Inhaltsverzeichnis

1 Einleitung

Energie ist für vieles nötig. Ohne Energie wären die bisherige industrielle Entwicklung und das Leben wie es bekannt ist, nicht möglich. Erst Energieträger wie Öl, Erdgas und vor allem elektrische Energie sollen den wirtschaftlichen und technischen Fortschritt erst ermöglichen haben [vgl. Bpb2013 S. 4 – S. 6]. 2014 wurden 684 TWh Strom erzeugt. Die Quelle der erzeugten elektrischen Energien wurden vor allem durch fossile Brennstoffe erzeugt [vgl. EDG S.38].

Aufgrund des Energieverbrauchs finden auch Klimaänderungen durch Umweltverschmutzung, verursacht durch Treibhausgase wie Kohlendioxid, statt. Dies hat nicht nur eine steigende globale Durchschnittstemperatur, sondern auch extreme Wetterereignisse, wie Stürme und Überflutungen zur Folge [vgl. Ven2013]. Die globale Durchschnittstemperatur soll um bis zu vier Grad Celsius steigen, wenn nichts dagegen unternommen wird.

Die Reduzierung des Stromverbrauchs hat nicht nur klimaschützende, sondern auch Kosten reduzierende Wirkung, da verminderte Nachfrage und Verbrauch auch entsprechend niedrigere Ausgaben für Strom bedeuten. Die Kontrolle des eigenen Energieverbrauchs wird daher immer wichtiger. Damit ist nicht nur die Überwachung der eigenen Energiekosten möglich, sondern auch die Steigerung der Energieeffizienz von Anlagen [vgl. pae]. Zudem wird durch die Energieverbrauchskontrolle die Möglichkeit eröffnet, die Umwelt zu schonen. Die Reduzierung des eigenen Stromverbrauchs und die Vermeidung von Spitzen sorgen dafür, dass die Stromversorgung und somit auch die Zulieferung und Produktion von Strom eingedämmt wird. Umweltschutz wird also nicht nur dadurch gefördert, wie und mit welchen Mitteln Strom gewonnen wird, z.B. durch erneuerbare Energien, sondern auch durch die Nachfrage danach. Um zum Umweltschutz beizutragen kommt es also nicht nur darauf an, wie der Strom erzeugt wird, sondern auch wie viel Energie verbraucht wird. Es hängt somit auch vom eigenen Verbrauch ab, weil die Menge des produzierten Stroms vermindert wird. Da die Stromerzeugung auch von der Nachfrage abhängig ist.

Hier kommt die in dieser Arbeit entwickelte Softwarelösung zum Einsatz. Mit der S0-Schnittstelle, nach DIN EN 62053-31, kann nicht nur der aktuelle Verbrauchstand von den Zählern direkt abgelesen, sondern auch über ganze Zeiträume hinweg erfasst werden, die dann aufbereitet und Visualisiert werden. So werden die Verbrauchsdaten nicht nur für einen bestimmten Zeitpunkt dargestellt, sondern für ganze Zeiträume.

Bei der Visualisierung der Energieverbrauchsdaten über die Zeit können die Verbrauchsdaten analysiert werden. Durch die Analyse können unnötige Verbräuche erfasst, dargestellt, eventuelle Zusammenhängen erkannt und so der Energieverbrauch optimiert werden.

2 Aufgabenstellung

Die Optimierung des Stromverbrauchs wird immer wichtiger. Dadurch kann die Umwelt geschützt und der eigene Energieverbrauch gesenkt werden. Ersteres wird aufgrund der immer weiter steigenden Umweltverschmutzung bei der Energiegewinnung wichtiger.

Zur Reduzierung der Umweltverschmutzung bei der Energiegewinnung und damit auch beim Energieverbrauch kann beigetragen werden, indem der eigene Verbrauch analysiert, unnötige Verbräuche ermittelt und optimiert werden können. Die Reduzierung des Energieverbrauchs hat auch einen Ausgaben reduzierenden Charakter.

Dazu soll in dieser Bachelorarbeit eine Monitorringlösung entwickelt werden, die über vier S0- Schnittstellen, den Energieverbrauch in Gebäuden automatisch am Zähler misst, verarbeitet und abspeichert. In dieser Arbeit soll mit dem Aufsatz SD0, direkt über seinen vier S0-Kanälen, von den am Stromzähler angebrachten S0-Schnittstellen der Energieverbrauch gemessen werden. Die Abspeicherung der Verbrauchswerte soll sowohl lokal, als auch auf einem Server erfolgen können. Dazu gibt der Aufsatz die Verbrauchswerte an den Rasperry Pi weiter, wo die Verbrauchswerte abgespeichert werden. Vom Raspberry Pi aus können, wenn nötig, die Verbrauchsdaten an einen Server weitergeleitet und dort abgelegt zu werden. Die Visualisierung der Verbrauchswerte soll über eine Webdarstellung vorgenommen werden.

Die Verbrauchsmessung und deren Darstellung ermöglicht, dass die Verbrauchsdaten analysiert und der Stromverbrauch des Nutzers bzw. des Systems optimiert werden können.

Die aufgenommenen S0-Impulse können Werte von Elektro-, Wasser- oder Gaszählern repräsentieren. Für die Möglichkeit der Konfiguration der Impulskonstante und der Messleitung ist eine Einstellmöglichkeit herauszusuchen. Die entwickelte Lösung ist prototypisch zu implementieren und in einem Testaufbau zu überprüfen.

3 Fachliches Umfeld

In diesem Kapitel werden die für diese Arbeit eingesetzte Hardware und die eingesetzten Schnittstellen UART und I²C erläutert. Zusätzlich werden die Datenhalungs bzw. -darstellungsform beschrieben. Bei der Abschlussarbeit wird Raspberry Pi und die Erweiterung SD0 von Busware eingesetzt.

Da für die Entwicklung der Softwarelösung der Aufsatz SD0 von Busware verwendet werden muss, trifft die Wahl des Einplatinencomputers auf den Raspberry Pi, weil der Aufsatz für den Raspberry Pi konzipiert worden ist [vgl. bsd0].

3.1 Beschreibung des Raspberry Pi

Als Platine für die Erweiterung wird der Raspberry Pi eingesetzt, da die Erweiterung SD0 dafür ausgerichtet ist. In diesem Kapitel wird entschieden welches Modell des Raspberry Pi ausgewählt wird und dieses beschrieben.

Tabelle 3.1: Eigenschaften der verschiedenen Raspberry Pi Modellen

Eigenschaften		Modell A	Modell A+	Modell B	Modell B+	Raspberry Pi 2 Model B
Gesamtgrösse (in mm)	Länge	93	70,4	93		
	Breite	63,5	57,2	63,5		
	Höhe	17	12	20		
SoC		Broadcom BCM2835				Broadcom BCM2836
CPU	Typ	ARM1176JZF-S				ARM Cortex-A7
	Kern	1				4
	Takt	700 MHz				900 MHz
	Architektur	ARMv6				ARMv7
Arbeitsspeicher		256 MB		512 MB	512 MB	1024 MB
Speicher		Kartenleser für Full SD	Micro SD	Kartenleser für Full SD	Micro SD	Micro SD
Anzahl der USB 2.0 Anschlüsse		1		2	4	
Ethernet		-		10 und 100 MBit		
Pin		26	40	26	40	
GPIO-Pins		17	26	17	26	
Weitere Schnittstellen		1 × CSI, 1 × DSI, 1 × I²S				
Stromversorgung		5,0 V; über einen Micro-USB-Anschluss (Micro-USB-B)				

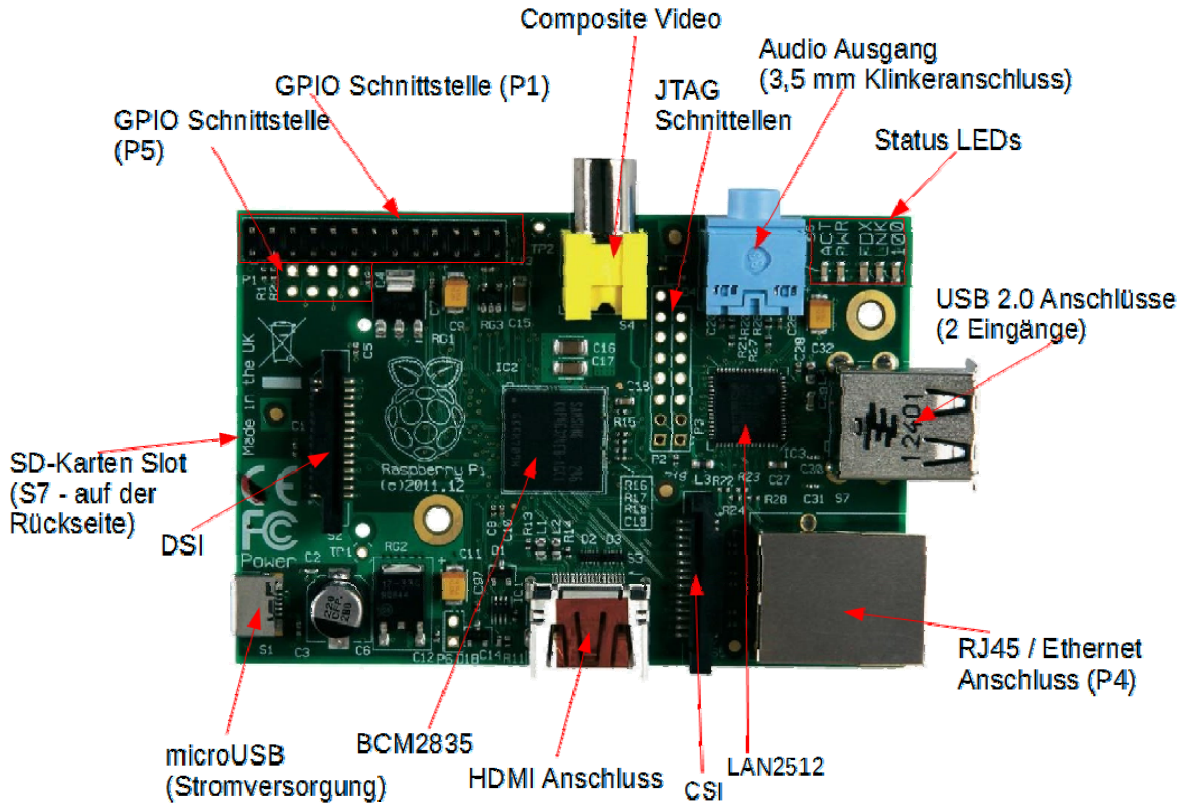
(Quelle: [vgl. wRPi und RpiM] Modifiziert)

Aufgrund der Anzahl der Pins und der Anordnung der Komponenten auf dem Board kann zwischen den Raspberry Pi Modellen A und B ausgewählt werden, da der Aufsatz SD0 an sich 26 Pins hat. Bei den anderen Raspberry Pi Modellen, also beim Raspberry Pi A+, B+ und Raspberry Pi 2 Modell B würde es beim Aufsetzen auf die Raspberry Pi Modellen mit mehr als 26 Pins die anderen Pins umbiegen bzw. zerstören. Zudem würde der Display Connection Port (DSI), das Aufstecken des Aufsatzes SD0 behindern. Der Aufsatz hat eine entsprechende Einbuchtung für den DSI Port bei Raspberry Pi A und B.

Da der Raspberry Pi Modell B, im Gegensatz zum Modell A einen höheren Arbeitsspeicher und einen Netzwerkanschluss hat, wird Modell B eingesetzt. Der Netzwerkanschluss ist für die Darstellung des Energieverbrauchs wichtig, weil die Daten entweder für die Darstellung an einen Server gesendet oder auf dem Raspberry Pi präsentiert werden.

In diesem Kapitel wird nicht nur der SoC BCM2835, welcher den Kern des Raspberry Pis bildet erläutert, sondern auch die Komponenten die für den Einsatz der zu entwickelnden Softwarelösung benötigt werden. Dabei handelt es sich vor allem um die Ein- und Ausgänge, insbesondere den GPIO Pins mit den benutzten Funktionen. Der Grafikprozessor, mit seinen Ausgängen wird nur am Rande erwähnt.

Abbildung 3.1: Raspberry Pi Model B



(Quell: [vgl. RpiB]; Modifiziert)

3.1.1 System on a Chip (SoC)

Der Raspberry Pi Model B basiert auf dem BCM2835 von Broadcom, dessen Kern die CPU ARM1176JZF-S ist. Die CPU gehört der ARM11 Generation an, welche auf der Architektur ARMv6 mit 700 MHz basiert [Upt2014, S.19 und Kof2015 S. 359]. Sie hat eine Adressbreite von 32 Bit [vgl. Kof2015 S. 359]. Beim BCM2835 handelt es sich um einen SoC, also um einen System on a Chip. Dies bedeutet, dass es nicht nur die CPU sondern auch andere Teilsysteme beinhaltet, so dass diese nicht einzeln auf dem Raspberry Pi verbaut sind. Neben dem CPU beinhaltet es den Grafikprozessor (GPU) und den Arbeitsspeicher (SDRAM) in der Größe von 512 MB [vgl. Kof2015 S. 359 und Dem2013 S. 56 - S. 57] sowie die Verbindungen zu den GPIO Pins. Die CPU besitzt zudem separate Cache-Speicher für Daten und Befehle [vgl. Dem2013 S. 61]. Beim Arbeitsspeicher handelt es sich um einen Internen Speicher,

sodass die CPU keine externen Datenverbindungen besitzt, da keine externen Arbeitsspeicher angeschlossen werden [vgl. Dem2013 S. 63].

Beim Grafikprozessor handelt es sich um einen VideoCore IV welches den H.264 Standard mit einer 40 MBits/s unterstützt, die Videos mit einer Datenkompression von H.263 mit einer Rate von bis zu 40 MBits/s darstellen kann. Der Grafikspeicher soll automatisch ein Teil des Arbeitsspeichers mitbenutzt werden [vgl. Kof2015 S. 359]. Die Darstellung kann über drei verschiedene Ausgänge erfolgen. Bei den Ausgängen handelt es sich um HDMI, Composite Video und DSI [vgl. Dem2013 S. 68].

Die mit dem Grafikprozessor verbundenen Comosite-Video und HDMI Anschlüsse können direkt eingesetzt werden. Der Display Serial Interface benötigt spezielle Hardware [vgl. Upt2014 S. 22] wie einem Folienkabel [vgl. Dem2013 S. 92].

Das DSI ist eine serielle Steckerleiste mit 15 Pins, an dem ein Display angeschlossen und der RPi autonom benutzt werden kann, ohne dass es an einen Monitor oder Display, wie sie von normalen Computern her bekannt ist, angeschlossen werden muss [vgl. Upt2014 S. 24 und vgl. Dem2013 S. 72]. Die Klemmhalterung dient der Aufnahme eines Folienkabels.

In dieser Arbeit wird weder der Grafikprozessor noch die Anschlüsse des Grafikprozessors verwendet. Lediglich der Jumper des DSI muss abmontiert werden, da sonst der Aufsatz SD0 von Busware nicht richtig auf die GPIO-Leiste des Raspberry Pis aufgesteckt werden kann.

3.1.2 Stromversorgung

Der Raspberry Pi Modell B sollte mit einem Strom von mindestens 700 mA (Millampere) und einer Spannung von 5 Volt versorgt werden. Eine Polyfuse verhindert eine Überversorgung von über einem Ampere, also 1000 Milliampere [vgl. Dem2013 Kapitel 3.8].

3.1.3 Datenspeicher des Raspberry Pis

In diesem Unterkapitel wird nicht der Arbeitsspeicher behandelt, sondern der Daten- und Programmspeicher. Da der Raspberry Pi keinen Speicher bzw. keine Festplatte besitzt, benötigt er eine SD-Karte [vgl. Upt2014 S. 27]. Auf der SD-Karte werden nicht nur Programme und Daten abgespeichert; darauf läuft auch das Betriebssystem.

Es können drei verschiedene SD-Karten-Standards am Raspberry Pi Model B angeschlossen werden, da diese die gleichen Abmessungen besitzen. Anzumerken ist, dass der Schreibschutz an den Karten vom Raspberry Pi nicht ausgewertet wird. An den SD-Karten Slot können SD-Karten mit den Abmessungen 32 mm x 24mm x 2,4mm angeschlossen werden. Bei den einsetzbaren SD-Karten handelt es sich um folgende Karten [vgl. Dem2013 S. 64]:

SD-Karte mit einer Kapazität von 8 MB bis 2 GB

SDHC (SD High Capacity) Card mit einer Kapazität von 4 GB bis 32 GB und

SDXC (SD eXtended Capacity) Card mit einer Kapazität von 64 GB bis 2 TB

Bei der SDHC-Karte ist wiederum zu beachten, dass sie verschiedene Klassen besitzt, also eine Abstufung der Mindestübertragungsrate. Bei den Abstufungen handelt es sich um folgende [vgl. Dem2013 S. 66]:

Class 2, mit 2 MByte/s Mindestübertragungsrate

Class 4, mit 4 MByte/s Mindestübertragungsrate

Class 6, mit 6 MByte/s Mindestübertragungsrate

Class 8, mit 8 MByte/s Mindestübertragungsrate

Mit den unteren Übertragungsraten, also mit Class 2 und 4 soll es kaum Probleme geben. Mit den höherklassigen SD-Karten, also mit Class 6 und 8, kann es jedoch Kompatibilitätsprobleme geben. Die höheren Klassen werden bei der Wiedergabe von ruckelfreien Videowiedergaben bevorzugt [vgl. Dem2013 S. 65 bis S. 66]. Für das Betriebssystem des Raspberry Pi werden mindestens 4 GB gebraucht [vgl. Upt2014 S. 27]. Es wird jedoch 8 GB empfohlen [vgl. Upt2014 S. 27].

Das SD-Karten Slot, das sich unterhalb des Raspberry Pi befindet, ist direkt mit dem SoC BMC2835 verbunden. Der eigentliche Card-Controller befindet sich auf der SD-Karte. Neben der Masseleitung des Blechslots (MTG1 und MTG2) und den Masseleitung für die Karte selbst (Vss1 und Vss2) wird der Slot bzw. die Karte über die Leitung Vdd mit Spannung versorgt. Die Datenübertragung erfolgt taktgesteuert. Die Karte wird über die Leitung CLK mit dem Takt vom BCM2835 versorgt. Die Datenübertragung an sich findet über die Datenleitungen DAT0, DAT1, DAT2, DAT3 statt. Die Leitung DAT3 kann gleichzeitig zur Kartenerkennung (CD: Card Detection) benutzt werden. Ob auf die Karte geschrieben oder von der Karte gelesen werden soll, wird von der Leitung CMD bestimmt [vgl. Dem2013 S.66].

3.1.4 Betriebssystem

Für den Raspberry Pi existieren unterschiedliche Betriebssysteme, die für verschiedene Einsatzzwecke konzipiert wurden. Bei den meisten handelt es sich um Betriebssysteme von Drittanbietern [vgl. RPiD].

Neben Betriebssystemen, wie OpenELEC (Open Embedded Linux Entertainment Centre) oder OSMC (Open Source Media Centre) [vgl. RPiD] die als Mediacenter, also für das Streaming von Medien, wie Musik und Filmen genutzt werden können, [vgl. Kof2015 S. 55] gibt es Betriebssysteme, die auf verschiedenen Linux-Distributionen basieren. Dabei handelt es sich beispielsweise um Raspbian, RISC OS, Pidora oder ArchLinux ARM. Das Betriebssystem wird vor allem im Hinblick auf die Unterstützung des Raspberry Pi und evtl. späterer Änderungen und Erweiterungen. des im Rahmen dieser Arbeit erstellten Softwarelösung ausgewählt. Zudem spielt die Portabilität der entwickelten Softwarelösung auf andere Einplatinencomputer ein wichtiges Kriterium.

Pidora, ist ein auf der Fedora-Distribution basierendes Betriebssystem [vgl. Kof2015 S.54]. Pidora soll langsam arbeiten und im Betrieb sollen merkliche Fehler auftauchen [vgl. Dem2013 S.96].

Beim ArchLinux handelt es sich um einen an Einplatinencomputer, wie BeagelBone und Raspberry Pi optimierte Arch-Distribution [vgl. Kof2015 S.55]. Es soll jedoch eine geringe Anzahl an Softwarepaketen anbieten [vgl. Dem2013 S.96], was nicht optimal für spätere Erweiterungen der entwickelten Software sein kann.

RiscOS ist ein nicht auf Linux basierendes, im Jahre 1987 von Acorn entwickeltes Betriebssystem. Für dieses Betriebssystem soll es wenig Open-Source-Software geben [vgl. Dem2013 S.96].

Beim Raspbian handelt es sich um ein vom Raspberry Foundation empfohlenes Betriebssystem [vgl. Upt2014 S. 48]. Bei dieser Arbeit wird Raspbian eingesetzt. Das hat verschiedene Gründe. Raspian soll einerseits ein stabiles, auf Debian basierendes und andererseits speziell auf den Broadcom BCM2835 angepasstes Betriebssystem sein, [vgl. Dem2013, S. 95] das auf ein direktes Zusammenspiel mit der Hardware des Raspberry Pi ausgerichtet ist [vgl. EPB1]. Es unterstützt beispielsweise den Floating Point Unit [vgl. Dem2013, S. 95]. Auch in Hinblick auf den oben genannten Punkt der Erweiterung der zu entwickelnden Software, erfüllt es das Kriterium. Mit Raspbian werden Softwarepakete mitgeliefert, die entweder nur ihm zur Verfügung stehen oder in anderen Distributionen kompiliert werden müssen, falls diese dort zum Einsatz kommen sollen [vgl. Kof2015 S.55].

3.1.5 I/O – Anschlüsse

Der Raspberry Pi benutzt den Micro USB-Anschluss und den SD-Karten-Slot zum Betrieb. Auf der SD-Karte läuft, wie im Kapitel 3.1.2„Datenspeicher des Raspberry Pis“ erwähnt, das Betriebssystem. Über den Micro USB-Anschluss wird der Raspberry Pi mit Strom versorgt. Darüber hinaus kann es auch mit einem Computer verbunden werden. In diesem Fall wird es über den Computer mit Strom versorgt. Über den SD-Karten Slot wird eine SD-Karte angeschlossen, die das Betriebssystem für den Raspberry Pi beinhaltet. Die SD-Karte wird nicht nur für das Betriebssystem, sondern auch als Speicher benutzt.

Neben den Anschlüssen, die für den Betrieb gebraucht werden, besitzt der Raspberry Pi Model B jeweils einen Audio- und Video Composit-Ausgang und zwei USB-Anschlüsse, über die beispielsweise Tastatur oder Maus angeschlossen werden können. Daneben besitzt er einen HDMI-Anschluss für eine hochauflösende, verlustfreie Video- und Audiosignalübertragung [vgl. ITW]. Wie in der Tabelle 3.1 im Kapitel 3.1 dargestellt, besitzt der Raspberry Pi Model B auch einen Ethernet - Anschluss, für einen Anschluss an ein Netzwerk.

Der Raspberry Pi Model B hat neben den fünf Status LEDs und dem Ethernet Anschluss noch die Verbindungen, die von S1 bis S7 und P1 bis P6 bezeichnet werden. In den folgenden beiden Tabellen werden die Schnittstellen erläutert und mit konkreten Anschlüssen dargestellt.

Tabelle 3.2: Schnittstellen des Raspberry

Anschluss	Beschreibung
S1	Micro USB-Anschluss (für die Stromversorgung)
S2	DSI-Anschluss
S3	HDMI Anschluss, Typ A
S4	Composite Video Anschluss (RCA)
S5	CSI Anschluss
S6	Audio-Ausgang; 3,5 Klinker Anschluss
S7	SD Karten-Slot (befindet sich unterhalb des RPi)

Die Folgende Tabelle Listet die verfügbaren GPIO Pins des Raspberry Pis auf.

Tabelle 3.3: GPIO Anschlüsse des Raspberry

Anschluss	Beschreibung
P1	GPIO Steckerleiste; 2x 13 Pins
P2	JTAG für GPU; Für 8 Pins (keine angeschlossenen Pins)
P3	JTAG für LAN9512; Für 7 Pins (keine angeschlossenen Pins)
P4	RJ45 Ethernet Anschluss
P5	GPIO Erweiterungspins; 8 Anschlüsse ohne Pins
P6	Reset-Button-Anschluss; (keine angeschlossenen Pins)

In dieser Arbeit werden die Anschlüsse S1, S7, P1 und P4 verwendet. Die anderen Anschlüsse werden der Vollständigkeit halber erläutert.

3.1.6 GPIO Anschlüsse

Bei einer General Purpose Input/Output (kurz GPIO) Schnittstelle handelt es sich, ganz allgemein betrachtet, um eine Schnittstelle die nichts über die einzelnen Pins, noch über deren elektrische Daten oder die Art und Weise der Programmierung aussagt. Erst durch den Einsatz auf einer bestimmten Platine und deren Bestückung gewinnt die GPIO an Bedeutung [vgl. Dem2013 S. 74 bis S.75]. Beim GPIO des Raspberry Pi handelt es sich um eine Schnittstelle, über die der Raspberry Pi mit Schaltungen, Platinen usw. erweitert werden kann.

Wie aus der Tabelle 3.2 zu entnehmen ist, existieren neben der eigentlichen GPIO-Schnittstelle mit der Bezeichnung P1 noch weitere Schnittstellen, die in unterschiedlichen Bereichen eingesetzt werden.

Bei den Schnittstellen mit der Bezeichnung P2 und P3 handelt es sich um JTAG Schnittstellen (Joint Test Action Group). Diese werden bei der Produktion des Raspberry Pi benutzt, um die Funktionalität des Raspberry Pi zu testen [vgl. Kof2015 S. 364 bis S.365]. Die Schnittstelle mit der P2-Bezeichnung wird für den GPU also für die Grafikeinheit benutzt. Mit der P3-Schnittstelle wird die Funktionalität des LAN2512-Chips getestet. Der LAN2512 beinhaltet den USB- Hub, mit 2 USB2.0-Ports und den LAN Controller.

Die GPIO Schnittstelle mit der Bezeichnung P5 besitzt acht Kontakte, die nicht mit Pins bestückt sind. Diese GPIO Schnittstelle steht erst ab der 2 Revision zur Verfügung. Über die ersten beiden Kontakte werden Spannungen zur Verfügung gestellt. Beim ersten Kontakt sind 5 Volt und über den zweiten Kontakt 3,3 Volt verfügbar. Die letzten beiden Kontakte bilden die Masse. Die restlichen vier Kontakte der Schnittstelle bieten zwei Funktionen an: die Die Primärfunktion bildet das Bussystem I²C über die Kontakte 3 und 4. Dieser ist der zweite I²C Kanal, auf dem Raspberry Pi. Die zweite bzw. alternative Funktion bilden die Kontakte 5 und 6. Über diese ist der I²S Bus benutzbar. Dieses Bussystem überträgt Audiosignale [vgl. Kof2015 S.365].

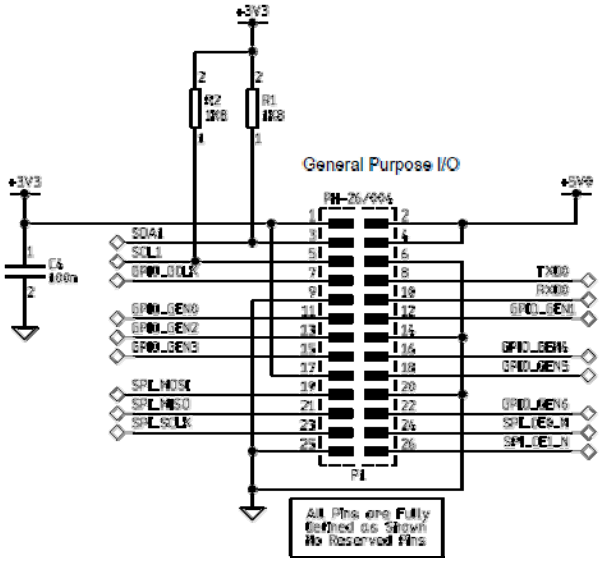
Die Schnittstelle P6 bietet die Möglichkeit eines Hardware-Rests an. Darüber wird also die CPU neugestartet [vgl. Kof2015 S.365]. Dazu müssen die beiden Kontakte miteinander kurzgeschlossen werden [vgl. Kof2015 S.1032].

Die GPIO Schnittstelle mit der Bezeichnung P4 ist mit der RJ45 bzw. Ethernet Buchse und dem LAN-Controller des LAN9512 verbunden. Daher wird diese Schnittstelle beim Raspberry Pi Modell B für Ethernet Anschluss verwendet und ist beim Modell B standardmäßig mit dem jeweiligen Port belegt [vgl. Kof2015 S.366]. Dieser wird zur Übertragung bzw. zur Visuallisierung der abgespeicherten Verbrauchsdaten benutzt.

Die GPIO-Schnittstelle mit der Bezeichnung P1 stellt die Basis für weitergehende Projekte dar. Durch die GPIO-Schnittstelle kann der Raspberry Pi mit Schaltungen, Platinen usw. um Funktionalitäten erweitert werden. In diesem Kapitel wird diese GPIO- Schnittstelle mit der Anschlussbezeichnung P1 näher beschrieben, da diese Schnittstelle nicht nur für die eigentlichen Erweiterungen, sondern auch in dieser Arbeit für die Erweiterung SD0 von Busware verwendet wird.

Die Schnittstelle selbst arbeitet intern mit einer Spannung von 3,3 Volt. Die Zufuhr von höheren Spannungen als 3,3 Volt oder der Kurzschluss der 5 Volt Pins, mit den anderen GPIO-Pins könnte den Raspberry Pi zerstören [vgl. Upt2014 S. 222].

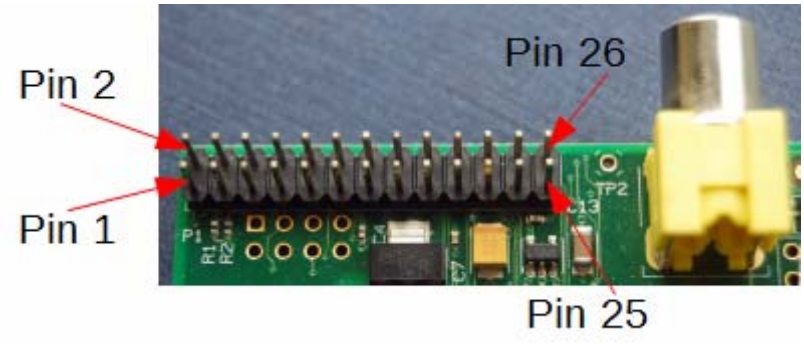
Abbildung 3.2: Schaltung des GPIO Steckerleiste von Raspberry Pi Model B



(Quelle: [vgl. Rpi Seite 2]; Modifiziert bzw. Kopiert)

Beim Raspberry Pi erfüllt jede GPIO Pin eine bestimmte Aufgabe. Einige Pins haben Doppelfunktionen. Alle Signale werden vom BCM2835 bereitgestellt. Die GPIO-Schnittstelle des Raspberry Pi Modell B, welches sich in der Abbildung 3.1 auf der oberen linken Ecke des Raspberry Pis befindet, besteht aus zwei Reihen mit jeweils 13 Pins.

Abbildung 3.3: Nummerierung der GPIO-Schnittstelle



(Quelle: [vgl. Rpi pins] Modifiziert)

Diese Pins besitzen einen Abstand von einem Inch, also 2,54 mm zueinander [vgl. Upt2014 S. 220].

Die Pins auf der einen Seite sind mit ungeraden Zahlen durchnummeriert, auf der anderen Seite mit geraden Zahlen durchnummeriert.

Abbildung 3.4: Pin-Belegung der GPIO Steckleiste des Raspberry Pi Modell B

Funktion/Signal	BCM - Bezeichnung		BCM - Bezeichnung	Funktion/Signal
3,3 Volt	-	1 2	-	5 Volt
I²C: SDA	GPIO2	3 4	-	5 Volt
I²C: SCL	GPIO3	5 6	-	Masse (0 Volt)
GPIO_Clock	GPIO4	7 8	GPIO14	UART: TXD
Masse (0 Volt)	-	9 10	GPIO15	UART: RXD
GPIO_GEN0	GPIO17	11 12	GPIO18	GPIO_GEN1 / PWM
GPIO_GEN2	GPIO27	13 14	-	Masse (0 Volt)
GPIO_GEN3	GPIO22	15 16	GPIO23	GPIO_GEN4
3,3 Volt	-	17 18	GPIO24	GPIO_GEN5
SPI: MOSI	GPIO10	19 20	-	Masse (0 Volt)
SPI: MISO	GPIO9	21 22	GPIO25	GPIO_GEN6
SPI: CLK	GPIO11	23 24	GPIO8	SPI: CE0
Masse (0 Volt)	-	25 26	GPIO7	SPI: CE1

GPIO Pin Masse 5 Volt 3,3 Volt

(Quelle: [vgl. Dem2013, S. 76 – 77 und Upt2014 S.221 und Kof2015 S. 361 und Rpi num] Modifiziert)

Von den 26 Pins sind 5 Pins frei verfügbar, das heißt, es handelt sich dabei um echte bzw. universelle GPIO Pins. Diese Pins können alternativ entweder als Ein- oder als Ausgang konfiguriert werden. Dabei handelt es sich um die Pins 11, 15, 16, 18 und Pin 22 [vgl. Dem2013, S. 76 ff.]. Diese Pins können also entweder den Logischen Zustand HIGH oder LOW haben. Dies bedeutet, dass diese Pins bei High, die

Betriebsspannung unter der die Schnittstelle läuft, also 3,3 Volt, liefern. Beim Low-Zustand dient der jeweilige Pin als Masse bzw. 0 Volt [vgl. Upt2014 S. 222].

Neben den GPIO-Pins werden auch Pins mit zwei verschiedenen Versorgungsspannungen zur Verfügung gestellt, die für Erweiterungen benutzt werden können. Neben 3,3 Volt, an den Pins 1 und 17, wird auch 5,5 Volt an den Pins 2 und 4 zur Verfügung gestellt [vgl. Kof2015 S.360]. Dabei dürfen die Pins 1 und 3 zusammen nicht mit mehr als insgesamt 50 Milliampere belastet werden. Die Pins mit einer Versorgungsspannung von 5 Volt besitzen über poly fuse, also selbstrückstellende Sicherungen. Das bedeutet, dass bei einer zu starken Strombelastung der Raspberry Pi heruntergefahren und nach einer Weile wieder hochgefahren wird. Bei den Pins 6, 9, 14, 20 und 25 handelt es sich um Masse-Leitungen. Bei Verwendung zur Steuerungsaufgaben (bei Verwendung der jeweiligen Pins als Ausgänge) darf jeder Pin mit maximal acht Milliampere bzw. 50 Milliampere für die gesamte GPIO-Schnittstelle, die Pins eins und sieben mit eingeschlossen, belastet werden [vgl. Kof2015 S.363].

An den GPIO-Schnittstelle befinden sich auch die Funktionsgruppen der Bussysteme wie SPI, I²C und UART. Bei UART werden lediglich die Signale TXD und RXD zur Verfügung gestellt. Über die Pins 3 und 5 können I²C-Komponenten eingesetzt werden. Beide Pins, die mit einem 1,8 Kiloohm Pull-Up Widerstand angeschlossen sind, eignen sich als Eingänge [vgl. Kof2015 S. 364].

Der Pin 7 kann als Clock, also als Taktgeber verwendet werden. Der Pin kann auch für den 1Wire-Bus benutzt werden [vgl. Kof2015 S. 364]. Dieser Pin kann ebenfalls als ein universeller GPIO-Pin eingesetzt werden, wie es weiter oben beschrieben wurde [vgl. Dem2013 S. 78].

An den Pins 8 und 10 wird standardmäßig eine serielle Schnittstelle, der UART, zur Verfügung gestellt. Bei einer Verbindung über den Bussystem UART werden die Daten von Pin 8 gesendet, am Pin 10 können Daten empfangen werden [vgl. Upt2014 S. 222].

Die Pins 12 und 13 können, wie der Pin 7, als universelle GPIO –Pins eingesetzt werden. Alternativ kann der Pin 12 als PWM bzw. PCM Taktleitung und der Pin 13 als PCM Datenleitung benutzt werden [vgl. Dem2013 S. 78].

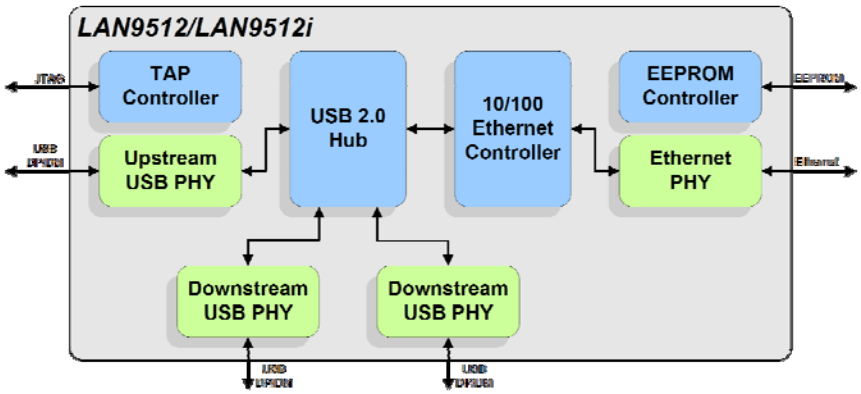
Zusätzlich gibt es die Möglichkeit die Pins 11, 12 und 13 als zweite SPI- Schnittstelle (SPI-Kanal 1) zu benutzen [vgl. Kof2015 S.364].

Die letzten Pins mit den Nummerierungen 19, 21, 23 ,24 und 26 werden als SPI-Bus , als SPI-Kanal 0,benutzt. Die Pins 24 und 26 dienen als Chip Select-Signale, das bedeutet, dass an diesen beiden Pins zwei Chips angeschlossen werden können, die mit SPI ausgestattet sind [vgl. Kof2015 S. 364]. Der Pin 19 wird beim SPI, zum Senden von Daten benutzt und Pin 21 zum Empfangen. Dabei dient der Pin 23 als Clock [vgl. Kof2015 S. 456].

3.1.7 LAN9512

Das in dieser Arbeit eingesetzte Raspberry Pi Model B hat einen LAN9512. Der LAN9512 ist neben den SoC BCM2835 der zweite Chip auf dem Raspberry Pi Model B. Es beinhaltet verschiedene Einheiten, wie einen LAN-Controller und den USB Hub.

Abbildung 3.5: Prinzipschaltbild des LAN9512



(Quelle: [vgl. LAN9512, S. 6])

Anzumerken ist, dass der SoC BCM2835 als USB-Controller und der LAN9512 nur als USB-Hub mit zwei Ports dient [vgl. LAN9512, S. 6 und Dem2013, S. 81].

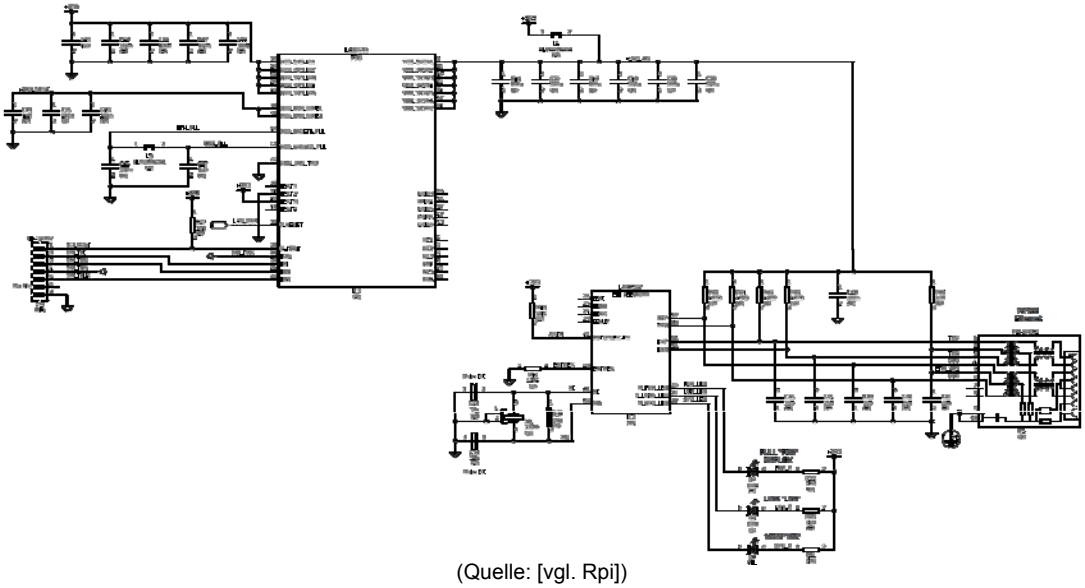
Der USB-Anschluss wird in dieser Arbeit nicht verwendet. Aber jedoch der Ethernetanschluss.

3.1.8 Ethernet

Der im LAN9512 eingesetzte Ethernet-Controller unterstützt die Ethernet-Standards 10Base-T mit 10Megabit/s, die im IEEE802.3 und 100Base-TX mit 100Megabit/s, welches im IEEE 802.3u spezifiziert sind [vgl. LAN9512 S.7 und Dem2013 S.84]. Zusätzlich besteht die Funktion „Wake on LAN“. Dies bedeutet, dass der Raspberry Pi über das angeschlossene Netzwerk aktiviert werden kann, wenn sich dieser im Ruhe Modus befindet.

100Base-TX unterstützt die automatische Detektion (Auto Negotiation) der Betriebsart. Es erkennt also die Verdrahtung des Ethernet-Kabels, das an der RJ45-Buchse angesteckt ist. Es wird auch Full-Dublex unterstützt, es kann somit gleichzeitig gesendet und empfangen werden [vgl. Dem2013 S.84].

Abbildung 3.6: Ethernet-Schaltung des Raspberry Pi Model B



(Quelle: [vgl. Rpi])

3.2 Datenbusse

Der Raspberry Pi Modell B stellt verschiedene Bussysteme zur Verfügung. Bussysteme, auch als Bus abgekürzt, werden für die Datenübertragung zwischen verschiedenen Teilnehmern benutzt [vgl. Kof2015 S. 455]. Je nach verwendetem Bussystem werden die Daten seriell, also hintereinander über dieselben Datenleitungen oder parallel, also gleichzeitig über verschiedenen Datenleitungen übertragen. Auf dem Raspberry Pi sind folgende fünf Bussysteme verfügbar, bei dem alle Daten seriell übertragen werden [vgl. Kof2015 S. 455],

- UART, Abk. für Universal Asynchronous Receiver/Transmitter.
- SPI, Abk. für Serial Peripheral Interface
- I²C, Abk. für Inter Integrated Circuit
- I²S, Abk. für Integrated Interchip Sound
- 1Wire, Eindraht-Verbindung.

Da in dieser Arbeit der UART und I²C verwendet werden, werden diese zwei Bussysteme beschrieben. Die vollständige Erläuterung der Bussysteme würde den Rahmen dieser Arbeit sprengen, daher werden nur die grundlegenden Punkte erläutert.

Die Bussysteme unterscheiden sich nicht nur, aber vor allem in der Datenübertragung. Dabei soll es zwei grundsätzliche Formen der Datenübertragung geben, die sich vor allem bei der Synchronisation durch einen Takt unterscheiden [vgl. Bei2011 S. 202 bis S. 203]:

Bei der synchronen Datenübertragung wird parallel zur Datenübertragung der Takt übertragen. Durch die Taktübertragung können die Kommunikationspartner den Beginn und das Ende einer Datenübertragung erkennen [vgl. Bei2011 S. 202 bis S. 203].

Bei der asynchronen Datenübertragung existiert keine separate Taktleitung. Die Daten werden als einzelne Zeichen übertragen. Dabei werden diese in Start- und Stoppbits verpackt. Die Start- und Stoppbits können variieren [vgl. Bei2011 S. 203].

Der Vollständigkeit halber wird im Folgenden der 1Wire, SPI und I²S beschrieben.

Beim 1-Wire Bus handelt es sich um einen Eindraht-Bus, deshalb auch der Name des Bussystems [vgl. Kof2015 S.491]. Der ursprünglich vom Unternehmen Dallas (heute Maxim) [vgl. Gei S. 3] entwickelte 1-Wire-Bus besitzt im Vergleich zu SPI und I²C keine zusätzliche Taktleitung. Dabei handelt es sich also um eine asynchrone Datenübertragung. 1-Wire-Bus führt lediglich neben der einzelnen Leitung zur Buskommunikation, also der Datenleitung mit der Bezeichnung DQ, noch die Masse Leitung [vgl. Gei S. 3 und Kof2015 S.491]. Die Datenleitung wird nicht nur dazu benutzt, um Daten zu senden und zu empfangen, sondern auch um die Erweiterungen mit einer Versorgungsspannung von 3,3 Volt zu versorgen [vgl. msx und Kof2015 S.491]. Die Übertragung erfolgt dabei seriell, bidirektional und asynchron, da über die Datenleitung entweder nur gesendet oder empfangen werden kann und dies auch ohne Taktleitung erfolgt [vgl. Gei S. 3].

Der Raspberry Pi bietet ab der Revision 2 den Soundbus I²S, Abk. für Integrated Interchip Sound. Der I²S Bus transportiert Audiosignale zwischen integrierten Schaltkreisen. [vgl. Kof2015 S.489].

Das ursprünglich von Motorola entwickelte Bussystem Serial Peripheral Interface (SPI) arbeitet nach dem Master-Salve-Prinzip, mit einem Master und einer theoretisch unbegrenzten Anzahl an Slaves. Dabei erfolgt die Kommunikation Synchron. [vgl. Bei 2011 S. 208]

3.2.1 UART

Der Universal Asynchrounous Receiver/Transmitter (kurz UART) Protokoll wird bei der Kommunikation von RS232- oder RS485-Schnittstellen eingesetzt. Der UART wird auch zur Datenübertragung mit Mikrocontrollern und wie hier mit Einplatinen-Computern benutzt.

Die Daten werden beim UART, im Gegensatz zum USART, der eine synchrone Datenübertragung erlaubt, asynchron übertragen. Das heißt dass es keine Taktleitung gibt. Der UART besitzt im Ruhezustand einen High-Pegel, was der logischen 1 entspricht [vgl. Pla uart]. Die Daten werden mit einem Start- und Stoppbit versehen. Es kann zusätzlich einen Paritätsbit geben. Üblicherweise werden acht bis neun Bits übertragen [vgl. MUA]. Dabei werden also insgesamt 10 bis 12 Bits zusammen übertragen. Diese werden auch als Wort bzw. Datenwort bezeichnet. Die Reihenfolge der übertragenen Bits sieht beim UART folgendermaßen aus:

Abbildung 3.7: Aufbau des UART-Datenwortes

Startbit	D0	D1	D2	D3	D4	D5	D6	D7	(D8)	(Paritätsbit)	Stoppbit

(Quelle: [vgl. MUA]; modifiziert)

Bei der obigen Abbildung zum Aufbau des UART-Datenwortes ist jedes Kästchen für ein Bit gedacht. Der Startbit wird als eine Logische 0 gesendet [vgl. Pla uart]. Zwischen zwei Datenworten können sich beliebig lange Pausen befinden, da jedes Wort am Startbit erkannt wird [vgl. Pla uart]. Die Bits D8, also der neunte Datenbit und die Paritätsbits sind gelb hintermalt, da diese optional sind.

Das Paritätbit überprüft, ob die Daten richtig übertragen wurden. Dabei wird zwischen odd parity und even parity unterschieden. Bei odd parity wird der Paritätsbit auf eins gesetzt, wenn es in den Datenbits eine gerade Anzahl an Einsen gibt [vgl. Mül2009 S. 6].

Even parity bildet den Gegensatz zur odd parity. Bei einer geraden Anzahl von Datenbits wird das Paritätsbit auf Null und bei einer ungeraden Anzahl auf eins gesetzt [vgl. Mül2009 S. 6].

Beim Stoppbit muss erwähnt werden, dass auch zwei Stoppbits hinter ein Datenwort eingefügt werden können, um den Pausenpegel zwischen zwei Datenworten bei der Übertragung zu trennen [vgl. Bei2011 S. 262]. Da die Kommunikation beim UART asynchron abläuft, synchronisieren sich die Kommunikationsteilnehmer für jeden Transfer neu [vgl. Pla uart]. Dies geschieht durch die oben erwähnten Start- und Stoppbits. Vor der Datenübertragung liegt der Pegel auf der Übertragungsleitung auf high [vgl. Pla uart]. Das bedeutet, dass die Leitung sich auf 3,3 Volt befindet.

Vor der Kommunikation zwischen Sender und Empfänger müssen diesen die Anzahl der Datenbits, Startbits und der Stoppbits sowie der Berechnung der Parität und die Frequenz des Übertragungstaktes der Daten kenntlich gemacht werden. Soll etwas übertragen werden, wird dies dem Empfänger der Daten durch den Startbit kenntlich gemacht. Anhand des High-Low- Pegels wird der Sendevorgang kenntlich gemacht, wodurch Empfänger und Sender synchronisiert sind. Der Sendevorgang wird durch einen Stoppbit beendet [vgl. Pla uart].

Der UART am Raspberry Pi besitzt die Leitungen TxD (am Pin 8) zum Senden und RxD (am Pin 10) zum Empfangen der Daten [vgl. Upt2014 S. 222]. Das Raspberry besitzt keine Leitung für einen Hardware-Handshake [vgl. Pla uart], zur Erkennung der Übertragung. Wird ein hardwarebasierter Handshake benötigt, kann auf einen freien GPIO-Pin zugegriffen werden [vgl. Pla uart].

Ein Baudratengenerator bzw. programmierbarer Timer sorgt für die Übertragungsgeschwindigkeit, auch Baud oder Baudrate genannt, der Daten [vgl. Bei2011 S. 262]. Die Baudraten müssen sowohl auf Empfängerseite als auch auf der Senderseite übereinstimmen, da die Daten sonst nicht korrekt übertragen werden können. Das hat den Hintergrund, dass der UART – wie oben erwähnt – keine Taktleitung hat und der Empfänger nicht weiß, wann über UART Daten geschickt werden und wann nicht.

3.2.2 I²C

Das im Jahre 1982 von Philips Semiconductor entwickelte Bussystem Inter Intergratet Circuit (I²C) wird für die Verbindung von integrierten Schaltungen auf Platinen bzw. innerhalb von Geräten eingesetzt [vgl. Bei2011 S. 209 und Dem2013 S. 185]. Beim I²C handelt es sich um eine serielle Schnittstelle [vgl. Dem2013 S. 185], bei der die Datenübertragung sowohl synchron als auch asynchron erfolgt [vgl. Bei2011 263 bis S. 264].

Das Bussystem kann mit zwei bzw. drei Leitungen eingesetzt werden. Eine Leitung dient als Taktleitung, eine zweite als Datenleitung, wobei die dritte Leitung die Masseleitung ist, die als Bezugspegel dient [vgl. Bei2011 S. 209]. I²C besitzt keine separate Leitung zum Auswählen der einzelnen Teilnehmer. Die einzelnen Teilnehmer werden durch Adressen ausgewählt [vgl. Kof2015 S.476]. Beim Raspberry Pi wird der I²C Datenbus auch vom BCM2835, also dem SoC, bereitgestellt [vgl. Upt2014 S. 223]. Der I²C besitzt folgende Leitungen:

SDA Serial Data- Line, hierüber werden die Daten zwischen den Kommunikationspartnern übertragen
SCL Serial Clock Line, Taktleitung des Bussystems

Die Übertragung der Daten erfolgt aus einer Mischung aus synchroner als auch asynchroner Übertragung. Das bedeutet, dass bei der Übertragung der einzelnen Datenpakete diese jeweils mit Start und Stoppbits versehen werden [vgl. Bei2011 263 bis S. 264]. Die Übertragung der einzelnen Bits der Datenpakete wird jedoch mit dem Taktsignal der SCL-Leitung des jeweiligen Masters synchronisiert übertragen [vgl. Bei2011 S. 264].

An das Bussystem können ein oder mehrere Master angeschlossen werden, von denen jedoch mindestens einer die Kommunikation innerhalb des I²C steuert [vgl. Bei2011 S. 209]. Ein Bus, der mehrere Master besitzt, wird auch als Multimasterbus bezeichnet [vgl. Dem2013 S. 187]. Am Bus können bis zu 128 Slaves angeschlossen werden, die eine im Bussystem eindeutige Adresse von sieben bzw. zehn Bit Länge besitzen [vgl. Dem2013 S. 189], wodurch die Slaves einzeln angesprochen werden können [vgl. Bei2011 S. 209 bis 210]. Die 10 Bit Adressierung soll kaum genutzt werden [vgl. Dem2013 S. 189]. Die Slaves können dadurch immer zum Senden bzw. Empfangen von Daten durch den Master aufgefordert werden. Der Empfang der Aufforderung wird dem Master durch ein Quittierungsbit (auch Acknowledge bezeichnet) bestätigt [vgl. Bei2011 S. 210].

Da I²C-Busse in der Regel Open-Collector-Eingänge besitzen, versorgen die I²C-Bausteine keine Spannungspegel an ihren Pins. Beim Raspberry Pi werden die I²C

Pins jedoch durch Pull-Up-Widerstände auf 3,3 Volt gezogen, dieser bildet dann den positiven Signalpegel [vgl. Bei2011 210]. Bei der Kommunikation über die I²C-Schnittstelle werden diese Leitungen auf Masse gezogen [vgl. Kof2015 S.476], das bedeutet, dass diese Leitungen dann auf 0 Volt herunter gezogen werden und somit auch einen LOW-Pegel besitzen [vgl. Kof2015 S.476].

Abbildung 3.8: Prinzipielles Aufbau des I²C Datenformates

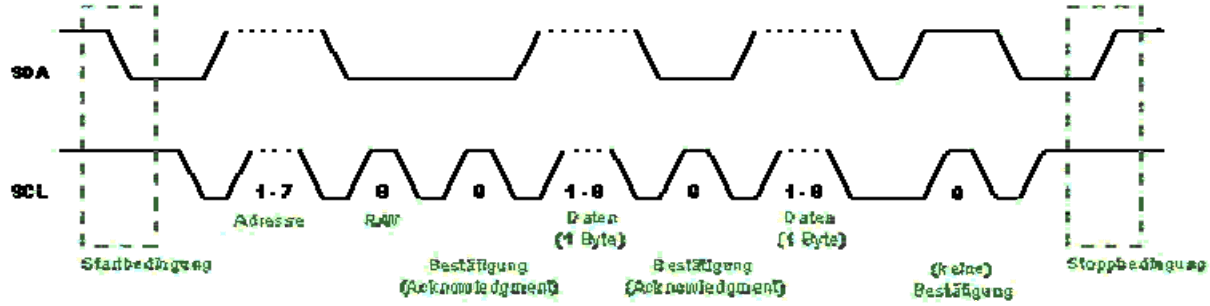
Start	Slave Adresse	Read/Write	Ackn.	Daten	Ackn.	Daten	Ackn.	Stop

(Quelle: [vgl. rni dat]; modifiziert)

In der Abbildung befindet sich der Startbit ganz links und der Stoppbit ganz rechts. Beim „Ackn." handelt es sich um den Quittierungsbit, was nach jeder Datenübertragung vom Slave an den Master erfolgt.

Beim I²C kann jede Baugruppe, je nach Funktion, entweder als Sender oder als Empfänger arbeiten. Dabei können mehrere Master im System existieren. Die Erzeugung des Taktes erfolgt immer durch den Master, wobei jeder Master seine eigene Taktfrequenz besitzt [vgl. Dem2013 S. 187 bis S. 188].

Abbildung 3.9 Prinzipielles Beispiel der vollständigen Kommunikation desI²C



(Quelle: [vgl. rni dat])

Da der Bus vom Raspberry Pi mit einer Spannung von 3,3 Volt versorgt wird, befinden sich die Leitungen SDL und SCL im High-Pegel wenn keine Busaktivität stattfindet [vgl. Kof2015 S.476]. Es hat also den logischen Wert 1. Sollen Daten übertragen werden geht das Datensignal SDA von High auf Low über, gleichzeitig befindet sich jedoch die Taktleitung auf High. Dabei handelt es sich um den Startbit. Der Startbit ist in der Abbildung 3.8 auf der linken Seite umrahmt. Im Gegensatz dazu bildet der Stoppbit einen Low zum High-Übergang des SDL bei gleichzeitigem High Pegel der Taktleitung. Dieser ist in der Abbildung 3.8 auf der rechten Seite umrahmt [vgl. rni].

Mit der Slave Adresse, die in der Regel sieben Bit groß ist, wird das jeweilige Slave Bauelement des I²C angesprochen. Der Read/Write Bit gibt die Richtung der Datenübertragungen an. Bei Read (logische 1), liest der Slave ein und bei Write (logische 0), schreibt der Slave. Die Richtung der Datenübertragung wird vom Slave an den Master gemeldet [vgl. rni und Dem2013 S. 189].

Bei der Kommunikation über den I²C erfolgt die Bestätigung, also der Quittierungsbit (in der Abbildung 3.8 als „Ackn." dargestellt), für die Übertragungsrichtung durch den Slave. Bei der Datenübertragung an sich bestätigt der Empfänger der Daten dem Sender den Empfang der Daten. Falls der Slave die Daten empfängt, setzt der Master den SDA auf High und erwartet vom Slave, dass dieser den SDA auf Low

zieht. Geschieht das nicht, wird die Übertragung gestoppt. Im umgekehrten Fall, wenn der Master der Empfänger ist, setzt der Slave den SDA auf High, falls es nicht bestätigt wird, erfolgt der Abbruch der Datenübertragung [vgl. Dem2013 S. 189]. Die Bestätigung muss innerhalb von neun Takten des jeweiligen Masters erfolgen [vgl. rni]. Bei der Datenübertragung an sich muss der Takt selbst High sein, wenn ein Bit übertragen wird und dieser als gültig gelten soll [vgl. rni].

Auf dem Raspberry Pi existieren zwei I²C-Schnittstellen [vgl. Upt2014 S. 223]. Der erste befindet sich in der GPIO-Schnittstelle mit der Bezeichnung P1. Der zweite befindet sich an der GPIO Schnittstelle P5, welcher nicht für den allgemeinen Gebrauch bestimmt ist [vgl. Upt2014 S. 223]. Der Aufbau der zweiten I²C Schnittstelle wird am Anfang der Kapitels GPIO-Anschlüsse beschrieben. Die erste I²C Schnittstelle kann über die GPIO-Schnittstelle mit der Bezeichnung P1 über zwei Pins benutzt werden. Dazu dient der Pin 3 als SDA, also als Datenleitung und der Pin 5 als SCL, also um den Takt für die Synchronisation der Kommunikation zu Übertragen. Die Pins sind mit zwei Pull-Up-Widerständen verschaltet [vgl. Dem2013 S. 188].

Der I²C arbeitet nach dem Master-Slave Prinzip. Beim Master handelt es sich um die aktive Komponente, beim Slave um die passive. Der Master steuert also die gesamte Kommunikation, sodass die Slaves lediglich auf die Anfragen des Masters ansprechen müssen.

3.3 Beschreibung des Aufsatzes SD0

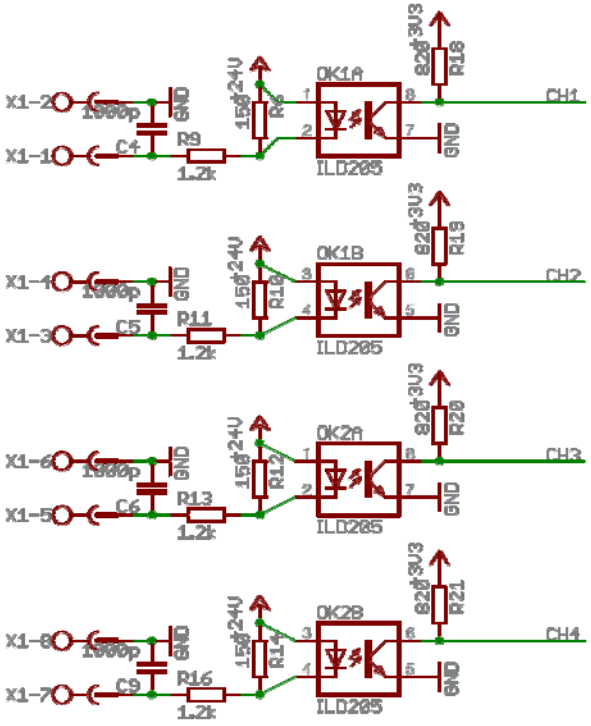
Beim Aufsatz handelt es sich um eine Erweiterung für den Raspberry Pi. Aufsätze werden bei Raspberry Pi dazu verwendet, um diesen mit zusätzlichen Funktionalitäten auszustatten oder zu erweitern.

Bei der benutzten Erweiterung handelt es sich um den Aufsatz SD0 des Unternehmens Busware. Der Aufsatz SD0 wird auf die als P1 bezeichneten General Purpose Input Output (GPIO) Pins des Raspberry Pi aufgesetzt. Diese GPIO-Leiste des Raspberry Pi wird zur Erweiterung des Raspberry Pi verwendet. Bei dieser Bachelorarbeit werden über den Aufsatz SD0 Energieverbrauchsdaten über dessen S0-Schnittstellen erfasst. Der SD0 wird verwendet, da dieser durch die Aufgabenstelölung vorgegeben ist. In diesem Kapitel wird der Aufsatz SD0 beschrieben. Da der aktuelle Schaltplan nicht zur Verfügung steht, wird der SD0 basierend auf dem Schaltplan Version 1.1 beschrieben.

3.3.1 S0-Schnittstellen des SD0

Neben der Steckleiste, mit der der Aufsatz SD0 auf die GPIO Steckleiste mit der Bezeichnung P1 des Raspberrry Pi aufgesteckt wird, besitzt der Aufsatz vier S0-Schnittstellen, auch Chanels (engl. für Kanal) genannt, über die die Verbrauchsdaten erfasst werden.

Abbildung 3.10: S0-Kanäle des SD0



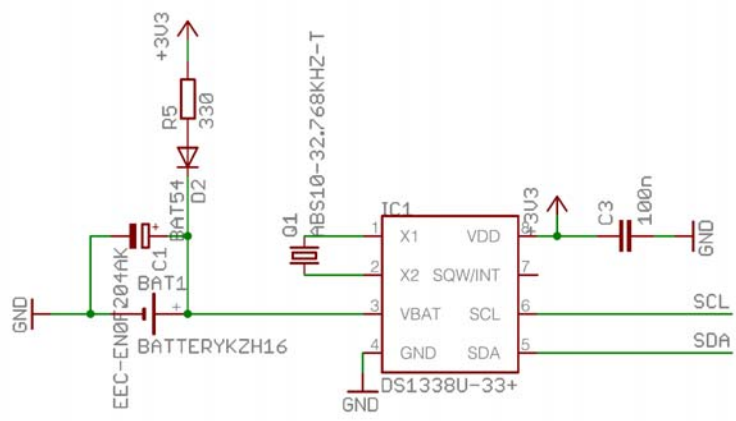
(Quelle: [vgl. SD0s]; modifiziert)

Zusätzlich befindet sich eine RJ-10-Schnittstelle auf dem SD0. Daneben ist die Erweiterung mit einem Mikrocontroller, dem Atmega1284p bestückt. Dieser ist mit der seriellen Schnittstelle RJ-10-Schnittstelle, den vier S0-Schnittstellen nach DIN 43864 / EN 62053-31 und der Steckverbindungsleiste verbunden. Zusätzlich besitzt die Erweiterung eine Real Time Clock (RTC), die für Echtzeitanwendungen oder hier dem Raspberry Pi zu gute kommen kann [vgl. bsd0 Kapitel Specs].

3.3.2 Real Time Clock (RTC)

Bei Real Time Clock, kurz RTC bezeichnet, handelt es sich um Echtzeituhren die Systemzeiten bereitstellen. RTCs stellen somit Datum und Uhrzeit bereit [vgl. Bei2011 278]. Beim Real Time Clock des SD0 handelt es sich um den DS1338U33. Da der Raspberry Pi ohne einen Internetzugang seine Uhrzeit nicht abgleichen kann, bietet sich die Möglichkeit an, die Uhrzeit beim RTC des SD0 zu beziehen, wobei der RTC durch den Atmega1284p dem Rasperry Pi bereitgestellt wird.

Abbildung 3.11: Echtzeituhr des SD0



(Quelle: [vgl. SD0s]; modifiziert)

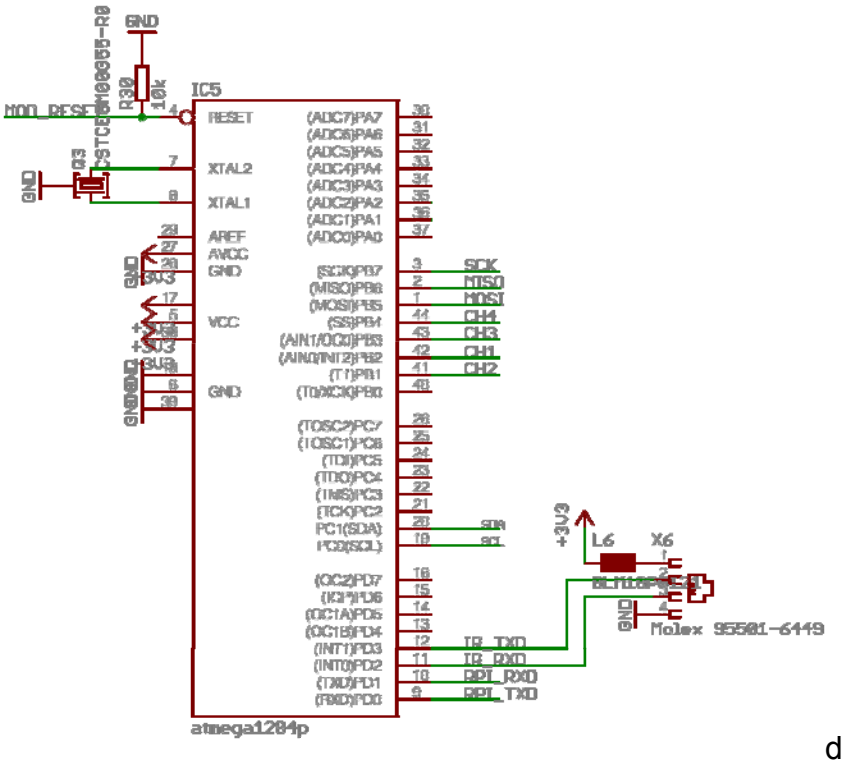
Die Verbindung zur Echtzeituhr wird über eine I²C Verbindung erstellt. Dafür stehen, wie die in der Abbildung 3.11 dargestellten SCL und SDA bereit, welche jeweils als Takt- und Datenleitung des I²C verwendet werden. Es bietet sowohl die Uhrzeit in den Einheiten Sekunden, Minuten, Stunden als auch den Tag, Monat und Jahr als Datum an. Die Uhrzeit wird entweder im 12- oder 24 Stunden-Format bereitgestellt. Beim 12-Stunden-Format indiziert es die AM/PM [vgl. DS1338 S. 1 General Description]. Die Monatsenden werden bei Monaten mit weniger als 31 Tage automatisch angepasst. Es beinhaltet auch die Erkennung von Schaltjahren. Der RTC DS1338 besitzt einen Stromerkennungssensor und schaltet auf einen 220 mF Kondensator um, wenn es mit zu wenig Strom versorgt wird [vgl. DS1338 S. 1].

Die RTC wird dazu verwendet, um die Uhrzeit des Raspberry Pi zu setzen, da der Raspberry Pi nur bei Internetverbindungen seine Uhrzeit setzen kann.

3.3.3 Mikrocontroller des SD0

Neben dem im vorherigen Kapitel beschriebenen RTC ist der SD0 mit einem Mikrocontroller bestückt. Bei dem 8-Bit Mikrocontroller handelt es sich um den Atmega1284p. Der 8-Bit Mikrocontroller basiert auf dem AVR [atm1284 S. 3]. Über das Atmega1284p werden die an den S0-Schnittstellen empfangenen Daten aufbereitet und an den Ausgängen bereitgestellt [vgl. bsd0 Kapitel Specs].

Abbildung 3.12: Mikrokontroller des SD0



d

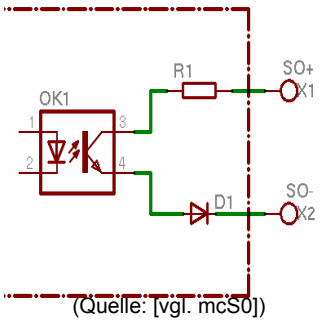
(Quelle: [vgl. SD0s]; modifiziert)

Wie der Raspberry besitzt der Atmega 1284p eine GPIO Schnittstelle, beim Atmega1284p besitzt sie jedoch 32 Pins, von denen nur 26 Pins auf dem SD0, als Steckleiste für den Anschluss P1 des Raspberry Pis, bereitstehen. Durch den Atmega1284p wird auch das RTC bereitgestellt. Der Mikrocontroller kann über zwei USART und eine SPI Schnittstelle kommunizieren [vgl. atm1284 S. 4].

3.4 S0- Schnittstelle

Die S0-Schnitstelle wird nach DIN EN 62053-31 definiert und in der Gebäudeautomatisierung verwendet. Die Schnittstelle wird in Elektrizitätszählern eingebaut. Mit Impulseinrichtungen wie die S0-Schnittstelle werden Impulse, die der aktuell verbrauchten Energiemenge entsprechen, an den Empfänger übertragen [s0 norm S.4].

Abbildung 3.13: S0-Schnitstelle in Stromzählern



(Quelle: [vgl. mcS0])

Die Impulseinrichtungen geben eine bestimme Anzahl von Impulsen pro Wattstunde. Dabei gibt es zwei Arten bzw. Klassen von Impulsausgängen [s0 norm S.5]:

Impulsausgang der Klasse A: Übertragung über größere Entfernungen;

Impulsausgang der Klasse B: geringe Entfernung und geringer Energieverbrauch

Dabei kann die Schnittstelle sowohl im Stromzähler als auch im Gas- oder Wasserzähler eingesetzt werden. Bei der Übertragung von Messwerten wird für jedes verbrauchtes Watt pro Stunde (Wph) ein gewichteter Impuls übertragen. Die Gewichtung des Impulses ist von Zähler zu Zähler unterschiedlich [vgl. S0S]. Für die zwei Arten bzw. Klassen von Impulsausgängen sind den Impulsen verschiedene Grenzen gesetzt:

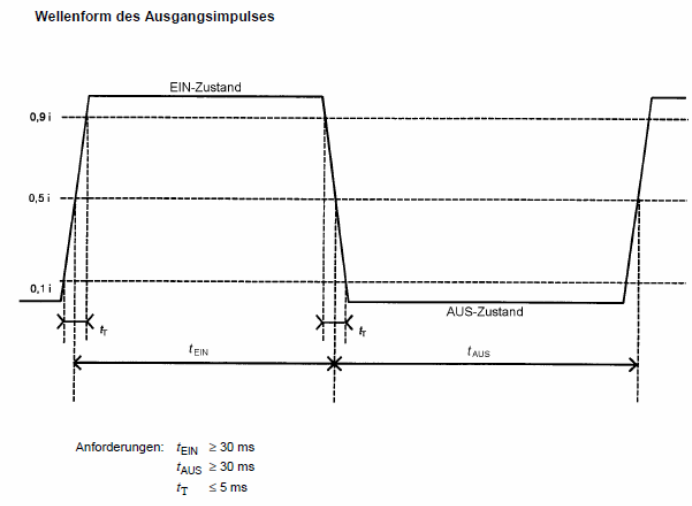
Tabelle 3.4: Betriebsbedingungen der Impulseinrichtungen

Parameter	Impulseinrichtung der Klasse A	Impulseinrichtung der Klasse B
Maximale Spannung (U_{max})	27 V DC	15 V DC
Maximaler Strom im EIN-Zustand	27 mA	15 mA
Minimaler Strom im EIN-Zustand	10 mA	2 mA
Maximaler Strom im Aus-Zustand	2 mA	0,15 mA

(Quelle: [vgl. s0 norm S.5])

Die Impulse können zwei Zustände haben: High- bzw. Ein- Zustand und Low- bzw.- Aus-Zustand, für zwei aufeinanderfolgende Impulse.

Abbildung 3.14: S0-Schnitstelle in Stromzählern



(Quelle: [vgl. s0Imp])

Für die Zustände sind die folgenden Impulslängen definiert:

Ein Impuls, im High- bzw. Ein- Zustand, ist mit $t_{EIN} \geq 30$ ms, also größer gleich 30 ms definiert und der

Aus- bzw- Low-Zustand, für die Zeit zwischen zwei aufeinanderfolgenden Impulsen, mit $t_{AUS} \geq 30$ ms, also größer gleich 30 ms

Dabei muss die Übergangszeit der Anstiegs- oder Abfallzeit zwischen den einzelnen Zuständen, also der Zustandswechsel 5 ms sein.

3.5 Datendarstellung

Bei der Datenhaltung, also der Datenspeicherung, gibt es verschiedene Möglichkeiten: die Daten können beispielsweise als CSV, also als Comma Separated Values, abgespeichert werden. Es gibt jedoch auch die Möglichkeiten die Daten in JSON oder aber auch in SQL abzulegen. Die Daten in SQL abzuspeichern lohnt sich erst, wenn die Daten auch lokal auf dem Raspberry Pi weiterverarbeitet werden sollen. Dabei muss auch bedacht werden, dass ein SQL-Server benötigt wird. Dies würde dazu führen, dass durch den SQL-Server mehr Ressourcen verwenden würden. Das heißt ein SQL-Server würde nicht nur Speicher auf der SD-Karte einnehmen, sondern auch, wenn der Server läuft, den Arbeitsspeicher und die CPU beanspruchen.

Beim JSON ist bei der Programmierung des Quellcodes, die auf dem Raspberry Pi läuft, nur ein Libary nötig, der die Daten in eine. JSON-Datei schreibt. JSON wird meist im Web Bereich eingesetzt.

3.5.1 Comma Separated Values (CSV)

Beim Comma Separated Values (CSV)-Format handelt es sich nicht um einen allgemeinen Standard oder bestimmte Spezifikation, sondern es kann in unterschiedlichen Spezifikationen eingesetzt werden. Das CSV-Format kann dann eingesetzt, wenn die abgespeicherten Daten unabhängig von der Technologie eingesetzt werden sollen. Also wenn die Daten beispielsweise in einem Tabellenprogramm dargestellt und in einem Quellcode bearbeitet werden sollen [vgl. RFC4180].

Die Daten können entweder durch einfache Kommata voneinander getrennt in der .csv Datei abgelegt werden, die Werte an sich können zusätzlich in Anführungsstriche bzw –zeichen gesetzt werden. Genauso könnten die einzelnen Daten mit Semikolon voneinander getrennt abgespeichert werden [vgl. RFC4180]. Zu beachten ist jedoch, dass bei Kommazahlen im deutschen ein Komma benutzt wird und im englischsprachigen Raum ein Punkt. Dies könnte evtl. zu Problemen beim Lesen bzw. Parsen der Datei im Quellcode führen.

3.5.2 JavaScript Object Notation (JSON)

JavaScript Object Notation, kurz JSON, ist ein Datenaustauschformat. Dabei handelt es sich um ein von Programmiersprachen unabhängiges Textformat. Daten können also programmiersprachenunabhängig ausgetauscht werden. JSON folgt jedoch der JavaScript Notation. Dabei handelt es sich bei JSON um eine Untermenge der Skriptsprache JavaScript [vgl. ecma2013].

Das Datenaustauschformat basiert dabei auf zwei Strukturen [vgl. ecma2013]:

- *Name-Wert-Paare*

Dabei ist jedem Namen bzw. Bezeichnung mindestens ein Wert zugewiesen.

- *Geordnete Liste von Werten*

Zusätzlich gibt es in JSON, wie auch in Programmiersprachen, Typen. Die Typen beziehen sich auf die Werte. Die folgende Auflistung mit entsprechenden Zeichen [vgl ecma2013]

- *Objekte*

Beinhaltet eine ungeordnete Menge an Name-Werte-Paare. Es beginnt mit einer öffnenden geschweiften Klammer „{“ und einer schließenden geschweiften Klammer „}“. Die Name-Werte-Paare sind mit einem Doppelpunkt getrennt [vgl ecma2013]:

Name:Wert

Jedes Paar ist wiederum zueinander mit Kommata (,) voneinander getrennt [vgl ecma2013].

- *Arrays*

Ein Array beinhaltet eine geordnete Liste von Werten, die einem Namen zugewiesen werden. Arrays beginnen mit einer [(öffnenden eckigen Klammer) und enden mit einer] (schließenden eckigen Klammer). Die Werte an sich sind mit , (Komma) zu einander getrennt [vgl ecma2013].

- *Werte*

Werte werden durch Objekte, Arrays, Zeichenketten (Strings), Zahlen, den Wahrheitswerten true und false oder Null gebildet [vgl ecma2013].

- *Zeichenketten*
-

Strings, also Zeichenketten, sind Unicode Zeichen [vgl ecma2013].

- *Nummern*

Bei den Zahlen in JSON handelt es sich um Dezimalzahlen. Den Zahlen kann ein – (Minus) vorangestellt werden. Gefolgt von entweder einem 0 (Null) oder einer beliebigen Zahl von 1 bis 9 in beliebiger Weiderholung. Gleitkommerzahlen werden mit einem Punkt getrennt [vgl ecma2013].

Im Folgenden ist ein beispielhafter Aufbau einer JSON-Datei dargestellt:

Listing 3.1: Beispielhafter Aufbau einer JSON Datei

```
{
        “books”: [
                “book”: {
                        “language”:”Java”,
                        “edition”:”second”,
                        “price”: 10,
                        “author”: [“Hans”, ”Peter”, “Müller”]
                },
                “book”: {
                        “language”:”C++”,
                        “lastName”:”fifth”
                        “price”: 11.12,
                        “author”: [“Stroustroup”]
                },
                “book”: {
                        “language”:”C”,
                        “lastName”:”third”
                        “price”: 8,
                        “author”: [“Georg”, ”Petrus”]
                }
 ]
```

(Quelle:[vgl. tpj]; modifiziert)

Bei Books handelt es sich um ein Objekt, das verschiedene book-Objekte besitzt. Books ist ein Array, was an den eckigen Klammern ([und]) zu erkennen ist.

Die Book ist an sich auch ein Objekt, das wiederum kein Array als Wert besitzt, sondern Namen-Wert Paare.

In jeder Zeile eines jeden Book Objekts befindet sich ein Name-Wert-Paar. Das heißt, wie bei allen drei book Objekten zu sehen ist, besitzen diese die gleichen Namen. Das sind language, edition, price und author. Was nach dem Doppelpunkt (:) folgt sind dann die Werte.

4 Anforderungsdefinition

In diesem Kapitel werden die Anforderungen an die Software erfasst, die das zu entwickelnde Software erfüllen soll.

4.1 Zielbeschreibung

Für das Monitoring von Energieverbrauchswerten soll eine Softwarelösung entwickelt werden. Es soll über die S0-Schnittstellen von Zählern Energieverbrauchswerte erfassen, anschließend verarbeiten und lokal abspeichern.

Die abgespeicherten Verbrauchswerte sollen an einen Server gesendet werden.

Die Visualisierung der Verbrauchswerte erfolgt grafisch über eine Weboberfläche. Dabei werden die Energieverbrauchswerte entweder für einen bestimmten, vom Benutzer ausgewählten, Zeitraum abgefragt. Oder die aktuellen Verbrauchswerte werden fortlaufend von der Messeinheit, also vom Aufsatz SD0 und dem Raspberry Pi abgefragt und auf dem Raspberry Pi dargestellt.

Die Verbrauchswerte werden über die S0-Schnittstellen des Aufsatzes SD0 gemessen. Die gemessenen Daten werden anschließend vom Aufsatz über die serielle Schnittstelle, also dem UART, des Raspberry Pi an diesen übertragen. Auf dem Raspberry Pi werden die S0-Impulse verarbeitet und anschließend mit dem jeweiligen Zeitstempel des Zeitpunktes der Messung versehen und abgespeichert.

Die abgespeicherten Verbrauchswerte werden anschließend an einen Server weitergeleitet, wo die Verbrauchswerte für einen, vom Benutzer bestimmten, Zeitraum grafisch dargestellt werden. Der Server kann sich physisch an einem anderen Ort als der Raspberry Pi befinden. Dabei erfolgt die Kommunikation über die Ethernet Schnittstelle. Alternativ kann die Webdarstellung auf dem Raspberry erfolgen. Für die fortlaufende grafische Darstellung des Energieverbrauchs muss sich die Webdarstellung auf dem Raspberry Pi befinden.

4.2 Muss-Kriterium

M – ERF010: Erfassen der S0-Impulse am Zähler
Durch S0-Impulse soll der Verbrauch am Stromzähler erfasst werden.

M – ERF011: Anzahl der verschiedenen, gemessenen Stromzählern
Mit dem Aufbau muss möglich sein, an bis zu vier verschiedenen Stromzähler Stromzählern mit S0-Schnittstellen die S0-Impulse erfasst zu können.

M-ERF020: Übergabe der erfassten S0-Impusle an den Raspberry Pi
Die erfassten S0-Impulse werden nach dem Messvorgang dem Raspberry Pi über die GPIO-Schnittstelle übergeben.

M-VERA030: Umrechnung der übergebenen S0-Impusle
Die gemessenen S0-Impusle, die an den Raspberry Pi übergeben wurden, müssen auf dem Raspberry Pi in Energieverbrauchswerte (Watt pro Stunde) umgerechnet werden.

M-VERA040: Abspeichern von umgerechneten Energieverbrauchswerten.
Die von S0-Impuslen nach Energieverbrauchswerte umgerechneten Werte müssen auf dem Raspberry Pi abgespeichert werden.

M-SEN050: Senden der abgespeicherten Energieverbrauchsdaten
Alle abgespeicherten Dateien sollen an einen Server versendet werden können.

M-SEN050-10: Zeiten für das Senden der abgespeicherten Energieverbrauchsdaten
Die abgespeicherten Dateien sollen zu bestimmten Uhrzeiten an einen Server versendet werden können.

M-SEN050-11: Zeiten für das Senden der abgespeicherten Energieverbrauchsdaten
Die abgespeicherten Dateien sollen permanent an einen Server versendet werden können.

M-BEA060: Betriebsart der Software
Die Software muss als Dienstprogramm laufen.

M-BEA070: Dauerhafter Betrieb der Software
Die Software muss dauerhaft auf dem Raspberry Pi laufen.

M-BEA080: Starten der Software
Die Software muss mit dem Systemstart der Raspberry Pi bzw. dessen Betriebssystems gestartet werden.

M-BEA090: Beenden der Software
Die Software muss vom Benutzer durch Angabe der Prozessnummer der Software beendet werden können.

M-BEA100: Neustart der Software
Die Software muss vom Benutzer neugestartet werden können.

M-KONF110: Einlesen einer Konfigurationsdatei
Die Software muss während der Laufzeit eine Konfigurationsdatei einlesen können.

M- KONF120: Einlesen der Impulskonstante einer Konfigurationsdatei
Die Software muss während der Laufzeit der Impulskonstante von bis zu vier unterschiedlichen Stromzählern aus einer Konfigurationsdatei einlesen können.

M- KONF130: Einlesen der Empfängeradresse aus einer Konfigurationsdatei
Zum senden der abgespeicherten Energieverbrauchswerte soll die IP-Adresse des Empfängers der Verbrauchsdaten, aus einer Konfigurationsdatei eingelesen werden können.

M- KONF130-10: Einlesen der Empfängeradresse aus einer Konfigurationsdatei
Wird für die IP-Adresse, an denen die Daten gesendet werden sollen als Localhost oder 127.0.01 angegeben, verbleiben die Daten auf dem Raspberry Pi. Bei der Adresse handelt es sich um den Raspberry Pi selbst [vgl. loc].

M-DAR140: Darstellung der umgerechneten Energieverbrauchswerte
Dem Benutzer sollen die umgerechneten Energieverbrauchsdaten grafisch dargestellt werden.

M-DAR140-10: Darstellung der umgerechneten Energieverbrauchswerten
Die grafische Darstellung der umgerechneten Energieverbrauchsdaten muss auf Webbasis geschehen.

M-DAR150: Grafische Darstellung der umgerechneten Energieverbrauchswerten
Die grafische Darstellung der umgerechneten Energieverbrauchsdaten muss auf einem Koordinatensystem geschehen.

M-DAR160: Auswahl der Zähler für die Darstellung der Energieverbrauchswerte
Der Benutzer soll für die grafische Darstellung die einzelnen Zähler auswählen können. Die Verbrauchswerte der ausgewählten Zähler werden dann auf einem Koordinatensystem dargestellt.

M-DAR160-10: Auswahl mehrerer Zähler für die Darstellung der Energieverbrauchswerte
Der Benutzer soll für die grafische Darstellung mehrere Zähler gleichzeitig auswählen können. Die Verbrauchswerte der ausgewählten Zähler werden dann gleichzeitig grafisch auf einem Koordinatensystem dargestellt.

M-DAR160-20: Auswahl aller Zähler für die Darstellung der Energieverbrauchswerte
Der Benutzer soll für die grafische Darstellung alle Zähler auswählen können. Die Verbrauchswerte aller Zähler werden, auf einem Koordinatensystem dargestellt.

M-DAR170: Auswahl des Zeitraumes für die Darstellung der Energieverbrauchswerte
Der Benutzer muss für die grafische Darstellung den Zeitraum auswählen können, in dem die Verbrauchsdaten angezeigt werden sollen.

M-DAR170-10: Auswahl des Startdatums für die Darstellung der Energieverbrauchswerte
Der Benutzer muss für die grafische Darstellung das Startdatum für den Zeitraum Auswählen können, bei dem die Verbrauchswerte angezeigt werden sollen, die ab dem ausgewählten Startdatum abgespeichert wurden.

M-DAR170-20: Auswahl des Enddatums für die Darstellung der Energieverbrauchswerte
Der Benutzer muss für die grafische Darstellung das Enddatum für den Zeitraum auswählen können, bei dem die Verbrauchswerte angezeigt werden sollen, die bis zum dem ausgewählten Enddatum abgespeichert wurden.

M-DAR170-30: Keine Auswahl des Datums für die Darstellung der Energieverbrauchswerte
Ein bestimmtes Datum kann nicht ausgewählt werden, wenn für dieses Datum keine Energieverbrauchsdaten abgespeichert wurden.

4.3 Kann-Kriterien

K-DAR180: Darstellung der aktuellen Energieverbrauchswerte
Dem Benutzer kann die Möglichkeit gegeben werden, die Energieverbrauchswerte darstellen zu lassen, die aktuell verbraucht werden.

K- DAR190: Export der Verbrauchsdatendarstellung in Bilddatei
Dem Benutzer kann die Möglichkeit gegeben werden, die von diesem betrachtete grafische Darstellung der Energieverbrauchsdaten in eine Bilddatei zu exportieren.

4.4 Technische Kriterien

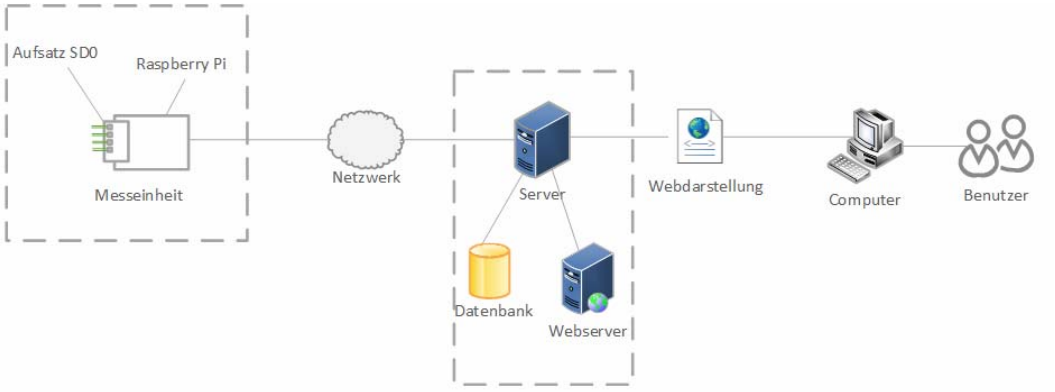
MT010: Den Aufsatz SD0 benutzen
Für die Messung der S0-Impulse muss der Aufsatz SD0 von Busware benutzt werden.

MT020: Datenempfang vom Aufsatz SD0
Die gemessenen S0-Impulse sollen vom Aufsatz SD0 von Busware über die serielle Schnittstelle empfangen werden.

5 Entwurf

In diesem Kapitel werden die Anforderungen an das System analysiert und eine Softwarelösung entworfen. Dies erfolgt vor allem durch Anwendungsfall- und Klassendiagramme.

Abbildung 5.1: Prinzipschaltbild des Messaufbaus



In der obigen Abbildung 5.1 zum Messaufbau gibt es zwei verschiedene Bereiche, für die Software entwickelt werden soll. Einerseits gibt es die Messeinheit, die vom Raspberry Pi und der Erweiterung SD0 von Busware gebildet wird. Der Aufsatz SD0 besitzt vier S0-Anschlüsse mit jeweils zwei Leitungen, die in der Abbildung auf der linken Seite mit grünen Linien dargestellt werden. Bei der Messeinheit werden die Verbrauchswerte, über die S0-Schnittstellen der Erweiterung, in Form von S0-Impulsen ermittelt. Anschließend werden diese an den Raspberry Pi über die serielle Schnittstelle (UART) übertragen, wo dann die Daten ausgewertet und abgespeichert werden. Dabei ist zu beachten, dass für die Erweiterung die eigene Firmware benutzt wird, da es die benötigte Aufgabe erledigt. Vom Raspberry Pi können die Daten an einen Server weitergesendet und dort abgespeichert werden. Der Server kann sich sowohl im lokalen Netzwerk befinden, aber auch über das Internet erreichbar sein.

Den zweiten Bereich bildet der Server. Die Verbrauchswerte sollen auch auf einem Server vom Raspberry Pi empfangen, abgespeichert und dargestellt werden können. Die Darstellung, also das Monitoring der Daten kann daher sowohl auf dem Raspberry Pi als auch auf einem von der Messeinheit entfernten Server erfolgen.

Bei den in der Abbildung 5.1 als Webdarstellung bezeichneten Einheit handelt es sich um das Monitoring der Energieverbrauchswerte, also um die grafische Darstellung der Verbrauchsdaten im Webbrowser des Benutzers.

Die beiden Grafiken, die sich in der Abbildung 5.1 auf der rechten Seite befinden, handelt es sich um den Rechner und den Benutzer des Systems, der die Darstellung der Verbrauchsdaten betrachten kann.

5.1 Anwendungsfälle der zu entwickelnden Software

Wie eingangs im Kapitel fünf beschrieben, besteht die zu entwickelnde Softwarelösung aus zwei Bereichen. Einerseits aus der Software die auf dem Raspberry Pi läuft und die Verbrauchsdaten von der Erweiterung empfängt, verarbeitet und abspeichert. Andererseits aus einer Software, die auf einem Webserver läuft und die Verbrauchswerte darstellen soll.

In diesem Kapitel werden als erstes Anwendungsfalldiagramme für diese Bereiche entworfen und beschrieben.

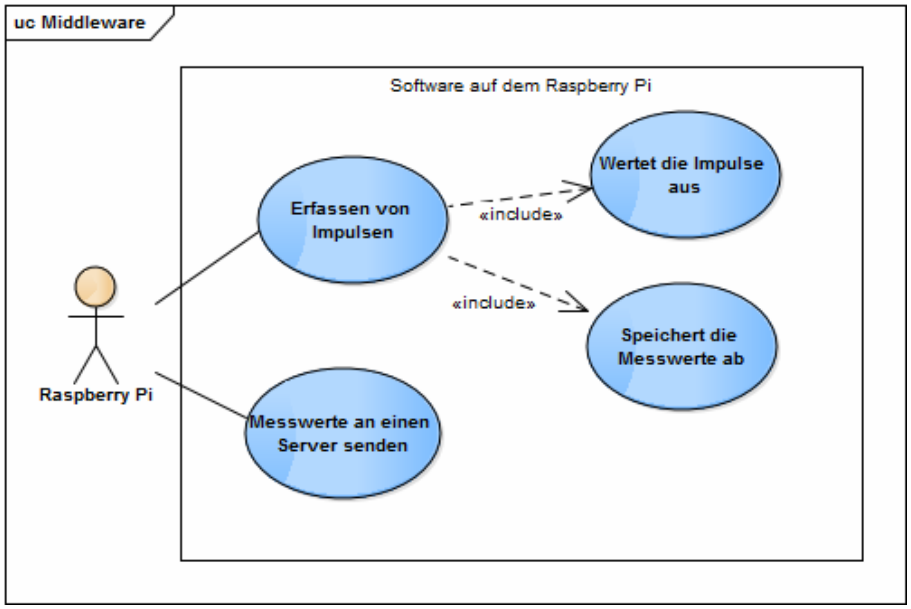
Die Beschreibungen der Anwendungsfälle erfolgt nach der in [Bal2011 S. 138] vorgeschlagenen Schablone.

5.1.1 Software der Messeinrichtung

In diesem Kapitel werden die Anwendungsfälle für die Software, die auf dem Raspberry Pi läuft, erstellt. Die Software, die sich auf dem Aufsatz SD0 befindet, ist eine Firmware, die vom Hersteller des Aufsatzes bereitgestellt und wird deshalb nicht weiter behandelt wird.

5.1.1.1 Anwendungsfalldiagramm

Abbildung 5.2: Use-Case-Diagramm für die Software der Messeinrichtung



5.1.1.2 Beschreibung der Anwendungsfälle

Die Use-Case-Beschreibungen für den Use-Case im Abbildung 5.3. ist im nächsten Kapitel zu finden. Hier bedarf es jedoch vorab der Beschreibung der Assoziationen bzw. Verbindungen zwischen den einzelnen Anwendungsfällen.

Die Anwendungsfälle „Wertet die Impulse aus“ und „Speichert die Messwerte ab“ sind Teile des Anwendungsfalls „Erfassen von Impulsen“, daher auch die jeweiligen Verbindungen.

5.1.1.2.1 Beschreibung: Erfassen von „Erfassen von Impulsen“

Tabelle 5.1: Use-Case Beschreibung, Erfassen von Impulsen

Use-Case:	**Erfassen von Impulsen.**
Ziel	Aktuellen Energieverbrauch fortlaufend über die serielle Schnittstelle, vom Aufsatz der Messeinheit, empfangen, in Verbrauchsdaten umrechnen und abspeichern.
Kategorie	-
Vorbedingung	Verbindung über die serielle Schnittstelle konnte aufgebaut werden und Empfang von S0-Impulsen.
Nachzustand Erfolg	Aktuellen Energieverbrauch fortlaufend empfangen und anzeigen.
Nachzustand Fehlschlag	Keine abgespeicherten Daten
Akteur	Raspberry Pi
Auslösendes Ereignis	Hochfahren des Raspberry Pis, da die Software als Dienstprogramm läuft, das beim Systemstart startet
Beschreibung:	
1. Die Software startet mit dem Betriebssystem des Raspberry Pis zusammen. 2. Eine Verbindung über die serielle Schnittstelle (UART) wird zum Aufsatz aufgebaut. 3. Impulse werden empfangen. 4. Umrechnen der Impulse in Watt pro Stunde. 5. Abspeichern der umgerechneten Werte. 6. Mit Schritt zwei fortfahren, solange die Option zur Darstellung der aktuellen Verbrauchdaten ausgewählt ist.	
Erweiterung/Alternative	
2a Bei erfolgloser Verbindung Fehlermeldung ausgeben und Verbindungsaufbau nochmals versuchen.	

5.1.1.2.2 Beschreibung: Erfassen von „Messwerte an einen Server versenden“

Tabelle 5.2: Use-Case Beschreibung, Messwerte an einen Server versenden

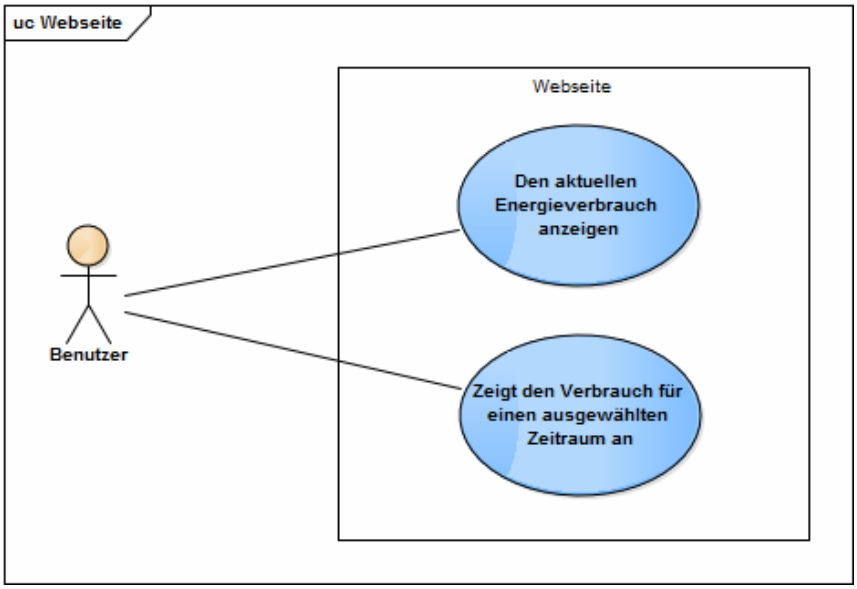
Use-Case:	**Messwerte an einen Server versenden.**
Ziel	Aktuellen Energieverbrauch fortlaufend über die serielle Schnittstelle, vom Aufsatz SD0 empfangen, in Verbrauchsdaten umrechnen und abspeichern.
Kategorie	-
Vorbedingung	Anwendungsfall „Erfassen von Messwerten“; S0-Impulse wurden erfasst umgewandelt und abgespeichert.
Nachzustand Erfolg	Abgespeicherte Messwerte wurden an einen Server versendet
Nachzustand Fehlschlag	-
Akteur	Raspberry Pi
Auslösendes Ereignis	Automatisch zyklisches Versenden der Daten durch Zeitangabe des Benutzers
Beschreibung:	
1. Verbindungsaufbau zum Server. 2. Überprüfen, ob Dateien vorhanden sind, die Übertragen werden sollen. 3. Dateien an den Server senden. 4. Mit Schritt zwei an den Zeitpunkten fortfah-ren, die vom Benutzer angegeben werden	
Erweiterung/Alternative	
2a Bei erfolgloser Verbindung eine Fehlermeldung ausgeben und Verbindungsaufbau nochmals versuchen.	

5.1.2 Darstellung der Verbrauchsdaten

In diesem Kapitel werden die Anwendungsfälle für die Darstellung der Daten analysiert.

5.1.2.1 Anwendungsfalldiagramm

Abbildung 5.3: Use-Case-Diagramm zur Darstellung der Verbrauchsdaten



5.1.2.2 Beschreibungen der Anwendungsfälle

Das Anwendungsfalldiagramm stellt die „Darstellung der Verbrauchswerte“ dar. Die Darstellung unterteilt sich in zwei verschiedene Darstellungsmöglichkeiten, die sich jedoch auf derselben Webseite befinden. Die unterschiedlichen Darstellungsformen können vom Benutzer ausgewählt werden.

5.1.2.2.1 Darstellung der aktuellen Energieverbrauchsdaten

Tabelle 5.3: Anwendungsfallbeschreibung, aktuelle Energieverbrauchsdaten darstellen

Use-Case:	**Den aktuellen Energieverbrauch anzeigen.**
Ziel	Aktuellen Energieverbrauch fortlaufend über die serielle Schnittstelle, vom Aufsatz der Messeinheit, empfangen und anzeigen.
Kategorie	-
Vorbedingung	Option zur Darstellung der aktuellen Energieverbrauchsdaten muss aktiviert werden und die Webseite zur Darstellung der Energieverbrauchdaten muss auf dem Raspberry Pi, also auf der Messeinheit laufen, auf dem die Auswertungssoftware läuft
Nachzustand Erfolg	Aktuellen Energieverbrauch fortlaufend empfangen und anzeigen.
Nachzustand Fehlschlag	Keine Visualisierung des aktuellen Energieverbrauchs
Akteur	Benutzer
Auslösendes Ereignis	Benutzer hat die Option zur Darstellung der aktuellen Verbrauchsdaten ausgewählt.
Beschreibung:	
1. Benutzer betätigt die Option zur Darstellung der aktuellen Energieverbrauchsdaten 2. Eine serielle Verbindung wird zur Erweiterung aufgebaut. 3. Impulse werden über die serielle Schnittstelle empfangen 4. Umrechnen der Impulse in Watt pro Stunde 5. Empfangene Daten darstellen 6. Mit Schritt zwei fortfahren, solange die Option zur Darstellung der aktuellen Verbrauchdaten ausgewählt ist.	
Erweiterung/Alternative	
2a Bei erfolgloser Verbindung Fehlermeldung ausgeben und Verbindungsversuch abbrechen.	

5.1.2.2.2 Darstellung der Energieverbrauchsdaten für einen bestimmten Zeitraum

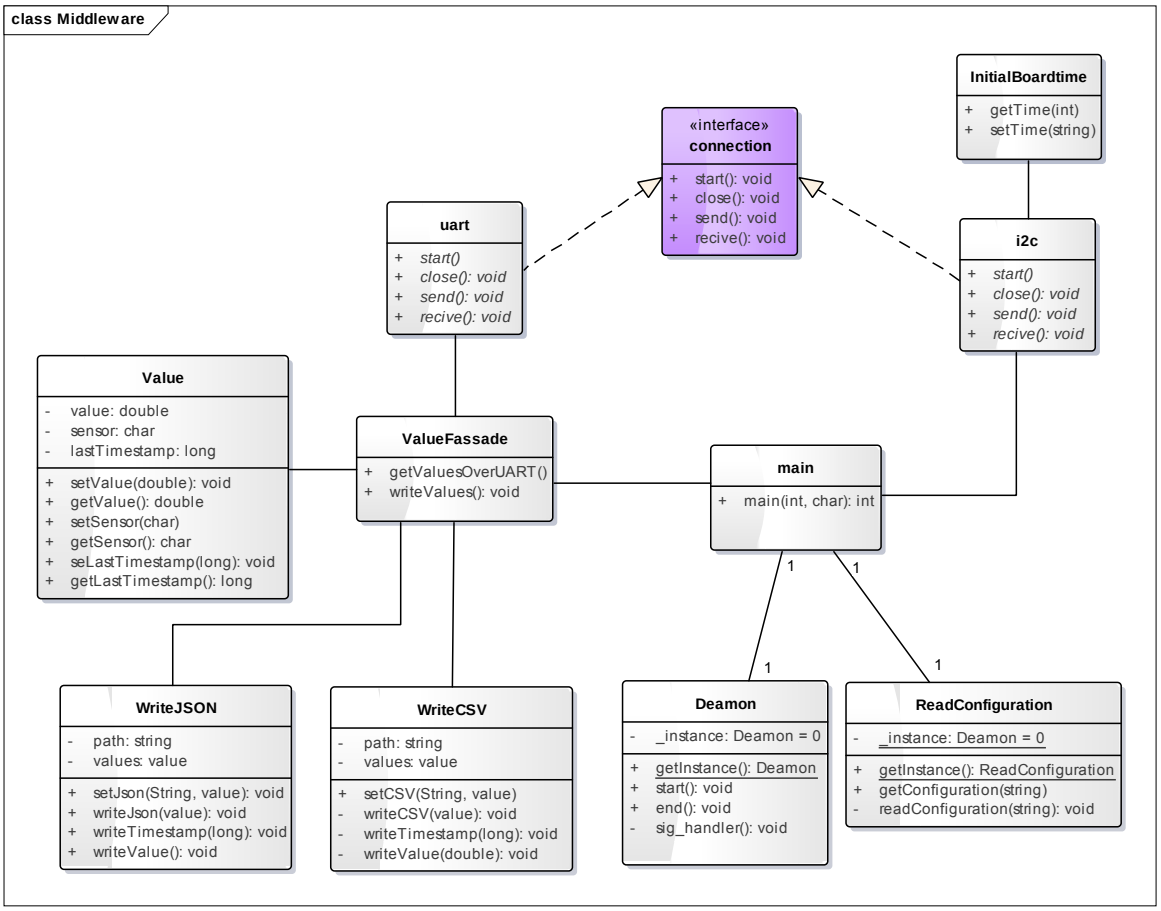
Tabelle 5.4: Anwendungsfallbeschreibung, Energieverbrauchsdaten eines bestimmten Zeitraum darstellen

Use-Case:	**Zeigt den Energieverbrauch für einen bestimmten Zeitraum an.**
Ziel	Vergangene Energieverbrauche wurden, für den vom Benutzer ausgewählten Zeitraum darstellen.
Kategorie	-
Vorbedingung	Daten für den abgefragten Zeitraum wurden abgespeichert und stehen zur Verfügung.
Nachzustand Erfolg	Grafische Darstellung der abgespeicherten Energieverbrauchsdaten für den ausgewählten Zeitraum.
Nachzustand Misserfolg	Keine Darstellung der Energieverbrauchsdaten für den ausgewählten Zeitraum
Akteur	Benutzer
Auslösendes Ereignis	Benutzer hat den Zeitraum für die Darstellung ausgewählt und Befehl zur Darstellung der Verbrauchsdaten gegeben.
Beschreibung:	
1. Benutzer wählt den Beginn des Zeitraums für die Darstellung aus. 2. Der Benutzer wählt das Ende des Zeitraums aus 3. Benutzer wählt einen Kanal aus. 4. Benutzer gibt den Befehl zur Darstellung der Verbrauchsdaten 5. Daten werden grafisch dargestellt.	
Erweiterung/Alternative	
3a Alle Stromzähler auswählen. 5a Falls keine Daten vorhanden sind, soll eine Fehlermeldung ausgegeben werden	

5.2 Klassendiagramm

In diesem Kapitel wird das entworfene Klassendiagramm vorgestellt und beschrieben. Das Klassendiagramm wird für die Software, die auf dem Raspberry Pi läuft, entwickelt.

Abbildung 5.4: Klassendiagramm für die Software auf dem Raspberry Pi



Die Interface connection bietet Methoden bzw. Schnittstellen für Klassen an, die über Bussysteme Daten empfangen sollen.

Die Klasse Values soll neben dem Verbrauchswert auch die Eigenschaft Sensor beinhalten, von dem der Energieverbrauchswert gemessen worden ist. Der Klasse ist auch der Zeitpunkt des letzten Messwertes bekannt. Bei dieser Klasse handelt es sich also um eine Datenklasse, da in der Klasse Werte bzw. Daten abgelegt und nicht bearbeitet werden.

Die Klasse UART implementiert die Interface connection. Das bedeutet, dass es die Methoden der Schnittstelle implementieren muss. Die Aufgabe der UART-Klasse ist es, über die serielle Schnittstelle, die S0-Impulse über den Aufsatz SD0 zu empfangen. Diese Klasse wird in der Klasse Value Fassade aufgerufen.

Die WriteCSV-Klasse beschreibt eine csv-Datei mit Daten, die es in der ValueFassade, von der Klasse Values, erhalten hat. WriteCSV wird also in der Klasse ValueFassade aufgerufen und nicht umgekehrt. Die WriteCSV empfängt die Daten, die von der Klasse Values gekapselt werden, in der Klasse ValueFassade. Anschließend werden die Daten in eine CSV Datei geschrieben.

Die Daten werden nicht nur in eine csv-Datei. Sondern auch in eine JSON-Datei abgespeichert. Die Klasse WriteJSON wird, wie die Klasse WriteCSV in der Klasse ValueFassade implementiert, wo es die Daten von der Klasse Value erhält und diese in eine JSON Datei parst.

In der ValueFassade werden die S0-Impulse, die über die serielle Schnittstelle von der Klasse UART empfangen, ausgewertet und in der Klasse Values abgelegt. In der ValueFassade werden auch die Werte, die in die Klasse Value abgelegt werden, an die Klassen Klassen WriteCSV und WriteJSON übergeben. Diese Klassen beschreiben dann .csv- und JSON-Dateien.

Die Klasse i2c implementiert auch die Methoden der Interface connections. Über diesen sollen die Zeiten, die vom Real Time Clock, also der Echtzeituhr des SD0, gesendet werden, empfangen und der Klasse InitialBoardtime übergeben werden. Die Klasse i2c wird in der main aufgerufen.

Die Klasse InitialBoardtime empfängt die Zeit der RTC des SD0 und setzt die Uhrzeit des Raspberry Pis, da der Raspberry Pi seine Uhrzeit über das Internet beziehen muss.

In der Klasse ReadConfiguration wird eine Konfigurationsdatei eingelesen. Dies dient dazu, Parameter während der Laufzeit der Software einzulesen und nicht direkt einzuprogrammieren. Dadurch ist ein flexiblerer Lösungsansatz geschaffen. Wenn die Parameter in die Software einprogrammiert werden, müsste die Software von Neuem kompiliert und gelinkt werden.

Durch die Konfigurationsdatei werden die Impulskonstanten für die Zähler eingelesen, von denen die S0-Impulse empfangen werden sollen. Zudem steht auch die Adresse des Servers in der Konfigurationsdatei, zu dem die abgespeicherten Daten gesendet werden sollen.

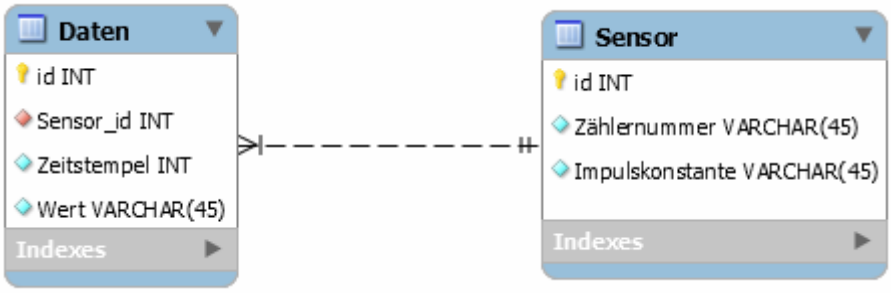
Die Klasse Deamon erzeugt aus dem ganzen Programm ein Dienstprogramm. Dienstprogramme laufen unter Linux im Hintergrund und erfüllen bestimmte Aufgaben, die von Programm zu Programm unterschiedlich sind. Zudem starten diese auch mit dem Raspberry Pi bzw. wenn dessen Betriebssystem hochgeladen wird. Die Klasse Deamon wird in der Main Klasse aufgerufen [vgl. Wol2006 dpz und Wol2006 af].

5.3 Datenmodell

Die Verbrauchsdaten sollen abgespeichert werden. Daher muss eine Datenbank oder Datenhaltungsmöglichkeit ausgewählt werden. Dadurch ist es dem Benutzer möglich die Verbrauchswerte für bestimmte Zeiträume anzeigen zu lassen, falls die abgefragten Zeiten vorhanden sind. Durch diese Möglichkeit können die Verbrauchsdaten jederzeit wieder verwendet oder miteinander verglichen werden.

Für den Entwurf der Datenhaltung wird ein ER-Diagramm erstellt. An dieser Stelle soll jedoch keine Festlegung auf eine bestimmte Datenhaltungsform stattfinden. Das ER-Diagramm stellt eine Übersicht für die benötigten Tabellen und deren Eigenschaften dar.

Abbildung 5.5: ER-Diagramm für die Datenhaltung



Die Datenhaltung beinhaltet die beiden Tabellen Daten und Sensor mit deren Eigenschaften. Diese werden in der Abbildung 5.4 dargestellt. Sie wurden aus der Aufgabe und Anforderungsdefinition hergeleitet.

Die Tabelle Daten besitzt neben der Eigenschaft id, bei dem es sich um einen Primary Key handelt, das bedeutet dass die einzelnen Eintragsätze eindeutig Identifiziert werden können, auch den Foreigen Key Sensor_id. Der Foreigen Key ist also der Fremdschlüssel, mit dem auf andere Tabellen verwiesen wird, es handelt sich hierbei also um die eindeutige Identifikation einer anderen Tabelle bzw. deren Datensätze. Hierdurch werden die erfassten Werte, einem bestimmten Sensor zugeordnet. Beim Aufsatz SD0 können bis zu vier S0-Schnittstellen benutzt werden. Es gibt also vier Sensoren, über die gemessen wird.

Zusätzlich beinhaltet die Tabelle Daten die Eigenschaft timestamp, zu Deutsch auch Zeitstempel genannt, wodurch der Zeitpunkt an dem ein Verbrauchswert gemessen wird, bekannt wird. In der Eigenschaft value werden die Verbrauchswerte abgelegt.

Die Verbindung zwischen den Tabellen Daten und Sensor besitzt die Multiplizität keine zu viele. Das bedeutet, dass keine, eine oder mehrere Daten einem bestimmten Sensor zugewiesen werden können.

Die Tabelle Sensor beinhaltet wie die Tabelle Daten auch die Eigenschaft id, die zur eindeutigen Identifikation benutzt wird. Die Eigenschaft wird von der Tabelle Daten referenziert. Daneben besitzt der Sensor eine Zählernummer, mit dem jeder Sensor in der Regel voneinander unterscheidbar ist. Die Tabelle besitzt auch die Eigenschaft Impulskonstante, die zur Umrechnung der S0-Impulse zu Watt pro Stunde benutzt werden kann. Die Impulskonstante wird von einer Konfigurationsdatei gelesen und während der Laufzeit in die Eigenschaft abgelegt.

5.4 Visualisierung der Verbrauchsdaten

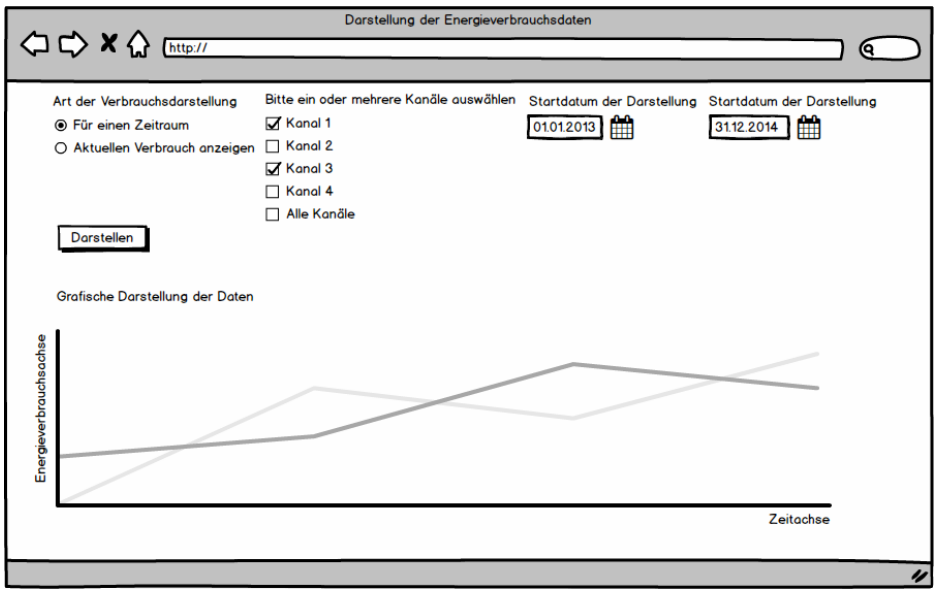
In diesem Kapitel wird die Oberfläche für die Darstellung der Energieverbrauchsdaten vorgestellt.

Es gibt eine einzelne Oberfläche, über diesen dem Benutzer die Möglichkeit gegeben werden soll, entweder zwischen einem Zeitraum oder einer Darstellung des aktuellen Energieverbräuche zu wählen. Bei der grafischen Darstellung werden also zwei Arten unterscheiden. Diese beiden Arten unterscheiden sich in puncto Zeit, da es sich entweder bei den abgespeicherten Werten um vergangene Werte oder um die Darstellung der aktuellen Verbrauchswertene handelt. Hier werden die Verbrauchsdaten direkt nach dem Messvorgang an den Raspberry Pi gesendet, wo diese dann unmittelbar nach dem Empfangen ausgewertet und dargestellt werden können.

Bei der Darstellung hat der Benutzer drei Auswahlgruppen: einerseits handelt es sich, wie oben beschrieben, um die Art der grafischen Darstellung. Die zweite Auswahlgruppe ist die Auswahl der jeweiligen Kanäle, die dargestellt werden sollen. Bei der letzten Auswahlgruppe handelt es sich um einen bestimmten Zeitraum, die dargestellt werden soll. Die letzte Auswahl hängt von der ersten Auswahl ab.

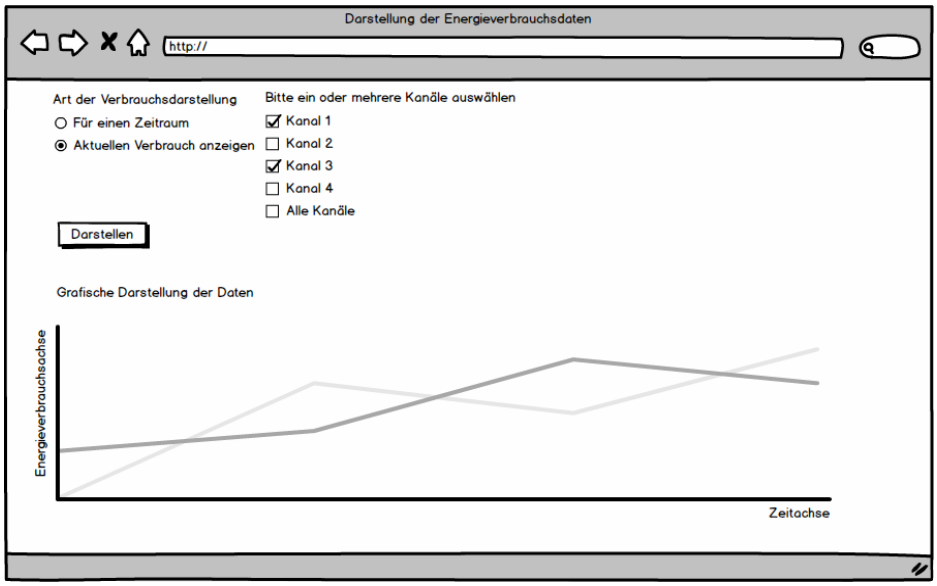
Bei der Darstellung des aktuellen Energieverbrauchs werden die jeweiligen Verbrauchswerte über die UART-Schnittstelle der GPIO-Steckleiste übernommen, ausgewertet und direkt dargestellt. Gleichzeitig werden die Daten dabei unabhängig von der Darstellung, weiterhin von der Software, die auf dem Raspberry Pi liegt, über den UART empfangen, ausgewertet und abgespeichert. Für die Darstellung der aktuellen Verbrauchswerte befindet sich die Webseite auf dem Raspberry Pi.
Die Kanäle können einzeln, in unterschiedlichen Kombinationen oder alle Kanäle zusammen ausgewählt werden. Die Kanäle, die über die Weboberfläche ausgewählt werden, orientieren sich an den jeweiligen S0-Kanälen des Aufsatzes SD0.

Abbildung 5.6: Beispielhafter Oberflächenaufbau der Verbrauchsdatenvisualisierung für eine Zeitraum



Die Abbildung 5.6 zeigt ein beispielhaftes Oberflächendesign, für die Verbrauchsdatendarstellung über einen bestimmten Zeitraum. Die Auswahl der Darstellungsart erfolgt auf der linken oberen Ecke der Abbildung 5.4.gefolgt von den möglichen Kanälen. Anschließend kann, in der oberen rechten Ecke der Zeitraum für die Darstellung als Datum ausgewählt werden.

Abbildung 5.7: Beispielhafter Oberflächenaufbau der Visualisierung der aktuellen Verbrauchsdaten



Den Kontrast zur Abbildung 5.6 stellt die Abbildung 5.7 dar, da hier nicht der Verbrauch über einen bestimmten Zeitraum dargestellt werden soll, sondern der aktuelle Verbrauch. Bei der Darstellung der aktuellen Verbrauchsdaten sollen die Daten, die gemessen wurden, nicht von der Darstellung verschwinden. Wenn ein Verbrauchswert dargestellt wird und anschließend ein weiterer, soll der ältere Verbrauch im Graphen einen zeitlichen Stellenwert nach rechts verschoben werden. Je größer der zeitliche Abstand zwischen dem alten und neuen Wert ist, desto länger ist die gerade Line zwischen diesen Punkten.

6 Realisierung

In diesem Kapitel wird die Realisierung der prototypischen Entwicklung beschrieben.

Um die Verbrauchswerte zu messen gibt es schon ein Projekt, den Volkszähler. Dabei handelt es sich um ein Open Source Projekt [vgl. vza]. Mit dem Volkszähler können Energieverbrauchswerte gemessen und auf dessen Darstellung Verbrauchswerte von Wasser, Strom Gas usw. angezeigt werden [vgl. Zoe], mit dem die Nutzer die eigenen Verbrauchdaten wie Strom, Temperatur, Wasser am Zähler messen können. Die gemessenen Verbrauchsdaten werden dann über die Schnittstellen an das Messgerät übertragen, dort gespeichert und auf einer Website ausgewertet. Ein geeignetes Messgerät muss vorhanden sein. In dieser Arbeit handelt es sich dabei um den SD0 von Busware, zusammen mit dem Raspberry Pi. Die Messung findet dabei am Zähler über die S0-Schnittstelle statt. Die Übertragung findet über Kupferkabel statt. Die Speicherung und Auswertung finden dann durch die Software statt. Bei der Software handelt es sich um den Volkszähler. Mit dem Volkszähler können auch Verbrauchsdaten über S0-Schnittstellen gemessen werden [vgl. vzg]. Dazu gibt es zwei Softwares, um S0-Impulse zu messen. Dabei handelt es sich um s0vz [vgl. s0vz] und s0enow [vgl. s0enow]. Aus dem Quellcode bzw. der Datei s0enow.cpp [vgl. s0ec], in der sich auch die Main-Funktion befindet, geht hervor, dass sich beim Projekt s0enow um einen nach GNU General Public Licence der Version 3 Lizenzierte Software handelt. Das bedeutet, dass der Quellcode verwendet bzw. Modifiziert werden darf. Dabei muss aber die Software, die Teile der Software, die nach GNU General Public Licence der Version 3 lizenziert wurde, verwendet auch als solche lizenziert werden [vgl. s0ec Codezeilen 23 bis 38].

In dieser Arbeit werden einige Funktionen, die im Softwareprojekt s0enow vorhanden sind, verwendet. Dadurch müssen diese nicht von neuem entwickelt werden. Neben der Bibliothek, die zum Einlesen der Konfigurationsdatei dient, werden ähnliche Daten eingelesen, wie im Projekt s0enow. Dies ist der Tatsache geschuldet, dass die Software die in dieser Arbeit entwickelt wird, eine ähnliche Aufgabe erfüllt, wie das Projekt s0enow. Dabei handelt es sich um die Funktion cfile(). Diese Datei befindet sich in den Codezeilen 204 bis 309 im s0enow.cpp [vgl. s0ec].
In der Software, die in dieser Arbeit entwickelt wird, werden neben dem Datafolder, die Messstellenname, die Mittelwertszeit auch die Impulskonstanten ausgelesen.

Zusätzlich gibt es Überschneidungen mit den Projekten s0enow und s0vz. Wie in diesen Projekten wird die Software, die auf dem Raspberry Pi laufen soll, als Dienstprogramm bzw. Deamon entwickelt So genannte Deamons laufen unter Linux im Hintergrund und starten, wenn das Betriebssystem hochgefahren wird [vgl. Wol2006 dpz und Wol2006 af]. Diese Funktionen befinden sich in den Codezeilen 111 bis 202, im Projekt s0enow [vgl. s0ec]. Bei den Funktionen handelt es sich um die drei folgenden Funktionen,

```
void signal_handler(int sig)
```

Behandlung von Neustart, Beendigung Anfragen.

```
void daemonShutdown()
```

Schließt bzw. beendet ein laufendes Dienstprogramm.

Void daemonize(char *rundir, char *pidfile)
Erstellt den Deamon.

Bei den Funktionen, die die Konfigurationsdatei einlesen und den Dienstprogramm bzw. Deamon erstellen sind sowohl im s0enow, als auch im s0vz vorhanden.

Zusätzlich werden noch folgende Funktionen verwendet,

void update_average_values(struct valuePack *vP)

Diese Funktion befindet sich in der Datei, s0enow.cpp [vgl. s0ec Zeile 471 bis 470]. Diese Funktion wandelt die Empfangenen S0-Impulse in Energieverbrauchsdaten um [vgl. s0ec].
Eingesetzt wird diese Funktionen, in dieser arbeit zu entwickelnden Software in der Klasse *Uart*.

Int appendToFile(const char *filename, char *str)

Diese Funktion speichert die gemessenen und ausgewerteten Verbrauchsdaten, in .csv, also in Coma Separated Values, ab. Beim filename handelt es sich um den Dateipfad, in der die .csv-Datei abgelegt wird. Ist die Datei nicht vorhanden, wir eine neue Datei angelegt. Dabei wird für jeden Tag eine Datei erstellt. Ist schon eine Datei für einen Tag vorhanden, wird in diesen die im Parameter *str* enthaltene Zeichenkette eingefügt. Ansonsten wird davor eine Datei erzeugt.

6.1 Entwicklungsumfeld

Der Prototyp der Software wird unter Ubuntu, mit der Entwicklungsumgebung Eclipse CDT entwickelt. Nach der Entwicklung des Software mit Eclipse wird die Software auf den Raspberry Pi übertragen. Nach der Übertragung wird die Software kompiliert.
Da zur Programmierung die Programmiersprache C++ verwendet, wird der Compiler g++ mit der Version 4.8.4 benutzt. Nach der Programmierung wird die Software auf den Raspberry Pi übertragen und dort ebenfalls mit Hinzuziehung des g++ Compilers auf dem Raspberry Pi übersetzt und kompiliert.

Beim Entwurf der Arbeit wird das Programm Enterprise Architect von Sprax Systems, zur Erstellung der Anwendungsfälle und des Klassendiagramms benutzt.

Zusätzlich kommen Bibliotheken zum Einlesen der Konfigurationsdatei und ein JSON Parser hinzu. Diese bieten Funktionalitäten an, um Konfigurationsparameter einzulesen und JSON-Dateien zu beschreiben und Konfigurationsdateien zu lesen

6.2 Verwendete Hardware

Bei der Hardware, die als Prototyp eingesetzt wird, handelt es sich um den Raspberry Pi Model B und den Aufsatz SD0 von Busware. Der Aufsatz SD0 wird als Anforderung definiert, weshalb der Auswahl der Plattform auf den Raspberry Pi Mo-

del B fällt, welche im Kapitel „Fachliches Umfeld“ beschrieben wird. An dieser Stelle sei auf das Kapitel 3.1 Beschreibung des Raspberry Pi hingewiesen.

Zum Testen des Prototypen wird der Zähler DPM1L32-D von Voltcraft verwendet. Dieser besitzt eine S0-Schnittstelle unterhalb de Zählers.

6.3 Firmware bzw. Bootloader des Aufsatzes SD0

Eine Firmware ist eine Software, die für die Grundfunktionalität der Hardware verantwortlich ist [vgl. gfi]. Für den Aufsatz SD0 werden zwei unterschiedliche Firmware angeboten. Dabei handelt es sich um die Software in den Ordnern mit der Bezeichnung Firmware und Firmware_VZ. In dieser Arbeit wird die Firmware im Ordner Firmware_VZ benutzt. Darin befindet sich ein Pearl Skript, um die Firmware auf den Aufsatz SD0 zu bringen, welcher sich dabei auf dem Raspberry Pi befinden muss.

Die Firmware für den SD0 muss selbstständig auf den SD0 geladen werden. Dazu wird vom Hersteller ein Perl Skript vorgegeben.

6.4 Datenhaltung

Die Datenhaltung der Messwerte erfolgt sowohl in .csv, als auch auf JSON- Basis. Die Datenhaltung auf .csv Basis hat den Nutzen, dass die Daten spezifikations- und technologieunabhängig genutzt werden können [vgl. RFC4180].

Die Daten werden in JSON geschrieben, da diese spätestens bei der Webdarstellung der Verbrauchsdaten in JSON konvertiert werden müssen. Durch das frühen Parsen wird erreicht, dass die Daten auf dem Raspberry Pi schon von Anfang an in JSON beschrieben werden und nicht erst bei der Darstellung in .JSON konvertiert werden müssen. Dadurch wird ermöglicht, dass die Daten auch bei anderen Systemen als JSON-Datei vorliegen, die auf JSON basierend arbeiten. Des Weiteren handelt es sich bei JSON um ein standardisiertes Format, so dass die Daten in einem spezifischen Format abgelegt werden können.

Als Bibliothek für den JSON-Parser wird der JsonCPP verwendet.

6.4.1 JSON-Bibliothek

Um JSON-Dateien zu parsen, also schreiben zu können, wird eine JSON-Parser-Bibliothek benutzt. Die Auswahl basiert auf dem Benchmarktests die von Santiago Alessandri [vgl. Ale2014] durchgeführt und präsentiert werden. Die durchgeführten Tests werden, nach den Kriterien Performance und Serialisierung auf der Webseite präsentiert. Dort werden die folgenden drei JSON Parser getestet:

JsonCpp, es soll sich um einen der gängigsten Json- Parser für C++handeln. [vgl. Ale2014]

Casablanca, soll ein von Microsoft gepflegter SDK zur Client-Server Kommunikation handeln. Dabei handelt es sich also um mehr als ein JSON Parser.
Um Cassablanca benutzen zu können wird Boost benutzt. [vgl. Ale2014]

Json Spirit, ist ein auf dem Parsergenerator Boost Spirit basierender Parser.

Auf den, auf der Webseite [vgl. Ale2014] dargestellten Tests, wird die Bibliothek JsonCpp ausgewählt, um JSON-Dateien zu parsen.

Bei JsonCpp handelt es sich um einen für die Programmiersprache C++ geschriebenen Parser für Json, mit dem die Json Daten manipuliert werden können. Des Weiteren kann nach Zeichenketten serialisiert werden können [vgl. jcd Introduction].

Die Struktur einer beispielhaften Json Datei hat dabei folgende Struktur:

Listing 6.1: Beispielhafte Json-Struktur für die Datendarstellung

```
{

  “data”:[{
          “Timestamp”:
    A:
    B:
    C:
    D:
  }]
}
```

Es werden keine expliziten JSONObjekte für die einzelnen Zähler bzw. Kanäle des SD0 erzeugt, da diese mit den zugehörigen Werten zusammen in Json abgelegt werden.

6.4.2 Datenhaltung in SQL

Im *Kapitel 5.3 Datenmodell* wurde ein ER-Diagramm erzeugt. Die Struktur dieses ER-Diagramms wird in eine SQL-Datenbank überführt. Die Datenbank, die in dieser Arbeit erzeugt wird, heißt *bachelorarbeit*. In dieser Datenbank sind, wie im ER-Diagramm dargestellt zwei Tabellen vorhanden. Diese heißen *daten* und *sensor* Die Tabellen werden in der Datenbank folgendermaßen erstellt:

Listing 6.2: Erzeugung der Tabelle für die Verbrauchskanäle

```
1  CREATE TABLE sensor
2  (
3    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
4    zaehlername varchar(40),
5    impulskonstante INT
6  );
```

Die Tabelle, die die Energieverbrauchswerte beinhaltet, besitzt folgende SQL-Struktur

Listing 6.3: Erzeugung der Tabelle zur Verbrauchsdatenhaltung

```
1  CREATE TABLE werte
2  (
3    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
4    sensor_ID INT,
5    zeitstempel DATETIME,
6    wert DOUBLE,
7   CONSTRAINT FOREIGN KEY werte (sensor_ID) REFERENCES sensor(ID)
```

```
8 );
```

Zuerst muss die Tabelle sensor erstellt werden, da die zweite Tabelle werte, auf die Tabelle sensor, verweist. Der Verweis erfolgt durch einen Fremdschlüssel.

6.5 Bibliothek zum Einlesen von Konfigurationsdateien

Die Konfigurationsdatei wird von der Software, die auf dem Raspberry Pi läuft verwendet. Die Konfigurationsdatei wird zum Lesen der Impulskonstanten, die zum Umrechnen der S0-Impulsen in Energieverbrauchswerte dienen, dem lesen der Serveradresse usw. verwendet. Mit Konfigurationsdateien ist es möglich, Software mit Parametern zu versorgen ohne diese wieder, aufgrund der Parameter, zu kompilieren.

Neben der Benutzerabfrage, sei es durch eine grafische Oberfläche oder eine Abfrage über eine Konsole, gibt es zwei grundlegende Möglichkeiten, ein Programm mit Konfigurationsparametern zu versorgen ohne die Software neu kompilieren zu müssen [vgl. Mou2011 S. 2]. Eine Benutzerabfrage ist bei der Software, die auf dem Raspberry Pi läuft nicht vorgesehen, da dieser in keinem, zumindest keinem direkten Kontakt mit einem menschlichen Benutzer steht.

Einerseits ist es möglich das Programms während des Programmstarts über die Main-Funktion des Programm, mit Parametern zu versorgen. Dabei können die Parameter direkt angegeben werden. Falls die Parameter direkt übergeben werden, können diese nicht mehr geändert werden.

Andererseits kann dem Programm der Pfad zu einer Datei übergeben werden, in der die Konfigurationsdaten stehen. Die Parameter werden dann aus der Datei eingelesen und benutzt. Dazu wird ein Parser verwendet.

Da sowohl in der Software s0vz [vgl. s0vc Zeile 43] als auch in s0enow [vgl. s0ec Zeile 53] die Konfigurationsbibliotheken libconfig verwendet wird, wird in dieser Arbeit auch die Bibliothek libconfig verwendet.

Mit der Libconfig-Bibliothek können Konfigurationsdateien gelesen, verändert und geschrieben werden. Mit der für die Programmiersprachen C und C++ entwickelten Parser-Bibliothek für Konfigurationsdateien, können unterschiedliche, voneinander unabhängige Konfigurationsdaten gleichzeitig in unterschiedlichen Threads gelesen werden und auf unterschiedlichen Betriebessystemen benutzt werden. Bei den Konfigurationsdateien, die von libconfig geparst werden, soll es sich um einfache, strukturierte und lesbare Dateien handeln [vgl. Lin2015 S. 5].

Beim Libconfig handelt es sich jedoch um einen Parser, der nicht threadsafe ist. Das bedeutet, dass, wenn mehrere Threads auf eine Konfigurationsdatei zugreifen müssen, der Zugriff der jeweiligen Threads auf die Konfigurationsdatei koordiniert werden muss [vgl. Lin2015 S. 6].

Da der Parser einige Bibliotheken der Programmiersprache C benutzt die nicht synchronisationssicher sind soll der Parser auch nicht synchronisationssicher sein. Daher sollen die signal handler keine Anfragen an den Parser stellen können [vgl. Lin2015 S. 6].

Die Konfigurationsbibliothek liest Name-Werte-Paare ein. Einem Namen sind meist ein oder mehrere Werte zugewiesen. Das können bei Libconfig Werte folgende Arten sein [vgl. Lin2015 S. 5 bis S.8]:

Scalare, Ganzzahlen, Fließkommerzahlen, Wahrheitswerte und Zeichenketten

Felder (auch array gennant), eine Sequenz an Scallaren selben Typs

Gruppen (auch als group bezeichnet) eine Sammlung von Konfigurationen

Listen (auch Lists), eine Liste kann weitere Listen, aber auch Werte unterschiedlicher Typen haben

6.6 Auf dem Raspberry Pi laufendes Software

In diesem Kapitel wird die Software erläutert, die auf dem Rasppberry Pi läuft.

6.6.1 Verbindung über die GPIO-Schnittstelle

Zur Kommunikation mit Schaltungen und Erweiterungen wird, wie bekannt, die GPIO-Schnittstelle des Raspberry Pis benutzt. Darüber wird nicht nur die Uhrzeit und das Datum des Raspberry Pi durch die Echtzeituhr über den Datenbus I²C gesetzt sondern auch die Verbrauchswerte, die am Stromzähler in Form von S0-Impulsen gemessen werden, an den Raspberry Pi über die serielle Schnittstelle, dem UART, gesendet. Dabei wird die Programmiersprache C++ benutzt.

6.6.1.1 Verbindung über die serielle Schnittstelle (UART)

Zur Benutzung und Steuerung der seriellen Schnittstelle, also der UART-Schnittstelle, des Raspberry Pis gibt es drei Möglichkeiten [vgl. Kof2015 S. 787]. Diese Möglichkeiten beziehen sich vor allem auf die Benutzung bestimmter Bibliotheken. Bei diesen Bibliotheken handelt es sich um die folgenden [vgl. Kof2015 S. 787]:

sysfs-Dateisystem: Dabei handelt es sich um eine Möglichkeit, die ganz ohne externe Bibliotheken auskommen soll [vgl. Kof2015 S. 787]. Dabei werden die vom Linux-Kernel bereitgestellten, in der Programmiersprache C geschriebenen Funktionen benutzt. Damit soll eine möglichst direkte Kommunikation mit der Hardware möglich sein [vgl. Dem2013 S. 173].

WiringPi-Bibliothek: Bei dieser Bibliothek handelt es sich um eine, an den Raspberry Pi angepasste Bibliothek, die alle Grundfunktionen, auch die der von Raspberry Pi zur Verfügung gestellter Datenbusse, der GPIO-Schnittstellen unterstützt [vgl. Kof2015 S. 787].

BCM2835-Bibliothek: Dabei handelt es sich um eine Bibliothek, deren Funktionsumfang vergleichbar mit der der WiringPi-Bibliothek sein soll. Es ist eine speziell auf den SoC BCM2835 angepasste Bibliothek, womit diese auch auf anderen Systemen eingesetzt werden kann, die auf dem BCM2835 basieren [vgl. Kof2015 S. 787].

Um eine Portierbarkeit auf andere Einplatinencomputer wie den BabanaPi oder BeagleBone, auf denen auch Betriebssysteme mit Linux-Kernel laufen, ermöglichen zu können, erfolgt die Entwicklung der Kommunikation bzw. Benutzung der GPIO-Steckleiste mit dem sysfs-Dateisystem. Dadurch wird die Abhängigkeit vom benutzten Einplatinencomputer soweit wie möglich reduziert, da die Linux eigenen Funktionen verwendet werden.

Bei der Kommunikation über die UART-Schnittstelle wird der Port bzw. die Schnittstelle wie ein Datenstrom behandelt.

Listing 6.4: Öffnen eines Ports über den UART

```
1  start(const char *device ){
2          int uart0_filestream;
3
4          Uart0_filestream = open(device, O_RDWR | O_NOCTTY | O_NDELAY);
5          if (uart0_filestream == -1)
6          {
7                  //Serieller Prot konnte nicht geöffnet werden
8                  printf("Error – Unable to open UART.  Ensure it is not in use by another
9  application\n");
10         }
11         struct termios options;
12         tcgetattr(uart0_filestream, &options);
13         options.c_cflag = B38400 | CS8 | CLOCAL | CREAD; //<Set baud rate
14         options.c_iflag = IGNPAR;
15         options.c_oflag = 0;
16         options.c_lflag = 0;
17         tcflush(uart0_filestream, TCIFLUSH);
18         tcsetattr(uart0_filestream, TCSANOW, &options);
19
20         this->device = device;
21         running = true;
22
23         if (pthread_create(&runningThread, NULL, BuswareS0::callRunFunction,
24 this) != 0) {
25                 printf("Running Thread can not be create.");
26         }else {
27                 printf("Thread gestartet.");
28         }
29         return true;
30 }
```

Im Listing 6.4 wird über die UART Schnittstelle eine Verbindung aufgebaut. Dies geschieht in der Zeile vier des Listings. Dort wird eine Verbindung mit dem Befehl open(…) geöffnet. Der Befehl beinhaltet zwei Parameter, die mit Kommata getrennt sind. Der erste Parameter gibt an, wie die Verbindung zustande kommt, also mit welcher Schnittstelle bzw. welchem Datenbus die Verbindung aufgebaut werden soll. Bei dieser Verbindung kann sowohl gelesen als auch geschrieben werden (mit Angabe des Parameters O_RDWR). Durch Angabe von O_NOCTTY wird eine Terminaltverbindung initialisiert. Der Parameter O_NDELAY bedeutet, dass es keine Verzögerungen in der Verbindung geben soll. Falls über die Verbindung keine Daten empfangen werden, kann es sein, dass eine Fehlermeldung eintritt [vgl. ieo]. Für den ersten Parameter des Befehls open(…), der Parameter device kann auch direkt

```
/dev/ttyAMA0
```

eingetragen werden. Dies repräsentiert die UART Schnittstelle [vgl. Pla uart]. Falls die Verbindung erfolgreich geöffnet werden konnte, liefert open(…) einen positiven Wert zurück. Bei dem Rückgabewert handelt es sich um einen Wert größer oder gleich 0, falls eine Verbindung aufgebaut werden konnte. Anderenfalls wir eine nega-

tive Zahl zurückgegeben. Der Rückgabewert wird in einer Variablen abgelegt. Diese Variable dient dann als File descriptor, mit der die Verbindung referenziert werden kann [vgl. ieo].

Zwischen den Zeilen 11 und 21 werden die Einstellungen für die UART Verbindung konfiguriert. Die Konfiguration der UART Verbindung geschieht durch den Befehl termios. Dabei handelt es sich um eine Struktur. Hier werden die Strukt-Komponenten, durch den Variablennamen des Strukts, die options lautet, gesetzt. Es werden vier Komponenten der Struktur termios gesetzt. Dazu werden zunächst die Einstelllungen in der Zeile 12 abgefragt und in einem Zeiger auf options abgelegt. Hier muss auf den Filedescriptor im ersten Parameter referenziert werden. Mit der options variable werden dann die einzelnen Einstellungen vorgenommen.

Mit der Komponente c_cflag werden die Kontroll- und Verbindungseinstellungen vorgenommen. Mit dieser Komponente werden nicht nur die Baudrate, also die Verbindungsgeschwindigkeit, eingestellt sondern auch die Zeichengröße, die übertragen (CS8) werden soll. Es werden acht Zeichnen mit einer Baudrate von 38400 übertragen. Zudem wird die Modemsteuerung ausgeschaltet. Da bei der Verbindung nur gelesen werden soll, wird der Parameter CREAD mit angegeben. Die Optionen werden durch den Befehl tcsetattr in Zeile 18 gesetzt [vgl. Wol2006].

Mit c_iflag werden die Einstellungen für die Eingabesteuerung bzw. zum Lesen der ankommenden Daten gesetzt. Hier werden nur die empfangenen Bytes mit Paritätsüberprüfungsfehler ignoriert. Da keine Daten gesendet (Komponente c_oflag) und die Eigenschaften des Terminalfensters (c_lflag) nicht gesetzt werden sollen, werden die jeweiligen Komponenten mit einer Null (0) versehen [vgl. Wol2006].

Der Empfang der Daten erfolgt folgendermaßen:

Listing 6.5: Datenempfang über die UART-Schnittstelle

```
1  unsigned char rx_buffer[255];
2  unsigned char proto_buffer[255];
3  int position = 0;
4  if (uart0_filestream != -1)
5  {
6          printf("\nOpen %s. \n", device);
7  }
8  if (((BuswareS0 *)This)->uart0_filestream != -1)
9  {
10 int rx_length = 0;
11         while (((BuswareS0 *)This)->running)
12                 {
13                         rx_length = read(((BuswareS0 *)This)->uart0_filestream,
14                                                                 (void*)rx_buffer, 255);
15                         if (rx_length < 0)
16                         {
17                                 //An error  (will occur if there are no bytes)
18                                 //printf("Error! UART.cpp:  ");
19                         }
20                         else if (rx_length == 0)
21                         {
22                                 //No data waiting
```

```
23                              printf("buswareS0.cpp: Data! No data expected! The
24  Cape are sending no data");
25                      }
26                      else
27                      {
28                              position = 0;
29                              for (int i=0; © < rx_length; i++)
30                              {
31                                      printf("Buffer mit dem Index %i: %.2x %s", ©,
32  rx_buffer[i], rx_buffer);
33                                      printf(„Inhalt des Gesammten Buffers %s“,
34  rx_buffer);
35                                      proto_buffer[position] = rx_buffer[i];
36                                      position++;
37                              }
38                              proto_buffer[4] = 0;
39                              printf("\n");
40
41  if(proto_buffer[0] == 'A' && proto_buffer[0] == '\r' && proto_buffer[0] == '\n'){
42          //Erster Chanel (A)
43          printf("A Chanel else-schleife, %s", proto_buffer);
44          update_average_values(&this->values[0]);
45  }else if(proto_buffer[0] == 'B' && proto_buffer[0] == '\r' && proto_buffer[0] == '\n'){
46          //zweiter Chanel (B)
47          printf("B Chanel else-schleife, %s", proto_buffer);
48          update_average_values(&this->values[1]);
49  }else if(proto_buffer[0] == 'C' && proto_buffer[0] == '\r' && proto_buffer[0] == '\n'){
50          //dritter Chanel (C)
51          printf("C Chanel else-schleife, %s", proto_buffer);
52          update_average_values(&this->values[2]);
53  }else if(proto_buffer[0] == 'D' && proto_buffer[0] == '\r' && proto_buffer[0] == '\n'){
54          //Vierter Chanel (D)
55          printf("D Chanel else-schleife, %s", proto_buffer);
56          update_average_values(&this->values[3]);
57  }else{
58          printf("UART.cpp: Function: run(void *This): Received Value! The over"+
59                  "UART received values are invalid. Received values:\n %s",
60                  proto_buffer);
61          }
62  }
63          sleep(1);
64  }
65          }
66          close(((BuswareS0 *)This)->uart0_filestream);
```

Der Empfang von Daten wird im Listing 6.5 dargestellt. Um Daten von der seriellen Schnittstelle zu lesen, werden vier Variablen benötigt: Die erste Variable, die in der ersten Zeile deklariert wird, wird zum Puffern der empfangenen Daten benutzt. In die zweite Variable, die in der Zeile zwei deklariert ist, werden die gepufferten Daten abgelegt. Anhand dieser Variable werden die Daten dann verarbeitet. Mit der in der Zeile drei deklarierten Variable werden die gepufferten Daten in den proto_buffer gela-

den. Die rx_length-Variable, vom Typ int ist eine Ganzzahl-Variable, die die Länge der empfangenen Daten beinhaltet. Anhand dieser Variable kann überprüft werden, ob überhaupt Daten empfangen werden.

In den Zeilen sieben bis zehn wird überprüft, ob überhaupt eine Verbindung über die UART-Schnittstelle aufgebaut wurde. Die Variable uart0_filestream in Zeile 12 wird im Listing 6.1 sowohl deklariert als auch verwendet. Falls eine Verbindung aufgebaut werden konnte, können Daten empfangen werden. Bei der Variable running in der Zeile 14 handelt es sich um einen Typ von boolean, der nur Wahrheitswerte true (richtig oder 1) oder false (0, also nicht richtig) beinhalten kann. Sie ist standardmäßig auf true gesetzt. Sollen keine Daten mehr empfangen werden, kann diese auf false gesetzt werden. Das geschieht mit der Funktion stopp(), die ebenfalls in der Klasse Busware enthalten ist.

Wie schon erwähnt, wird in der Variable rx_length die Länge der empfangenen Daten abgelegt. Falls ein Fehler auftritt, wird die negative Zahl -1 zurückgegeben. Bei der Zahl 0 werden keine Daten empfangen. Wenn die Zahl größer als 0 ist, werden Daten empfangen [vgl. mre]. Der Empfang der Daten erfolgt mit dem Befehl read(…). Beim ersten Parameter handelt sich um den filedescriptor, der die Schnittstelle referenziert, von der gelesen werden soll. Dies ist in diesem Fall der uart0_filestream. Der zweite Parameter ist der Puffer, über den die Daten empfangen werden. Der Puffer hat beim Befehl read(…) den Datentyp void. Die Zahl 255 ist der dritte Parameter. Diese Zahl ist die Größe der zu empfangenen Daten in Bytes [vgl. mre].

Zwischen den Zeilen 22 bis 31 wird überprüft, ob Daten empfangen werden. Dies geschieht anhand der Länge der empfangenen Daten. In der Zeile 22 wird überprüft, ob ein Fehler aufgetreten ist. In der Zeile 27 wird überprüft, falls kein Fehler aufgetreten ist, ob Daten empfangen werden oderv nicht. Falls Daten empfangen werden (ab Zeile 25), werden die in dem Puffer empfangenen Daten in den proto_buffer geladen. Das ist wichtig, weil der rx_buffer einen Datentyp von void hat, also weder eine Zahl noch eine Zeichenkette ist.

In den Zeilen 40 bis 68 werden die Daten entsprechend den empfangenen Daten einsortiert. Dabei werden folgende Zeichen empfangen:

A 13 10

Statt A kann auch B, C oder D empfangen werden. Die jeweiligen empfangenen Zeichen hängen davon ab, an welchem Kanal die Verbrauchswerte empfangen werden. Bei 13 und 10 handelt es sich um einen Zeilenumbruch unter Windows. Dazu kann die Hexadezimalzahl 0x0D0A [vgl. zum], also die einzelnen Hexadezimal zahlen 0D und 0A mit 13 und 10 und der ASCII Tabelle [vgl. ASCII] verglichen werden. Falls Daten, abgesehen von den oben angegebenen, empfangen werden, werden diese in den Zeilen 64 bis 68 verworfen.

6.6.1.2 Verbindung zur Echtzeituhr des Aufsatzes SD0 über den I2C

Da bei einem Start bzw. Neustart der Raspberry Pi nicht die richtige Zeiteinstellung besitzt, bietet sich die Echtzeituhr des SD0 an. Das hat den Grund, dass der Raspberry Pi seine Zeit über das Internet setzt und es beim Einsatz der Messeinheit Fälle geben kann, in denen der Raspberry Pi nicht immer einen Internetzugang hat [vgl. Kof2015 S. 527]. Die Zeit des SD0 wird über den Bus I²C bereitgestellt [vgl. bsd0 Kapitel Specs].

Die Verbindung über den I²-Bus kann über zwei Wege erfolgen. Entweder ähnlich wie beim UART, im vorherigen Kapitel.6.6.1.1. oder über Konsolenbefehle [vgl. pla rtc].

Da über den I²C-Bus nur die Uhrzeit des RTC empfangen und auf dem Raspberry Pi gesetzt werden soll und keine Weiterverarbeitung stattfindet, wird der zweite Weg gewählt. Das heißt es werden lediglich betriebssystemeigene Werkzeuge verwendet.

Um die Verbindung über den I²C herzustellen und die Uhrzeit zu setzen, müssen zunächst einige Einstellung vorgenommen werden. Zunächst muss der I²C Bus konfiguriert werden.

I²C Konfigurieren

Um den I²C-Bus benutzen zu können, muss die Bibliothek i2c-tools installiert werden. Dies geschieht durch die Eingabe folgenden Befehls,

sudo apt-get install i2c-tools

in die Konsole. Die Bibliothek dient dazu, den I²C Bus über die Kommandozeile benutzen zu können [vgl. pla rtc].

Anschließend muss der I²C Bus aktiviert werden. Dazu wird in der Datei /etc/modprobe.d/raspi-blacklist.conf eine Zeilen auskommentiert. Diese sieht dann folgendermaßen aus:

 # blacklist i2c-bcm2708

Um nach jedem Neustart über den I²C-Bus auf den RTC zuzugreifen, werden in die Datei /etc/module folgende Zeilen hinzugefügt [vgl. pla rtc und pla i2c]:

modprobe i2c_bcm2708
modprobe i2c_dev
modprobe rtc-ds1307

Der Treiber rtc-ds1307 in der dritten Zeile, soll den Treiber des RTC DS1338 mit zur Verfügung stellen. Dabei soll es sich um einen bereits auf dem Raspbian installierten Treiber handeln [vgl. hbt].

RTC einbinden

Um die Uhrzeit des RTC DS1338 dauerhaft einzubinden müssen folgende Zeilen in die Datei rc.local eingefügt werden:

```
echo “ds1307 0x68” > /sys/class/i2c-adapter/i2c-1/new_device
hwclock –systohc –D –noadjfile –utc
```

Durch die erste Zeile wird auf den RTC des SD0 zugegriffen. Die zweite Zeile setzt die Uhrzeit des Raspberry Pi durch die Uhrzeit des RTC. Dazu überträgt der Raspberry Pi dem RTC die Zeitsortseinstellung [vgl. pla rtc].

6.6.2 Umwandlung der S0-Impulse

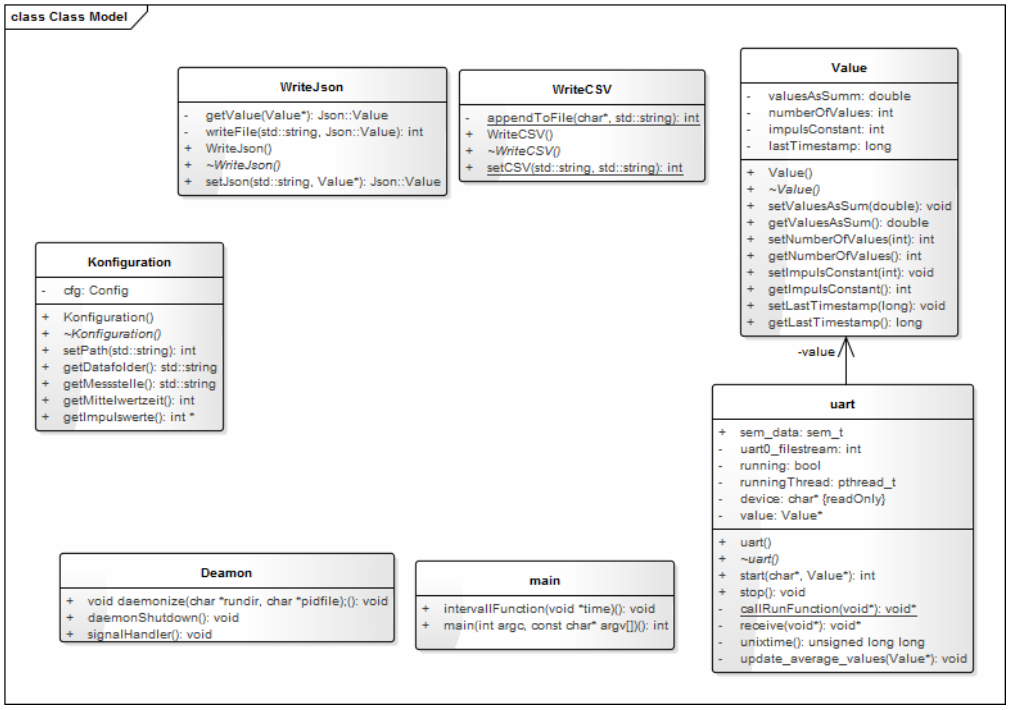
Wie bereits in der Einführung des Kapitels 6 erwähnt, gibt es schon ein Projekt, dass ähnliche Aufgaben durchführt, wie es in dieser Arbeit durchgeführt wird. Im Quellcode von s0enow, werden schon S0-Impulse in Energieverbrauchsdaten umgewandelt. Da für die Umwandlung von S0-Impulsen im Projekt s0enow die Funktion

void update_average_values(struct valuePack *vP)

existiert [vgl. s0ec Zeilen 451 bis 470], wird diese Funktion verwendet.

6.6.3 Klassendiagramm der entwickelten Software

Abbildung 6.1: Klassendiagramm der entwickelten Software



Um den Klassendiagram für die programmierte Software zu erstellen, werden die entwickelten Dateien in das Enterprise Architekt importiert. Enterprise Architekt importiert nur die Headerdateien der C++ Klassen.

Die main und deamon werden nicht importiert, da diese keine Klassen sind, weshalb diese im Enterprise Architekt per Hand erstellt werden müssen. Der unterschied zum Klassendiagramm, das im Kapitel 5.2 vorgestellt wird, ist dass die Klassen ValueFassade, InitialBoardTime und i2c nicht nötig sind.

Die Rolle des ValueFassade nimmt die main.cpp, also die Datei, die die main-Methode beinhaltet, ein.

Die Klassen InitialBoardTime und i2c werden nicht benötigt, da die Uhrzeit des Raspberry Pis nicht über den C++ Code an sich gesetzt, sondern über Konsolenbefehle. Dies basiert darauf, dass die über den I²C empfangenen Daten nicht weiterverarbeitet werden sollen. Über den I²C soll nur die Uhrzeit empfangen werden, die dazu benutzt wird, die Uhrzeit des Raspberry Pi gesetzt zu setzen.

Die Schnittstelle interface connection wird nicht gebraucht, da nur eine einzelne Klasse zur Kommunikation über einen Datenbus benötigt wird.

Die Klasse Deamon wird auch nicht als Klasse erstellt, da es zu beim Signalhandler zu Problemen führte.

Beim importieren wird nur eine Verbindung, durch den Enterprise Architekt, erstellt. Dabei handelt es sich bei der Verbindung zwischen dem ValuePack und der Klasse UART erstellt.

Die Klassen WriteCSV, WriteJson und Konfiguration sowie deamon werden in der main Datei aufgerufen. Diese wird jedoch avom Enterprise Architekt nicht gezeichnet.

6.7 Übertragung der abgespeicherten Verbrauchsdaten an einen Server

Im vorangegangenen Unterkapitel 6.6 wird die Realisierung der Software beschrieben, die die empfangenen S0-Impulse empfängt und auswertet. Bei den ausgewerteten S0-Impulsen handelt es sich um Energieverbrauchsdaten. Diese Verbrauchsdaten werden dann abgespeichert.

In diesem Unterkapitel wird beschrieben, wie die abgespeicherten Verbrauchsdaten an einen Server gesendet werden.

Die Übertragung der abgespeicherten Energieverbrauchsdaten besteht aus zwei bereichen: erstens, die Übertragung der Verbrauchsdaten, an den Server; zweitens, der Import der Verbrauchsdaten in eine Datenbank. Die Datenbank kann eine SQL Datenbank oder aber auch eine NoSQL Datenbank sein.

Wie die Daten übertragen werden, wird in diesem Kapitel beschrieben.

6.7.1 Senden der Verbrauchsdaten vom Raspberry Pi aus an den Server

Die Verbrauchsdaten werden in einem Ordner abgespeichert, die in einer Konfigurationsdatei angegebenen wurde, die von der Software eingelesen werden soll. Der Pfad zum Speicherort wird durch die, in C++ geschriebene Software, ausgelesen, um dort die Energieverbrauchsdaten abzuspeichern. Von dort werden die abgespeicherten Daten, mit einem Skript mit einem auf dem Server liegenden Ordner abgeglichen.

Es gibt zwei Möglichkeiten, die abgespeicherten Dateien an einen Server zu senden: entweder mit der Bibliothek cURL oder mit dem rsync.

Um mit cURL, die abgespeicherten Daten zu übertragen, muss die Bibliothek unter Raspbian installiert werden. Durch Einbindung der Bibliothek ist es möglich, die Übertragung der Dateien, direkt im Quellcode zu programmieren. Es überträgt Dateien in Rechnernetzen und soll unterschiedliche Protokolle, wie FTP SFTP, IMAP usw., unterstützen [vgl. cURL].

Bei der zweiten Möglichkeit geht es darum, die Verbrauchsdaten mit rsync zu übertragen. Es handelt sich dabei um ein Programm, das Dateien und Ordner lokal und über Netzwerk erreichbare Pfade abgleicht. Dabei werden Erstellungsdatum und Größe der Dateien, in der Quell- und Zielpfad verglichen. Haben in dem Quellverzeichniss- bzw. –datei Änderungen stattgefunden, werden Quelle und Ziel abgeglichen [vgl. rsync].

Anzumerken ist, dass Änderungen einseitig übertragen werden. Das heißt, ändert sich etwas in der Quelle, wird es mit dem Ziel abgeglichen, aber nicht umgekehrt [vgl. lwry].

Die Verbrauchsdaten mit rsync zu übertragen bietet sich an, da es nicht nur Dateien an sich übertragen soll, sondern auch verändert Dateiteile.

Hierbei bietet es sich an, dass im Projekt s0enow mitgeliefertes Shell Script, mit dem Namen s0enowSync.sh [vgl. s0enow sync] als Grundlage zu benutzen, da dadurch der Skript zur Synchronisation nicht von Grund auf, von neuem entwickelt werden müsste.

Datenübertragung durch einen Shell Skript als Cron-Job

Das Skript, das die Verbrauchsdaten an einen Server sendet, wird als Cronjob konfiguriert. Ein Cron bzw. ein Cronjob soll Aufgaben automatisch, zu bestimmten Zeiten ausführen [vgl. crol]. Dadurch werden die Daten, in regelmäßigen Abständen, an einen Server gesendet, wo diese dann dargestellt werden sollen. Auf dem Server werden die Daten, in einen bestimmten Ordner gelegt.

Für das Senden liest das Skript aus der Konfigurationsdatei die Serveradresse, an den die Daten gesendet werden sollen und den Ordner, aus dem die Daten gesendet werden sollen.

Da für den Server ein Passwort gebraucht werden kann, bietet es sich an, die Public Key Authentication zu verwenden.

6.7.2 Import der Verbauchsdaten auf dem Server

Auch auf dem Server werden die Daten durch einen Skript in eine MySQL Datenbank Importiert. In die MySQL-Datenbank werden die .csv Dateien Importiert. Die Skripte, die zum Empfang der Daten und dem Import der csv Dateien in eine MySQL-Datenbank zum Einsatz kommen, werden von dem Komillitonen Rene Galow zur Verfügung gestellt, da er schon ein ähnliches Projekt durchgeführt hatte. So müssen die Skripte, die zum importieren Gebraucht werden nicht komplett neu entwickelt werden sondern auf Grundlage von vorhandenen Skripten erstellt werden.

Der Empfang bzw. Import der Verbrauchsdaten besteht aus drei Skripten. Die Erste Datei dient dem start des Importvorgangs. In diesem Skript wird ein Skript mit der angabe des Kompleten Pfades aufrufen. Gleichzeitig wird dem aufgerufenen Skript zwei Parameter übergeben. Der erste parameter ist der Pfad in die die Daten importiert werden. Der zweite Parameter ist eine Konfigurationsdatei, in der die Datenbank und die Tabellen stehen, in die die Verbrauchsdaten Importiert werden sollen.

6.8 Webdarstellung der Verbrauchsdaten

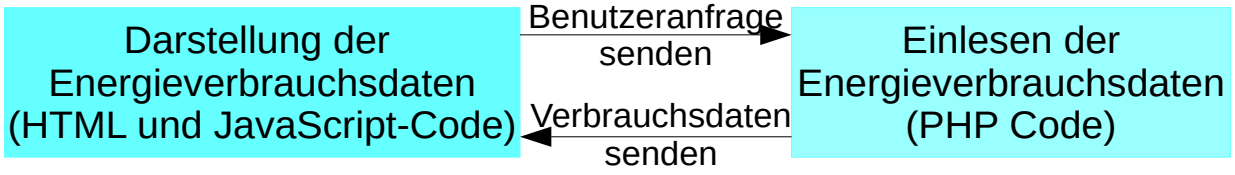
Die Verbrauchsdaten werden über eine Weboberfläche im Browser angezeigt. Dargestellt werden die Verbrauchsdaten mit dem Framework Hightcharts. Hightcharts bietet die Funktionalität an die Verbrauchdaten in Graphen darzustellen und erzeugt dabei ein Koordinatensystem, beidem eine Achse die Zeitachse ist, die den Verbrauchszeitpunkt darstellt. Die andere Achse repräsentiert die Verbrauchswerte. Die Verbräuche werden dann als Punkt im Koordinatensystem, im Verhältnis Energieverbrauch pro Zeiteinheit gezeichnet. Die Punkte werden, durch Hightchart, dann mit Strichen miteinander verbunden.

Um die Verbrauchsdaten darzustellen müssen diese gelesen werden. Zum Einlesen der Verbrauchsdaten gibt es zwei Möglichkeiten entweder werden die Daten über einen SQL-Server geladen oder der Verbrauch kann über Json Dateien gelesen werden.

HTML bildet also das Grundgerüst für die Webdarstellung. Hightcharts zeichnet die Verbrauchsdaten und PHP baut eine Verbindung zur Datenbank auf. Die Auswahl trifft bewusst die Skriptsprache PHP. Nicht nur weil PHP einer der meist benutzten Skriptsprachen sein soll. PHP wird nicht vom Browser interpretiert, sondern erst vom Webserver geparst, wenn es vom Webbrowser aus angefordert wird. Im Browser wird nicht der PHP-Code dargestellt, sondern das Ergebnis des PHP-Skriptes, nachdem es in Bytecode übersetzt, interpretiert und ausgeführt wird. Zudem bietet PHP auch die Anbindung an Datenbanken, wie MySQL [vgl. Balweb2011 S.182]. Das bedeutet, dass die Abläufe nicht sichtbar sind, wenn die PHP-Datei ausgeführt wird.

Für die Webdarstellung gibt es drei Dateien, welche den programmierten Code, die die Verbrauchsdaten darstellen sollen, beinhalten. Die erste Datei ist die HTML-Datei, die im Webbrowser dargestellt wird. Auf der HTML-Seite wird die Darstellungsart ausgewählt und der Zeitraum für die Darstellung vergangener Verbrauchswerte angegeben. Falls die aktuellen Verbrauchsdaten dargestellt werden sollen, ist die angebe des Zeitraumes, also des (Anfangs- und Enddatums nicht möglich. Zusätzlich müssen die kanäle angegeben werden, die Dargestellt werden sollen. Die zweite Datei beinhaltet den JavaScript Code, der die Verbrauchsdaten darstellt. Die JavaScriptdatei wird von der HTML-Seite eingebunden.

Abbildung 6.2: Zusammenhang der Benutzerabfrage und Verbrauchsdatenabfrage



Die Anfrage des Benutzers wird mit einem Button, an eine Javascript Funktion, die sich in einer JavaScript Datei befindet. Bei der Funktion handelt es sich um drawChart(). Diese Funktion ruft auch die JavaScript Funktion InitHighChart(von, bis) auf, die in Listing 6.6 dargestellt ist. Dieser Zeichnet die Verbrauchsdaten. Dazu ruft es für die Datenabfrage die dritte Datei, die PHP-Datei auf. Die Verbrauchsdaten werden von der PHP-Datei gelesen und an InitHighChart(von, bis), zurückgesendet, die die Energieverbrauchsdaten dann darstellt.

Als Webserver wird XAMPP von Apache verwendet, da es sowohl einen PHP-Parser, als auch ein MySQL-Datenbank anbietet. Es kann jedoch auch jede beliebiger Webserver benutzt werden, der eine Datenbankschnittstelle und einen PHP-Parser anbietet.

6.8.1 Zeichnen der Energieverbrauchsdaten

Die Darstellung der Energieverbrauchsdaten erfolgt durch Hightcharts. Der folgende Code-Ausschnitt zeichnet die Verbrauchsdaten in einem Koordinatensystem auf der HTML Seite:

Listing 6.6: Hightchart Code, zum Zeichnen des Graphen für Verbrauchswerte

```
1  function InitHighChart(von, bis) {
2                          var checkbox_value = "";
3                          $(":checkbox").each(function () {
4                            var ischecked = $(this).is(":checked");
5                            if (ischecked) {
6                              checkbox_value += $(this).val()+"=true" + "&";
7                            }else{
8                              checkbox_value += $(this).val()+"=false" + "&";
```

```
                    }
                  });

                  var allPOSTparameter =
"beginDatum="+von+"&endDatum="+bis+"&"+checkbox_value;

                  var options = {
                    chart: {
                      renderTo: 'chart',
                    },
                    credits: {
                      enabled: false
                    },
                    title: {
                      text: 'Timestmp Overview',
                      x: -20
                    },
                    xAxis: {
                      categories: [{}]
                    },
                    tooltip: {
                      formatter: function() {
                        var s = '<b>'+ this.x +'</b>';

                        $.each(this.points, function(i, point) {
                          s += '<br/>'+point.series.name+': '+point.y;
                        });

                        return s;
                      },
                      shared: true
                    },
                    navigation: {
                      buttonOptions: {

                        enabled: true
                      }
                    },

                    series: [{}, {}, {}, {}]
                  };

                  $.ajax({
                    url: "http://localhost/Visualisierung/php/getgraph.php",
                    data: allPOSTparameter,
                    type:'post',
                    dataType: "json",
                    success: function(data){
                      options.xAxis.categories = data.timestamp;
                      options.series[0].name = 'ch1';
                      options.series[0].data = data.ch1;
```

```
60                      options.series[1].name = 'ch2';
61                      options.series[1].data = data.ch2;
62                      options.series[2].name = 'ch3';
63                      options.series[2].data = data.ch3;
64                      options.series[3].name = 'ch4';
65                      options.series[3].data = data.ch4;
66                      var chart = new Highcharts.Chart(options);
67                  }
68              });
69          }}
```

Im Listing 6.6 wird zwichen den Zeilen drei bis zehn überprüft, welche Kanäle ausgewählt wurden. In den Zeilen 12 bis 13 wird ein String zusammengesetzt, um es an die PHP Datei zu schicken. Die PHP Datei wird im in der Zeile 52, aufgerufen. Die PHP Datei baut eine Verbindung zur MySQL Datenbank auf, startet eine Datenbankanfrage und sendet die empfangenen Daten an den Graphen, welcher dann die Verbrauchsdaten zeichnet. Die vom Hightchart empfangenen Daten werden dann zwischen den Zeilen 55 und 66 einsortiert.

6.8.2 Abruf von Energieverbrauchsdaten

Um die abgespeicherten Energieverbrauchsdaten, darstellen zu können, wird eine Anbindung an eine Datendarstellungsform wie JSON oder eine Datenbankanbindung zu einem SQL-Server benötigt.

Die PHP-Datei zur Anbindung zur Datenbank sieht folgendermaßen aus:

Listing 6.7: Code, zum Verbinden zur einer Datenbank

```
<?php

//Variablen
   $servername = "localhost“
   $username = "root";
   $password = "";
   $dbname = "bachelorarbeit";
   $str = "";
   $graph_data = array();
   $categories = array();

   // Verbindung zum Server und Datenbank aufbauen.
   $conn = new mysqli($servername, $username, $password, $dbname);

   if ($conn->connect_error) {
      die("Connection failed: " . $conn->connect_error);
   }
   $str.= " daten.Zeitstempel <= '".$_POST['endDatum']."' AND daten.Zeitstempel
>='".$_POST['beginDatum']."'";
      $channels = array();
      if ($_POST['A'] == "true") array_push($channels, "'A'");
      if ($_POST['B'] == "true") array_push($channels, "'B'");
```

```
        if ($_POST['C'] == "true") array_push($channels, "'C'");
        if ($_POST['D'] == "true") array_push($channels, "'D'");
        $str .= " AND sensor.Zaelernummer in (" . join(',', $channels) . ")";

    $sql = "SELECT daten.Zeitstempel, daten.Wert, sensor.Zaelernummer
                FROM daten INNER JOIN sensor
                  ON daten.Sensor_ID = sensor.ID
                WHERE " .$str ;
    //Datenbank abfrage
    $result = $conn->query($sql);
  //Überprüfen, ob Daten empfangen wurden
    if ($result->num_rows > 0) {
        //Empfangene Daten Einsortieren
        $ch1 = array();
        $ch2 = array();
        $ch3 = array();
        $ch4 = array();
        while($row = $result->fetch_assoc()) {

            //Zeitstempel einem Array übergeben
            $timestamp = "";
            if($timestamp != $row["Zeitstempel"]){ //Array wird einem
                $timestamp = $row["Zeitstempel"];
                array_push($categories, $row["Zeitstempel"]);
            }

            $graph_data['timestamp']= $categories;

            if  ($_POST['A'] == "true" && $row["Zaelernummer"]  == 'A') {
                array_push($ch1, floatval($row["Wert"]));

            }

            if  ($_POST['B']== "true" && $row["Zaelernummer"]  == 'B') {
                array_push($ch2, floatval($row["Wert"]));

            }

            if  ($_POST['C']== "true" && $row["Zaelernummer"]  == 'C') {
                array_push($ch3, floatval($row["Wert"]));

            }
            if  ($_POST['D']== "true" && $row["Zaelernummer"]  == 'D') {
                array_push($ch4, floatval($row["Wert"]));

            }
        }
        $graph_data['ch1']= $ch1;
        $graph_data['ch2']= $ch2;
        $graph_data['ch3']= $ch3;
        $graph_data['ch4']= $ch4;
```

```
    }

    $conn->close();
    echo json_encode($graph_data);
exit;
?>
```

Die Programmiersprache PHP bietet selbst die Möglichkeit, mit Datenbanken eine Verbindung aufzubauen [vgl. Balweb2011 S.182], weshalb für die Datenbankanbindung an sich kein Framework benötigt wird.

In den Zeilen 12 bis 16 wird die Verbindung zu einem Server ($servername) aufgebaut, auf dem sich eine Datenbank befindet. Dazu muss man auch den Benutzernamen ($username) und Passwort (password) sowie die Datenbank ($dbname) gesetzt werden. Die dazu nötigen Variablen werden in den Zeilen 3 bis 9 gesetzt. In den Zeilen 17 bis 29 wird eine Datenbankanfrage aufgebaut, um die Verbrauchsdaten in der Zeile 31 anzufordern, die dargestellt werden sollen. Dazu werden in den Zeile 17 und 18 das Anfangs- und Enddatum abgefragt, das auf der HTML-Seite im Webbrowser, vom Benutzer eingegeben wurden, sowie is in den Zeilen 20 bis 24, die Checkboxen überprüft, welche angeklickt wurden, um die Werte der entsprechenden Kanäle darzustellen. Anzumerken ist hierbei, dass die Kanäle den S0-Anschlüssen des Busware SD0 entsprechen. In den Zeilen 26 bis 29 wird die Benutzereingabe mit einer vorgefertigten SQL-Abfrage zusammengefasst und in Zeile 31 dem Datenbankserver gesendet.

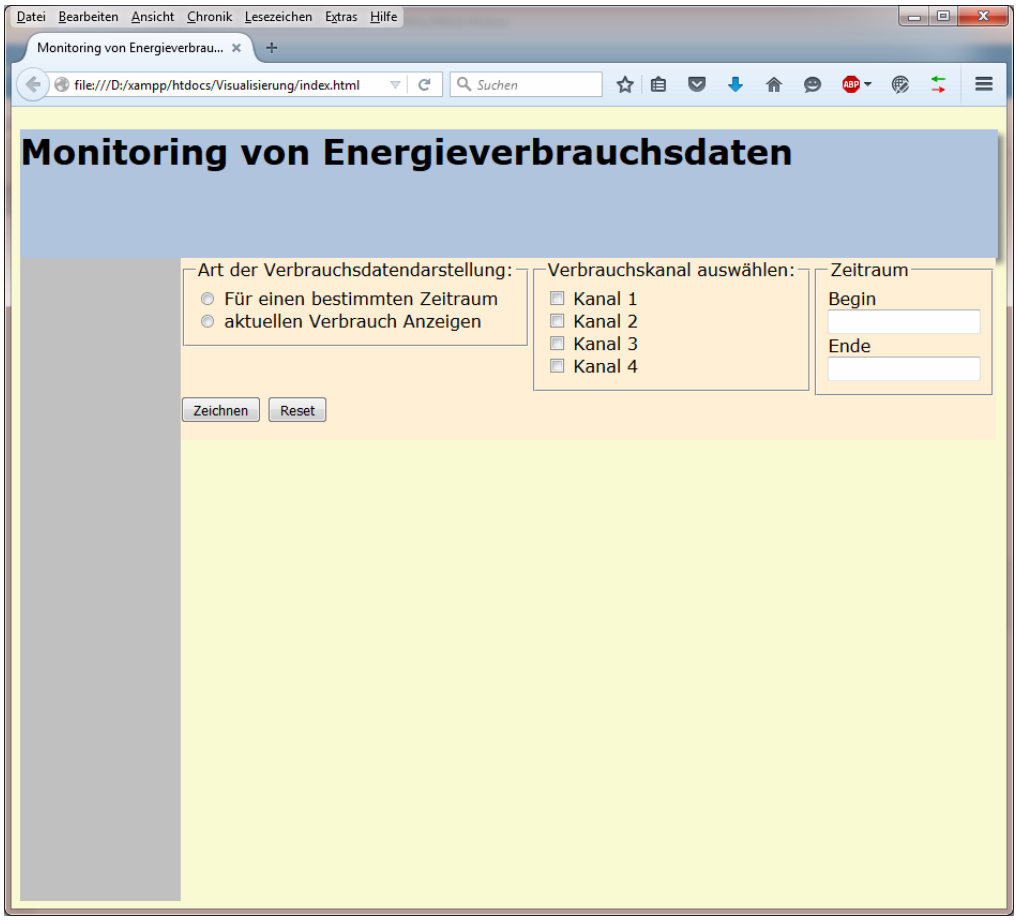
Für die grafische Darstellung werden lediglich der abgespeicherte Zeitstempel und die Verbrauchswerte benötigt. Die Zählernummer wird dazu benötigt, um die Energieverbrauchswerte bei der Darstellung entsprechend richtig einsortieren zu können.

Zwischen den Zeilen 39 bis 73 werden die abgefragten Energieverbrauchswerte einsortiert. Dazu wird in der Zeile 39 zunächst überprüft, ob für die Datenbankanfrage Daten vorhanden sind. Solange Daten vorhanden sind werden die Daten, zwischen den Zeilen 43 bis 72, auch einsortiert.

Die Datenbankabfrage gibt das Abfrageergebnis zeilenweise zurück. Beim Ergebnis handelt es sich um einen assoziativen Array. Das bedeutet, dass auf die einzelnen Werte bzw. Zellen der zurück gelieferten Datenzeile, durch Angabe der Spaltenüberschrift zugreifen kann.

In der Zeile 75 wird die Verbindung zum Server geschlossen und die Daten mit der Angabe des Befehls echo json_encode($graph_data); an die JavaScript Funktion gesendet, die die PHP-Datei aufgerufen hat.

Abbildung 6.3: Abbildung der Weboberfläche



7 Inbetriebnahme

Im vorherigen Kapitel wird die Realisierung der Softwarelösung beschrieben. In diesem Kapitel wird die Inbetriebnahme erläutert.

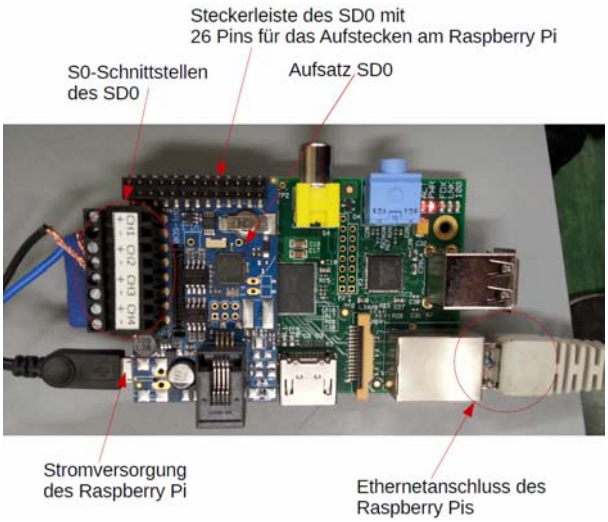
Um die Softwarelösung in Betrieb zu nehmen, muss zunächst das Betriebssystem Raspbian auf eine SDHC-Karte installiert werden. Dies geschieht an einem Computer, auf dem das Betriebssystem Windows 7 Läuft. Raspbian wird mit dem Programm Win32 Disk Imager auf die SDHC-Karte gebracht. Dazu wird die SD-Karte in einen Kartenleser gesteckt, der mit dem Computer verbunden ist.

7.1 Technischer Aufbau

Nachdem Raspbian auf die SD-Karte gebracht wird, wird die SD-Karte in den SD-Karte in den SD-Karten Slot des Raspberry Pis gesteckt. Nach der Vorbereitung der SDHC-Karte wird der Aufsatz SD0 von Busware, wird auf die GPIO-Steckleiste des Raspberry Pi aufgesteckt. Anschließend werden die S0-Schnittstellen des Aufsatzes SD0, mit den S0-Schnittstellen der Zählern angeschlossen, von denen die Verbrauchswerte gemessen werden sollen. Der Raspberry Pi wird anschließend, über dessen RJ45-Buchse, mit einem Ethernet Kabel an einem Netzwerk bzw. Internet angeschlossen.

Danach folgt der Stromanschluss des Raspberry Pis. Damit fährt der Raspberry Pi hoch und das Betriebssystem Raspbian startet.

Abbildung 7.1: Technischer Aufbau der Inbetriebnahme



7.2 Softwaretechnische Inbetriebnahme

Nach dem Start des Raspberry Pi und dem Hochfahren des Betriebsystems muss zunächst der g++ Compiler installiert werden, da für die Programmierung C++ benutzt wird. Dies geschieht mit dem Befehl,

sudo apt-get install g++

Der Aufsatz SDO hat eine eigene Firmware. Das Aufladen der SD0 eigenen Firmwares erfolgt mit einem Perl-Skript. Sowohl die Firmware als auch das Perl-Skript, werden von Busware bereitgestellt. Mit dem Perl-Skript wird die Firmware vom Raspberry Pi aus auf den Aufsatz aufgeladen. Um den Aufsatz SD0 mit der Firmware aufzuladen, muss daher ein Perl-Compiler installiert werden. Die Installation des Perl-Skriptes erfolgt folgendermaßen:

sudo apt-get install perl

Um anschließend die entwickelte Software einsetzen zu können, wird libconfig installiert. Dies geschieht mit der folgenden Kommandozeile:

sudo apt-get install libconfig++-dev

Der Parameter libconfig++-dev installiert die Konfigurationsbibliothek Libconfig. Mit Hilfe der Bibliothek wird die Konfigurationsdatei eingelesen, die für den Betrieb der Software benötigt wird.

7.3 Einbinden des I²C

Um über den I²C-Bus den RTC einbinden zu können wird der Treiber für den Bus installiert:

sudo apt-get install i2c-tools

Anschließend wird der I²C-Bus aktiviert. Dazu wird in der Datei /etc/modprobe.d/raspi-blacklist.conf folgende Zeilen auskommentiert:

 # blacklist i2c-bcm2708

Um nach jedem Neustart über den I²C-Bus auf den RTC zuzugreifen, werden in die Datei /etc/module folgende Zeilen hinzugefügt:

modprobe i2c_bcm2708
modprobe i2c_dev
modprobe rtc-ds1307

Um die RTC dauerhaft einsetzen zu können muss die entsprechende Schnittstelle eingefügt werden. Dies geschieht folgendermaßen:

echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
sudo hwclock –s

Die aufgelisteten Zeilen müssen am Ende der jeweiligen Datei, noch vor der Zeile exit 0, eingefügt werden.

7.4 Einsatz der entwickelten Software

Die Verbrauchswerte werden über die serielle Schnittstelle (UART) übertragen. Um den UART frei zu schalten muss in der Datei /etc/inittab am Anfang der Zeile,

T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100

ein Raute (#) Zeichen eingefügt werden [vgl. fua].

Die Freigabe erfolgt in der Datei rz.local im Linux Verzeichnis /etc/. Dorthin müssen die Folgenden Zeilen eingefügt werden:

chmod 777 /dev/ttyAMA0

Diese Zeile ändert die Zugriffsrechte auf den UART, dies ist notwendig, da sonst die entwickelte Software auf dem Raspberry Pi nicht auf den UART zugreifen könnte. Anschließend wird die entwickelte Software kompiliert, die zuvor auf dem Raspberry Pi geladen wird. Die Kompilierung kann mithilfe eines Makefiles geschehen. Makefiles dienen der Ausführung von Konsolenbefehlen bzw. dem Kompilieren des Quellcodes. Der Makefile wird beim erstellten Eclipse-Project mit erzeugt. Die C++ Software wird im Verzeichnis

/usr/local/src/bachelorarbeit

abgelegt. Dieses Verzeichnis dient als Ablageort für Quellcodes lokal installierter Software. In das Verzeichnis,

/*usr/local/sbin/*

wird dann die Objektdatei abgelegt.

Die Konfigurationsdatei wird im Linux-Verzeichnis /etc/ abgelegt. Dieses Verzeichnis bietet sich an, da die Software, die auf dem Raspberry Pi läuft, als Dienstprogramm bzw. Deamon läuft und in dem Verzeichnis Konfigurationsdateien vorgesehen sein sollen [vgl. lvz].

Um die abgespeicherten Verbrauchsdaten an einen Server zu senden und diese dort in eine Datenbank zu Importieren, werden die Skripte benutzt, die im Kapitel 6.7. erläutert wurden.
Diese Skripte werden als Cronjobs realisiert.
Der Synchronisierungen Intervall der abgespeicherten Daten sollte gleich oder kleiner oder gleich groß sein als die Zeit, indem die Verbrauchsdaten auf dem Raspberry Pi abgespeichert werden. Würde der Übertragungsintervall größer als dass die Verbrauchsdaten abgespeichert werden, wären einige Synchronisationsversuche umsonst, da zwischen mindestens zwei Synchronisationsversuchen keine Verbrauchsdaten abgespeichert werden würden.

Die Erstellung der cronejobs erfolgt über die Konsole. Dort muss der Befehl,

crontab –e

angegeben werden.

Falls es bisher keinen Standarteditor gibt, wird nachgefragt, welcher Editor benutzt werden soll. Falls schon vorher ein editor verwendet wurde, wird dieser verwendet.

In der Inbetriebnahme wird der Editor vim, auf Grund der vorhandenen Erfahrung, benutzt.

Um das Skript zum Senden der abgespeicherten Verbrauchsdaten als Cronjob zu realisieren, wird folgende Zeile am Ende der Datei eingefügt:

** 1 * * * /home/pi/sync.sh*

Hier werden die Verbrauchsdaten stündlich gesendet. Beim /home/pi/sync.sh handelt es sich um einen absoluten Pfad zum Skript der als Cronjob ausgeführt werden soll.

Visualisierung der Verbrauchsdaten

Um die Verbrauchsdaten zu visualisieren, wird XAMPP verwendet. Neben dem Apache Webserver und einem PHP-Parser, werden auch zusätzliche Softwares auf dem Server installiert. Dazu zählt auch die Datenbank MySQL, die für Abfragen verwendet wird, welche von der Weboberfläche durch den Benutzer gesendet werden.

Um XAMPP zu installieren wird es zuerst con dessen Webseite in den Ordner /Downloads herunter geladen. Anschließend wir XAMPP durch Angabe des Befehls,

sudo ./xampp-linux-1.8.3-4-installer.run

in der Konsole, installiert. Nach dem Start der Installation müssen die Anweisungen der grafischen Oberfläche befolgt werden, um die Installation abzuschließen.

Die programmierte Webseite wird im Webserver, in dessen Ordner /htdocs abgelegt. Dazu wird dort ein Unterordner mit der Bezeichnung Visualisierung erstellt. Die Darstellung wird dann folgendermaßen erreicht:

http://Serveradresse/Visualisierung/

Auf dem Server werden die in .csv Dateien abgespeicherten Verbrauchsdaten durch einen Skript in die MySQL Datenbank importiert. Diese Skripte sind auch als Cronjob realisiert. Dieser Cronjob wird ähnlich wie der Cronjob zum Senden der abgespeicherten Verbrauchsdaten erzeugt. Was sich ändert ist der Pfad zum Skript und der Benutzer, unter dem der Cronjob erzeugt wird.

Für die Inbetriebnahme wird der Zähler Voltcraft DPM32L1-D verwendet. Dieser hat eine Impulskonstante von 0,5 Wh/Imp. Laut Datenblatt erzeugt es 2000 Impulse pro Watt Stunde [vgl. dpm].

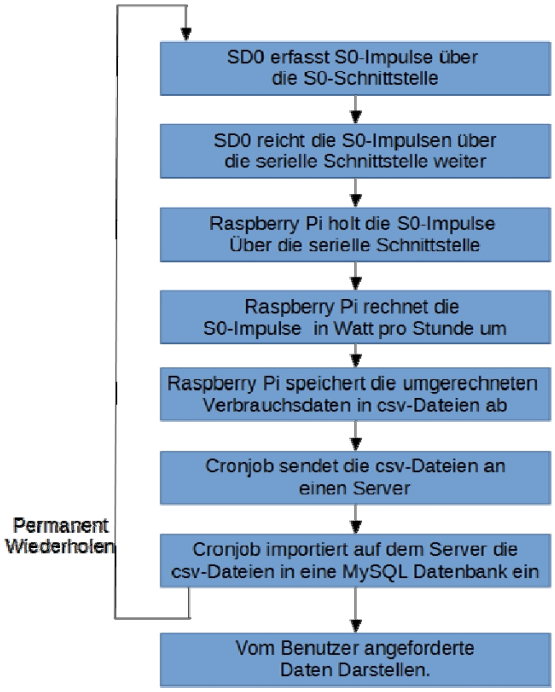
Das Ergebnis der Inbetriebnahme ist im nächsten Kapitel sichtbar.

8 Test

Bisher wird die Entwickelung, Realisierung und Inbetriebnahme beschrieben. In diesem Kapitel werden die Tests beschrieben, die zur Überprüfung der realisierten Software durchgeführt werden. Der Test hat den Sinn, die Funktionsfähigkeit von Softwareteilen bzw. des Softwares zu validieren, ob diese macht, was diese machen soll.

Neben den Unit Test gibt es die Möglichkeit, die Funktionalität der entwickelten Lösung, durch die Inbetriebnahme zu validieren. Dabei werden die Ergebnisse durch Nachprüfen überprüft.

Abbildung 8.1: Ablaufplan von der Erfassung der S0-Impulsen bis zur Visualisierung



Wie aus der Abbildung 8.1. hervorgeht, werden die S0-Impulse vom Aufsatz erfasst und über die serielle Schnittstelle, also dem UART, an den Raspberry Pi weiter gegeben. Anschließend werden die S0-Impulse, in Watt pro Stunde Umgerechnet. Diese Verbrauchswerte werden dann in .csv-Dateien abgespeichert. Die abgespeicherten Verbrauchsdaten werden von einem Skript an einen Server versendet, von wo aus diese Dargestellt werden können.

Die csv Daten sehen folgendermaßen aus:

2015-06_29 14:01:13;3.559;0.000;0.000;0.000

Bis zum Ersten Semikolon handelt es sich um einen Timestamp. Anschließend werden die Verbrauchswerte der einzelnen Daten, getrennt mit Semikolon, dargestellt.

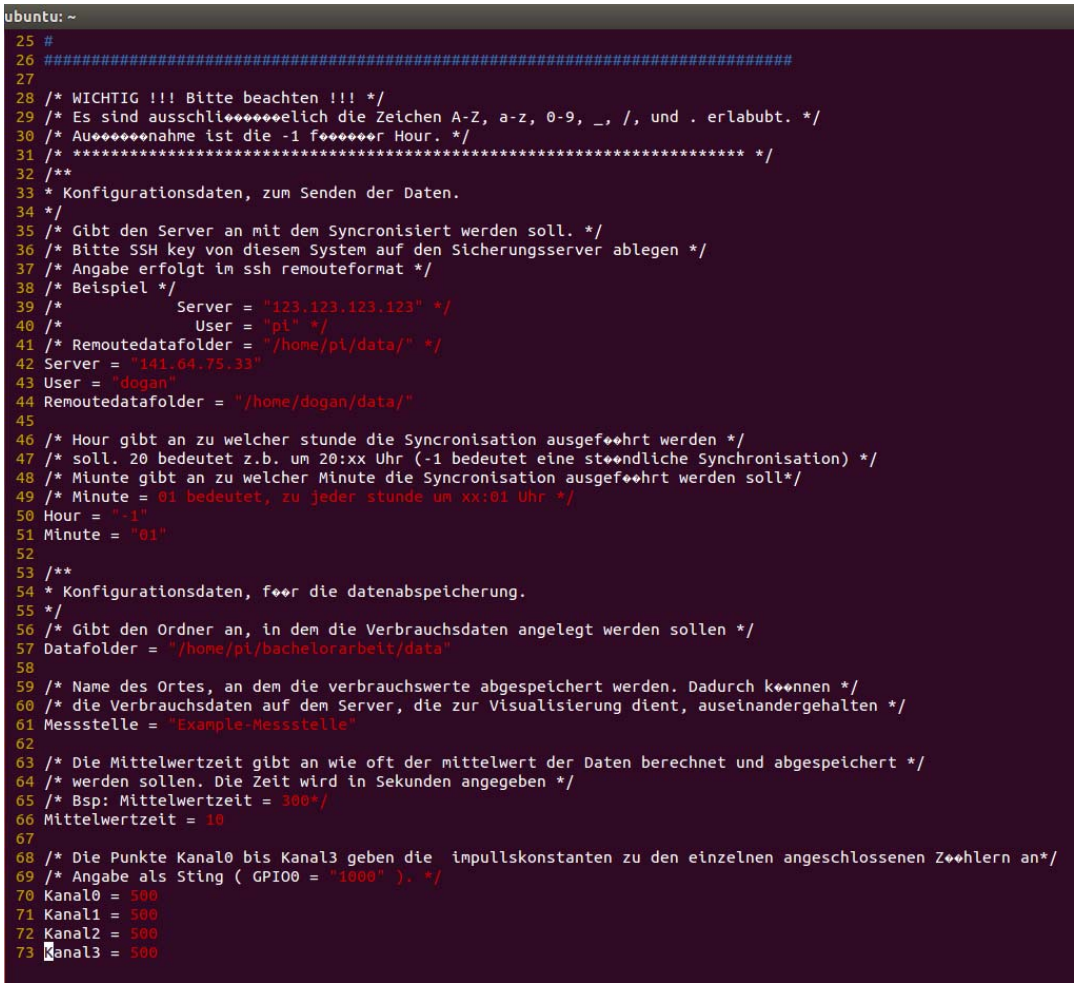
Die einzelnen Wertebereiche sind mit einem Semikolon getrennt. Der erste Bereich, auf der Linken Seite, stellt den Zeitstempel dar. Dass heißt den Datum und Uhrzeit, an den die Verbrauchsdaten abgespeichert wurden. Gefolgt von vier weiteren Bereichen, in die die einzelnen Verbrauchswerte abgespeichert werden, die von den Zählern gemessen wurden.

Nach dem Zeitstempel folgt der Verbrauchswert, die am ersten Kanal des SD0 gemessen wurde. Darauf der Verbrauchswert vom zweiten Kanal bis zum vierten Kanal des SD0, der sich an letzter Stelle befindet.

8.1 Überprüfungsbedingungen

Um die Funktionalität zu validieren müssen als erstes die Parameter der Konfigurationsdatei festgelegt werden.

Abbildung 8.2: Inhalt der Konfigurationsdatei



Diese Parameter sind in der Abbildung 76.der Konfigurationsdatei sichtbar.

Der Inhalt der Konfigurationsdatei ist in der Abbildung 8.2 sichtbar. Der Speicherort der csv Dateien wird durch die in der Konfigurationsdatei festgelegten Werte der Namen Datafolder und Messstelle gebildet. Die csv Dateien werden also im Ordner

/home/pi/bachelorarbeit/data/Example-Messstelle/

Abgelegt. Dort werden dann die Verbrauchsdaten in die csv-Dateien eingefügt. Von hier werden dann die csv-Dateien an einen Server Versendet.

Die csv-daten werden an den, in der Zeile 42 stehende Serveradresse versendet. Dazu wird der Konfigurationsname User, in der Zeile 43, für die Authentifizierung auf dem Server benutzt. Dort werden die Daten unter dem Pfad, die unter Remoutedata-folder in der Zeile 44 angegeben wird, abgelegt.

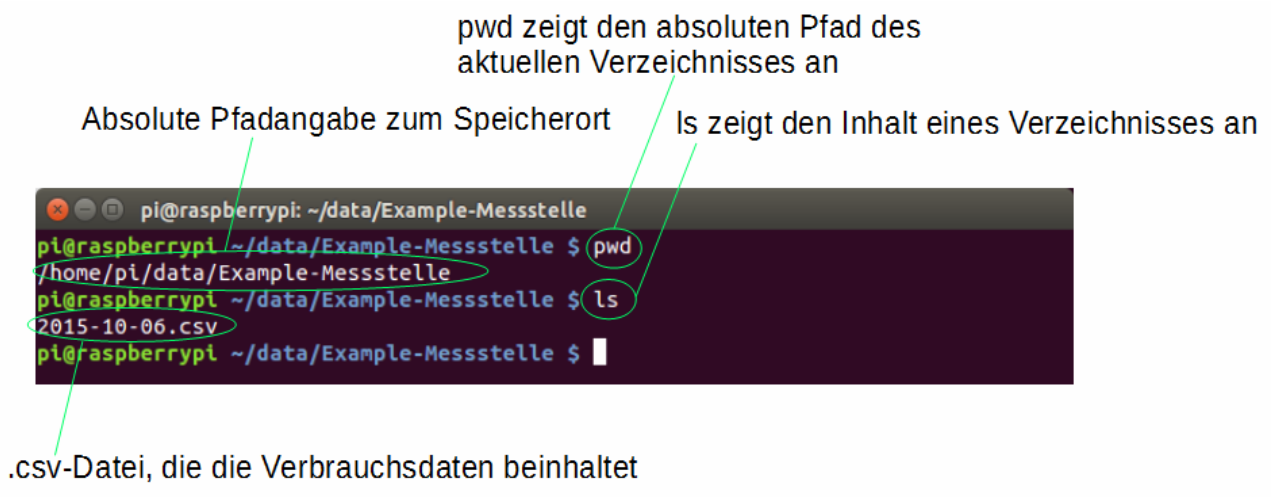
8.2 Überprüfung der Funktionalität

Nach dem Ablaufplan, in der Abbildung 8.1. werden folgende Überprüfungsfragen abgeleitet:

- werden die csv-Dateien, in denen die Energieverbrauchsdaten abgespeichert werden, im richtigen Pfad abgelegt?
- Erfolgt die Übertragung der Verbrauchsdaten an den, in der Konfigurationsdatei festgelegten Rechner?
- Werden die Daten in den MySQL Server importiert?
- Werden die Verbrauchsdaten Visualisiert?

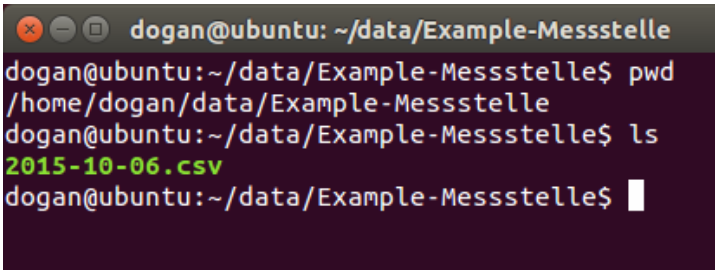
Auf dem Raspberry Pi werden die Verbrauchsdaten gemessen und abgespeichert.

Abbildung 8.3: Auf dem Raspberry Pi befindliche csv-Dateien



Die Abbildung 8.3 zeigt die auf dem Raspberry abgespeicherte .csv-Datei, die die verbrauchswerte beinhaltet.

Abbildung 8.3: Auf dem Server befindliche csv-Dateien



Um die Funktionalität der Webseite zu validieren, werden Testdaten mit einem Pythonskript generiert, in eine MySQL Datenbank importiert und dargestellt.

Zur Validierung können dann einzelne Verbrauchswerte auf der Darstellung ausgewählt und mit den Verbrauchswerten verglichen werden. Dazu müssen der Datum und der Wert eines einzelnen Kanals gemerkt werden.

8.3 Ergebnis der Überprüfung

In diesem Unterkapitel werden die Ergebnisse der Funktionsüberprüfungen behandelt. Für die, im vorherigen Kapitel festgelegten Überprüfungsfragen, wird folgende Tabelle aufgestellt:

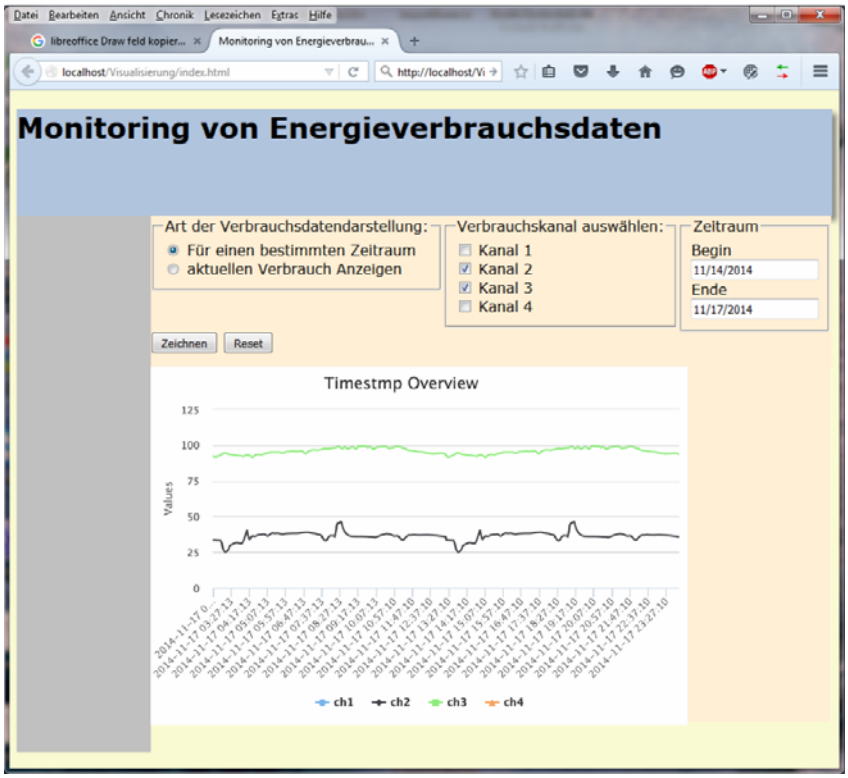
Tabelle 8.1: Prüftabelle der Inbetriebnahme

Nr.	Überprüfung	Status
1	Die .csv Dateien werden in der im Konfigurationsdatei festgelegten Pfad abgespeichert.	Ok
2	Übertragung der .csv Dateien, an den im Konfigurationsdatei festgelegten Server.	Ok
3	Import der .csv Dateien in eine MySQL Datenbank	Ok
4	Visualisierung der vorher übertragenen Verbrauchsdaten	Ok

Die mittlere Spalte, also die in der Spalte Überprüfungen gelisteten Kriterien, sind die Prüfkriterien. Diese Wirden im kapitel 8.2 in form von fragen erstellt. Bei der rechten Spalte, also die Spalte mit der Überschrift Status, handelt es sich darum, ob der erfolgreich war oder nicht. Hier bedeutet Ok, dass der Test erfolgreich ist.

Die Webseite mit der Visualisierung der Verbrauchsdaten sieht folgendermaßen aus:

Abbildung 8.4: Auf dem Server befindliche csv-Dateien



9 Zusammenfassung und Ausblick

In dieser Abschlussarbeit wurde eine, aus zwei Bereichen bestehende, Softwarelösung entwickelt, die den eigenen Energieverbrauch visualisiert und dadurch die Möglichkeit aufgreift den eigenen Energieverbrauch zu analysieren. Die auf der Grundlage eines Linux-Betriebssystems entwickelte Software, welche auf einem Raspberry Pi läuft, wertet die empfangenen S0-Impulse in Energieverbrauchswerte um. Auf der Grundlage, von diesen als Linux-Treiber (auch Deamon bezeichnet) entwickelten Software ausgewerteten Verbrauchswerten, findet die Visualisierung, der ausgewerteten Energieverbrauchswerte statt. Die Visualisierungseinheit, also die Webseite befindet sich dann auf einem Webserver, der entweder auf einem Raspberry Pi oder einem vom Raspberry Pi physisch unabhängigen Server läuft. Die entwickelte Visualisierungseinheit bietet den Vorteil, dass nicht nur der bisherige Gesamtverbrauch, vom Zähler als Zahl abgelesen werden kann, sondern die Verbrauchswerte für einen bestimmten Zeitraum grafisch darzustellen werden. Zudem ist es durch die Auswahl der einzelnen Kanälen, die der Anzahl der S0-Anschlüssen der Erweiterung SD0 entsprechen, möglich die Verbrauchswerte verschiedener Zähler, gleichzeitig zu analysieren und miteinander zu vergleichen. Es wurde eine Anforderungsanalyse durchgeführt, die die Anforderungen an die entwickelte Softwarelösungen definieren soll. Mit deren Hilfe wurden dann im anschließenden Kapitel die Lösungen entworfen. Ausgehend vom Entwurf, der die Grundlage bzw. den Ausgangspunkt der realisierten Lösungen bildet, wurde die Lösung realisiert. Es wurde auch ein Parser zum Einlesen von Konfigurationsdateien verwendet. Dadurch kann die Software während der Laufzeit an verschiedene Zähler angepasst werden. Die Software, die auf dem Raspberry Pi läuft, wurde modular aufgebaut. Dadurch ist später eine einfache Erweiterung der schon entwickelten Software möglich.

Der Vorteil bei der in dieser Abschlussarbeit eingesetzten S0-Schnittstelle ist, dass die Schnittstelle nicht nur für den Energieverbrauch, sondern auch für andere Verbrauchswerte wie Wärme, Wasser und Gas verwendet werden kann. Die entwickelte Software kann also in Verbindung mit den S0-Schnittstellen vielseitig verwendet werden.

Eine Möglichkeit den Prototyp zu erweitern wäre, weitere Darstellungsformen und –varianten hinzuzufügen. Das wäre beispielsweise die Darstellung der Verbrauchsdaten in einer App, also auf einem Smartphone oder Tablet. Es wäre auch möglich, weitere Daten, wie Tarifinformationen und bisherige Kosten für den Energieverbrauch, darzustellen. Es wäre auch möglich, die aktuellen Energieverbrauchswerte mit den Verbrauchswerten des selben Zeitraums vergangener Jahre zu vergleichen.

Quellenverzeichnis

[Dem2013] Dembowski, K., „Raspberry Pi – Das Handbuch“, Springer Fachmedien Wiesbaden 2013

[Kof2015] M. Kofler, C. Kühnast,, C. Scherbeck, „Raspberry Pi : Das umfassende Handbuch ; [aktuell zu allen Versionen, inkl. Modell B ; Grundlagen verstehen ; spannende Projekte realisieren ; Schnittstellen des Pi, Schaltungsaufbau, Steuerung mit Python ; Erweiterungen für den Pi: Gertboard, PiFace, Quick2Wire u.a. in Hardware-Projekten einsetzen]“ 1. Aufl., 1. korrigierter Nachdr. Ed., Galileo Press, Bonn, 2015.

[Upt2014] E. Upton, G. Halfacree, „Raspberry Pi User Guide“ (2. Auflage), John Wiley &Sons Ltd, Chichester, West Sussex 2014

[Mes2003] H. Messmer, K. Dembowski, „PC Hardwarebuch. Aufbau, Funktionsweise, Programmierung“, 7.Auflage, Addison-Wesley, München,2003

[Bei2011] Beierlein, Thomas, Hagenbruch,Olaf, u.a. „Taschenbuch Mikroprozzesoretchnik“, Carl Hanser Verlag, München, 2011

[Bal2011] Balzert, Heide, „Lehrbuch der Objektmodellierung- Analyse und Entwurf mit der UML 2“, Spektrum Akademischer Verlag, Heidelberg, 2005 (Nachdruck 2011)

[Bpb2013] Berkel, Manuel, Beck, Prof. Dr.-Ing Hans-Peter, Matthes, Dr. Felix Chr. Matthes, Pötter, Bernhard, Schulz, Prof. Dr.-Ing. habil. Detlef, Schulz, Dr. Karen, Springmann, Dr. Jens-Peter, Ziesing, Dr. Hans-Joachim, „Energie und Umwelt“, Heft 319/2013, Bundeszentrale für politische Bildung(bpb), Fürth, 2013

[Balweb2011] Balzert, Heide, „Basiswissen Web-Programmierung : XHTML, CSS, JavaScript, XML, PHP, JSP, ASP.NET, Ajax “, 2. Auflage, W3L-Verlag, Witten, 2011

[s0 norm] „Einrichtung zur Messung der Elektrischen Energie (AC) – Besondere Anforderungen –Teil31: Impulseinrichtungen für Induktionszählern oder elektronische Zähler. (nur Zweidrahtsysteme) (IEC 62053-31:1998); Deutsche Fassung EN 62053-31:1998, Teil 3-31“; VDE Vorschriftenwerk –elektronische Fassung, April 1999

[Mül2009] Lother Müller, Skript „Binaercodes“, Berlin, 2009

Online-Quellen

[EDG] Bundesministerium für Wirtschaft und Energie, „Energiedaten: Gesamtausgabe“, http://www.bmwi.de/BMWi/Redaktion/PDF/E/energiestatistiken-grafiken,property=pdf,bereich=bmwi2012,sprache=de,rwb=true.pdf, stand April 2015, gelesen am: 29.07.2015

[ITW] itwissen.info, „HDMI (high definition multimedia interface)“, http://www.itwissen.info/definition/lexikon/high-definition-multimedia-interface-HDMI-HDMI-Schnittstelle.html gelesen am: 03.05 2015

[Ven2013] Venjakob, Maike, Mersmann, Florian, „ Kosten des Klimawandels” http://www.bpb.de/gesellschaft/umwelt/klimawandel/38487/kosten-

des-klimawandels, Bundeszentrale für Politische Bildung, 23.5.2013,

[ieo] http://linux.die.net, “open(2) – Linux man page”, http://linux.die.net/man/2/open gelesen am: 03.05.2015

[mre] linux.die.net, „read(2) – Linux man page”,http://linux.die.net/man/2/read gelesen am: 03.05.2015

[RpiD] Raspberry Pi Foundation, „Downloads“, https://www.raspberrypi.org/downloads/ gelesen am:01.06 2015

[RFC4180] Y. Shafranovich, Network Working Group (The Internet Society), “Common Format and MIME Type for Comma-Separated Values (CSV) Files”, https://tools.ietf.org/html/rfc4180 Erstellt am October 2005, gelesen am 28.04.2015

[Rpi] Raspberry Pi Foundation, „Raspberry Pi – Revision 2.0“, https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/Raspberry-Pi-Rev-2.0-Model-AB-Schematics.pdf , gelesen am 28.04.2015

[RpiB] Patrick Schnabel, „Raspberry Pi Model B“;http://www.elektronik-kompendium.de/sites/raspberry-pi/bilder/19052511.jpg Abgerufen am: 01.06.2015

[Rpi pins] Raspberry Pi Foundation, „gpio-pins.jpg“, https://github.com/raspberrypi/documentation/blob/master/usage/gpio/images/gpio-pins.jpg; Gelesen am: 28.04.2015

[Rpi num] Raspberry Pi Foundation, “a-and-b-physical-pin-numbers.png”, https://raw.githubusercontent.com/raspberrypi/documentation/master/usage/gpio/images/a-and-b-physical-pin-numbers.png Gelesen am: 28.04..2015

[SD0s] Busware, „SD0 V1.1 schematics”, busware.de/tiki-download_file.php?fileId=69 Gelesen am: 12.03.2015

[tpj] tutorialspoint.com, http://www.tutorialspoint.com/json/json_overview.htm; gelesen am 11.06.2015

[s0vz] volkszaehler.org,“ RaspberryPi-Erweiterung“, http://wiki.volkszaehler.org/hardware/controllers/raspberry_pi_erweiterung?s[]=s0vz gelesen am 11.06.2015

[Ale2014] Alessandri, Santiago, „Efficiency Comparison of C++ JSON Libraries“, https://blog.thousandeyes.com/efficiency-comparison-c-json-libraries/ gelesen am 10.06.2015

[gfi] Golem.de, „Firmware“, http://www.golem.de/specials/firmware/, gelesen am 11.07.2015

[S0S] Mikrocontroller (bzw. Andreas Schwarz im Imperisum), „S0-Schnittstelle“, http://www.mikrocontroller.net/articles/S0-Schnittstelle; gelesen am:22.06.2015

[Gei] Geisler, Andreas, „1-Wire Bussystem Grundlagen, Tipps und Hintergrundinformationen“,http://home.arcor.de/RoBue/1-Wire/Download/1-Wire%20Bussystem_Grundlagen_Tipps.pdf, gelesen am 28.06.15

[ecma2013] Ecma International, „Einführung in JSON – ECMA-404 The JSON Data Interchange Standard.“,://json.org/json-de.html Gelesen am: 03.08.2015

[msx] msxfaq, „1-Wire Bus”, http://www.msxfaq.de/verschiedenes/bastelbude/1wirebus.htm, gelesen am: 28.06.2015

[EPB1] Gerstl, Sebastian, „15 Betriebssysteme für den Raspberry Pi” http://www.elektronikpraxis.vogel.de/embedded/articles/488934/ Geleesn am: 29.06.2015

[AN2141] Maxim Intergrated, „Determining Clock Accuracy Requirements for UART Communications“, http://pdfserv.maximintegrated.com/en/an/AN2141.pdf Gelesen am:30.06.2015

[MUA] Mikrocontroller (bzw. Andreas Schwarz im Imperisum), „S0-Schnittstelle“,, „UART“, http://www.mikrocontroller.net/articles/UART Gelesen am: 30.06.2015

[Pla uart] Plate, Prof. Jürgen, „Raspberry Pi: Serielle Schnittstelle“, http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_Serial.html, gelesen am: 30.06.2015

[rni] rn-wissen, „I²C“, http://rn-wissen.de/wiki/index.php/I²C gelesen am: 30.06.2015

[rni dat] rn-wissen, „I2C_Datenuebertragung.png“,http://rn-wissen.de/wiki/images/f/ff/I2C_Datenuebertragung.png 30.06.2015

[DS1338] Maxim integrated, „DS1338 IC RTC with 56-Byte NV RAM – Maxim”,http://datasheets.maximintegrated.com/en/ds/DS1338-DS1338Z.pdf Gelesen am: 30.06.2015

[bsd0] tostmann, „SD0“ http://busware.de/tiki-index.php?page=SD0 gelesen am: 30.06.2015

[pla rtc] Plate, Prof. Jürgen, Goll, Jonas, Müller, Tobias, Wang, Xiao „Raspberry Pi: Real Time Clock“ http://www.netzmafia.de/skripten/hardware/RasPi/Projekt-RTC/index.html gelesen am: 30.06.2015

[pla i2c] Plate, Prof. Jürgen, „Raspberry Pi: I2C-Konfiguration und – Programmierung“ http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_I2C.html gelesen am: 30.06.2015

[hbt] HobbyTronics Ltd, „Atmega1284P Datasheet“, http://www.hobbytronics.co.uk/raspberry-pi-rtc-module Gelesen am: 30.06.2015

[loc] TechDivision, „Localhost“, https://www.techdivision.com/glossar/l/localhost.html, abgerufen am:06.07.2015

[vza] volkszaehler.org, „About“, http://volkszaehler.org/about gelesen am: 06.07.215

[LAN9512] SMSC, “LAN9512 Data Sheet“, http://ww1.microchip.com/downloads/en/DeviceDoc/9512.pdf Gelese am: 16.06.2014

[atm1284] Atmel Corporation, “8-bit Microcontroller with 128K Bytes In-System Programmable Flash- ”
,http://www.atmel.com/Images/doc8059.pdf

[Zoe] Zoerner, Thorsten, „Volkszähler trifft Discovergy (und Smappee)“, https://blog.stromhaltig.de/2014/08/volkszaehler-trifft-discovergy-und-smappee/
gelesen am 06.07.2015
Gelese am: 16.06.2014

[vzg] volkszaehler.org, „Eigenen Volkszaehler Installieren“, http://wiki.volkszaehler.org/howto/getstarted
gelesen am: 06.07.2015

[pae] K-Zeitung Online„Profi-Analyse des Energie-verbrauchs“http://www.k-zeitung.de/profi-analyse-des-energieverbrauchs/150/1085/88125/
gelesen am: 06.07.2015

[s0vc] Wellschmidt, Henrik, „s0vz/s0vz.c“, https://github.com/w3llschmidt/s0vz/blob/master/s0vz.c
Gelesen am: 06.07.2015

[s0ec] Galow, Rene, „ s0enow/bin/src/s0enow.cpp”, https://github.com/slayerrensky/s0enow/blob/master/bin/src/s0enow.cpp
Gelesen am: 06.07.2015

[Mou2011] Mouton, Claire, „Configuration File Parser Library“,http ://arxiv.org/pdf/1103.3021.pdf
gelesen am: 06.06.2015

[Lin2015] Lindner, Mark A, „libconfig – A Library For Processing Structured Configuration Files“, http://www.hyperrealm.com/libconfig/libconfig.pdf
gelesen am: 06.06.2015

[jcd] Jsoncpp, „JsonCpp”, https://github.com/open-source-parsers/jsoncpp
Gelesen am: 06.07.2015

[cURL] haxx.se ,“cURL“, http://curl.haxx.se/
Gelesen am: 21. 08.2015

[lvz] ubuntu Deutschland e.V., Verzeichnisstruktur, https://wiki.ubuntuusers.de/verzeichnisstruktur,
gelesen am: 25.08.2015

[crol] JC Pollman, „Cron“, http://www.selflinux.org/selflinux/html/cron01.html
Gelesen am: 25.08.2015

[s0lmp] volkszaehler.org, wellenform.png, http://wiki.volkszaehler.org/_detail/hardware/channels/meters/wellenform.png?id=misc%3As0_schnittstelle

[rsync]. Ubuntu Deutschland e.V., „Rsync“, https://wiki.ubuntuusers.de/rsync, gelesen am: 25.08.2015

[lwry] Henryk Gerlach, „Rsync“, http://linuxwiki.de/rsync, gelesen am: 25.08.2015

[Wol2006 dpz] Wolf, Jürgen, „Dämonprozesse“, http://openbook.rheinwerk-verlag.de/linux_unix_programmierung/Kap07-011.htm#RxxKap07011040002021F048100
Aus: Wolf, Jürgen, „Linux-UNIX-Programmierung – Das umfassen-

	de Handbuch“, 2. Auflage, Rheinwerk Verlag GmbH, Bonn, 2006 Gelesen am: 10.06.2015
[Wol2006 af]	Wolf, Jürgen, „Rund um die Ausführung von Prozessen“, http://openbook.rheinwerk-verlag.de/linux_unix_programmierung/Kap07-012.htm#RxxKap07012040002031F04A100 Aus: Wolf, Jürgen, „Linux-UNIX-Programmierung – Das umfassende Handbuch“, 2. Auflage, Rheinwerk Verlag GmbH, Bonn, 2006 Gelesen am: 10.06.2015
[mcS0]	Mikrocontroller (bzw. Andreas Schwarz im Imperisum), „S0-Schnittstelle“, http://www.mikrocontroller.net/wikifiles/e/ed/SOint.png Gelesen am 21.06.2015
[fua]	raspberrypi-spy.co.uk, „Free Your Raspberry Pi Serial Port 10“, http://www.raspberrypi-spy.co.uk/2013/12/free-your-raspberry-pi-serial-port/ gelesen am: 06.08.2015
[zum]	Quality First Software GmbH, „Zeilenumbrüche in Unix und Windows“, https://www.qfs.de/qftest/manual/de/tech_linebreaks.html Gelesen am 06.08.2015
[ASCII]	asciitable.com, „ASCII Table and Description“,http://www.asciitable.com/ gelesen am: 30.04.2015
[wRPi]	Wikipedia.org, „Raspberry Pi – Eigenschaften“ https://de.wikipedia.org/wiki/Raspberry_Pi#Eigenschaften
[RpiM]	Raspberry Pi Foundation, “Raspberry Pi Model Specifications”, https://www.raspberrypi.org/documentation/hardware/raspberrypi/models/specs.md
[dpm]	conrad.com, „DPM1L32-D- Technisches Datenblatt / Bedienungsanleitung“, http://www.produktinfo.conrad.com/datenblaetter/125000-149999/125355-an-01-de-VOLTCRAFT_DPM_1L32_D_WECHSELSTROMZAEH.pdf gelesen am:01.06.2015
[s0enow]	Github.de (Benutzer: slayerrensky), „s0enow/+“ https://github.com/slayerrensky/s0enow Gelesen am: 01.04.2015
[vgl. s0enow sync]	Github.de (Benutzer: slayerrensky), „/s0enow/sync/+“ https://github.com/slayerrensky/s0enow/tree/master/sync Gelesen am: 01.04.2015

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Anhang

A. Schaltplan des Raspberry Pi Model B

B. Schaltplan des SD0

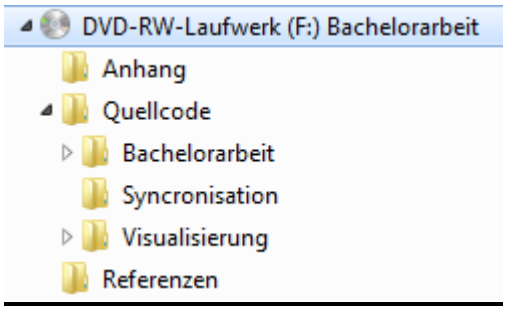
C. Hilfsfunktionen zum zeichnen der Energieverbrauchsdaten

Die Hier dargestellte Code befindet sich auf der Beiliegenden CD, im Ordner Quellcode\Visualisierung\js\, in der Datei *page.js*

```
1   /**
2    * Startet den Zeichenvorgang
3    */
4   function drawChart()
5   {
6
7     if ( document.getElementById('aktuell').checked == true )   //Aktuelle
8   Verbrauchsdaten sollen angezeigt werden.
9     {
10      //Datumseingabe Deaktivieren, da aktuelle Verbrauchsdaten angezeigt werden
11     //sollen.
12      document.getElementById('beginDatum').value = '';
13      document.getElementById('beginDatum').disabled = true;
14      document.getElementById('endDatum').value = '';
15      document.getElementById('endDatum').disabled = true;
16    }
17    else if (document.getElementById('zeitraum').checked == true ) //Verbrauchsdaten,
18  für einen Zeitraum,sollen angezeigt werden.
19    {
20      //Verbrauchsdaten für einen Zeitraum angezeigen.
21      var anfangsZeit =
22  datepicker2Timestamp(document.getElementById('beginDatum').value,'00:00:00');
23      var endZeit =
24  datepicker2Timestamp(document.getElementById('endDatum').value,'23:59:59');
25      InitHighChart(anfangsZeit, endZeit);
26    }
27  }
28  })
```

D. Inhalt und Struktur der Beiliegenden CD

Struktur der CD:



Inhalt der CD: