Bernburg
Dessau
Köthen

Hochschule Anhalt
Anhalt University of Applied Sciences

emw
Fachbereich
Elektrotechnik, Maschinenbau
und Wirtschaftsingenieurwesen

Bachelorarbeit

Zur Erlangung des akademischen Grades Bachelor of Engineering (B. Eng.)

Peixiang Liu

Vorname Nachname

Elektro- & Informationstechnik, 2014, 4061440

Thema:

Visualisierung von Energieverbräuchen als Web-Oberfläche im Gebäudemanagement unter Zuhilfenahme eines Messclusters

Prof. Dr. Ingo Chmielewski

Vorsitzender der Bachelorprüfungskommission/1. Prüfer

Prof. Dr. Michael Brutscheck

2. Prüfer

17.11.2017

Abgabe am

Selbstständigkeitserklärung

Hiermit erkläre ich, dass die Arbeit selbstständig verfasst, in gleicher oder ähnlicher Fassung noch nicht in einem anderen Studiengang als Prüfungsleistung vorgelegt wurde und keine anderen als die angegebenen Hilfsmittel und Quellen, einschließlich der angegebenen oder beschriebenen Software, verwendet wurden.

Köthen, den 17.11.2017

Ort, Datum

Unterschrift des Studierenden

Sperrvermerk

Sperrvermerk: ja [ ] nein [X]

wenn ja: Der Inhalt der Arbeit darf Dritten ohne Genehmigung der/des (Bezeichnung des Unternehmens) nicht zugänglich gemacht werden. Dieser Sperrvermerk gilt für die Dauer von X Jahren.

Köthen, 17.11.2017

Ort, Datum

Unterschrift des Studierenden

Kurzfassung

Für diese Bachelorarbeit wurde bei Professor Chmielewski das Projekt Visualisierung von Energieverbräuchen als Web-Oberfläche im Gebäudemanagement unter Zuhilfenahme eines Messclusters aufgenommen.
Das hauptsächliche Ziel dieser Arbeit ist der Aufbau einer Schnittstelle zwischen Messgerät (NAS-8) und Smart-Meter-System (Volkszähler), um die Datenflüsse aus dem Messgerät in das Frontend des Volkszählers darzustellen. Eine weitere Aufgabe bestand in der Realisierung der Authentifikation der Benutzer in der Web-Oberfläche, um die Leistungsverbräuche in der individuellen Website (Data Visualisierung) der Benutzer deutlich prüfen zu können.
Um die Aufgaben lösen zu können, muss man zuerst die Grundlagen von Linux und der Python-Programmierung lernen, dann muss man noch die existierende Software im Beaglebone Black und die Datenstruktur des Messwertes analysieren.

Anmerkung: Die folgende Kapitel 1 – 3 sind der Inhalt des Berufspraktiums
(01.04.2017 – 31.07.2017)
Die folgende Kapitel 4 – 6 sind der Inhalt der Bachelorarbeit
(01.08.2017 – 31.10.2017)

Inhaltsverzeichnis

1 Motivation und Zielsetzung

Energie, nämlich Strom, Wasser, Erdgas und Erdöl, sind sehr nützlich und wertvoll für die Menschen, deshalb muss man mit diesem kostbaren Gut sehr achtsam umgehen und sparsam sein. Hier gibt es ein Messgerät (NAS-8), das als Stromzähler benutzt werden kann, aber es läuft nicht wie ein normaler Stromzähler. Das Messgerät besteht aus drei Hauptteilen, einer verbundenen Leiterplatte, einem LCD und einem Beaglebone Black. Durch diese Teile kann das Gerät zuerst den Stromwert messen und es zeigt die Messwerte gleichzeitig am Frontend des Smart-Meter-Systems (Volkszähler) an. Das Ziel des Bachelorprojektes ist die Visualisierung von Energieverbräuchen als Web-Oberfläche unter Zuhilfenahme eines Messclusters und die Realisierung der Authentifikation der Benutzer in der Web-Oberfläche, um den Leistungsverbrauch des Benutzers in seiner individuellen Website (Data Visualisierung) deutlich prüfen zu können. Um die Aufgaben lösen zu können, muss man sich zuerst die Grundlagen von Linux aneignen, erlernen und verstehen. Dann muss man die existierende Software im Messgerät analysieren, um die Datenstruktur zu ermitteln.

Abb. 1.1: Gehäuse des LENA-Messgerätes (NAS-8)

Quelle: eigene Abbildung

2 Module und Kommunikation des Messgerätes

2.1 Allgemeines

Das NAS-8 ist ein Leistungs- und Energiemessgerät mit integriertem Netz- und Anlagenschutz nach VDE-AR-4105 und BDEW Mittelspannungsrichtlinie. Es kann als integrierter oder zentraler NA-Schutz in Niederspannungs- oder Mittelspannungserzeugungsanlagen eingesetzt werden. Es erfasst alle relevanten Werte zur Leistungs- und Energiemessung in einem 3-phasigen Drehstromnetz und führt die notwendigen Überwachungs- und Schutzfunktionen aus. Mit einem optional einsetzbaren Miniatur-PC (BBB-PC) kann die Verbindung zu einer übergeordneten Steuerung hergestellt werden. Das Gerät kann zur Steuerung von Leistungsbezug oder Leistungsabgabe eingesetzt werden.

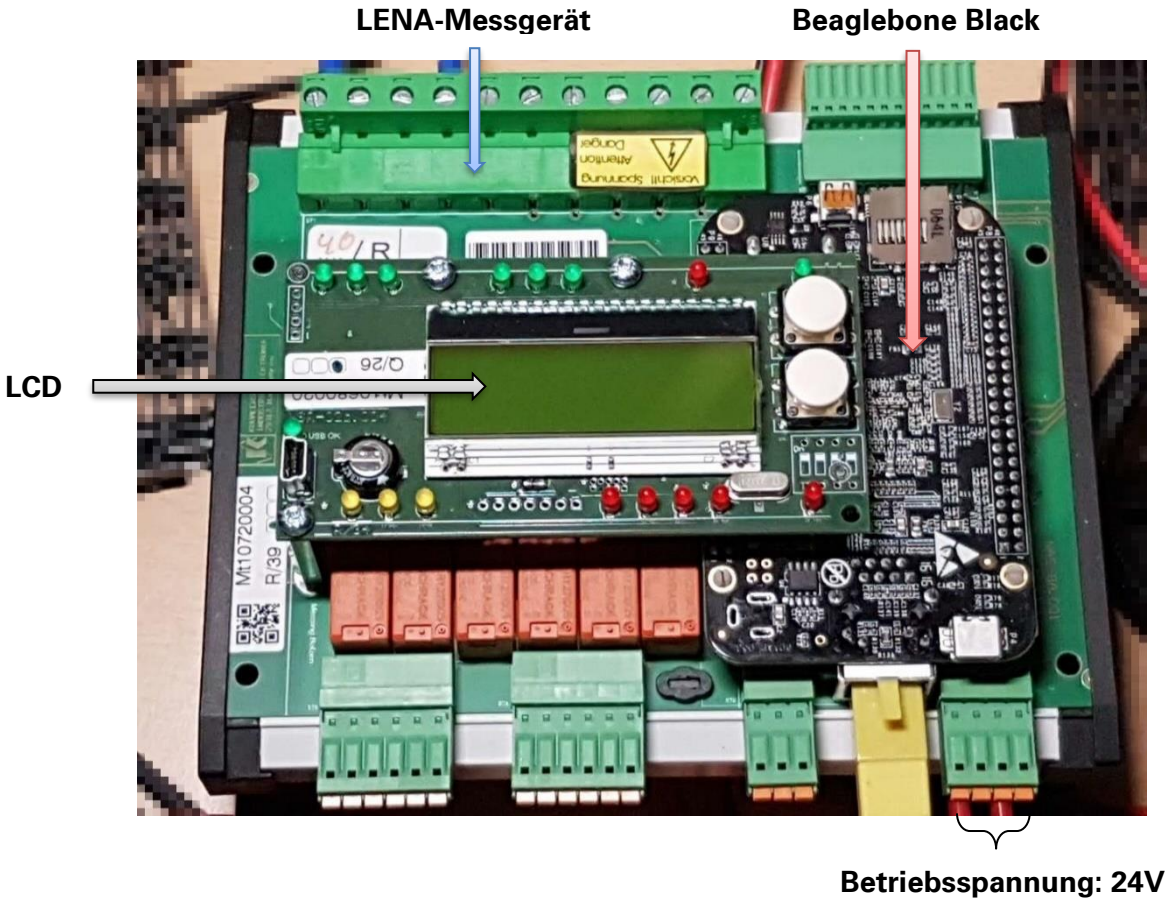


Abb. 2.1: Innere Struktur des LENA-Messgerätes (NAS-8)

Quelle: eigene Abbildung

2.2 Messung der LENA-CPU

2.2.1 LENA-CPU

Die LENA-CPU ist der Hauptteil des Messgerätes, nämlich ein Stromzähler, mit dem man den Stromwert, den Spannungswert und den Leistungswert direkt messen kann. Mehrere technische Daten sind in folgender Tabelle aufgeführt:

Betriebsspannung:	24V DC (18...36V), galvanisch getrennt
Absicherung:	4A träge
Leistungsaufnahme:	ca. 8W bei 24V
Messbereich Spannung:	ca. 10 bis 280 V AC Sternpunktspannung ca. 16 bis 484 V AC Außenleiterspannung 35,0 bis 75,0 Hz besser 0,2 % vom Endwert / Klasse 0,2
Messbereich Frequenz:	Frequenzerkennung ab ca. 10 V Sternpunktspannung, 35,0 Hz bis 75,0 Hz besser 0,01 Hz absolut
Messbereich Strom:	Ca. 150 mA bis 6000 mA, 35,0 bis 75,0 Hz besser 0,5 % vom Endwert / Klasse 0,5
Digitale Eingänge:	4 Stück mit gemeinsamer Masse LowActive (Kontaktspannung 12 V 5mA DC, optoentkoppelt) Leitungen nicht länger als 3 m
Relaisausgänge:	230 V / 50 Hz / 2 A - 4 x Schließer mit gemeinsamer Wurzel - 3 x neutrale Wechsler
Analogausgänge 1+2 (0 .. 10 V):	0(2) .. 10 V DC +/- 0,05 V max. 10,5 V RLast >= 1 kOhm
Analogausgänge 2 (0(4) .. 20 mA):	0(4) .. 20 mA DC +/- 0,1 mA V max. 21 mA RLast >= 400 Ohm
Datenschnittstelle RS-485:	2 Leiter, galvanisch getrennt gegen Versorgungsspannung, Werkseitig: Koralewski-Erweiterungs-Protokoll; Anwenderspezifisches Protokoll möglich

Tabelle 1: Technische Daten von NAS-8

Quelle: Anleitung der NAS-8

2.2.2 Spannungs- und Strommessung

Die Spannungs- und Strommessung ist eine echte Effektivwertmessung. Es werden alle sechs Messpfade simultan mit 32 Abtastungen je Periode gemessen. Die Spannungsmessung arbeitet bis zu einer Spannungsuntergrenze von ca. 10V. Sobald eine Messspannung erkannt wurde, leuchtet die LED der jeweiligen Phase.

2.2.3 Frequenzmessung

Die Frequenz aller drei Spannungen werden jeweils separat erfasst und ausgewertet. Die Frequenzmessung beginnt ab einer Sternpunktspannung von ca.10 V.

2.2.4 3-Leiter - und 3-Leiter + N-Netze

Durch die Wahl des Messverfahrens kann mit oder ohne Sternpunkt gemessen werden. Bei der Messung ohne Sternpunkt ist es nicht notwendig, einen Nullleiter anzuschließen. Bei der 3-Leiter + N-Messung ist durch eine spezielle interne Beschaltung der Klemmen der Wegfall des Nullleiters erkennbar und wird in Form von Spannungsasymmetrie oder Unterspannung Lx angezeigt.

2.2.5 Verhalten bei kleinen Spannungen

Unterhalb einer Messspannung von etwa 20V nimmt die Genauigkeit der Spannungsmessung und der Winkelmessung ab. Bei ca.10V ist die Messspannungsuntergrenze erreicht. Für Frequenz und Spannung wird dann 0 angezeigt.

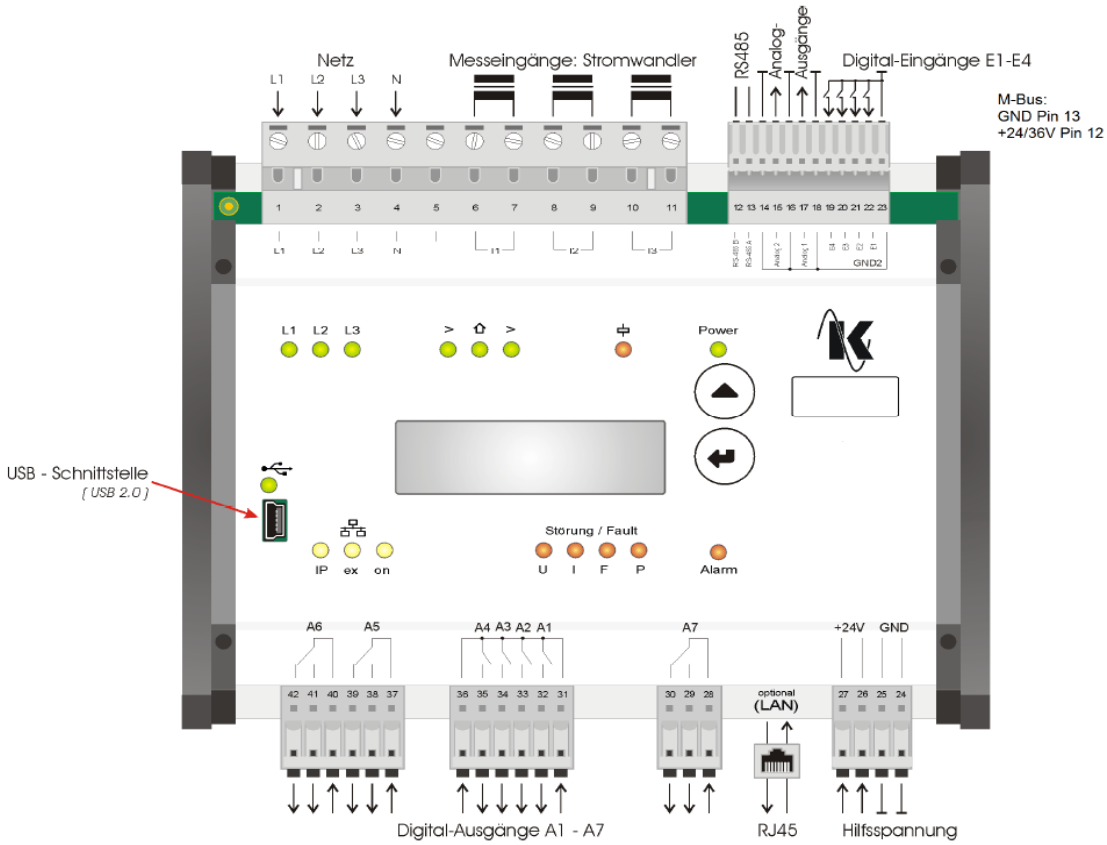


Abb. 2.2: Anschlussplan

Quelle: Anleitung des NAS-8

2.3 Miniatur-PC (Beaglebone Black)

Das Beaglebone Black (BBB) ist auch ein wichtiger Bestandteil des NAS-8, um alle Messwerte aus der LENA-CPU empfangen, speichern und übertragen zu können. Warum wählt man das Beaglebone Black aus?

Heutzutage ist die Open-Source-Hardware ein Trend. Die Open-Source-Hardware ist mit der Open-Source-Software kombiniert, was die Entwicklung einfacher macht. Eine dieser Open-Source-Hardware ist das Beaglebone Black.
Bei dem Beaglebone Black handelt es sich um eine Entwicklungsplattform, die einen günstigen „Sitara" Prozessor AM3358 nutzt, einen Singlecore ARM Cortes-A8 von Texas Instruments. Dieser Einplatinencomputer kann unter Linux betrieben werden und ist ein leistungsstarker, Energie sparender und günstiger Computer, in der Größe einer Kreditkarte, mit allen Fähigkeiten der heutigen Desktop-Computer, aber ohne Lärm. Das Beaglebone Black ist vergleichbar mit anderen Embedded Systemen wie z. B. dem Raspberry Pi und dem Banana Pi. Es besitzt mehrere GPIO (auf Englisch: Generalpurpose input/output oder auf Deutsch Allzweckeingabe/-ausgabe). Es gibt maximal 69 GPIO und einige haben einen speziellen Verwendungszweck, z. B. SPI, I2C, CAN-Bus usw. Die folgende Tabelle 2.3 gibt die wichtigen Eigenschaften des Beaglebone Black wieder.

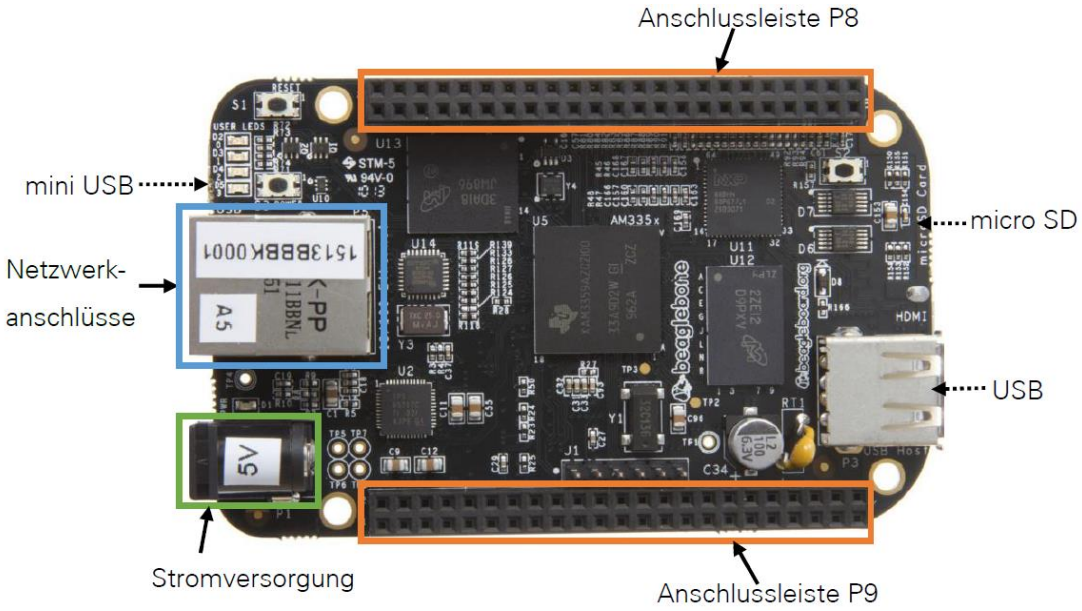


Abb. 2.3: Anschlüsse auf dem Beaglebone Black

Quelle: vgl. mit Beaglebone Black System Reference Manual Rev C.1.pdf, S. 120

Prozessor	Sitara AM3358BZCZ100, 1GHz
DRAM	512MB DDR3L, 800MHz
Onboard Flash	4GB, 8bit Embedded eMMC
SD/MMC	microSD, 3,3V
Leistung	210-460 mA@5V
Energiequelle	Mini USB oder DC Jack
Erweiterungsstecker	Energiequelle: 5V, 3,3V, VDD_ADC(1,8V) 3,3V auf alle I/O, GPIO (max. 69), SPI, I2C,7 AIN, 4 Timer, 4 Serial Port
Video/Audio Out	HDMI 1920x1080@24Hz
Abmessung	3,1inch x 2,1inch, 1,4oz (ca. 40g)

Tabelle 2: Eigenschaften des Beaglebone Blacks

Quelle: vgl. Mit Beaglebone Black System Reference Manual Rev C.1.pdf, S.30

Es gibt zwei Varianten für den Anschluss zwischen dem Beaglebone Black und dem PC. Die erste Variante ist die, dass das Beaglebone Black direkt per USB-Kabel am PC angeschlossen werden kann. Die zweite Variante ist die, dass man das Beaglebone Black mithilfe des Ethernets unter LAN am PC anschließt. Die LENA-CPU und der BBB-PC sind bereits durch den Modbus angeschlossen worden. Aber das BBB passt nicht direkt per USB-Kabel an den PC, deshalb wird die zweite Variante ausgewählt.

Weil das Betriebssystem des BBB Linux ist, muss man alle Arbeiten unter Linux durchführen. Für die Verbindung zwischen dem BBB und dem PC kann man direkt mithilfe des Ethernets (LAN) unter dem Linux-System arbeiten oder eine Software unter Windows am PC installieren, um BBB zu verbinden, z. B. PuTTY. Die Verbindungsart ist „SSH“, die IP-Adresse lautet 192.168.2.112 und die Portnummer ist 22. Die Einstellung zeigt folgende Abbildung 2.4:

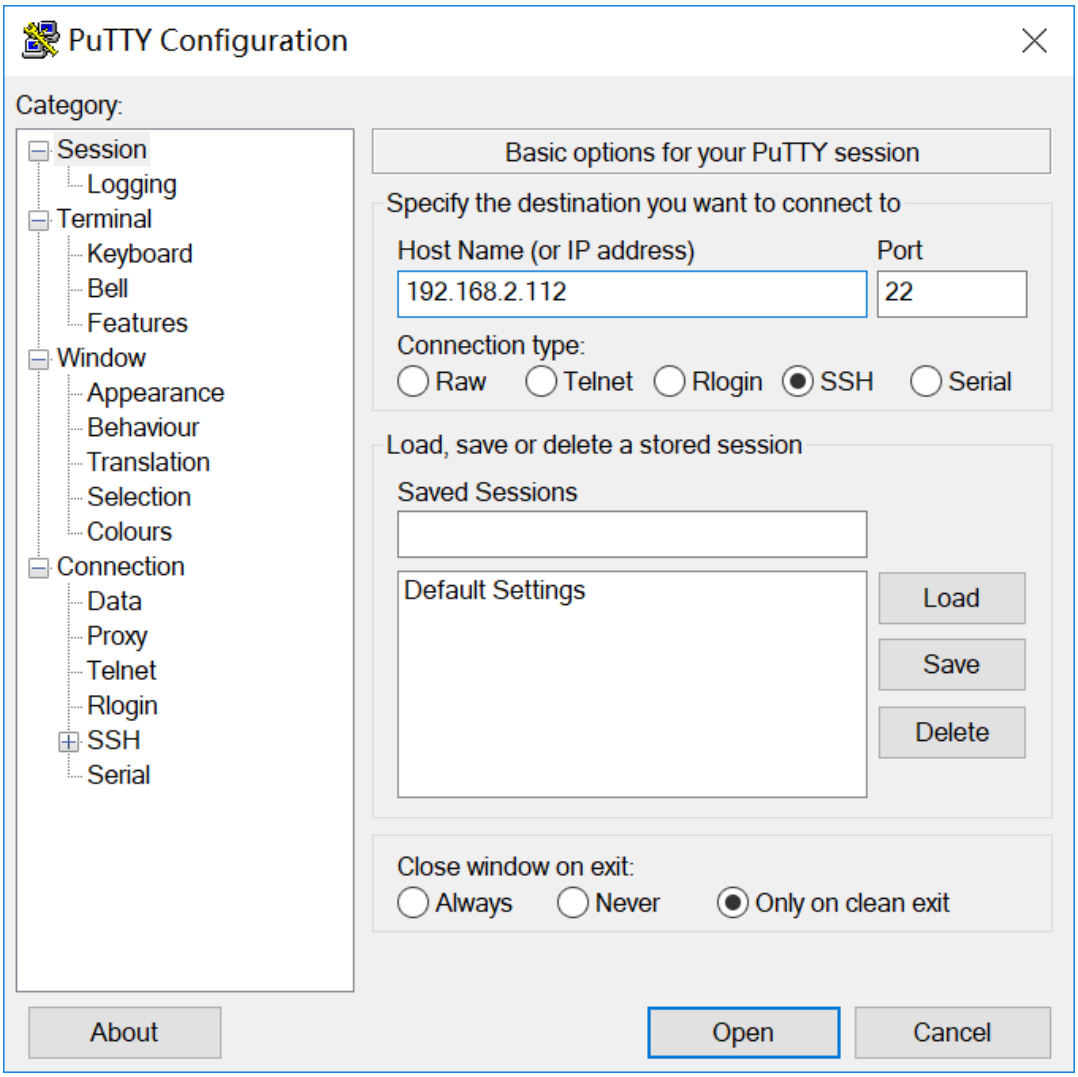


Abb. 2.4: Verbindung zum Beaglebone Black mit PuTTY

Quelle: eigene Abbildung

Danach erscheint ein schwarzes Dialogfeld, das eine Linux-Konsole ist, über die man sich auf dem Beaglebone Black anmelden muss. Verwendet man den Benutzernamen „root“, dann braucht man auch das Passwort „root“ zur Anmeldung.
Dann kann man Linux-Kommandozeilenbefehle direkt auf dem Beaglebone Black nutzen.

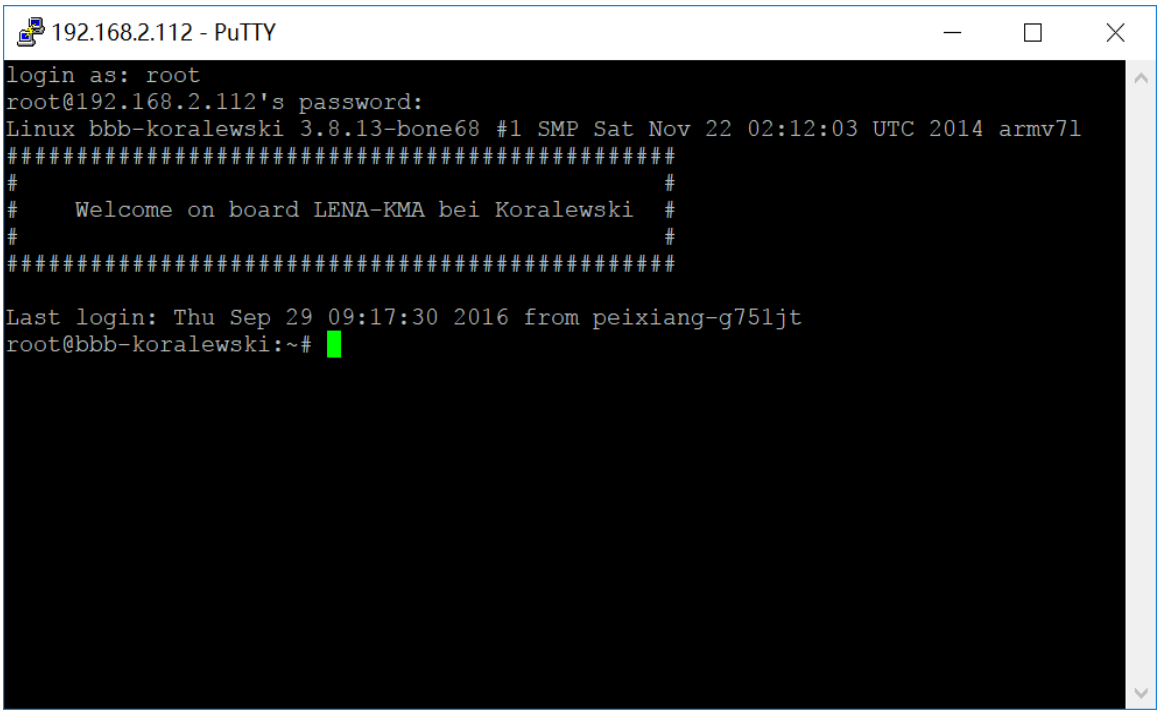


Abb. 2.5: Linux-Konsole bei PuTTY

Quelle: eigene Abbildung

3 Grundlagen der Software und Datenanalyse

3.1 Linux-Grundlage

3.1.1 Allgemeines

Linux ist ein kostenloses Betriebssystem und man kann es sich selbst online herunterladen. Es gibt auch viele Varianten, bekannte Auflagen sind Ubuntu, Debian und CentOS.
Als Linux oder GNU/Linux bezeichnet man in der Regel freie, Unix-ähnliche Mehrbenutzer-Betriebssysteme, die auf dem Linux-Kernel und im Wesentlichen auf der GNU-Software basieren. Die weite, auch kommerzielle Verbreitung, wurde ab 1992 durch die Lizenzierung des Linux-Kernels unter der freien Lizenz GPL ermöglicht. Einer der Initiatoren von Linux war der finnische Programmierer Linus Torvalds. Er nimmt bis heute eine koordinierende Rolle bei der Weiterentwicklung des Linux-Kernels ein und wird auch als Benevolent Dictator for Life bezeichnet.

Linux wird vielfältig und umfassend eingesetzt, beispielsweise auf Arbeitsplatzrechnern, Servern, Mobiltelefonen, Routern, Netbooks, Embedded Systems, Multimedia-Endgeräten und Supercomputern. Dabei wird Linux unterschiedlich häufig genutzt: So ist Linux im Server-Markt wie auch im mobilen Bereich eine feste Größe, während es auf den Desktops und Laptops eine noch geringe, aber wachsende Rolle spielt.
Linux wird von zahlreichen Nutzern verwendet, darunter private Nutzer, Regierungen und Organisationen wie das Französische Parlament, die Stadt München und das US-Verteidigungsministerium, Unternehmen wie Samsung, Siemens, Google, Amazon, Peugeot usw.

Abb. 3.1: Betriebssystem-Linux

Quelle: http://d.youth.cn/tech_focus/201407/t20140728_5568191.html (18.08.2017)

3.1.2 Häufige Kommandos von Linux (Unix)

Linux-Systeme zeichnen sich durch eine Vielzahl von Kommandos aus, mit denen sich über eine Shell das Betriebssystem bedienen lässt. Wenn man diese Kommandos geschickt benutzt, kann man sehr schnell und direkt seine Ziele erreichen. In der folgenden Tabelle 3 sind einige häufige Kommandos von Linux aufgeführt:

cp	Datei kopieren (CoPy)
ls	Dateien in einem Verzeichnis anzeigen (LiSt)
mv	eine Datei verschieben oder umbenennen (MoVe)
rm	Löschen einer Datei (ReMove)
mkdir	erzeugt ein neues Verzeichnis (MaKe DIRectory)
rmdir	löscht ein (leeres) Verzeichnis (ReMove DIRectory)
cd/cd..	wechselt in ein anderes Verzeichnis (Change Directory)/Zurück
pwd	Anzeige des aktuellen Verzeichnispfades (Print Working Directory)
su	Benutzer wechseln, standardmäßig wird zu root gewechselt (Substitute User)
sudo	Kommando mit besonderen Rechten ausführen
passwd	Benutzerpasswort ändern (siehe auch chsh)
cat	Ausgabe und/oder Verkettung von Textdateien in der Kommandozeile (conCATenate)
echo	Ausgabe
tail	Ausgabe der letzten Zeilen von der Eingabe (dies ist üblicherweise eine Datei)
vi/vim	(VIsual editor)/ (Vi IMproved)
ping	schickt ein ping an einen anderen Rechner
netstat	Anzeige der Netzverbindungen
ifconfig	Netzwerkschnittstellenkonfiguration
kill	Einen Prozess beenden oder andere Signale an ihn senden
uname	Ausgabe von Informationen über Betriebssystem und Rechner

Tabelle 3: Linux - Kommandos (Auswahl)

Quelle: https://de.wikipedia.org/wiki/Unix-Kommando

3.1.3 Embedded Linux

Als Embedded Linux bezeichnet man ein eingebettetes System mit einem auf dem Linux-Kernel basierenden Betriebssystem. Dies impliziert nicht den Gebrauch bestimmter Bibliotheken oder Anwendungen mit diesem Kernel.
Embedded-Linux-Systeme werden normalerweise aufgrund ihrer verschiedenen Systemeigenschaften und nicht aufgrund ihrer Einsatzorte eingeteilt. Das können u. a. die Skalierbarkeit, die Unterstützung für bestimmte Prozessoren, der Stromverbrauch, das Zeitverhalten (Echtzeitfähigkeit), der Grad der möglichen Nutzerinteraktionen oder andere wesentliche Faktoren sein.

3.2 Kommunikation zwischen LENA-Gerät und Beaglebone Black

Die Funktion des Beaglebone Black ist die Speicherung und Übertragung der Messwerte aus dem LENA-Gerät. Das LENA-Gerät misst zuerst die Strom- oder Spannungswerte, danach sendet es diese Werte mithilfe des Modbus an das Beaglebone Black. Im Inneren des Beaglebone Blacks gibt es einige Software, um die Speicherung und Übertragung der Messwerte zu realisieren. Diese Software läuft beidseitig vom Modbus bis zur IPC.
Folgende Abbildung zeigt den Prozess der inneren Kommunikation:

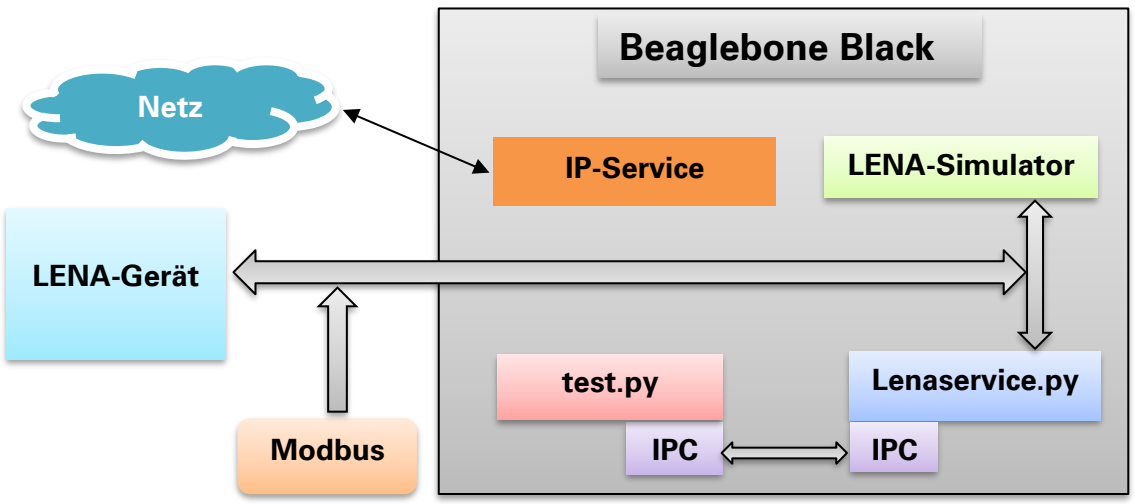


Abb. 3.2: Innere Kommunikation des Messgerätes (Nas-8)

Quelle: eigene Abbildung

3.2.1 Modbus

Das Modbus-Protokoll ist ein Kommunikationsprotokoll, das auf einer Master / Slave- bzw. einer Client / Server-Architektur basiert. Es wurde 1979 von Gould-Modicon für die Kommunikation mit seinen speicherprogrammierbaren Steuerungen ins Leben gerufen. In der Industrie hat sich der Modbus zu einem De-facto-Standard entwickelt, da es sich um ein offenes Protokoll handelt. Seit 2007 ist die Version Modbus TCP Teil der Norm IEC 61158.
Mittels Modbus können ein Master (z. B. ein PC) und mehrere Slaves (z. B. Mess- und Regelsysteme) verbunden werden. Es gibt zwei Versionen: Eine für die serielle Schnittstelle (EIA-232 und EIA-485) und eine für Ethernet.

Bei der Datenübertragung werden drei verschiedene Betriebsarten unterschieden:

- Modbus RTU
- Modbus ASCII
- Modbus TCP

3.2.1.1 RTU-Modbus

Die Modbus RTU (RTU: Remote Terminal Unit, entfernte Terminaleinheit) überträgt die Daten in binärer Form. Dies sorgt für einen guten Datendurchsatz, allerdings können die Daten nicht direkt vom Menschen ausgewertet werden, sondern müssen zuvor in ein lesbares Format umgesetzt werden.

Start	**Adresse**	**Funktion**	**Daten**	**CR-Check**	**Ende**
Wartezeit (min.3,5 Zeichen)	1 Byte	1 Byte	n Byte	2 Byte	Wie Start

Tabelle 4: Protokollaufbau des RTU-Modbus

Quelle: https://de.wikipedia.org/wiki/Modbus

3.2.1.2 Modbus/TCP

Modbus/TCP ist RTU sehr ähnlich, allerdings werden TCP/IP-Pakete verwendet, um die Daten zu übermitteln. Der TCP-Port 502 ist für Modbus/TCP reserviert. Modbus/TCP ist seit 2007 in der Norm IEC 61158 festgelegt und wird in IEC 61784-2 als CPF 15/1 referenziert.

Transaktions-nummer	Protokoll-kennzeichen	Zahl der noch folgenden Bytes	Adresse	Funk-tion	Daten
2 Byte	2 Byte(immer0x0000)	2 Byte (n+2)	1 Byte	1 Byte	n Byte

Tabelle 5: Protokollaufbau der Modbus/TCP

Quelle: https://de.wikipedia.org/wiki/Modbus

3.2.1.3 ASCII-Modbus

Im Modbus ASCII wird keine Binärfolge, sondern der ASCII-Code übertragen. Dadurch ist es direkt für den Menschen lesbar, allerdings ist der Datendurchsatz im Vergleich zu RTU geringer.

Start	Adresse	Funktion	Daten	LR-Check	Ende
1 Zeichen (:)	2 Zeichen	2 Zeichen	N Zeichen	2 Zeichen	2 Zeichen(CRLF)

Tabelle 6: Protokollaufbau des ASCII-Modbus

Quelle: https://de.wikipedia.org/wiki/Modbus

3.2.2 IPC (Interprozesskommunikation)

Der Begriff Interprozesskommunikation bedeutet in der Informatik verschiedene Verfahren des Informationsaustausches zwischen den Prozessen eines Systems. Mithilfe eines Shared Memory erfolgt die Kommunikation dadurch, dass mehrere Prozesse auf einen gemeinsamen Datenspeicher zugreifen können, beispielsweise gemeinsame Bereiche des Arbeitsspeichers. Bei einer Message Queue dagegen werden Nachrichten von einem Prozess an eine Nachrichtenschlange geschickt, von wo diese von einem anderen Prozess abgeholt werden kann.

Beispiel der IPC:

Message Queue (Nachrichtenschlange)

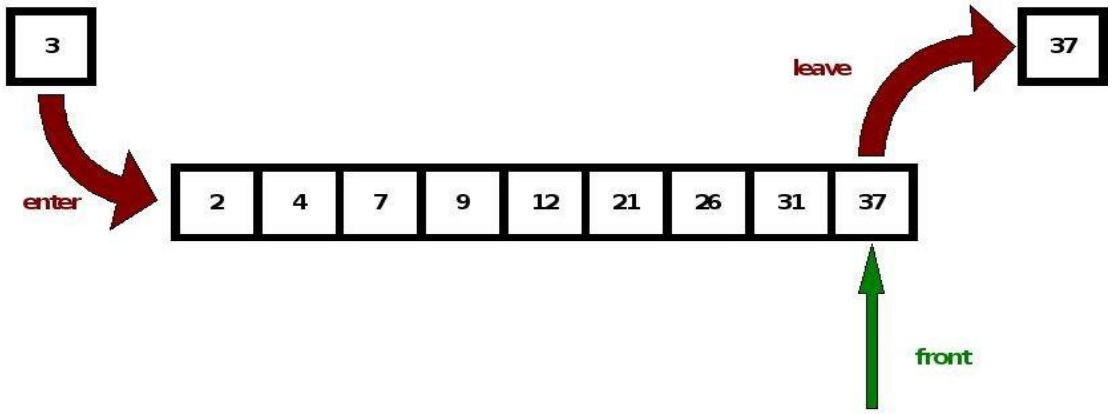


Abb. 3.3: Kommunikation über eine Nachrichtenschlange

Quelle: https://upload.wikimedia.org/wikipedia/commons/4/45/Queue_algorithmn.jpg

(Namenlose) Pipes

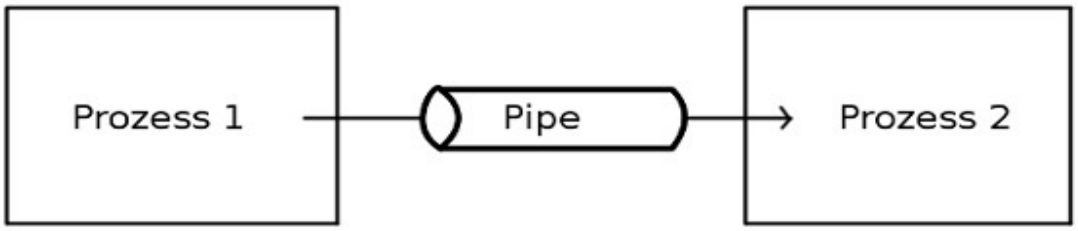


Abb. 3.4: Kommunikation zweier Prozesse über eine Pipe

Quelle: https://upload.wikimedia.org/wikipedia/commons/2/22/Piping.jpg

Weitere Arten sind **benannte Pipes (FIFO-Pipes)** und **Sockets**.

3.3 Analyse der existierenden Software im Messgerät (LENA)

3.3.1 Beziehung der existierenden Software im Messgerät (LENA)

In diesem Projekt gibt es einen wichtigen und schwierigen Prozess, nämlich den, dass man die existierende Software des Messgerätes analysieren muss. Durch diese Software kann man die Beziehung zwischen dem LENA-Gerät und dem BBB erkennen und die Datenstruktur aus den Messwerten erhalten. Diese Software ist eigentlich ein Python-Programm.

LENA--Software	**socierIPC**	socierIPC.py
	socieerMsg	socieerMsg.py
	socierWSGI	socierWSGI.py
	socieerLENA	analyse.py
		appurtenance.py
		configure.py
		lenainstrument.py
		minimodbus.py
		socieerLENA.py
		test.py
		test_cl.py
		test_sluh.py

Tabelle 7: Die existierende Software im Messgerät (LENA)

Quelle: eigene Tabelle

Der Typ „socieerLENA“ ist ein Testeffekt, der Messdaten misst und sammelt. Der Typ „IPC“ ist ein Werkzeug, um die Kommunikation zwischen Prozess und Netzwerk zu realisieren. Und die Methode der IPC ist Pipe. Der Typ „WSGI“ ist eine Datenbank, um Messdaten zu speichern und anzuzeigen.

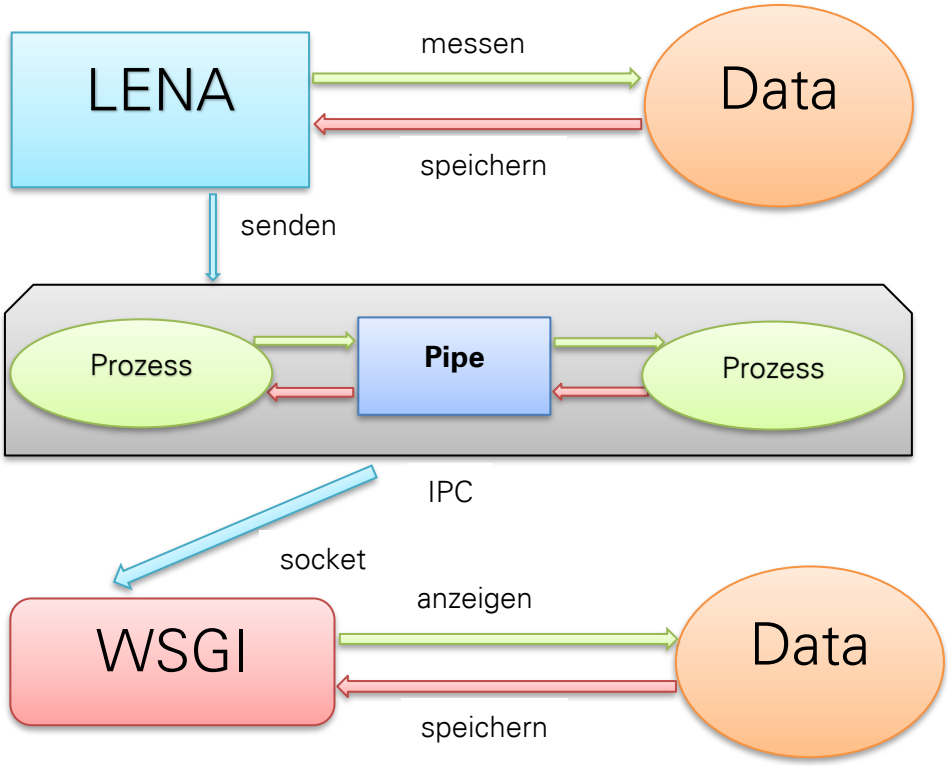


Abb. 3.5: Kommunikation zwischen LENA - Programmen

Quelle: eigene Abbildung

3.3.2 Analyse des Programmes „test.py“

Wenn das Messgerät jeweils anläuft, läuft ein Programm immer automatisch, das unter Typ „socieerLENA“ „test.py“ heißt, und es ist ein zusammenfassendes Programm.

Kerncode von „test.py“

…

```
try:
    self.mBackup.backup_timestamp(format_payload(0,time.time(), descrip-
    tion="timestamp"), rewrite=True)
    response = self.dev.getCyclicvalue()
    self.m1.mtype = 65
```

```
            self.write(str(response))
            logging.warning('<#> Function 65 with response: %s\n', response)
            self.m1.payload = (';'.join(str(v) for v in response))
            self.msgr_c.send(self.m1)
            self.mBackup.backup_fkt_cycle_value(format_payload(65,response,de scrip-
            tion="fkt_get_cycle_values")
            response = self.dev.getActualvalue()
            self.m1.mtype = 66
            self.write(str(response))
            logging.warning('<#>Function 66 with response: %s\n', response)
            self.m1.payload = (';'.join(str(v) for v in response))
            self.msgr_c.send(self.m1)
            self.mBackup.backup_fkt_actual_value(format_payload(66,response,descrip-
            tion="fkt_get_actual_values"))
            self.lsl[0] = analyse.getNetworkstatusbits(self.ns.isIP, self.ns.isNetwork)
            self.mBackup.backup_network_status(format_payload(71,[self.ns.isIP,
            self.ns.isNetwork], description="network status"))
            response = self.dev.setSystemstatus(self.lsl)
            logging.warning('<#> Function 71 with response: %s\n', response)
        except (ValueError, TypeError, IOError) as ErrorString:
            logging.error('* Error info: {0!r}'.format(ErrorString))
        else:
            pass
        ct = ct + 1
...
```

Die Funktion des Programmes hängt vom Kommunikationsablauf ab. Die Kommunikation zwischen dem BBB-PC und der LENA-CPU läuft zyklisch und azyklisch ab. Es werden kontinuierlich, mindestens einmal pro Sekunde folgende Telegramme abgerufen:

FKT_CYCLIC_VALUE 65(0x41)
FKT_ACTUAL_VALUE 66(0x42)
FKT_STATUS_SET 71(0x47)

Weitere Telegramme werden azyklisch dazwischen abgearbeitet.

Die Telegrammfolge sieht dann beispielweise folgendermaßen aus:

65, 66, 71, xx, xx, xx, 65, 66, 71, xx, 65, ……

3.3.3 Analyse des Programmes „lenainstrument.py“

Das Programm „lenainstrument.py“ ist auch sehr wichtig, weil man mit dessen Hilfe die Funktionen erkennen kann. Das ist die Fortsetzung des Programmes „minimodbus.py“. Das Modell ist natürlich gleich wie Modbus, nämlich das Modbus RTU Protokoll. Die Funktionstabelle gehört zur „Funktionszone“. Die geordnete Funktionstabelle ist nachfolgend dargestellt:

Funktionscode	**Kurzname**	**Funktion**
0	----------	Timestamp
65	FKT_CYCLIC_VALUE	Zustände zyklisch holen
66	FKT_ACTUAL_VALUE	Aktuelle Messwerte holen
67	FKT_CONFIG_GET	Konfiguration lesen
68	FKT_CONFIG_SET	Konfiguration schreiben
69	FKT_TIME_GET	Zeit lesen
70	FKT_TIME_SET	Zeit setzen
71	FKT_STATUS_SET	Status setzen
72	FKT_EA_GET	Ein-Ausgang lesen
73	FKT_VALUES1_GET	Messwert 1 holen
74	FKT_VALUES2_GET	Messwert 2 holen
75	FKT_TEXT_GET	Text aus der Konfiguration lesen
100—109	FKT_ERROR_GET100—109	Fehler/Zustandstelegramme holen
110	FKT_STATIC_ERROR_GET	Statische Fehler/Zustände holen
111	FKT_CONFIG_CHANGED	Geänderte Parameter holen
112	FKT_LENA_INFO	LENA-CPU Info holen

Tabelle 8: Funktionsüberblick aus dem Programm „lenainstrument.py“

Quelle: eigene Tabelle

Das Programm ist eine wichtige Bibliothek im Programm „test.py“.

3.4 Beschreibung der Datenstruktur

Teilcode von „test.py“:

…

```
def write(self, string):
        with open("/tmp/lena_tmp","a") as outfile:
            outfile.write(string)
```

…

Im Programm „test.py“ kann man sehen, dass ein Dokument „lena_tmp“ unter „/tmp/“ geöffnet ist und der Inhalt viele Daten, nämlich die Messwerte aus dem LENA-Gerät beinhaltet.

```
root@bbb-koralewski:/tmp# tail -f lena_tmp
[2415921152L, 0][0, 0, 0, 0, 0, 0, 0, 0, 0, 0][2048, 3][0, 0, 0,
 0, 0][2048, 12][0, 0, 0, 0, 0, 0, 0, 0, 0, 0][2048, 15][0, 0, 0
 0, 0, 0][2048, 22][0, 0, 0, 0, 0, 0, 0, 0, 0, 0][268437504, 24]
, 0, 0, 0, 0, 0, 0, 0][2048, 32][0, 0, 0, 0, 0, 0, 0, 0, 0, 0][2
, 0, 0, 0, 0, 0, 0, 0, 0][2048, 44][0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0][2048, 55][0, 0, 0, 0, 0, 0, 0, 0, 0
][0, 0, 0, 0, 0, 0, 0, 0, 0, 0][2048, 66][0, 0, 0, 0, 0, 0, 0, 0
 73][0, 0, 0, 0, 0, 0, 0, 0, 0, 0][2048, 76][0, 0, 0, 0, 0, 0, 0
```

Abb. 3.6: Messdaten unter „lena_tmp“

Quelle: eigene Abbildung

3.4.1 Direkte Analyse der Messdaten

Es gibt drei verschiedene Gruppen in diesen Messdaten. Jede Gruppe steht in einer eckigen Klammer und repräsentiert unterschiedliche Bedeutungen. Die erste Gruppe ist z. B. [268437504, 24]. Die zweite Gruppe ist z. B. [2048, 3] und die dritte Gruppe ist [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]. Diese drei Gruppen laufen kontinuierlich und azyklisch ab. Die Gruppe [2048, Y] und [X, X, X, X, X, X, X, X, X, X] laufen fast ständig ab (fünf Mal in einer Sekunde) und die Gruppe [268437504, Y] läuft selten ab (fünf bis zehn Sekunden einmal). „Y“ steigt immer bis 65535(2^{16}) als Ende einer Periode, danach kommt sie auf 0 zurück. Hier wird deutlich, dass die Gruppe [2048, Y] von dem Funktionscode 65 „Zustände zyklisch holen“ passt und die Gruppe [268437504, Y] passt vom Funktionscode 0 „Timestamp“. Die Gruppe [X, X, X, X, X, X, X, X, X, X] von dem Funktionscode 66 ist der aktuelle Messwert aus dem LENA-Gerät.

3.4.2 Begründung der Analyse in der existierenden Software

An Ende des Python-Programmes „appurtenance.py“ kann man deutlich sehen, dass alle wichtigen Funktionscodes, Funktionen und die Bedeutung der jeweiligen Messwerte geschrieben worden sind.

Teil der Pythoncode „appurtenance.py“

…

```
    if fktNr == 0:
        return "timestamp = {0!r}\n".format(payload)
    if fktNr == 65:
        return "status_code = {0!r}\n" \
               "update_counter = {1!r}\n".format(payload[0], payload[1])
    if fktNr == 66:
        return "active_power = {0!r}\n" \
               "apparent_power = {1!r}\n" \
               "reactive_power = {2!r}\n" \
               "cosine_phi = {3!r}\n" \
               "star_point_voltage_L1N = {4!r}\n" \
               "star_point_voltage_L2N = {5!r}\n" \
               "star_point_voltage_L3N = {6!r}\n" \
               "conductor_current_L1 = {7!r}\n" \
               "conductor_current_L2 = {8!r}\n" \
               "conductor_current_L3 = {9!r}\n".format(payload[0], payload[1], payload[2], payload[3],
                                                       payload[4], payload[5], payload[6], payload[7],
                                                       payload[8], payload[9])
    if fktNr == 71:
```

```
        return "is_ip = {0!r}\n" \
               "is_network = {1!r}\n".format(int(payload[0]), int(payload[1]))
```

…

Im Funktionscode 66 gibt es zehn Typen von Messwerten. Das sind Werkleistung P, Scheinleistung S, Blindleistung Q, Cosphi, Sternpunktspannung L1-N, Sternpunktspannung L2-N, Sternpunktspannung L3-N, Leiterstrom L1, Leiterstrom L2 und Leiterstrom L3.

Deutsche Bezeichnung	Englische Bezeichnung	Erklärung
Werkleistung P	active_power	Aktuelle Wirkleistung in Watt
Scheinleistung S	apparent_power	Aktuelle Scheinleistung in VA
Blindleistung Q	reactive_power	Aktuelle Blindleistung in var
Cosphi	cosine_phi	Aktueller CosPhi
Sternpunktspannung L1-N	star_point_voltage_L1N	Aktuelle Spannung L1-N in ganzen V
Sternpunktspannung L2-N	star_point_voltage_L2N	Aktuelle Spannung L2-N in ganzen V
Sternpunktspannung L3-N	star_point_voltage_L3N	Aktuelle Spannung L3-N in ganzen V
Leiterstrom L1	conductor_current_L1	Aktueller Leiterstrom L1 in mA
Leiterstrom L2	conductor_current_L2	Aktueller Leiterstrom L2 in mA
Leiterstrom L3	conductor_current_L3	Aktueller Leiterstrom L3 in mA

Tabelle 9: Typen der Messwerte aus dem Funktionscode 66

Quelle: eigene Tabelle

Die aktuellen Messdaten im Dokument „lena_output“, die im Python geschrieben werden, kommen aus dem LENA-Messgerät und die Formel ist gleich wie die im Python-Programm „appurtenance.py“.

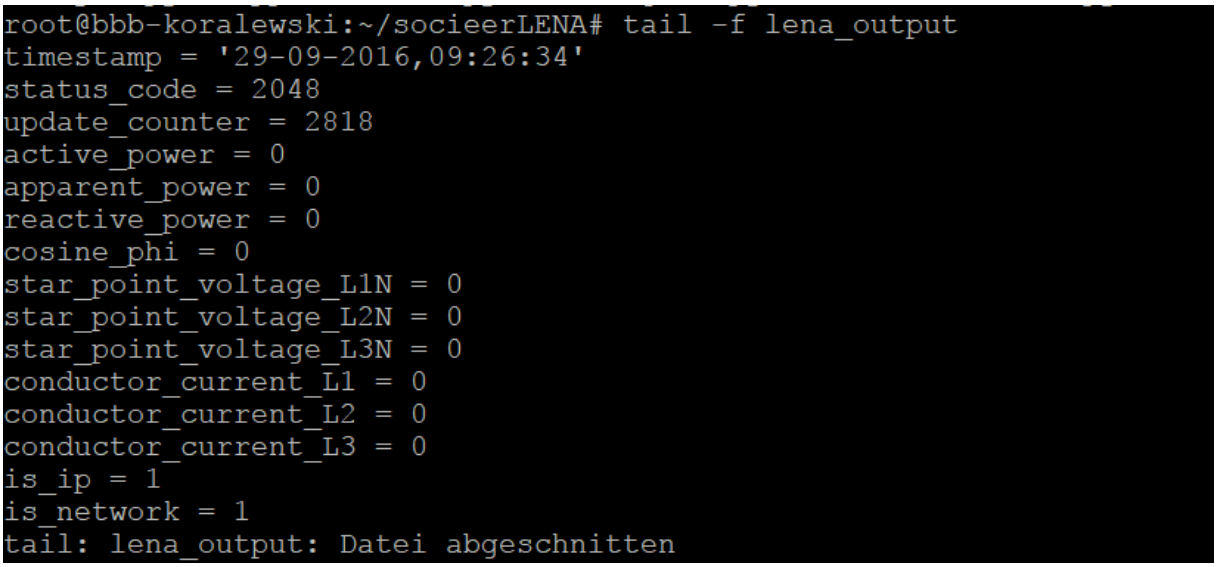


Abb. 3.7: lena_output: - Aktuelle Messdaten aus dem LENA-Messgerät

Quelle: eigene Abbildung

4 Visualisierung von Energieverbräuchen als Web-Oberfläche

Die in den zuvor behandelten Kapiteln umfassten die Analyse der existierenden Messdaten und die Datenstruktur im Messgerät. Das Ziel der nächsten Aufgabe ist die Visualisierung von Energieverbräuchen im Messgerät als Web-Oberfläche. Um das Ziel erreichen zu können, musste zuerst das Smart-Meter-System (Volkszähler) in dem Server installiert und dann die Übertragungsschnittstelle zwischen dem Beaglebone Black und dem Volkszähler aufgebaut werden.

4.1 Überblick über das Smart-Meter-System (Volkszähler)

Volkszähler ist ein freies Smart Meter (hier: intelligenter Stromzähler) im Selbstbau. Alle anfallenden Daten bleiben dabei unter der Kontrolle des Nutzers.

4.1.1 Was ist ein Volkszähler?

Volkszähler ist ein freier intelligenter Stromzähler im Selbstbau, bei dem die anfallenden Stromprofile unter der Kontrolle des Nutzers verbleiben. Die Daten des Volkszählers sind nicht durch den Versorger auslesbar. Mit einem Materialeinsatz von ca. EUR 100, etwas Geschick und Zeit lässt sich ein solcher Volkszähler auf Basis eines Standard-µC-Moduls aufbauen.

Abb. 4.1: Startseite des Volkszählers

Quelle: http://volkszaehler.org/

4.1.2 Warum benötigt man einen Volkszähler?

Wer seinen Energiebedarf analysieren möchte, braucht dazu genaue Messwerte. Bei heutigen Hausinstallationen mit konventionellen Drehstromzählern sind für den Stromverbrauch diese Messwerte nicht vorhanden. Es wird also ein intelligenter Zähler benötigt, der in der Lage ist, den Energiebedarf über sehr kurze Zeiträume zu messen und zu speichern.

4.1.3 Woraus besteht ein Volkszähler?

Der Volkszähler besteht aus vier Modulen: Messen, Übertragen, Speichern und Auswerten. Für die meisten dieser Module gibt es verschiedene Möglichkeiten der Umsetzung. Die wichtigsten Varianten in diesem Projekt sind in nachfolgender Darstellung zu finden.

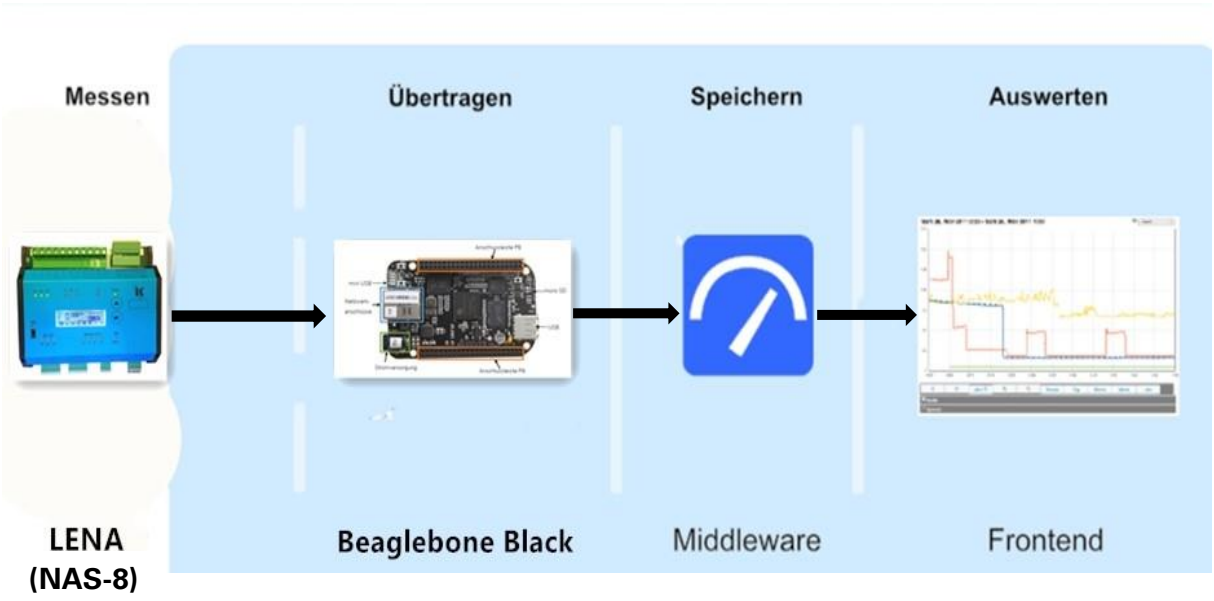


Abb. 4.2: Vier Modulen des Volkzählers dieses Projektes

Quelle: https://wiki.volkszaehler.org/overview

- **Messen:**

Das erste Modul des Volkszählers ist Messen. In diesem Modul brauchen wir natürlich einem Messgerät, um die Messwerten zu erhalten und hier spielt LENA eine wichtige Rolle.

- **Übertragen:**

Die nächste Komponente erfasst die Daten vom Messgerät und reicht sie zur Speicherung und Übertragung weiter und hier braucht einen Mikrocontroller nämlich Beaglebone Black.

- **Speichern:**

Zentraler Bestandteil des Volkszählers ist die Middleware, dort werden die Daten gespeichert und ausgewertet. Die Middleware muss im Server installiert werden und die IP-Adresse des Servers im Labor ist 192.168.10.113.

- **Auswerten:**

Letztlich sollen die gemessenen Daten in einer für uns angenehmen Weise aufbereitet angezeigt werden. Dazu wird ein Frontend verwendet. Der Volkszähler hat ein Standard-Frontend, das die Messdaten in einem hübschen Graphen anzeigen kann und erlaubt, Zeitraum und Auflösung der Darstellung frei zu wählen. Die Messdaten aus dem Messgerät(NAS-8) werden hier dargestellt.

4.2 Aufbau Schnittstelle Beaglebone Black/Smart-Meter-System

4.2.1 Analyse Datenfluss vom Beaglebone Black bis Smart-Meter-System

Im Kapitel 3 konnte man erfahren, dass es im Funktionscode 66 zehn Typen von Messwerten aus dem Messgerät gibt. Um den Leistungsverbrauch erhalten zu können, musste der Output der sieben Typen von Messwerten (Werkleistung P, Sternpunktspannung L1-N, Sternpunktspannung L2-N, Sternpunktspannung L3-N, Leiterstrom L1, Leiterstrom L2 und Leiterstrom L3) und Timestamp durchgeführt werden. Der Prozess der allgemeinen Module des Datenflusses ist in nachfolgender Abbildung dargestellt:

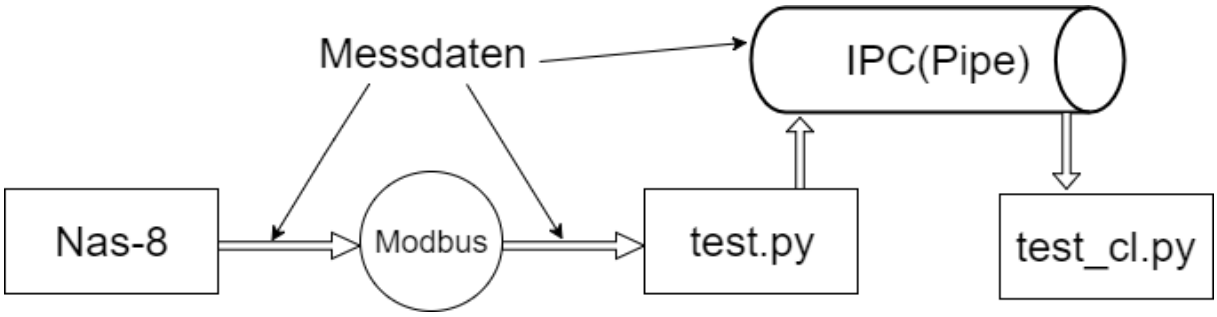


Abb. 4.3: Prozess der allgemeinen Module des Datenflusses

Quelle: eigene Abbildung

Das Messgerät (NAS-8) misst zuerst die Messdaten, die durch den Modbus zum Programm „test.py“ gesendet werden. Am Ende werden diese Messdaten an das Programm „test_cl.py“ übertragen.

Die Programme „test.py“ und „test_cl.py“ sind im Prozess des Beaglebone Blacks automatisch geschrieben worden. Wenn man am Anfang das Beaglebone Black anschließt, laufen diese beiden Programme gleichzeitig automatisch ab. Hier kann man in der Linux-Konsole ein Kommando „ps aux“ eingeben, dann erscheinen alle laufenden Prozesse im Beaglebone Black und die Programme „test.py“ und „test_cl.py“ sind dazwischen. Das bedeutet, dass der Prozess vom allgemeinen Modul des Datenflusses immer läuft und zwar vom Anschluss bis zur Ausschaltung des Beaglebone Blacks.

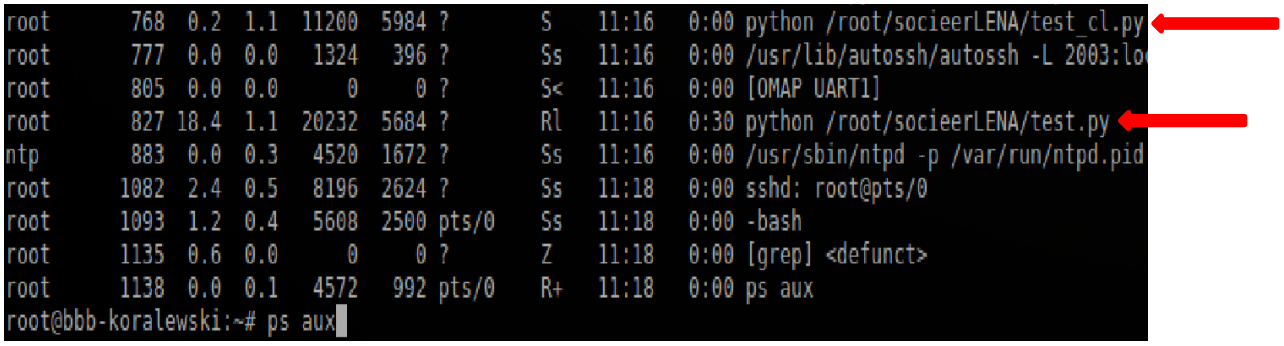


Abb. 4.4: Laufende Prozesse unter dem Anschluss des Beaglebone Blacks

Quelle: eigene Abbildung

4.2.2 Versuch 1 (Entnahme der Messdaten direkt aus „lena_output“)

In den zuvor behandelten Abschnitten hat man bereits erfahren, dass die aktuellen Messdaten im Dokument „lena_output“, welche im Python geschrieben werden, aus dem LENA-Messgerät kommen. Das bedeutet mit einfachen Worten, dass die Messdaten direkt aus dem Dokument „lena_output“ genommen werden können.
Es folgt nun eine Abbildung von dem Prozess des ersten Versuches:

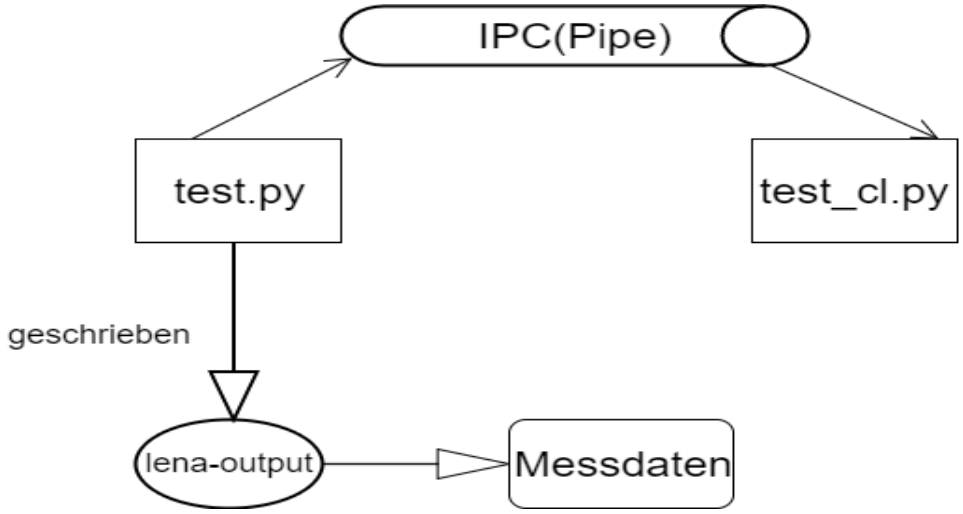


Abb. 4.5: Entnahme der Messdaten direkt aus „lena_output“

Quelle: eigene Abbildung

Jetzt wurde ein Versuch anhand einer Idee durchgeführt. Es sollte zuerst nur ein Typ der Messwerte ausprobiert werden, deshalb wurde der Typ „Sternpunktspannung L1-N“ ausgewählt. Danach wurden zwei Kabel für den Anschluss zwischen dem Messgerät (NAS-8) und der Spannungsquelle ausgesucht und die Spannungsquelle auf 20 Volt eingestellt. Schließlich konnte man im LCD des Messgerätes deutlich sehen, dass hier „Sternpunktspannung L1-N“ 20 Volt schon angezeigt war.

Abb. 4.6: Messwert „Sternpunktspannung L1-N“

Quelle: eigene Abbildung

Wenn jetzt das Dokument „lena_output“ lief, konnte man auch hier „star_point_voltage_L1N“ 20V sehen. Das bedeutet, dass der Messwert von dem Beaglebone Black gleichzeitig gelesen wurde und “Sternpunktspannung L1-N“ schon in das Dokument „lena_output“ des Programms „test.py“ eingeschrieben war. Dies ist in folgender Abbildung zu sehen:

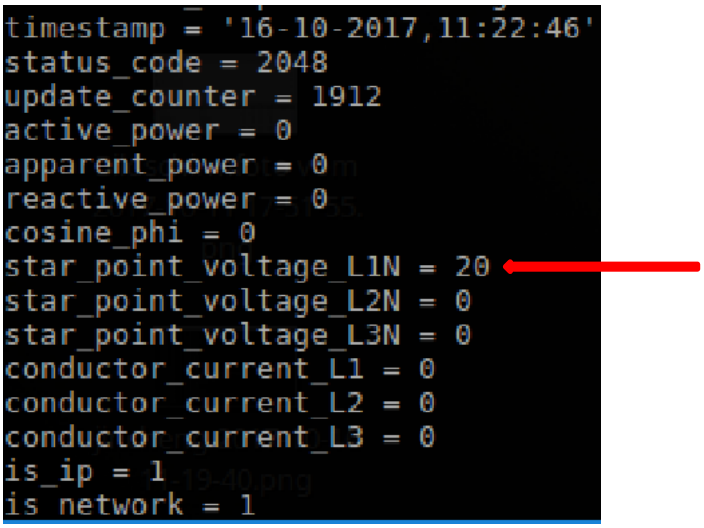


Abb. 4.7: Anzeige des Messwertes „star_point_voltage_L1N“ im Dokument „lena_output“

Quelle: eigene Abbildung

Aber das Ziel besteht darin, die Anzeige des Messwertes am Smart-Meter-System (Volkszähler) darzustellen. Deshalb muss ein Python-Programm geschrieben werden, um diese Funktion zu realisieren.

Nachfolgend ist ein kleines Python-Programm „Regulärer Ausdruck.py“ aufgeführt:

```
import time
import urllib2     [1]
import os
import re          [2]

delay = 2

t = 0
while True:
try:
   with open('/root/socieerLENA/lena_output','r') as f:     [3]
      data = f.read()
      time.sleep(delay)
      print data
      num = re.search(r'(?<=star_point_voltage_L1N = )\d+',data)     [4]
      print num
      if not num:
         print int(num)
      else:
         print int(num.group(0))
except Exception as e:
   print(e)

   V = int(num.group(0))
   print int(num.group(0))
   uuid = "695bb500-7537-11e7-a8d9-63bc96e02c17"     [5]
   timestamp = time.time()*1000       [6]
   ts = timestamp
   path = 'http://192.168.10.113/middleware.php/data/%s.json?ts=%d&value=%d' % (uuid,
ts, V)       [7]
   req = urllib2.Request(url = path, data = '')
```

```
f = urllib2.urlopen(req)    [8]
t = t + 1
time.sleep(delay)
```

Erläuterung des Python-Code:

[1]: Einführen der Bibliothek für die URL (Uniform Resource Locator)

[2]: Einführen der Bibliothek für „Regulärer Ausdruck“

[3]: Öffnen des Dokuments „lena_output“ unter „/root/socieerLENA/“

[4]: Nutzung des „Regulären Ausdruckes“, um Messwert (20 Volt) „Sternpunktspannung L1-N“ zu erhalten

[5]: Einzige uuid (Kanaladdresse) der „Sternpunktspannung L1-N“ aus dem Volkszähler

[6]: Aktueller Zeitpunkt

[7]: Regel der URL (uuid, Timestamp, Zahl)

[8]: Senden der URL an die Middleware des Servers des Volkszählers

Nun lief das Python-Programm „Regulärer Ausdruck.py“. Aber das erste bis fünfte Mal wurden die Messwerte aus dem „Regulären Ausdruck“ gar nicht angezeigt und „None“ sowie die Grafik am Volkszähler waren auch gar nicht dargestellt. Aber das sechste Mal wurde ein Messwert plötzlich aus dem „Regulären Ausdruck“ genommen und 20 Volt waren zu einem Zeitpunkt im Volkszähler dargestellt. Aber weitere Messwerte gab es nicht mehr. Dann wurden die Versuche viele Male ausprobiert und die Messwerte waren manchmal dargestellt und manchmal auch überhaupt nicht.

Die angezeigten Messwerte im Volkszähler waren auch nicht kontinuierlich und nicht synchron mit der Einstellung der Strom/Spannungsquelle.

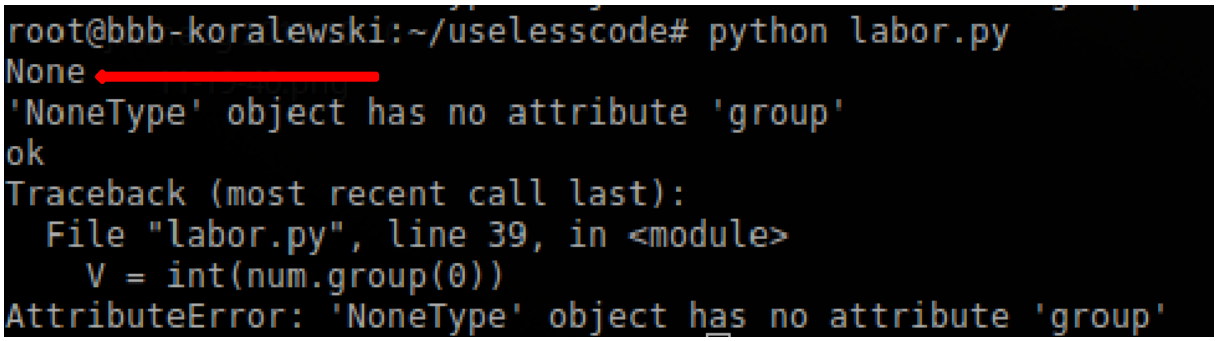


Abb. 4.8: Keine Messwerte aus dem „Regulären Ausdruck“ angezeigt

Quelle: eigene Abbildung

Für diese Situation war das geschriebene Dokument „lena_output“ verantwortlich, da es nicht immer zyklisch und stabil abgelaufen ist, und manchmal gab es eine Blockierung in diesem geschriebenen Dokument. Deshalb konnte der „Reguläre Ausdruck“ nicht immer die Messwerte bekommen.

Zum Schluss musste auf den ersten Versuch verzichtet und noch eine andere, bessere Methode herausgefunden werden. Aber aus diesem Versuch hat man viel gelernt. Das war eine sinnvolle und kostbare Erfahrung für die zukünftige Arbeit.

4.2.3 Versuch 2 (Entnahme der Messdaten aus „test_cl.py“)

Vorher wurde das Modul des Datenflusses im Beaglebone Black analysiert. Das existierende Python-Programm „test_cl.py“ fungierte als Empfänger und empfing die Messdaten immer durch den Modbus aus dem Programm „test.py“. Dieser Prozess dauerte immer vom Anschluss bis zur Ausschaltung des Beaglebone Blacks. Deshalb konnte angenommen werden, dass die Messdaten aus dem Python-Programm „test_cl.py“ entnommen und diese dann durch URL an das Smart-Meter-System (Volkszähler) geschickt werden. Folgende Abbildung stellt den Prozess des zweiten Versuches dar:

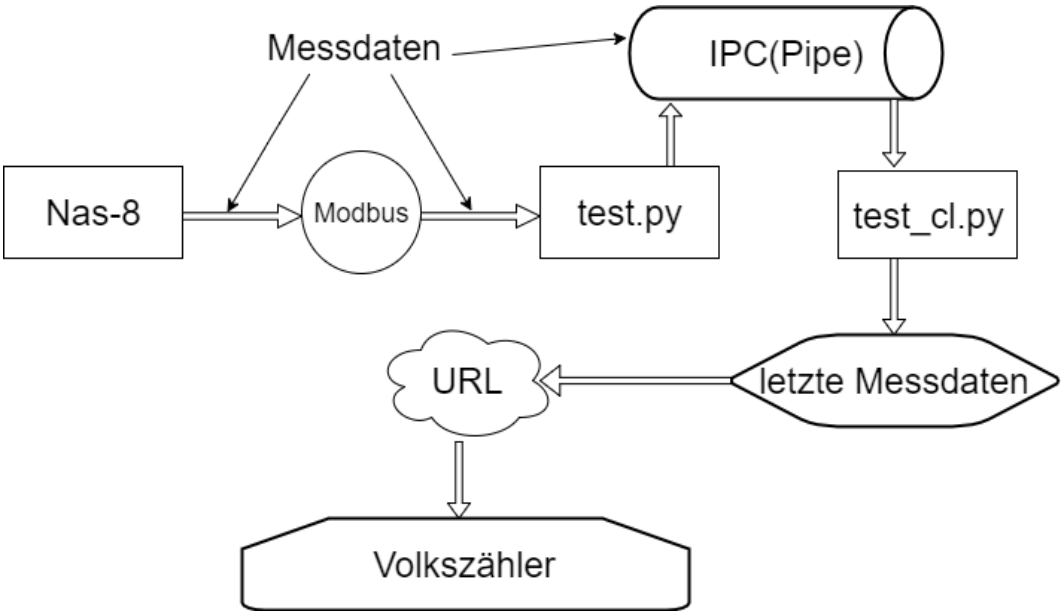


Abb. 4.9: Erhalt der Messdaten aus dem Programm „test_cl.py“

Quelle: eigene Abbildung

Jetzt erfolgte der zweite Versuch mithilfe dieser neuen Idee. Genau wie beim ersten Versuch wurde zuerst auch nur der Messwert „Sternpunktspannung L1-N“ ausprobiert und danach wurden wieder zwei Kabel für den Anschluss zwischen dem Messgerät (NAS-8) und der Spannungsquelle benötigt und diese dann auf 20 Volt eingestellt.
Der wichtigste Schwerpunkt dieses Programms „test_cl.py“ ist im Folgenden dargestellt:

```
…
def main():
    while True:
        print "reading...."
        m2 = ipc.recv()   [1]
        if m2 is not None:
            if m2.mtype == 66:
                logging.info("# get the message from ICP: {0!r}".format(m2.payload))
                if CARBON_IO == True:
                    t = 0
                    datas = m2.payload.split(";")   [2]
                    print(datas)   [3]
…
```

Anhand des Python-Codes **[1]** wurde herausgefunden, dass „m2“ aus IPC kam. In Python-Code **[2]** fand man die Gruppe der Messdaten in einer Liste. Diese waren mit einigen Semikolons getrennt. Jetzt wurden die Messdaten „datas“ **[3]** ausgedruckt, die [0; 0; 0; 0; 2; 0; 0; 0; 0; 0] angezeigt haben. Die fünfte Messzahl in der Liste „2“ war der eingestellte Spannungswert 20V der Spannungsquelle.

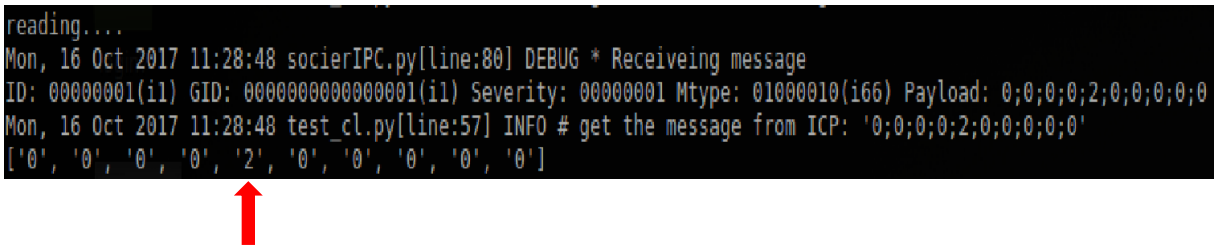


Abb. 4.10: Die Messzahl „2“ aus dem laufenden Programm „test_cl.py“

Quelle: eigene Abbildung

Nun lief das Python-Programm „test_cl.py“ an. Die Messdaten erschienen immer kontinuierlich und zyklisch. Wenn die Spannungsquelle auf 30 Volt eingestellt war, wurde sie von „2“ auf „3“ geändert.

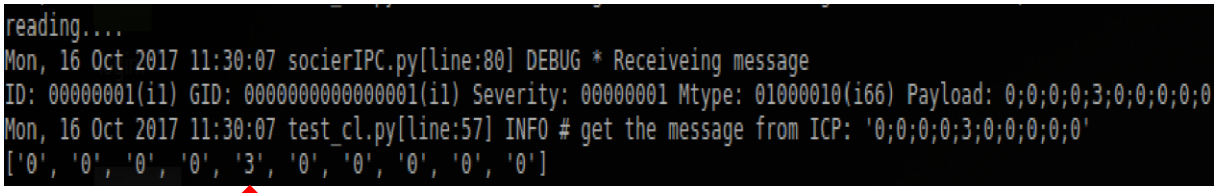


Abb. 4.11: Änderung der Messzahl von „2“ auf „3“

Quelle: eigene Abbildung

Wenn 10 Volt eingestellt wurden, änderte sich die Messzahl ebenfalls von „3“ auf „1“.

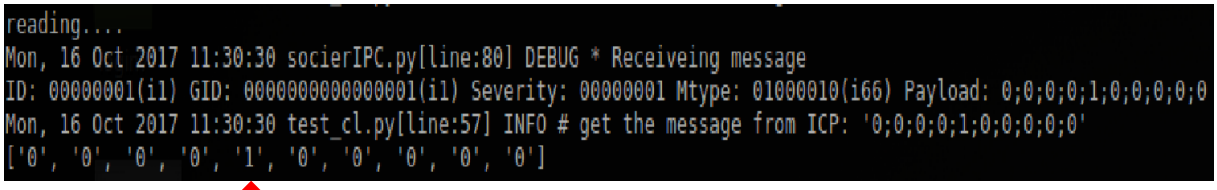


Abb. 4.12: Änderung der Messzahl von „3“ auf „1“

Quelle: eigene Abbildung

Jetzt konnte man sagen, dass die Messdaten aus dem Programm „test_cl.py“ immer synchron mit den Änderungen des Spannungswertes erschienen. Das war ein großer Erfolg!

Um die Wirkleistung zu bekommen, mussten zuerst andere 5 Typen der Messwerte (Sternpunktspannung L2-N, Sternpunktspannung L3-N, Leiterstrom L1, Leiterstrom L2 und Leiterstrom L3) geben, dann produzierte ich andere fünf Kabel für den Anschluss zwischen dem Messgerät und der Strom/Spannungsquelle.

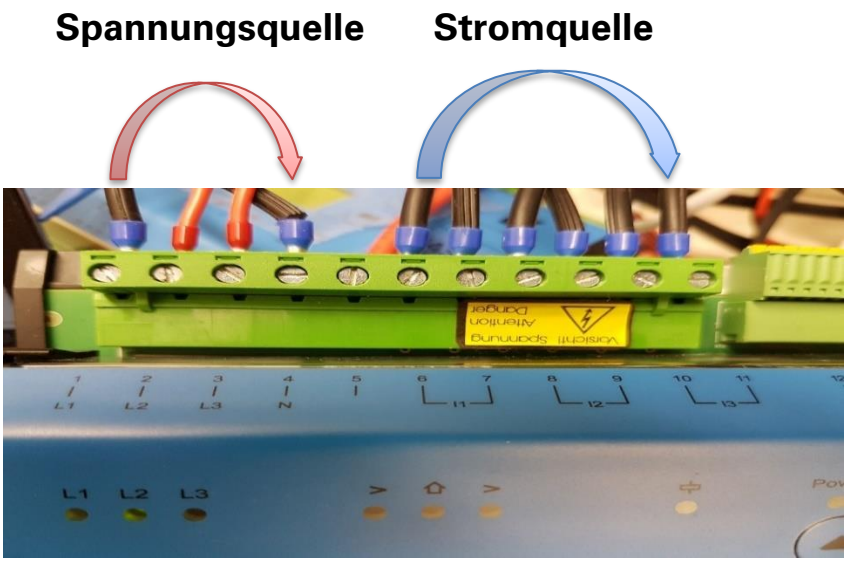


Abb. 4.13: Sechs Kabel für Anschluss zwischen Messgerät und Strom/Spannungsquelle

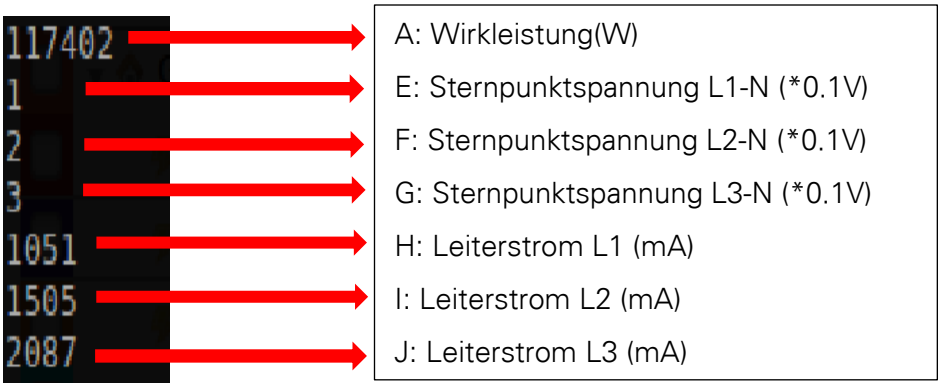
Quelle: eigene Abbildung

Nun wurden alle sieben Messwerte (Werkleistung P, Sternpunktspannung L1-N, Sternpunktspannung L2-N, Sternpunktspannung L3-N, Leiterstrom L1, Leiterstrom L2 und Leiterstrom L3) allein ausgewählt und jeweils ausgedruckt. Nachfolgend ist ein kleiner Teil des Python-Code aus „test_cl.py“ dargestellt.

```
...
datas = m2.payload.split(";")
                A = (int(datas[0]))
                B = (int(datas[1]))
                C = (int(datas[2]))
                D = (int(datas[3]))
                E = (int(datas[4]))
                F = (int(datas[5]))
                G = (int(datas[6]))
                H = (int(datas[7]))
                 I = (int(datas[8]))
                J = (int(datas[9]))
                print A
                print E
                print F
```

```
print G
print H
print I
print J
```

Dann hat man diese Messwerte erhalten:



Als nächste Aufgabe folgte die Darstellung dieser Messwerte am Smart-Meter-System (Volkszähler).

4.2.4 Darstellung der Messwerte am Smart-Meter-System (Volkszähler)

Nachdem die Middleware des Volkszählers an die Serveradresse (hier ist 192.168.10.113) installiert wurde, erschien eine Frontendwebsite wie die folgende Abbildung zeigt:

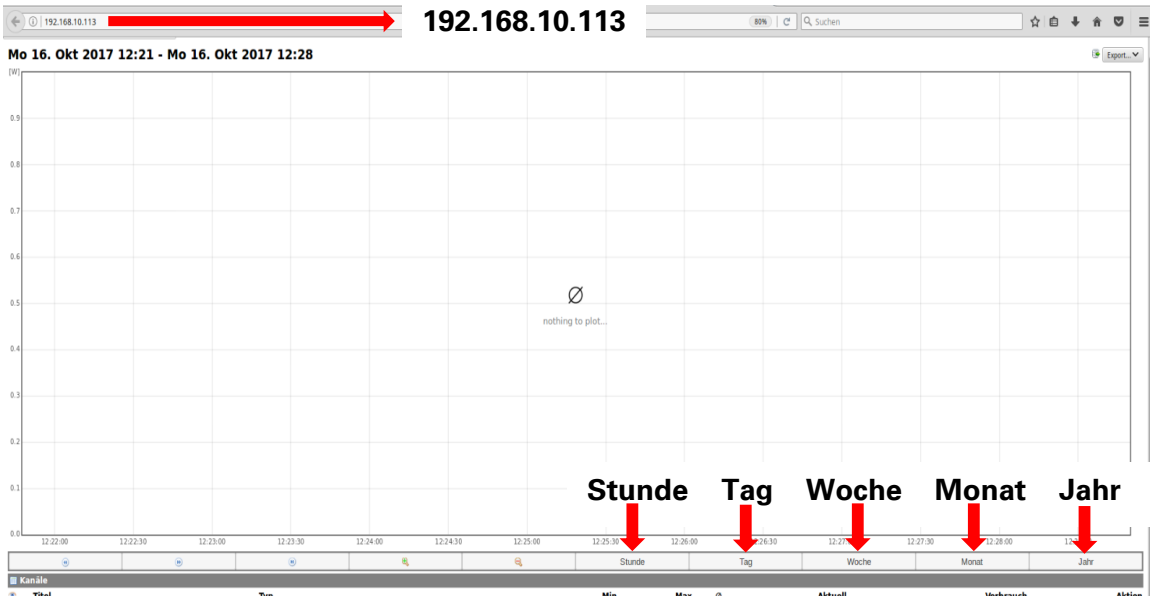


Abb. 4.14: Frontend des Volkszählers

Quelle: eigene Abbildung

Um die jeweiligen Messwerte darzustellen, musste man zuerst die Kanäle für diese Messwerte hinzufügen, dann wählte man den geeigneten Typ des Messwertes aus, z. B. „Spannungssensor“, „Stromsensor“ und „Leistungswert“. Anschließend bekam man den jeweiligen Kanal, ein „uuid“. Dieser „uuid“ fungierte als ein individueller URL für die Datenübertragung.

Folgende Abbildung zeigt den Prozess der Einstellung des Kanals:

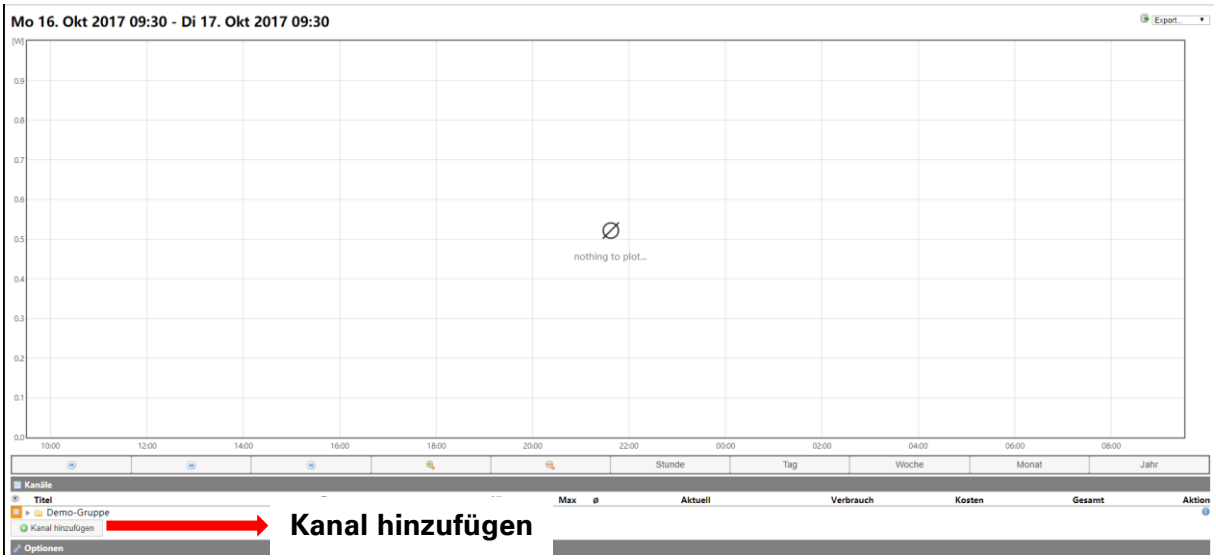


Abb. 4.15: Hinzufügen der Kanäle

Quelle: eigene Abbildung

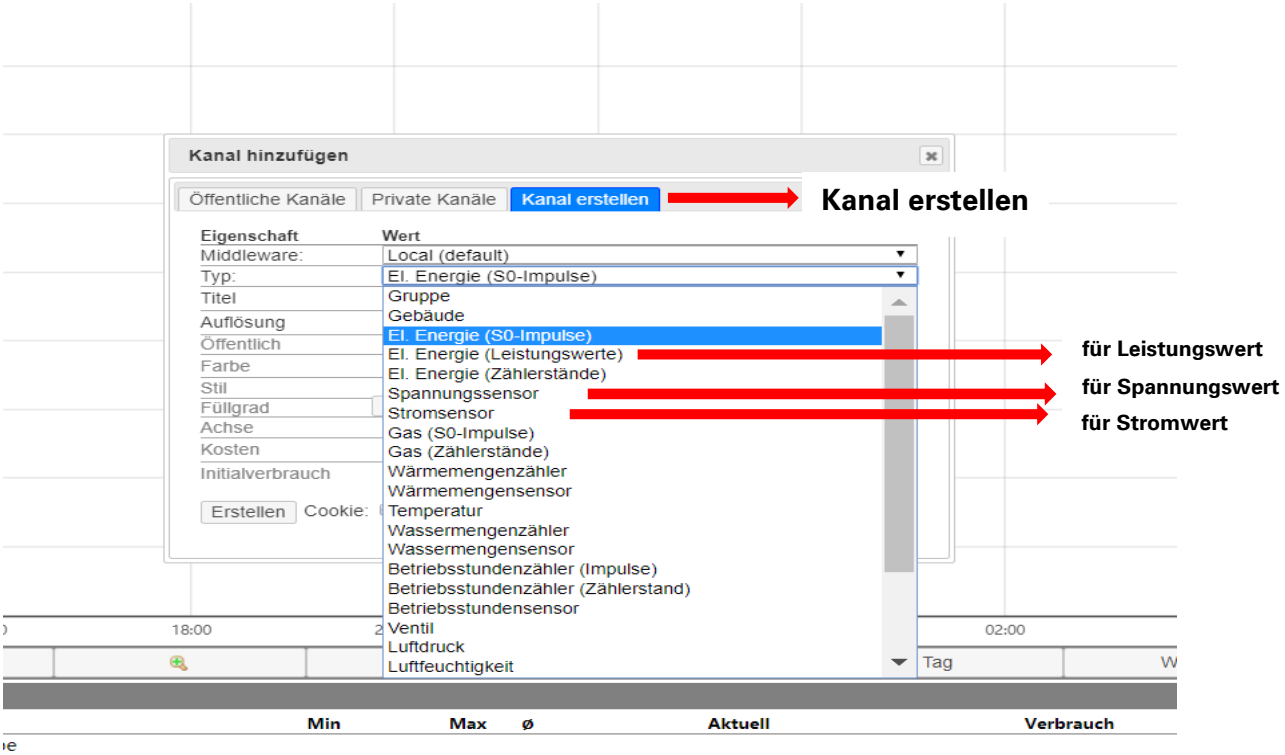


Abb. 4.16: Auswahl der unterschiedlichen Kanäle für Messwerte

Quelle: eigene Abbildung

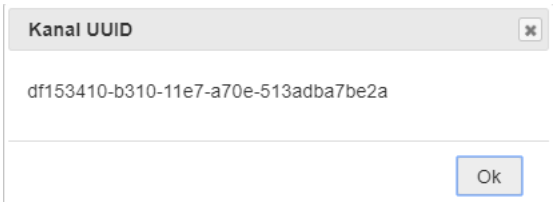


Abb. 4.17: Erhalten des individuellen Kanals „uuid“ der Messwerte

Quelle: eigene Abbildung

Wann man alle Kanäle fertig erstellt hatte, erschien eine Anzeige wie in folgender Abbildung dargestellt:

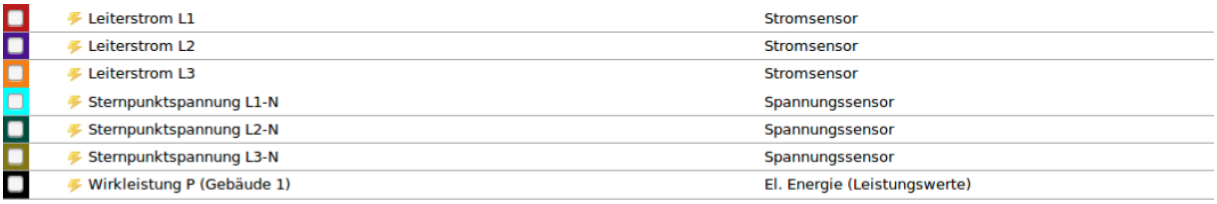


Abb. 4.18: Alle Kanäle der Messwerte aus dem Messgerät

Quelle: eigene Abbildung

Jetzt konnte dieser „uuid“ in das Python-Programm „test_cl.py“ hinzugefügt werden, um die sieben Typen der Messwerte an den Volkszähler zu übertragen.
Der folgende kleine Python-Code ist die Übertragung des Messwertes „Leistungswert“ aus dem Programm **„test_cl.py“**. Die anderen Messwerte sind ähnlich wie dieser Teil:

```
uuid0 = "4a2b4000-a8d1-11e7-842a-f9c408836b1e" [1]

timestamp = time.time()*1000   [2]

ts0 = timestamp

path  =  'http://192.168.10.113/middleware.php/data/%s.json?ts=%d&value=%d'  %  (uuid0,

ts0, A) [3]

req = urllib2.Request(url = path, data = '')
```

```
f = urllib2.urlopen(req) [4]
```

Erklärung des Codes:

[1]: Individueller „uuid“ des Messwertes „Leistungswert“ aus dem Volkszähler

[2]: Erhalt des aktuellen Timestamp

[3]: Erhalt des individuellen URL

[4]: Senden des URL und der Messwerte an den Server (192.168.10.113)

Nun lief das Programm „test_cl.py“, und es gab sieben Linien in der Graphik. Diese Linien liefen automatisch mit der aktuellen Zeit. Am wichtigsten war es, dass der Leistungsverbrauch auch hier automatisch berechnet wurde. Das war ganz prima!
Folgende Abbildung ist eine Darstellung der konkreten Situation:

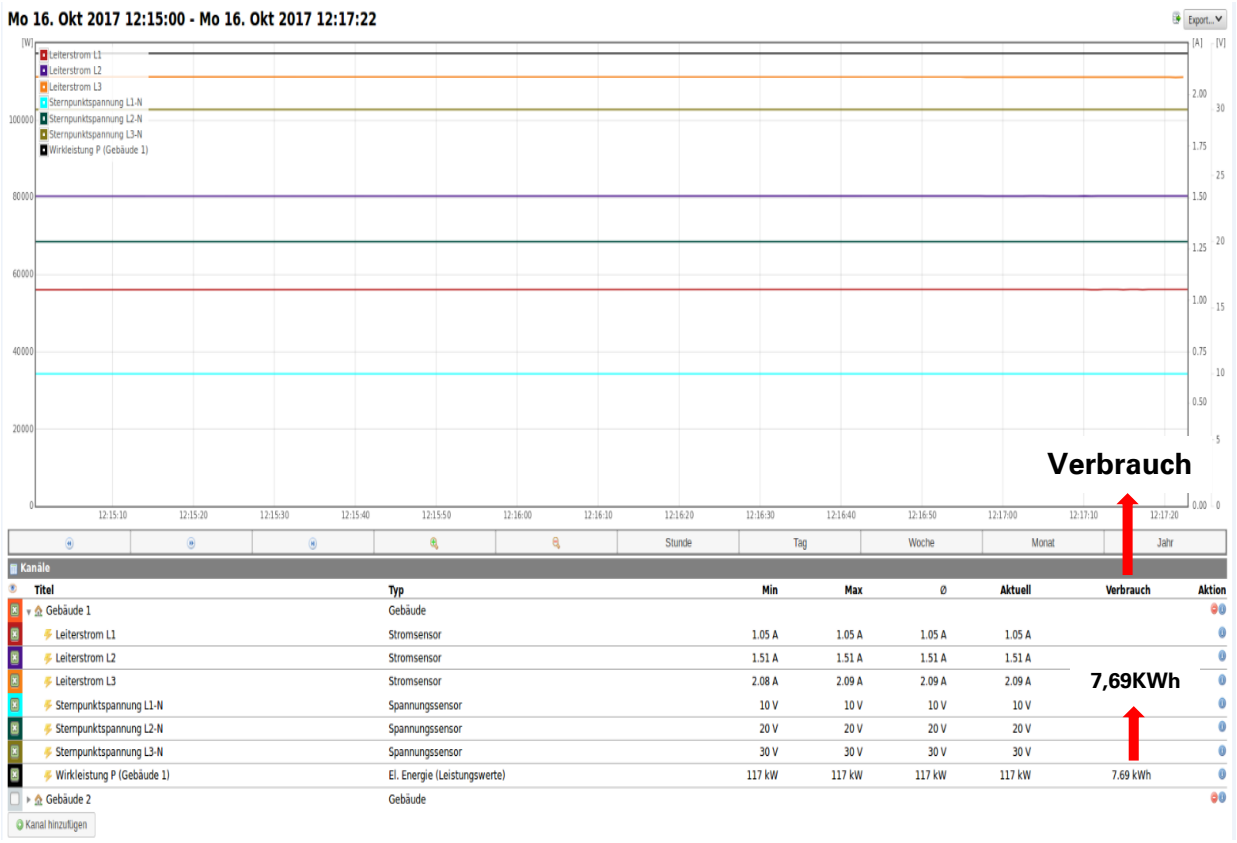


Abb. 4.19: Darstellung aller Messwerte und berechneter Verbrauch an den Volkszähler

Quelle: eigene Abbildung

5 Authentifikation der Benutzer in der Web-Oberfläche

5.1 Zielsetzung der Authentifikation

Das Messgerät (NAS-8) hat die Aufgabe, den Leistungsverbrauch in einem Gebäude für einen Benutzer zu messen. Für die Messung in mehreren Gebäuden für mehrere Benutzer müssen natürlich mehrere Messgeräte (NAS-8) geliefert werden. Wenn man den Leistungsverbrauch seines Gebäudes wissen möchte, kann man sich in einer Anmeldungsseite selbst einloggen und die Situation seines Leistungsverbrauches prüfen. Das endgültige Ziel des Bachelorprojektes ist die Realisierung der Authentifikation für verschiedene Benutzer in der Web-Oberfläche.

Folgende Abbildung stellt die Zielsetzung der Authentifikation für verschiedene Benutzer dar.

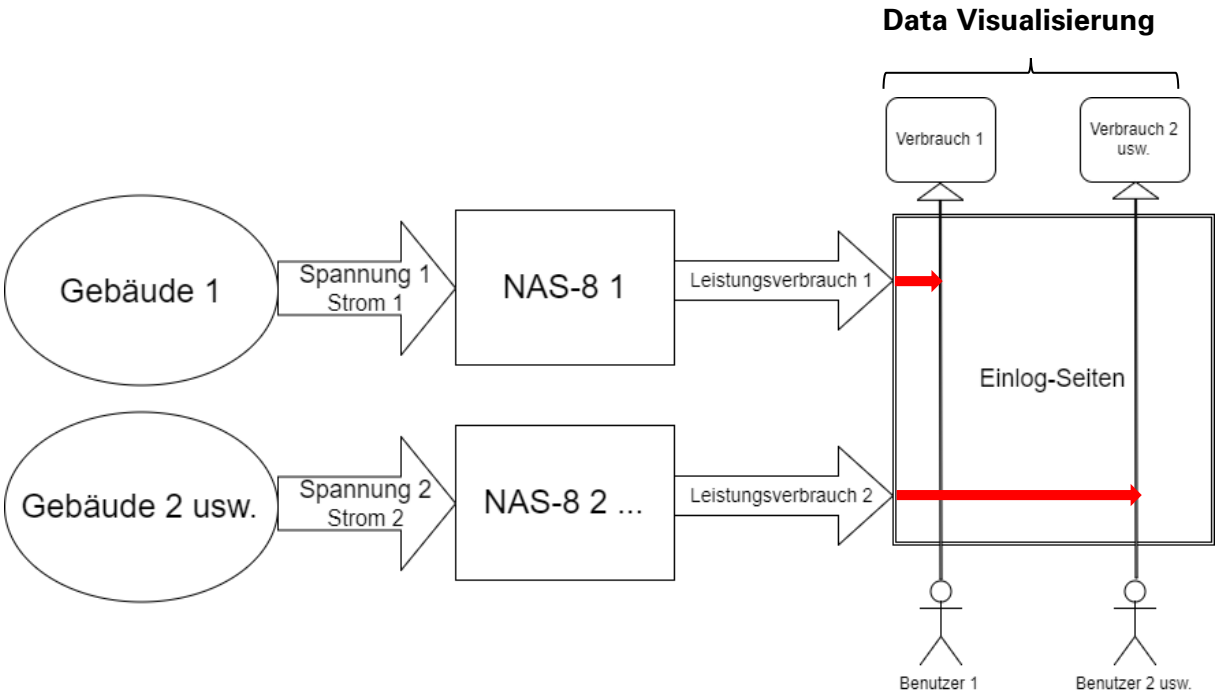


Abb. 5.1: Zielsetzung der Authentifikation für verschiedene Benutzer

Quelle: eigene Abbildung

5.2 Realisierung der Webseite der Authentifikation für die Benutzer

Um mit dem Versuch zu beginnen, musste zuerst der Anschluss zwischen dem zweiten Messgerät und den Spannungsquellen bzw. Stromquellen realisiert werden. Das erste Messgerät maß für das Gebäude 1 und das zweite für das Gebäude 2, wie folgende Abbildung zeigt:

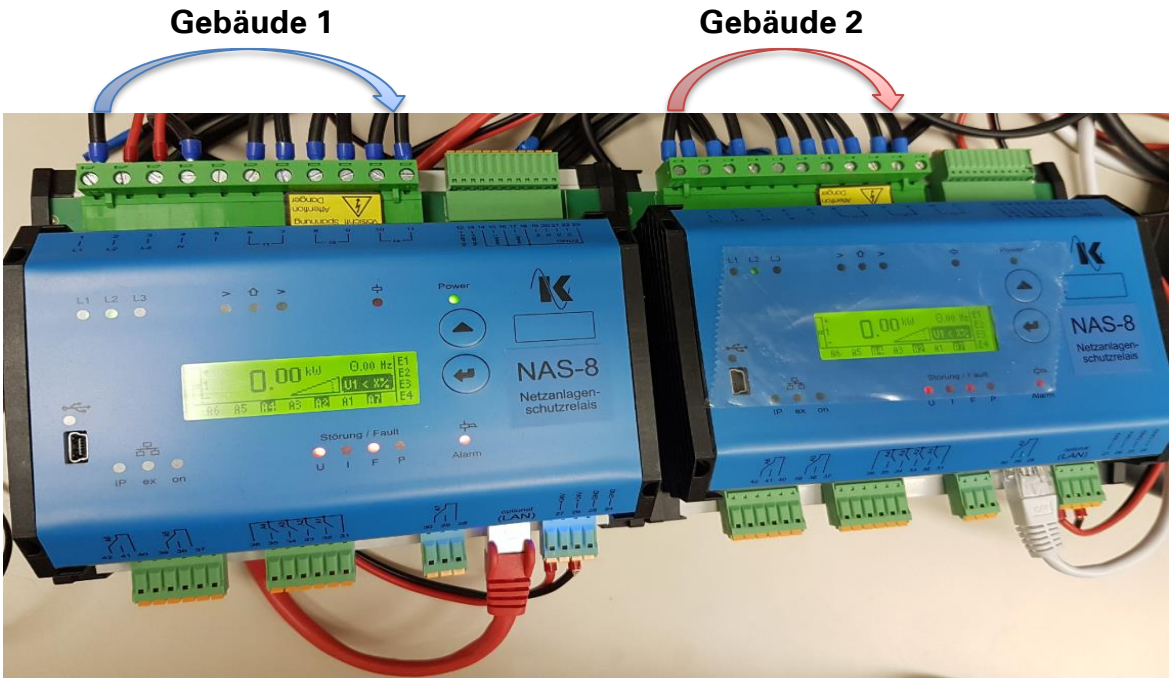


Abb. 5.2: Zwei Messgeräte für die Messungen von zwei Gebäuden

Quelle: eigene Abbildung

Dann wurde auch die Schnittstelle des Datenflusses zwischen dem zweiten Messgerät und dem Volkszähler aufgebaut. Der Prozess der Produktion war ganz genauso wie bei dem ersten Messgerät. Als alles fertiggestellt war, konnten die verschiedenen Messwerte auch synchron im Frontend des Volkszählers dargestellt werden. Die konkrete Situation ist in der folgenden Abbildung dargestellt:

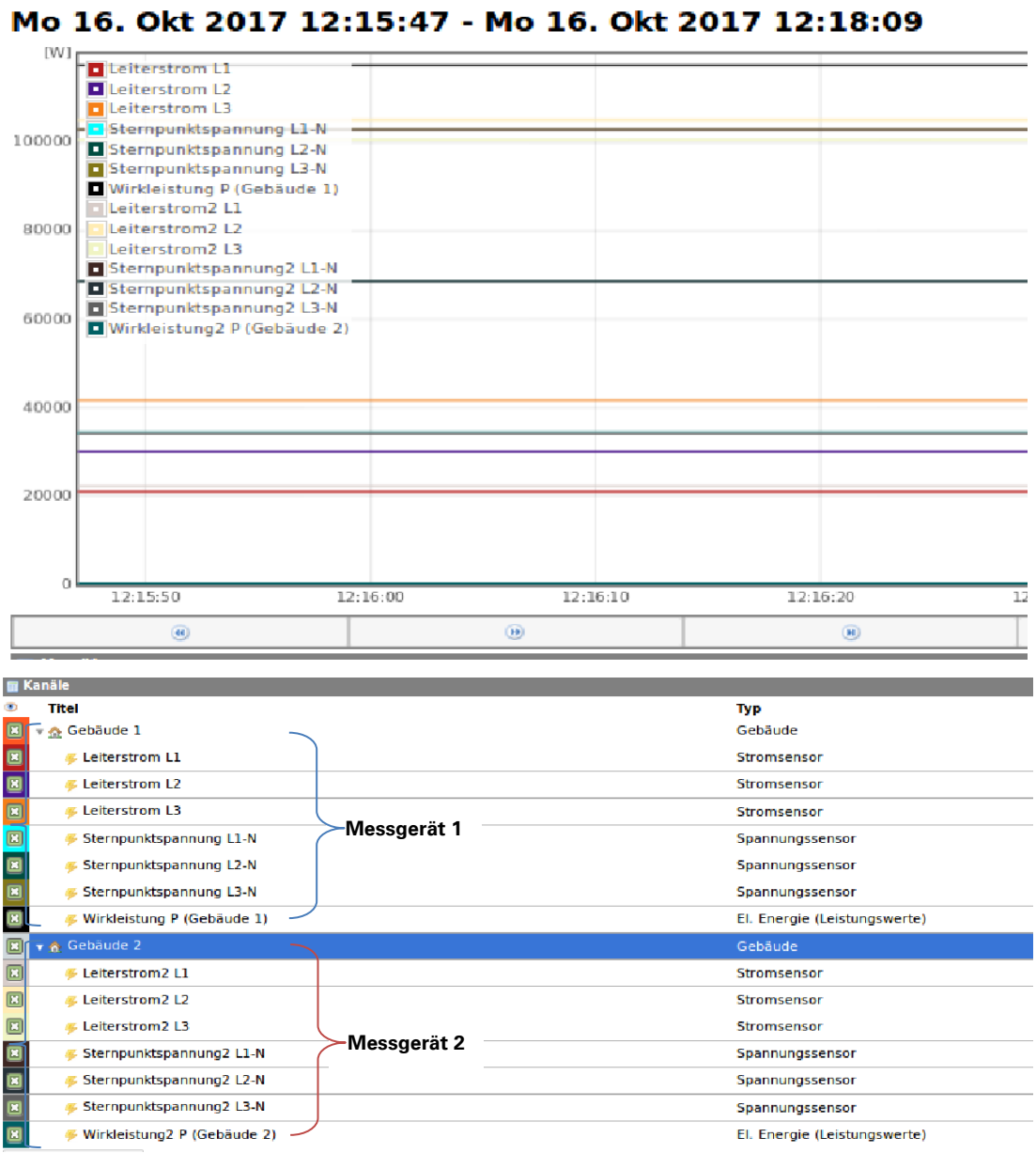


Abb. 5.3: Darstellung aller Messwerte im Frontend des Volkszählers

Quelle: eigene Abbildung

Es gibt die Website „Data Visualisierung“. Wenn man hier den „uuid“ und „Middleware des Servers“ des Leistungswertes eingibt, können die Messwerte und der Leistungsverbrauch in sechs Stunden, einem Tag, einer Woche, einem Monat und drei Monaten dargestellt werden. Die Quelle der Website und die Darstellung des Leistungswertes sind in nachfolgender Abbildungen dargestellt:

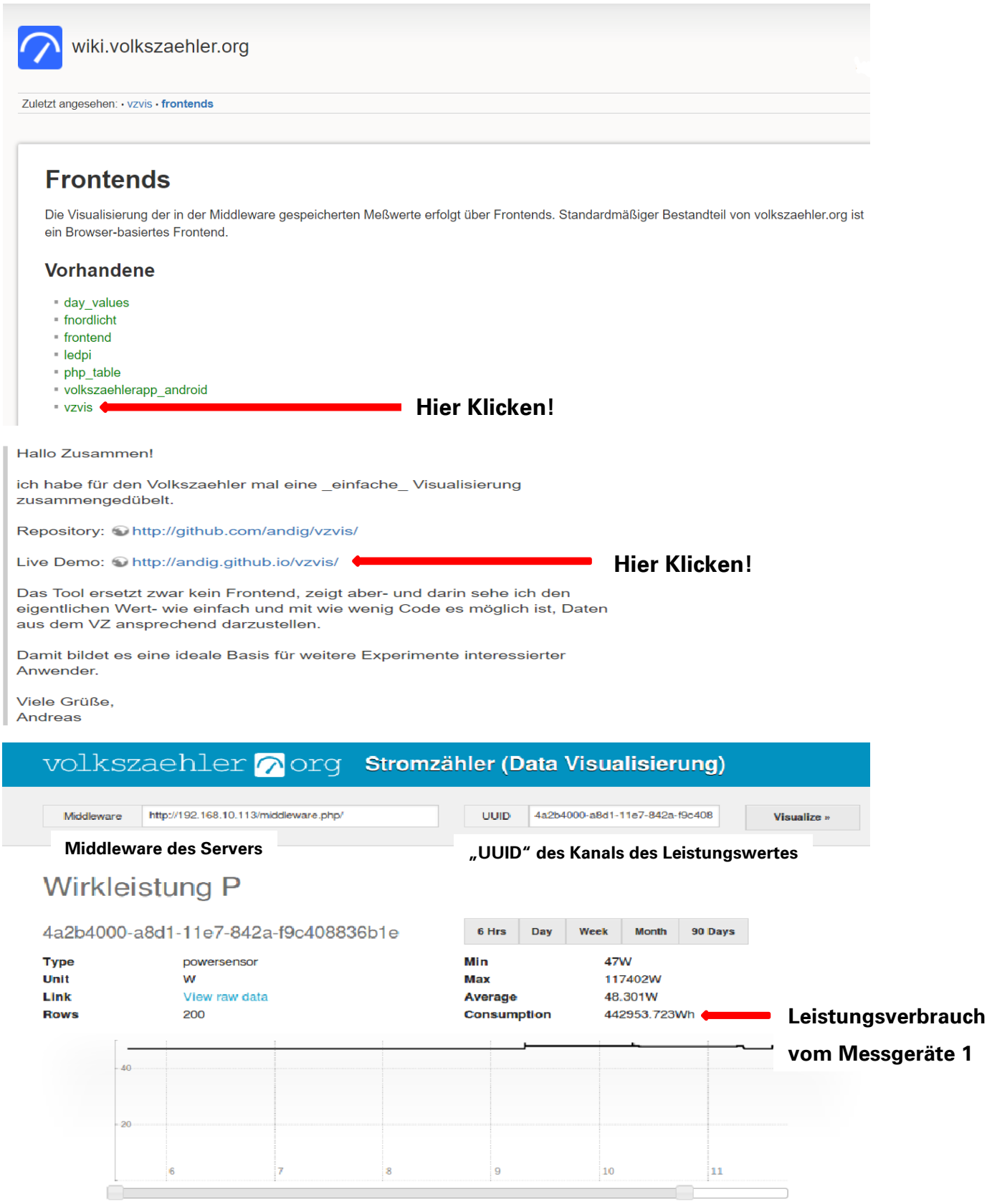


Abb. 5.4: Quelle der Website und Darstellung des Leistungswertes

Quelle: https://wiki.volkszaehler.org/software/frontends/vzvis

Jetzt konnten zwei Leistungsverbräuche von den beiden Messgeräten sowie die Quelle der Website und die Darstellung des Leistungswertes (Siehe obere Abbildung) erhalten werden. Die nächste Aufgabe bestand in der Produktion der Anmeldungsseite. Um eine Anmeldungsseite zu realisieren, musste man zuerst eine Tabelle in der Datenbank erstellen. In diesem Projekt wurde eine Tabelle „users“ in der Datenbank „volkszaehler“ erstellt, um die Registerinformationen der Benutzer zu ergänzen. Die Tabelle wurde in sieben Spalten untergliedert, nämlich „id“, „first_name“, „last_name“, „email“, „password“, „hash“ und „active“.

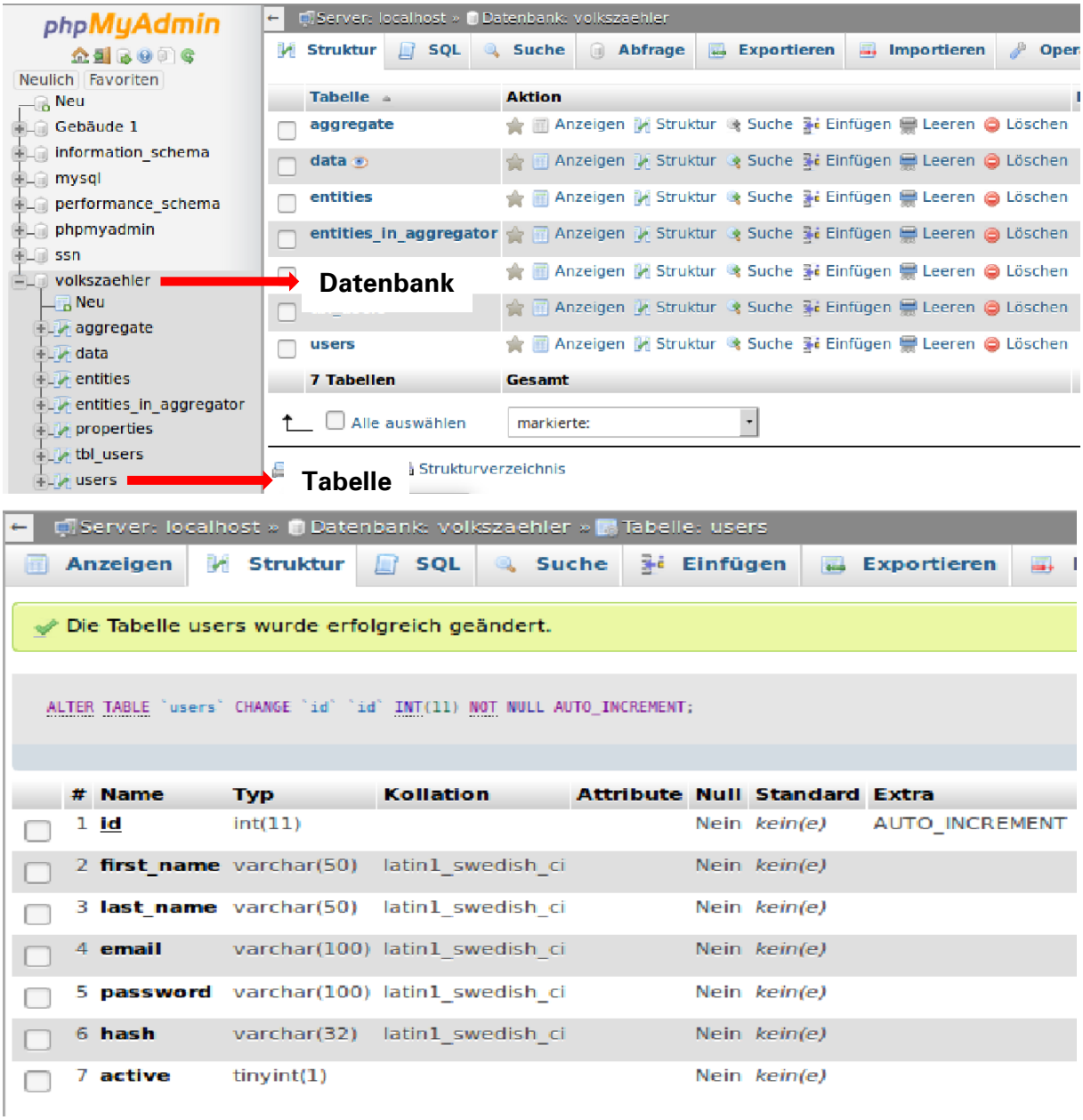


Abb. 5.5: Tabelle der Datenbank „volkszaehler“ und die Struktur der Tabelle „users“

Quelle: eigene Abbildung

Dann können einige php-Programme unter dem Server des Volkszählers (/var/www/volkszaehler.org/htdocs/login7/) geschrieben werden. Folgende Abbildung stellt den allgemeinen Prozess der Authentifikation des Benutzers dar.

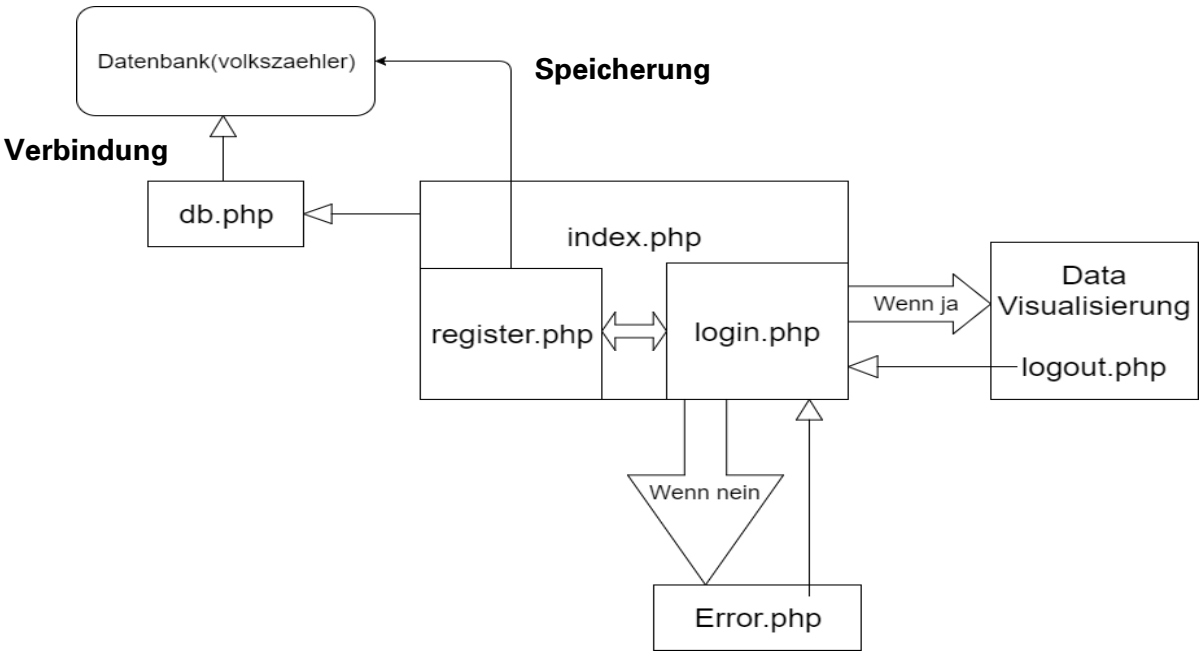


Abb. 5.6: Prozess der Authentifikation des Benutzers

Quelle: eigene Abbildung

Zuerst entstand das Programm „db.php“ für die Verbindung der Datenbank (volkszaehler). Das nachfolgende Programm hieß „db.php“:

```
<?php
/* Database connection settings */
$host = 'localhost'; // Datenbank gehört zum Lokalhost
$user = 'root';  // Nutzername der Website (phpmyadmin)
$pass = 'root';  // Passwort dsr Nutzernamen
$db = 'volkszaehler'; // Datenbank der Benutzer
$mysqli = new mysqli($host,$user,$pass,$db) or die($mysqli->error);
```

Dann wurde das Programm „register.php“ für das Register des Benutzers geschrieben. Wenn man hier seinen Vornamen, Nachnamen, die E-Mail-Adresse und das Passwort einschrieb, waren diese Informationen automatisch in der Tabelle „users“ der Datenbank gespeichert und beim nächsten Mal konnte man sich direkt mit der E-Mail-Adresse und Passwort selbst einloggen. Das Passwort wurde im Programm „register.php“ mit md5 bezeichnet.

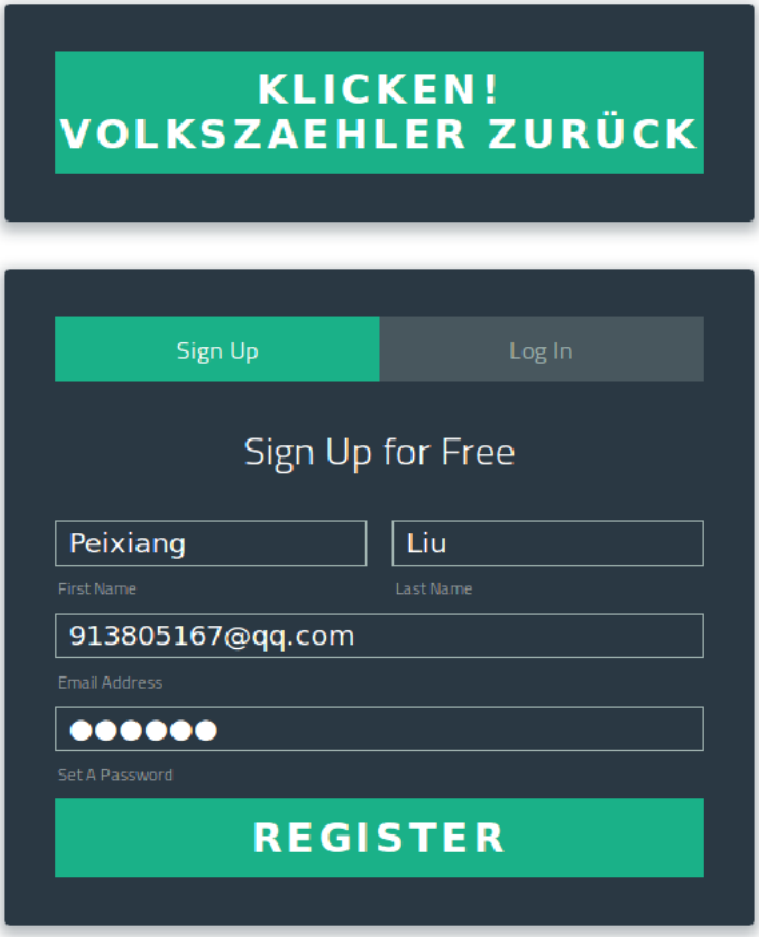


Abb. 5.7: Registerseite für den Benutzer

Quelle: eigene Abbildung

Hier wurden zwei Benutzer für zwei Messgeräte benannt. Wenn man sich mit seiner E-Mail-Adresse einloggte, durfte man nur die Situation des Leistungsverbrauchs von seinem Messgerät erfahren.

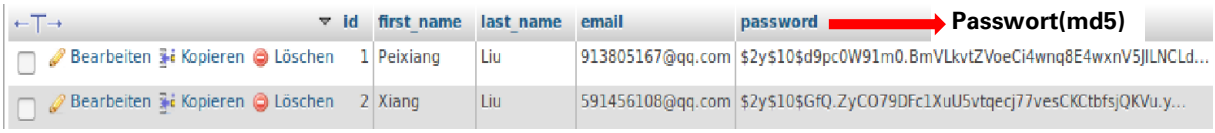


Abb. 5.8: Speicherung der beiden Benutzer in der Datenbank

Quelle: eigene Abbildung

Dann bekam man zwei Netzadressen der beiden Leistungsverbräuche aus der Website „Data Visualisierung“ und schrieb diese danach in das Programm „login.php“ für die Anmeldungsseite. Das bedeutete, dass die Website seines individuellen Leistungsverbrauchs automatisch erschien, wenn man seine eigene E-Mail-Adresse und sein Passwort hier korrekt einschrieb.

Im Folgenden ist ein kleiner Teil des Programmes „login.php“ aufgezeigt:

```
if ( password_verify($_POST['password'], $user['password']) && $user['id'] == "1" ) {
    $_SESSION['id'] = $user['id'];
    $_SESSION['email'] = $user['email'];
    $_SESSION['first_name'] = $user['first_name'];
    $_SESSION['last_name'] = $user['last_name'];

    // This is how we'll know the user is logged in
    $_SESSION['logged_in'] = true;
    header("location: http://192.168.10.113/login7/index.html?middleware=http://192.168.10.113/middleware.php/&channels=4a2b4000-a8d1-11e7-842a-f9c408836b1e");
```

Wenn alle Informationen des Benutzers richtig sind, erscheint die Website des individuellen Leistungsverbrauchs automatisch.

Netzadresse des individuellen Leistungsverbrauchs

Jetzt wurde es mit zwei verschiedenen Benutzernamen ausprobiert und man konnte deutlich sehen, dass mit der Eintragung der unterschiedlichen Anmeldungsinformationen verschiedene Leistungsverbräuche in der Website „Data Visualisierung“ zu sehen waren.

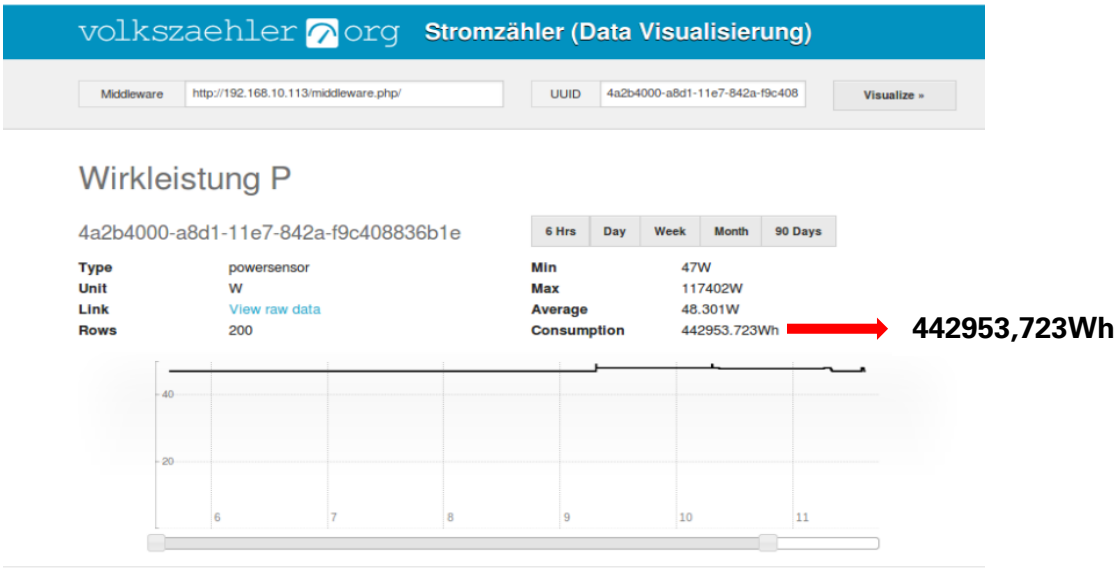
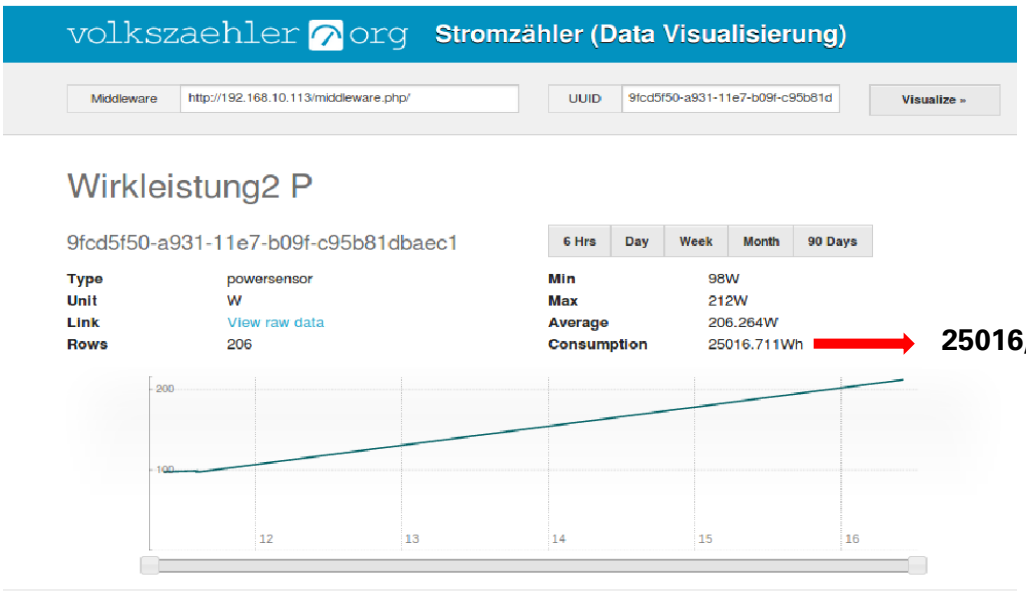


Abb. 5.9: Leistungsverbrauch vom ersten Benutzer (erstes Messgerät)

Quelle: eigene Abbildung



S

Abb. 5.10: Leistungsverbrauch vom zweiten Benutzer (zweites Messgerät)

Quelle: eigene Abbildung

Wenn man sich danach abmelden wollte, konnte man oben links in der Webseite „Log Out“ anklicken und auf der Anmeldungsseite zurückgehen. Dann konnte man sich neu registrieren oder sich mit einem neuen Benutzernamen anmelden.

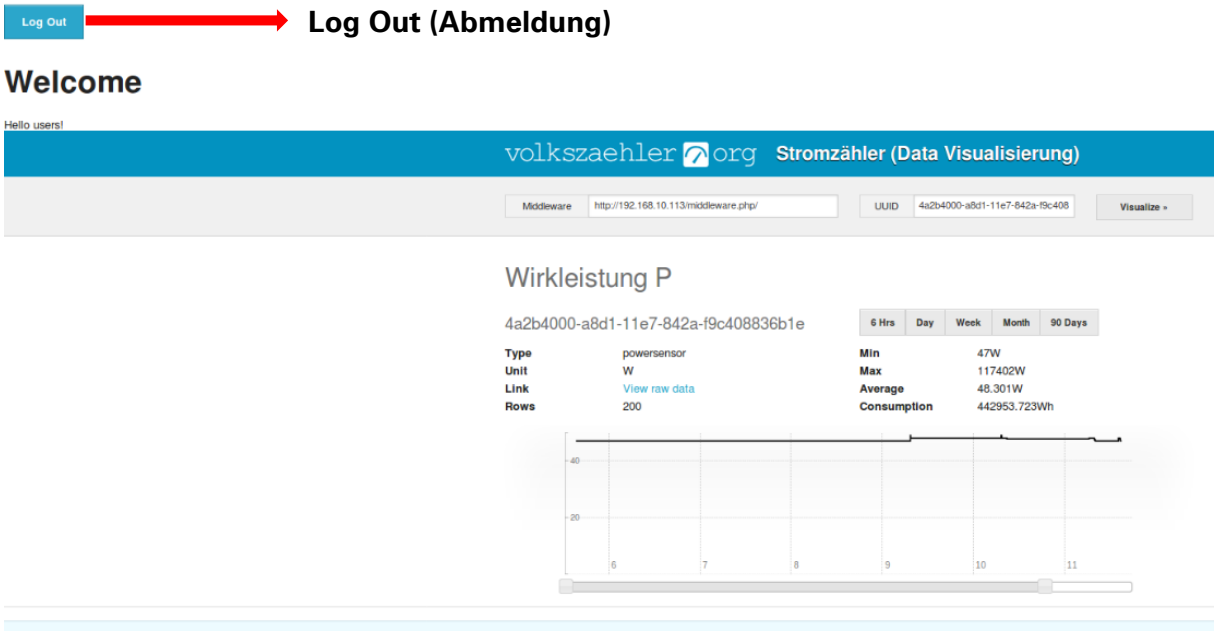


Abb. 5.11: Abmeldung des Benutzers in der Website „Data Visualisierung“

Quelle: eigene Abbildung

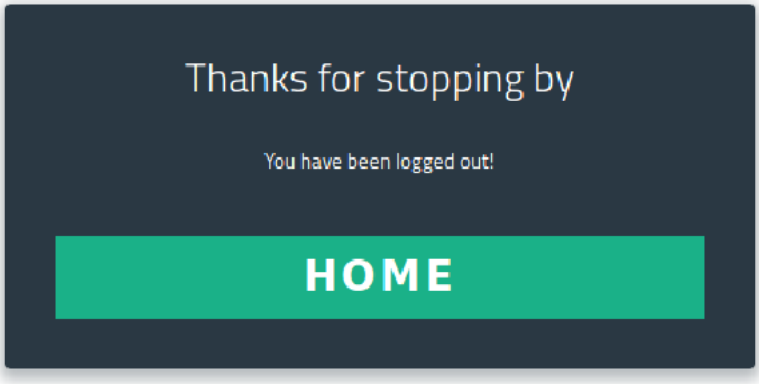


Abb. 5.12: Abmeldung des Benutzers und zurück zur Anmeldungsseite

Quelle: eigene Abbildung

Jetzt war die Authentifikation der Benutzer in der Web-Oberfläche realisiert, und wenn es mehrere Benutzer gab, konnten sie sich selbst anmelden, um den Leistungsverbrauch in ihrer individuellen Website „Data Visualisierung“ zu prüfen!

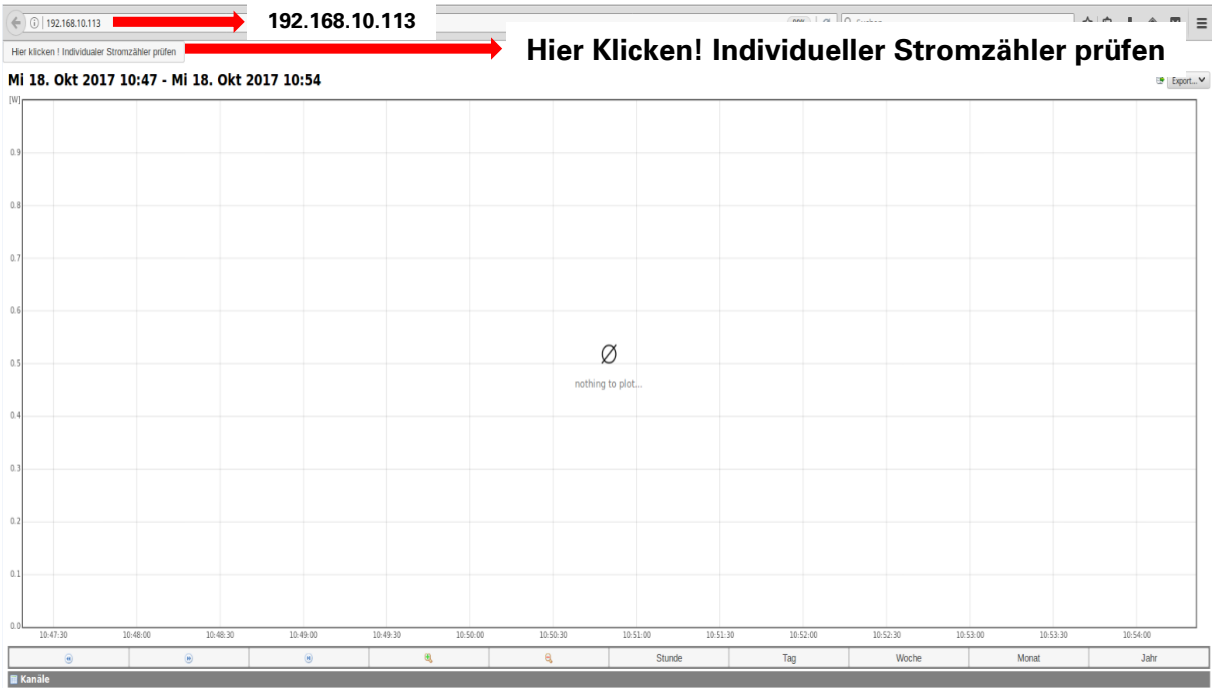


Abb. 5.13: Link der Anmeldungsseite auf dem Server des Volkszählers

Quelle: eigene Abbildung

6 Zusammenfassung und Ausblick

In den letzten drei Kapiteln wurden die Module und die Kommunikation des Messgerätes, die Grundlagen der Software und der Datenanalyse, die Visualisierung von Energieverbräuchen als Web-Oberfläche und Authentifikation der Benutzer in der Web-Oberfläche vorgestellt. Die Hauptaufgabe bestand im Aufbau der Schnittstelle zwischen Messgerät (NAS-8) und Smart-Meter-System (Volkszähler), um die Datenflüsse aus dem Messgerät in das Frontend des Volkszählers genauer darstellen zu können. Dann erfolgte die Realisierung der Authentifikation der Benutzer in der Web-Oberfläche, damit die Benutzer die Leistungsverbräuche in ihrer individuellen Website (Data Visualisierung) ganz genau prüfen können.

Die Arbeit war sehr sinnvoll und lehrreich. Man hat sich nicht nur die Kenntnisse über die Grundlagen von Linux angeeignet, sondern konnte auch das Niveau der Python-Programmierung erhöhen. Auch die Produktionsmethode der Website und die Authentifikation des Benutzers wurden umfangreich studiert, und das war sehr interessant!

Im Verlauf dieses Projektes wurde ganz deutlich, dass man sich mithilfe dieser Technik nicht nur die Leistungsverbräuche anzeigen, sondern auch Wasserverbräuche, Gasverbräuche, Temperaturmessungen und Luftdruckmessungen realisieren kann. Es wird in der Zukunft so sein, dass viele Familien ganz bequem von zu Hause aus nur per Mausklick ihren Energieverbrauch erfahren und kontrollieren können. Das ist ziemlich günstig für die Einwohner und die Technik muss in der Zukunft weltweit populär und nützlich sein.

Abkürzungsverzeichnis

Abkürzung	Bezeichnung
BDEW	Bundesverband der Energie- und Wasserwirtschaft
LCD	Liquid Crystal Display
VDE	Verband der Elektrotechnik
LENA	Leistungsmesser/Energiezähler NA-Schutz
USB	Universal Serial Bus
RTU	Remote Terminal Unit
TCP	Transmission Control Protocol
LED	Light Emitting Diode
IPC	Inter-Process Communication
IP	Internet Protocol
UUID	Universally Unique Identifier
URL	Uniform Resource Locator

Abbildungsverzeichnis

Tabellenverzeichnis

Quellen- und Literaturverzeichnis

[1]
NAS-8: Anleitung der NAS-8 bei Koralewski/NAS-8_004.pdf (13.08.2017)

[2]
Beaglebone Black: http://beagleboard.org/black (14.08.2017)

[3]
Linux: https://de.wikipedia.org/wiki/Linux (15.08.2017)

[4]
Linux-Kommando: https://de.wikipedia.org/wiki/Unix-Kommando (15.08.2017)

[5]
Embedded-Linux: https://de.wikipedia.org/wiki/Embedded_Linux (15.08.2017)

[6]
Modbus: https://de.wikipedia.org/wiki/Modbus (16.08.2017)

[7]
Interprozesskommunikation:
https://de.wikipedia.org/wiki/Interprozesskommunikation (17.08.2017)

[8]
Volkszähler:
https://wiki.volkszaehler.org/ (17.10.2017)

[9]
Module des Volkszählers:
https://wiki.volkszaehler.org/overview (18.10.2017)

[10]
Website der „Data Visualisierung:
https://wiki.volkszaehler.org/software/frontends/vzvis (19.10.2017)

[11]
Produktionsmethode der Authentifikation der Benutzer:
https://www.youtube.com/watch?v=Pz5CbLqdGwM&t=424s (20.10.2017)

[12]
Interne Kommunikation zwischen LENA und Beaglebone Black:
Anleitung der internen Kommunikation bei Koralewski/ LENA01_BB_Kommunikation_004.pdf

Anhang

Anhang 1: Python-Programm der Schnittstelle der Übertragung des Messwertes

test_cl.py

Author: Peixiang Liu/Zhichao Wang

```
import sys
import time
import logging
sys.path.append('../')
from socieerMsg import Message
from socieerIPC import messanger
from socket import socket
import urllib2

LOG_FORMAT = ('%(levelname) -10s %(asctime)s %(name) -8s %(funcName) '
              '-8s %(lineno) -5d: %(message)s')
LOGGER = logging.getLogger(__name__)
logging.basicConfig(level=logging.DEBUG,
                    format='%(asctime)s  %(filename)s[line:%(lineno)d]  %(levelname)s  %(mes-
sage)s',
                    datefmt='%a, %d %b %Y %H:%M:%S')

mess_str = [
    'realpower', 'reactivepower', 'complexpower', 'cosphi', 'firstvoltage',
    'secondvoltage', 'thirdvoltage', 'firstcurrent', 'secondcurrent', 'thirdcurrent']
```

```
time_slots = ['', '_day_', '_month_', '_year_']

agg_method = ['average']


ipc = messanger(Message)

ipc.init_pipe(is_server=False)

CARBON_SERVER = '10.40.8.1'

CARBON_PORT = 2003

CARBON_IO = True


message = Message()

message.id = 1

message.gid = 1

message.mtype = 1

message.severity = 1

message.payload = "test"


sock = socket()


try:

    sock.connect((CARBON_SERVER, CARBON_PORT))

except:

    logging.error(

        "# Couldn't connect to {0!r} on port {1!r}, is carbon-agent.py running?".format(CAR-

BON_SERVER, CARBON_PORT))

#   CARBON_IO = False
```

```
def main():
    while True:
        print "reading...."
        m2 = ipc.recv()
        print m2
        if m2 is not None:
            if m2.mtype == 66:
                logging.info("# get the message from ICP: {0!r}".format(m2.payload))



                if CARBON_IO == True:
                    t = 0
                    datas = m2.payload.split(";")
                    A = (int(datas[0]))
                    B = (int(datas[1]))
                    C = (int(datas[2]))
                    D = (int(datas[3]))
                    E = (int(datas[4]))
                    F = (int(datas[5]))
                    G = (int(datas[6]))
                    H = (int(datas[7]))
                    I = (int(datas[8]))
                    J = (int(datas[9]))
                    print A
                    print E
                    print F
```

```
            print G

            print H

            print I

            print J


            now = int(time.time())

            uuid0 = "4a2b4000-a8d1-11e7-842a-f9c408836b1e"

            timestamp = time.time()*1000

            ts0 = timestamp

            path = 'http://192.168.10.113/middleware.php/data/%s.json?ts=%d&value=%d' %
(uuid0, ts0, A)

            req = urllib2.Request(url = path, data = '')

            f = urllib2.urlopen(req)


            uuid4 = "4c220990-a8d0-11e7-bb32-3b4d249025a0"

            timestamp = time.time()*1000

            ts4 = timestamp

            path = 'http://192.168.10.113/middleware.php/data/%s.json?ts=%d&value=%d' %
(uuid4, ts4, E)

            req = urllib2.Request(url = path, data = '')

            f = urllib2.urlopen(req)


            uuid5 = "86e1e7b0-a8d0-11e7-baaa-bd02ebbc147c"

            timestamp = time.time()*1000

            ts5 = timestamp
```

```
        path = 'http://192.168.10.113/middleware.php/data/%s.json?ts=%d&value=%d' %
(uuid5, ts5, F)
        req = urllib2.Request(url = path, data = '')
        f = urllib2.urlopen(req)

        uuid6 = "9fdf9a40-a8d0-11e7-be4a-d76168f04031"
        timestamp = time.time()*1000
        ts6 = timestamp
        path = 'http://192.168.10.113/middleware.php/data/%s.json?ts=%d&value=%d' %
(uuid6, ts6, G)
        req = urllib2.Request(url = path, data = '')
        f = urllib2.urlopen(req)

        uuid7 = "02051b70-a8d1-11e7-a427-5b90a6ec68dc"
        timestamp = time.time()*1000
        ts7 = timestamp
        path = 'http://192.168.10.113/middleware.php/data/%s.json?ts=%d&value=%d' %
(uuid7, ts7, H)
        req = urllib2.Request(url = path, data = '')
        f = urllib2.urlopen(req)

        uuid8 = "a45c8940-a8d1-11e7-b6c6-b3fa994249b6"
        timestamp = time.time()*1000
        ts8 = timestamp
        path = 'http://192.168.10.113/middleware.php/data/%s.json?ts=%d&value=%d' %
(uuid8, ts8, I)
```

```
            req = urllib2.Request(url = path, data = '')
            f = urllib2.urlopen(req)

            uuid9 = "2f2c8180-a8d1-11e7-838f-2998923c62a1"
            timestamp = time.time()*1000
            ts9 = timestamp
            path = 'http://192.168.10.113/middleware.php/data/%s.json?ts=%d&value=%d' %
(uuid9, ts9, J)
            req = urllib2.Request(url = path, data = '')
            f = urllib2.urlopen(req)



            t = t + 1
            time.sleep(0.2)
            # We're gonna report all three loadavg values
            j = 0
            for mstr in (mess_str):
                lines = []
                for tslots in (time_slots):
                    if tslots is "":
                        namebase = "socieerlena.KMAatPSW." + mstr
                        lines.append(
                            "%s %s %d" % (namebase, datas[j], now))
                    else:
                        for agm in (agg_method):
                            namebase = "socieerlena.KMAatPSW." + mstr + tslots + agm
```

```
                    lines.append("%s %s %d" % (namebase, datas[j], now))
                j=j+1
                #all lines must end in a newline
                message = '\n'.join(lines) + '\n'
                print(message)
                logging.info("# Sending message into database!")
                print("# message:{0!r}".format(message))

            else:
                pass
        else:
            logging.info(
                "# get not usefull message from ICP: {0!r}".format(m2.payload))
    else:
        logging.info('# There is no message')
    time.sleep(0.1)


if __name__ == "__main__":
    main()
```

Anhang 2: Zusammenfassendes Python-Programm im Beaglebone Black

test.py

Author: Zhichao Wang

```
import lenainstrument
import analyse
import logging
import sys
import time
from appurtenance import BackupIntoFile, format_payload
sys.path.append('../')
from socieerMsg import Message  # for pickled messages
from socieerIPC import messanger  # ipc interface
import signal


lenainstrument.minimodbus.DEBUGLEVEL = 3


LOG_FORMAT = ('%(levelname) -10s %(asctime)s %(name) -8s %(funcName) '
              '-8s %(lineno) -5d: %(message)s')
LOGGER = logging.getLogger(__name__)


def signal_handler(signal, frame):
    print('You pressed Ctrl+C!')
    analyse.EXIT_FLAG = 1
    sys.exit(0)
```

```
signal.signal(signal.SIGINT, signal_handler)


class socierLENA():
    """
    This is a class for lena service
    """


    def __init__(self):
        self.dev = lenainstrument.Lena()
        self.ls = [0, 10, 3, 3, 0, 5]
        self.lsl = [49, 1000, 20000, 100000, 200000, 1000000, 2000000, 3000000]
        self.ns = analyse.NetzwerkStatus()
        self.th = analyse.MyThread(self.ns)
        self.mBackup = BackupIntoFile()
        logging.warning(
            '<#> Deivies Infos: %s ', self.dev)
        self.MSGnet = True
        if self.MSGnet is True:
            logging.warning(
                '<#> configure the messager handler')
            self.msgr_c = messanger(Message)
            self.msgr_c.init_pipe(is_server=True)
            self.m1 = Message()
            self.m1.id = 1
            self.m1.gid = 1
            self.m1.mtype = 1
```

```
            self.m1.severity = 1
            self.m1.payload = "test"
        else:
            logging.error(
                '* The MSGnet parameter must be true. Given {0!r}.'.format(self.MSGnet))


    def write(self, string):
        with open("/tmp/lena_tmp","a") as outfile:
            outfile.write(string)


    def test(self):
        """
        """
        self.th.start()
        logging.warning('<#> run the lena test module using the test methode()')
        ct = 0
        while True:
            try:
                self.mBackup.backup_timestamp(format_payload(0, time.time(), descrip-
tion="timestamp"), rewrite=True)
                response = self.dev.getCyclicvalue()
                self.m1.mtype = 65
                #errorcode = analyse.getErrorfunctioncode(response[0])
                self.write(str(response))
                logging.warning('<#> Function 65 with response: %s\n', response)
                self.m1.payload = (';'.join(str(v) for v in response))
```

```
            self.msgr_c.send(self.m1)
            self.mBackup.backup_fkt_cycle_value(format_payload(65, response, description="fkt_get_cycle_values"))
            response = self.dev.getActualvalue()
            self.m1.mtype = 66
            self.write(str(response))
            logging.warning('<#>Function 66 with response: %s\n', response)
            self.m1.payload = (';'.join(str(v) for v in response))
            self.msgr_c.send(self.m1)
            self.mBackup.backup_fkt_actual_value(format_payload(66, response, description="fkt_get_actual_values"))
            self.lsl[0] = analyse.getNetworkstatusbits(self.ns.isIP, self.ns.isNetwork)
            self.mBackup.backup_network_status(format_payload(71, [self.ns.isIP, self.ns.isNetwork],
                                                description="network status"))
            response = self.dev.setSystemstatus(self.lsl)
            logging.warning('<#> Function 71 with response: %s\n', response)

        except (ValueError, TypeError, IOError) as ErrorString:
            logging.error('* Error info: {0!r}'.format(ErrorString))
        else:
            pass
        ct = ct + 1

    def stop(self):
        logging.info(
```

```
            '* stop run the lenaservice module')
        sys.exit(0)

if __name__ == "__main__":
    logging.basicConfig(level=logging.NOTSET, format=LOG_FORMAT)
    lenas = socierLENA()
    lenas.test()
```

Anhang 3: php-Programm der Registerseite des Benutzers

register.php

Author: Clever Techie

```
<?php
/* Registration process, inserts user info into the database
   and sends account confirmation email message
 */

// Set session variables to be used on profile.php page
$_SESSION['email'] = $_POST['email'];
$_SESSION['first_name'] = $_POST['firstname'];
$_SESSION['last_name'] = $_POST['lastname'];

// Escape all $_POST variables to protect against SQL injections
$first_name = $mysqli->escape_string($_POST['firstname']);
$last_name = $mysqli->escape_string($_POST['lastname']);
$email = $mysqli->escape_string($_POST['email']);
$password = $mysqli->escape_string(password_hash($_POST['password'], PASS-
WORD_BCRYPT));
$hash = $mysqli->escape_string( md5( rand(0,1000) ) );

// Check if user with that email already exists
$result = $mysqli->query("SELECT * FROM users WHERE email='$email'") or
die($mysqli->error());
```

```
// We know user email exists if the rows returned are more than 0
if ( $result->num_rows > 0 ) {

    $_SESSION['message'] = 'User with this email already exists!';
    header("location: error.php");

}
else { // Email doesn't already exist in a database, proceed...

    // active is 0 by DEFAULT (no need to include it here)
    $sql = "INSERT INTO users (first_name, last_name, email, password, hash) "
          . "VALUES ('$first_name','$last_name','$email','$password', '$hash')";

    // Add user to the database
    if ( $mysqli->query($sql) ){

        $_SESSION['active'] = 0; //0 until user activates their account with verify.php
        $_SESSION['logged_in'] = true; // So we know the user has logged in
        $_SESSION['message'] =

             "Confirmation link has been sent to $email, please verify
             your account by clicking on the link in the message!";

        // Send registration confirmation link (verify.php)
        $to      = $email;
        $subject = 'Account Verification ( qq.com )';
```

```
        $message_body = '

        Hello '.$first_name.',


        Thank you for signing up!


        Please click this link to activate your account:


        http://192.168.10.113/login7/verify.php?email='.$email.'&hash='.$hash;


        mail( $to, $subject, $message_body );


        header("location: profile.php");


    }


    else {

        $_SESSION['message'] = 'Registration failed!';

        header("location: error.php");

    }


}
```

Anhang 4: php-Programm der Anmeldungsseite des Benutzers

Login.php

Author: Clever Techie

```php
<?php

/* User login process, checks if user exists and password is correct */


// Escape email to protect against SQL injections

session_start();

$email = $mysqli->escape_string($_POST['email']);

$result = $mysqli->query("SELECT * FROM users WHERE email='$email'");


if ( $result->num_rows == 0 ){ // User doesn't exist

    $_SESSION['message'] = "User with that email doesn't exist!";

    header("location: error.php");

}

else { // User exists

    $user = $result->fetch_assoc();


if ( password_verify($_POST['password'], $user['password']) && $user['id'] == "1" ) {

    $_SESSION['id'] = $user['id'];

    $_SESSION['email'] = $user['email'];

    $_SESSION['first_name'] = $user['first_name'];

    $_SESSION['last_name'] = $user['last_name'];

    $_SESSION['active'] = $user['active'];
```

```
    // This is how we'll know the user is logged in
    $_SESSION['logged_in'] = true;
    header("location: http://192.168.10.113/login7/index.html?middle-ware=http://192.168.10.113/middleware.php/&channels=4a2b4000-a8d1-11e7-842a-f9c408836b1e");
    }
    elseif ( password_verify($_POST['password'], $user['password']) && $user['id'] == "2" ) {
        $_SESSION['id'] = $user['id'];
        $_SESSION['email'] = $user['email'];
        $_SESSION['first_name'] = $user['first_name'];
        $_SESSION['last_name'] = $user['last_name'];
        $_SESSION['active'] = $user['active'];

        // This is how we'll know the user is logged in
        $_SESSION['logged_in'] = true;

        header("location: http://192.168.10.113/login7/index.html?middle-ware=http://192.168.10.113/middleware.php/&channels=9fcd5f50-a931-11e7-b09f-c95b81dbaec1");
    }
    else {
        $_SESSION['message'] = "You have entered wrong password, try again!";
        header("location: error.php");
    }
}
```