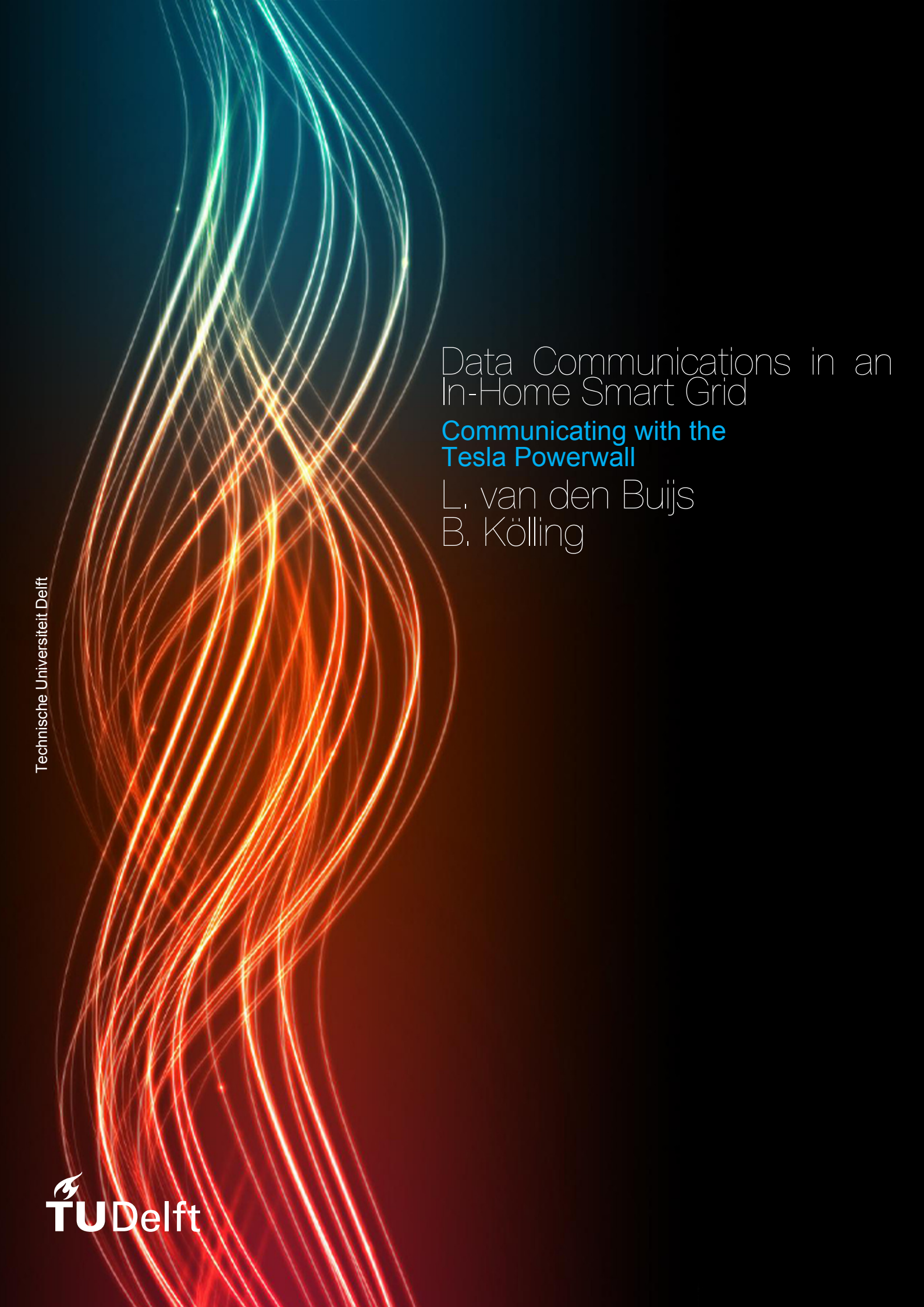


Data Communications in an In-Home Smart Grid

Communicating with the Tesla Powerwall

L. van den Buijs
B. Kölling

Technische Universiteit Delft

TU Delft

Data Communications in an In-Home Smart Grid

Communicating with the Tesla Powerwall

June 15, 2016

by

L. van den Buijs
B. Kölling

4155947
4228219

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended on Thursday June 30, 2016 at 11:00 AM.

Supervisors:	Prof. dr. eng. P. Bauer	TU Delft
	Dr. ir. L.M. Ramirez Elizondo	TU Delft
Thesis committee:	Dr. ir. I.E. Lager,	TU Delft
	Dr. ir. L.M. Ramirez Elizondo	TU Delft

Some of the information contained within this Thesis has been redacted due to confidentiality.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.
Cover Source: www.freehdwallpapers.com

TU Delft

Abstract

More insight and control is warranted over an in-home energy management system, consisting in our case mainly of a Tesla Powerwall and a SolarEdge Inverter, in order to allow for further research and development on the possibilities within the field. Not much is known yet on the Tesla Powerwall and its communication, as they aren't widely available yet and documentation is lacking. In order to achieve this insight, the communicated signals within the system have to read, analysed and used for possible charging commands. Software has been written on a single board computer, which establishes a connection, logs the messages send, displays the retrieved information on a locally hosted webpage and computes a charging command based on the retrieved information and entered user preferences. Even though the analysis of the logged communication signals was lacking due to the Powerwall not initializing, the achieved results should offer a good foundation for further research on the Tesla Powerwall and in-home energy management systems.

L. van den Buijs
B. Kölling
Delft, June 2016

Acknowledgements

First we would like to thank our supervisors, Pavol Bauer and Laura Ramirez Elizondo. We also would like to thank Laurens Mackay, PhD student from the DC systems, Energy Conversion and Storage department, for helping us when we had any questions. Furthermore, we would like to thank PhD student Victor Vega Garita and the technicians from the department, Joris Koeners and Bart Roodenburg for their support during this project.

Contents

1

Introduction

This chapter will both introduce the subject of in-home energy management with the Tesla Powerwall, as well as introduce the assignment the Bachelor Graduation Group has received. First, it will clarify the subject matter and the goals of the group. After that, the division of tasks between the subgroups and the assignment of the communications subgroup will be further elaborated upon. Additionally, the structure of the thesis will be discussed.

1.1. Background

Over the past years, the use of photovoltaic (PV) systems has grown significantly. In the past decade the price of a PV system declined by 75%, and the annual PV market volume has multiplied by over 40 times[1][2].

Even though this is great for the environment, pressure on the stability of the grid is growing because of it. In the current situation the surplus energy generated via solar panels is fed back into the central electrical grid in exchange for a financial compensation. However, all these injections of extra energy are difficult to predict, for they are not solely dependent on the weather and the time of the day, but also on the energy consumption of households, and on the number of households with installed solar panels within a district. This creates an increasing load on the management of the electrical grid, resulting in the rejection of this surplus solar energy by several energy companies. A great example of overgeneration is happening in California; an continuing increase in the generation of solar power, has resulted in an extra 13000 MW that has to be generated in merely 3 hours time. This has resulted in the so-called 'Duck chart', as can be seen from figure 1.1[3]. In anticipation of similar problems, The Netherlands will stop financially compensating feed-in starting in 2020 [4].

Electrical Energy Storage for consumers could help to alleviate these problems. On April 2015 Elon Musk announced this solution, as Tesla's Powerwall. The Powerwall is essentially a consumer use lithium-ion battery pack capable of storing 7 kWh of energy, with the main intention to store the surplus energy harvested by solar panels [6]. The energy can then be used when less energy is generated than used by the household. A commercial use version of the Powerwall has also been released, the Tesla Powerpack, which is a theoretical infinitely scalable version of the Powerwall, thus being able to store enough energy to power a complete industrial-level building [7]. And Tesla is not the only company doing this; next to several start-ups that want to introduce their own battery system, Mercedes-Benz introduced their own Personal Power Pack [8]. These home battery solutions could help solve the challenges resulting from an increase of personal sustainable energy generation, and also opens up new possibilities for research.

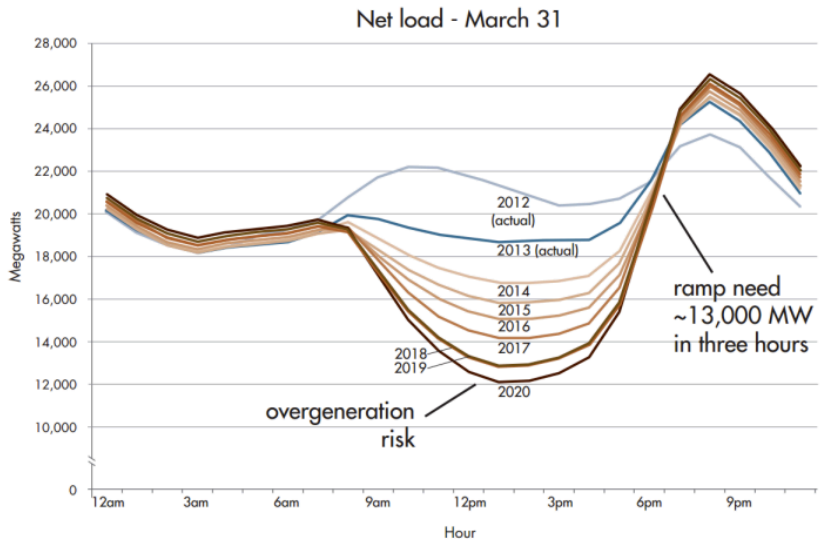


Figure 1.1: The California 'Duck Chart', showing an increasing problem with a growing amount of installed PV systems in California. Source: CAISO 2013[5]

1.2. Group Assignment

Eneco has supplied the TU Delft with a test setup with both a Tesla Powerwall and an accompanying SolarEdge Inverter. The Bachelor group was tasked to test the capabilities of the Powerwall and prepare cases for future research. The group split into three groups each with their own sub-assignment in order to tackle the challenges. The collected efforts of the three subgroups will result in a better understanding of the Tesla Powerwall System and will provide a base for implementing other and more advanced systems.

The first subgroup (PV emulation) is tasked with emulating a photovoltaic array which is to be connected to the Tesla Powerwall. Additionally, they will create a simulation of the array. The ability to emulate the photovoltaic array connected to the test setup offers the possibility of a realistic, but controllable, test bench.

The second subgroup (DC in-home grid) focuses on the design of a DC in-home grid. This includes an energy storage device in the form of a battery. The product will be a simulation which enables the analyses of the behaviour of a DC in-home grid.

The third, and final, subgroup (Communications) look into the communication within the supplied test setup. The aim will be to read the communicated data and to offer more control and insight on the energy management system.

In figure 1.2 a simplified overview of the provided setup is depicted, with in it the focus of each subgroup visually depicted.

1.3. Communications Subgroup Assignment

This thesis will focus on the objectives set for and results of the communication-oriented subgroup. Before listing the subgroup goals, the state-of-the-art analysis, on which they are based, is discussed.

1.3.1. State-of-the-art Analysis

Research has been done in controlling microgrids on a neighborhood where storage solutions are shared [9]. Methods have been proposed to for distributed control of a microgrid by letting local controllers communicate with each other and the central grid [10]. Full control by a connected system of the in-home energy flow can result in cost-reduction for the consumer [11].

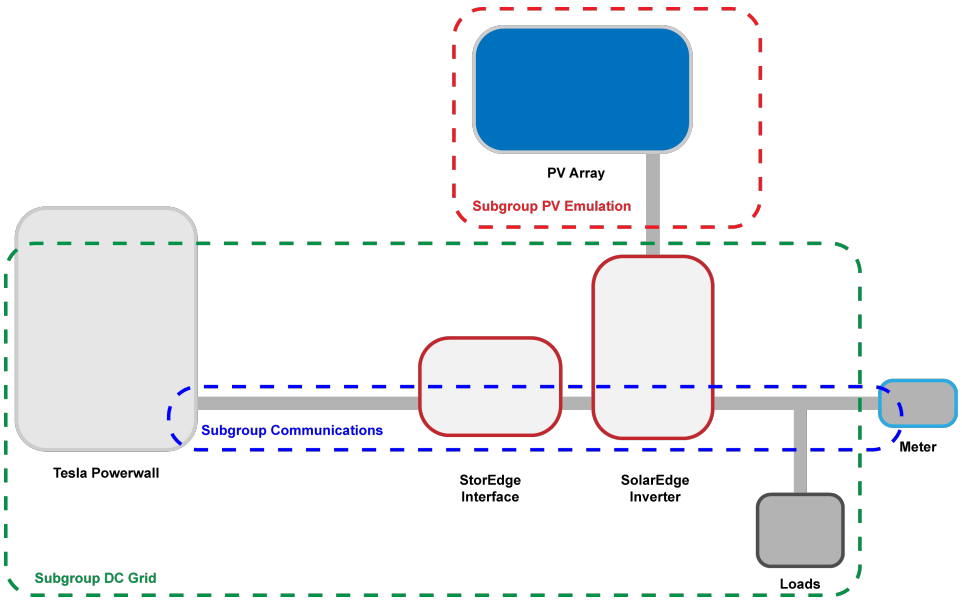


Figure 1.2: Overview of the three subgroups in the project and their focus in this project.

A recurring assumption in previous studies is that information about the state of the distributed generators, storage solutions and loads is available and they are controllable to a certain degree. In-home solutions available on the market at this time like the SolarEdge system offer very little interconnectivity. Some information can already be reported to a central server, however decisions about whether or not to store energy or and how much to use from and supply to the grid are done by the system in the home and are not influenced by outside information.

Electrical energy can't be stored efficiently enough yet, because of this the energy price is directly dependent on fluctuations in supply shortages [12]. Due to this rapid change in prices, it has become attractive for larger companies to dedicate departments to saving money on their energy costs [13]. Seeing the variation in the prices and the relatively low costs of solar electricity generation has prompted many companies and people to generate a large amount of their energy themselves [14]. Germany serves as an interesting test case, as high subsidies and feed-in tariffs have encouraged many households to install their own solar panels [15]. Several of the challenges society is faced with regarding the future grid can be observed. One of which is the cost that come with it. Because of the increase of variation in the demand of energy from the grid energy prices have become very volatile. The German government has decided to scale back the feed-in tariffs [16].

Another important challenge the increase of personal power plants warrants is the stress that it brings to the central grid. In order to balance the system the power delivered by the central power plants has to be adjusted. But if a local outage or a voltage spike or some other grid disturbance occurs, protective circuitry quickly shuts down the photovoltaics' inverters. And that in turn can lead to cascading system wide instabilities. Germany has invested millions in smarter systems to cope with this problem [14]. These personal grids have to be connected in an intelligent way in order to ensure an affordable and stable energy supply. This newly designed network would have to be able to work with all of these sudden changes. In order to allow for this to work it is essential that the all of the relevant information is retrieved in real time. Most likely it will result in a change in role of the central energy grid.

Intelligent in-home energy management systems with the help of the Tesla Powerwall, should be able to tackle these central issues. Before these issues can be solved however, great insight into the possibilities of the in-home energy systems has to be acquired.

Information about the in-home energy network is becoming more and more accessible due to *smart meters*. These meters supply information to the user through a display, website or smartphone application. They offer very little actual control over the network.

The setup supplied by Eneco includes smart meter. Additionally, the inverter is currently able to control

the behaviour of the Tesla Powerwall. The current system is severely limited though. Not only is the documentation available on the matter severely lacking, but the control appears to be really limited. Furthermore, only a selection of the status information in displayed.

In order to allow for future research and development of solutions to the discussed challenges, a product has to be developed which looks into the possibilities of the provided setup with the Tesla Powerwall and the SolarEdge Inverter. It has to retrieve the available status information and provide the basis for future control-oriented computations.

1.3.2. Goals

The subgroup is tasked with researching and testing the possibilities of communication within the provided setup. The product which is to be designed is thus not meant to be consumer-ready. The final product has to give insight into the capabilities of the current communication lines and demonstrate possible uses for it.

Based on this description the following list of main goals have been established:

1. Retrieve and read the information communicated over the communication lines
2. Display the retrieved information
3. Allow for user preferences
4. Compute and send (dis)charging commands

Complete these goals would lay the groundwork for future research and development of applications. Further requirements and limitations of the to be designed product are listed in the Programme of Requirements (chapter 2).

1.4. Structure of the Thesis

Firstly the Program of Requirements is discussed in chapter 2. In this chapter several assumptions for the project are made and a clear overview is given of the capabilities and other requirements of the final product. Then the system is described and assumptions for the project are clarified in chapter 3. Next, the design process is discussed in chapter 4. Here the choices made to satisfy the program of Requirements as described. After that the implementation and validation of the prototype is described in chapter 5. In chapter 6 the results of the implementation are discussed. Finally chapter 7 includes the conclusions from the project and also includes recommendations for future work. Finally in the Appendix a guide on how to run the software and the code is included.

2

Programme of Requirements

In the original setup the Tesla Powerwall is connected to the grid and PV panels via the SolarEdge systems. The components of the system communicate via a serial line. The Tesla Powerwall sends out its status specifications and this information is in turn used by the SolarEdge Inverter to control the in-home grid. The product which is to be designed should improve on the original setup by combining the status information communicated over the serial lines and preferences of the user to derive charging instructions for the Powerwall. The retrieved information has to be easily available for the user. The product is meant to be used for further research into the current and future capabilities of the Tesla Powerwall and isn't meant to be consumer-ready.

2.1. Functional requirements

1. The system must be able to operate with the Tesla Powerwall and the SolarEdge systems
2. The status information of the Powerwall and the smart meter have to be displayed
3. The display has to be refreshed automatically in order to show the current information
4. The system must be able to send a controlling command
5. The system must be able to operate it's key tasks without an Internet connection
6. The system has to be able to compute a controlling command based on the retrieved data
7. The system must allow for user input
8. The system must be able to use user configurable preferences in its control
9. The system should be adaptable to a newly designed inverter

2.2. Ecological embedding in the environment

1. If the system fails, the default design should be able to continue working
2. The computation must take the durability of the lithium-ion battery in the Tesla Powerwall into account

2.3. System requirements

1. The system must be easily installable within the original setup
2. The user preferences shouldn't be able to overrule the safety regulations

3

System Description

This chapter will elaborate on the current setup. To do so, the chapter will first discuss the complete system with a more zoomed in focus on the communication lines following after that. Additionally, assumptions and definitions used throughout the thesis will be discussed.

3.1. Current System

The energy storage system is situated at the Electrical DC systems, Energy conversion & Storage department (DCE&S) on the TU Delft Campus. The system consists of several parts: The Tesla Powerwall, the SolarEdge Inverter, the SolarEdge StorEdge interface and a Wattnode® Meter. An overview of the system can be seen in figure 3.1.

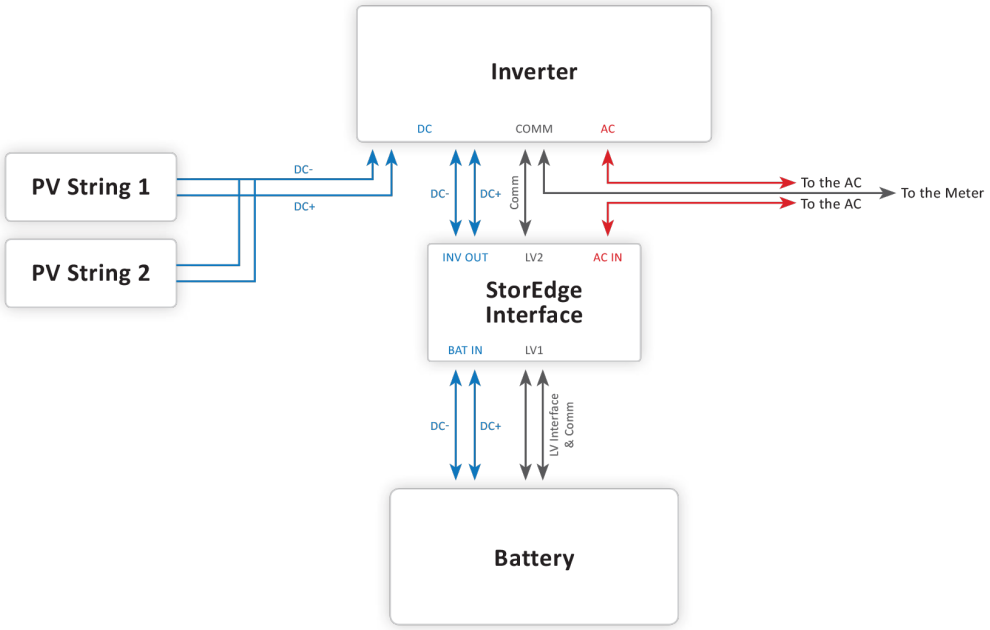


Figure 3.1: Overview of the connections between the Battery, the Inverter, StorEdge Interface, Meter and PV strings [17].

Storage of electrical energy is handled by the Tesla Powerwall. The Powerwall is a 7 kWh Lithium-Ion battery pack created for domestic use [Redacted]. It is normally mounted on a wall or other vertical surface. It is charged and discharged over a 350-450 volt DC line. It has two 12 volt inputs, one to

power the thermal management system and one to power the logic. For communication it has two RS-485 connections over which it communicates with the Modbus protocol, the details of which shall be discussed in sections 3.2.2 and 3.2.3 respectively.

The DC line from the Powerwall goes into the SolarEdge StorEdge Interface [17]. Here there are fuses located and the 12 volt for the thermal and logic of the Powerwall is generated. The StorEdge is used for safe decoupling during maintenance or emergencies.

The SolarEdge Inverter [18] is the central point in this system. The inverter collects data from the other systems and controls the flow of power. It uses the data from a meter to limit the feed-in to the grid if requested from the user. It has a connection to the Internet, with which it can send information to an online portal where the user can monitor the system.

The connection from the AC network to the grid is done via the meter. In the current system a Wattnode Modbus meter [19] is installed. This meter provides the Inverter with information about the incoming or outgoing power over the RS-485 line.

3.2. Communication Overview

The Tesla Powerwall uses a 12 volt circuit interface for its communication and control. It consists of several parts: an enable line, Thermal Power, Logic Power and Communications. An overview of the connections can be seen in figure 3.2.

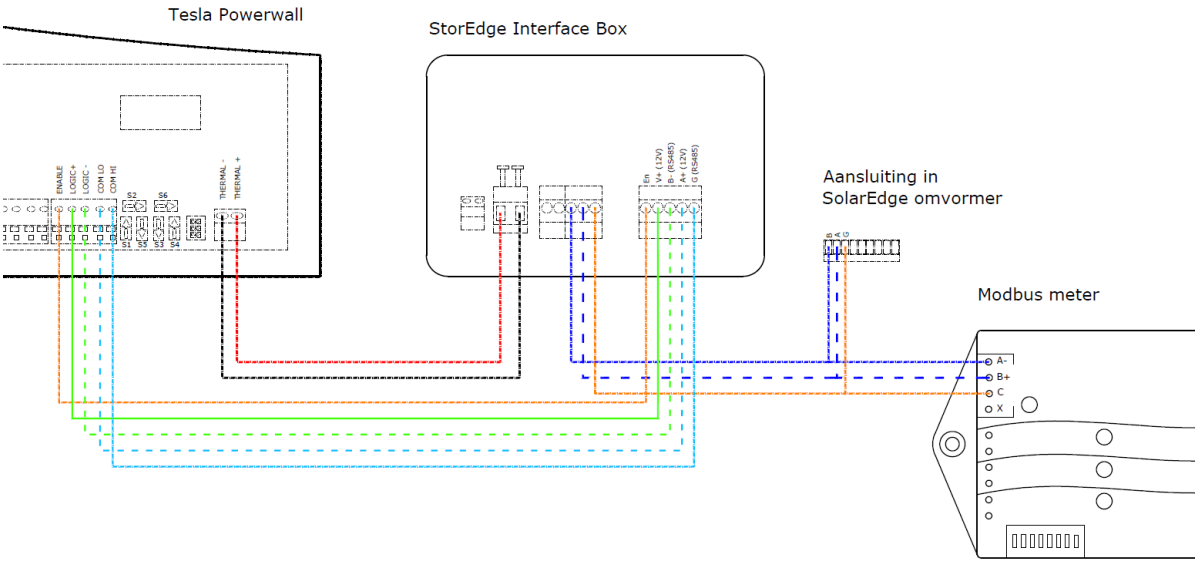


Figure 3.2: The RS-485 connections between the Battery, the Inverter, StorEdge Interface and Meter. Source: [20]

The switches seen in figure 3.2 on the Tesla Powerwall are only altered during the installation of the system and the required configuration depends on the system.

The enable line signals whether or not the Powerwall should be enabled at all. Once enabled the battery shall operate in accordance within its programed limits. In the enabled state, the Powerwall shall be capable of heating the battery when needed. The enable line shall be broken with a switch when the bottom access panel of the battery system is removed, ensuring no DC link current into or out of the system.

3.2.1. Logic and Thermal Power

The Logic+ and Logic- lines power the logic interface. The logic interface shall consume less than 3 watt at 12 volt in all operating states. The Thermal+ and Thermal- are used for the thermal power. The thermal interface shall consume less than 50 W at 12 V. Both terminals are mechanically incompatible. In all cases, the Powerwall will attempt to maintain optimal thermal conditions to optimize system efficiency and battery life. In cases where thermal power is not provided, and the Powerwall cannot manage its thermal condition, its power capability may be reduced. 12 V power may also be provided with a single connection, in lieu of two separate connections. The single connection shall provide the combined power consumption for both the thermal and logic interface, i.e. 53 W at 12 V. It shall be possible to daisy chain the 12 V Logic Power for up to eight Powerwalls. The 12 V Thermal power connection shall be provided separately for each Powerwall and may not be daisy-chained [Redacted].

3.2.2. RS-485

RS-485, or *TIA-485-A*, is a standard for serial communication lines [21]. It can be used for a linear bus topology with only two wires and a ground reference. One of the data pins is called the A, D+, or non-inverting pin. The other is called the B, D- or the inverting pin. The two ends of the bus have to be terminated for optimal operation. It can also be used in full duplex mode with four wires.

In the Tesla Powerwall the two communication signals are specified as DATA+ and DATA- with a third ground wire, that also functions as LOGIC-, where LOGIC is the 12 volt supply for the internal logic of the Powerwall [Redacted]. The symbol rate specified is 9600 baud with no parity bits.

3.2.3. Modbus RTU

Modbus is a protocol developed by Modicon (Currently Schneider Electric) in 1979. Communication within this protocol happens between a master/client and slave/server or multiple slaves/servers. The protocol has different versions, for example *Modbus TCP/IP* and *Modbus Remote Terminal Unit (RTU)*. It is often used in Supervisory Control And Data Acquisition (SCADA) systems.

In table 3.1 the Modbus RTU protocol is displayed in the 7-layer Open Systems Interconnection (OSI) model [22]. Modbus only occupies the Application and the Data Link Layer.

7.	Application Layer	Modbus Application Protocol
6.	Presentation Layer	Empty
5.	Session Layer	Empty
4.	Transport Layer	Empty
3.	Network Layer	Empty
2.	Data Link Layer	Modbus Serial Line Protocol
1.	Physical Layer	RS-485

Table 3.1: The OSI model for the Modbus RTU Protocol.

A client initializes the communication by sending a message to the bus. Each server receives it but only the server with the correct address processes the message and can reply to it. In table 3.2 the message format that Modbus RTU uses is shown [23]. A message always starts and ends with at least 3.5 character times of silence.

START	ADDRESS	FUNCTION	DATA	CRC CHECK	END
Silent	8 bits	8 bits	n x 8 bits	16 bits	Silent

Table 3.2: The Modbus message format.

The Tesla Powerwall has the address *0x18* (24) in the Modbus connection [24]. Both a message from and to the Powerwall will have the address 0x18. The function code determines what the slave should do with the data it gets from the master. For example it can read or write bits or registers. At the end

System Parameter	Data Type	Scale	Units
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX

Table 3.3: Contents of communicated Powerwall registers [Redacted]

a Cyclic Redundancy Check (CRC) is added, specifically CRC-16-ANSI. A CRC is an error detecting code based on division by a polynomial.

The Modbus protocol uses individual bits and registers to store data. There are two types of bits: *Input Bits* that can't be altered but can only be read with a Modbus request and *Coils* that can be read and written to. The structure of the registers is similar. *Input Registers* are read-only registers in Modbus and *Holding Registers* are also written to.

3.2.4. Communicated Variables

The Tesla Powerwall communicates, upon request, the contents of a list of registers. The contents of these registers can be found in table 3.3.

The Wattnode Modbus meter functions in the same manner as the Powerwall, when it comes to the communication lines. The meter communicates a set of registers, the content of which can vary. The meter itself is capable of measuring a wide variety of variables. Furthermore, it allows the user to remap the register address in order to retrieve the variables wanted [19]. In the documentation available on the current system however, the information on which registers are requested during communication aren't described. The contents of the communicated registers within the current setup will be derived by analysing the signals. The results of which will be discussed in section 5.4.1.

3.3. Assumptions and Definitions

The assumption is made that without interference the Tesla Powerwall and inverter will operate with a focus on self-consumption, as the current system does now. When in self-consumption the Powerwall attempts to optimize the energy retrieved from the attached solar panels. This results in the storing surplus energy rather than feeding it back into the central grid. Additionally, the use of the central grid is minimized by attempting to supply the complete demand via the solar panels and stored energy.

One of the goals described in section 1.3.2 is that the designed product should be able to send out

a charging command. At the moment it isn't clear where this signal should be send to be processed. It is possible that the Tesla Powerwall or the SolarEdge Inverter are capable of this process, but the available documentation don't provide enough evidence for this. It could also be possible to design a controlling DC-DC converter with the described capabilities. For now, the element processing the charging command will be defined as the *Computation Module*, whether this resides within an existing component or a still to be designed element.

4

Design Process

In this chapter the choices made while designing the product are discussed. The program of Requirements forms the basis on which the choices in this chapter are based. Firstly the method with which data is retrieved from the Tesla Powerwall is explained. Then the choices that were made to determine how the data should be displayed to the user are described. Furthermore, the design of the computation process and how the product is to be integrated with the existing system are discussed. Lastly, the decision for an early test setup is expanded upon.

4.1. Retrieving System Information

From the information that was available on the Tesla Powerwall [Redacted] it was clear it has one method of communicating with other devices in this configuration, an RS-485 connection with the Modbus RTU protocol. The symbol rate specified is 9600 baud with no parity bits. This section begins with information regarding RS-485 and Modbus, then it describes the choices made for retrieving the necessary information from the system.

For development purposes several Odroid C-1 Single Board Computers [25] have been made available. These computers run a Linux distribution as their operating system. They are versatile and powerful enough for this project. The design will not specifically be for this computer but will be able to run on any Linux-powered device with the required I/O capabilities.

To communicate over RS-485 an USB adapter is needed. For this there are a lot of options on the market. The *USB-COM485-PLUS1* [26] from Future Technology Devices International Ltd has proven to work in similar situations. It has a USB type-B port on one side and a DE-9P D-subminiature connector on the other. It also works out-of-the-box without installing any drivers in Linux so it is chosen.

The adapter is used to connect to the two communication lines of the Powerwall, DATA+ and DATA- with a third ground wire.

4.1.1. Libraries

There are multiple libraries available suitable for creating a program that uses Modbus. In order to come to a conclusion on which library to use, several options have been taken into consideration.

For embedded systems there is *FreeMODBUS*, a C library for somewhat specific applications. A second option is the *libmodbus* C library. The support for this library seems good, as the source is available on GitHub and there is enough documentation to aid in bug fixing. Libmodbus has also been used for a different Modbus application by the DCE&S department, thus there is some expertise available. An option when using Java is *jamod*. This library can be used for serial or IP Modbus implementations in Java applications. It seems to not be actively in development anymore. *FieldTalk*^{TM} is a library for C++, it has a lot of features and is well documented. It is however an expensive licence to purchase just for

this project. Lastly there is also a library for Python called *PyModbus*. The features seem similar to libmodbus, but it seems to not be actively in development.

In this subgroup a thing to consider is experience with different programming languages. C and C++ are both part of the Electrical Engineering curriculum. Knowledge about Java and Python is limited and learning it might be a hurdle when developing the product. An overview of the discussed advantages and disadvantages can be found in table 4.1.

Library	Language	Pro's	Cons
FreeMODBUS	C	C experience in team	Less Support
libmodbus	C	Actively developed Many examples C experience in team	
jamod	Java		No Java experience in team
FieldTalk™	C++	Professional Software C++ experience in team	Purchase required
PyModbus	Python	Python can be easy to learn	No active development Not widely implemented

Table 4.1: Libraries for Modbus and their Pro's and Cons

The choice was made to use **libmodbus** as the Modbus library considering the available support and the experience within the team. The software is thus written in C. There are several standards when it comes to this programming language, for this project C11 is used. This is the most recent standard with good support from compilers. Older standards can be inconvenient to use as they for example don't support inline commenting.

4.2. User Interface

There are multiple ways in which to implement the user interface. The user interface is meant to both display the system information and allow for user input, in order to meet requirements 2.1.2 and 2.1.7 respectively. The options which will be considered are an attached display, a local webpage and a smart-phone application.

A display attached to the communication module isn't a very user-friendly option, as it would only allow for a central location to display the information. A locally hosted webpage would solve this issue. With the correct IP information it would be possible to reach the information from anywhere within the local network without the use of any external system. One extra addition to that could be the use of the *multicast Domain Name System* (mDNS) [27] to allow the user to not enter an IP but enter a host name ending in *.local*. It is possible to take this further and develop a smart-phone application which makes use of the local network. The benefit of this proposed implementation is that allows for the display to be even more user friendly. An important downside, however, is that security and privacy become an issue as strangers shouldn't be able to alter the settings or retrieve the information hosted.

The choice was made for the locally hosted webpage, as the user-friendliness isn't worth the extra effort at the time to develop a smart-phone application and the attached display is to limited in its capabilities. Additionally, the benefit of locally hosting the webpage is that the system can operate fully without making use of the Internet, meeting requirement 2.1.5.

4.3. Control of System

As stated in requirements 2.1.4 and 2.1.6, the information derived from the Powerwall and the user interface are used to compute and send out a signal and control the system. The status information from the Tesla Powerwall and the Wattnode Modbus meter supply the current information. Combined with the user preferences, the computation protocol has to send out a charging command while taking safety and durability of the system into account. An option is to let the user specify certain thresholds

(maximum power, minimum State of Energy) in the system to modify its behaviour. This gives the user a lot of freedom but could let the system underperform when not configured correctly. Another option is to give the user the choice between a predetermined number of modes, each with their own purpose. This option seem preferable, it could give the user enough flexibility without compromising the effectiveness of the system and still meets requirement 2.1.8.

4.3.1. Modes

To define the different *modes* for the system, first the purposes the system should serve are determined. It's important to note that the modes which are to be implemented in the prototype have the goal to test and showcase the general possibilities of the communication module. They aren't necessarily the modes the consumer would receive as possible options.

Backup power in case of a black-out is a great use for a battery. Here the battery is in a quite passive state most of the time. The system needs to be ready to provide power when the grid fails.

Self-consumption is what the existing system is mostly designed for. Here the goal is to make the most of installed PV panels or other renewable energy systems. When feed-in is limited power generated is best stored for later use.

Smart Selling of energy can become a major factor on the energy market in the near future. When an abundance of solar or wind energy floods the market energy becomes cheap or even free. The system can use this information to then let the battery charge and when prices become higher again let it discharge. Here a connection to another server is required to monitor the energy price is needed.

Safety is an important factor to be considered within the control computation. The Powerwall shouldn't be forced to go beyond its limits or ignore the safety requirements of the Powerwall.

Durability has to be kept in mind when computing charging commands. Constantly forcing the battery-pack to charge or discharge will wear on the system.

Considering the aforementioned purposes the system could serve, a choice of modes is made. First of all, both *safety* and *durability* limitations are integrated within all of the modes in order to comply with requirements 2.2.2 and 2.3.2. Four modes will be implemented. **Backup** will maintain a high SOE so the system is prepared for a black-out. **Self-consumption** will let the system optimally use the renewable energy sources in the system. **Smart Selling** will use the energy prices to determine when to charge and when to discharge the battery. Finally **Hybrid** will be a mode where part of the capacity is reserved for backup power and the rest is dynamically sold to the grid by checking the energy prices.

4.3.2. Charging Command

The form of the charging command, which is essential a control signal, still has to be determined. The control signal won't influence the process directly, as it does in a traditional dynamic system. The signal forms a command which is to be processed by the *Computation Module*. The options to be considered are two implementations of *Bang-Bang Control* and *Proportional Control*.

Bang-Bang Control is a form of control with only two possible states. In case of the Powerwall these states would be charging and discharging at a fixed rate. The selected mode would have to determine an upper and lower limit to the state of energy. If the state of energy reaches the upper limit the Bang-Bang control would tell the Powerwall to discharge. Once the state of energy reaches its lower limit, the Powerwall will be told to charge again. An advantage of Bang-Bang control is that, in combination with the frequency of charging commands, it is quite easy to keep the state of energy within the determined limits. An important downside however is that the Powerwall would never be in rest and would always be actively charging or discharging. This would result in a large amount of unnecessary charging cycles, which would put a lot of stress on the capabilities of the battery pack conflicting with requirement 2.2.2. Furthermore, this manner of control could force in in-home energy grid to deliver power to the central grid. Not being able to control this in a more sophisticated manner serves limits the advantages of an in-home energy management system.

While the mentioned disadvantageous hold up when it is only possible to either send a charge or discharge command, there is another implementation of *Bang-Bang Control* possible. In section 3.3

the assumption is mentioned that the existing system will operate with a focus on self-consumption if no control command is send. Adding the option to not send a signal to the discussed Bang-Bang control opens up more possibilities. In practice, it would result in a charging command after a lower limit, defined by the selected mode, is reached and is switched back to self-consumption after the upper limit is reached. Virtually, the only reason to give a discharge command is when the *Smart Selling* is selected, the energy are relatively high and there is a surplus in energy available in the Powerwall. This implementation of Bang-Bang control would share the benefits of the former implementation, but would overcome the problem of the constant charging cycles.

Proportional Control offers a more elegant way of controlling the Powerwall. Proportional control would be achievable by communicating a charging factor. This factor, ranging from -1 to 1, would be multiplied with the maximum charging speed, as allowed by the Tesla Powerwall. The most important benefit of this system is that the Tesla Powerwall is free to focus on self-consumption, while it is more subtly steered according to the entered user preferences when needed. Moreover, it is possible to give more weight to a charging command if necessary, e.g. when changing from operating mode. A downside to this system is however, that it necessitates the controlling algorithm to be more complex. It wouldn't suffice to simple define an upper and lower limit, but it has to compute the necessary speed as well.

After considering both of the discussed options, the choice was made for the implementation of *Bang-Bang control* which allows for the possibility of not sending a command at all. The benefits of this choice are that it can easily ensure a certain state of energy within the Powerwall. Furthermore, not requiring a command to be send allows the system to meet requirement 2.2.1. Additionally, it prevents constant charging cycles while also keeping the demands on the computation protocol relatively simple.

4.4. System Integration

Requirement 2.1.1 states that the designed system to be easily integratable in the original set-up as developed by Tesla and SolarEdge. Furthermore, requirement 2.3.1 warrants the system to be easily installable. A choice has to be made where the system will be placed. The RS-485 bus is accessible from any point in the system, as it forms one single bus.

The Odroid can be plugged into existing connectors next to the wires that are already in place, in the inverter, StorEdge, Powerwall or meter. It is also possible to connect it to the wiring between the elements in the system. Then alterations have to be made to the existing wiring. The requirements, however, state that the product should be easily installable on the original system this isn't a wise choice.

RS-485 in normally implemented with *Unshielded Twisted Pair* (UTP) wires. This should be a requirement in the final product to minimize disturbances from outside electromagnetic sources. However when testing UTP wiring is not a requirement as reliability is less of an issue for these shorter tests.

4.5. Early Test Setup

Before connecting the designed communication module to the Tesla Powerwall, the communication system will have to be tested separately at first. This is important as unknown factors could cause unsuccessful test results when testing directly with the Powerwall.

One option to do this is by connecting two Odroids together, illustrated in figure 4.1. The USB to RS-485 adapters are used with both. Here one Odroid functions as the client and one functions a the server. The client requests data from the server, hosts the website and computes actions to be taken. The server responds to the queries and uses the command given by the client to alter the data, acting like a surrogate for a Powerwall system. Here communication signals from the Powerwall can be simulated in a controlled environment.

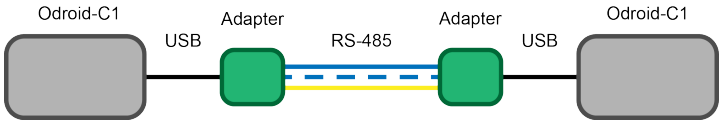


Figure 4.1: A schematic overview of the test setup with two Odroids. They are connected via RS-485.

Testing over Ethernet with the *Modbus TCP* protocol would be simpler. This leaves the adapters out of the picture and the focus can be on the software itself. Testing over RS-485 with the adapters however would mean an easier transition to testing with the actual Powerwall. The choice is made to connect the two Odroids with RS-485 during testing.

5

Prototype Implementation

In this chapter the methods used to implement the system designed in chapter 4 are discussed. An overview of the software is given first. Then the implementation is discussed per part of the software. Testing the prototype is also described in this chapter. Tests are done with the Powerwall and with that a test bench is constructed that emulates a Powerwall system.

5.1. Software Overview

The software required to implement the discussed requirements, consists of a multitude of functions. An overview of the software topology can be seen in figure 5.1. The figure describes the software run on the Odroid and as a whole is initiated by running the function *Main*.

The *Main* function calls *CSS Create* and *PHP Create*. These two functions create two files, *mystyle.css* and *test.php*. The file *mystyle.css* is the style sheet the website uses and doesn't change during the operation, so it only has to be created once. The file *test.php* contains the PHP code required for passing the user's input to the rest of the system.

Most importantly, *Main* initiates the function *Tesla Client*. *Tesla Client* opens the *Modbus* connection and retrieves the status information communicated over the *Modbus* connection by the Powerwall. The data is entered in array *Powerwall_Data[]*.

First *HTML Display* is called which is used to create *index.html*. *Index.html* receives both its code and the array *Powerwall_Data[]* from *HTML Display*. It uses the styles defined in *mystyle.css* to host the webpage. The webpage doesn't just display the values within *Powerwall_Data[]*, but it also allows for user input by making use of a form. The user selects a mode and triggers *test.php*.

Tesla Client initiates the required computation algorithm dictated by *Mode* and sends it *Powerwall_Data[]*. The computation algorithm computes the signal *Command* and sends it to the required location. *Tesla Client* loops until the users shuts down the program.

The code is included in appendix B.

5.2. User Interface

As discussed in the section 4.2, it is necessary to host a local webpage for both displaying the system information as well as serving as an interface for the user input. It is important to be able to update the content of the webpage, it is dealt with in a separate function, as discussed in section 5.1. As it is unnecessary to recreate the style sheet every few seconds, this function has been implemented within the main function. For the PHP code allowing for the user input, the same applies. An Apache HTTP server is used to host the webpage. How this is installed is explained in appendix A. In figure 5.2 a screenshot of the User Interface can be seen

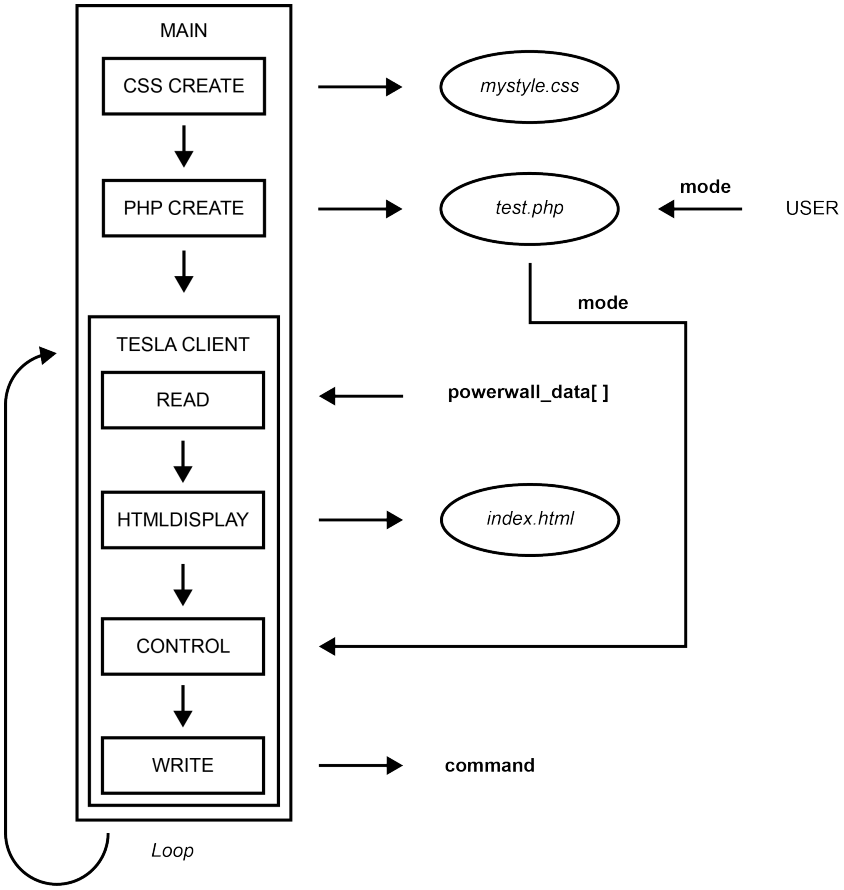


Figure 5.1: A schematic overview of the software implementation of the client.

5.2.1. Displaying Data

The easiest way to communicate the to be displayed values to the content-generating function is by making them argument for the function. As there are many variables which are worthwhile to be displayed, an array has been put in place. The data received via the Modbus protocol are placed within the array, after the reference to the array is used as the argument of the hosting function. Within the function a simple loop is used to print all of the data. Data is put into the files with the *fprintf()* function, an example from the function *htmldisplay()*, included in appendix B.6, is given here:

```
fprintf(fp, "<div id=\"header\">");
fprintf(fp, "<h1>Tesla Powerwall</h1>");
fprintf(fp, "</div>");
```

Additionally, an automated refresh command has been added to the webpage, thus meeting requirement 2.1.3. This way the webpage always states the current information and it isn't required for the user to keep refreshing manually. A refresh every 2 seconds in HTML is done like this:

```
<meta http-equiv="refresh" content="2" >
```

The function has been test in two different ways. Firstly, a set array of variables was entered to see if they were displayed correctly on the webpage. After concluding that no complications arose a second test was run. This time, however, the test variables increased slightly with every iteration of the script. Due to this addition it was possible to test whether or not the webpage refreshed itself with the updated data. The second test showed that the function worked as envisioned.

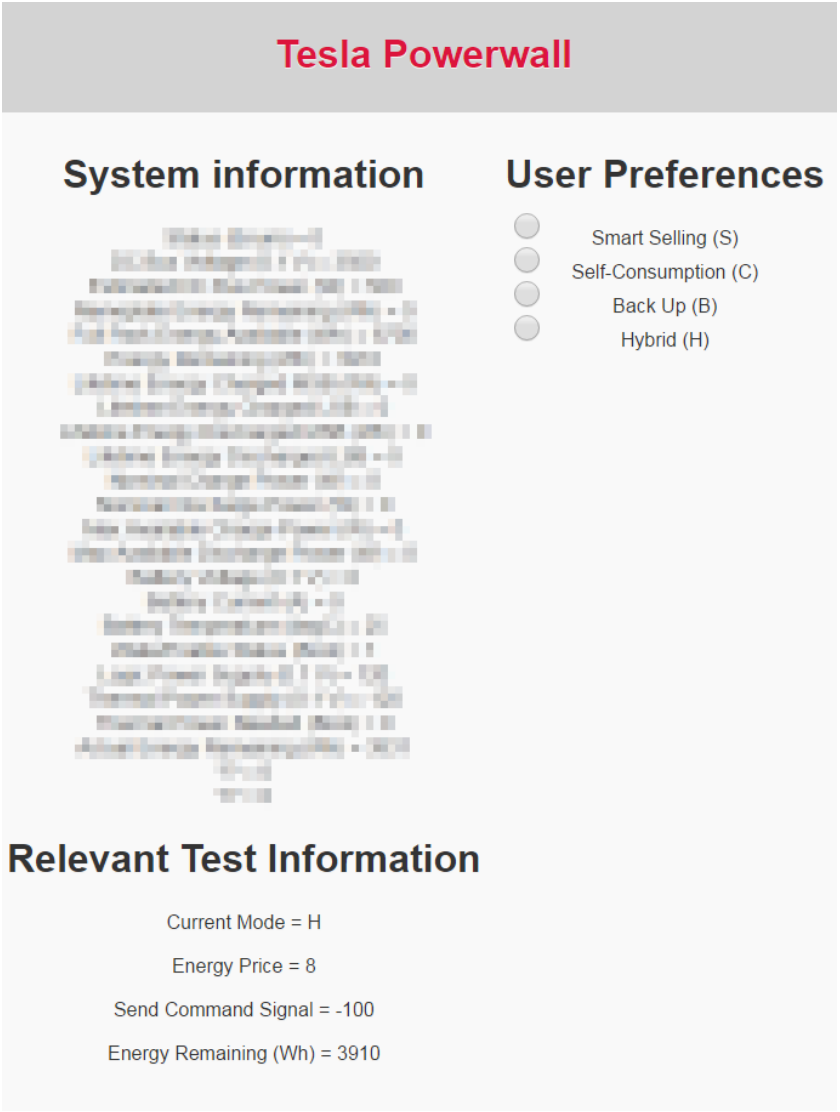


Figure 5.2: The designed User Interface displaying the Powerwall variables, relevant test values and an user input form

5.2.2. User Input

As can be seen in figure 5.2, within the webpage hosting function a form has been initialised. This results in several checkboxes to be displayed on the webpage. Selecting a mode activates PHP code which has to retrieve the information. As mentioned before the PHP code is initialized once and this is done in the main program.

The form on the webpage communicates a character value to the PHP code, depending on the selection made. *Test.php* requests the character associated with the submitted mode and writes the character *Mode* in *userpref.txt*. A separate function has been designed to read the character written in the text file and to draw conclusions based on the result.

The implemented protocol has been test by manually selecting various modes and checking both the text file as well as the function meant to read it. From the tests it became clear that the PHP code overwrites the text file with the new information whenever it is requested. Additionally, the protocol was able to read the contents from the text file correctly. To be certain the permissions for the files are correct, the *chmod* function is called. The implementation of the PHP code is done like this:

```
<?php
chmod("/var/www/html/userpref.txt", 0777);
$fp = 'userpref.txt';
```

```
file_put_contents (  $fp, $_GET["mode"]);
header("Location:{$_SERVER["HTTP_REFERER"]}");
?>
```

5.3. Testing Communication with two Odroids

In the section 4.5 the plans to run tests between two Odroids have been described. The test setup described in this section was created and used to test the required functionalities regarding communication.

5.3.1. Establishing a connection

In order to establish a connection, and thus testing the physical connection discussed above, standard client and server code, supplied with the *libmodbus* library, was run. After it was clear that the system allowed for a connection, the code for the proposed system was developed.

Establishing a connection with the libmodbus library is done with the following lines of code:

```
modbus_t *ctx
ctx = modbus_new_rtu("/dev/ttyUSB0", 9600, 'N', 8, 1);
modbus_set_debug(ctx, TRUE);
modbus_set_error_recovery(ctx, MODBUS_ERROR_RECOVERY_LINK | MODBUS_ERROR_RECOVERY_PROTOCOL);
modbus_set_slave(ctx, TESLA_ID);
```

Here *modbus_new_rtu* opens a Modbus RTU context with 9600 baud, 8 data bits and 1 stop bit on the USB connection *ttyUSB0*. Furthermore, *modbus_set_debug* is enabled to monitor traffic on the bus, *modbus_set_error_recovery* sets the error recovery protocol to the standard libmodbus protocol and *modbus_set_slave* specifies the slave ID. TESLA_ID is defined as 24 in the header file, which is included in Appendix B. On both the server and client side the slave ID is set to be the same. This way the server knows its own address and the client knows which address to send its messages to.

The connection needs to be closed at the end of a program, this is done with *modbus_close* and *modbus_free* is used to free the memory used:

```
modbus_close(ctx);
modbus_free(ctx);
```

On the server a mapping is made. This allocates memory for the bits and registers used in the Modbus protocol. It uses the addresses of the bits and registers and the amount of bits and registers defined in the header file to do this:

```
mb_mapping = modbus_mapping_new( UT_BITS_ADDRESS + UT_BITS_NB, UT_INPUT_BITS_ADDRESS +
    UT_INPUT_BITS_NB, TESLA_ADDRESS + TESLA_REGISTERS_NB, TESLA_INPUT_REGISTERS_ADDRESS +
    TESLA_INPUT_REGISTERS_NB);
if (mb_mapping == NULL) {
        fprintf(stderr, "Failed to allocate the mapping: %s\n", modbus_strerror(errno));
        modbus_free(ctx);
        return -1;
}
```

An error can occur during the initialization phase of the program. To counter this and ensure a connection the program was altered to start over and reattempt to establish a connection whenever this occurs. The test following this alteration showed that a connection could always be initialized without any other complications.

5.3.2. Communicating Variables

In order to reach the set goals, it is essential to be able to communicate variables over an RS-485 connection. The main Odroid sends out a request to read the contents of specific registers on the second Odroid. The second Odroid responds by sending the content of the registers in the same way the Tesla Powerwall communicates with the SolarEdge Inverter.

The function *modbus_read_input_registers()* reads a number of registers from a specific register address from a specific slave and stores the values. It returns the amount of registers it has read. The implementations is shown below, where *ctx* is the Modbus context, *TESLA_INPUT_REGISTERS_ADDRESS* is the starting address of the registers to be read, *TESLA_INPUT_REGISTERS_NB* is the number of registers to be read and *tab_rp_registers* is the array pointer where the values should be stored.

```
rc = modbus_read_input_registers(ctx, TESLA_INPUT_REGISTERS_ADDRESS, TESLA_INPUT_REGISTERS_NB
    , tab_rp_registers);
ASSERT_TRUE(rc == TESLA_INPUT_REGISTERS_NB, "");
```

ASSERT_TRUE determines if the correct amount of registers have been read. Tests showed that the correct variables were send. Additionally, the program was integrated with the earlier discussed webpage. This offered a clear way of reviewing the test results. No complications arose from these tests. Additional tests were done with slowly increasing variables. These tests showed that the communication of the changing variables is successfully.

5.3.3. Logging Communicated Signals

In the existing system with the SolarEdge inverter directly communicating with the Powerwall is difficult. As discussed previously, Modbus allows for one master within the system and a set of slaves. There already is a master on the communication line in the original setup, the SolarEdge Inverter. It is, however, possible to track every signal sent over the communication line. As communication works by broadcasting the signals over the RS-485 line with the message themselves dictating who is to respond it is possible to log all of the signals sent. A function has been implemented which prints each signal into a text file. The results are easily verifiable if they are compared to the libmodbus library and the communicated variables.

5.4. Testing with the Tesla Powerwall

The RS-485 line coming from the Odroid is connected to the line running from the StorEdge Interface to the Powerwall, an overview is shown in figure 5.3. The software used to test with two Odroids is ran including the function used to log the communicated signals. There was communication with two slaves in the system, one on address 2 and one on address 24, which is the Powerwall's address. The messages to and from the Tesla Powerwall could now be analysed.

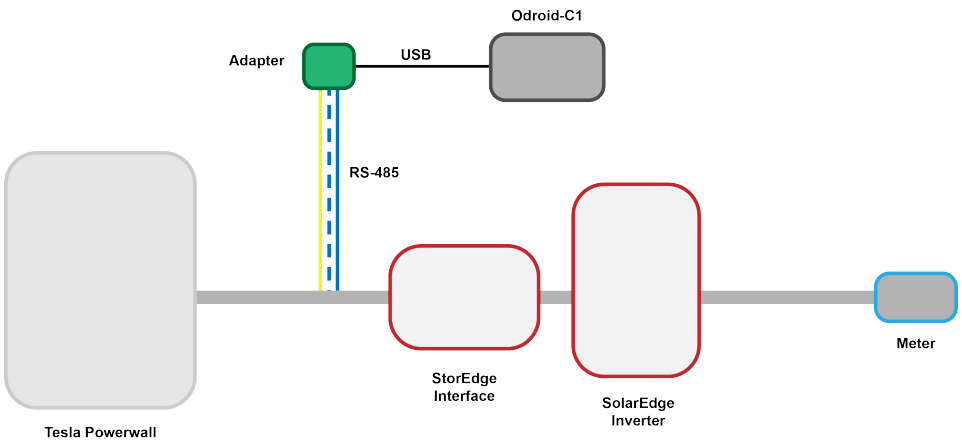


Figure 5.3: A schematic overview of the Tesla Powerwall system data connections during testing.

5.4.1. Message Analysis

From the intercepted communications the message is found where the request for the data from the Powerwall done. The master in the system sends out a request to the Powerwall with the following message: *18 04 00C8 0018 73F7*. The information this message contains is:

```
18 Slave Address (24)
04 Function Code Read Input Registers (4)
00C8 Starting address (200)
0018 Quantity (24)
73F7 CRC
```

The inverter asks to read 24 Input Registers of the Powerwall (48 bytes) starting from address 201. In [Redacted] there are certain variables specified which the Powerwall returns, this is however limited to XX registers (XX bytes) of data. The message the Powerwall returned in the first test is: *18 04 30 0007 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 FFFF FFFF 0027 0001 007D 007A 0000 0000 0000 0017 5333*. Using [Redacted] an estimation is made to what which register corresponds. Some fluctuation was spotted in the [Redacted] and [Redacted] registers indicating the data to be accurate and updated regularly. The data corresponds to:

```
18 Slave Address (24)
04 Function Code Read Input Registers (4)
30 Byte Count (48)
0007 XX
0000 XX
0000 XX
0000 XX
0000 XX
0000 0000 XX
0000 0000 XX
0000 XX
0000 XX
0000 XX
0000 XX
FFFF XX
FFFF XX
0027 XX
0001 XX
007D XX
007A XX
0000 XX
0000 XX
0000 XX
0017 XX
5333 CRC
```

Most of the data gives no information, although some registers give interesting values. In [Redacted] there are X statuses specified without any decimal or hexadecimal value specified. The value here is 7 and the inverter displayed the *init* status. The [Redacted] is reported as [Redacted], this is a possible value although it did not seem to change over time. The [Redacted] register corresponds to the Enable signal the Powerwall gets from the StorEdge Interface. The [Redacted] are most likely displaying the correct values. They are supplied with two separate 12 volt connections and they can fluctuate some tenths of a volt over time. The last two registers before the CRC are unknown and not described in the documents available to us.

It could very well be possible the Powerwall doesn't display the data from most of the registers during the init state. All of the register except for the [Redacted] did not change during monitoring of a large set of messages. Only the [Redacted] register could be changed from TRUE to FALSE by disabling the SolarEdge StorEdge Interface.

It is also possible the Powerwall is in a deep discharge state and is possibly broken beyond repair. There has been correspondence with Eneco and SolarEdge but they could not confirm this scenario at this time.

The Master also request data from the Wattnode Modbus Meter. This is done with the message: *02 03 04 0E 00 06 A5 08*. The information in this message is:

```
02 Slave Address (2)
03 Function Code Read Holding Registers (3)
040E Starting address (1039)
0006 Quantity (6)
A508 CRC
```

In firmware update 22 of the meter the registers starting at address 1039 were added. They contain the power flowing into the in-home network specified as a IEEE-754 Floating-Point in hexadecimal notation. The Least Significant Bits are returned first, then the Most Significant Bits for each phase. One of the returned messages was: *02 03 0C C2FE 4157 0000 0000 0000 0000 943E*. This corresponds to:

```
02 Slave address (2)
03 Function code Read Holding Registers (3)
0C Byte count (12)
C2FE LSB PowerAFast
4157 MSB PowerAFast
0000 LSB PowerBFast
0000 MSB PowerBFast
0000 LSB PowerCFast
0000 MSB PowerCFast
943E CRC
```

The value for PowerAFast is 4157C2FE or 13.4851 in decimal notation. This means there was a bit more then 13 watt flowing into the system. Only one phase is connected to the meter so PowerBFast and PowerCFast will always be 0. The value in PowerAFast was also reported by the inverter during testing, validating that this is indeed the message we receive.

After the response from the meter the master makes another request to the Powerwall. The message is: *18 10 03E8 0009 12 0003 1090 10A4 0014 0014 0000 00FA AA55 001E 32CE*. This time is has function code 16, indicating a *Write Holding Registers* request. It requests to write 9 registers (18 bytes) starting at address 1001. The documentation available does not specify what these registers contain or why they are written. The full message is as follows:

```
18 Slave Address (24)
10 Function Code Write Holding Registers (16)
03E8 Starting address (1001)
0009 Quantity (9)
12 Byte count (18)
0003 Register Values
1090 ..
10A4 ..
0014 ..
0014 ..
0000 ..
00FA ..
AA55 ..
001E ..
32CE CRC
```

This write request may very well be only occurring during the initialization of the system. Because it never got past this state this can't yet be determined. The response from the Powerwall is: *18 10 03E8 0009 8276*. This confirms to the inverter the registers have been written to.

The message that follows the write request is another read request to the meter. After that the cycle of four messages, *Read Multiple Holding Registers Meter -> Read Input Registers Powerwall -> Read Multiple Holding Registers Meter -> Write Multiple Holding Registers Powerwall -> Repeat*, repeats endlessly without pause.

Unfortunately, the Powerwall was never able to leave the initialisation state. The combined efforts of the DCE&S department, the Bachelor Graduation Group, Eneco, Tesla and SolarEdge weren't enough to overcome this setback in time. The situation has prevented further and more elaborate research to the full meaning of the logged communications.

5.4.2. Displaying the Variables

When the read request is done by the Inverter the variables can still be displayed on the webpage. This was done during the tests with the Tesla Powerwall by looking at the header of the messages. By checking if the header specified the address 0x18, the function code 0x04 and the byte count 0x30 the correct message is used. The data it contains is then given to the function *htmldisplay()* and it is put on the website.

5.5. Prototype Implementation

The final prototype orientates itself towards an adaptation of the designed module in within a newly designed inverter, meeting requirement 2.1.9, in which the designed communication module would function as the *Master* on the RS-485 communication lines. To implement this, the same physical set-up was used as was used to test basic communication between two Odroids 5.3. An overview of that setup can be seen in figure 4.1. The first Odroid will serve as the communication module, while the second Odroid emulates the Powerwall. Meanwhile the first Odroid will host the local webpage as well.

In order to properly discuss the Prototype Implementation, first the Control Computation will be elaborated upon. Secondly, the created Test bench is explained. After that, the test results are discussed.

5.5.1. Control Computation

As discussed in the design process in section 4.3.2 the choice was made to implement the charging command as a Bang-Bang control signal, with the possibility of not sending a signal at all. The result of this decision is that the command variable has three possible values: 100 (charge), -100 (discharge) and 0 (self-consumption). Within the prototype implementation the command signal is send to the Powerwall-emulating Odroid.

The necessary control command is computed as dictated by the user selected mode. The implemented modes are *Back Up*, *Self-Consumption*, *Smart Selling* and *Hybrid* as discussed in section 4.3.1. These modes use the following variables to compute the control command: *Energy Remaining (kWh)*, *Battery Temperature (degC)* and, depending on the mode, *Energy Price (€)*. The first two of these are part of the variables the Powerwall communicates. The energy price however, has to be retrieved form an additional external source. As the reason of including the energy price within the computation, is to demonstrate possible uses of the communication model. Below the computation algorithms of each mode is discussed in more detail.

The *Back Up* mode ensures a large amount of energy is stored in the Powerwall. It does this by sending the charge command whenever *Energy Remaining* drops below 6000 kWh. Additionally, it is required for the battery temperature to be at least 0 degrees Celsius. If one of these conditions isn't met, the Powerwall will enter start self-consumption.

The *Self-Consumption* mode is equivalent to the default state of the Powerwall. A control command is thus not necessary.

While in *Smart Selling* mode the energy prices are checked. When the energy prices are high enough, the Powerwall is told to discharge. To prevent technical problems due to the Powerwall trying to discharge beyond its capabilities, *Energy Remaining* has to equal at least 1 kWh. When the energy price is low, the Powerwall is told to seize the opportunity to charge with an upper limit of 6.5 kWh. In all other cases the Powerwall will maximise self-consuming.

The *Hybrid* mode is designed to combine the previously discussed modes. It combines the focus on energy prices, while also ensuring a certain amount of back up energy. First of all, the algorithm considers the case in which the energy price is low. In this case computation mode will work like in the

Back Up mode. The Powerwall is told to charge as long as the battery temperature is above 0 degrees Celsius and the energy remaining is less than 6 kWh. If either of these requirements aren't adhered to, the Powerwall will enter self-consumption. Secondly, the case is considered in which the energy price is high. In this case the Powerwall will be told to discharge as long as at least 3 kWh of energy is left in the battery pack. Charging is only used to have an acceptable amount of back up energy available and is thus only instructed if the energy remaining drops below 2.95 kWh. In all other cases the Powerwall will focus on self-consuming.

As displayed below a switch statement was implemented to select a computation algorithm. The complete code for the client is included in appendix B.3.

```
switch(mode)
{
        case 'B'  : // Full Backup Mode
                command = mode_backup(tab_rp_registers);
                break;
        case 'S'  : // Sell Energy at certain threshold
                command = mode_sell(tab_rp_registers, energyprice);
                break;
        case 'C'  : // Self Consumption (leave it to the inverter)
                command = 0;
                break;
        case 'H'  : // Hybrid Backup-Consumption
                command = mode_hybrid(tab_rp_registers, energyprice);
                break;
        default :
                printf("Invalid mode\n" );
                command = 0;
                break;
}
```

5.5.2. Test bench

In order to emulate the Powerwall on the second Odroid the code run on it establishes a connection as a slave on the RS-485 communication line. This is done in the same manner as described at section 5.3. The server acting as Powerwall has the ID 24, like the Powerwall. It also uses the same input registers to store the data, starting at address 200 (0xC8) with 24 (0x18) registers. These were derived from the log analysis in section 5.4.1. The registers contain values as dictated by the designed test bench.

```
const uint16_t TESLA_INPUT_REGISTERS_ADDRESS = 0xC8;
const uint16_t TESLA_INPUT_REGISTERS_NB = 0x18;
```

For receiving commands 16 (0x10) holding registers are reserved at address 3273 (0x0CC9). These can also be allocated elsewhere as they are for the test bench only.

```
const uint16_t TESLA_ADDRESS = 0x0CC9;
const uint16_t TESLA_REGISTERS_NB = 0x10;
```

The server receives the command from the client and processes it. The command is added to the energy remaining, increasing or decreasing it if it isn't 0. A snippet from the code is where this happens displayed below:

```
if (mb_mapping->tab_registers[TESLA_ADDRESS] != 0) // No Self-consumption
{
        mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 5] = mb_mapping->
            tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 5] + (int16_t) mb_mapping->
            tab_registers[TESLA_ADDRESS];
        printf("Charging with %" PRId16 "\n", (int16_t) mb_mapping->tab_registers[
            TESLA_ADDRESS]);
...
```

All of the registers can be given a new value when a command arrives. At this time most of the registers sent back to the client still contain static values. Data from the actual behaviour of the Powerwall is needed to determine how these values change and how they could be implemented in the test bench. The complete code for the test bench server is included in appendix B.2.

For testing the energy price is generated in the client itself and not retrieved from an outside source. The test bench for the energy price is included in the function *Main*. The price variable increases from 2 to 13 and then drops back to 2. The value of 6 is determined to be the difference between a low and a high price for these tests.

5.5.3. Test Results

For the tests the energy price was set to rise and fall with a set pattern. For testing purposes the self-consumption state was defined to have a discharge rate of 30 Wh per message cycle. This means it is assumes more power is consumed by the loads than is generated. First, tests were done to test each mode individually. The test were successful, each mode performed as expected. After this longer tests were performed.

In figure 5.4 the results from a test are displayed. Here the webpage is used to switch modes. It starts in the *Backup* mode until 40 messages have passed. The energy is kept at a steady level by sometimes recharging the battery. Then it is put in *self-consumption* mode for 15 message cycles and it discharges steadily. At 55 messages it is switched to *Smart Selling* mode and it is kept there for 37 messages. During this time it goes into self-consumption if the energy price is low and it discharges if the price is high, that can be seen in the Charge Command plot. At the 92 messages point the client is instructed by the user to go into *Hybrid* mode. Here it first starts charging to a set level to provide backup power if needed. At around 120 message cycles it hovers around that point because the energy price it too high to charge extra. When the price is lower again it starts charging until the price is high and it discharges.

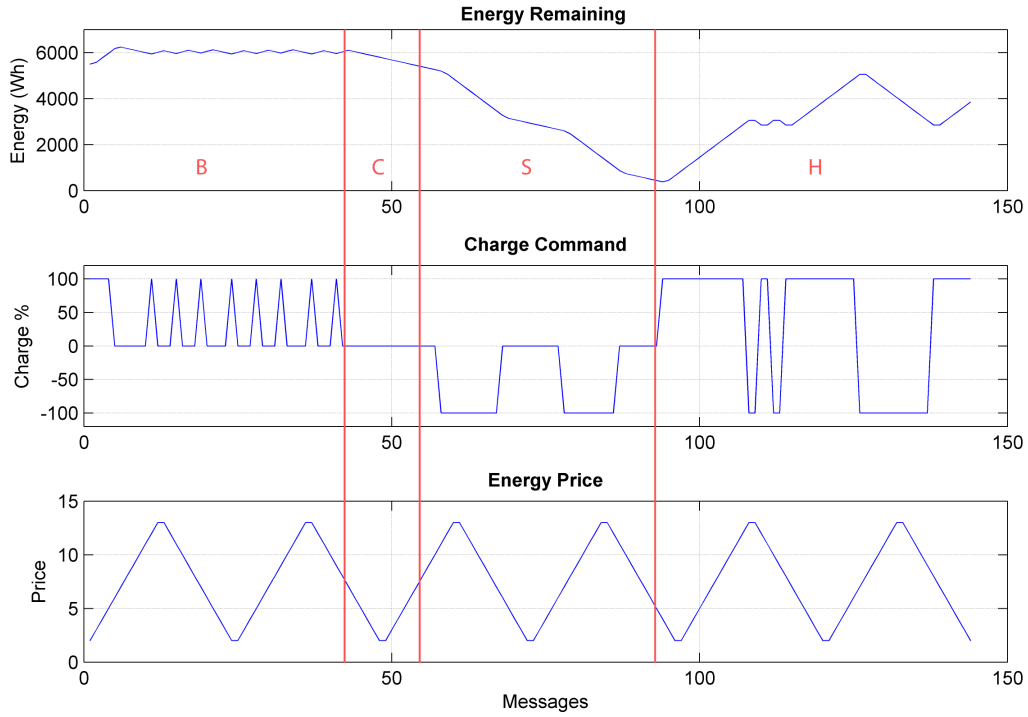


Figure 5.4: The data from testing with the test bench. On top the energy remaining is shown, in the middle the charge command that is computed and on the bottom the fluctuating energy price is shown. 'B' indicates the backup mode, 'C' indicates the self-consumption mode, 'S' indicates the smart selling mode and 'H' indicates the hybrid mode.

During testing in some cases a timeout occurred and the connection between the two Odroids was reset. This happened when a display was connected and the processing power of the Odroid wasn't enough to respond to a query in time. When no display was attached the communication was fine.

6

Discussion of the Results

In this chapter the results of the different implementations, as described in chapter 5, are discussed. The result have partially already been discussed in the previous chapter, but this chapter offers a complete and dedicated overview.

6.1. User Interface

The User Interface served to complete two separate tasks: it had to display the data retrieved from the system and process user input.

Testing showed that the programme was able to host a local webpage and to display the received variables on it. Additionally, by automatically refreshing the webpage the variables were able to be kept up to date. Though this implementation meets the basic needs for this project, the variables aren't as up to date as wished. The programme would have to be altered to show the data in real time rather than with a small delay. Furthermore, the webpage as it is only displays the data received by the Powerwall, not by the smart meter. The reason for this, is that it wouldn't add much to the performed tests and would work in the exact same manner.

The mode selected by the user was able to be communicated and used throughout the rest of the software, as discussed in the test results. Even though it serves it purpose, the implemented user form is quite limited. Mostly because it doesn't give immediate feedback on the selected mode. The selected mode is displayed within the list of variables, but due to the limitations discussed there is a small delay present.

Furthermore, he webpage turned out to be accessible both from a local connection and from the Internet. For the short tests performed this is fine. When longer testing is done restricting access would provide a safer environment for tests.

6.2. Testing Communication with two Odroids

The initial program implementation on the two Odroids had to test three different objectives.

First of all, the possibility of establishing a connection with the use of Modbus. After the discussed loop the function enters whenever an error occurs was implemented, connections can be initialized without complications. Secondly, the communication of variables has been tested. The results confirmed that the program was working as expected. An important discussion point on these results is that it doesn't guarantee a successful connection to the existing communication lines. Though, the basis is the same, the product can't actively send out requests as the Inverter already serves as a master.

Lastly, the functionality of logging all of the communicated signals has been tested via the Odroids. The writing of these signals into a designated text file worked successfully. This positive result allows the function to be used when researching communication over the RS-485 line.

6.3. Testing with the Tesla Powerwall System

The analysis of the logged communication signals allows for a lot of insight into the communication with the Powerwall. First of all, the analysis showed that the planned way of reading the communicated variables works. Many of the registers turned up empty. The values that did return were the values one would expect.

The reason for the empty registers seems to be that the Powerwall was never fully operational during the performed tests. The Powerwall wasn't able to leave its initiation state. Attempts have been made by all of the involved parties to solve this issue, but unfortunately no solution was found prior to writing this document. As a result, no proper analysis could be done on the entirety of the existing communications network.

There are still multiple unidentified registers of which the meaning might have been discovered if elaborate testing would have been possible.

6.4. Prototype Implementation

The charging command computation has been tested as part of the overall prototype implementation. As an input for the main Odroid a second Odroid was used which ran a test bench as a surrogate Powerwall as discussed in section 5.5.2. The test results, as discussed in section 5.5.3, show that the program works as expected. Even though, the test bench checked most of the possible situations, one condition hasn't been tested. One of the safety requirements is that the Tesla Powerwall isn't allowed to charge itself when the temperature of the battery is below 0 °C. This limit has been implemented within the command computations, but hasn't been explicitly tested. This isn't a large problem however, as the register holding the temperature works in the exact same manner as the register holding the state of energy.

7

Conclusions

In this chapter the conclusions of the project are discussed. This is done by evaluating the set goals and whether or not they were met, followed by a section on future work and recommendations.

7.1. Goals

The goals set at the beginning of the project will be discussed one by one in a comparison with the achieved results. The goals were defined at the start of the project as follows:

1. Retrieve and read the information communicated over the communication lines
2. Display the retrieved information
3. Allow for user preferences
4. Compute and send (dis)charging commands

Retrieving and reading the information communicated over the communication lines is perfectly possible. This can be done for two different implementations. First of all, it is possible to log all of the communicated signals for further analysis. Secondly, the program allows the product to take up the position of the master on the Modbus network and actively request the relevant status information.

The retrieved information can be displayed on the locally hosted webpage and updates itself without any additional interference by the user. Also, the user preferences can be entered on the webpage. By selecting one of four modes a computation algorithm is set in motion controlling the system. The selected algorithm uses the retrieved variables to compute a charging command. Even though, no dedicated control module has been identified to which the command is to be send, the command can be send and thus meets the set goal.

Overall the goals set for the project have been accomplished, but more insight into the existing system would have been appreciated.

7.2. Future Work and Recommendations

The main recommendation for future work is that further analysis of the existing communication is required. The delivered programme can be used to create a log of the send signals, which in turn can be used for further analysis. In order to allow for this, it is essential for the Tesla Powerwall to be working first. The analysis can complete the picture of what it is that is communicated to and from the Powerwall.

Furthermore, as discussed earlier, the final product is capable of serving as the master on a Modbus connection with the Tesla Powerwall. Because of this, the final product can be used as a module within a possible inverter designed by the DCE&S department, thus meeting requirement 2.1.9. In a similar

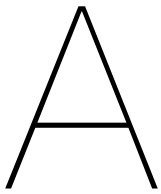
way, the product can be implemented on a newly designed battery pack, as long as it communicates via an RS-485 connection.

There is also more work to be done when it comes to the implementation of the user preferences. As of right now, it is only possible to select one of four different profiles. Even though other profiles can easily be added, it might be worthwhile, for research purposes, to offer the possibility to manually set upper and lower limitations for several variables.

It could also be advised to develop a smartphone application and to set up profiles with modes linked to certain time stamps. This recommendation is however, very consumer oriented and wouldn't offer more insight into the possibilities of the system. Therefore, it shouldn't be prioritized.

Additionally, very important work is still left when it comes to connection multiple in-home energy management systems. Monitoring the status information of multiple energy management systems and being able to use a portion of the battery packs for grid stabilizing purposes, can open up a lot of important possibilities for future applications.

Lastly, a Simulink model of the designed prototype will have to be designed in order to integrate the prototype with the results of the other two subgroups of the Bachelor Graduation Group. Combined with the simulation developed by both groups, a well grounded model can be created of a controllable in-home energy management system.



Setup Guide

In this appendix the steps are described to install and configure the required software on a Linux based computer to interface with the Tesla Powerwall system. This guide assumes the Linux computer is configured with an Internet connection and the user has access to the command line either through a local shell or by connecting to is via an Secure Shell (SSH) connection.

A.1. Git Repository

If not already installed, install git with the command:

```
sudo apt-get install git
```

The installation can be verified with:

```
which git
```

After that the name and email git should use can be specified:

```
git config --global user.name "name"
git config --global user.email "name@mail.com"
```

Now the repository from which the software needs to be pulled is added. In this case it is on the *dcgrid* repository and only the Powerwall part needs to actually be downloaded. For this a sparse checkout is used. So first create an empty repository in a directory:

```
mkdir dcgrid
cd dcgrid
git init
git remote add -f dcgrid https://github.com/laurensss/dcgrid.git
```

To enable sparse checkout type:

```
git config core.sparsecheckout true
```

Now configure the directory of the software to be included:

```
echo software/powerwall/ >> .git/info/sparse-checkout
```

Finally the pull request can be sent:

```
git pull dcgrid master
```

If the powerwall branch has not been merged with the main branch, a switch of branches is needed.

```
git checkout powerwall
```

Then the pull request can be sent like this:

```
git pull dcgrid powerwall
```

Now all of the most recent files from the *powerwall* branch are on the system.

A.2. Webserver

For displaying the webpage the software uses an Apache Webserver. To install it simply type:

```
sudo apt-get install apache2
```

The installation can be verified by typing *localhost* into a web browser and checking if a page is displayed.

Now to install PHP for the webserver. Use the following command to install it:

```
sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt
```

To make sure the software has access to the files it needs for the web server, permissions need to be changed in the /var/www/ directory. Navigate to it and use:

```
sudo chmod -R 777 .
```

A.3. Libraries

The libmodbus library is included in the repository however it still needs to be installed. navigate to the directory */powerwall/lib/libmodbus-master/* to start. Use *autogen.sh* to generate the configure script.

```
sudo ./autogen.sh
```

To install the library, type:

```
sudo ./configure
sudo make install
```

To make the library work properly *modbus.h* needs to be copied to */usr/local/include*.

```
sudo cp modbus.h /usr/local/include
```

And also *libmodbus.pc* needs to be placed in */usr/lib/pkgconfig*. This is done with:

```
sudo cp libmodbus.pc /usr/lib/pkgconfig
```

Now the library cache needs to be updated. To update the library cache use:

```
sudo ldconfig
```

The libmodbus library is now ready to use.

A.4. Compiler

As a compiler GCC is used, with as a special addition flags for pkg-config indicating it should use the libmodbus library. The command to compile *main.c* for example would be:

```
gcc main.c -o main `pkg-config --libs --cflags libmodbus`
```

A.5. Test Bench

To run the test bench two devices connected by via a RS-485 line on their USB ports are required. Navigate to the subfolder /powerwall/testbench/. On the client, compile the software with:

```
gcc main.c -o main `pkg-config --libs --cflags libmodbus`
```

On the server device, compile the server software:

```
gcc testbench-server.c -o testbench-server `pkg-config --libs --cflags libmodbus`
```

On the server, run the software with:

```
sudo ./testbench-server
```

A log of the communications happening will be created in *serverlog.txt*. It also logs the the data it sends in *datalog-s.txt*.

On the client, run the software with:

```
sudo ./main
```

It also logs the data, this time in *datalog-c.txt*. The software can now be monitored by going to *localhost* in a browser on the client device or navigating to the IP address of the client, if the firewall of the network it is on permits traffic to port 80 of the device.

B

Software

B.1. testbench.h

This header file used in the test bench.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* BAP Group I: Testing the Tesla Powerwall                                  */
/* Bart Kolling & Ludo van den Buijs                                         */
/* Header file of ”testbench-client.c” and ”testbench-server.c”              */
/*                                                                           */
/* The base of the communication is provided by the libmodbus library.       */
/* Copyright © 2008-2014 Stéphane Raimbault <stephane.raimbault@gmail.com>   */
/* SPDX-License-Identifier: BSD-3-Clause                                     */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _TESTBENCH_H_
#define _TESTBENCH_H_

#define HAVE_INTTYPES_H 1
#define HAVE_STDINT_H 1

#ifdef HAVE_INTTYPES_H
#include <inttypes.h>
#endif
#ifdef HAVE_STDINT_H
# ifndef _MSC_VER
# include <stdint.h>
# else
# include ”stdint.h”
# endif
#endif

#define TESLA_ID          24

/* Server allocates address + nb */
// Define address and number of bits
const uint16_t UT_BITS_ADDRESS = 0x130;
const uint16_t UT_BITS_NB = 0x25;

// Define address and number of input bits
const uint16_t UT_INPUT_BITS_ADDRESS = 0x1C4;
const uint16_t UT_INPUT_BITS_NB = 0x16;

// Define address and number of input registers of Powerwall
const uint16_t TESLA_INPUT_REGISTERS_ADDRESS = 0xC8;
const uint16_t TESLA_INPUT_REGISTERS_NB = 0x18;
const uint16_t TESLA_INPUT_REGISTERS_TAB[] = { 0x000A };

// Define address and number of holding registers of Powerwall
const uint16_t TESLA_ADDRESS = 0x0CC9;
```

```
const uint16_t TESLA_REGISTERS_NB = 0x10;

int tesla_client(int Energyprice);
#endif /* _TESTBENCH_H_ */
```

B.2. testbench-server.c

This code is used to emulate the Powerwall with a test bench.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* BAP Group I: Testing the Tesla Powerwall                                    */
/* Bart Kolling & Ludo van den Buijs                                           */
/* Test bench Server                                                           */
/* - Establish Connection                                                      */
/* - Emulate Powerwall                                                         */
/* - Process Charge Command                                                    */
/* - Communicate Variables upon request                                        */
/*                                                                             */
/* The base of the communication is provided by the libmodbus library.         */
/* Copyright © 2008-2014 Stéphane Raimbault <stephane.raimbault@gmail.com>     */
/* SPDX-License-Identifier: BSD-3-Clause                                       */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <modbus.h>
#ifdef _WIN32
# include <winsock2.h>
#else
# include <sys/socket.h>
#endif

#include <time.h>
#include <inttypes.h>

/* For MinGW */
#ifndef MSG_NOSIGNAL
# define MSG_NOSIGNAL 0
#endif

#include "testbench.h"

int main(void)
{
        modbus_t *ctx;
        modbus_mapping_t *mb_mapping;
        int rc;
        int i;
        uint8_t *query;

        int p_selfcon = -30; // Power generated during self-consumption

        // Initialize the Modbus Context
        ctx = modbus_new_rtu("/dev/ttyUSB0", 9600, 'N', 8, 1);
        modbus_set_slave(ctx, TESLA_ID);
        query = malloc(MODBUS_RTU_MAX_ADU_LENGTH);
        modbus_set_debug(ctx, TRUE);

        mb_mapping = modbus_mapping_new( UT_BITS_ADDRESS + UT_BITS_NB, UT_INPUT_BITS_ADDRESS
            + UT_INPUT_BITS_NB, TESLA_ADDRESS + TESLA_REGISTERS_NB,
            TESLA_INPUT_REGISTERS_ADDRESS + TESLA_INPUT_REGISTERS_NB);
        if (mb_mapping == NULL) {
                fprintf(stderr, "Failed to allocate the mapping: %s\n", modbus_strerror(errno
                    ));
                modbus_free(ctx);
                return -1;
```

```
}

rc = modbus_connect(ctx);
if (rc == -1) {
        fprintf(stderr, "Unable to connect %s\n", modbus_strerror(errno));
        modbus_free(ctx);
        return -1;
}

// Place values in registers
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS] = 0;
                    // Status
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 1] = 3500;
                    // 0.1 V dc_v
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 2] = 500;
                    // W estimated DC power
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 3] = 0;
                    // Wh nameplate energy remailing BOL
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 4] = 6700;
                // Wh full pack energy available
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 5] = 5500;
                    // Wh energy remaining
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 6] = 0x0000;
            // Lifetime energy charged MSB
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 7] = 0x0000;
            // Lifetime energy charged LSB
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 8] = 0x0000;
            // Lifetime energy discharged MSB
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 9] = 0x0000;
            // Lifetime energy discharged LSB
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 10] = 0;
                    // Nominal charge power
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 11] = 0;
                    // Nominal discharge power
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 12] = 0;
                    // Max charge power
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 13] = 0;
                    // Max discharge power
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 14] = 0;
                    // 0.1 V Battery Voltage
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 15] = 0;
                    // 0.1 A Battery Current
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 16] = 25;
                    // degC Battery Temperature
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 17] = 1;
                    // Bool Wake/Enable Status
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 18] = 120;
                    // 0.1 V Logic Voltage
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 19] = 120;
                    // 0.1 V Thermal Voltage
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 20] = 0;
                    // Bool Thermal Power Needed
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 21] = 5400;
            // Wh Actual Energy remaining
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 22] = 0;
                    // Unknown
mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 23] = 0;
                    // Unknown

for (;;) {

        do {
                rc = modbus_receive(ctx, query);

                // Get Time
                time_t timer;
                char buffer[26];
                struct tm* tm_info;
                time(&timer);
                tm_info = localtime(&timer);
                strftime(buffer, 26, "%Y:%m:%d %H:%M:%S", tm_info);
```

```
                // Print Log
                FILE *log;
                log = fopen("serverlog.txt","a");
                i = 0;
                while (i < 50)
                {
                        fprintf(log, "%0X ", query[i]);
                        i++;
                }
                fprintf(log, "   %s\n", buffer);
                fclose(log);

        } while (rc == 0); // Filtered query gives 0

        /* The connection is not closed on errors which require on reply such as
        bad CRC in RTU. */
        if (rc == -1 && errno != EMBBADCRC) {
                break;
        }

        rc = modbus_reply(ctx, query, rc, mb_mapping);
        if (rc == -1) {
                break;
        }

        // Test bench
        if (mb_mapping->tab_registers[TESLA_ADDRESS] != 0) // No Self-consumption
        {
                mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 5] =
                    mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS +
                    5] + (int16_t) mb_mapping->tab_registers[TESLA_ADDRESS];
                printf("Charging with %" PRId16 "\n", (int16_t) mb_mapping->
                    tab_registers[TESLA_ADDRESS]);

                if ((int16_t) mb_mapping->tab_input_registers[TESLA_ADDRESS] > 0) //
                    Charging
                {
                        mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS
                             + 1] = 4000;                       // 0.1 V dc_v
                        mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS
                             + 2] = 20*mb_mapping->tab_registers[TESLA_ADDRESS];
                                        // W estimated DC power

                        mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS
                             + 14] = mb_mapping->tab_input_registers[
                            TESLA_INPUT_REGISTERS_ADDRESS + 1];
                                // Battery Voltage
                        mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS
                             + 15] = mb_mapping->tab_input_registers[
                            TESLA_INPUT_REGISTERS_ADDRESS + 14]/mb_mapping->
                            tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 2];
                                        // Battery Current
                }

                if ((int16_t) mb_mapping->tab_input_registers[TESLA_ADDRESS] < 0) //
                    Discharging
                {
                        mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS
                             + 1] = 3500;                       // 0.1 V dc_v
                        mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS
                             + 2] = 20*mb_mapping->tab_registers[TESLA_ADDRESS];
                                        // W estimated DC power

                        mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS
                             + 14] = mb_mapping->tab_input_registers[
                            TESLA_INPUT_REGISTERS_ADDRESS + 1];
                                // Battery Voltage
                        mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS
                             + 15] = mb_mapping->tab_input_registers[
                            TESLA_INPUT_REGISTERS_ADDRESS + 14]/mb_mapping->
```

```
                                tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 2];
                                                // Battery Current
                        }
                }
                else // Self Consumption
                {

                        if (mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS +
                            21] > 10)
                        {
                                mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS
                                     + 5] = mb_mapping->tab_input_registers[
                                    TESLA_INPUT_REGISTERS_ADDRESS + 5] + p_selfcon;
                        }
                        printf("Self-consumption\n");
                }
                mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 21] = 0.98*
                    mb_mapping->tab_input_registers[TESLA_INPUT_REGISTERS_ADDRESS + 5];


                // Printing the values of the registers
                i = 0;
                while (i < 3)
                {

                        printf("%d = %0X\n", i, mb_mapping->tab_registers[TESLA_ADDRESS + i])
                            ;
                        i++;
                }

                // Logging the data
                FILE *log;
                log = fopen("datalog-s.txt","a");
                i = 0;
                while (i < 24)
                {
                        fprintf(log, "%" PRId16 " ", mb_mapping->tab_input_registers[
                            TESLA_INPUT_REGISTERS_ADDRESS + i]);
                        i++;
                }

                fprintf(log, "     %" PRId16 "\n", (int16_t) mb_mapping->tab_registers[
                    TESLA_ADDRESS]);
                fclose(log);

        }

        printf("Quit the loop: %s\n", modbus_strerror(errno));

        // Free the memory
        modbus_mapping_free(mb_mapping);
        free(query);

        // Close the connection
        modbus_close(ctx);
        modbus_free(ctx);

        return 0;
}
```

B.3. testbench-client.c

This code has is used to connect over RS-485 and request variables.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* BAP Group I: Testing the Tesla Powerwall                                    */
/* Bart Kolling & Ludo van den Buijs                                           */
/* Client                                                                      */
/* - Establish a connection                                                    */
```

```
/* - Request variables                                                    */
/* - Retrieve the selected Mode                                           */
/* - Initialize "localhost"                                               */
/* - Compute command based on Mode and Variables                          */
/*                                                                        */
/* The base of the communication is provided by the libmodbus library.    */
/* Copyright © 2008-2014 Stéphane Raimbault <stephane.raimbault@gmail.com> */
/* SPDX-License-Identifier: BSD-3-Clause                                  */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <modbus.h>

#include "testbench.h"
#include "htmldisplay.h"
#include "htmldisplay.c"

#define BUG_REPORT(_cond, _format, _args ...) \
        printf("\nLine %d: assertion error for '%s': " _format "\n", __LINE__, # _cond, ##
            _args)

#define ASSERT_TRUE(_cond, _format, __args...) {  \
                if (_cond) {                                    \
                        printf("OK\n");                             \
                } else {                                        \
                        BUG_REPORT(_cond, _format, ## __args);      \
                        goto restart;                               \
                }                                               \
        };

char getmode(void) // Returns Mode preference by user
{
        FILE *fp;

        fp = fopen("/var/www/html/userpref.txt", "r");
        if(fp == 0)
        {
                printf("Could not get user mode!");
                fclose(fp);
                return 'C'; //Return Selfconsumption
        }

        char mode = fgetc(fp);
        printf("Got mode %c \n", mode);
        fclose(fp);
        return mode;
}

int mode_backup(uint16_t *tab_rp_registers)
{
        if (tab_rp_registers[5] < 6000 & tab_rp_registers[16] > 0)
        {
                return 100;
        }
        else
        {
                return 0;
        }
}

int mode_sell(uint16_t *tab_rp_registers, int energyprice)
{
        if (tab_rp_registers[5] < 1000 | energyprice < 7)
        {
                return 0;
        }
        else if (energyprice < 3 & tab_rp_registers[5] < 6500)
```

```
        {
                return 100;
        }
        else
        {
                return -100;
        }
}

int mode_hybrid(uint16_t *tab_rp_registers, int energyprice)
{
        if (energyprice < 7)
        {
                if (tab_rp_registers[5] < 6000 & tab_rp_registers[16] > 0)
                {
                        return 100;
                }
                else
                {
                        return 0;
                }
        }
        else if (energyprice > 6)
        {
                if (tab_rp_registers[5] > 3000)
                {
                        return -100;
                }
                else if (tab_rp_registers[5] < 2950 & tab_rp_registers[16] > 0)
                {
                        return 100;
                }
                else
                {
                        return 0;
                }
        }
        else
        {
                return 0;
        }
}

int tesla_client(energyprice)
{
        uint16_t *tab_rp_registers = NULL;
        modbus_t *ctx = NULL;
        int nb_points;
        int rc;
        int n = 0;
        int i = 0;
        FILE *log;
        uint32_t old_response_to_sec;
        uint32_t old_response_to_usec;

restart:
        // Open the communications
        ctx = modbus_new_rtu("/dev/ttyUSB0", 9600, 'N', 8, 1);
        if (ctx == NULL) {
                fprintf(stderr, "Unable to allocate libmodbus context\n");
                return -1;
        }
        modbus_set_debug(ctx, TRUE);
        modbus_set_error_recovery(ctx, MODBUS_ERROR_RECOVERY_LINK |
            MODBUS_ERROR_RECOVERY_PROTOCOL);
        modbus_set_slave(ctx, TESLA_ID);

        modbus_get_response_timeout(ctx, &old_response_to_sec, &old_response_to_usec);
        if (modbus_connect(ctx) == -1)
        {
                fprintf(stderr, "Connection failed: %s\n", modbus_strerror(errno));
```

```
        modbus_free(ctx);
        goto restart;
}

// Allocate and initialize the memory to store the registers
nb_points = (TESLA_REGISTERS_NB > TESLA_INPUT_REGISTERS_NB) ? TESLA_REGISTERS_NB :
    TESLA_INPUT_REGISTERS_NB;
tab_rp_registers = (uint16_t *) malloc(nb_points * sizeof(uint16_t));
memset(tab_rp_registers, 0, nb_points * sizeof(uint16_t));

// Requesting data
printf("--Request Powerwall Data--\n");
rc = modbus_read_input_registers(ctx, TESLA_INPUT_REGISTERS_ADDRESS,
    TESLA_INPUT_REGISTERS_NB, tab_rp_registers);
ASSERT_TRUE(rc == TESLA_INPUT_REGISTERS_NB, "");

// Print the received values
while (n < rc){
        printf("POWERWALL DATA: %0X\n", tab_rp_registers[n]);
        n++;}

// Compute action to be taken
char mode;
int16_t command = 0; // from -100 to +100, percentage of charging capacity that goes
    into the battery
mode = getmode();

switch(mode)
{
case 'B'  : // Full Backup Mode
        command = mode_backup(tab_rp_registers);
        break;
case 'S'  : // Sell Energy at certain threshold
        command = mode_sell(tab_rp_registers, energyprice);
        break;
case 'C'  : // Self Consumption (leave it to the inverter)
        command = 0;
        break;
case 'H'  : // Hybrid Backup-Consumption
        command = mode_hybrid(tab_rp_registers, energyprice);
        break;
        default :
        printf("Invalid mode\n" );
        command = 0;
        break;
}

// Update Webpage
htmldisplay(tab_rp_registers, mode, command, energyprice);

// Log the data
log = fopen("datalog-c.txt","a");
while (i < 24)
{
        fprintf(log, "%" PRId16 " ", tab_rp_registers[i]);
        i++;
}
fprintf(log, "     %d   %c   %d\n", command, mode, energyprice);
fclose(log);

// Give command for test bench
rc = modbus_write_register(ctx, TESLA_ADDRESS, command);
ASSERT_TRUE(rc == 1, "");

// Free the memory
free(tab_rp_registers);

// Close the connection
modbus_close(ctx);
modbus_free(ctx);
```

```
        return 0;
}
```

B.4. main.c

Main.c is used to create the CSS and PHP file and to initialize the other functions.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* BAP Group I: Testing the Tesla Powerwall                      */
/* Bart Kolling & Ludo van den Buijs                             */
/* Main function                                                 */
/* - Create Stylesheet                                           */
/* - Create PHP file                                             */
/* - Initialze "tesla_client"                                    */
/*  - Energyprice Test bench (Only used in final prototype)      */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <modbus.h>
#include <time.h>
#include "testbench.h"
#include "testbench-client.c"

// Create the Stylesheet
void css_create(void)
{
        FILE * fp;
        fp = fopen("/var/www/html/mystyle.css", "w+");

        printf("Creating stylesheet!");
        if(fp == 0)
        {
                printf("Could not create stylesheet!");
        }

        fprintf(fp, "#header {");
        fprintf(fp, "background-color:lightgray;");
        fprintf(fp, "color:crimson;");
        fprintf(fp, "text-align:center;");
        fprintf(fp, "padding:5px;}");
        fprintf(fp, "#nav {");
        fprintf(fp, "line-height:30px;");
        fprintf(fp, "background-color:lightgray;");
        fprintf(fp, "height:300px;");
        fprintf(fp, "width:142px;");
        fprintf(fp, "float:left;");
        fprintf(fp, "padding:5px; }");
        fprintf(fp, "#navr {");
        fprintf(fp, "line-height:30px;");
        fprintf(fp, "background-color:lightgray;");
        fprintf(fp, "height:300px;");
        fprintf(fp, "width:142px;");
        fprintf(fp, "float:right;");
        fprintf(fp, "padding:5px; }");
        fprintf(fp, "#sectionl {");
        fprintf(fp, "padding:10px;");
        fprintf(fp, "text-align:center;");
        fprintf(fp, "float:left;}");
        fprintf(fp, "#sectionr {");
        fprintf(fp, "padding:10px;");
        fprintf(fp, "text-align:center;");
        fprintf(fp, "float:left;}");
        fprintf(fp, "#footer {");
        fprintf(fp, "background-color:black;");
        fprintf(fp, "color:white;");
        fprintf(fp, "clear:both;");
```

```
        fprintf(fp, "text-align:center;");
        fprintf(fp, "padding:5px; }");
        fprintf(fp, "body {background-color: whitesmoke;}");
        fclose(fp);
}

// Create the PHP file
void php_create(void)
{
        FILE * fp;
        fp = fopen("/var/www/html/test.php", "w+");

        if(fp == 0)
        {
                printf("Could not create PHP!");
        }

        fprintf(fp, "<!DOCTYPE html>\n");
        fprintf(fp, "<html>\n");
        fprintf(fp, "<body>\n");
        fprintf(fp, "<?php \n");

        fprintf(fp, "chmod(\"/var/www/html/userpref.txt\", 0777);\n");
        fprintf(fp, "$fp  = 'userpref.txt';\n");

        fprintf(fp, "file_put_contents ( $fp, $_GET[\"mode\"]);");
        fprintf(fp, "header(\"Location: {$_SERVER[\"HTTP_REFERER\"]}\");");
        fprintf(fp, " ?>\n");

        fprintf(fp, "</body>\n");
        fprintf(fp, "</html>\n");
        fclose(fp);
}

int main()
{
        int energyprice = 1;
        int priceswitch = 0;

        // Initialize the files
        css_create();
        php_create();

        while(1){

                // Energy Price Fluctuation: Only used in final prototype
                switch(priceswitch)
                {
                case 1:
                        if (energyprice > 2)
                        {
                                energyprice = energyprice - 1;
                        }
                        else
                        {
                                priceswitch = 0;
                        }
                        break;

                case 0:
                        if (energyprice < 13)
                        {
                                energyprice = energyprice + 1;
                        }
                        else
                        {
                                priceswitch = 1;
                        }
                        break;
                }
```

```
                // Start the communications
                tesla_client(energyprice);
                sleep(1);
        };


        return 0;
}
```

B.5. htmldisplay.h

This header file is used for htmldisplay.c.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* BAP Group I: Testing the Tesla Powerwall    */
/* Bart Kolling & Ludo van den Buijs           */
/* Header file of "htmldisplay.c"              */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef HTMLDISPLAY_H
#define HTMLDISPLAY_H

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

void htmldisplay(uint16_t variables[], char mode, int16_t command, int Energyprice);

#endif
```

B.6. htmldisplay.c

Thes script is used to host the user interface including the retrieved variables and the mode selection form.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* BAP Group I: Testing the Tesla Powerwall                */
/* Bart Kolling & Ludo van den Buijs                       */
/* Hosting the webpage                                     */
/* - initialize the webpage                                */
/* - keep refreshing the webpage with new data             */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#include "htmldisplay.h"

#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <modbus.h>

void htmldisplay(uint16_t variables[], char mode, int16_t command, int Energyprice)
{
        printf("Creating Webpage!\n");
        FILE * fp;
        fp = fopen("/var/www/html/index.html", "w+");

        if(fp == 0)
        {
                printf("Could not create page!");
        }

        const char *regname[24];
```

```
regname[0] = "XX";
regname[1] = "XX";
regname[2] = "XX";
regname[3] = "XX";
regname[4] = "XX";
regname[5] = "XX";
regname[6] = "XX";
regname[7] = "XX";
regname[8] = "XX";
regname[9] = "XX";
regname[10] = "XX";
regname[11] = "XX";
regname[12] = "XX";
regname[13] = "XX";
regname[14] = "XX";
regname[15] = "XX";
regname[16] = "XX";
regname[17] = "XX";
regname[18] = "XX";
regname[19] = "XX";
regname[20] = "XX";
regname[21] = "XX";
regname[22] = "XX";
regname[23] = "XX";

fprintf(fp, "<!DOCTYPE html>");
fprintf(fp, "<html>");

fprintf(fp, "<head>");
fprintf(fp, "<meta http-equiv=\"refresh\" content=\"3\" >");
fprintf(fp, "<link rel=\"stylesheet\" href=\"mystyle.css\">");
fprintf(fp, "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">
    ");
fprintf(fp, "<link rel=\"stylesheet\" href=\"http://code.jquery.com/mobile/1.4.5/
    jquery.mobile-1.4.5.min.css\">");
fprintf(fp, "<script src=\"http://code.jquery.com/jquery-1.11.3.min.js\"></script>");
fprintf(fp, "<script src=\"http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.
    min.js\"></script>");
fprintf(fp, "</head>\n");

fprintf(fp, "<body>\n");

fprintf(fp, "<div id=\"header\">");
fprintf(fp, "<h1>Tesla Powerwall</h1>");
fprintf(fp, "</div>");

fprintf(fp, "<div id=\"sectionl\">");
fprintf(fp, "<h1>System information</h1>\n");

int n = 0;
while (n<24){
        fprintf(fp, "<p style=\"margin-top: 0; margin-bottom: 0;\"> %s = %d </p>\n",
            regname[n], variables[n]);
        n++;
}
fprintf(fp, "<h1>Relevant Test Information</h1>\n");
fprintf(fp, "\n<p> Current Mode = %c </p>\n", mode);
fprintf(fp, "<p> Energy Price = %d </p>\n", Energyprice);
fprintf(fp, "<p> Send Command Signal = %d", command);
fprintf(fp, "<p style=\"margin-top: 0; margin-bottom: 0;\"> %s = %d </p>\n", regname
    [5], variables[5]);
fprintf(fp, "</div>");

fprintf(fp, "<div id=\"sectionr\">");
fprintf(fp, "<h1>User Preferences</h1>");

fprintf(fp, "<form action=\"test.php\" method=\"GET\">");
fprintf(fp, "<input onChange='this.form.submit();' type=\"radio\" name=\"mode\" value
    =\"S\" > Smart Selling (S)<br> <?php echo $_SERVER[\"REQUEST_URL\"]; ?> ");
fprintf(fp, "<input onChange='this.form.submit();' type=\"radio\" name=\"mode\" value
    =\"C\"> Self-Consumption (C)<br> <?php echo $_SERVER[\"REQUEST_URL\"]; ?> ");
```

```
        fprintf(fp, "<input onChange='this.form.submit();' type=\"radio\" name=\"mode\" value
            =\"B\"> Back Up (B)<br> <?php echo $_SERVER[\"REQUEST_URL\"]; ?> ");
        fprintf(fp, "<input onChange='this.form.submit();' type=\"radio\" name=\"mode\" value
            =\"H\"> Hybrid (H) <?php echo $_SERVER[\"REQUEST_URL\"]; ?> ");
        fprintf(fp, "</form>");

        fprintf(fp, "</div>");

        fprintf(fp, "</body>");
        fprintf(fp, "</html>");
        fclose(fp);
}
```

Bibliography

[1] M. Rekinger, F. Thies, G. Masson, and S. Orlandi, “Global market outlook for solar power 2015-2019,” tech. rep., SolarPower Europe and Becquerel Institute, jun 2015.

[2] International Energy Agency, *Snapshot of Global PV Markets*, 2014.

[3] G. B. Paul Denholm, Matthew O’Connell and J. Jorgenson, “Overgeneration from solar energy in california: A field guide to the duck chart,” tech. rep., National Renewable Energy Laboratory, nov 2015.

[4] Holland Solar, *Regelgeving rond salderen in Nederland*, mar 2016.

[5] California Independent System Operator (CAISO), *Fast Facts: What the duck curve tells us about managing a green grid*, 2016.

[6] Tesla Motors Inc., *Tesla Powerwall*, 2015. https://www.teslamotors.com/.

[7] C. Taylor, *Elon Musk unveils Tesla Powerwall batteries to ’change the world’*. Mashable, may 2015.

[8] A. Mamiit, *Mercedes-Benz Unveils Personal Power Pack: How Does It Compare With Tesla’s Powerwall?* Tech Times, jun 2015.

[9] B. Jiang and Y. Fei, “Smart home in smart microgrid: A cost-effective energy ecosystem with intelligent hierarchical agents,” *IEEE Transactions on Smart Grid*, vol. 6, pp. 3–13, Jan 2015.

[10] A. L. Dimeas and N. D. Hatziargyriou, “Operation of a multiagent system for microgrid control,” *IEEE Transactions on Power Systems*, vol. 20, pp. 1447–1455, Aug 2005.

[11] D. Livengood and R. Larson, “The energy box: Locally automated optimal control of residential electricity usage,” *Service Science*, vol. 1, no. 1, pp. 1–16, 2009.

[12] “Five global trends shaping the future of energy white paper,” tech. rep., Schneider Electric Professional Services, 2013. [Accessed 22 April 2016].

[13] “Buying energy is unique: A look at the complexities of energy procurement,” tech. rep., Schneider Electric Professional Services, 2013. [Accessed 22 April 2016].

[14] J. Kumagai, “The rise of the personal power plant.” IEEE Spectrum, May 2014. [Accessed 22 April 2016].

[15] J. Lipp, “Lessons for effective renewable electricity policy from denmark, germany and the united kingdom,” *Energy Policy*, vol. 35, no. 11, pp. 5481 – 5495, 2007.

[16] V. Smil, “A skeptic looks at alternative energy,” jun 2012.

[17] SolarEdge Technologies, Inc., *SolarEdge StorEdge Interface Datasheet*, October 2015.

[18] SolarEdge Technologies, Inc., *SolarEdge Single Phase Inverters Datasheet*, December 2015.

[19] Continental Control Systems, LLC., *WattNode Modbus Installation and Operation Manual*, December 2011.

[20] Eneco Installatiebedrijven, *Tekening 0012257-200-00, Aansluitingen*, apr 2016.

[21] Texas Instruments, *RS-422 and RS-485 Standards Overview and System Configurations*, May 2010.

[22] Modbus.org, *MODBUS over Serial Line Specification and Implementation Guide V1.02*, December 2006.

[23] MODICON, Inc., Industrial Automation Systems, One High Street North Andover, Massachusetts 01845, USA, *Modicon Modbus Protocol Reference Guide, PI–MBUS–300 Rev. J*, June 1996.

[24] Tesla Motors, Inc., 3500 Deer Creek Road, Palo Alto, CA 94304, *Powerwall Installation and User's Manual*, 2015.

[25] Hard Kernel Ltd., 704 Anyang K-Center, Gwanyang, Dongan, Anyang, Gyeonggi, South Korea, 431-815, *USER MANUAL ODROID-C1*, 2015.

[26] Future Technology Devices International Ltd, *USB-COM485-PLUS1*, October 2010. Version 1.21.

[27] S. Cheshire and M. Krochmal, "Multicast dns," February 2013.