FRANKFURT UNIVERSITY OF APPLIED SCIENCES

BACHELOR THESIS

Entwicklung einer Reset-Lösung für Raspberry Pi Clusterknoten und Konstruktion einer netzwerkgesteuerten Steckdosenleiste

---

Autor:
Henry-Norbert COCOS
Mat.Nr. 1021897

Referent:
Prof. Dr. Christian BAUN

Korreferent:
Prof. Dr. Thomas GABEL

Eingereicht zum Erlangen
des akademischen Grads Bachelor of Science

am

Fachbereich 2
Informatik

22. März 2018

Eidesstattliche Erklärung

Ich, Henry-Norbert COCOS versichere die Bachelorarbeit mit dem Titel „**Entwicklung einer Reset-Lösung für Raspberry Pi Clusterknoten und Konstruktion einer netzwerkgesteuerten Steckdosenleiste**“selbstständig und lediglich unter Benutzung der angegebenen Quellen und Hilfsmittel verfasst zu haben.

Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Unterschrift:

Datum:

Zusammenfassung

Bachelor of Science

Entwicklung einer Reset-Lösung für Raspberry Pi Clusterknoten und Konstruktion einer netzwerkgesteuerten Steckdosenleiste

von Henry-Norbert COCOS

Diese Arbeit stellt eine Reset-Lösung für Raspberry Pi Cluster-Knoten vor. Diese Lösung besteht aus einer Schaltung, die mithilfe eines Optokopplers den Reset am Raspberry Pi auslöst. Angeschlossen wird diese Schaltung an die GPIO-Ports des Raspberry Pi. Die Überwachung der Knoten realisiert das Linux Kommando `ping`. Weiter wird in dieser Arbeit eine netzwerkgesteuerte Steckdosenleiste entwickelt und konstruiert. Eine detaillierte Anleitung beschreibt den Aufbau und die Steuerung der Steckdosenleiste mit einer Weboberfläche und einem REST-Webservice. Es wird gezeigt, wie ein Webserver für die Weboberfläche und den Webservice mithilfe des Python Frameworks Flask implementiert wird.

Abstract

Bachelor of Science

Entwicklung einer Reset-Lösung für Raspberry Pi Clusterknoten und Konstruktion einer netzwerkgesteuerten Steckdosenleiste

by Henry-Norbert COCOS

This work covers the construction of a reset solution for the Raspberry Pi. The solution consists of a circuit, which uses an optocoupler to realize the reset. The circuit is attached to the Raspberry Pis GPIO-Ports. The Linux command `ping` is used to monitor and control the cluster nodes. This work also covers the construction of a network controlled power strip. Therefore a detailed manual with the nessacary steps to build such a power strip is described. The power strip is controlled via a website and a REST-Webservice. Therefore it is described how to program a webserver, which hosts a website and a webservice with the Python Framework Flask.

Danksagung

An dieser Stelle möchte ich all jenen danken, die durch ihre fachliche und persönliche Unterstützung zum Gelingen dieser Bachelor Thesis beigetragen haben.

Danken will ich meiner Familie für ihre Unterstützung während meines Studiums. Besonders will ich meinem Vater Marian-Theodor Cocos und meinem Bruder Samuel-Robert Cocos danken. Sie haben mir wichtige Hinweise auf dem Gebiet der Elektrotechnik und Elektronik gegeben.

Nach den Arbeiten beim Aufbau des Clusters **BRAIN 128** in den Semesterferien des Sommersemesters 2017, wurde mir dieses Thema von Herrn Prof. Dr. Baun vorgeschlagen. Ich möchte Herrn Prof. Dr. Baun danken, dass er mir die Möglichkeit gab meine Kenntnisse der Elektronik in diesem spannenden Projekt weiter zu vertiefen. Außerdem möchte ich Herrn Prof. Dr. Gabel danken, dass er meine Abschlussarbeit als Korreferent begleitet hat.

Weiterhin möchte ich der Frankfurt University of Applied Sciences, speziell dem Fachbereich 2 danken, welche mir das Labor 1-149 zur Verfügung gestellt hat, um meine Versuchsaufbauten zu testen und diese Arbeit zu verfassen.

Mein Dank gilt auch Rosa-Maria Spanou für die gute Zusammenarbeit und unentwegte Unterstützung während des Tutoriums und dem Aufbau des Clusters **BRAIN 128**.

Danken will ich Herrn Huth und Herrn Rüth für die hilfreichen Anmerkungen und das Korrekturlesen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

ARM - Advanced RISC Machine
BIOS - Basic Input Output System
DIN - Deutsches Institut für Normung
DIP - Dual in-line package
GPIO - General Purpose Input/Output
HAC - High Availability Cluster
HPC - High Performance Computing
HTML - Hypertext Markup Language
HTTP - Hypertext Transfer Protocol
IP - Internet Protocol
JSON - Java Script Object Notation
LAN - Local Area Network
LED - Light Emitting Diode
MBit/s - Megabit pro Sekunde
NAS - Network Attached Storage
pip - pip installs packages
RAM - Random-Access Memory
REST - Representational State Transfer
RISC - Reduced Instruction Set Computer
RSA - Rivest, Shamir und Adleman
SD-Card - Secure Digital Memory Card
SOAP - Simple Object Access Protocol
SSH - Secure Shell
URI - Unique Resource Identifier
URL - Unique Resource Locator
USB - Universal Serial Bus
VDE - Verband der Elektrotechnik, Elektronik und Informationstechnik
WLAN - Wireless Local Area Network
XML - Extensible Markup Language

Symbolverzeichnis

C	Kapazität	F
I	Strom	A
U	Spannung	V
R	Widerstand	Ω
P	Leistung	W
f	Frequenz	Hz

Einleitung

Der Fachbereich 2 der Frankfurt University of Applied Sciences betreibt einen Cluster aus 128 Raspberry Pi 3 Einplatinencomputern. Dieser Cluster wird für Forschungszwecke und studentische Projekte verwendet. Abbildung 1.1 zeigt diesen Cluster.

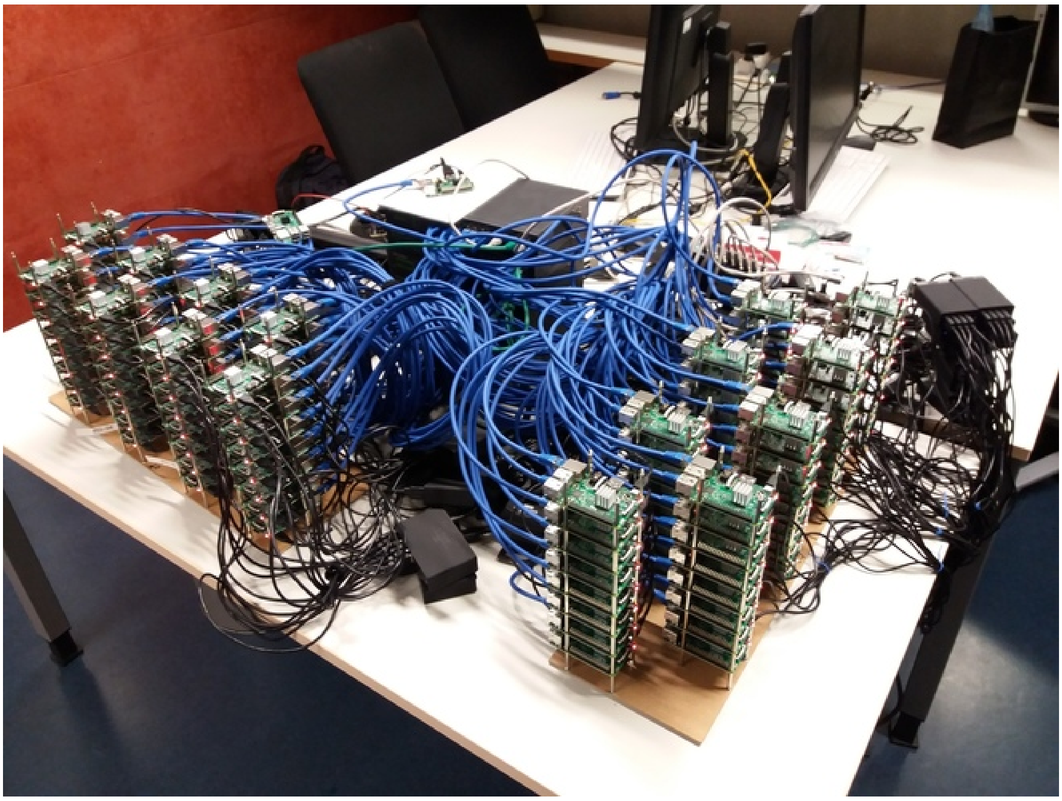


ABBILDUNG 1.1: Cluster des Fachbereich 2

Cluster sind vernetzte eigenständige Rechner, die gemeinsam einen Rechnerverbund darstellen. Sie können für unterschiedliche Einsatzzwecke verwendet werden. Mit solchen Cluster-Systemen lassen sich hochverfügbare Cluster, sogenannte **HA**-Cluster (High Availability Cluster), realisieren. Sie zeichnen sich durch eine hohe Ausfallsicherheit und Verfügbarkeit aus. Weiter lassen sich auch **HPC**-Cluster (High Performance Computing) realisieren. HPC-Cluster sind durch ihre hohe Rechenleistung ideal für die Abarbeitung von großen Rechenaufgaben geeignet.

Mit ihnen ist es möglich Rechenaufgaben parallel über mehrere unabhängige Rechensysteme, sogenannte **Knoten** oder engl. „**Nodes** “, zu verteilen.[1]

Cluster aus Einplatinencomputern sind geeignet um Projekte auf dem Gebiet des parallelen Rechnens zu testen. Sie eignen sich durch ihre niedrigen Anschaffungskosten und den geringen Stromverbrauch für den Einsatz im universitären Umfeld [1]. Durch die Verwendung des Raspberry Pi 3 lässt sich ein großes, verteiltes System günstig aufbauen. Der in Abbildung 1.1 gezeigte Rechnerverbund dient als Testumgebung für viele unterschiedliche Projekte. Unter anderem wurden Laborübungen für Studenten des Master-Studiengangs entworfen. Diese Übungen geben den Studenten die Möglichkeit praktische Aufgaben mit einem realen, verteilten System zu absolvieren. Während des Betriebs des Clusters kam es jedoch zu Problemen. Ein immer wiederkehrendes Problem war der Ausfall von Knoten, während der Berechnung von Aufgaben. Standardmäßig bietet der Raspberry Pi 3 keine Möglichkeit zum Hardware-Reset. Es stellte sich als große Herausforderung dar diesen Cluster zu administrieren, da es nicht möglich war einzelne Knoten des Clusters neu zu starten.

Aus diesem Grund sollte nach einer Lösung gesucht werden, die es einem Administrator ermöglicht, einzelne Knoten per Fernzugriff neu zu starten. Diese Arbeit beschäftigt sich mit diesem Problem und zeigt unterschiedliche Möglichkeiten zur Realisierung einer Lösung auf.

Das Kapitel 3 zeigt wie man einen Cluster aus Einplatinencomputern aufbaut. Dieser Cluster dient als Testumgebung und stellt den Cluster **BRAIN 128** nach. Der Aufbau dieser Testumgebung sorgt dafür, dass der Betrieb des Clusters im Labor störungsfrei abläuft und bietet einen kleinen repräsentativen Aufbau zur Demonstration des Resets.

Kapitel 4 erklärt die Implementation der Reset Funktionalität mithilfe von Shell-Skripten und die Konstruktion einer Schaltung, welche den Hardware-Reset ermöglicht. Es werden die nötigen Grundlagen vermittelt und eine detaillierte Beschreibung der Lösung dargestellt.

Weiter zeigt Kapitel 5 die Konstruktion einer netzwerkgesteuerten Steckdosenleiste. Durch diese steuerbare Steckdosenleiste lässt sich der Cluster per Fernzugriff An- und Ausschalten. Es wird aufgezeigt, wie man einen Webserver mit **Python** erstellt. Dieser Server stellt eine Webseite als auch einen Webservice bereit. Eine detaillierte Beschreibung des Aufbaus und der Implementation dieser Steckdosenleiste wird in Kapitel 5 aufgezeigt.

Abschließend bietet diese Arbeit ein Fazit und einen Ausblick. Dieses Kapitel fasst die wichtigsten Erkenntnisse der beiden vorangegangenen Kapitel nochmals zusammen. Zukünftige Weiterentwicklungen werden aufgezeigt und diskutiert.

[1] `https://de.wikipedia.org/wiki/Rechnerverbund` (abgerufen 02.03.2018)

Grundlagen

Dieses Kapitel behandelt die Grundlagen zur verwendeten Hardware und Software. Der aktuelle Stand der Technik wird ermittelt. Eine Vorstellung des **Raspberry Pi** und seiner **GPIO-Ports** erklärt deren Merkmale. Das Betriebssystem **Raspbian** und die Bibliotheken **RPi.GPIO** und **Flask** werden beschrieben. Außerdem wird eine Gegenüberstellung zwischen einer erhältlichen Steckdosenleiste und der in dieser Arbeit konstruierten Lösung präsentiert.

2.1 Stand der Technik

Der Einplatinencomputer Raspberry Pi besitzt standardmäßig keine Möglichkeit hardwaremäßig neu gestartet zu werden. Anders als handelsübliche Rechnersysteme bietet der Raspberry Pi keinen Schalter zum Ab- oder Anschalten. Eine mögliche Erklärung dafür ist wohl das der Raspberry Pi kompakt und einfach zu verwenden sein soll. Aufgrund seines niedrigen Preises ist der Raspberry Pi beispielsweise nicht mit einem BIOS bestückt [2]. Jedoch wurde der Raspberry Pi mit Pins bestückt die zum Reset des Prozessors dienen.

Auf der Platine sind zwei Anschlusspins aufgebracht, die für einen Reset geplant wurden. Ein Kurzschluss an diesen Pins startet den Prozessor neu. Dadurch lässt sich ein Schaltermechanismus nachrüsten [3]. beschreibt diesen Port namens **P6** und zeigt, wie man einen Taster am Raspberry Pi montiert. Es wird auch darauf eingegangen, dass das Betriebssystem aus einem heruntergefahrenen Zustand wieder hochfährt. Diese Tatsache zeigt, dass es sich bei dieser Art von Reset um einen **Hard-Reset** handelt. Das bedeutet, dass anders als beim **Soft-Reset**, die Hardware des Pi neu gestartet wird und keine Software Routinen den Neustart auslösen.

In [4] werden nähere Informationen zur Hardware bereitgestellt. Dort werden Pläne des Raspberry Pi bereitgestellt, die den P6 Port und seinen Aufbau detailliert beschreiben. In diesem Plan ist ersichtlich, dass diese Anschlusspins direkt mit dem Prozessor verbunden sind. Ein extern herbeigeführter Kurzschluss zieht die Versorgungsspannung des Prozessors ab. Nachdem dieser Kurzschluss aufgehoben wird, kann der Prozessor wieder mit Spannung versorgt werden und neu starten. Der vollständige Plan kann im Anhang A.1 eingesehen werden.

Die aufgeführten Informationen zeigen, dass ein Hard-Reset beim Raspberry Pi nachgerüstet werden kann. Ein Taster wie er in [3] beschrieben ist, ist für den Einsatzzweck im Umfeld eines Clusters nicht sonderlich hilfreich. Durch einen Taster kann man einen Knoten zwar neu starten, jedoch muss hierfür eine Person in unmittelbarer Nähe zum Cluster sein. Dadurch wird die Administration des Clusters nicht erleichtert, da der gleiche Effekt durch das Abziehen des USB-Stromversorgungskabels erreicht werden kann.

Aus den Überlegungen zum Reset der Cluster-Knoten entstand die Idee eine Möglichkeit zum Starten und Herunterfahren des Clusters zu finden. Der Cluster befindet sich im Dauerbetrieb und kann nur manuell gestartet werden. Eine Lösung um den Cluster anzuschalten lag daher nahe. Die einfachste Lösung dieses Problems sind schaltbare Steckdosen. Unter diesen Steckdosen existieren auch Versionen, die sich über das Netzwerk schalten lassen. Diese Steckdosen sind unter dem Namen „IP-Steckdose“ oder „intelligente Steckdose“ auf dem Markt erhältlich. Diese Steckdosen haben eine Anbindung ans Netzwerk und betreiben einen Webserver mit Weboberfläche, über den sich die Steckdose schalten lässt. Die Preise für solche Steckdosen reichen von 50 bis 110 € [5]. Diese Produkte lassen sich in ein bestehendes WLAN-Netzwerk integrieren und steuern.

Es existieren auch Lösungen mit mehreren Steckdosen. Solche schaltbaren Steckdosenleisten bieten die Möglichkeit mehrere Geräte an eine Leiste anzuschließen und über eine Weboberfläche anzusteuern. Auch hier wird intern ein Webserver betrieben. Die Preise für solche Steckdosenleisten reichen von 180 bis 270 € [6]. Der Nachteil der in [5] und [6] gezeigten Produkte, liegt in der festen Installation der implementierten Software. In den dort verfügbaren Produkten ist die Webserversoftware nicht austausch- oder erweiterbar. Die Bedienoberfläche dieser Produkte ist meist nicht anpassbar.

Daher sind alternative Systeme zu evaluieren. Durch den Einsatz eines Raspberry Pi lässt sich ein vollwertiges Linux Betriebssystem verwenden. Dadurch können Webserversoftware und Schnittstelle gewählt werden, da der Raspberry Pi über eine Ethernet- und WLAN-Schnittstelle verfügt [7]. zeigt eine detaillierte Bauanleitung für eine netzwerkgesteuerte Steckdosenleiste. Dort wird das Einsetzen und Verkabeln der Steckdosen erklärt. Die Steuerung übernimmt ein Raspberry Pi. Zum Schalten der Steckdosen wird eine Transistor Schaltung aufgebaut und diese mit dem Raspberry Pi über seine GPIO-Ports angesteuert. Diese Steckdosenleiste wird anschließend in das System OpenHAB eingebunden. OpenHAB ist eine Open Source Lösung für die Gebaudeautomation [8]. Die in [7] vorgestellte Lösung bietet eine detaillierte Beschreibung der Arbeitsschritte zum Aufbau einer Steckdosenleiste mit dem Raspberry Pi. Der Aufbau ist jedoch durch die verwendete Holzkonstruktion sehr groß und nicht besonders mobil. Außerdem geht diese Anleitung nicht auf die Implementation der verwendeten Software ein.

2.2 Gegenüberstellung mit erhältlichen Steckdosenleisten

Um einen Vergleich der in dieser Arbeit konstruierten Steckdosenleiste zu erhalten, wird diese Lösung mit einem auf dem Markt erhältlichen Produkt verglichen. Dabei wurde eine Steckdosenleiste herangezogen, die beim Internethändler Reichelt erhältlich ist. Im Funktionsumfang ist dieses Produkt mit der hier vorgestellten Lösung vergleichbar. Abbildung 2.1 (b) zeigt die hier verglichene Steckdosenleiste. In Abbildung 2.1 (a) sieht man die selbst konstruierte Lösung.

(a) selbstgebaute Steckdosenleiste

(b) erhältliche Steckdosenleiste[1]

ABBILDUNG 2.1: Netzwerksteckdosenleiste

Die zum Vergleich herangezogene Steckdosenleiste **ALLNET ALL4176** hat einen Ethernet-Anschluss und eine WLAN-Schnittstelle. Durch diese Schnittstellen lässt sich dieses Produkt über WLAN oder LAN an das Netzwerk anbinden. Die in dieser Arbeit konstruierte Lösung bietet zusätzlich eine Bluetooth-Schnittstelle. Außerdem bietet sie neun Steckdosen, drei mehr als die verglichene Steckdosenleiste.

TABELLE 2.1: Vergleich der Steckdosenleisten

	selbstgebaute Steckdosenleiste	**ALLNET ALL4176**
Schnittstellen	LAN,WLAN,Bluetooth,USB	LAN,WLAN,USB
Steuerung	Taster,Webseite,Webservice	Taster,Webseite
Erweiterbarkeit	Software erweiterbar	geschlossenes System
Anzahl Steckdosen	9	6
Schaltstrom	10 A	10 A
Preis	160 €	270 €

[1]Quelle: https://www.reichelt.de/Messen-Steuern-Regeln/ALLNET-ALL4176/3/index.html?ACTION=3&LA=446&ARTICLE=209651&GROUPID=5828&artnr=ALLNET+ALL4176&SEARCH=steckdosenleiste%2Bnetzwerk&trstct=pos_4 (aufgerufen 02.03.2018)

Der Vorteil der Steckdosenleiste ALLNET ALL4176 ist, dass jede einzelne Steckdose über einen Taster oder über den integrierten Webserver angesteuert werden kann. Bei der selbst gebauten Lösung lässt sich die Ansteuerung der Steckdosen über den Webserver umprogrammieren, jedoch bietet der eingebaute Taster nur die Möglichkeit alle Steckdosen an- und auszuschalten. Die selbst erstellte Lösung bietet aber im Gegensatz zur Netzwerksteckdose ALLNET ALL4176 einen Webservice und ein Linux Betriebssystem. Durch den Einsatz eines Linux Betriebssystems kann das System erweitert werden. Der Preis, der in dieser Arbeit konstruierten Steckdosenleiste, liegt bei 160 €, wohingegen ALLNET ALL4176 einen Preis von 270 € hat. Somit ist der Preis der hier erarbeiteten Lösung sogar geringer als der der erhältlichen Lösung. Die Zusammensetzung des Preises, der selbst gebauten Steckdosenleiste, kann Tabelle 5.1 in Kapitel 5 entnommen werden.

Die hier aufgeführten Punkte zeigen, dass eine selbst gebaute Steckdosenleiste zu erhältlichen Produkten vergleichbar ist. Die Erweiterungsmöglichkeiten der hier vorgestellten Steckdosenleiste und ihr Preis machen sie zu einer ernsthaften Alternative zu erhältlichen Produkten. Tabelle 2.1 zeigt die verglichenen Merkmale der beiden Steckdosenleisten.

2.3 Raspberry Pi 3

Der Raspberry Pi ist ein Einplatinencomputer der Raspberry Pi Foundation. Das erste Modell kam im Jahr 2012 auf den Markt. Seitdem ist er mittlerweile in der dritten Revision erhältlich. Dieses System wurde mit dem Ziel entwickelt jungen Menschen die Möglichkeit zu geben Programmiererfahrungen zu sammeln. Außerdem kann der Raspberry Pi auch für Experimente auf Hardwareebene eingesetzt werden [9].

Der Raspberry Pi hat den Formfaktor einer Kreditkarte und eignet sich durch seine kompakte Bauweise für unterschiedlichste Projekte. Durch den Raspberry Pi können Projekte auf dem Gebiet des **IoT** (Internet of Things) erstellt werden. Dieses Gebiet strebt die Vernetzung von immer mehr alltäglichen Gegenständen an. Ein weiteres beliebtes Einsatzgebiet ist der Bereich des **Physical Computing**. Dabei wird der Raspberry Pi mithilfe von Sensoren dazu verwendet mit seiner physischen Umgebung zu interagieren. Dabei werden Softwarekomponenten mit Sensoren und Motoren vereint und können im Verbund mit ihrer Umgebung interagieren [10].

Der Raspberry Pi zeichnet sich durch einen günstigen Preis aus. Sein Anschaffungspreis von 35 € machen ihn zu einem günstigen Einsteigergerät. Abbildung 2.2 zeigt den Raspberry Pi in der Version 3.

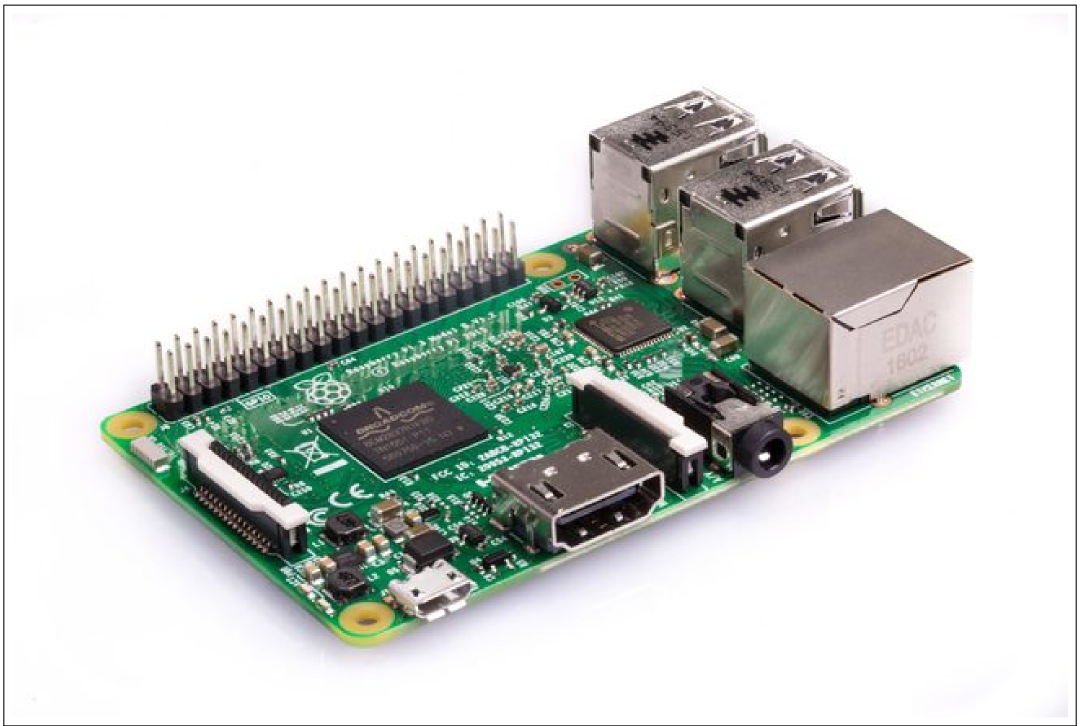


ABBILDUNG 2.2: Raspberry Pi 3
Quelle: `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/` (abgerufen 02.03.2018)

Der Raspberry Pi 3 bietet folgende Hardware Spezifikationen:

- Quad Core ARM Prozessor 1.2 GHz Broadcom BCM2837
- 1 GB RAM
- Micro-SD Karten Slot für Bootmedium
- 100 MBit/s Ethernet Port
- WLAN und Bluetooth Schnittstelle
- 4 USB 2.0 Anschlüsse
- 40 GPIO Pins

2.4 GPIO-Ports

GPIO steht für General Purpose Input/Output. Diese Pins sind Anschlüsse, die frei programmierbar sind und zum Aussenden und Empfangen von Signalen genutzt werden können. Außerdem können die GPIO-Ports als Stromversorgung für kleine Geräte genutzt werden. Abbildung 2.3 zeigt das Schema der Pinbelegung der GPIO-Ports.

Das Bild zeigt, dass manche Pins mit einer Spannung von 5 Volt und manche Pins mit einer Spannung von 3,3 Volt betrieben werden. Die Pins des Raspberry Pi können mit Jumperkabeln auf deutsch Drahtbrücken verbunden werden. Dadurch lassen sich kleine Schaltungen mit Leuchtdioden oder auch kleine Elektromotoren betreiben und steuern [11]. Diese lassen sich mithilfe von verschiedenen Bibliotheken ansprechen. So können die GPIO-Ports mit der Programmiersprache C, Python und sogar mit Shell Skripten angesteuert werden.

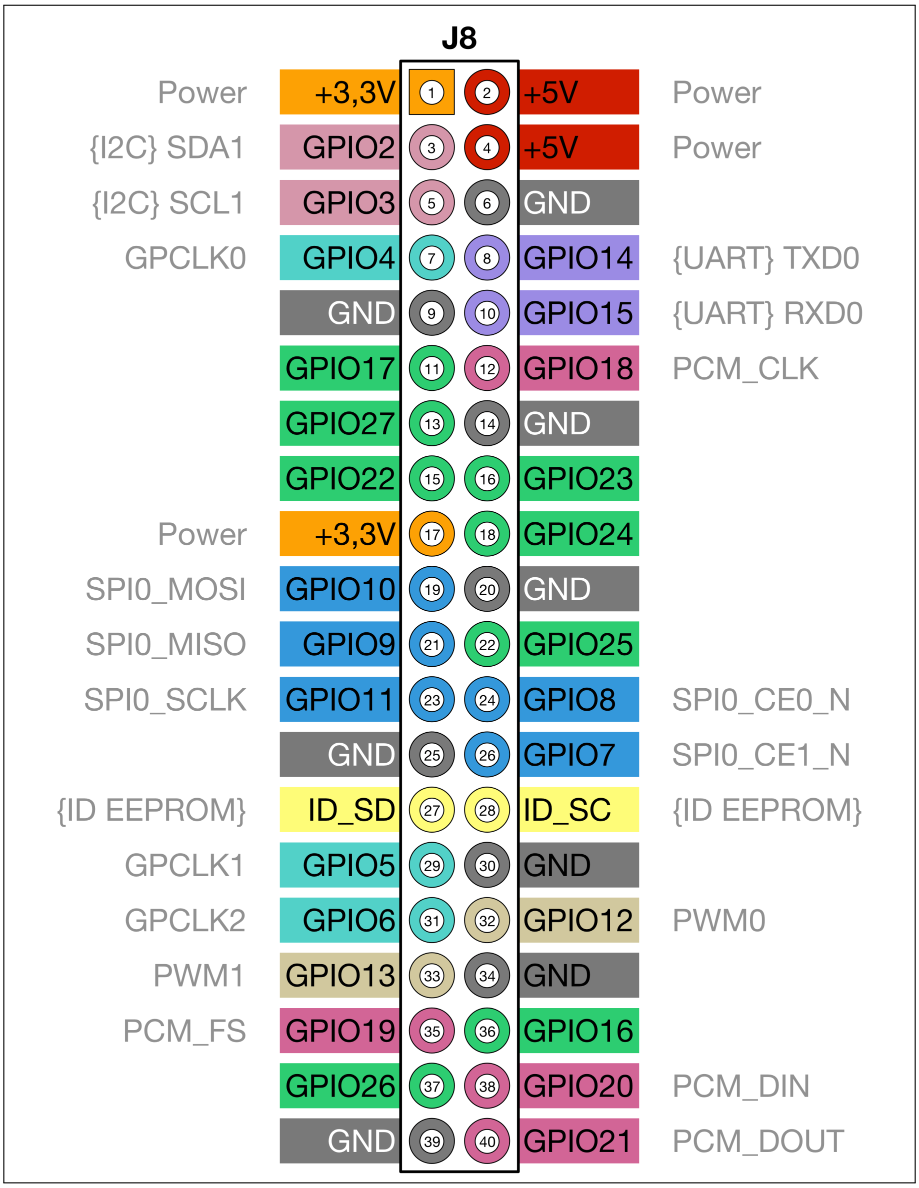


ABBILDUNG 2.3: GPIO Pinbelegung des Raspberry Pi 3
Quelle: `https://indibit.de/raspberry-pi-die-gpio-schnittstelle-grundlagenbelegung/` (abgerufen 02.03.2018)

Die GPIO-Ports sind durch ihre Mehrfachbelegung in der Lage unterschiedliche Aufgaben zu erledigen. So bieten beispielsweise die GPIO-Ports 2 und 3 eine I^2C-Schnittstelle.

Diese Schnittstelle eignet sich zur seriellen Kommunikation zwischen Geräten und wurde in den 80er Jahren des letzten Jahrhunderts von Philips für Unterhaltungselektronik entwickelt [12].

Mit diesen Ports lassen sich Geräte für die serielle Kommunikation auf kurzen Strecken adressieren und ansprechen. Diese und andere Kommunikationsschnittstellen sorgen für einen großen Einsatzbereich des Raspberry Pi in unterschiedlichen Projekten.

2.5 Raspbian

Raspbian ist das verbreitetste Betriebssystem für den Raspberry Pi. Es ist ein Fork des Betriebssystems Debian und für den Raspberry Pi angepasst worden. Es steht unter einer Open Source Lizenz und kann kostenlos verwendet werden. Für das Betriebssystem Raspbian existieren über 35.000 Softwarepakete, die zur freien Verfügung stehen [13]. Zusätzlich bietet es die Möglichkeit die GPIO-Ports des Raspberry Pi mit Shell Skripten anzusteuern.

2.6 RPi.GPIO

RPi.GPIO ist eine Bibliothek zum Ansteuern der GPIO-Ports des Raspberry Pi. Es ist in Python geschrieben und kann in jedes Python Skript eingebunden werden. Diese Bibliothek bietet Standardfunktionen zum Ansprechen der GPIO-Ports und erleichtert die Implementation von Hardware Steuerungen [14]. Es existieren weitere Bibliotheken für die Programmiersprache C, wie zum Beispiel WiringPi.

WiringPi ist eine Erweiterung für den Raspberry Pi, die den Einsatz von C und C++ ermöglicht. Dabei richtet sich diese Bibliothek an erfahrene C/C++ Programmierer und steht unter einer freien Open Source Lizenz [15].

2.7 Flask

Flask ist ein Python Framework zum Erstellen von Webanwendungen mit Python. Dabei bietet Flask einen leichtgewichtigen Webserver mit Debugging Funktion. Es lassen sich Webseiten mit dynamischen Inhalten erstellen. Dabei können die dynamischen Inhalte auf den HTML Webseiten in Python implementiert werden. Außerdem lassen sich REST-ful Webservices mithilfe dieser Bibliothek erstellen [16].

2.8 REST-ful Webservices

REST-ful-Webservices beschreiben einen Architekturstil für Webbasierte Anwendungen. Dieser Architekturstil baut auf vier Methoden von HTTP (Hypertext Transfer Protocol) auf. Dabei werden alle Operationen auf Ressourcen über die HTTP-Methoden `GET`,`POST`,`UPDATE` und `DELETE` abgebildet. Die Informationen über den angebotenen Service werden in der URI (Uniform Resource Identifier) gespeichert. Durch dieses Prinzip lassen sich einfache und leichtgewichtige Webanwendungen erstellen [17].

Eine weitere Möglichkeit zum Erstellen eines Webservice bietet **SOAP** (Simple Object Access Protocol). Dieses Protokoll basiert auf XML-Dateien und definiert Schnittstellen zum Ansprechen des Webservice. Anders als SOAP benötigt ein REST-Webservice keine XML-Datei zur Definition der angebotenen Ressourcen [18]. Ein REST-Webservice ist einfacher zu implementieren und bietet sowohl server- als auch clientseitig eine einfachere Darstellung des angebotenen Webservice.

Aufbau einer Testumgebung

Dieses Kapitel beschreibt den Aufbau eines Test-Clusters. Dieser Cluster dient als Testumgebung für die Funktion der Reset-Lösung aus Kapitel 4. Zuerst wird die Topologie des Clusters gezeigt und erklärt. Anschließend wird gezeigt, wie man einen Cluster installiert und konfiguriert.

3.1 Topologie des Clusters

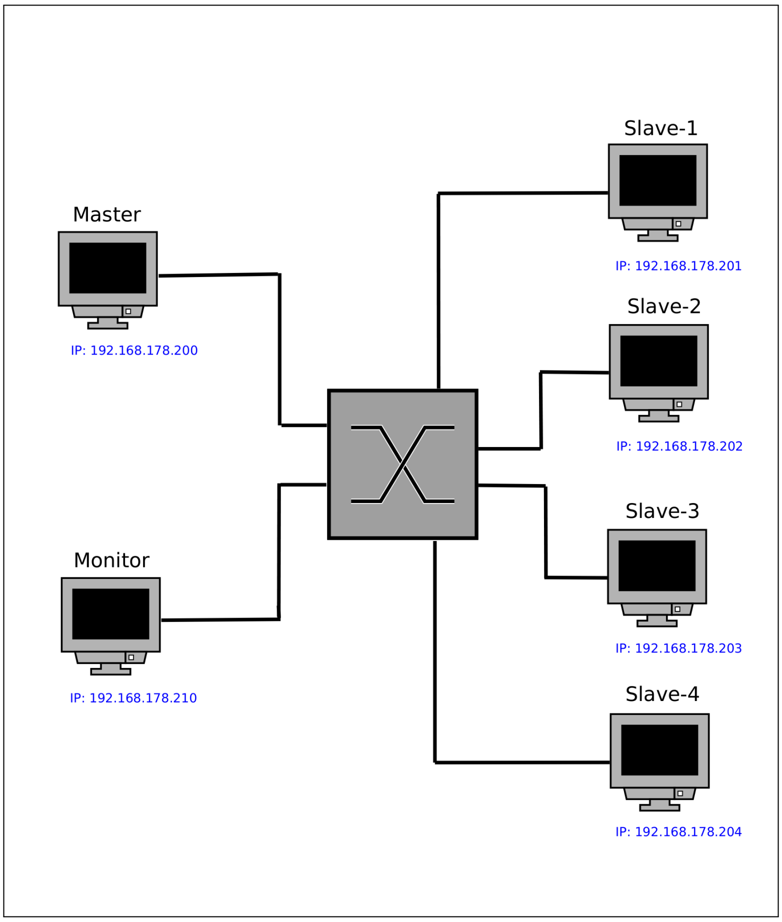


ABBILDUNG 3.1: Netzwerktopologie des Test-Clusters

Abbildung 3.1 zeigt die Netzwerktopologie der Testumgebung. Sie dient im weiteren Verlauf der Arbeit als Referenzumgebung und ersetzt den Cluster des Fachbereich 2.

Da sich der Cluster zum Zeitpunkt der Erstellung dieser Arbeit in Benutzung befand, wurde dieser Cluster aufgebaut. In seiner Konfiguration entspricht er dem Cluster im Raum 1-149. Dabei wurden jedoch die auf dem Cluster **BRAIN 128** installierten Softwarepakete ausgelassen, weil sie für diese Testzwecke nicht benötigt werden.

Die Hostnamen und IP-Adressen der Knoten des Clusters sind die folgenden:

- **Master** - IP: 192.168.178.200
- **Monitor** - IP: 192.168.178.210
- **Slave-1 bis 4** - IP: 192.168.178.201 bis 192.168.178.204

Der Rechner mit dem Hostnamen **Master** dient als Administrator des gesamten Clusters. Die Knoten **Slave-1 bis 4** werden über den Host Master gesteuert und administriert. Der Raspberry Pi mit dem Hostnamen **Monitor** dient der Überwachung des Clusters und soll im späteren Verlauf dafür sorgen, dass jeder Knoten des Clusters neu gestartet wird, sollte er ausfallen. Alle Raspberry Pis des Clusters sind über einen Switch miteinander verbunden. Eine Verbindung zum Internet ist zwar möglich, jedoch für Testzwecke nicht erforderlich. Im nächsten Abschnitt wird erläutert, wie dieser Cluster eingerichtet wird. Die Installation und Konfiguration des Clusters wird mit einem Linux System ausgeführt.

3.2 Installation und Konfiguration

Auf dem Cluster wird das Betriebssystem Rasbian installiert. Dazu bespielt man die SD-Karte des Raspberry Pi mit dem Betriebssystemabbild. Hierzu steckt man die SD-Karte in den SD-Karten Slot des Rechners mit dem die Installation durchgeführt werden soll und hängt die Partition aus. Man erreicht das mit dem Kommando `umount /dev/mmcblk0p1`.

Als nächstes wird das Image mit dem Kommando `dd` auf die SD-Karte kopiert. `dd` ist ein Linux Programm, welches das Kopieren von Images erlaubt. Das Kommando zum Kopieren eines Images mit dem Namen `RASPBIAN.img` würde folgendermaßen lauten:

Prompt:`# dd bs=4M if=/home/henry/RASPBIAN.img of=/dev/mmcblk0`

Dieses Kommando muss mit Root-Rechten ausgeführt werden. Der Parameter `bs` bezeichnet die Blockgröße. Das obige Beispiel definiert eine Blockgröße von 4 Megabyte. Somit werden vom Speichermedium jeweils 4 Megabyte große Blöcke geladen und auf das Zielmedium kopiert.

Der Parameter `if` bezeichnet die Eingabedatei und `of` steht für die Ausgabedatei. Nach Abschluss des Kopierens sollte das Kommando `sync` abgesetzt werden. Dieses Kommando schreibt die im Hauptspeicher gepufferten Daten in die Ausgabedatei.

Die erste Partition der SD-Karte hängt man mit dem Kommando `mount /dev/mmcblk0p1 /media/test` aus. Das Verzeichnis `/media/test` wird gewählt, um einen Ort zum Bearbeiten der Dateien zu haben. Es kann auch ein anderes Verzeichnis gewählt werden. In dieser Partition muss die Datei `ssh` erstellt werden. Diese Datei aktiviert den SSH-Server. Die Aktivierung kann auch über die grafische Oberfläche von Raspbian erfolgen, jedoch werden die Raspberry Pis in diesem Szenario Headless betrieben, was bedeutet, dass sie ohne grafische Oberfläche betrieben werden. Seit dem Raspbian Image **Raspbian 2016-11-25** muss man den SSH-Server aus Sicherheitsgründen eigenständig aktivieren.[1] Aus diesem Grund muss die Datei `ssh` erstellt werden.

In der Partition `/dev/mmcblk0p2` der SD-Karte müssen die Dateien `/etc/hostname`, `/etc/hosts`, `/etc/network/interfaces` und `/etc/resolv.conf` angepasst werden. Die Datei `hostname` beinhaltet den Namen des Rechners. Die Datei `hosts` beinhaltet alle Knoten des Clusters. Listing 3.1 zeigt die Beispielkonfiguration des Master.

```
# /etc/hosts

192.168.178.200  Master
192.168.178.210  Monitor

192.168.178.201  Slave-1
192.168.178.202  Slave-2
192.168.178.203  Slave-3
192.168.178.204  Slave-4
```

LISTING 3.1: `/etc/hosts` Datei des Master

Im Listing 3.2 ist die Netzwerkkonfiguration des Masters dargestellt. Dort ist die Schnittstelle `eth0` definiert. Diese Schnittstelle ist die Netzwerkschnittstelle des Raspberry Pi. Zu sehen ist, dass diese mit einer statischen IP-Adresse definiert ist. Das rührt daher, dass diese Konfiguration die Administration des Clusters einfacher macht und die Ansteuerung mithilfe von Skripten dadurch erleichtert wird.

[1] https://linuxundich.de/raspberry-pi/ssh-auf-dem-raspberry-pi-aktivieren-jetzt-unter-raspian-noetig/ (abgerufen 02.03.2018)

```
# /etc/network/interfaces

auto eth0
iface eth0 inet static
  address 192.168.178.200
  netmask 255.255.255.0
  gateway 192.168.178.1
  dns-nameservers 192.168.178.1
```

LISTING 3.2: `/etc/interfaces` Datei des Master

Die Datei `/etc/resolv.conf` beinhaltet die Informationen über die bevorzugte Art der Verbindung. In diesem Fall wird eine kabelgebundene Verbindung bevorzugt, weshalb hier die Konfiguration `domain lan` verwendet wird. Außerdem wird hier der Nameserver angegeben.

Nachdem die Netzwerkkonfiguration für alle Knoten des Clusters durchgeführt wurde, kann der Cluster in Betrieb genommen werden. Mithilfe von SSH kann man sich auf die Raspberry Pis einwählen.

Der Aufruf hierfür lautet:

Prompt:$ `ssh pi@Master`

Das standardmäßig vergebene Passwort des Raspberry Pi lautet `raspberry`. Einstellungen am System nimmt man nach dem Einwählen auf den Raspberry Pi mit dem Kommando `sudo raspi-config` vor.

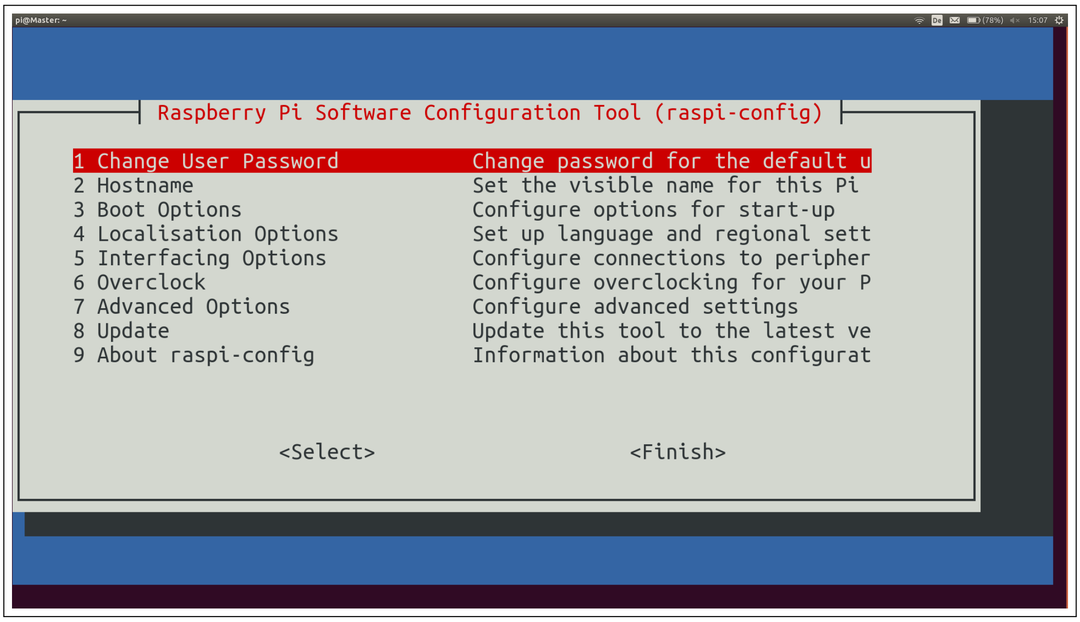


ABBILDUNG 3.2: Menü von raspi-config

Abbildung 3.2 zeigt die verfügbaren Optionen von `raspi-config`. Unter anderem können hier Einstellungen zum Dateisystem vorgenommen werden. Die Option `Expand Filesystem` vergrößert das Dateisystem und passt es an die Größe der eingesteckten SD-Karte an. Abbildung 3.3 zeigt das Menüfenster zum Anpassen der Größe des Dateisystems. Die Option `Change Timezone`, das unter `Localisation Options` zu finden ist, stellt die Zeitzone ein. Die Auswahl der Option Europe/Berlin stellt die Zeit für die deutsche Zeitzone ein. Nachdem das System fertig konfiguriert ist, startet Raspbian das System neu.

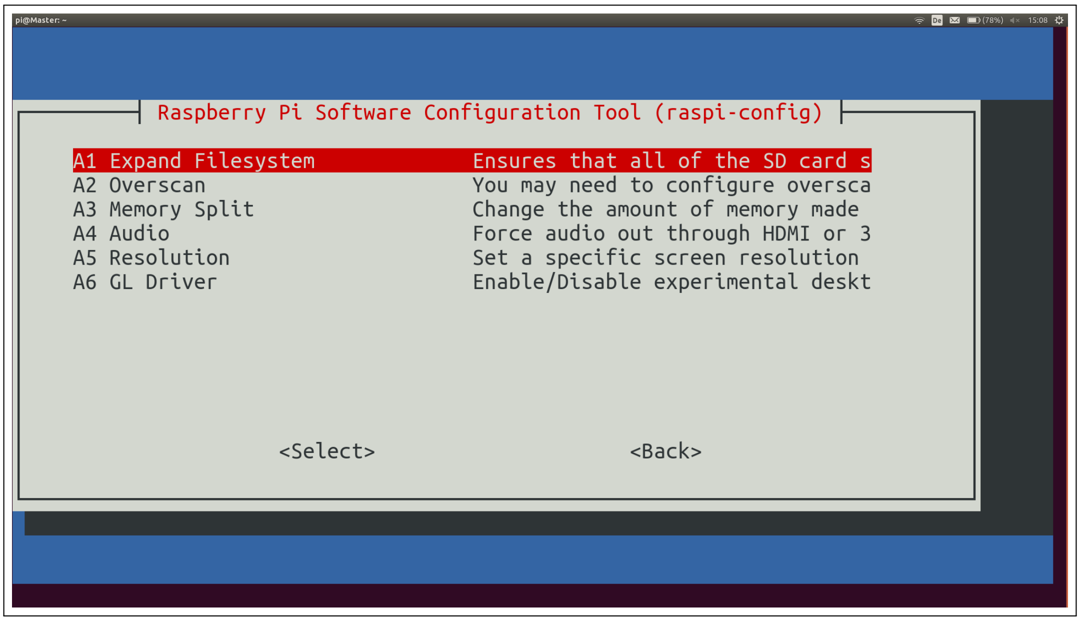


ABBILDUNG 3.3: Option Expand Filesystem

Vom Master aus wird auf die anderen Hosts des Clusters Passwortlos zugegriffen. Das funktioniert mit dem Programm SSH. SSH steht für Secure Shell und dient der Administration von Server-Systemen. Durch Einwahl auf einen Server können Dateien oder Kommandos auf dem Server ausgeführt werden. Dadurch lässt sich der Server über das Terminal wie ein gewöhnlicher Linux Rechner bedienen.

Das Passwortlose Einwählen funktioniert durch das Verteilen von öffentlichen Schlüsseln. Diese Schlüssel lassen sich mit dem Kommando `ssh-keygen -t rsa` generieren. Der Parameter `rsa` zeigt, dass die erstellten Schlüssel RSA Schlüssel sind. Das RSA-Verschlüsselungverfahren gehört zu den asymmetrischen Verschlüsselungsverfahren und beinhaltet einen öffentlichen und einen privaten Schlüssel. Im weiteren Verlauf muss der öffentliche Schlüssel auf die Knoten, die Teilnehmer im Cluster sind, verteilt werden. Die Schlüsselverteilung führt das Kommando `ssh-copy-id -i USER@HOSTNAME` aus.

Im Beispiel des Monitors des Test-Clusters ist das Kommando `ssh-copy-id -i pi@Monitor`. Nach dem Ausführen erscheint die Abfrage des Passworts. Hier wird das Passwort `raspberry` eingegeben.

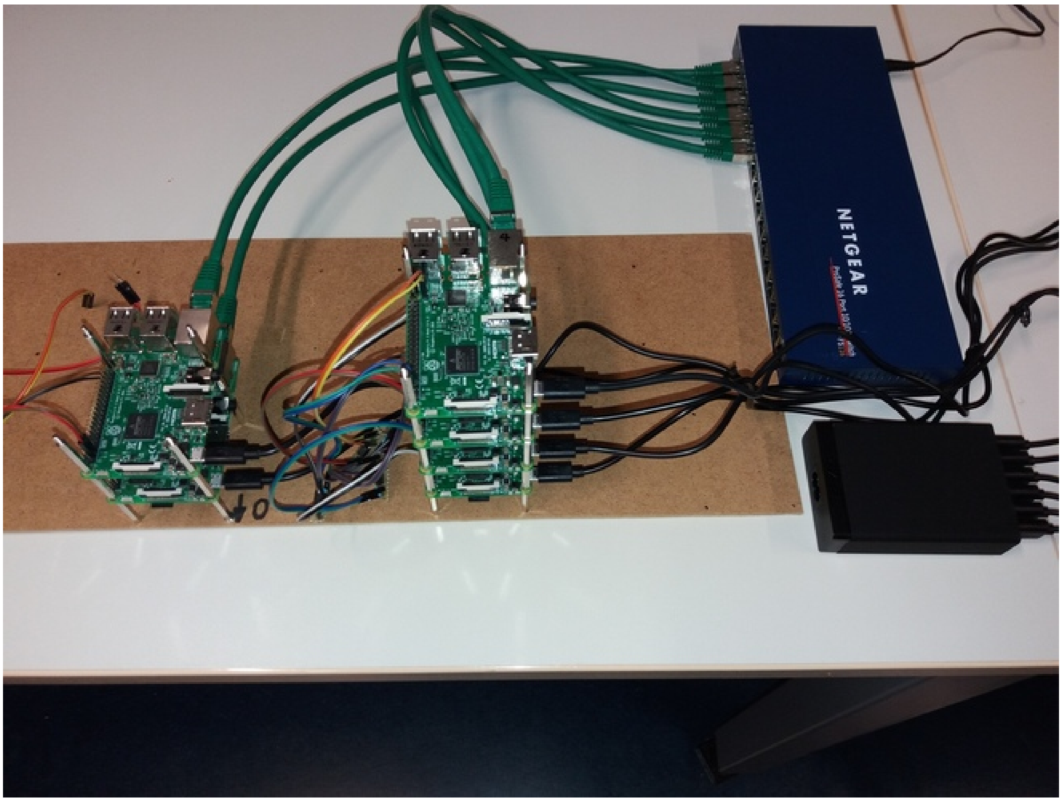


ABBILDUNG 3.4: Bild des Test-Clusters

Abbildung 3.4 zeigt den Aufbau des Testclusters mit angeschlossenen Reset-Schaltungen. Die Raspberry Pis wurden mit Abstandsbolzen zu Türmen aus zwei und vier Knoten zusammengefasst. Der Turm mit zwei Knoten beinhaltet den Master und den Monitor des Clusters. Dabei ist der untere Knoten der Master und der obere der Monitor. Der Turm aus vier Raspberry Pis beinhaltet die Knoten Slave-1 bis Slave-4 in aufsteigender Reihenfolge. Die Türme sind aus Stabilitätsgründen auf eine Holzplatte geschraubt. Angeschlossen sind alle Knoten an einen 16-Port Gigabit-Switch von Netgear. Die Raspberry Pis werden über eine 6-Port USB-Stromversorgung von Anker mit Strom versorgt. Diese hat eine Leistung von 60 Watt.

Die in diesem Kapitel gezeigten Arbeitsschritte stellen eine Testumgebung bereit. Die Schaltung und Skripte aus Kapitel 4 werden auf diesem Test-Cluster erprobt. Die Arbeiten und Tests der Resetschaltung führen dank dieser Testumgebung zu keinen Beeinträchtigungen im Ablauf des eigentlichen Clusters im Raum 1-149.

Lösung zum Reset der Cluster-Knoten

Dieses Kapitel beschreibt die Konzeption und Realisierung einer Reset Lösung für den Raspberry Pi. Dabei beschäftigt sich ein Abschnitt mit unterschiedlichen Möglichkeiten für den Reset der Raspberry Pis. Anschließend wird der Aufbau einer konkreten Lösung mithilfe von Optokopplern erläutert. Ein weiterer Abschnitt zeigt das Ansteuern von GPIO-Ports mit Shell-Skripten. Abschließen wird dieses Kapitel mit der Implementation von Skripten zum automatischen Neustart von ausgefallenen Knoten.

4.1 Möglichkeiten zum Neustarten des Raspberry Pi

Die Anforderungen, die an eine Reset-Funktionalität gestellt werden, machen den Bau einer Schaltung erforderlich. Diese Schaltung muss in der Lage sein die beiden Pins des P6-Ports kurz zuschließen. Am anderen Ende der Schaltung wird keine Spannung geschaltet, weil dort nur die Spannung vom Prozessor abgezogen werden soll. Somit benötigt man ein elektronisches Bauteil, das diese Funktion realisiert. Diese Schaltung kann durch ein Ausgangssignal eines GPIO-Ausgangs beschaltet werden.

Zur Auswahl stehen folgende Möglichkeiten:

- Relais
- Transistoren
- Optokoppler

Relais sind Schalter, die durch ein Signal am Eingang des Relais geschaltet werden können. Dabei wird im inneren des Relais eine Spule zum Schalten eines Kontakts verwendet. Durch anlegen einer Spannung am Eingang kann dieser Kontakt geschlossen und geöffnet werden. Durch Relais lassen sich einfache Schaltungen realisieren.

Sie haben jedoch den Nachteil, dass sie mechanisch einen Kontakt herstellen und dadurch stoßempfindlich sind. Dadurch unterliegen sie Bauart bedingt einem Verschleiß, der durch häufige Schaltvorgänge zum Ausfall des Bauteils führen kann. Außerdem sind sie groß und langsam.

Eine weitere Möglichkeit zum Schalten stellen Transistoren dar. Transistoren sind Halbleiter-Bauelemente, die zum Schalten von niedrigen Spannungen geeignet sind. Transistoren stellen einen sehr wichtigen und weit verbreiteten Typ von Schalter dar. So werden Transistoren in integrierten Schaltungen verwendet, um Prozessoren und Mikrocontroller herzustellen.

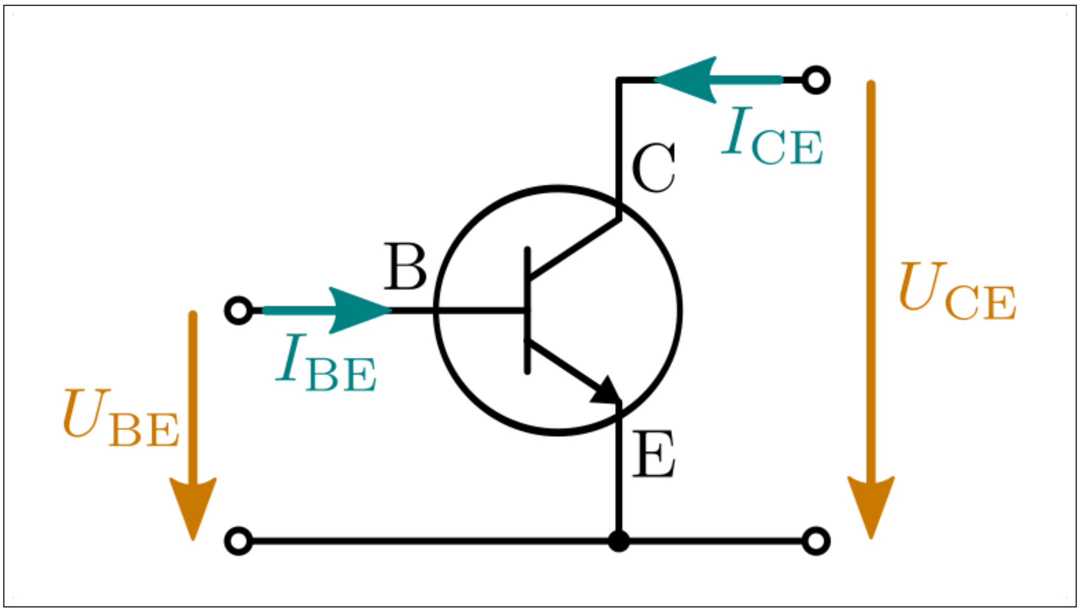


ABBILDUNG 4.1: Aufbau eines npn-Transistors
Quelle: `https://www.grund-wissen.de/elektronik/bauteile/transistor.html` (abgerufen 02.03.2018)

Abbildung 4.1 zeigt den Aufbau eines npn-Transistors. Das Bild zeigt den Kollektor, die Basis und den Emitter (in absteigender Reihenfolge). Wird zwischen Emitter und Basis eine Spannung von 0,7 Volt angelegt, so wird der Transistor zwischen Emitter und Kollektor leitfähig und der Transistor schaltet in dieser Richtung durch. Sobald die Basis-Emitter-Spannung (U_{BE}) abgezogen wird, sperrt der Transistor. Dabei wird der Kollektor-Emitter Spannungskreis als Arbeitsspannungskreis bezeichnet (U_{CE}) [19]. Der Name Transistor leitet sich vom Zusammenschluss der beiden Wörter Transfer und Resistor ab. Durch ihre Bauweise sind Transistoren in der Lage Strom zu sperren und zu schalten.

Eine Schaltung mit Transistoren ist prinzipiell möglich, jedoch schließen Transistoren den Stromkreis. Durch dieses Verhalten muss im Kollektor-Emitter Stromkreis ein Widerstand aufgebracht werden, der den durchgeschalteten Strom ableiten muss.

Optokoppler bieten im Unterschied zu Transistoren eine galvanische Trennung. Das heißt, dass zwischen Eingang und Ausgang keine elektrische Verbindung besteht. Somit sind der Eingangs- und Ausgangsstromkreis untereinander **potentialfrei**. Zwischen Eingang und Ausgang kann somit keine Spannung fließen. Optokoppler werden überall da eingesetzt, wo eine Schaltung erforderlich ist, jedoch zwei Stromkreise voneinander getrennt bleiben müssen.

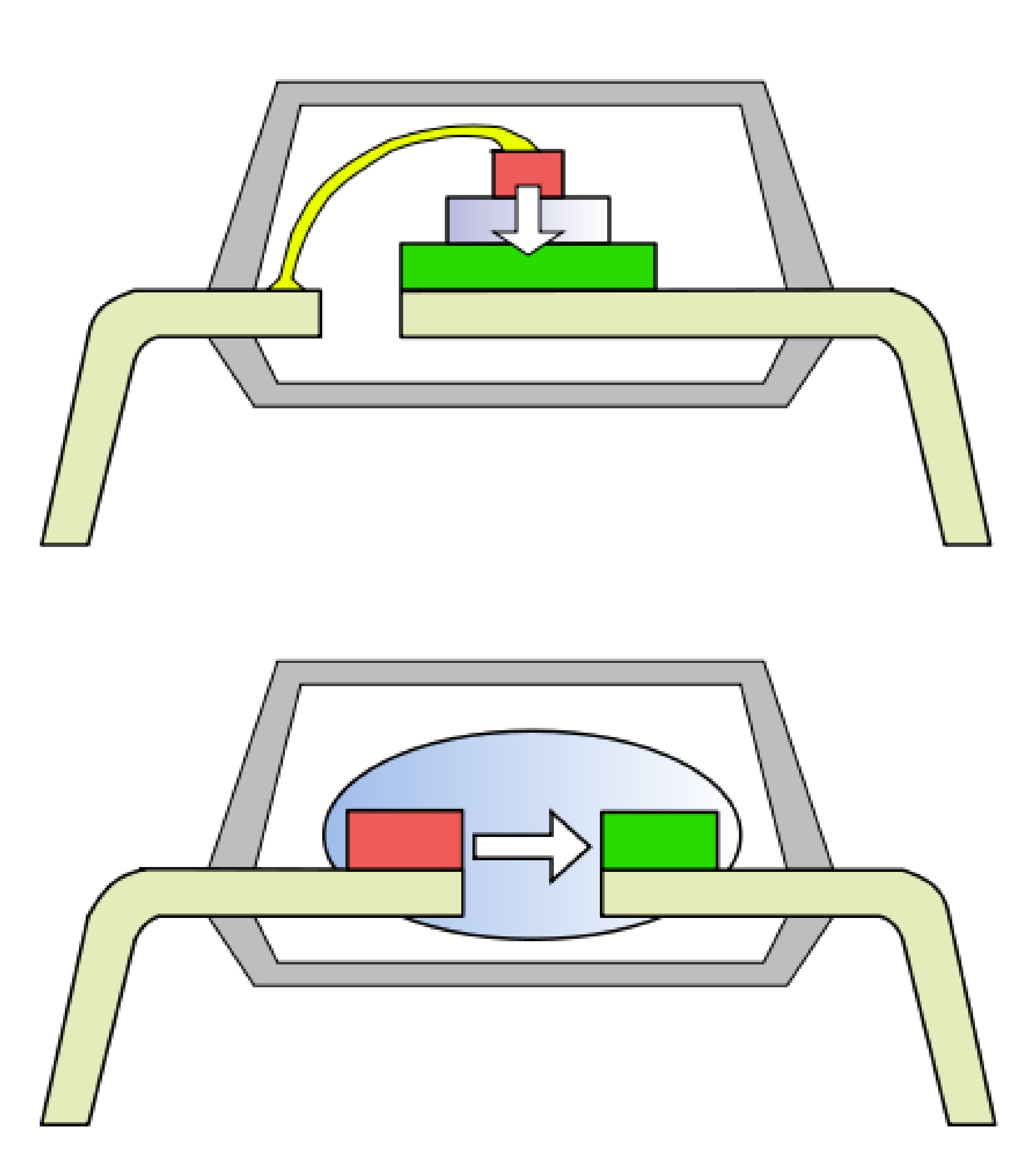


ABBILDUNG 4.2: interner Aufbau eines Optokopplers
Quelle: `https://de.wikipedia.org/wiki/Optokoppler` (abgerufen 02.03.2018)

Abbildung 4.2 zeigt den internen Aufbau eines Optokopplers. Das Funktionsprinzip ist sehr einfach. Auf der einen Seite befindet sich eine Leuchtdiode die bei anliegender Spannung einen Lichtimpuls aussendet der auf der anderen Seite auf eine Photodiode oder einen Phototransistor trifft. Dadurch wird der Ausgang geschaltet und der Ausgangsstromkreis wird geschlossen.

Das elektrische Eingangssignal wird in ein optisches Signal umgewandelt [20]. Außerdem lassen sich digitale und analoge Signale mit einem Optokoppler übertragen, wodurch sie sich für Anwendungen in der Nachrichtentechnik eignen.

Aus den angeführten Eigenschaften des Optokopplers zeigt sich, dass sich dieses elektronische Bauteil zur Realisierung einer Reset-Schaltung eignet. Durch die Verwendung eines Optokopplers lässt sich eine galvanische Trennung zwischen dem GPIO-Ausgang und dem P6-Port des Raspberry Pi bewerkstelligen. Da es zu keiner elektrischen Verbindung kommt, lässt sich eine Schaltung zwischen zwei Raspberry Pi erstellen. Dadurch ist der sichere Betrieb beider Raspberry Pis gewährleistet.

4.2 Aufbau einer Resetschaltung mit Optokopplern

Da eine Resetschaltung für Cluster-Knoten erarbeitet werden soll, muss zunächst eine Methode zum Schalten des Resets gefunden werden. Ein ausgefallener Knoten kann sich nicht selbst schalten, da er sich in einem Zustand befindet, indem er inaktiv ist. Daher bietet sich die Verschaltung von zwei Knoten an. Dabei wird der GPIO-Ausgang eines Knotens an den Reset-Port eines anderen Knotens geschaltet und umgekehrt. Das heißt, dass jeder Rapsberry Pi Knoten eine Optokoppler Schaltung benötigt.

Dieser Abschnitt beschreibt den Aufbau einer Schaltung zum Neustart von zwei Knoten. Dabei wird von den GPIO-Ausgängen der Raspberry Pis Verwendung gemacht. Jeweils zwei GPIO-Ausgänge werden mit dem Eingang des Optokopplers verbunden. An den Ausgängen der Schaltung werden jeweils zwei Leitungen mit dem P6-Port verbunden. Anhang A.2 zeigt eine schematische Darstellung dieser Schaltung.

Die Schaltung aus Abbildung 4.3 besteht aus einem 39 Ω Widerstand, einem 10 kΩ Widerstand und einem DIP-4 Optokoppler. Der Optokoppler wurde verwendet, da er eine galvanische Trennung bietet, wie sie bei Transistoren nicht vorgefunden werden kann. Außerdem ist diese Schaltung in seinen Abmessungen kompakter als eine vergleichbare Relaisschaltung. Formel 4.1 zeigt das Ohmsche Gesetz. Mit dem Ohmschen Gesetz lassen sich die Werte für Spannung (U), Widerstand (R) und Strom (I) ermitteln. Das Ohmsche Gesetz stellt eines der Grundgesetze der Elektrotechnik dar und ist unverzichtbar in der Berechnung von Schaltungen. Der 39 Ω Vorwiderstand wird benötigt, um die Spannung zum Schalten des Optokopplers zu erhalten (siehe Rechnung 4.4).

Damit dieser spezifische Optokoppler schalten kann, braucht dieser eine Spannung im Bereich von 1,3 Volt bis 1,65 Volt an der Sendediode des Optokopplers. Die typische Spannung zum Schalten liegt laut Datenblatt bei 1,35

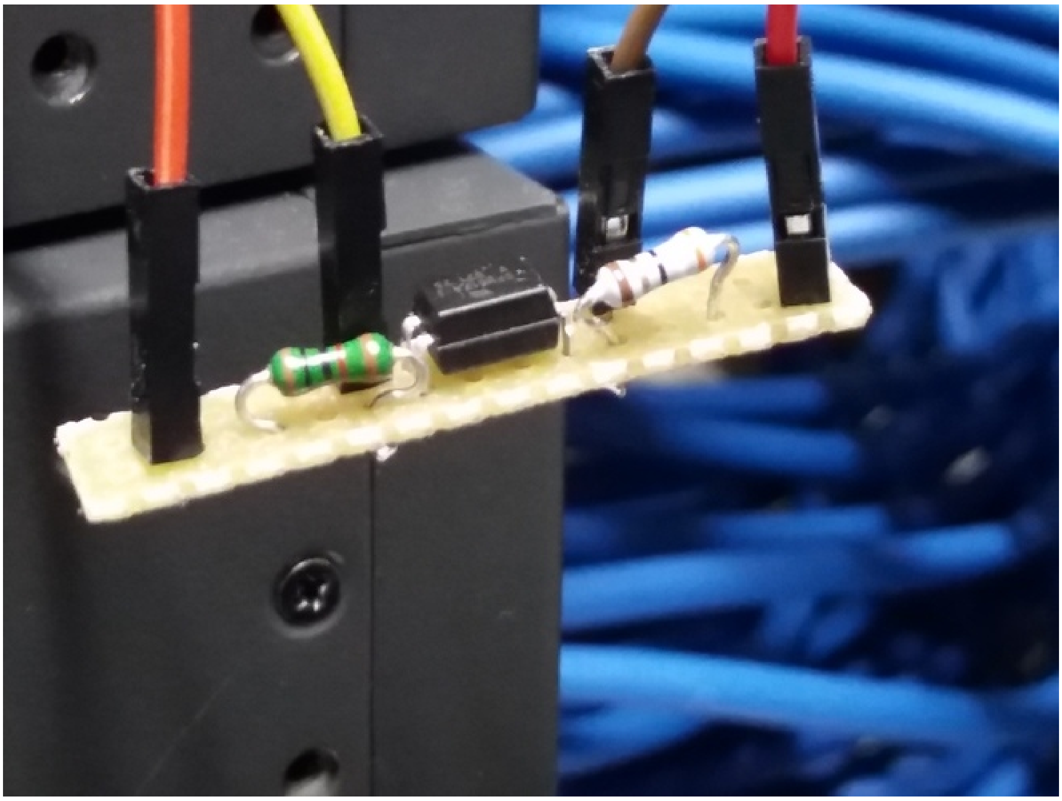


ABBILDUNG 4.3: Prototyp Schaltung mit Optokoppler

Volt [21]. Die GPIO-Ports 2 und 3 liefern jeweils 3,3 Volt Spannung und daher benötigt man diesen Vorwiderstand.

$$U = R * I \tag{4.1}$$

$$U_{\text{Widerstand}} = 3,3\ V - 1,35\ V = 1,95\ V \tag{4.2}$$

$$I_{\text{GPIO}} = 50\ mA \tag{4.3}$$

$$R_{\text{Vorwiderstand}} = \frac{U_{\text{Widerstand}}}{I_{\text{GPIO}}} = \frac{1,95\ V}{0,05\ A} = 39\ \Omega \tag{4.4}$$

Rechnung 4.2 zeigt die Spannung $U_{\text{Widerstand}}$, welche durch einen geeigneten Widerstand abgezogen werden muss. Rechnung 4.4 zeigt den Wert des Widerstands für diese Schaltung bei einem angenommenen maximal Strom von 50 mA. Der 10 kΩ Widerstand wird zum Begrenzen des entstehenden Stroms der internen Hardware benötigt und soll den Hauptprozessor des Raspberry Pi schützen. Die Berechnungen wurden auf eine Stromstärke von 50 mA am GPIO-Ausgang ausgelegt. Jeder GPIO-Port des Raspberry Pi liefert eine Stromstärke von 16 mA und die Raspberry Pi Foundation gibt die maximale kumulierte Stromstärke aller GPIO-Ports mit 50 mA an [22]. Aus Sicherheitsgründen wurde mit der maximalen Stromstärke gerechnet, um den sicheren Betrieb des Optokopplers zu gewährleisten. Dieser Wert wurde als Ausgangssituation für das Messen einer Testschaltung angenommen.

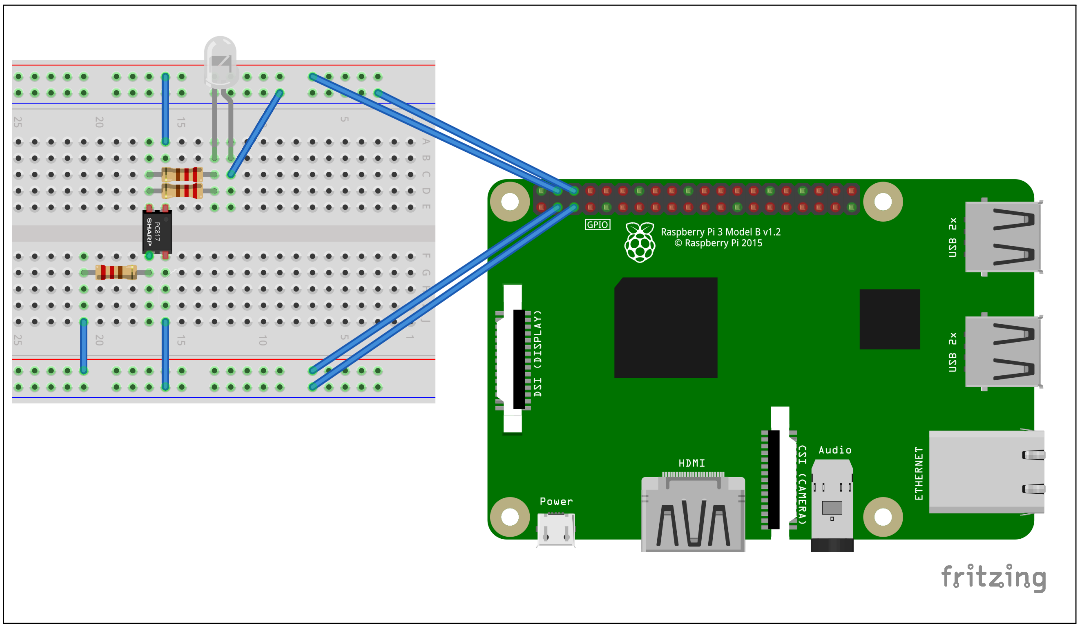


ABBILDUNG 4.4: Darstellung der Schaltung zum Messen des Vorwiderstands

Die oben aufgestellten Formeln dienen als Grundlage zum Auslegen der Schaltung. Die errechneten Werte dienen als erster Ausgangswert und werden auf einem Steckbrett getestet. Abbildung 4.4 zeigt das Schema dieser Schaltung. Dazu wird eine Schaltung mit einem 39 Ω Vorwiderstand und einem Optokoppler zusammen gesteckt. Diese Schaltung dient als Testschaltung. Die Werte für Strom und Spannung an dieser Schaltung werden mithilfe eines Multimeters gemessen. U_{IN} ist die Spannung, die am Eingang des Optokopplers anliegt. I_{IN} steht für den Strom, der am Eingang des Optokopplers gemessen wird. Am Ausgang des Optokopplers werden eine LED und ein Vorwiderstand von 165 Ω für die LED geschaltet. Die Leuchtdiode am Ausgang des Optokopplers dient als Last und Indikator für die Funktion der Schaltung. Die Leuchtdiode des Ausgangs wird mit dem 5 Volt Ausgang des Raspberry Pi betrieben. Die unten aufgeführten Werte zeigen die Ergebnisse der Messungen.

$$R1 = 39\,\Omega \quad U_{\mathrm{IN}} = 1,23\ V \quad I_{\mathrm{IN}} = 16,5\ mA \tag{4.5}$$

$$R2 = 100\,\Omega \quad U_{\mathrm{IN}} = 1,23\ V \quad I_{\mathrm{IN}} = 11,5\ mA \tag{4.6}$$

$$R3 = 1000\,\Omega \quad U_{\mathrm{IN}} = 1,05\ V \quad I_{\mathrm{IN}} = 2\ mA \tag{4.7}$$

Die gemessenen Werte zeigen, dass eine Schaltung mit 39 Ω eine Stromstärke von ca. 16,5 mA ergeben. Zwar ist im Datenblatt des Optokopplers angegeben, dass ein maximaler Strom von 60 mA am Eingang des Optokopplers anliegen dürfen [21]. Diese Stromstärke führt jedoch zu einer erhöhten Belastung am GPIO-Port des Raspberry Pi. Die Werte der Messung mit einem Widerstand von 1 kΩ führen zu einer Stromstärke von 2 mA. Dieser Wert ist ein idealer Wert in Bezug auf die Port Belastung des Raspberry Pi. Hierbei sinkt jedoch die gemessene Schaltspannung am Eingang des Optokopplers ab und ist zu gering, um diesen zuverlässig zu schalten. Durch den hohen Vorwiderstand am Eingang sinkt die Spannung über den Optokoppler stark ab. Die Werte aus Messung 4.6 sind besser geeignet, um die Belastung für den GPIO-Port gering zu halten und die Schaltleistung des Optokopplers zu liefern. Daher ist eine Schaltung mit einem 100 Ω Vorwiderstand am Eingang des Optokopplers zu bevorzugen.

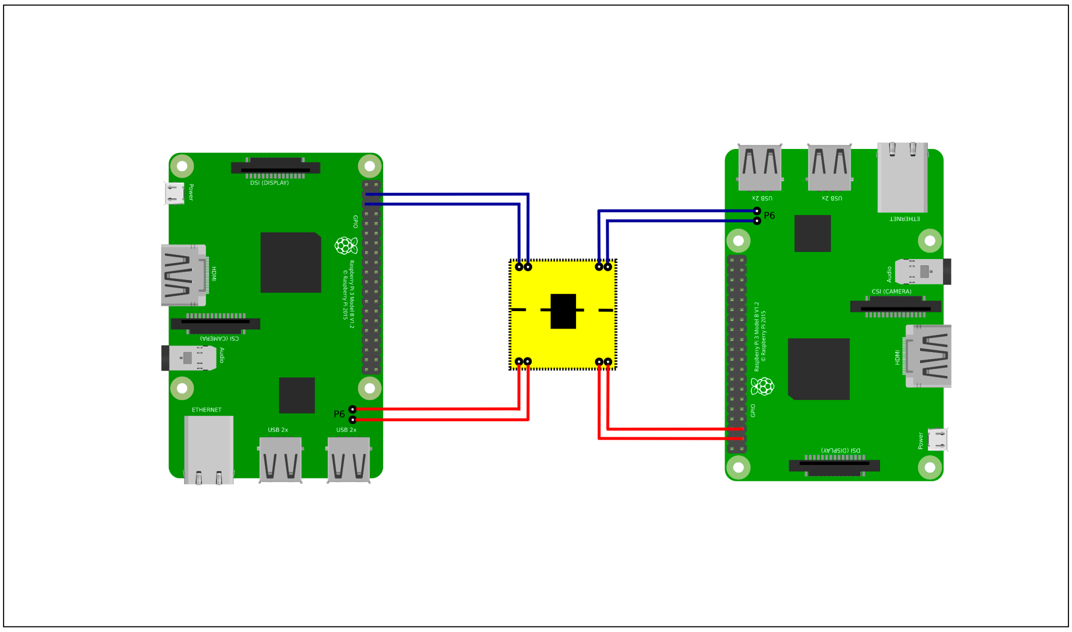


ABBILDUNG 4.5: Darstellung der Reset Schaltung mit 2 Raspberry Pi

Abbildung 4.3 zeigt den Prototyp für einen unidirektionalen Reset. Diese Schaltung kann einen Knoten von einem anderen Raspberry Pi aus neu starten. Abbildung 4.5 zeigt, wie zwei Raspberry Pi durch eine zusammenhängende Reset Schaltung verbunden sind. Dabei zeigen die blauen Linien die Verbindung der GPIO-Ports 2 und 3 des linken Raspberry Pi mit dem P6 Port des rechten Raspberry Pi. Die roten Linien zeigen die Verbindung in anderer Richtung. Das gelbe Quadrat zwischen den beiden Raspberry Pi steht für die Reset Schaltung. So können sich beide Raspberry Pi im Fehlerfall gegenseitig neu starten.

Im nachfolgenden Abschnitt wird erklärt, wie die GPIO-Ports mithilfe von Shell-Skripten angesteuert werden können. Dabei wird eine konkrete Implementation zum Schalten des Resets gezeigt und erläutert.

4.3 Ansteuerung der GPIO-Ports mit Shell Skripten

Zum Schalten des Reset wird ein Shell Skript verwendet. Das Betriebssystem Raspbian beinhaltet eine Möglichkeit die GPIO-Ports ohne Installation von zusätzlichen Bibliotheken anzusteuern. Zugegriffen wird dabei auf die Dateien im Verzeichnis `/sys/class/gpio/`. In diesem Verzeichnis befinden sich zwei Dateien namens `export` und `unexport` [23]. Der `echo`-Befehl spricht den Port an, den man setzen will. Wichtig ist hierbei zu beachten, dass die hier angegebenen Ports die GPIO-Port Nummerierung sind und nicht die Pin-Nummerierung.

```
1  #!/bin/sh
2
3  # Dieses Skript sorgt dafuer dass nach dem
4  # booten die GPIO Ports 2 und 3 initialisiert werden
5
6  for Port in 2 3
7  do
8    echo "$Port" > /sys/class/gpio/export
9    echo "out" >/sys/class/gpio/gpio${Port}/direction
10   echo "0" >/sys/class/gpio/gpio${Port}/value
11   chmod 660 /sys/class/gpio/gpio${Port}/direction
12   chmod 660 /sys/class/gpio/gpio${Port}/value
13 done
```

LISTING 4.1: Listing zum Initialisieren der GPIO-Ports 2 und 3

Das Listing 4.1 zeigt ein kurzes Skript, welches die GPIO-Ports 2 und 3 initialisiert. In Zeile 8 wird auf die Datei `export` zugegriffen, wodurch die Datei `gpio${Port}` erstellt wird. In Zeile 9 und 10 wird die Art des Ports und sein Wert festgelegt. Zur Auswahl der Art stehen die Möglichkeiten „in“ und „out“.

Das legt fest, ob es sich um einen Eingang oder Ausgang handelt. Der Wert des Ausgangs aus Zeile 10 wird auf „0 " gesetzt. Dadurch wird keine Spannung an den Ausgang gelegt, somit gibt es kein Ausgangssignal.

Der Aufruf von `chmod 660` setzt die Rechte der GPIO-Ports für den Besitzer und die Gruppe auf les- und schreibbar. Damit dieses Skript ausgeführt werden kann, muss das Kommando `sudo chmod +x <SKRIPTNAME>.sh` ausgeführt werden. Das in Listing 4.1 gezeigte Skript initialisiert zum Start des Raspberry Pi die GPIO-Ports. Das erreicht man dadurch, dass man die erstellte Datei in das Verzeichnis `/etc/init.d/` kopiert. Der Aufruf von `sudo update-rc.d <SKRIPTNAME> defaults` teilt dem Betriebssystem mit das Skript mit dem Namen `SKRIPTNAME` beim Booten auszuführen. Die GPIO-Ports sind initialisiert und können verwendet werden.

Das Kommando `echo "1" >/sys/class/gpio/gpio2/value` setzt ein Ausgangssignal auf den GPIO-Port 2 und löst den Reset aus. Die Datei `/sys/class/gpio/gpio2/value` muss wieder auf 0 gesetzt werden, sonst liegt ein Dauersignal am GPIO-Port an und der Raspberry Pi kann nicht starten.

Dieses kurze Beispiel zeigt, wie einfach die GPIO-Ports über die Kommandozeile angesteuert werden können. Das Skript 4.1 zeigt die Möglichkeit eines Neustarts des Raspberry Pi auf. Dieses Skript ist in Verbindung mit der in Abschnitt 4.2 vorgestellten Schaltung eine hilfreiche Methode, um abgestürzte Knoten neu zu starten. Jedoch sollte durch Verwendung von Shell Skripten ein Potenzial zur Automatisierung dieses Prozesses bestehen. Abschnitt 4.4 geht auf die Implementation einer automatischen Reset-Lösung ein und testet diese Skripte auf der Testumgebung aus Kapitel 3.

4.4 Erstellen eines automatisierten Skripts

Die Ansteuerung der GPIO-Ports mittels Shell Skripten ist eine einfache und schnelle Möglichkeit, die Cluster-Knoten neu zu starten. Mit Shell Skripten können große Ansammlungen von Kommandos zusammengefasst und automatisiert werden. Aus diesem Grund soll auch der Reset der Knoten automatisiert werden. Die Grundidee hinter einem automatischen Skript ist, dass das Skript eigenständig prüfen kann, ob die Knoten des Clusters laufen oder nicht. Wichtig für eine automatisierte Lösung ist auch die Tatsache, dass das Skript automatisch und sicher handelt, sollte ein Knoten nicht funktionieren. Diese Punkte führen zu der Überlegung, welche Methoden bestehen, um automatisch zu reagieren.

Das Kommando `ping` des Betriebssystems schickt Anfragen über das Netzwerk an einen Rechner. Im weiteren Verlauf wird dieser Rechner **Host** genannt. `ping` sendet diese Anfragen und wertet sie aus.

Ist ein Host erreichbar, so gibt `ping` eine Nachricht darüber aus. Diese Eigenschaft macht `ping` zu einem geeigneten Tool zur Überwachung der Cluster-Knoten [24].

Das Skript 4.2 benutzt das Kommandozeilenprogramm `ping`, um den Master des Clusters kontinuierlich abzufragen und seine Erreichbarkeit zu prüfen.

```
1  #!/bin/bash
2  # Skript pingt Master an und betaetigt Reset
3  # wenn Master nicht erreichbar
4
5  sleep 60
6
7  while true
8  do
9    COUNT=$(ping -c 5 Master | grep 'received' | awk -F',' '{ print
      $2 }' | awk '{print $1}')
10   if [ $COUNT -eq 0 ]; then
11     echo "Host: Master ist ausgefallen! Datum: $(date)" >> ~/Logs/
      PingLog.txt
12     echo "Host: Master wurde neu gestartet! Datum: $(date)" >> ~/
      Logs/PingLog.txt
13     echo "1" >/sys/class/gpio/gpio2/value
14     sleep 0.01
15     echo "0" >/sys/class/gpio/gpio2/value
16   fi
17 sleep 120
18 done
```

LISTING 4.2: Shell Skript für automatischen Reset des Masters

Zeile 7 zeigt eine Endlosschleife, die über das gesamte Skript reicht. Diese Endlosschleife sorgt dafür, dass die darauf folgenden Anweisungen durchgängig ausgeführt werden. Die Überprüfung der Erreichbarkeit soll immer gewährleistet sein.

Zeile 9 zeigt den Aufruf des Kommandos `ping` mit dem Parameter `-c 5`, was bedeutet, dass `ping` fünf Anfragen sendet. Die Ausgabe von diesem Aufruf dient als Eingabe für das Kommando `grep`. Das Zeichen | ist der sogenannte Pipe-Operator, der dazu dient Ausgaben eines Kommandos an ein weiteres Kommando zu leiten. Somit ist zum Beispiel bei `KOMMANDO1 | KOMMANDO2` die Ausgabe von `Kommando1` die Eingabe von `Kommando2`.

`grep` ist ein Programm, dass in Linuxsystemen dazu dient reguläre Ausdrücke in Dateien zu suchen. Hier sucht `grep` nach dem Wort „`received`“. Der Aufruf von `awk -F',' '{ print $2}'` setzt den sogenannten **Field Seperator**. In diesem Fall lautet der Field Separator ','. Die Rückgabe von `ping` trennt die einzelnen Parameter durch ein Komma, daher trennt `awk` die Ausgabe an diesem Zeichen. Weiter ist in dem Kommando der Parameter `'{ print $2}'` angegeben. Dieser Parameter gibt an, dass der zweite String ausgelesen werden soll.

Am Ende der Zeile steht ein weiterer Aufruf der Form `awk '{print $1}'`. Dabei wird die Rückgabe des vorangegangenen `awk` nochmals unterteilt. Die Variable `COUNT` ist eine ganze Zahl und speichert die Rückgabe dieser verschachtelten Kommandos. `COUNT` stellt die Zahl der erfolgreichen Anfragen dar, die bei Erfolg ungleich 0 ist. Wenn keine Anfrage angekommen ist, also der Host nicht erreichbar ist, wird dieser Wert 0.

Hier wird jede einzelne Ausgabe jedes Kommandos farblich markiert:

Kommando: `ping -c 1 192.168.178.200`
Ausgabe: `1 packets transmitted, 1 received, 0% packet loss, time 0ms`
Kommando: `grep 'received'`
Ausgabe: `1 packets transmitted,1 received, 0% packet loss, time 0ms`
Kommando: `awk -F',' '{ print $2 }'`
Ausgabe: `1 received`
Kommando: `awk '{print $1}'`
Ausgabe: `1`

In Zeile 10 des Skripts prüft die if-Bedingung, ob der Wert 0 ist, die Anfragen also nicht ankamen. Ist das der Fall schreiben die `echo`-Kommandos aus Zeile 11 und 12 den Hostnamen und das Datum mit Uhrzeit in die Datei `PingLog.txt`, welche sich im Verzeichnis `Logs` befindet. Zeile 13 bis 15 sorgen für den Neustart des Masters, indem eine 1 in die Datei `/sys/class/gpio/gpio2/value` geschrieben wird. Nach einer Wartezeit von 10ms schaltet das Kommando `echo "0" >/sys/class/gpio/gpio2/value` den GPIO-Port 2 auf 0. Somit hat dieser Port kein Ausgangssignal mehr. Die Wartezeit von 10ms wurde gewählt, da sichergestellt werden soll, dass der Raspberry Pi und die angeschlossene Schaltung genug Zeit haben zu schalten. Zeile 17 lässt das Skript 120 Sekunden warten, bevor die Anfragen erneut gestellt werden. Die Wartezeit von 60 Sekunden in Zeile 5 soll das Neustarten des Masters bei einem gleichzeitigen Hochfahren der beiden Raspberry Pis verhindern. Ein schnellerer Boot-Vorgang des Monitors könnte den Master sonst ungewollt neu starten

```
Host: Master ist ausgefallen! Datum: Tue 6 Feb 12:20:25 CET 2018
Host: Master wurde neu gestartet! Datum: Tue 6 Feb 12:20:25 CET
   2018
Host: Master ist ausgefallen! Datum: Tue 6 Feb 12:26:55 CET 2018
Host: Master wurde neu gestartet! Datum: Tue 6 Feb 12:26:55 CET
   2018
Host: Master ist ausgefallen! Datum: Tue 6 Feb 12:32:55 CET 2018
Host: Master wurde neu gestartet! Datum: Tue 6 Feb 12:32:55 CET
   2018
```

LISTING 4.3: Logdatei des automatischen Reset Skripts

Das Skript 4.2 zeigt, dass sich ein automatischer Reset mit weniger als 20 Zeilen Quellcode implementieren lässt. Listing 4.3 zeigt die erzeugten Logdateien. Darin sind die wichtigsten Informationen für den Administrator zusammengefasst. Dadurch lassen sich Schlüsse über die Zuverlässigkeit des Knotens treffen und Statistiken anlegen.

Ein weiteres Shell Skript soll die Cluster-Knoten neu starten. Hierfür wird das Skript 4.2 erweitert. Das Listing 4.4 zeigt die Änderungen, die notwendig sind um die Reset-Funktionalität auf die Cluster-Knoten zu erweitern. In Zeile 1 sieht man eine weitere Schleife, welche die Hostnamen der einzelnen Knoten aus einer Textdatei namens `Slaves.txt` liest. Diese Datei beinhaltet die Hostnamen der Knoten des Clusters. Für die zugrundeliegende Testumgebung sind das die Knoten **Slave-1** bis **Slave-4**. Diese Knoten werden in Zeile 3 mit dem `ping` Kommando abgefragt. Diese Zeile ist bis auf die Variable für die Hosts identisch zum Skript 4.2. Auch in diesem Skript werden die Knoten des Clusters fünfmal abgefragt und wenn das Ergebnis der Anfragen gleich 0 ist, wird die if-Abfrage ausgeführt. Die Zeilen 5 und 6 schreiben den Hostnamen und das Datum in eine Logdatei namens `PingLog.txt`.

```
1  while read HOST
2  do
3    COUNT=$(ping -c 5 $HOST | grep 'received' | awk -F',' '{ print $2
      }' | awk '{print $1}')
4    if [ $COUNT -eq 0 ]; then
5      echo "Host: "$HOST "ist ausgefallen! Datum: $(date)" >> ~/Logs/
     PingLog.txt
6      echo "Host: "$HOST "wurde neu gestartet! Datum: $(date)" >> ~/
     Logs/PingLog.txt
7
8      SLAVENO=$(echo "$HOST" | awk -F'-' '{ print $2}')
9
10     if (( $SLAVENO % 2 ==  1 )); then
11     ssh -n Slave-$(( $SLAVENO + 1 )) "echo "1" >/sys/class/gpio/
     gpio2/value"
12     sleep 0.01
13     ssh -n Slave-$(( $SLAVENO + 1 )) "echo "0" >/sys/class/gpio/
     gpio2/value"
14
15     else
16     ssh -n Slave-$(( $SLAVENO - 1 )) "echo "1" >/sys/class/gpio/
     gpio2/value"
17     sleep 0.01
18     ssh -n Slave-$(( $SLAVENO - 1 )) "echo "0" >/sys/class/gpio/
     gpio2/value"
19     fi
20   fi
21 done < Slaves.txt
```

LISTING 4.4: Shell Skript für automatischen Reset der Slaves

Zeile 8 zeigt den Unterschied zum Skript 4.2. In dieser Zeile kommt das Programm `awk` zum Einsatz um den Hostnamen des ausgefallenen Knoten zu manipulieren. Die Trennung der Hostnamen findet am Zeichen „- “ statt. Anschließend wird das zweite Argument in der Variablen `SLAVENO` gespeichert. Darin befindet sich die Zahl nach dem Bindestrich des Hostnamen. Aus diesem Grund wurde das Format der Hostnamen gewählt. Das erleichtert die Arbeit mit dem Programm `awk`.

Mit der Nummer des Hostnamen kann in Zeile 10 eine Modulo-Operation ausgeführt werden. Diese Berechnung ermittelt, ob es sich um einen geraden oder einen ungeraden Knoten handelt. Das ist wichtig, da die Verschaltung der Knoten immer paarweise ausgeführt ist. Dadurch schaltet jeder ungerade Knoten seinen Nachbarn mit einer geraden Nummer und jeder gerade Knoten seinen ungeraden Nachbarn. Aus diesem Grund ermittelt die Modulo-Operation, ob die Zahl im Hostnamen des Knoten gerade oder ungerade ist.

Zeile 11 bis 13 zeigen den Reset für einen ungeraden Knoten. Dabei wird das Kommando `ssh -n Slave-$(( $SLAVENO + 1 ))` verwendet, um den ausgefallenen ungeraden Knoten neuzustarten. So wird beispielsweise Slave-1 durch Slave-2 neugestartet. Ist Slave-1 ausgefallen wird die Nummer von Slave-1 um eins erhöht und Slave-2 schaltet Slave-1. Das Kommando `ssh - n` führt Kommandos auf einem entfernten Rechner aus. Zeile 15 bis 18 zeigen den Fall für einen ungeraden Knoten. Dabei wird die Nummer des Knotens um eins erniedrigt.

Durch diese Änderungen am Reset Skript 4.2 können sich die Cluster-Knoten wechselseitig neu starten. Die Anordnung der Schaltung und die Auswahl der Hostnamen sorgen für eine einfache Implementation der Reset-Funktionalität für Cluster-Knoten.

```
Host: Slave—2 ist ausgefallen! Datum: Sun 11 Feb 18:49:49 CET 2018
Host: Slave—2 wurde neu gestartet! Datum: Sun 11 Feb 18:49:49 CET
    2018
Host: Slave—3 ist ausgefallen! Datum: Sun 11 Feb 18:50:04 CET 2018
Host: Slave—3 wurde neu gestartet! Datum: Sun 11 Feb 18:50:04 CET
    2018
Host: Slave—1 ist ausgefallen! Datum: Sun 11 Feb 18:52:05 CET 2018
Host: Slave—1 wurde neu gestartet! Datum: Sun 11 Feb 18:52:05 CET
    2018
Host: Slave—4 ist ausgefallen! Datum: Sun 11 Feb 18:52:28 CET 2018
Host: Slave—4 wurde neu gestartet! Datum: Sun 11 Feb 18:52:28 CET
    2018
Host: Slave—1 ist ausgefallen! Datum: Sun 11 Feb 18:54:43 CET 2018
Host: Slave—1 wurde neu gestartet! Datum: Sun 11 Feb 18:54:43 CET
    2018
Host: Slave—3 ist ausgefallen! Datum: Sun 11 Feb 18:55:01 CET 2018
Host: Slave—3 wurde neu gestartet! Datum: Sun 11 Feb 18:55:01 CET
    2018
```

LISTING 4.5: Logdatei des automatischen Reset Skripts für Slaves

Die beiden vorgestellten Shell Skripte werden mithilfe eines Linux eigenen Mechanismus zum Start des Betriebssystems ausgeführt. Dazu benutzt man das Programm `cron` von Linux. `cron` verwaltet eine Tabelle in der wiederkehrende Ereignisse aufgelistet werden. Der Aufruf von `crontab -e` öffnet diese Tabelle und die Bearbeitung dieser Tabelle führt man mit einem Texteditor aus. Die Pfade und gewählten Optionen schreibt man ans Ende dieser Datei. Die hier vorgestellten Skripte sollen zum Start des Betriebssystems ausgeführt werden. Aus diesem Grund wird `@reboot /home/pi/autoReset.sh` in der `crontab` eingetragen. Dadurch führt das Betriebssystem das Skript `autoReset.sh` zum Start des Betriebssystems aus. Das Gleiche wird mit dem Skript zum Reset der Cluster-Knoten eingerichtet und sorgt für eine automatische Ausführung der Skripte.

Das Listing 4.5 zeigt die erstellte Logdatei des Reset Skript. Die wichtigsten Informationen sind darin enthalten. Durch diese Informationen lassen sich Schlüsse über den Cluster ziehen. Dieses Skript soll für einen Cluster aus 128 Knoten verwendet werden und ein automatisches Skript in Kombination mit einer Logdatei helfen bei der Administration von großen Clustern. Durch die Logdatei kann beobachtet werden, ob bei den Ausfällen statistische Häufungen auftreten. Dadurch können Aussagen zur Zuverlässigkeit der einzelnen Knoten getroffen werden. Die automatische Ausführung dieses Skripts sorgt für eine einfache Administration, da er ohne manuellen Eingriff ausgeführt wird. Durch diese beiden Funktionen ist eine Erleichterung beim Betrieb des Clusters entstanden, die zuverlässig die Funktion der Knoten überwacht und Neustarts protokolliert.

Konstruktion einer netzwerkgesteuerten Steckdosenleiste

Dieses Kapitel beschäftigt sich mit der Konstruktion einer netzwerkgesteuerten Steckdosenleiste. Dabei wird der Aufbau der interne Hardware erklärt. Außerdem geht dieses Kapitel auf die Funktionalität der Steckdosenleiste ein. Ein Webserver wird mithilfe von Flask konfiguriert und implementiert. Eine Erklärung der Thematik Webservices und eine konkrete Implementation zeigen die Möglichkeiten dieser Technologie auf.

5.1 Aufbau der netzwerkgesteuerten Steckdosenleiste

Für den Aufbau dieser Steckdosenleiste werden Bauteile aus dem Baumarkt verwendet. Eine handelsübliche Abzweigdose dient als Gehäuse für die komplette Hardware. Abzweigdosen dienen für gewöhnlich dazu mehrere Kabel zusammenzuführen, deshalb besitzen sie Durchführungen an denen Kabel eingeführt werden können. Sie ist geeignet um eine Steckdosenleiste zu bauen, da sie aus einem nicht leitenden Material besteht und der Deckel durch Schrauben geschlossen werden kann. Abbildung 5.1 zeigt eine Abzweigdose.

Verbaut werden handelsübliche Steckdosen. Dabei kommen unterputz Steckdosen zum Einsatz. Sie können in die Abzweigdose eingelassen werden. Dadurch schließen die Steckdosen bündig mit dem Gehäuse ab und verhindern ein versehentliches Berühren von leitenden Teilen. Das ist wichtig bei der Konstruktion einer Steckdosenleiste, da sichergestellt werden muss, dass bei der Verwendung der Steckdose kein Mensch zu schaden kommt. Beim Zusammenbau von elektrotechnischen Apparaturen muss besonders Wert auf die Sicherheit gelegt werden. Da hier 230 Volt Spannung fließen, muss man den Benutzer vor Verletzungen durch Strom schützen.

ABBILDUNG 5.1: Abzweigdose
Quelle: https://www.bauhaus.info/schalterdosen-deckel/spelsberg-feuchtraum-abzweigdose-abox/p/22682031?q=abzweigdose%20250 (abgerufen 02.03.2018)

Die beiden essentiellen Komponenten wurden bereits genannt. Um die Steckdosen mit Strom zu versorgen werden Kabel benötigt. Dazu werden Kabel mit einem Durchmesser von 1,5 mm^2 verwendet. Laut DIN VDE 0100-430 sind Kabel mit einem Querschnitt von 1,5 mm^2 bis zu einer Stromstärke von 16 A ausgelegt, der im Innenbereich durch Leitungsschutzschalter abgesichert ist [25]. Außerdem wird ein Kabel zum Anschließen der Steckdosenleiste an den Strom benötigt. Der Einbau eines Schalters dient zum Steuern der Stromzufuhr. Dieser Schalter dient zum Ein- und Ausschalten des Gesamtsystems.

Die internen Komponenten zum Steuern der Steckdosen bestehen aus einem Raspberry Pi 3, einer Relaiskarte, einem Netzteil für die Stromversorgung des Raspberry Pi und der Relaiskarte und diversen Jumperkabeln zum Anschließen des Raspberry Pi und der Relaiskarte. Dabei dient der Raspberry

Pi als zentrale Einheit zum Steuern der Steckdosen. Die Relaiskarte wird genutzt, um die Steckdosen zu schalten. Die Steuerung der Relaiskarte übernimmt ein Raspberry Pi über seine GPIO-Ports. Die Relais sorgen dafür, dass eine Spannung von 230 Volt geschaltet werden kann. Der Raspberry Pi ist selbst nicht in der Lage eine solche Spannung zu schalten.

Der Aufbau der Steckdosenleiste beginnt zunächst mit dem Einbau der Steckdosen in die Abzweigdose. Dabei werden die Durchführungen gebohrt und die unterputz Steckdosen eingesteckt. Anschließend werden sie mit dem Gehäuse verschraubt. Da hier unterputz Steckdosen verbaut sind befindet sich der Großteil der Steckdose innerhalb der Abzweigdose. Dadurch befinden sich die Anschlüsse zur Verkabelung innerhalb des Gehäuses. Die Verkablung innerhalb des Gehäuses wird von jeder Steckdose zu den Anschlussklemmen der Relaiskarte geführt. Um die Verkabelung zu organisieren werden die Kabel von den Steckdosen über die Innenseite des Gehäuses zu den Anschlussklemmen der Relaiskarte geführt. Dort werden die Kabel mit den Klemmen verbunden. Abbildung 5.2 zeigt den inneren Aufbau der Steckdosenleiste.

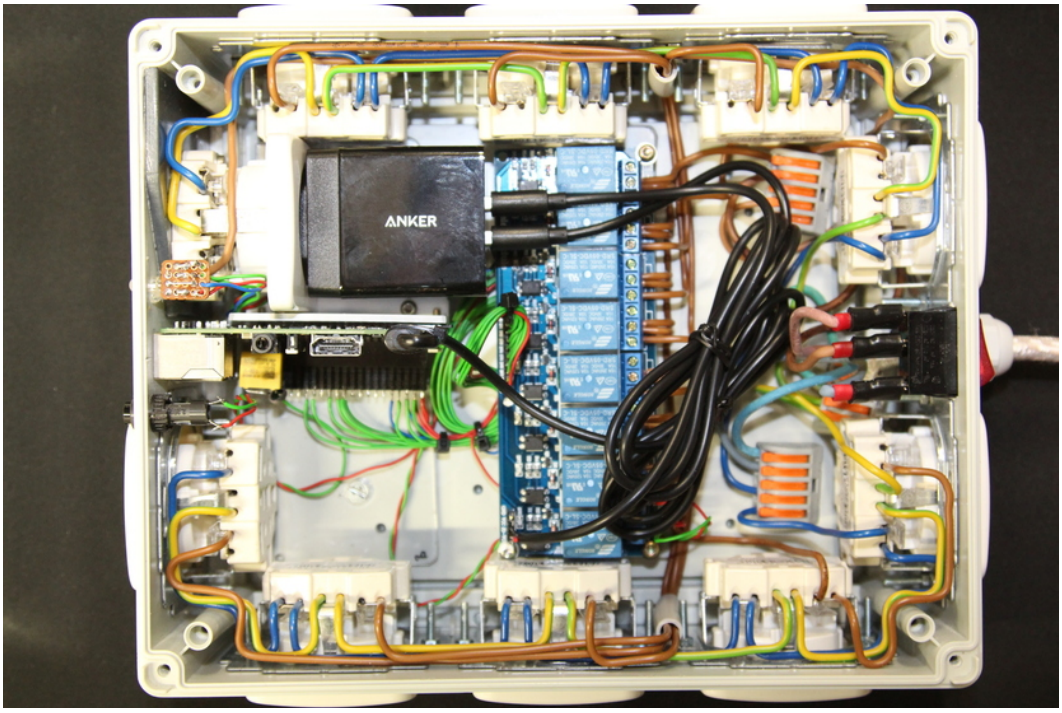


ABBILDUNG 5.2: Steckdosenleiste Innen

Einer der Steckdosen ist umgekehrt eingebaut. Diese Steckdose dient zur Stromversorgung für das Netzteil. Das Netzteil speist den Raspberry Pi und die Relaiskarte. Um sicherzustellen, dass keine elektrischen Kabel außerhalb des Gehäuses zur Gefährdung eines Benutzers führen, wird das entstehende Loch mit einer Kunststoffplatte verschlossen. Diese Platte ist mit dem Gehäuse verklebt und kann daher nicht nachträglich entfernt werden. Das trägt zur Sicherheit für den Benutzer bei.

An dieser umgekehrt angebrachten Steckdose befindet sich ein USB-Netzteil von Anker mit 24 Watt Leistung. Dieses Netzteil verfügt über zwei USB-Ports und speist den Raspberry Pi mit Strom. Der andere Port wird benutzt um die Relaiskarte mit Strom zu versorgen. Dabei wurde ein USB-Kabel abisoliert und die einzelnen Adern des Kabels mit dem Eingang der Relaiskarte verbunden. Dieser Anschluss ist wichtig, da die Relaiskarte eine externe Stromversorgung benötigt, um die Leistung zum Schalten der 230 Volt am Ausgang des Relais zu erhalten.

Auf einer Seite des Gehäuses ist ein Anschlusskabel eingebaut. Dieses Kabel versorgt die gesamten Komponenten der Steckdosenleiste mit Strom. Oberhalb des Kabels ist ein Schalter angebracht, der die Stromzufuhr regelt. Mit diesem Schalter kann die Steckdosenleiste und damit das Gesamtsystem an und ausgeschaltet werden. Abbildung 5.3 zeigt den Schalter und die Zuleitung der Steckdosenleiste.

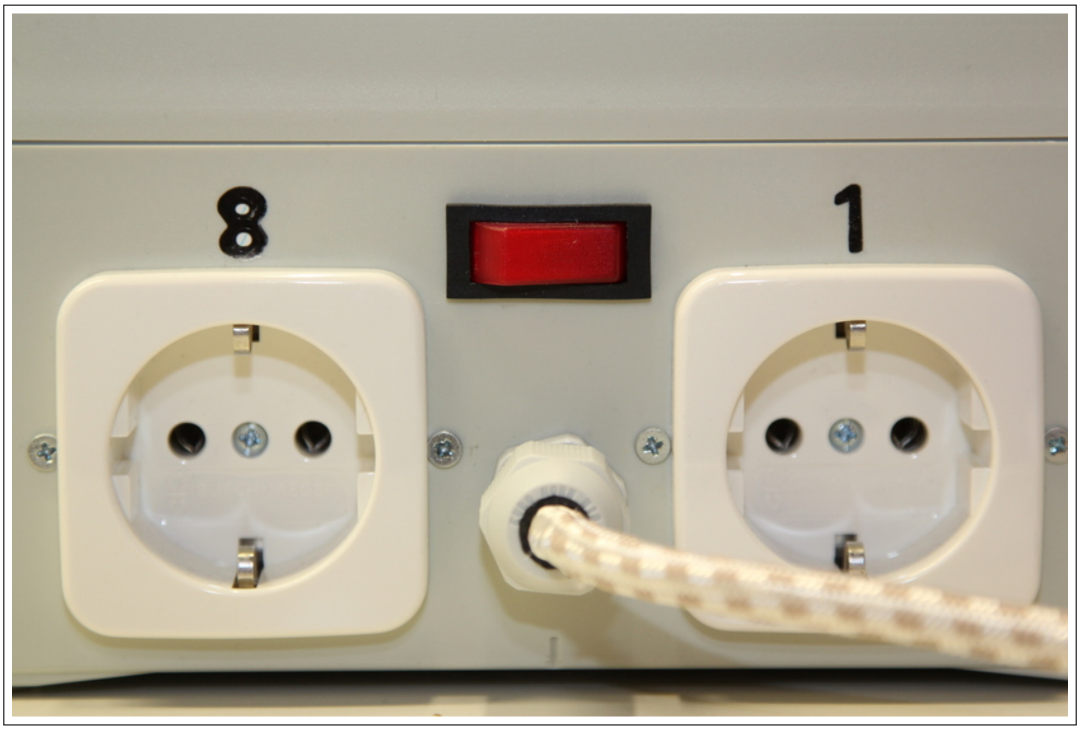


ABBILDUNG 5.3: Schalter zum Anschalten und Zuleitung

Auf dem Deckel der Steckdosenleiste sind Warnsymbole und Aufschriften aufgebracht. Da nach DIN VDE 0100-410 der Grenzwert für Berührungsspannungen bei 50 Volt Wechselstrom liegt, werden die Warnsymbole und Aufschriften verwendet [25]. Sie sollen klar kennzeichnen, dass intern eine lebensgefährliche Spannung fließt. Abbildung 5.4 zeigt den Deckel der Steckdosenleiste mit den Warnsymbolen und Aufschriften.

Der Raspberry Pi und die Relaiskarte sind auf Polycarbonat Platten aufgebracht. Dieses Material ist einfach zu zuschneiden und sehr widerstandsfähig. Außerdem ist dieses Material nicht leitfähig. Diese Merkmale

ABBILDUNG 5.4: Bild des Deckels der Steckdosenleiste

machen es zu einem geeigneten Material für die Aufnahme dieser Komponenten. Die Relaiskarte ist zentral auf dem Boden des Gehäuses moniert. Dadurch können alle Kabel der Steckdosen zu dieser zentral angebrachten Komponente gelegt werden.

Der Raspberry Pi ist hochkant angebracht. Durch diese Anordnung kann Platz gespart werden. Außerdem wird ein Loch durch das Gehäuse der Abzweigdose gesägt, um die Anschlüsse des Raspberry Pi erreichbar zu machen. Somit sind der Ethernet Anschluss und die USB Anschlüsse des Raspberry Pi verwendbar. Abbildung 5.5 zeigt die Front der Steckdosenleiste. Dort ist eine eingelassene Plastikabdeckung angebracht, da die dort eingebaute Steckdose umgekehrt eingebaut ist. Dadurch soll eine versehentliche Berührung durch den Benutzer unterbunden werden.

Die Anschlüsse auf den GPIO-Ports des Raspberry Pi sind auf einer Buchsenleiste aufgelötet. Dadurch kann eine ideale elektrische Verbindung sichergestellt werden und die Buchsenleiste kann wie ein Stecker auf die GPIO-Ports des Raspberry Pi gesteckt werden. Die Anschlusssicherheit wird erhöht, da die Pinbelegung nicht mehr beachtet werden muss. Der Stecker Mechanismus sorgt dafür, dass keine Vertauschung der Pinbelegung stattfindet.

Neben dem Raspberry Pi ist ein Taster angebracht, der mit den GPIO-Ports des Raspberry Pi angeschlossen ist. Dieser Taster wird verwendet, um eine Möglichkeit zu haben die Steckdosen zu schalten ohne eine Verbindung zum

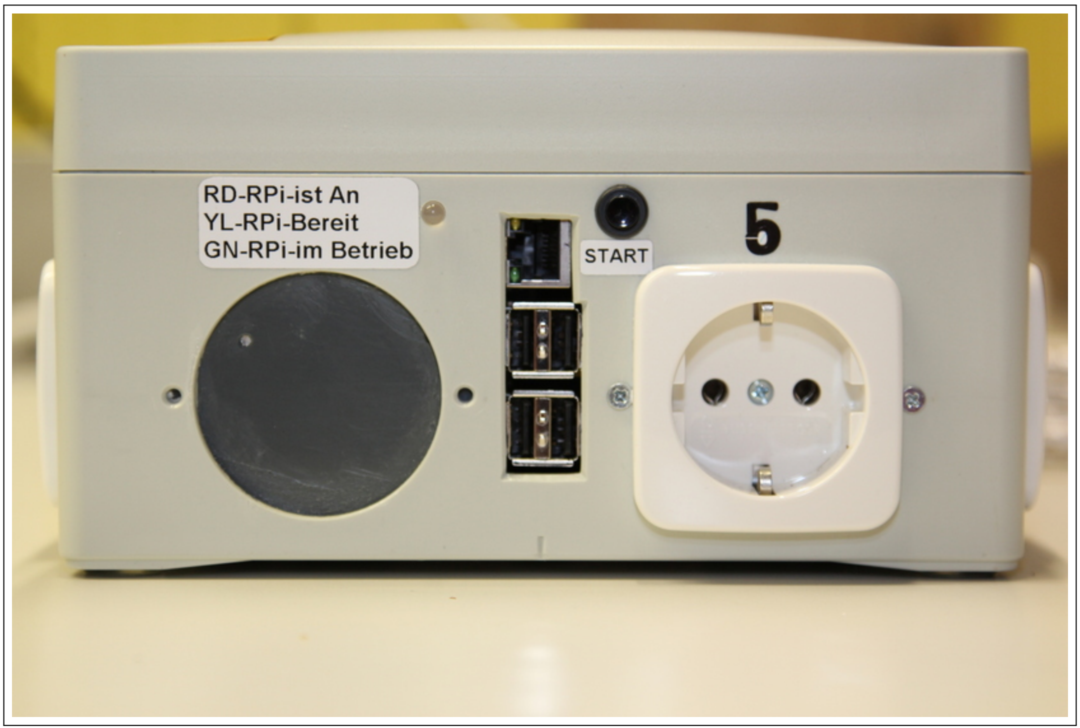


ABBILDUNG 5.5: Front der Steckdosenleiste

Raspberry Pi zu haben. Betätigt man diesen Taster schaltet der Raspberry Pi alle Steckdosen nacheinander mit einem Versatz von einer Sekunde an. Betätigt man den Taster danach nochmals werden alle Steckdosen nacheinander mit einer Sekunde Versatz abgeschaltet. Listing 5.4 zeigt das Skript, das diese Funktion realisiert.

Weiter ist eine Leuchtdiode neben dem Raspberry Pi angebracht. Diese LED dient dazu dem Anwender Informationen über den Status der Steckdosenleiste zu geben. Beim Hochfahren des Raspberry Pi ist diese LED rot. Während des Hochfahrens kann der Raspberry Pi keine Befehle ausführen, daher signalisiert die rote Farbe der LED, dass noch keine Kommandos abgegeben werden können. Nachdem der Raspberry Pi hochgefahren ist, signalisiert die LED durch seine gelbe Farbe die Betriebsbereitschaft der Steckdosenleiste. Nun können die Steckdosen angeschaltet werden. Wenn die Steckdosen angeschaltet sind, leuchtet die Leuchtdiode grün. Dabei ist die LED mit dem achten Relais der Relaiskarte verbunden und ändert seine Farbe zu grün, wenn dieses Relais geschaltet wird.

Tabelle 5.1 zeigt eine Auflistung der Teile mit den Kosten der einzelnen Bauteile. Insgesamt belaufen sich die Kosten auf 160 €. Für diesen Preis erhält man eine Steckdosenleiste, die frei programmierbar ist und an die Bedürfnisse des Benutzers angepasst werden kann. Anhang B.1 zeigt einen detaillierten Schaltplan der Steckdosenleiste. Darin ist die genaue Verschaltung der einzelnen Komponenten und eine Liste der verwendeten GPIO-Ports mit Legende gezeigt.

TABELLE 5.1: Liste der Bauteile und Kosten

Bauteil	**Stückzahl**	**Preis**
Raspberry Pi	1	35 €
Micro-SD Karte	1	7 €
USB-Kabel	2	5 €
Abzweigdose	1	25 €
Steckdosen (unterputz)	10	30 €
Anschlusskabel 3x1,5 10m	2	10 €
Zuleitung	1	7 €
Verbindungsklemmen	2	4 €
Kleinteile Montage	-	10 €
Taster	1	2 €
Schalter an/aus	1	2 €
Status-LED	1	1 €
Relaiskarte	1	8 €
Buchsenleiste 40 Pin	1	2 €
USB-Stromversorgung	1	12 €
Summe		160 €

Im weiteren Verlauf des Kapitels wird auf Implementationsdetails eingegangen. Dabei wird gezeigt, wie man einen Taster zum an und ausschalten der Steckdosenleiste implementiert und einen Webserver einrichtet, der in der Lage ist die Steckdosen über eine Weboberfläche zu schalten. Ein Webservice zum Schalten der Steckdosen wird implementiert.

5.2 Implementationsbeispiele in Python

Zum Verständnis der Skripte hilft Tabelle 5.2. In dieser Tabelle sind die Pinbelegungen der GPIO-Pins aufgelistet. Dabei wird in den nachfolgenden Python Skripten Verwendung von der GPIO-Port Nummerierung gemacht. Die Pinbelegung ist in Tabelle 5.2 angegeben, da sie hardwaremäßig auf den Pins verkabelt wurden. Anhang B.1 zeigt den Plan der Steckdosenleiste und bietet einen detaillierten Einblick in den Aufbau.

TABELLE 5.2: Tabelle der GPIO-Ports und Pinbelegungen

GPIO-Port	**Pin**	**Belegung**
GPIO 2	Pin 3	Relais 1
GPIO 3	Pin 5	Relais 2
GPIO 4	Pin 7	Relais 3
GPIO 5	Pin 29	Relais 4
GPIO 6	Pin 31	Relais 5
GPIO 7	Pin 26	Relais 6
GPIO 8	Pin 24	Relais 7
GPIO 9	Pin 21	Relais 8
GPIO 25	Pin 22	Status LED
GPIO 26	Pin 37	Taster

Die Listings 5.1 und 5.2 zeigen Python Skripte, die verdeutlichen sollen, wie man die Steckdosen anschalten kann. Sie dienen exemplarisch dazu die wichtigsten Funktionen der **RPi.GPIO** Bibliothek zu zeigen.

Zeile 1 des Listings 5.1 zeigt wie man die Bibliothek in Python einbindet. Durch den Aufruf `as GPIO` wird dem Python Interpreter mitgeteilt, dass die Funktionen dieser Bibliothek mit dem Wort `GPIO` getrennt durch einen Punkt aufgerufen werden können. Der Aufruf von `GPIO.setmode(GPIO.BCM)` definiert die Zählweise der GPIO-Ports. Wird hier der Parameter `GPIO.BOARD` statt `GPIO.BCM` übergeben ist die Nummerierung anhand der Pin Nummern und nicht der GPIO-Port Nummern definiert. Zeile 9 schaltet die Warnstufe beim Ansprechen der GPIO-Ports niedriger. Wird diese Funktion mit dem Parameter `True` aufgerufen, werden beim Ansprechen der GPIO-Ports Warnungen auf der Kommandozeile ausgegeben. Der Parameter `False` unterdrückt diese Warnungen. Das Unterdrücken dieser Warnmeldungen ist deshalb implementiert worden, da die Ausgabe der Warnungen auf der Kommandozeile störend ist und im späteren Verlauf der Implementation zu viele Ausgaben produzieren.

```
1  import RPi.GPIO as GPIO
2
3  import time
4
5  # Setzt das Schema auf GPIO Nummern
6  GPIO.setmode(GPIO.BCM)
7
8  # Schaltet Warnungen auf der Kommandozeile aus
9  GPIO.setwarnings(False)
10
11 # Definiert alle Ports als Ausgang und
12 # Setzt sie auf HIGH
13 for port in range(2, 10):
14   GPIO.setup(port, GPIO.OUT)
15   GPIO.output(port, GPIO.HIGH)
16
17 print("Ports wurden gesetzt")
18
19 # Schaltet die Ports an mit 5 Sekunden Unterbrechung zwischen jedem
       Port
20 for port in range(2, 10):
21   GPIO.output(port, GPIO.LOW)
22   print ('Port ', port, ' gestartet')
23   time.sleep(0.5)
```

LISTING 5.1: Beispiel Skript zum Starten der Steckdosen

In Zeile 13 wird eine **for-each** Schleife benutzt um in der darauf folgenden Zeile mit der Funktion `GPIO.setup(port, GPIO.OUT)` die Ports 2 bis 9 als Output zu definieren. Das bedeutet, dass diese Ports Signale aussenden. In Zeile 15 werden diese Ports auf logisch `HIGH` gesetzt. Die Relais sind so konfiguriert, dass sie mit einem logischen `LOW` angeschaltet werden. Zeile 21 zeigt das Setzen der Ports auf `LOW`, womit die Relais angeschaltet werden. Listing 5.2 zeigt den einzigen Unterschied, der zwischen dem Anschalten und dem Ausschalten besteht. Dort werden die Ports in einer Schleife auf logisch `HIGH` gesetzt, was dazu führt, dass die Relais ausgeschaltet werden.

```
for port in range(2, 10):
  GPIO.setup(port, GPIO.HIGH)
```

LISTING 5.2: Auszug Beispiel Skript zum Ausschalten der Steckdosen

Das Skript 5.3 dient dazu die Status LED der Steckdosenleiste zu starten. Dabei wird der GPIO-Port 25 auf logisch `HIGH` gesetzt. Dieses Skript wird im Verzeichnis `/etc/init.d/` gespeichert und wird während des Hochfahrens ausgeführt. Das Kommando sudo chmod +x <SKRIPTNAME> macht das Skript ausführbar. Durch Eingabe des Kommandos sudo update-rc.d <SKRIPTNAME> defaults wird das Betriebssystem aufgefordert dieses Skript nach dem Hochfahren zu starten. Ist der Raspberry Pi hochgefahren sendet er ein `HIGH` Signal auf GPIO-Port 25, was dazu führt, dass die LED gelb leuchtet. Das signalisiert, dass der Raspberry Pi hochgefahren ist und die Steckdosenleiste bereit zur Verwendung ist. Diese Farbe setzt sich aus der Mischung der Farben rot und grün zusammen. Die verbaute LED ist eine RG-LED, welche die Farben rot und grün aussenden kann. Ist der Raspberry Pi nicht hochgefahren leuchtet nur der rote Anteil der RG-LED. Werden die Relais geschaltet führt das Schalten des achten Relais zu einem Eingangssignal für die RG-LED, welche sich dadurch grün färbt und den Betrieb der Steckdosen signalisiert.

```
import RPi.GPIO as GPIO


# Setzt das Schema auf GPIO Nummern
GPIO.setmode(GPIO.BCM)

# Schaltet Warnungen auf der Kommandozeile aus
GPIO.setwarnings(False)

GPIO.setup(25, GPIO.OUT)
GPIO.output(25, GPIO.HIGH)
```

LISTING 5.3: Steuern der Status LED

Das Listing 5.4 zeigt ein Skript zum Schalten der Relais mithilfe eines Tasters. Dabei ist der GPIO-Port 26 als Eingang definiert (Zeile 2). Bei der Definition des Eingangs wird mit dem Parameter `pull_up_down = GPIO.PUD_DOWN` ein sogenannter **Pull-Down Widerstand** definiert. Ein Pull-Down Widerstand zieht dabei die Spannung am Eingang herunter auf die Masse [26]. Wird ein Signal mit dem Pegel `HIGH` an den Eingang gelegt, zieht der interne Pull-Down Widerstand die Spannung gegen die Masse und kann ausgewertet werden.

Zeile 6 zeigt die Definition einer Funktion. Die Funktion definiert einen **Interrupt**. Ein Interrupt ist eine Routine, die die Abarbeitung eines Prozesses unterbricht und dafür eine Routine ausführt. Hier ist dieser Interrupt in Zeile 35 angemeldet. Angemeldet bedeutet, dass die Routine mit dem Aufruf der Funktion `GPIO.add_event_detect(...)` definiert wird [27]. Dabei ist der erste Parameter der Funktion 26. Dieser Parameter steht für den verwendeten GPIO-Port. Der zweite Parameter `GPIO.RISING` zeigt an, dass ein Event ausgeführt wird, wenn am GPIO-Port eine **steigende Flanke** erkannt wird. Eine steigende Flanke bedeutet, dass ein Signal durch den Taster einen `HIGH` Pegel an den Eingang setzt. Der dritte Parameter `callback=isr26` definiert die Funktion `isr26()` als **Callback-Funktion**.

Eine Callback-Funktion ist eine Funktion, die asynchron aufgerufen wird. Dadurch kann sie bei erkennen eines Events ausgeführt werden. Der vierte Parameter `bouncetime=500` beschreibt die Zeit die der Eingang nach erkennen eines Events pausiert bis er wieder Signale empfängt. Diese Zeit ist angegeben, um den verwendeten Taster zu **entprellen**.

Entprellen heißt, dass die mechanischen Schwingungen des Taster am Eingang zu mehreren Schaltvorgängen führen. Dadurch das hier eine Entprellzeit von 500 ms gewählt ist kann sichergestellt werden, dass ein Eingangssignal nicht mehrfach ankommt.

```
1  # Port 26 wird als Input mit Pull Down Widerstand gesetzt
2  GPIO.setup(26, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
3
4
5  # Interrupt Routine zum Schalten der Relais
6  def isr26(channel):
7
8    button_Push_Flag = 0
9
10   if GPIO.input(9) == 0:
11
12     button_Push_Flag = 1
13
14     print("\nPower OFF!\n")
15     for port in range(2, 10):
16                         GPIO.output(port, GPIO.HIGH)
17                         time.sleep(1)
18
19   if GPIO.input(9) == 1 and button_Push_Flag == 0:
20
21     print("\nPower ON!\n")
22     for port in range(2, 10):
23       GPIO.output(port, GPIO.LOW)
24       time.sleep(1)
25
26   # Der Interrupt wird geloescht und aschliessend wieder angemeldet
27   # Muss gemacht werden weil der Interrupt sonst doppelt
      ausgefuehrt wird
28   GPIO.remove_event_detect(26)
29   GPIO.add_event_detect(26, GPIO.RISING, callback=isr26, bouncetime
      =500)
30
31
32 # Definiert den Interrupt
33 GPIO.add_event_detect(26, GPIO.RISING, callback=isr26, bouncetime
      =500)
```

LISTING 5.4: Python Skript zum Anschalten per Taster

Die in Zeile 6 definierte Funktion realisiert das Auswerten des Eingangssignals. Dabei dient die boolesche Variable `button_Push_Flag` als Indikator über den Zustand des Tasters. Da auf dem Taster sowohl die An- als auch die Ausschaltfunktion liegen, wird diese Variable verwendet um zu ermitteln, ob der Taster zum An oder Ausschalten gedrückt wurde. Die if-Abfrage `if GPIO.input(9) == 0:` dient zum Ermitteln, ob das Relais 8 an ist oder nicht. Ist der Wert 0 so sind die Relais an und werden in der Schleife aus Zeile 15 ausgeschaltet. Ist der Wert 1 so sind die Relais aus und werden in Zeile 24 angeschaltet. Zeile 17 und 26 führen zu einer Verzögerung von einer Sekunde.

Das wird implementiert, da diese Steckdosenleiste die Knoten des Clusters anschalten soll. Dadurch, dass die Steckdosen nacheinander geschaltet werden soll die Last für das Labornetz reduziert werden und so ein Auslösen der Sicherung vermeiden.

In Zeile 30 der Interrupt Routine wird der Interrupt wieder abgemeldet und in der darauf folgenden Zeile erneut angemeldet. Dieser Vorgang ist implementiert, da der Interrupt sonst doppelt ausgeführt wird. Dadurch würde beim Einschalten der Steckdosen die erneute Ausführung des Interrupt die Steckdosen wieder ausschalten. Diese Maßnahme verhindert dieses Verhalten.

Damit der Taster nach hochfahren der Steckdose verwendet werden kann, wird das Skript zum Start ausgeführt. Das Programm `supervisorctl` kann diese Funktionalität anbieten. Dieses Programm hilft Programme als Daemon auszuführen. Das bedeutet, dass das Programm im Hintergrund abläuft und einen Dienst anbietet. In Betriebssystemen gibt es viele solcher Daemons, die im Hintergrund ablaufen und Dienste ausführen, ohne das der Benutzer sie explizit starten muss. `supervisorctl` führt den Daemon aus und steuert ihn. Um `supervisorctl` verwenden zu können, muss das Programm vorher installiert werden. Das Kommando `sudo apt-get -y install supervisorctl` installiert das Programm. Im Verzeichnis `/etc/supervisor/conf.d/` wird eine Konfigurationsdatei erstellt. Für das Programm aus 5.4 heißt das Skript `KASKADEd.conf`.

```
[program:KASKADE.py]
directory=/home/pi/pythonskripte/
environment=ENV_VARIABLE=kaskade,OTHER_ENV_VARIABLE=cascade
command=python3 KASKADE.py
autostart=true
autorestart=true
```

LISTING 5.5: Konfigurationsdatei für Taster Daemon

Listing 5.5 zeigt die Konfigurationsdatei, die das Programm zum Daemon macht. Darin wird der Pfad zum Programm und der Programmname mit dem Aufruf von `python3` zum Ausführen des Programms angegeben. Die Option `autostart=true` führt das Programm zum Start aus. Auch wird eine Umgebungsvariable für die Kommandozeile angegeben. Nach dem Erstellen dieser Konfigurationsdatei übernimmt man die Konfiguration mit dem Kommando `sudo supervisorctl update`. Mit dem Kommando `sudo supervisorctl restart` wird das Programm neu gestartet und der Daemon wird ausgeführt. Dadurch kann der Taster nach Hochfahren des Systems verwendet werden.

Beim Aufbau der Steckdose ist beim Testen der Funktion ein Fehler aufgefallen. Wenn im Betrieb ein Verbraucher in eine Steckdose eingeführt wird, führt das zum Ausschalten der Steckdosen. Das ist dadurch zu erklären, dass beim Einstecken ein Funke entsteht, der ein hochfrequentes Signal aussendet. Dieses Signal wird über die ungeschirmte Leitung des Tasters an den Eingang des Rapsberry Pi geleitet und führt zum Ausführen des Interrupts. Diesem Problem wurde entgegen gewirkt, indem am Eingang des Raspberry Pi ein Kondensator angebracht wurde. Dieser Kondensator hat eine Kapazität von 100 nF. Er dient zur Abnahme dieses hochfrequenten Störsignals. Das Signal lädt den Kondensator auf und dadurch, dass der Kondensator gegen die Masse auf Pin 39 geschaltet ist, kann er diese Ladung gegen die Masse des Raspberry Pi abführen. Durch den Einsatz des Kondenstors können Stecker angeschlossen werden, ohne dass diese zu einem fehlerhaften Schalten des Tasters führen.

Durch das Skript 5.4 kann die Steckdosenleiste bereits in diesem Stadium wie eine handelsübliche Steckdosenleiste verwendet werden. Ein Benutzer kann die Steckdosenleiste anschalten und über den Taster alle Steckdosen anschalten. Im nächsten Abschnitt wird beschrieben, wie man einen Webserver konfiguriert und implementiert. Dieser Webserver soll dazu verwendet werden die Steckdosen über eine Weboberfläche zu schalten.

5.3 Webserver zum Steuern der Steckdosen

Dieser Abschnitt beschäftigt sich mit dem Installieren und Konfigurieren eines Webservers auf dem Raspberry Pi. Dabei kommen unterschiedliche Möglichkeiten wie Apache oder nginx in Betracht. Hier wird jedoch der Webserver von **Flask** verwendet. Flask ist ein Python Framework zum Erstellen von Webanwendungen in Python. Die Entscheidung für Flask wurde getroffen, da die Verwendung von Python eine Ansteuerung der GPIO-Ports vereinfacht. Es wird gezeigt, wie sich die Bibliothek RPi.GPIO in Flask Webanwendungen einbinden lässt. Mit diesem Framework lassen sich dynamische Webseiten mithilfe von Python Skripten implementieren.

Die Installation von Flask auf dem Raspberry Pi ist sehr einfach. Dazu wird das Python Paketverwaltungsprogramm **pip** (pip installs packages) verwendet. Dieses Programm verwaltet Pakete in Python und dient der Installation von Erweiterungen für Python. Das kann unter Linux mit dem Kommando `sudo pip install Flask` ausgeführt werden. Dadurch wird Flask systemweit installiert und kann durch Einbinden der Module in jedem Python Skript verwendet werden [28].

Mit Unterstützung von Flask lassen sich leichtgewichtige Webserver Anwendungen erstellen. Außerdem lassen sich dynamische Anteile von HTML Webseiten mit Python erstellen. Bevor auf Quellcode Beispiele eingegangen wird, soll zuerst die Verzeichnisstruktur der hier präsentierten Anwendung erklärt werden. Dazu wurde die Anwendung mithilfe von Python Modularisiert. Das bringt den Vorteil, dass die einzelnen Dateien besser geordnet sind [29].

Die hier gezeigte Webanwendung hat folgende Verzeichnisstruktur:

- `\Anwendungsverzeichnis` - ein Ordner in dem alle Dateien und weiteren Unterverzeichnisse sind
- `\Anwendungsverzeichnis\static` - hier befinden sich alle statischen Dateien wie Bilder oder Stylesheets
- `\Anwendungsverzeichnis\templates` - in diesem Ordner sind alle HTML Seiten

Das Listing 5.6 zeigt das Start Skript der Webanwendung. Dabei wird das Modul `webserver` geladen. Anschließend wird der Webserver mit der IP-Adresse 192.168.178.220 und dem Port 80 gestartet. Der Server startet im Debug Modus, um Informationen über den Status auf der Kommandozeile auszugeben. Dieses Modul ist das Anwendungsverzeichnis, indem sich alle relevanten Dateien der Webanwendung befinden. Das beinhaltet auch die Dateien die in Listing 5.7 und 5.9 gezeigt werden. Dabei stellen sie den Hauptteil der Anwendung dar. Die Datei aus Listing 5.9 ist die HTML Datei, die für die Darstellung im Webbrowser dient und sich im Unterverzeichnis `static` befindet. Die Quellcode Datei aus 5.7 zeigt die dynamischen Anteile der statischen HTML-Seite.

```python
#!flask/bin/python

from webserver import webserver

webserver.run(debug=True, host='192.168.178.220',port=80)
```

LISTING 5.6: Starten der Webserver Anwendung

Listing 5.7 zeigt ein Python Skript, welches die Funktion der Webseite implementiert. Dabei sieht man in Zeile 1, dass das Modul `webserver` importiert wird. Im vorangegangenen Absatz wurde bereits erklärt, wie die Verzeichnisstruktur der Anwendung aussieht. Zeile 2 zeigt den Import der Funktion `render_template` aus dem `flask` Modul. Diese Funktion wird von Flask bereitgestellt, um HTML Seiten zu rendern. Dadurch können HTML Seiten im Browser angezeigt werden. Zeile 3 und 4 zeigen den Import der `RPi.GPIO` und `time` Bibliothek. Diese werden zum Ansteuern der GPIO-Ports benötigt und um die Funktion `time.sleep()` zu verwenden.

In Zeile 10 wird ein assoziatives Array mit dem Namen `dosen` definiert. Dabei werden Schlüssel-Wert Paare definiert. Die Schlüssel sind die Nummern der GPIO-Ports. In Tabelle 5.2 sieht man die verwendeten GPIO-Ports. Jedem dieser Ports ist ein weiteres assoziatives Array zugewiesen. Darin sind für jeden Port ein Name und ein Zustand definiert. So ist in Zeile 11 der Wert `Steckdose Nr. 1` und der Zustand `GPIO.HIGH` definiert. Der Wert wird im späteren Verlauf verwendet, um den Namen der Steckdose auf der Webseite anzuzeigen. Der Status dient dem Anzeigen des Zustands der Steckdosen. Dadurch können mit diesem Array alle Informationen, die notwendig sind um sie auf der Webseite anzuzeigen, gespeichert werden.

```
1  from webserver import webserver
2  from flask import render_template
3  import RPi.GPIO as GPIO
4  import time
5
6  GPIO.setmode(GPIO.BCM)
7  GPIO.setwarnings(False)
8
9  # Global werden die Ports und dazugehoerige Werte gesetzt
10 dosen = {
11   2 : {’name’ : ’Steckdose Nr. 1’, ’status’ : GPIO.HIGH},
12   3 : {’name’ : ’Steckdose Nr. 2’, ’status’ : GPIO.HIGH},
13   4 : {’name’ : ’Steckdose Nr. 3’, ’status’ : GPIO.HIGH},
14   5 : {’name’ : ’Steckdose Nr. 4’, ’status’ : GPIO.HIGH},
15   6 : {’name’ : ’Steckdose Nr. 5’, ’status’ : GPIO.HIGH},
16   7 : {’name’ : ’Steckdose Nr. 6’, ’status’ : GPIO.HIGH},
17   8 : {’name’ : ’Steckdose Nr. 7’, ’status’ : GPIO.HIGH},
18   9 : {’name’ : ’Steckdose Nr. 8’, ’status’ : GPIO.HIGH}}
19
20 # In Schleife werden jetzt die Dosen gesetzt
21 for dose in dosen:
22   GPIO.setup(dose, GPIO.OUT)
23   GPIO.output(dose, GPIO.HIGH)
24
25 # index Datei
26 @webserver.route(’/’)
27 @webserver.route(’/index’)
28 def index():
29
30   isOn = False
31
32   # liest den Status aller Ports aus
33   for dose in dosen:
34     dosen[dose][’status’] = GPIO.input(dose)
35
36   # Der Status der Dosen wird in Dictionary gespeichert und
37   # an HTML weitergereicht
38   dynamicData = {
39   ’buttonState’ : isOn,
40   ’dosen’ : dosen
41   }
42
43   return render_template("index.html", **dynamicData)
```

LISTING 5.7: Ausschnitt aus Python Skript für dynamische Webseite

Der Vorteil dieser Datenstruktur liegt auch darin, dass man sie mithilfe einer Schleife manipulieren kann. Aus diesem Grund wird in Zeile 21 eine Schleife verwendet, um die Steckdosen auf `HIGH` zu setzen. Das bedeutet, das die Steckdosen Initial aus sind. Das wird erreicht, indem auf den Schlüssel des assoziativen Arrays zugegriffen wird. Diese stellen die GPIO-Ports dar, die wiederum mit den Relais verbunden sind. Somit sind die Steckdosen definiert und initialisiert und können im weiteren Verlauf des Programms verwendet werden.

Zeile 28 definiert eine Funktion namens `index()`. Diese Funktion stellt die Index Datei der Webanwendung bereit. Oberhalb dieser Funktion sieht man zwei Zeilen, die mit einem `@` beginnen. Diese Zeilen werden als Dekorator bezeichnen und signalisieren dem Flask Framework, dass es sich hier um eine Webanwendung handelt. Das ist das Einzige, das die Funktion zusätzlich markiert, um zu signalisieren, dass es sich um eine Webanwendung handelt. Die Parameter `/` und `/index` in der Funktion `route()` zeigen das Ziel der Webanwendung an. Da es sich hier um die Index Datei der Webanwendung handelt, sind hier nur die beiden Zeichen angegeben.

In Zeile 33 wird der aktuelle Status der Steckdosen ausgelesen und in das assoziative Array unter dem Schlüssel `status` gespeichert. Eine Schleife führt dies für alle Steckdosen durch. In Zeile 38 wird im assoziativen Array `dynamicData` der Status des HTML Buttons und das assoziative Array `dosen` gespeichert. Dieses assoziative Array zeigt den Status der Steckdosen auf der HTML Seite an. Um das zu bewerkstelligen, wird die index.html Datei zusammen mit dem assoziativen Array an die Flask Funktion `render_template()` übergeben. Diese Funktion befindet sich in der Rückgabe Anweisung der Funktion `index()`. Durch die Rückgabe mit der Funktion `render_template()` wird die HTML Datei zusammen mit den dynamischen Teilen der Python Anwendung gerendert. Dadurch kann man diese Seite auf dem Browser anzeigen, indem man die IP-Adresse des Webservers im Browser eingibt. Listing 5.9 zeigt die HTML Seite **index.html**.

Listing 5.8 zeigt eine Funktion, die den Wert für einen Button übergeben bekommt. Die if-Abfrage prüft, ob der Wert des Buttons auf „on “ steht. Wenn das der Fall ist, wird die Variable `response` mit dem Wert „`Cluster angeschaltet`“ befüllt. Anschließend werden alle Steckdosen angeschaltet. Ist das nicht der Fall wird die Variable `response` mit dem Wert „`Cluster ausgeschaltet`“ befüllt. Anschließend werden alle Steckdosen ausgeschaltet. Zeile 26 speichert den aktuellen Status der Steckdosen und Zeile 29 speichert den Status der Steckdosen, den Status des Buttons und der Antwort in der Variablen `response` im Array `dynamicData`.

```python
@webserver.route('/index/<button>')
def switchCluster(button):

  # Wenn der Button im HTML Dokument
  # gedrueckt wurde wird sein state auf
  # True gesetzt und die Steckdosen
  # werden mit einem Delay von 1 Sekunde
  # angeschaltet
  # else werden sie abgeschaltet
  if button == "on":

    isOn = True
    response = "Cluster angeschaltet"
    for dose in dosen:
      GPIO.output(dose, GPIO.LOW)
      time.sleep(1)
  else:

    isOn = False
    response = "Cluster ausgeschaltet"
    for dose in dosen:
      GPIO.output(dose, GPIO.HIGH)
      time.sleep(1)

  # Der Status der Steckdosen wird im Dictionary gespeichert
  for dose in dosen:
    dosen[dose]['status'] = GPIO.input(dose)

  dynamicData = {
  'dosen' : dosen,
  'buttonState' : isOn,
  'response':response
  }


  return render_template('index.html', **dynamicData)
```

LISTING 5.8: Ausschnitt aus Python Skript zum Auslesen des Button

Die HTML Datei aus 5.9 zeigt die Darstellung der Webseite. Dabei wird ein Stylesheet verwendet, auf welches nicht weiter eingegangen wird. Es sorgt nur für eine Trennung zwischen den Darstellungsattributen und der HTML-Seite. Dadurch kann der HTML-Code kürzer gehalten werden. Interessanter ist die Darstellung des dynamischen Python Skripts auf der Webseite. Die Anweisungen, die in Python geschrieben sind, werden von einer geschweiften Klammer und einem Prozentzeichen umschlossen. So zeigt Zeile 8 eine Schleife, die den Namen der Steckdosen aus dem assoziativen Array `dynamicData` auf der Webseite anzeigt. Anders als in Python muss hier die for-Anweisung mit einem `{% endfor %}` beendet werden. Genauso wie jede if-Anweisung durch ein `{% endif %}` beendet werden muss. Zeile 19 und 22 definieren einen Button, um die Relais an und wieder auszuschalten. Dabei wird die Funktion aus Listing 5.8 aufgerufen.

```html
<html>
  <head>
    <link rel="stylesheet" href="/static/stylesheet.css">
    <title>Stem Webserver Power Management</title>
  </head>
  <body>
      <h1>Stem Power Management</h1>
      {% for dose in dosen %}
      <p>Die {{ dosen[dose].name}}
      {% if dosen[dose].status == true %}
  ist momentan ausgeschaltet
      {% else %}
  ist momentan angeschaltet
      {% endif %}
      </p>
      {% endfor %}

      {% if buttonState == False %}
      <a href="/on"> <button> Cluster anschalten</button></a>
      <br></br>
      {% else %}
      <a href="/off"> <button>Cluster ausschalten</button></a>
      <br></br>
      {% endif %}
      {% if response %}
      {{ response }}
      {% endif %}
  </body>
</html>
```

LISTING 5.9: Index Seite der Anwendung

Der Aufruf der Index Seite führt zur Darstellung der Webseite. Abbildung 5.6 zeigt die Darstellung im Browser. Durch betätigen des Buttons auf der Webseite lassen sich die Steckdosen der Reihe nach an und ausschalten. Unterhalb des Buttons wird eine Nachricht über die Aktion angezeigt. Auf der Webseite wird der Status für jede Steckdose angezeigt. Dabei korrespondieren die Nummern auf der Webseite mit der Nummerierung auf dem Gehäuse der Steckdosenleiste.

Dieser Webserver startet nach dem Hochfahren automatisch. Dazu verwendet `supervisorctl` eine Konfigurationsdatei, die den automatischen Start definiert. Dabei ist die Konfigurationsdatei bis auf den Pfad und den Programmnamen identisch zur Konfiguration aus Listing 5.5. Durch das Neustarten von `supervisorctl` wird der Webserver automatisch gestartet.

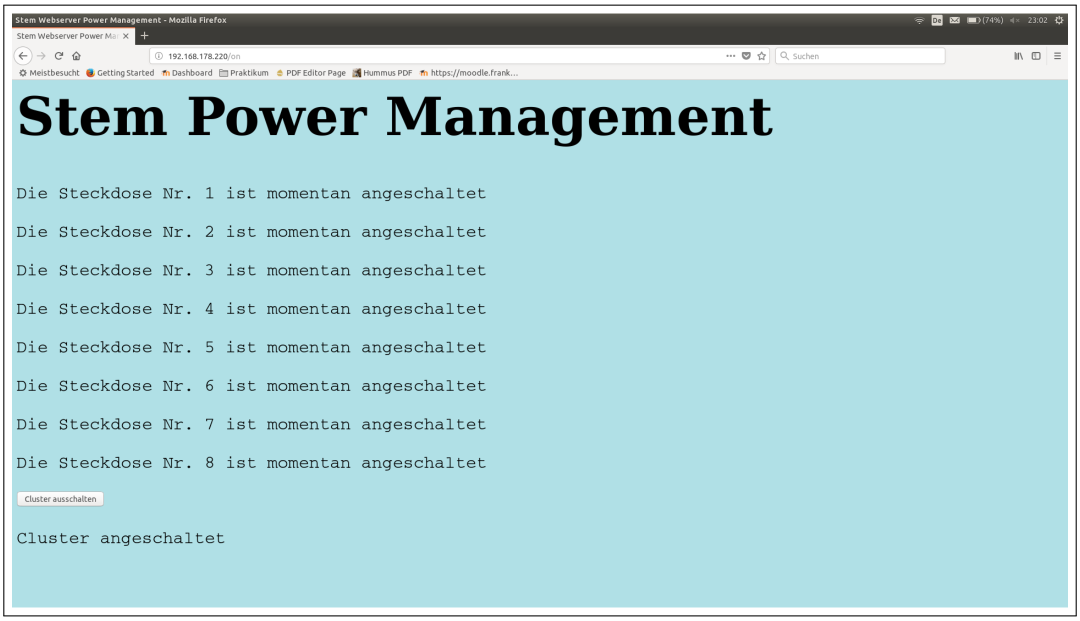


ABBILDUNG 5.6: Anzeige der Webseite im Browser

Diese Webseite macht es möglich den Cluster aus der Ferne zu starten. Hierdurch kann der Cluster zu jeder Tages und Nachtzeit administriert werden. Natürlich lässt sich die Webseite umprogrammieren. So kann zum Beispiel jede Steckdose einzeln angesteuert werden. Die in diesem Abschnitt vorgestellte Webseite hatte zum Ziel einen großen Cluster zu schalten. Die Einsatzmöglichkeiten dieser Steckdosenleiste sind vielfältig und können mithilfe des Webservers Flask weiter angepasst werden. Im folgenden Abschnitt wird die Möglichkeit gezeigt, wie Flask dazu verwendet werden kann um einen REST-Webservice bereit zu stellen. Ein kleiner Beispiel Webservice zeigt die Implementation und Funktion eines REST-Webservice.

5.4 Webservice zum Steuern der Steckdosen

Dieser Abschnitt zeigt den Aufbau eines REST-Webservice mithilfe von Flask. Diese Webservices zeichnen sich durch ihr leichtgewichtiges Design aus. Sie sind einfach zu bedienen und sind primär für die Maschine-zu-Maschine Interaktion geeignet. Alle Anfragen an den Server werden dabei zustandslos übertragen. Das heißt, dass alle nötigen Informationen ohne Zwischenspeicherung an den Server oder den Client geschickt werden. Jeder Service einer REST-konformen Anwendung ist dadurch gekennzeichnet, dass ein Service über eine eindeutige URL (Uniform Resource Locator) erreichbar ist. Weiter zeichnet sich ein REST-Service dadurch aus, dass er mit den 4 HTTP-Methoden `GET`,`PUT`,`POST` und `DELETE` auskommt. Dadurch kann eine einfache Architektur für Webservices erschaffen und einfach genutzt werden [17].

Das Python Framework Flask kann nicht nur dazu verwendet werden einen Webserver zu implementieren, mit diesem Framework lassen sich auch REST-konforme Webservices entwickeln [30]. Dabei bietet Flask Funktionen und Datentypen, die bei der Implementation eines Webservices hilfreich sind. So bietet es die Möglichkeit **JSON**-Objekte zu erstellen. JSON steht für Java Script Object Notation und wird zum Übertragen von Daten zwischen einem Client und einem Server verwendet. JSON eignet sich durch seine gute Struktur zum seriellen Übertragen von Daten und bietet eine kompakte Schreibweise.

Das Listing 5.10 zeigt den Auszug eines Webservices zum Ansteuern der Steckdosen. Dabei wurde die Definition und Initialisierung der Steckdosen weggelassen, da schon Abschnitt 5.3 erklärt, wie das geschieht. Anders als die Webserver Anwendung wurde bei diesem Webservice auf eine Modularisierung verzichtet, da hier keine externen Daten, wie zum Beispiel eine HTML Seite, definiert werden. Es müssen keine Dateien außer der Datei für den Webservice selbst administriert werden, weshalb eine Modularisierung in diesem Zusammenhang einen erhöhten Aufwand bringen würde.

Wie bei der Webserver Anwendung aus 5.3 werden hier Python Funktionen implementiert, die den eigentlichen Service bereitstellen. Der Dekarator definiert eine URI für den Service. So definiert Zeile 5 die URI `/webservice/stem/status`. Der zweite Parameter der Funktion `route()` definiert die HTTP-Methode, mit der auf den Service zugegriffen wird. In diesem Fall wird die Methode `GET` verwendet. Der Aufruf dieses Services gibt den Status aller GPIO-Ports als JSON-Objekt zurück. Dazu dient die Funktion `jsonify` in der Return-Anweisung der Funktion. Diese Funktion ist eine Standardfunktion des Flask-Frameworks und wandelt die Daten des übergebenen Arrays in JSON-Objekte um, die an den Client gesendet werden. Das lässt sich mit dem Kommandozeilenprogramm `curl` testen.

```
app = Flask(__name__)

# gibt den Status aller Steckdosen als JSON Objekt zurueck
@app.route('/webservice/stem/status', methods=['GET'])
def status():

  # liest den Status aller Ports aus
  for dose in dosen:
    dosen[dose]['status'] = GPIO.input(dose)

  return jsonify({'Dose':dosen})


# schaltet alle Steckdosen der Reihe nach an
@app.route('/webservice/stem/on/all', methods=['GET'])
def allOn():

  for dose in dosen:
    GPIO.output(dose, GPIO.LOW)
    time.sleep(1)

  return "Alle Steckdosen angeschaltet!\n"


# schaltet alle Steckdosen der Reihe nach aus
@app.route('/webservice/stem/off/all', methods=['GET'])
def allOff():

  for dose in dosen:
    GPIO.output(dose, GPIO.HIGH)
    time.sleep(1)

  return "Alle Steckdosen ausgeschaltet!\n"


if __name__ == '__main__':
    app.run(debug=True, host='192.168.178.200',port=2789)
```

LISTING 5.10: Auszug Webservice zum Steuern der Steckdosen

Listing 5.11 zeigt einen Auszug der Antwort des Webservice nach dem Absetzen des Kommandos:

Prompt:$ `curl -i http://192.168.178.220:2789/webservice/stem/status`

Dieses Kommando setzt eine `GET`-Anfrage an den Webservice. Wie bereits im vorangegangenen Absatz erwähnt, wird die Antwort als JSON-Objekt zurückgegeben. Dieses Objekt beginnt ab Zeile 7 und beinhaltet den Status aller Steckdosen. Der Status 1 deutet daraufhin, dass die Steckdosen momentan ausgeschaltet sind. Zwischen Zeile 1 und 5 sieht man Statusinformationen, wie den HTTP-Statuscode 200, der für eine erfolgreiche Anfrage steht. Außerdem werden Informationen zum Server angegeben.

Genau nach dem selben Prinzip funktionieren die anderen zur Verfügung gestellten Services. So zeigt Listing 5.12 die Antwort des Webservice zum Anschalten der Steckdosen. Hier werden auch die Statusinformationen des Servers und der HTML-Statuscode zurück gegeben. Der Unterschied liegt darin, dass die Nachricht in diesem Fall kein JSON-Objekt sondern ein String, das als HTML-Text zurück gegeben wird, ist.

```
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 570
Server: Werkzeug/0.9.6 Python/3.4.2
Date: Wed, 28 Feb 2018 22:10:12 GMT

{
  "Dose": {
    "2": {
      "name": "Steckdose Nr. 1",
      "status": 1
    },
    "3": {
      "name": "Steckdose Nr. 2",
      "status": 1
    },
```

LISTING 5.11: Auszug Antwort des Webservice (Status)

Die hier gezeigten Webservices zeigen, wie einfach Webservices zum Schalten von Steckdosen implementiert werden können. Dabei vereinfacht die Verwendung von Flask die Implementation enorm. Diese Webservices können dazu verwendet werden, um die Steckdose über einen Rechner zu steuern, ohne dabei einen Webbrowser zu verwenden. Denkbar ist dabei ein Szenario, indem ein Raspberry Pi die Steckdosenleiste steuert. Dazu braucht der Raspberry Pi keine grafische Oberfläche oder einen Terminalbrowser wie `lynx`. Die Anfragen an die Steckdosenleiste können mithilfe von `curl` realisiert werden.

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 30
Server: Werkzeug/0.9.6 Python/3.4.2
Date: Wed, 28 Feb 2018 22:09:34 GMT

Alle Steckdosen angeschaltet!
```

LISTING 5.12: Auszug Antwort des Webservice (on/all)

Der in diesem Abschnitt vorgestellte Webservice lässt sich noch erweitern. So kann man diesem Webservice so umbauen, dass jede Steckdose einzeln angesteuert werden kann. Dazu müssen nur die Informationen, die an den Webservice gesendet werden, bearbeitet werden. So könnte eine `POST`-Anfrage mit der Nummer der gewünschten Steckdose selbige anschalten. Weiter könnte eine Funktion zum Abfangen von Fehlern eingebaut werden, die einen Fehlercode zurück gibt, wenn eine Ressource angefragt wird, die nicht existiert.

Fazit

Diese Arbeit hat gezeigt wie man einen Test-Cluster aufbaut. Ein Netzwerk-Switch dient der Vernetzung eines Clusters aus sechs Raspberry Pis. Es wurden die nötigen Schritte vorgestellt, um das Betriebssystem Raspbian auf dem Raspberry Pi mithilfe des Kommandozeilenprogramms `dd` zu installieren. Auch wurden die nötigen Konfigurationen am Betriebssystem und an der Netzwerkschnittstelle des Raspberry Pi dokumentiert. Mit diesen Konfigurationen lässt sich eine Testumgebung aufbauen, die den Cluster **BRAIN 128** simuliert und eine Möglichkeit bietet Tests auszuführen, ohne dabei den laufenden Betrieb des Clusters zu stören.

Die Evaluation unterschiedlicher technischer Möglichkeiten und eine konkrete Realisierung einer Reset-Schaltung mit Optokopplern zeigt die Verwendung des P6-Ports. Dabei wurden die einzelnen Komponenten der Schaltung berechnet und Messergebnisse mit verschiedenen Widerständen dargestellt. Die Ansteuerung der realisierten Schaltung mithilfe von Shell Skripten hat gezeigt, dass die GPIO-Ports des Raspberry Pi ohne Installation von zusätzlichen Paketen angesteuert werden können. Ein automatisches Skript, das mit dem Kommando `ping` arbeitet, ist in der Lage den Reset eines ausgefallenen Raspberry Pi ohne manuellen Eingriff zu bewerkstelligen. Dabei kam das Programm `cron` zum Einsatz. Die daraus resultierende automatische Ausführung des Reset vereinfacht die Administration des Clusters.

Auch wurde die Konstruktion einer netzwerkgesteuerten Steckdosenleiste präsentiert. Eine detaillierte Anleitung zum Bau einer Steckdosenleiste zeigte die Verwendung von einem Raspberry Pi und einer Relaiskarte. Ferner wurde gezeigt, wie man Python dazu benutzt die GPIO-Ports mit der Python Bibliothek RPi.GPIO anzusteuern. Es wurde gezeigt, wie man den Raspberry Pi dazu benutzen kann eine netzwerkgesteuerte Steckdosenleiste zu bauen. Das Python Framework Flask dient als Grundlage zum Aufbau eines Webservers und eines Webservices. Diese Technologien wurden benutzt, um Steckdosen über eine Weboberfläche und einen Webservice aus der Ferne zu schalten. Weiter ermöglicht dieses Framework das Einbinden von Python Skripten in HTML Dateien zum Erstellen von dynamischen Webseiten.

6.1 Ausblick

Die in dieser Arbeit entwickelte Reset-Lösung hat eine Möglichkeit zum Neustarten der Raspberry Pis gezeigt. Die Tests der automatischen Skripte konnten auf einem Testcluster erprobt werden und die Funktion dieser Lösung nachgewiesen werden. Der nächste Schritt ist die Installation dieser Lösung auf dem Cluster des Fachbereich 2. Dabei muss eine Lösung für die Fertigung der Schaltungen gefunden werden. Das Anfertigen der Schaltung durch manuelles löten birgt eine Fehlerquelle, da für den Cluster 128 Schaltungen zusammen gelötet werden müssen. Bei 128 Schaltungen muss eine Möglichkeit gefunden werden diese Schaltungen im besten Fall maschinell zu fertigen. Sie sollte kompakt gebaut sein und das Anschließen sollte einfacher sein. Auch zeigte sich, dass Drahtbrücken als Anschlussmöglichkeit im experimentellen Betrieb geeignet sind, jedoch der Kontakt der Anschlüsse schlecht ist. Im Betrieb sind sie locker und eine Verpolung des Anschlusses ist leicht möglich. Von daher sollte beim Design der Anschlüsse eine Buchse verwendet werden, welche die Kontakte vorgibt und den Pluspol des Eingangs farblich markiert.

Weiter zeigte die in dieser Arbeit konstruierte Steckdosenleiste die Möglichkeiten des Raspberry Pi für Steuerungsaufgaben auf. Eine Weboberfläche zum Schalten der Steckdosen macht ihre Verwendung für einen Benutzer sehr einfach. Die Zuverlässigkeit der erstellten Lösung muss durch Langzeittests weiter erprobt werden. Dazu müssen geeignete Lastzustände getestet und ausgewertet werden. Es gilt zu ermitteln, ob der Langzeitbetrieb der Steckdosenleiste reibungslos funktioniert und die erstellte Lösung eine zufriedenstellende Haltbarkeit aufweist. Auch können Tests, die an die Belastungsgrenze der Steckdosenleiste gehen, aufzeigen wie sich diese im Härtefall verhält. Dadurch können hilfreiche Beobachtungen gemacht werden, die zu Verbesserungen der Hard- und Software beitragen.

Die Webseite der Steckdosenleiste kann durch einen Login vor unautorisierter Benutzung geschützt werden. Auch kann der Webservice durch die Abfrage von einem Benutzernamen und einem Passwort im gleichen Rahmen geschützt werden. Ein weiterer Punkt, der in dieser Arbeit nicht weiter verfolgt wurde, ist die Synchronisation der bereitgestellten Dienste. Da die GPIO-Ports des Raspberry Pi eine begrenzte Ressource darstellen, die von verschiedenen Prozessen angefragt werden, können die Dienste untereinander Blockaden hervorrufen. So kann der Webservice die Steckdosen einschalten und die Weboberfläche oder der Taster können die Steckdosen währenddessen ausschalten. Von daher muss eine Möglichkeit gefunden werden diese Prozesse miteinander zu synchronisieren.

Der Raspberry Pi besitzt eine Auswahl an Schnittstellen. Hier wurde nur Verwendung des Ethernet Ports des Raspberry Pi 3 gemacht. Jedoch besitzt der Raspberry Pi 3 eine WLAN- und Bluetooth-Schnittstelle. Diese Schnittstellen könnten eingesetzt und erprobt werden. Auch besitzt der Raspberry Pi 3 vier USB-Ports, die eine serielle Kommunikation erlauben.

Auch wurden in dieser Arbeit alle Steckdosen nacheinander geschaltet, da der Kontext eine solche Schaltung verlangte. Eine Anpassung des Quellcodes ermöglicht das Ein- und Ausschalten jeder einzelnen Steckdose. Dadurch könnte die Steckdosenleiste für die Verwendung im Haushalt eingesetzt werden. Das ist ein beliebter Anwendungsfall, da hierdurch beispielsweise Geräte wie Fernseher, Rechner oder andere Geräte über das Netzwerk angeschaltet werden können. So könnte ein Benutzer beispielsweise seinen Heimrechner mit angebundenem NAS-Laufwerk bedarfsgerecht anschalten, um Daten vom Heimnetzwerk auf seinen Rechner im Internet zu übertragen.

Eine weitere Möglichkeit die Steckdosenleiste zu erweitern wäre eine Möglichkeit zum Messen von Leistung. Das wäre eine hilfreiche Funktion zum Analysieren von Verbrauchern im Heimnetzwerk. So könnte eine Messung der Leistung helfen Verbraucher, die im Leerlauf Strom verbrauchen, abzuschalten. Ein Programm könnte implementiert werden, das Aufzeichnungen und Statistiken zum Stromverbrauch von Geräten im Haushalt anfertigen könnte. Diese Funktion ließe sich mit einer Smartphone-App erweitern. So könnte eine Smartphone-App die Benutzung der Steckdose vereinfachen. Diese Anwendung könnte zusammen mit anderen Geräten in eine Haussteuerung eingebunden werden.

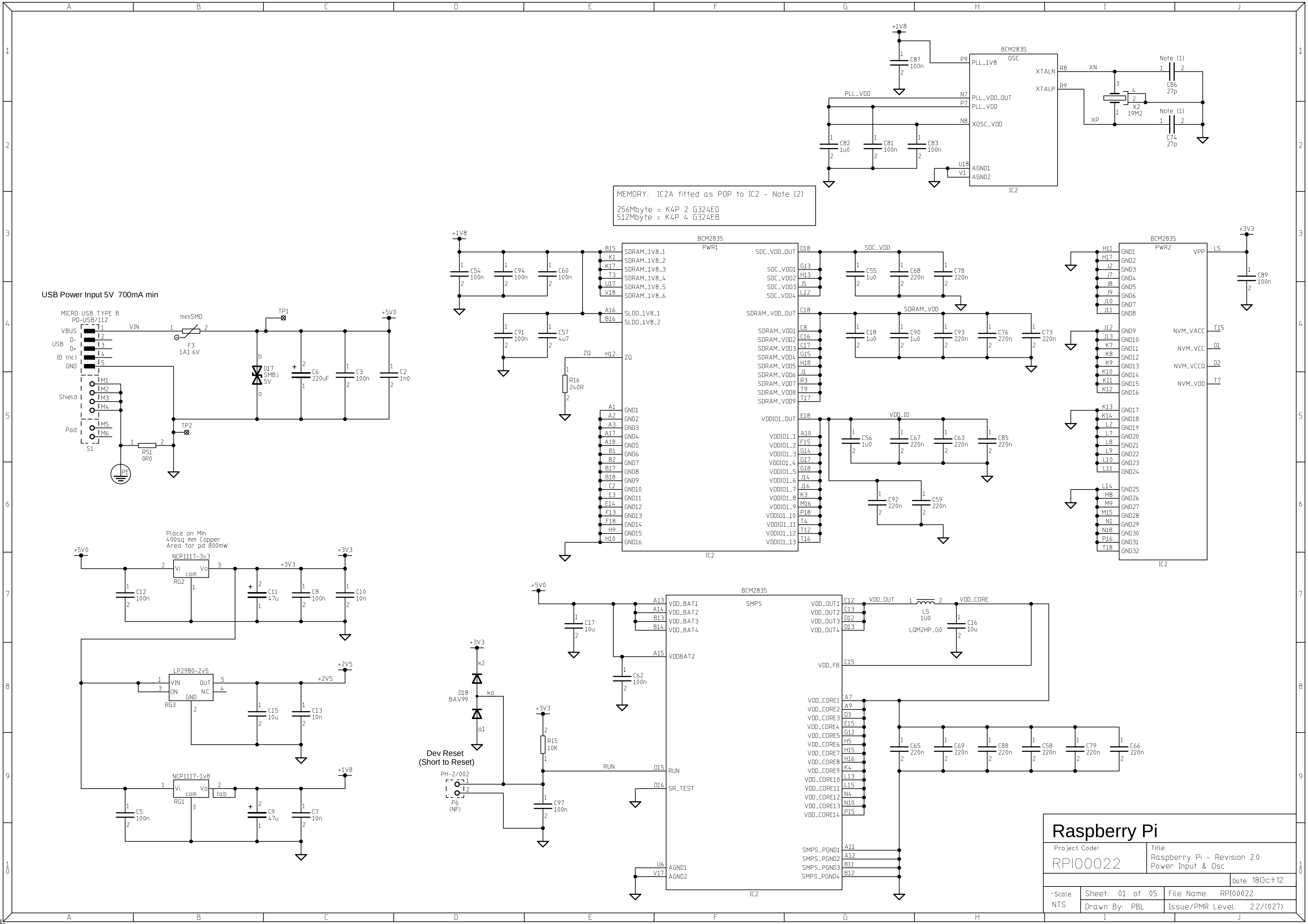
Die oben aufgezeigten Möglichkeiten stellen den Anwendungsfall für die Gebäudeautomation dar. Das Schlagwort „Smarthome“ ist ein aktuelles Thema der Industrie und die hier dargestellte Steckdosenleiste fügt sich in diesen Themenkomplex ein. Netzwerkgesteuerte Steckdosen sind für viele Benutzer ein geeignetes Produkt zur Erweiterung ihrer Haussteuerung und sind zusammen mit intelligenten Heizungssteuerungen zentrale Komponenten einer „Smarthome“ Anwendung. Die Steuerung und Überwachung dieser beiden Prozesse bietet ein großes Potenzial für Einsparungen im Haushalt.

Anhang A

A.1 Auszug: Revision 2.0 Raspberry Pi Model B

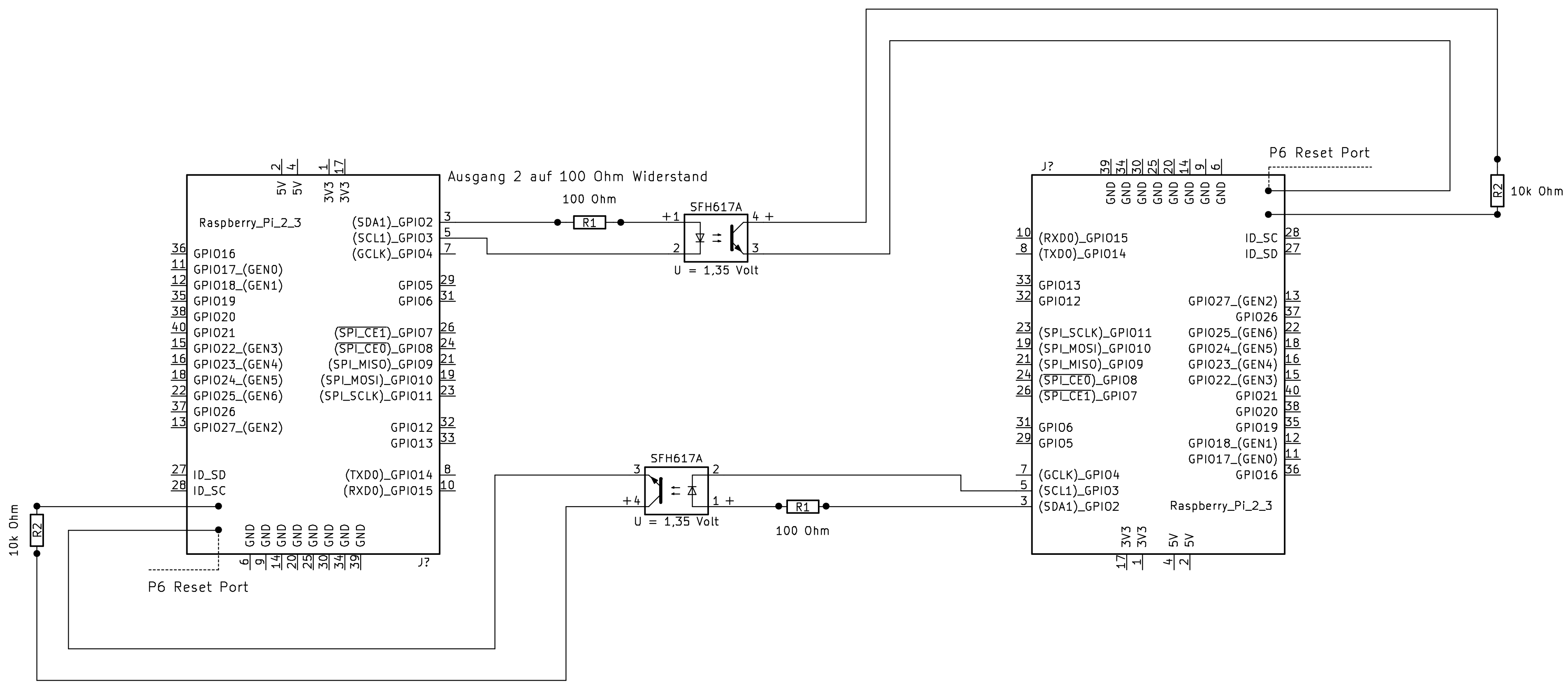
Quelle: Raspberry Pi Foundation

Link: https://www.raspberrypi.org/blog/model-b-revision-2-0-schematics/ (abgerufen 02.03.2018)



Raspberry Pi
Project Code: RPI00022
Title: Raspberry Pi - Revision 2.0 Power Input & Osc
Date: 18Oct12
Scale: NTS
Sheet: 01 of 05
File Name: RPI00022
Drawn By: PBL
Issue/PMR Level: 2.2/(027)
USB Power Input 5V 700mA min
MICRO USB TYPE B
PD-USB/112
VBUS
USB D-
D+
ID (nc)
GND
Shield
Pad
miniSMD
F3
1A1 6V
D17
SMBJ
5V
C6
220uF
C3
100n
C2
1n0
R51
0R0
MEMORY: IC2A fitted as POP to IC2 - Note [2]
256Mbyte = K4P 2 G324ED
512Mbyte = K4P 4 G324EB
BCM2835
OSC
PLL_1V8
PLL_VDD_OUT
PLL_VDD
XOSC_VDD
AGND1
AGND2
XTALN
XTALP
Note [1]
C86
27p
C74
27p
X2
19M2
PWR1
PWR2
SMPS
Place on Min
400sq mm Copper
Area for pd 800mW
NCP1117-3v3
LP2980-2v5
NCP1117-1v8
C11
47u
C9
47u
C15
10u
LQM2HP_G0
L5
1U0
VDD_OUT
VDD_CORE
VDD_FB
VDDBAT2
RUN
SR_TEST
Dev Reset
(Short to Reset)
PH-2/002
P6
(NF)
D18
BAV99
R15
10K
C97
100n
IC2

A.2 Schema Reset Schaltung mit Raspberry Pi



Frankfurt University of Applied Sciences
Henry-Norbert Cocos

Sheet: /
File: Reset_pi.sch

Title: Schaltung Reset Funktion Raspberry Pi

Size: A4	Date:	**Rev:**
KiCad E.D.A.	kicad 4.0.7-e2-6376\u0305\u03055\u03058ubuntu16.04.1	Id: 1/1

Anhang B

B.1 Schaltplan Steckdosenleiste

Anlagenbezeichnung: **Sequenziell Verteiler**

Einspeisung	:	230VAC
Steuerspannung	:	5V DC
Baujahr	:	2018

Gezeichnet :	Henry-Norbert Cocos	Geprüft :	....	Freigegeben :	....
Datum :	04.03.2018	Datum :	....	Datum :	....
Unterschrift :	....	Unterschrift :	....	Unterschrift :	....

	Datum	04.03.2018	Sequenziell Verteiler		Deckblatt	= Nini Schule	
	Bearb.	MARI				+ D	
04.03.2018	Gepr.						Bl. 1
letzte Änderung	Norm						11 Bl.

0	1	2	3	4	5	6	7	8	9

Anlagendaten

Nennspannung : 230 VAC

Nennisolationsspannung :

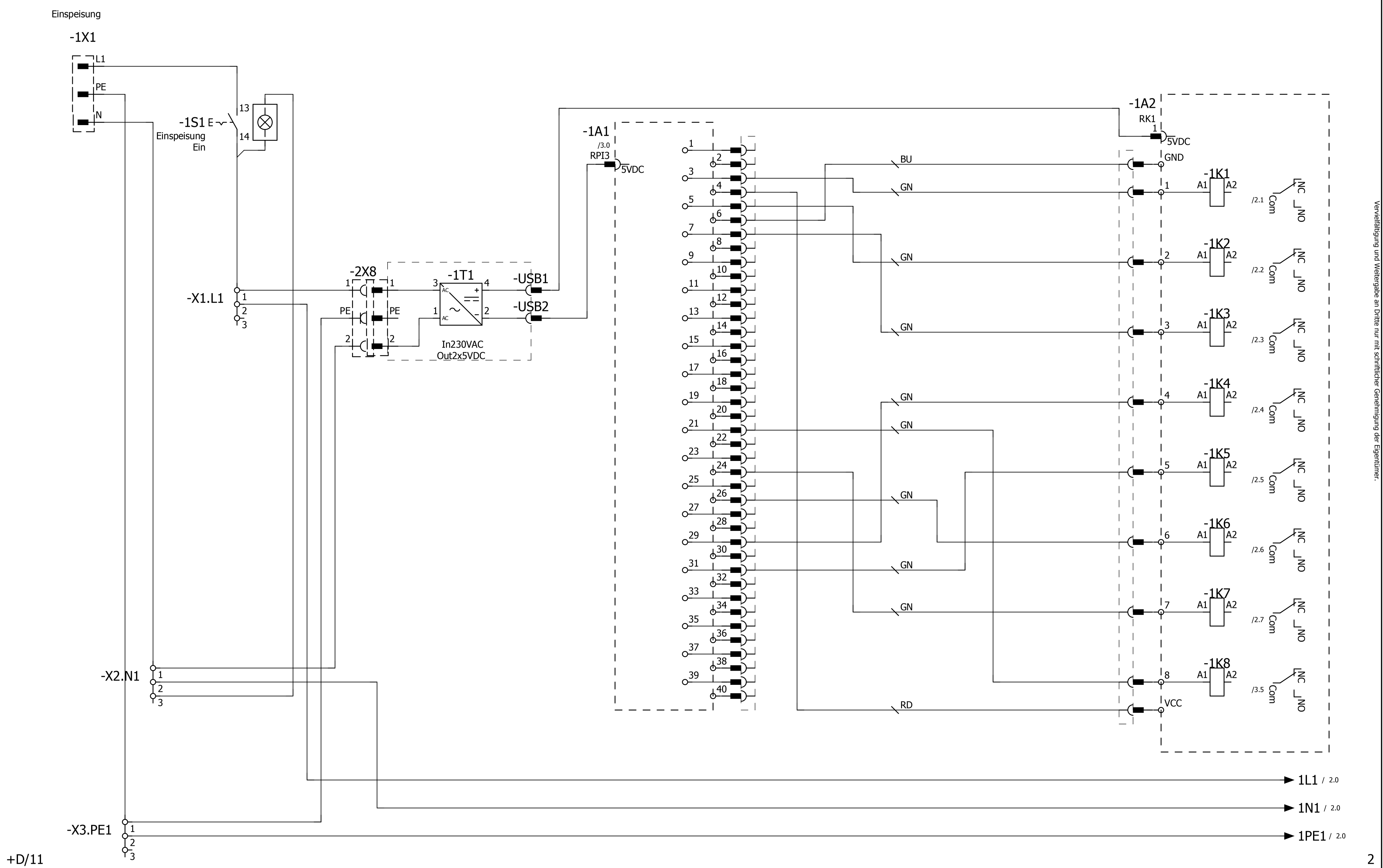
Nennstrom :

Nennkurzschlußstrom :

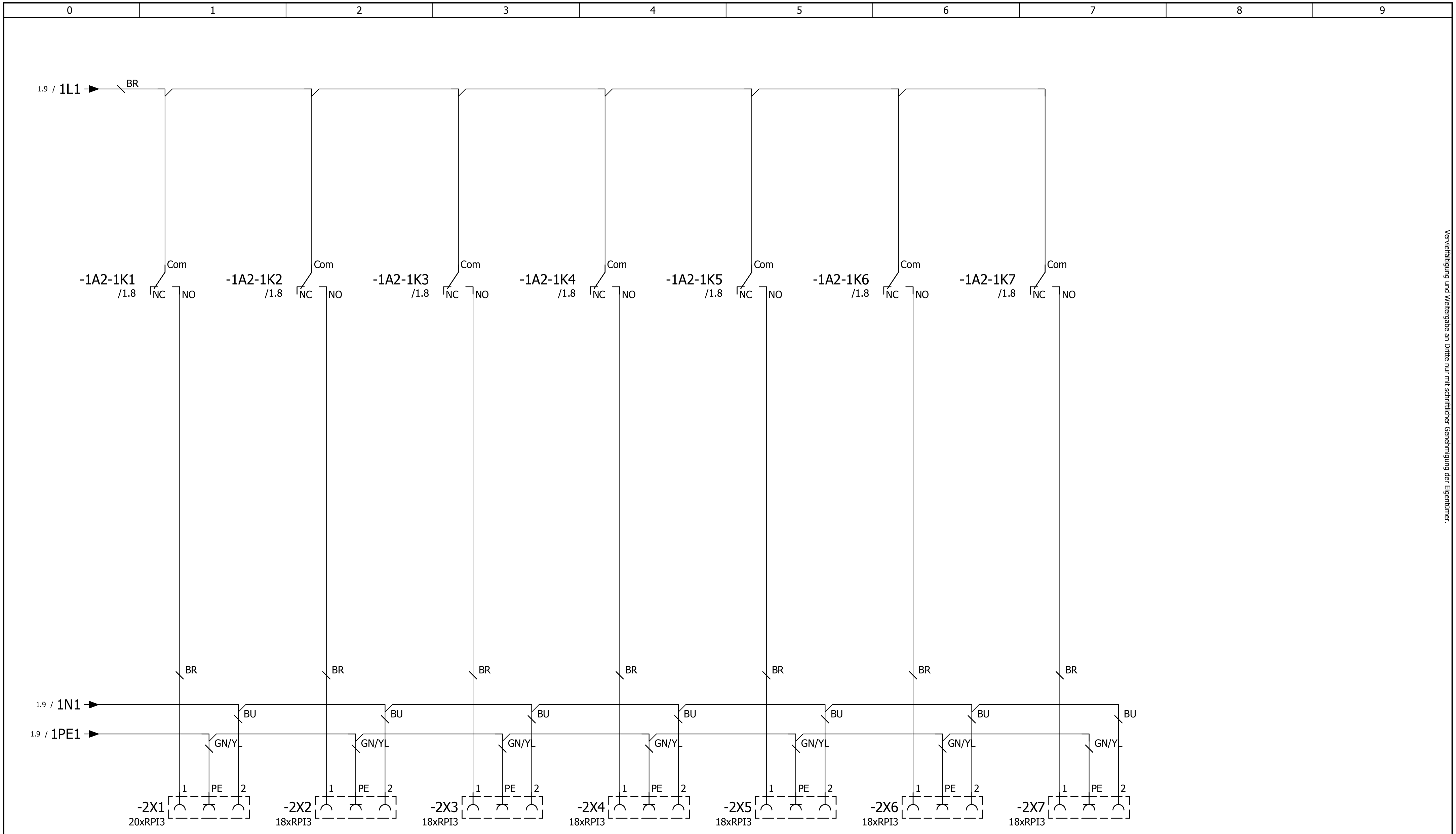
Steuerspannung : 3,3 VDC

Meldespannung :

RPI3 Pin	Ader Farbe	Funktion	Gerät
1	YL	+3V3	
2	RD	+5V VCC	
3	GN	GPIO2	1R1
4	RD	+5V VCC	
5	GN	GPIO3	1R2
6	BU	GROUND	
7	GN	GPIO4	1R3
9	BU	GROUND	
21	GN	GPIO9	1R8
22	GN	GPIO25	OUT
24	GN	GPIO8	1R7
26	GN	GPIO7	1R6
29	GN	GPIO5	1R4
31	GN	GPIO6	1R5
37	GN	GPIO26	IN

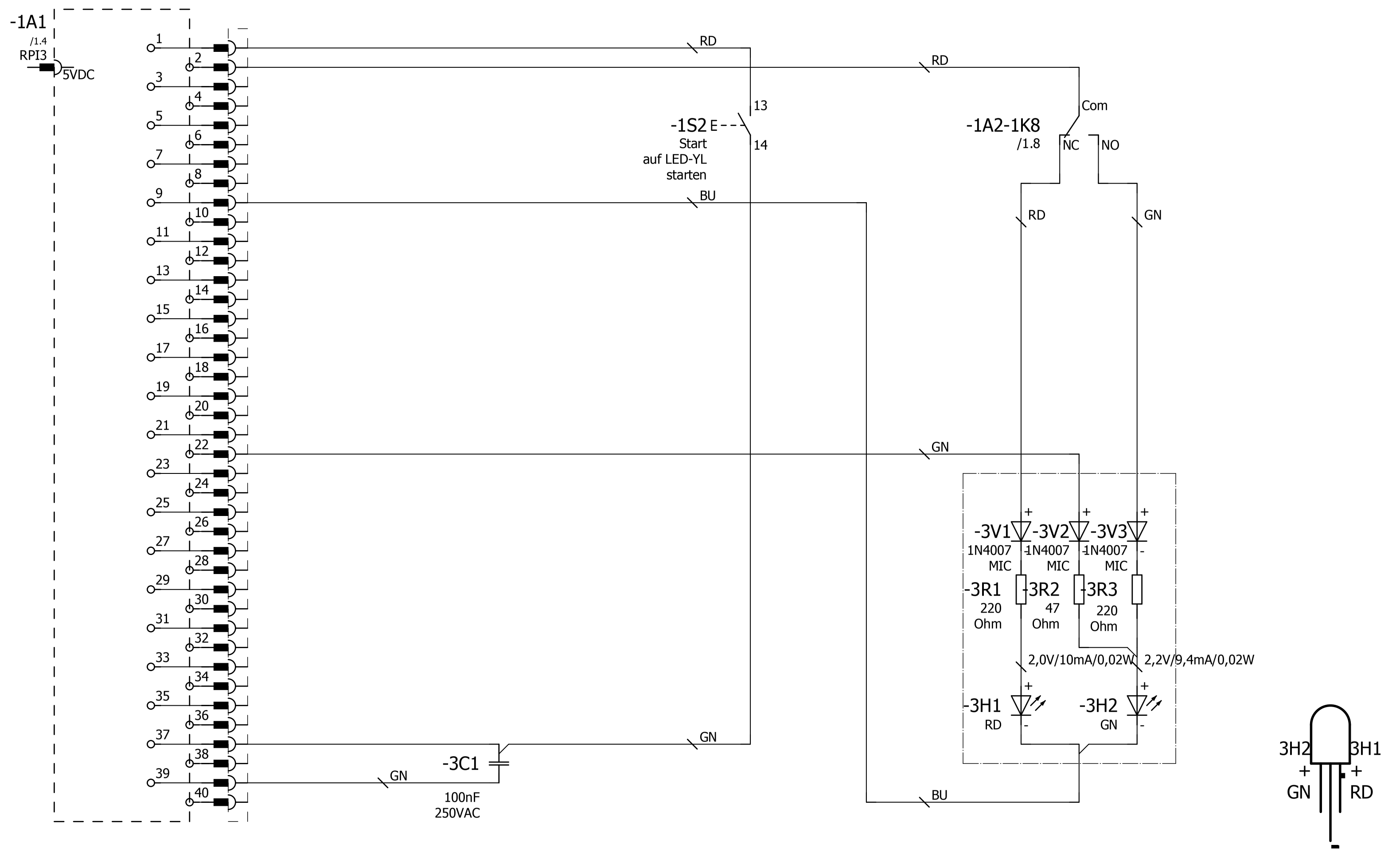


		Datum	04.03.2018	Sequenziell Verteiler	Einspeisung	= Nini Schule	
		Bearb.	MARI			+ F1	
04.03.2018		Gepr.				Bl.	1
letzte Änderung		Norm					7 Bl.



1 3

	Datum	04.03.2018	Sequenziell Verteiler	Steckdosesteuerung	= Nini Schule		
	Bearb.	MARI			+ F1		
04.03.2018	Gepr.					Bl.	2
letzte Änderung	Norm						7 Bl.



LED Farbe	RPI3	Funktion stufe
RD	ist An	RPI3 Boot ~21Sec.
RD+GN= YL	Bereit	RPI3 ist Hochgefahren.
GN	im Betrieb	Sequenziellesteurung ist fertig -1A2-1K8 ist an.

2

4

	Datum	04.03.2018	Sequenziell Verteiler	Start Taste LED Meldung
	Bearb.	MARI		=Nini Schule
04.03.2018	Gepr.			+F1
letzte Änderung	Norm			Bl. 3 / 7 Bl.

Literatur

[1] C. Baun, H.-N. Cocos und R.-M. Spanou. „Erfahrungen beim Aufbau von großen Clustern aus Einplatinencomputern für Forschung und Lehre“. In: *Informatik-Spektrum* 1 (2018). https://link.springer.com/article/10.1007/s00287-017-1083-9 (abgerufen 02.03.2018), 1–11.

[2] *Understanding the Raspberry Pi Boot Process (Englisch)*. http://wiki.beyondlogic.org/index.php?title=Understanding_RaspberryPi_Boot_Process (abgerufen 02.03.2018).

[3] *Making a Reset Switch for your Rev 2 Raspberry Pi (Englisch)*. http://raspi.tv/2012/making-a-reset-switch-for-your-rev-2-raspberry-pi (abgerufen 02.03.2018).

[4] *What are the RUN pin holes on Raspberry Pi 2? (Englisch)*. https://raspberrypi.stackexchange.com/questions/29339/what-are-the-run-pin-holes-on-raspberry-pi-2 (abgerufen 02.03.2018).

[5] *Reichelt Suche ip Steckdose*. https://www.reichelt.de//2/index.html?ACTION=2&GROUPID=5828&START=0&OFFSET=16& (abgerufen 02.03.2018).

[6] *Reichelt Suche ip schaltleisten*. https://www.reichelt.de/Messen-Steuern-Regeln/2/index.html?ACTION=2&LA=514&GROUPID=5828 (abgerufen 02.03.2018).

[7] T. Klein und D. Wenzel. *Schaltbare Steckdosenleiste mit dem Raspberry Pi*. https://klenzel.de/5190 (abgerufen 02.03.2018).

[8] *OpenHAB (Englisch)*. https://www.openhab.org/ (abgerufen 02.03.2018).

[9] *Raspberry Pi*. https://de.wikipedia.org/wiki/Raspberry_Pi (abgerufen 02.03.2018).

[10] *Physical Computing*. https://de.wikipedia.org/wiki/Physical_Computing (abgerufen 02.03.2018).

[11] *An introduction to GPIO and physical computing on the Raspberry Pi (Englisch)*. https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/README.md (abgerufen 02.03.2018).

[12] *Der-I2C-Bus-des-Raspberry-Pi*. http://www.raspberry-pi-geek.de/Magazin/2015/01/Der-I2C-Bus-des-Raspberry-Pi-Teil-1 (abgerufen 02.03.2018).

[13] *Welcome to Raspbian (Englisch)*. https://www.raspbian.org/ (abgerufen 02.03.2018).

[14] *GPIO beim Raspberry Pi per Programm ansteuern*. http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_GPIO_C.html (abgerufen 02.03.2018).

[15] *WiringPi (Englisch)*. http://wiringpi.com/ (abgerufen 02.03.2018).

[16] *Flask (Englisch)*. http://flask.pocoo.org/ (abgerufen 02.03.2018).

[17] R. T. Fielding. „Architectural Styles and the Design of Network-based Software Architectures (Englisch)". https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (abgerufen 02.03.2018). Diss. University of California, 2000.

[18] *Simple Object Access Protocol (SOAP) 1.1 (Englisch)*. https://www.w3.org/TR/2000/NOTE-SOAP-20000508/ (abgerufen 02.03.2018).

[19] *Bipolarer Transistor*. https://www.elektronik-kompendium.de/sites/bau/0201291.htm (abgerufen 02.03.2018).

[20] *Optokoppler / Opto-Koppler*. https://www.elektronik-kompendium.de/sites/bau/0411091.htm (abgerufen 02.03.2018).

[21] *SFA617A - Optocpoupler Datasheet (Englisch)*. https://www.vishay.com/docs/83740/sfh617a.pdf (abgerufen 02.03.2018).

[22] *GPIO pads control (Englisch)*. https://de.scribd.com/doc/101830961/GPIO-Pads-Control2 (abgerufen 02.03.2018).

[23] *GPIO beim Raspberry Pi per Kommando ansteuern*. http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_GPIO_Shell.html (abgerufen 02.03.2018).

[24] *Simple Linux and UNIX Shell Script Based System Monitoring With ping Command (Englisch)*. https://www.cyberciti.biz/tips/simple-linux-and-unix-system-monitoring-with-ping-command-and-scripts.html (abgerufen 02.03.2018).

[25] K.-H. Krefter und H. Schmolke. *DIN VDE 0100 : Daten und Fakten für das Errichten von Niederspannungsanlagen*. Berlin, 2012.

[26] *Was ist ein Pullup- und ein Pulldown-Widerstand?* https://www.elektronik-kompendium.de/public/schaerer/pullr.htm (abgerufen 02.03.2018).

[27] *Raspberry Pi: Interrupt-Verarbeitung mit Python*. http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_GPIO_int.html (abgerufen 02.03.2018).

[28] *Flask Installation (Englisch)*. http://flask.pocoo.org/docs/0.12/installation/#installation (abgerufen 02.03.2018).

[29] *The Flask Mega-Tutorial Part I: Hello, World! (Englisch)*. https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world (abgerufen 02.03.2018).

[30] *Designing a RESTful API with Python and Flask (Englisch)*. https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask (abgerufen 02.03.2018).