CHARGING STATION FOR ISO / IEC 15118 PROTOCOL

A PROJECT REPORT

Submitted by

JIZTOM FRANICS K - 311113106022

NIVAS GOKUL M - 311113106040

In partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

LOYOLA – ICAM COLLEGE OF ENGINEERING AND TECHNOLOGY, LOYOLA CAMPUS, NUNGAMBAKKAM, CHENNAI - 600034

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2017

BONAFIDE CERTIFICATE

Certified that this project report **"CHARGING STATION FOR ISO / IEC 15118 PROTOCOL"** is the bonafide work of "**JIZTOM FRANICS K (311113106022) and NIVAS GOKUL M (311113106040)"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Prof. Dr. S. Saraswathi Janaki	Mr I. Suman Maria Tony
HEAD OF THE DEPARTMENT	**ASST. PROFESSOR/ PROJECT GUIDE**
Dept. Of Electronics and Communication Engineering	Dept. Of Electronics and Communication Engineering
Loyola-ICAM College Of Engineering And Technology, Nungambakkam, Chennai – 600 034	Loyola-ICAM College Of Engineering And Technology Nungambakkam, Chennai – 600 034

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We wish to express our sincere thanks to our beloved Director **Rev. Dr. Ignacy Arockyaa SJ** for providing a great opportunity to do our final year project at Hochschule Heilbronn (University of Heilbronn), Germany.

We proudly render our heartfelt thanks to our Principal **Dr. Jose Swaminathan** for the constant encouragement and support given by him for the progress and completion of our project.

We would also like to express our deep sense of gratitude to our Dean of studies **Rev. Dr John Pragasam SJ**, and the Controller of Examination **Dr. D. Caleb Chanthi Raj** for having given their motivation in completing this project work.

Our immense thanks is also due to the Head of the Department **Prof. Dr. S Saraswathi** Janaki for her effective leadership, encouragement and guidance in the project.

We are deeply indebted and wish to extend our sincere thanks to our Project mentor, **Prof. Dr.-Ing Ansgar Meroth** from Heilbronn University, Germany for his valuable ideas accompanied by moral support throughout the conduct of the project.

We are very much thankful to our guide **Mr. I. Suman Maria Tony** for his valuable ideas which makes us to finish our work in a most successful manner.

We duly acknowledge the help extended by **Mr. Robert Rajkumar**, the project co-ordinator for his timely guidance in the documentation and presentations related to the project work.

We wish to extend our sincere thanks to all Faculty members of the Department of Electronics and Communication Engineering for their valuable suggestions and their kind co-operation for the successful completion of our project.

We also wish to acknowledge the help of **Mr. Raphael Scholz** from Heilbronn University, Germany who has helped us visualize the working of the concepts of the project.

ABSTRACT

The Project work focuses mainly on the need of ISO /IEC 15118 and gives clear testimony of why it was standardized and the use of the protocol in the Electric Car Charging Station. It defines the protocols and all the related technologies (hardware and software) required to run a working third generation Car Charging Station. The existing car charger station running on IEC 61851 does not allow for user to pay for the charge used and is being given as a free option to existing Electric car users at public places. The new protocol improves this situation by allowing the user to set the parameters and also allowing the car to communicate with the charging station via powerline communication thereby allowing better charging efficiency. The main task was to design a working HMI based on the requirements as is mentioned and the process of designing and implementing the same in the working prototype model by way of embedded hardware and Linux/C language integration of the same which is done most successfully for charging, billing and power grid management to safeguard against the blackout of the electric grids due to overdraft of available power etc.. The project in overall has successfully designed the HMI and the backend process to communicate the Raspberry pi with the Electric Vehicle Supply Equipment (EVSE) and synchronize all the three systems (Raspberry Pi, EVSE and EV).This forms as the solid foundation for any next levels of this main project already implemented. The state Diagram developed along with the UML diagrams help in further visualizing the process.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	BONAFIDE CERTIFICATE	**i**
	ACKNOWLEDGEMENT	**ii**
	ABSTRACT	**iii**
	TABLE OF CONTENTS	**iv**
	LIST OF TABLES	**vii**
	LIST OF FIGURES	**viii**
	LIST OF ABBREVATIONS	**ix**
1.	**INTRODUCTION**	**1**
1.1	SHORT VERSION	1
1.2	TASK	2
1.3	CHAPTER OVERVIEW	3
2.	**LITERATURE SURVERY**	**4**
3.	**PROBLEM STATEMENT / OBJECTIVE**	**6**
3.1	INTRODUCTION	6
3.2	OBJECTIVE	7
4.	**METHODOLOGY**	**8**
4.1	IEC 62196: VEHICLE PLUG	8
4.2	IEC 61851	10
4.3	ISO 15118	12
4.4	TCP/IP COMMUNICATION	14
4.5	UML	16
4.5.1	INTRODUCTION	16
4.5.1.1	UML 2.0	16
4.5.1.2	TYPES OF UML DIAGRAMS	16
4.5.2	STATE MACHINE DIAGRAM	17
4.5.2.1	DIFFERENCE BETWEEN A STATE DIAGRAM AND A FLOW CHART	18

4.5.2.2	STEPS TO DRAWING A STATE DIAGRAM	18
4.5.3	SEQUENCE DIAGRAM	19
4.5.3.1	HOW TO USE SEQUENCE DIAGRAMS	19
4.6	AUTOMATIC STATE DIAGRAM	20
4.6.1	INTRODUCTION	20
4.6.2	STATE DIAGRAM FOR THE ISO 15118	22
5.	**SOFTWARE DESCRIPTION**	**23**
5.1	FileZilla	23
5.2	GEANY	25
5.3	PuTTY TERMINAL	26
5.4	EB GUIDE	27
5.4.1	BENEFITS	27
5.4.2	FEATURES	28
5.5	www.draw.io	29
5.6	VISUAL PARADIGM 14.0	30
5.6.1	INTRODUCTION	30
5.6.2	BENEFITS	30
5.7	COMPILING THE PROGRAM	31
6.	**HARDWARE DESCRIPTION**	**33**
6.1	RASPBERRY PI 3	33
6.1.1	INTRODUCTION	33
6.1.2	RASPBERRY PI 7” OFFICIAL TOUCH SCREEN	34
6.2	EVACharge SE BOARD	35
6.2.1	INTRODUCTION	35
6.2.2	SPECIFICATIONS	36
6.2.3	APPLICATIONS	36
6.3	COMMUNICATION BETWEEN EVSE AND EV	37
6.4	COMMUNICATION BETWEEN EVSE AND RASPBERRY PI	38
6.5	INTEGRATING THE SYSTEM	39
7.	**RESULTS**	**40**

8.		**CONCLUSION**	**41**
9.		**BIBLIOGRAPHY**	**42**
		APPENDIX	**43**
		APPENDIX 1	43
	a.	UML SEQUENCE DAIGRAM	43
	b.	UML STATE DIAGRAM	44
	c.	THE STATE DIAGRAM FOR THE EVSE SIDE	45
	d.	THE STATE DIAGRAM FOR THE RASPBERRY SIDE	46
		APPENDIX 2	47
		APPENDIX 3	63
		APPENDIX 4	81
	a.	Raspberry pi	81
	b.	EVSE output	83
	c.	EV output	87
		APPENDIX 5	90

LIST OF TABLES

TABLE NO.	TITLE	PAGE NO.
Table 4.1	PIN details of IEC 62196	9
Table 4.2	EC 61851 STATES EXPLAINED	11
Table 5.1	GENERAL NUTSHELL COMMANDS	32
Table 6.1	SPECIFICATION OF EVACharge SE BOARD	36

LIST OF FIGURES

FIG. NO.	TITLE	PAGE NO.
Figure 4.1	CAR CHARGING PLUG MODELS	8
Figure 4.2	VOLTAGE DIVIDER STATES	10
Figure 4.3	ISO 15118 STATES WORING DETAILS	12
Figure.4.4	OPENV2G STATE DIAGRAM	13
Figure 4.5	TCP/IP DATA FLOW DIAGRAM	14
Figure 4.6	LAYERS OF TCP/IP	15
Figure 4.7	UML STATE MACHINE DIAGRAM	17
Figure 4.8	EXAMPLE OF CREATING A STATE DIAGRAM	18
Figure 4.9	UML SEQUENCE DIAGRAM	19
Figure 4.10	COFFEE MACHINE AUTOMATIC STATE DIAGRAM	20
Figure 4.11	CASE EXAMPLE OF COFFEE MACHINE	21
Figure 4.12	BASIC STATE DIAGRAM FOR ISO 15118	22
Figure 5.1	FileZilla INTERFACE	24
Figure 5.2	Geany INTERFACE	25
Figure 5.3	PuTTY INTERFACE	26
Figure 5.4	EB GUIDE 6.3 INTERFACE	28
Figure 5.5	www.draw.io INTERFACE	29
Figure 5.6	VISUAL PARADIGM INTERFACE	30
Figure 5.7	LOGIN SHELL FOR EVAChargeSE USING SSH	31
Figure 5.8	COMPILING THE PROGRAM ON GCC	32
Figure 6.1	RASPBERRY PI 3 MODEL B SPECIFICATION	33
Figure 6.2	OFFICIAL 7"TOUCH SCREEN	34
Figure 6.3	PARTS OF THE TOUCH SCREEN WITH DRIVER	34
Figure 6.4	EVACharge SE BOARD	35
Figure 7.1	THE WORKING MODEL WITH THE SIGNAL ON EV	40
Appendix 1.1	UML SEQUENCE DIAGRAM	43
Appendix 1.2	UML STATE DIAGRAM	44
Appendix 1.3	STATE DIAGRAM -EVSE	45
Appendix 1.4	STATE DIAGRAM -RASPBERRY PI	46

LIST OF ABBREVATIONS

CPLT	Control Pilot Line
EV	Electric Vehicle
EVCC	Electric Vehicle Communication Controller
EVSE	Electric Vehicle Supply Equipment
HMI	Human Machine Interface
HPGP	HomePlug Green Phy
IP	Internet Protocol
OSI	Open Systems Interconnect
RPi	Raspberry Pi
SECC	Supply Equipment Communication Controller
TCP	Transmission Control protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
V2G	Vehicle-to-Grid
V2GTP	Vehicle-to-Grid Transfer Protocol
V2IoG	Vehicle-to-Grid Infrastructure over Grid

CHAPTER I

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.

1.1 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.

1.2 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:

- Analyse 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.

1.3 CHAPTER OVERVIEW

The first chapter gives the basic introduction of the project and its contents and a better brief.

The second chapter deals with the Literature survey with all the papers taken into consideration to get idea of implementing the project. It gives clear cut ideas on improving the system thereby allowing reduction of possible problems.

The third chapter deals with the problem statement and it clearly introduces why this protocol was brought into existence and also clearly define the task that was to be done on the project.

The fourth chapter deals with the methodologies and other technologies incorporated into the project. It clearly defines the protocol and gives brief explanation on what it is and how it works.

The fifth chapter deals with the software components used in the project. This explains all the software used and the reason why it was used along with a description of its interface and how it works.

The sixth chapter deals with the hardware components used in the project and also the way they communicate between each other i.e. the hardware interfacing.

The seventh chapter deals with the Result of the project followed by the Conclusion in the eighth chapter. The ninth chapter tells all the references used while writing the report.

The annexure allows us to show the output produced as form of state diagram, C code outputs and HMI display screen shots.

CHAPTER II

2. LITERATURE SURVERY

[1] Peter Hank, Steffen Müller, Ovidiu Vermesan, Jeroen Van Den Keybus, ‘Automotive Ethernet: in-vehicle networking and smart mobility’, DATE '13 Proceedings of the Conference on Design, Automation and Test in Europe Pages 1735-1739

Inference:

This paper explain the possibilities opened by using Ethernet for connecting the automobile network with any charging setup. It explicitly tells the use of Green PHY the powerline communication to improve the communication process and also reduce additional requirements for inter automotive communication. This was used in designing the communication process between the vehicle and the electric car charger.

[2] B O Chen , Keith S. Hardy, Jason D. Harper, Daniel S. Dobrzynski , ‘ Towards standardized Vehicle Grid Integration’ , Transportation Electrification Conference and Expo (ITEC) ,2015 ,IEE.

Inference:

This paper speaks about the standards needed to be considered while the vehicle and the grid will communicate. This is one of the major part of the V2G project and looks at integrating the smart grid to the Vehicle charging. This forms the basis of the entire ISO 15118 protocol.

[3] Dr. Andreas Heinrich, Michael Schwaiger, Daimler Ag and BMW group , ‘ISO 15118 – charging communication between plug-in electric vehicles and charging infrastructure’, Grid Integration of Electric Mobility, 1st international ATZ Conference 2016 , pp 213-227

Inference:

The paper presented as collaboration between Daimler AG and BMW group discusses on the point of how vehicle to be communicated to the grid and why ISO 15118 is very essential. It speaks about the potential issues the increased no of electrical cars can bring and how the protocol will help to tackle this issue.

[4] M. Sc. Michael Tybel, Dr. –Ing Andrey Popov, Dr. –Ing Michael Schugt , Scienlab electronic systems, Bochum, ‘Assuring Interoperability between Conductive EV and EVSE Charging Systems’, Popov.com.

Inference:

This paper deals with the appHandshake assurance between the EV and EVSE and make sure that the proper powerline communication takes place. It focuses on the idea of socket communication with data loss prevention techniques to have a 100% successful communication of the required data between the EV and EVSE.

[5] D. Wellisch, J Lenz, A Faschingbauer, R. Posch, S. Kunze , Deggendorf Institute of Technology, Freyung. Research gate Publication, ‘ Vehicle-to-grid AC charging Station’ An Approach for Smart Charging Development.’

Inference:

This paper is the logical case study of the project being implemented. It spoke about the possibilities of the V2GTP and the possible states that need to be considered to obtain a good model. This is one of the bases used in designing the Electric car charging station.

CHAPTER III

3. PROBLEM STATEMENT / OBJECTIVE

3.1 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.

3.2 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:

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

CHAPTER IV

4. 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".

4.1 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.

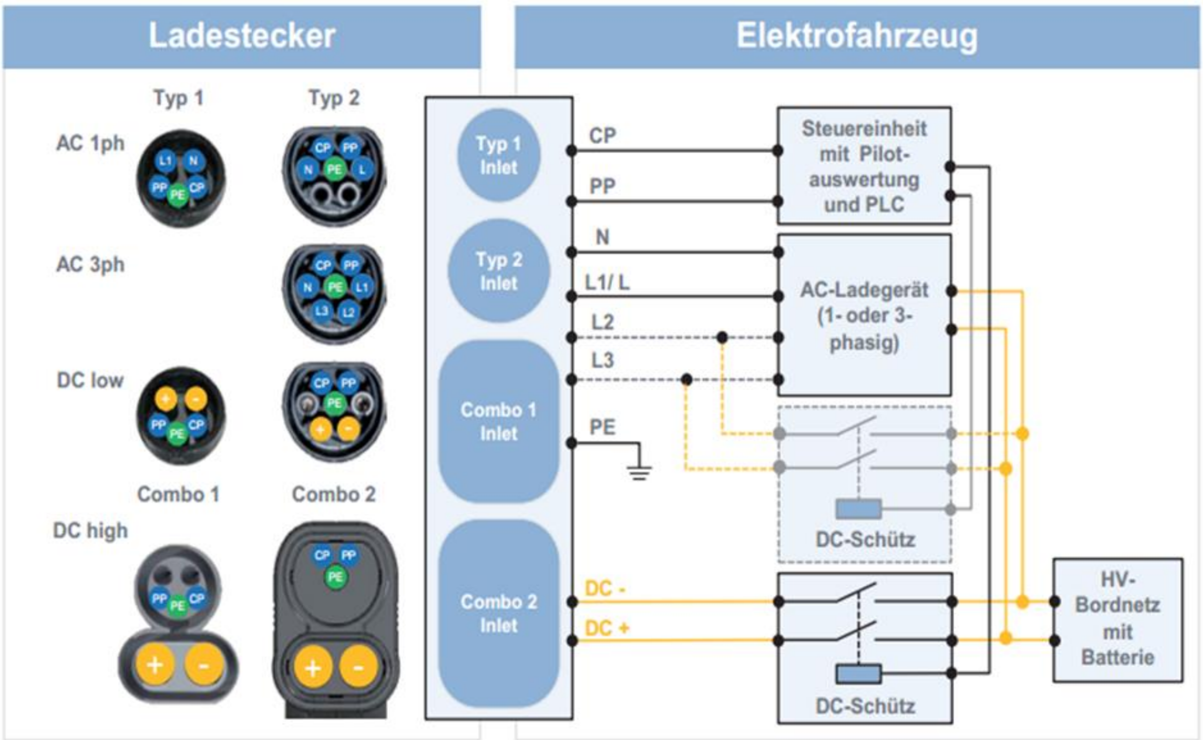


Figure 4.1 CAR CHARGING PLUG MODELS

The Type 1 charging plug, which in Figure 4.1 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 4.1. 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.

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

Table 4.1 PIN details of IEC 62196

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.

4.2 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.

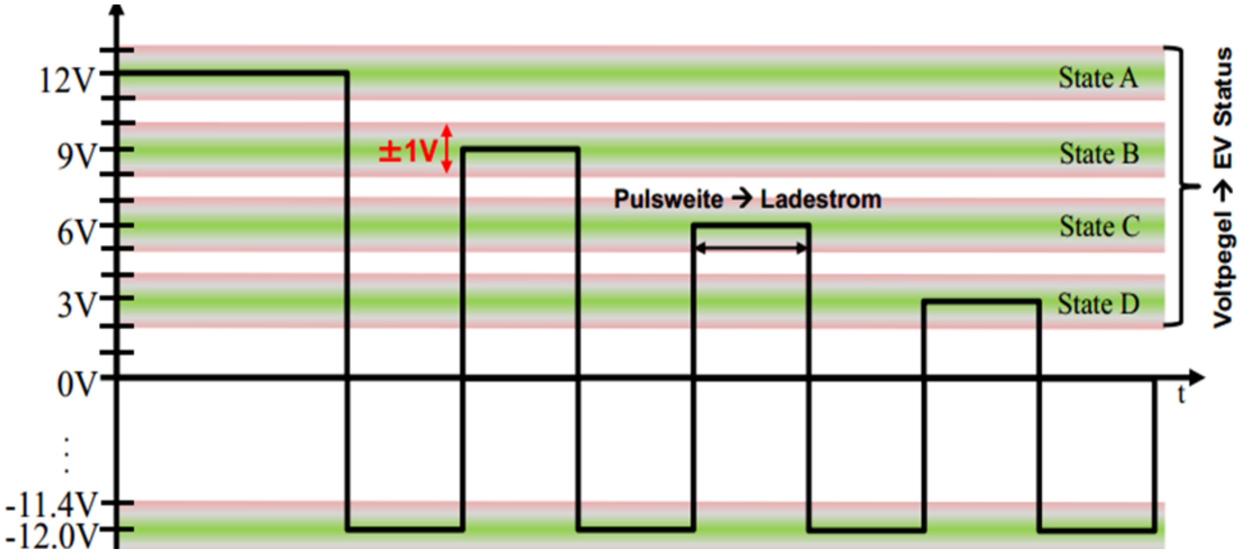


Figure 4.2 VOLTAGE DIVIDER STATES

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 4.1 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 4.2. Please note that the negative voltage value is permanently -12V, and only the positive values change. A definition of the individual states is Table 4.2.

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

Table 4.2 EC 61851 STATES EXPLAINED

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.

4.3 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 4.3 assigned

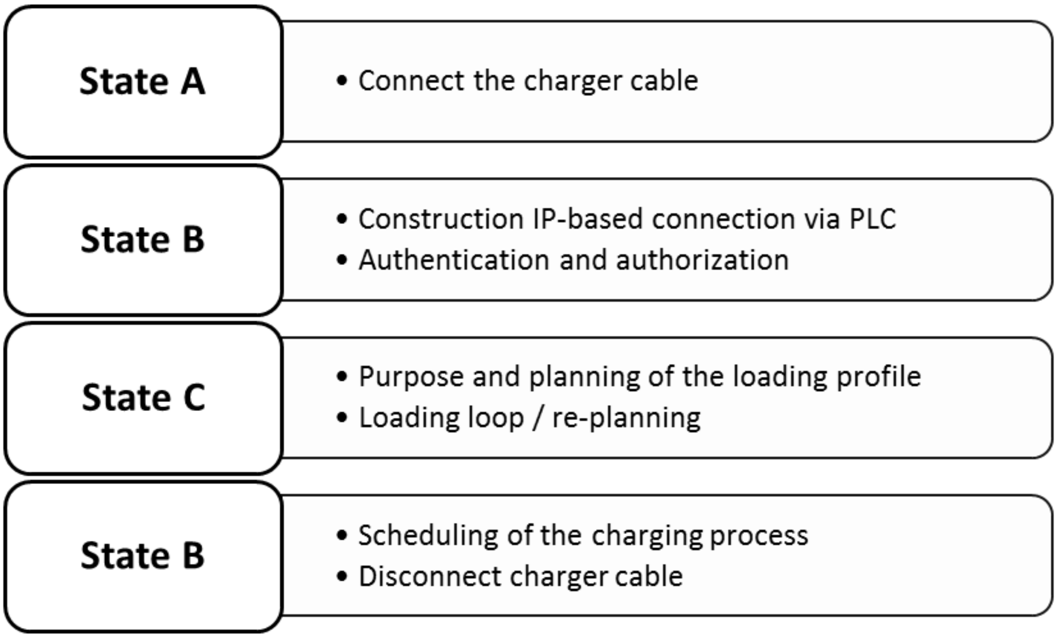


Figure 4.3 ISO 15118 STATES WORING DETAILS

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)

4.3.1 OPENV2G 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.

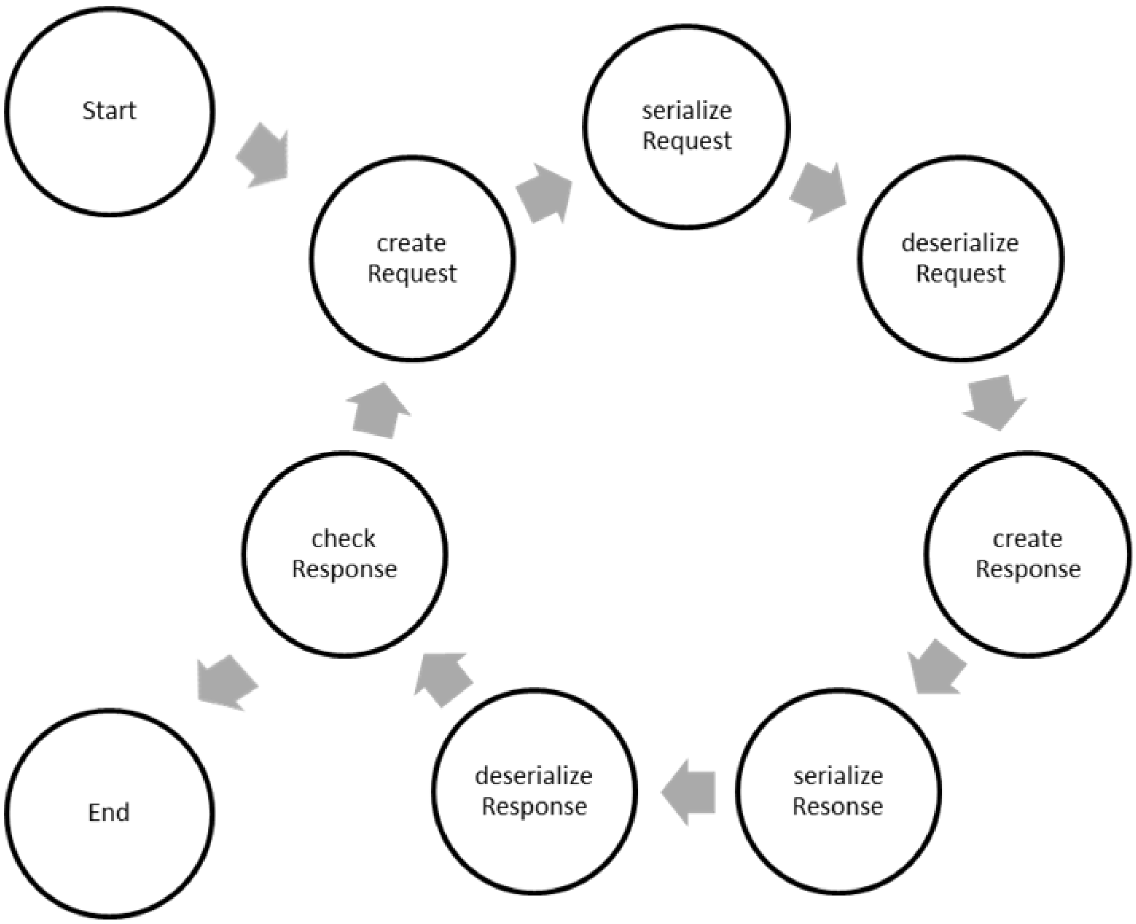


Figure 4.4 OPENV2G STATE DIAGRAM

4.4 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.

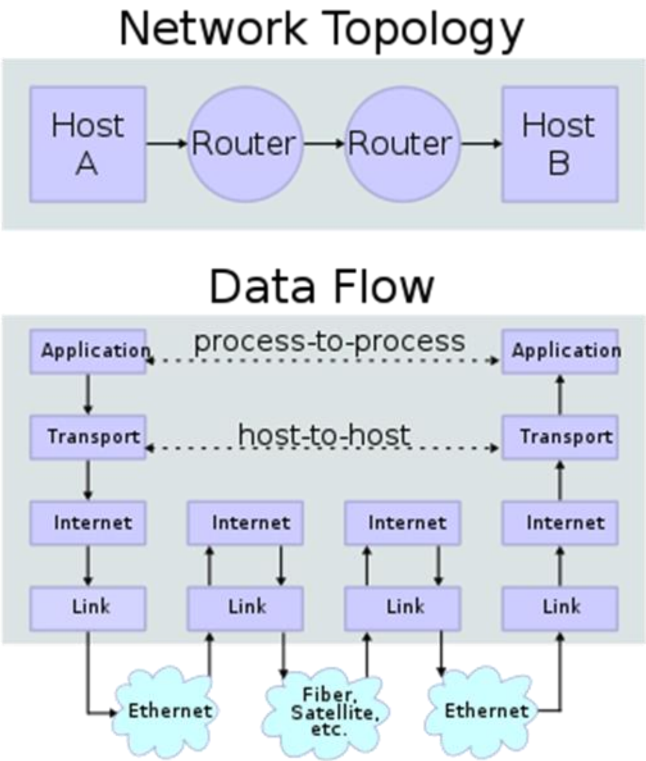


Figure 4.5 TCP/IP DATA FLOW DIAGRAM

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.

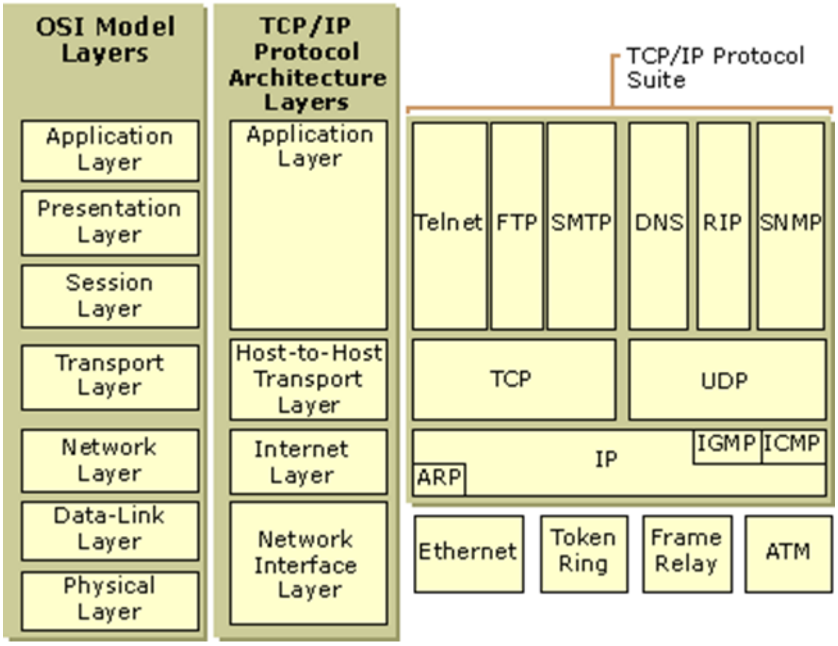


Figure 4.6 LAYERS OF TCP/IP

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.

4.5 UML

4.5.1 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.

4.5.1.1 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.

4.5.1.2 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.

a. Structural UML diagrams
   - Class diagram
   - Package diagram
   - Object diagram
   - Component diagram
   - Composite structure diagram

- Deployment diagram

b. 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.

4.5.2 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.

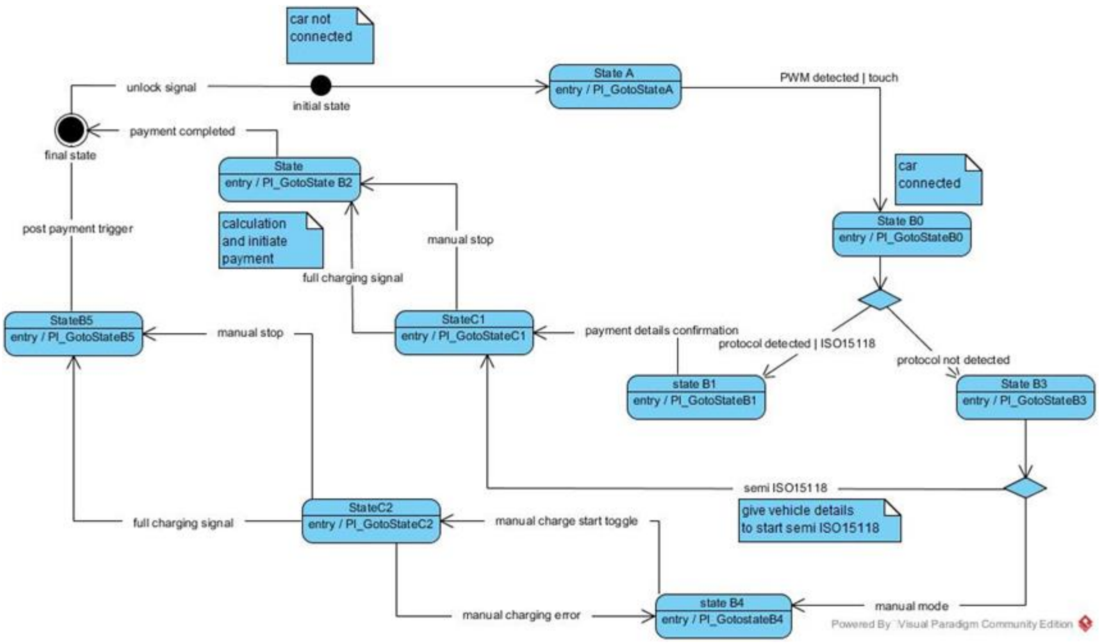


Figure 4.7 UML STATE MACHINE DIAGRAM

4.5.2.1 DIFFERENCE BETWEEN A STATE DIAGRAM AND A FLOW CHART

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.

4.5.2.2 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.

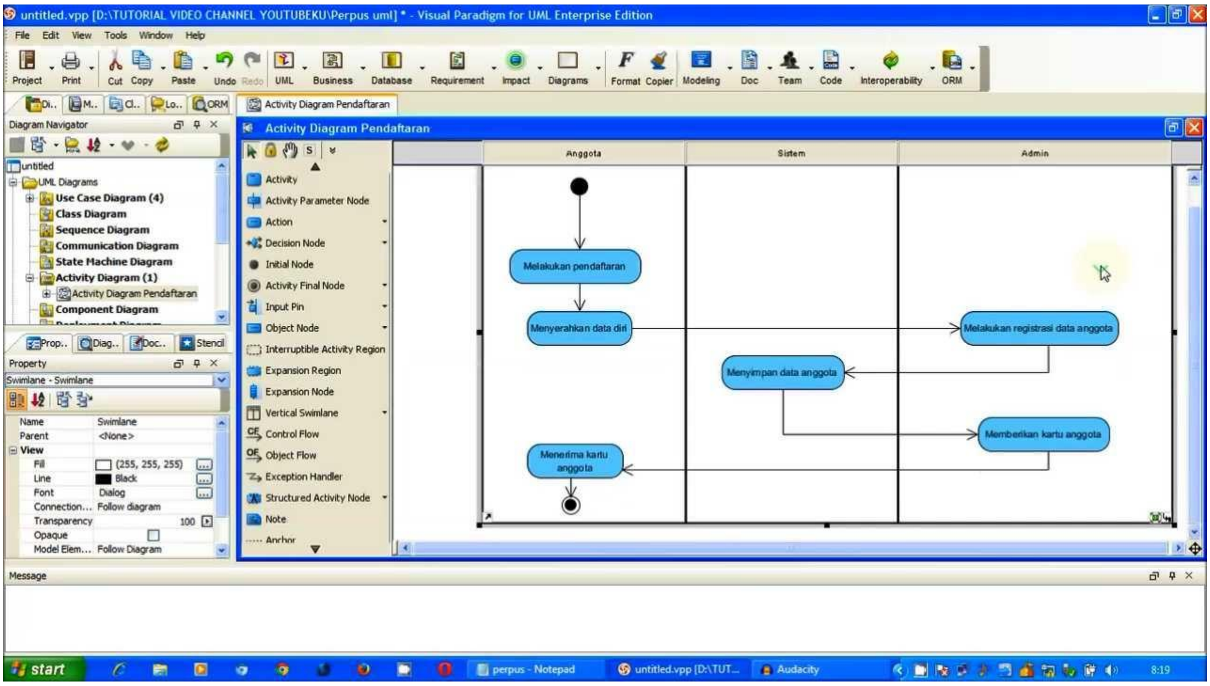


Figure 4.8 EXAMPLE OF CREATING A STATE DIAGRAM

4.5.3 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

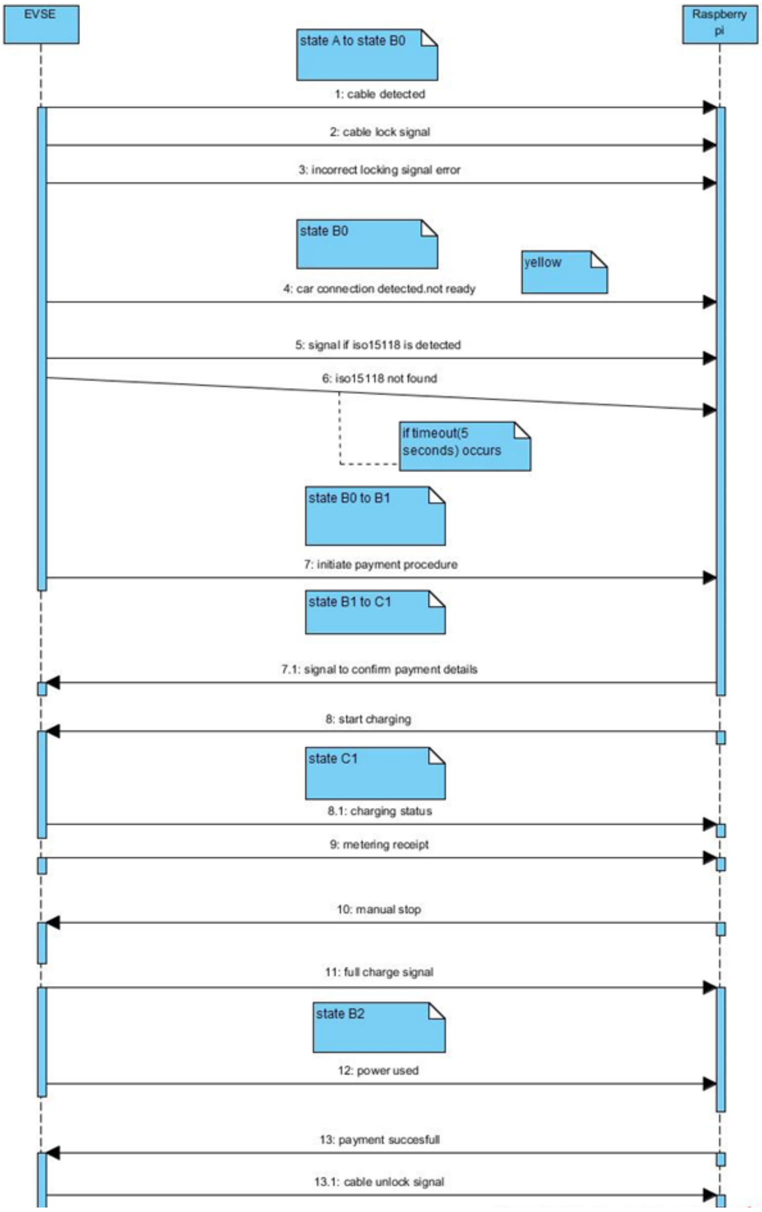


Figure 4.9 UML SEQUENCE DIAGRAM

4.5.3.1 HOW TO USE SEQUENCE DIAGRAMS

- Model and document how your system will behave in various scenarios
- Validate the logic of complex operations and functions

4.6 AUTOMATIC STATE DIAGRAM

4.6.1 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.

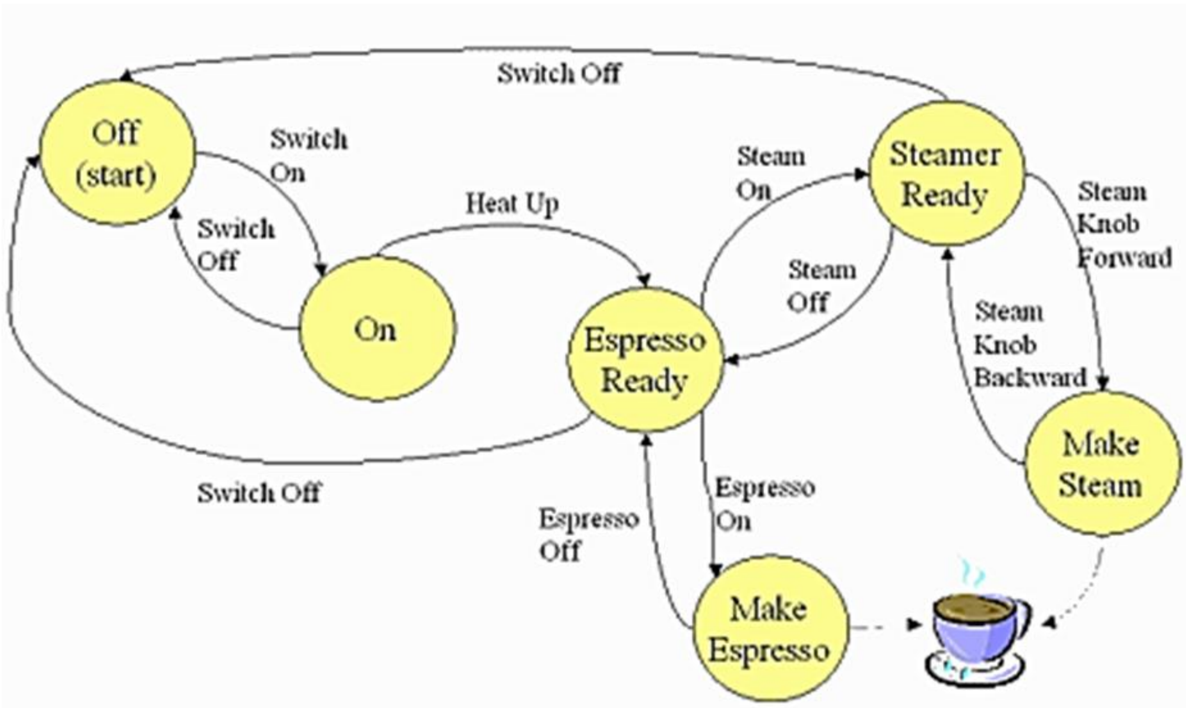


Figure 4.10 COFFEE MACHINE AUTOMATIC STATE DIAGRAM

An illustration of such an automatic state machine is provided by a coffee machine as shown in Figure 4.10. 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 4.10 is shown in Figure 4.11.

```
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;

...

...
```

Figure 4.11 CASE EXAMPLE OF COFFEE MACHINE

4.6.2 STATE DIAGRAM FOR THE ISO 15118

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.

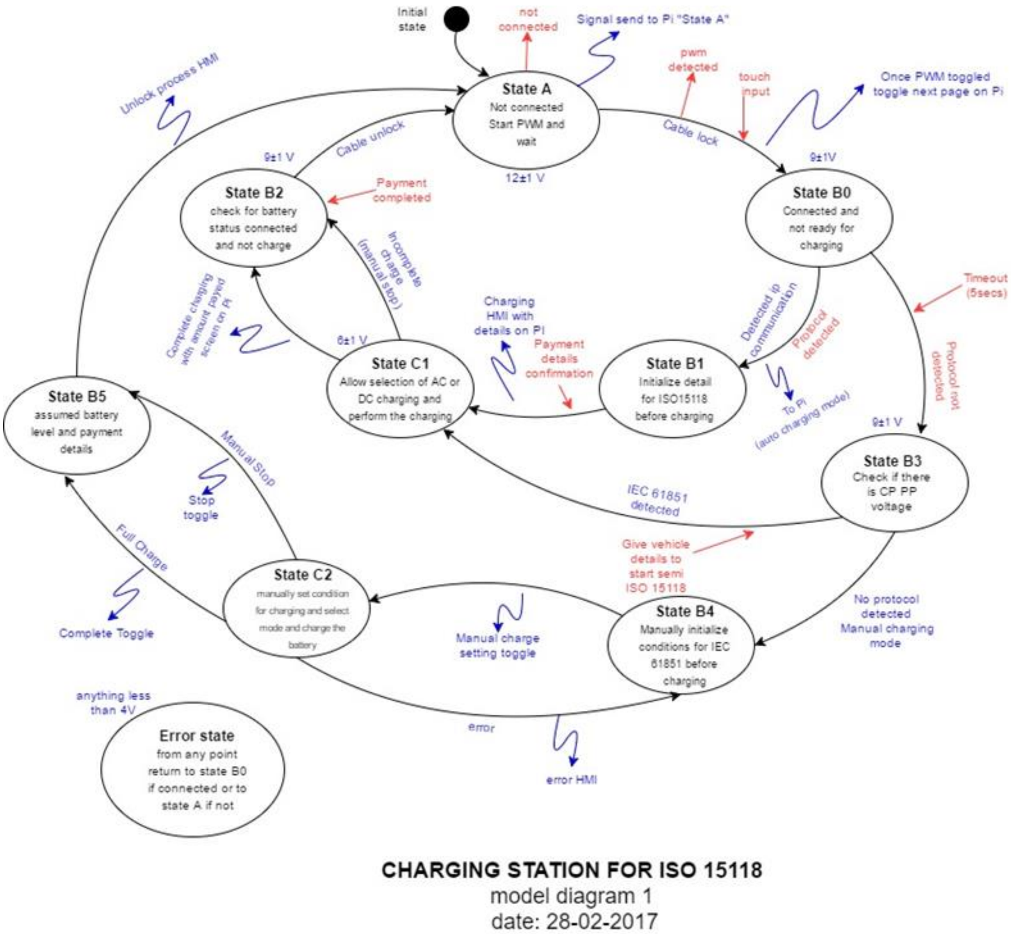


Figure 4.12 BASIC STATE DIAGRAM FOR ISO 15118

CHAPTER V

5. SOFTWARE DESCRIPTION

5.1 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.

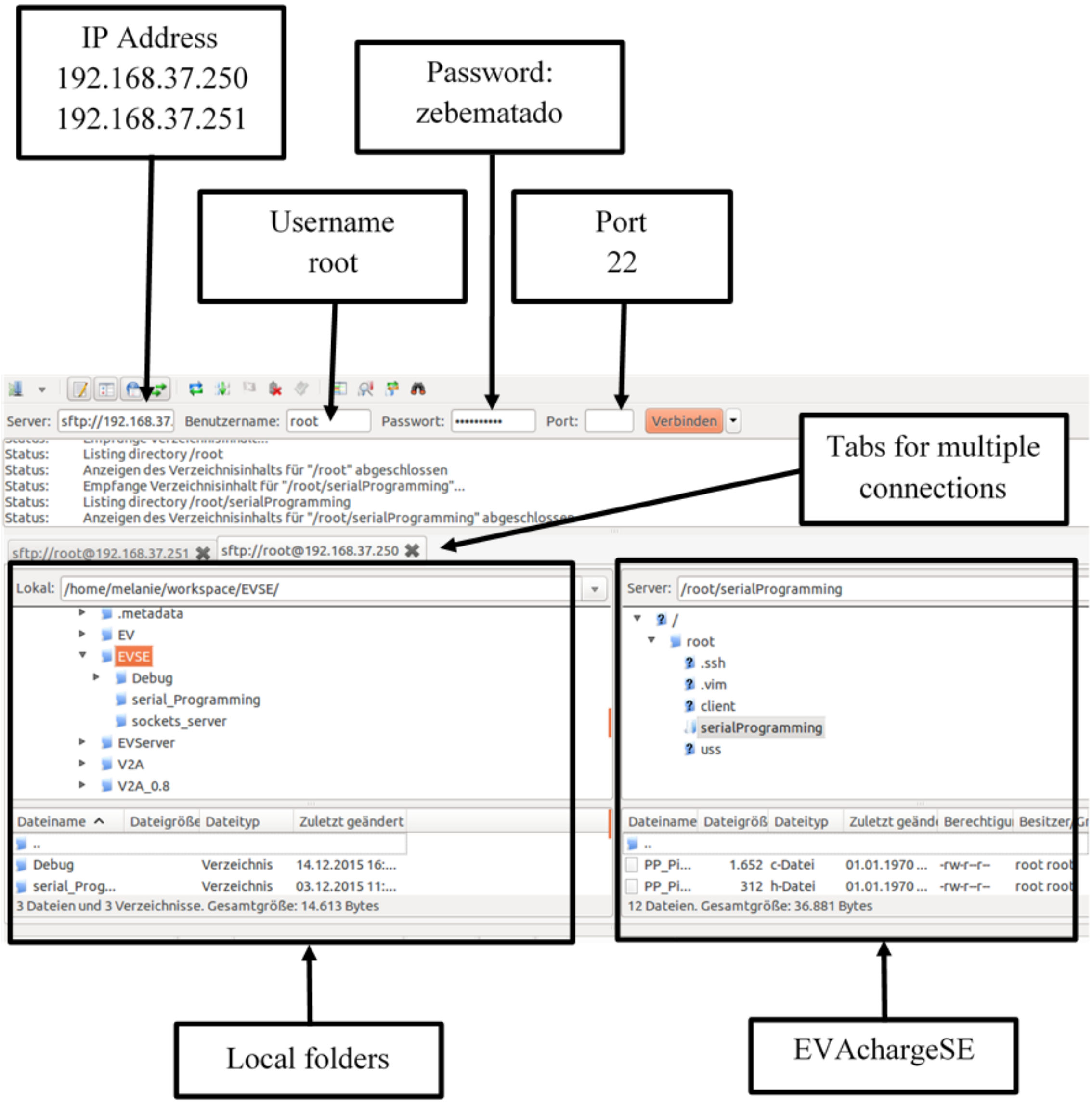


Figure 5.1 FileZilla INTERFACE

5.2 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. The codes in both APPENDIX 2 and 3

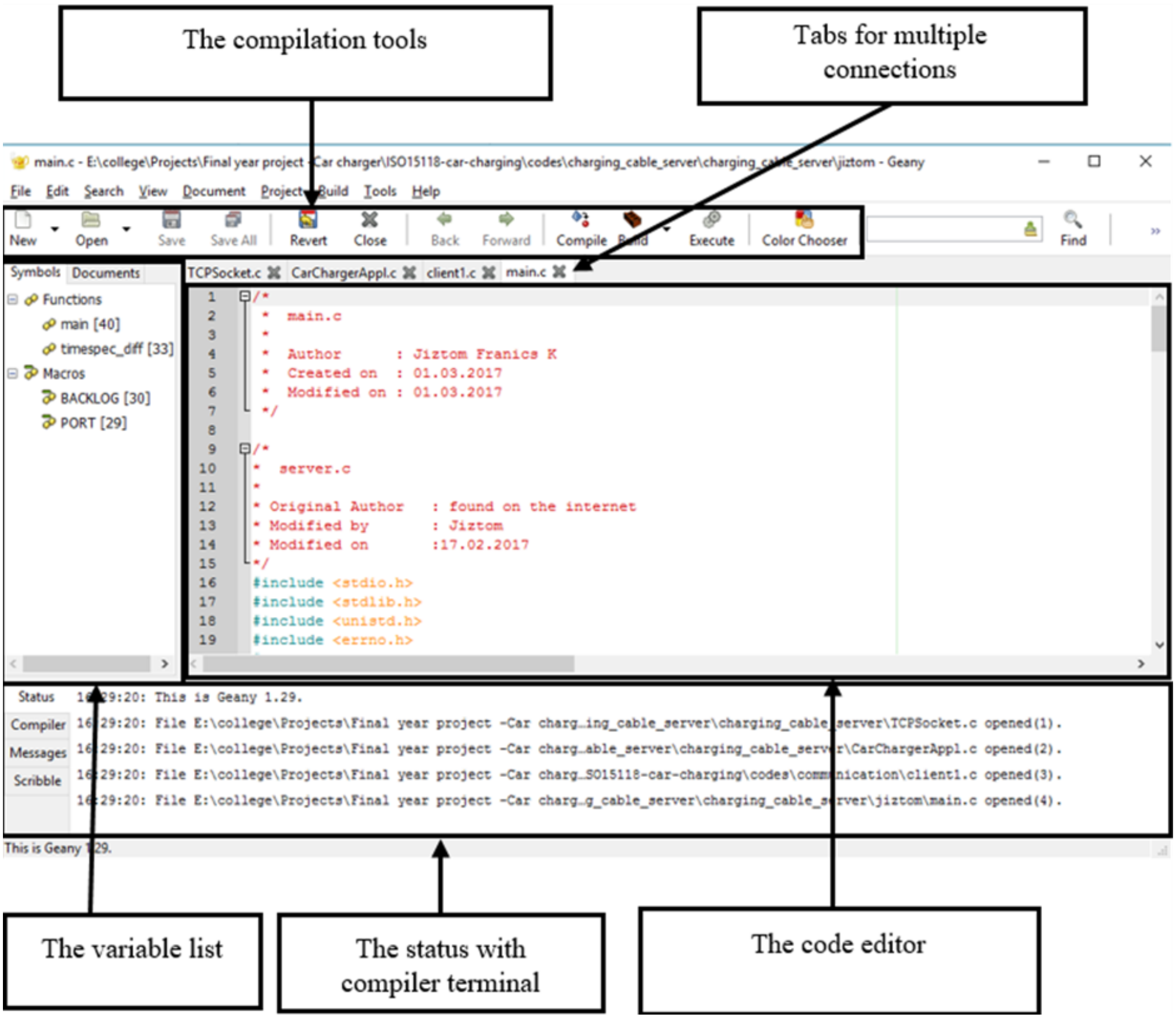


Figure 5.2 Geany INTERFACE

were written using this software.

5.3 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.

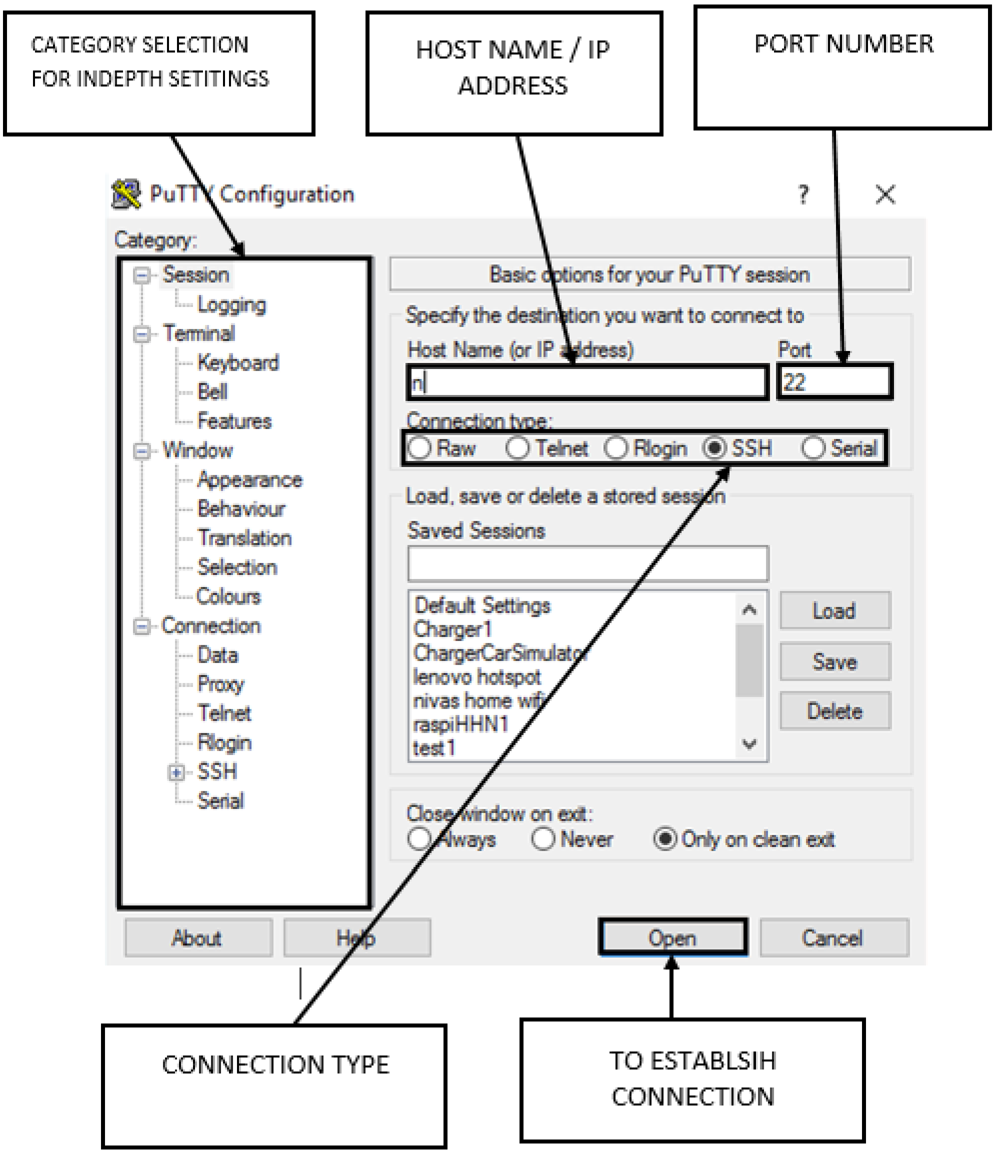


Figure 5.3 PuTTY INTERFACE

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

5.4 EB GUIDE

Human machine interface (HMI)

EB GUIDE is the technology behind some of the best industrial user interfaces in today's cars. With EB GUIDE it is possible to create the best of breed head unit and instrument cluster HMI's. EB GUIDE is more than a tool – it enables an automotive SW development process, leading to world class automotive HMI

5.4.1 BENEFITS

- **Save time and money during development**

  EB GUIDE lets you control HMI development and do so across multiple suppliers, car makers, or car models. With EB GUIDE, you can model and simulate the HMI on your PC and deploy it easily on your target—all with the same look and feel. Moreover, EB GUIDE allows multiple users and distributed teams to work on the same model.

- **Create a compelling state-of-the-art user experience**

  Seamless integration of graphical, haptic, and voice user interfaces enables you to create a consistent user experience. Deliver the advanced user interfaces your customers want, including 3D support, animations and effects, as well as the latest speech technology.

- **Benefit from worldwide support from an experienced provider**

  For more than ten years, EB GUIDE and EB experts have been assisting HMI developers worldwide. We continue to be an industry innovator. EB works with leading international chip vendors and technology providers to extend and improve EB GUIDE continuously according to the market's needs.

5.4.2 FEATURES

- All-in-one tool for specification, modelling, prototyping, and mass production of HMIs for digital instrument clusters, infotainment systems, heads-up displays (HUDs), and industry applications
- Provides a single tool for multimodal modelling of graphical, haptic, and voice user interfaces
- Supports you in developing HMIs with 3D graphics, effects, and animations
- Enables you to use the latest speech dialog technology
- Allows you to integrate HTML 5 content into your native HMI
- WYSIWYG interface lets you evaluate your HMI in an early development stage

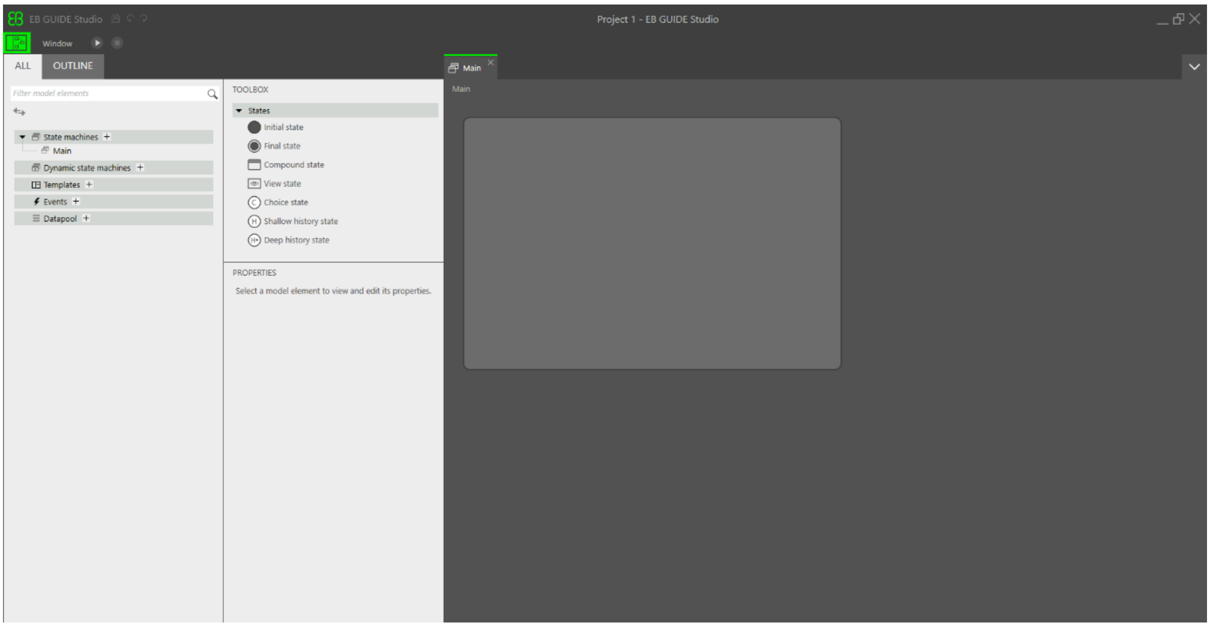


Figure 5.4 EB GUIDE 6.3 INTERFACE

5.5 www.draw.io

The draw.io is a free online diagram editor with support to various formats which includes mock-up, flowcharts, sequence diagrams, and so on.

This was a project started by a group of students and was later incorporated by google as one of their free online editors. Since it does not require any pre installation and can work in any browser (the interface shown in Figure 6 www.draw.io interface), it has become a popular tool for simple and easy.

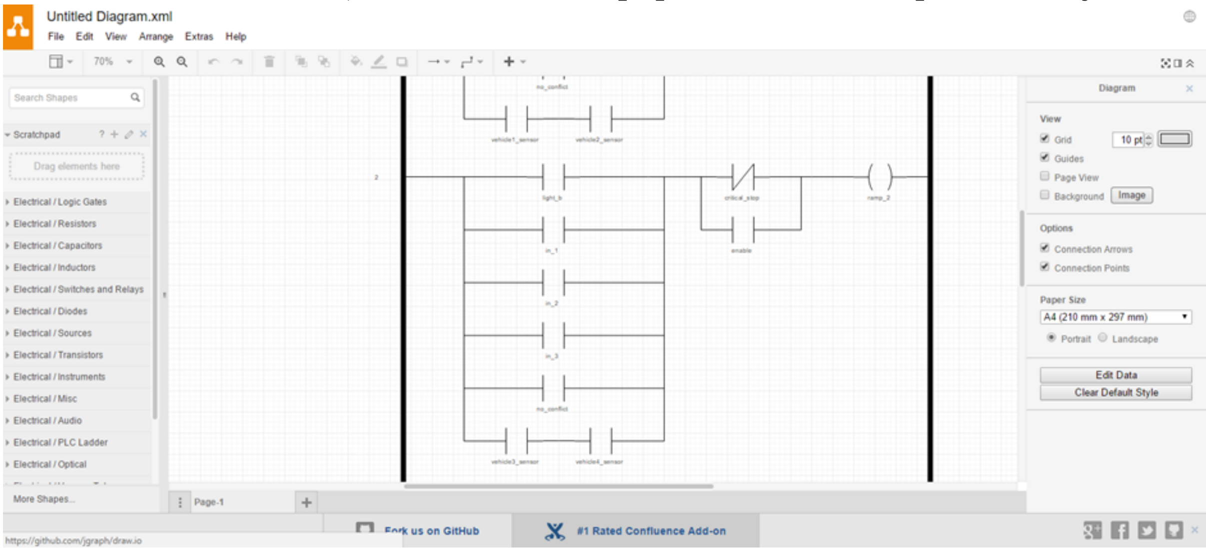


Figure 5.5 www.draw.io INTERFACE

In this project the state diagram (Appendix 1.3 and Appendix 1.4) and basic sequence diagrams was created using this software.

5.6 VISUAL PARADIGM 14.0

5.6.1 INTRODUCTION

Visual Paradigm is a word-wide leading award-winning enterprise management and software development suite. This product provides all the feature you needs for enterprise architecture, project management, software development and team collaboration in a one-shop-stop solution.

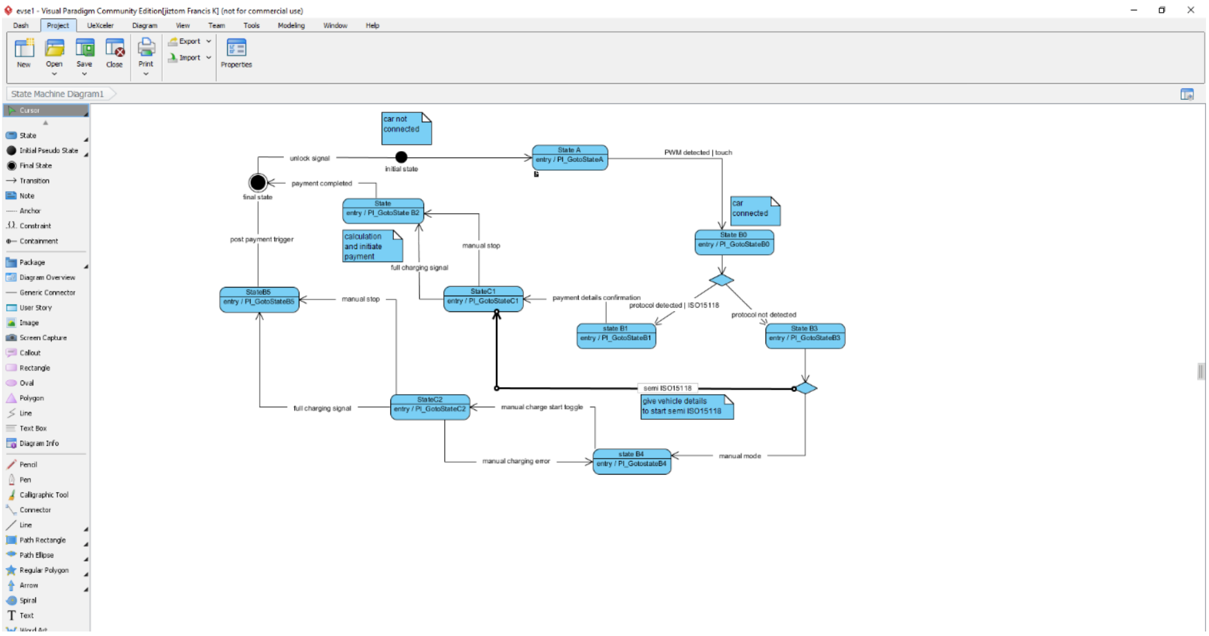


Figure 5.6 VISUAL PARADIGM INTERFACE

5.6.2 BENEFITS

Visual Paradigm provides the following key features so as to help you simplify your application development:

- Persistence Made Easy
- Sophisticated Object-Relational Mapping Generator
- Model Driven Development
- Extensive Database Coverage
- Database Reverse Engineering
- Class Reverse Engineering
- IDE Integration

5.7 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 5.7. The Linux terminal in which this call was executed now represents a command line of the EVAchargeSE board.

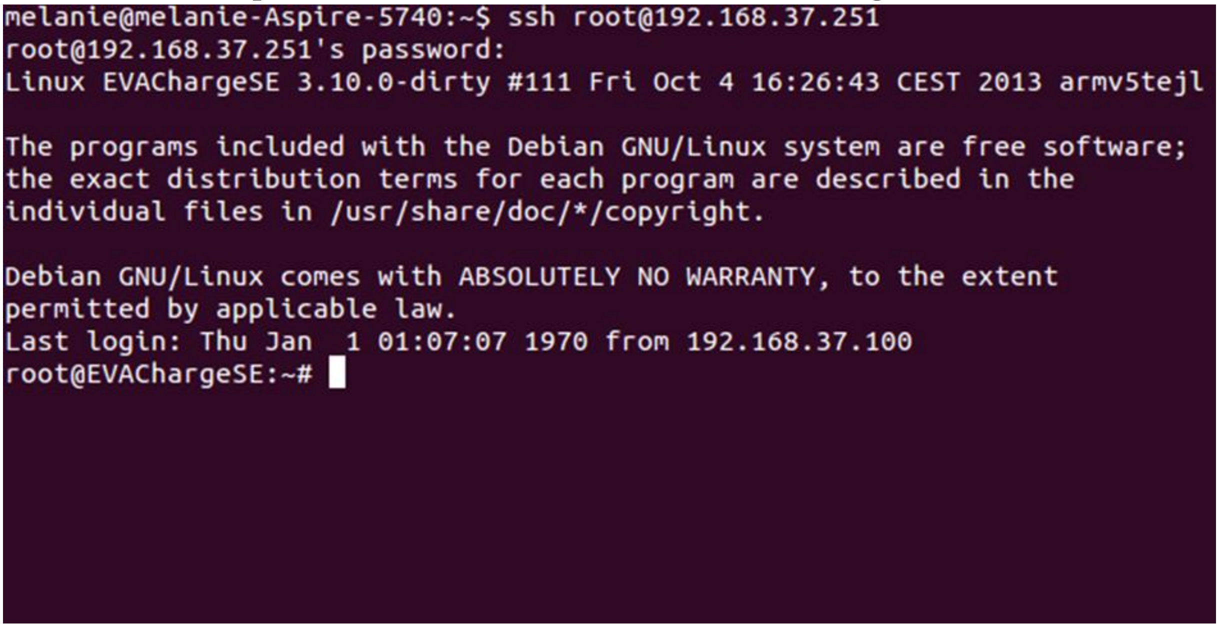
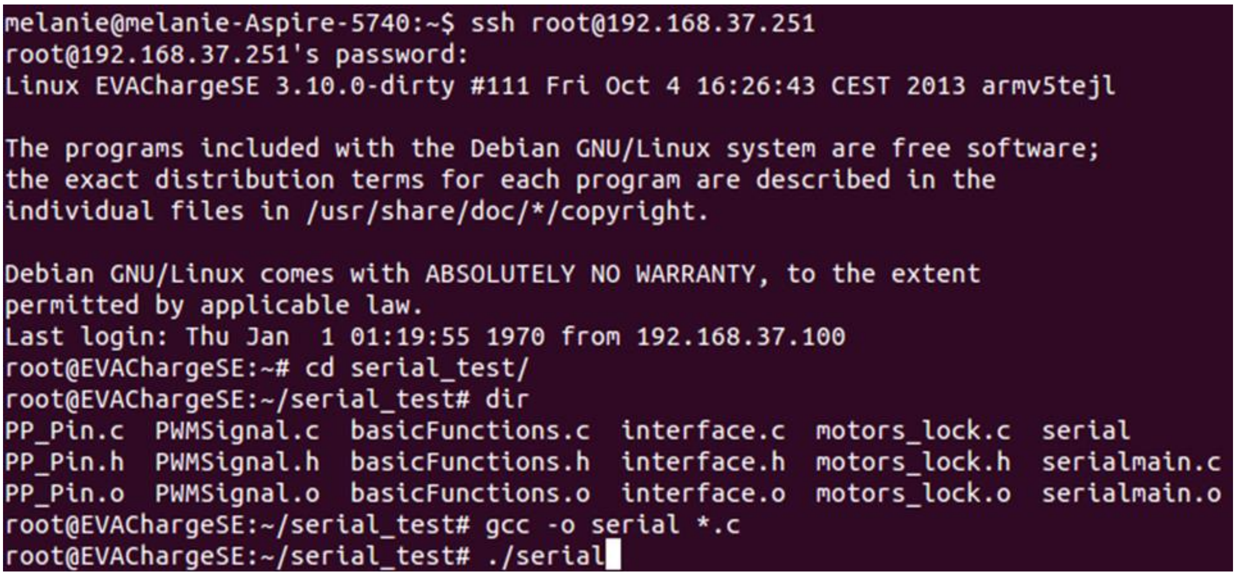


Figure 5.7 LOGIN SHELL FOR EVACharge SE USING SSH

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 5.8.



```
melanie@melanie-Aspire-5740:~$ ssh root@192.168.37.251
root@192.168.37.251's password:
Linux EVAChargeSE 3.10.0-dirty #111 Fri Oct 4 16:26:43 CEST 2013 armv5tejl

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Jan  1 01:19:55 1970 from 192.168.37.100
root@EVAChargeSE:~# cd serial_test/
root@EVAChargeSE:~/serial_test# dir
PP_Pin.c  PWMSignal.c  basicFunctions.c  interface.c  motors_lock.c  serial
PP_Pin.h  PWMSignal.h  basicFunctions.h  interface.h  motors_lock.h  serialmain.c
PP_Pin.o  PWMSignal.o  basicFunctions.o  interface.o  motors_lock.o  serialmain.o
root@EVAChargeSE:~/serial_test# gcc -o serial *.c
root@EVAChargeSE:~/serial_test# ./serial
```

Figure 5.8 COMPILING THE PROGRAM ON GCC

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**".

General Nutshell commands	Explanation
ssh root@192.168.37.250	Establishing a secure shell connection
sftp 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

Table 5.1 GENERAL NUTSHELL COMMANDS

CHAPTER VI

6. HARDWARE DESCRIPTION

6.1 RASPBERRY PI 3

6.1.1 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

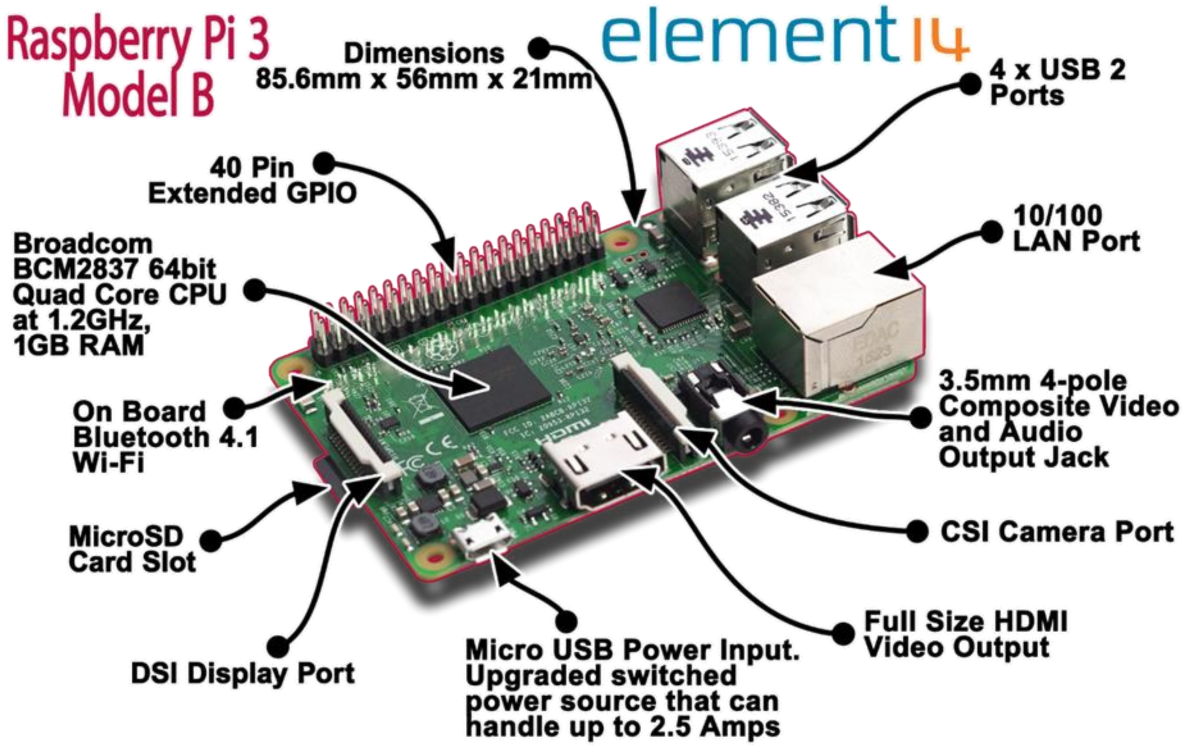


Figure 6.1 RASPBERRY PI 3 MODEL B SPECIFICATION

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.

6.1.2 RASPBERRY PI 7" OFFICIAL TOUCH SCREEN

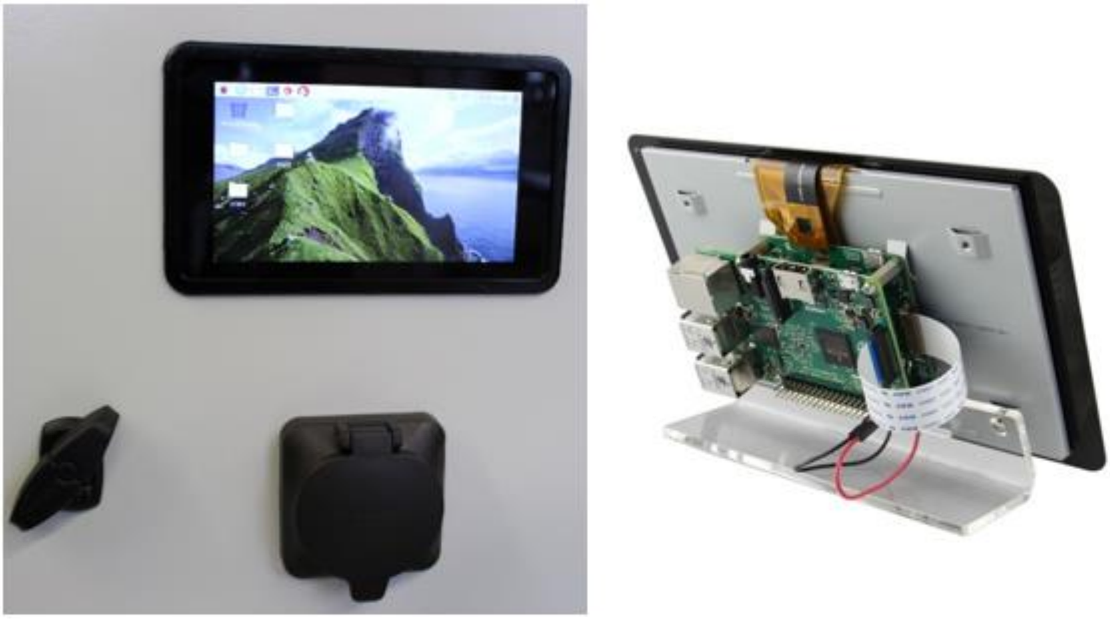


Figure 6.2 OFFICIAL 7"TOUCH SCREEN

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.

Figure 6.3 PARTS OF THE TOUCH SCREEN WITH DRIVER

6.2 EVACharge SE BOARD

6.2.1 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 centres 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.

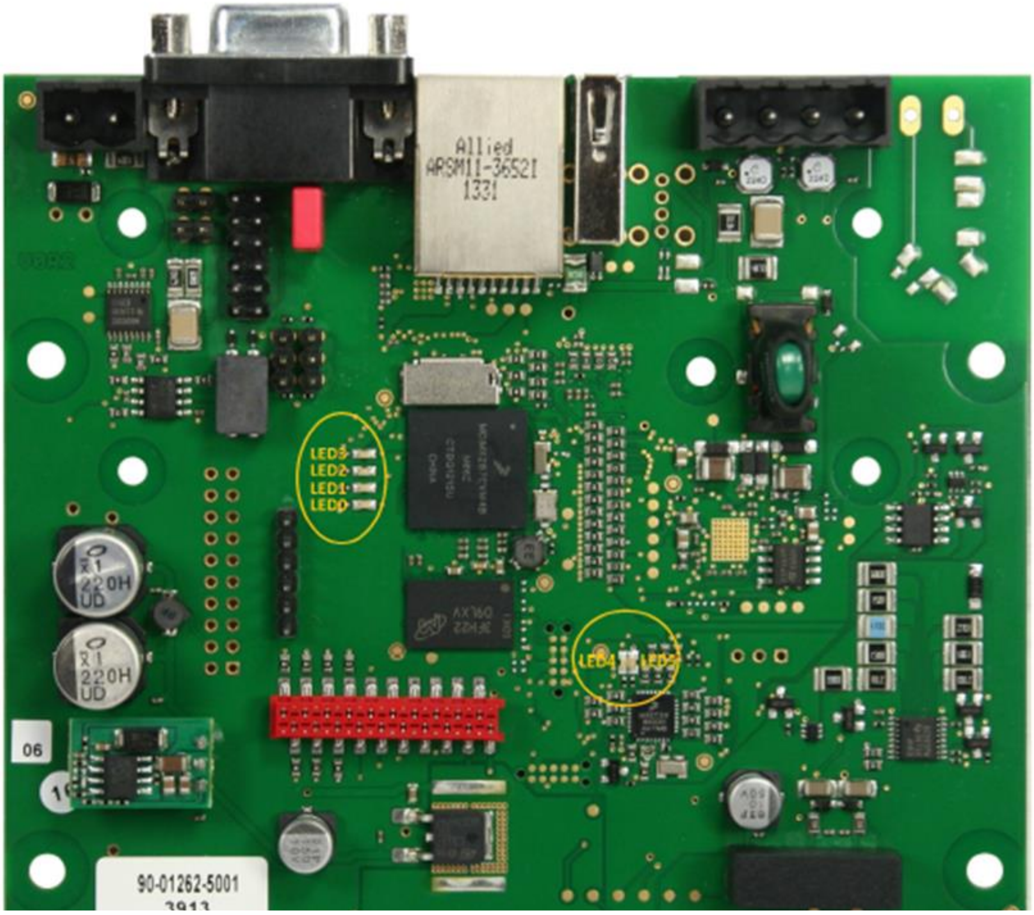


Figure 6.4 EVACharge SE BOARD

6.2.2 SPECIFICATIONS

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

Table 6.1 SPECIFICATION OF EVACharge SE BOARD

6.2.3 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

6.3 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).

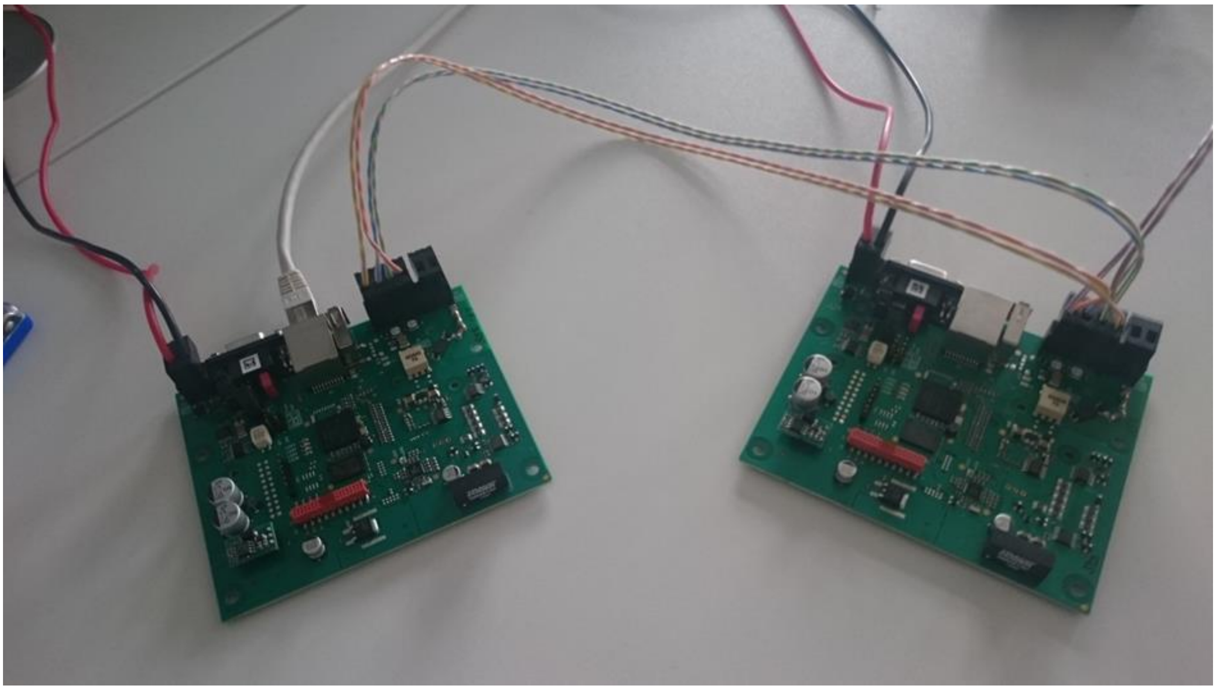


Figure 6.5 EVSE EV COMMUNICATION

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, and 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, and 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.

6.4 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 Section 5.7.

Figure 6.6 EVSE AND RASPBERRY PI ON THE MODULE

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

6.5 INTEGRATING THE SYSTEM

The integration of the system refers to the logic implementation of the state diagram. The first step was to make sure there is a connection established between the Raspberry pi and the EVSE. This was made as the first connection before the handshake protocol for the Green PHY is initiated. The language selection on the HMI is used as the key to start the IOS 15118 handshake procedure. This also ensures that the EVSE will not run the ISO 15118 protocol throughout and thereby saving processing time and reducing standby power consumed.

The next aspect would be the synchronization of the EVSE –EV communication with the Raspberry pi to physically show the user what happens. The cable detection and lock signal are all important parameters which are required to show the user that the process is running well. Also from the V2GProject has the payment process as a part of the code but to reduce the load this is converted into an internal process within the HMI. It will invoke the payment on the HMI and after the payment details are satisfied it will send the authorization signal which will continue the process.

The EV will allow the choice of which type of charging needs to be done and it will communicate to the EVSE the status of charging and the same will be pushed to the HMI as string data to be displayed to the user. There is also a manual stop option to stop the charging process from the HMI, which allows the user to stop the charging at a short notice.

After the charging is done it will invoke the payment process and only after the payment is successfully done will the EVSE allow to unlock the cable on both the EV and EVSE side. This works as a way of preventing people from performing free charge.

After this, the EV will be disconnected and the EV and Raspberry pi will go to the initial not connected state. So again the language selection mode is found.

CHAPTER VII

7. 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 3.

Separate output code of EV and EVSE is also attached in APPENDIX 3.

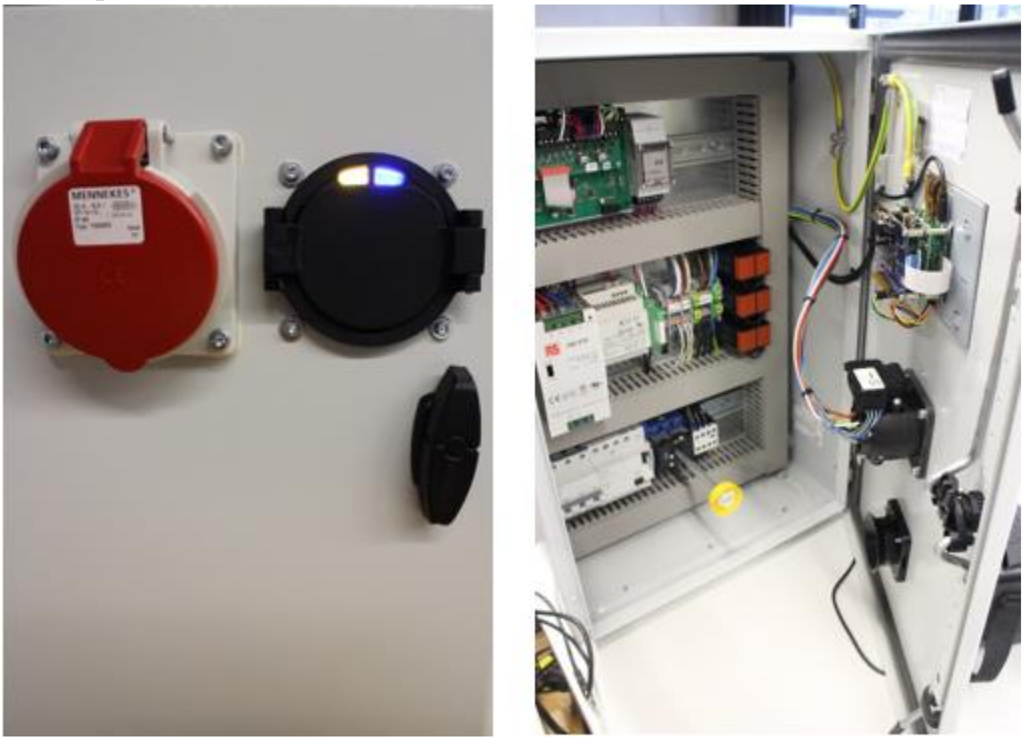


Figure 7.1 THE WORKING MODEL WITH THE SIGNAL ON EV

The output of the HMI model created using EB GUIDE is attached in Appendix 5.

CHAPTER VIII

8. 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 ever 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…

CHAPTER IX

9. BIBLIOGRAPHY

[1] IEC 61851---1, Electric vehicle conductive charging system — Part 1 General requirements

[2] http://www.iso.org/iso/catalogue_detail.htm?csnumber=55366

[3] http://www.jsae.or.jp/e07pub/yearbook_e/2014/docu/28_industry_standards.pdf

[4] http://www.power---up.org/wp---content/uploads/2012/07/PowerUp_D3.2_Final_V2G_Architecture.pdf

[5] http://www.insys---icom.co.uk/bausteine.net/f/10599/DS_en_INSYS---Powerline--GP_130715_eBook.pdf?fd=0

[6]http://www.iaria.org/conferences2012/filesSECURWARE12/Electric_Vehicle_Charging_Infrastructure_Keynote%20RainerFalk.pdf

[7] http://dlms.com/documents/ARCHIVE/Excerpt_GB6.pdf

[8] http://www.openchargealliance.org/?q=node/4170

[9] https://en.wikipedia.org/wiki/IEC_61850

[10] http://blog.iec61850.com/2012/06/iec---61850---90---8---object---models---for---e.html

[11] http://www.smartv2g.eu/

[12] Von Alexander Barth & Michael Röhrich, Studienarbeit ISO 15118, Bachelors Thesis 2013, Hochschule Heilbronn, Heilbronn, Germany.

[13] www.wikipedia.org

[14] Peter Hank, Steffen Müller, Ovidiu Vermesan, Jeroen Van Den Keybus, 'Automotive Ethernet: in-vehicle networking and smart mobility', DATE '13 Proceedings of the Conference on Design, Automation and Test in Europe Pages 1735-1739

[15] B O Chen , Keith S. Hardy, Jason D. Harper, Daniel S. Dobrzynski , ' Towards standardized Vehicle Grid Integration' , Transportation Electrification Conference and Expo (ITEC) ,2015 ,IEE.

[16] Dr. Andreas Heinrich, Michael Schwaiger, Daimler AG and BMW group, 'ISO 15118 – charging communication between plug-in electric vehicles and charging infrastructure', Grid Integration of Electric Mobility, 1st international ATZ Conference 2016, pp 213-227

[17] M. Sc. Michael Tybel, Dr. –Ing Andrey Popov, Dr. –Ing Michael Schugt , Scienlab electronic systems, Bochum, 'Assuring Interoperability between Conductive EV and EVSE Charging Systems', Popov.com.

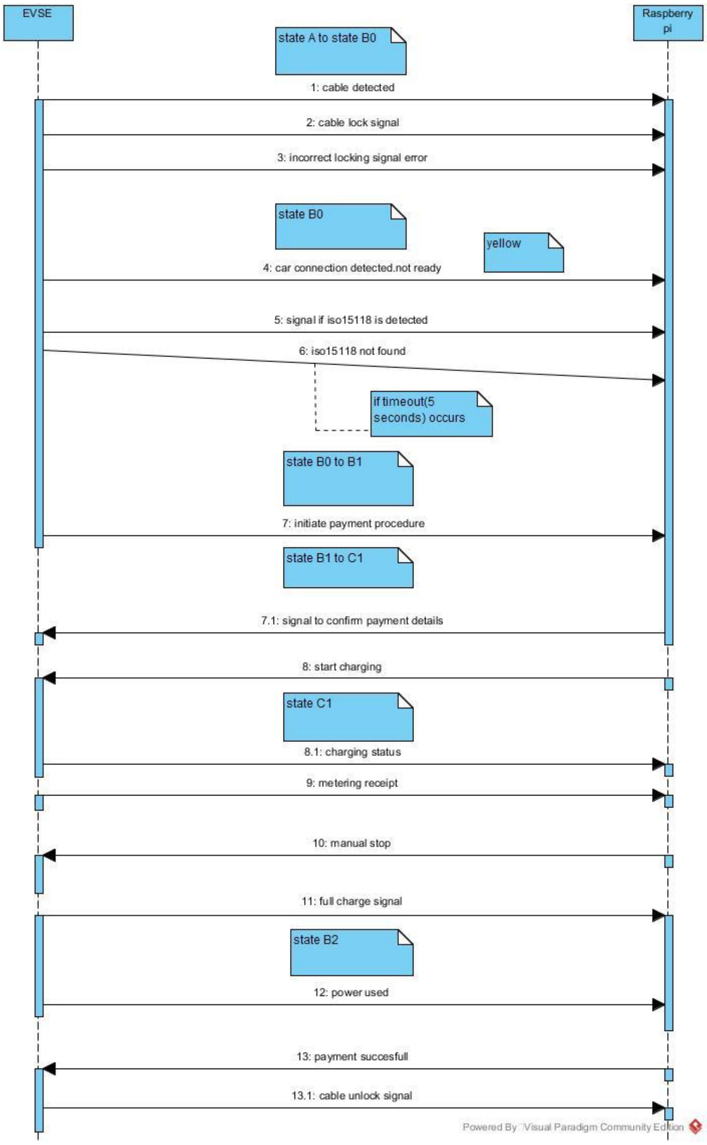
[18] D. Wellisch, J Lenz, A Faschingbauer, R. Posch, S. Kunze , Deggendorf Institute of Technology, Freyung. Research gate Publication, 'Vehicle-to-grid AC charging Station' An Approach for Smart Charging Development.'

APPENDIX

APPENDIX 1

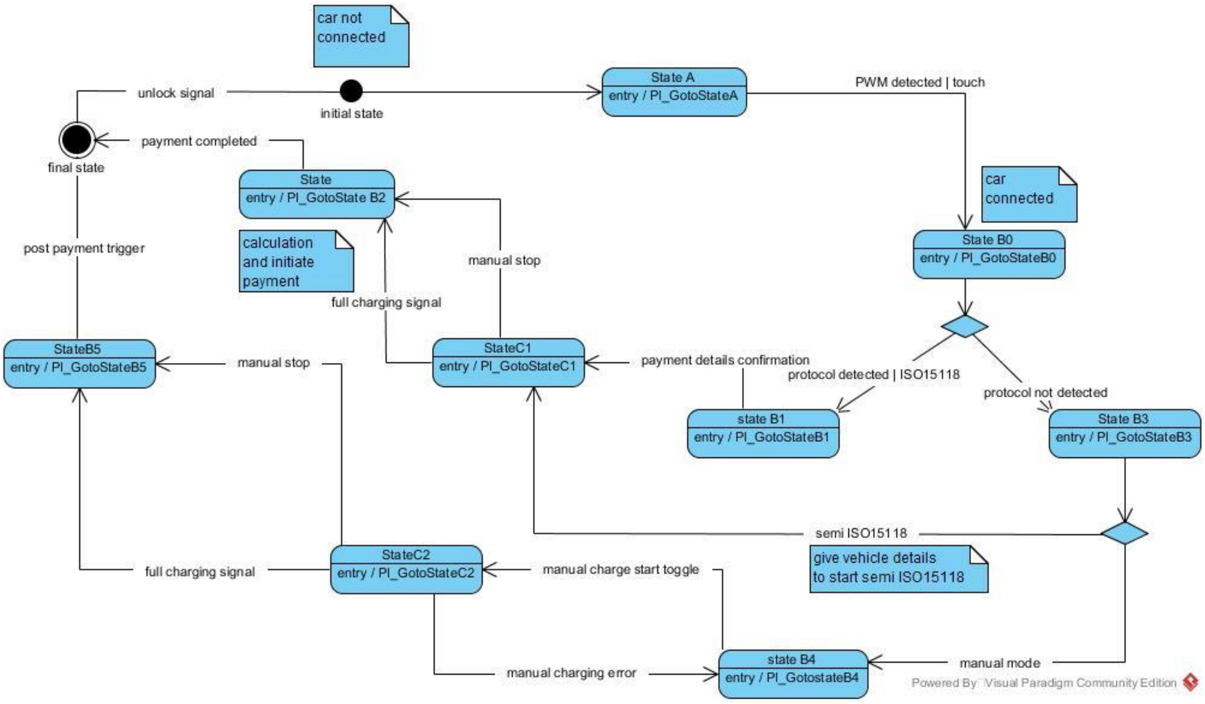
The State diagrams and the UML diagram generated as a part of the project are shown here

a. UML SEQUENCE DAIGRAM



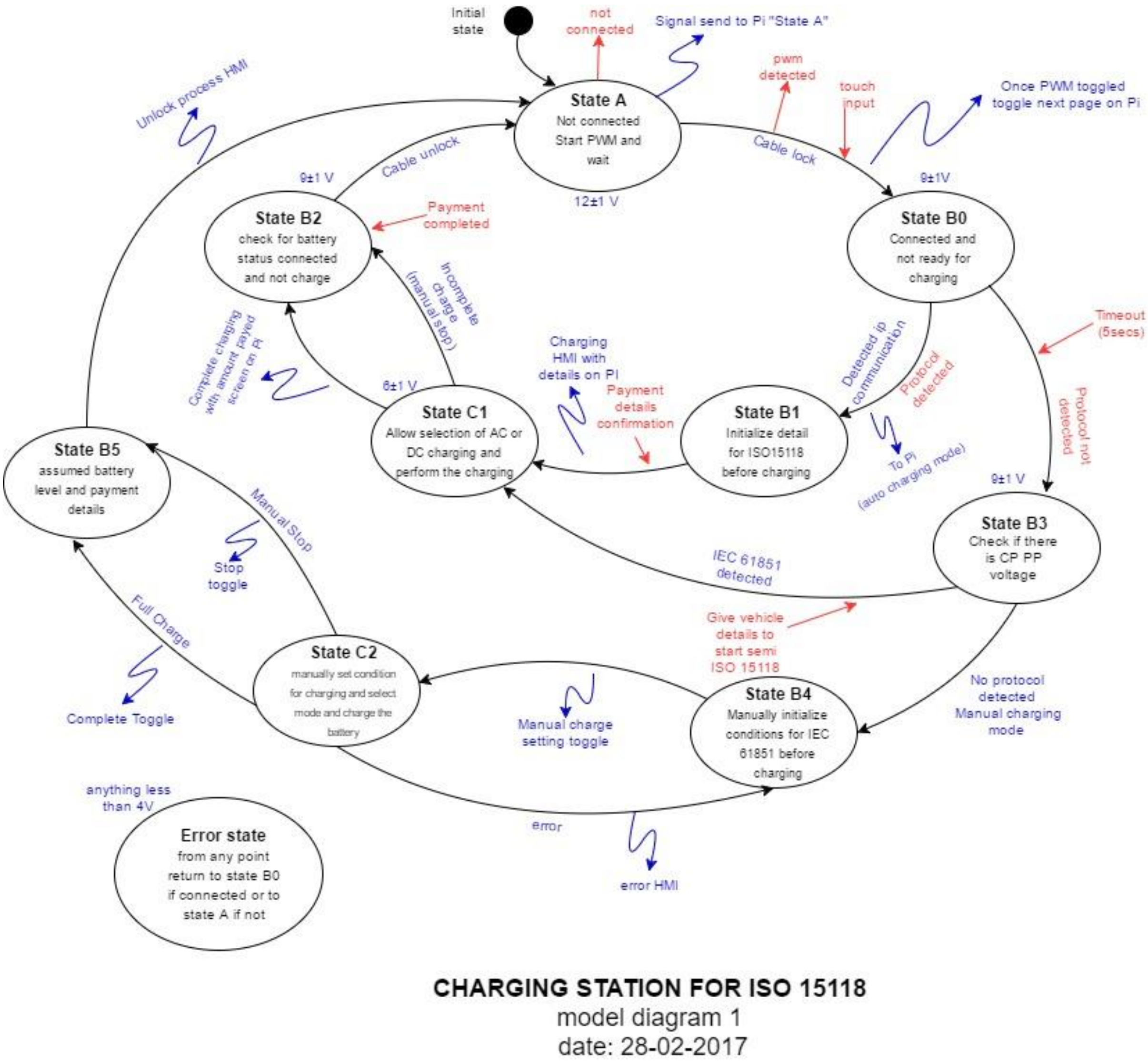
Appendix 1.1 UML SEQUENCE DIAGRAM

b. UML STATE DIAGRAM



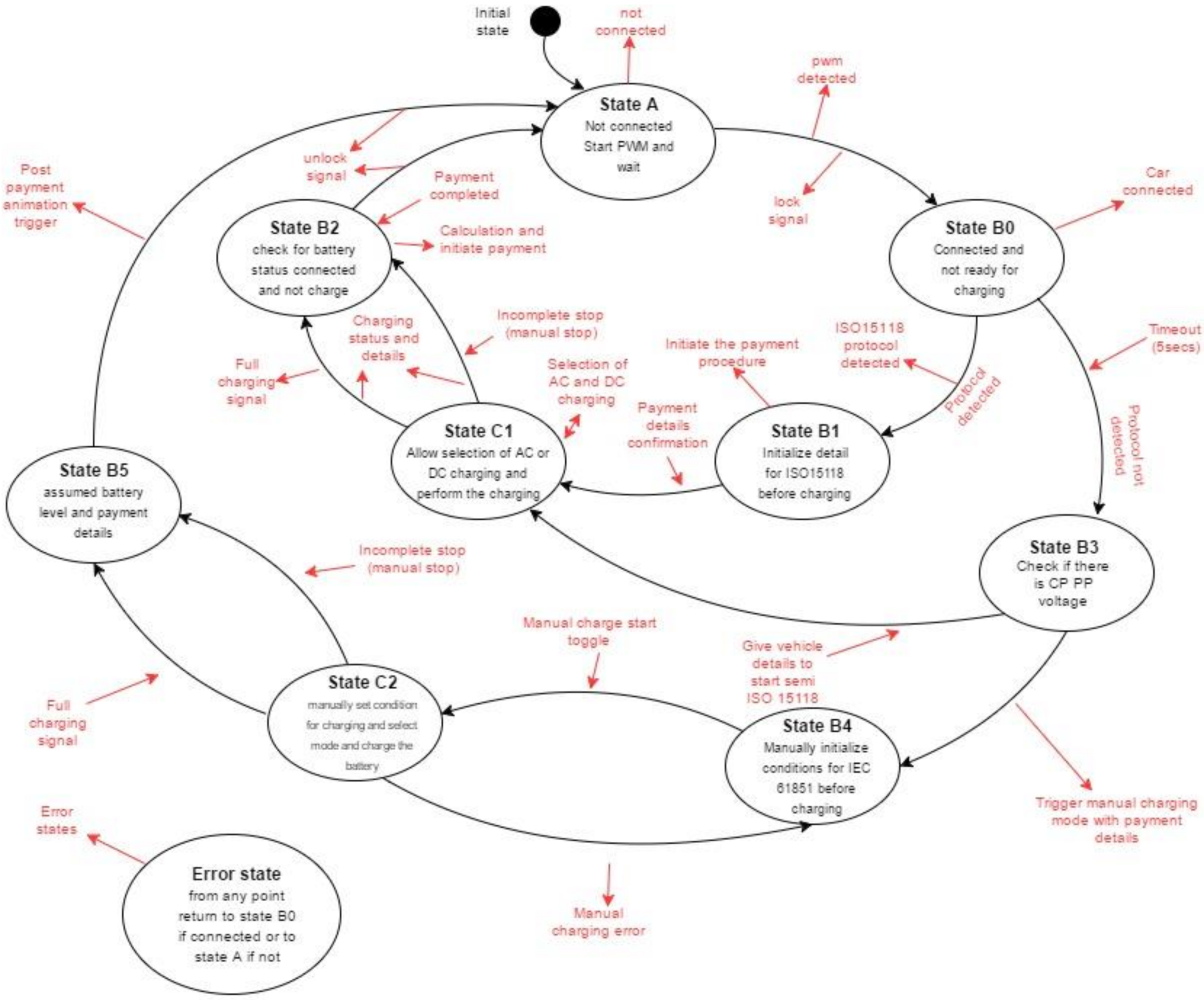
Appendix 1.2 UML STATE DIAGRAM

c. THE STATE DIAGRAM FOR THE EVSE SIDE



Appendix 1.3 STATE DIAGRAM -EVSE

d. THE STATE DIAGRAM FOR THE RASPBERRY SIDE



Appendix 1.4 STATE DIAGRAM -RASPBERRY PI

APPENDIX 2

This includes the code in the raspberry pi

```
/*
 * main.h
 *
 * Created on      : 06.03.2017
 *          Author : Jiztom Francis
 *
 *
 */

#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 Franics 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);
```

```c
/*
 *  switchpi.h
 *
 *  header file for switchpi.c
 *
 *    Author             : Jiztom
 *    Created on   : 01.03.2017
 * modified on     :
 */


# 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_*/
```

```c
/*
*
* 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;
}
```

```c
/*
*      transfer.h for client
*
*      Author      : Jiztom
*      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 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
*      Created on   : 17.02.2017
*      Modified on  : 17.02.2017
*/

#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 3

```
/*
 * 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
 *  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
```

```c
/*
 * 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.c
haractersLen);
		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_isUs
ed){
                                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
 *    Edited by  : Jiztom Francis
 *  Modified on : 08.02.2017
 */

# 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);
}
```

```c
/*
*     transfer.h for EVSE
*
*     Author          : Jiztom
*     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
*     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 4

The OUTPUT from the three systems on implementing the codes.

a. 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 :
```

b. EVSE output

```
root@EVAChargeSE:~/modified 08.03.2017# ./EVSE

EVServer: socket()...
 EVServer: bind()...
 Server: listen()...
 Server an Port 5000 wartet...
Len: 3
Server:Msg Received

+++ Start protocol example ISO 15118 +++
Pos Voltage 12.035000 V Neg Voltage 8.845000 V
+++ check level on CP for state B: EV detected, 12.000000 +++

Server:Msg being sent: :11
+++ End of example +++
Lock Status 0
Server:Msg being sent: :45

+++ Start protocol example ISO 15118 +++
^[[A^C
root@EVAChargeSE:~/modified 08.03.2017# ./EVSE

EVServer: socket()...
 EVServer: bind()...
 Server: listen()...
 Server an Port 5000 wartet...
^X^C
root@EVAChargeSE:~/modified 08.03.2017# ./EVSE

EVServer: socket()...
 EVServer: bind()...
 Server: listen()...
 Server an Port 5000 wartet...
Len: 3
Server:Msg Received

+++ Start protocol example ISO 15118 +++
Pos Voltage 8.874000 V Neg Voltage 8.584000 V
+++ check level on CP for state B: EV detected, 8.000000 +++

Server:Msg being sent: :11
Lock Status 1
Server:Msg being sent: :12
+++ release for charging: State B: vehicle detected +++
+++ Start application handshake protocol example +++

receive stream......received stream
EVSE side: List of application handshake protocols of the EV
	Protocol entry #=1
		ProtocolNamespace=urn:iso:15118:2:2010:MsgDef
		Version=1.0
		SchemaID=1
		Priority=1
	Protocol entry #=2
```

```
            ProtocolNamespace=urn:din:70121:2012:MsgDef
            Version=1.0
            SchemaID=2
            Priority=2
Server:Msg being sent: :51
EVSE side: send response to the EV
send stream... ...sent
+++ Terminate application handshake protocol example +++

receive stream......received stream
EVSE side: sessionSetup called
      Received data:
      Header SessionID=0 0 0 0 0 0 0 0
             EVCCID=10
send stream... ...sent
receive stream......received stream
EVSE side: serviceDiscovery called
      Received data:
      Header SessionID=1 2 3 4 5 6 7 8
             ServiceCategory=1
send stream... ...sent
receive stream......received stream
EVSE side: serviceDetail called
      Received data:
      Header SessionID=1 2 3 4 5 6 7 8
             ServiceDetailID=22
send stream... ...sent
receive stream......received stream
EVSE side: paymentServiceSelection called
      Received data:
      Header SessionID=1 2 3 4 5 6 7 8
             SelectedPaymentOption=ExternalPayment
             ServiceID=1
             ServiceID=22
             ParameterSetID=1
Server:Msg being sent: :21
send stream... ...sent
receive stream......received stream
EVSE side: paymentDetails called
      Received data:
             eMAID=1
             ID=dD
             Certificate=Ce
             SubCertificate 1=Su
             SubCertificate 2=Su
send stream... ...sent
receive stream......received stream
EVSE: Authorization called
      Received data:
                   GenChallenge=1
                   ID=Id2
Len: 1
Server:Msg Received
send stream... ...sent
receive stream......received stream
EVSE side: chargeParameterDiscovery called
```

```
      Received data:
              EVRequestedEnergyTransferType=0
              DepartureTime=17508
              EAmount=100
              EVMaxCurrent=200
              EVMaxVoltage=400
              EVMinCurrent=500
send stream... ...sent
Pos Voltage 7.047000 V Neg Voltage 8.613000 V
+++ check level on CP for state C: EV connected, ready, 7.000000 +++

+++ Start Communication State C +++

receive stream......received stream
EVSE side: powerDelivery called
      Received data:
               ChargeProgress=0
               SAScheduleTupleID=10
send stream... ...sent
receive stream......received stream
EVSE side: chargingStatus called
Server:Msg being sent: :32
send stream... ...sent
receive stream......received stream
EVSE side: meteringReceipt called
      Received data:
              ID=Id3
              SAScheduleTupleID=12
              SessionID=0
              MeterInfo.MeterStatus=2
              MeterInfo.MeterID=3
              MeterInfo.isused.MeterReading=1
              MeterReading.Value=100
              MeterInfo.TMeter=123456789
Server:Msg being sent: :36
send stream... ...sent
receive stream......received stream
Len: 1
Server:Msg Received !
EVSE side: powerDelivery called
      Received data:
               ChargeProgress=0
               SAScheduleTupleID=123
send stream... ...sent
Pos Voltage 8.874000 V Neg Voltage 8.584000 V
+++ check level on CP for state B: EV detected, 8.000000 +++

+++ Start Communication State B +++

receive stream......received stream
EVSE side: sessionStop called
      Received data:
      Header SessionID=1 2 3 4 5 6 7 8
              ChargingSession=1
Server:Msg being sent: :41
Len: 3
```

```
Server:Msg Received *
send stream... ...sent
+++ End of example +++
Lock Status 0
Server:Msg being sent: :45

+++ Start protocol example ISO 15118 +++
```

c. EV output

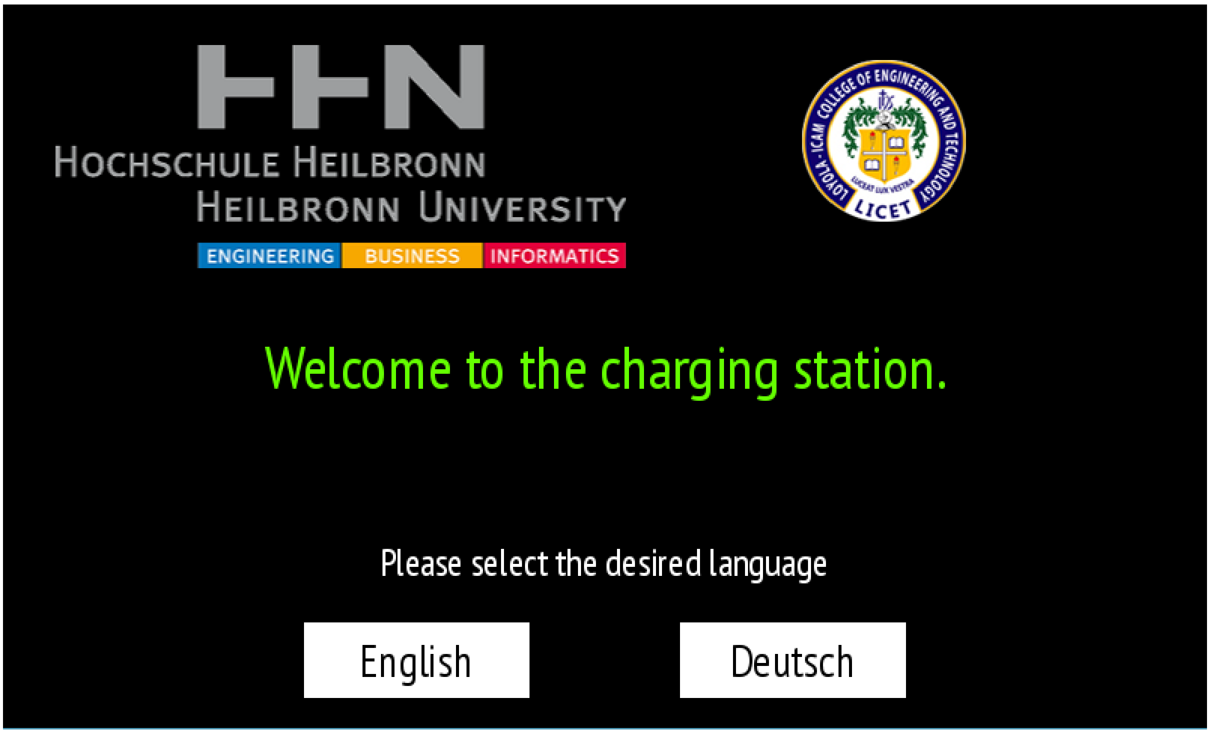
```
+++ level on CP: standby, 1900.196000 +++


 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
```

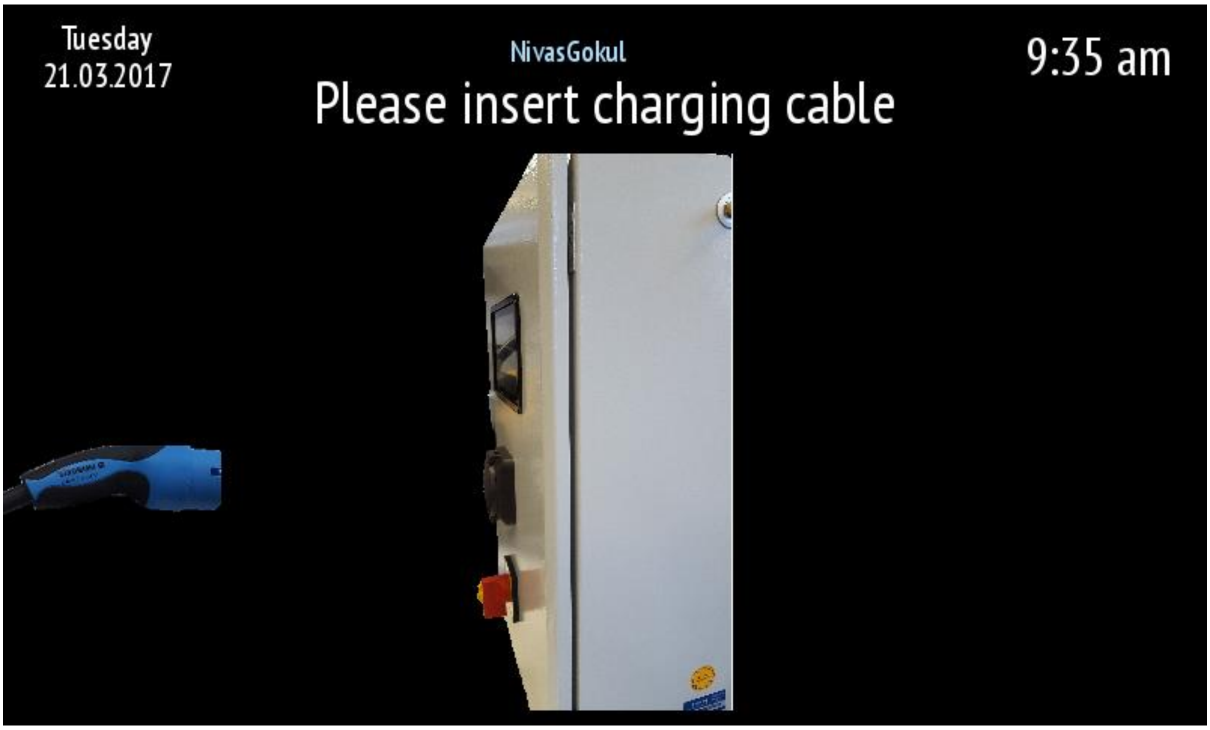
```
EV side: call EVSE ServiceDetail
send stream... ...sent
receive stream......received stream
EV side: received response message from EVSE
      Header SessionID=1 2 3 4 5 6 7 8
       ResponseCode=0
       ServiceID=1234
            Length=2
            ServiceSetID=1
            Parameters=2
                   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
```

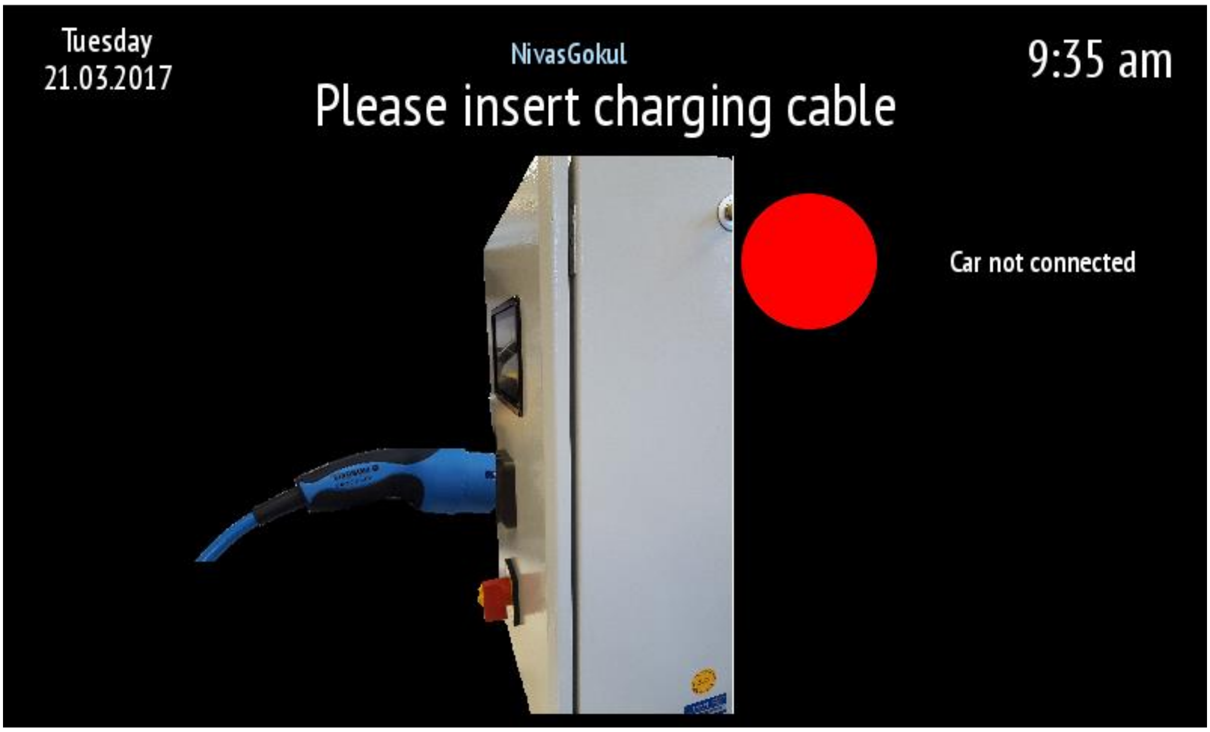
```
EV side: received response message from EVSE
      Header SessionID=1 2 3 4 5 6 7 8
       ResponseCode=0
      EVSEStatus:
            RCD=0
            EVSENotification=3
            NotificationMaxDelay=12
EV side: call EVSE chargingStatus
send stream... ...sent
receive stream......received stream
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 +++
```

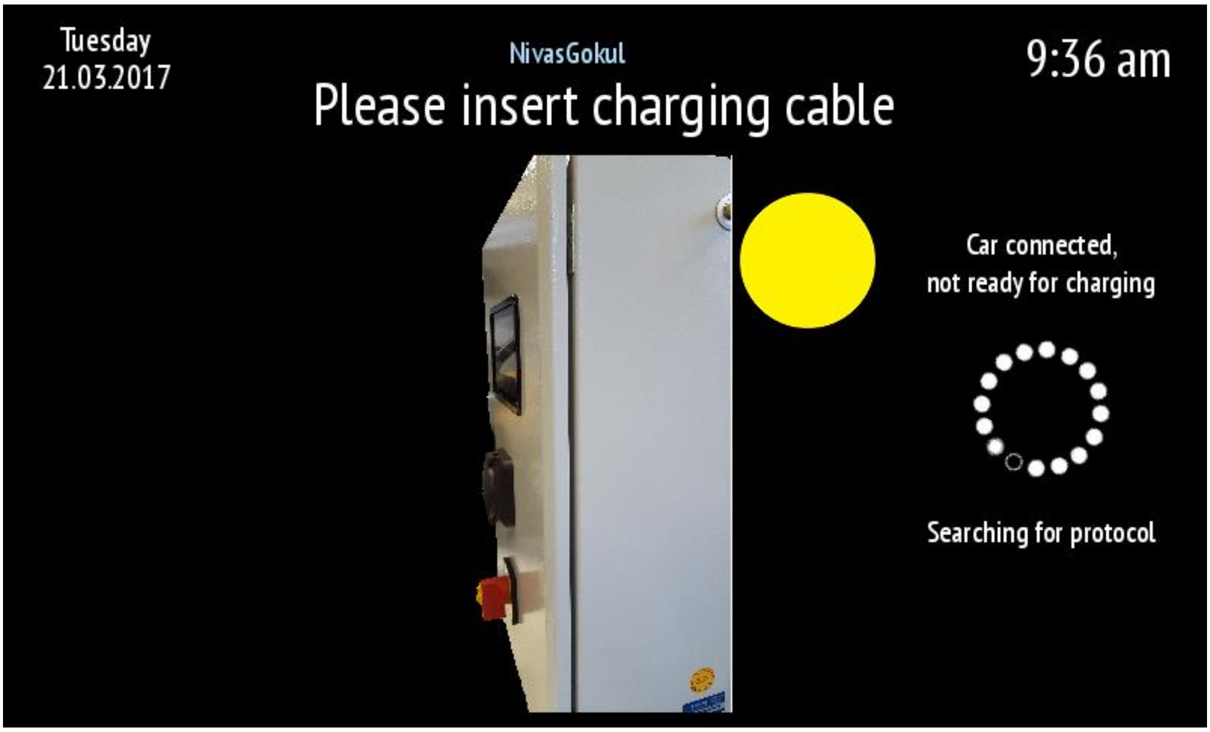
APPENDIX 5

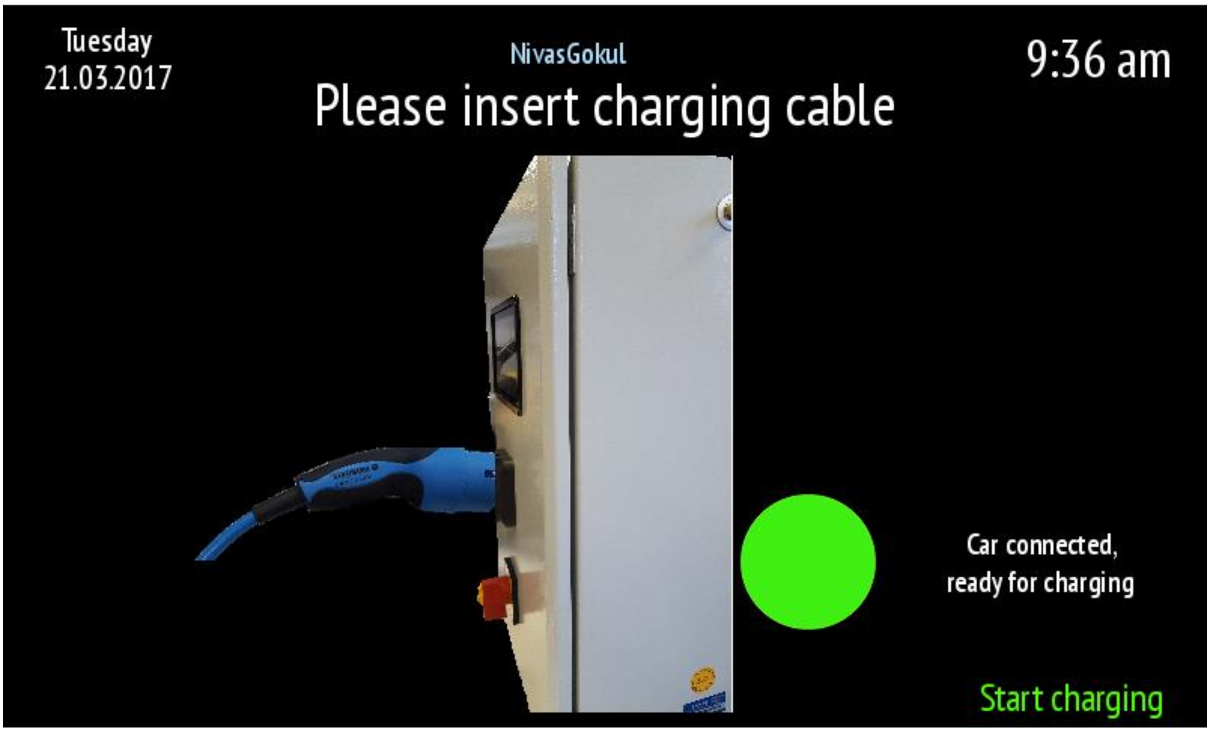
The screens from the HMI created using EB GUIDE



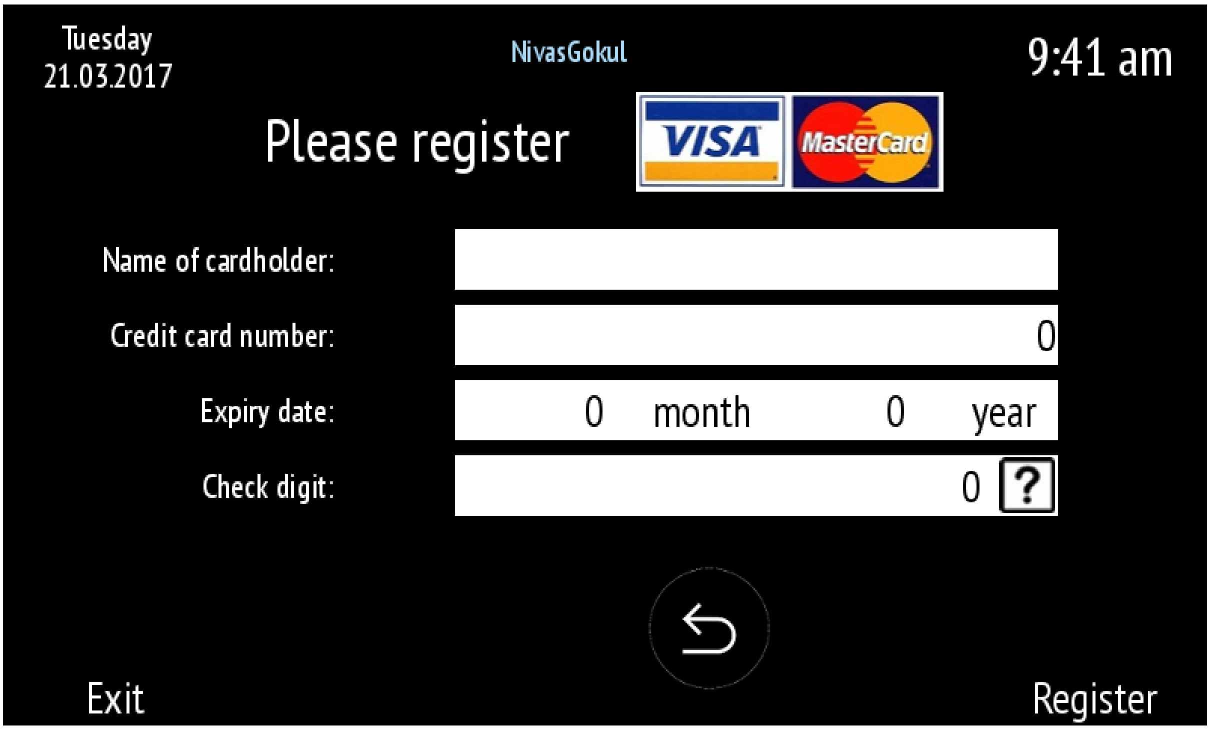



Tuesday
21.03.2017
NivasGokul
Please insert charging cable
9:35 am
Car not connected

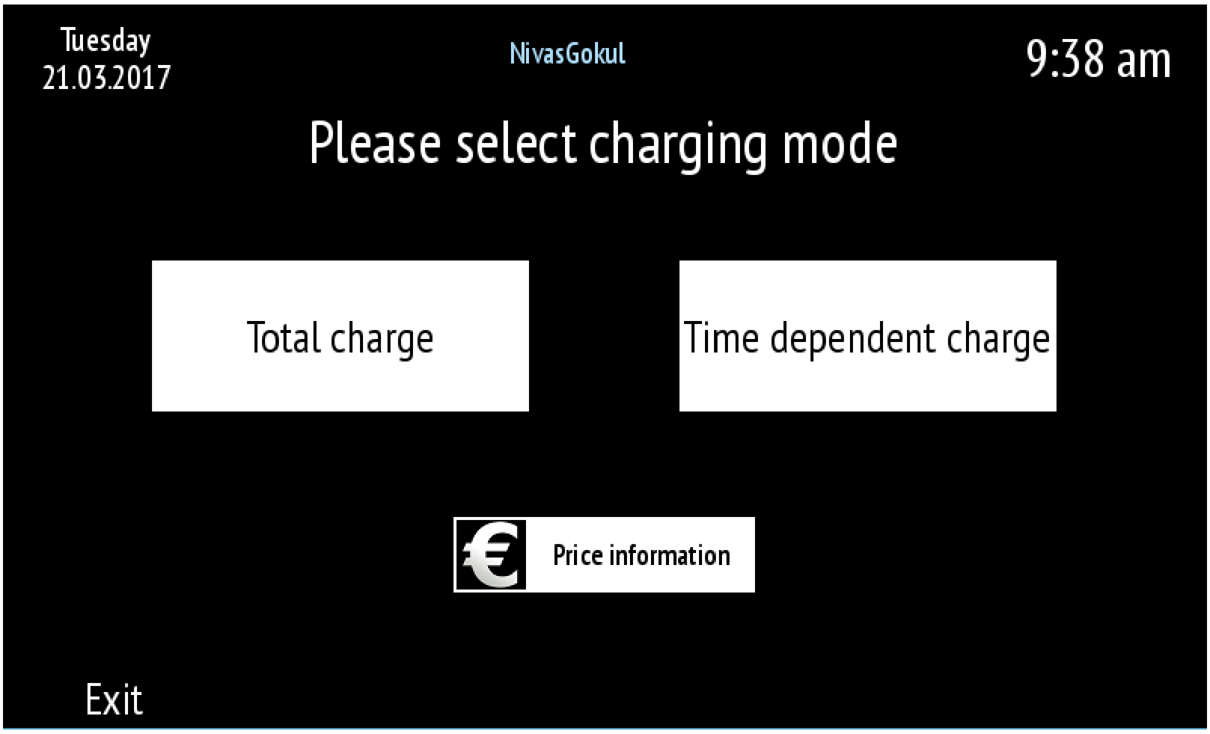

Tuesday
21.03.2017
NivasGokul
Please insert charging cable
9:36 am
Car connected,
not ready for charging
Searching for protocol

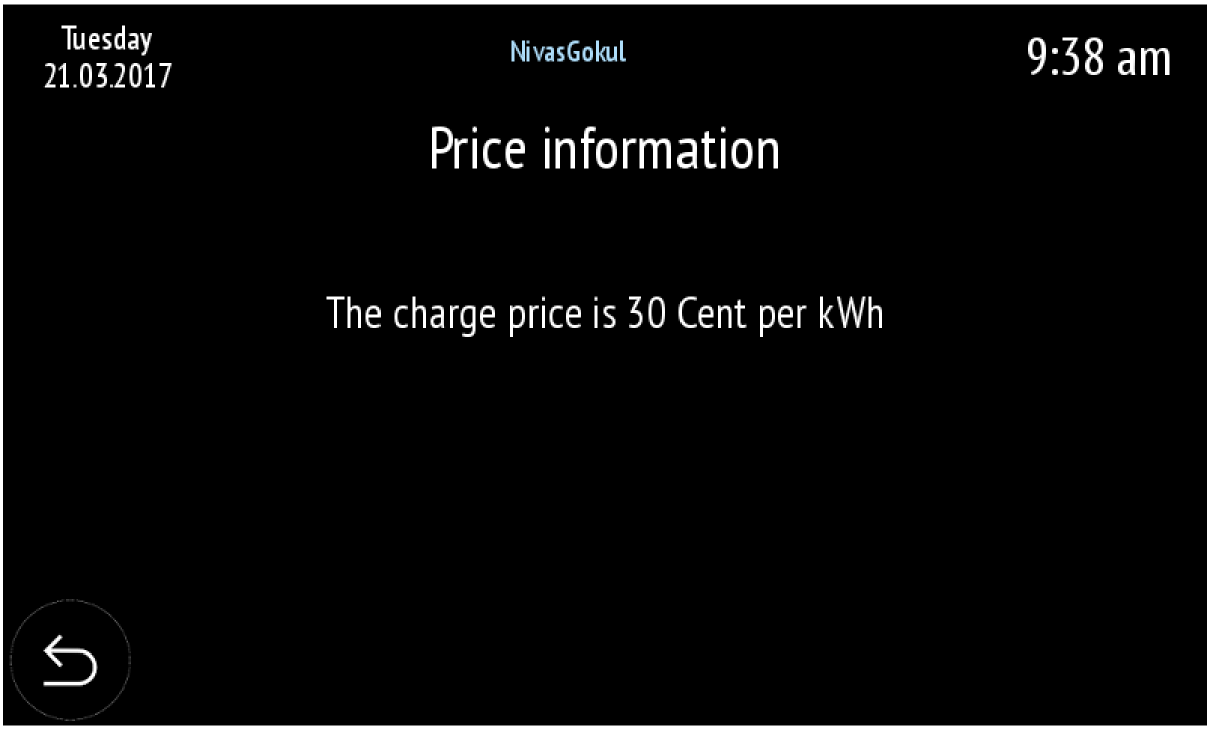

Tuesday
21.03.2017
NivasGokul
Please insert charging cable
9:36 am
Car connected,
ready for charging
Start charging

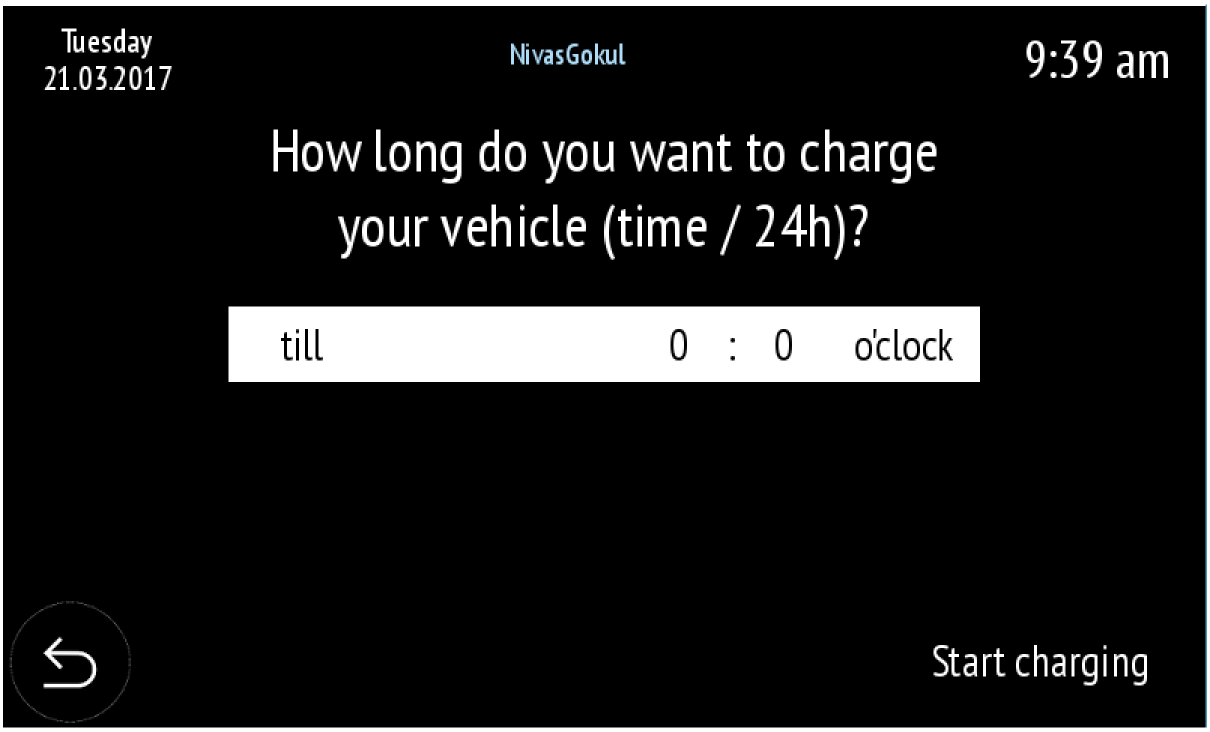

Tuesday
21.03.2017
NivasGokul
9:37 am
Please sign in
User name:
nego
Password:
123
You do not have a user account?
Then please register here
Exit
Sign in

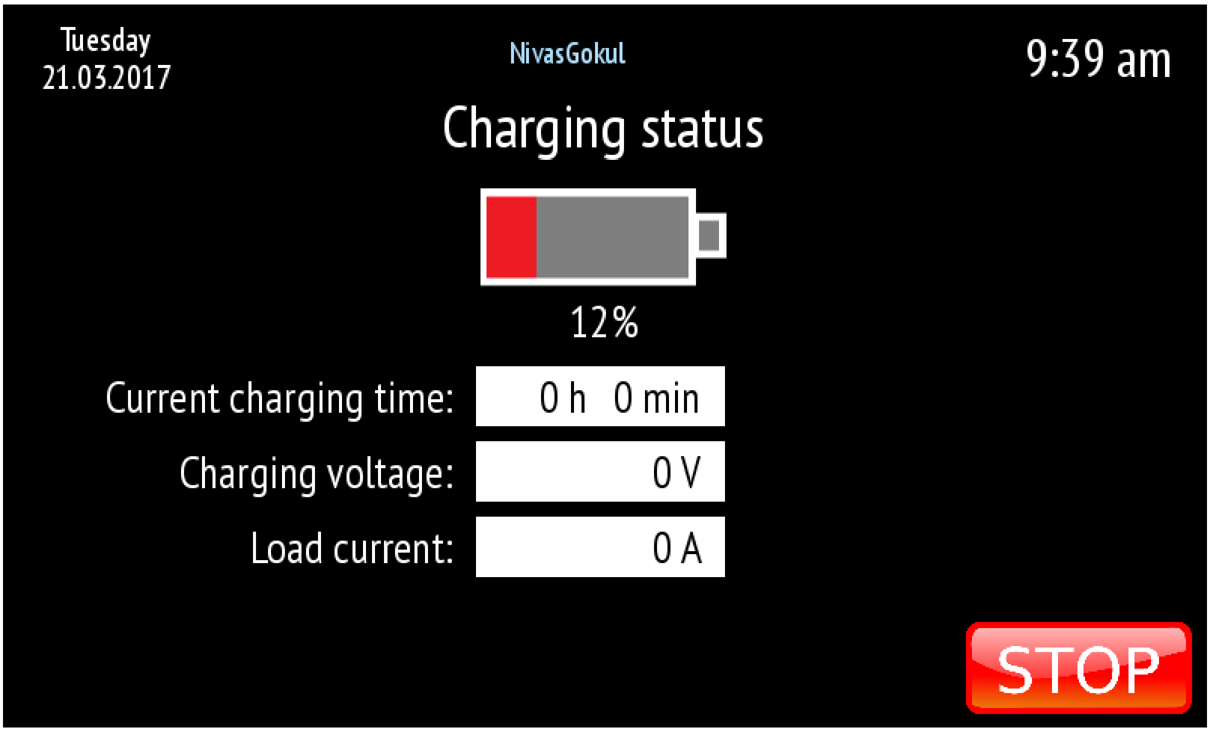

Tuesday
21.03.2017
NivasGokul
9:41 am
Please register
VISA
MasterCard
Name of cardholder:
Credit card number:
0
Expiry date:
0 month 0 year
Check digit:
0 ?
Exit
Register

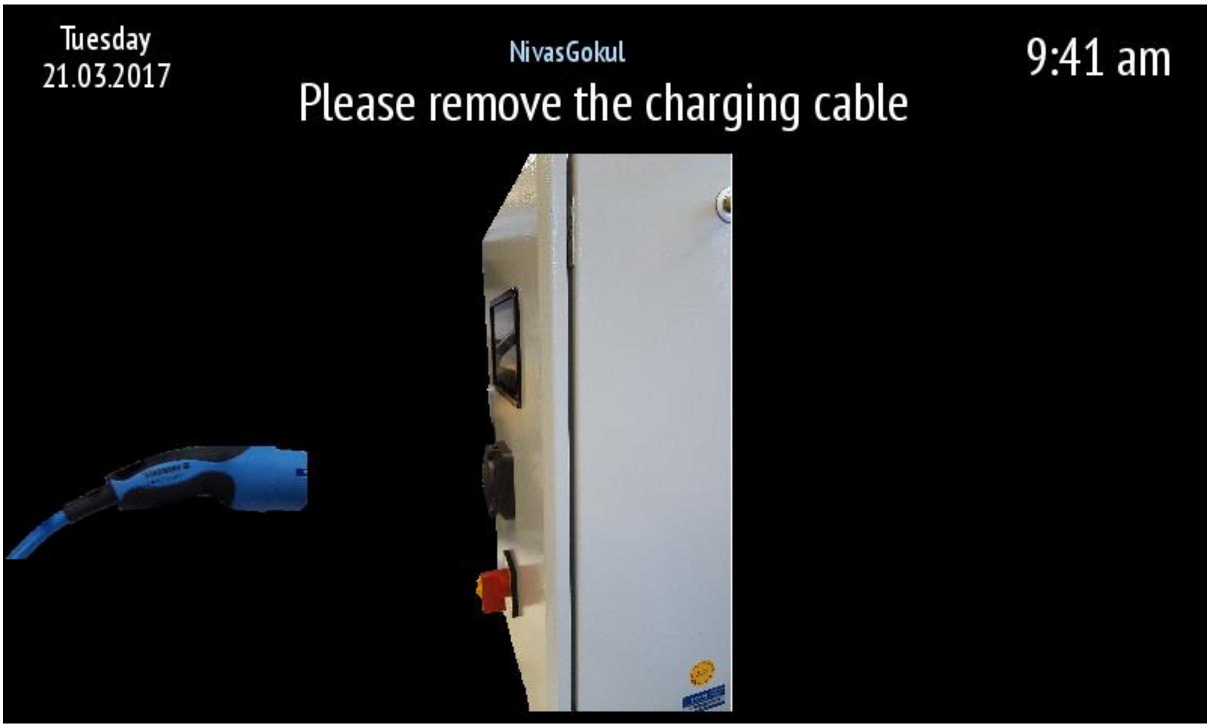

Tuesday
21.03.2017
NivasGokul
9:42 am
Here you can find the check digit
AUTHORIZED SIGNATURE
Not Valid Unless Signed
321
CARD


Tuesday
21.03.2017
NivasGokul
9:38 am
Please select charging mode
Total charge
Time dependent charge
Price information
Exit


Tuesday
21.03.2017
NivasGokul
9:38 am
Price information
The charge price is 30 Cent per kWh


Tuesday
21.03.2017
NivasGokul
9:39 am
How long do you want to charge your vehicle (time / 24h)?
till 0 : 0 o'clock
Start charging


Tuesday
21.03.2017
NivasGokul
9:39 am
Charging status
12%
Current charging time: 0 h 0 min
Charging voltage: 0 V
Load current: 0 A
STOP


Tuesday
21.03.2017
NivasGokul
Please remove the charging cable
9:41 am


Tuesday
21.03.2017
9:43 am