Control Network of Bi-Directional DC/DC Converters

Maxim Marchal, Matthijs Poot, Martijn Vermeulen

Bachelor Graduation Thesis

TU Delft Delft University of Technology

Control Network of Bi-Directional DC/DC Converters

Bachelor Graduation Thesis

Maxim Marchal, Matthijs Poot, Martijn Vermeulen
4360842, 4342569, 4390784

June 19, 2017

Faculty of Electrical Engineering, Mathematic and Computer Science (EEMCS)
Delft University of Technology


TU Delft
Delft
University of
Technology

Executive Summary

This document is the bachelor graduation thesis of BAP Group B1. Together with BAP Group B2, the objective of this project was to create a network of bi-directional flyback DC/DC converters. This document is specifically concerned with the control network, which must read the current and voltage signals from the converter and display it on a website. On this website the user is able to change the power flow of the converters.

To accomplish this, the control network consists of three parts: Server, Master and Slave. The slave is a microcontroller connected to the power converter which reads and sends control signals from and to the converter; it is able to change the outputs of the converter. Next, the master is a mini computer which is the commander in chief of the network. All microcontrollers are connected to this unit, and the master passes this data through to the server. The server shows all data acquired from each individual slave and the user can change the power output of all the power converters connected to the designed system.

To check these features, five design requirements need to be met: Robustness, Modularity, Expandability, Interactivity, Power Management. Robustness is the system being able to run for years without failure. Modularity; connecting new microcontrollers with simple automatic setup. Expandability; future proofing the network so other teams can take over and expand the system and the ease of reproducing the designed system. Interactivity; which is visualizing all commands and output of the system for the user to easily understand. Finally, power management; the system being able to control the power flow of all connected DC/DC converters and the power throughput in the entire system.

The design of the control network needs to account for the design requirements, therefore several decision were made: The slave microcontroller was chosen based on group experience to ensure progress and the programming language C so future groups could easily continue expanding with a very capable language. Furthermore, the slave can decide what to do with the connected device, thus changing its power output, using a Finite State Machine, independently of the master. Furthermore, the communication protocol between the slave and master is I^2C due to its simplicity and flexibility as it communicates using only a clock and data signal. Finally, the data retrieved by the master should be uploaded every second to the server to ensure there was 'real time' output for the end user.

The delivered prototype is robust, expandable, and resembles the designed system. However, the system was not tested in combination with the converters power connected to the slave microcontrollers. To account for this fact, several tests were done using similar output characteristics as the converter. Therefore, the system is able to perform its core tasks and thus, a frame work for future research is completed.

Table of Contents

Problem

1-1 Problem Scope

Historically, in electric grids, energy demands were small and predictable, while the energy supplies were large. Nowadays, new complex, unpredictable energy sources like solar panels and windmills are added to the system and the energy consumption is rising more and more, while the energy infrastructures are still the same. Loads connected to the network assume that the grid is an infinite bus, meaning it can provide an infinite amount of power. However, when too many loads are connected and the consumption exceeds the supply, the energy system can start malfunctioning or even have a blackout. Current electric grids do not make use of all the technologies available in the 21st century, which could prevent such malfunctioning and create better energy systems.

In new systems, not only power, but also information is exchanged in the infrastructure. This allows analysis of the power flow, and control and scheduling of the enclosed system. Energy demands can be adjusted to meet the available supplies, whereas in the current energy systems, demand is not controllable.

This Bachelor Graduation Thesis focuses on a control network of DC/DC power converters. The practical goal will be to create a system that is able to monitor and control an electric bike charging station at the University of Technology in Delft. This involves designing a system that can potentially:

- Control power converters with control signals coming from microcontrollers.
- Connect all the microcontrollers to one master computer.
- Control energy throughput over all power converters.
- Output data of all statuses of the converters externally and saving the data for analysis.
- Control all power converters externally, using the master and a web server.

1-2 Technical Review

1-2-1 Energy infrastructures

Currently, in the electrical grids, big power plants deliver the power to the electrical grid [1]. The AC power delivered from the plants is transformed into a lower voltage numerous times, until it reaches the customer. No information about demand is transferred back to the suppliers, thus, current electrical grids are a one-way flow system.

1-2-2 Energy control networks

In the last few years, more attention is paid to integrating more information into the power grids. Some even try to integrate electricity smart grids into an overall smart energy system [2] and show the benefits of such a smart multi-energy grid [3]. Also web-based dispatcher information systems [4] have been presented before.

1-2-3 Ensuring network reliability

When designing a smart grid, one must regulate the charging to prevent unexpected problems such as continuous charging of the system, which causes stress on the entire network for too many connected devices. In the paper 'Smart EV charging schedules' [5] several intelligent algorithms are already explored with time scheduled charging. Furthermore, using *Vehicle to Grid power* suggests discharging the Electronic Vehicle (EV) when unused such as 3 AM in the morning, to add additional power to the grid. And charging the EV back when the grid is more 'relaxed', when there is more available power such as the sun powering the solar panels. Thus reduce the maximum level of stress on the network.

1-2-4 Data Visualization

The mini computer gathers information from all the slave units and can produce several interesting figures for both the user and the network administrator. One of the goals of this project is to provide insight into the information that is gathered.
Data visualization software and hardware for energy is abundantly present on the market; to name a few: Toon from Eneco [6] and the Nest smart thermostat [7]. These devices make electricity and gas consumption visually attractive and give customers valuable insight in their energy consumption.
During the previous year, electrical engineering students at the TU Delft preceded this group and already did the ground work for the master controller and data visualization. The server part of the project aims to optimize what has already been done using previous years work [8] [9] [10].

1-3 Design Requirements

The control network will be designed with the following five requirements:

- Robustness
- Modularity
- Expandablity
- Interactivity
- Power Management

Here follows a description of what is required of each and if possible, how it can be measured:

1-3-1 Robustness

The control system of itself is relatively easy to build, however, the challenge of the project is in the robustness of the system. One of the less obvious challenges of making this system work is ensuring reliability for an indefinitely long period of time. Other groups using the master computer have encountered issues involving the system crashing after a time period of typically $T_{crash} = 3 - 4$ hours [8][9][10]. Special care has to be taken when it comes to *error handling* and *memory management*. Error handling is the ability to deal with exceptions, for example a disruption in the internet connection of the devices. Memory management is primarily concerned with small memory leaks, which (in the long run) will accumulate and cause the system to run out of memory. [11] and [12] provide information about how to approach this problem.

Since the master computer is in use for design and testing purposes most of the time, it is impractical to let it run for multiple weeks or months without human intervention. However, it is sensible to state that if T_{crash} is above a certain threshold value, its value will tend to infinity. Taking into account the typical value of T_{crash} of the previous groups (three to four hours), the threshold value is set to $25 \cdot T_{crash,typ}$, which is one hundred hours.

1-3-2 Modularity

The system must be able to connect at least ten slaves and must do so without human intervention (except for someone who connects a power cable to each slave). The first application the system will be used in is the electric bike solar charging station, which is already built in front of EWI at the TU Delft. The amount of ten slaves is derived from a reasonable maximum amount of bicycles connected to an e-bike charging station. Having the ability to add and remove parts of the system on the go without having to redesign it, is advantageous, because it creates flexibility and reduces maintenance costs.

1-3-3 Expandability

This project is part of the Solar E–bike station led by Dr. Ir. P. Bauer [13]. Therefore it is important for future project to improve and increase the number of functions. A new team must be able to quickly understand the system, thus the code must be easily readable and expandable. This means all design choices should take expanding the system as the main criteria.

Figure 1-1: Solar E-bike Station at the TU Delft

1-3-4 Interactivity

For the purpose of the E–bike station, it is useful if the voltage and current outputs of the converters can be retrieved remotely through an internet connection. The requirement for the interactivity of the prototype is that anyone with an internet connection can retrieve the voltages and currents of all the converters that are connected, as well as modify the state of the converters (On, Off, and the direction of power flow).

Additionally, when a consumer charges his/her bike, it should be easy to understand the features of the system. Because this thesis is not part of a bachelor of arts, the main focus will be functionality and ease of use, without complicated and subjectively attractive control panels.

1-3-5 Power Management

Today, the regulatory systems adhere the principle "supply-follows-demand" [14]. This essentially involves 'demand prediction'. Such a system is a one way system with generation planning. However, with limited energy certainty, this principle can put a lot of strain on the power source, like a battery, if a large amount of power is required. Therefore, the solution is "demand-follows-supply", where the control system regulates the output based on the available supply of energy in the electrical system.

To test this requirement, the system must be able read out and adjust the maximum power output per microcontroller and change the output of the all other slaves accordingly either by user or autonomously. This feature will ensure the possibility of future, proper power management in systems based on the designs proposed in this project.

Design

2-1 Overview

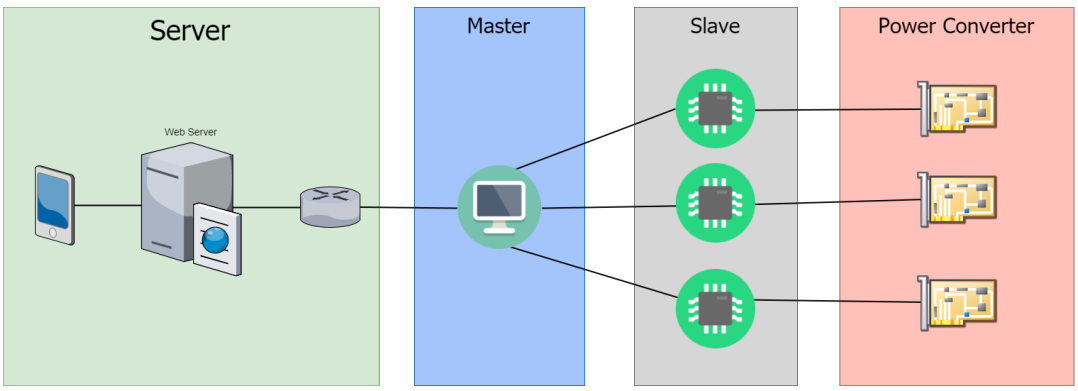
The network of bi-directional flyback DC/DC converters consists of four parts:

1. Bi-directional Flyback DC/DC Converter [15], a printed circuit board (PCB)
2. Slave Microcontroller
3. Master Single Board Computer
4. Server

This thesis exists of points two to four, with the power converter being designed by the other half of the group. This project is a control network of slave microcontrollers which regulate the power output of the converters. The power regulation can be done either by the slave itself or it can be overwritten by the master computer. The master, acting as a central hub, connects to all slaves and processes the gathered information from all the slaves and sends this information to a web server, which is accessible using an Internet connection. The server is also capable of sending commands to the master, which on its turn will send instructions to the slave. Via a website, a user can control the power output of all converters connected and read how much current and voltage is flowing through the converters.

A detailed design description of each individual component will now follow.

Figure 2-1: Overview of designed system



2-2 Slave

The core task of the microcontroller is to control the currents and voltages that flow through the bi–directional flyback converter as well as providing information about the device that is connected to the converter. The microcontroller should control the PCB output and find out the current State of Charge (SOC) or proper nominal power consumption of the device attached to the PCB.

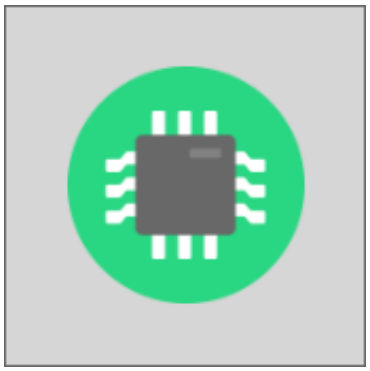


Figure 2-2: The 'slave', as part of the system in figure 2-1

To perform its task, the microcontroller must:

- Be able to receive and interpret commands from a master (section 2-2-2).
- Provide the master controller with measurements of the input and output currents and voltages that are present on the converter PCB (section 2-2-3).
- Provide the bi–directional flyback converter with necessary control signals to alter the power flow (section 2-2-4).
- Provide information about the SOC of the device that is connected to the converter (section 2-2-5).

In this chapter, the implementation of these tasks is described in detail. Design considerations that have been made are mentioned explicitly, including the selection of a suitable microcontroller.

2-2-1 Selecting a Microcontroller Unit

As shown in figure 2-1, there is communication between the slave and the power converter of BAP Group B2. Between these two system is were this project is divided. To be exact, the decision was made to let the control network design all digital signals, and the power team to design all analog signals.

In order to perform its core tasks, the microcontroller Unit (MCU) must feature Pulse Width Modulation (PWM) hardware, due to the features explained in section 2-2-4, as well as an Analog to Digital Converter (ADC) module, because of the input signals, explained in section 2-2-3. Hardware that facilitates communication is also required. MCUs with a large userbase are preferred, due to the fact that this makes it convenient for other engineers to familiarize themselves with the software and thus be able to create designs with this project at its core. There is a plethora of options available when it comes to picking one that is suitable for this purpose. Some possibilities are presented in the section below.

Options

The **TI C2000** family is attractive due to the fact that it makes use of Control Loop Acceleration to service time-critical control loops, while the main CPU is free to perform miscellaneous tasks

such as communication and diagnostics [16]. Furthermore, Texas Instruments provides software for this board that is especially designed for control applications [17]. The TMS320F2027 board of this family features seven 12–bit ADC channels and 4 PWM–channels.

Another possibility is the **Arduino Nano** [18], which features eight 10–bit ADC–channels and 6 PWM–channels. The advantage of this unit is that it has a very large userbase. This is advantageous because code written by the Arduino community can be used. The availability of many Arduino libraries makes this option viable. A downside to this option is the fact that Arduinos are programmed in a high level language; this makes it harder to understand the inner workings of the written code. Besides, the Arduino libraries are not always as efficient and waste RAM and CPU cycles [19].

NXP Semiconductor's **LPC1343** (an ARM cortex M3 board) is an option as well. The LPC1343 is a basic 32 bit MCU with 8 ADC-channels and 5 PWM–channels. It is programmable through a CodeBlocks plugin via a USB connection. While the quality of this board does not soar over the other options, it is chosen because of the extended experience the project group has with it. The datasheet for this microcontroller can be found in [20]. The fact that this microcontroller is used in the Electrical Engineering Bachelor program at the University of Technology in Delft makes it a convenient choice for future projects based upon this project, as all students will have experience with this type of controller.

The LPC1343 is programmed in C [21], a low–level language that is versatile, fast and highly portable. Since most microcontrollers are able to use C, the decision to use this language was made, to make it possible to deploy the code to other microcontrollers as well, while only needing to make minor changes in the setup. Compilation of the code is done with a GCC compiler for ARM which is embedded into Code::Blocks (an integrated development environment). Programming is done by flashing the machine code onto the board using special executables. The GCC compiler, Code::Blocks plugin and the special executables are all provided by the Computer Architecture and Organisation (EE3D11) course lab of the TU Delft [22]. Finally, due to the requirement of scalability, future versions must be able to download and use the written code.

2-2-2 Communication with the Single Board Computer

I^2C

The communication method that is used for this task is the Inter Integrated Circuits (I^2C) protocol [23]. This is a two–wire communication protocol where one controller can issue commands to a number of listeners. In I^2C lingo, these are called *master* and *slave* respectively.

There is a number of reasons why I^2C is chosen as the communication method. Firstly, it has a very low overhead since the setup involves connecting just two wires and a common ground. Furthermore, by exploiting the fact that the wires are connected via a pull up resistor to V_{DD}, a high number of slaves can be pooled together on the same bus. In the used configuration, more than the ten required slaves, a requirement from section 1-3, can be controlled by the single board computer.

One limitation of the usage of I^2C is the limit of one byte per package. This is accounted for by using a custom designed I^2C protocol. This protocol makes the sender split the large, multiple–byte integer in parts of one byte. These bytes are saved in a buffer. When all the data needed is sent and received, the different bytes are combined to integers with more bits.

Master Requests and Commands

The master is capable of requesting data from slaves as well as writing data to them. The first byte that is sent to a slave is always a state byte. This informs the slave about the intentions of the master. Then, a byte is sent by either the slave or the master (depending on what the state byte was). This byte contains the length of the information that is to be sent or received. Subsequent bytes that travel along the I^2C bus contain the necessary data. When the full message is sent, the data is processed by either the master or the slave.

Dynamic address allocation

The communication between the master and its numerous slaves is designed with modularity as its foundation. Any given converter should be able to connect to a network that has a master regulating the I^2C bus, even if many slaves are already connected. Therefore, it is necessary to implement a dynamic address allocation protocol. Normally, an address is assigned to a listening device when manufactured, but when multiple devices with the same address are connected and listening on the same bus, data transfer is corrupted. The master should assign an unused address to a listening device when it first connects. The slave microcontrollers are programmed to listen to the same address at first. Then, when it connects to the master, the master will send a new, unused address back to slave. The slave will then change the address it is listening to, to the newly sent address.

One foreseen problem with this technique, is that upon connecting two new slaves at the same time, there is a chance both will receive the same address making future communication with either of these two slaves impossible. A possibility to diminish the probability of occurrence of the above-mentioned situation, is to limit the time period in which the slave is listening to the master to some smaller period with random time intervals. When multiple slaves are connected at the same time, but start listening at random intervals, the chance of these periods of the distinct slaves colliding will decrease. Random time intervals in between communication signals are widely applied in combination with collision detection in various communication networks and has been shown to be a reliable protocol, even when a high amount of slaves are trying to connect [24].

For this design, it is chosen not to implement the option of time intervals for listening to the master. It is assumed that the users of the first prototypes will be well informed of this issue, as this problem only occurs when first connecting the converter and not the devices connected to the PCB. Not implementing the random time periods on its part removes possible increase in detection time. The possible increase in detection time is caused by the facts that the master will not be scanning for new slaves continuously. Consequently, it is possible that the slave microcontroller was not listening to the I^2C bus at the moment the master tried to connect.

Possible commands

The custom I^2C protocol that was developed for this project features several commands that the slave software recognizes. These can be separated in two different categories: commands in which information flows from the slave to the master and commands where it flows the other way around.

Master to Slave commands occur when the master wishes to control certain variables in the slave. All commands of this type are listed below:

State	Description
VOLT_READ_STATE	The *maximum* output voltage is set equal to the payload received
CURRENT_READ_STATE	The *maximum* output current is set equal to the payload received
TURN_ON_OFF	The master turns the slave on or off
MASTER_SET_DIRECTION	The master sets the direction of power flow on the converter.
SET_PWM_X_STATE	Overwrites the duty cycle of either PWM–channel 1 or 2.
NEW_ADDRESS_STATE	The 7–bit address of the I^2C–module is set equal to the payload. This is illustrated further in section 2-2-2

Slave to Master commands occur when the master wishes to retrieve information that is gathered by the slave. All commands of this type are listed below.

State	Description
READ_POWER_STATE	The slave tells the master whether it is turned on or off
READ_PWM_X_STATE	The duty cycle of PWM signal X is transmitted to the master
VOLT_X_WRITE_STATE	The voltage at the input of the converter (X = 1) or the output of the converter (X = 2) is transmitted to the master
CURRENT_X_WRITE_STATE	The current at the input of the converter (X = 1) or the output of the converter (X = 2) is transmitted to the master
READ_CHARGE_STATE	The state in which the FSM is (see section 2-2-4 is transmitted to the master
READ_POWER_STATE	The slave tells the master whether the converter is turned on or off
READ_PWM_X_STATE	The duty cycle of PWM–channel 1 or 2 is sent to the master.

2-2-3 Measuring Voltages and Currents

The converter PCB provides the microcontroller with four separate analog signals between 0 and 3.3V. These signals represent the following physical quantities:

1. Input voltage (V_{in}).
2. Output voltage (V_{out}).
3. Input Current (I_{in}).
4. Output Current (I_{out}).

The true currents that are flowing are related to the voltage signal through a transresistance value $R_{measure}$. The true voltages on the terminals are related to the voltage signal through a constant $K_{measure}$. The signals are converted to the digital domain via the MCU's ADCs.

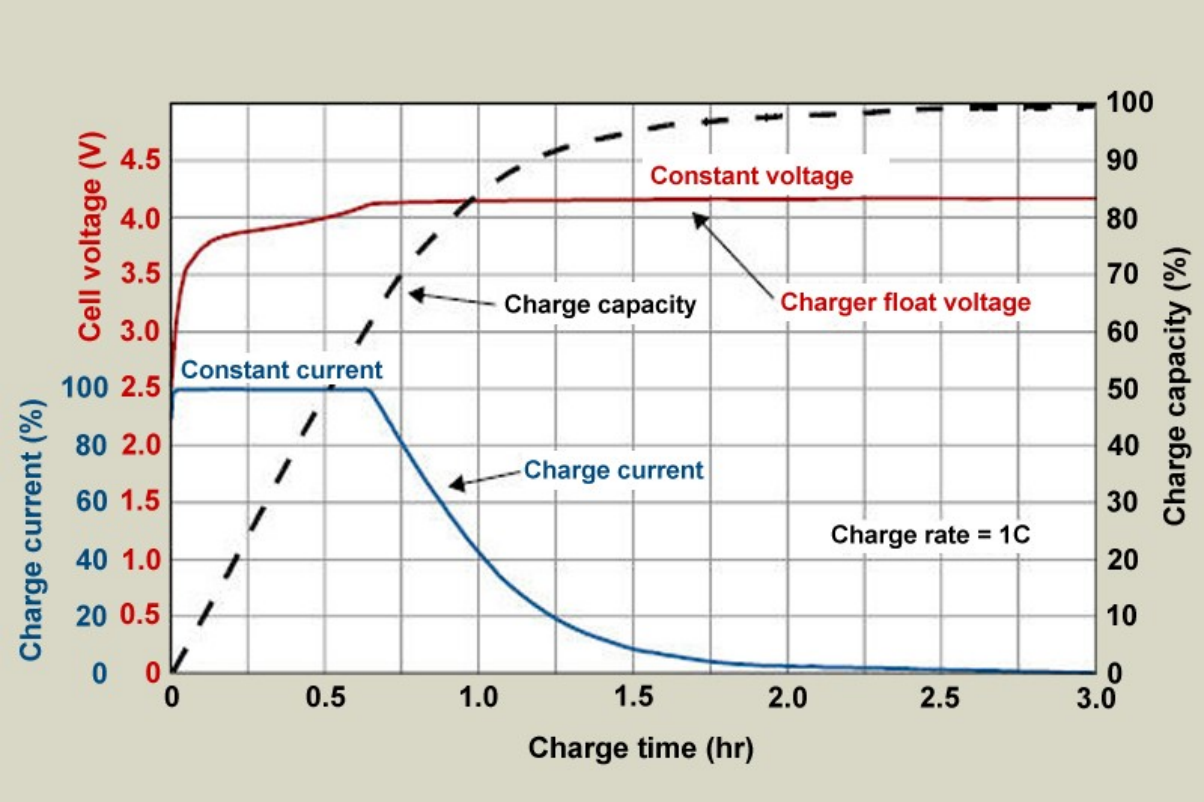


Figure 2-3: I-V curves of a typical Lithium–ion battery. Adapted from [25].

The Olimex LPC1343 comes packed with an ADC with a resolution of at most 10 bits. The software can connect the ADC to 8 different channels. The sampling frequency is adjustable, but is limited to 409 kHz.

The noise introduced by quantization is uniformly distributed across $\left[-\frac{LSB}{2}, +\frac{LSB}{2}\right]$. The voltages on the converter are between 0 and 50 Volts. The Least Significant Bit (LSB) is calculated using equation 2-1.

$$LSB = \frac{|V_{max} - Vmin|}{2^N} \tag{2-1}$$

where N is the number of bits. With $V_{max} = 3.3V$ and $V_{min} = 0V$, this equation evaluates to an ADC resolution of 49 millivolts. The noise introduced by quantization can thus be modeled as a uniform distribution with $\mu = 0\ V$ and $\sigma^2 = 2.0 \times 10^{-4}\ V^2$. This fact is later exploited in section 2-2-5.

Since the microcontroller is operating near a DC/DC converter that is operating at high frequencies, it is possible that the sensors pick up high frequency noise. This can destabilize the control FSM, which is undesirable. Therefore, the sensor values are filtered by summing over the last N samples and taking the average. This is sometimes called a moving average filter.

2-2-4 Control Signals

Composition of signals

When the controller receives all the incoming voltage and current measurements, the correct output control signals need to be defined. These output control signals will be sent back to the

power converter so that the PCB regulates the power flow correctly. The integrated circuits on the PCB that regulate the power flow, require a DC voltage signal as control. This signal will be compared to a reference voltage. The resulting output will be a portion of the maximum available output. This portion of output is related to the ratio of the control signal and the reference signal given to the integrated circuit on the PCB.
The chosen microcontroller is not able to output a continuous analog voltage. Instead, a digital PWM signal is sent to one of the controllers output. The digital signal is then connected to a digital to analog converter (DAC) on the PCB, which will give an analog reference control signal. By changing the duty cycle of the PWM signal, the analog output of the DAC is changed to the average value across one period of the PWM signal [26].
Bidirectionality is enabled by switching the input and output depending on the *direction* variable. This way, power can flow in both directions depending on the value of a user–defined variable.

Determining correct output

When a voltage is sensed at the terminals of the converter, this means a device was connected. At this point, there are two possibilities. The first possibility is that the load is purely resistive, meaning it will have a constant voltage–current relationship when a voltage is applied. This type of load requires a constant DC–voltage.
When a battery is connected, the process requires a more sophisticated control system. Figure 2-3 shows the charging pattern of a typical Li–ion battery cell. When voltage is low, it can be seen that the current supplied is constantly high. The voltage rises during this time and when it reaches the threshold value, the current is throttled. This is called 'Constant Current, Constant Voltage Charging' and it is the foundation for the control system that is employed in the system.

The diagram of figure 2-4 shows the charging algorithm that is running on the microcontroller. The states are to be interpreted as follows:

The **reset** state is where the converter is idle. The output current is set to 0. Whenever the Open Circuit Voltage exceeds the threshold value, this means that a load has been connected to the converter. The next state will then be **V_ramp**, and charging will commence.

During the V_ramp state, the voltage is incremented over time. If the current reaches the maximum allowable value, V_nominal is entered. In **V_nominal**, the output voltage is kept constant. This is where most of the charging happens. When the output current dips below a certain threshold value, **V_pinch off** is entered. This state transition happens in figure 2-3 at $ChargeTime = 1\text{hr}$.

V_pinch off is what is called the Constant Voltage state in most literature. This name is not used here, because the voltage of the charging battery is increasing, albeit slowly. Only when the battery is fully charged, the voltage will be constant. This event (the voltage being constant) is detected by using a technique as described in algorithm 1. This algorithm counts how many consecutive cycles the voltage of the load has been steady. When this has happened, the state will change to **Done**. When the control system is in the state **Done**, this indicates the charging process is finished. The current is set to zero so that no overcharging occurs. When the charged device is disconnected from the converter, the system returns to the **reset** state.

Algorithm 1 Tracking the output voltage

$V_{track} \leftarrow N$
while State $= V_{nominal}$ **do**
  if $V_{now} \leq V_{previous}$ **then**
    $V_{track} \leftarrow V_{track} - 1$
  end if
  if $N = 0$ **then**
    $state \leftarrow V_{nominal}$
  end if
end while

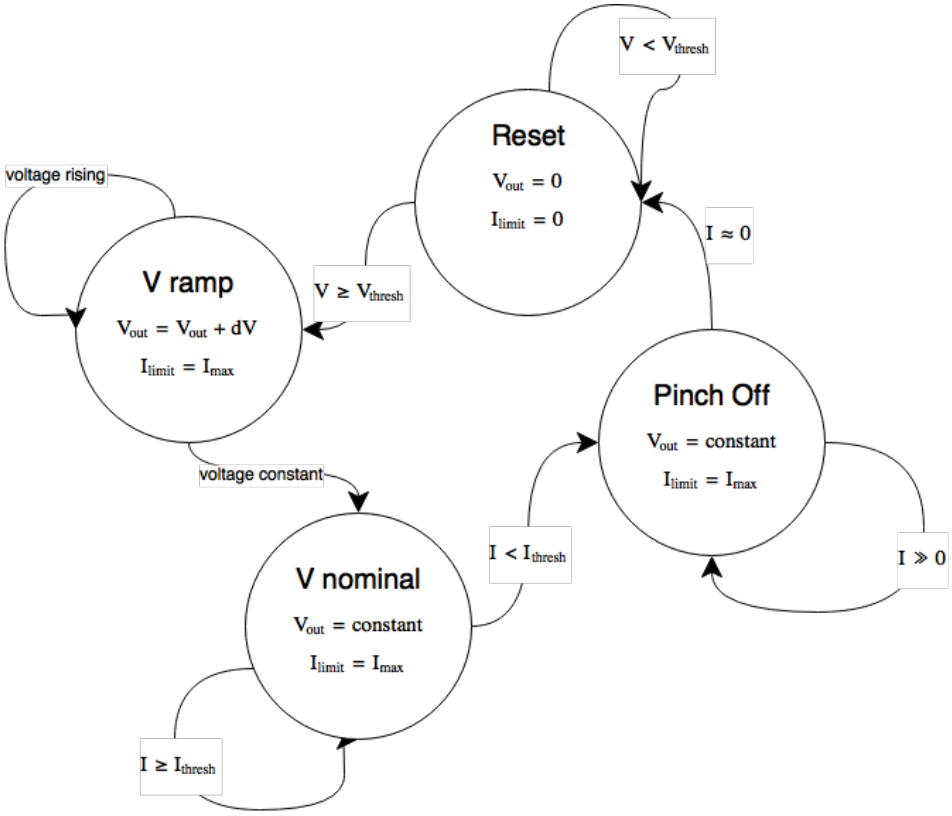


Figure 2-4: Finite State Diagram of the charging process.

2-2-5 State of Charge Determination

The SOC of a battery system is information of considerable importance, since it provides users with information about how much energy they have received from the system. Several methods for determining the SOC have been proposed in the past. These include: Coulomb counting, Kalman Filtering and the voltage method [27]. With state of the art technology, a mean error of 3% is achievable [28].

Kalman Filtering

The Kalman Filter is an algorithm to estimate the inner states of a dynamic system [29]. It requires a suitable model for the battery and a precise identification of the parameters. These parameters are not available during this project, which is the reason why the Kalman Filter will not be used to determine the SOC.

Voltage Method

The SOC of a battery can be determined using a discharge test under controlled conditions. The reading of the battery voltage can be converted to an equivalent SOC using a known discharge curve. However, this voltage is dependent on the battery current due to the battery's electrochemical kinetics and temperature [27]. Measuring the battery's temperature is a convoluted process and it is not worth considering when other methods are available.

Coulomb Counting

By integrating the current that is injected into the battery pack, the total charge that is supplied to the battery can be calculated, which provides information about difference between the initial and final SOC. This method has a disadvantage: offsets in sensor values can accumulate over time due to the integration of this value. Another problem is that the initial SOC is not determined by knowing only the integrated current. Nevertheless, this method is easy to implement and requires no knowledge about the temperature and other specific parameters of the battery. By determining the error that is introduced by the accumulation of sensor-inaccuracy and offset, it can be decided whether this method is feasible.
From section 2-1, it was concluded that the error introduced by quantization is uniformly distributed with $\mu = 0$. Let x_k be the true voltage at measurement instant k. Then, the voltage that is measured at time instant k is equal to $\hat{x_k} = x_k + n_k + C$, where n_k is the noise voltage at time instant k and C is the constant offset of the sensor. Integrating $\hat{x_k}$ over k yields:

$$\sum_{k=0}^{k=N} x_k = \sum_{k=0}^{k=N} x_k + \sum_{k=0}^{k=N} n_k + \sum_{k=0}^{k=N} C \tag{2-2}$$

Assuming N is large, the definition of mean value can be used to eliminate the quantization noise term:

$$\frac{1}{N} \sum_{k=0}^{k=N} \hat{x_k} = \mu_n = 0 \tag{2-3}$$

Where μ_n is the expected value of the quantization noise. Recall that this quantity is equal to zero (see section 2-2-3).
The constant error term consists partly of the sensors bias. This term can be calculated by shorting the input sensor to ground and measuring the voltage it measures. This output is averaged over N samples (where $N \approx 1000$). The average value that the sensor measures when it is connected to ground is 3.7mV. By subtracting this value from every measurement, the sensor bias can be accounted for.

In conclusion, Coulomb counting is useful in determining the difference in SOC. By using equation 2-4, the amount of energy that is added to the device connected to the converter can be calculated.

$$\Delta E = \sum_{k=0}^{N} P_k \Delta t \tag{2-4}$$

where $P_k = V_k I_k$ and Δt is the time between measurements k and $k-1$.

Even though it is not possible to determine the exact SOC of a battery through the use of Coulomb counting, it *is* possible to know how much energy was added during a cycle.

2-3 Master

In this design, a top–level master single board computer is implemented. The master is central in the design of the system, and the brain of the network. It receives and sends the information, which it translates to workable data for the microcontroller and server. Its core tasks are:

- Reading the statuses of all the connected slaves on the I^2C bus.
- Outputting these statuses locally.
- Being able to control the connected slaves, either decided locally or due to a command from the web server.
- Sending the statuses to a web server.

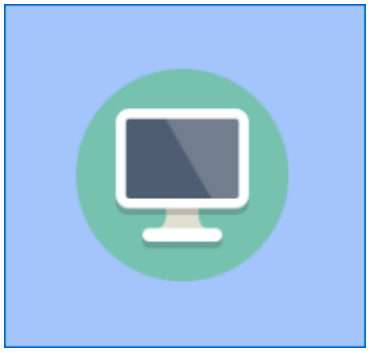


Figure 2-5: The 'master', as part of the system in figure 2-1

2-3-1 Single Board Computer

The chosen single board computer is the ODROID C1+[30]. The ODROID is a versatile, robust and cheap single board computer that fits well in the configuration of this project, as it has I^2C capabilities, is able to run the program designed and can be easily connected to the internet using the Ethernet port on the board. Moreover, the ODROID was used in previous, comparable projects as well [9], thus making it a confident choice. Furthermore its has a relatively low cost yet high performance and quick Internet connection.
The Internet connection is an important part of the system as the computer will create a local dashboard that exchanges information with a TU Delft server. The information on this server consists all the previous power flow characteristics on the system. Saving this information enables the possibility of analysis and prediction, features that might be realized in future designs.

2-3-2 Connecting with the Slaves

Communication protocol

The protocol for communication was described in 2-2-2. Every byte received from the slave will be saved correspondingly for further use.

Scanning for connected slaves

Before the master is able to communicate with the slaves using the defined protocol, the addresses of the slaves need to be known. The master will send a single byte to every address in the predefined address range. If a slave is connected and listening, it will respond with an acknowledgement signal when a byte is sent to the slave's address. The master will save this particular address, as it is associated with a connected converter.

Modularity

One of the main aspects of this design is the modularity. Multiple slaves can be added on the communication bus and the master will dynamically add the devices to its memory to control and observe from that point on.
Every slave device with I^2C enabled will have its 7 bit address that it will respond to when called for by the master. As stated before in 2-2-2, this setup works with dynamic address allocation. The master has a predefined subset of possible address, with one general address assigned to every slave when first connected. Used addresses are stored in the master and a new unused address will be assigned to a newly connected slave.

Reliability

Both the master and the slave should continue working when disconnected to ensure robustness. All the connected parts of the system are able to function autonomously, except the hardware part of the bi-directional flyback DC/DC converter.

The source code of the master was written in C++[31]. This programming language was chosen because of its many object-oriented programming features and prior experience with this language. Especially for one of the purposes of the project, namely being able to easily add multiple converters to one master, programming using objects and classes is very useful. Apart from usefulness, using C++, with classes and its Standard Library function and data structures [32] ensures that there will be no memory leakage and the program will be able to run continuously, as long as there is an active power supply to the master. The code is designed in a way that there is no dynamic memory allocation [33] (and thus not having to deallocate memory manually), reducing the risks of failures even further, whilst keeping all the desired features.

2-3-3 Power management & Control

Reading status

The main feature of the master is reading the status of every connected slave and parsing this information for further use. In this project, the output is given in simple text format, however, user friendly user interfaces can be used to output the data as well, as shown in literature [8]. But, most users will use the web pages on the server, which are easily accessible with any device with a web browser, such as a mobile phone.

Controllable features

Beside data coming from the slave going to master, a reversed data stream is also possible. Possible functions to call from the master include, but are not limited to: powering off the slave, limiting the output and set the direction of the power flow. This is further clarified in 2-3-4.

Maximum power output & other applications

Combing both the abilities of reading statuses and the controllable features, more advanced commands can be implemented. One of these implementation would be a maximum power output across multiple connected devices. This setup of devices creates the ability to manage power more precisely and more conveniently.

As real life example, a setup with multiple electric bikes as loads and a solar panel as power source can be realized. When the power from the power source is not enough to fully charge both loads, trade-offs can be made by the master to keep the setup running, even if it would be at a lower power rate.

2-3-4 Online Connection

Protocol

To keep flexibility, the master uses JavaScript Object Notation (JSON) [34] to serialize the data that needs to be sent and then send it as a simple text package to the server. Using JSON does not limit the size of the data packages that are sent. Consequentially, this suits our demands, as the amount of slaves connected to the master and amount of data needed to be sent is variable-sized.

Logging status

The master sends a JSON package containing two arrays of information to the server. The first array actually is a multidimensional array, because it contains an array with all the integer variables (such as voltages, currents and power state) of a single connected slave. The second array in the JSON package is a simple integer array containing the corresponding I^2C slave addresses of the connected converters. For the sake of interactivity, it is important to update the data in real time, thus once every second should be enough.

Receiving control signals

It is possible for the master to receive a JSON data package from the server as well. The message contains three variables.

- Slave address
- Instruction type
- Data to be transferred, if instruction type requires a value (e.g., a maximum voltage output)

On receiving this data, the master decides which instruction needs to be sent to which slave.

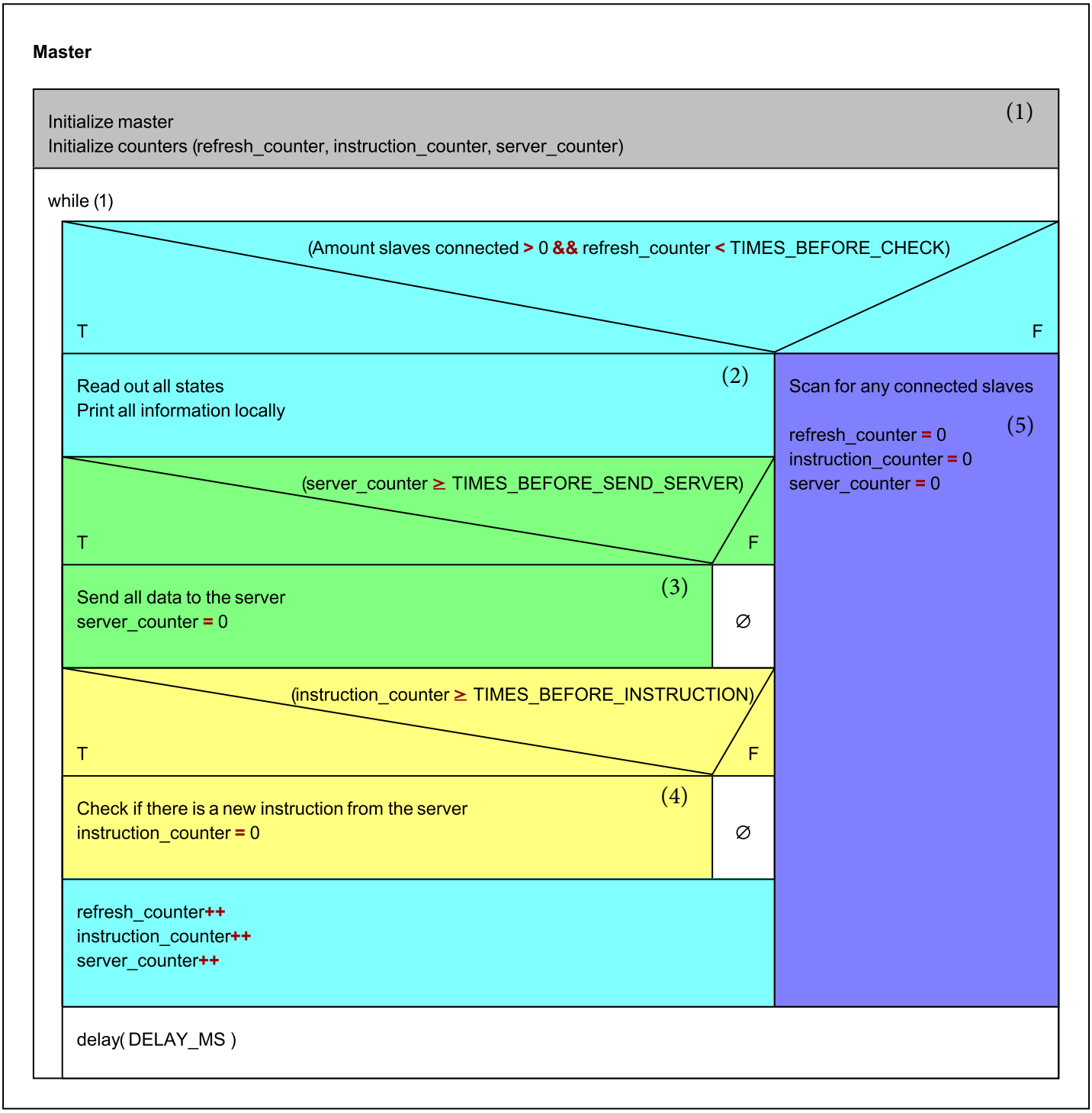
2-3-5 Code Implementation

Having defined the features of the master computer in the previous section, this subsection will describe how these features interact. A Nassi-Schneiderman diagram [35] of the workings of the master is shown in figure 2-6. The diagram elucidates the functioning in pseudocode. The numbers in between the parentheses (), will point to the code section in the figure.

The master starts with storing the server URLs (1). Then the master starts scanning for connected slaves. If no slave is connected, the master will keep scanning for slaves. After a list of connected slaves is known and if any slave is connected, the master device will read all the statuses from the connected slaves (2). The statuses get updated locally every iteration. The next operations depend on the predefined checking thresholds. Not every operation is done every iteration. The two possible operation are sending the data to the server (3) and checking for a new instruction from the server (4). When the counters for one of the two operations exceed the thresholds, the corresponding function is executed. When sending the data, all the received statuses are stored into one data package (as described in section 2-3-4) and sent to the server. On checking for an instruction, if there was a new instruction, the instruction will be parsed and using the protocol described in 2-2-2, the instruction is sent to the right slave. The master will then loop the steps mentioned above.

Scanning the I^2C bus does not have to happen every iteration, due to the fact that the converter will work autonomously without a connection to the master as well. To save time and consequently have a faster rate of reading statuses, scanning the happens once after a predefined number of iterations (5).

Figure 2-6: Nassi–Shneiderman diagram of the functioning of the master



Lower case words connected with underscores are variables. Upper case words with underscores are predefined values.

2-4 Server

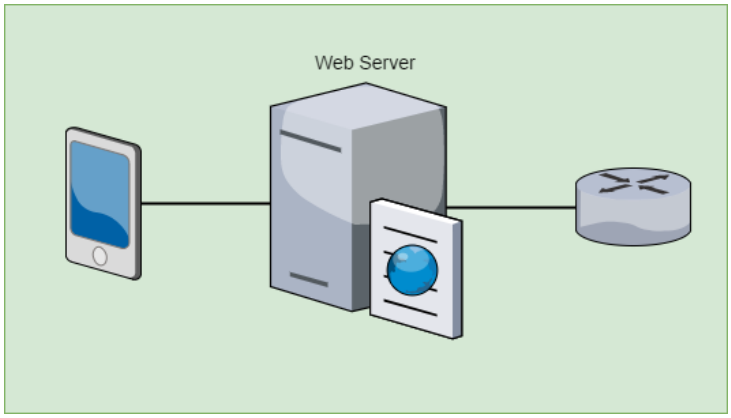


Figure 2-7: The 'server', as part of the system in figure 2-1

The server is the third part of this project. The concept 'server' will be the component that is accessible with an Internet connection and can be viewed in a web browser. It will be a web server with a database to store all the data sent from the master. All the real-time statuses from the converters are available on a web page, which parses the last sent data and displays it in a convenient way. The user is also able to send different instructions to the devices from this web page.

2-4-1 Data Storage & Parsing

The master sends its statuses as a single package (see section 2-3-4) to the server URL as post data [36]. This data package is then stored as text together with an index number and a time stamp as a row in a table of a MySQL [37] database. The data from this MySQL row is then retrieved by the web page and used to display the current status. MySQL is considered the second most popular database engine [38] and due to its wide support across many web hosts, it is used for this project.

The advantage of saving all the received data into a database, is that there is a possibility to retrieve all the historical data and analyze it, for the sake of future prediction of power flows. This makes a contribution to the Power Management part of the design requirements 1-3.

2-4-2 User Interface

Mobile users will be the main users of the web page. Therefore, a convenient user interface is necessary to enable the full capabilities of the project. However, due to the fact that an interface is not particularly the scope of this thesis, only a basic interface was designed, with a limited set of features. These features will mainly be used to show the functionality of a future prototype.

Main View

The main page of access should provide an overview of connected slaves and a directory to the current setup. Furthermore it contains a clock which is updated each time the Master data is uploaded to the server. If the clock does not correspond to the actual time/it stopped working, this indicates the ODROID has crashed or is not able to connect to the assigned web server.

Features

When a user accesses the interface of the designed system, these must be the required features:

- Select the converter the user is using
- Read out the current status of that particular converter
- Turn this converter on/off
- Switch the direction of power flow
- Set the maximum power output

All the above-mentioned feature will be realized using large, user-friendly buttons on the web page. The maximum power flow can be adjusted using sliders on the page.

2-4-3 Code Implementation

The server is designed using PHP [39]. PHP is a server side programming language and is capable of interacting with the MySQL databases of the host as well. The server and master are connected through two given URLs. The master sends its data to the receiving URL of the server. This data is then put in the MySQL table.

To access the data, a user visits the homepage of the project. This page retrieves the latest entry of the MySQL table and then parses it and displays it as HTML [40] & JavaScript [41], which is displayed in the web browser.

2-5 Use

The control network can be utilized in a variety of sectors. This is attributable due to the fact that the system is scalable and designed in a modular fashion. This allows other projects in other fields, for example automotive, to apply and expand the design presented in this design report.

The network is to be used in a solar powered E–bike charging station. The station contains approximately four converters. When the user has plugged in their E–bike, it will begin charging the battery. They can then visit the website (www.bap2017b.tk) to monitor how much energy was deposited into their device.

Evaluation

3-1 Overview

This chapter describes the final results of the designed system. This includes testing of the prototype and validating if the specifications are met. As of writing the thesis, the converter was not yet finished to test with the control network, therefore, the system currently is similar to figure 3-2. To measure if design requirements, several test setups were implemented to validate the system. In table 3-1 the design requirements of section 1-3 with summary are shown. Validation of the specifications are described in Boolean, thus either true or false if it is working or not.

Requirement	**Description**
Robustness	System must be able to run for seven consecutive days without crashing
Modularity	Connecting of new slaves, while the network maintains the same level of performance
Expandability	Code should be structured and easy to grasp concepts for future teams to expand
Interactivity	Data communication from slave input to server output and visualization for end-user
Power Management	Adjustable power output for each connected slave by either the control system on the microcontroller or by the user on the website

Table 3-1: Design Requirements Summary

3-2 Prototype

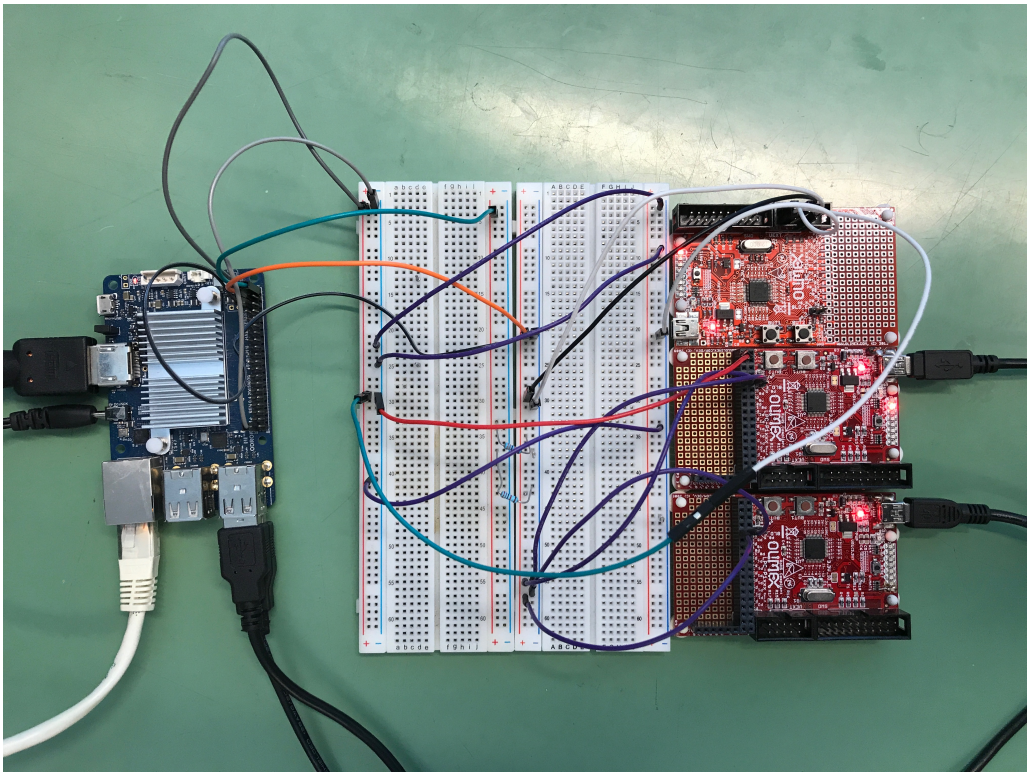


Figure 3-1: Prototype with three connected slaves

The finalized prototype is shown descriptively in figure 3-2. The purpose of the device is to register input current and voltage signals of a device connected to the slave, in this case the converter designed by the other half of BAP Group B. Then, this slave is able to autonomously process this data to determine if connected device is either a active or passive source. If it is a passive source, for example a bicycle battery, the slave will use a FSM to supply it with power until it is full. Using the LEDs on the slave, several light patterns were created to visualize the state the slave finds itself in: searching, supplying power and withdrawing power.

The Master also receives the input data using the I^2C protocol. Furthermore it can change the power output from nothing to maximum output. Furthermore this output can be controlled by a user by accessing the website and manually altering the power flow. The website shows all slaves registered with corresponding output and input signals at the master. Finally the website shows if the computer is still running.

All the designs in chapter 2 were implemented by programming in different programming languages, on different controllers and machines. The code for the slave microcontroller was written in C [21]. The source code for the master was written in C++[31] and the server was implemented on a web server using MySQL [37], HTML [40], PHP [39] and JavaScript [41] .
All the code is to be found in appendices A, B and C. All the written programs are reflections of the designs and the control flows are elucidated using comments in the code itself. Only specific code design choices are highlighted in the next sections, as it not necessary to go over structure of the code, as explained in chapter 2, again.

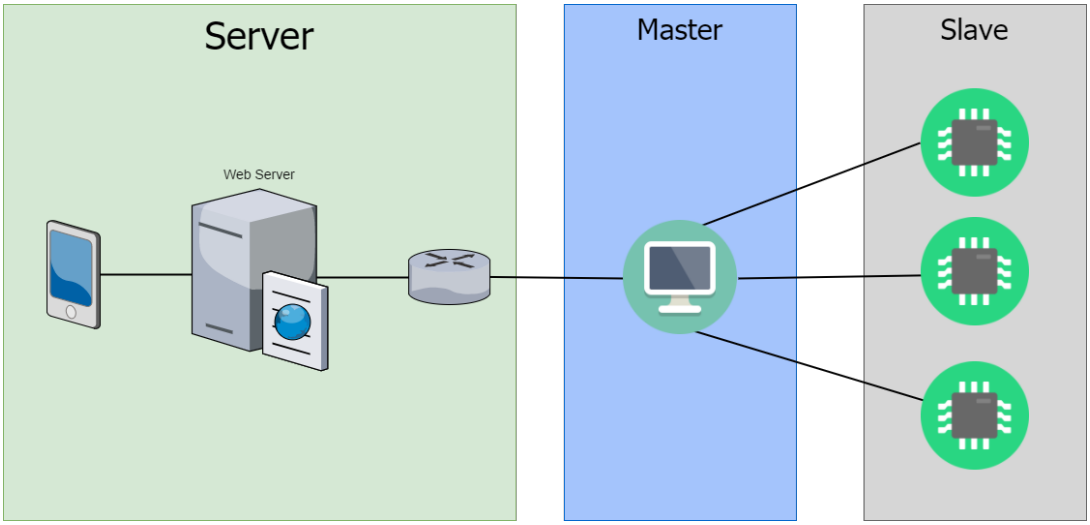


Figure 3-2: Final System schematic as shown in figure 3-1

3-2-1 Slave

The code that is programmed on the slave can be found in appendix A. It is written in C and makes use of several libraries which allow the code to run on the chosen microcontroller (LPC1343). These libraries are provided by the manufacturer, and contain device specific information.

The microcontroller must perform tasks in parallel, such as communication with the master and controlling the PWM duty cycles. With a procedural programming language like C, this can be tricky because every line of code is executed after the other. For this reason, Interrupt Service Routines (ISRs) are employed throughout the code. ISRs are procedures that are executed upon specific events. An example is when the I^2C senses a start condition on the bus; the module will issue an interrupt, which will trigger the I^2C interrupt routine. This function can be custom–made by the programmer. The advantage of these constructs is that they are executed asynchronously, which allows procedural languages to execute in quasi–parallel. Another advantage is that the code can be structured more neatly by placing the ISR–functions into a separate file, and thus keeping the main function compact and readable.
Examples of modules that are coded using ISRs are:

I^2C

The I^2C communication with the master operates in the background by writing the functionalities inside of the ISR. In this subprocedure, the interrupt status code is read out and depending on what this code was, the correct bits in the I^2C–module are (re)set.

Timers

In the prototype, different types of LED patterns are programmed to be displayed, depending on what state the Finite State Machine is in. For example, when nothing is connected, the LEDs are turned on one by one signifying that the microcontroller is 'scanning' for a load. When a load is connected, the number of LEDs that is on increases periodically, signifying that the load is receiving energy.

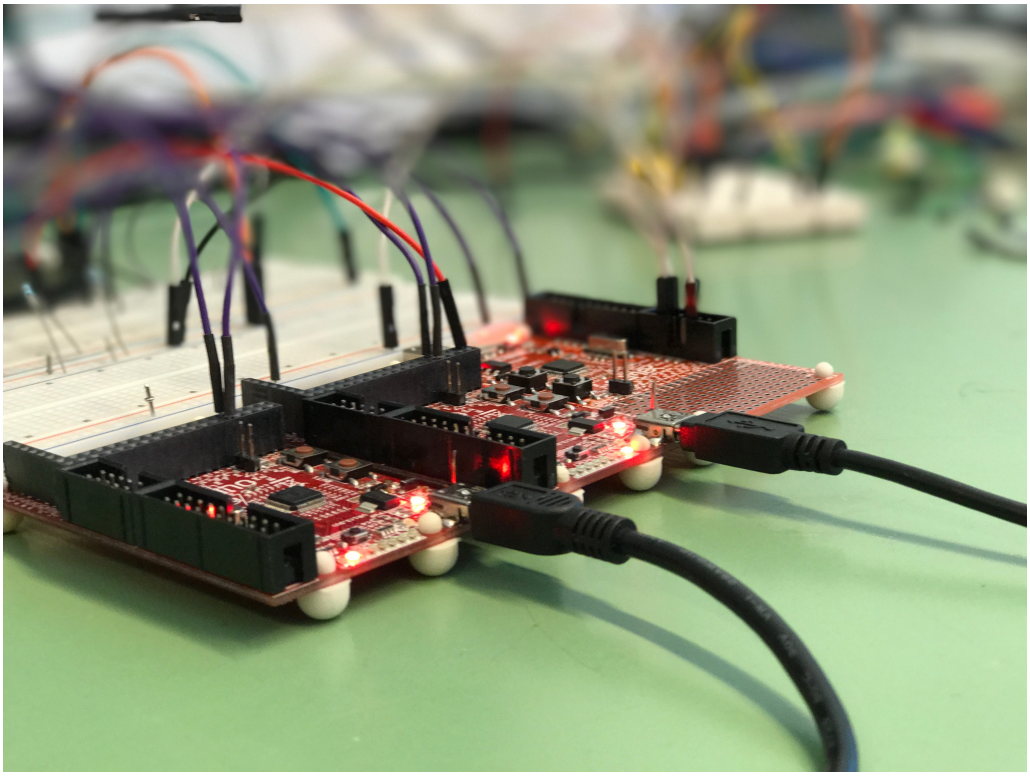


Figure 3-3: Prototype of three connected slaves to the master. On the left: the ODROID single board computer. On the right: three LPC1343 microcontrollers.

These functions are implemented to enable debugging; since the microcontroller does not have an HDMI or VGA port, it can be difficult to keep track of what is happening inside the MCU.

PWM

The two PWM–channels are controlled through a 32–bit counter and several 32–bit match registers (MR). A match register is set equal to the period of the PWM–signal (expressed in clock cycles of the main CPU). Another match register is set equal to the duty cycle multiplied by the period. Then, using the routine described in figure 3-4, the PWM output is set either low or high.

In the main code (appendix A.1), it can be seen that many variables are declared globally. This is done for two reasons. Firstly, the interrupt service routines described in the sections do not take arguments (since they are never called directly by the programmer). Thus, these ISRs can only read and write global and local variables. By making variables globally available, the ease of use of ISRs is greatly increased.
Another reason to declare globals is to make the code appear more tidy. Custom functions do not require excessive amounts of input arguments, which makes code far more readable.

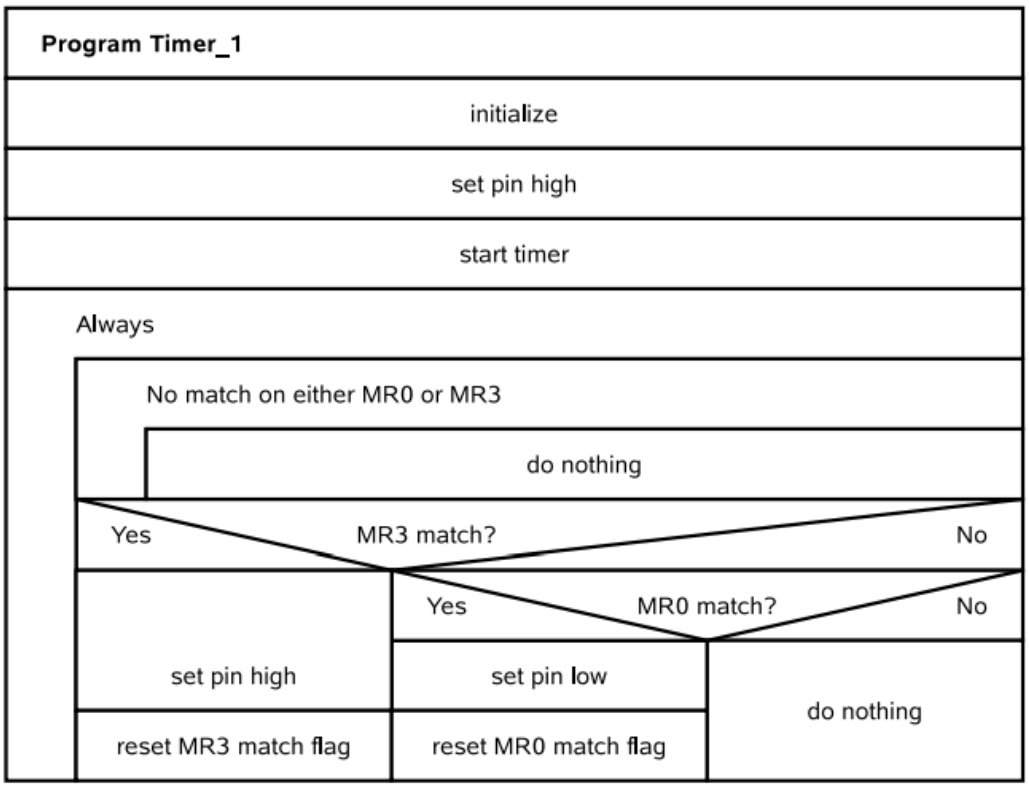


Figure 3-4: Nasi-Schneidermann diagram of how the PWM signals are generated. Adapted from [22]

3-2-2 Master

The source code of the master is to be found in appendix B.

Structure

The master program consists of five different parts. The main code and four classes (mastermind, power_manager, slave_list, slave_handler). These classes are designed to break a bigger problem into smaller sub-problems. Breaking up pieces of functionality into their own classes and encapsulating all the logic makes the easier to maintain, test and reuse different parts of the features. This adds to the expandability 1-3-3 part of the project. The main code is the top-level control loop. Then, the mastermind class keeps all of high-level functions as shown in figure 2-6. To make all these functions possible, the other classes (power_manager, slave_list, slave_handler) are used. The power_manager class implements all the commands and request from section 2-2-2. The class slave_list is a designed to keep track of all the connected converters in a organized way. And the class slave_handler implements all the communication at the lowest level.

Control loop

As seen in figure 2-6, not every operation is executed every iteration of the main loop. The thresholds and timing is defined in lines 6-9 in B.1. The main loop will scan for slaves after 60 iterations, will check for a new instruction coming from the user every iteration and send the status to the server every 3 iterations. These compromises have been made in a way that the user has almost continuous output ($>$ 1 Hz refresh rate) on the web page and even faster

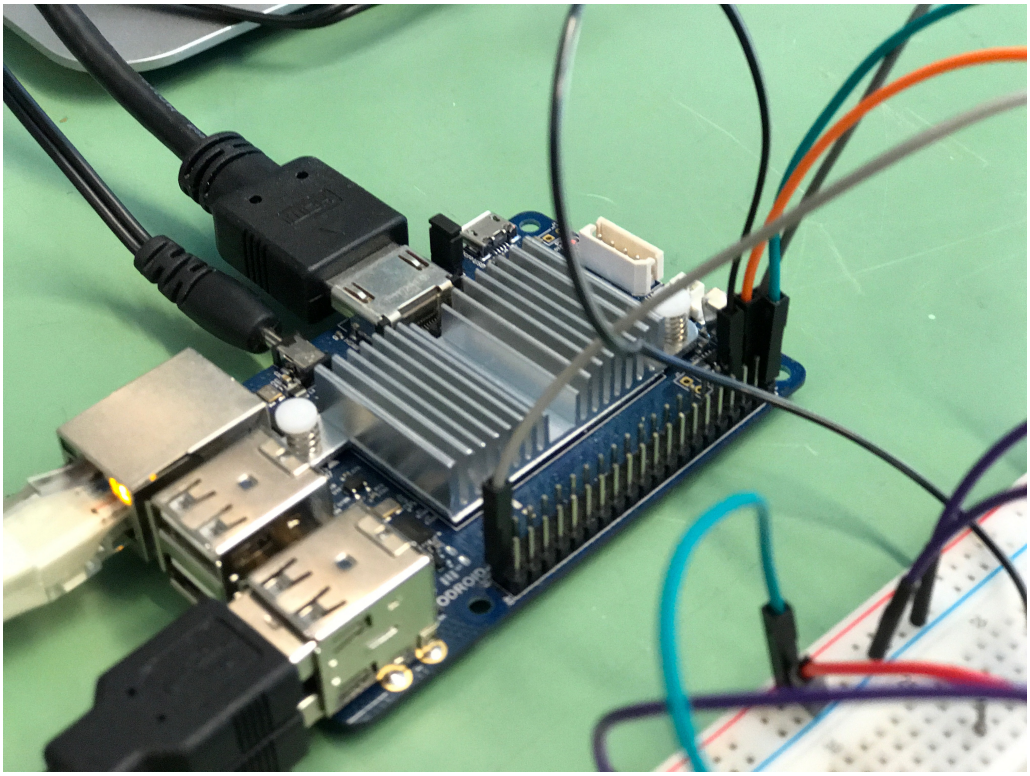


Figure 3-5: The ODROID is the master

response of instructions. Output faster than 1 second is found to be not useful, as the user cannot respond faster than a second and analysis is more focused on steady state behaviour.

Sending & receiving data

To encode the data that needs to be sent in a JSON package (section 2-3-4), [42] is used. This library allows the programmer to parse JSON package with only one header file included.
To send the encoded data to the server, cURLpp is used [43]. The maximum time-out for the request is set to 5 seconds. When the package is not sent or received correctly, the request is stopped and the master continues without using the Internet connection.

The master receives its instruction types as strings. Examples for instruction strings are: 'write-voltage' or 'turnoff'. One of the disadvantages of strings in C++is that they can not be used inside of the switch statement in line 229 of listing B.3. Therefore, the strings have to be converted to numbers, which is done in lines 5-27 (in B.3 again).

I^2C

For the I^2C communication, the library WiringPi [44], a I^2C for single board computers is used. This enables writing and reading bytes from the communication bus with only one line of code.

To work around the fact that this communication protocol is only capable of sending one byte at a time, the designed protocol from section 2-2-2 is implemented. To split the values that need to be sent into bytes, a bit mask and then a shift is used to get correct parts, as shown for example in line 16 in code B.5.

3-2-3 Server

The source code of the server is to be found in appendix C.

Interaction with the user

The main accessible pages of the website contain an overview of the currently connected slave and a page with instructions and status of one specific connected controller. The pages consists of a static 'skeleton' of tags (lines 26-44 in C.1 & lines 106-196 in C.2, which compose the layout of the website.
Using jQuery [45], a JavaScript library designed to simplify client-side scripting, the web page loads the latest status of all the slaves as a JSON package, parses it and dynamically adds new elements to the static frame of the main page. This status is loaded every 500 milliseconds (line 61-91 in listing C.1). For every slave connected, it adds a new button to navigate to the converter page, as shown in figure 3-7.
On the page with instructions and status of a single controller, the latest status for that specific controller is parsed and shown. The following instructions are possible to execute by tapping a button on that page:

- Turn on
- Turn off
- Switch direction
- Set maximum voltage (with a slider for the amount)
- Set maximum current (with a slider for the amount)

On tapping a button, the correct instruction will be uploaded to the database to later be received by the master (line 14-77 in C.2).
The buttons and layout are put together by using Bootstrap [46], Font Awesome [47], jQuery UI [48] and range-touch [49]. Combining all these libraries creates a mobile-user friendly web page with fast response.

Interaction with the master & database

In every interaction with the MySQL database, the configuration file in C.3 is called to make the connection. After the connection is made, request to the specific data where the data is stored can be made.
When the users executes a new instruction, the target slave address, instruction type and data to be sent along with the instruction is sent from the user page to a server page (listing C.7). This page then stores this data in a table called 'instructions' in the database. When a new instruction is added, the list of recipients of the instructions is emptied.
The file from listing C.4 is loaded by the master to check for new instructions. When it has loaded the new instruction, this master IP gets stored into a list of recipients, so no instructions are received double and executed without any use. This feature also allows to add multiple masters to the same database in later design iterations.
To get dynamic output on the user pages, the server side script listed in C.6 displays the latest status received by the server. The master adds this status by posting its status to the server page listed in C.5.

Extra feature

As an extra feature, an additional directory takes the user to a 3D model of the charging station. Here the model can be viewed using rewritten code of [50]. The 3d image has been made in Microsoft 3D builder [51].

3-3 Testing & Results

The design requirements of section 1-3 are evaluated in this section.

3-3-1 Robustness

In order to put the robustness of this system to the test, the ODROID is connected to two microcontrollers. Sensor data from both microcontroller is read out every second and stored in a database. Then, the system is left alone for one hundred hours.

This longest test was carried out during a period of time that the access to the ODROID was limited. The test started on Wednesday may 24^{th}, 16:00 and ended on Monday may 29^{th} 10:00. The ODROID had been running for 114 hours, and was still running. At this time the test was terminated. The system has passed the robustness requirement, and a lower bound on the crash time has been obtained, namely $T_{crash} < 114h$.

3-3-2 Modularity

To test whether the prototype is expandable with minimal human interference, ten microcontroller units are connected to the ODROID. Each MCU is powered via a USB–cable. The SDA and SCL ports are connected to a bus located on a breadboard.

Testing was done using seven external LPC1343 boards that were supplied by the EE3D11 practical. Unfortunately, connecting any of these boards to the I^2C bus caused the bus to malfunction. The reason for this was the fact that previous soldering on these boards caused a large capacitance, jamming the communication (see figure 3-6).

Connecting three boards (which did *not* have bad soldering) to the bus *did* produce the

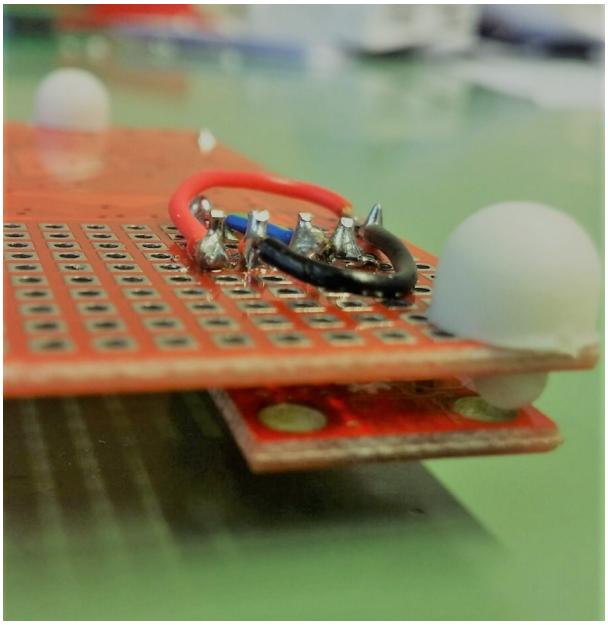


Figure 3-6: One of the 'dirty' microcontroller boards. The soldering causes a high capacitance, which jams the I^2C bus.

wanted results. In summary, this test proved inconclusive due to the lack of enough clean microcontroller boards.

3-3-3 Expandability

During the design process a lot of time was taken to make sure the system is easily expandable and therefore future-proof. Furthermore it should also be easy to grasp the ideas and expand upon the foundation build in this project. Several design choices confirm this fact:

- Data communication between master and slave using I^2C protocol, which is not only robust but also enables easy connection with new devices. The code is written to expand on the number of commands if the future designer deems this necessary.
- Programming the microcontroller in C, since this is a common language widely used in computer architecture especially at the TU Delft Electrical Engineering department.
- Programming the mini computer in C++, which is an enhanced version of C which introduces object oriented programming. This makes communications protocols easy to comprehend for new groups.

3-3-4 Interactivity

Through the link www.bap2017b.tk, anyone with an active Internet connection can view the slaves that are currently attached to the ODROID.

Figure 3-7 shows the main page of the site. There is a clickable button for ever slave device that is connected. Clicking one of these buttons brings the user to a page where they can set the maximum voltage and current, change the direction of power flow (if possible), and turn the converter on or off.

To verify the interactivity requirement, access to the website is tested via a Windows PC, an Android smartphone and an iPhone. Each of these devices can access the website and use all of its features.

3-3-5 Power Management

To control power management, the microcontroller can set the converter's voltage and set its maximum output current. The converter provides the microcontroller with the input and output voltages and currents, which are processed by the Finite State Machine.
The microcontroller sends these quantities to the master via the I^2C protocol, where they are displayed on the website. Via the website, the maximum voltage and current can be set.
To simulate these two functions, two separate tests were implemented:

The test setup described below is to validate whether the control loop (dictated by the FSM) does what it is expected to do.
To verify this for battery–type loads, a simple resistor-capacitor (RC) circuit was made. This consisted of a resistor placed parallel with a capacitor seen in figure 3-8. It can be considered a battery, because applying a voltage to it will charge it. This method is from Electrical Energy Conversion Practical [52]. The converter PCB was not available yet, so it was simulated by passing the PWM–signal through a lowpass filter. By doing so, the DC–component of the signal is extracted and the following relationship is:

$$V_o \propto D \tag{3-1}$$

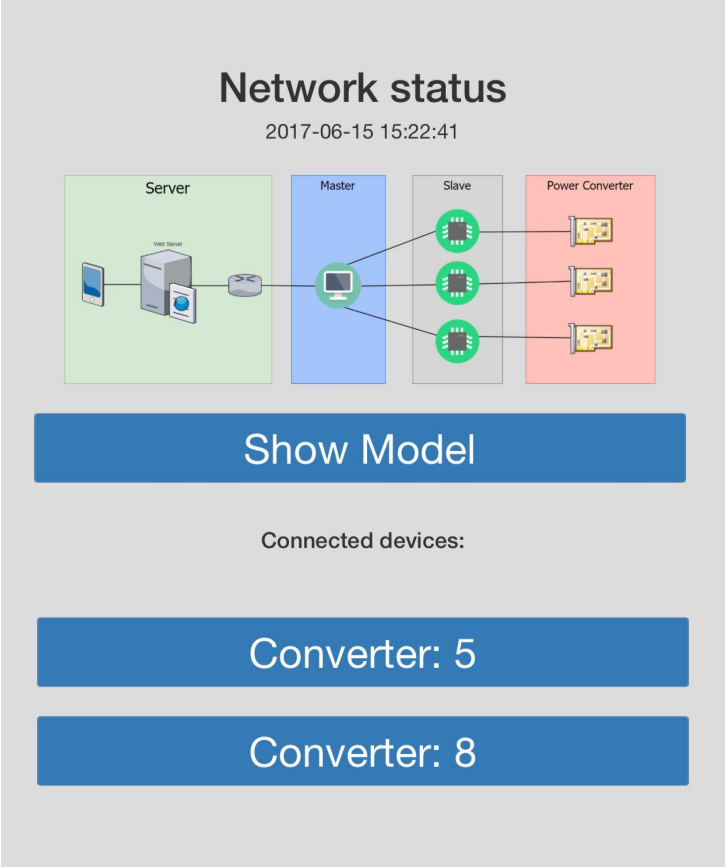


Figure 3-7: The main window of the web page.

where D is the duty cycle.
This setup also shows a DAC separately so the microcontroller can read the changing power flow direction. With the test setup, no current flow can be measured directly. The current measurement is thus performed manually, by connecting the current measurement pin to 3.3V and ground.

The eight LEDs on the LPC1343 board are configured by software to blink in a certain pattern depending on the state the FSM is in. By looking at these blinking patterns, it was possible to determine that the states were looped through as expected, confirming that the FSM is functioning as expected.

To test whether the voltage measurements are sent to the master correctly, the voltage across the RC–network is measured using a multimeter with a precision of ± 0.1mV. The voltage measured by this device was compared to the voltage displayed on the webpage. The mean deviation between the two values was 10 mV, which proves that the correct voltages are being transmitted to the master.

The conclusions that can be drawn from these two test setups are:

1. The system's FSM is moving along its states as expected.
2. The measurements of the control signals are sent to the master with an accuracy of $\pm 10mV$.

To verify the expected behaviour for a resistive load, a resistor was attached to the output of the

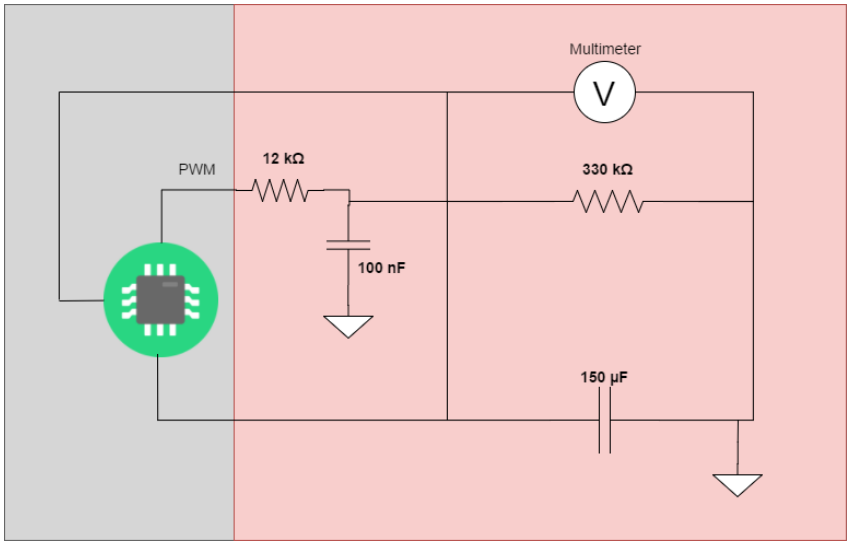


Figure 3-8: RC Circuit to simulate a battery.

simulated converter PCB. When this happens, the voltage quickly ramps up to the maximum value, as is to be expected. Nevertheless, it has to be taken in consideration that not all resistive loads require the same voltage level.

With these two setups, several conclusions can be drawn:

1. The states of the Finite State Machine are being walked through as is to be expected from its design.
2. When a load is being supplied with power by the simulated converter PCB and the direction variable is changed, this load is no longer being supplied with power. This confirms that the microcontroller software correctly implements the bi–directional power flow that it is programmed to do.

At the time of writing the thesis, it was not yet possible to connect the control network with the converter of the other group. If this were possible, the following test setup would have been used to show the bi–directionality of the system shown in figure 3-9. This would show the bi–directional characteristic in combination the adjustable power output at the converters.

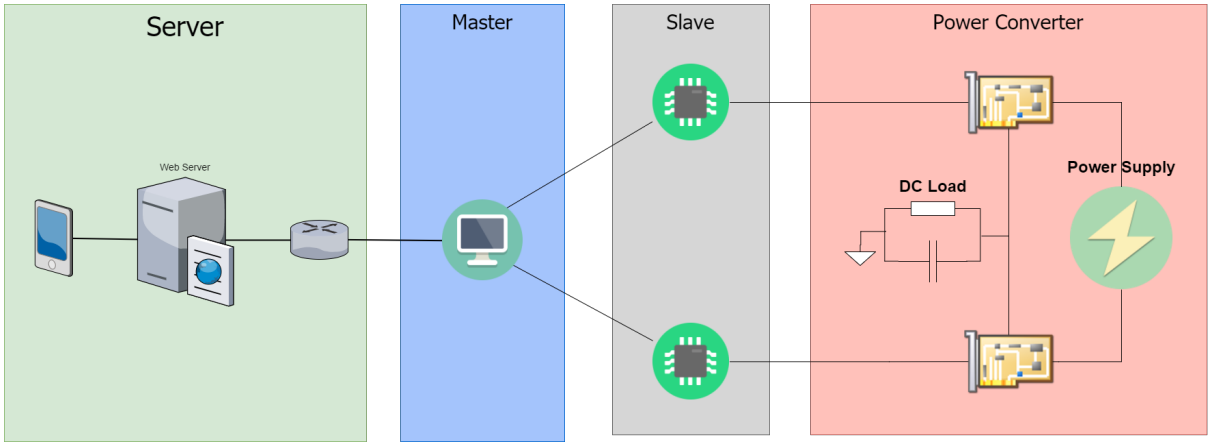


Figure 3-9: Test setup for control network combined with bi–directional converter

3-4 Assessment

Section 1-1 described the challenges of the control network in combination with the DC/DC converter. After the prototype was developed, it can regulate the power control signals from microcontrollers. Furthermore, all the microcontrollers can be connected to one master computer, which controls energy throughput over all power converters. The output data of all slaves is saved, however, the system does not (visually) analyze the power consumption data. The power converters were not yet connected to the network and thus it is not yet verified whether the actual PCB can be controlled via the network.

However, the realized prototype shows the potential of the designed control network of DC/DC converters. The testing results show that the system is robust and expandable. The setup was able to keep running without errors for the required test intervals.

All the necessary features to do the desired power management are present in the system, but is not set up and tested for real-life use yet. It is possible to externally control the different connected controllers and its control signals, but due to the fact that have not been combined with the power converter PCBs, no real power flow management has been done yet.

No SOC determination method is implemented as proposed in 2-2-5. Unfortunately, the first application of the system was meant to involve charging batteries, so this feature was desirable. However, the method of output determination implemented, as given in section 2-2-4, suffices for determining the nominal voltage of a load or battery connected. Together with the possible commands from the user, this design can be used in the desired application, an electrical bike charging system.

The interactivity of the system suffices as well, as the user is able to access the control website from any device with a web browser and Internet connection. The user interface is not meant for a bigger userbase yet, as it is not as convenient as interfaces reviewed in 1-2-4. The current interface enables total control the networks currents and voltages, as where some features should be secluded in future designs and replaced by simpler, safe control options, depending on the type of user (e.g. only turning on/off the charger).

Overall, the created framework for the control network is functioning as desired and shows potential as framework for future projects.

3-5 Next Steps

The control network was designed for expandablity for future use. As explained in section 2-5, the project can be used in a number of different applications so future projects could expand on what is possible with the ground work layed out.

The robustness of the system should be tested under more stressful conditions. In section 3-3-1, the system was tested with three slave devices connected. It is worth exploring if the same uptime can be achieved while more than three devices are connected.

Testing the modularity (section 3-3-2) proved inconclusive for more than three slave devices, due to the unavailability of more 'clean' microcontrollers. It is of considerable importance that the design can be verified for at least ten devices. Hardware on the microcontroller boards allows software to multiplex I^2C channels ([20] Table 216). With this technique it is possible to simulate multiple MCUs on a single microcontroller.
If it proves impossible to connect the minimum amount of slave units on the same bus at a frequency of 100 kHz, it can be worth investigating if reducing the bus frequency can decrease communication latency. As a last resort, other communication protocols can be explored.

The interactivity of the website could be greatly expanded upon by adding more features to appeal to the user visually. The interface could be designed for specific purposes. So, for instance, the State of Charge could be uploaded and visualized on the website, when the energy system involves charging batteries. Additionally, when used in a solar charging station, a function could be added that calculates the time that remains until units are fully charged (based on the sun's intensity).

Currently, power management is achieved by manually setting the voltages and currents for each connected converter. In future iterations, the design can be altered so that the user only needs to set a maximum power value. The software could then automatically calculate the currents and voltages for each slave unit such that the total power drawn is less than the maximum power that is set. This is especially useful in applications where peak loads are much larger than the average load. Having the possibility to 'smooth out' the peak consumption is advantageous because systems can be designed for lower peak loads, reducing their cost drastically.
Another field where power management is useful, is in systems that are driven by unpredictable power sources, like solar and wind energy.

Bibliography

[1] "Netherlands_gb_dl_33pct_v10_161231_hires," https://www.tennet.eu/fileadmin/user_upload/Company/Publications/Gridmaps/ENG/Gridmap_Netherlands_ENG.pdf, (Accessed on 06/15/2017).

[2] H. Lund, A. N. Andersen, P. A. Ã~stergaard, B. V. Mathiesen, and D. Connolly, "From electricity smart grids to smart energy systems â€“ a market operation based approach and understanding," *Energy*, vol. 42, no. 1, pp. 96 – 102, 2012, 8th World Energy System Conference, WESC 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360544212002836

[3] P. Mancarella, "Smart multi-energy grids: Concepts, benefits and challenges," in *2012 IEEE Power and Energy Society General Meeting*, July 2012, pp. 1–2.

[4] S. Eren, D. Küçük, C. Ünlüer, M. Demircioğlu, Y. Arslan, and S. Sönmez, "A web-based dispatcher information system for electricity transmission grid monitoring and analysis," in *2015 9th International Conference on Electrical and Electronics Engineering (ELECO)*, Nov 2015, pp. 986–990.

[5] G. Lacey, G. Putrus, and E. Bentley, "Smart ev charging schedules: supporting the grid and protecting battery life," *IET Electrical Systems in Transportation*, vol. 7, no. 1, pp. 84–91, 2017.

[6] "Toon, de slimme thermostaat," https://www.eneco.nl/toon-thermostaat/, (Accessed on 06/15/2017).

[7] "Meet the nest learning thermostat | nest," https://nest.com/thermostat/meet-nest-thermostat/?from-chooser=true, (Accessed on 06/15/2017).

[8] D. Brouwer and D. Veselka, "Project sunrise gebruikersinterface voor het e-bike oplaadstaton," Master's thesis, Delft University of Technology, 6 2016, groep D, Team HTML.

[9] T. H. en Job van Staveren, "Project sunrise gebruikersinterface voor het e-bike oplaadstaton," Master's thesis, Delft University of Technology, 6 2016, groep D, Team ODROID.

[10] A. el Mehdi and T. de Moor, "Project sunrise gebruikersinterface voor het e-bike oplaadstaton," Master's thesis, Delft University of Technology, 6 2016, groep D, Team Server.

[11] D. L. Heine and M. S. Lam, "A practical flow-sensitive and context-sensitive c and c++ memory leak detector," in *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, ser. PLDI '03. New York, NY, USA: ACM, 2003, pp. 168–181. [Online]. Available: http://doi.acm.org/10.1145/781131.781150

[12] C. Erickson, "Memory leak detection in embedded systems," *Linux J.*, vol. 2002, no. 101, pp. 9–, Sep. 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=566949.566958

[13] P. Bauer, "Solar e-bike station," http://www.ewi.tudelft.nl/en/the-faculty/departments/electrical-sustainable-energy/dc-systems-energy-conversion-storage/research/solar-e-bike-station/.

[14] H. la Poutre, "Smart grid and ict," 2016.

[15] T. Gerrits, G. Koolman, and B. van der Werk, "Design of a bi–directional flyback dc/dc converter," Master's thesis, Delft University of Technology, June 2017, groep B.

[16] Wikipedia, "Texas instruments tms320 — wikipedia, the free encyclopedia," 2017, [Online; accessed 1-June-2017]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Texas_Instruments_TMS320&oldid=765888736

[17] "Controlsuite controlsuite™software suite: Essential software and development tools for c2000™microcontrollers | ti.com," http://www.ti.com/tool/controlsuite, (Accessed on 06/01/2017).

[18] "Arduino - arduinoboardnano," https://www.arduino.cc/en/Main/ArduinoBoardNano, (Accessed on 06/01/2017).

[19] "Why i'm ditching the arduino software platform - alan's ramblings," http://bleaklow.com/2012/02/29/why_im_ditching_the_arduino_software_platform.html, (Accessed on 06/18/2017).

[20] NXP, "Um10375 datasheet," https://blackboard.tudelft.nl/bbcswebdav/pid-2765680-dt-content-rid-9451117_2/courses/40206-161703/UM10375.pdf, June 2009, (Accessed on 06/14/2017).

[21] B. W. Kernighan, *The C Programming Language*, 2nd ed., D. M. Ritchie, Ed. Prentice Hall Professional Technical Reference, 1988.

[22] X. van Rijnsoever, *Lab Manual Computer Architecture and Organisation*, March 2017.

[23] F. Leens, "An introduction to i2c and spi protocols," *IEEE Instrumentation Measurement Magazine*, vol. 12, no. 1, pp. 8–13, February 2009.

[24] E. Ziouva and T. Antonakopoulos, "Csma/ca performance under high traffic conditions: throughput and delay analysis," *Computer Communications*, vol. 25, no. 3, pp. 313 – 321, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366401003693

[25] Unknown, "Charging lithium-ion batteries," http://batteryuniversity.com/learn/article/charging_lithium_ion_batteries, 2010.

[26] F. Zhou and W. Xiong, "Using pwm output as a digital-to-analog converter on dsp," in *2010 International Conference on System Science, Engineering Design and Manufacturing Informatization*, vol. 2, Nov 2010, pp. 278–281.

[27] M. Murnane and A. Ghazel, "A closer look at state of charge and state health estimation techniques," http://www.analog.com/media/en/technical-documentation/technical-articles/A-Closer-Look-at-State-Of-Charge-and-State-Health-Estimation-Techniques-....pdf, 2013, (Accessed on 06/01/2017).

[28] O. Gérard, J.-N. Patillon, and F. d'Alché Buc, *Neural network adaptive modeling of battery discharge behavior*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 1095–1100. [Online]. Available: http://dx.doi.org/10.1007/BFb0020299

[29] W. Z. Caiping Zhang and S. M. Sharkh, "Estimation of state of charge of lithium-ion batteries used in hev using robust extended kalman filtering," http://www.mdpi.com/1996-1073/5/4/1098/htm, 2012, (Accessed on 06/06/2017).

[30] "Odroid | hardkernel," http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143703355573&tab_idx=2, (Accessed on 06/06/2017).

[31] B. Stroustrup, *The C++ Programming Language*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[32] N. M. Josuttis, *The C++ standard library: a tutorial and reference*. Addison-Wesley, 2012.

[33] "new operator (c++)," https://msdn.microsoft.com/en-us/library/kewsb8ba(VS.71).aspx, (Accessed on 06/18/2017).

[34] D. Crockford, "The application/json media type for javascript object notation (json)," 2006.

[35] I. Nassi and B. Shneiderman, "Flowchart techniques for structured programming," *ACM Sigplan Notices*, vol. 8, no. 8, pp. 12–26, 1973.

[36] "Post (http) - wikipedia," https://en.wikipedia.org/wiki/POST_(HTTP), (Accessed on 06/08/2017).

[37] M. Widenius and D. Axmark, *Mysql Reference Manual*, 1st ed., P. DuBois, Ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2002.

[38] "Db-engines ranking - popularity ranking of relational dbms," https://db-engines.com/en/ranking/relational+dbms, (Accessed on 06/18/2017).

[39] "Php: Hypertext preprocessor," https://secure.php.net/, (Accessed on 06/19/2017).

[40] "Iso/iec 15445:2000(e) iso-html," https://www.scss.tcd.ie/misc/15445/15445.html, (Accessed on 06/19/2017).

[41] Wikipedia, "Javascript — wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=JavaScript, 2017, [Online; accessed 19-June-2017].

[42] N. Lohmann, "Github - nlohmann/json: Json for modern c++," https://github.com/nlohmann/json, (Accessed on 06/19/2017).

[43] J.-P. Barrette-LaPierre, "curlpp by jpbarrette," http://www.curlpp.org/, (Accessed on 06/19/2017).

[44] G. Henderson, "Wiringpi," http://wiringpi.com/, (Accessed on 06/19/2017).

[45] "jquery," https://jquery.com/, (Accessed on 06/19/2017).

[46] "Bootstrap · the world's most popular mobile-first and responsive front-end framework." http://getbootstrap.com/, (Accessed on 06/19/2017).

[47] "Font awesome, the iconic font and css toolkit," http://fontawesome.io/, (Accessed on 06/19/2017).

[48] "jquery ui," https://jqueryui.com/, (Accessed on 06/19/2017).

[49] dwyl, "Github - dwyl/range-touch: Use html5 range input on touch devices (iphone, ipad & android) without bloatware!" https://github.com/dwyl/range-touch, (Accessed on 06/19/2017).

[50] M. Wieser, "Webgl 3d model view using three.js," https://manu.ninja/webgl-3d-model-viewer-using-three-js, (Accessed on 06/16/2017).

[51] "3d builder," https://www.microsoft.com/nl-nl/store/p/3d-builder/9wzdncrfj3t6, (Accessed on 06/19/2017).

[52] P. I. P. Bauer, "Ee2e11 electrical energy conversion student manual," 2015-2016.

Appendices

Appendix A: Slave Source Code

Listing A.1: main.c

```
1  /* Includes */
2  #include <stdbool.h>
3  #include <stdint.h>
4  #include "LPC13xx.h"
5  #include "definitions.h"
6  #include "prototypes.h"
7
8  /* Global Variables */
9  int state = RESET_STATE;
10 int totalLength = 0;
11 int lengthLeft = 0;
12 int I2C_addr = I2C_SLAVE_ADDRESS;
13 int cstate = cRESET, cnew_state;
14 unsigned int voltage1 = 0;
15 unsigned int current1 = 0;
16 unsigned int voltage2 = 0;
17 unsigned int current2 = 0;
18
19 unsigned int next_i = 0;
20 unsigned int current1_buffer[TAKE_AVERAGE] = {0};
21 unsigned int current2_buffer[TAKE_AVERAGE] = {0};
22 unsigned int voltage1_buffer[TAKE_AVERAGE] = {0};
23 unsigned int voltage2_buffer[TAKE_AVERAGE] = {0};
24
25 unsigned int maxVoltage = 1024;
26 unsigned int maxCurrent = 1024;
27 unsigned int powered_on = 1;
28 unsigned int PWM1 = PWM1_START;
29 unsigned int PWM2 = PWM2_START;
30
31 unsigned int output_volt = 0;
32 unsigned int output_current = 0;
33
34 bool directionValue = true; // Left to Right = true, Right to left = false
35 uint8_t tmp_rcv;
36 uint8_t toSend;
37 int bufferArray[numberOfElementsInMyArray]; // Set maximum of 10 bytes, easy.
38 int control;
39 unsigned int Timeout_ms = 0;
40 unsigned int Timeout_ms2 = 0;
41 unsigned int ADC_Data[4] = {0, 0, 0, 0};
42 /* END OF Global Variables */
43
44 int main (void)
45 {
```

```
46      /*  The Charging routine is executed in the main() function
47          An in-depth explanation of this Finite State Machine is
48          given in the design report.
49      */
50      initI2C();
51      initTimer();
52      initADC();
53      init_leds();
54      init_PWM();
55      pin_high_1();
56      pin_high_2();
57      timer_start_1();
58      timer_start_2();
59
60      int I_track = INCREMENT_THRESHOLD;
61      int V_track = INCREMENT_THRESHOLD;
62      unsigned int I_previous = 0, V_previous = 0;
63      unsigned int currentReadOut = 0;
64      unsigned int voltageReadOut = 0;
65
66      while(1)
67      {
68          if((Timeout_ms%100)==0)
69          {
70              if(directionValue)
71              {
72                  currentReadOut = current1;
73                  voltageReadOut = voltage1;
74              }else{
75                  currentReadOut = current2;
76                  voltageReadOut = voltage2;
77              }
78              switch(cstate)
79              {
80              case cRESET:
81                  I_track = INCREMENT_THRESHOLD;
82                  V_track = INCREMENT_THRESHOLD;
83                  lightshow(SCAN);
84                  output_volt = 0;
85                  output_current = 0;
86                  if (voltageReadOut > 500)
87                  {
88                      cnew_state = cRAMP_VOLTAGE;
89                  }else{
90                      cnew_state = cRESET;
91                  }
92                  break;
93              case cRAMP_VOLTAGE:
94                  lightshow(DISCHARGING);
95                  if(output_volt < maxVoltage)
96                  {
97                      output_volt += dV;
98                  }
99                  if(output_current < maxCurrent)
100                 {
101                     output_current += dV;
102                 }
103
104                 if (currentReadOut ≤ I_previous)
105                 {
106                     I_track--;
107                 }
108                 if (I_track > 0)
```

```
109                 {
110                     cnew_state = cRAMP_VOLTAGE;
111                 }else{
112                     cnew_state = cNOMINAL;
113                 }
114                 break;
115             case cNOMINAL:
116                 lightshow(DISCHARGING);
117                 output_volt += 0;
118                 output_current += maxCurrent;
119                 if (currentReadOut < 500)
120                 {
121                     cnew_state = cPINCH_OFF;
122                 }else{
123                     cnew_state = cNOMINAL;
124                 }
125                 break;
126             case cPINCH_OFF:
127                 lightshow(DISCHARGING);
128                 output_volt += 0;
129                 output_current += 0;
130                 if (currentReadOut < 50)
131                 {
132                     cnew_state = cDONE;
133                 }else{
134                     cnew_state = cPINCH_OFF;
135                 }
136                 if (voltageReadOut ≤ V_previous)
137                 {
138                     V_track--;
139                 }else{
140                     V_track ++;
141                 }
142                 if (V_track > 0)
143                 {
144                     cnew_state = cPINCH_OFF;
145                 }else{
146                     cnew_state = cDONE;
147                 }
148                 break;
149             case cDONE:
150                 lightshow(FLICKER);
151                 output_volt += 0;
152                 output_current = 0;
153                 if (voltageReadOut < 50)
154                 {
155                     cnew_state = cRESET;
156                 }else{
157                     cnew_state = cDONE;
158                 }
159                 break;
160             default:
161                 set_leds(42);
162                 break;
163             }
164             if ((currentReadOut<50) && (voltageReadOut<50))
165             {
166                 cnew_state = cRESET;
167             }
168
169             if(!powered_on)
170             {
171                 cnew_state = cRESET;
```

```
172                 }
173                 I_previous = currentReadOut;
174                 V_previous = voltageReadOut;
175                 PWM1 = MIN(output_volt,maxVoltage);
176                 PWM2 = MIN(output_current,maxCurrent);
177                 update_PWM_1();
178                 update_PWM_2();
179                 cstate = cnew_state;
180                 renew_ADC();
181                 Timeout_ms = 1;
182             }
183         }
184 }
```

Listing A.2: implementations.c

```
1  #ifndef IMPLEMENTATIONS_C
2  #define IMPLEMENTATIONS_C
3
4  #include "prototypes.h"
5  #include "LPC13xx.h"
6  #include "definitions.h"
7  #include <stdbool.h>
8  extern unsigned int PWM1, PWM2, powered_on;
9
10 /** Custom I2C Protocol Helper Functions **/
11 void receiveEvent(int x)
12 {
13     /*  This function executes actions when databytes are received
14         from the master. It does this in accordance with the custom
15         I2C protocol that has been developed for this project.
16     */
17     extern int totalLength, lengthLeft, state;
18     extern int bufferArray[numberOfElementsInMyArray];
19     int tempI = totalLength - lengthLeft;
20     switch (state)
21     {
22     case RESET_STATE:
23         if (x > 0 && x <= MAX_BEGIN_STATE)
24         {
25             /*  If the received data x is within a range that is legal for ...
                states,
26                 the state is set to x.
27             */
28             state = x;
29         }
30         break;
31     case VOLT_READ_STATE:
32     case CURRENT_READ_STATE:
33     case TURN_ON_OFF:
34     case MASTER_SET_DIRECTION:
35     case SET_PWM_1_STATE:
36     case SET_PWM_2_STATE:
37     case NEW_ADDRESS_STATE:
38     /*  The number of I2C packets that is going to be sent/received by the ...
           master
39         is given by lengthLeft
40     */
41         if (totalLength == 0)
42         {
43             totalLength = x;
44             lengthLeft = x;
```

```
45          }
46          else
47          {
48              bufferArray[tempI] = x;
49              if (lengthLeft == 1)
50              {
51                  done_with(state);
52              }
53              else
54              {
55                  lengthLeft--;
56              }
57          }
58          break;
59      /* Below are all slave --> master writes
60      */
61      case VOLT1_WRITE_STATE:
62      case VOLT2_WRITE_STATE:
63      case CURRENT1_WRITE_STATE:
64      case CURRENT2_WRITE_STATE:
65      case READ_SLAVE_DIRECTION:
66      default:
67          state = RESET_STATE;
68          break;
69      }
70
71  }
72  void requestEvent()
73  {
74      /*  This function executes actions when databytes are sent
75          to the master. It does this in accordance with the custom
76          I2C protocol that has been developed for this project.
77      */
78      extern int totalLength, lengthLeft, state,cstate;
79      extern unsigned int voltage1, voltage2, current1, current2, powered_on, ...
            PWM1, PWM2;
80      extern int bufferArray[numberOfElementsInMyArray];
81      extern bool directionValue;
82      extern uint8_t toSend;
83
84      switch (state)
85      {
86      case VOLT1_WRITE_STATE:
87      case VOLT2_WRITE_STATE:
88      case CURRENT1_WRITE_STATE:
89      case CURRENT2_WRITE_STATE:
90      case READ_PWM_1_STATE:
91      case READ_PWM_2_STATE:
92      case READ_CHARGE_STATE:
93          if (totalLength == 0)
94          {
95              /*  Integers are being sent over the I2C bus.
96                  These are typically 4 bytes long, but for portability reasons...
                        ,
97                  their length is defined through a constant called ...
                        INT_SIZE_BYTE
98              */
99              totalLength = INT_SIZE_BYTE;
100             lengthLeft = INT_SIZE_BYTE;
101             toSend = INT_SIZE_BYTE;
102         }
103         else
104         {
```

```
105             /* Decide which measurement (volt1, current1, etc..) to send
106             */
107             int value;
108             if (state == VOLT1_WRITE_STATE)
109             {
110                 value = voltage1;
111             }
112             else if (state == VOLT2_WRITE_STATE)
113             {
114                 value = voltage2;
115             }
116             else if (state == CURRENT1_WRITE_STATE)
117             {
118                 value = current1;
119             }
120             else if (state == CURRENT2_WRITE_STATE)
121             {
122                 value = current2;
123             }
124             else if (state == READ_PWM_1_STATE)
125             {
126                 value = PWM1;
127             }
128             else if (state == READ_PWM_2_STATE)
129             {
130                 value = PWM2;
131             }
132             else if (state == READ_CHARGE_STATE)
133             {
134                 value = cstate;
135             }
136             /* Sends the variable 'value' by sliding the 8bit window 0x00FF ...
                    over it
137             */
138             toSend = (uint8_t)((value & (0x00FF << (totalLength - lengthLeft)...
                     * 8)) >> (totalLength - lengthLeft) * 8);
139             lengthLeft--;
140
141             if (lengthLeft == 0)
142             {
143                 done_with(state);
144             }
145         }
146         break;
147     case READ_SLAVE_DIRECTION:
148         /* The direction is a single bit, thus only one I2C packet needs to ...
                be sent,
149         which is why totalLength is set to 1
150         */
151         if (totalLength == 0)
152         {
153             totalLength = 1;
154             lengthLeft = 1;
155             toSend = 1;
156         }
157         else
158         {
159             toSend = directionValue;
160             done_with(state);
161         }
162         break;
163
164     case READ_POWER_STATE:
```

```
165         if (totalLength == 0)
166         {
167             totalLength = 1;
168             lengthLeft = 1;
169             toSend = 1;
170         }
171         else
172         {
173             toSend = powered_on;
174             done_with(state);
175         }
176         break;
177 
178     default:
179         /*  If the received data was other than expected, return -1 so the ...
                master
180             knows something has gone wrong.
181         */
182         state = RESET_STATE;
183         toSend = -1;
184         break;
185     }
186 }
187 void reset_buffer()
188 {
189     /*  This function clears the I2C-buffer by writing
190         zeros to its elements. It also puts the slave's
191         I2C state into the reset mode.
192     */
193     extern int state, totalLength, lengthLeft;
194     extern int bufferArray[numberOfElementsInMyArray];
195     state = RESET_STATE;
196     totalLength = 0;
197     lengthLeft = 0;
198     for(int i = 0; i<numberOfElementsInMyArray; i++)
199     {
200         bufferArray[i]=0;
201     }
202 }
203 void done_with(int state)
204 {
205     /* This routine is executed at the end of a successful I2C conversation.
206     The received packets are interpreted and, depending on which state the ...
            slave is in,
207     commands are executed from the master.
208     */
209     extern unsigned int maxVoltage, maxCurrent, powered_on, PWM1, PWM2;
210     extern int bufferArray[numberOfElementsInMyArray];
211     extern void initI2C();
212     extern int cnew_state, cstate;
213     extern int I2C_addr;
214     extern int totalLength;
215     extern bool directionValue;
216     extern void update_PWM_1();
217     extern void update_PWM_2();
218     extern void delay_ms();
219 
220     unsigned int temp_value = 0;
221     for(int i = 0; i<totalLength; i++)
222     {
223     /* This loop 'interprets' the array of 8bit integers into a 32bit (or ...
            more) integer
224     */
```

```
225             temp_value += bufferArray[i] << (8*i);
226         }
227         /* Executes the command that was issued by the master
228         */
229         if (state == VOLT_READ_STATE)
230         {
231             maxVoltage = temp_value;
232         }
233         else if (state == CURRENT_READ_STATE)
234         {
235             maxCurrent = temp_value;
236         }
237         else if(state == TURN_ON_OFF)
238         {
239             powered_on = temp_value;
240             if(!powered_on)
241             {
242                 PWM1=0;
243                 PWM2=0;
244                 cnew_state = cRESET;
245                 cstate = cRESET;
246             }
247             update_PWM_1();
248             update_PWM_2();
249         }
250         else if(state == MASTER_SET_DIRECTION)
251         {
252             directionValue = temp_value;
253             cnew_state = cRESET;
254             cstate = cRESET;
255         }
256         else if(state == SET_PWM_1_STATE)
257         {
258             PWM1 = temp_value;
259             update_PWM_1();
260         }
261         else if(state == SET_PWM_2_STATE)
262         {
263             PWM2 = temp_value;
264             update_PWM_2();
265         }
266         else if(state == NEW_ADDRESS_STATE)
267         {
268             I2C_addr = temp_value; //
269             initI2C();
270         }
271         reset_buffer();
272     }
273     /** End of Custom I2C Protocol Functions **/
274
275     /** LED Functions **/
276     void init_leds (void)
277     {
278         /* This function initializes the LED pins
279         */
280         /*  The direction (IN or OUT) of the leds has to be set before use.
281             Writing a 1 to the correct bit in the DIR register sets the pin as ...
                    output
282         */
283         // Leds 0-3 are on PIO3_0-PIO3_3
284         LPC_GPIO3->DIR |= (1<<3) | (1<<2) | (1<<1) | (1<<0);
285         // Leds 4-7 are on PIO2_4-PIO2_7
286         LPC_GPIO2->DIR |= (1<<7) | (1<<6) | (1<<5) | (1<<4);
```

```
287 }
288 void led_on (void)
289 {
290     LPC_GPIO3->DATA = LPC_GPIO3->DATA & ~(0x02);
291 }
292
293 void set_leds (uint8_t leds)
294 {
295     /*  Controls the array of 8 LEDs as follows:
296         the variable 'leds' is to be interpreted as an 8bit number.
297         Positions where 'leds' is 1 correspond to the LEDs that are
298         turned on on the microcontroller development board.
299     */
300     uint32_t leds_low_data  = ~leds & 0x0F;
301     uint32_t leds_high_data = ~leds & 0xF0;
302     LPC_GPIO3->DATA = (LPC_GPIO3->DATA & ~0x0F) | leds_low_data;
303     LPC_GPIO2->DATA = (LPC_GPIO2->DATA & ~0xF0) | leds_high_data;
304 }
305 void lightshow(int mode)
306 {
307     extern unsigned int Timeout_ms2;
308     if(!powered_on)
309     {
310         set_leds(0);
311         return;
312     }
313     /* T_ms is the period at which the LEDs will perform their routine
314     */
315     int T_ms = 1000;
316     int k;
317     /*  The lightshows have different 'states' depending on the local timer
318         Timeout_ms2 MODULO the period T_ms.
319     */
320     int T_mod = Timeout_ms2%T_ms;
321     switch(mode)
322     {
323     case FLICKER:
324         /* Toggles all the LEDs every half period of T_ms
325         */
326         if(T_mod > T_ms / 2)
327         {
328             set_leds(511);
329         }
330         else
331         {
332             set_leds(0);
333         }
334         break;
335     case CHARGING:
336         /* Turns on k leds when the time MODULO the period is k/9
337         */
338         for (k=0; k<9; k++)
339         {
340             if ( (T_mod > (T_ms*k/9)) && (T_mod <(T_ms*(k+1)/9)))
341             {
342                 set_leds(ones(k));
343                 break;
344             }
345         }
346         break;
347     case DISCHARGING:
348         /* Turns on 9-k leds when the time MODULO the period is k/9
349         */
```

```
350             for (k=0; k<9; k++)
351             {
352                 if ( (T_mod > (T_ms*k/9)) && (T_mod <(T_ms*(k+1)/9)))
353                 {
354                     set_leds(ones(9-k));
355                     break;
356                 }
357             }
358             break;
359         case SCAN:
360             /* Turns on LED number k when the time MODULO the period is k/9
361             */
362             for (k = 0; k<9; k++)
363             {
364                 if ( (T_mod > (T_ms*k/9)) && (T_mod <(T_ms*(k+1)/9)))
365                 {
366                     set_leds(1<<k);
367                     break;
368                 }
369             }
370             break;
371         default:
372             /*  When an incorrect argument is passed to this function,
373                 turns on a specific set of LEDs
374             */
375             set_leds(42);
376             break;
377         }
378         return;
379     }
380     int ones(int k)
381     {
382         /*  Recursively calculates the binary number that
383             consists of k ones (for example 3, 15, 511).
384             The recursive relationship that is used is:
385             N_k = 2*N_(k-1) + 1
386         */
387         if (k==1 || k==0)
388         {
389             return 1;
390         }
391         else
392         {
393             return ((ones(k-1)<<1)+1);
394         }
395     }
396     /** End of LED Functions **/
397
398     /** PWM Helper Functions **/
399     void pin_low_1()
400     {
401         LPC_GPIO1->DATA = LPC_GPIO1->DATA & ~(1<<6);
402     }
403     void pin_high_1()
404     {
405         LPC_GPIO1->DATA = LPC_GPIO1->DATA | (1<<6);
406     }
407     void pin_low_2()
408     {
409         LPC_GPIO1->DATA = LPC_GPIO1->DATA & ~(1<<7);
410     }
411     void pin_high_2()
412     {
```

```
413     LPC_GPIO1->DATA = LPC_GPIO1->DATA | (1<<7);
414 }
415 void timer_start_1()
416 {
417     LPC_TMR32B0->TCR = 0b01;
418 }
419 void timer_start_2()
420 {
421     LPC_TMR32B1->TCR = 0b01;
422 }
423 void timer_stop_1()
424 {
425     LPC_TMR32B0->TCR = 0b10;
426 }
427 void timer_stop_2()
428 {
429     LPC_TMR32B1->TCR = 0b10;
430 }
431 void update_PWM_1()
432 {
433     extern unsigned int powered_on, PWM1, PWM2;
434     timer_stop_1();
435 
436     if(powered_on==0)
437     {
438         pin_low_1();
439         return;
440     }
441 
442     if(PWM1>TOTALOUTPUTS-MAX_RANGE_OUTPUT)
443     {
444         LPC_TMR32B0->MR0 = TOTALOUTPUTS*2;
445     }else{
446         LPC_TMR32B0->MR0 = PWM1;
447     }
448     if(PWM1==0)
449     {
450         /* The PWM should stay off when it is 0
451         */
452     }
453     else
454     {
455         timer_start_1();
456     }
457 
458 }
459 void update_PWM_2()
460 {
461     extern unsigned int PWM1, PWM2;
462     timer_stop_2();
463     if(powered_on==0)
464     {
465         pin_low_2();
466         return;
467     }
468     if(PWM2>TOTALOUTPUTS-MAX_RANGE_OUTPUT)
469     {
470         LPC_TMR32B1->MR0 = TOTALOUTPUTS*2;
471     }
472     else
473     {
474         LPC_TMR32B1->MR0 = PWM2;
475     }
```

```
476     if(PWM1==0)
477     {
478
479     }
480     else
481     {
482         timer_start_2();
483     }
484 }
485 /** End of PWM Helper Functions **/
486
487 /** ADC Functions **/
488 extern unsigned int next_i;
489 extern unsigned int current1_buffer[TAKE_AVERAGE];
490 extern unsigned int current2_buffer[TAKE_AVERAGE];
491 extern unsigned int voltage1_buffer[TAKE_AVERAGE];
492 extern unsigned int voltage2_buffer[TAKE_AVERAGE];
493 extern unsigned int voltage1,voltage2,current1,current2;
494 void renew_ADC()
495 {
496     /*  renew_ADC() updates the global variables voltage1,2 and current1,2
497         when it is called. The data is filtered (by using a moving average)
498         so that high frequency noise is rejected. The filter length is ...
                defined
499         by TAKE_AVERAGE
500     */
501     unsigned int temp_current1 = 0;
502     unsigned int temp_current2 = 0;
503     unsigned int temp_voltage1 = 0;
504     unsigned int temp_voltage2 = 0;
505
506     /*  Reads out the ADC Data registers. The 10 data bits that we are
507         interested in are located at positions 6 to 15, so these are
508         extracted using a bit mask (0x3FF).
509     */
510     voltage1_buffer[next_i] = (LPC_ADC->DR0 & (0x3FF<<6))>>6;
511     voltage2_buffer[next_i] = (LPC_ADC->DR1 & (0x3FF<<6))>>6;
512     current1_buffer[next_i] = (LPC_ADC->DR2 & (0x3FF<<6))>>6;
513     current2_buffer[next_i] = (LPC_ADC->DR3 & (0x3FF<<6))>>6;
514
515     next_i++;
516     if(next_i==TAKE_AVERAGE)
517     {
518         next_i = 0;
519     }
520
521     /*  The moving average filter operation is performed below by adding
522         the most recent TAKE_AVERAGE elements and dividing by TAKE_AVERAGE
523     */
524     for(int i = 0; i<TAKE_AVERAGE; i++)
525     {
526         temp_current1 += current1_buffer[i];
527         temp_current2 += current2_buffer[i];
528         temp_voltage1 += voltage1_buffer[i];
529         temp_voltage2 += voltage2_buffer[i];
530     }
531     current1 = temp_current1/TAKE_AVERAGE;
532     current2 = temp_current2/TAKE_AVERAGE;
533     voltage1 = temp_voltage1/TAKE_AVERAGE;
534     voltage2 = temp_voltage2/TAKE_AVERAGE;
535 }
536 /** End of ADC Functions **/
537
```

```
538 /** Delay Functions **/
539 /*  Provided by the TU Delft EE3D11 Course Lab
540 */
541 static uint32_t ticks_in_ms = CLK_FREQ/1000;
542 static uint32_t ticks_in_us = CLK_FREQ/1000000;
543 void init_delay (void)
544 {
545     SystemCoreClockUpdate ();
546     ticks_in_ms = (SystemCoreClock/1000);
547     ticks_in_us = (SystemCoreClock/1000000);
548 }
549 void delay_us (uint32_t us) __attribute__ ((optimize("Os"), noclone));
550 void delay_ms (uint32_t ms) __attribute__ ((optimize("Os"), noclone));
551 void delay_ms (uint32_t ms)
552 {
553     while (ms--)
554     {
555         delay_us (1000);
556     }
557 }
558 void delay_us (uint32_t us)
559 {
560     static uint32_t local_ticks_in_us;
561     while (us--)
562     {
563         local_ticks_in_us = ticks_in_us/8;
564         do { __NOP (); __NOP (); } while (--local_ticks_in_us);
565     }
566 }
567 /** End of Delay Functions **/
568 #endif // IMPLEMENTATIONS_C
```

Listing A.3: ISR_Handlers.c

```
1  #ifndef ISR_HANDLERS_C
2  #define ISR_HANDLERS_C
3  #include "LPC13xx.h"
4  #include "definitions.h"
5  #include "prototypes.h"
6  extern uint8_t tmp_rcv, toSend;
7  extern int control;
8  extern int lengthLeft;
9  void TIMER_32_0_Handler()
10 {
11     /*  This function asserts and deasserts the PWM signal of TMR32B0
12      based on an NSD given in the Design Report.
13     */
14
15     if(LPC_TMR32B0->IR & (0b1<<3))    // If MR3 is set
16     {
17         pin_high_1();
18         LPC_TMR32B0->IR = (1<<3);
19     }
20     if(LPC_TMR32B0->IR & 0b1)      // If MR0 is set
21     {
22         pin_low_1();
23         LPC_TMR32B0->IR = 0b1;
24     }
25 }
26 void TIMER_32_1_Handler()
27 {
28     /*  This function asserts and deasserts the PWM signal of TMR32B1
```

```
29       based on an NSD given in the Design Report.
30      */
31      if(LPC_TMR32B1->IR & (0b1<<3))    // If MR3 is set
32      {
33          pin_high_2();
34          LPC_TMR32B1->IR = (1<<3);
35      }
36      if(LPC_TMR32B1->IR & 0b1)      // If MR0 is set
37      {
38          pin_low_2();
39          LPC_TMR32B1->IR = 0b1;
40      }
41  }
42  void TIMER_16_0_Handler()
43  {
44      /*  This function increments Timeoutms_1,2
45          every time it is called.
46      */
47      extern int Timeout_ms2,Timeout_ms;
48      Timeout_ms++;
49      Timeout_ms2++;
50      LPC_TMR16B0->IR = 1;
51  }
52  void I2C_Handler()
53  {
54      /*  This function allows communication by following the I2C
55          protocol when an event happens on the I2C bus.
56          LPC_I2C->STAT contains information about which event
57          happened.
58      */
59
60      switch(LPC_I2C->STAT)
61      {
62      //  Own Slave Address + write bit has been received.
63      //  ACK returned.
64      case 0x60:
65          LPC_I2C->CONSET = AA;
66          break;
67
68      //  Data has been received.
69      //  ACK returned.
70      case 0x80:
71          tmp_rcv = LPC_I2C->DAT;
72          LPC_I2C->CONSET = AA;
73          receiveEvent(tmp_rcv);
74          break;
75
76      //  Own Slave Address + read bit has been received
77      //  ACK returned.
78      case 0xA8:
79          LPC_I2C->CONSET = AA;
80          requestEvent();
81          LPC_I2C->DAT = toSend;          // Loads data into I2CDAT register
82          control = ONLINE;
83          break;
84
85      //  Data has been transmitted.
86      //  ACK received.
87      case 0xB8:
88          if (lengthLeft==0)
89          {
90              LPC_I2C->CONCLR = AA;
91          }
```

```
92          break;
93      default:
94          break;
95      }
96      LPC_I2C->CONCLR = SI; // Clear the interrupt register after the ISR has ...
            been executed
97  }
98  #endif // ISR_HANDLERS_C
```

Listing A.4: definitions.h

```
1   #ifndef DEFINITIONS_H
2   #define DEFINITIONS_H
3
4   /** I2C Interrupt Register Codes **/
5   #define AA (1<<2)                   //Assert Acknowledge
6   #define SI (1<<3)                   //System Interrupt Flag
7   #define STO (1<<4)                  //Stop Condition
8   #define STA (1<<5)                  //Start Condition
9   #define I2EN (1<<6)                 //Enable I2C module
10
11  /** MCU Specific Definitions **/
12  #define CLK_FREQ    72000000
13
14  /** Custom I2C states **/
15  #define I2C_SLAVE_ADDRESS 3
16  #define RESET_STATE 0
17  #define VOLT_READ_STATE 1
18  #define CURRENT_READ_STATE 2
19  #define VOLT1_WRITE_STATE 3
20  #define VOLT2_WRITE_STATE 4
21  #define CURRENT1_WRITE_STATE 5
22  #define CURRENT2_WRITE_STATE 6
23  #define READ_SLAVE_DIRECTION 7
24  #define MASTER_SET_DIRECTION 8
25  #define TURN_ON_OFF 9
26  #define READ_POWER_STATE 10
27  #define SET_PWM_1_STATE 11
28  #define SET_PWM_2_STATE 12
29  #define READ_PWM_1_STATE 13
30  #define READ_PWM_2_STATE 14
31  #define NEW_ADDRESS_STATE 15
32  #define READ_CHARGE_STATE 16
33  #define INT_SIZE_BYTE 4
34  #define MAX_BEGIN_STATE READ_CHARGE_STATE
35  #define numberOfElementsInMyArray 10
36  #define ONLINE 999
37  #define OFFLINE 888
38
39
40  /** PWM Variable Definitions **/
41  #define PreScaleC 0
42  #define TOTALOUTPUTS 1023
43  #define MAX_RANGE_OUTPUT 7
44  #define PWM1_START 256
45  #define PWM2_START 512
46
47
48  #define TAKE_AVERAGE 50
49
50  /** FSM Charging Definitions **/
51  #define cRESET 0
```

```
52 #define cRAMP_VOLTAGE 1
53 #define cNOMINAL 2
54 #define cREADY 3
55 #define cPINCH_OFF 4
56 #define cDONE 5
57 #define dV 5
58 #define INCREMENT_THRESHOLD 100
59 #define DIFF_THRESHOLD 1.20
60
61 /** LED Visual Effect Definitions **/
62 #define FLICKER 0
63 #define SCAN 1
64 #define CHARGING 2
65 #define DISCHARGING 3
66
67
68 /** Miscellaneous Function Definitions **/
69 #define MIN(a,b) ((a) < (b) ? (a) : (b))
70 #define MAX(a,b) ((a) > (b) ? (a) : (b))
71
72 #endif
```

Appendix B: Master Source Code

Listing B.1: main.cpp

```
1  #include <iostream>
2
3  #include "mastermind.h"
4
5  // Define different variables that influence the timing of the program
6  #define TIMES_BEFORE_CHECK 60
7  #define TIMES_BEFORE_INSTRUCTION 1
8  #define TIMES_BEFORE_SEND_SERVER 3
9  #define DELAY_MS 300
10
11 using namespace std;
12
13 int main()
14 {
15
16     // Initialize the 'mastermind', this is the main framework of the program...
           , where to urls are pages to send data to and load instructions from, ...
           the two ints give range of possible I2C addresses
17     mastermind meester("http://solarpoweredbikes.tudelft.nl/bap2017/receive....
           php","http://solarpoweredbikes.tudelft.nl/bap2017/instructions.php",3,...
            25);
18
19     // Init all the used counters
20     int refresh_counter = 0;
21     int instruction_counter = 0;
22     int server_counter = 0;
23
24     while(1)
25     {
26         // Fail-safe, try-catch system to catch errors on web request error
27         try
28         {
29             if(meester.slave_size()>0 && refresh_counter<TIMES_BEFORE_CHECK)
30             {
31                 meester.get_states(); // Possibility to output the statuses ...
                       to a local interface.
32                 meester.print_all();
33
34                 // Send data to server, if done too many times, data storage ...
                       will become a problem
35                 if(server_counter≥TIMES_BEFORE_SEND_SERVER)
36                 {
37                     meester.send_to_server();
38                     server_counter = 0;
39                 }
```

```
40
41                  // Read instruction from server
42                  if(instruction_counter≥TIMES_BEFORE_INSTRUCTION)
43                  {
44                      meester.check_for_instructions();
45                      instruction_counter = 0;
46                  }
47
48                  // Increment all counters
49                  refresh_counter++;
50                  instruction_counter++;
51                  server_counter++;
52              }
53              else
54              {
55                  // If no slaves were connected or refresh_counter hit ...
                        TIMES_BEFORE_CHECK, scan for slaves
56                  meester.check_slaves();
57                  refresh_counter = 0;
58                  instruction_counter = 0;
59                  server_counter = 0;
60              }
61
62              // Wait some time before restart
63              delay(DELAY_MS);
64          }
65          catch (...)
66          {
67              cout << "Error. Starting over." << endl;
68          }
69      }
70
71      return 0;
72  }
```

Listing B.2: mastermind.h

```
1  #ifndef MASTERMIND_H
2  #define MASTERMIND_H
3
4  #include "power_manager.h"
5  #include "slave_list.h"
6
7  // Basic libraries for data structures, string handling and data streams
8  #include <vector>
9  #include <string>
10 #include <sstream>
11 #include <iostream>
12
13 // Libraries needed for web request and data encoding
14 #include <curlpp/cURLpp.hpp>
15 #include <curlpp/Easy.hpp>
16 #include <curlpp/Options.hpp>
17 #include <curlpp/Exception.hpp>
18 #include "json.hpp"
19
20 // For convenience, shorter code
21 using json = nlohmann::json;
22 using namespace std;
23
24 class mastermind
25 {
```

```
26      public:
27          mastermind(string url_new, string instruction_url_new, int newMin, ...
                int newMax);
28          virtual ¬mastermind();
29          void check_slaves();
30          void get_states();
31          void print_all();
32          void send_to_server();
33          int slave_size();
34          void check_for_instructions();
35      protected:
36      private:
37          slave_list list_of_slaves; // Instance of slave list, this is a class...
                 to keep track of all connected slaves and check for new slaves. ...
                See the slave_list class;
38
39          power_manager p_manager; // Instance of power manager, this is a ...
                class to read and write instructions to a slave address. See the ...
                power_manager class;
40
41          int amount_of_slaves; // The amount of currently connected slaves
42          vector< vector<int> > states_vec; // This is the vector with all the ...
                different values of inputs, outputs and states of each slave
43
44          string server_url;
45          string instructions_url;
46          void parseInstructions(string data);
47          int newAddress, maxAddress, minAddress;
48          void new_device();
49  };
50
51  #endif // MASTERMIND_H
```

Listing B.3: mastermind.cpp

```
1   #include "mastermind.h"
2
3
4   // Initialize different instruction cases with enums and hashing
5   enum string_code
6   {
7       nothing,
8       writevoltage,
9       writecurrent,
10      setdirection,
11      turnoff,
12      turnon,
13      setpwm1,
14      setpwm2
15  };
16
17  string_code hashit(string const& inString)
18  {
19      if(inString == "writevoltage") return writevoltage;
20      if(inString == "writecurrent") return writecurrent;
21      if(inString == "setdirection") return setdirection;
22      if(inString == "turnoff") return turnoff;
23      if(inString == "turnon") return turnon;
24      if(inString == "setpwm1") return setpwm1;
25      if(inString == "setpwm2") return setpwm2;
26      return nothing;
27  }
```

```
28 // End init of enums
29
30
31 // Initialize function, set up all the needed variables and check for slaves
32 mastermind::mastermind(string url_new, string instruction_url_new, int newMin...
       , int newMax)
33 {
34     // Variable part
35     list_of_slaves = slave_list(true, newMin, newMax);
36     minAddress = newMin;
37     maxAddress = newMax;
38     newAddress = newMin+1;
39     server_url = url_new;
40     instructions_url = instruction_url_new;
41
42     // First check for slaves and get the status for the first time
43     check_slaves();
44     get_states();
45     //ctor
46 }
47
48 mastermind::~mastermind()
49 {
50     // No need to delete any elements (such as dynamic arrays), standard ...
           library handles the vectors
51     //dtor
52 }
53
54
55 // Function to check all slaves
56 void mastermind::check_slaves()
57 {
58     list_of_slaves.add_slaves();
59     amount_of_slaves = list_of_slaves.get_size();
60
61     // Checks if a new slave with the minAddress is connected, if so, send a ...
           new address to it
62     cout << "Checking slaves, amount " << amount_of_slaves << endl;
63     if(amount_of_slaves>0&&list_of_slaves[0]==minAddress){
64         cout << "New device!" << endl;
65         new_device();
66     }
67 }
68
69 // The function to send a new dynamic address to a newly connected slave
70 void mastermind::new_device(){
71     p_manager.change_address(list_of_slaves[0]); //First connect to the slave...
           at minAddress
72     p_manager.write_address(newAddress); //Send its new address
73     list_of_slaves[0]=newAddress;
74     newAddress++;
75     if(newAddress>maxAddress){
76         newAddress = minAddress+1;
77     }
78 }
79
80 // Get the status updates for all the connected slaves
81 void mastermind::get_states()
82 {
83     states_vec.clear();
84     for(int i=0; i<amount_of_slaves; i++)
85     {
86         p_manager.change_address(list_of_slaves[i]); // Select a slave from ...
```

```
                the list
87
88          // Create a new row and read out all different values
89          vector<int> row;
90          row.push_back(p_manager.get_power_status());
91          row.push_back(p_manager.get_direction());
92          row.push_back(p_manager.read_current(1));
93          row.push_back(p_manager.read_current(2));
94          row.push_back(p_manager.read_voltage(1));
95          row.push_back(p_manager.read_voltage(2));
96          row.push_back(p_manager.read_pwm(1));
97          row.push_back(p_manager.read_pwm(2));
98          row.push_back(p_manager.read_charge_state());
99
100         // Put the result row into the states vector
101         states_vec.push_back(row);
102     }
103
104     //Check if disconnected
105     int i = 0;
106     while(i<amount_of_slaves)
107     {
108         if(states_vec[i][0]==-1)
109         {
110             //Disconnected!
111             list_of_slaves.remove_slave(list_of_slaves[i]);
112             states_vec.erase(states_vec.begin()+i);
113             amount_of_slaves--;
114         }
115         else
116         {
117             i++;
118         }
119     }
120 }
121
122
123  // Simply print all statues, create simple outputs
124 void mastermind::print_all()
125 {
126     for(int i=0; i<amount_of_slaves; i++)
127     {
128         cout << "Slave " << list_of_slaves[i] << ": ";
129         for(int j = 0; j<states_vec[i].size();j++){
130             cout << states_vec[i][j] << " ";
131         }
132         cout << endl;
133     }
134
135 }
136
137
138
139 // Function to take all data from the states_vec and put it into a json data ...
        package (text). Then send it to the predefined server url.
140 void mastermind::send_to_server()
141 {
142
143     json data(states_vec);
144     json slaves(list_of_slaves.return_vector());
145
146     json json_to_send;
147     json_to_send["slaves"] = slaves;
```

```
148     json_to_send["data"] = data;
149
150     // Output the data that will be sent to terminal, optional
151     //cout << json_to_send << endl;
152
153     try
154     {
155         curlpp::Cleanup clean;
156         curlpp::Easy request;
157         request.setOpt<curlpp::options::Url>(server_url);
158         list<string> header;
159         header.push_back("Content-Type: application/x-www-form-urlencoded");
160         request.setOpt(new curlpp::options::HttpHeader(header));
161         string test = "status="+json_to_send.dump();
162         request.setOpt(new curlpp::options::PostFields(test));
163         request.setOpt(new curlpp::options::Timeout(5L));
164         request.setOpt(new curlpp::options::PostFieldSize(test.length()));
165
166         ostringstream os;
167         os << request;
168
169         // Optional output of result
170         /*string data = os.str();
171         cout <<"Result: " << data << endl;*/
172     }
173     catch(curlpp::RuntimeError& e)
174     {
175         cout << "Error making request of type " << e.what();
176     }
177 }
178
179
180 int mastermind::slave_size()
181 {
182     return amount_of_slaves;
183 }
184
185
186 // Read the instruction from the instructions url
187 void mastermind::check_for_instructions()
188 {
189     try
190     {
191         curlpp::Cleanup clean;
192         curlpp::Easy request;
193         request.setOpt<curlpp::options::Url>(instructions_url);
194         request.setOpt(new curlpp::options::Timeout(5L));
195
196         // Perform the request and put its output into a stream, then convert...
                it to a string
197         ostringstream os;
198         os << request;
199         string data = os.str();
200
201         // Parse the received data in another function
202         parseInstructions(data);
203     }
204     catch(curlpp::RuntimeError& e)
205     {
206         cout << "Error making request of type " << e.what();
207     }
208 }
209
```

```
210
211 // Parsing the received data
212 void mastermind::parseInstructions(string data)
213 {
214     json temp = json::parse(data);
215
216     // If the received data doesn't consist of a slave, instruction type and ...
        data, quit (3 values needed). For example, during disconnection
217     if(temp.size()<3)
218     {
219         return;
220     }
221
222     // Get the slave number and connect to that slave
223     int temp_slave= stoi(temp["slave"].get<string>());
224     p_manager.change_address(temp_slave);
225
226     int data_to_send;
227
228     // Depending on the instruction, do something, case-switch works well ...
        with enums, so hash it to make life easier
229     switch (hashit(temp["instruct_type"].get<string>()))
230     {
231     case writevoltage:
232         // Convert data string to an int
233         data_to_send = stoi(temp["data"].get<string>());
234         p_manager.write_voltage(data_to_send);
235         break;
236     case writecurrent:
237         data_to_send = stoi(temp["data"].get<string>());
238         p_manager.write_current(data_to_send);
239         break;
240     case setdirection:
241         data_to_send = stoi(temp["data"].get<string>());
242         p_manager.set_direction(data_to_send);
243         break;
244     case turnoff:
245         p_manager.turn_off();
246         break;
247     case turnon:
248         p_manager.turn_on();
249         break;
250     case setpwm1:
251         data_to_send = stoi(temp["data"].get<string>());
252         p_manager.set_pwm(1,data_to_send);
253         break;
254     case setpwm2:
255         data_to_send = stoi(temp["data"].get<string>());
256         p_manager.set_pwm(2,data_to_send);
257         break;
258     default:
259
260         break;
261     }
262 }
```

Listing B.4: power_manager.h

```
1 #ifndef POWER_MANAGER_H
2 #define POWER_MANAGER_H
3
4 #include "slave_handler.h"
```

```
5
6  class power_manager
7  {
8  public:
9
10     // First init function and connect to first address
11     power_manager(int first_address = 0)
12     {
13         address = first_address;
14         handler.set_addr(address);
15     };
16
17     // Connect to a new slave address
18     void change_address(int new_address){
19         address = new_address;
20         handler.set_addr(address);
21     };
22
23     int read_voltage(int which = 1);
24     int read_current(int which = 1);
25     int write_voltage(int new_voltage);
26     int write_current(int new_current);
27     int get_direction();
28     int set_direction(int new_direction);
29     int turn_off();
30     int turn_on();
31     int get_power_status();
32     int set_pwm(int which, int value);
33     int read_pwm(int which);
34     int write_address(int new_address);
35     int read_charge_state();
36     virtual ¬power_manager();
37 protected:
38 private:
39     int address; // For saving address of currently connected slave
40
41     slave_handler handler; // I2C slave handler, see class slave_handler
42 };
43
44 #endif // POWER_MANAGER_H
```

Listing B.5: power_manager.cpp

```
1  #include "power_manager.h"
2
3  power_manager::¬power_manager()
4  {
5      //dtor
6  }
7
8  // Write max voltage to a slave. State code = 1
9  int power_manager::write_voltage(int new_voltage)
10 {
11     vector<int> temp_data;
12
13     // ODroid works with 32 bit integers. So split into 4 bytes, and then ...
           send them sequentially
14     for(int i=0; i<4; i++)
15     {
16         uint8_t toSend = (uint8_t)((new_voltage & (0x00FF << (i*8))) >> (i*8)...
               );
17         temp_data.push_back(toSend);
```

```
18      }
19      handler.send_data(1,4,temp_data);
20      return 1;
21
22  }
23
24  // Write max current to a slave. State code = 2
25  int power_manager::write_current(int new_current)
26  {
27      vector<int> temp_data;
28      for(int i=0; i<4; i++)
29      {
30          uint8_t toSend = (uint8_t)((new_current & (0x00FF << (i*8))) >> (i*8)...
                );
31          temp_data.push_back(toSend);
32      }
33      handler.send_data(2,4,temp_data);
34      return 1;
35  }
36
37  // Read voltage, states codes 3 and 4, reuse function below
38  int power_manager::read_voltage(int which)
39  {
40      return read_current(which-2);
41  }
42
43  // Read current, states codes 5 and 6
44  int power_manager::read_current(int which)
45  {
46      vector<int> temp_data = handler.read_data(4+which);
47      int sum = 0;
48
49      // Master will receive as seperate bytes. Combine them by bit shifting ...
            every data package accordingly and sum to get the original integer
50      for(unsigned int i =0; i<temp_data.size(); i++)
51      {
52          sum += (temp_data[i] << i*8);
53      }
54
55      return sum;
56  }
57
58  // Read the current direction, state code 7
59  int power_manager::get_direction()
60  {
61      vector<int> temp_data = handler.read_data(7);
62
63      if(temp_data.size()==1)
64      {
65          return (bool)temp_data[0];
66      }
67      else
68      {
69          cout << "Error getting direction" << endl;
70          return -1;
71      }
72  }
73
74  // Set the direction, state code 8
75  int power_manager::set_direction(int new_direction)
76  {
77      vector<int> temp_data;
78      temp_data.push_back(new_direction);
```

```
79
80      handler.send_data(8,1,temp_data);
81      return 1;
82  }
83
84  // Turn off, state code 9
85  int power_manager::turn_off()
86  {
87      vector<int> temp_data;
88      temp_data.push_back(0);
89
90      handler.send_data(9,1,temp_data);
91      return 1;
92  }
93
94  // Turn on, state code 9, but with different data
95  int power_manager::turn_on()
96  {
97      vector<int> temp_data;
98      temp_data.push_back(1);
99
100     handler.send_data(9,1,temp_data);
101     return 1;
102 }
103
104 // Read if the devices is turned on, state code 10
105 int power_manager::get_power_status()
106 {
107     vector<int> temp_data = handler.read_data(10);
108
109     if(temp_data.size()==1)
110     {
111         return (bool)temp_data[0];
112     }
113     else
114     {
115         cout << "Error getting power status" << endl;
116         return -1;
117     }
118 }
119
120 // Overwrite PWM outputs, state code 11 and 12, only useful for debugging
121 int power_manager::set_pwm(int which, int value)
122 {
123     vector<int> temp_data;
124     cout << endl << "Setting PWM" << which << " to " << value <<endl;
125     for(int i=0; i<2; i++)
126     {
127         uint8_t toSend = (uint8_t)((value & (0x00FF << (i*8))) >> (i*8));
128         temp_data.push_back(toSend);
129     }
130
131     if(which==1)
132     {
133         handler.send_data(11,2,temp_data);
134     }
135     else if(which==2)
136     {
137         handler.send_data(12,2,temp_data);
138     }
139     return 1;
140 }
141
```

```
142 // Read the current PWM outputs, state codes 13 and 14
143 int power_manager::read_pwm(int which)
144 {
145     vector<int> temp_data = handler.read_data(12+which);
146     int sum = 0;
147 
148     for(unsigned int i =0; i<temp_data.size(); i++)
149     {
150         sum += (temp_data[i] << i*8);
151     }
152 
153     return sum;
154 }
155 
156 // Write a new address to the slave, this makes dynamic addressing possible, ...
        state code 15
157 int power_manager::write_address(int new_address)
158 {
159     vector<int> temp_data;
160     temp_data.push_back(new_address);
161     handler.send_data(15,1,temp_data);
162     return new_address;
163 }
164 
165 // Read in which state the charging process is, state code 16
166 int power_manager::read_charge_state()
167 {
168     vector<int> temp_data = handler.read_data(16);
169     int sum = 0;
170 
171     for(unsigned int i =0; i<temp_data.size(); i++)
172     {
173         sum += (temp_data[i] << i*8);
174     }
175 
176     return sum;
177 }
```

Listing B.6: slave_list.h

```
1  #ifndef SLAVE_LIST_H
2  #define SLAVE_LIST_H
3  
4  // Output and vector
5  #include <iostream>
6  #include <vector>
7  
8  // I2C
9  #include <wiringPi.h>
10 #include <wiringPiI2C.h>
11 #include <unistd.h>
12 
13 // To check connection by reading a byte or sending a byte
14 #define WRITE_MODE 1
15 #define READ_MODE 0
16 
17 using namespace std;
18 
19 class slave_list
20 {
21     public:
22         slave_list(bool write_set = true, int set_min = 3, int set_max = 127)...
```

```
            ;
23          virtual ¬slave_list();
24          void add_slaves();
25          void print_slaves();
26          bool remove_slave(int device_number);
27          int get_size();
28          int get_addr(int i);
29          vector<int> return_vector();
30          int& operator[] (int x){
31              return slaves[x];
32          }
33      protected:
34      private:
35          vector<int> slaves; // List of connected slaves
36          int mode; // Read or write mode when connecting
37          int size_array; // Counter of the amount of slaves
38          int min_device;
39          int max_device;
40  };
41
42  #endif // SLAVE_LIST_H
```

Listing B.7: power_list.cpp

```
1   #include "slave_list.h"
2
3   // Slave list init, setting which mode is used for check and set the address ...
        range
4   slave_list::slave_list(bool write_set, int set_min, int set_max)
5   {
6       if(write_set)
7       {
8           mode = WRITE_MODE;
9       }
10      else
11      {
12          mode = READ_MODE;
13      }
14      size_array = 0;
15      min_device = set_min;
16      max_device = set_max;
17      //ctor
18  }
19
20  slave_list::¬slave_list()
21  {
22      //dtor
23  }
24
25  // Slave scanning function
26  void slave_list::add_slaves()
27  {
28      slaves.clear();
29      size_array = 0;
30
31      // Check every devices in range (inclusive)
32      for(int device=min_device; device≤max_device; device++)
33      {
34          int fd = wiringPiI2CSetup(device);
35
36          if(fd>−1)
37          {
```

```
38              if(mode==WRITE_MODE)
39              {
40                  // If write mode, write a byte
41                  int temp = wiringPiI2CWrite(fd,-1);
42
43                  // If there is a response, push connected slave to the list
44                  if(temp>-1)
45                  {
46                      slaves.push_back(device);
47                      size_array++;
48                  }
49              }
50              else
51              {
52                   // If read mode, read a byte
53                  int temp = wiringPiI2CRead(fd);
54
55                  // If there is a response, push connected slave to the list
56                  if(temp>-1){
57                      slaves.push_back(device);
58                      size_array++;
59                  }
60              }
61          }
62          close(fd);
63      }
64  }
65
66  // Print all the connected slaves, used for debugging
67  void slave_list::print_slaves()
68  {
69      cout << "List of slaves: " << endl;
70      for(int i=0; i<size_array; i++)
71      {
72          cout << "Slave " << i << ": " << slaves[i] << endl;
73      }
74  }
75
76  // Removing a slave from the list (when disconnecting for instance)
77  bool slave_list::remove_slave(int device_number)
78  {
79      // Iterate through list
80      for(int i=0; i<size_array; i++)
81      {
82          // If the current slave is the one to delete, delete it and decrease ...
                counter of slaves
83          if(slaves[i]==device_number)
84          {
85              slaves.erase(slaves.begin()+i);
86              size_array--;
87              return true;
88          }
89      }
90
91      return false;
92  }
93
94  int slave_list::get_size()
95  {
96      return size_array;
97  }
98
99  int slave_list::get_addr(int i){
```

```
100     return slaves[i];
101 }
102
103 vector<int> slave_list::return_vector(){
104     return slaves;
105 }
```

Listing B.8: slave_handler.h

```
1  #ifndef SLAVE_HANDLER_H
2  #define SLAVE_HANDLER_H
3
4  // Standard libraries needed for output and data structure
5  #include <iostream>
6  #include <vector>
7
8  // wiringPi library for easy I2C connection
9  #include <wiringPi.h>
10 #include <wiringPiI2C.h>
11
12 //Needed for closing file handlers
13 #include <unistd.h>
14
15 using namespace std;
16
17 class slave_handler
18 {
19     public:
20         slave_handler(int device_number = 0);
21         int send_data(int send_mode, int send_length, vector<int> send_data);
22         vector<int> read_data(int read_mode);
23         void set_addr(int new_address);
24         virtual ¬slave_handler();
25     protected:
26     private:
27         int device;
28 };
29
30 #endif // SLAVE_HANDLER_H
```

Listing B.9: power_handler.cpp

```
1  #include "slave_handler.h"
2
3  slave_handler::slave_handler(int device_number)
4  {
5      device = device_number;
6      //ctor
7  }
8
9  slave_handler::¬slave_handler()
10 {
11     //dtor
12 }
13
14 int slave_handler::send_data(int send_mode, int send_length, vector<int> ...
       send_data)
15 {
16
17     // Check for the connection
18     int fd = wiringPiI2CSetup(device);
```

```
19      if(fd<0)
20      {
21          cout << "Can't connect. " << endl;
22          return -2;
23      }
24
25      // Check if writeable
26      int temp = wiringPiI2CWrite(fd,-1);
27      if(temp<0)
28      {
29          cout << "Can't write to device. " << endl;
30          close(fd);
31          return -1;
32      }
33
34
35      // Send which state code is used
36      int return_value = wiringPiI2CWrite(fd,send_mode);
37      if(return_value==-1)
38      {
39          cout << "Send mode not written correctly. " << endl;
40          return -3;
41      }
42
43      // Send which how many bytes will be sent
44      return_value = wiringPiI2CWrite(fd,send_length);
45      if(return_value==-1)
46      {
47          cout << "Send length not written correctly. " << endl;
48          return -4;
49      }
50
51      // Send all the bytes
52      for(int i=0; i<send_length; i++)
53      {
54          return_value = wiringPiI2CWrite(fd,send_data[i]);
55
56      }
57
58      close(fd);
59      return 1;
60  }
61
62  //  Read Data function, this functions returns a vector with all the received...
        bytes
63  vector<int> slave_handler::read_data(int read_mode)
64  {
65      vector<int> output;
66
67      // Check if connection is possible
68      int fd = wiringPiI2CSetup(device);
69      if(fd<0)
70      {
71          cout << "Can't connect. " << endl;
72          return output;
73      }
74
75      // Send read_mode, which state
76      int return_value;
77      int temp = wiringPiI2CWrite(fd,read_mode);
78      if(temp<0)
79      {
80          close(fd);
```

```
81          return output;
82      }
83
84      // Receive the length of bytes to read
85      int read_length = wiringPiI2CRead(fd);
86
87      // Receive bytes and put them into output
88      for(int i=0; i<read_length; i++)
89      {
90          return_value = wiringPiI2CRead(fd);
91          if(return_value>-1)
92          {
93              output.push_back(return_value);
94          }
95      }
96
97      // Close the connection to the device
98      close(fd);
99
100     return output;
101 }
102
103 // End Read function
104
105 // Change connected address
106 void slave_handler::set_addr(int new_address)
107 {
108     device = new_address;
109 }
```

Appendix C: Server Source Code

Listing C.1: index.php

```
<!DOCTYPE html>
<html>

<head>
    <title>BAP 2017 - Groep B</title>
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.css" rel="stylesheet" type="text/css">
    <script src="https://code.jquery.com/jquery-3.1.1.js">
    </script>
    <script>
        function submitinstr() {
            $.ajax({
                url: "new_instruction.php?slave_addr=" + $("#instruction").find('#slave_addr').val() + "&instruction_type=" + $("#instruction").find('#instruction_type').val() + "&data=" + $("#instruction").find('#data').val()
            }).done(function(data) {
                console.log(data);
            });
            return false;
        }
    </script>
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" type="text/css">
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js">
    </script>
    <link href="style.css" rel="stylesheet" type="text/css">
</head>

<body>
    <div class="main">
        <center>
            <h1>Network status</h1>
            <div class="time"></div>
            <br>
            <img src="overview.png" width="100%" style="max-width:800px;">
            <br>
            <br>
            <a href="http://solarpoweredbikes.tudelft.nl/bap2017/Model/3dmodel.html">
                <button type="button" class="btn btn-primary slave">Show Model</button>
            </a>
            <br>
```

```
38              <br>
39              <h2>Connected devices:</h2>
40              <br>
41              <br>
42              <div class="devices"></div>
43          </center>
44      </div>
45      <script>
46          var oldOptions = {};
47          var newOptions = {};
48          var times = [];
49          var values = [];
50          var dataArray = [];
51          var xData = [];
52          var x = 0;
53          var which = 0;
54          var oldtime;
55
56          function pressed(num) {
57              console.log(num);
58              window.location = "next.php?num=" + num;
59          }
60
61          function refresh() {
62              $.ajax({
63                  url: "lateststatus.php"
64              }).done(function(data) {
65                  var arr = data.split('////');
66                  if (arr[0] == oldtime) {
67                      console.log("Al gehad");
68                      return;
69                  }
70                  oldtime = arr[0];
71                  $(".time").html(arr[0]);
72                  times.push(arr[0]);
73                  var obj = JSON.parse(arr[1]);
74                  values.push(obj);
75
76                  var jsonLength = obj["slaves"].length;
77                  var tempHTML = "";
78
79                  for (var i = 0; i < jsonLength; i++) {
80                      var newDiv = '<button onclick="pressed(' + obj["slaves"][...
                            i] + ')" type="button" class="btn btn-primary slave">...
                            Converter: ' + obj["slaves"][i] + '</button><br><br>';
81                      tempHTML += newDiv;
82
83                  }
84
85                  $(".devices").html(tempHTML);
86                  $(".slave").css("width", "90%");
87                  $(".btn").css("font-size", "200%");
88                  $(".slave").css("display", "inline-block");
89              });
90          }
91          setInterval(refresh, 500);
92          $(".overlay").hide();
93      </script>
94  </body>
95
96  </html>
```

Listing C.2: next.php

```
<!DOCTYPE html>
<html>

<head>
    <title>BAP 2017 - Groep B</title>
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css.../font-awesome.css" rel="stylesheet" type="text/css">

    <script type="text/javascript" src="//cdnjs.cloudflare.com/ajax/libs/.../jquery/2.1.4/jquery.min.js"></script>
<script type="text/javascript" src="//cdnjs.cloudflare.com/ajax/libs/jqueryui.../1.11.4/jquery-ui.min.js"></script>
<script type="text/javascript" src="//cdnjs.cloudflare.com/ajax/libs/jqueryui.../-touch-punch/0.2.3/jquery.ui.touch-punch.min.js"></script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.4.0/Chart..../min.js"></script>
    <script>
        function power_on() {
            $.ajax({
                url: "new_instruction.php?slave_addr=<?php echo $_REQUEST['.../num'];?>&instruction_type=turnon&data="
            }).done(function(data) {
                console.log(data);
            });
        }

        function power_off() {
            $.ajax({
                url: "new_instruction.php?slave_addr=<?php echo $_REQUEST['.../num'];?>&instruction_type=turnoff&data="
            }).done(function(data) {
                console.log(data);
            });
        }

        function max_volt() {
            $.ajax({
                url: "new_instruction.php?slave_addr=<?php echo $_REQUEST['.../num'];?>&instruction_type=writevoltage&data="+$( "#maxvolt..." ).slider( "value" )
            }).done(function(data) {
                console.log(data);
            });
        }

        function max_current() {
            $.ajax({
                url: "new_instruction.php?slave_addr=<?php echo $_REQUEST['.../num'];?>&instruction_type=writecurrent&data="+$( "#.../maxcurrent" ).slider( "value" )
            }).done(function(data) {
                console.log(data);
            });
        }

        function power_off() {
            $.ajax({
                url: "new_instruction.php?slave_addr=<?php echo $_REQUEST['.../num'];?>&instruction_type=turnoff&data="
            }).done(function(data) {
                console.log(data);
```

```
51              });
52          }
53
54          function switch_direction() {
55              if (direction) {
56                  direction = 0;
57              } else {
58                  direction = 1;
59              }
60
61              $.ajax({
62                  url: "new_instruction.php?slave_addr=<?php echo $_REQUEST['...
                        num'];?>&instruction_type=setdirection&data=" + direction
63              }).done(function(data) {
64                  console.log(data);
65              });
66
67          }
68
69
70          function submitinstr() {
71              $.ajax({
72                  url: "new_instruction.php?slave_addr=" + $("#instruction").....
                        find('#slave_addr').val() + "&instruction_type=" + $("#...
                        instruction").find('#instruction_type').val() + "&data=" +...
                         $("#instruction").find('#data').val()
73              }).done(function(data) {
74                  console.log(data);
75              });
76              return false;
77          }
78      </script>
79      <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap...
            .min.css" rel="stylesheet" type="text/css">
80      <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap...
            .min.js">
81      </script>
82      <script src="https://cdn.rawgit.com/nelsonic/range-touch/master/range-...
            touch.min.js">
83      </script>
84      <link href="style.css" rel="stylesheet" type="text/css">
85      <link href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8/themes/base...
            /jquery-ui.css" rel="stylesheet" type="text/css">
86      <script>
87    $( function() {
88      $( "#maxvolt" ).slider({
89          min: 0,
90        max: 1024,
91        value: 1024
92        });
93    } );
94
95    $( function() {
96      $( "#maxcurrent" ).slider({
97          min: 0,
98        max: 1024,
99        value: 1024
100       });
101   } );
102   </script>
103 </head>
104
105 <body>
```

```
106     <div class="backButton">
107         <a href="index.php">
108 
109                 <button type="button" class="btn btn-primary btn-sm">
110                 <li class="fa fa-arrow-left fa-4x"></li>
111             </button>
112         </a>
113     </div>
114 
115     <div class="main">
116         <center>
117             <h1>Converter <?php echo $_REQUEST['num'];?></h1>
118             <div class="time"></div>
119             <br>
120             <br>
121             <div class="devices">
122                 <div class='slave'>
123                     <table style='width:80%'>
124                         <tr>
125                             <td>Power:</td>
126                             <td id="power_slave"></td>
127                         </tr>
128                         <tr>
129                             <td>Direction:</td>
130                             <td id="direction_slave"></td>
131                         </tr>
132                         <tr>
133                             <td>Charge state:</td>
134                             <td id="charge_state"></td>
135                         </tr>
136                         <tr>
137                             <td> </td>
138                         </tr>
139 
140                         <tr>
141                             <td>Voltage 1:</td>
142                             <td id="volt1_slave"></td>
143                         </tr>
144                         <tr>
145                             <td>Current 1:</td>
146                             <td id="current1_slave"></td>
147                         </tr>
148 
149                         <tr>
150                             <td> </td>
151                         </tr>
152                         <tr>
153                             <td>Voltage 2:</td>
154                             <td id="volt2_slave"></td>
155                         </tr>
156                         <tr>
157                             <td>Current 2:</td>
158                             <td id="current2_slave"></td>
159                         </tr>
160 
161                         <tr>
162                             <td> </td>
163                         </tr>
164                         <tr>
165                             <td>Output 1:</td>
166                             <td id="output1_slave"></td>
167                         </tr>
168                         <tr>
```

```
169                             <td>Output 2:</td>
170                             <td id="output2_slave"></td>
171                         </tr>
172                     </table>
173 
174                     <br>
175                     <br>
176                     <button onclick="power_on()" type="button" class="btn btn...
                            -primary slave">Turn on</button>
177                     <br>
178                     <br>
179                     <button onclick="power_off()" type="button" class="btn ...
                            btn-primary slave">Turn off</button>
180                     <br>
181                     <br>
182                     <button onclick="switch_direction()" type="button" class=...
                            "btn btn-primary slave">Switch Direction</button>
183                     <br>
184                     <br>
185                     <div id="maxvolt"></div>
186                     <br><br>
187                     <button onclick="max_volt()" type="button" class="btn btn...
                            -primary slave">Update Maximum Volt</button>
188                     <br><br>
189                     <div id="maxcurrent"></div>
190                     <br><br>
191                     <button onclick="max_current()" type="button" class="btn ...
                            btn-primary slave">Update Maximum Current</button>
192                 </div>
193 
194             </div>
195         </center>
196     </div>
197     <script>
198 
199 
200         var oldOptions = {};
201         var newOptions = {};
202         var times = [];
203         var values = [];
204         var dataArray = [];
205         var xData = [];
206         var x = 0;
207         var which = 0;
208         var oldtime;
209         var num = "<?php echo $_REQUEST['num'];?>";
210         var direction = -1;
211         var power = -1;
212         var current1 = -1;
213         var current2 = -1;
214         var volt1 = -1;
215         var volt2 = -1;
216         var output1 = -1;
217         var output2 = -1;
218         var charge_state = "";
219 
220 $(".slave").css("width", "90%");
221                 $(".btn").css("font-size", "200%");
222                 $(".slave").css("display", "inline-block");
223         function refresh() {
224             $.ajax({
225                 url: "lateststatus.php"
226             }).done(function(data) {
```

```
227                 var arr = data.split('////');
228                 if (arr[0] == oldtime) {
229                     console.log("Al gehad");
230                     return;
231                 }
232                 oldtime = arr[0];
233                 $(".time").html(arr[0]);
234                 times.push(arr[0]);
235                 var obj = JSON.parse(arr[1]);
236                 values.push(obj);
237
238                 var jsonLength = obj["slaves"].length;
239                 var tempHTML = "";
240                 for (var i = 0; i < jsonLength; i++) {
241                     //console.log(obj["data"][i]);
242                     newOptions[obj["slaves"][i]] = obj["slaves"][i];
243
244                     if (obj["slaves"][i] == num) {
245                         power = obj["data"][i][0];
246                         direction = obj["data"][i][1];
247                         current1 = obj["data"][i][2];
248                         current2 = obj["data"][i][3];
249                         volt1 = obj["data"][i][4];
250                         volt2 = obj["data"][i][5];
251
252                         output1 = obj["data"][i][6];
253                         output2 = obj["data"][i][7];
254
255                         switch(obj["data"][i][8]){
256                             case 0:
257                                 charge_state = "Reset";
258                             break;
259                             case 1:
260                                 charge_state = "Ramp Voltage";
261                             break;
262                             case 2:
263                                 charge_state = "Nominal";
264                             break;
265                             case 3:
266                                 charge_state = "Ready";
267                             break;
268                             case 4:
269                                 charge_state = "Pinch off";
270                             break;
271                             case 5:
272                                 charge_state = "Done";
273                             break;
274
275                             default:
276                                 charge_state = "";
277                             break;
278                         }
279
280                         $("#power_slave").html(power);
281                         $("#direction_slave").html(direction);
282                         $("#charge_state").html(charge_state);
283
284                         $("#volt1_slave").html(Math.round((volt1 + 1) / 1024 ...
                                * 1000 * 3.3) / 1000 + "V");
285                         $("#current1_slave").html(Math.round((current1 + 1) /...
                                 1024 * 1000 * 3.3) / 1000 + "V");
286                         $("#volt2_slave").html(Math.round((volt2 + 1) / 1024 ...
                                * 1000 * 3.3) / 1000 + "V");
```

```
287                         $("#current2_slave").html(Math.round((current2 + 1) /...
                                1024 * 1000 * 3.3) / 1000 + "V");
288
289                         $("#output1_slave").html(Math.round((output1 + 1) / ...
                                1024 * 1000 * 3.3) / 1000 + "V - " + output1);
290                         $("#output2_slave").html(Math.round((output2 + 1) / ...
                                1024 * 1000 * 3.3) / 1000 + "V - " + output2);
291                     }
292
293
294
295                 }
296
297
298             });
299         }
300         setInterval(refresh, 500);
301     </script>
302 </body>
303
304 </html>
```

Listing C.3: config.php

```
1  <?php
2     define('DB_SERVER', 'localhost');
3     define('DB_USERNAME', 'bap2017');
4     define('DB_PASSWORD', 'zA23w!4f');
5     define('DB_DATABASE', 'jkoeners_bap2017');
6     $salt = "bapb";
7     $mysqli = new mysqli(DB_SERVER,DB_USERNAME,DB_PASSWORD,DB_DATABASE);
8     /* check connection */
9  if ($mysqli->connect_errno) {
10     printf("Connect failed: %s\n", $mysqli->connect_error);
11     exit();
12 }
13 ?>
```

Listing C.4: instructions.php

```
1  <?php
2  include "config.php";
3
4  $query  = "SELECT * FROM device_received WHERE ip='" . $_SERVER['REMOTE_ADDR'...
       ] . "'";
5  $result = $mysqli->query($query);
6
7  if ($result->num_rows > 0) {
8      echo "{}";
9  } else {
10     $query  = "SELECT * FROM instructions ORDER BY id DESC LIMIT 1";
11     $result = $mysqli->query($query);
12
13     if ($result->num_rows == 1) {
14         $row = $result->fetch_assoc();
15         echo $row['instruct_text'];
16     } else {
17         echo "{}";
18     }
19
```

```
20      $query = "INSERT INTO device_received (ip) VALUES ('" . $_SERVER['...
            REMOTE_ADDR'] . "')";
21
22
23      if ($mysqli->query($query)) {
24
25      } else {
26      }
27  }
28  ?>
```

Listing C.5: receive.php

```
1   <?php
2   include "config.php";
3
4   if(isset($_POST['status'])) {
5       $json_data = $_POST['status'];
6       // Doe iets met de data (beveiligen)
7
8       if($mysqli->query("INSERT INTO status (json_data) VALUES ('".$json_data."...
            ')")){
9           die("Nieuwe status toegevoegd");
10      } else {
11                  echo "INSERT INTO status (json_data) VALUES ('".$json_data."...
                        ')";
12          die("Er ging iets fout bij status invoegen");
13      }
14  }
15  else {
16     die('no post data to process');
17  }
18
19  ?>
```

Listing C.6: lateststatus.php

```
1   <?php
2   include "config.php";
3
4   $query = "SELECT * FROM status ORDER BY index_number DESC LIMIT 1";
5   $result = $mysqli->query($query);
6
7   if($result->num_rows == 1){
8       $row = $result->fetch_assoc();
9       echo $row["time_stamp"];
10      echo '////';
11      echo $row['json_data'];
12  }
13  ?>
```

Listing C.7: new_instruction.php

```
1   <?php
2   include "config.php";
3   if (isset($_REQUEST['slave_addr']) && isset($_REQUEST['instruction_type']) &&...
        isset($_REQUEST['data'])) {
4       $arr           = array(
5           'slave' => $_REQUEST['slave_addr'],
```

```
6           'instruct_type' => $_REQUEST['instruction_type'],
7           'data' => $_REQUEST['data']
8       );
9       $json_encoded = json_encode($arr);
10
11      $query = "INSERT INTO instructions (instruct_text) VALUES ('" . ...
            $json_encoded . "')";
12      if ($mysqli->query($query)) {
13          $query = "TRUNCATE device_received";
14          $mysqli->query($query);
15      } else {
16          die("Er ging iets fout bij instructie invoegen");
17      }
18  } else {
19      echo "Niet een goede instructie.";
20  }
21  ?>
```

Listing C.8: style.css

```
1   body,
2   html {
3       background-color: rgba(220, 220, 220, 1.00);
4       height: 100%;
5       width: 100%;
6   }
7   .main {
8       margin: auto;
9       margin-top: 100px;
10      font-size: 200%;
11  }
12  h1,
13  .btn {
14      font-size: 200%;
15  }
16  .overlay {
17      width: 95%;
18      height: 95%;
19      position: absolute;
20      margin-left: auto;
21      margin-right: auto;
22      left: 0;
23      right: 0;
24      z-index: 100;
25      display: block;
26      border-radius: 10px;
27      border-style: solid;
28      border-color: rgba(54, 36, 36, 1.00);
29  }
30  .backButton {
31      position: absolute;
32      top: 20px;
33      left: 20px;
34  }
35  input[type range] {
36      -webkit-appearance: none;
37      width: 100%;
38      margin: 24px 0;
39  }
40  input[type range]: focus {
41      outline: none;
42  }
```

```
43 input[type range]::-webkit-slider-runnable-track {
44     width: 100%;
45     height: 2px;
46     cursor: pointer;
47     box-shadow: 1px 1px 1px #000000, 0px 0px 1px # 0 d0d0d;
48     background: #3071a9;
49     border-radius: 1.3px;
50     border: 0.2px solid # 010101;
51 }
52 input[type range]::-webkit-slider-thumb {
53     box-shadow: 1px 1px 1px #000000, 0px 0px 1px # 0 d0d0d;
54     border: 1px solid #000000;
55     height: 50px;
56     width: 20px;
57     border-radius: 3px;
58     background: # ffffff;
59     cursor: pointer;
60    -webkit-appearance: none;
61     margin-top: -24.2px;
62 }
63 input[type range]: focus::-webkit-slider-runnable-track {
64     background: #367ebd;
65 }
66 input[type range]::-moz-range-track {
67     width: 100%;
68     height: 2px;
69     cursor: pointer;
70     box-shadow: 1px 1px 1px #000000, 0px 0px 1px # 0 d0d0d;
71     background: #3071a9;
72     border-radius: 1.3px;
73     border: 0.2px solid # 010101;
74 }
75 input[type range]::-moz-range-thumb {
76     box-shadow: 1px 1px 1px #000000, 0px 0px 1px # 0 d0d0d;
77     border: 1px solid #000000;
78     height: 50px;
79     width: 20px;
80     border-radius: 3px;
81     background: # ffffff;
82     cursor: pointer;
83 }
84 input[type range]::-ms-track {
85     width: 100%;
86     height: 2px;
87     cursor: pointer;
88     background: transparent;
89     border-color: transparent;
90     color: transparent;
91 }
92 input[type range]::-ms-fill-lower {
93     background: #2a6495;
94     border: 0.2px solid # 010101;
95     border-radius: 2.6px;
96     box-shadow: 1px 1px 1px #000000, 0px 0px 1px # 0 d0d0d;
97 }
98 input[type range]::-ms-fill-upper {
99     background: #3071a9;
100    border: 0.2px solid # 010101;
101    border-radius: 2.6px;
102    box-shadow: 1px 1px 1px #000000, 0px 0px 1px # 0 d0d0d;
103 }
104 input[type range]::-ms-thumb {
105    box-shadow: 1px 1px 1px #000000, 0px 0px 1px # 0 d0d0d;
```

```
106     border: 1px solid #000000;
107     height: 50px;
108     width: 20px;
109     border-radius: 3px;
110     background: # ffffff;
111     cursor: pointer;
112     height: 2px;
113 }
114 input[type range]: focus::-ms-fill-lower {
115     background: #3071a9;
116 }
117 input[type range]: focus::-ms-fill-upper {
118     background: #367ebd;
119 }
```