmodulation realisiert. Durch Veränderung der Impedanz des Transponders verändert sich

[12] Digitale Modulationsart, engl. **B**inary **P**ulse **L**ength **M**odulation

die Energie, welche dem Magnetfeld entzogen wird. Das kann über eine Messung der Spannung zwischen Spule und Kondensator des Reihenschwingkreises der Antenne gemessen werden. Durch Schalten eines Widerstandes auf dem Transponder kann die Basisstation also zwei verschiedene Zustände unterscheiden. Mithilfe einer Manchester- oder einer differentiellen Manchester-Kodierung kann in dieser Art und Weise eine Übertragungsrate von bis zu 4 kbit/s erreicht werden. Folgende Abbildungen 2.3 und 2.4 stellen Kodierung und Funktionsweise der Lastmodulation (LM) dar.

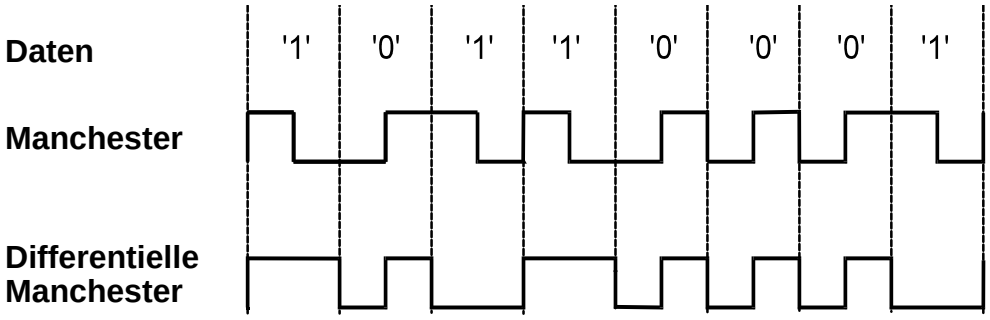

Abbildung 2.3: Kodierung LM im Transponder, sinngemäß nach [19, Seite 5]

Die obere Hälfte der Abbildung 2.4 stellt die Transponderkommunikation als Ersatzschaltbild eines Transformators dar. Mit Schließen des Schalters wird dem Magnetfeld durch den Lastwiderstand R_L mehr Energie entzogen. Die untere Hälfte der Abbildung stellt die Spannungsverhältnisse zwischen Spule und Kondensator des Reihenschwingkreises der Basisstation dar. Während V_{LF-Low} das Spannungslevel bei zugeschalteter Last repräsentiert, zeigt $V_{LF-High}$ das Spannungslevel ohne Last.

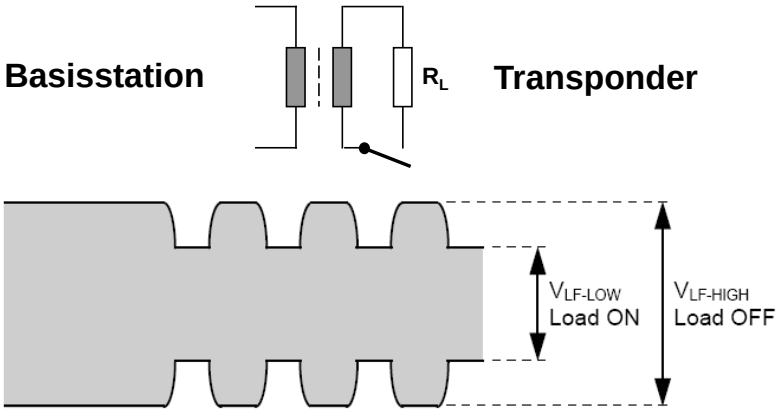

Abbildung 2.4: Funktionsweise LM, sinngemäß nach [19, Seite 6]

Die in Abbildung 2.4 gezeigten Spannungsunterschiede V_{LF-Low} und $V_{LF-High}$ verdeutlichen die Funktionsweise, stellen aber keine realistischen quantitativen Verhältnisse dar. Diese bewegen sich im mV-Bereich und setzen einen Empfänger mit hoher Sensitivität voraus.

2.2.6 Authentifikation nach dem Challenge-Response-Prinzip

Folgende Beschreibung basiert sinngemäß auf [29, Seite 109 ff.].

In einem Challenge-Response-Verfahren erfolgt eine eindeutige Authentifikation auf Grundlage einer einmaligen Frage, welche von Teilnehmer A (Alice) einem Teilnehmer B (Bob) gestellt wird. Diese Frage kann Bob nur beantworten, wenn er über eine bestimmte geheime Information, den sogenannten Schlüssel („Secret Key“), verfügt. Dabei darf die Antwort den Schlüssel selbst nicht preisgeben.

Das Challenge-Response-Verfahren wird bei der elektronischen Wegfahrsperre mit einer symmetrischen Verschlüsselung realisiert. Das bedeutet, dass Alice und Bob die gleiche geheime Information teilen. Bob möchte sich bei Alice authentifizieren und sendet seine Identität in Form einer Seriennummer oder Vergleichbarem. Wenn nun Alice Bob eine Zufallszahl sendet („Challenge“), beide auf diese Zufallszahl sowie auf Bobs Identität ihren geheimen Schlüssel anwenden und Bob sein verschlüsseltes Ergebnis an Alice zurücksendet („Response“), ist Alice in der Lage sich durch Vergleich der Ergebnisse von Bobs Authentizität zu überzeugen.

Innerhalb der elektronischen Wegfahrsperre ist Alice die fahrzeugseitige Elektronik und Bob der Fahrzeugschlüssel. Es erfolgt eine gegenseitige Authentifikation. Je nach Transponderprotokoll sind verschiedene Abläufe, Zeitverhalten, Schlüssellängen und Verschlüsselungsverfahren festgelegt.

Konkret kommt für *HT-2* ein Verfahren mit 48 Bit Schlüssellänge und einer von NXP entwickelten Verschlüsselung zum Einsatz. Ein Authentifikationsvorgang dauert typischerweise 40 ms.

2.3 NXP Automobil-Mikrocontrollerfamilie S32K

2.3.1 S32K Mikrocontroller

S32K ist ein Portfolio von 32-Bit-Mikrocontrollern. Basierend auf ARM Cortex Architekturen bietet sie Strukturen für typische Automobilanwendungen wie zum Beispiel der CAN- und LIN-Bus oder ein Betrieb im ULP[13]-Modus. Für die Programmierung existiert die Entwicklungsumgebung S32-Design-Studio inklusive eines

[13]Stromsparender Betriebsmodus des Mikrocontrollers, engl. **U**ltra **L**ow **P**ower

code-generierenden SDKs[14], welches durch eine grafische Benutzeroberfläche bedient wird. Eine Übersicht über dieses Portfolio ist in [11] gegeben.

Die S32K Mikrocontrollerfamilie ist ein COTS[15]-Produkt, welches auch in der realen Applikation zum Einsatz kommen könnte. Die Erstellung des neuen *ABIC2*-Referenzdesigns soll mit dem *S32K144* als steuernden Mikrocontroller erfolgen.

2.3.2 S32K144 Evaluierungsboard

Der *S32K144* wird im Rahmen dieses Projektes auf dem zugehörigen Evaluierungsboard betrieben. Folgende Beschreibung basiert sinngemäß auf [10].

Das Evaluierungsboard verfügt über vorbereitete Schnittstellen für die Bussysteme und über Arduino™ UNO kompatible Steckerleisten sowie Drucktaster und „Touch"-Elektroden für die Benutzerinteraktion. Mit OpenSDA[16] bietet das Evaluierungsboard außerdem die Möglichkeit der seriellen Kommunikation zwischen dem Mikrocontroller und einem Host über die USB-Verbindung. Das erlaubt neben der direkten Flash-Programmierung auch einen Debug-Modus innerhalb der IDE[17]. Ein Foto des Evaluierungsboards ist in Abbildung A.1 dargestellt.

Die Auswahl der Softwaremodule des *S32K144* für dieses Projekt erfolgt in Abschnitt 4.1. Eine finale Übersicht der verwendeten Pins gibt Anhang B.

[14] Set von Werkzeugen und Bibliotheken zur Softwareentwicklung, engl. **S**oftware **D**evelopment **K**it
[15] Seriengefertigte Produkte, engl. **C**ommercial **O**ff-**T**he-**S**helf
[16] **Open** Standard **S**erial and **D**ebug **A**dapter, für weiterführende Informationen sei auf [12] verwiesen
[17] Integrierte Entwicklungsumgebung, engl. **I**ntegrated **D**evelopment **E**nvironment

2.4 Fachtermini

Für eine bessere Übersichtlichkeit und eine saubere Trennung werden folgende Bezeichnungen für die Thesis eingeführt.

> *LPC93x-Design* beschreibt das existierende Referenzdesign,
> gesteuert durch den NXP Mikrocontroller *89LPC93*.
>
> *S32-Design* beschreibt das zu erstellende Referenzdesign,
> gesteuert durch den NXP Mikrocontroller *S32K144*.

Darüber hinaus werden Bezeichnungen und Abkürzungen, teils aus dem Englischen, für folgende Fachbegriffe verwendet.

Immobilizer -
: Elektronische Wegfahrsperre

Basisstation -
: Fahrzeugseitige Komponenten der elektronischen Wegfahrsperre

HT -
: NXP Transponderprotokoll ***HIT****AG*

μC -
: Mikrocontroller

CEB (engl. **C**ustomer **E**valuation **B**oard) -
: Kundenevaluierungsboard

GUI (engl. **G**raphical **U**ser **I**nterface) -
: Graphische Benutzerschnittstelle

Alle Abbildungen in dieser Thesis ohne Quellenangabe wurden vom Autor im Zuge dieses Projektes erstellt.

3 Analyse der Aufgabenstellung

3.1 Ziel des Projektes

Das Referenzdesign des *ABIC2* soll unter Verwendung des *S32K144* in Betrieb auf dem zugehörigen CEB erneuert werden. Ein angepasstes CEB wurde für den *ABIC2* bereits entwickelt. Die Hardware ist somit definiert. Eine detaillierte Beschreibung und eine Aufstellung der sich daraus ergebenden Anforderungen an die Software folgen in Unterabschnitt 4.1.3.

Die durch das *LPC93x-Design* gegebene Funktionalität wird auf das *S32-Design* übertragen. Die Steuerung mit dem deutlich leistungsfähigeren Mikrocontroller *S32K144* erlaubt darüber hinaus eine funktionale Erweiterung. Eine Aufstellung neuer Funktionen gibt ebenfalls Unterabschnitt 4.1.3.

Ergänzend soll eine Schnittstelle entwickelt werden, welche dem Kunden eine schnelle Inbetriebnahme und Konfiguration des *ABIC2* außerhalb der IDE ermöglicht. Eine Aufstellung der Anforderungen erfolgt in Unterabschnitt 5.2.1.

3.2 Rahmenbedingungen des Projektes

Die Aufgabenstellung ist unter Einhaltung folgender Bedingungen zu bearbeiten.

- Die Entwicklung der Software orientiert sich an den Regeln und Methoden der Produktlinie „Secure Car Access“ in [15].
- Die Programmierung des *S32K144* erfolgt in der Umgebung des S32-Design-Studios.
- Es wird nur lizenzkostenfreie Software verwendet. Im Rahmen der Evaluierung dürfen für den Kunden keine Kosten anfallen.

3.3 Anforderungsanalyse des Projektes

Das zu erstellende Referenzdesign soll dem Kunden eine schnelle Inbetriebnahme des *ABIC2* ermöglichen und Vorbild für die Implementierung im eigenen System sein. Durch Nutzung eines aktuellen Mikrokontrollers lassen sich eine ganze Reihe

an zusätzlichen Möglichkeiten realisieren. Dabei darf der Fokus auf das eigentliche Produkt, den *ABIC2*, aber nicht verloren gehen. Das Referenzdesign soll weiterhin nachvollziehbar und möglichst schlank bleiben.

Unter Umständen ist die Implementierung auf dem *S32K144* für einen Kunden nicht von Bedeutung, da im eigenen System längst ein anderer steuernder Controller feststeht. Für diesen Kunden wäre die zeitintensive Installation und Einarbeitung in die IDE nicht notwendig. Eine Evaluation des *ABIC2* soll demnach auch ohne Kenntnisse über den *S32K144* möglich sein.

Für Kunden, welche im Moment der Übergabe ohne eigenes System arbeiten, könnte das Referenzdesign jedoch auch mit Blick auf den Controller von Interesse sein. Der *S32K144* ist ein Mehrzweck-Mikrocontroller im Automobilbereich, ein eigenes Produkt mit entsprechend umfangreicher Dokumentation. Das *ABIC2*-Referenzdesign kann nur ein Einstiegspunkt im Sinne der Immobilizer-Applikation sein. Gleichzeitig kann es aber ebenso Ausgangspunkt für die Nutzung des *S32K144* sein, welches Möglichkeiten sowie Art und Weise der Programmierung aufzeigt und nicht in einem abgeschlossenen Rahmen lediglich die *ABIC2*-Funktionalität herstellt. Merkmale dessen wären eine ausführliche Kommentierung sowie die Implementierung unterstützender Funktionalität zum Betrieb des *ABIC2*, wie zum Beispiel die Nutzung der im *S32K144*-CEB eingebauten LED oder die Vorbereitung von Interrupt-Routinen vorhandener Taster.

Diese Anforderungen führen zu einer Art Zweistufigkeit. Im *S32-Design* könnte auf der einen Seite das mitgelieferte S32-Design-Studio als Programmierumgebung und auf der anderen Seite eine schlichte grafische Oberfläche für einen direkten Betrieb des *ABIC2* dienen. Dieser Ansatz wird im Folgenden konkretisiert und auf einem hohen Abstraktionslevel kurz ausformuliert.

(1) *Das S32-Design muss eine Beispielimplementierung des ABIC2 in einer Immobilizer-Applikation sein. In der Umgebung des S32-Design-Studios muss in die Funktionsweise des ABIC2 eingeführt und weiterführende Möglichkeiten des S32K144 aufgezeigt werden.*

(2) *Es muss eine GUI entwickelt werden, welche eine grundlegende Konfiguration des S32-Designs ermöglicht. Aktuelle Konfiguration sowie die Ausgabedaten müssen visualisiert werden. Die Oberfläche muss ohne Installation des S32-Design-Studios nutzbar sein und sich durch eine schnelle, unkomplizierte Inbetriebnahme auszeichnen.*

Dabei gilt es zu beachten, dass die GUI funktional auf dem im S32-Design-Studio entwickelten Softwareprojekt aufbaut. Daraus ergibt sich eine klare Reihenfolge im Arbeitsablauf sowie im Aufbau dieser Thesis. Die Beschreibung der Umsetzung von Anforderung (1) erfolgt in Kapitel 4. Anforderung (2) ist in Kapitel 5 beschrieben.

Für die Entwicklung der GUI wird das V-Modell als Vorgehensmodell gewählt. Eine Einführung in das V-Modell sowie dessen Anwendung in diesem Projekt ist in Abschnitt 5.1 erläutert.

Beide Anforderungen sind Softwareprojekte mit dem finalen Ziel der Verteilung an den Kunden. Diese Thesis dokumentiert die Entwicklung von Prototypen, welche unter Version 0.9 firmenintern zur Verfügung gestellt werden. Notwendige Schritte bis zur Produktreife umreißt das Kapitel 7 im Ausblick.

4 Portierung und funktionale Erweiterung

Im folgenden Abschnitt werden aus der vorhandenen Softwarestruktur des *LPC93x-Designs* und der gegebenen Hardware Anforderungen an die Software des *S32-Designs* erarbeitet. Die erforderlichen Anpassungen sind im Abschnitt 4.2 beschrieben. Umsetzung und Testphase werden in 4.3 und 4.4 geschildert.

4.1 Anforderungsanalyse

4.1.1 Analyse der vorhandenen Software

Softwarearchitektur

Die Software des *LPC93x-Designs* ist in einer modularen Architektur aufgebaut, welche sich mit einem Schichtenmodell beschreiben lässt. Die folgende Aufstellung unterteilt die Schichten und beschreibt kurz deren Aufgabe.

- **Anwendungsschicht**
  Programmablauf des Referenzdesigns zur Demonstration der Funktionalität des *ABIC2*, Repräsentation durch Main-Routine
- **HT-Schicht**
  Transponderkommunikation gemäß der Transponderprotokolle
- **Transponder-Schicht**
  Umsetzung der Transponderkommunikation mit Kommandos des *ABIC2*
- **Transceiver-Schicht**
  Kommunikation zwischen Controller und *ABIC2* über SPI oder LIN
- **Controller-Schicht**
  Steuerung der Module und GPIOs des Controllers

Ziel der modularen Architektur ist die Unabhängigkeit der Softwaremodule der einzelnen Transponder von denen des *ABIC2*. Die Transponder-Schicht überführt die allgemeinen Sende- und Empfangskommandos der Transponderkommunikation in die

spezifischen *ABIC2*-Kommandos. Dardurch wird beispielsweise eine Integration eines weiteren LF-Transceivers ohne Änderung der HT-Schicht möglich. Gleichzeitig stellt die Transponder-Schicht eine definierte Schnittstelle für Aufrufe der übergeordneten HT-Schicht zur Verfügung. Eine Verzweigung auf die verschiedenen Transponderprotokolle erfolgt ausschließlich in der Anwendungsschicht und unabhängig von den unteren Schichten.

Abbildung 4.1 visualisiert das Schichtenprinzip und verdeutlicht das Aufrufschema. Jede Schicht, mit Ausnahme der Transponder-Schicht, ruft Funktionen der Controller-Schicht, z.B. die Warteroutine, auf. Aus Gründen der Übersichtlichkeit wurde auf diese Verbindungen sowohl innerhalb dieser Abbildung als auch in den folgenden Abbildungen der Schichtenmodelle verzichtet.

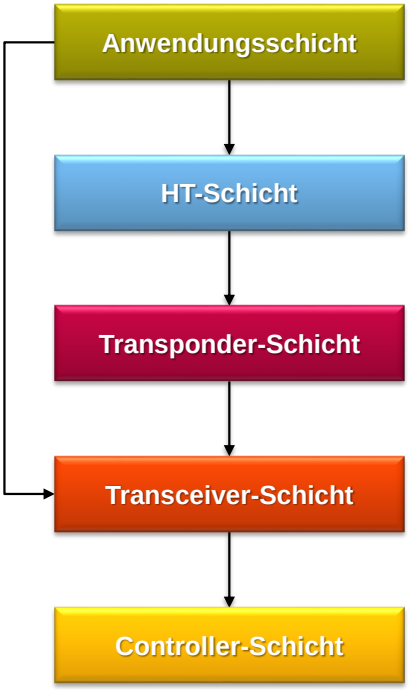

Abbildung 4.1: Schichtenmodell der Software des *LPC93x-Designs*

Ein vollständiger Aufruf von oben nach unten entspricht einer Kommunikation zwischen Basisstation und Transponder. Aufrufe der Anwendungsschicht direkt zur Transceiver-Schicht sind Konfigurationsanweisungen an den *ABIC2*. Für die Transceiver-Schicht existiert in der Architektur eine zusätzliche Aufteilung in High-Level und Low-Level. Während auf dem Low-Level das *ABIC2*-Kommandoset implementiert ist, sind umfangreiche Konfigurationsanweisungen, bestehend aus mehreren Kommandos, auf dem High-Level implementiert. Folgende Abbildung 4.2 erweitert Abbildung 4.1 um die Aufteilung der Transceiver-Schicht.

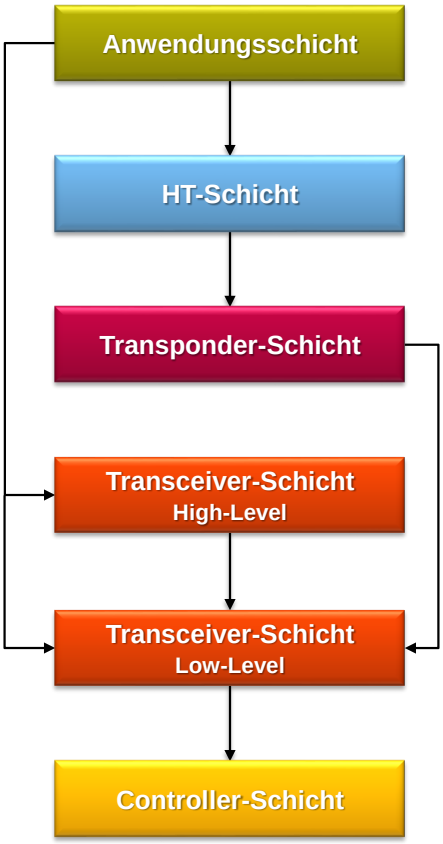

Abbildung 4.2: Erweitertes Schichtenmodell der Software des *LPC93x-Designs*

Programmablauf

Die Anwendung folgt einem zyklischen Vorgehen. Nach der Initialisierung des Controllers verbleibt die Anwendung in einer Endlosschleife. In dieser erfolgt eine Aktivierung und Initialisierung des *ABIC2*, gefolgt von einer Authentifikation mit dem Transponder sowie, je nach gewähltem Transponderprotokoll, weiteren Transponderkommandos. Nach abgeschlossener Transponderkommunikation wird der *ABIC2* deaktiviert und alle relevanten Informationen über RS232 ausgegeben. Danach beginnt der Zyklus von vorn.

Abbildung C.1 visualisiert den Ablauf auf dem *LPC93x-Design* vereinfacht mithilfe eines Aktivitätsdiagramms. Innerhalb eines Zyklus werden folgende Programmablaufphasen unterschieden.

- Initialisierungsphase - Anfang der Endlosschleife bis Aktivierung *ABIC2*
- Betriebsphase - Aktivierung *ABIC2* bis Deaktivierung *ABIC2*
- Ausgabephase - Deaktivierung *ABIC2* bis Ende der Endlosschleife

Diesem Ablauf geht die Wahl des Transponderprotokolls und der verwendeten Schnittstelle zwischen Controller und *ABIC2* voraus. Mithilfe von vorbereiteten compilierten HEX-Dateien muss der steuernde μC dementsprechend geflasht werden. Innerhalb des Quelltextes wurde diese Auswahl aufgrund von Speichermangel mithilfe von „Compile-Flags“ realisiert. Ein Wechsel während des Betriebs ist nicht möglich. Die Wahl der Schnittstelle ist darüber hinaus noch mit einem Jumper auf dem *ABIC2*-CEB abgesichert.

Nach jedem Zyklus kann der Programmablauf durch Eingabe eines 'a' auf einem verbundenen Terminal-Programm unterbrochen werden. Eine vorbereitete Routine ermöglicht das Setzen der *ABIC2*-Register mit selbstgewählten Werten durch eine Eingabe auf der Konsole.

Anhang E zeigt Screenshots der zyklischen Konsolenausgabe einer erfolgreichen sowie einer fehlgeschlagenen Transponderkommunikation des *LPC93x-Designs*.

4.1.2 Beschreibung der Hardware des S32-Designs

Der *ABIC2* wird in diesem Projekt auf einem speziell entwickelten CEB betrieben. In Abbildung A.2 ist dieses Board dargestellt.

Die Verbindungen zwischen beiden Platinen wird, mit Ausnahme der LIN-Schnittstelle, über die Stift- bzw. Buchsenleisten, das sogenannte „Shield“, realisiert. Durch ein Aufstecken des *ABIC2*-Boards, dargestellt in Abbildung A.3, entsteht eine physikalische Verbindung der einzelnen Pins.

Die Abbildung A.4 zeigt das vollständige Referenzdesign mit Beschriftung der Komponenten.

Jeder μC-Pin des Shields kann als GPIO oder mit einer Sonderfunktion konfiguriert werden. Dabei besitzt jeder Pin maximal fünf fest definierte Sonderfunktionen. Eine freie Zuweisung ist nicht möglich. Die verwendeten Softwaremodule und GPIOs des *S32K144* sind in diesem Projekt demnach zu großem Teil durch die gegebenen Hardwareverbindungen zum *ABIC2*-CEB vorgegeben.

4.1.3 Anforderungsdefinition

Die Anforderungen gliedern sich in drei Teilbereiche.

- Portierung der Controller-Schicht
- Funktionale Erweiterung
- Globale Anpassungen

Portierung der Controller-Schicht

Zunächst werden Anforderungen an die Controller-Schicht abgeleitet und Module ausgewählt. Eine Übersicht der verwendeten Pins des *S32K144* sowie deren Position auf dem CEB gibt Anhang B.

- Timer

  Im Softwareprojekt des *LPC93x-Designs* wurde für eine Wartefunktion und für einen Timeout-Interrupt ein gemeinsamer Timer mit einer Auflösung von 35 μs und einem Maximalwert von 2,3 s verwendet. Im *S32K144* sind verschiedene Timer-Module integriert. Die Wahl wird durch keinerlei Bedingung beschränkt.

- Kommunikation mit dem *ABIC2* über SPI

  Die Anforderungen an die SPI-Kommunikation wird durch die Spezifikation im Datenblatt vorgegeben. Für weitere Informationen sei auf [14, Seite 25] verwiesen. Das *ABIC2*-CEB gibt die Nutzung des Moduls *LPSPI0*[18] vor.

- Kommunikation mit dem *ABIC2* über LIN

  Die Anforderungen an die LIN-Kommunikation wird durch die Spezifikation im Datenblatt vorgegeben. Für weitere Informationen sei auf [14, Seite 26 ff. und Seite 46] verwiesen. Durch die Hardware ist die Nutzung des Moduls *LPUART2*[19] in Verbindung mit dem *TJA1027* vorgegeben.

- Kommunikation mit dem Benutzer-PC

  Die Kommunikation erfolgt auf Basis von ASCII-Zeichen mit einem Terminal-Programm auf Seiten des Benutzers. Das Modul *LPUART1*[20] bietet die Möglichkeit einer Kommunikation über die USB-Schnittstelle unter Verwendung des OpenSDA.

- Integration in das Schichtmodell

  Im Sinne einer modularen Architektur muss die Controller-Schicht klare Schnittstellen für einen Aufruf von übergeordneten Schichten bereitstellen, welche weiterführende Möglichkeiten mit dem Controller aufzeigen. Direkte Registeranweisungen dürfen nur innerhalb der Controller-Schicht verwendet werden.

[18] *S32K144-Modul „**L**ow **P**ower **S**erial **P**eripheral **I**nterface **0**“*,
für weiterführende Informationen sei auf [20, Seite 1433 ff.] verwiesen
[19] *S32K144-Modul „**L**ow **P**ower **U**niversal **A**synchronous **R**eceiver/**T**ransmitter **2**“*,
für weiterführende Informationen sei auf [20, Seite 1535 ff.] verwiesen
[20] *S32K144-Modul „**L**ow **P**ower **U**niversal **A**synchronous **R**eceiver/**T**ransmitter **1**“*,
für weiterführende Informationen sei auf [20, Seite 1535 ff.] verwiesen

Funktionale Erweiterung

Folgende Anforderungen wurden in Zusammenarbeit mit der aufgabenstellenden Firma erarbeitet.

- Entfernung der „Compile-Flags“ für die Wahl des Transponderprotokolls

  Im Betrieb des *S32-Designs* muss das verwendete Transponderprotokoll während der Laufzeit auswählbar sein. Alle „Compile-Flags“ sowie die damit verbundenen Betriebsmodi müssen entfernt werden.

- Entfernung der „Compile-Flags“ - SPI / LIN

  Im Betrieb des *S32-Designs* muss eine Wahl zwischen SPI und LIN ohne Neukompilieren möglich sein. Durch einen Jumper, welcher je nach Schnittstelle ein definiertes Spannungslevel an einen GPIO des *S32K144* gibt, muss die Auswahl automatisch erkannt und umgesetzt werden. Das Umstecken des Jumpers bei laufendem Betrieb ist nicht empfohlen und muss innerhalb der Software abgefangen werden.

- Konfiguration der Geschwindigkeit - SPI / LIN

  Die Taktrate der SPI-Schnittstelle und die Baudrate der LIN-Schnittstelle müssen bei Initialisierung des jeweiligen Softwaremoduls einstellbar sein. Über Makros müssen vordefinierte Konfigurationen einen Überblick über die Möglichkeiten des *ABIC2* geben.

- LEDs auf dem *ABIC2*-CEB

  Auf dem für das *S32-Design* entwickelten *ABIC2*-CEB existieren zwei LEDs für die Anzeige des Ergebnisses der Transponderkommunikation. Die LEDs müssen über GPIOs des *S32K144* angesteuert werden.

- Messung der Zeit für eine Authentifizierung

  Die Zeit für eine Authentifizierung muss gemessen und ausgegeben werden.

Globale Anpassungen

Im Zuge der Portierung bietet sich eine Anpassung des gesamten Softwareprojektes an. Folgende Aufstellung konkretisiert die Richtlinien in [15] und definiert Anforderungen.

Alle Dateien der HT-Schicht werden für die Prototyp-Version 0.9, welche Ziel dieses Projektes ist, nur für eine Gewährleistung der Funktionalität angepasst. Eine detaillierte Übersicht über die Behandlung einzelnen Quelldateien gibt Anhang D.

- Projekt- und Ordnerstruktur

  Das Softwareprojekt des *S32-Designs* muss der Struktur in [15, Seite 3 ff.] folgen. Die Applikation inkludiert folgende Komponenten.

  - HT2
  - HT3
  - HTAES
  - HTPRO
  - ABIC2
  - S32

- Konventionen zur Namensgebung

  Die Namen von Dateien und Funktionen müssen sich an den Richtlinien in [15, Seite 5] orientieren. Für die Controller-Schicht und für neue Dateien der Transceiver-Schicht gelten folgende Musterschablonen.

  - Dateinamen
    - S32_<Modulname>_<Anwendung in diesem Projekt>
      z.B. *S32_LPUART2_lin.c*
  - Funktionsnamen
    - S32_<Modulname>_<Funktionsname>
      z.B. *S32_LPUART2_writeByte()*

- Lizenz

  Jede Quelldatei muss mit dem aktuellen NXP-Disclaimer beginnen.

- Programmierstil

  Der Programmierstil muss sich an [16] orientieren. Darüber hinaus muss auf eine umfangreiche Kommentierung und Einheitlichkeit geachtet werden. Die Programmierung muss in englischer Sprache erfolgen.

- Dokumentation innerhalb des Quelltextes mit Doxygen[21]

  Innerhalb des Quelltextes muss sich die Kommentierung zum Zwecke der Dokumentation an [15, Seite 8] orientieren. Die vorhandene Dokumentation auf der Transponder-, Transceiver- und Anwendungsschicht muss in den Doxygen-Stil überführt werden. Die Doxygen-Dokumentation muss in englischer Sprache erfolgen.

- Datentypen

  Die Portierung macht eine Erneuerung der Definition der Datentypen in der Headerdatei *types.h* notwendig.

4.2 Softwarearchitektur

4.2.1 Anpassungen des Schichtenmodells

Eine parallele Implementierung von SPI und LIN erfordert eine Umstrukturierung der vorhandenen Softwarearchitektur. Für beide Schnittstellen existiert dasselbe Kommandoset im Low-Level der Transceiver-Schicht. Da im *LPC93x-Design* stets nur der Code einer der beiden Schnittstellen kompiliert wurde, tragen die Funktionen identische Namen und Übergabeparameter. Ein Aufruf der übergeordneten Schichten erfolgt ohne Wissen der genutzten Schnittstelle. Diese Charakteristik muss beibehalten werden.

Mithilfe einer globalen Variable, welche in Abhängigkeit des Spannungslevels am Pin *PTB10*[22] die verwendete Schnittstelle speichert, kann bei Aufruf eines *ABIC2*-Kommandos die entsprechende Schnittstelle angesprochen werden.

Das Beschreiben der Variable erfolgt nur einmal vor Beginn der Endlosschleife des Programmablaufs gemäß Unterabschnitt 4.1.1. Eine Veränderung der Jumperposition wird über einen Interrupt detektiert und fordert den Nutzer zu einem Reset des Controllers auf.

Die Abfrage der Variable erfolgt bei jedem Kommandoaufruf auf einer neu implementierten Hardwareabstraktionsschicht[23] zwischen High-Level und Low-Level der Transceiver-Schicht. Abbildung 4.3 zeigt das angepasste Schichtenmodell. Die ursprünglich genutzten Funktionsnamen rufen nun Funktionen des HALs auf, welche die

[21] Sinngemäß nach [7] ein Dokumentationssystem für verschiedene Programmiersprachen, u.a. C, Java. Doxygen erstellt eine Dokumentation aus kommentierten Quelltexten in u.a. HTML, für weitere Informationen sei auf die Quelle verwiesen

[22] *PTB10* ist über das Shield mit dem Signal *VIOSense* des *ABIC2*-CEB verbunden. Über einen Jumper auf dem *ABIC2*-CEB lässt sich die verwendete Schnittstelle wählen.

[23] Im Folgenden bezeichnet mit HAL, engl. **H**ardware **A**bstraction **L**ayer

entsprechenden Funktionen des SPI- oder LIN-Kommandosets aufruft. Diese werden mit dem Präfix *SPI_* oder *LIN_* versehen.

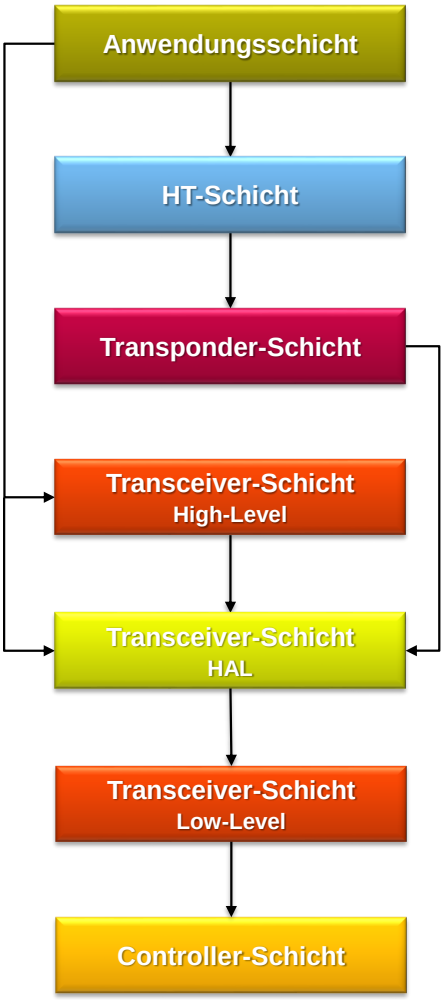

Abbildung 4.3: Schichtenmodell der Software des *S32-Designs*

4.2.2 Anpassungen des Programmablaufs

Das Wechseln des verwendeten Transponderprotokolls erfordert keine Umstrukturierung des Schichtenmodells, da eine Verzweigung auf die einzelnen Transponder lediglich in der Main-Routine erfolgt. Im Programmablauf müssen die Verzweigungen, welche momentan durch den Präprozessor erfolgen, mittels *switch-case*-Anweisungen modelliert werden. Transponderspezifische Größen, wie beispielsweise die Umsetzung des Zeitverhaltens, wurden im *LPC93x-Design* durch eine Variable bzw. ein Makro repräsentiert und in Abhängigkeit des gewählten Transponderprotokolls beschrieben bzw. definiert. Im *S32-Design* werden für jeden Transponder eigene Variablen

und Makros angelegt. Eine Unterscheidung gewährleisten die Präfixe *HT2_*, *HT3_*, *HTAES_* und *HTPRO_*.

Der Programmablauf des *S32-Designs* ist in Abbildung C.2 dargestellt. Eine globale Variable repräsentiert das verwendete Transponderprotokoll. Eine Initialisierung in Abhängigkeit dieser Variable gewährleistet zu Anfang des Kommunikationszyklus die richtigen Rahmenbedingungen auch nach Änderung des Transponderprotokolls. Der Onboard-Taster *SW3* des *S32K144*-CEBs wird für ein Umschalten auf das nächste Transponderprotokoll vorbereitet. Der durch den Taster geworfene Interrupt ändert die globale Variable.

Die bestehende Möglichkeit des Setzens der *ABIC2*-Register über eine Konsoleneingabe wird entfernt. Im *S32-Design* kann eine Veränderung der Registerwerte mithilfe der zu entwickelnden Benutzerschnittstelle veranlasst werden.

4.2.3 Auswahl des Timer-Moduls

Gemäß den Anforderungen wird die Wahl des Timer-Moduls nicht durch die Hardware vorgegeben. Der *S32K144* besitzt mehrere Timer-Module unterschiedlicher Größe und mit verschiedenen Möglichkeiten der Konfiguration. Das Modul *LPIT0*[24] eignet sich aufgrund einer Kanalgröße von 32 Bit für dieses Projekt. Die Nutzung des *LPIT0* garantiert Zeitspannen über 100 s mit einer sehr genauen Auflösung von unter einer μs. Durch die vier Kanäle des Moduls werden darüber hinaus alle Anforderungen an den Timer mit demselben Modul erfüllt.

4.3 Programmierung und Implementierung

Die Quelldateien lassen sich hinsichtlich der Portierung in verschiedene Kategorien der Bearbeitung aufteilen. Diese Kategorien beschreiben den Freiheitsgrad in der Umsetzung und ordnen den einzelnen Dateien Anforderungen aus dem Teilbereich 'Globale Anpassungen' zu. In Anhang D ist diese Kategorisierung der Quelldateien dargelegt.

Für Einsicht in den Code sei auf die Doxygen-Dokumentation auf der beigelegten CD verwiesen. Detaillierte Informationen sind in Anhang I zu finden.

[24] *S32K144-Modul „**L**ow **P**ower **I**nterrupt **T**imer **0**“*,
für weiterführende Informationen sei auf [20, Seite 1361 ff.] verwiesen

4.4 Funktionstests

Es erfolgt eine Einteilung der Quelldateien in zwei Kategorien. Die Reihenfolge der Tests entspricht dem Schichtenmodell von unten nach oben.

- Kategorie 1: Quelldateien unterhalb des HALs der Transceiver-Schicht inklusive des HALs
  - Testen der C-Funktionen mit dafür erstellten Testfunktionen anhand von Oszilloskopaufnahmen oder innerhalb des Debug-Modus der IDE
- Kategorie 2: Quelldateien oberhalb des HALs der Transceiver-Schicht
  - Testen mit Blick auf die Gesamtfunktionalität anhand der Konsolenausgaben, dargestellt in Anhang E

Eine Übersicht über Kategorie 1 gibt die Exceltabelle auf der beigelegten CD.

Kategorie 2 wird durch das Herstellen einer erfolgreichen Transponderkommunikation mit allen Transponderprotokollen unter Verwendung von LIN sowie SPI geprüft. Mit einem Terminal-Programm muss eine stabile und fehlerfreie Ausgabe, wie dargestellt in Abbildung E.1, bis zu einer Reichweite von mehreren Zentimetern reproduzierbar sein.

4.5 Auswertung

Die definierten Anforderungen wurden mithilfe der beschriebenen Tests verifiziert. Sowohl die portierte Controller-Schicht als auch alle funktionalen Erweiterungen wurden erfolgreich implementiert.

5 Entwicklung einer grafischen Benutzerschnittstelle

Eine grafische Benutzerschnittstelle ist eine Erweiterung des *LPC93x-Designs*, welche im Rahmen dieses Projektes entwickelt wird. Die GUI ermöglicht auch ohne IDE des Controllers eine grundlegende Konfiguration des *S32-Designs*. Hintergrund und Definition der Anforderungen an das Referenzdesign sind in Abschnitt 3.3 aufgezeigt.

Im folgenden Abschnitt sind Vorüberlegungen und daraus abgeleitete Entscheidungen dargelegt. Der Entwicklungsprozess ist in Abschnitt 5.2 beschrieben.

5.1 Vorbetrachtung

5.1.1 Rahmenbedingungen

Aus den Rahmenbedingungen des Projektes, aufgeführt in Abschnitt 3.2, resultieren folgende Rahmenbedingungen für die Entwicklung einer GUI.

- Es wird nur lizenzkostenfreie Software verwendet. Im Rahmen der Evaluierung dürfen für den Kunden keine Kosten anfallen.
- In [15, Seite 5] wird eine Verwendung der Programmiersprache Java und des Swing-Frameworks[25] empfohlen.
- In [15, Seite 5] wird in Bezug auf das Softwaredesign eine Orientierung am MVC-Konzept[26] empfohlen.

Für die Kommunikation eignet sich auf Seiten des *S32-Designs* das innerhalb der Portierung verwendete *LPUART1*-Modul. Grundlage ist der Aufbau einer seriellen Verbindung mit dem zugehörigen COM-Port des *S32-Designs*. Demzufolge ist ein Zugriff auf die seriellen Schnittstellen des Benutzer-PCs notwendig.

Java ist eine plattformunabhängige Sprache, welche zur Laufzeit in einer virtuellen Umgebung ausgeführt wird. Systemnahe Eigenschaften sind nicht sichtbar und ein Zugriff auf Hardware ist unter Verwendung der Standardbibliotheken zum Zeitpunkt

[25] Java Bibliothek zur Programmierung von GUIs
[26] **M**odel-**V**iew-**C**ontroller-Konzept, Entwurfsmuster objektorientierter Software, siehe 5.1.3

dieses Projektes nicht möglich. Für einen Zugriff auf die serielle Schnittstelle muss auf zusätzliche Bibliotheken zurückgegriffen werden. Dieser Absatz ist sinngemäß aus [30] entnommen.

RXTX ist sinngemäß nach [32] eine Java Bibliothek, welche Klassen für eine serielle und parallele Kommunikation bereitstellt. Verschiedene Quellen, unter anderem [33] und [25], beschreiben ähnliche Anwendungsfälle und geben ausführliche Hilfe mit Beispielprogrammen. Aufgrund der Vorgabe der Verwendung von kostenfreier Software und dem zukünftigen Einsatz in einem kommerziellen Zusammenhang ist eine gründliche Sichtung der Lizenz alternativlos.

Die folgende Beschreibung der Lizenz wurde teilweise und sinngemäß aus [8] übersetzt. *RXTX* ist freie Software, welche unter den Bestimmungen der GNU LGPL[27] Version 2.1 verteilt und modifiziert werden kann. Für eine ausführbare Datei, welche nicht von *RXTX* oder einem Teil davon abgeleitet ist, aber über eine dynamische Verlinkung mit dieser Bibliothek arbeitet, erfolgt eine Einteilung in die Kategorie „work that uses that library“. Für diese Kategorie definieren die Lizenzbedingungen von *RXTX*, einsehbar in Anhang F, eine zusätzliche Ausnahme.

Diese Ausnahme gibt die Freiheit, eine ausführbare Datei, welche faktisch ein Derivat der Bibliothek ist, unter eigenen Bestimmungen zu kopieren und zu vertreiben. Voraussetzung ist das Mitliefern einer vollständigen Kopie des Quellcodes von *RXTX* in der verwendeten Version. Gemäß GNU LGPL muss darüber hinaus ein Urheberrechtsvermerk sowie ein Haftungsausschluss zur Verwendung der Bibliothek innerhalb der GUI dargestellt werden.

Zusammenfassend erfüllt *RXTX* die Rahmenbedingungen für die Entwicklung einer GUI und kann, solange an der Bibliothek keine Änderungen vorgenommen werden, aus lizenzrechtlicher Sicht in diesem Projekt verwendet werden.

Der in Unterabschnitt 4.1.1 beschriebene modulare Ansatz, welcher eine Implementierung weiterer Basisstation-ICs, wie dem Vorgängermodell *ABIC*, ermöglichen soll, gilt auch für die GUI.

In Anlehnung wird der Name „*Base Station User Interface*“, kurz *BUI*, gewählt und in der Thesis verwendet.

5.1.2 Verwendung RXTX – eine Machbarkeitsstudie

Vor dem eigentlichen Entwicklungsprozess wird durch eine Machbarkeitsstudie die Realisierbarkeit der Kommunikation zwischen *S32-Design* und der Java IDE Eclipse mithilfe der *RXTX*-Bibliothek überprüft. Es wird angenommen, dass mit einem erfolgreichen Senden und Empfangen von Zeichenketten verschiedener Länge unter einer

[27] GNU **L**esser **G**eneral **P**ublic **L**icense, Lizenz zur Verwendung von Bibliotheken

Geschwindigkeit von 115200 Baud eine Realisierbarkeit gewährleistet ist. Der Fokus dieses Tests liegt nur auf dem Kommunikationsvorgang, da davon ausgegangen werden kann, dass sowohl für die Vorbereitung der zu sendenden Daten als auch für die Verarbeitung der empfangenen Daten an beiden Endpunkten programmiertechnische Lösungen gefunden werden.

Konkret wird mithilfe des Beispielprogramms aus [31] eine Verbindung zwischen der Konsole der Java IDE und dem *S32-Design* aufgebaut. Mithilfe einer Echo-Funktion auf Seiten des *S32-Designs* können gesendete und empfangene Nachrichten innerhalb der Konsole verglichen werden. Folgende Abbildung 5.1 zeigt einen im Zuge der Machbarkeitsstudie erstellten Screenshot.

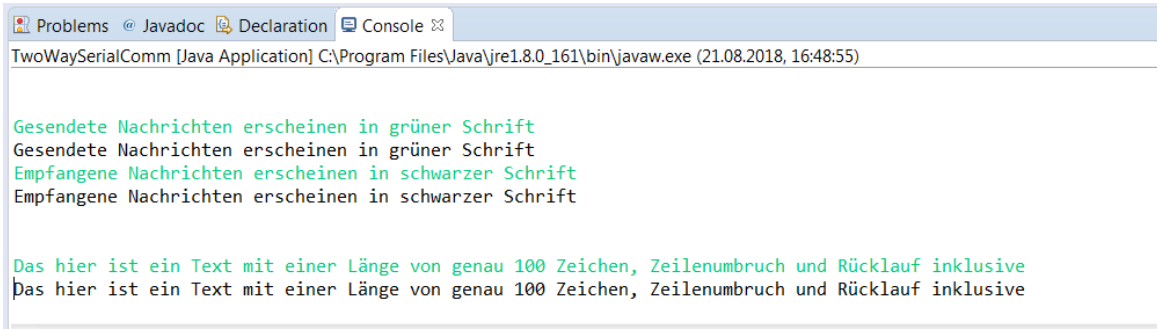

Abbildung 5.1: Konsole der Eclipse Java IDE, Screenshot

Der Test zeigt, dass *RXTX* auch funktional für einen Einsatz in diesem Projekt geeignet ist.

5.1.3 Model-View-Controller-Konzept

Dieser Unterabschnitt gibt eine Einführung in das verwendete MVC-Konzept für den Softwareentwurf von grafischen Benutzeroberflächen. Die Einführung basiert sinngemäß auf [6].

Das MVC-Konzept ist ein Entwurfsmuster zur Strukturierung von Software in der objektorientierten Programmierung. Durch die weitgehende Trennung von Daten-Modell und dessen grafischer Repräsentation wird Übersichtlichkeit, Wartbarkeit und Wiederverwendbarkeit erhöht. Dabei sind drei architektonische Bestandteile zu unterscheiden.

- Das Modell (engl. model) beschreibt die Datenquelle der Anwendung. Die Daten sind unabhängig vom Erscheinungsbild definiert.
- Die Präsentation (engl. view) beschreibt die Darstellung der Daten mithilfe einer grafischen Oberfläche. Die Art und Weise der Darstellung wird nicht vom Modell beeinflusst.
- Die Steuerung (engl. controller) ist die vermittelnde Komponente zwischen Modell und Präsentation. Sie verarbeitet Nutzereingaben, aktualisiert die Daten des Modells und beinhaltet darüber hinaus die Programmlogik.

Abbildung 5.2 visualisiert die drei Bestandteile des Konzeptes und deren Beziehungen zueinander.

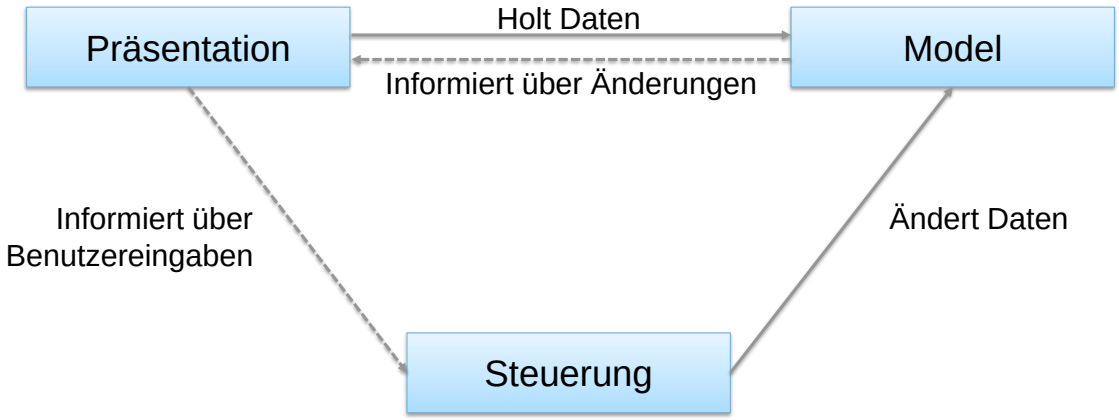

Abbildung 5.2: MVC-Konzept, sinngemäß nach [6]

Der Controller ändert die Daten im Modell gemäß der Programmlogik oder in Folge einer Benutzereingabe. Diese wird dem Controller durch die Präsentation mitgeteilt.

Das Model informiert die Präsentation über eine Änderung der Daten. Nach Abholen der Daten visualisiert die Präsentation diese Änderung ohne direkte Aktion des Controllers.

5.1.4 V-Modell

Dieser Unterabschnitt gibt eine Einführung und erläutert die einzelnen Phasen sowie deren Anwendung in diesem Projekt. Die allgemeinen Beschreibungen basieren sinngemäß auf [28].

Das V-Modell ist ein Entwicklungsprozessmodell für Soft- und Hardwareentwicklung. Während der konstruktiven Phasen erfolgt bereits eine Spezifizierung der zugehörigen Tests. Diese Charakteristik erhöht die Wahrscheinlichkeit, unvollständige Spezifikationen in frühen Stadien der Entwicklung zu erkennen. Die Anordnung der Phasen, visualisiert in Abbildung 5.3, gibt dem Modell seinen Namen. Auf der linken Seite sind die konstruktiven Phasen zu erkennen, welche auf gleichem Grad der Detaillierung den entsprechenden Testphasen gegenübergestellt sind. Lediglich die untere Spitze repräsentiert die eigentliche Programmierung.

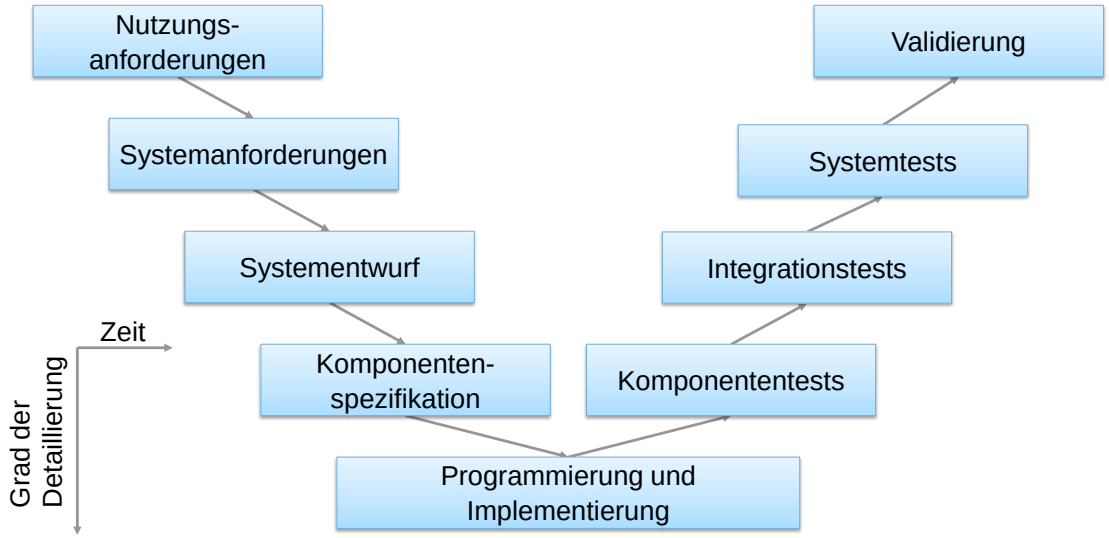

Abbildung 5.3: V-Modell, sinngemäß nach [28]

Definition der Nutzungsanforderungen

Nutzungsanforderungen beschreiben notwendige Tätigkeiten des Nutzers in der *BUI*, um ein bestimmtes Handlungsziel zu erreichen. Sie stellen keine konkrete Beschreibung einer Lösung dar.

Eine Validierung der Nutzungsanforderungen erfolgt in der letzten Phase des V-Modells, indem die ausformulierten Anforderungen als erfüllt oder nicht erfüllt eingestuft werden.

Definition der Systemanforderungen

Systemanforderungen beschreiben die Umsetzung der Nutzungsanforderungen durch das System. Dabei wird das System wie eine „Black-Box“ betrachtet, um externe Schnittstellen zu definieren. Eine detaillierte Beschreibung der technischen Umsetzung fällt nicht in diese Phase.

In diesem Projekt werden die Systemanforderungen aufgrund der gegebenen Systemumgebung und der überschaubaren Größe des Projektes mit den Nutzungsanforderungen kombiniert. Die Nutzungsanforderungen werden mit Blick auf Schnittstellen und durch konkrete Umsetzungen auf der grafischen Oberfläche präzisiert. Eine Prüfung der Spezifikation erfolgt in einer kombinierten Testphase, welche 'Systemtests und Validierung' genannt wird.

Grobarchitektur, technischer Systementwurf

Die Beschreibung der Begriffe 'System-Architektur' und 'Software-Architektur' basiert sinngemäß auf [27] bzw. [26]. Da es sich bei diesem Projekt um keine „standalone“ Software handelt, müssen beide Prozesse unterschieden werden. Um eine bessere Trennung zu ermöglichen, wird im Folgenden für 'Software-Architektur' die Bezeichnung 'Software-Design' und für 'Komponente einer Software-Architektur' die Bezeichnung 'Modul' verwendet.

In dieser Phase des V-Modells ist die technische Umsetzung der definierten Anforderungen beschrieben. Die System-Architektur beschreibt, aus welchen Komponenten das Endprodukt zusammengesetzt ist und wie diese Komponenten über Schnittstellen miteinander in Beziehung stehen. Das Software-Design umfasst den inneren Aufbau eines Softwaresystems mit Identifizierung der Module sowie deren Zusammenspiel und Abhängigkeiten.

Sowohl die *BUI* als auch die Software des *S32-Designs* stellen eigene Softwaresysteme und damit Komponenten des Systems dar. Innerhalb der Systemarchitektur wird die Kommunikation zwischen den öffentlichen Schnittstellen dieser Softwaresysteme betrachtet. Das macht die Entwicklung eines Übertragungsprotokolls notwendig. In diesem Projekt erfolgt das Softwaredesign aufgrund der absehbaren Komplexität vollständig in der Feinarchitektur. Die zur Systemarchitektur zugehörigen Integrationstests prüfen die Kommunikation der öffentlichen Schnittstellen der *BUI* und des *S32-Designs*.

Feinarchitektur, Komponentenspezifikation

In diesem Projekt trennt sich die Feinarchitektur in das Softwaredesign der *BUI* und des *S32-Designs*. Die *BUI* wird in Java unter Anwendung des MVC-Konzeptes programmiert. Neben einem Klassendiagramm ist eine Definition der grafischen Oberflächen und Elemente notwendig. Für das *S32-Design* beschränkt sich die Betrachtung auf die Sende- und Empfangsroutine für eine Kommunikation gemäß dem Übertragungsprotokoll sowie dessen Integration in den Programmablauf. Die zugehörigen Komponententests prüfen, ob die Vorgaben der jeweiligen Spezifikation erfüllt sind.

Programmierung und Implementierung

Die Verwirklichung der Komponentenspezifikationen in Programmcode erfolgt in dieser Phase des V-Modells.

Eine Dokumentation des Codes wird mit Doxygen realisiert und orientiert sich an [15, Seite 8]. Der innerhalb der Portierung verwendete Programmierstil wird in Anbetracht der objektorientierten Sprache adaptiert.

5.2 Entwicklungsprozess unter Anwendung des V-Modells

5.2.1 Definition der Nutzungs- und Systemanforderungen

Die folgenden Anforderungen wurden in Zusammenarbeit mit der aufgabenstellenden Firma im Zuge dieses Projektes formuliert.

(I) Der Nutzer muss die *BUI* innerhalb von 30 Minuten in Betrieb nehmen können.

(II) Der Nutzer muss die *BUI* ohne Installation des S32-Design-Studios in Betrieb nehmen können.

(III) Der Nutzer muss eine Verbindung zwischen *S32-Design* und PC über USB aufbauen können. Die Verbindung erfolgt über die serielle Schnittstelle mit einer Geschwindigkeit von 115200 Baud und 8N1 Protokollparameter (8 Datenbits, 1 Stoppbit, kein Paritätsbit). Der Nutzer muss aus einer Dropdown-Liste verfügbare Ports selektieren und diese Liste aktualisieren können.

(IV) Der Nutzer muss zwischen einem erfolgreichen Verbindungsaufbau sowie zwischen folgenden auftretenden Fehler beim Verbindungsaufbau unterscheiden können:

i) Es existiert kein Port für einen Verbindungsaufbau.

ii) Der selektierte Port existiert zum Zeitpunkt des Verbindungsaufbaus nicht mehr.

iii) Der selektierte Port ist zum Zeitpunkt des Verbindungsaufbaus in Benutzung.

iv) Der selektierte Port ist kein serieller Port.

(V) Der Nutzer muss über eine abgebrochene Verbindung nach einem erfolgreichen Verbindungsaufbau durch ein Popup-Fenster hingewiesen werden und einen erneuten Verbindungsaufbau auslösen können.

(VI) Der Nutzer muss auf eine fehlerhafte Übertragung zwischen *BUI* und *S32-Design* durch ein Popup-Fenster hingewiesen werden.

(VII) Der Nutzer muss nach erfolgreichem Verbindungsaufbau die Transponderprotokolle *HT-2*, *HT-3*, *HT-AES* und *HT-Pro* in einer Dropdown-Liste selektieren und eine Änderung des verwendeten Transponderprotokolls im Betrieb des *S32-Designs* veranlassen können.

(VIII) Der Nutzer muss nach erfolgreichem Verbindungsaufbau selbstgewählte Werte für den geheimen Schlüssel in hexadezimaler Form eingeben und eine Änderung des verwendeten geheimen Schlüssels im Betrieb des *S32-Designs* veranlassen können.

(IX) Der Nutzer muss nach erfolgreichem Verbindungsaufbau selbstgewählte Werte für die Challenge in hexadezimaler Form eingeben und eine Änderung der Challenge im Betrieb des *S32-Designs* veranlassen können.

(X) Der Nutzer muss die Größe des geheimen Schlüssels und der Challenge in Bits erkennen können.

(XI) Der Nutzer muss nach erfolgreichem Verbindungsaufbau selbstgewählte Werte für die Register des *ABIC2* wählen und eine Neubeschreibung dieser Register im Betrieb des *S32-Designs* veranlassen können. Ein Kenntnisstand der Registeradressen darf dabei nicht erforderlich sein.

(XII) Der Nutzer muss nach Änderungen des geheimen Schlüssels, der Challenge oder der *ABIC2*-Registerwerte die Möglichkeit haben, zur Standardkonfiguration zurückzukehren. Ein Kenntnisstand der Standardkonfiguration darf dabei nicht erforderlich sein.

(XIII) Der Nutzer muss auf zu lange Zeichenketten einer hexadezimalen Eingabe hingewiesen werden. Der Hinweis muss optisch erfolgen und einen Rückschluss auf das Eingabefeld zulassen, in dem der Fehler erkannt wurde. Zu kurze Zeichenketten werden bis zur korrekten Länge mit führenden Nullen aufgefüllt.

(XIV) Der Nutzer muss auf semantische Fehler der hexadezimalen Eingabe hingewiesen werden. Der Hinweis muss optisch erfolgen und einen Rückschluss auf das Eingabefeld zulassen, in dem der Fehler erkannt wurde.

(XV) Der Nutzer muss den Standard-Programmablauf des *S32-Designs* einmalig innerhalb der *BUI* starten können.

(XVI) Der Nutzer muss den Programmablauf des *S32-Designs* in Endlosschleife innerhalb der *BUI* starten können.

(XVII) Der Nutzer muss innerhalb der *BUI* veranlassen können, dass der Programmablauf des *S32-Designs* einmalig auf Druck eines Tasters des *S32K144*-CEB startbar ist.

(XVIII) Der Nutzer muss Ausgaben[28] des *S32-Designs* erkennen können. Die Ausgaben müssen innerhalb der *BUI* wie bei Verwendung eines Terminal-Programms visualisiert und aktualisiert werden.

[28] Konsolenausgaben wie dargestellt in Anhang E

(XIX) Der Nutzer muss das Resultat (erfolgreich / nicht erfolgreich) der einzelnen Transponderkommandos (*AUTHENT*, *READ_PAGE*, *WRITE_PAGE*) unabhängig von den Ausgaben des *S32-Designs* erkennen können.

(XX) Der Nutzer muss während Benutzung der *BUI* die Versionsnummer der *BUI* sowie einen Hinweis zu der Verwendung von *RXTX* gemäß Lizenzbedingungen einsehen können.

(XXI) Der Nutzer muss nach erfolgreichem Verbindungsaufbau den verbundene Basisstation-IC und die Versionsnummer des *S32-Designs* einsehen können.

(XXII) Die Nutzung der *BUI* darf das Zeitverhalten der Transponderkommunikation des *S32-Designs* nicht beeinflussen.

Definition der Systemtests und Validierung

Einen Überblick über die Rahmenbedingungen zur Prüfung der Anforderungen gibt die Exceltabelle auf der beigelegten CD. Detaillierte Informationen sind in Anhang I zu finden.

5.2.2 Grobarchitektur - Systemarchitektur

Systemübersicht

Abbildung 5.4 zeigt das System im Sinne der Grobarchitektur. Aufgrund der Vollständigkeit ist der Nutzer und dessen Schnittstellen zum PC mit dargestellt. Die beiden Systemkomponenten *BUI* und *S32-Design* sind über die serielle Schnittstelle miteinander verbunden. Für diese Kommunikation bedarf es der Entwicklung eines Übertragungsprotokolls.

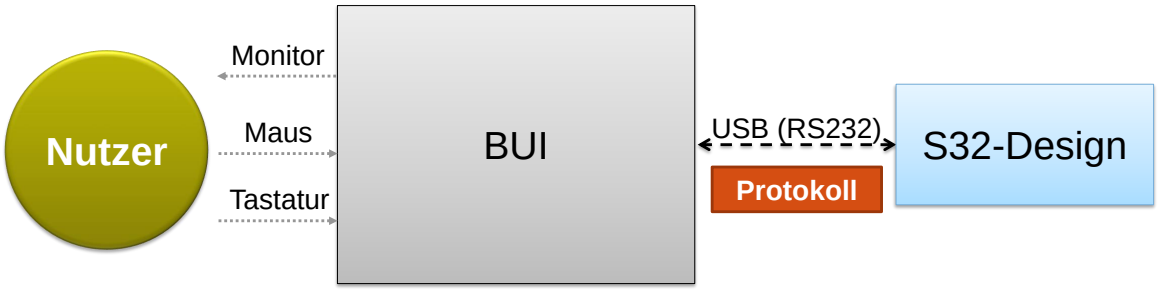

Abbildung 5.4: Systemübersicht Grobarchitektur

Entwicklung eines Übertragungsprotokolls

Die Übertragung erfolgt hardwareseitig über die USB-Verbindung. Durch OpenSDA kann diese Verbindung softwareseitig wie eine serielle Schnittstelle angesprochen werden. Die Parameter sind gemäß (III) festgelegt. Das zu entwickelnde Protokoll definiert den zeitlichen Ablauf und die Interpretation der übertragenen Daten an beiden Endpunkten.

Mit Blick auf die Nutzungsanforderungen empfiehlt sich eine Trennung in zwei Zustände, welche den Status der Verbindung repräsentieren.

- Zustand 1 (Gelb): Es besteht keine aktive Verbindung. Es wurde noch keine Verbindung aufgebaut oder die Verbindung ist abgebrochen.
- Zustand 2 (Blau): Es besteht eine aktive Verbindung. Ein erfolgreicher Verbindungsaufbau ist vorangegangen.

Der Verbindungsaufbau zum entsprechenden COM-Port gewährleistet die Punkt-zu-Punkt Verbindung. Eine Identifikation des *S32-Designs* sowie weiterführend eine Versionsunterscheidung erfordert zusätzlich einen expliziten Verbindungsaufbau auf Applikationsebene. Abbildung 5.5 verdeutlicht den Zustandsübergang. Der zeitliche Ablauf ist von oben nach unten unter farblicher Trennung der Zustände dargestellt. Horizontale Pfeile beschreiben eine Interaktion des Nutzers an der *BUI* bzw. eine Kommunikation zwischen *BUI* und *S32-Design*.

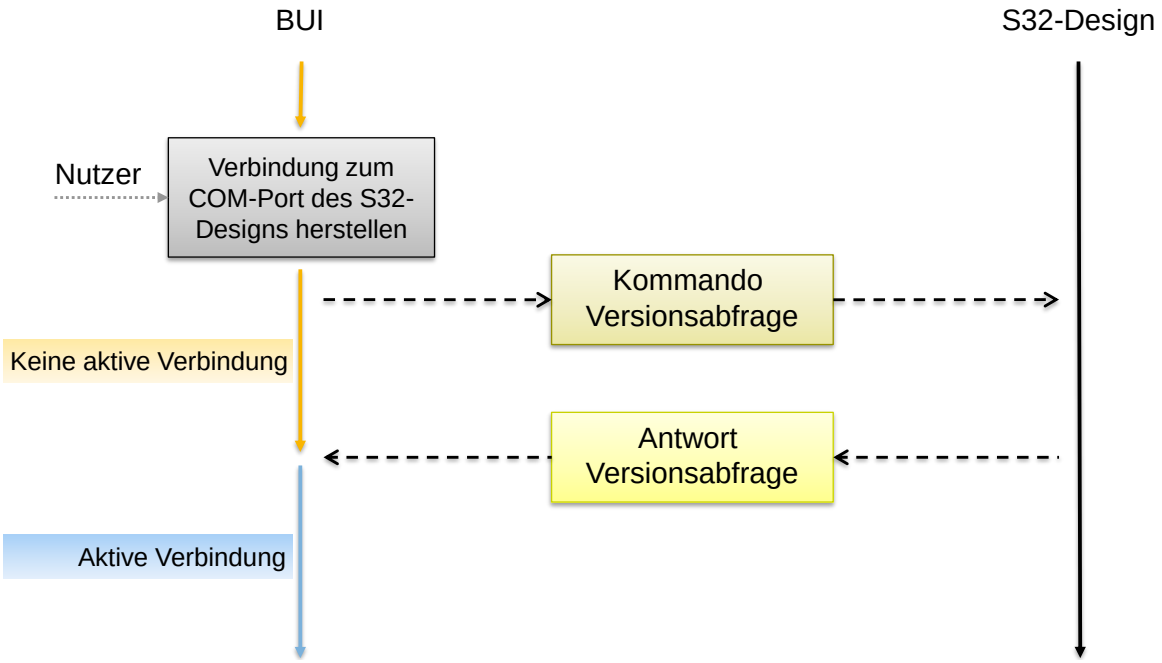

Abbildung 5.5: Ablauf eines expliziten Verbindungsaufbaus

Gemäß (VI) muss das *S32-Design* fähig sein, einen Übertragungsfehler zu erkennen und das Auftreten der *BUI* mitzuteilen. Eine Einteilung der Antwortmöglichkeiten des *S32-Designs* ist somit folgendermaßen möglich.

- Antwortmöglichkeit 1: Kommando erfolgreich empfangen (ACK[29])
- Antwortmöglichkeit 2: Kommando nicht erfolgreich empfangen (NAK[30])
- Antwortmöglichkeit 3: Keine Antwort

Mit den definierten Zuständen und Antwortmöglichkeiten lässt sich ein Zustandsdiagramm für die *BUI* entwerfen, welches in Abbildung 5.6 dargestellt ist. Jegliche Kommunikation wird von der *BUI* initiiert. *Kommando xxx* beschreibt ein beliebiges Kommando. Das Abfragen der Version impliziert den vorangegangenen Verbindungsaufbau zu einem COM-Port. Die maximale Wartezeit auf eine Antwort wird innerhalb der Feinarchitektur definiert.

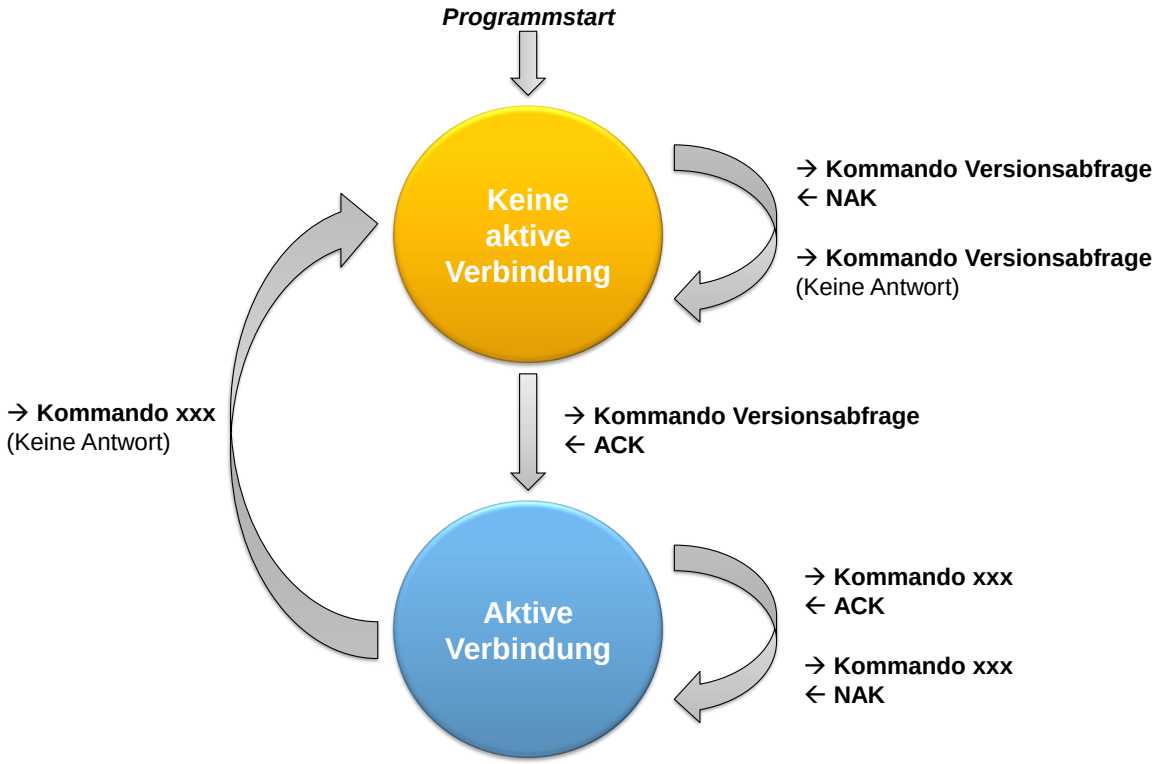

Abbildung 5.6: Zustandsdiagramm *BUI*

[29] Positive Bestätigung, engl. **Ack**nowledgement
[30] Negative Bestätigung, engl. **N**egative **Ack**nowledgement

Die zwei Zustände werden durch unterschiedliche grafische Oberflächen der GUI repräsentiert. Eine positive Bestätigung der Versionsabfrage geht mit der Übertragung einer Kennung und einer Version des verbundenen Basisstation-ICs einher. Dadurch wird eine Zuordnung angepasster Oberflächen zum angeschlossenen Design möglich. Der Zustandsübergang ist in Abbildung 5.7 in Bezug zu Abbildung 5.6 dargestellt. Diese Verfahrensweise erlaubt einen modularen Aufbau und die Implementierung mehrerer Basisstation-ICs unter verschiedenen Versionen.

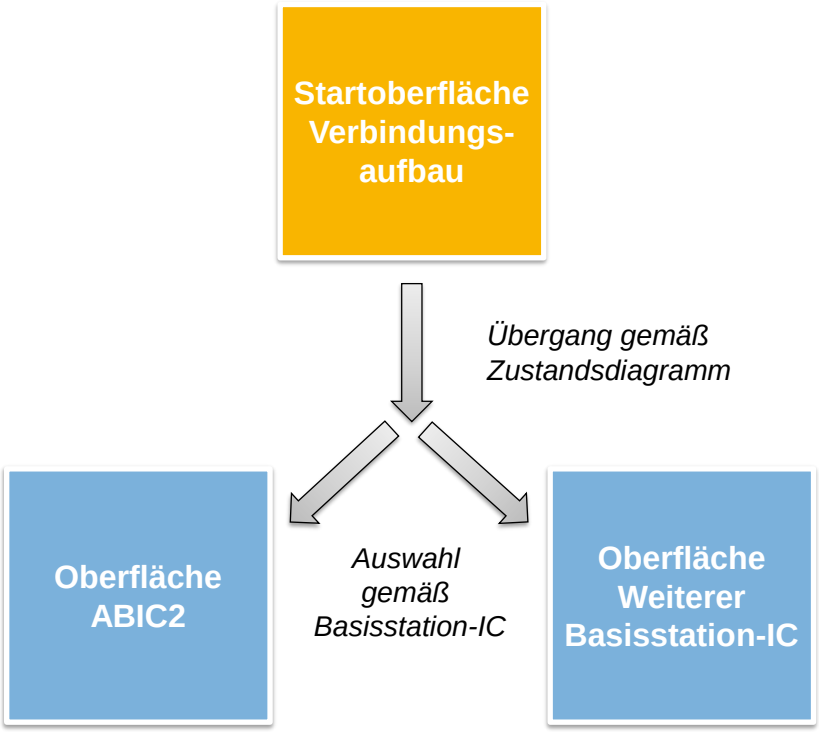

Abbildung 5.7: Zustandsübergang *BUI*

Vor einer Definition von Abläufen und Kommandos muss zunächst die Einordnung in den zeitlichen Programmablauf des *S32-Designs* betrachtet werden. Genannte Programmablaufphasen entsprechen der Definition in Unterabschnitt 4.1.1.

Das verwendete Empfangsmodul besitzt einen FIFO mit einer Größe von 4 Byte. Aufgrund (XXII) kommt eine Verwendung von Interrupts nicht in Frage. Größere Datenmengen erfordern demzufolge eine Routine, welche sich ausschließlich dem Datenempfang widmet. Die Implementierung muss zudem außerhalb der Betriebsphase erfolgen, um (XXII) zu erfüllen. Die Nutzung der Kapazität des FIFOs bietet sich dennoch für die Ankündigung eines Kommandos an.

Abbildung 5.8 verdeutlicht die Übertragung eines Kommandos nach erfolgreichem

Verbindungsaufbau. Der zeitliche Ablauf ist von oben nach unten unter farblicher Trennung der Zustände und Programmablaufphasen dargestellt. Horizontale Pfeile beschreiben eine Interaktion des Nutzers an der *BUI* bzw. eine Kommunikation zwischen *BUI* und *S32-Design*. Die geschweifte Klammer deutet an, dass eine auf eine Nutzereingabe folgende Ankündigung zu einem beliebigen Zeitpunkt während der Betriebs- oder Ausgabephase stattfinden kann. Die Kommandoübertragung erfolgt in der Initialisierungsphase.

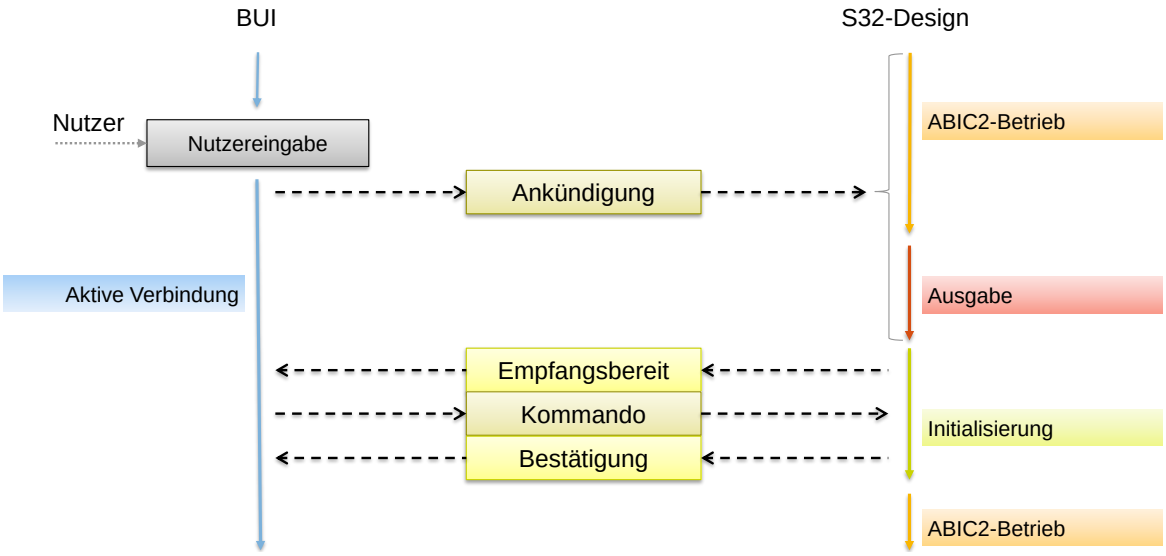

Abbildung 5.8: Ablauf einer Kommandoübertragung

Gemäß (XII) muss eine Rückkehr zur Standardkonfiguration möglich sein. Dies lässt sich realisieren, indem das *S32-Design* stets mit der Standardkonfiguration initialisiert wird. Mitgeteilte Änderungen durch die *BUI* werden darauffolgend überschrieben. Da die Initialisierung transponderbezogen erfolgt, muss die Wahl des Transponderprotokolls zwingend vorher erfolgen. Die genaue Implementierung ist Teil der Feinarchitektur und in Unterabschnitt 5.2.3 detailliert beschrieben.

Folglich besitzt das *S32-Design* kein Gedächtnis und reagiert unabhängig von vorangegangenen Kommandos. Das wirkt sich direkt auf das Protokolldesign aus. Die Startbedingungen gemäß (XV), (XVI) und (XVI) müssen durch verschiedene Kommandos realisiert werden. Neben der Startbedingung werden stets das Transponderprotokoll und nach Bedarf abweichende Parameter gemäß (VIII), (IX) und (XI) gesendet. Eine Verwendung der Standardkonfiguration wird durch einen nicht mitgeteilten Parameter impliziert.

Bei der Definition des Protokollrahmens erfolgt eine Orientierung an der SPI-

Kommunikation des *JOKERs*, dem Nachfolger des *ABIC2*. Die Dokumentation des Kommandosets ist in [17] einsehbar. Die Kommunikation ist kommandobasiert und wird mit einer CRC-8[31] Prüfsumme abgesichert. Ein vorangestelltes Byte bestimmt die Länge der nachfolgenden Nachricht. Das zweite Byte spezifiziert das Kommando und damit die zu erwartenden folgenden Datenbytes. Das letzte Byte enthält die Prüfsumme. Die Antwort auf ein Kommando folgt der gleichen Struktur und wird um ein Statusbyte erweitert. In Abbildung 5.9 ist der Aufbau von Kommando und Antwort dargestellt.

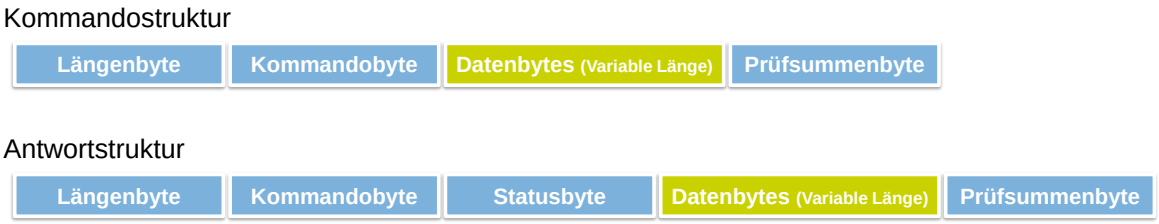

Abbildung 5.9: Nachrichtenstruktur

Längenbyte und Prüfsummenbyte lassen eine Fehlerüberprüfung vor der Auswertung auf applikativer Ebene zu. Das Statusbyte zeigt in diesem Projekt als Ergebnis entsprechend ein ACK oder NAK an. Des Weiteren wird festgelegt, dass sinnwidrige Kommandos vom *S32-Design* mit NAK beantwortet und sinnwidrige Antworten von der *BUI* als NAK interpretiert werden.

Da das *S32-Design* kontinuierlich Daten zur direkten Ausgabe an die *BUI* sendet, müssen die Kommandoantworten zusätzlich eingerahmt werden. Die Zeichen '~' und '@' eignen sich aufgrund der Nichtexistenz innerhalb der Ausgabedaten als Start- und Endmarkierung. Dies erlaubt Protokollkommunikation von Ausgabedaten zu trennen und weiterzuverarbeiten.

Die Definition des Protokollkommandosets auf Basis der beschriebenen Vorüberlegungen ist in einem Foliensatz auf der beigelegten CD dokumentiert. Detaillierte Informationen sind in Anhang I zu finden.

Definition der Integrationstests

Es erfolgt eine Einteilung der Integrationstests in zwei Kategorien. Einerseits wird jeweils die Sende- und Empfangsfunktionalität anhand festgelegter Datenmuster überprüft. Andererseits wird ein Test des Zusammenspiels durch Nachbildung aller Transitionen des Zustandsdiagramms in Abbildung 5.6 überprüft. Um alle Möglichkeiten

[31] **C**yclic **R**edundancy **C**heck-8, Verfahren zur Fehlererkennung mithilfe einer 8 Bit Prüfsumme

abzudecken, werden Übertragungsfehler und ausbleibende Antworten erzwungen. Eine konkrete Übersicht gibt die Exceltabelle auf der beigelegten CD. Detaillierte Informationen sind in Anhang I zu finden.

5.2.3 Feinarchitektur - Softwaredesign

Definition der grafischen Oberflächen und Elemente

Die in der Grobarchitektur beschriebenen Zustände der *BUI* werden durch verschiedene grafische Oberflächen repräsentiert. Folgende Aufstellung ordnet den Anforderungen Elemente auf der Startoberfläche zu. Abbildung 5.10 visualisiert die Beschriftung und Anordnung dieser Elemente.

- Oberfläche 1: Keine aktive Verbindung
  - (III) Anzeige und Auswahl der COM-Ports → *Dropdown-Liste*
  - (III) Aktualisierung der Auswahl der COM-Ports → *Button Aktualisierung*
  - (III) Auslösung des Verbindungsaufbaus → *Button Verbindungsaufbau*
  - (IV) Nutzerkommunikation → *Ausgabefeld*
  - (XX) Anzeige der Lizenzbedingungen → *Menüleiste*

Die Lizenzbedingungen von NXP und von *RXTX* können in einem Zusatzfenster, aufrufbar durch einen Eintrag in der Menüleiste, eingesehen werden.

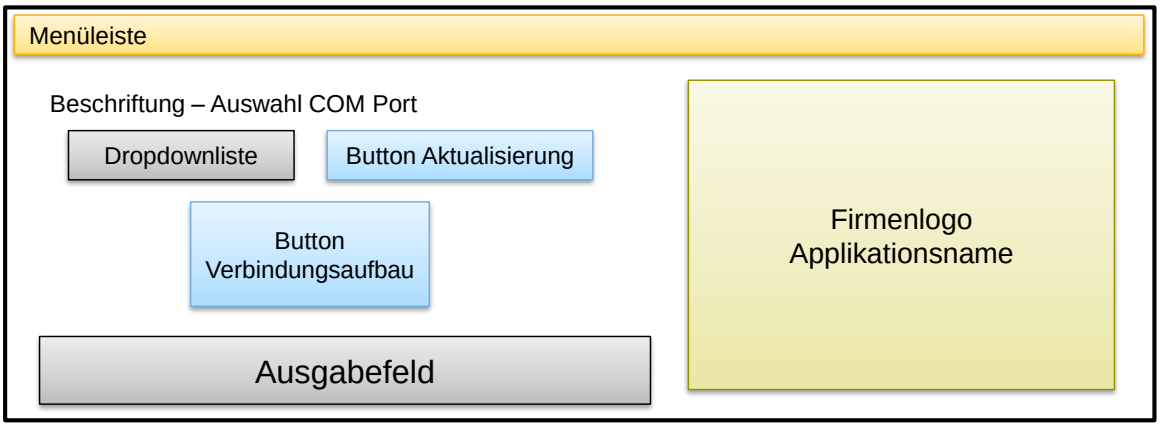

Abbildung 5.10: Anordnung der Elemente - Oberfläche 1: Verbindungsaufbau

Nach erfolgreichem Verbindungsaufbau mit einem COM-Port und Identifizierung des *S32-Designs* wechselt die Oberfläche. Zum Wohle der Übersichtlichkeit erfolgt in der folgenden Zuordnung eine weitere Aufteilung hinsichtlich der Parameter.

- Oberfläche 2: Aktive Verbindung
  - Transponderprotokoll
    - (VII) Auswahl → *Dropdown-Liste*
  - Geheimer Schlüssel
    - (VIII) Eingabe → *Eingabefeld*
    - (X) Einsehen der Länge → *Beschriftungselement*
    - (XII) Rückkehr zur Standardkonfiguration → *Kontrollkästchen*
  - Challenge
    - (IX) Eingabe → *Eingabefeld*
    - (X) Einsehen der Länge → *Beschriftungselement*
    - (XII) Rückkehr zur Standardkonfiguration → *Kontrollkästchen*
  - *ABIC2*-Register
    - (XI) Auswahl Register → *Button-Array*
    - (XI) Setzen der Registerwerte → *Zusatzfenster*
    - (XII) Rückkehr zur Standardkonfiguration → *Kontrollkästchen*
  - Datenübertragung / Startbedingungen
    - (XV) Auslösung einmaliger Programmablauf → *Button Single*
    - (XV) Auslösung Programmablauf in Endlosschleife → *Button Schleife*
    - (XVII) Auslösung Programmablauf per Tastendruck → *Button Taster*
  - Ausgabe
    - (XVIII) Ausgaben des *S32-Designs* → *Ausgabefeld*
    - (XIX) Resultat der Transponderkommandos → *3 Bilder Resultat*
    - (XX) Anzeige der Lizenzbedingungen → *Menüleiste*

Abbildung 5.11 visualisiert die Beschriftung und Anordnung der Elemente. Kontrollkästchen sind als Quadrate mit einem X darin dargestellt. Ein nicht selektiertes Kontrollkästchen deaktiviert das zugehörige Eingabefeld und signalisiert die Verwendung der Standardkonfiguration. Nach der Aktivierung kann ein selbstgewählter Wert eingegeben werden. Ein Übertrag der eingegebenen Daten in das Modell sowie eine Datenübertragung zum *S32-Design* erfolgt durch die drei einzelnen Startbuttons oberhalb des Ausgabefeldes. Im Fall des Button-Arrays für die *ABIC2*-Register unterbindet ein

nicht selektiertes Kontrollkästchen zusätzlich die Möglichkeit, ein Zusatzfenster für das Setzen des Registers zu öffnen.

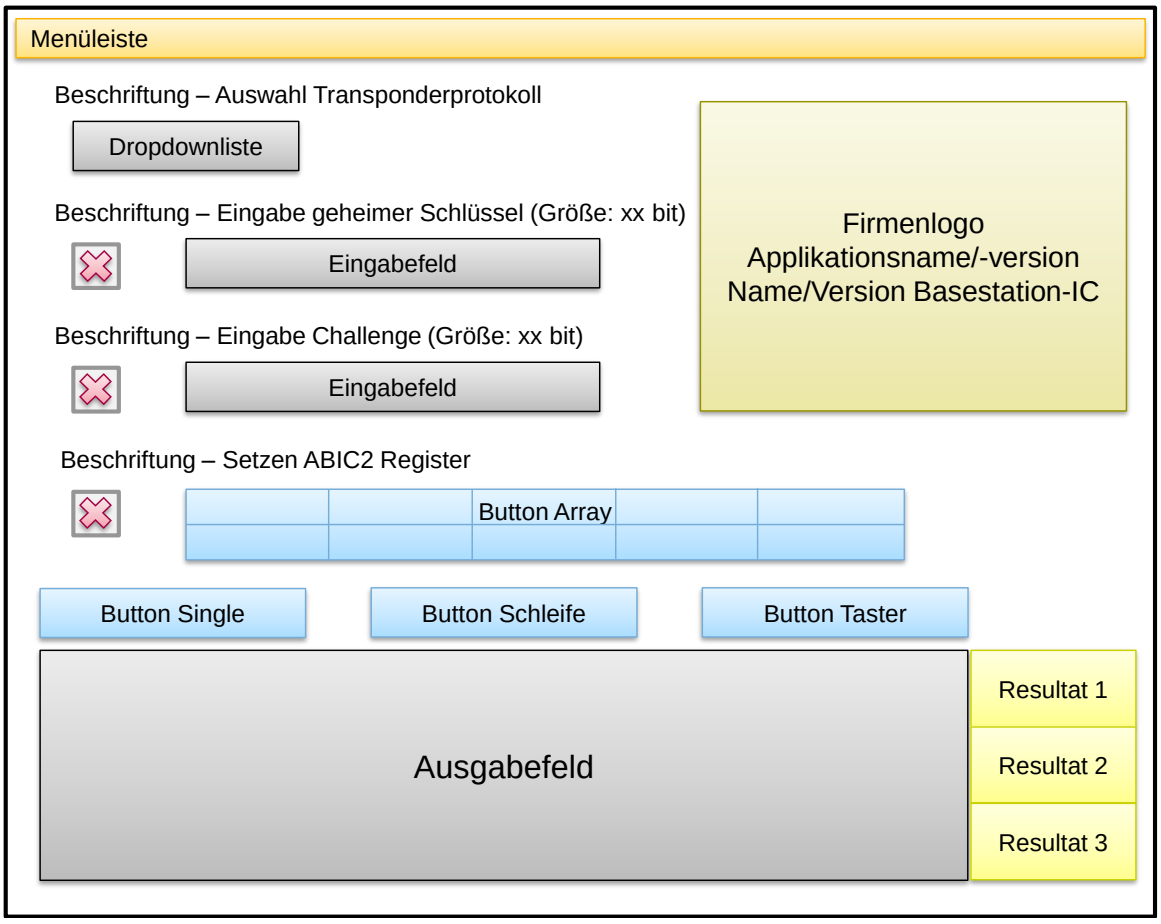

Abbildung 5.11: Anordnung der Elemente - Oberfläche 2: Aktive Verbindung

Ein Zusatzfenster ermöglicht genügend Platz, um eine bitweise Eingabe der *ABIC2*-Registerwerte unter Darstellung der Bitnamen sowie deren Standardkonfiguration zu realisieren. Jeder Button des Button-Arrays öffnet das Zusatzfenster mit den spezifischen Werten des entsprechenden Registers. Abbildung 5.12 visualisiert die Beschriftung und Anordnung der Elemente. Ein Klick auf einen Button mit der 0 bzw. 1 negiert das Bit. Ein inaktiver Button zeigt ein nicht änderbares Bit an. Auf Abbildung 5.12 wurde dafür beispielhaft Bit 7 gewählt. Der Übertrag in das Datenmodell erfolgt mit dem 'Button Bestätigung', welcher gleichzeitig den aktuell konfigurierten Wert in hexadezimaler Form anzeigt.

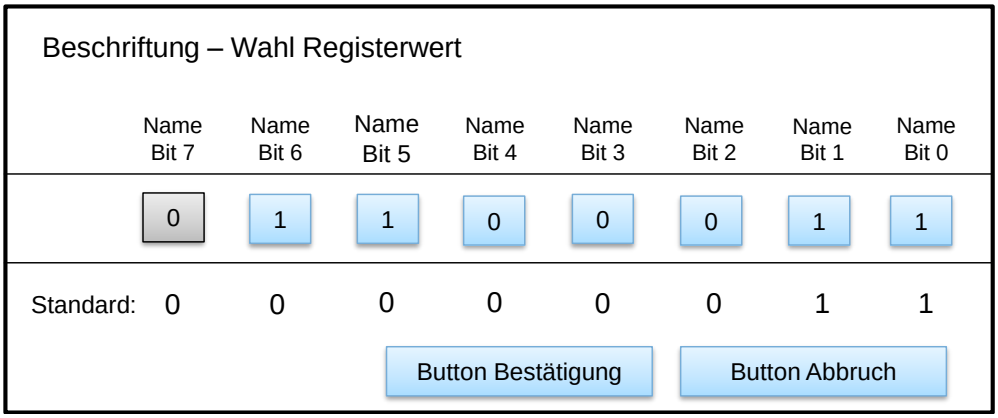

Abbildung 5.12: Anordnung der Elemente - Zusatzfenster: *ABIC2*-Register

Mithilfe einer übergeordneten Design-Klasse lassen sich alle grafischen Elemente gleichen Typs gemeinsam formatieren. Das Farbschema folgt dem Design des firmeneigenen Foliensatzes. Für detaillierte Informationen sei direkt auf die Doxygen-Dokumentation der Designklasse auf der beigelegten CD verwiesen.

Screenshots der *BUI* sind in Anhang G dargestellt.

Entwurf eines Klassendiagramms

Ein Klassendiagramm modelliert die funktionalen Anforderungen in Verbindung mit den definierten grafischen Oberflächen und Elementen unter Beachtung des MVC-Konzeptes. Mithilfe von existierenden Java Superklassen[32] und „Interfaces“[33] lassen sich sowohl die Beziehungen der architektonischen Bestandteile als auch die Integration des *RXTX*-Moduls auf vorhandenen Strukturen aufbauen. Abbildung 5.13 zeigt vereinfacht die Realisierung der MVC-Struktur in diesem Projekt in Bezug zu Abbildung 5.2.

Eine Assoziation ist sinngemäß nach [3] die Eigenschaft zweier Objekte, Nachrichten oder Anforderungen untereinander auszutauschen. Weiterführende Informationen zu den verwendeten Superklassen und „Interfaces“ geben [22], [23] und [24].

[32] Sinngemäß nach [4] übergeordnete Klasse der Vererbungshierarchie
[33] Sinngemäß nach [2] eine Schnittstelle, um einer Klasse eine bestimmte Funktionalität zur Verfügung zu stellen

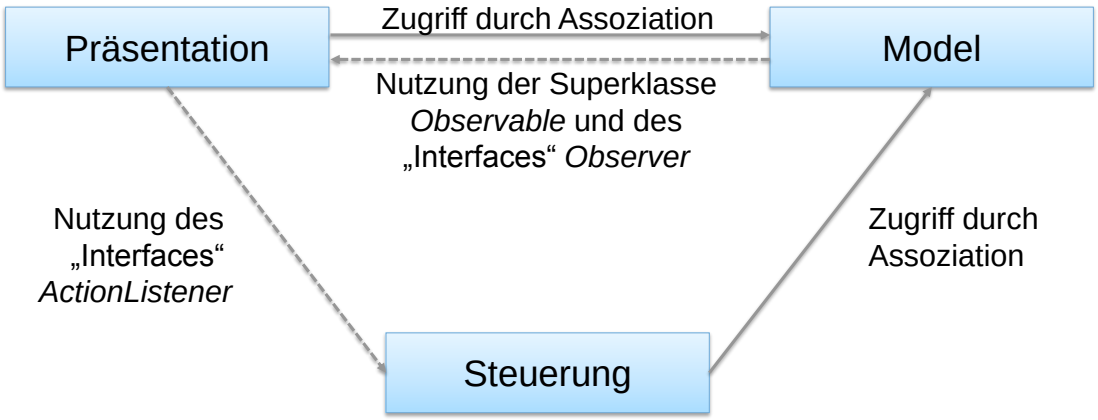

Abbildung 5.13: MVC-Struktur, Java-Implementierung

Die Verbindung zu *RXTX* erfolgt im Controller durch Verwendung von Java Ein- und Ausgabeklassen. Eingehender und ausgehender Datenstrom werden in jeweils nebenläufigen Prozessen ausgeführt. *RXTX* verbindet die Datenströme mit der realen Hardware.

Das Klassendiagramm der *BUI* ist auf der beigelegten CD dokumentiert. Detaillierte Informationen sind in Anhang I zu finden. Die Notation orientiert sich an der beschriebenen UML-Darstellung in [5]. Für weiterführende Information zu verwendeten Java Superklassen und „Interfaces“ sei auf die Originaldokumentation der Firma Oracle verwiesen. Beispielhaft sind in dieser Thesis [22], [23] und [24] angegeben.

Im Zuge der Erarbeitung des Klassendiagramms wurde großen Wert auf eine einfache Implementierung weiterer Basisstation-ICs unter Anwendung des entwickelten Protokolls gelegt. Wiederverwendbare Programmteile, wie die Kommunikation und der Prozess des Verbindungsaufbaus, sind klar von den spezifischen Definitionen in Modell und Präsentation getrennt. Gleichzeitig bleiben die geschaffenen Strukturen des MVC-Konzeptes unberührt. Die allgemeinen Klassen *SelectionParam*, *InputParam* und *Register* unterstützen eine Generalisierung bei Entwurf des Datenmodells. Durch die Softwarearchitektur sind notwendige Anpassungen bei Implementierung eines weiteren Basisstation-ICs klar auf einzelne Klassen und Methoden lokalisierbar. Anhang H beschreibt eine mögliche Vorgehensweise. Zudem lassen sich durch eine Verzweigung in Abhängigkeit der Version des verbundenen *S32-Designs* Modifikationen oder NXP-interne Versionen realisieren.

Das entwickelte Übertragungsprotokoll ist in Modell und Controller implementiert. Während das Modell anhand eines übergebenen Parameters das Kommando identifiziert und die Datenbytes auf Grundlage der Modelldaten konstruiert, rahmt der

Controller diesen Datenstrom mit Längen- und Prüfsummenbyte ein. Das erlaubt spezifische Kommandodefinitionen für jeden Basisstation-IC innerhalb des Modells.

Definition der maximalen Wartezeiten

Die Umsetzung des Zustandsdiagramms aus Abbildung 5.6 macht eine Definition der maximalen Wartezeiten auf eine Antwort des *S32-Designs* notwendig. Folgende Zeiten gelten mit Blick auf die dargestellten Programmablaufphasen in Abbildung 5.14 als Erweiterung der Abbildung 5.8.

- Maximale Wartezeit auf *SET_READY_TO_RECEIVE* (ACK1): 2 *s*
- Maximale Wartezeit auf andere Kommandos (ACK2): 0.5 *s*

Die Wartezeit auf ein ACK1 ist verhältnismäßig hoch, da das Kommando zu jedem Zeitpunkt des Programmablaufes des *S32-Designs* gemäß Definition in Unterabschnitt 4.2.2 ausgelöst werden kann. Demnach ist die Zeit bis zur Verarbeitung im ungünstigsten Fall so groß wie die Durchlaufzeit eines Zyklus. Die Wartezeit auf ein ACK2 richtet sich hauptsächlich nach der zu erwartenden Verarbeitungszeit auf dem *S32-Design*. Diese hängt hauptsächlich von der implementierten Warteroutine zwischen Empfang eines Kommandos und der Antwort ab. Beide Wartezeiten sind sehr sicher dimensioniert, da in Anbetracht der zur erwartenden Verzögerung bis zur Signalisierung der Empfangsbereitschaft des *S32-Designs* keine Notwendigkeit einer maximal schnellen Übertragung besteht.

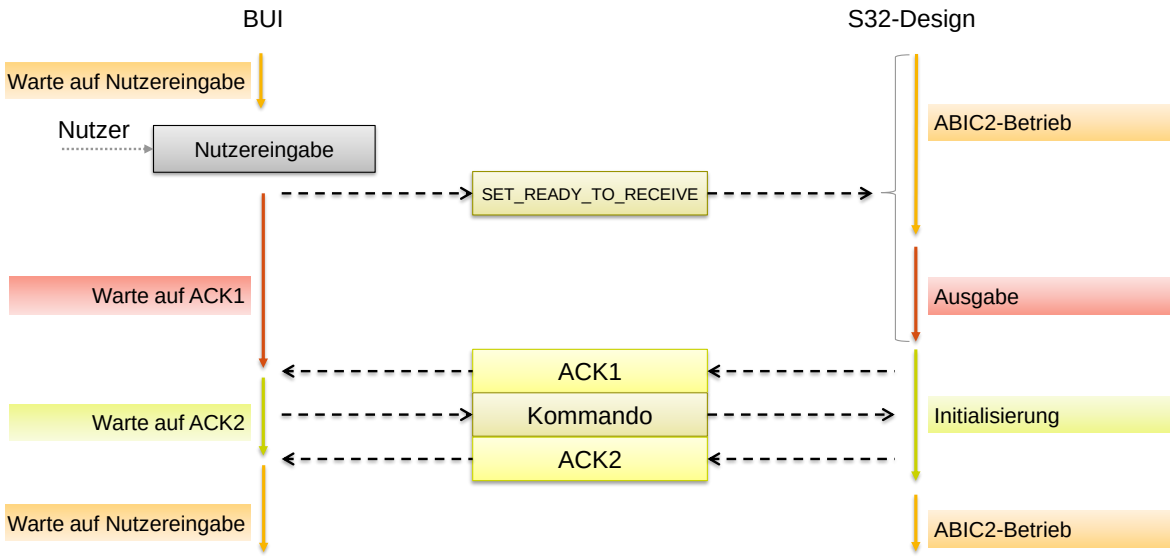

Abbildung 5.14: Kommandoübertragung, Programmablauf *BUI*

Es wird festgelegt, dass bis zum Abschluss einer Übertragung nach Abbildung 5.14 keine erneute Nutzereingabe möglich ist. In Folge eines NAKs wird der Benutzer gemäß (VI) durch ein Popup-Fenster benachrichtigt. Ein Timeout führt gemäß (V) neben der Benachrichtigung des Benutzers zu einer Rückkehr zur Startoberfläche.

Auf Seiten des *S32-Designs* erfolgt die Einrichtung eines Timeouts bei ausbleibender Antwort der *BUI* nach dem Signalisieren der Empfangsbereitschaft (ACK1).

- Maximale Wartezeit auf BUI-Kommando nach ACK1: 3 s

Die Wartezeit wurde in dieser Größenordnung gewählt, um Verzögerungen der Multitasking-Umgebung des Benutzer-PCs abzufangen. Bei Eintreten eines Timeouts kehrt das *S32-Design* in den Standardprogrammablauf zurück und ist bereit für eine weitere Kommunikation mit der *BUI*. Die aktuelle Konfiguration bleibt erhalten.

Integration in den Programmablauf des S32-Designs

Auf Basis der Vorüberlegungen in Unterabschnitt 5.2.2 muss der Standardprogrammablauf in Endlosschleife, dargestellt in Abbildung C.2, angepasst werden. Abbildung C.3 zeigt den Programmablauf nach Implementierung der Kommunikation mit der *BUI*.

Für die Realisierung der Protokollkommunikation wird das Sende- und Empfangsmodul *LPUART1* des *S32-Designs* um folgende Funktionalität erweitert.

- Abfrage, ob neue Daten im FIFO
- Empfang gemäß des Übertragungsprotokolls
- Senden gemäß des Übertragungsprotokolls

Für weitere Informationen sei auf das Modul *S32K144* innerhalb der Doxygen-Dokumentation auf der beigelegten CD verwiesen.

Sowohl die Kommunikation mit der *BUI* als auch die Umsetzung der empfangenen Konfiguration wird auf applikativer Ebene integriert. Für weitere Informationen sei auf das Modul *MAIN* innerhalb der Doxygen-Dokumentation auf der beigelegten CD verwiesen.

Detaillierte Informationen zur beigelegten CD sind in Anhang I zu finden.

Definition der Komponententests

Die Komponententests sind in folgende Kategorien eingeteilt. Da Abhängigkeiten berücksichtigt werden müssen, erfolgt die zeitliche Abarbeitung der Unterpunkte in der aufgeführten Reihenfolge und direkt in Anschluss an die Fertigstellung der involvierten Klassen bzw. Funktionen.

- *BUI*
  - MVC-Model
  - MVC-Präsentation
  - MVC-Controller
  - Oberfläche 1: Verbindungsaufbau
  - Oberfläche 2: Aktive Verbindung
- *S32-Design*
  - *LPUART2*-Modul
  - Applikative Integration der *BUI*-Kommunikation

Eine Übersicht gibt die Exceltabelle auf der beigelegten CD. Detaillierte Informationen sind in Anhang I zu finden.

5.2.4 Programmierung und Implementierung

Um eine langfristige Nutzung der Software zu gewährleisten, gilt es, sowohl die Versionen der Java Laufzeitumgebung, die der Entwicklungsumgebungen Eclipse Java und des S32-Design-Studios als auch des Moduls *RXTX* festzulegen und einzufrieren. Das garantiert eine definierte Nutzungs- bzw. Entwicklungsumgebung und vermeidet Kompatibilitätsprobleme mit kommenden Versionen.

Die Programmierung der *BUI* erfolgt unter Nutzung der **„Eclipse Java EE IDE for Web Developers“** in der Version **Oxygen.3a Release (4.7.3a)** und unter Verwendung der **JRE 1.8.0_161**. Die Verbindung zur seriellen Schnittstelle wurde unter Anwendung des Moduls **RXTX 2.2** realisiert.

Die Programmierung des *S32-Designs* erfolgt unter Nutzung des **„S32 Design Studio for ARM“** in der Version **2018.R1**.

Eine Einsicht in den Code gibt die Doxygen-Dokumentation auf der beigelegten CD. Detaillierte Informationen sind in Anhang I zu finden.

5.2.5 Komponententests

Alle definierten Tests wurden mit positiven Ergebnis durchgeführt.

5.2.6 Integrationstests

Alle definierten Tests wurden mit positiven Ergebnis durchgeführt.

5.2.7 Systemtests und Validierung

Alle definierten Anforderung wurden nach Überprüfung mit erfüllt eingestuft.

Für das Abprüfen von Anforderung (I) und (II) wurde das Softwareprojekt der *BUI* als eine lauffähige JAR[34]-Datei exportiert. Der Start erfolgt durch Doppelklick auf eine erstellte Batch-Datei, welche eine Verwendung der eingefrorenen Version der Java Laufzeitumgebung garantiert. Vor dem Start der *BUI* sind folgende Schritte im Rahmen der erstmaligen Inbetriebnahme notwendig:

- Anschließen des *S32-Designs*, Installation der OpenSDA-Treiber
- Installieren der Java Laufzeitumgebung 1.8.0_161 in den Standardpfad (C:\Program Files\java\jre1.8.0_161\...)
- Kopieren der Datei *RXTXcomm.jar* in den Pfad (C:\Program Files\java\jre1.8.0_161\lib\ext\...)
- Kopieren der Datei *rxtxSerial.dll* in den Pfad (C:\Program Files\java\jre1.8.0_161\bin\...)
- Kopieren der Datei *rxtxParallel.dll* in den Pfad (C:\Program Files\java\jre1.8.0_161\bin\...)

Der Testlauf der Inbetriebnahme auf einem vom Entwicklungsprozess unabhängigen Laptop dauerte knapp 14 Minuten. Zur Reproduktion befinden sich die notwendigen Dateien auf der beigelegten CD. Detaillierte Informationen sind in Anhang I zu finden.

Mit Erfüllung aller Nutzungsanforderungen wird die Entwicklung der Benutzerschnittstelle „*Basestation User Interface*“ als erfolgreich betrachtet.

Screenshots der *BUI* sind in Anhang G dargestellt.

[34] **J**ava **Ar**chive, Dateiformat zur Verteilung von Java Klassen

6 Fazit

Die in Kapitel 3 definierten Anforderungen sind mit der Fertigstellung dieser Thesis erfüllt. Sowohl die Beispielimplementierung des *ABIC2* in einer Immobilizer-Applikation als auch eine grafische Benutzerschnittstelle zur grundlegenden Konfiguration sind als Prototypen in der Version 0.9 verfügbar. Die verbleibenden Schritte bis zur Produktreife nennt Kapitel 7.

Die Anwendung des *S32K144* transportiert das vorhandene Design auf einen zeitgemäßen Stand und zeigt dem Kunden zielgerichtet weiterführende Möglichkeiten mit dem Controller auf. Als Beispiel dafür seien allgemeine Funktionen zur Konfiguration von GPIO-Pins sowie vorbereitete Code-Ausschnitte einer Zeitmessung oder Erstellung eines Oszilloskop-Triggersignals genannt.

Eine Wahl des Transponderprotokolls und der Schnittstelle zwischen Controller und *ABIC2* ist mit dem erstellten Design ohne Neu-Kompilierung möglich. Mit der Möglichkeit der Einstellung der Schnittstellengeschwindigkeit sowie der Messung der Zeit für eine Authentifikation wurde das Referenzdesign funktional erweitert.

Eine strengere Umsetzung des modularen Schichtenmodells in der Software-Architektur vereinfacht zudem eine Wiederverwendung einzelner Programmteile in anderen Projekten oder eine Integration weiterer Basisstation-ICs.

Der Entwicklungsprozess der grafischen Schnittstelle unter Anwendung des V-Modells zeigt die Vorteile einer frühen Spezifikation der Testphasen. Mithilfe des MVC-Konzeptes konnte eine Klassenstruktur entwickelt werden, welche eine Wiederverwendbarkeit der Kommunikation zwischen PC und *S32K144* auch für andere Projekte erlaubt.

Eine Änderung des verwendeten geheimen Schlüssels sowie das Setzen einer festen Challenge sind neue Möglichkeiten des erstellten Designs. Darüber hinaus lässt sich der Kommunikationszyklus einer Transponderkommunikation nun auch einzeln oder auf Tastendruck starten.

Die von der IDE unabhängige Steuerung des Referenzdesigns durch die *BUI* erlaubt eine zügige Inbetriebnahme zum Zwecke initialer Validierungen. Erfahrungsgemäß spielt bei der Bewertung eines technischen Produkts nämlich meist die Zeit zwischen Auspacken und Funktionieren eine übergeordnete Rolle.

7 Ausblick

Das erstellte Referenzdesign dient als Beispielimplementierung und unterliegt dementsprechend einem hohen Qualitätsanspruch. Besonders die Portierung bedarf deshalb einer mehrstufigen Überprüfung. Vor einem Release der Version 1.0 sind folgende Schritte notwendig.

- Funktionales Review der Portierung inklusive der Konfiguration mithilfe der *BUI* durch den produktverantwortlichen Ingenieur des *ABIC2*
- Review der Doxygen-Dokumentationen durch den produktverantwortlichen Ingenieur des *ABIC2*
- Review der Softwarestruktur und des Programmierstils durch einen Softwareingenieur
- Überführung der Kommentierung des Quelltextes der HT-Schicht in den Doxygen-Stil
- Aktualisierung der Bedienungsanleitung für das Referenzdesign
- Firmenspezifische Release-Prozedur

Die Einarbeitung in das Thema und in den *ABIC2* erlaubte einen Blick auf das Softwareprojekt ähnlich dem eines Kunden. Im Prozess der Portierung wurden resultierend Optimierungsmöglichkeiten entdeckt, welche nach diesem Projekt zur Diskussion gestellt werden können.

Bei der firmeninternen Vorstellung der *BUI* entstanden schnell zwei neue Anforderungen, welche nach diesem Projekt umgesetzt werden können.

- Aktivierung und Deaktivierung erweiterter Debug-Ausgaben des *S32-Designs*
- Speichern und Laden eines definierten *ABIC2*-Registersets

A Fotos der Hardware

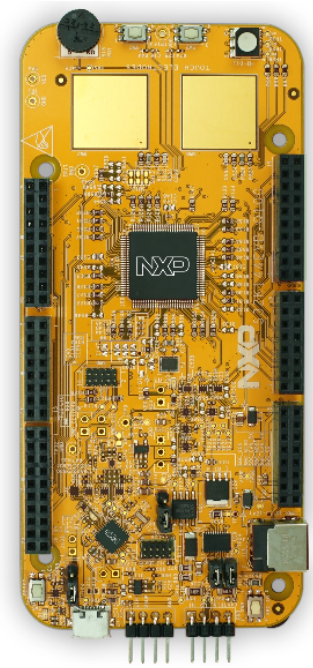

Abbildung A.1: *S32K144*-CEB, [10]

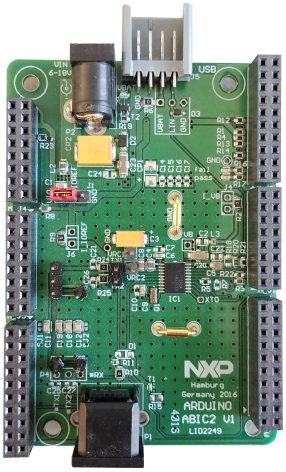

Abbildung A.2: *ABIC2*-CEB

Abbildung A.3: *ABIC2*-CEB, aufgesteckt auf *S32K144*-CEB

Für eine Auswahl der Schnittstelle zwischen Controller und *ABIC2* wird der rote Jumper entsprechend positioniert. In der sichtbaren Position wird SPI verwendet.

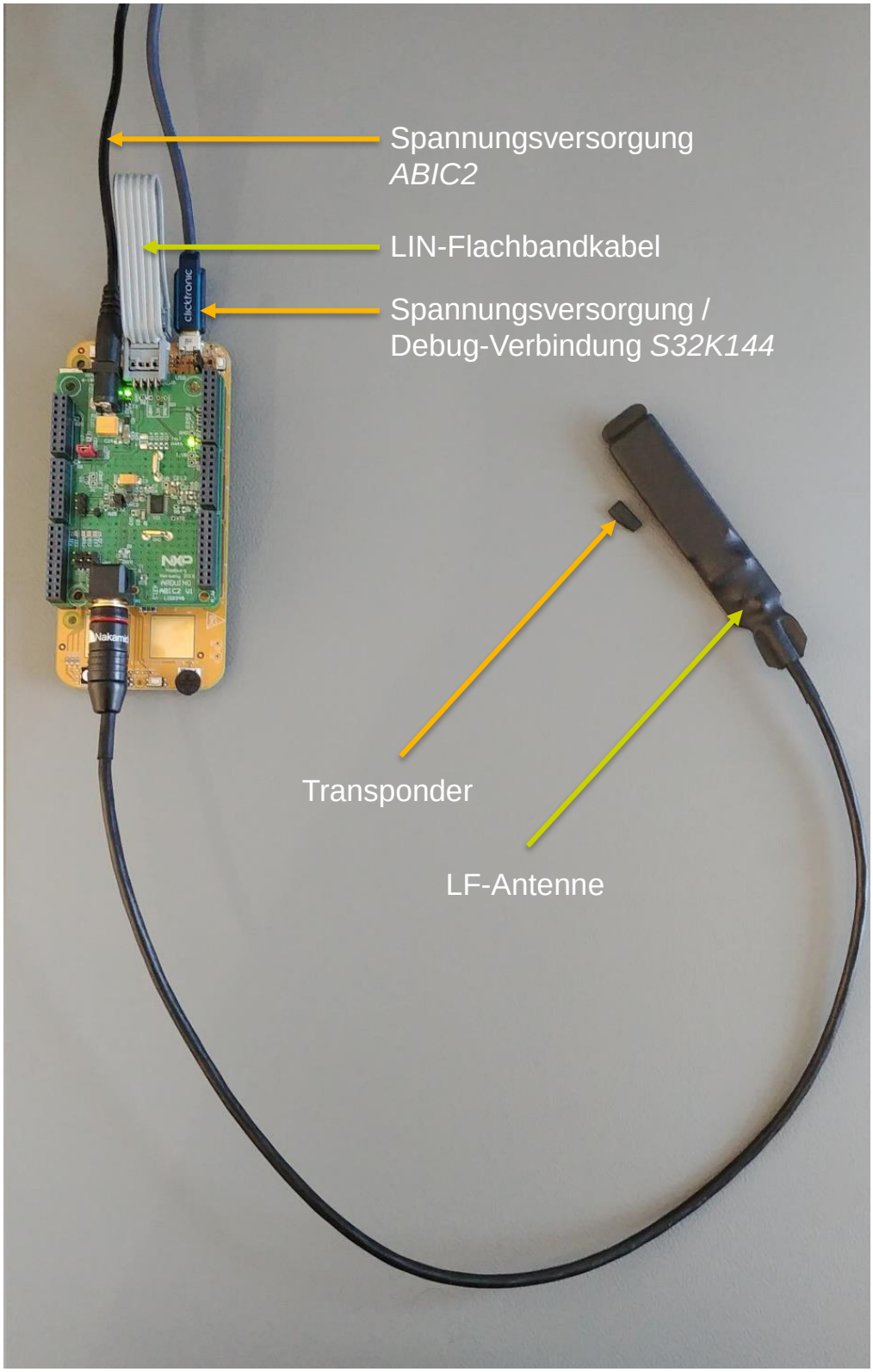

Abbildung A.4: *S32-Design* mit Beschriftung der Komponenten

B Verwendete Pins und Onboard-Komponenten des S32K144-CEB

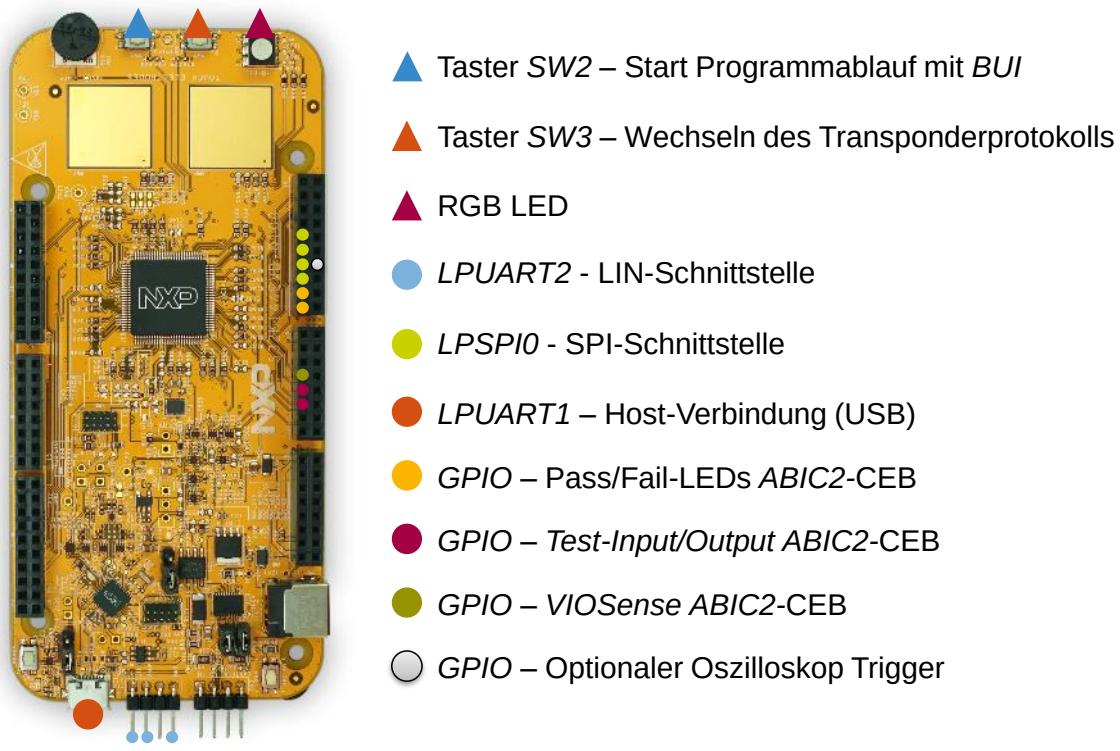

Abbildung B.1: In diesem Projekt verwendete Pins und Onboard-Komponenten des *S32K144*-CEB, Abbildung basierend auf [10]

C Programmablaufpläne

Die folgenden Abbildungen beschreiben den Programmablauf der Main-Routine der Software des Referenzdesigns mithilfe von Aktivitätsdiagrammen. Der Ablauf wurde im Sinne einer Einführung in die Applikation und der getätigten Änderungen sehr vereinfacht und gekürzt. Der Präfix *var* kennzeichnet eine Variable in Pseudocode.

Programmablauf des LPC93x-Designs

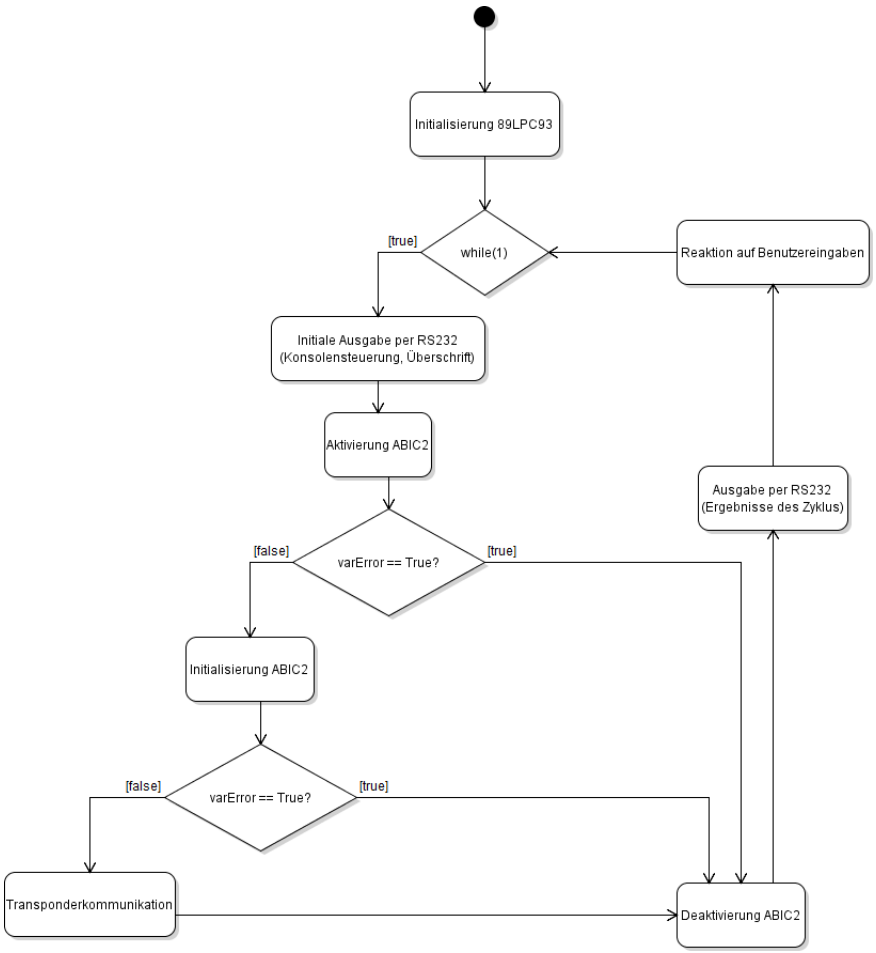

Abbildung C.1: Programmablauf des *LPC93x-Designs* als Aktivitätsdiagramm

Programmablauf des S32-Designs

Um eine Verzweigung auf die einzelnen Transponderprotokolle zu realisieren, ist eine neue Variable *varTransponderprotokoll* notwendig.

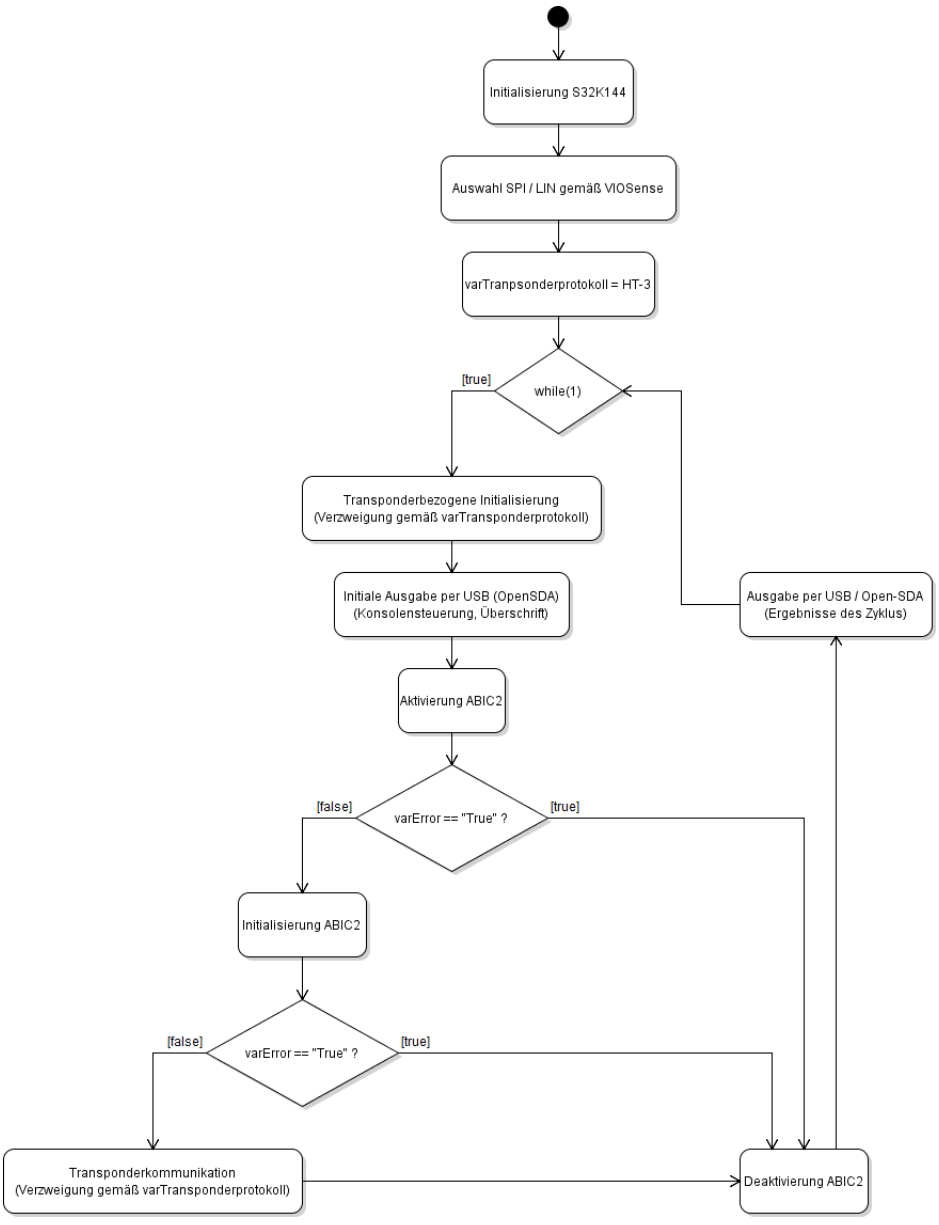

Abbildung C.2: Programmablauf des *S32-Designs* als Aktivitätsdiagramm

Programmablauf des S32-Designs nach Integration der BUI

Um die 3 Startbedingungen gemäß (XV), (XVI) und (XVII) zu realisieren, ist eine neue Variable *varProgrammablauf* notwendig, welche in Abhängigkeit des empfangenen Startkommandos gesetzt wird und Unterscheidungen im Programmablauf zulässt. Für den einmaligen Start ist eine Warteschleife durch Erweiterung der bedingten Anweisung implementiert. Mithilfe einer zusätzlichen globalen Variable *varTaster* ist eine Kommunikation mit dem Interrupt-Handler des Tasters *SW2* möglich. Das ermöglicht eine Verwendung des Tasters als Starttrigger für den Programmablauf.

Das Aktivitätsdiagramm befindet sich aus Platzgründen auf der folgenden Seite.

Aufgrund der beschränkten Größe der Abbildungen sind die Aktivitätsdiagramme auch auf der beiliegenden CD gespeichert. Einen detaillierten Überblick gibt Anhang I

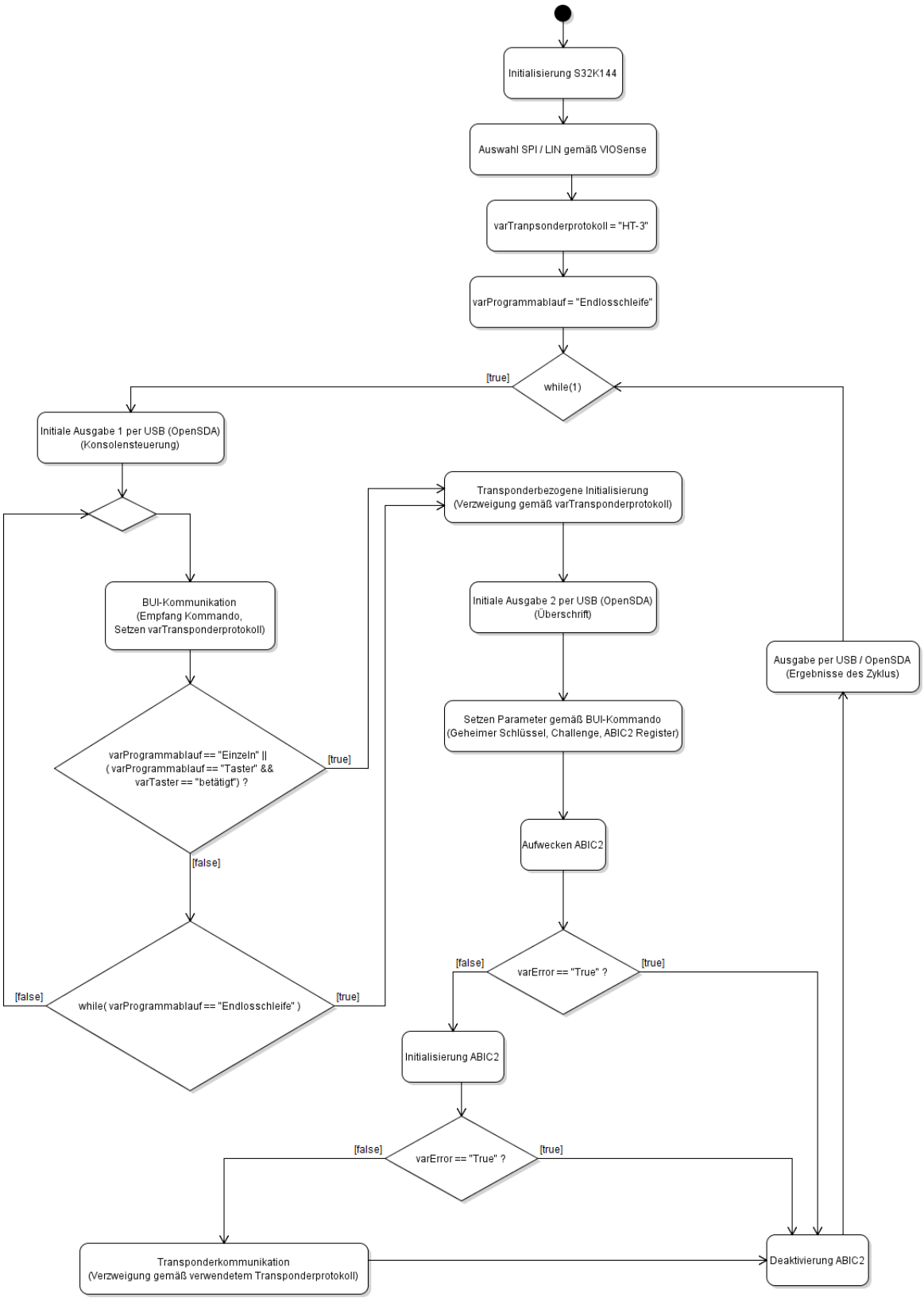

Abbildung C.3: Programmablauf des *S32-Designs* nach Integration der *BUI* als Aktivitätsdiagramm

D Kategorisierung der Quelldateien

Die Abbildung D.1 auf der folgenden Seite visualisiert die Kategorisierung der Quelldateien. Die Abbildung wurde aus Platzgründen quer dargestellt. Bezogen auf die Kategorien ergeben sich folgende Rahmenbedingungen während der Programmierung und Implementierung hinsichtlich der Portierung.

- Kategorie 'Neu' (Gelb)
  - Erstellung in diesem Projekt
  - Berücksichtigung der Konventionen zur Namensgebung
  - Einfügen des aktuellen NXP-Disclaimers
  - Berücksichtigung des Programmierstils
  - Dokumentation mit Doxygen
- Kategorie 'Sinngemäß neu' (Rot)
  - Erstellung in diesem Projekt nach Vorbild der Implementierung im *LPC93x-Design*
  - Einfügen des aktuellen NXP-Disclaimers
  - Berücksichtigung des Programmierstils
  - Überführung der vorhandenen Dokumentation in den Doxygen-Stil
- Kategorie 'Portierung' (Grau)
  - Nur für Funktionalität notwendige Änderungen
  - Einfügen des aktuellen NXP-Disclaimers
  - Überführung der vorhandenen Dokumentation in den Doxygen-Stil mit Ausnahme der HT-Schicht
- Kategorie 'Notwendig für S32K144' (Blau)
  - Unveränderte Einbindung in das Softwareprojekt

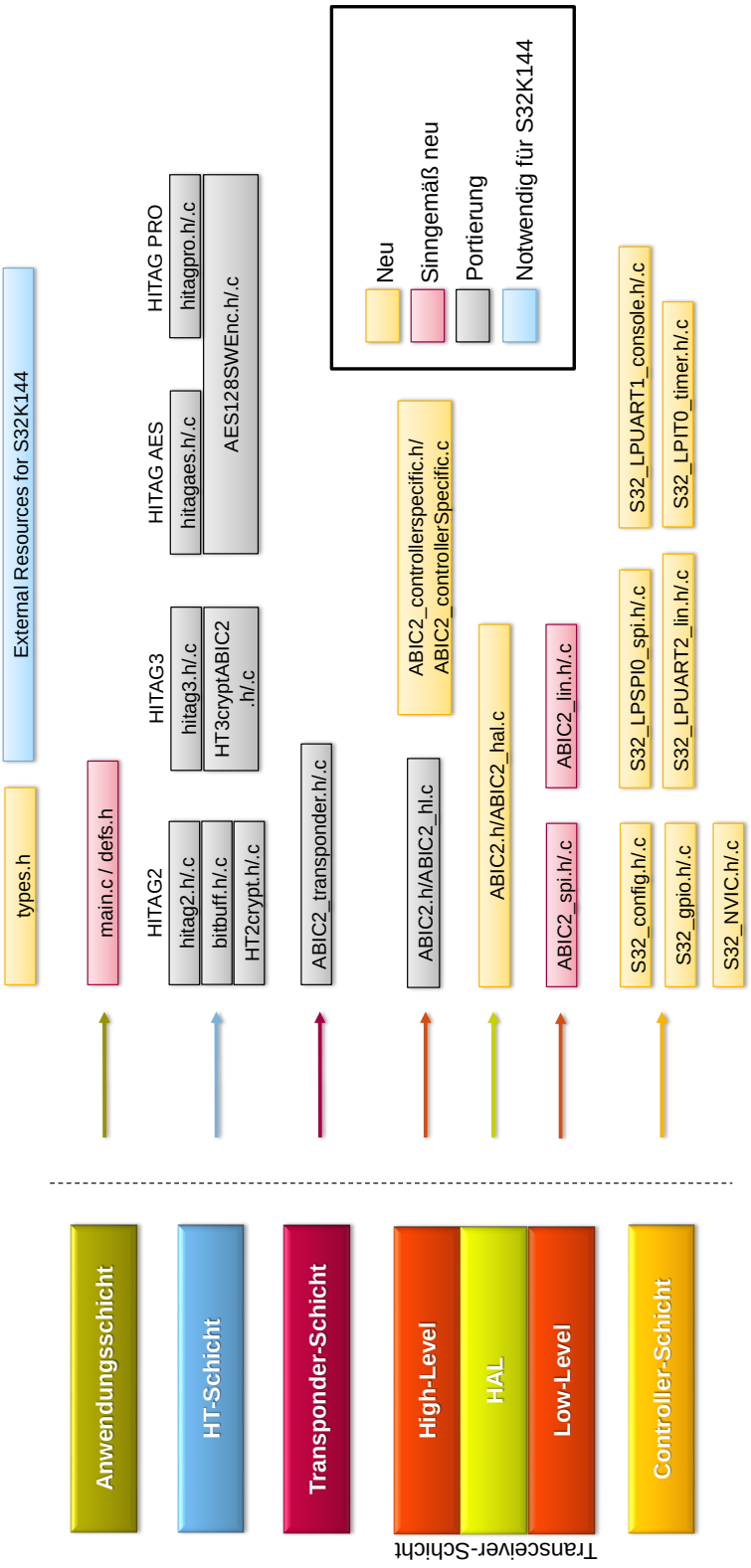

Abbildung D.1: Kategorisierung der Quelldateien

E Ausgaben auf Konsole

Die folgenden Abbildungen zeigen die Ausgaben eines Zyklus des *LPC93x-Designs* über RS232. Für einen Eindruck der Ausgaben des *S32-Designs* sei auf die Screenshots der *BUI* in Anhang G verwiesen.

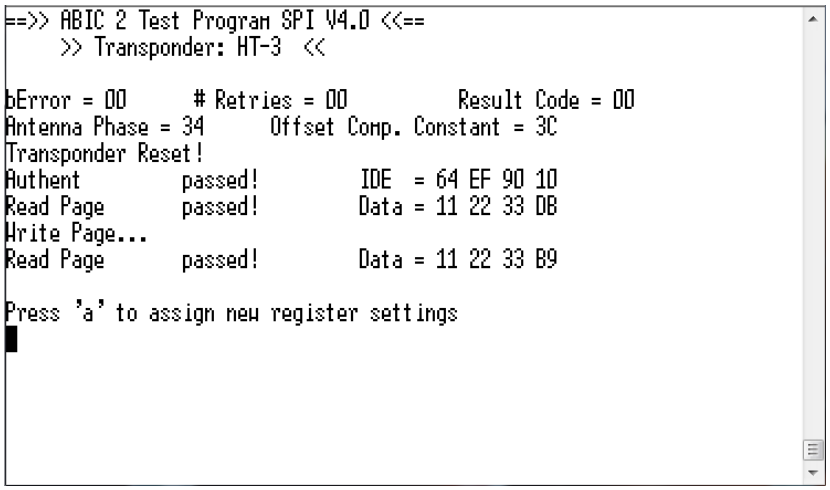

Abbildung E.1: Ausgaben *LPC93x-Design*, erfolgreiche Transponderkommunikation

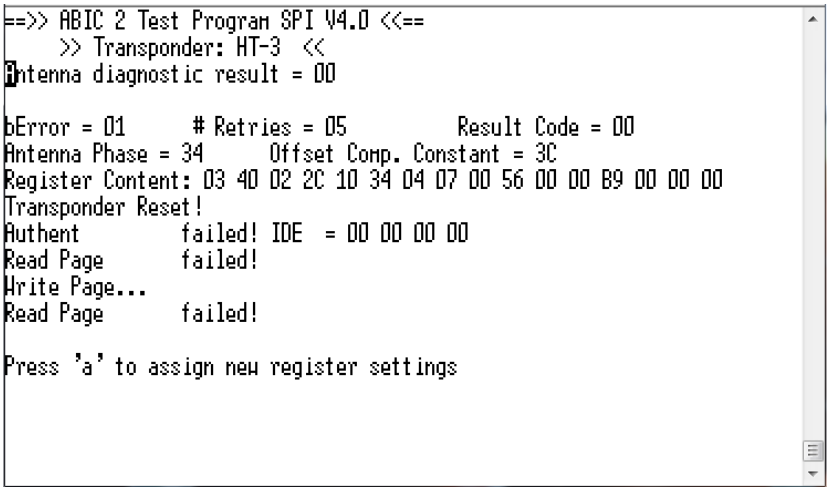

Abbildung E.2: Ausgaben *LPC93x-Design*, fehlgeschlagene Transponderkommunikation

F Lizenz RXTX

Die folgenden Lizenzbedingen sind aus [8] entnommen.

RXTX License v 2.1 - LGPL v 2.1 + Linking Over Controlled Interface. RXTX is a native interface to serial ports in java. Copyright 1997-2007 by Trent Jarvi tjarvi@qbang.org and others who actually wrote it. See individual source files for more information.

A copy of the LGPL v 2.1 may be found at http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html on November 21st 2007

A copy is here for your convenience.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

An executable that contains no derivative of any portion of RXTX, but is designed to work with RXTX by being dynamically linked with it, is considered a "work that uses the Libraryßubject to the terms and conditions of the GNU Lesser General Public License.

The following has been added to the RXTX License to remove any confusion about linking to RXTX. We want to allow in part what section 5, paragraph 2 of the LGPL does not permit in the special case of linking over a controlled interface. The intent is to add a Java Specification Request or standards body defined interface in the future as another exception but one is not currently available.

http://www.fsf.org/licenses/gpl-faq.html#LinkingOverControlledInterface

As a special exception, the copyright holders of RXTX give you permission to link RXTX with independent modules that communicate with RXTX solely through the Sun Microsytems CommAPI interface version 2, regardless of the license terms of these independent modules, and to copy and distribute the resulting combined work under terms of your choice, provided that every copy of the combined work is accompanied by a complete copy of the source code of RXTX (the version of RXTX used to produce the combined work), being distributed under the terms of the GNU Lesser General Public License plus this exception. An independent module is a module which is not derived from or based on RXTX.

Note that people who make modified versions of RXTX are not obligated to grant this special exception for their modified versions; it is their choice whether to do so. The GNU Lesser General Public License gives permission to release a modified version without this exception; this exception also makes it possible to release a modified version which carries forward this exception.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

All trademarks belong to their respective owners.

G Screenshots der BUI

Bei Start öffnet sich die Oberfläche, dargestellt in Abbildung G.1, welche eine Auswahl und einen Verbindungsaufbau zu einem COM-Port ermöglicht.

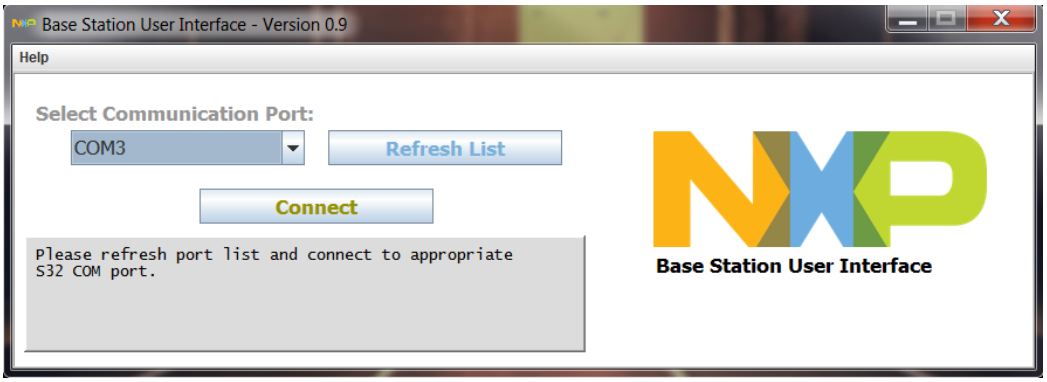

Abbildung G.1: *BUI*, Startoberfläche

Nach Klicken des Buttons 'Connect' erfolgt ein expliziter Verbindungsaufbau mit dem *S32-Design*. Nach erfolgreicher Erkennung des *ABIC2*-Referenzdesigns erscheint das Popup-Fenster, welches in Abbildung G.2 visualisiert ist.

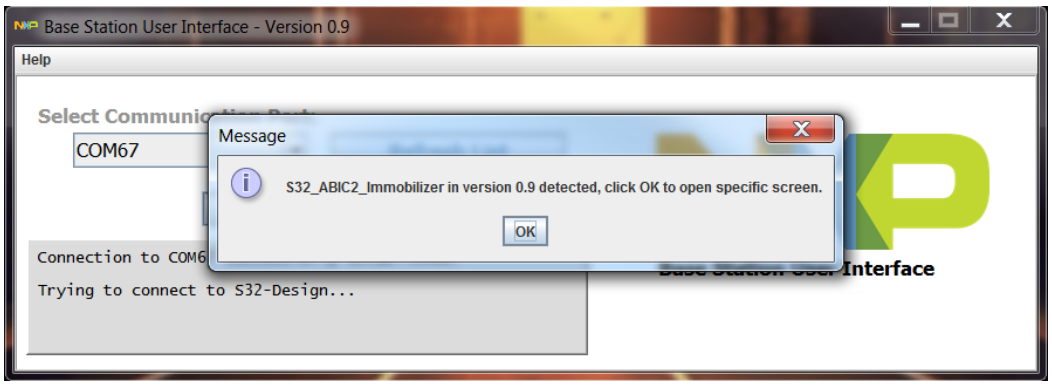

Abbildung G.2: *BUI*, Oberfläche nach erfolgreichem Verbindungsaufbau

Nach der Bestätigung des Popup-Fensters öffnet sich die spezifische Oberfläche des *ABIC2*-Referenzdesigns. Eine erfolgreiche Transponderkommunikation unter Verwendung von SPI und *HT-3* ist auf Abbildung G.3 zu erkennen.

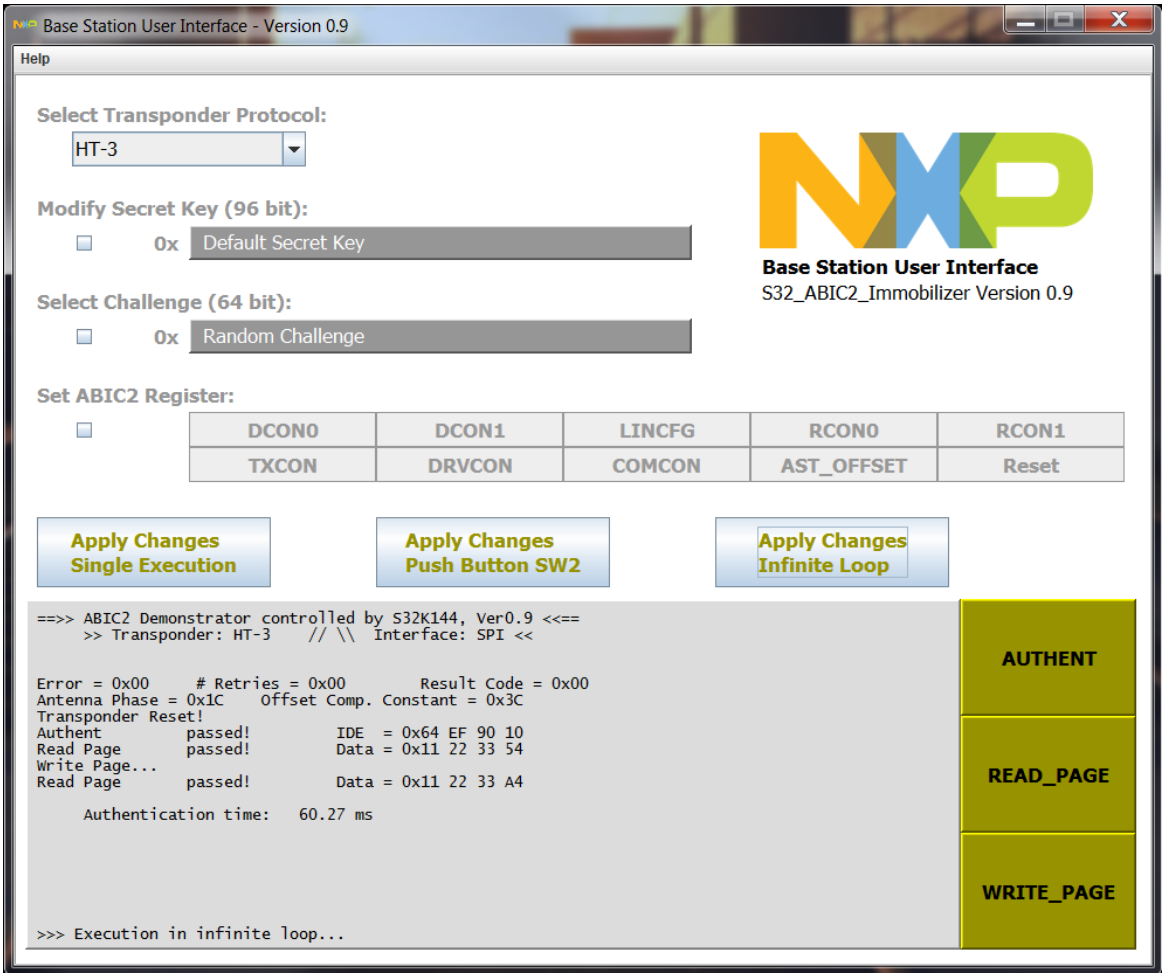

Abbildung G.3: *BUI*, Oberfläche *ABIC2*-Referenzdesign 1

Ein Wechsel auf *HT-AES* führt zu einer fehlgeschlagenen Transponderkommunikation, da sich ein *HT-3*-Transponder im Feld befindet. Darüber hinaus wurde der Zyklus per Taster gestartet. Abbildung G.4 visualisiert die *BUI* Oberfläche in diesem Moment.

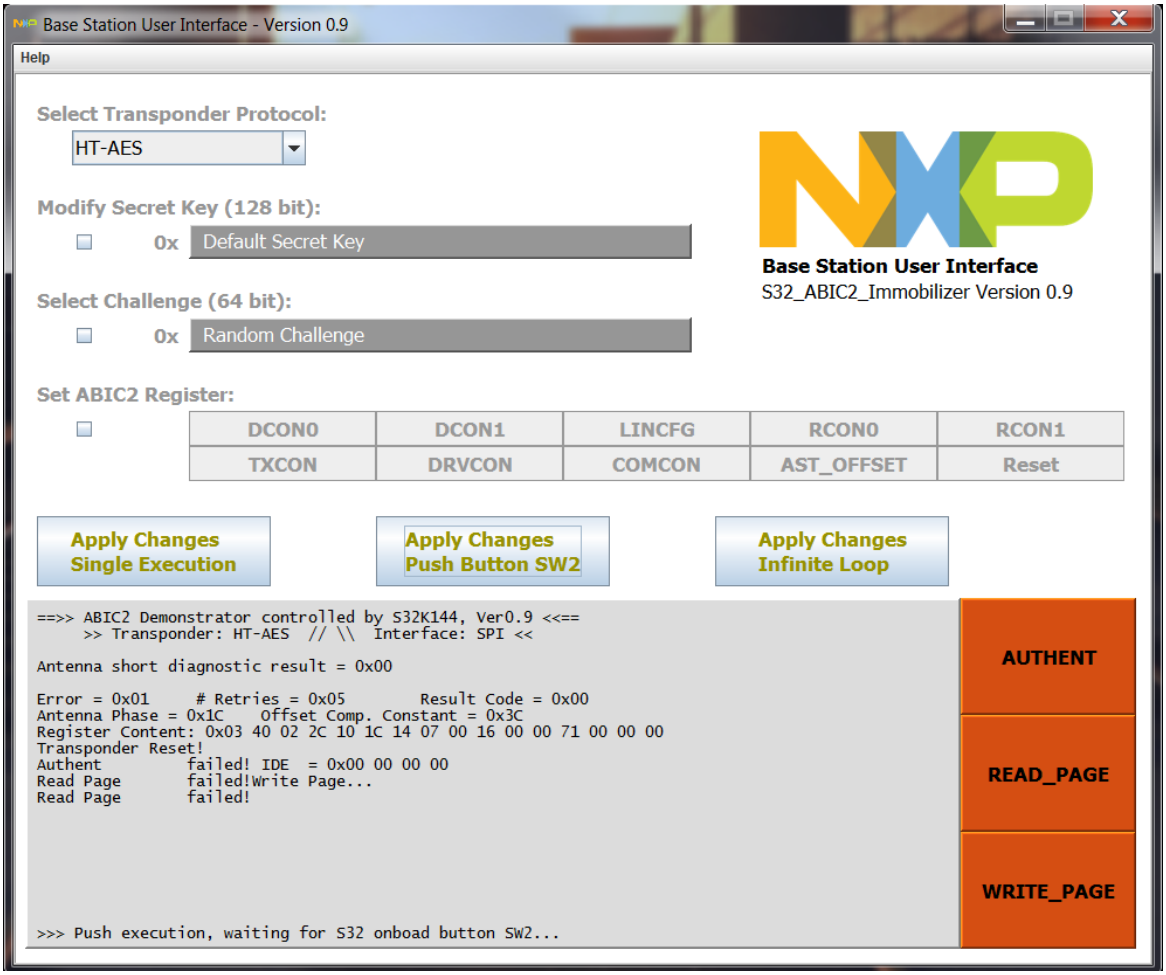

Abbildung G.4: *BUI*, Oberfläche *ABIC2*-Referenzdesign 2

Eine beispielsweise Änderung der Parameter führt zu der in Abbildung G.5 dargestellten Oberfläche. Der Programmablauf wurde einmalig in der *BUI* durch den Button 'Apply Changes Single Execution' gestartet.

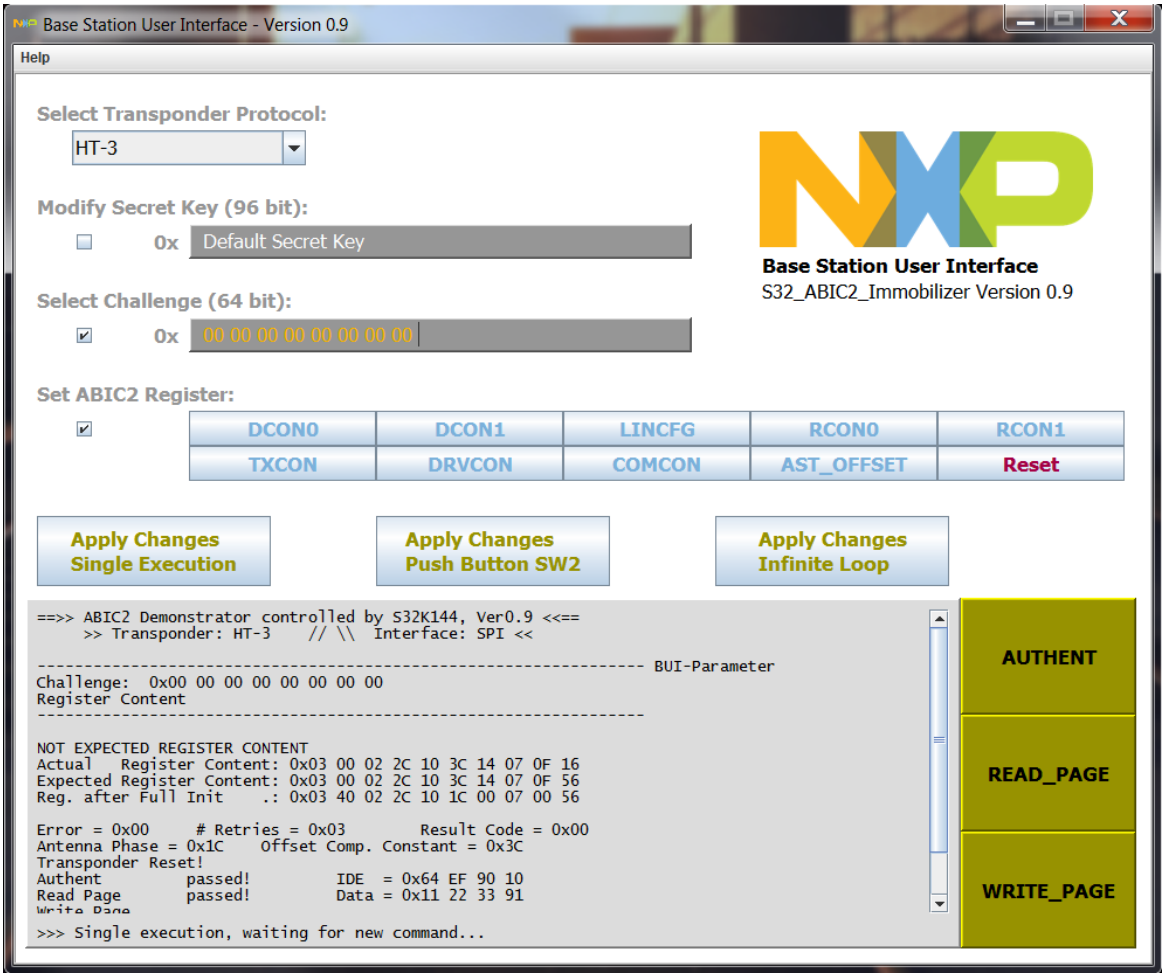

Abbildung G.5: *BUI*, Oberfläche *ABIC2*-Referenzdesign 3

Ein Klick auf den Button 'DCON0' öffnet das Zusatzfenster, mit welchem der entsprechende Registerwert innerhalb des Datenmodells der *BUI* gesetzt werden kann. Abbildung G.6 stellt das Zusatzfenster dar.

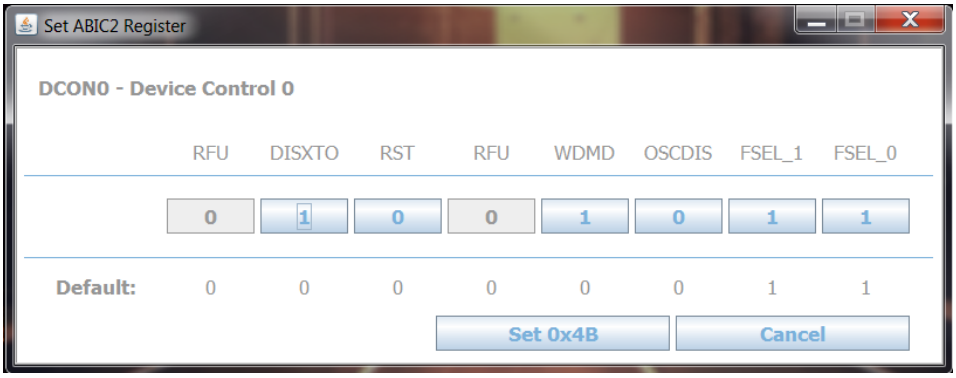

Abbildung G.6: *BUI*, Zusatzfenster *ABIC2*-Register

Über die Menüleiste lässt sich ein Fenster öffnen, dargestellt in Abbildung G.7, welches Einsicht in den NXP-Disclaimer sowie in die Lizenzbedingungen von *RXTX* ermöglicht.

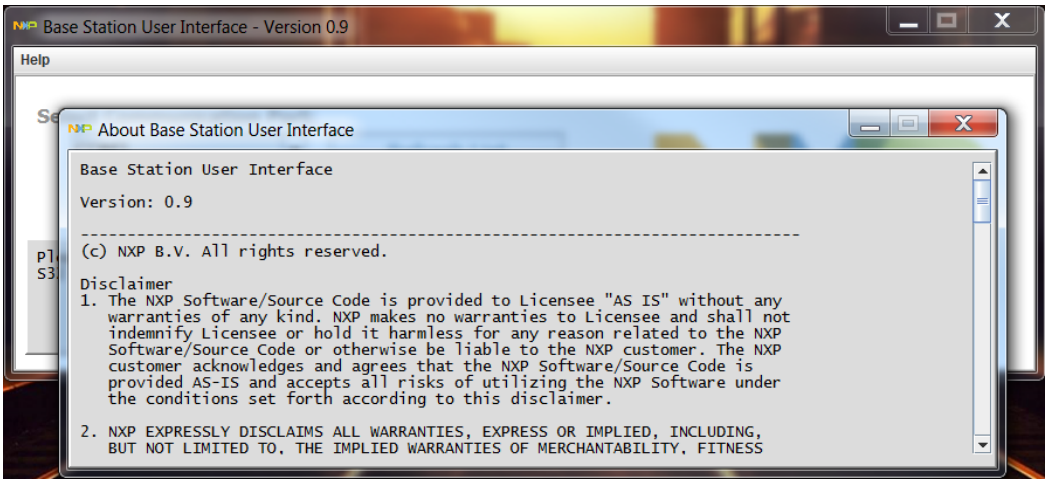

Abbildung G.7: *BUI*, Zusatzfenster Lizenzinformationen

H Implementierung eines Basisstation-ICs in die BUI

Die folgende Aufstellung beschreibt eine Lokalisierung sowie eine kurze Beschreibung der notwendigen Anpassungen bei Implementierung eines weiteren Basisstation-ICs. Die Begriffe Modell, Präsentation und Controller beziehen sich auf die Bestandteile des MVC-Konzeptes.

- Erstellung eines Modells
  - Vererbung von der abstrakten Klasse *MVC_Model*
  - Implementierung der abstrakten Methoden *getMaxResponseTime()* und *makeCommand()* zur Umsetzung spezifischer Kommandos gemäß des entwickelten Übertragungsprotokolls
- Erstellung einer Präsentation
  - Vererbung von der abstrakten Klasse *MVC_View*
  - Implementierung der abstrakten Methode *setActionListener()* zur Herstellung der Verbindung zum Controller für eine Auswertung von Benutzeraktionen
  - Implementierung der Methode *update()* zur Definition des Umgangs mit Änderungen der Daten innerhalb des Modells
- Anpassung der Klasse *MVC_Controller*
  - Erstellung der Referenzierungen des neuen spezifischen Modells und der neuen spezifischen Präsentation
  - Anpassung der Methode *connectToS32Design()*
    - Implementierung der Transition zum spezifischen Modell und zur spezifischen Präsentation des zu implementierenden Basisstation-IC
  - Anpassung der Methode *update()*
    - Implementierung des Umgangs mit empfangenen Daten des zu implementierenden Basisstation-IC
  - Anpassung der Methode *actionPerformed()*
    - Implementierung der Behandlung von Benutzeraktionen

I Beigelegte CD

An dieser Stelle folgt eine Übersicht über die Inhalte auf der beigelegten CD.

Aufgrund der Geheimhaltung der proprietären Verschlüsselung der Transponderprotokolle *HT-2* und *HT-3* sind alle Quelldateien dieser Module nicht in der Doxygen-Dokumentation auf der CD enthalten.

Aufgrund der firmeninternen Verwendung des Klassendiagrammes und der Übersicht über das entwickelte Übertragungsprotokoll ist der Inhalt dieser Dateien in englischer Sprache.

- Ordner *1_Thesis*
  - Enthält diese Thesis als .pdf-Datei
- Ordner *2_Anhang_C*
  - Enthält die Aktivitätsdiagramme als .html-Datei
- Ordner *3_Anhang_I*
  - Enthält die Doxygen-Dokumentationen in den jeweiligen Ordnern *Doxygen_BUI* und *Doxygen_ImmobilizerBspImplementierung* als .html-Dateien
    - Ein Öffnen der jeweiligen Dokumentation erfolgt durch Starten der .bat-Datei
  - Enthält das Klassendiagramm der *BUI* als .html-Datei
  - Enthält eine Übersicht über das entwickelte Übertragungsprotokoll der *BUI* als .pdf-Datei
  - Enthält die jeweiligen Testpläne als Exceltabelle im .xlsx-Format
  - Enthält die *BUI* als startbare .jar-Datei und die notwendigen *RXTX*-Ressourcen
    - Ein Start der *BUI* erfolgt durch Starten der .bat-Datei
- Ordner *4_Quellen*
  - Enthält alle Offline-Quellen als .pdf-Dateien

Die dieser Arbeit beigelegte CD ist bei Prof. Dr. Heike Neumann und Dirk Besenbruch (NXP Semiconductors Germany GmbH) verfügbar.

Literaturverzeichnis

[1] AEC Component Technical Committee. *AEC Documents [online].* URL: http://www.aecouncil.com/AECDocuments.html (besucht am 04. 10. 2018).

[2] Chris Wagner. *Interfaces – Verwendung von Schnittstellen in Java [online].* URL: http://www.programmierenlernenhq.de/interfaces-in-java/ (besucht am 28. 08. 2018).

[3] Christian Ullenboom. *Java ist auch eine Insel - Assoziationen zwischen Objekten [online].* URL: http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_05_007.htm (besucht am 28. 08. 2018).

[4] Christian Ullenboom. *Java ist auch eine Insel - Vererbung [online].* URL: http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_05_008.htm#mjf017bb7a74c678fbdf843629b9781be8 (besucht am 28. 08. 2018).

[5] Christian Ullenboom. *Java ist auch eine Insel [online].* URL: http://openbook.rheinwerk-verlag.de/javainsel9/index.htm#_top (besucht am 28. 08. 2018).

[6] dpunkt.verlag GmbH. *Model View Controller [online].* URL: https://www.dpunkt.de/java/Programmieren_mit_Java/Oberflaechenprogrammierung/40.html (besucht am 10. 08. 2018).

[7] Dimitri van Heesch. *Doxygen [online].* URL: http://www.doxygen.nl/ (besucht am 04. 10. 2018).

[8] Keane Jarvi. *RXTX License v 2.1 - LGPL v 2.1 + Linking Over Controlled Interface [online].* URL: http://users.frii.com/jarvi/rxtx/license.html (besucht am 10. 08. 2018).

[9] Juliane Barjenbruch. *Funktionale Sicherheit elektronischer Systeme in Kraftfahrzeugen [online].* URL: https://www.uni-koblenz-landau.de/de/koblenz/fb4/ist/AGZoebel/Lehre/ss2011-ordner/sem_asida/barjenbruch (besucht am 04. 10. 2018).

[10] *NXP Mikrokontroller S32K144 [online].* URL: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/s32-automotive-platform/s32k144-evaluation-board:S32K144EVB?lang_cd=en (besucht am 29. 04. 2018).

[11] *NXP Mikrokontrollerfamilie S32K [online].* URL: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/s32-automotive-platform/32-bit-automotive-general-purpose-microcontrollers:S32K (besucht am 29. 04. 2018).

[12] *NXP OpenSDA [online].* URL: https://www.nxp.com/support/developer-resources/run-time-software/kinetis-developer-resources/ides-for-kinetis-mcus/opensda-serial-and-debug-adapter:OPENSDA) (besucht am 29. 04. 2018).

[13] NXP Semiconductors. *ABIC 2 Application Note*. Rev. 3.02. 2011.

[14] NXP Semiconductors. *ABIC 2 Data Sheet*. Rev. 2015-10-8. 2015.

[15] NXP Semiconductors. *Best Practices for Software Development*. Ver. 17.1. 2014.

[16] NXP Semiconductors. *C Coding Styles*. 2012.

[17] NXP Semiconductors. *NJJ29C0D - SPI Command Set*. Rev. 11 Shortened Version. 2016.

[18] NXP Semiconductors. *NXP HITAG [online].* URL: https://www.nxp.com/products/identification-and-security/smart-label-and-tag-ics/hitag:MC_42027 (besucht am 29. 04. 2018).

[19] NXP Semiconductors. *PCF7936A HT2 Training Slides*. Shortened Version. 2018.

[20] NXP Semiconductors. *S32K1xx Series Reference Manual*. Rev. 6. 2017.

[21] NXP Semiconductors. *TJA1027 Data Sheet*. Rev. 2. 2013.

[22] Oracle. *Class Observable [online].* URL: https://docs.oracle.com/javase/7/docs/api/java/util/Observable.html (besucht am 28. 08. 2018).

[23] Oracle. *Interface ActionListener [online].* URL: https://docs.oracle.com/javase/7/docs/api/java/awt/event/ActionListener.html (besucht am 28. 08. 2018).

[24] Oracle. *Interface Observer [online].* URL: https://docs.oracle.com/javase/7/docs/api/java/util/Observer.html (besucht am 28. 08. 2018).

[25] Nicolas Pannwitz. *RxTx für Java [online].* URL: https://pic-projekte.de/blog/rxtx-fur-java/ (besucht am 10. 08. 2018).

[26] Prof. Dr. Christian Johner. *Software-Architektur [online].* URL: https://www.johner-institut.de/blog/tag/software-architektur/ (besucht am 11. 08. 2018).

[27] Prof. Dr. Christian Johner. *Systemarchitektur [online].* URL: https://www.johner-institut.de/blog/systems-engineering/systemarchitektur-fuer-medizinprodukte/ (besucht am 11. 08. 2018).

[28] Prof. Dr. Christian Johner. *V-Modell vs. Wasserfallmodell für Hardware- und Softwareentwicklung [online].* URL: https://www.johner-institut.de/blog/iec-62304-medizinische-software/v-modell/ (besucht am 11. 08. 2018).

[29] Prof. Dr. Heike Neumann. *Skript - Kryptographie in Soft- und Hardware.* 2018.

[30] Rheinwerk Verlag GmbH. *1.3 Eigenschaften von Java [online].* URL: http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_01_003.htm (besucht am 10. 08. 2018).

[31] RXTX community. *Beispielprogramm - Two way communcation with the serial port [online].* URL: http://rxtx.qbang.org/wiki/index.php/Two_way_communcation_with_the_serial_port (besucht am 10. 08. 2018).

[32] RXTX community. *RXTX Wiki - FAQ [online].* URL: http://rxtx.qbang.org/wiki/index.php/FAQ (besucht am 10. 08. 2018).

[33] RXTX community. *RXTX Wiki - Projects [online].* URL: http://rxtx.qbang.org/wiki/index.php/Projects (besucht am 10. 08. 2018).

[34] SparkFun Electronics®. *SPI - Serial Peripheral Interface [online].* URL: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi (besucht am 29. 04. 2018).

[35] Christian Wüst Spiegel. *Zukunft: Bei Anruf stopp, 1998 [online].* URL: http://www.spiegel.de/spiegel/spiegelspecial/d-7240000.html (besucht am 29. 04. 2018).

[36] Vector Informatik GmbH. *Einführung in LIN [online].* URL: https://elearning.vector.com/index.php?wbt_ls_kapitel_id=1329934&root=376493&seite=vl_lin_introduction_de (besucht am 29. 04. 2018).

[37] *Zulieferpyramide Automobilbranche [online].* URL: https://ecosio.com/de/blog/2017/03/10/Was-ist-ein-Tier-Supplier-oder-Tier-Lieferant/ (besucht am 29. 04. 2018).

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, 8. Oktober 2018	
Ort, Datum	Unterschrift