



Charging Station for ISO / IEC 15118 Protocol

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

*IEC 61851*

Bachelor's Project

**Presented by**

Jiztom Kavalakkatt Francis

Nivas Gokul Manimurugesan

Raphael Scholz

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

Hochschule Heilbronn / Heilbronn University

Faculty of Mechanics und Electronics

Automotive Systems Engineering

# **Table of Contents**

[Table of Contents II](#_Toc477484994)

[Abstract V](#_Toc477484995)

[List of Tables VI](#_Toc477484996)

[List of Figures VI](#_Toc477484997)

[List of Symbols and Abbreviations VII](#_Toc477484998)

[1. Introduction - 1 -](#_Toc477484999)

[1.1. Short Version - 1 -](#_Toc477485000)

[**1.2.** **Task** - 2 -](#_Toc477485001)

[1.3. Chapter Overview - 3 -](#_Toc477485002)

[2. Literature Survey - 4 -](#_Toc477485003)

[2.1. Scholastic Survey - 4 -](#_Toc477485004)

[a. Automotive Ethernet: in-vehicle networking and smart mobility - 4 -](#_Toc477485005)

[**b.** **Towards standardized Vehicle Grid Integration:** - 6 -](#_Toc477485006)

[**c.** **ISO 15118 – charging communication between plug-in electric vehicles and charging infrastructure** - 7 -](#_Toc477485007)

[**d.** **Assuring Interoperability between Conductive EV and EVSE Charging Systems Authors:** - 8 -](#_Toc477485008)

[**e.** **Vehicle-to-Grid AC Charging Station: AN approach for Smart Charging Development** - 9 -](#_Toc477485009)

[2.2. Reference - 10 -](#_Toc477485010)

[3. Problem statement /Objective - 11 -](#_Toc477485011)

[3.1. Introduction: - 11 -](#_Toc477485012)

[3.2. Objective: - 12 -](#_Toc477485013)

[4. Methodology - 13 -](#_Toc477485014)

[4.1. IEC 62196: vehicle plug - 13 -](#_Toc477485015)

[4.2. IEC 61851 - 15 -](#_Toc477485016)

[4.3. ISO 15118 - 17 -](#_Toc477485017)

[4.3.1. Open V2G Project - 18 -](#_Toc477485018)

[4.4. TCP/IP Communication: - 19 -](#_Toc477485019)

[4.5. UML - 21 -](#_Toc477485020)

[4.5.1. Introduction - 21 -](#_Toc477485021)

[4.5.1.1. UML 2.0 - 21 -](#_Toc477485022)

[4.5.1.2. Types of UML Diagrams - 21 -](#_Toc477485023)

[4.5.2. State Machine Diagram - 22 -](#_Toc477485024)

[4.5.2.1. Difference between a State Diagram and a Flowchart - 22 -](#_Toc477485025)

[4.5.2.2. Steps to Drawing a State Diagram - 22 -](#_Toc477485026)

[4.5.2.3. The Charging Station – State Machine Diagram - 23 -](#_Toc477485027)

[4.5.3. Sequence Diagram - 24 -](#_Toc477485028)

[4.5.3.2. The Charging Station – The State Machine Diagram - 25 -](#_Toc477485029)

[4.6. Automatic State Diagram: - 26 -](#_Toc477485030)

[4.6.1. Introduction - 26 -](#_Toc477485031)

[4.6.2. State diagram of the charging station - 28 -](#_Toc477485032)

[5. Software Description - 30 -](#_Toc477485033)

[5.1. FileZilla - 30 -](#_Toc477485034)

[5.2. Geany - 32 -](#_Toc477485035)

[5.3. Putty Terminal - 33 -](#_Toc477485036)

[5.4. EB GUIDE 6.3 - 34 -](#_Toc477485037)

[5.5. www.draw.io - 35 -](#_Toc477485038)

[5.6. Visual Paradigm 14.0 - 36 -](#_Toc477485039)

[5.7. Compiling the program - 37 -](#_Toc477485040)

[6. Hardware Description - 39 -](#_Toc477485041)

[6.1. Raspberry Pi 3 - 39 -](#_Toc477485042)

[6.2. EVAChargSE Board - 41 -](#_Toc477485043)

[6.2.2. EVACharge SE board - 42 -](#_Toc477485044)

[6.2.3. Applications - 42 -](#_Toc477485045)

[6.3. Communication between EVSE and EV - 43 -](#_Toc477485046)

[6.4. Communication between EVSE and Raspberry Pi - 45 -](#_Toc477485047)

[6.5. Integrating the system - 46 -](#_Toc477485048)

[7. Results - 47 -](#_Toc477485049)

[8. Conclusion - 48 -](#_Toc477485050)

[9. Summary & outlook - 49 -](#_Toc477485051)

[Appendix I](#_Toc477485052)

[Appendix - A I](#_Toc477485053)

[Appendix - B XVII](#_Toc477485054)

[**Appendix - C** XXXVI](#_Toc477485055)

[**Raspberry pi** XXXVI](#_Toc477485056)

[**EVSE** XXXVII](#_Toc477485057)

[**EV** XXXVIII](#_Toc477485058)

[**Appendix - D** XXXIX](#_Toc477485059)

[Bibliography XL](#_Toc477485060)

# **Abstract**

# **List of Tables**

# **List of Figures**

# **List of Symbols and Abbreviations**

1. Introduction

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

* 1. Short Version

The present project work serves to set up a working model for the current standardization of the vehicle to grid communication according to ISO / IEC 15118 along with the HMI display running on raspberry pi platform.

The built-up working model consists of two interconnected EVACharge SE boards and a Raspberry Pi with a 7” inch screen for the display. .

The software used is derived from an existing stack and adapted for a defined application of ISO / IEC 115118. A dummy state model is used to test the veracity of the code.

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.

The HMI is configured to choose the language and allows the user to have smooth transition for payment and charging the vehicle. Additional option to fully charge and also timed charging with full receipt details of the charge is displayed at the end of the process.

* 1. **Task**

The main task is to implement a basic working model of the car charging station with the HMI interfaced into the system. It involves the study of the basic working of the ISO 15118 protocol and defining the possible working structure of the Charging station.

The task involves the design of the communication mechanisms and processes between the main processor of an electric vehicle supply equipment (EVSE) – or called charging station - and a computer that runs the HMI on that EVSE (HMI), and the electric vehicle to be charged (EV)

The targets to be achieved:

- Analyze the requirements of ISO 15118 and IEC 61851 based on the work of the references.

- Design, discuss and finalize the state machine and the communication process between EVSE and HMI together with a German student (Raphael Scholz)

- Learn about UML as a description language for state machines and communication sequences

- Use TCP/IP communication between the Raspberry pi and EVSE to send information between HMI and EVSE.

- Write Software program based on C language to interface the HMI and the EVSE as a backend process.

- Implement the entire task to a fully functional final product.

* 1. Chapter Overview

1. Literature Survey
   1. Scholastic Survey

The following were considered for the initial study of the Car Charging Process

* + 1. Automotive Ethernet: in-vehicle networking and smart mobility

**Authors:**

* Peter Hank NXP Semiconductors, Hamburg, Germany
* Steffen Müller NXP Semiconductors, Hamburg, Germany
* Ovidiu Vermesan SINTEF, Oslo, Norway
* Jeroen Van Den Keybus Triphase NV, Leuven, Belgium

**Proceedings:**

DATE '13 Proceedings of the Conference on Design, Automation and Test in Europe

Pages 1735-1739

Grenoble, France — March 18 - 22, 2011

**Inference:**

This paper discusses novel communication network topologies and components and describes an evolutionary path of bringing Ethernet into automotive applications with a focus on electric mobility. For next generation in-vehicle networking, the automotive industry identified Ethernet as a promising candidate besides CAN and FlexRay. Ethernet is an IEEE standard and is broadly used in consumer and industry domains. It will bring a number of changes for the design and management of in-vehicle networks and provides significant re-use of components, software, and tools. Ethernet is intended to connect inside the vehicle high-speed communication requiring sub-systems like Advanced Driver Assistant Systems (ADAS), navigation and positioning, multimedia, and connectivity systems. For hybrid (HEVs) or electric vehicles (EVs), Ethernet will be a powerful part of the communication architecture layer that enables the link between the vehicle electronics and the Internet where the vehicle is a part of a typical Internet of Things (IoT) application. Using Ethernet for vehicle connectivity will effectively manage the huge amount of data to be transferred between the outside world and the vehicle through vehicle-to-x (V2V and V2I or V2I+I) communication systems and cloud-based services for advanced energy management solutions. Ethernet is an enabling technology for introducing advanced features into the automotive domain and needs further optimizations in terms of scalability, cost, power, and electrical robustness in order to be adopted and widely used by the industry.

* + 1. **Towards standardized Vehicle Grid Integration:**

**Authors:**

* BO chen Argonne National Laboratory, USA
* Keith S. Hardy Argonne National Laboratory, USA
* Jason D. Harper Argonne National Laboratory, USA
* Daniel S. Dobrzynski Argonne National Laboratory, USA

**Published in:** Transportation Electrification Conference and Expo (ITEC), 2015 IEEE

**Inference:**

This paper studies what are needed to enable the standardization of Vehicle Grid Integration (VGI). The requirements of interoperable VGI are examined at multiple interoperability layers defined by reference architecture models, including European Commission's Mandate 490 (EU-M490), National Institute of Standards and Technology (NIST) Smart Grid Architectural Methodology (SGAM), and the Institute of Electrical and Electronic Engineers (IEEE) 2030 Smart Grid Interoperability Reference Model (SGIRM). The current status of standards and technology development is reviewed and VGI demonstrations are discussed. The paper identifies barriers for the implementation of an interoperable VGI and provides recommendations to address these challenges.

* + 1. **ISO 15118 – charging communication between plug-in electric vehicles and charging infrastructure**

**Authors:**

* Dr. Andreas Heinrich Daimler AG, Holzgerlinen, Germany
* Michael Schwaiger BMW Group, Munich, Germany

**Book Title:** Grid Integration of Electric Mobility

**Book Subtitle:** 1st international ATZ Conference 2016

**Pages:** pp 213-227

**Inference:**

* + 1. **Assuring Interoperability between Conductive EV and EVSE Charging SystemsAuthors:**
* M. Sc. Michael Tybel Scienlab electronic systems, Bochum
* Dr.-Ing Andrey Popov Scienlab electronic systems, Bochum
* Dr.-Ing Michael Schugt Scienlab electronic systems, Bochum

**Link to document:**

<http://www.p0p0v.com/science/downloads/TybelPopovSchugt15.pdf>

**Inference:**

The development and deployment rate of electric vehicles (EV) and plug-in electric vehicles (PEV) substantially depends on the corresponding EV supply equipment (EVSE). The facts that vehicles are intrinsically mobile and hence require interoperability between manufacturers, countries and charging points, implies that the components of the charging systems should extensively be tested, in order to allow access of the companies to the global market.

The two most common automotive charging communication standards are the pulse width modulation (PWM) based IEC 61851-1 [1] and the CAN based CHAdeMO [2]. The third and latest approach is a power line communication (PLC) using V2Gprotocol specified by the ISO 15118 [3]. The norm has been published in 2014 and will become the standard in Europe and North America for DC- and AC-Charging within the next years since all major OEMs have decided to apply it. In order to implement the new standard in a way, that supports all specified use cases (e.g. private or public charging, plug & charge or external identification/payment) and simultaneously assure operation between all EV/EVSE communication controllers of different origins, dedicated verification techniques, and routines are required.

This paper introduces the ISO 15118 norm and suggests independent and reproducible test methods that allow developers and quality managers to achieve a high degree of interoperability.

* + 1. **Vehicle-to-Grid AC Charging Station: AN approach for Smart Charging Development**

**Authors:**

* D. Wellisch Deggendorf Institute of Technology, Freyung
* J. Lenz Deggendorf Institute of Technology, Freyung
* A. Faschingbauer Deggendorf Institute of Technology, Freyung
* R. Pöschl Deggendorf Institute of Technology, Freyung
* S. Kunze Deggendorf Institute of Technology, Freyung

**Link to the document:** [researchgate.com](https://www.researchgate.net/profile/Rainer_Poeschl/publication/282846691_Vehicle-to-grid_AC_Charging_Station_An_Approach_for_Smart_Charging_Development/links/561e15f808aec7945a253e1c.pdf)

**Inference:**

The use of electric powered vehicles is increasing steadily. This also leads to new challenges for the power grid. An electric powered vehicle provides heavy stress for the grid, especially when many vehicles are loading their accumulators simultaneously. To counteract these negative effects, smart charging is developed. With intelligent vehicle-to-grid communication, the stress for the grid, during the charging process, can be reduced. This is especially important when renewable energy sources are utilized. Using new software protocols and suiting hardware applications, smart charging can harmonize the needs of renewable energy sources and electromobility. In this paper, a smart charging capable AC charging station for hardware and software evaluation is proposed. This system is based on OCPP 2.0 and the ISO 15118 standard.

* 1. Reference

1. Problem statement /Objective
   1. Introduction:

The European Union is looking forward to reduce the no. of oil based vehicles by 40% and replace them with electric vehicles by 2025. This allows countries to reduce their carbon footprint and also improve the pollution crisis which some of the cities are facing worldwide.

This new idea on implementing results in another growing concern, i.e. the power required to charge the newly added electric cars. If all the cars are charged simultaneously it will result in a blackout due to overdraft of available power. This is not an acceptable scenario.

This led to the realization of **ISO/IEC 15118** which focus on how to distribute the power rather than on how to increase the speed of charging. The main idea is to integrate the smart grid into the charging station and make a platform on which all types of standard AC or DC charging can be done. The OpenV2G project is the community project which focuses on developing the codes for the integration of the smart grid with the Charging Station.

Once implemented, it may increase the time required to charge the electric vehicles but allows to reduce the possibility of potential blackout and also integrate all forms of renewable energy sources into the system.

* 1. Objective:

The car charging station based on ISO/IEC 15118 has already been implemented and communication between the EVSE and EV is done using Green PHY (powerline communication).Now the next objective is to integrate an HMI to the system which allows the user to interact and charge the vehicle.

Conditions to be followed:

1. Another platform should be used to run and interface the HMI as the EVSE has enough processes to run.
2. The EVSE should be a client and should have the code modified in minimal invasive method.
3. The three devices should simultaneously run when the specified conditions are satisfied.
4. Should satisfy all the state and event condition as mentioned in the OpenV2G project.

1. Methodology

This chapter provides an overview of work and information to which the project is worked up. These include, inter alia, the former way of loading a vehicle as well as the previous exchange of information and the different vehicle connectors used for loading of electric vehicles. Further more information on this work is a study work, which is describes the ISO 15118 accurately and a dissertation of Dr. Marc Mültin which is engaged in the electric vehicle as a "flexible consumers and energy storage device in the smart home".

* 1. IEC 62196: vehicle plug

Connector types and charging modes of electric vehicles are defined by the International Electrotechnical Commission in IEC 62196 (Wiki\_plug, 2016),

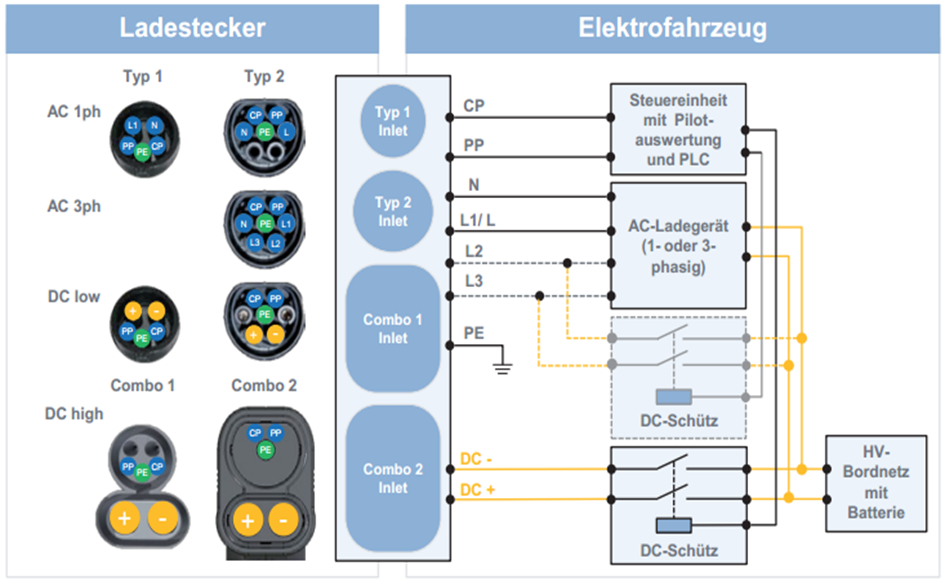
The second part of the standard was published in 2011 and includes different types of connectors. This includes three of the most popular at this time charging plug.

The Type 1 charging plug, which in Figure 2.1 is shown, takes its specification of the SAE J1772. This was first published in 1996 by the Society of Automotive Engineers and has since been expanded and maintained by this. The disadvantage of this connector type is found in the contacts since these do not allow a three-phase charging with alternating current.

Type 2 of the standard charging plug is the currently the most built-up type of charging plug systems and found in figure 2.1. The plug finds its origins through a collaboration of the connector manufacturer Mennekes with the power company RWE and the carmaker Daimler. The naming of the Mennekes plug thus receives this by its manufacturer.

The third plug-in type plugged into the standard, the EV Plug Alliance, was defined by a consortium led by French and Italian companies. Due to the low demand, the further production of the plug was discontinued.

For all defined types of connectors as defined in Type 1 Signal contacts CP (Control Pilot) and PP are (Proximity pilot) included which allow charging to IEC 61851



In the Figure Signal contacts shown are defined as follows:

|  |  |  |
| --- | --- | --- |
| abbreviation | Contact | function |
| CP | Control pilot | Control signals charging station🡪electric vehicle |
| PP | Proximity pilot | Check the presence of a charging cable |
| N | Neutral | For AC charging |
| L1, L2, L3 | Current-carrying phases | For AC charging with a (L1 / L) or three (L1, L2, L3) Phases |
| PE | Protective Earth | protective conductor |
| DC +/- | Current-carrying phases | For DC charging |

* 1. IEC 61851

The IEC 62196 is an international standard for a number of types of plugs and charging modes for electric vehicles and of the International Electrotechnical Commission maintained (IEC). The standard is valid in Germany as a DIN standard DIN EN 62196. It consists of several parts which have been passed in succession. The third part was published in June 2014. In June 2015, the standardization process for part 4 (light- weight electrical connections) began.

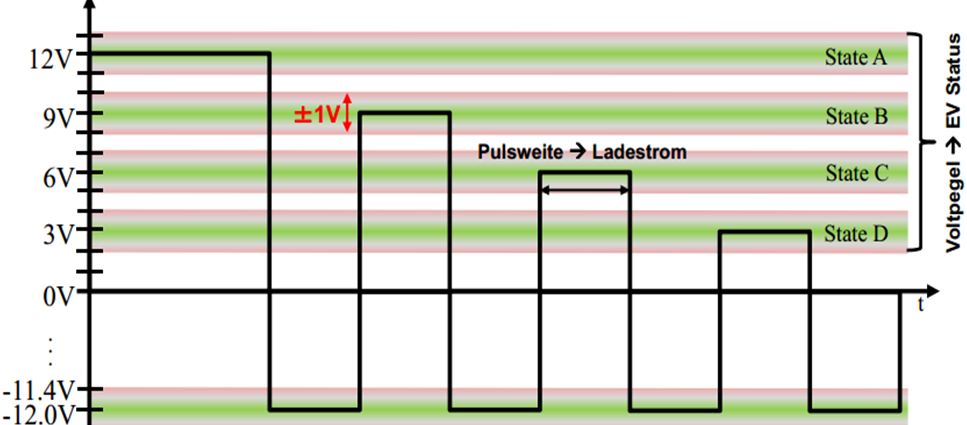
The standard adopts the IEC 61851 definition for a signal pin that switches the charging current - the charging station remains de-energized until an electric vehicle is connected. During the charging process, the vehicle cannot be put into operation.

Prior to the definition of a charging process according to ISO / IEC 15118, the charging parameters required for the charging process were defined using a PWM signal according to IEC 61851. The signals of the Control Pilot (CP), Protective Earth (PE) and Proximity Pin (PP) contacts described in section 2.1 are required to determine the parameters required for loading.

For charging the vehicle, both communication subscribers are first connected to one another. A 1 kHz signal with 12V is generated on the CP contact from the side of the charging column. The pulse width of the signal indicates which maximum power can be provided by the charging column. In this case, 10% max. 10A, 25% 16A, 50% max. 32A and 90% quick charge (Wiki\_Stecker, 2016).

On the vehicle side, resistors are connected between CP and PE or PP and PE. Different charging states are indicated by different switchable levels of the voltage between the CP and PP contacts, as shown in Figure 2.2. Please note that the negative voltage value is permanently -12V, and only the positive values ​​change. A definition of the individual states is Table 2.2.

Lastly, a vehicle-side resistance between the PP and the PE contact indicates the maximum possible charging current of the electric vehicle. The greater the resistance used, the lower the maximum charging current. Specifically, for a 1.5kΩ resistor, a maximum charging current of 13A, a maximum of 20A with a resistance of 680Ω, at 220Ω the maximum charging current 32A and 63A is at 100Ω.



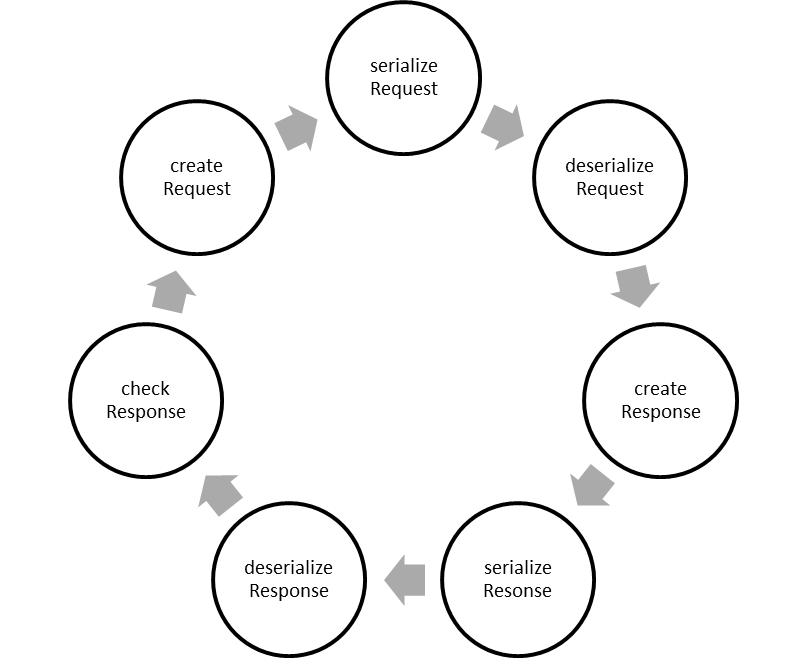
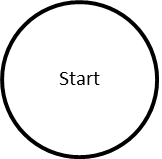
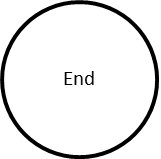
|  |  |  |
| --- | --- | --- |
| Level | State | Condition Description |
| 12 ± 1 V | State A | Electric vehicle is not connected |
| 9 ± 1 V | State B | connected electric vehicle, not charging Ready |
| 6 ± 1 V | State C | connected electric vehicle, ready to charge |
| 3 ± 1 V | State D | connected electric vehicle, ready for loading, ventilation needed |
| 0 ± 1 V | State E | Network problem, PP Short to earth |
| -12V | State F | Vehicle unavailable Error |

* 1. ISO 15118

The International Organization for Standardization (ISO) and the International Electronic Commission (IEC) in 2009 started to describe the standardization of a "digital IP-based communication protocol" between electric vehicle and charging station( Mültin, 2014), This should be a "plug-and-charge" mechanism for authentication, authorization, accounting, and for load control, so that needed to load enable parameters are stored in the vehicle and the user both communication parties must connect only. The individual communication Content will be the level of tension control pin signal from Chapter2.2 correspondingly Figure 2.3 assigned.

The full schedule of communication stacks for AC or DC charging an electric vehicle according to ISO / IEC 15118 can be found in Figure 7.1 to Figure 7.4, an overview of the variables contained in the messages within the AC communication stack is described together with an overview of the ISO / IEC 15118 in a previous study, work (Barth, 2015)

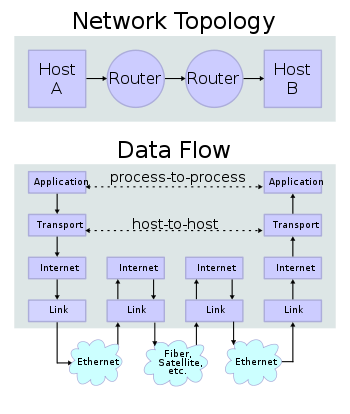
* + 1. Open V2G Project

An already far-reaching example for the implementation of a communication pack according to ISO / IEC 15118 has already been initiated by the support of Siemens Corporate Technology as Open Source project (OpenV2G, 2016). Both the loading column and the vehicle side are displayed in a program code and the messages are generated, checked, and the next message is generated. At the current status (version 0.9.3), the sequence of the individual requests and responses, as well as the message contents to the direct current and alternating current charge, can be derived very well. It is one of the objectives of this thesis to divide this code into a program for every communication user.

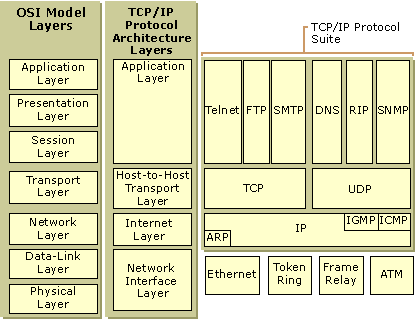
* 1. TCP/IP Communication:

The Internet protocol suite is the conceptual model and set of communications protocols used on the Internet and similar computer networks. It is commonly known as TCP/IP because the original protocols in the suite are the Transmission Control Protocol (TCP) and the Internet Protocol (IP).

The Internet protocol suite provides end-to-end data communication specifying how data should be packetized, addressed, transmitted, routed and received. This functionality is organized into four abstraction layers which are used to sort all related protocols according to the scope of networking involved. From lowest to highest, the layers are the link layer, containing communication methods for data that remains within a single network segment (link); the internet layer, connecting independent networks, thus providing internetworking; the transport layer handling host-to-host communication; and the application layer, which provides process-to-process data exchange for applications.

Technical standards specifying the Internet protocol suite and many of its constituent protocols are maintained by the Internet Engineering Task Force (IETF). The Internet protocol suite model is a simpler model developed prior to the OSI model.

TCP/IP is a two-layer program. The higher layer, Transmission Control Protocol, manages the assembling of a message or file into smaller packets that are transmitted over the Internet and received by a TCP layer that reassembles the packets into the original message. The lower layer, Internet Protocol, handles the address part of each packet so that it gets to the right destination. Each gateway computer on the network checks this address to see where to forward the message. Even though some packets from the same message are routed differently than others, they'll be reassembled at the destination.



TCP/IP (Transmission Control Protocol/Internet Protocol) is the basic communication language or protocol of the Internet. It can also be used as a communications protocol in a private network (either an intranet or an extranet). TCP/IP uses the client/server model of communication in which a computer user (a client) requests and is provided a service (such as sending a Web page) by another computer (a server) in the network. TCP/IP communication is primarily point-to-point, meaning each communication is from one point (or host computer) in the network to another point or host computer. TCP/IP and the higher-level applications that use it are collectively said to be "stateless" because each client request is considered a new request unrelated to any previous one (unlike ordinary phone conversations that require a dedicated connection for the call duration). Being stateless frees network paths so that everyone can use them continuously. (Note that the TCP layer itself is not stateless as far as any one message is concerned. Its connection remains in place until all packets in a message have been received.)

* 1. UML
     1. Introduction

UML stands for Unified Modelling Language. UML is a way of visualizing a software program using a collection of diagrams. The notation has evolved from the work of Grady Booch, James Rumbaugh, Ivar Jacobson, and the Rational Software Corporation to be used for object-oriented design, but it has since been extended to cover a wider variety of software engineering projects. Today, UML is accepted by the Object Management Group (OMG) as the standard for modelling software development.

* + - 1. UML 2.0

UML 2.0 helped extend the original UML specification to cover a wider portion of software development efforts including agile practices.

Here are some of the changes made to UML diagrams in UML 2.0:

• Improved integration between structural models like class diagrams and behaviour models like activity diagrams.

• Added the ability to define a hierarchy and decompose a software system into components and sub-components.

• The original UML specified nine diagrams; UML 2.x brings that number up to 13. The four new diagrams are called: communication diagram, composite structure diagram, interaction overview diagram, and timing diagram. It also renamed state chart diagrams to state machine diagrams, also known as state diagrams.

* + - 1. Types of UML Diagrams

The current UML standards call for 13 different types of diagrams: class, activity, object, use case, sequence, package, state, component, communication, composite structure, interaction overview, timing, and deployment.

These diagrams are organized into two distinct groups: structural diagrams and behavioural or interaction diagrams.

1. Structural UML diagrams

• Class diagram

• Package diagram

• Object diagram

• Component diagram

• Composite structure diagram

• Deployment diagram

1. Behavioural UML diagrams

• Activity diagram

• Sequence diagram

• Use case diagram

• State diagram

• Communication diagram

• Interaction overview diagram

• Timing diagram

In our project we deal with behavioural UML diagram to explain how the Charging station components behave at different states and kinds. In Behavioural UML the state machine diagram and the sequence diagram is well used.

* + 1. State Machine Diagram

A state diagram shows the behaviour of classes in response to external stimuli. Specifically a state diagram describes the behaviour of a single object in response to a series of events in a system. Sometimes it's also known as a Harel state chart or a state machine diagram. This UML diagram models the dynamic flow of control from state to state of a particular object within a system.

* + - 1. Difference between a State Diagram and a Flowchart

A flowchart illustrates processes that are executed in the system that change the state of objects. A state diagram shows the actual changes in state, not the processes or commands that created those changes.

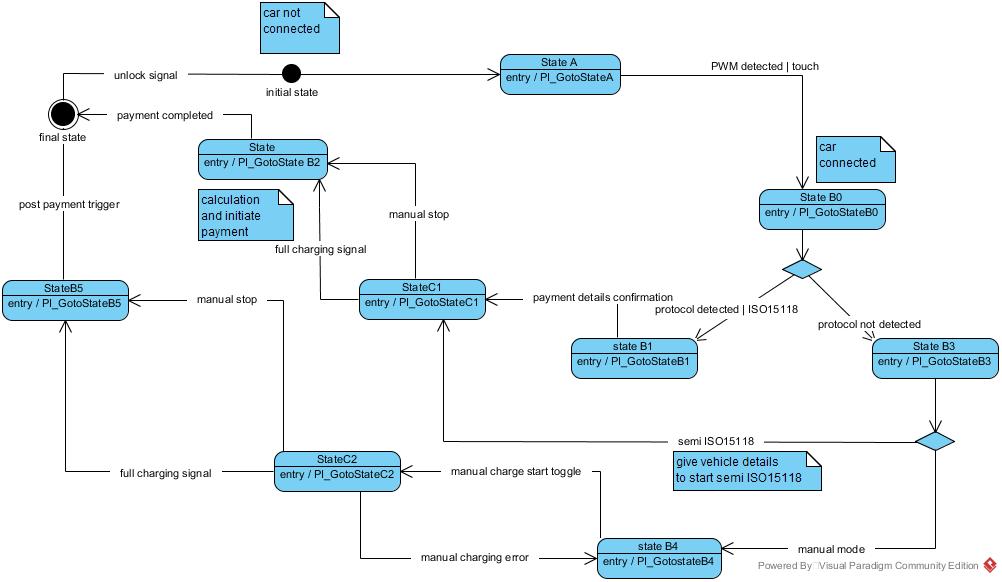
* + - 1. Steps to Drawing a State Diagram

Before you begin your drawing find the initial and final state of the object in question.

Next, think of the states the object might undergo. For example, in e-commerce a product will have a release or available date, a sold out state, a restocked state, placed in cart state, a saved on wish list state, a purchased state, and so on.

Certain transitions will not be applicable when an object is in a particular state, for example a product can be in a purchased state or a saved in cart state if its previous state is sold out.

* + - 1. The Charging Station – State Machine Diagram



* + 1. Sequence Diagram

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time. They're also called event diagrams. A sequence diagram is a good way to visualize and validate various runtime scenarios. These can help to predict how a system will behave and to discover responsibilities a class may need to have in the process of modelling a new system.

* + - 1. How to Use Sequence Diagrams

• Model and document how your system will behave in various scenarios

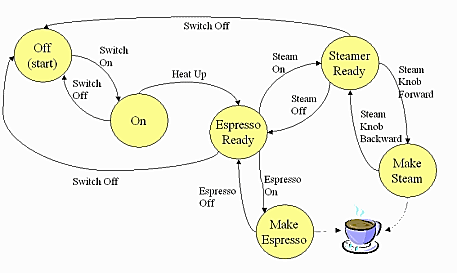
• Validate the logic of complex operations and functions

* + - 1. The Charging Station – The State Machine Diagram



* 1. Automatic State Diagram:
     1. Introduction

An automatic state machine consists of states, status transitions, and actions. The purpose of these tools is to implement the control of a system which takes into account past, present and future events. Each state is associated with actions that occur when it is entered or exited. A state must be defined at any time during the runtime of the system. A state transition, on the other hand, describes the connections of the individual states to one another as well as the event which must occur in order to switch between the states.



An illustration of such an automatic state machine is provided by a coffee machine as shown in Figure 2.6. The state machine starts with the start state, which in the present example is the switched off state of the coffee machine. Here, a status change is only possible by the switch-on transition. Depending on the user's input, the machine can be set to "Espresso ready", "Steamer ready" or "OFF". This example shows particularly well the inclusion of different time forms. To be ready for operation, the coffee machine had to be switched on in advance and brought to a defined temperature. Which state is assumed in the further course depends on unforeseeable events. It is also clearly shown that the machine cannot activate the individual states at any time. In order to be able to assume a particular state, this must be connected to that of a state transition from the current state. Thus, in the given example, no coffee can be prepared as long as the coffee machine is in the "ON" state. The programming of an automatic state machine can be implemented with the switch case function. A basic state is already defined in advance. As soon as an event that might cause a state change occurs, the function is started. The currently defined state is queried and the state change is defined in the corresponding case in order to reach a new state. An example for the coffee machine Figure 2.6 is shown in Figure 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;

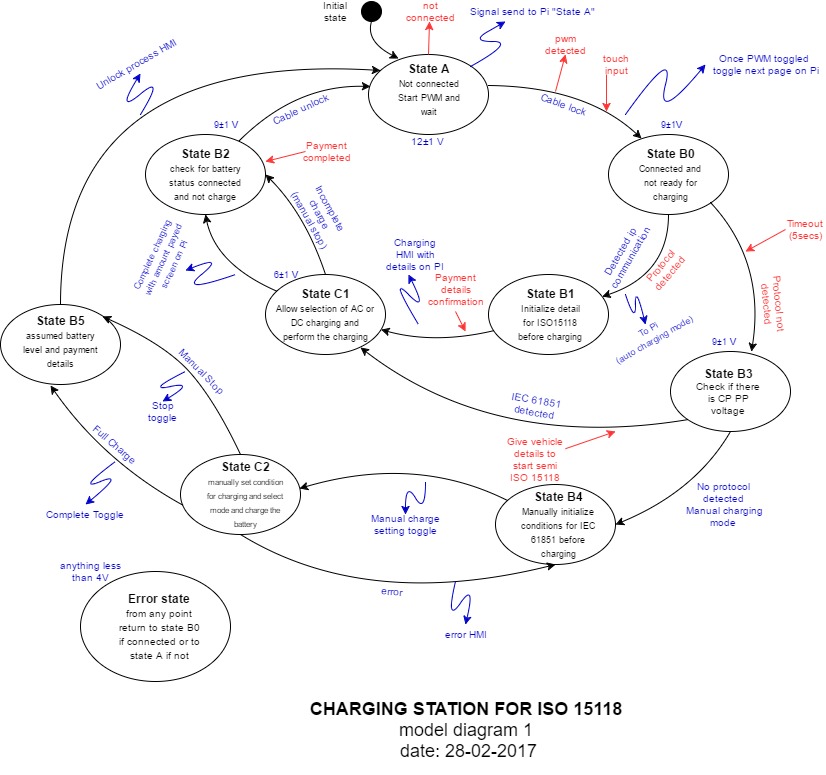
…

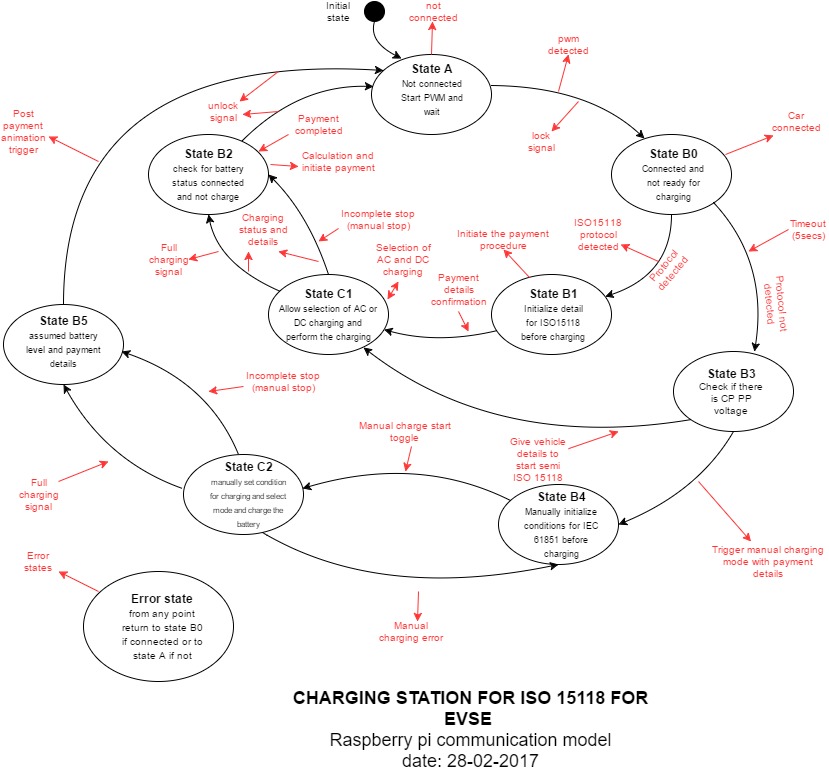
…

* + 1. State diagram of the charging station

The charging station requires the need of an automatic state diagram to explain its working. It helps the developers visualize the process and set up the required process faster. In the state diagram designed we have used the online drawing tool **www.draw.io**. It is an easy to use tool with several options.

The finalized state diagram is shown in figure … and also in figure … There is a separate state diagram for the EVSE process and the signal the Raspberry pi will work with.



The Following diagram show the possible signals which exists from each state of the system and what should be interfaced with the system.

1. Software Description
   1. FileZilla

FileZilla is a free software, cross-platform FTP application, consisting of FileZilla Client and FileZilla Server. Client binaries are available for Windows, Linux, and macOS, server binaries are available for Windows only. The client supports FTP, SFTP and FTPS (FTP over SSL/TLS).

FileZilla's source code is hosted on Source Forge and the project was featured as Project of the Month in November 2003.However, there have been criticisms that Source Forge bundles malicious software with the application; and that FileZilla stores users' FTP passwords insecurely.

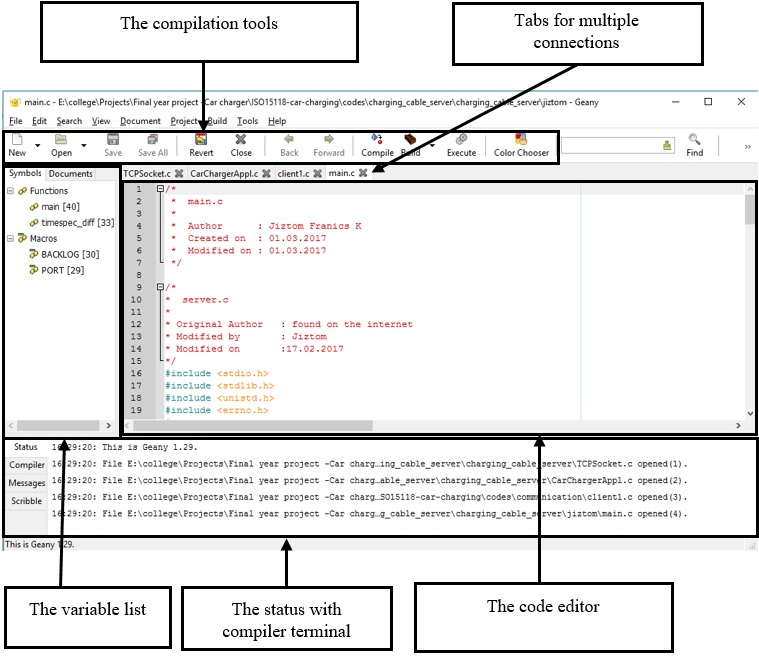
These are some features of FileZilla.

* Transfer files in FTP, SFTP, encrypted FTP such as FTPS and SFTP.
* Support IPv6 which is the latest version of internet protocol.
* Available in 47 languages worldwide.
* Supports resume which means the file transfer process can be paused and continued.
* Tabbed user interface for multitasking, to allow browsing more than one server or even transfer files simultaneously between multiple servers.
* Site Manager to manage server lists and transfer queue for ordering file transfer tasks.
* Bookmarks for easy access to most frequent use.
* Drag and drop to download and upload.
* Directory comparison for comparing local files and server files in the same directory. When the file doesn't have the same information (name not match, or size not match) it will highlight that file in colour.
* Configurable transfer speed limits to limit the speed transferring the files, which helps reducing error of transferring
* Filename filters, users can filter only specific files that have the conditions they want.
* Network configuration wizard, help configuring confusing network settings in form of step-by-step wizard
* Remote file editing, for quickly edit file on server side on-the-fly. No need to download, edit on the computer and re-upload back to the server.
* Keep-alive, if the connection has been idle for the long time it will check by sending keep-alive command.
* HTTP/1.1, SOCKS5 and FTP-Proxy support
* Logging to file
* Synchronised directory browsing
* Remote file search to search file on the server remotely.



* 1. Geany

Geany is a lightweight GUI text editor using Scintilla and GTK+, including basic IDE features. It is designed to have short load times, with limited dependency on separate packages or external libraries on Linux. It has been ported to a wide range of operating systems, such as BSD, Linux, mac OS X, Solaris and Windows. Because Windows lacks a virtual terminal equivalent, the Windows port lacks an embedded terminal window. Also missing from the Windows version are the external development tools present under UNIX, unless installed separately by the user. Among the supported programming languages and mark-up languages are C, C++, C#, Java, JavaScript, PHP, HTML, LaTeX, CSS, Python, Perl, Ruby, Pascal, Haskell, Erlang, Vala and many others.

In contrast to traditional Unix-based editors like Emacs or Vim, Geany more closely resembles a small and fast IDE.

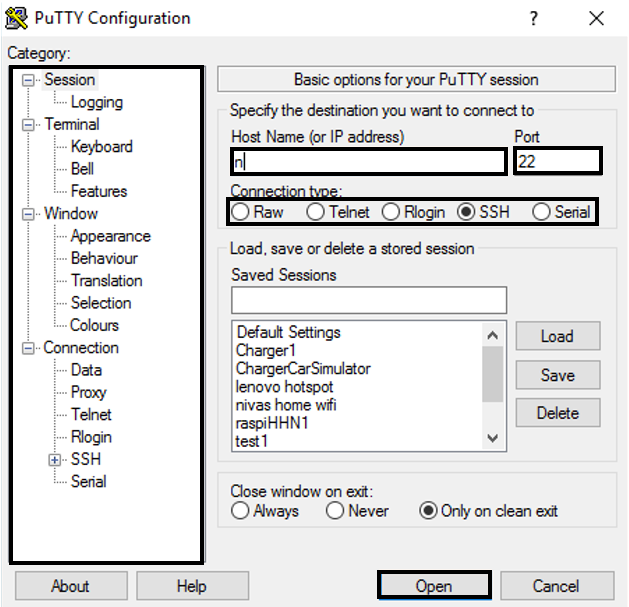
.

* 1. Putty Terminal

PuTTY is a free and open-source terminal emulator, serial console and network file transfer application. It supports several network protocols, including SCP, SSH, Telnet, rlogin, and raw socket connection. It can also connect to a serial port. The name "PuTTY" has no definitive meaning.

PuTTY was originally written for Microsoft Windows, but it has been ported to various other operating systems. Official ports are available for some Unix-like platforms, with work-in-progress ports to Classic Mac OS and mac OS, and unofficial ports have been contributed to platforms such as Symbian, Windows Mobile and Windows Phone.

PuTTY was written and is maintained primarily by Simon Tatham and is currently beta software.



* 1. EB GUIDE 6.3
  2. [www.draw.io](http://www.draw.io)
  3. Visual Paradigm 14.0
  4. Compiling the program

To compile and run a program, you must first establish a connection between the local computer and the respective development board. A secure shell (SSH) is used to make locally available a remote command line available. This is a network protocol that creates an encrypted network connection (Wiki\_SSH, 2016).

The function call to call such a connection is similar to the one described in chapter 3.3. First, the secure shell is set up with the call "**ssh <Username> @ <IP address>**" in the Linux terminal. For security reasons, a password request is also carried out in the next step.

Both usernames (**root**) and passwords (**zebematado**) are used for both boards. However, the IP addresses must have differences (192.168.37.250 and 192.168.37.251). If the password is correct, the current date and the last login will be displayed as shown in Figure 3.5. The Linux terminal in which this call was executed now represents a command line of the EVAchargeSE board.

To compile the previously transmitted C-code must first be coordinated into the corresponding directory. The commands "pwd" and "cd" are used according to Table 3.4 for the overview of the current directory path and for navigation into other directories. The command "**gcc -o <NAME> \* .c**" is called for the EVAchargeSE board to compile the sourcefiles in the directory. The variable <NAME> used here can be named as desired and contains the start file. To start the compiled code, the previously defined start file is called by **"./ <NAME>"**. A Linux terminal, which is compiled and started in the serial\_Programming directory, is shown in Figure 3.6.

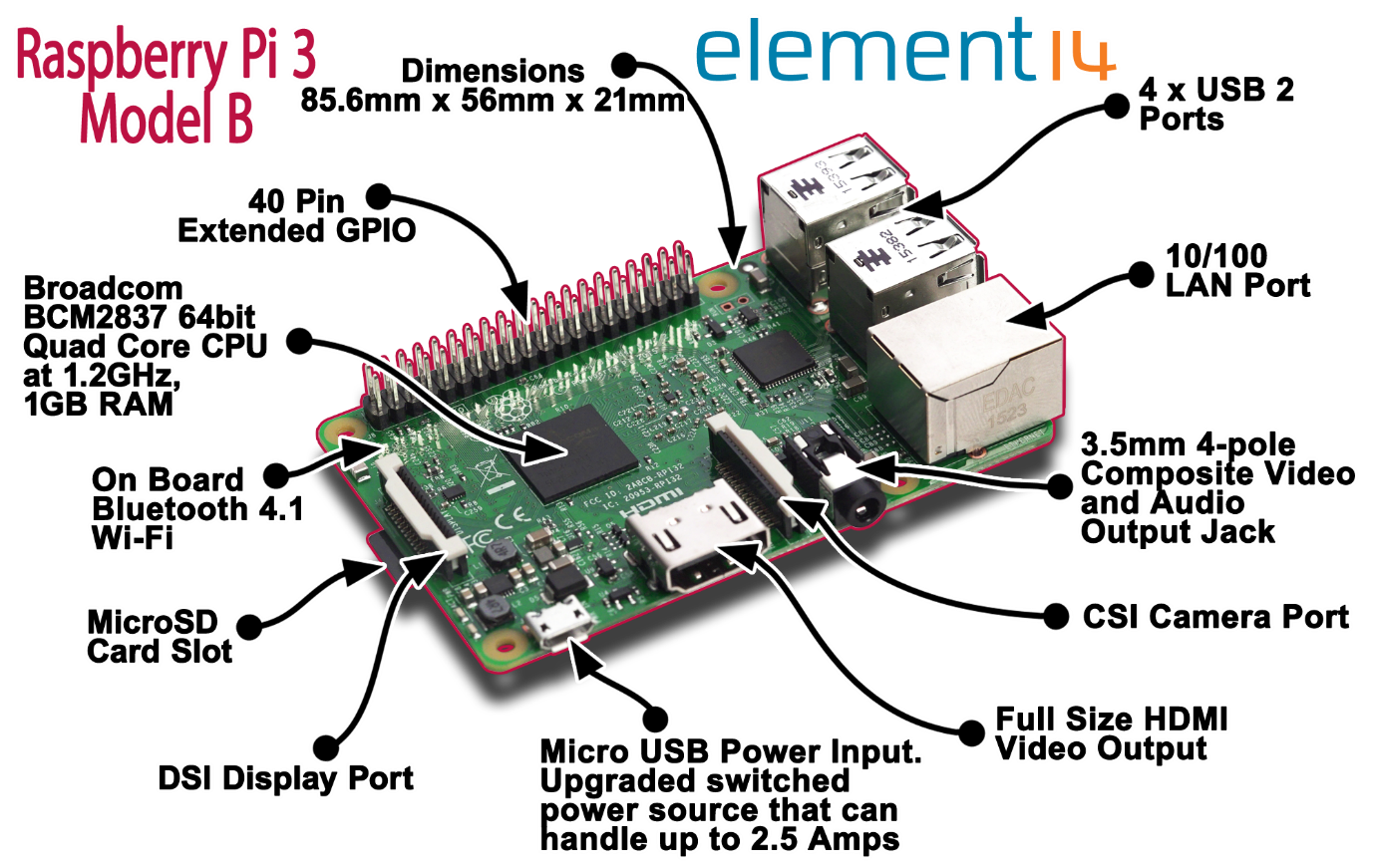
If an executable file is to be interrupted, this can be done using the key combination "Ctrl" and "C". Once the work in the terminal of the development board is completed, the SSH connection can be terminated by "exit".

* + 1. General Nutshell commands

|  |  |
| --- | --- |
| Allgemeine Nutshell Befehle |  |
| ssh [root@192.168.37.250](mailto:root@192.168.37.250) | Establishing a secure shell connection |
| sftp [root@192.168.37.250](mailto:root@192.168.37.250) | Connection setup of an SFTP protocol |
| ifconfig | Configuration and status display of all available network interfaces |
| cd <Ordnername> | Change Directory: To a subdirectory of the current folder |
| cd .. | Change to parent directory |
| cd /home/user | Change to the / home / user file |
| vi <Dataname> | Open a file in the vi editor. To return to the terminal from the editor, press "ESC", enter ": q" and confirm with Return. |
| gcc –o <NAME> \*.c | Compile of all source files accordingly |
| ./<NAME> | Starting from a program accordingly |
| ping <IP-address> | Sends data packets to an IP address to check the presence of a connection |
| ping –I qca0 <IP-Address> | Ping command over network interface qca0 |
| dir | Displays all folders and files in the current directory. |
| pwd | Returns current folder path |
| rm <Data> | Delete a file |

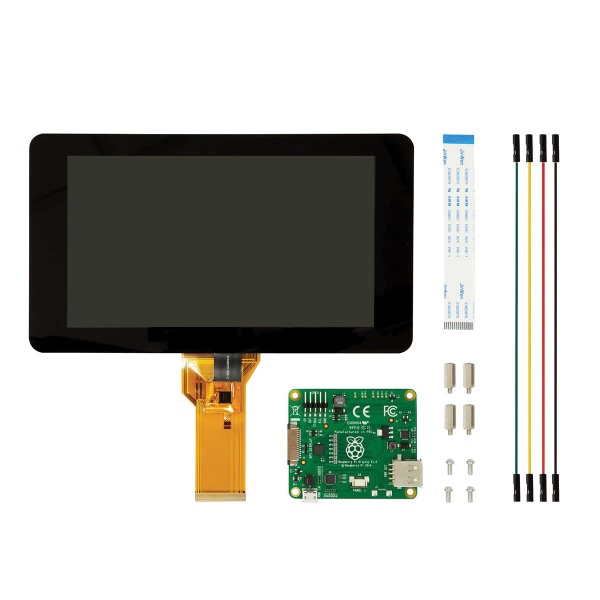
1. Hardware Description
   1. Raspberry Pi 3
      1. Introduction

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. The original model became far more popular than anticipated, selling outside of its target market for uses such as robotics. Peripherals (including keyboards, mice and cases) are not included with the Raspberry Pi. Some accessories however have been included in several official and unofficial bundles

All models feature a Broadcom system on a chip (SoC), which includes an ARM compatible central processing unit (CPU) and an on-chip graphics processing unit (GPU, a VideoCore IV). CPU speed ranges from 700 MHz to 1.2 GHz for the Pi 3 and on board memory range from 256 MB to 1 GB RAM. Secure Digital (SD) cards are used to store the operating system and program memory in either the SDHC or Micro SDHC sizes. Most boards have between one and four USB slots, HDMI and composite video output, and a 3.5 mm phone jack for audio. Lower level output is provided by a number of GPIO pins which support common protocols like I²C. The B-models have an 8P8C Ethernet port and the Pi 3 has on board Wi-Fi 802.11n and Bluetooth.

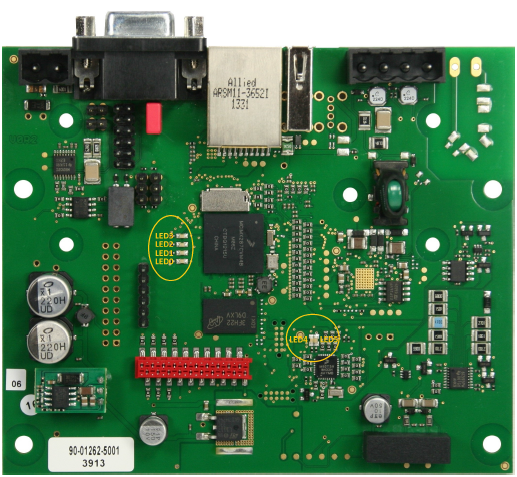
* + 1. Raspberry pi 7” Official Touch screen

**Key Features:**

* Screen Dimensions: 194mm x 110mm x 20mm (including standoffs)
* Viewable screen size: 155mm x 86mm
* Screen Resolution 800 x 480 pixels
* 10 finger capacitive touch.
* Connects to the Raspberry Pi board using a ribbon cable connected to the DSI port.
* Adapter board is used to power the display and convert the parallel signals from the display to the serial (DSI) port on the Raspberry Pi.
* Will require the latest version of Raspbian OS to operate correctly.

* 1. EVAChargSE Board
     1. Introduction

Plug-in Electric Vehicle (PEV) charging is a major Smart Grid application, with the goal to provide the means of charging PEVs at home, at work, and in public areas such as shopping centers and airports. This is also a global initiative for many major auto manufacturers.

Car\_150w.png In 2011, global auto manufacturers Audi, BMW, Daimler, Ford Motor Company, General Motors, Porsche and Volkswagen all put their considerable weight behind the HomePlug Green PHY specification for connectivity with concurrent electric vehicle charging.

* + 1. EVACharge SE board

EVACharge SE is an ISO 15118 compliant controller for electric vehicle charging stations. The board contains the PLC communication via CP with PWM generation and HomePlug Green PHY integration. The board will be provided with a Linux operating system. The board can act as EVSE as well as PEV.

• Based on the Freescale i.MX287

• Storage: eMMC 4 GB

• Network interface: Fast Ethernet

• Operating system: Debian jessie, Kernel 3.10 (or newer)

• RAM: 128 Mbyte

|  |  |
| --- | --- |
| Parameter | Value |
| Power Supply | 12 V |
| Power Consumption | Max. 4 W (2.6 W in idle mode) - Plus Power for USB devices |
| Temperature range | -40 °C to +85 °C |
| Air humidity | 95% rel. humidity (non-condensing) |
| Outline Dimension | 100mm x 120 mm x 20 mm |
| Weight | 92g |
| RoHS | EVACharge SE is manufactured RoHS compliant |

* + 1. Applications

EVACharge SE is a communication platform for Electric Vehicle Supply Equipment (EVSE) as well as plug-in electric vehicles (PEV). It enables the charge controller to communicate with electric vehicles (EVs) that are ISO 15118 / DIN 70121 compliant. For communication between EVSE and PEV it supports CP (control pilot) and PP (proximity pilot) signalling including Green PHY communication. The PP signal can also be used to simulate cables with different charge current capability. Possible Applications:

• Charge controller in electric vehicle supply equipment (EVSE)

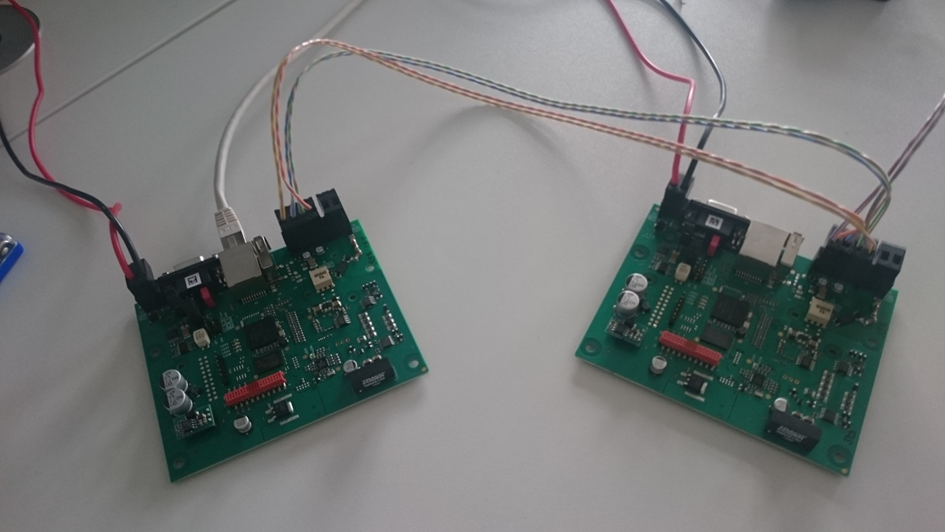
• Charge controller in plug-in electric vehicles (PEV)

• Simulators for tests of PEV or EVSE

* 1. Communication between EVSE and EV

A UART interface is used for communication between the i.MX 28 and the KL02. Since the i.MX 28 subsequently controls the program sequence, it is defined as the master. For this reason, the KL02 is already programmed as a slave in the delivery state. This means that he only responds to requests according to chapter 5.7.

To establish the connection, the settings are used according to the "Board Support Package" document (I2SE, 2016). This defines a baud rate of 57600 Bd with 8 data bits and 1 stop bit. Furthermore, the modem device over which the data is to be transmitted must be known. When the port is opened, the connection is saved in a file handler so that a differentiation can take place in the case of several existing connections. For the initialization of such a communication protocol, there are already existing sources which are used after changing the configurations (Sweet, 2016).

After initialization of the UART interface messages can be transmitted. To send a message, it is written to an array. With the command **<write (filehandle, array, number of bytes to be transferred)>**, the array is sent via UART. The return value of the function indicates the number of bytes sent. **<Read (filehandle, array, number of bytes to be received)>**, the data to be received are written to the previously defined array. In both cases, you must know how many bytes are expected or sent. The frames defined in 5.7 are thus always written, sent and also received again in an array.

* 1. Communication between EVSE and Raspberry Pi

The communication is done based on the TCP/IP model as explain in Chapter 4.4. The TCP/IP communication is done over the LAN. The system is designed with the Raspberry pi as the Server and the EVSE as the client.

As per the logic the Raspberry pi will wait for the client to connect and will accept the connection followed by binding the socket to the IP. Then it will listen to the request or send a message.

The designed model is a bi-directional in nature so both the Raspberry pi and the EVSE can both send and receive the messages. The messages are send in a way it satisfies the state diagram. The initial model will be designed using with the dummy parameters with the main focus to just satisfy the ISO 15118 Protocol with conditional opening for older generation of charging systems / protocols.

The entire process will be written in C language and will be compiled and run using the gcc compiler mentioned in chapter 5.7

The output and the code used to achieve this link is attached in the appendix.

* 1. Integrating the system

1. Results

The HMI was developed satisfying the state diagram using EB GUIDE 6.3 and the Raspberry pi was integrated with the EVSE using TCP/IP communication and the software codes was successfully implemented.

The three Linux systems was successfully synchronised with the dummy variables required by the HMI display. The output of the code is attached in the appendix C

Separate output code of EV and EVSE is also attached additionally.

The output of the HMI model created using EB GUIDE is attached in Appendix - D

1. Conclusion

The ISO/IEC 15118 protocol was successful implemented on an example parameter model and the further implementation of the project includes the synchronization of the CAN bus signal produced by the Control Unit in the car.

This project on the long run maybe the solution to the increasing pollution and over dependence of the depleting fossil fuels. In India, it will take a bit more time to implement as the Smart GRID is yet to be implemented and needs a few more advanced infrastructure.

This shows high prospects into harnessing the available renewable resources within the country, also encouraging the citizens to join the smart grid to improve the livelihood and move towards a country with no smokes and fumes and thus a clean air and green country.

As there are not conditions set for the type of charging used it could be modified to suit other applications like paid heating service, charging larger vehicles and so on…

1. Summary & outlook

Appendix

Appendix - A

/\*

\* main.h

\*

\* Created on : 06.03.2017

\* Author : Jiztom Francis K

\*

\*

\*/

#ifndef MAIN\_H\_

#define MAIN\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <netdb.h>

#include <time.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include "transfer.h"

#include "switchpi.h"

#define ST\_PLUG 1

#define ST\_SIGN\_IN 2

#define ST\_CHARGE 3

#define ST\_POST\_CHARGE 4

#define HMI\_READY 5

#define CABLE\_DETECTED 11

#define CABLE\_LOCK 12

#define PROTOCOL\_DETECT 13

#define LOCKING\_ERROR 14

#define REGISTER 21

#define AUTHORIZATION 22

#define START\_CHARGE 31

#define CHARGING\_STATUS 32

#define MANUAL\_STOP 33

#define FULL\_CHARGE 34

#define METER\_RECEIPT 36

#define INITIATE\_PAYMENT 41

#define PAYMENT\_SUCESSFUL 42

#define PAYMENT\_UNSUCESSFUL 43

#define CABLE\_UN\_LOCK 45

#define ISO15118\_DETECTED 51

#define IEC61851\_DETECTED 52

#define NO\_PROTOCOL\_DETECTED 53

#define SEMI\_ISO15118 61

#define MANUAL\_CHARGING 62

#endif /\*MAIN\_H\_\*/

/\*

\* main.c

\*

\* Author : Jiztom Francis K

\* Created on : 01.03.2017

\* Modified on : 08.03.2017

\*/

/\*

\* server.c

\*

\* Original Author : found on the internet

\* Modified by : Jiztom

\* Modified on :17.02.2017

\*/

#include "main.h"

///////global variable //////////////////////////////////////

int state = 0;

//int socket\_fd;

long timespec\_diff(struct timespec a, struct timespec b)

{

long diff;

diff = (a.tv\_sec - b.tv\_sec);

return diff;

}

int main()

{

struct sockaddr\_in server;

struct sockaddr\_in dest;

struct timespec start;

struct timespec end;

long t\_diff;

int status, socket\_fd, client\_fd,num;

socklen\_t size;

int i=0;

char buffer[1024];

char buff[100];

char buff2[100];

// memset(buffer,0,sizeof(buffer));

unsigned char code;

unsigned int value;

/////////////variables for inner loop////////////

int language;

int condition = 0;

int detect = 0;

int lock\_condition = 0;

int signal = 0;

socket\_fd = init\_tcp();

if (!socket\_fd) exit(1);

while(1)

{

size = sizeof(struct sockaddr\_in);

/\* if (accept\_client(socket\_fd)==-1)

{

fprintf(stderr, "Accept did not work\n");

exit(1);

}\*/

if ((client\_fd = accept(socket\_fd, (struct sockaddr \*)&dest, &size))==-1 )

{

perror("accept");

return(-1);

}

printf("\nServer got connection from client %s\n", inet\_ntoa(dest.sin\_addr));

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

while(1)

{

if(condition == 0) ////language selection/////

{

printf(" Please choose the language to be selected");

printf(" \n 1. English \t 2. German \n");

printf(" your option please : \n");

scanf("%d", & language );

printf("\n");

sendd(client\_fd , HMI\_READY );

/////----->>>>> send signal to the EB guide for language selection

condition++;

}

if(condition == 1)////// plug detection and intialization//////

{

printf(" \n please insert the plug into the system \n");

init\_statemachine();

detect = fire\_event(CABLE\_DETECTED , 0 ,client\_fd);

if( detect == 1)

{

printf("\n the cable has been connected and the car has been detected");

///////the signal from the EVSE for the lock status ////////

signal = receivee(socket\_fd , &code, &value);

if(fire\_event( CABLE\_LOCK , signal,client\_fd) == 1)

{

condition++;

}

else

{

init\_statemachine();

condition = 1;

}

}

else

printf(" \n the cable has not been detected continue loop" );

}

if(condition == 2)

{

printf("\n the vehicle status is :");

//signal =0;/////signal for the protocol detected////////////

signal = receivee(client\_fd , &code, &value);

fire\_event( PROTOCOL\_DETECT , code , client\_fd);

condition++;

/\*if (code == ISO15118\_DETECTED)

{

printf("\n the ISO 15118 was detected and proceeding to next state \n");

fire\_event(PROTOCOL\_DETECT , ISO15118\_DETECTED ,client\_fd);

condition++;

}

else if(code == IEC61851\_DETECTED )

{

printf("\nthe IEC 61851 was detected\n");

fire\_event(PROTOCOL\_DETECT, IEC61851\_DETECTED,client\_fd);

////condition = //////////////// ;

}

else if(code == MANUAL\_CHARGING)

{

printf("\n no protocol detected will need to move towards manual charging\n");

fire\_event(PROTOCOL\_DETECT, MANUAL\_CHARGING,client\_fd);

//////condition = //////////

}

else /// is this even required?

{

printf ("\n error in detection. Lost communication\n resetting connection \n");

init\_statemachine();

condition =0;

}\*/

}

if(condition == 3)

{

printf("\n the protocol has been detected . Now initiating the information and account details process\n");

fire\_event(REGISTER , 0 , client\_fd);

printf("\n the payment and the initial requirement has been done\n");

fire\_event(AUTHORIZATION , 0 , client\_fd);

condition++;

}

if( condition == 4)

{

printf(" \n the car is ready for charging.\n\n please press the button to charge the vehicle\n");

//fire\_event(START\_CHARGE, 0 , client\_fd);

//do

//{

fire\_event(CHARGING\_STATUS , 0 , client\_fd);

fire\_event(MANUAL\_STOP,0,client\_fd);

//}while((fire\_event(FULL\_CHARGE,0 , client\_fd)|| fire\_event(MANUAL\_STOP,0,client\_fd)) == 1);

printf("\nthe car has stopped charging");

printf("\nthe payment details are as follows:");

fire\_event(METER\_RECEIPT , 0,client\_fd);

condition++;

}

if(condition == 5)

{

printf("\nThe payment will be processed now");

///////try the payment using the details logged before////

fire\_event(INITIATE\_PAYMENT,0,client\_fd);

fire\_event(PAYMENT\_SUCESSFUL,0,client\_fd);

condition++;

/\*if ( fire\_event(INITIATE\_PAYMENT , 0 ,client\_fd) == 1)

{

printf(" the payment was sucessful");

fire\_event(PAYMENT\_SUCESSFUL , 0,client\_fd);

condition++;

}

else

{

printf(" the payment was unsucessful\n");

fire\_event(PAYMENT\_UNSUCESSFUL,0,client\_fd);

//condition = ;////special error case

}\*/

}

if (condition == 6)

{

printf("\n the cable will be unlocked now ");

fire\_event(CABLE\_UN\_LOCK , 0,client\_fd);

condition = 0;

printf("\n the charging process has been completed \n Thankyou please use me again\n\n\n\n");

printf("++++++++++++++++++++++++++++++++++++\n");

}

} //End of Inner While...

//Close Connection Socket

close(client\_fd);

} //Outer While

close(socket\_fd);

return 0;

}

//End of main

//unsigned char receivee(int client\_fd,unsigned char \*code, unsigned int \*value);

/\*

\* switchpi.h

\*

\* header file for switchpi.c

\*

\* Author : Jiztom Francis K

\* Created on : 01.03.2017

\* modified on : 08.03.2017

\*/

# include <stdio.h>

# include <netinet/in.h>

# include <sys/types.h>

# include <sys/socket.h>

#include "transfer.h"

#ifndef SWITCHPI\_H\_

#define SWITCHPI\_H\_

int fire\_event ( int event, int param , int socket\_fd);

unsigned char get\_state();

void init\_statemachine();

#endif /\*SWITCHPI\_H\_\*/

/\*

\*

\* switchpi.c

\*

\* the switch case function for the raspberry pi

\* Author : Jiztom Francis K

\* Created on : 01.03.2017

\* Modified on : 08.03.2017

\*/

#include "switchpi.h"

#define ST\_PLUG 1

#define ST\_SIGN\_IN 2

#define ST\_CHARGE 3

#define ST\_POST\_CHARGE 4

#define CABLE\_DETECTED 11

#define CABLE\_LOCK 12

#define PROTOCOL\_DETECT 13

#define LOCKING\_ERROR 14

#define REGISTER 21

#define AUTHORIZATION 22

#define START\_CHARGE 31

#define CHARGING\_STATUS 32

#define MANUAL\_STOP 33

#define FULL\_CHARGE 34

#define METER\_RECEIPT 36

#define INITIATE\_PAYMENT 41

#define PAYMENT\_SUCESSFUL 42

#define PAYMENT\_UNSUCESSFUL 43

#define CABLE\_UN\_LOCK 45

#define ISO15118\_DETECTED 51

#define IEC61851\_DETECTED 52

#define NO\_PROTOCOL\_DETECTED 53

#define SEMI\_ISO15118 61

#define MANUAL\_CHARGING 62

unsigned char state ;

void init\_statemachine()

{

state=ST\_PLUG;

}

unsigned char get\_state()

{

return state;

}

int fire\_event ( int event, int param , int socket\_fd)

{

int j =0;

char i;

unsigned char code;

unsigned int value;

switch(state)

{

case ST\_PLUG:

switch(event)

{

case CABLE\_DETECTED:

do

{

i = receivee(socket\_fd, &code , &value);

}while(code!= CABLE\_DETECTED);//(strcmp(i,CABLE\_DETECTED)!= 1);

printf("\nthe car has been detected ");

j= 1;

return j;

break;

case CABLE\_LOCK:

if(param == 1)

{

printf("\nthe cable has been locked");

j= 1;

}

else

{

printf("\nthere is a cable lock error");

j = 0;

}

return j;

break;

case PROTOCOL\_DETECT:

if(param == ISO15118\_DETECTED)

{

printf("\nProtocol ISO15118 detected.\n Car is not ready to charge");

state = ST\_SIGN\_IN;

}

else if ( param == IEC61851\_DETECTED)

{

state = SEMI\_ISO15118;

}

else if ( param == NO\_PROTOCOL\_DETECTED)

{

state = MANUAL\_CHARGING;

}

//state = ST\_SIGN\_IN;

break;

case LOCKING\_ERROR:

printf("\nRestarting the locking preocess");

break;

}

case ST\_SIGN\_IN:

switch(event)

{

case REGISTER:

printf("\ninitialize the payment procedure with the sign in details");

do

{

i=receivee(socket\_fd ,&code ,&value);

}while(code!=REGISTER);//(strcmp(i,REGISTER)!=1);

break;

case AUTHORIZATION:

printf(" \nAuthorize the charging station to start the charging process");

sendd(socket\_fd , AUTHORIZATION);

state = ST\_CHARGE;

break;

}

case ST\_CHARGE:

switch(event)

{

case START\_CHARGE:

printf("\nto start the charging process");

sendd(socket\_fd, START\_CHARGE);

break;

case CHARGING\_STATUS:

printf("\nthe charging status of the car with all necessary details");

do

{

i = receivee(socket\_fd , &code ,&value);

}while(code!= CHARGING\_STATUS);//(strcmp(i,CHARGING\_STATUS)!= 1);

break;

case FULL\_CHARGE:

printf("\nthe car has been fully charged ");

do

{

i = receivee(socket\_fd , &code , &value);

}while(code!= FULL\_CHARGE);//(strcmp(i,FULL\_CHARGE)!= 1);

break;

case MANUAL\_STOP:

printf("\nthe car charging should be stopped immediately");

sendd(socket\_fd , MANUAL\_STOP);

break;

case METER\_RECEIPT:

printf("\nthe receipt of the power and duration the vehicle has been charged");

do

{

i=receivee(socket\_fd ,&code,&value);

}while(code!= METER\_RECEIPT);//(strcmp(i,METER\_RECEIPT)!= 1);//maybe an internal process

state = ST\_POST\_CHARGE;

break;

}

case ST\_POST\_CHARGE:

switch(event)

{

case INITIATE\_PAYMENT:

printf(" \n the payment based on the meter receipt in initiated \n ");

do

{

i = receivee(socket\_fd , &code , &value);

}while(code!= INITIATE\_PAYMENT);//(strcmp(i,INITIATE\_PAYMENT) != 1);

printf(" \n the payment will be assesed for completion and will proceed to next stage \n");

break;

case PAYMENT\_SUCESSFUL:

printf(" \n The payment was sucessful and the sucessful display is displayed \n");

sendd(socket\_fd , PAYMENT\_SUCESSFUL);

break;

case PAYMENT\_UNSUCESSFUL:

printf(" if the payment is unsicessful please retry to pay");

///////////requires an internal code to replace pre used account details////////

printf("if failed thrice allow user to re enter their card details. Car will not be released until the payment is sucessful");

break;

case CABLE\_UN\_LOCK:

printf("\n the car is being unlocked");

do

{

i = receivee(socket\_fd , &code ,&value);

}while(code!= CABLE\_UN\_LOCK);//(strcmp(i,CABLE\_UN\_LOCK)!= 1);

printf("\ncharging has been completed");

break;

}

}

return j;

}

/\*

\* transfer.h for client

\*

\* Author : Jiztom Francis K

\* Created on : 18.02.2017

\* Modified on : 08.03.2017

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <netdb.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#ifndef TRANSFER\_H\_

#define TRANSFER\_H\_

#define PORT 3490

#define BACKLOG 10

char sendd(int sockfd , char data);

unsigned char receivee(int client\_fd,unsigned char \*code, unsigned int \*value);

int init\_tcp();

int accept\_client(int socket\_fd);

#endif /\* TRANSFER\_H\_\*/

/\*

\* transfer.c

\*

\* Author : Jiztom Francis K

\* Created on : 17.02.2017

\* Modified on : 08.03.2017

\*/

#include "transfer.h"

int status, socket\_fd, client\_fd,num;

struct sockaddr\_in server;

struct sockaddr\_in dest;

socklen\_t size;

int init\_tcp()

{

int yes =1;

if ((socket\_fd = socket(AF\_INET, SOCK\_STREAM, 0))== -1) {

fprintf(stderr, "Socket failure!!\n");

return(0);

}

if (setsockopt(socket\_fd, SOL\_SOCKET, SO\_REUSEADDR, &yes, sizeof(int)) == -1) {

perror("setsockopt");

return(0);

}

memset(&server, 0, sizeof(server));

memset(&dest,0,sizeof(dest));

server.sin\_family = AF\_INET;

server.sin\_port = htons(PORT);

server.sin\_addr.s\_addr = INADDR\_ANY;

if ((bind(socket\_fd, (struct sockaddr \*)&server, sizeof(struct sockaddr )))== -1)

{ //sizeof(struct sockaddr)

fprintf(stderr, "Binding Failure\n");

return(0);

}

if ((listen(socket\_fd, BACKLOG))== -1)

{

fprintf(stderr, "Listening Failure\n");

return(0);

}

return socket\_fd;

}

int accept\_client(int sfd)

{

if ((client\_fd = accept(socket\_fd, (struct sockaddr \*)&dest, &size))==-1 )

{

perror("accept");

return(-1);

}

else fprintf(stdout,"Accepted %s\n",inet\_ntoa(dest.sin\_addr));

return(client\_fd);

}

unsigned char receivee(int client\_fd,unsigned char \*code, unsigned int \*value)

{

int num;

static char buffer[20+1];

if ((num = recv(client\_fd, buffer, 1024,MSG\_DONTWAIT))== -1) {

}

else if (num == 0)

{

printf("Connection closed\n");

//So I can now wait for another client

return(0);

}

buffer[num] = '\0';

if (num > 0)

{ printf("Len: %d\n",num);

printf("Server:Msg Received %s\n", buffer);

}

\*code = buffer[0];

\*value= (unsigned int)buffer[1]<<8 | buffer[2];

return 1;

}

char sendd(int client\_fd , char data)

{

char data\_to\_send[1];

data\_to\_send[0]=data;

if ((send(client\_fd,data\_to\_send, strlen(data\_to\_send),0))== -1)

{

fprintf(stderr, "Failure Sending Message\n");

close(client\_fd);

return(0);

}

printf("Server:Msg being sent: :%d\n",data);

return(1);

}

Appendix - B

The main codes used in EVSE

/\*

\* EVSE\_main.h

\*

\* Created on : 30.06.2016

\* Author : Melanie

\* Modified by : Jiztom Francis K

\* Modified on : 08.03.2017

\*/

#ifndef EVSE\_MAIN\_H\_

#define EVSE\_MAIN\_H\_

# include <stdio.h>

# include <netinet/in.h>

# include "PWMSignal.h"

# include "interface.h"

# include "motors\_lock.h"

# include "PP\_Pin.h"

# include "serversockets.h"

# include "v2gEXIDatatypes.h"

# include "response.h"

# include "hardware.h"

# include "v2gtp.h"

# include <sys/types.h>

# include <sys/socket.h>

# include "ISO\_EVSE\_main.h"

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <netdb.h>

#include <arpa/inet.h>

#include "transfer.h"

# define PWM\_CTRL 1 // 0=disable --> EV-Side, 1=enable-->EVSE-Side, 2= query PWM-Signal

# define PWM\_DUTYCYCLE 250 //500 Hz of 1 kHz = 25% --> max current EVSE 15A

# define CODE\_EXAMPLE\_SOFTWARE 0

# define CODE\_EXAMPLE\_HARDWARE 1

# define CODE\_EXAMPLE CODE\_EXAMPLE\_HARDWARE

# define PORT\_NUMMER 5000

#define HMI\_READY 5

#endif /\* EVSE\_MAIN\_H\_ \*/

/\*

\* EVSE\_main.c

\*

\* Created on : 02.12.2015

\* Author : melanie

\* Edited by : Jiztom Francis K

\* Modified on : 08.03.2017

\*/

#include "EVSE\_main.h"

int main (void)

{

# if CODE\_EXAMPLE == CODE\_EXAMPLE\_HARDWARE

/\*init EVAchargeSE board \*/

int fd;

double UCP;

int round\_UCP;

fd=open\_serial();

//Init PWM Signal

control\_pwm(fd, PWM\_CTRL);

set\_pwm(fd, PWM\_DUTYCYCLE);

//get\_pwm(fd); // data EVSE --> 1kHz, Duty cycle depending on max current of the EVSE

# endif

int ev;

int i;

/////\* the variables for the TCP IP connection\*////////////////////////

init\_tcp();

/////\*start ISO 15118 protocol\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/////////////////

int new\_socket, server\_socket;

int laenge;

struct sockaddr\_in clientinfo;

struct v2gEXIDocument exiIn;

struct v2gEXIDocument exiOut;

int errn;

server\_socket = socket\_serverconnect(PORT\_NUMMER);

//pi\_server = pi\_connect(PORTPI\_NUMBER);

do

{

i=receivee(sockfd,&code,&value);

}while(code!=HMI\_READY);

//////the TCP IP establishment between raspberry pi and the EVSE //

////////////////////////////////////////////////////////////////////////

while(1)

{

printf("\n+++ Start protocol example ISO 15118 +++\n");

laenge = sizeof(clientinfo);

new\_socket = accept(server\_socket, (struct sockaddr \*) &clientinfo, (socklen\_t\*)&laenge);

/\*DIN Test --> optional\*/

/\*XMLDSIG Test --> optional\*/

# if CODE\_EXAMPLE == CODE\_EXAMPLE\_HARDWARE

UCP = get\_Ucp(fd) \* 10;

round\_UCP = UCP;

UCP = round\_UCP / 10;

printf("+++ check level on CP for state B: EV detected, %f +++\n\n", UCP);

sendd(sockfd , CABLE\_DETECTED);

////// --------------->>>>>>event CABLE\_DETECTED

#else

int fd =0;

double UCP = 9;

# endif

if (((UCP>8)||(UCP==8)) && ((UCP<10)||(UCP==10))){ //Level for State B

do{

errn = Lock\_Cable(fd);

} while (errn !=1);

printf("+++ release for charging: State B: vehicle detected +++\n");

printf("+++ Start application handshake protocol example +++\n\n");

ev = appHandshake(new\_socket);

if (ev == 0){

ev = STATE\_B1;

}

printf("+++ Terminate application handshake protocol example +++\n\n");

while (ev == STATE\_B1){

ev = Communication\_State\_B1(new\_socket, &exiIn, &exiOut);

}

# if CODE\_EXAMPLE == CODE\_EXAMPLE\_HARDWARE

UCP = get\_Ucp(fd) \* 10;

round\_UCP = UCP;

UCP = round\_UCP / 10;

printf("+++ check level on CP for state C: EV connected, ready, %f +++\n\n", UCP);

#else

double UCP = 6;

# endif

if (((UCP>5)||(UCP==5)) && ((UCP<7.5)||(UCP== 7.0)) && (ev == STATE\_C))////get\_Ucp -->6V state C Communication ToDo Pegel auf 7.1V-> Toleranz 6+-1V

{

printf("+++ Start Communication State C +++\n\n");

while (ev == STATE\_C){

ev = Communication\_State\_C(fd, new\_socket, &exiIn, &exiOut);

}

# if CODE\_EXAMPLE == CODE\_EXAMPLE\_HARDWARE

UCP = get\_Ucp(fd) \* 10;

round\_UCP = UCP;

UCP = round\_UCP / 10;

printf("+++ check level on CP for state B: EV detected, %f +++\n\n", UCP);

#else

double UCP = 9;

# endif

printf("+++ Start Communication State B +++\n\n");

if (((UCP>8)||(UCP==8)) && ((UCP<10)||(UCP==10)) && (ev == STATE\_B2)) ////get\_Ucp --> 9 V state B Communication

{

while (ev == STATE\_B2){

ev = Communication\_State\_B2(new\_socket, &exiIn, &exiOut);

}

}

}

}

if (ev == STATE\_ERROR)

{

printf(" Error during ISO 15118 Communication. \nPlease restart ");

}

printf("+++ End of example +++ \n");

ev = STATE\_A;

do{

errn = Unlock\_Cable(fd);

} while (errn !=1);

} //end while

socket\_close(server\_socket);

close(sockfd);

return 0;

} //end main

/\*

\* ISO\_main.h

\*

\* Created on : 03.01.2016

\* Author : Melanie

\* Modified by : Jiztom Francis K

\* Modified on : 08.03.2017

\*/

#ifndef ISO\_COMM\_ISO\_EVSE\_MAIN\_H\_

#define ISO\_COMM\_ISO\_EVSE\_MAIN\_H\_

# include "ErrorCodes.h"

# include "EXITypes.h"

# include "v2gEXIDatatypes.h"

# include "EVSE\_main.h"

# include "transfer.h"

/\*Define shold basic status of the Signal 10-15\*/

# define STATE\_A 10

# define STATE\_B1 11

# define STATE\_C 12

# define STATE\_B2 13

# define STATE\_ERROR 15

/\*Define charging state AC/DC \*/

# define AC\_CHARGING 21

# define DC\_CHARGING 22

/\*Define States in state machines for cases B1, C, B2 \*/

/\*State B1: 100-110\*/

# define STATE\_B1\_SUPPORTED\_APP\_PROTOCOL 100

# define STATE\_B1\_SESSION\_SETUP 101

# define STATE\_B1\_SERVICE\_DICOVERY 102

# define STATE\_B1\_SERVICE\_AND\_PAYMENT\_SELECTION 103

# define STATE\_B1\_PAYMENT\_DETAILS 104

# define STATE\_B1\_CONTRACT\_AUTHENTICATION 105

# define STATE\_B1\_CHARGE\_PARAMETER\_DISCOVERY 106

/\*State C: 111 - 120\*/

# define STATE\_C\_BEGIN\_POWER\_DELIVERY 111

# define STATE\_C\_AC\_CHARGING\_STATUS 112

//# define STATE\_C\_AC\_METERING\_RECEIPT 113

# define STATE\_C\_DC\_CABLE\_CHECK 114

# define STATE\_C\_DC\_PRE\_CHARGE 115

# define STATE\_C\_DC\_CURRENT\_DEMAND 116

# define STATE\_C\_END\_POWER\_DELIVERY 117

/\*State B2: 121 - 130\*/

# define STATE\_B2\_SESSION\_STOP 121

# define STATE\_B2\_DC\_WELDING\_DETECTION 122

#define CABLE\_DETECTED 11

#define CABLE\_LOCK 12

#define PROTOCOL\_DETECT 13

#define LOCKING\_ERROR 14

#define REGISTER 21

#define AUTHORIZATION 22

#define START\_CHARGE 31

#define CHARGING\_STATUS 32

#define MANUAL\_STOP 33

#define FULL\_CHARGE 34

#define METER\_RECEIPT 36

#define INITIATE\_PAYMENT 41

#define PAYMENT\_SUCESSFUL 42

#define PAYMENT\_UNSUCESSFUL 43

#define CABLE\_UN\_LOCK 45

#define ISO15118\_DETECTED 51

#define IEC61851\_DETECTED 52

#define NO\_PROTOCOL\_DETECTED 53

#define SEMI\_ISO15118 61

#define MANUAL\_CHARGING 62

/\*Start ISO Communication

\* @params: socket\_number from ethernet communikation\*/

int appHandshake(int socket\_number);

int Communication\_State\_B1(int socket\_number,struct v2gEXIDocument\* Input, struct v2gEXIDocument\* Output);

int Communication\_State\_B2(int socket\_number,struct v2gEXIDocument\* Input, struct v2gEXIDocument\* Output);

int Communication\_State\_C(int fd, int socket\_number,struct v2gEXIDocument\* Input, struct v2gEXIDocument\* Output);

#endif /\* ISO\_COMM\_ISO\_EVSE\_MAIN\_H\_ \*/

/\*

\* ISO\_main.c

\*

\* Created on : 03.01.2016

\* Author : Melanie

\* Modified by : Jiztom Francis K

\* Modified on : 08.03.2017

\*

\*/

# include <stdio.h>

# include <netinet/in.h>

#include "ISO\_EVSE\_main.h"

# include "appHandEXIDatatypes.h"

# include "v2gtp.h"

# include "Convert.h"

# include "appHandEXIDatatypesDecoder.h"

#include "appHandEXIDatatypesEncoder.h"

#include "serversockets.h"

# include "response.h"

# include "hardware.h"

# include <sys/types.h>

# include <sys/socket.h>

#include "transfer.h"

#define BUFFER\_SIZE 256

static int next\_state;

static int charging\_state;

static int basic\_state;

int appHandshake(int socket\_number) {

bitstream\_t iStream;

bitstream\_t oStream;

uint16\_t payloadLengthDec;

uint16\_t pos1 = V2GTP\_HEADER\_LENGTH; /\* v2gtp header \*/

uint16\_t pos2 = 0;

int errn, i;

uint8\_t buffer1[BUFFER\_SIZE];

uint8\_t buffer2[BUFFER\_SIZE];

iStream.size = BUFFER\_SIZE;

iStream.data = buffer1;

iStream.pos = &pos1;

oStream.size = BUFFER\_SIZE;

oStream.data = buffer2;

oStream.pos = &pos2;

struct appHandEXIDocument appHandResp;

struct appHandEXIDocument exiDoc;

errn = receive\_message(socket\_number, &iStream);

if ( (errn = read\_v2gtpHeader(iStream.data, &payloadLengthDec)) == 0) {

\*iStream.pos = V2GTP\_HEADER\_LENGTH;

if( (errn = decode\_appHandExiDocument(&iStream, &exiDoc)) ) {

/\* an error occured \*/

return errn;

}

}

printf("EVSE side: List of application handshake protocols of the EV\n");

for(i=0;i<exiDoc.supportedAppProtocolReq.AppProtocol.arrayLen;i++) {

printf("\tProtocol entry #=%d\n",(i+1));

printf("\t\tProtocolNamespace=");

printASCIIString(exiDoc.supportedAppProtocolReq.AppProtocol.array[i].ProtocolNamespace.characters, exiDoc.supportedAppProtocolReq.AppProtocol.array[i].ProtocolNamespace.charactersLen);

printf("\t\tVersion=%d.%d\n", exiDoc.supportedAppProtocolReq.AppProtocol.array[i].VersionNumberMajor, exiDoc.supportedAppProtocolReq.AppProtocol.array[i].VersionNumberMinor);

printf("\t\tSchemaID=%d\n", exiDoc.supportedAppProtocolReq.AppProtocol.array[i].SchemaID);

printf("\t\tPriority=%d\n", exiDoc.supportedAppProtocolReq.AppProtocol.array[i].Priority);

}

/\* prepare response handshake response:

\* it is assumed, we support the 15118 1.0 version :-) \*/

sendd( sockfd, ISO15118\_DETECTED);

///////--------->>>>> event PROTOCOL\_DETECTED 15118

appHandResp.supportedAppProtocolReq\_isUsed = 0u;

appHandResp.supportedAppProtocolRes\_isUsed = 1u;

appHandResp.supportedAppProtocolRes.ResponseCode = appHandresponseCodeType\_OK\_SuccessfulNegotiation;

appHandResp.supportedAppProtocolRes.SchemaID = exiDoc.supportedAppProtocolReq.AppProtocol.array[0].SchemaID; /\* signal the protocol by the provided schema id\*/

appHandResp.supportedAppProtocolRes.SchemaID\_isUsed = 1u;

\*oStream.pos = V2GTP\_HEADER\_LENGTH;

if( (errn = encode\_appHandExiDocument(&oStream, &appHandResp)) == 0) {

errn = write\_v2gtpHeader(oStream.data, (\*oStream.pos)-V2GTP\_HEADER\_LENGTH, V2GTP\_EXI\_TYPE);

printf("EVSE side: send response to the EV\n");

errn = transmit\_message(socket\_number, &oStream);

}

/\*init static state machine params\*/

next\_state = STATE\_B1\_SESSION\_SETUP;

charging\_state = 0;

return errn;

}

int Communication\_State\_B1(int socket\_number,struct v2gEXIDocument\* Input, struct v2gEXIDocument\* Output){

int errn = 0;

char parameter ;

errn = deserializeStream2EXI(Input, socket\_number);

if((errn == 0) && (Input->V2G\_Message\_isUsed)) {

init\_v2gEXIDocument(Output);

switch (next\_state){

case STATE\_B1\_SESSION\_SETUP:

if (Input->V2G\_Message.Body.SessionSetupReq\_isUsed) {

errn = sessionSetup(Input, Output);

next\_state = STATE\_B1\_SERVICE\_DICOVERY;

basic\_state = STATE\_B1;

}

break;

case STATE\_B1\_SERVICE\_DICOVERY:

if (Input->V2G\_Message.Body.ServiceDiscoveryReq\_isUsed) {

errn = serviceDiscovery(Input, Output);

next\_state = STATE\_B1\_SERVICE\_AND\_PAYMENT\_SELECTION;

basic\_state = STATE\_B1;

}

break;

case STATE\_B1\_SERVICE\_AND\_PAYMENT\_SELECTION:

if (Input->V2G\_Message.Body.ServiceDetailReq\_isUsed) {

errn = serviceDetail(Input, Output);

basic\_state = STATE\_B1;

} else if (Input->V2G\_Message.Body.PaymentServiceSelectionReq\_isUsed) {

errn = paymentServiceSelection(Input, Output);

next\_state = STATE\_B1\_PAYMENT\_DETAILS;

basic\_state = STATE\_B1;

sendd(sockfd , REGISTER);

/////------>>>>> event REGISTER

}

break;

case STATE\_B1\_PAYMENT\_DETAILS:

if (Input->V2G\_Message.Body.CertificateInstallationReq\_isUsed){

basic\_state = STATE\_B1;

//missing request

}else if (Input->V2G\_Message.Body.CertificateUpdateReq\_isUsed){

basic\_state = STATE\_B1;

//missing request

} else if (Input->V2G\_Message.Body.PaymentDetailsReq\_isUsed) {

errn = paymentDetails(Input, Output);

next\_state = STATE\_B1\_CONTRACT\_AUTHENTICATION;

basic\_state = STATE\_B1;

} break;

case STATE\_B1\_CONTRACT\_AUTHENTICATION:

if (Input->V2G\_Message.Body.AuthorizationReq\_isUsed) {

errn = authorization(Input, Output);

next\_state = STATE\_B1\_CHARGE\_PARAMETER\_DISCOVERY;

basic\_state = STATE\_B1;

do

{

parameter = receivee(sockfd , &code , &value);

}while(code!= AUTHORIZATION);

//(strcmp(parameter,AUTHORIZATION) != 1);

/////----->>>>> event AUTHORIZATION

}

break;

case STATE\_B1\_CHARGE\_PARAMETER\_DISCOVERY:

if (Input->V2G\_Message.Body.ChargeParameterDiscoveryReq\_isUsed) {

errn = chargeParameterDiscovery(Input, Output);

if(Input->V2G\_Message.Body.ChargeParameterDiscoveryReq.AC\_EVChargeParameter\_isUsed){

next\_state = STATE\_C\_BEGIN\_POWER\_DELIVERY;

charging\_state = AC\_CHARGING;

basic\_state = STATE\_C;

} else {

next\_state = STATE\_C\_DC\_CABLE\_CHECK;

charging\_state = DC\_CHARGING;

basic\_state = STATE\_C;

}

//(strcmp(parameter,START\_CHARGE) != 1) ;

/////----->>>>> event START\_CHARGE

}

break;

}

}

if (errn == 0){

errn = serializeEXI2Stream(Output, socket\_number);

}

if (errn == 0){

return basic\_state;

} else return STATE\_ERROR;

}

int Communication\_State\_C(int fd, int socket\_number,struct v2gEXIDocument\* Input, struct v2gEXIDocument\* Output){

int errn;

errn = deserializeStream2EXI(Input, socket\_number);

char parameter;

switch (next\_state){

case STATE\_C\_BEGIN\_POWER\_DELIVERY:

if (charging\_state == AC\_CHARGING){

/\*do

{

parameter = receivee(sockfd , &code ,&value);

}while(code!=START\_CHARGE);\*/

do{

errn = Close\_Contractors(fd);

} while (errn !=0);

errn = powerDelivery(Input, Output);

next\_state = STATE\_C\_AC\_CHARGING\_STATUS;

basic\_state = STATE\_C;

} else if (charging\_state == DC\_CHARGING){

if (Input->V2G\_Message.Body.PowerDeliveryReq\_isUsed){

do{

errn = Close\_Contractors(fd);

} while (errn !=0);

errn = powerDelivery(Input, Output);

next\_state = STATE\_C\_DC\_CURRENT\_DEMAND;

basic\_state = STATE\_C;

} else if (Input->V2G\_Message.Body.PreChargeReq\_isUsed){

errn = preCharge(Input, Output);

}

}

break;

case STATE\_C\_AC\_CHARGING\_STATUS:

if (Input->V2G\_Message.Body.ChargingStatusReq\_isUsed) {

errn = chargingStatus(Input, Output);

sendd(sockfd , CHARGING\_STATUS);

/////----->>>>> event CHARGING\_STATUS

next\_state = STATE\_C\_END\_POWER\_DELIVERY;

basic\_state = STATE\_C;

}

break;

case STATE\_C\_DC\_CABLE\_CHECK:

if (Input->V2G\_Message.Body.CableCheckReq\_isUsed){

errn = cableCheck(Input, Output);

next\_state = STATE\_C\_DC\_PRE\_CHARGE;

basic\_state = STATE\_C;

}

break;

case STATE\_C\_DC\_PRE\_CHARGE:

if (Input->V2G\_Message.Body.CableCheckReq\_isUsed){

errn = cableCheck(Input, Output);

} else if (Input->V2G\_Message.Body.PreChargeReq\_isUsed){

errn = preCharge(Input, Output);

next\_state = STATE\_C\_BEGIN\_POWER\_DELIVERY;

basic\_state = STATE\_C;

}

break;

case STATE\_C\_DC\_CURRENT\_DEMAND:

if(Input->V2G\_Message.Body.CurrentDemandReq\_isUsed){

errn = currentDemand(Input, Output);

next\_state = STATE\_C\_END\_POWER\_DELIVERY;

basic\_state = STATE\_C;

}

break;

case STATE\_C\_END\_POWER\_DELIVERY:

if (charging\_state == AC\_CHARGING){

if (Input->V2G\_Message.Body.ChargingStatusReq\_isUsed) {

errn = chargingStatus(Input, Output);

} else if (Input->V2G\_Message.Body.MeteringReceiptReq\_isUsed) {

errn = meteringReceipt(Input, Output);

sendd(sockfd , METER\_RECEIPT);

/////----->>>>> event METERING RECEIPT

} else if (Input->V2G\_Message.Body.PowerDeliveryReq\_isUsed){

do{

errn = Open\_Contractors(fd);

} while (errn !=0);

do

{

parameter = receivee(sockfd, &code, &value);

if(code == FULL\_CHARGE)

break;

////////full cahrge condition//////////

}while(code!=MANUAL\_STOP);//(strcmp(parameter, MANUAL\_STOP)!=1);

errn = powerDelivery(Input, Output);

next\_state = STATE\_B2\_SESSION\_STOP;

basic\_state = STATE\_B2;

}

} else if (charging\_state == DC\_CHARGING){

if(Input->V2G\_Message.Body.CurrentDemandReq\_isUsed){

errn = currentDemand(Input, Output);

} else if (Input->V2G\_Message.Body.PowerDeliveryReq\_isUsed){

do{

errn = Open\_Contractors(fd);

} while (errn !=0);

errn = powerDelivery(Input, Output);

next\_state = STATE\_B2\_DC\_WELDING\_DETECTION;

basic\_state = STATE\_B2;

}

}

break;

default: errn = STATE\_ERROR; break;

}

if (errn ==0){

errn = serializeEXI2Stream(Output, socket\_number);

}

if (errn ==0){

return basic\_state;

} else return errn;

}

int Communication\_State\_B2(int socket\_number,struct v2gEXIDocument\* Input, struct v2gEXIDocument\* Output){

int errn;

errn = deserializeStream2EXI(Input, socket\_number);

char parameter;

switch (next\_state){

case STATE\_B2\_SESSION\_STOP:

if (Input->V2G\_Message.Body.SessionStopReq\_isUsed) {

errn = sessionStop(Input, Output);

sendd(sockfd , INITIATE\_PAYMENT);/////----->>>>> event INTIATE\_PAYMENT

/////----->>>>> event PAYMENT\_SUCCESSFUL

/////----->>>>> event PAYMENT\_UNSUCCESSFUL

do

{

parameter = receivee(sockfd , &code , &value );

} while(code!=PAYMENT\_SUCESSFUL); //(strcmp(parameter,PAYMENT\_SUCESSFUL)!=1);

next\_state = 0;

basic\_state = STATE\_A;

}

break;

case STATE\_B2\_DC\_WELDING\_DETECTION:

if (Input->V2G\_Message.Body.WeldingDetectionReq\_isUsed) {

errn = weldingDetection(Input, Output);

next\_state = STATE\_B2\_SESSION\_STOP;

basic\_state = STATE\_B2;

}

break;

}

if (errn ==0){

errn = serializeEXI2Stream(Output, socket\_number);

}

if (errn ==0){

return basic\_state;

} else return errn;

}

/\*

\* Cable\_lock\_device.c

\*

\* Created on : 31.03.2016

\* Author : Melanie

\* Modified by : Jiztom Francis K

\* Modified on : 08.03.2017

\*/

# include "hardware.h"

# include <stdio.h>

# include "interface.h"

# include "EVSE\_main.h"

# include "transfer.h"

# include "ISO\_EVSE\_main.h"

#define UNLOCK 0

#define LOCK 1

#define CHECK 2

char data[6];

char result[6];

static void Change\_cable(int fd, int state){

data[0]=0x02;

data[1]=0x04;

data[2]=0x00;

data[3]=0x17;

data[4] = state; //query

data[5]=build\_checksum(data,5);

uart\_send\_data(fd,data,6);

read(fd,result,6);

}

int Lock\_Cable(int fd){

int errn = -1;

# if CODE\_EXAMPLE == CODE\_EXAMPLE\_HARDWARE

Change\_cable(fd, LOCK);

if (result[4] == LOCK){

if((result[0]==0x02)&& ((result[3] == 0x97)|(result[3] == 0x98)))

{

printf("Lock Status %x\n",(result[4]));

errn = 1;

}

sendd(sockfd , CABLE\_LOCK);

////////---------->>>>>>>event CABLE\_LOCK

}

# else

errn =1;

# endif

return (errn);

}

int Unlock\_Cable(int fd){

int errn =-1;

# if CODE\_EXAMPLE == CODE\_EXAMPLE\_HARDWARE

Change\_cable(fd, UNLOCK);

if (result[4] == UNLOCK){

if((result[0]==0x02)&& ((result[3] == 0x97)|(result[3] == 0x98)))

{

printf("Lock Status %x\n",(result[4]));

sendd(sockfd , CABLE\_UN\_LOCK);

errn = 1;

}

/////------------->>>>>>>>event CABLE\_UNLOCK

}

# else

errn =1;

# endif

return (errn);

}

/\*

\* transfer.h for EVSE

\*

\* Author : Jiztom Francis K

\* Created on : 18.02.2017

\* Modified on : 18.02.2017

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <netdb.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#ifndef TRANSFER\_H\_

#define TRANSFER\_H\_

# define PORT 3490

# define MAXSIZE 1024

char buff[1024];

char buffer[1024];

int sockfd;

unsigned char code;

unsigned int value;

void init\_tcp();

char sendd(int sockfd , char data);

unsigned char receivee(int client\_fd,unsigned char \*code, unsigned int \*value);

#endif /\* TRANSFER\_H\_\*/

/\*

\* transfer.c for EVSE

\*

\* Author : Jiztom Francis K

\* Created on : 17.02.2017

\* Modified on : 08.03.2017

\*/

#include "transfer.h"

void init\_tcp()

{

int num;

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

struct sockaddr\_in remoteaddr;

remoteaddr.sin\_family = AF\_INET;

remoteaddr.sin\_addr.s\_addr = inet\_addr("192.168.37.253");

remoteaddr.sin\_port = htons(PORT);

connect(sockfd, (struct sockaddr \*)&remoteaddr, sizeof(remoteaddr));

}

unsigned char receivee(int client\_fd,unsigned char \*code, unsigned int \*value)

{

int num;

static char buffer[20+1];

if ((num = recv(client\_fd, buffer, 1024,MSG\_DONTWAIT))== -1) {

printf("error");

}

else if (num == 0)

{

printf("Connection closed\n");

////So I can now wait for another client

return(0);

}

buffer[num] = '\0';

if (num > 0)

{ printf("Len: %d\n",num);

printf("Server:Msg Received %d\n", buffer[0]);

}

\*code = buffer[0];

\*value= (unsigned int)buffer[1]<<8 | buffer[2];

return 1;

}

char sendd(int client\_fd , char data)

{

char data\_to\_send[1];

data\_to\_send[0]=data;

if ((send(client\_fd,data\_to\_send, strlen(data\_to\_send),0))== -1)

{

fprintf(stderr, "Failure Sending Message\n");

close(client\_fd);

return(0);

}

printf("Server:Msg being sent: :%d\n",data);

return(1);

}

**Appendix - C**

The output from the successful implementation of the codes.

**Raspberry pi**

Server got connection from client 192.168.37.251

Please choose the language to be selected

1. English 2. German

your option please :

1

Server:Msg being sent: :5

please insert the plug into the system

Len: 1

Server:Msg Received

the car has been detected

the cable has been connected and the car has been detected

the cable has been locked

the vehicle status is :

the protocol has been detected . Now initiating the information and account details process

initialize the payment procedure with the sign in detailsLen: 1

Server:Msg Received

Len: 1

Server:Msg Received 3

Len: 1

Server:Msg Received

the payment and the initial requirement has been done

Authorize the charging station to start the charging processServer:Msg being sent: :22

the car is ready for charging.

please press the button to charge the vehicle

the charging status of the car with all necessary detailsLen: 1

Server:Msg Received

the car charging should be stopped immediatelyServer:Msg being sent: :33

the car has stopped charging

the payment details are as follows:

the receipt of the power and duration the vehicle has been chargedLen: 1

Server:Msg Received $

The payment will be processed now

the payment based on the meter receipt in initiated

Len: 10

Server:Msg Received )k���\*��8�

the payment will be assesed for completion and will proceed to next stage

The payment was sucessful and the sucessful display is displayed

Server:Msg being sent: :42

the cable will be unlocked now

the car is being unlockedLen: 1

Server:Msg Received -

charging has been completed

the charging process has been completed

Thankyou please use me again

++++++++++++++++++++++++++++++++++++

Please choose the language to be selected

1. English 2. German

your option please :

**EVSE**

**EV**

+++ level on CP: standby, 1900.196000 +++

Verbindung nicht hergestellt root@EVAChargeSE:~/EV# ./EV

+++ level on CP: standby, 11.890000 +++

+++ change level on CP for state B: EV detected, 11.890000 +++

+++ Start application handshake protocol example +++

EV side: setup data for the supported application handshake request message

EV side: send message to the EVSE

send stream... ...sent

receive stream......received stream

EV side: received message from the EVSE

EV side: Response of the EVSE

ResponseCode=OK\_SuccessfulNegotiation

SchemaID=1

+++ Terminate application handshake protocol example +++

choose between AC-Charging[1], DC-Chargiung[2] or exit [3]: 1

+++ Start V2G client / service example for AC charging +++

EV side: call EVSE sessionSetup

send stream... ...sent

receive stream......received stream

EV side: received response message from EVSE

Header SessionID=1 2 3 4 5 6 7 8

ResponseCode=0

EVSEID=20

EVSETimeStamp=123456789

EV side: call EVSE serviceDiscovery

send stream... ...sent

receive stream......received stream

EV side: received response message from EVSE

Header SessionID=1 2 3 4 5 6 7 8

ResponseCode=0

ServiceID=1

ServiceName=AC\_DC

PaymentOption=Contract\_paymentOptionType

ChargeService.FreeService=True

EnergyTransferMode=AC\_single\_DC\_core

EnergyTransferMode=AC\_single\_phase\_core\_EnergyTransferModeType

Value added service list:

ServiceID=22

ServiceName=WWW

ServiceCategory=Internet

FreeService=True

ServiceID=33

ServiceName=HTTPS

ServiceCategory=Internet

EV side: call EVSE ServiceDetail

send stream... ...sent

receive stream......received stream

EV side: received response message from EVSE

Header SessionID=1 2 3 4 5 6 7 8

ResponseCode=0

ServiceID=1234

Length=2

ServiceSetID=1

Parameters=2

1: ParameterName=Protocol

1: IntValue=15119

2: ParameterName=Name

ServiceSetID=2

Parameters=1

1: ParameterName=Channel

1: PhysicalValue=1234 (0)

EV side: call EVSE ServicePaymentSelection

send stream... ...sent

receive stream......received stream

EV side: received response message from EVSE

Header SessionID=1 2 3 4 5 6 7 8

ResponseCode=0

EV side: call EVSE PaymentDetails

send stream... ...sent

receive stream......received stream

EV side: received response message from EVSE

Header SessionID=1 2 3 4 5 6 7 8

ResponseCode=0

EVSETimeStamp=123456

GenChallenge=1

EV side: call EVSE Authorization

send stream... ...sent

receive stream......received stream

EV side: received response message from EVSE

Header SessionID=1 2 3 4 5 6 7 8

ResponseCode=0

EVSEProcessing=Finished

+++ change level on CP for state C: EV connected, ready, 6.960000 +++

EV side: call EVSE chargeParameterDiscovery

send stream... ...sent

receive stream......received stream

EV side: received response message from EVSE

Header SessionID=1 2 3 4 5 6 7 8

ResponseCode=0

EVSEStatus:

RCD=1

EVSENotification=0

NotificationMaxDelay=123

EVSEProcessing=1

EVSEMaxCurrent=100

EVSENominalVoltage=300

EV side: call EVSE powerDelivery

send stream... ...sent

receive stream......received stream

EV side: received response message from EVSE

Header SessionID=1 2 3 4 5 6 7 8

ResponseCode=0

EVSEStatus:

RCD=0

EVSENotification=3

NotificationMaxDelay=12

EV side: call EVSE chargingStatus

send stream... ...sent

receive stream......received stream

EV side: received response message from EVSE

Header SessionID=1 2 3 4 5 6 7 8

ResponseCode=0

EVSEStatus:

RCD=1

EVSENotification=0

NotificationMaxDelay=123

ReceiptRequired=1

EVSEID=12

SAScheduleTupleID=10

EVSEMaxCurrent=400 (3 2)

isused.MeterInfo=1

MeterInfo.MeterID=2

MeterInfo.MeterReading.Value=5000

MeterInfo.MeterStatus=4321

MeterInfo.TMeter=123456789

MeterInfo.SigMeterReading.data=123

EV side: call EVSE meteringReceipt

send stream... ...sent

receive stream......received stream

EV side: received response message from EVSE

Header SessionID=1 2 3 4 5 6 7 8

ResponseCode=0

EV side: call EVSE powerDelivery

send stream... ...sent

receive stream......received stream

EV side: received response message from EVSE

Header SessionID=1 2 3 4 5 6 7 8

ResponseCode=0

EVSEStatus:

RCD=0

EVSENotification=3

NotificationMaxDelay=12

+++ change level on CP for state B: EV detected, 8.816000 +++

EV side: call EVSE stopSession

send stream... ...sent

receive stream......received stream

EV side: received response message from EVSE

Header SessionID=1 2 3 4 5 6 7 8

ResponseCode=0

+++Terminate V2G Client / Service example for AC charging +++

**Appendix - D**

The EB GUIDE Screen shots:

Bibliography