Hardware Hub

Energy System Integration Demonstrator

R.A. Siemensma
P.J. Rozestraten

June 19, 2020

TU Delft

Hardware Hub

Energy System Integration Demonstrator

by

R.A. Siemensma
P.J. Rozestraten

to obtain the degree of Bachelor of Science

at the Delft University of Technology,

Student number:	4712358 (R.A. Siemensma)	
	4725352 (P.J. Rozestraten)	
Project duration:	April 20, 2020 – June 19, 2020	
Thesis committee:	Dr. M. Cvetkovic,	TU Delft, supervisor
	Dr. ir. A.A. van der Meer,	TU Delft, supervisor
	Prof. Dr. A.H.M. Smets,	TU Delft
	Dr. M. Taouil	TU Delft

TU Delft

Abstract

This report discusses the design and implementation of a subsystem required for an energy system integration demonstrator. The purpose of this subsystem is emulating energy systems using physical table-top model representations. This system has been realised using a Raspberry Pi 3B+, which communicates information using the I^2C protocol to the table-top models. The physical models can actuate information regarding the energy systems from the Raspberry Pi with DACs, and sense real-time information with ADCs.

keywords: hub, I^2C, energy system emulation, table-top model, Raspberry Pi 3B+, Python

Preface

This report written for the Bachelor Graduation Project of Electrical Engineering at Delft University of Technology. It details the design process of the Hardware Hub, a part of the Energy System Integration Demonstrator project. The Demonstrator is a tool which can be used by educators to emulate the power grid, and to make it more tangible. The Hardware Hub is the physical part of the demonstrator, to which you can attach physical models of energy systems, such as wind turbines and solar panels, and emulate their behaviour.

A special thanks to our supervisors Milos Cvetkovic and Arjen van der Meer for proposing this project, giving lots of freedom and providing a great development setup. We would also like to thank our other project members, Ali Ahmad Sohrab Ashraf, Victor van Doorn, Benjamin Visser, and Daniel van Paassen, for their great cooperation and a nice time together.

Delft, June 2020
Ruben Siemensma
Jean-Paul Rozestraten

Contents

1 Introduction

1.1. Problem Description

The knowledge of how a city or neighbourhood could operate on sustainable sources, is often not known to the common people. Nonetheless it is an important global transition that must be achieved. The energy demand in the world is continuously rising and because of the direction it going in now due to climate change, something must change. Renewable implementations for energy are very popular already [1], and increasing still which means the market will expand. In order to positively stimulate the use of renewable sources, and tackle the complexity the integration of these sources demands, a way of explaining in simple terms is required. Energy grids are always gaining components, for example electric vehicles (EVs) have starting getting popular and are appearing more commonly since the 20th century.

This additional component creates more energy distribution options within a grid and demands extra integration. Such developments require investments and primarily more apprehension. However, demanding the common people to inform themselves about renewable sources independently, as well as an integrated solution would be quite the challenge. A simplified, but accessible and easy-to-use visualizing tool is necessary to cross the gap of knowledge about integrated neighbourhoods powered by renewable sources. A tool like this will try to present a neighbourhood or other cases in such a fashion that little to no prior understanding of engineering is required. This tool visualizes all essential characteristics that are needed to understand and motivate people to innovate in the energy market.
The creation of such a tool will consist of multiple components presented visually in Figure 1.1. Different models must be implemented and be able to work together to simulate the powers

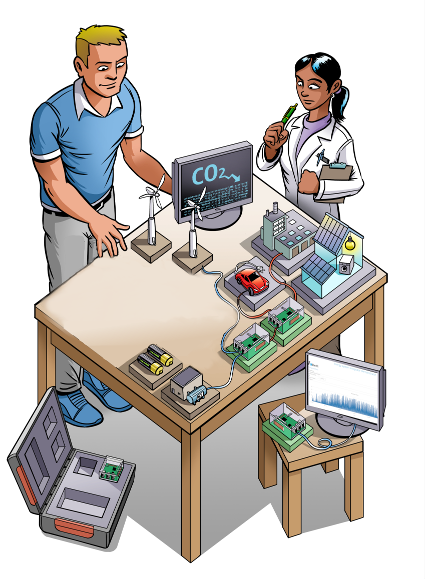

Figure 1.1: The energy integration Demonstrator in clip art image.

in different circumstances and show what influences the renewable sources, and how to operate independently on those.

1.2. State of the art analysis

The transition into renewable sources is an active field of research. Being one sustainable development goal from UNESCO, the development of educating about green energy has been encouraged greatly. For example, the ”Changing with the climate network” project, which contains several school activities for educating about the renewable energy use [2]. In the discussion about the conferences of the Engineering Education for Sustainable Development (EESD), it becomes clear that now there is a need for participatory methods, and interaction, with sustainable energy technologies [3]. Many developed demonstrating tools around sustainable energies are mostly about market-analysis, prototype analysis, or policy making and how the public would adopt the renewable technologies [4]. All of these categories are not aimed at the public, but the parties already involved in renewable technologies. Only a few tools are made to offer insight about these technologies to the public with a demonstrating tool. ESDL, the Energy System Description Language project [5], has created such a model (called ESSIM) that offers insight about how a system with renewable sources works. This tool has enough customizability of the system. For example, storage options can be added. Also, it will provide a lot of information about the system as a whole; during which periods there is an excess or shortage of energy; and the CO2 emission of the system. This tool however, is completely online, and some interactive part is lost due to this. There is no interaction possible with physical table-top models and the visualization is only digital. Another model is the System Advisor Model (SAM) [6]. This computer model calculates performance and financial metrics of renewable energy systems. This model is very accurate and powerful, but is made for policymakers, researchers, and project developers. These are audiences that have some pre-existing knowledge about the workings of a renewable system. This makes the tool unsuitable as an educational tool for the common people.

1.3. Demonstrator Overview

The visualisation and demonstration of power flows throughout a grid is done using a Demonstrator kit. In Figure 1.2 an overview of the Demonstrator is shown. The Demonstrator consists of Raspberry Pis, depicted in the colour raspberry. Each client Pi can be connected to a ”Hardware Hub” to which table-top models are connected visualized by the images. The server Pi however is connected to all client Pis and this is also where the visualization, called ”Dashboard”, is located using a monitor. This Demonstrator is able to show power production and consumption and the influence of changing inputs on these powers. This visualization is not only done by looking at a monitor but also has a more touchy side with the addition of a Hardware Hub. This is a component which will resemble what is going on in the grid by means of actuators and sensors. The Demonstrator will then consist of 3 sub modules. Below is a small elaboration on each sub module existing in the Demonstrator kit.

1.3.1. Dashboard

The Dashboard is where the Demonstrator shows the simulated neighbourhood outputs. The Dashboard is also the place where a user can change the neighbourhood configuration. The Dashboard depicts several power flows over the course of a single year (2019), but is also able to zoom to an arbitrary day or month. This provides the user with more specific performance data for a shorter time line. Furthermore the Dashboard uses all this received power data to make an emission analysis and a financial analysis. From these rigorous analyses the user can make decisions regarding their neigh-

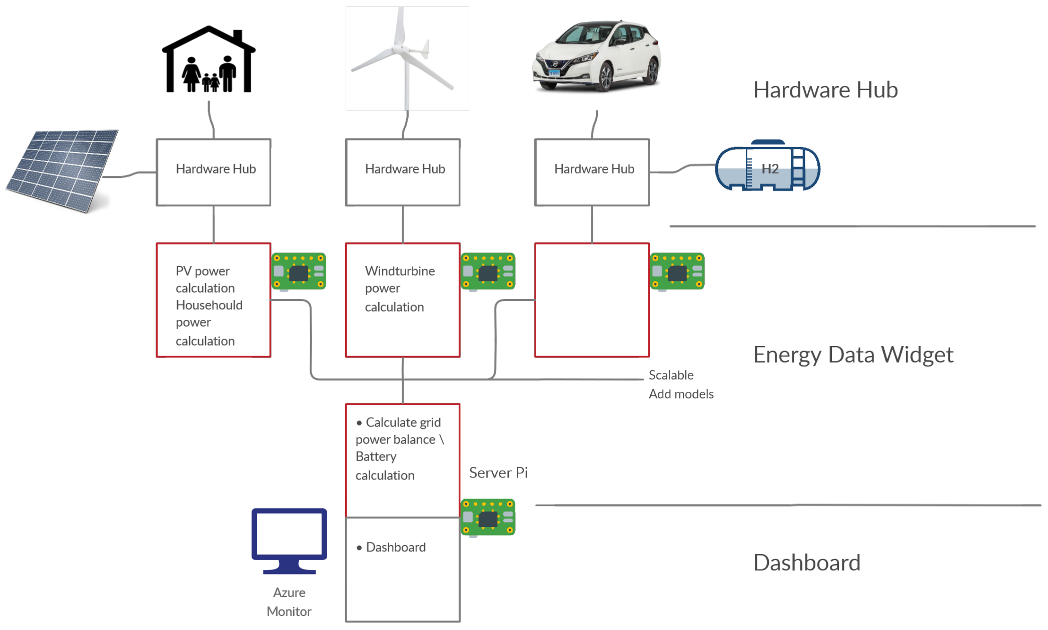
Hardware Hub
Hardware Hub
Hardware Hub
Hardware Hub
H2
PV power calculation Househould power calculation
Windturbine power calculation
Energy Data Widget
Scalable
Add models
• Calculate grid power balance \ Battery calculation
Server Pi
• Dashboard
Azure Monitor
Dashboard

Figure 1.2: Schematic overview of the Demonstrator

bourhood configuration. For example adding more houses, down-scaling the amount of wind turbines or even completely disconnecting all hydrogen tanks.

1.3.2. Energy Data Widget

The Energy Data Widget is going to be the backbone of the Demonstrator. It will calculate all values regarding powers and energies by means of Python. This software is implemented on Raspberry Pis and is able to handle 2 types of inputs, coming from the Dashboard or from table-top models. When using Dashboard as main input all relevant weather and consumption patterns are retrieved through an online database as well as types and amounts of components within your grid.
Besides the Dashboard the table-top models are also able to create inputs. However, they just output actual weather data created by sensors and can not change the type. The models should be able to operate in both modes.

1.3.3. Hardware Hub

The Hardware Hub is a physical setup that is able to emulate energy systems by using table-top models. If there is a lot of wind power being generated, the table-top wind model is able to actuate this by rotating a DC-motor, or if houses are using a lot of energy a lamp will shine brighter. The Hardware Hub allows users to connect and interchange table-top models freely and see how new energy systems influence other energy systems. It is also possible to interact and play with these table-top models, it is for instance possible to change loads by rotating potentiometers, add or remove electric vehicles by using switches, and influence generated wind power by rotating a DC-motor.

1.4. Structure of Thesis

In this Thesis the Hardware Hub will be discussed. This Thesis starts with the general requirements of the entire system, and the requirements of the Hardware Hub solely.

In Chapter 3 a general overview is given of what the Hardware Hub is, and what components it needs to operate and function. Also it discusses the power supply and how it is integrated in the hardware bus.

In Chapter 4 the communication design between the Raspberry Pi and the table-top models is discussed. This Chapter elaborates on some design choices and how these choices have been implemented in the actual design. This Chapter discusses how I^2C has been implemented and then continues to the testing of I^2C, and ends with a discussion and conclusion.

Chapter 6 discusses all the models that are designed and can be connected to the Hardware Hub. This is done by first starting with a general description of all the models, since many designs that are discussed in the general description are reused in the design for different models. After the general Sections the specific models are explained. Some of these models like the solar panel and the wind turbine are extensive enough to require their own testing, results, and discussion Section.

The thesis ends with a conclusion and an exploration of possible future work.

2 Program of Requirements

2.1. Requirements for the entire system

This Chapter will describe the requirements which the Demonstrator must have and which it would preferably have. Below, the requirements are split up for the complete system and the sub module. These requirements are then divided up into mandatory requirements, which are necessary to function, and trade-off requirements, which enrich the product.

Mandatory requirements

These are the requirements the system should comply with in order for all sub modules to work together correctly and to have a fully functioning Demonstrator.

1. The Demonstrator must be able to calculate the power flows from real-time simulated weather data coming from the hardware hub (table-top models).
2. The Demonstrator must show all power flows going into and out of the system and its individual components (for example a solar panel or wind turbine).
3. The product has to be able to interact with the designed table-top models, a communication needs to be established.
4. Table-top models must be able to visualize information regarding power and energy of the specific model.
5. The table-top models must be interactive so that the user can physically influence the individual components such as the wind turbine or solar panel.
6. The Dashboard must be interactive as well, so that the parameters can be changed (such as the amount of EVs or solar panels), and this leads to a change in output.

Trade-off requirements

These are the qualities or attributes which the product must have, such as performance, security, usability, maintainability, etc.

1. The resolution of the power flows from the table-top models should have an hourly resolution.

2. The dashboard should be easily operable, with only a keyboard and mouse.
3. The system should be easy to understand for laymen.
4. The system should be built with low cost, no more than €100.
5. The performance should not be too slow, preferably within seconds for a new simulation.
6. The system should not be too large to make it easy to carry in a briefcase for example.
7. The product should preferable also handle extreme cases, such as a hack on the wind turbine, disabling all power flow from that component within the system.

2.2. Requirements for the Hardware Hub

These are the requirements for the Hardware Hub solely.

Mandatory requirements

These are the requirements that have to be fulfilled by the Hardware Hub, in order to have it functional and working.

1. The table-top models are interchangeable on the Hardware Hub.
2. The table-top models are able to sense and or actuate information from and to a Raspberry Pi.
3. The table-top models are interactable, so the user can physically influence the individual table-top models.
4. The Hardware Hub is able to function using only the I/O ports of one Raspberry Pi.
5. The Hardware Hub can be build using only existing integrated circuits.
6. The Hardware Hub's power supply must provide power to all the table-top models.

Trade off requirements

1. The system is as modular as possible, meaning it can be easily expanded into a bigger system.
2. The table-top models can be connected to the Hardware Hub using as little as possible connections.
3. The Hardware Hub gives future users enough tools to keep expanding and improving the Hardware Hub.

3 Hardware Hub Overview

The main design feature when designing the Hardware Hub was to create system that is as modular as possible, where as many as 32 energy systems can be emulated on one Raspberry Pi. This feature has been realised using three main components: The Raspberry Pi, the Hardware Bus, and the table-top models. The overview of the Hardware Hub can be seen in Figure 3.1.

The Raspberry Pi is the place where all the information, regarding power flow and energy, is calculated. This information is presented to the Hardware Hub using only two I/O pins present on the Raspberry Pi. The Raspberry Pi is also able to receive information coming from the Hardware Hub using the same I/O pins. These two pins are the I^2C pins. The Hardware bus is practically a large bus which allows the table-top models to be connected to the communication bus. The Hardware bus is also responsible for powering all the models and provide the three different supply voltages (excluding ground) required by the models.

The table-top models are the components where all the conversions take place. Here I^2C signal is translated to an analog signal and the analog signals are translated back to I^2C. The table-top models are also the place where the energy system models are actuating and sensing information. The table-top models can be connected to the Hardware Hub bus freely, and up to 32 different table-top models can be connected in total. However, each Hardware Bus PCB will only have a limited number of places where models can be attached, but since there are connections to the top and bottom of each Hardware Bus it is possible to connect multiple Hardware busses together to create one large bus. This expandable system has been made because for most Hardware Hub applications only 4 to 10 table-top models will be connected, and it would be a waste of hardware to leave 28 ports unused. This design also gives the power supply some freedom, since it is possible to connect multiple power sources. It is important to keep in mind not to connect the power bus together if multiple power sources are used.

3.1. Power Supply

The models require power to actuate signals and for their conversions. The Raspberry Pi has a 3.3 V pin and a 5 V pin to power electronic devices attached to the GPIO pins. However these cannot provide enough power to all models. The 5 V output is connected to USB 5 V rail, and can supply a maximum of 1.2 A over that rail. 6.0 W is not enough to power more than 2 models and is thus not sufficient. The Raspberry Pi also has no proper current limiting capabilities, so if too much power is drawn, the

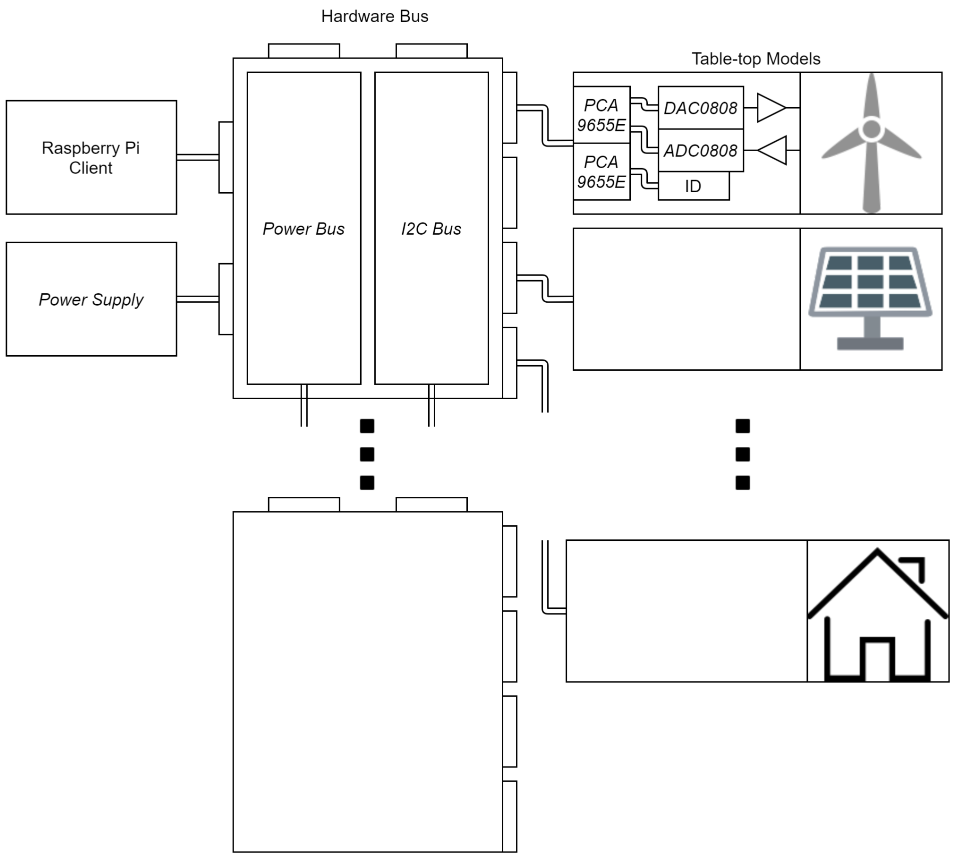

Figure 3.1: The Hardware Hub overview.

processor and other components of the Pi will not get enough power and might malfunction. [7]
The Raspberry Pi is also not able to create 12V and -12V voltage sources. Therefore, a separate power supply is necessary for these.

First, two models are designed to determine the requirements of the power supply. These are the solar cell model from Section 6.3 and the wind turbine model from Section 6.4. The power drawn by these models and the required voltage levels will be assumed to be typical for all models. This is then extrapolated to a maximum of 32 models per Hardware Hub.

From Table 3.1 the maximum power consumption for a solar PV model is 2.86 W and for the wind turbine model this is 4.86 W. The average component will then use around 3.86W meaning that for 32 models a power supply of 123.52 W is necessary. But since the Hardware Hub design allows for

Component	Maximum power consumption	Amount	Total maximum power
PCA9655E [8]	0.6 W	2	1.2 W
ADC0808 [9]	0.015 W	1	0.015 W
DAC0808 [10]	0.305 W	1	0.305 W
μA741 [11]	0.085 W	4	0.34 W
3V DC-motor	3 W	1	3 W
5V lamp	1 W	1	1 W

Table 3.1: Power consumption per component

multiple power supplies, and generally a user will not use all 32 table-top models, a pre-made power supply has been chosen [12] that is able to deliver 64.6 W at the required voltages of 12 V, -12 V, and 5 V. This makes it capable of powering 16 models.

Another implementation which has been considered was using a USB-C based power supply. The main advantage is that a USB-C is a universally accepted connection, the main drawbacks are however complexity and some of the components required for this power supply are quite expensive. Further explanation of a possible implementation of USB-C will be discussed later in this thesis in Section 7.1.2.

4 Communication

In the Hardware Hub design it is required that multiple different table-top models can be connected to the hub freely and that these table-top models are interchangeable. The Raspberry Pi is also required to be able to send information regarding power and energy to the correct models, without being influenced by the topology in which these models are connected to the Hardware bus. The models are also able to sense information, this information needs to be sent back to the Raspberry Pi and also tell the Pi from which model this information is coming from. To solve these challenges the I^{2}C communication protocol has been implemented in the design. With this protocol it is possible for the Raspberry Pi to communicate with a large number of devices with just two communication lines to be connected to the Pi.

4.1. Communication Protocol

In this design the I^{2}C protocol has been used. I^{2}C uses a master slave relation, so here the master decides when to read or write from a specific slave. The components of a system using I^{2}C generally consist of a master controller and multiple slaves. The master and slaves are all connected using the same bus, this bus consists of only two lines. This bus uses a data signal (SDA) and a clock signal (SCL). These communication lines are pulled up by default, so when information needs to be transferred these lines are pulled down by the masters and slaves.
The communication starts when master issues the start condition by pulling both the SDA and the SCL down. The master can then choose which slave it wants to communicate with by sending 7 address bits together with a read or write bit. Only if a slave has a corresponding address it can interact with the bus. If a slave has a corresponding address it will send an acknowledge. Now the master is able to receive or send data from and to the slaves, depending on if it send a read or write bit.[13][14]

4.1.1. Different Communication Protocols

For this design multiple communication protocols have been considered. These were SPI, UART and the one that is used in this design I^{2}C.
SPI is a protocol that requires at least four connections, these connections are: master output slave input (MOSI), master input slave output (MISO), serial clock (SCLK) and a slave select (SS). The main downside to this protocol is that it requires a SS line for each new slave. Since the design of

the hardware hub requires a substantial amount of slaves using this protocol would have been sub-optimal. To implement this protocol it would require multiple hardware and software solutions to reduce the amount of lines.

UART is a 1 to 1 communication protocol, it requires two wires (TX and RX) and it is a simple and well known protocol. The main drawback of this protocol is that it can only communicate with one other device, so to implement this in the current hardware-hub design would require a system that distributes information correctly to multiple devices. This is something I^{2}C already does by default.

I^{2}C only needs two lines for a large amount of slaves, making I^{2}C an adequate communication protocol choice for the design. [13] [14]

4.2. I^{2}C Implementation

Figure 4.1 shows how the I^{2}C communication has been implemented on the table-top models. The lines going into the "model" block represent the wires coming from / going to the model side, so the converters and the actuators and sensors. And the lines going into the I^{2}C bus are the communications lines SDA and SCL and this bus is then directly connected to the Raspberry Pi. To convert the I^{2}C signal into 16 parallel bits and the other way around the PCA9655E has been used. The PCA9655E has 16 different digital inputs/outputs and these are divided into two subsets consisting of the pins [IO0$_0$ - IO0$_7$] and the pins [IO1$_0$ - IO1$_7$]. These 16 pins are able to read and set information depending on what commands the PCA9655E IC receives from the Raspberry Pi.

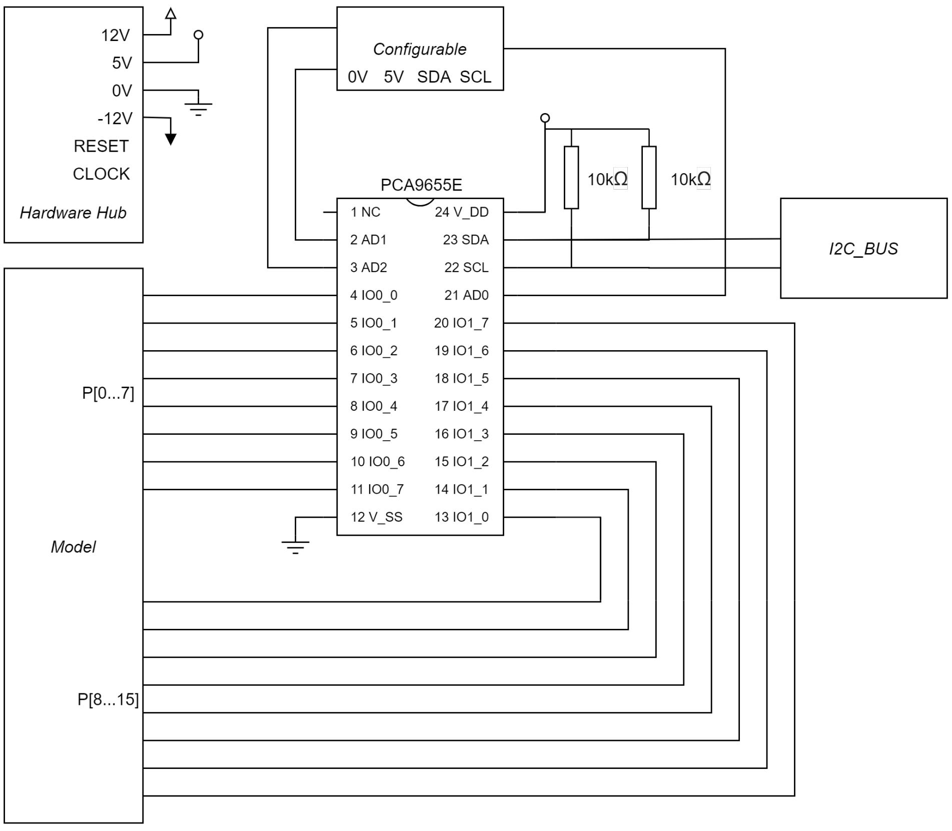

Figure 4.1: The implementation of the I2C converter

4.2.1. Model ID

In the general overview of the Hardware Hub in Section 3.1 can be seen that each model has two PCA9655E chips and an ID connected to one of these PCA9655E chips. This ID can be changed using switches and this information gets requested by the Raspberry Pi using I^{2}C. This enables the Pi to determine which energy system the each model is representing, and to which I^{2}C address each model corresponds. This ID uses 8-bits, which allows for 256 different table-top models. 8-bits seems like to much for 32 table-top models, but this gives the freedom to implement different model versions. So the user is now able to choose for instance from a high power solar PV, medium power solar PV or a low power solar PV. And this complies with the requirement of creating a as modular as possible design.

4.2.2. PCA9655E

The PCA9655E [8] is an IC that is able to translate I^{2}C to 16 parallel bits, divided into 2 sets of 8 output pins. Internally each I/O pin has four registers: An input register, an output register, a polarity inversion register and a configuration register. When the master communicates with one of its slaves, it sends a command byte. The command byte tells which sets of registers it is supposed to read/write from, this register can be different than from what is presented to the I/O ports. The different commands can be seen in Table 4.1.

Command	Register
0	Input Port 0
1	Input Port 1
2	Output port 0
3	Output port 1
4	Polarity Inversion port 0
5	Polarity Inversion port 1
6	Configuration Port 0
7	Configuration Port 1

Table 4.1: Command Byte Options [8]

Before it is possible to utilise this device it is important to first configure the I/O pins using the Configuration Port commands. By default this is set to 1, which means the device is only able to read from those pins. To be able to output information from the I/O pins it is required to set the Configuration Registers to 0. What this essentially does is connecting the right registers to the I/O ports of the device.

For writing to the device (Figure 4.2), the master sends 7 address bits with the read/write bit set to zero, meaning the master will write. After an acknowledge from the slave the master is required to send a command byte telling to which register it needs to write to. After that the master is able send the data to the chosen register.

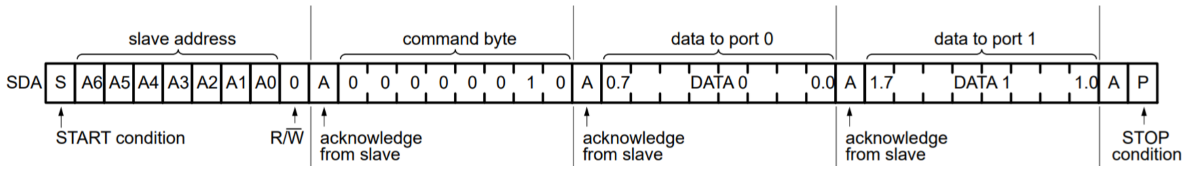

Figure 4.2: Writing to the PCA9655E (Figure from [8])

For reading it is required to send two sets of information (Figure 4.3). In the first set the master sends the address and a command byte telling from what register the master is supposed to read. For the master to be able to send commands the write/read bit needs to be set to write, so first the master writes to the device telling what register it is going to read on later. Then after sending the command byte the master only has to send the address together with a read bit, the slave will then send its information to the master.

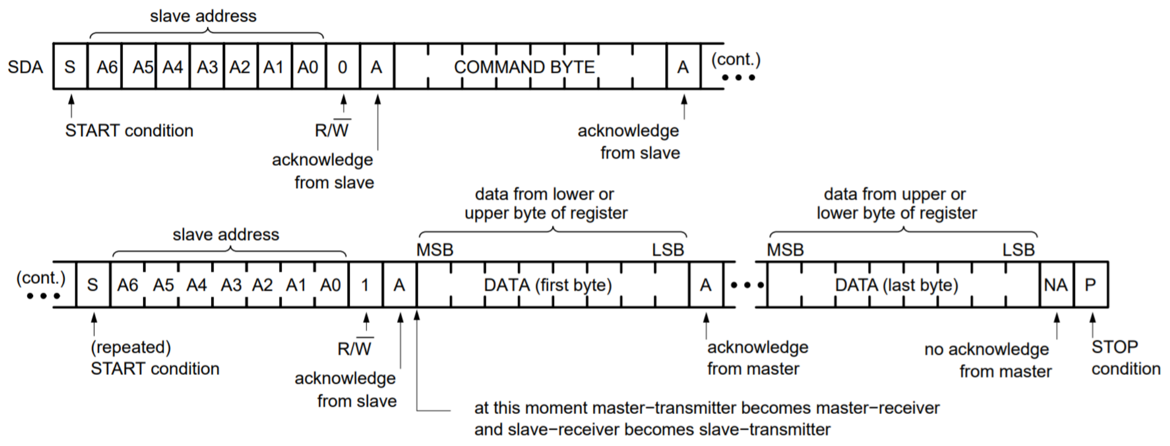

Figure 4.3: Reading from the PCA9655E (Figure from [8])

The maximum speed which the data can be read/written at depends on the SCL (clock) frequency. For this device the speed is set to 100kHz, since this is also the SCL frequency the Pi normally operates on. It is possible to increase this frequency, but for the current application 100kHz is more than enough, so the clock speed does not need to be adjusted. The sample frequency is dependent on how often the actuate function and sense function are called by the Energy Data Widget 1.3.2.

The PCA9655E is able to operate properly with 63 other slaves, for this to be possible the IC requires 64 different ways to configure its address. The PCA9655E has three configurable address pins [AD0 - AD2], it is possible to change the address by connecting the address pins in different ways to V_{DD}, ground, SDA and SCL. Each combination maps to a different address, the list of configurable addresses can be found in the datasheet [8].

4.3. Verification

As a result of the universities precaution measures regarding COVID-19, the setup with the PCA9655E couldn't be tested. Instead of the PCA9566E, three PCF8574 ICs [15] were used to get a general feel for how the Raspberry Pis would work and interact with a general I^{2}C setup. This allowed for some testing using the command line in the Raspberry Pi. The goals of these tests were to confirm if the ICs are able to read and write data correctly and if it is possible to see all the devices connected to the I^{2}C bus.

4.3.1. Setup

For this setup one Raspberry Pi 3B+ is required and two or more PCF8574 ICs, the PCF8574 with IO expander that is used in this test setup can be seen in Figure 4.4. It has four male ports to the left of the IO expander and four female ports to the right (as seen in the figure). This allows these components to be connected together easily and creating one bus. The IO expander also has some jumper caps making it easy to change the address of the IC. There are also the 8 input/output pins plus an interrupt.

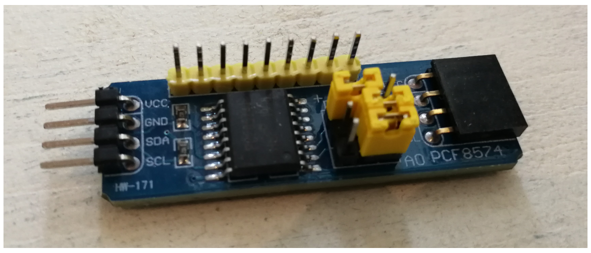

Figure 4.4: The PCF8574 IO expander

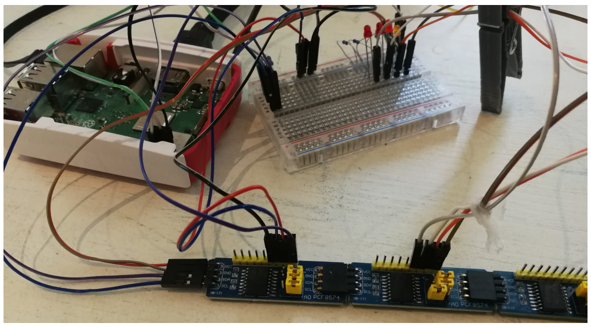

Figure 4.5: The I^{2}C test setup

The test setup consists of three of these IO expanders connected together. Each PCF8574 has been assigned a different address. One of the ends of the expander is connected to the Raspberry Pi. This creates an I^{2}C setup with one master and three slaves. The test setup can be seen in Figure 4.5. To make it possible to communicate with devices using I^{2}C it is required that the Pi's have I^{2}C communication enabled and that i2c-tools is installed. i2c-tools is a program that gives some basic commands and tools regarding I^{2}C.

The first test was to see if it was possible to see all the ICs connected to the device. This can be done by entering the command ”i2cdetect -y 1”.

The second test was to confirm if the IC was able to translate information from the Pi to its I/O ports, this was done by connecting LED's in series with a resistor to ground. This way if information gets passed to the I/O ports correctly the LED's should light up. This can be done by using the command ”i2cset -y 1 <DEVICE ADDRESS> <ADDRESS> <VALUE>”. It is important to know that the value entered is read by the Pi as a hexadecimal entry. The <DEVICE ADDRESS> had to be set to the IC its base address (this is internally decided and can't be changed) + the address set by the jumper-caps. <ADDRESS> is not used for this device and can be any integer, for the PCA9566E however it is used for commands. Generally <ADDRESS> is used for i2c multiplexers.

The third test was done by connecting wires to the I/O pins and directly connecting these to the ground or the 5V line. This would then represent a binary number. This value could then be retrieved by using ”I2cget -y 1 <DEVICE ADDRESS>”.

4.3.2. Results

The ICs were all detected on the I^{2}C bus, and the addresses visible were corresponding to the addresses assigned to the ICs. The information set to the I/O pins correctly, but the LED's were barely visible since the output voltage of these I/O pins is just 0.5V.

When the I/O pins were directly connected to a ground/5 V line, the information retrieval worked also. When a user tries to write to the IO pins while the IO pins are wired to the ground/5 V the write will not work, because it is directly overwritten by the configuration of the wires.

4.4. Discussion and conclusion

The results were good and as expected. These tests gave insight in how I^{2}C works and how to interact with it using the Raspberry Pis. This setup allowed for some basic testing and allowed for the basic tools to build the Python code around the communication. In the final product however a different IC will be used than the one used for testing (PCF8574). There are some significant differences between the PCF8574 and the PCA9655E, and these have to be accounted for. The PCA9655E has eight times as many addresses, and the I/O pins output voltages can range up to 5.5V. This means it is possible for an application to be directly connected to this device without having to amplify I/O port signals. Also, the extra addresses give more freedom, and remove the need of an extra I^2C multiplexer or a solution with a demux. The only real downside to the PCA9655E is that the communication protocols are more complex and require more intensive Python code than the PCF8574. Yet the PCA9655E will result in a cleaner complete design, since it does not require the extra hardware components like the I^2C multiplexers and demuxes. This is why the PCA9655E was chosen in the final design.

In the end the communication hardware design has been completed. The I^2C protocol is also working on the Raspberry Pi's and is able to function as master properly. The real test with the PCA9655E still have to be done.

5 Python Module

5.1. Introduction

The Hardware Hub needs a way to interface with the rest of the Demonstrator. Using I^{2}C functions in Python to directly communicate with the I^{2}C chips on the Hardware Hub is not very easy for the users if they are not familiar with the protocol. Therefore some classes and functions are made that should be easy to implement in other parts of the Demonstrator.

Besides being easy to use, the functions and classes should be fast, since at its core it should only be handling input and output. Controlling the models and handling the addresses should not significantly slow down the rest of the program of the Demonstrator.

The repository with the code is available at `https://github.com/danielvanpaass/Demonstrator` in Client/hhub.py.

5.2. Classes for the Table-Top Models

The Python module needs to provide a way for the program to control the table-top models. This is done by giving each model its own class. The classes have an associated ID, which corresponds to the hardware ID set by the switches as discussed in Chapter 4. The classes also have a method to set the address at which a model is found, and a boolean which indicates whether the model is connected.

The methods that communicate with the models over I^{2}C are made according to the configuration of the bits in Chapter 6.

To actuate a signal, the classes have an actuate() method. It accepts a power as an fraction, and scales it up to a range of 0-255, and rounds it down to a byte sized integer. The actuate() method then sends the byte over I^{2}C with the address stored in the class and the corresponding command from Section 4.2.2. The actuate() method is slightly different for the ElectricVehicle() and HydrogenStorage() class, since they actuate 16 bits.

To sense a signal, the classes also have a sense() method. For the ElectricVehicle() class this is called get_amount(), and for the HydrogenStorage() class get_soc(), since the bits are at a different location. The method collects a byte from the models and converts it to a fraction, or an amount of EVs in the case of get_amount().

The models are not required to be coupled to a set address, so the address needs to be able to be set from outside the class. That is what is set_i2c_address method is for.

In Section 4.2.2 it is explained that the configuration registers need to be set to make the I/O pins either an input or an output. The method configure() sets the configuration registers to 1 or 0, so the I/O pins can be used correctly by the actuate() and sense() methods.

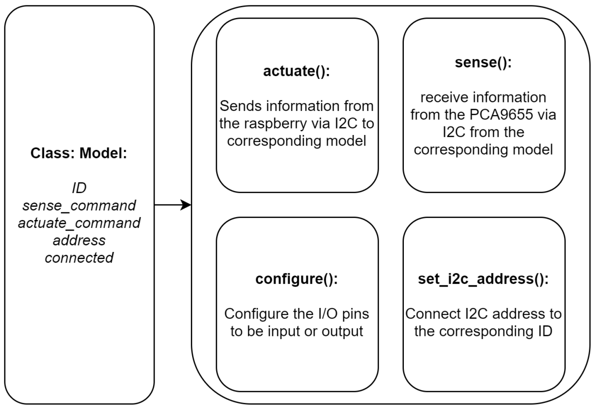
Class: Model:
ID
sense_command
actuate_command
address
connected
actuate():
Sends information from the raspberry via I2C to corresponding model
sense():
receive information from the PCA9655 via I2C from the corresponding model
configure():
Configure the I/O pins to be input or output
set_i2c_address():
Connect I2C address to the corresponding ID

Figure 5.1: Overview of the classes for the table-top models.

5.3. Startup Routine

Before the Hardware Hub can function correctly, it needs to get a reset signal on one of the GPIO pins of the Raspberry Pi. This reset signal is used by the ADC in Section 6.2.2. It also needs to send an extra clock signal (besides the SCL signal of the I^2C protocol), which is again used by the ADC. These functions are performed by calling hhub.startup(). When it is called, the GPIO pins will be set up correctly. The clock frequency has to be changed outside of the code however.

The startup() function also returns a list with objects of all the classes of the models. To access an object, the list has to be indexed with the ID of the table-top model, e.g. hhub.modelList[2] for the household, since the household is configured to have ID 2.

5.4. Update Routine

The Hardware Hub has to periodically check which models are connected. When it is time to check this, the update() has to be called. As an input it accepts a lists of objects of the model classes created by the startup() function.
It loops through all the possible I^2C addresses, and when it finds a model, it:

- sets the connected boolean of the corresponding object to True
- sets the I^2C address of the corresponding object
- configures the model by calling the configure() method

5.5. Verification

The pins and the I^2C protocol handling can not be tested, because the PCA9655E ICs are not available at the moment. However, the logic of the module can be tested by setting a global variable __USEPINS__ to False. When this variable is set to False in the file __init__.py, the module will not try to use the pins on the Raspberry Pi and instead will use dummy values and print output messages.

To test the module, a test script was written: Client/tests/test_hhub.py. In this test setup, the startup function initializes the modelList, and the update function updates the model objects.
After that the script tests two classes: SolarPV and ElectricVehicle. The other classes do not differ much from these classes and thus do not have to be tested.

```
SolarPV: Actuating power as byte 0xc0 to I2C address 0x11 with command 3
SolarPV: Sensing power as byte 0xa from I2C address 0x11 with command 0

modelList[0].sense() = 0.0390625

ElectricVehicle: Actuating state_of_charge as bytes ['0x0f', '0xff'] to I2C address 0x11
 ↪  with command 2
ElectricVehicle: Sensing amount as byte 0x69 from I2C address 0x10 with command 1

Amount of EVs = 4
```

5.6. Discussion and recommendations

The classes and function work as they are supposed to do. However, the code could be improved. The modelList contains objects which are not currently in use, although deleting and instantiating objects when models connects might increase runtime. The code also makes no use of inheritance, while all class objects are similar. This makes debugging very difficult, and creating a new class for a new model requires making that class from scratch. A base model should be added that takes minimal input to create a class with similar functionality.

6 Models

6.1. Standard Circuits

The table-top models carry the most components. Each model consists of two PCA9655Es, which expand the serial input to a parallel output. But after that, the models have a wide range of possibilities. Most need to actuate and sense, but some only need to actuate, and some others are reserved for special purposes. That is why four standard circuits have been developed, which require little configuration.

All these circuits use the same mechanism to change the addresses of the PCA9655Es. The PCA9655Es' I^2C addresses can be changed by three switches for both of the chips. Care has to be taken that the resulting address is divisible by 4 for the chip with the ID, and the other one has an address that is 2 higher. The configuration of these addresses is further explained in Chapter 4.

6.1.1. Standard Circuit 1: Actuating and Sensing

This standard circuit is for models that need to actuate a signal and need to sense something. Thus, this circuit needs both a DAC and an ADC. It will also need a way to change the ID of the device it is on, and the addresses of the PCA9655Es. Figure 6.1 shows the components and wiring of this circuit. It features two PCA9655Es, one of which is used for sensing and actuating, and the other for the ID of the model.

The general component "DAC + Actuator" converts the parallel 8 bits to a current, which is then either directly used by an actuator, or which is converted to a voltage and then used by an actuator. This part is explained more in depth in Section 6.2.1.

The general component "ADC + Sensor" has a sensor on it which outputs a current or voltage. This voltage or current is converted to a digital signal of parallel 8 bits. These bits are directly fed to the lower 8 bits of the PCA8655E. The specific ADC that is used also requires a clock and a startup signal. This is provided to the model circuit by the CLOCK and RESET signal which is coming from the Hardware Hub. This part is explained more in depth in Section 6.2.2.

The second PCA9655E has 8 switches attached to its lower 8 I/Os to control the ID of the model. The 8 switches together form an 8 bit unsigned integer. The upper 8 bits are left unconnected so they can be used in case the model needs to be expanded. The IDs need to be compliant with Section 4. Lastly,

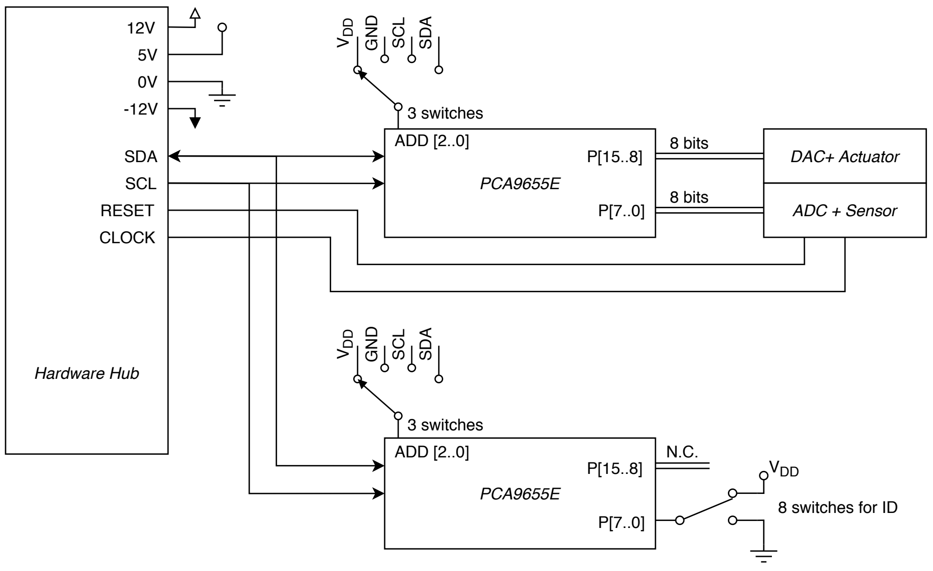

Figure 6.1: Standard Circuit 1 can actuate and sense signals via first the PCA9655E IC.

the model gets four power lines from the Hardware Hub, which can be used by any of the components if necessary.

6.1.2. Standard Circuit 2: Actuating with LEDs

This standard circuit is for models that output via a LED array. Models that use this circuit are the hydrogen battery and the electric vehicles. It is important for the power calculations on the Raspberry Pis to know how many electric vehicles are present, so this circuit needs a way to set that amount. Figure 6.2 shows the components of this circuit.

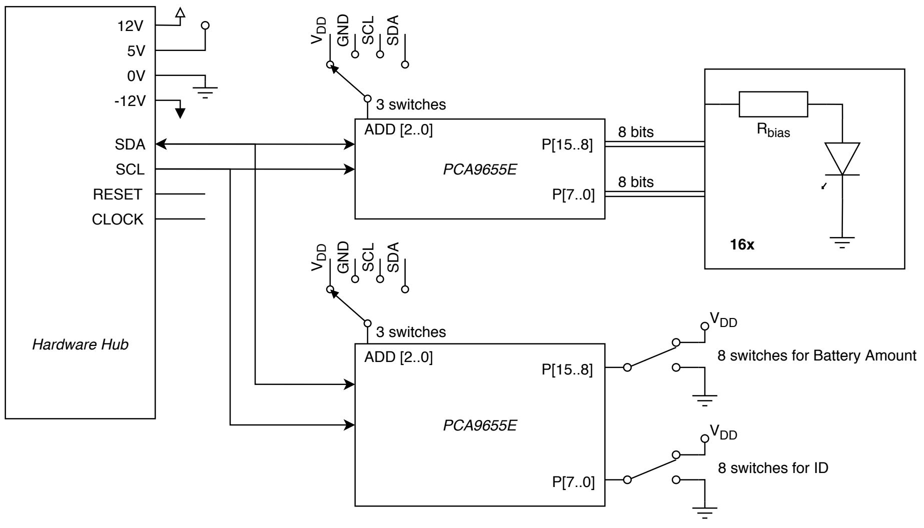

Figure 6.2: Standard Circuit 2 can display individual bits by lighting LEDs. Additionally, the amount of batteries can be changed by setting a binary number with switches.

The LEDs that are used are MCL053s [16][17][18]. They all have a luminosity of 50 mcd, and they operate at a typical forward current of 20 mA. The PCA9655E can drive these LEDs directly without the use of a transistor array to amplify the current. The PCA9655E can deliver 25 mA at its I/O pins [8], which is enough for the 20 mA of the LEDs.

To bias the LEDs at the right output, a resistor needs to be placed between the LED an the I/O port. The I/O port is expected to deliver voltage of 5 V. If you then make a large signal model of the LED, the equivalent circuit looks like Figure 6.3. From this the R$_{bias}$ is determined to be: 138 Ω for the green LED; 152 Ω for the orange LED; 158 Ω for the red LED.

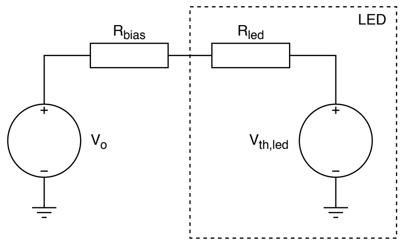

Figure 6.3: Model of a LED being driven by a 5 V voltage source.

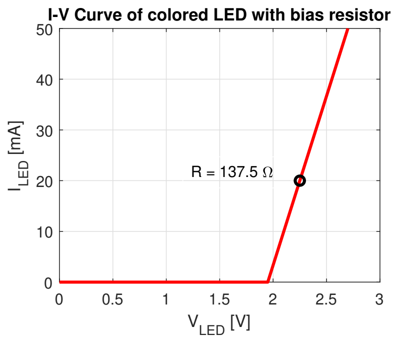

Figure 6.4: Simulated I-V curve of the green LED. The curve is modeled to look like the I-V curve in the datasheet [16]. Resulting I and V are shown for a biasing resistor of 137.5 Ω.

6.1.3. Standard Circuit 3: Custom Models

This standard circuit serves to be fully customizable. The only ICs that are on this circuit are the PCA9655Es. The I/O pins will be connected to 4 sets of 8 header pins on the circuit board.

The four power lines will also be readily available via banana plug, since these typically have a lower gauge, and thus allow more current, than jumper wires.

6.2. General Components

Several models use similar ways to fulfill their requirements. For example, the wind turbine and the solar panel models both use the same DAC and ADC for actuating and sensing. This Section describes these common components. These components will be referred to in the design of the individual models if they are used.

6.2.1. Actuator and Digital-Analog Conversion

The task of the actuator is to emulate how much power is being generated, stored or used by an energy system, this can be done using DC motors, lamps and LEDs. The information telling the actuators how they should react is calculated by the Raspberry Pis and presented to the actuators as 8 bit data by the PCA8655E chip as discussed in Chapter 4, so this data has to be read and translated to a current or voltage output. This is done by an digital to analog converter (DAC), in this design there is chosen for the DAC0808 [10]. The DAC0808 is one of the more simpler DACs, requires a small power of at most 0.305 mW and can be operated with a supply voltage of 5 V, implying that it fits the current design. This DAC will draw a current depending on the digital inputs. So here 1111 1111 will translate to the highest current drawn and 0000 0000 will translate to a current of zero.

The output current (I_{out}) is dependent on the reference voltage (V_{ref}) and R_{14}, according to the data sheet of the DAC0808 [10] the relation is defined according to:

$$\frac{V_{ref}}{R_{14}} = I_{out} = 2mA$$

In this design V_{ref} is chosen to be 5 V and R_{14} is chosen to be 2.5 kΩ, this implies that the maximum output current in this design is equal to 2 mA.

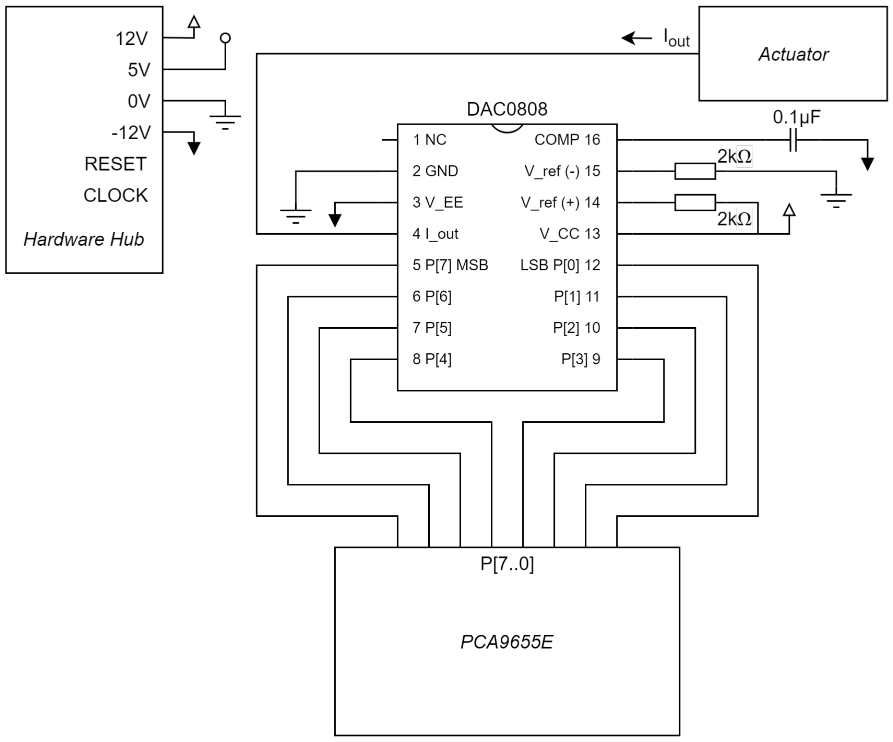

Figure 6.5: DAC

In between the DAC0808 and the actuator a transimpedance amplifier with a voltage follower has been designed as well (seen in Figure 6.6), since most actuators require a voltage instead of a current to run. The transimpedance amplifier has a potentiometer in the feedback loop, allowing the necessary output voltage to be adjusted for each actuator.

The voltage follower makes sure that no power can flow back into the DAC. Some devices act as both a sensor and actuator on the same line, so they are susceptible to accidentally transferring power into the DAC. Just using the transimpedance amplifier has a line in the feedback loop where current could be pushed to the DAC.

6.2.2. Sensor and Analog-Digital Conversion

The models need a sensor component to take the current surroundings into account in the computation. This means they need a device that outputs a voltage or current that is representative of the power they would be producing in a real world scenario. The voltage or current is then converted to a digital signal that can be read via the I^2C connection. Schematically it looks like Figure 6.7.

The eventual number that is returned by the Python function should be representative of the power, but does not need to be very accurate. This is so the user can get more intuition on what a power

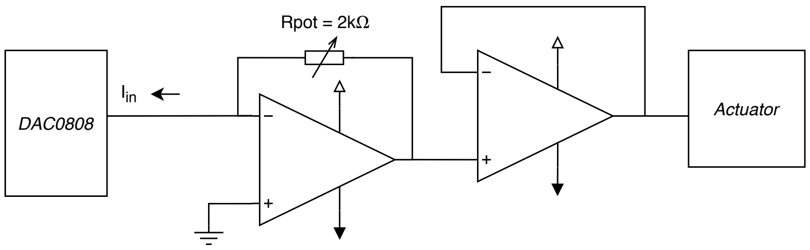

Figure 6.6: Transimpedance amplifier and voltage follower.

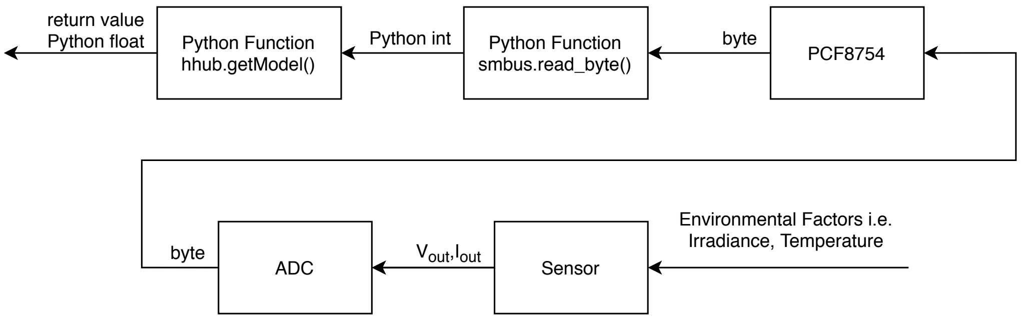

Figure 6.7: Schematic of the conversions done by the models needed for the sensing.

would mean in the real world. The idea is for example that the power from a solar cell goes up as the irradiance increases, or the power from a wind turbine increases as the motor is spun faster. Preferably, the return value from the function is linear with respect to the output power.

Following Figure 6.7, this linearity could be ensured by the function hhub.Model.sense() (from Chapter 5), or directly at the sensor. Picking a voltage or current that is linear with the output power is the best option here, because less assumptions have to be made about the behaviour of the sensing device.

The ADC device that is chosen is the ADC0808 [9]. And its implementation can be seen in Figure 6.8. It also comes with an accompanying application report [19].
A major downside to this design is that it requires two extra lines coming from the Hardware-Hub to operate. These lines are an extra clock signal and a startup/reset signal. The ADC0808 requires a clock to circle trough its internal states to eventually be able to convert the analog signal to digital data. It also has a ALE that requires a startup signal, ALE stands for address latch enable, and this input port needs be shortly triggered. This allows the ADC to read the address available on the pins [ADD A - ADD C] and then know from what input pin the ADC is supposed to read off.

The START also needs a pulse, this START is required for the control and timing of the ADC, and is responsible for the start of the data conversion. When a conversion cycle has finished the EOC is pulled up. This makes it possible to connect the EOC to the start and this allows the IC to run on itself, this use is also stated in the data sheet [9] and the application report [19]. To prevent the EOC from also triggering the ALE, a diode has been placed in between EOC and ALE.

6.3. Solar Cell

The modeling of the solar cell has been done using a small lamp and a 5 W solar PV. Hereby the lamp functions as the actuator and represents the sun. The brighter the lamp shines, the higher the solar irradiance is, resulting in more power being generated. The information telling how bright the lamp is

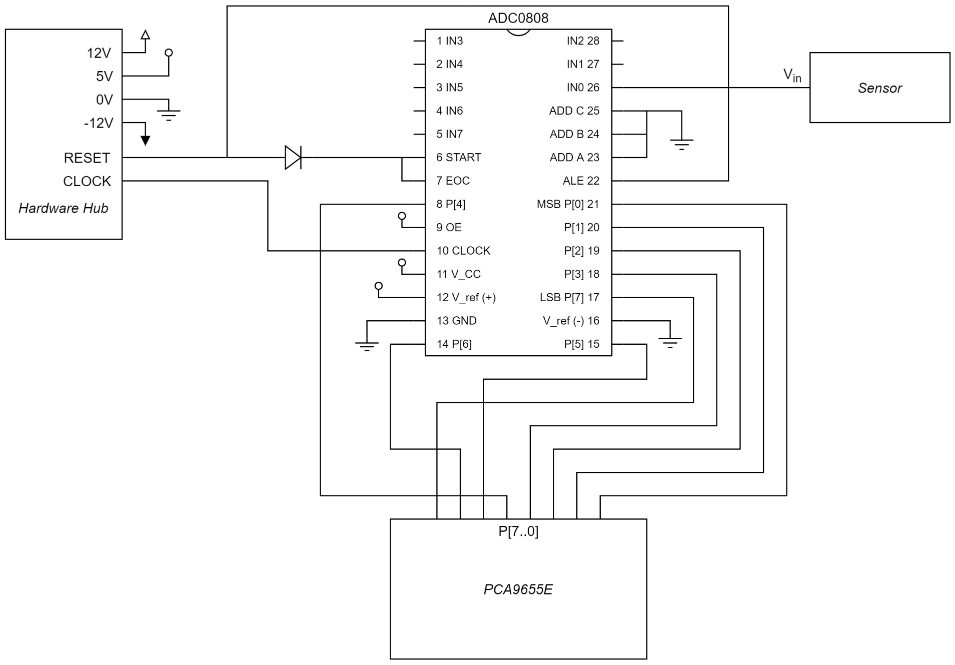

Figure 6.8: ADC0808 implementation

shining is coming from the Raspberry Pi and has no relation with the solar PV sensor unless this lamp is directly placed above the solar PV. For the sensor two design choice's were considered, sensing the light intensity using a LDR or using a real solar PV. Although sensing using the LDR would be easier to implement, it would be a bit dull. When using an actual solar PV however will result in a more accurate solar PV behaviour, and the actual table-top model will be more recognisable as a solar PV model.

6.3.1. Solar PV behaviour

A solar PV will output current and voltage based on multiple factors like irradiance, loads and temperature. The behaviour of a generic solar PV can be described using the I-V curves seen in Figure 6.9. A solar PV's I-V curve gets mainly influenced by two factors: the irradiance and the temperature. The temperature will influence the open circuit voltage, a higher temperature will result in a lower open circuit voltage. The short circuit current is influenced by the irradiance, a higher irradiance will result in a higher short circuit current. For this table-top model the goal is to measure the maximum output power of the solar PV, since most solar PV cells today are used in conjunction with maximum power point trackers (MPPTs). The relation between the power and the irradiance is quadratic, but can also approximated by a linear equation [20]. This makes it a good approximation to represent the power. Therefore measuring the short-circuit current is the objective of the solar PV sensor.

The output currents and voltages of the solar PV are also determined by what load it is connected to (Figure 6.10). A higher Resistance will create a lower current and a voltage closer to the open circuit voltage, so picking a high resistance could help measuring temperature changes. Since in this design the irradiance needs to be known, it is better to connect a low resistance load to the solar PV. [21][22][23]

6.3.2. Design Implementation

In the implementation of the solar PV sensor, a 5 W solar PV has been chosen and has a open circuit voltage of 6 V and produces a short circuit current of 0.866 A. There were more solar PV's considered,

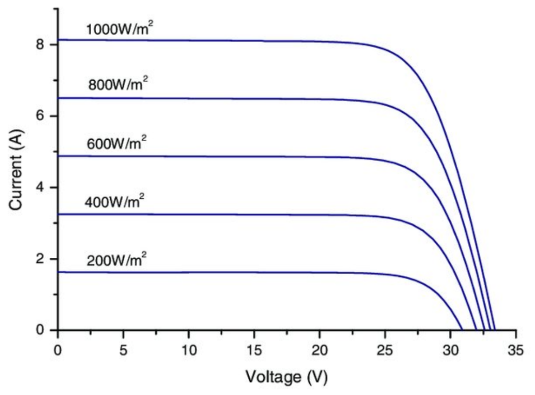

Figure 6.9: Influence of irradiance on IV-curves (Figure from [21])

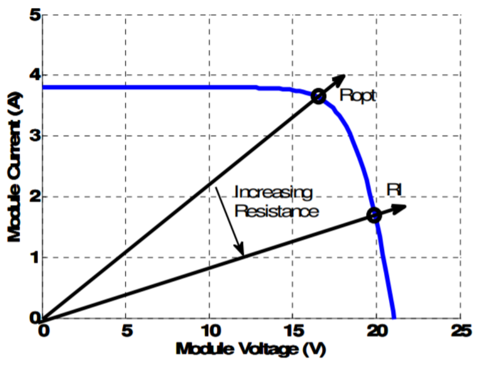

Figure 6.10: Influence of different loads on solar panel (Figure from [22])

those were ranging from 0.5 W to 40 W. The downside when choosing larger solar PV's is the price tag, a 40 W solar PV is quite expensive, since these PV's range between the 40 and 60 euros. And be course the requirements state that the system should be relatively affordable this wasn't an ideal option. The cheapest solar panels of 0.5 W have a price tag of around 2 euro. The downside with this cheap solar PV is that it would require very precise current measurements since the output currents would range between the 0 and 100 mA, als there are no data sheets are available and the power output could be non-linear. This is why the current solar PV option is a nice balance between the two extremes.

The implementation of the solar PV sensor can be seen in Figure 6.11, the design is build around the INA139[24]. The INA139 is an voltage comparator based current meter. That is able to output a voltage based on the following equation:

$$V_O = \frac{I_s * R_s * R_l}{1k\Omega}$$

Here is V_O the output voltage and I_s the current though the shunt resistor R_s (The resistor between $V_{in}+$ and $V_{in}-$). And R_l is the load Resistance that converts the output current of the INA139 to a voltage, here chosen to be 2 kΩ. The total resistance to which the solar PV is connected is in this design relatively low because this gives the best results when it comes to measuring irradiance, as discussed in 6.3.1.

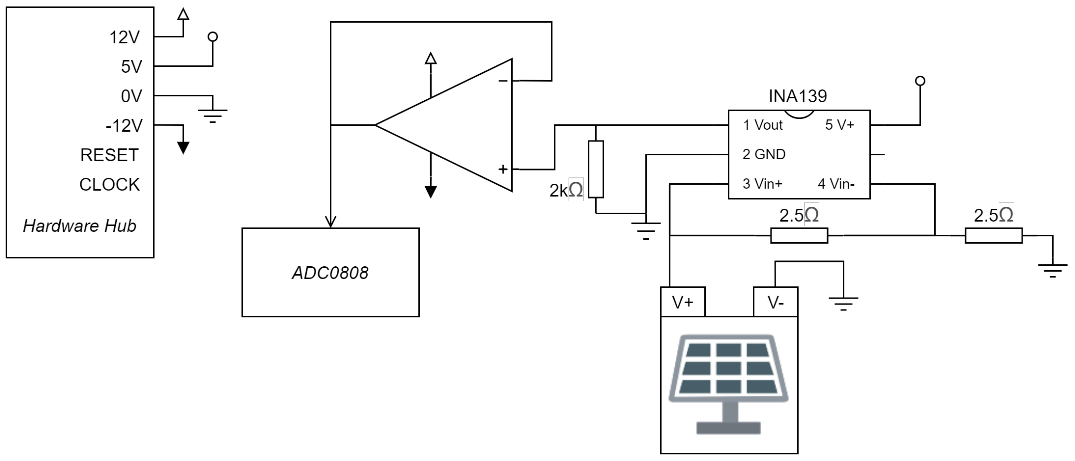

Figure 6.11: Solar PV Sensor Design

Since some solar panels might differentiate from their specifications, the output voltage of the INA139 has been chosen a bit lower than the maximum of 5 V (Since the ADC0808 cannot properly handle higher voltages).

Then a voltage follower is placed between V_{out} and the ADC0808. This to ensure that the input resistance of the ADC0808 doesn't influence the R_l, without the voltage follower the output current of the INA139 could travel through both R_l and the ADC, resulting in a lower than intended output voltage.

The design for the actuator is shown in Figure 6.12. This design uses the same transimpedance amplifier design as discussed in 6.2.1, only now without the voltage follower since the lamp wont be generating power and since the actuator isn't directly connected to the solar PV. For the lamp a generic 6 V lamp is chosen, this means that the potentiometer on the transimpedance amplifier has to be around 3 kΩ to ensure that the full range of the lamp is utilised.

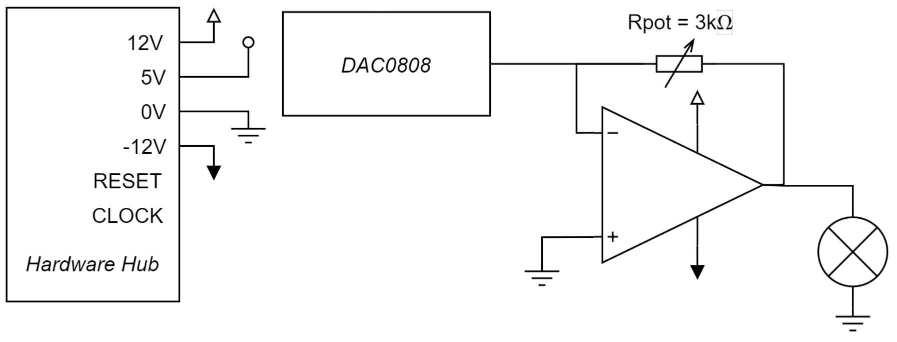

Figure 6.12: Solar PV Actuator Design

6.4. Wind Turbine

For modeling a wind turbine, a DC-motor has been used. This DC-motor will operate as an actuator and sensor, this because it will be more intuitive than a design that requires both some sort of generator and motor combination. The downside of using only one DC-motor is that most cheap DC-motors are designed for only outputting mechanical power. Or in the case of a dynamo only for generating electric power. This implies that the information given over the DC-motor is limited. Another downside is that the actuator is now also able to produce power, meaning that it has the possibility to make current flow back into the digital to analog converter, and damaging the IC's. These are some extra challenges but manageable with the right protection circuits.

In this design a 3 V DC motor has been used. It is also possible to use a 5 V motor but would require changing the potentiometers accordingly. The DC motor that was kept in mind when designing this system had an operating voltage range between 1.5 V and 3 V, and draws a power of maximal 3 W. It is important to keep in mind that it isn't possible to run a large quantity of these DC-motors on maximum speed since this would require more power than the power supply is able to deliver.

6.4.1. Design Implementation

In Figure 6.13 the design for implementing the DC-motor is shown. For this design the methodology was that different types of DC-motors (ranging from 1.5 V to 5 V) can be operated with this system. When using the motor as actuator, the analog signal is created by the DAC0808, this signal travels though the voltage controller and voltage follower as discussed in 6.2.1. For this design a 3V DC-motor is used, there it is recommended to set the potentiometer of the voltage controller to around 1.5

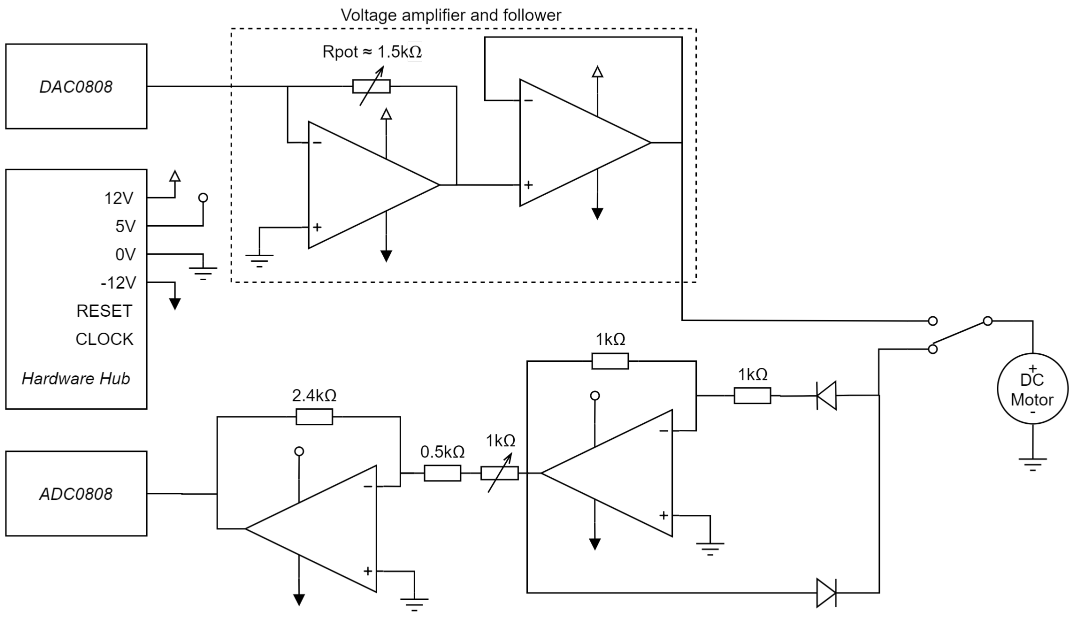

Figure 6.13: Implementation of the wind turbine

kΩ, this will give a maximum output voltage of 3 V and thus allows for using the entire range of the DC-motor. The DC-motor is also able to generate power while the switch is in actuator mode, to prevent this power from flowing back into the DAC the voltage follower was necessary to be implemented.

For using the DC-motor as a sensor/generator it is required to switch the slide switch and put the system into generator mode. This switch was necessary in the design because not including this switch would result in a faulty measurement. Since if a user were to stop the DC-motor from rotating the voltage created by the DAC would still be sensed, resulting into the data telling the Pi's that the wind turbine is rotating at some speed, while in reality the wind turbine is not rotating. So the solution for this was using the slide switch, and giving it two operating modes.

The ADC can only receive analog signals between 0 V and 5 V and since for this design a permanent magnet DC-motor has been used, the rotation in which the DC-motor is rotated determines the polarity of the generated voltage. To still be able to sense this data the voltage has to inverted. The generated voltage of the DC-motor will generally be between 0 V and 3 V, since the ADC has a reference voltage of 5 V, the signal has to be also amplified to ensure that the full range of the ADC gets utilised. And to add to that there is also the probability that someone will spin the DC-motor harder than it was designed for, this action can potentially create voltages above the recommended 5 V. The ADC0808 that is implemented in the design cannot handle voltages above 5 V. according to the data sheet [9], so the signal amplitude entering the ADC also needs to be limited to a maximum voltage of 5 V. The circuit that does all this can be seen in Figure 6.13, and is placed between the ADC and the DC-motor.

In this design there are some operational amplifiers required, for these amplifiers the µA741 IC's have been used.[11] This because these operational amplifiers are widely used, cheap and exist as model in PSpice allowing for simulation.

6.4.2. Test Setup

The design between the ADC and the DC-motor has been verified using P-spice, and the simulation setup can be seen in Figure 6.14. Here the potentiometer and the 0.5 kΩ resistor are combined as a single resistor of 1.2 kΩ meaning that the potentiometer in the design had to be set to 0.7 kΩ. The V_{in} in the PSpice setup represents the DC-motor. For the simulation a DC sweep was done for V_{in} and was done for the values between -8 V and 8 V. This represents a person rotating the permanently magnetised DC-motor counterclockwise to clockwise, likely this 8 V wont be reached by the 3 V motor, but this could be the case if the user connects for instance a 12 V DC-motor. So this setup will be able to check if the circuit is capable to handle higher voltages than anticipated. For the amplifiers the μA741 PSpice models have been used, since these are used in the design and gave better results than some other amplifier models like the lm324.

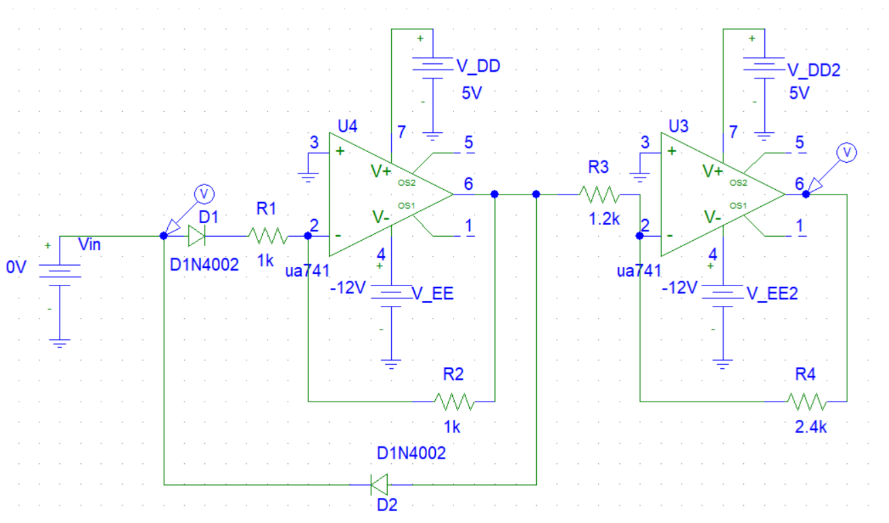

Figure 6.14: Simulation Setup for the Wind Turbine Model

6.4.3. Simulation Results

From the results in Figure 6.15, it is simulated that the output voltage has a range between 0 V and 4.6 V. The incoming negative voltages are converted into positive voltages correctly. It can also be seen that there is a small offset of about 0.15 V to the left, this implies that the output voltage graph is slightly off center. Another observation that can be made is that the input voltages between -0.8 V and 0.5 V give an output voltage of 0 V.

6.4.4. Discussion of the Results

From the results (Figure 6.15) it becomes clear that the output signal isn't perfect. The offset of 0.15 V is not a problem, since when the motor is being rotated, it is fairly difficult to tell how fast it is being rotated by a user. The fact that the output voltage is maximum 4.6 V has some negative impact, and this error is most likely caused by the imperfections of the operational amplifier. This lower voltage output will mean that the ADC now has a lower number of states it can digitally represent the rotational speed of the motor. Also since 5 V represents the maximum rotational speed, it would mean that some extra coding is required to make sure that the voltage input of 4.6 V represents the maximum rotating speed.

Another way to solve this problem is by feeding the right-amplifier a slightly higher supply voltage instead of the current 5V supply, but to realise this it would mean implementing a voltage divider together with a voltage follower to convert the 12 V to the desired supply voltage. But for this application it is not really worth the extra hardware.

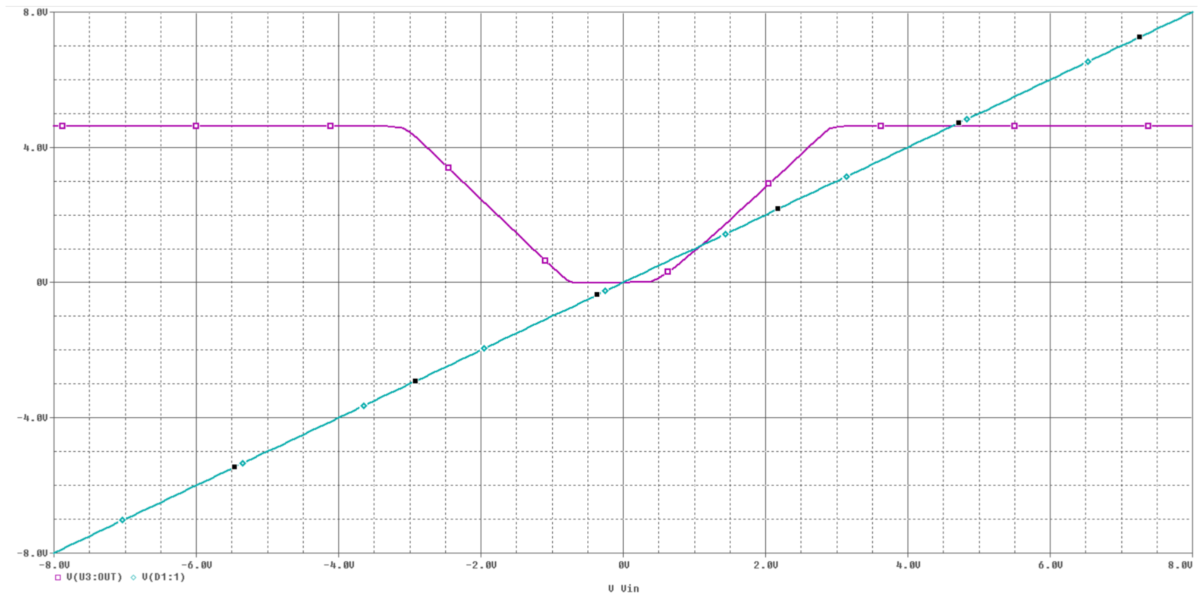

Figure 6.15: Simulation Results for the Wind Turbine Model

Another issue is the 0 V output between -0.8 V and 0.5 V, this is likely caused by the diodes. A way to solve this is to remove the inverting amplifier and diodes from the design, and create a mechanical solution that prevents turning the motor is one direction. But this would make the design less modular and requires a user to keep close attention to the orientation in which the DC-motor needs to be connected. Thus the current solution is good enough for the way it is used.

6.4.5. Recommendations

When actuating the DC-motor with no (mechanical) load attached it can rotate up to 14200 rpm, which isn't a really big problem, but it is a bit unrealistic for a wind turbine model. Also when using the DC-motor as sensor it would require the user to rotate it with similar speeds. This is why it is recommended to use a gearbox with a ratio of 1:64 and a handle to be able to rotate the DC-motor properly, since the handle and the gears also add some load, the rotational speed will be reduced significantly. This gear ratio is based on an existing electricity set which uses a DC-motor in a similar way [25].

6.5. Household

The household table-top model is a system that functions as a load only. This model uses the standard circuit 1 as discussed in 6.1.1, this model is thus able to sense and actuate. For actuating the power usage of an household a lamp was used in the same way the solar PV model actuates generated power 6.3.2. The sensor part was done using a potentiometer which allows users to change the load of the households by adjusting it.

6.5.1. Design Implementation

The implementation of the sensor can be seen in Figure 6.16. It consists of a voltage divider using a potentiometer and a voltage follower. The voltage follower ensures that the voltage divider doesn't also divide over the input resistance of the ADC. The potentiometer has been chosen at fairly high 10 kΩ so the system doesn't draw a to large current from the power supply.

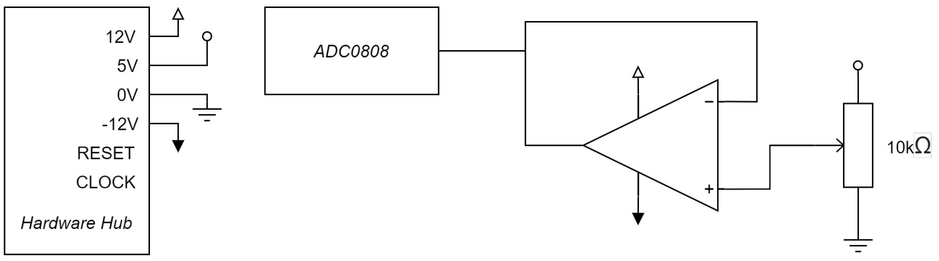

Figure 6.16: The Household Sensor Implementation

6.6. Battery

The battery table-top model is representing the Hydrogen Storage and Electric Vehicles present in the grid. Since both have a similar function it was decided that they have the same table-top model. Batteries store energy and the percentage of stored energy is what is being actuated by the models. The actuators should do this with a certain resolution, so the users are able to spot the different levels of state of charge easily. Standard Circuit 2 from Section 6.1.2 is used for this purpose.

Standard Circuit 2 uses 16 LEDs to output the SoC. Using 8 would have been too few, since the difference in SoC between any two levels would be 12.5%. The hydrogen storage battery and the electric vehicles will be kept at a SoC between There were some small design constraints however, one of which is that the battery has to be actuated with a certain precision. This implies that for instance eight different states for battery charge level will not be enough, since then jumps of 12.5% are made, and fluctuations in charge wont be that visible. So in the current design 16 states are used for visualising the battery charge. This visualisation has been done using LED’s, when all the LED’s are off it means the battery is empty of charge. When the first 8 LED’s are on it means the battery is charged for 50%, and when all the LED’s are on the battery is fully charged.

There can also be interacted with the batteries. Since electric vehicles can connect and disconnect from the grid, it means that the total battery capacity changes. This change in battery storage capacity can be changed in the design using switches, so when a switch is turned on it means that some vehicles are connected to the grid, or that an extra storage unit has been added to the grid.

7 Conclusion and Future Work

In the goal of this project was to design a physical energy system emulator, revered to as the Hardware Hub, which is capable of actuating and sensing information using table-top model representations. The Hardware Hub is also able to work together with the other subsystems to together create the Energy System Integration Demonstrator. These goals have been realised by creating a modular design, to which up to 32 different table-top models can be connected.
With the help of a Raspberry Pi 3B+ this design could be realised. A power supply has been integrated which is able to power 16 models, which is enough for most applications. It is still possible to add more power supplies to the design if needed. The Raspberry Pi is able to communicate with all table-top models using the I^2C protocol. The implementation of I^2C has been realised using the PCA9655E and Python code on the Raspberry Pi. The communication had to be tested using the PCF8574, so working code could be written. The table-top models are able to convert information between the analog and digital domain using the ADC0808 and DAC0808 implementations. Also, multiple table-top models have been designed, these models are the wind turbine, the solar PV, the household, the battery and the electric vehicles. These table-top models are able to emulate information satisfactorily.

7.1. Future Work

7.1.1. Power Grid Emulator

The next step to take is designing and implementing a power grid emulator. A system that allows the user to make their own grid topology and see through which lines the most power will flow. The main idea is that the topology can be changed by connecting wires from and to each model. This information regarding the topology will be retrieved by the Raspberry Pi. The Raspberry Pi is then able to calculate the power flows through all the lines. These power flow calculations can be done using models discussed in [26]. The primary challenge is the method in which information regarding the topology and connected components gets sent to the Raspberry Pi.

The current concept for solving this would be to reserve two I^2C addresses or to reserve model IDs for devices that determine grid topology. A simplistic routing algorithm could then be used to map the network.

7.1.2. USB-C Power Supply

A power supply using USB-C could be implemented using existing components. USB-C is better suited than other power supplies, since it uses a universally accepted connection, meaning that a even phone charger could be used to power the Hardware Hub.

The overview of such a power supply implementation can be seen in Figure 7.1. The USB-C power supply can differ depending on how many table-top models are used. If the user is only using two or four models, a 20 W power supply is more than enough. If a user is however actually using all the 32 possible table-top models, a power supply of at least 123.52 W is necessary, as discussed in 3.1.
For a USB-C charger or adapter to be connected to any device it needs to be able to preform a ”hand-shake”. So to connect a USB-C cable to the hardware hub it is required to be plugged into an USB-C trigger board. For this trigger board it is recommended to use the STUSB4500 [27]. This is a device that allows the user to select output voltages, ranging from 0 V to 22 V. This device is capable of out-putting powers up to 100 W, which is enough for at least 24 models. And if more power is required it is still possible to connect multiple power supplies to the Hardware Hub. A downside to this product however, is that it is quite expensive.

The different voltage supply levels, 12 V, -12 V and 5 V, can be partially realised by using buck con-verters. The 12 V power output would use the LM2596S [28], this device is cheap and able to deliver an output current of 3 A.
For the 5 V output a different buck converter has to be used since the 5 V power line powers a lot of components in the Hardware Hub, and the LM2596S is only able to output 15 W at 5 V. For four table top models this power would be enough, however since the design is required to power 32 table-top models, a larger output power would be ideal. Therefore it is recommended that for larger applications a buck converter is chosen that can output more than 15 W at 5 V.

What is still left to do is to create the -12 V power supply, and to verify the complete USB-C power supply design. A possible solution for the -12 V power supply is the P7805-S [29]. Which is able to create the -12 V output, but the drawback of this device is the low output current of 200 mA. Another solution would be by designing a buck converter using the design methodology described in [30] or [31]. The downsides of these implementations is that they were designed for lower voltages and may require extra boost converters to create the -12V output.

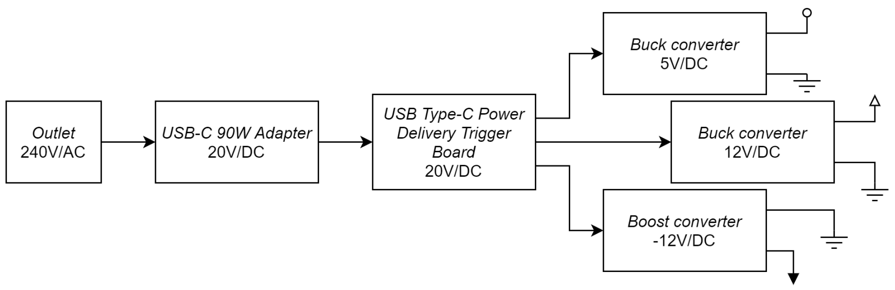

Figure 7.1: USB-C power supply

7.1.3. The Physical Product

As a result of university measures no physical product was allowed to be made, this is an important thing still left to do. It is still a goal to build, test and verify everything of the Hardware Hub and to also implement this project together with the other subsystems into the Demonstrator.

Bibliography

[1] D. Roberts, “New global survey reveals that everyone loves green energy - especially the chinese,” Nov 2017. [Online]. Available: https://www.vox.com/energy-and-environment/2017/11/20/16678350/global-support-clean-energy

[2] A. Alexandru, E. Tudora, O. Bica, and R. Mayer, “Educating young people for sustainable energy development,” *Bulletin of Electrical Engineering Faculty*, vol. 11, pp. 53–57, 11 2011.

[3] K. Mulder, J. Segalas, and D. Ferrer-Balas, “Educating engineers for/in sustainable development? what we knew,what we learned, and what we should learn,” *Thermal Science*, vol. 14, pp. 625–639, 01 2010.

[4] B. A. Bossink, “Demonstrating sustainable energy: A review based model of sustainable energy demonstration projects,” *Renewable and Sustainable Energy Reviews*, vol. 77, pp. 1349 – 1362, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032117302010

[5] “Esdl documentation.” [Online]. Available: https://energytransition.gitbook.io/esdl/

[6] N. Blair, A. P. Dobos, J. Freeman, T. Neises, M. Wagner, T. Ferguson, P. Gilman, and S. Janzou, “System advisor model, sam 2014.1.14: General description,” 2 2014.

[7] “Raspberry Pi Hardware - Raspberry Pi Documentation,” Raspberry Pi Foundation, Tech. Rep., [Accessed: 19 Jun. 2020]. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/

[8] “Remote 16-bit I/O Expander for I2C Bus with Interrupt,” PCA9655E Datasheet, ON Semiconductor, Mar. 2017, [Rev. 5].

[9] “ADC0808/ADC0809 8-Bit µP Compatible A/D Converters with 8-Channel Multiplexer,” ADC0808/ADC0809 Datasheet, Texas Instruments, Oct. 1999, [Rev. Mar. 2013].

[10] “8-Bit D/A Converter,” DAC0808 Datasheet, Texas Instruments, May 1999.

[11] “µA741 General-Purpose Operational Amplifiers,” µA741 datasheet, Texas Instruments, Nov. 1997.

[12] “65W Triple Output Switching Power Supply,” RT-65 datasheet, Mean Well, Jul. 2017.

[13] J.Mankar, C.Darode, K.Trivedi, M.Kanoje, and P.Shahare, “Review of i2c protocol,” *International Journal of Research in Advent Technology*, vol. 2, no. 1, p. 474–479, Jan. 2014. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.429.1402&rep=rep1&type=pdf

[14] A. van Genderen and J. Wong, “Digital Systems B Lec 6,” EE1D12, Delft Univ. of Technol., 2017-2018.

[15] “Remote 8-bit I/O Expander for I2C Bus,” PCF8574 Datasheet, Texas Instruments, Jul. 2001, [Rev. Mar. 2015].

[16] “LED, Green, 5 mm,” MCL053GT datasheet, Multicomp Pro, Sep. 2019.

[17] “Round LED, Orange, 5 mm,” MCL053AT datasheet, Multicomp Pro, Sep. 2019.

[18] “Round LED, Super Red, 5 mm,” MCL053SRD datasheet, Multicomp Pro, Sep. 2019.

[19] “ANB-247 Using the ADC0808/ADC0809 8-Bit µP Compatible A/D Converters with 8-Channel Analog Multiplexer,” Texas Instruments, Sep. 1980, [Rev. Apr. 2013].

[20] S. A. M. Abdullah, Siti Fauziah Toha, “Investigation on the Maximum Power Point in Solar Panel Characteristics Due to Irradiance Changes,” et al 2017 IOP Conf. Ser.: Mater. Sci. Eng.

[21] T. Ma, H. Yang, and L. Lu, “Pumped storage-based standalone photovoltaic power generation system: Modeling and techno-economic optimization,” *Applied Energy*, vol. 137, pp. 649–659, Jan. 2015.

[22] J. Durago, “Photovoltaic emulator adapatable to irradiance, temperature and panel-specific i-v curves,” Master’s thesis, Jun. 2011. [Online]. Available: https://digitalcommons.calpoly.edu/theses/541

[23] A. Smets, “Lecture 7 solar energy - photovoltaics,” EE2E21, Delft Univ. of Technol., 2018-2019.

[24] “High-Side Measurement CURRENT SHUNT MONITOR,” INA139 datasheet, Burr-Brown Products from Texas Instruments, Dec. 2000, [Rev. Nov. 2005].

[25] “Electriciteit,” S87803 electricity set, Clementoni, [Discontinued Product].

[26] M. Cvetkovic, “lec04 PowerSystemOperations,” EE2E21, Delft Univ. of Technol., 2018-2019.

[27] “Standalone USB PD sink controller with short-to-VBUS protections,” STUSB4500 datasheet, STMicroelectronics, Dec. 2019, [Rev. 3].

[28] “LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator,” LM2596S datasheet, Texas Instruments, Nov. 1999, [Rev. Feb. 2020].

[29] “NON-ISOLATED SWITCHING REGULATOR,” P7805-S datasheet, CUI inc., Feb. 2019.

[30] O. Bryant, “Creating a Negative Output Voltage Using a Buck Converter,” Application Note, VISHAY SILICONIX, [Rev. Feb. 2016].

[31] S. Gupta, “Creating an Inverting Power Supply Using a Synchronous Step-Down Regulator,” Application Report, Texas Instruments, Aug. 2011, [Rev. Jun. 2018].