

Charging Station for ISO / IEC 15118 Protocol

*Building working smart networked charging station with support for both ISO 15118 and IEC 61851*

Bachelors Project

**Presented by**

Jiztom Kavalakkatt Francis

Nivas Gokul Manimurugesan

Raphael Scholz

**Supervisor** Prof. Dr.-Ing. Ansgar Meroth, Hochschule Heilbronn

Hochschule Heilbronn

Faculty of Mechanics und Electronics

Automotive Systems Engineering

# Contents

[Contents III](#_Toc476173417)

[Directory of Tables V](#_Toc476173418)

[List of Figures V](#_Toc476173419)

[1 Introduction 1](#_Toc476173420)

[1.1 Short Version 1](#_Toc476173421)

[1.2 Task 2](#_Toc476173422)

[1.3 Kapitelübersicht 3](#_Toc476173423)

[2 Stand der Technik 4](#_Toc476173424)

[2.1 IEC 62196: Fahrzeugstecker 4](#_Toc476173425)

[2.2 IEC 61851: Ladevorgang 6](#_Toc476173426)

[2.3 ISO/IEC 15118 8](#_Toc476173427)

[2.3.1 Open V2G Projekt 9](#_Toc476173428)

[2.4 Powerline Kommunikation 10](#_Toc476173429)

[2.5 Zustandsautomaten 11](#_Toc476173430)

[3 Hardware 13](#_Toc476173431)

[3.1 Anforderungen/Eigenschaften 13](#_Toc476173432)

[3.2 EVAchargeSE 14](#_Toc476173433)

[3.2.1 Aufbau 15](#_Toc476173434)

[3.2.2 Erstellen serial\_Programming-Verzeichnisses 16](#_Toc476173435)

[3.3 Kommunikation zwischen Rechner und EVAchargeSE 18](#_Toc476173436)

[3.3.1 Flashen eines Programms 18](#_Toc476173437)

[3.3.2 Kompilieren und Ausführen eines Programms 20](#_Toc476173438)

[3.4 Aufgetretene Probleme und Maßnahmen zur Problembehebung 22](#_Toc476173439)

[3.5 Verzeichnisse verwendeter Befehle 23](#_Toc476173440)

[4 Software 25](#_Toc476173441)

[4.1 Kommunikation innerhalb des Boards 27](#_Toc476173442)

[4.2 IP-basierter Kommunikationsaufbau zwischen EV und EVSE 28](#_Toc476173443)

[4.3 Erstellen eines ISO/IEC 15118 Stacks 30](#_Toc476173444)

[4.3.1 Erweiterung um einen Zustandsautomaten 32](#_Toc476173445)

[5 Ergebnisse 34](#_Toc476173446)

[6 Zusammenfassung & Ausblick 37](#_Toc476173447)

[7 Anhang 38](#_Toc476173448)

[7.1 AC-Ladeablauf 38](#_Toc476173449)

[7.2 DC-Ladeablauf 40](#_Toc476173450)

[7.3 Programmablaufpläne der Ladesäule 42](#_Toc476173451)

[7.4 Programmablaufpläne des Elektrofahrzeugs 46](#_Toc476173452)

[8 Literaturverzeichnis 48](#_Toc476173453)

# Directory of Tables

Tabelle 2.1: Kontaktbezeichnungen 5

Tabelle 2.2: Zustände CP-PE (Wiki\_Stecker, 2016) 7

Tabelle 3.1: Beschreibung der in den Frames enthaltenen Parameter 16

Tabelle 3.2: Beschreibung der Services zur Kommunikation zwischen i.MX28 und KL02 17

Tabelle 3.3: I2SE-Befehle im Terminal 23

Tabelle 3.4: Linux Terminalbefehle (Shellbefehle, 2016) 24

# List of Figures

[Abbildung 2.1: Ladestecksysteme (Mültin, 2014) 5](#_Toc454570580)

[Abbildung 2.2: Spannung CP-PE (Lewandowski, 2014) 7](#_Toc454570581)

[Abbildung 2.3: Zuordnung der Kommunikationsinhalte zu den verschiedenen Ladestatus 8](#_Toc454570582)

[Abbildung 2.4: Programmablauf innerhalb des V2A-Codes (OpenV2G, 2016) 9](#_Toc454570583)

[Abbildung 2.5: moduliertes PWM-Signal 10](#_Toc454570584)

[Abbildung 2.6: Beispiel eines Zustandsautomaten (Bustamante, 2014) 11](#_Toc454570585)

[Abbildung 2.7: Implementierung eines Zustandsautomaten 12](#_Toc454570586)

[Abbildung 3.1: EVAchargeSE board (I2SE, 2016) 14](#_Toc454570587)

[Abbildung 3.2: Verbindung der Informationsleitungen 15](#_Toc454570588)

[Abbildung 3.3: Message Frames zwischen i.MX und KL02 (I2SE, 2016) 16](#_Toc454570589)

[Abbildung 3.4: FileZilla 19](#_Toc454570590)

[Abbildung 3.5: Erfolgreiche SSH-Verbindung 20](#_Toc454570591)

[Abbildung 3.6: Kompilieren und ausführen eines Programms 21](#_Toc454570592)

[Abbildung 3.7: Anzeige eines weiteren Powerline-Kommunikationssystems 22](#_Toc454570593)

[Abbildung 4.1: Ablauf der Softwareprogrammierung 25](#_Toc454570594)

[Abbildung 4.2: Typische Socket-Kommunikation(Groß, 2015) 28](#_Toc454570595)

[Abbildung 4.3: IP-Adressen der EVAchargeSE-Boards 30](#_Toc454570596)

[Abbildung 4.4: Zustandsautomat EVSE Teil 1 von 3 32](#_Toc454570597)

[Abbildung 4.5: Zustandsautomat EVSE Teil 2 von 3 32](#_Toc454570598)

[Abbildung 4.6: Zustandsautomat EVSE Teil 3 von 3 32](#_Toc454570599)

[Abbildung 5.1: Hardwareaufbau 33](#_Toc454570600)

[Abbildung 5.2: Anzeige des Linux-Terminals nach Starten von serial\_Programming 33](#_Toc454570601)

[Abbildung 5.3: Pegel des Control Pin bei Ladestatus A 34](#_Toc454570602)

[Abbildung 5.4: Pegel des Control Pin bei Ladestatus B 34](#_Toc454570603)

[Abbildung 5.5: Pegel des Control Pin bei Ladestatus C 35](#_Toc454570604)

[Abbildung 7.1: Nachrichtenübersicht für AC-Ladezyklus (1 von 2) 37](#_Toc454570605)

[Abbildung 7.2:Nachrichtenübersicht für AC-Ladezyklus (2 von 2) 38](#_Toc454570606)

[Abbildung 7.3:Nachrichtenübersicht für DC-Ladezyklus (1 von 2) 39](#_Toc454570607)

[Abbildung 7.4: Nachrichtenübersicht für DC-Ladezyklus (2 von 2) 40](#_Toc454570608)

[Abbildung 7.5: Programmablaufplan der Ladesäulenseite 41](#_Toc454570609)

[Abbildung 7.6: Programmablaufplan des Funktionsaufrufs State B1 Kommunikation 42](file:///C:\Users\Melanie\Desktop\HSHN\_Projektarbeit\abgabedok\Dokumentation.docx#_Toc454570610)

[Abbildung 7.7: Programmablaufplan des Funktionsaufrufs State C Kommunikation 43](file:///C:\Users\Melanie\Desktop\HSHN\_Projektarbeit\abgabedok\Dokumentation.docx#_Toc454570611)

[Abbildung 7.8: Programmablaufplan des Funktionsaufrufs State B2 Kommunikation 44](#_Toc454570612)

[Abbildung 7.9: Programmablaufplan des Elektrofahrzeugs (Teil 1) 45](file:///C:\Users\Melanie\Desktop\HSHN\_Projektarbeit\abgabedok\Dokumentation.docx#_Toc454570613)

[Abbildung 7.10: Programmablaufplan des Elektrofahrzeugs (Teil 2) 46](file:///C:\Users\Melanie\Desktop\HSHN\_Projektarbeit\abgabedok\Dokumentation.docx#_Toc454570614)

# Introduction

In this chapter a short summary of this work is described. In addition, the task description as well as an overview of the following chapters and their contents are given.

## Short Version

The present project work serves to set up a working model for the current standardization of the vehicle-column communication according to ISO / IEC 15118 along with the HMI display.

The built-up working model consists of two interconnected boards and a Raspberry Pi for teh display. Each communication user is represented by a circuit board.

The software used is derived from an existing stack and adapted for a defined application of ISO / IEC 115118.

According to the ISO variable parameters are listed as macros and documented, so that a change is possible at any time. If parameters are selected so that the resulting requirements can no longer be met by the respective other subscriber, a fault message is output.

## Task

The project is expected to result in a structure that demonstrates the communication staple of ISO / IEC 15118.

For this purpose an existing stack for the ISO is put into operation. The freely available stack is written so far for demonstration purposes of the software. For use at a demonstrator, it must be separated, reordered and modularized so that a stack for the vehicle as well as the load column side is created. The initially defined load parameters are defined as macros, so that a change is possible at any time. Furthermore, incorrect error messages are to be output in the event of incorrect inputs or not supporting functions of the charging column or the vehicle.

At the same time, a development environment to be used for demonstration purposes is to be established. First of all, hardware requirements must be defined. From the resulting specifications, approaches for implementation have to be found. The resulting results are evaluated and selected according to the previously defined viewpoints.

As soon as the selected hardware is ready for operation, it must be commissioned and all functions required for commissioning must be checked.

Finally, the functional stack is flashed to the development environment and tested for functionality.

The resulting project results are defined as follows:

* Integration into vehicle charging columns Communication according to ISO / IEC 15118
* Separation of an existing stack in the vehicle and loading column side
* Commissioning and demonstration of the separate stack
* Selection and commissioning of a hardware suitable for demonstration
* Start up the demonstrator with the previously created communication pack
* documentation of the results

## Kapitelübersicht

In *Kapitel* 1 werden eine Kurzfassung der vorliegenden Arbeit sowie die Aufgabenstellung und diese Kapitelübersicht dargestellt.

*Kapitel* 2 gibt einen Überblick über vorangegangene Arbeiten an der Hochschule sowie Dokumente, die zur Einarbeitung in das Thema beigetragen haben. Weiterhin werden verwendete Komponenten aus Hard- und Softwarebereich beschrieben.

Eine Übersicht der Hardwareanforderungen sowie die Inbetriebnahme ist in *Kapitel* 3 zu finden. Weiterhin wird ein Überblick über die Arbeit mit dem Linux-Terminal gegeben. Dies ist wichtig für die Kommunikation mit dem Board, besonders im Bereich der Datenverwaltung sowie für das Kompilieren und Starten eines Programms.

In Kapitel 4 werden die verschiedenen Teile der erstellten sowie der bearbeiteten Software beschrieben. Darin sind neu erstellte Files zur Kommunikation innerhalb des Boards sowie übernommene Programmteile zur Kommunikation nach ISO/IEC 15118. Neu erstellte Bestandteile enthalten Funktionen, welche im Besonderen für die Signalgenerierung nach IEC 61851 wichtig sind. Zum Datenaustausch der Kommunikationsteilnehmer nach ISO/IEC 15118 ist zudem ein Verbindungsaufbau notwendig, welcher ebenfalls in diesem Kapitel dargestellt gebracht wird.

Die durch diese Arbeit erreichten Ergebnisse werden in *Kapitel* 5 wiedergegeben. Zuletzt wird eine Zusammenfassung der Arbeit sowie ein Ausblick des Projektes in *Kapitel* 6 wiedergegeben.

Teile der ISO/IEC 15118 werden zusammen mit Programmablaufplänen in *Kapitel* 7 dargestellt. In beiden Fällen geschieht dies, um einen Überblick der Anforderungen nach DIN/ISO und der resultierenden Programme zu erhalten.

# Stand der Technik

Das Kapitel gibt einen Überblick über Arbeiten und Informationen, auf welche das Projekt aufarbeitet. Dazu gehören unter anderem die bisherige Art des Ladens eines Fahrzeugs sowie der bisherige Informationsaustausch, verschieden verwendete Fahrzeugstecker zum Laden elektronischer Fahrzeugen. Weiterhin informativ für diese Arbeit sind eine Studienarbeit, welche die ISO 15118 genauer beschreiben soll sowie eine Dissertation von Dr.-Ing Marc Mültin, welche sich mit dem Elektrofahrzeug als „flexibler Verbraucher und Energiespeicher im Smart Home“ beschäftigt.

## IEC 62196: Fahrzeugstecker

Steckertypen sowie Lademodis von Elektrofahrzeugen werden durch die International Electrotechnical Commission in der IEC 62196 definiert (Wiki\_Stecker, 2016).

Der zweite Teil der Norm wurde 2011 veröffentlicht und beinhaltet verschiedene Steckertypen. Darin enthalten sind drei der zu diesem Zeitpunkt meist genutzten Ladestecker.

Der Typ 1 Ladestecker, welcher in Abbildung 2.1 dargestellt ist, übernimmt seine Spezifikation aus der SAE J1772. Diese wurde erstmalig im Jahr 1996 von der Society of Automotive Engineers veröffentlicht und wird seither von dieser erweitert und gepflegt. Der Nachteil dieses Steckertyps findet sich in den Kontakten, da diese ein dreiphasiges Laden mit Wechselstrom nicht ermöglichen.

Typ 2 der genormten Ladestecker ist die aktuell am meisten verbaute Art der Ladestecksysteme und ebenfalls in Abbildung 2.1 zu finden. Der Stecker findet seine Ursprünge durch eine Zusammenarbeit des Steckerherstellers Mennekes mit dem Stromversorger RWE sowie dem Automobilhersteller Daimler. Die Namensgebung des Mennekes-Steckers erhält dieser somit durch seinen Hersteller.

Der dritte in der Norm aufgenommene Steckertyp, der EV-Plug-Alliance, wurde von einem Konsortium unter der Führung französischer und italienischer Firmen definiert. Aufgrund der geringen Nachfrage wurde die weitere Produktion des Steckers eingestellt.

Bei allen definierten Steckertypen sind die in Typ 1 definierten Signalkontakte CP (Control Pilot) und PP (Proximity Pilot) enthalten. Welche einen Ladevorgang nach IEC 61851 ermöglichen.

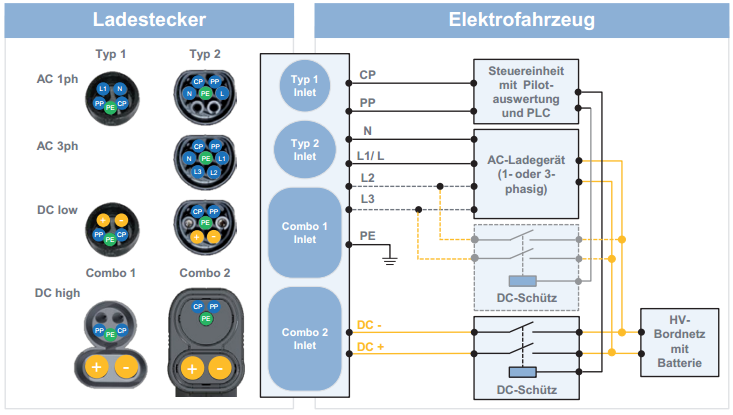


Abbildung .: Ladestecksysteme (Mültin, 2014)

Die in Abbildung 2.1 dargestellten Signalkontakte sind folgendermaßen definiert:

|  |  |  |
| --- | --- | --- |
| Abkürzung | Kontakt | Funktion |
| CP | Control Pilot | Steuersignale Ladesäule🡪Elektrofahrzeug |
| PP | Proximity Pilot | Anwesenheit eines Ladekabels prüfen |
| N | Neutralleiter | Für Wechselstrom-Ladevorgang |
| L1, L2, L3 | Stromführende Phasen | Zum Wechselstrom-Laden mit einer (L1/L) oder drei (L1, L2, L3) Phasen |
| PE | Protective Earth | Schutzleiter |
| DC+/- | Stromführende Phasen | Zum Gleichstrom-Laden |

Tabelle .: Kontaktbezeichnungen

## IEC 61851: Ladevorgang

Vor der Definition eines Ladevorgangs nach ISO/IEC 15118 wurden für den Ladevorgang notwendige Ladeparameter mithilfe eines PWM-Signals nach IEC 61851 definiert. Zur Bestimmung der zum Laden benötigten Parameter werden die Signale der in *Kapitel* 2.1 beschriebenen Kontakte Control Pilot (CP), Protective Earth(PE) und Proximity Pin(PP) benötigt.

Zum Laden des Fahrzeugs werden zunächst beide Kommunikationsteilnehmer miteinander verbunden. Daraufhin wird von der Seite der Ladesäule ein 1kHz-Signal mit 12V auf dem CP-Kontakt erzeugt. Die Pulsweite des Signals gibt an, welche maximale Leistung von der Ladesäule zur Verfügung gestellt werden kann. Dabei gilt bei 10% max. 10A, 25% 16A, 50% max. 32A und 90% Schnellladung (Wiki\_Stecker, 2016).

Fahrzeugseitig werden im weiteren Verlauf Widerstände zwischen CP und PE bzw. PP und PE geschalten. Durch verschieden schaltbare Pegel der Spannung zwischen dem CP- und PP-Kontakt entsprechend Abbildung 2.2 werden dabei verschiedene Ladezustände angegeben. Dabei ist zu beachten, dass der negative Spannungswert dauerhaft -12V beträgt, und sich lediglich die positiven Werte ändern. Eine Definition der einzelnen Zustände ist Tabelle 2.2.

Zuletzt gibt ein Fahrzeugseitiger Widerstand zwischen dem PP- und dem PE- Kontakt den maximal möglichen Ladestrom des Elektrofahrzeugs an. Dabei gilt je größer der verwendete Widerstand gewählt wird, desto geringer ist der maximale Ladestrom. Konkret bedeutet das für einen 1,5kΩ- Widerstand einen maximalen Ladestrom von 13A, maximal 20A bei einem Widerstand von 680Ω, bei 220Ω beträgt der maximale Ladestrom 32A und 63A bei 100Ω.

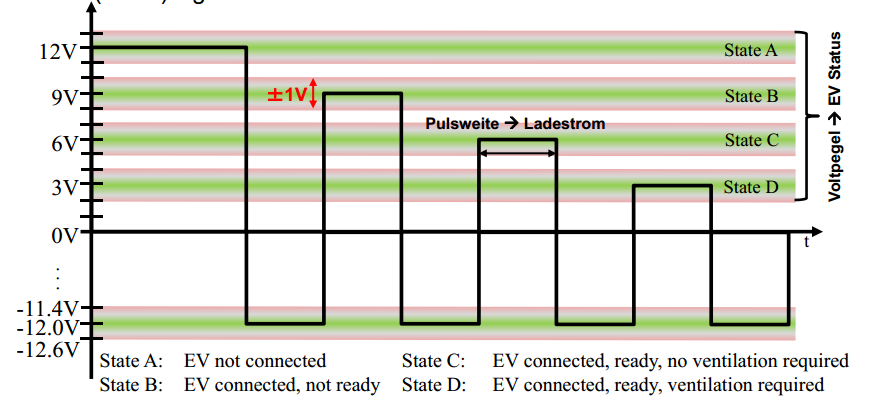


Abbildung .: Spannung CP-PE (Lewandowski, 2014)

|  |  |  |
| --- | --- | --- |
| Pegel | Zustand | Zustandsbeschreibung |
| 12 ± 1 V | State A | Elektrofahrzeug nicht verbunden |
| 9 ± 1 V | State B | Elektrofahrzeug verbunden, nicht Ladebereit |
| 6 ± 1 V | State C | Elektrofahrzeug verbunden, ladebereit |
| 3 ± 1 V | State D | Elektrofahrzeug verbunden, ladebereit, Ventilation notwendig |
| 0 ± 1 V | State E | Netzproblem, PP-Kurzschluss gegen Erde |
| -12V | State F | Fahrzeug nicht verfügbar, Error |

Tabelle .: Zustände CP-PE (Wiki\_Stecker, 2016)

## ISO/IEC 15118

Die Internationale Organisation für Normung (ISO) und die internationale elektronische Kommission (IEC) haben 2009 damit begonnen, die Standardisierung eines „digitalen IP-basierten Kommunikationsprotokolls“ zwischen Elektrofahrzeug und Ladestation zu beschreiben (Mültin, 2014). Diese soll einen „Plug-and-Charge“-Mechanismus zur Authentifizierung, Autorisierung, Abrechnung sowie zur Lastkontrolle darstellen, sodass die zur Ladefreigabe benötigten Parameter im Fahrzeug abgespeichert sind und der Benutzer lediglich die beiden Kommunikationsparteien miteinander verbinden muss. Die einzelnen Kommunikationsinhalte werden dabei den Pegelspannungen des Control Pin Signals aus Kapitel 2.2 entsprechend Abbildung 2.3 zugeordnet.

Abbildung .: Zuordnung der Kommunikationsinhalte zu den verschiedenen Ladestatus

Der vollständige Ablaufplan eines Kommunikationsstacks zum AC- bzw. DC-Laden eines Elektrofahrzeugs nach ISO/IEC 15118 findet sich in Abbildung 7.1 bis Abbildung 7.4. Eine Übersicht der in den Nachrichten enthaltenen Variablen innerhalb des Wechselstrom-Kommunikationsstack wird zusammen mit einem Überblick über die ISO/IEC 15118 in einer vorangegangenen Studienarbeit beschrieben (Barth, 2015).

### Open V2G Projekt

Ein bereits weit voran geschrittenes Beispiel zur Implementierung eines Kommunikationsstack nach ISO/IEC 15118 wurde bereits durch die Unterstützung der Siemens Corporate Technology als Open Source Projekt initiiert (OpenV2G, 2016). Dabei werden sowohl Ladesäulen- als auch Fahrzeugseite in einen Programmcode dargestellt und die Nachrichten entsprechend Abbildung 2.4 nacheinander generiert, geprüft und die nächste Nachricht generiert. Zum aktuellen Stand (Version 0.9.3) lassen sich daraus der Ablauf der einzelnen Requests und Responses sowie die Nachrichteninhalte zum Gleich- und Wechselstromladen sehr gut ableiten. Es gehört zu den Zielen der vorliegenden Arbeit, diesen Code so aufzuteilen, dass für jeden Kommunikationsteilnehmer ein Programm vorliegt.

****

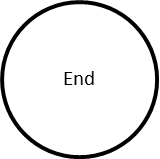
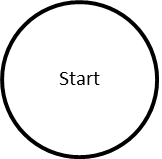
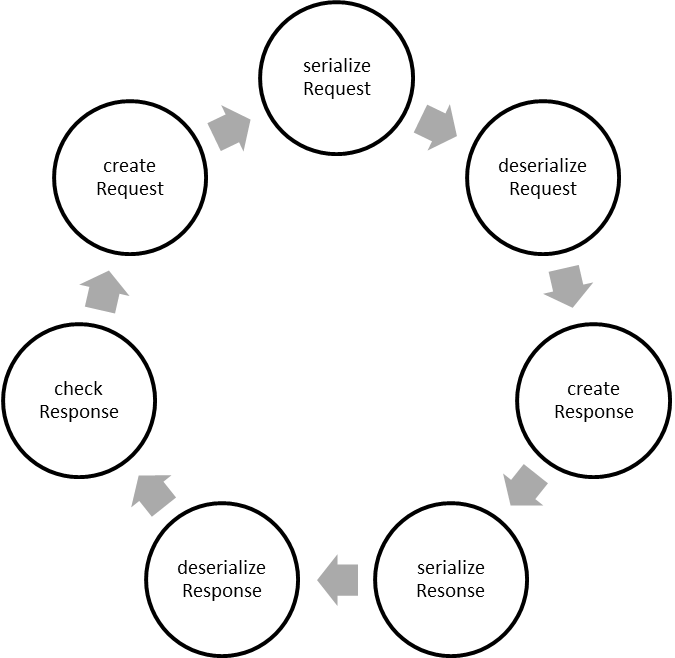
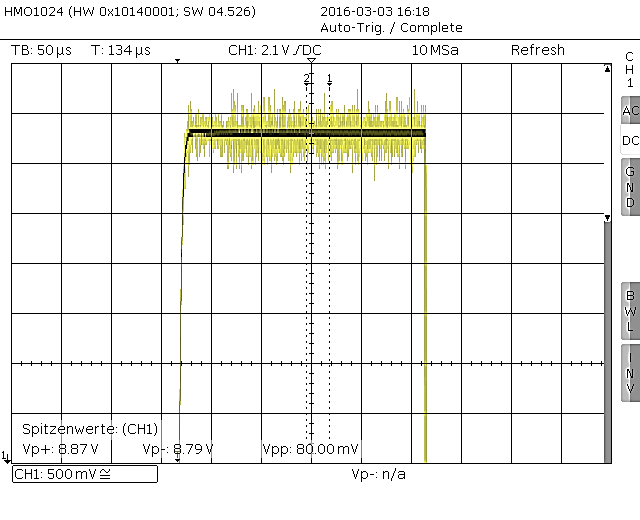


Abbildung .: Programmablauf innerhalb des V2A-Codes (OpenV2G, 2016)

## Powerline Kommunikation

Allgemein ist die Powerline Kommunikation (PLC) beschrieben als „Übertragung von Daten über das Stromkabel“ (El-Ko, 2016). Dabei wird diese Form der Signalübertragung schon länger zum Beispiel in Sprechanlagen eingesetzt. Durch die Verwendung dieser Kommunikationsvariante müssen keine weitere Pins und Leitungen in den vorhandenen Ladesteckern und -kabeln definiert und nachgerüstet werden.

Bei der Powerline-Kommunikation werden die zu übermittelnden Daten im Hochfrequenten Bereich auf das Stromkabel moduliert und Empfängerseitig demoduliert. Im Falle der ISO/IEC 15118 erfolgt die Modulation und Demodulation des Kommunikationsprotokolls entsprechend Abbildung 2.5 auf dem PWM-Signal des CP-Leiters.



**Abbildung 2.5: moduliertes PWM-Signal**

## Zustandsautomaten

Ein Zustandsautomat besteht aus Zuständen, Zustandsübergängen und Aktionen. Mithilfe dieser Werkzeuge soll die Steuerung eines Systems realisiert werden, welches Ereignisse der Vergangenheit, des aktuellen Zeitpunktes sowie der Zukunft berücksichtigt. Jedem Zustand sind dabei Aktionen zugeordnet, die beim Eintreten oder Verlassen dessen ausgeführt werden. Es muss zu jedem Zeitpunkt der Laufzeit des Systems ein Zustand definiert sein. Ein Zustandsübergang hingegen beschreibt die Verbindungen der einzelnen Zustände zueinander sowie das Ereignis, welches Eintreten muss um zwischen den Zuständen zu wechseln.

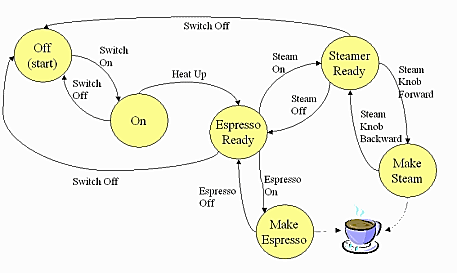


Abbildung .: Beispiel eines Zustandsautomaten (Bustamante, 2014)

Zur Veranschaulichung eines solchen Zustandsautomaten dient ein Kaffeeautomat entsprechend Abbildung 2.6. Dabei beginnt der Zustandsautomat mit dem Startzustand, welcher im vorliegenden Beispiel der ausgeschaltete Zustand der Kaffeemaschine ist. Hier ist eine Zustandsänderung lediglich durch den Übergang „Switch on“ möglich. Je nach Eingabe des Benutzers kann der Automat in den Zustand „Espresso ready“, „Steamer ready“ oder „OFF“ rücken.

Besonders gut zeigt dieses Beispiel die Miteinbeziehung von verschiedenen Zeitformen. Um Betriebsbereit zu sein musste die Kaffeemaschine im Vorfeld eingeschaltet und auf eine definierte Temperatur gebracht werden. Welcher Zustand im weiteren Verlauf angenommen wird hängt von nicht vorhersehbaren Ereignissen ab. Dabei wird auch deutlich gezeigt, dass der Automat nicht zu jedem Zeitpunkt die einzelnen Zustände nach Belieben aktivieren kann. Um einen bestimmten Zustand annehmen zu können, muss dieser mit dem einem Zustandsübergang von dem aktuellen Zustand aus gehend verbunden sein. Somit kann im gegebenen Beispiel kein Kaffee zubereitet werden solange sich die Kaffeemaschine im „ON“-Zustand befindet.

Die Programmierung eines Zustandsautomaten lässt sich mit der switch case Funktion realisieren. Ein Grundzustand wird bereits im Vorfeld definiert. Sobald ein Ereignis, welches eine Zustandsänderung hervorrufen könnte auftritt, wird die Funktion gestartet. Dabei wird der aktuell definierte Zustand abgefragt und in dem dazugehörigen case die Zustandsänderung definiert, um einen neuen Zustand erreichen zu können. Ein Beispiel zu dem Kaffeeautomaten Abbildung 2.6 zeigt Abbildung 2.7.

switch (current\_state)

case OFF: if (switch\_OFF) current\_state = switch\_ON; break;

case ON: if(Heat\_up) current\_state = Espresso\_ready; break;

case Espresso\_ready: if(Steam\_on) current\_state = Steamer\_ready;

else if(Espresso\_on) current\_state = Make Espresso;

break;

…

…

Abbildung .: Implementierung eines Zustandsautomaten

# Hardware

Ein Bestandteil der vorliegenden Arbeit ist der Aufbau eines Demonstrators. Für die Hardware wird aus diesem Grund jeweils Fahrzeug- und Ladesäulenseitig eine Platine benötigt, welche ein Signal nach IEC 61851 generieren und gleichzeitig eine Powerline basierte Kommunikation aufbauen kann.

## Anforderungen/Eigenschaften

Für die Realisierung des Demonstrators werden somit insgesamt zwei Platinen benötigt. Dabei soll eine für den Aufbau eines Ladesäulendemonstrators und die zweite für das Elektrofahrzeug verwendet werden. Beide Platinen benötigen zur Demonstration die in *Kapitel* 2.3 benötigten Pins zum Kommunikationsaufbau nach ISO/IEC 15118. Weiterhin müssen Leitungen zum Steuern der Stromzufuhr sowie dem Ansprechen der Verriegelung des Ladesteckers, um ein Abziehen während des Ladevorgangs zu verhindern, vorhanden sein.

Zur Kommunikation nach IEC 61851 muss Ladesäulenseitig die Möglichkeit bestehen, ein PWM-Signal zu generieren und die positiven Pegel von diesem zu verändern. Weiterhin muss dieser Kommunikationsteilnehmer die Pegel zu den anderen Pins messen können. Die Fahrzeugseite benötigt zur Ermittlung des aktuellen Ladezustandes die positive Spannung sowie die Pulsweite von diesem um den maximalen Ladestrom zu bestimmen. Weiterhin müssen zur Rückmeldung Widerstände zwischen den CP-, PP- und PE-Kontakten geschalten werden können.

Eine für dieses Projekt wichtige Eigenschaft ist zudem die Möglichkeit der Powerline-Kommunikation. Diese muss von Ladesäulen- und Elektrofahrzeugseite aus unterstützt werden, um die Nachrichten entsprechend ISO/IEC 15118 auf das Signal des Control Pins zu modulieren.

## EVAchargeSE

Das EVAchargeSE Board von I2SE wurde für die Kommunikation nach ISO/IEC 15118 entwickelt. Der Vorteil an dem Board besteht darin, dass es sowohl Fahrzeug- als auch Ladesäulenseitig eingesetzt werden kann. Dadurch können für den Demonstrationsaufbau zwei identische Platinen mit unterschiedlicher Konfiguration verwendet werden.

Auf dem Board ist ein i.MX28 verbaut mit einem Linux Betriebssystem. Die beiden zur Kommunikation benötigten GPIO´s CP, PP und PE sind an einen Stecker herausgeführt. Programme zum Kommunikationsstack können mithilfe der vorhandenen Ethernet-Schnittstelle übertragen werden. Diese werden in dem 4GB Speicher abgelegt und von dort aus gestartet.

Ein auf dem Board enthaltener KL02 Mikrocontroller kann von dem i.MX28 per UART angesprochen werden. Bei der Master/Slave-Verbindung wird der KL02 als Slave angesprochen und ist für das Generieren und Messen der Signale nach IEC 61851 zuständig. Weiterhin wird er verwendet um GPIO´s anzusprechen, welche einen Verriegelungsmechanismus der Ladekabel starten sollen.

Weiterhin ist das EVAchargeSE Board PLC-fähig. Dabei werden die zu versendenden Nachrichten über TCP/IP-sockets versendet. Ein QCA 7000 –Chip moduliert die gesendeten Daten im weiteren Verlauf auf das PWM-Signal des KL02.

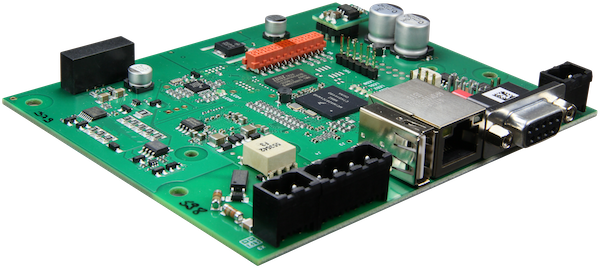


Abbildung 3.1: EVAchargeSE board (I2SE, 2016)

### Aufbau

Zur Inbetriebnahme der Platinen werden diese zunächst im Labor aufgebaut. Die Spannungsversorgung beträgt dabei 12V bei 0,3A. Zur Kommunikation zwischen den EVAchargeSE Boards werden die Anschlüsse von CP, PP und PE entsprechend Abbildung 3.2 miteinander verbunden. Weiterhin zeigt der Aufbau eine Ethernet-Verbindung zu einem Router und einem Rechner mit Linux-Betriebssystem. Dadurch wird ein Zugriff auf die Linux-Systeme, die sich auf den Boards befinden, ermöglicht. Zu diesem Zeitpunkt ist dieser Zugriff sinnvoll, da dies eine Programmierung in einer Umgebung wie Eclipse auf einem Rechner ermöglicht. Die Programme werden mithilfe von Eclipse zunächst auf dem Rechner gestartet, um die Lauffähigkeit zu prüfen. Im weiteren Verlauf werden die Programme für Fahrzeug- und Ladesäulenseite auf die jeweiligen Boards geflasht, kompiliert und von dort aus gestartet. Eine genaue Anleitung hierzu ist in Kapitel 3.3 zu finden.

Router

EV

EVSE

Linux

Eth

Eth

Eth

PE CP

PP CP

CP CP

CP

PP CP

PE CP

Abbildung .: Verbindung der Informationsleitungen

### Erstellen serial\_Programming-Verzeichnisses

Das serial\_Programming- Verzeichnis ist zuständig für die Kommunikation zwischen dem i.MX 28 und dem KL02, welcher unter anderem für die Generierung des PWM-Signals zuständig ist. Dadurch sind die Einstellungen des Microcontrollers wichtig bei der Unterscheidung zwischen Elektrofahrzeug und Ladestation. Wie bereits in Kapitel 3.2 erwähnt, erfolgt die Kommunikation zwischen den beiden Microcontrollern per UART. Dabei ist der i.MX 28 als Master und der KL02 als Slave deklariert. Dies bedeutet, dass der KL02 auf eine Befehlsaufforderung des ARM-Prozessors wartet, und auf diese Antwortet. Die Inhalte der übertragenen Nachrichten sind in Frames gegliedert. Dies bringt den Vorteil einer geringen zu übertragenden Datenmenge mit sich. Ein Frame setzt sich dabei entsprechend Abbildung 3.3 sowohl Fahrzeug- als auch Ladesäulenseitig zusammen. Die darin enthaltenen Parameter werden in Tabelle 3.1 genauer definiert.

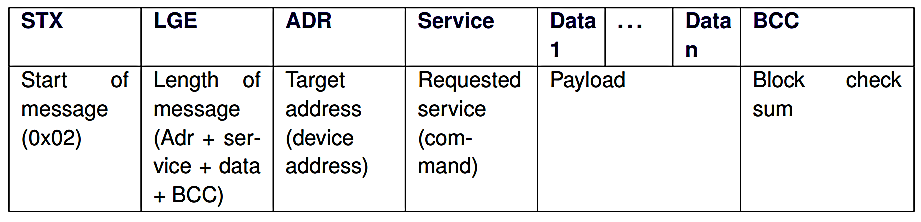


Abbildung .: Message Frames zwischen i.MX und KL02 (I2SE, 2016)

|  |  |
| --- | --- |
| Benennung im Frame | Beschreibung |
| STX | Zeigt den Start eines neuen Frames an |
| LGE | Gibt die Anzahl der Bytes an, welche in ADR, Service, Data und BBC enthalten sind |
| ADR | Nummer des Zielgerätes. |
| Service | Beschreibt den durchzuführenden Service entsprechend Tabelle 3.2 |
| Data 1…n | Enthält die zur Durchführung des Services benötigten Parameter bzw. gibt Angeforderte Werte zurück |
| BCC | Prüfsumme: Zur Vermeidung von Übertragungsfehlern.  Wird berechnet indem alle im Frame enthaltenen Bytes durch ein logisches XOR miteinander verbunden werden. |

Tabelle .: Beschreibung der in den Frames enthaltenen Parameter

|  |  |
| --- | --- |
| Service-ID | Servicebeschreibung |
| 0x01 | Testen der Verbindung. Hat als Rückgabewert Soft- und Hardwareversion des EVAchargeSE Boards |
| 0x04 | Weiterer Test der Verbindung. Enthält als Rückgabewert unter anderem den letzten Grund eines Resets. Eine Auflistung der möglichen Gründe lässt sich im Datenblatt finden. |
| 0x10 | Misst die Frequenz des PWM-Signals |
| 0x11 | Ändert die Frequenz des PWM-Signals |
| 0x12 | Definiert ob ein PWM-Signal generiert wird.  Diese Einstellung ist wichtig für die Unterscheidung zwischen Elektrofahrzeug und Ladesäule, da das Signal nur Ladesäulenseitig generiert werden darf. |
| 0x14 | Misst sowohl positiven als auch negativen Pegel des CP-Pins |
| 0x15 | Ändert den positiven Pegel des CP-Signals. Dieser Service darf nur Ladesäulenseitig aufgerufen werden, da dieser Parameter Fahrzeugseitig für die Bestimmung des aktuellen Ladezustandes wichtig ist. |
| 0x17/0x18 | Spricht Pins zur Ansteuerung von Motoren zur Ladekabelverriegelung an |
| 0x1A | Gibt den aktuellen Status der Motoren zur Ladekabelverriegelung an |
| 0x20 | Aktiviert eine automatische Messung und Rückgabe des PWM-Signals |
| 0x31 | Sorgt für eine Unterbrechung der Verbindung |
| 0x33 | Initiiert einen Neustart des KL02 |
| 0x50 | Definiert den Wert des Widerstandes zwischen PP und PE |
| 0x51 | Aktiviert den Widerstand zwischen PP und PE. Dieser Service darf nur Fahrzeugseitig aktiviert werden. Dadurch wird der maximale Ladestrom des Elektrofahrzeugs definiert. |
| 0x52 | Misst die Spannung zwischen PP und PE |

Tabelle .: Beschreibung der Services zur Kommunikation zwischen i.MX28 und KL02

## Kommunikation zwischen Rechner und EVAchargeSE

Auf dem Board ist ein Debian Betriebssystem von dem Hersteller des Boards I2SE installiert. Dieses „verwendet den Linux-Betriebssystemkern, aber die meisten grundlegenden Systemwerkzeuge stammen aus dem GNU-Projekt“ (Debian, 2016). Um einen Kommunikationsaufbau zu erleichtern wird für Arbeiten mit dem EVAchargeSE ausschließlich das Linux-System Ubuntu verwendet. Durch das Terminal wird dem Benutzer nach kurzer Einarbeitungszeit ein schneller Zugriff, Informations- und Datenaustausch auf andere verbundene Teilnehmer des Routers ermöglicht.

### Flashen eines Programms

Bevor ein Programm gestartet werden kann muss es auf den Microcontroller, auf welchem es im späteren Verlauf kompiliert und durchgeführt werden soll, geflasht werden.

Für Programme mit wenigen Files konnte hierfür das Secure File Transfer Protocol (SFTP) verwendet werden. Dieses Protokoll läuft auf Basis von Secure Shells und baut eine sichere Verbindung zur Übertragung von Daten auf. Im gegebenen Fall können Daten mithilfe von SFTP an die EVAchargeSE Boards übermittelt mithilfe eines Linux-Terminals übermittelt werden.

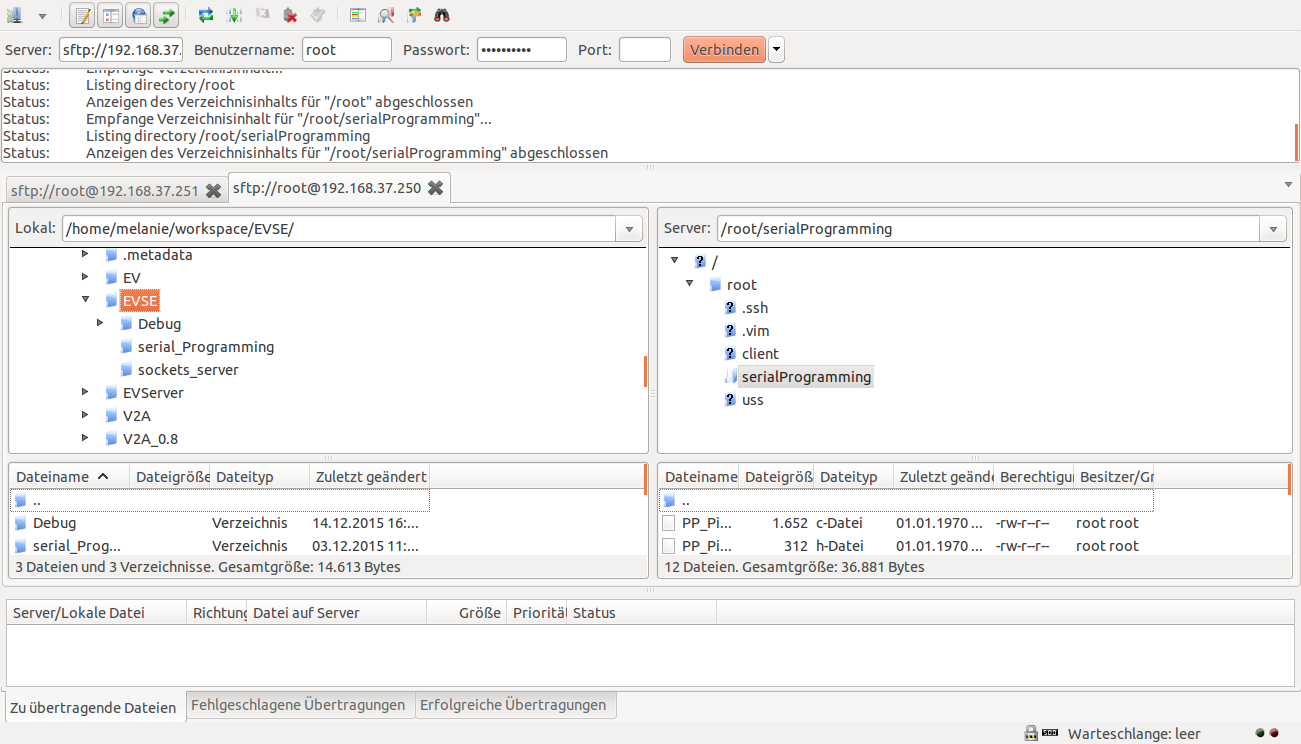
Zum Starten einer SFTP Verbindung wird der Befehl „sftp <Username>@<IP-Adresse>“ verwendet. Ist ein Teilnehmer mit den Kenndaten erreichbar wird eine Passworteingabe verlangt. Sobald dieses korrekt ist werden die im Terminal initiierten Befehle von dem User des Geräts der angegebenen IP-Adresse ausgeführt.

Somit kann Beispielsweiße durch den Befehl „cd“ das aktuelle Verzeichnis auf dem Board gewechselt werden. Soll jedoch ein anderes Verzeichnis auf dem Linuxsystem des Rechners aufgerufen werden, wird der Befehl mit einem Ausrufezeichen vor dem Befehl versehen („!cd“).

Mithilfe des Befehls „put <Dateiname>“ können Dateien von dem Linux-Rechner zum Board kopiert werden. Eine Kopie von einer auf dem Board existierenden Datei kann auf das Rechnerverzeichnis durch den Befehlssetz „get <Dateiname>“ geschrieben werden.

Aufgrund der steigenden Anzahl an Dateien ist eine solche Vorgehensweise zum Übertragen der einzelnen Files sehr zeitaufwendig. Aus diesem Grund werden Daten im weiteren Verlauf mithilfe des Programms FileZilla übertragen. Der FTP-Client ermöglicht eine einfache und übersichtliche Übertragung mehrerer Datenfiles. Gleichzeitig wird dem Benutzer durch die grafische Benutzeroberfläche die Möglichkeit geboten, eine Übersicht der vorhandenen Ordnerstrukturen und –verzeichnisse zu bekommen. Dadurch können lokale Files sowie die des verbundenen Kommunikationspartners aus verschiedenen Ordnern in kurzer Zeit gefunden und übertragen werden.

Zum Übertragen von Daten wird das Programm gestartet und die IP-Adresse, Benutzername, Passwort sowie Port des EVAchargeSE Boards entsprechend Abbildung 3.4 in die dafür vorgesehenen Felder eingetragen. Sobald eine Verbindung hergestellt ist erscheint neben der Ordnerstruktur des lokalen Rechners eine solche auch für das mit dem Rechner verbundenen Board. Durch Drag and Drop können nun Dateien in beide Richtungen der Verbindung ausgetauscht werden (FileZilla, 2016).



IP Adresse

192.168.37.250

192.168.37.251

Benutzername

root

Passwort:

zebematado

Port

22

Tabs für mehrere Verbindungen

Lokale Ordner

EVAchargeSE

Abbildung .: FileZilla

### Kompilieren und Ausführen eines Programms

Zum Kompilieren und Ausführen eines Programms muss zunächst eine Verbindung zwischen lokalem Rechner und dem jeweiligen Entwicklungsboard hergestellt werden. Um lokal eine entfernte Kommandozeile verfügbar machen zu können, wird eine Secure Shell (SSH) verwendet. Dies ist ein Netzwerkprotokoll, welches eine verschlüsselte Netzwerkverbindung herstellt (Wiki\_SSH, 2016).

Der Funktionsaufruf um eine solche Verbindung aufzurufen ist ähnlich zu dem in Kapitel 3.3. Zunächst wird die Secure Shell mit dem Aufruf „ssh <Username>@<IP-Adresse>“ in dem Linux-Terminal aufgebaut. Zur Sicherheit erfolgt auch hier im nächsten Schritt eine Passwortabfrage.

Dabei werden bei beiden Board identische Usernames (root) und Passwörter (zebematado) verwendet. Die IP-Adressen jedoch müssen Unterschiede (192.168.37.250 und 192.168.37.251) aufweisen. Bei korrektem Passwort wird das aktuelle Datum sowie das des letzten Logins entsprechend Abbildung 3.5 angezeigt. Das Linux-Terminal, in welchem dieser Aufruf durchgeführt wurde stellt nun eine Kommandozeile des EVAchargeSE Boards dar.

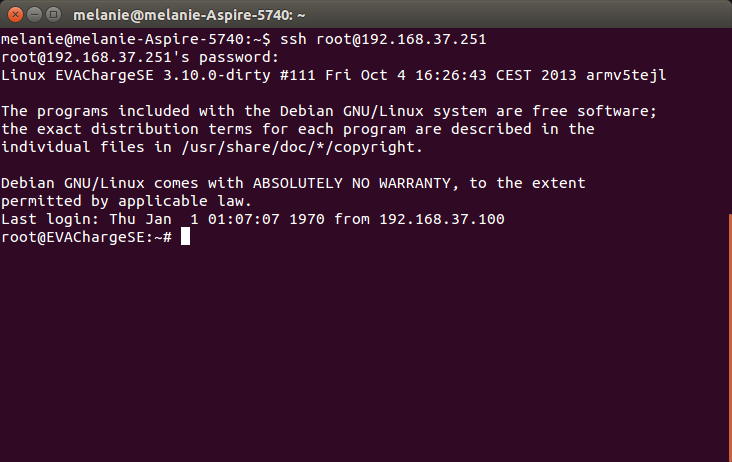


Abbildung .: Erfolgreiche SSH-Verbindung

Zum kompilieren des zuvor übertragenen C-Codes muss zunächst in das entsprechende Verzeichnis koordiniert werden. Zur Übersicht des aktuellen Verzeichnispfades und zur Navigation in andere Verzeichnisse werden die Befehle „pwd“ und „cd“ entsprechend Tabelle 3.4 verwendet. Damit das EVAchargeSE Board im weiteren Verlauf die in dem Verzeichnis vorhandenen Sourcefiles kompiliert, wird der Befehl „gcc –o <NAME> \*.c“ aufgerufen. Die hierbei eingesetzte Variable <NAME> kann dabei beliebig benannt werden und beinhaltet die Startdatei. Zum Starten des Kompilierten Codes wird nun die zuvor definierte Startdatei durch „ ./ <NAME>“ aufgerufen. Ein Linux-Terminal, welches in dem serial\_Programming-Verzeichnis dieses kompiliert und Startet ist in Abbildung 3.6 zu sehen.

Soll eine ausführende Datei unterbrochen werden kann dies mithilfe der Tastenkombination „Strg“ und „C“ durchgeführt werden. Sobald die Arbeiten im Terminal des Entwicklungsboards durchgeführt werden kann die SSH-Verbindung durch „exit “ beendet werden.

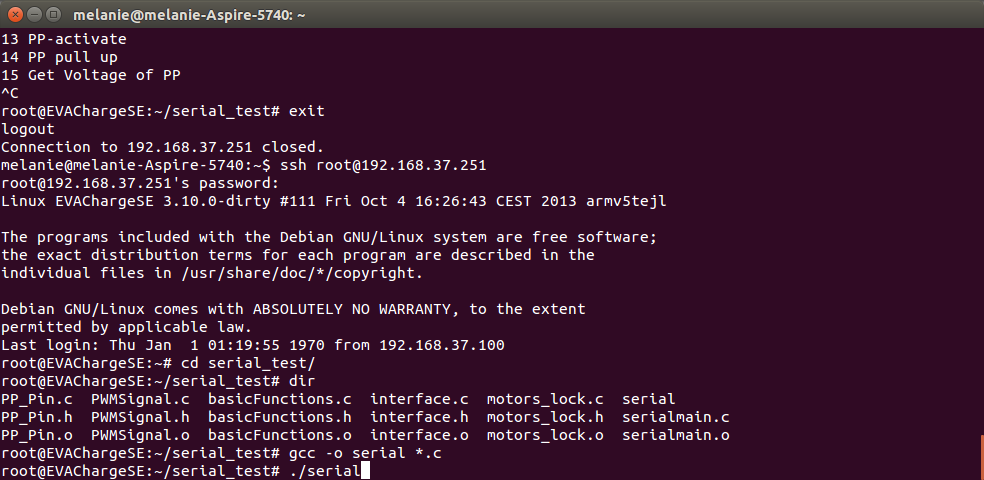


Abbildung .: Kompilieren und ausführen eines Programms

## Aufgetretene Probleme und Maßnahmen zur Problembehebung

Während der Inbetriebnahme der verwendeten Hardware sind Probleme aufgetreten, die nicht in dem Datenblatt der Boards zu beschrieben sind. Zur Nachverfolgung und Fehlervermeidung sind die wichtigsten im weiteren Verlauf beschrieben.

Im ersten Schritt zur Inbetriebnahme der Boards wird eine Verbindung zwischen PC und einem Board aufgebaut. Der Aufbau hierfür wird entsprechend Abbildung 3.2 mit nur einem Board realisiert. Bei dem Versuch, andere mit dem Router verbundene Teilnehmer mithilfe des „ping“-Befehls entsprechend Tabelle 3.4 anzusprechen, erhält der Linux-Rechner keine Antwort von dem Board. Der Grund dafür lässt sich in der IP des Routers und des EVAchargeSE-boards finden. Da die Grundeinstellung des Routers eine IP-Adresse in einem anderen Subnetz vorsieht. Durch Änderung der IP-Adresse des Routers können sich die Netzwerkteilnehmer erkennen und beim „ping“-Befehl erhält der Rechner eine Antwort des Boards.

Ein weiteres Problem lässt sich in einem nicht definierten Response Service der seriellen Kommunikation finden. Bei einem Fehler in der Übertragung des Frames von Master zu Slave gibt dieser als Response den nicht Dokumentierten Response-Service 0x99 zurück. Dabei kann in dem Datenblock 0x43 für einen Fehler der Prüfsumme oder 0x44 für einen unbekannten Fehler als Grund des Fehlers erkannt werden.

Zur Kommunikation der Boards miteinander wird im weiteren Verlauf ein gegenseitiges anpingen über die Powerline-Kommunikationsschnittstelle initiiert. Da sich die beiden Kommunikationsteilnehmer nicht finden konnten wurde der Rat von I2SE hinzugezogen. Durch den in Tabelle 3.3 beschriebenen Befehl „plcstat –i qca0 -t“ war ein Erkennen ebenfalls nicht möglich. Durch Auslesen des Parameter Information Blocks beider Boards konnte festgestellt werden dass beide Boards mit einer identischen MAC-Adresse vorbelegt wurden. Eine Änderung der Adresse ist ebenfalls in Tabelle 3.3 beschrieben und behebt diesen Fehler. Dadurch erfolgt bei einem erneuten Aufruf zur Anzeige aller verfügbaren Powerline Kommunikationspartner eine Antwort entsprechend Abbildung 3.7.

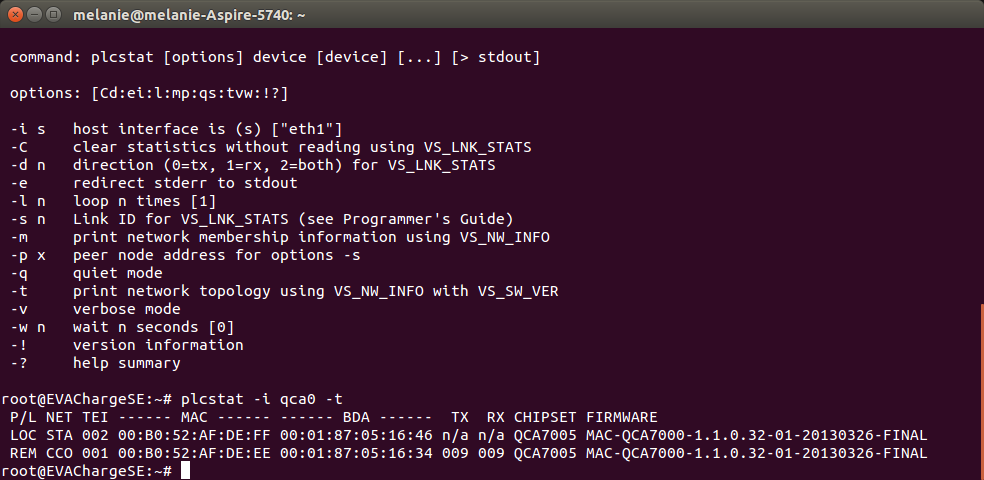


Abbildung .: Anzeige eines weiteren Powerline-Kommunikationssystems

## Verzeichnisse verwendeter Befehle

Wie bereits in Kapitel 3.2.2 erwähnt, gibt es verschiedene, von I2SE definierte Shell-Befehle. Diese sind besonders zur Konfiguration und Anzeige der Powerline-Verbindung notwendig. Durch die Kommunikation mit I2SE während der Bearbeitung des Projekts konnte die Übersicht in Tabelle 3.3 erstellt werden. Dabei werden immer die drei Hauptbefehle „plcstat“, „plctool“ sowie „modpib“ verwendet. Der Befehl „plcstat“ ist für Netzwerkeinstellungen zwischen den beiden Kommunikationspartnern der Powerline-Verbindung verantwortlich. Dazu gehören Einstellungen sowie die Anzeige von Parametern. „plctool“ ist für die internen Einstellungen der Powerline-Verbindung verantwortlich. Durch „modpib“ können definierte Parameter einer pib Datei gelesen und verändert werden.

|  |  |
| --- | --- |
| I2SE Befehle |  |
| Plcstat | PLC-Netzwerkeinstellungen |
| Plctool | Lokale Einstellungen der PLC-Kommunikation |
| plcstat –I qca0 –t | Erkennen der verfügbaren Powerline Verbindungen |
| plctool –i qca0 –p /tmp/pib local | Ändern der MAC Adresse |
| modpib –M<MAC> –v /tmp/pib |
| plcwait –R –I qca0 |
| /usr/local/bin/plcboot –N /root/Mac-QCA7000-v1.1.0-01-X-FINAL.nvm –P /tmp/pib –S |
| /root/NvmSoftloader-QCA7000-v1.1.0-01-FINAL.nvm –I qca0 –F |
| plctool –I qca0 –p /tmp/pib local | der QCA Einstellungen anzeigen |
| modpib –v /tmp/pib |

Tabelle .: I2SE-Befehle im Terminal

Um sich innerhalb der Linux-Terminalbefehle zurechtzufinden sind in Tabelle 3.4 die während der Bearbeitung meist verwendeten Shellbefehle zusammengestellt. Die detaillierte Beschreibung eines Befehls kann auch durch „man <Befehl>“ von dem Terminal ausgegeben werden. Weitere Unterstützung bei der Programmierung mit Nutshells ist im Literaturverzeichnis zu finden (Hekman, 1998).

|  |  |
| --- | --- |
| Allgemeine Nutshell Befehle |  |
| ssh [root@192.168.37.250](mailto:root@192.168.37.250) | Aufbau einer Secure Shell Verbindung (Kapitel 3.3.2) |
| sftp [root@192.168.37.250](mailto:root@192.168.37.250) | Verbindungsaufbau eines SFTP-Protokolls (sh. Kapitel 3.3) |
| Ifconfig | Konfigurations- und Statusanzeige aller verfügbaren Netzwerkschnittstellen |
| cd <Ordnername> | Change Directory: wechseln in ein Unterverzeichnis des aktuellen Ordners |
| cd .. | In Übergeordnetes Verzeichnis wechseln |
| cd /home/user | In den Dateipfand /home/user wechseln |
| vi <Dateiname> | Öffnen eines Files im vi-Editor. Um aus dem Editor wieder zurück ins Terminal zu kommen „ESC“-Taste drücken, „:q“ eingeben und mit Return bestätigen. |
| gcc –o <NAME> \*.c | Kompilieren von allen Source Files entsprechend Kapitel 3.3.2. |
| ./<NAME> | Starten von einem Programm entsprechend Kapitel 3.3.2. |
| ping <IP-adresse> | Sendet Datenpakete an eine IP-adresse um das Vorhandensein einer Verbindung zu prüfen |
| ping –I qca0 <IP-Adresse> | Ping-Befehl über Netzwerkschnittstelle qca0 |
| dir | Zeigt alle im aktuellen Verzeichnis enthaltenen Ordner und Dateien an. |
| pwd | Gibt aktuellen Ordnerpfad aus |
| rm <Datei> | Löschen einer Datei |

Tabelle .: Linux Terminalbefehle (Shellbefehle, 2016)

# Software

Wie bereits in Kapitel 1.2 erwähnt besteht ein Ziel des Projekts darin, Softwaretechnisch einen vorhandenen Kommunikationsstack zur ISO/IEC 15118 in Fahrzeug- und Ladesäulenseite in zwei Teile zu splitten und auf den EVAchargeSE-Boards zu Demonstrationszwecken lauffähig zu implementieren. Aus diesem Grund muss zusätzlich die Erstellung eines Verzeichnisses zur Generierung des PWM-Signals sowie der IP-basierte Kommunikationsaufbau berücksichtigt werden.

Abbildung **4.1** zeigt die dafür gewählte Vorgehensweiße während der Projektarbeit.

Für die Verwendung der Funktionen des serial\_Programming-Verzeichnisses muss dabei zunächst eine Kommunikation zwischen dem i.MX 28 und dem KL02 hergestellt werden. Die Inhalte der Nachrichten, welche in dieser Verbindung kommuniziert werden, beinhalten die in Kapitel 3.2.2 definierten.

Im weiteren Verlauf wird die Modulation der IP-basierten Nachrichten auf das PWM-Signal realisiert. Für die Umsetzung einer solchen Kommunikationsmethode werden sockets entsprechend Kapitel 4.1verwendet.

Im letzten Arbeitspacket wird wie bereits erwähnt der vorhandene Stack in Fahrzeug und Ladesäule aufgeteilt. Durch das Verbinden aller Softwarepakete soll der Kommunikationsstack nach ISO/IEC 15118 realisiert werden. Dazu gehört neben dem gesplitteten Stack das Generieren bzw. auslesen des PWM-Signals sowie die Kommunikation über TCP/IP.

**Abbildung 4.1: Ablauf der Softwareprogrammierung**

## Kommunikation innerhalb des Boards

Zur Kommunikation zwischen dem i.MX 28 und dem KL02 wird eine UART-Schnittstelle verwendet. Da der i.MX 28 im weiteren Verlauf den Programmablauf steuert, wird er als Master definiert. Aus diesem Grund ist der KL02 bereits im Auslieferungszustand als Slave Programmiert. Dies bedeutet, dass er lediglich auf Anfragen entsprechend Kapitel 3.2.2 antwortet.

Zum Aufbau der Verbindung werden die Einstellungen entsprechend des „Board Support Package“ Dokuments (I2SE, 2016) verwendet. Darin sind eine Baudrate von 57600Bd bei 8 Daten- und 1 Stoppbit definiert. Weiterhin muss das Modemdevice, über welches die Daten zu übertragen sind, bekannt sein. Beim Öffnen des Ports wird die Verbindung in einem Filehandle gesichert, sodass auch bei mehreren bestehenden Verbindungen eine Differenzierung stattfinden kann. Für die Initialisierung eines solchen Kommunikationsprotokolls existieren bereits Sources, welche nach Änderung der Konfigurationen verwendet werden (Sweet, 2016).

Nach Initialisierung der UART-Schnittstelle können Nachrichten übermittelt werden. Zum Senden einer Nachricht wird diese in ein Array geschrieben. Mit dem Befehl <write(Filehandle, Array, Anzahl zu übertragender Bytes)> wird das Array per UART gesendet. Der Rückgabewert der Funktion gibt an, wie viele Bytes gesendet wurden. Durch <read(Filehandle, Array, Anzahl zu empfangender Bytes)> werden die zu empfangenen Daten in das zuvor definierte Array geschrieben. In beiden Fällen muss bekannt sein, wie viele Bytes erwartet bzw. gesendet werden. Die in 3.2.2 definierten Frames werden somit immer in ein Array geschrieben, versendet und auch wieder empfangen.

## IP-basierter Kommunikationsaufbau zwischen EV und EVSE

Für den Aufbau einer TCP-Kommunikation wird das weit verbreitete LAN-Interface für Netzwerkanwendungen, ein Socket verwendet. Auch die in 3.3.1 und 3.3.2 beschriebenen Funktionen SFTP und SSH sind Applikationen, die ein Socket-Interface verwenden. Bei diesem Protokoll gilt das Client-Server Prinzip, wobei der Ladesäule die Serverseite zugeordnet wird (ITWissen, 2016).

Um Geräte, die an ein Netz angebunden sind, erreichen und unterscheiden zu können, wird ihnen eine IP-Adresse zugewiesen. Da manche Protokolle gleichzeitig von mehreren Nutzern gleichzeitig genutzt werden können, werden zur Unterscheidung der Prozesse zusätzlich Portnummern verwendet. Für den Aufbau einer Verbindung zwischen den zwei EVAchargeSE Boards muss die verwendete Portnummer beiden Seiten bekannt sein. Zusätzlich muss die Clientseite die IP-Adresse des anzusprechenden Servers kennen.

Für den Aufbau, die Kommunikation sowie die Beendigung der TCP-Verbindung werden die Socket-Funktionen entsprechend Abbildung 4.2 verwendet.

Die Funktion Socket() ist für die Einrichtung eines Kommunikationsendpunktes auf verantwortlich. Darin sind Parameter, welche die Adressfamilie, die Übertragungsart sowie das zur Übertragung zu verwendende Protokoll definieren, anzugeben.

Nachdem beide Seiten einen Socket errichtet haben wird dem Kommunikationsendpunkt des Servers eine Portnummer mithilfe des Befehls bind() zugewiesen. Diese Portnummer muss dem Client bekannt sein, um eine spätere Verbindung aufbauen zu können.

Auf Serverseite sind alle für die Kommunikation benötigten Einstellungen bekannt. Aus diesem Grund wartet dieser Kommunikationsteilnehmer mit listen() auf eine Verbindungsanforderung an dem zuvor definierten Port.

Sobald der Client einen Verbindungsaufbau mithilfe des Aufrufs connect() herzustellen versucht, muss die Serverseite diesen mit accept() akzeptieren, um eine Kommunikation zu ermöglichen.

Zum Versenden bzw. Empfangen werden nun die Funktionen send() und recv() verwendet. Wie bei der UART-verbindung muss auch hier bei beiden Funktionen die Größe des zu übermittelnden bzw. zu empfangenden Datenpaktes bekannt sein. Aus diesem Grund werden im gegebenen Fall structs verwendet, deren Inhalte und Inhaltsgrößen bereits im Vorfeld definiert wurden.

Zuletzt muss die Socket Verbindung nach beenden der Kommunikation beendet werden. Dies geschieht mit dem Aufruf von Close(). Dadurch wird der für die Kommunikation verwendete Port frei und kann erneut mit der listen()-Funktion auf eine Verbindungsanforderung warten.

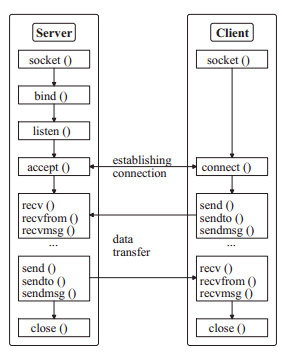


Abbildung .: Typische Socket-Kommunikation(Groß, 2015)

## Erstellen eines ISO/IEC 15118 Stacks

Um einen Stack zum Aufbau eines Demonstrators nicht komplett neu aufbauen zu müssen, wird auf ein Programm zur ISO/IEC 15118 aus einem Open-Source-Projekt aufgebaut(OpenV2G, 2016). Darin werden vor allem die Nachrichteninhalte und deren Datentypen beschrieben. Weiterhin werden beispielhafte Abläufe zum Laden mit Gleich- und Wechselstrom dargestellt.

Bei den bisherigen Versionen des Kommunikationsstack werden sowohl Fahrzeug- als auch Ladesäulenseite von einem Programm simuliert. Dies bringt eine hohe Anzahl an Files mit sich, die zudem bis zu 20.000 Zeilen Code enthalten. Zudem soll der finale Stand des Projektes zwei C-Programme beinhalten, welches die Einbindung in einen Demonstrator mit zwei Kommunikationsteilnehmern ermöglicht. Dadurch ist es unumgänglich, jeweils für Fahrzeug- und Ladesäulenseite ein Programm zu implementieren und somit den vorhandenen Stack aufzuteilen.

Zur Validierung der beiden Programme müssen diese miteinander kommunizieren können. Aus diesem Grund wird im weiteren Verlauf die in Kapitel 4.2 erstellte Bibliothek zur IP-basierten Kommunikation eingebunden. Für die Socketverbindung wird bei der Initialisierung des Clients die IP-Adresse 127.0.0.1 verwendet. Diese führt zu einem lokalen Host wodurch das Versenden von Nachrichten an den eigenen Rechner durch diese Adresse ermöglicht wird. Dadurch wird ein Programmstart Beispielsweiße von Eclipse auf einem Rechner ermöglicht, ohne Änderungen bezüglich der IP-Adresse vornehmen zu müssen. Eine Unterscheidung wird mithilfe der Makro „CODE\_EXAMPLE“ vorgenommen. Nimmt dieser Parameter den Wert der definierten Makro „CODE\_EXAMPLE\_SOFTWARE“ an, wird die IP-Adresse 127.0.0.1 verwendet. Für einen Kommunikationsaufbau mit dem Server muss „CODE\_EXAMPLE\_HARDWARE“ angegeben werden. Im vorliegenden Projekt ist dies die IP-Adresse der Ladesäule. Bei der Kommunikation der EVAchargeSE-Boards wird eine Powerline-Kommunikation aufgebaut. Für diesen Verbindungsaufbau wird deshalb nicht die bisherige IP-Adresse der Ethernet-Verbindung verwendet. Das Board, welches zur Verbindung mit einem Rechner mit der IP-Adresse 192.168.37.250 angesprochen wurde, verwendet die IP-Adresse 192.168.38.32 zur Powerline-Kommunikation. Das andere Board hingegen hat zur Powerline-Verbindung die IP-Adresse 192.168.38.33. Dies bedeutet, dass bei dem Flashen der Boards auf die IP-Adressen geachtet werden muss. Wird ein Server-Programm auf das Board mit der IP-Adresse 192.168.37.250 geflasht, hat dieses zur Powerline-Kommunikation die Adresse 192.168.38.32. Daraus folgt, dass das Board 192.168.37.251 die Clientseite mit der IP-Adresse 192.168.37.33 ist. Bei der Initialisierung der socket-Verbindung muss nun auf der Clientseite die IP-Adresse des Servers, also 192.168.37.32 angegeben werden. Für einen besseren Überblick diesbezüglich dient Abbildung 4.3.

Nachdem der Kommunikationsstack nach ISO/IEC 15118 in der Entwicklungsumgebung läuft, muss dieser im letzten Schritt auch Hardwareseitig implementiert werden. Zur Realisierung müssen die dafür notwendigen Funktionen aus der serial\_Programming Bibliothek eingebunden werden. Dabei ist darauf zu achten, zwischen Fahrzeug und Ladesäule zu unterscheiden. Auch diese Funktionen können durch die Makro „CODE\_EXAMPLE“ zu- bzw. abgeschaltet werden. Eine solche Abschaltung dient der Möglichkeit, die Source Files in einer Entwicklungsplattform zu starten und Funktionen, die einen Rückgabewert von anderen Kommunikationspartnern benötigen, überbrücken zu können.

Der resultierende Programmablaufplan des Elektrofahrzeugs ist im Anhang durch Abbildung 7.9 und Abbildung 7.10 dargestellt.

Powerline-IP

192.168.38.33

**Router**

**Client**

Powerline-IP

192.168.38.32

**Server**

Powerline-IP

192.168.37.250

Powerline-IP

192.168.37.251

Powerline-Verbindung für Kommunikation nach ISO/IEC 15118

SSH- SFTP-Verbindung zum Flashen, Kompilieren

und Starten von Programmen

Abbildung .: IP-Adressen der EVAchargeSE-Boards

### Erweiterung um einen Zustandsautomaten

Wie bereits in Kapitel 2.5 beschrieben, wechselt ein Zustandsautomat zwischen seinen Zuständen aufgrund eintretender Ereignisse. Des Weitern werden bei der Implementierung vorangegangene sowie zukünftige Zustände mit einbezogen. Zudem kann ein solcher Automat, wie Abbildung 2.6 zeigt, übersichtlich dargestellt werden. Dies bringt weiterhin den Vorteil mit sich, fehlende Elemente des Zustandsautomaten zu erkennen.

Besonders in dem vorliegenden Projekt ist die Implementierung eines solchen Zustandsautomaten auf Seite der Ladesäule vorteilhaft. Die Kommunikationsanfragen werden nacheinander von dem Elektrofahrzeug gestartet, wie durch Abbildung 7.9 und Abbildung 7.10 zu sehen ist. Da die Ladesäule in einem großen Teil der zur Verfügung stehenden Zustände mehrere Möglichkeiten eines darauffolgenden Zustandes hat, bietet ein Zustandsautomat aufgrund der Übersichtlichkeit eine gute Übersicht der einzelnen Verzweigungen. Weiterhin sind die einzelnen Zustände, die ein solcher Automat im vorliegenden Projekt annehmen kann, bereits in den Nachrichteninhalten in Abbildung 7.1 bis Abbildung 7.4 zu erkennen.

Ein Zustandsübergang wird dabei ausgelöst durch eine von dem Elektrofahrzeug empfangene Nachricht. Sollte diese Nachricht aus Informationen bestehen, die in keinem der Zustandsübergänge zum nächsten Zustand nach ISO/IEC 15118 definiert ist, wird ein Error-Zustand angenommen und die Kommunikation wird abgebrochen. Zuletzt muss berücksichtigt werden, dass ab dem Zeitpunkt des ersten Pegelwechsels in Ladestatus C eine Unterscheidung zwischen Gleich- und Wechselstromladung notwendig ist.

Bei der Implementierung wird zunächst eine Funktion „handshake“ aufgerufen. Darin enthalten ist bereits die Nachricht „supported App Protocol “. Nachdem die Funktion erfolgreich ausgeführt wurde, wird die Variable „next\_state“ mit dem ersten Status „Session Setup“ vorbelegt.

Eine Übersicht des Zustandsautomaten ist in Abbildung 4.4, Abbildung 4.5 und Abbildung 4.6 zu sehen. Zum Starten der Zustände in Abbildung 4.4 und Abbildung 4.5 muss der Pegel des CP-Signals dem Ladestatus B entsprechen. Für einen Start der Zustände in Abbildung 4.5 hingegen Ladestatus C.

Der Programmablaufplan der Ladesäule und die Funktionen, in welchen der Zustandsautomat realisiert ist, findet sich im Anhang beginnend mit Abbildung 7.5.



Abbildung .: Zustandsautomat EVSE Teil 1 von 3

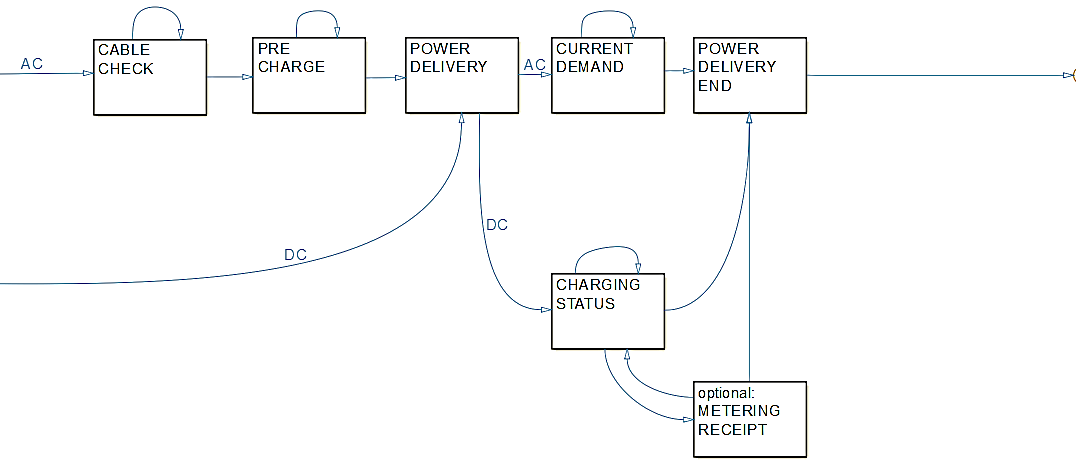


Abbildung .: Zustandsautomat EVSE Teil 2 von 3



Abbildung .: Zustandsautomat EVSE Teil 3 von 3

# Ergebnisse

Die Platinen von I2SE wurden entsprechend Abbildung 5.1 aufgebaut und die Pins, welche denen des Ladesteckers entsprechen, mit verdrillten Leitungen verbunden.

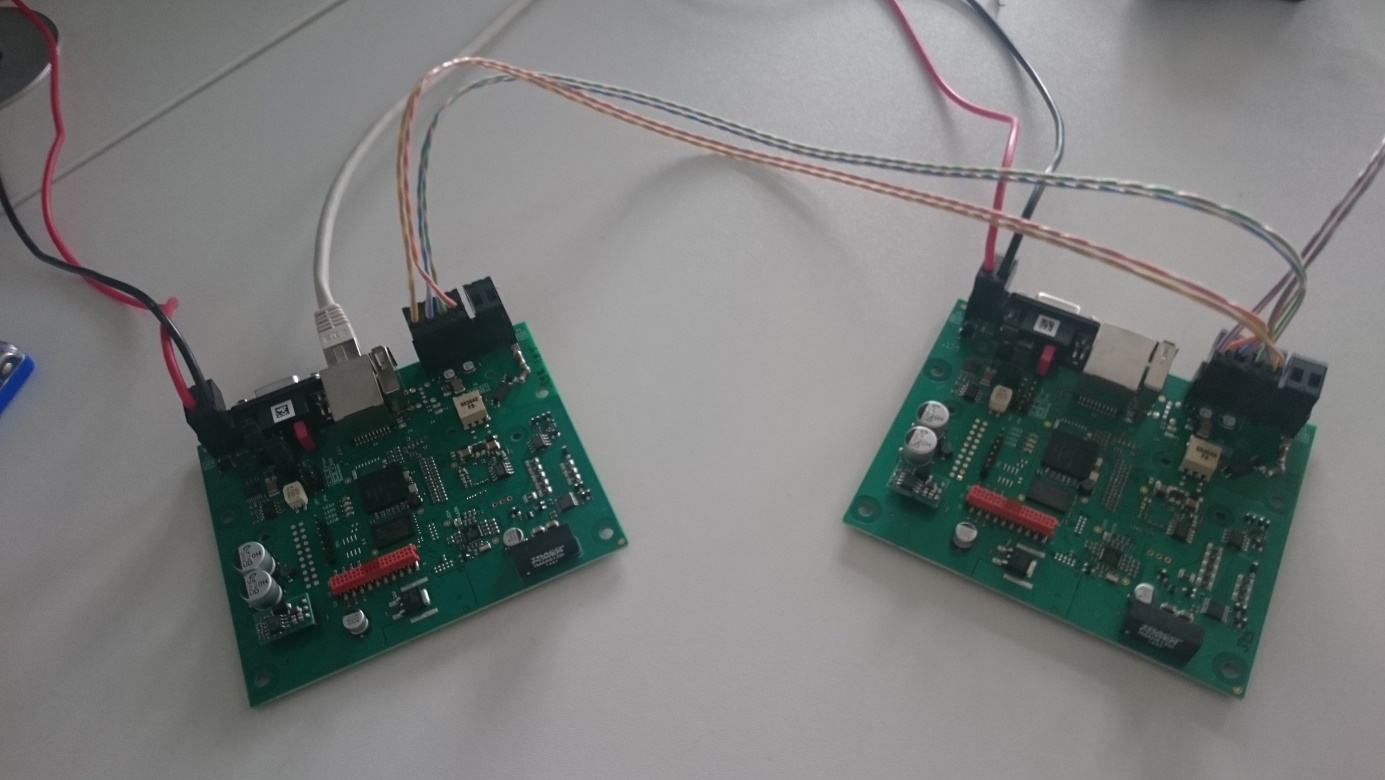
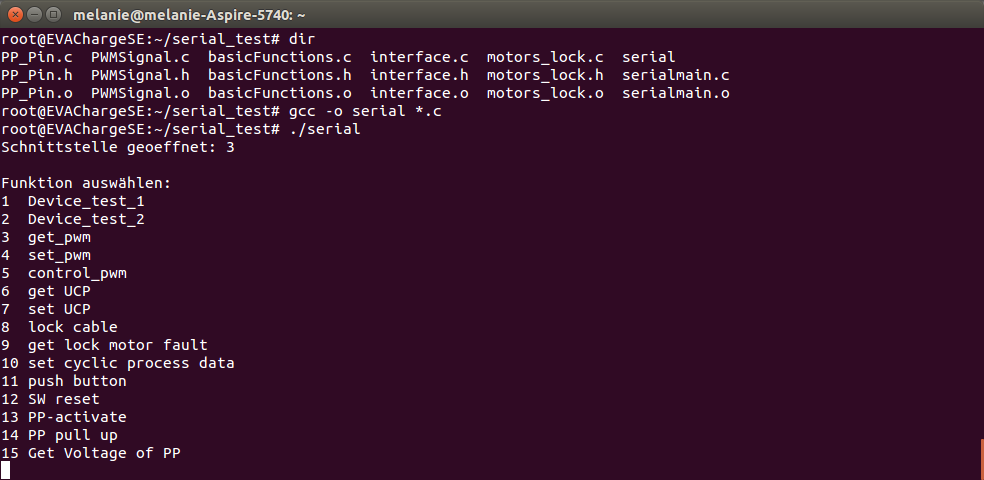


Abbildung .: Hardwareaufbau

Weiterhin konnte ein Aufstellen der serial\_Programming-Bibliothek erreicht werden. Sobald eine Platine mit einem Linux-Rechner verbunden ist, kann diese mithilfe eines Linux-Terminals gestartet werden. Nach dem Starten wird ein Menü entsprechend Abbildung 5.2 angezeigt, durch welches die verfügbaren Funktionen gestartet werden können.



**Abbildung 5.2: Anzeige des Linux-Terminals nach Starten von serial\_Programming**

Die für die ISO/IEC 15118 benötigten Funktionen aus der entstandenen Bibliothek sind in dem gleich benannten Unterverzeichnis des endgültigen Programms zu finden. Daraus entstehen Signale auf der Leitung des Control Pins entsprechend den Folgenden Abbildungen.

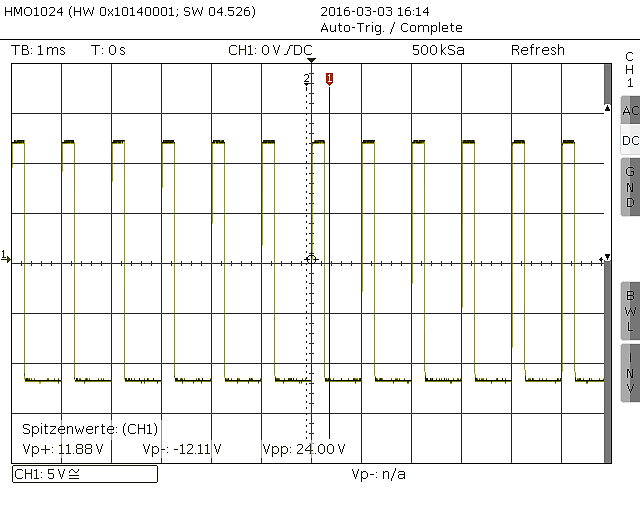


Abbildung .: Pegel des Control Pin bei Ladestatus A

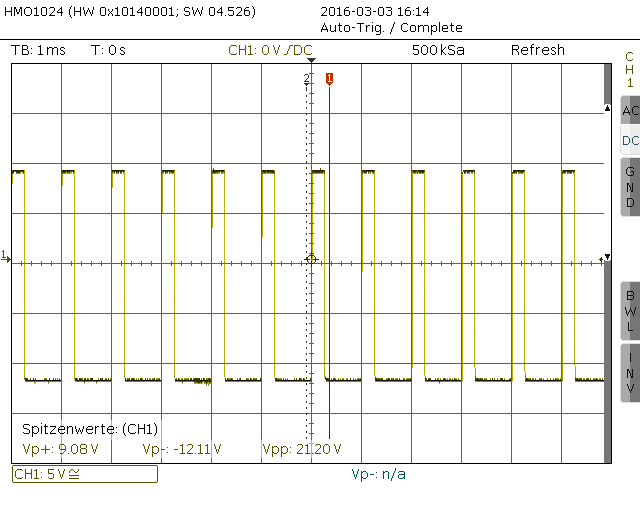


Abbildung .: Pegel des Control Pin bei Ladestatus B

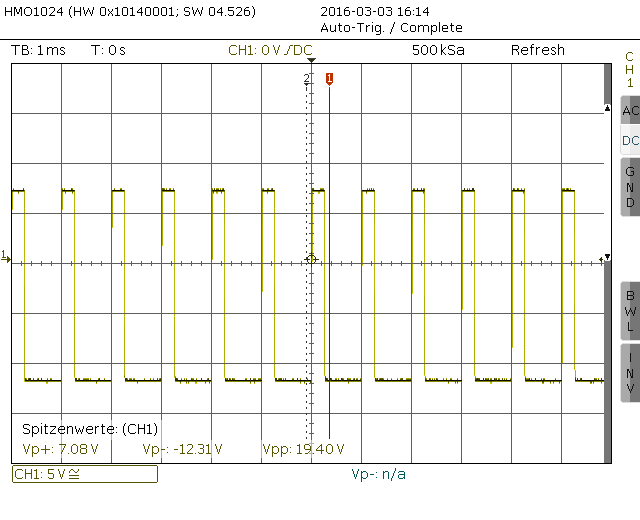


Abbildung .: Pegel des Control Pin bei Ladestatus C

Dabei zeigt Abbildung 5.3 den Ladestatus A, welcher einen Pegel von ±12±1Volt hat. Laut IEC 61851 entspricht dies einem nicht verbundenen Elektrofahrzeug. Im gegebenen Fall jedoch können die Widerstände entsprechend gesetzt werden, sodass der gezeigte Pegel erreicht werden kann, während eine Verbindung zwischen CP, PP und PE der Boards besteht.

In Abbildung 5.4 wird der Pegel bei verbundenem, jedoch nicht Ladebereitem Elektrofahrzeug dargestellt. Wie bereits in Kapitel 2.2 erwähnt, muss der negative Pegel -12±1V beibehalten, während im positiven Bereich bei dem vorliegenden Ladestatus B ein Bereich von 8±1V vorgegeben ist.

Abbildung 5.5 zeigt einen Grenzwert des Toleranzfeldes während des Ladestatus C, wodurch eine ladebereite Verbindung anzeigt wird. Das Toleranzfeld der Pegel ist mit min -12±1V und max. 6±1V angegeben, der positive Spitzenwert beträgt jedoch 7,08V. Da der Messwert für keine weitere Berechnungen benötig wird lediglich eine Nachkommastelle zur Bestimmung des Pegels betrachtet.

# Zusammenfassung & Ausblick

Das resultierende Ergebnis der vorliegenden Projektarbeit ist ein Demonstrator der ISO/IEC 15118 in Hard- und Software. Dafür wurde eine entsprechende Hardware definiert und in Betrieb genommen und ein Kommunikationsstack auf Fahrzeug- und Ladesäulenseite aufgeteilt. Weiterhin ist ein wichtiger Bestandteil der Aufbau des PWM-Signals sowie die Realisierung der IP-basierten Kommunikation zwischen EV und EVSE.

Im weiteren Verlauf wird im Umfang einer Studienarbeit der Aufbau einer „Wallbox“ realisiert. Ein Ladegerät für Elektrofahrzeuge dieser Art ist im privaten Gebrauch verbreitet, da sie keine Bezahlmöglichkeit bietet (Wiki\_Wallbox, 2016). Weiterhin soll eine App mit dem Protokoll verbunden werden, sodass ein Starten des Kommunikationsprotokolls und die Definition der darin enthaltenen Parameter per Smartphone übertragen werden kann.

# Anhang

## AC-Ladeablauf



Abbildung .: Nachrichtenübersicht für AC-Ladezyklus (1 von 2)

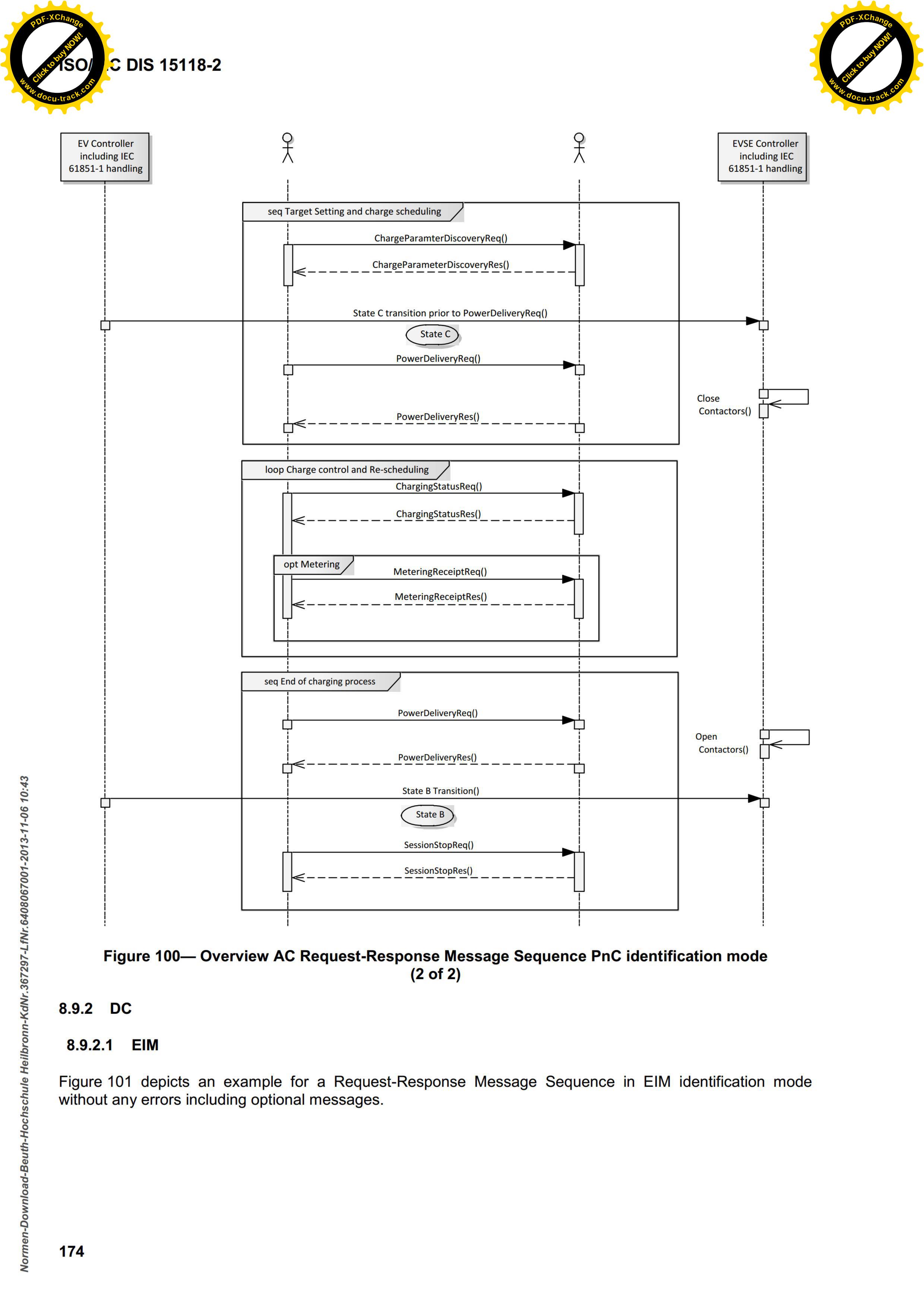


Abbildung .:Nachrichtenübersicht für AC-Ladezyklus (2 von 2)

## DC-Ladeablauf

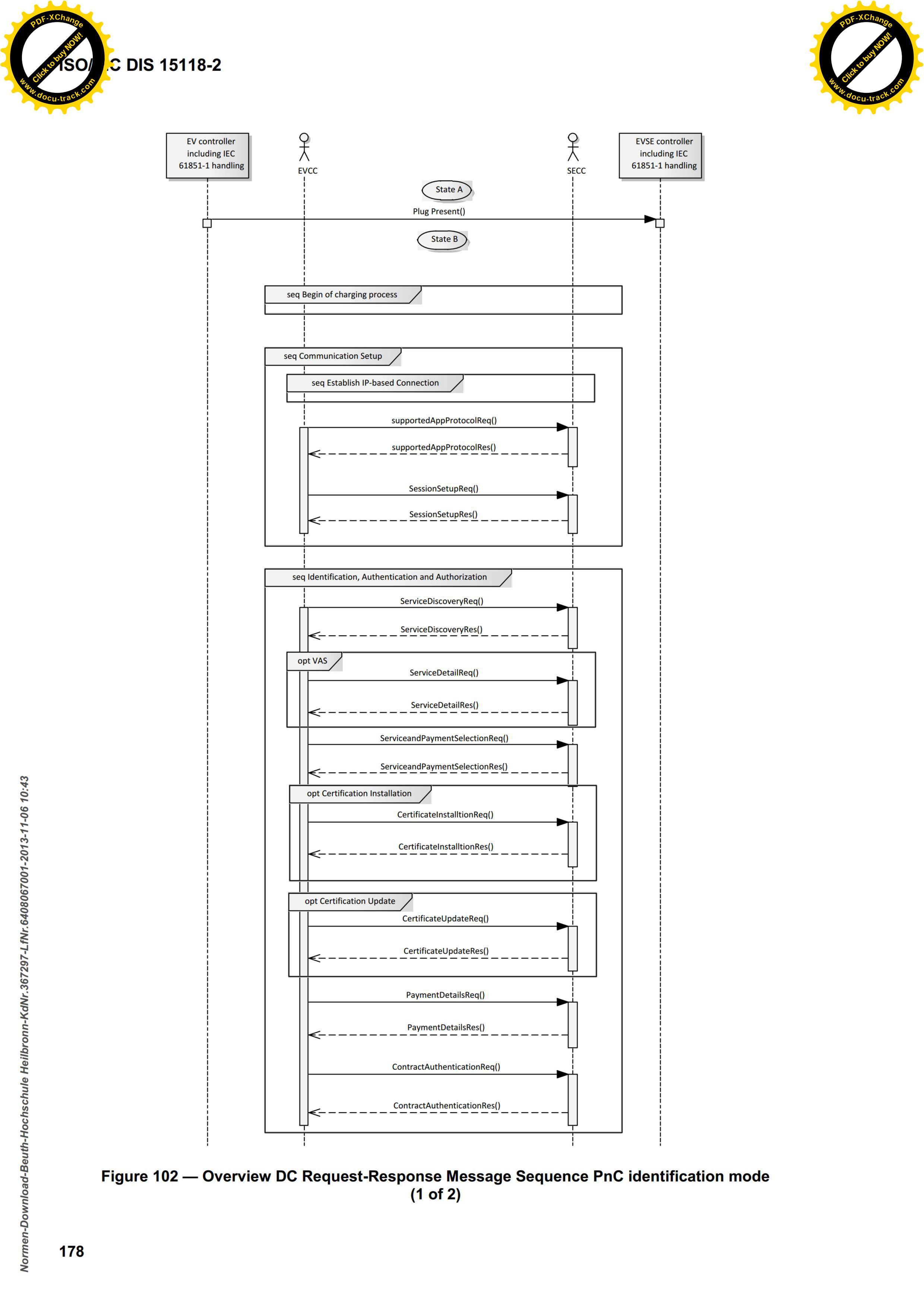


Abbildung .:Nachrichtenübersicht für DC-Ladezyklus (1 von 2)

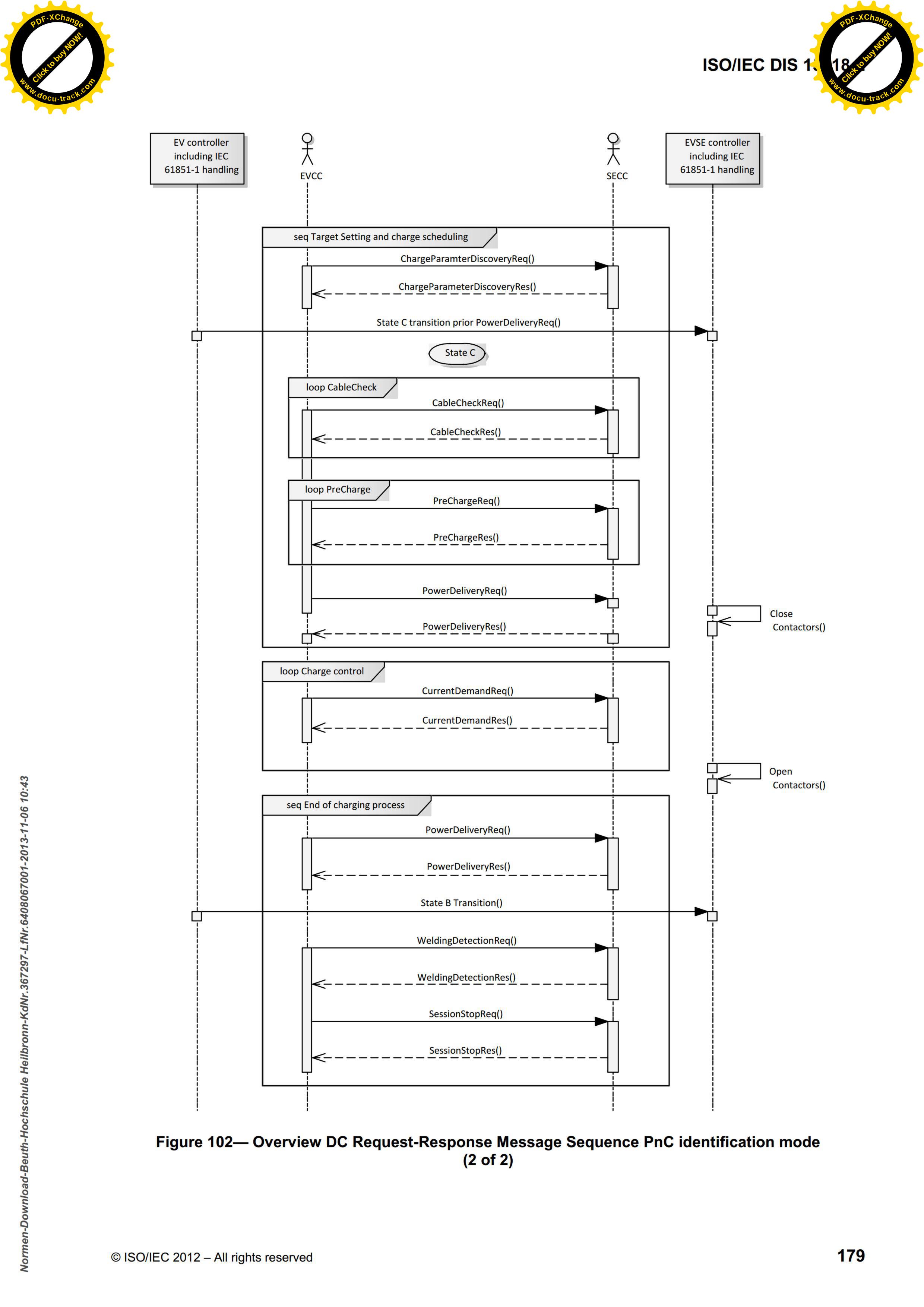


Abbildung .: Nachrichtenübersicht für DC-Ladezyklus (2 von 2)

## Programmablaufpläne der Ladesäule



Abbildung .: Programmablaufplan der Ladesäulenseite



Abbildung .: Programmablaufplan des Funktionsaufrufs State B1 Kommunikation



Abbildung .: Programmablaufplan des Funktionsaufrufs State C Kommunikation



Abbildung .: Programmablaufplan des Funktionsaufrufs State B2 Kommunikation

## Programmablaufpläne des Elektrofahrzeugs



Abbildung .: Programmablaufplan des Elektrofahrzeugs (Teil 1)



Abbildung .: Programmablaufplan des Elektrofahrzeugs (Teil 2)

# Literaturverzeichnis

|  |  |
| --- | --- |
| (Barth, 2015) | **Alexander Barth, Michael Röhrich**  Studienarbeit ISO 15118  (E-Mail vom 13.05.2015) |
| (Bustamante, 2014) | **Michele Leroux Bustamante**  Finite State Machines, Wizards, and the Web  <https://msdn.microsoft.com/en-us/library/aa478972.aspx>  (Veröffentlicht Februar 2004) |
| (Debian, 2016) | **Was ist Debian GNU?**  Debian Anwenderhandbuch  <http://debiananwenderhandbuch.de/wasistdebiangnu.html>  (Zugriff am 22.04.2016) |
| (El-Ko, 2016) | **Elektronik Kompendium**  Powerline-Kommunikation  (Zugriff am 13.04.2016) |
| (FileZilla, 2016) | **FileZilla**  <https://filezilla-project.org/>  (Zugriff am 25.04.2016) |
| (Groß, 2015) | **Prof. Dr. Siegmar Groß**  Skript Parallelverarbeitung  Kap. 4 Die Socket-Schnittstelle  Hochschule Fulda, Fachbereich Angewandte Informatik  (Zugriff am 21.06.2015) |
| (Hekman, 1998) | **Jessica Perry Hekman**  Linux in a Nutshell  O´Reilly Verlag  ISBN 3-930673-57-6  1. Nachdruck Januar 1998 |
| (I2SE, 2016) | **I2SE GmbH**  EVAchargeSE SE board  <https://www.i2se.com/product/evacharge-se/>  (Zugriff am 17.04.2016) |
| (ITWissen, 2016) | **ITWissen**  Socket  <http://www.itwissen.info/definition/lexikon/Socket-socket.html>  (Zugriff am 01.03.2016) |
| (Lewandowski, 2014) | **Christian Lewandowski.**  .Entwurf und Leistungsbewertung einer Kommunikationsarchitektur zur Erbringung von Regelleistung durch Elektrofahrzeuge  Technische Universität Dortmund  (Veröffentlicht am 17.10.2014) |
| (Mültin, 2014) | **Marc Mültin**  Dissertation: Das Elektrofahrzeug als flexibler Verbraucher und Energiespeicher im Smart Home  KIT-Fakultät für Wirtschaftswissenschaften des Karlsruher Instituts für Technologie(KIT)  (Zugriff am 08.04.2015) |
| (Opatz, 2006) | **Felix Opatz**  Netzwerkprogrammierung mit BSD-Sockets  Revision 1.15  http://www.zotteljedi.de/socket-buch/socket-buch.pdf  (Veröffentlicht am 08.07.2006) |
| (OpenV2G, 2016) | **www.Sourceforge.net**  Open-Source-Projekt zur Implementierung eines vollständigen ISO 15118 Kommunikationsprotokolls  http://openv2g.sourceforge.net/index.php/home.html  (Zugriff am 07.04.2016) |
| (Shellbefehle, 2016) | **Shellbefehle**  Übersicht der gängigsten Shellbefehle  http://www.shellbefehle.de/befehle/  (Zugriff am 27.04.2016) |
| (Shrestha, 2015) | **Nerad Shrestha**  10 sFTP Command Examples to Transfer Files on Remote Servers in Linux  http://www.tecmint.com/sftp-command-examples/  (Zuletzt aktualisiert 03.01.2015) |
| (Sweet, 2016) | **Michael R. Sweet**  Serial Programming Guide  5th Edition, 3re Revision  <https://support.dce.felk.cvut.cz/pos/cv5/doc/serial.html#config>  (Zugriff am 01.05.2016) |
| (Ubuntu\_SSH, 2016) | **Wiki.Ubuntuusers**  SSH  <https://wiki.ubuntuusers.de/SSH/>  (Zugriff am 22.04.2016)) |
| (Wiki\_SSH, 2016) | **Wikipedia**  Secure Shell  https://de.wikipedia.org/wiki/Secure\_Shell  (Zugriff am 25.04.2016) |
| (Wiki\_Stecker, 2016) | **Wikipedia**  Steckertypen und Lademodi für Elektrofahrzeuge  <https://de.wikipedia.org/wiki/IEC_62196>  https://de.wikipedia.org/wiki/IEC\_62196\_Typ\_2  (Zugriff am 08.03.2016) |
| (Wiki\_Wallbox, 2016) | Wikipedia  Wandladestation  <https://de.wikipedia.org/wiki/Wandladestation>  (Zugriff am 06.05.2016) |