

Projektaufgabe Embedded Systems

Dennis Sadkowski, 1006210

9. September 2024

Inhaltsverzeichnis

1 Einleitung	2
1.1 Motivation	2
1.2 Aufgabenstellung	3
2 Testplan	3
3 Installationsanleitung	4
3.1 Hardware	4
3.2 Versuchsaufbau	4
3.3 Software	6
3.4 Verzeichnisstruktur	6
3.5 TFTP	6
3.6 Netzwerkboot	7
3.7 Kernel	8
3.8 Buildroot	9
3.9 Einrichten Des MQTT-Dashboards	10
4 Systemtest	12
4.1 Komponententest	12
4.1.1 Testen des Netzwerkboots	12
4.1.2 Testen der Benutzeranmeldung	13
4.1.3 Testen des SSH-Zugangs	13
4.1.4 Testen des Access Pointes	14
4.1.5 Mosquitto Tests	15
4.2 Testen des Treibers	16
4.3 Test des Gesamtsystems	17
4.3.1 Testaufbau	17
4.3.2 Testdurchführung	17
4.3.3 Testergebnisse	17
5 Zusammenfassung	19
6 Literaturverzeichnis	20

Abbildungsverzeichnis

1	Verbreitung von IoT Geräten in letzten Jahren im Heimbereich	3
2	Aufbau des Systems	4
3	Anschluss des Ethernet Kabels am Switch	4
4	Schaltplan für die LED	5
5	Oberfläche des Rpi Imagers	7
6	Eeprom-config	8
7	Oberfläche der Brokerkonfiguration (links) und der Dashboard Oberfläche (rechts)	10
8	Konfiguration der Textkacheln (links) und des Slider Widgets (rechts)	11
9	Erfolgreicher Netzwerkboot des Raspberry Pis	12
10	Anmeldung des Benutzers root	13
11	Anmeldung des Benutzers Sadkowski	13
12	SSH Verbindung	14
13	Wlan	14
14	WlanPing	14
15	Links der mosquitto_pub Befehl mit dem entsprechenden mosquitto_sub Befehl rechts.	15
16	Die Abbildung zeigt die Erfolgreiche Ankunft der Daten über die Abonnierten Topics.	15
17	Ein und Ausgaben der Treiberfrequenz	16
18	Fertiges MQTT Dashboard mit angkommenden Daten	18

1 Einleitung

Das Projekt widmet sich der Umsetzung eines IoT-Projekts auf Basis des Raspberry Pi 4, bei dem die Steuerung der an den Raspberry Pi angeschlossenen Hardware über ein Smartphone erfolgt. Die Dokumentation soll es Dritten ermöglichen, das System nachzubauen und lauffähig zu machen.

1.1 Motivation

In den letzten Jahren hat die Bedeutung von vernetzten Geräten und IoT-Geräten (Internet of Things) in verschiedenen Altersgruppen stark zugenommen [@SmarthomeTrend]. Die Möglichkeit, Geräte über das Internet anzusteuern, bietet dem Heimanwender neue Möglichkeiten, intelligente Geräte wie Lampen, Steckdosen, Thermostate etc. zu steuern und deren Verbrauch zu überwachen, um insbesondere Energie und Geld zu sparen. Um die bestmögliche Lösung für die eigenen Bedürfnisse zu finden und möglichst unabhängig von Anbietern zu sein, die ausschließlich mit der Cloud arbeiten, sind insbesondere Kenntnisse über eingebettete Systeme notwendig. Die vorliegende Arbeit widmet sich der Umsetzung eines solchen IoT-Projektes auf Basis eines Raspberry Pi 4 mit dem Ziel, Grundkenntnisse im Embedded-Bereich zu vermitteln.

Der Smart-Home-Trend ist ungebrochen

Nutzen Sie Smart-Home-Anwendungen in Ihrem Haushalt?



Abbildung 1: Verbreitung von IoT Geräten in letzten Jahren im Heimbereich

1.2 Aufgabenstellung

Die Projektarbeit beschäftigt sich mit der Umsetzung eines IoT-Projektes auf Basis des Raspberry Pi 4. Mit diesem soll es möglich sein, eine LED mit Hilfe eines Smartphones ein- und auszuschalten sowie in einer bestimmten Frequenz blinken zu lassen, wobei die Blinkfrequenz einstellbar ist. Die Ansteuerung der LED erfolgt über einen Gerätetreiber, der beim Start automatisch geladen wird. Die Kommunikation zwischen dem Raspberry Pi und dem Startphone erfolgt über das MQTT-Protokoll. Der Raspberry Pi fungiert hierbei als Broker. Damit sich auch andere Geräte mit dem Broker verbinden können, baut der Raspberry Pi zusätzlich ein Wlan auf. Auf dem Smartphone läuft eine MQTT Dashboard App, über die die Daten visuell aufbereitet dargestellt werden können und über die die Ansteuerung der LEDs umgesetzt wird. Das System besteht aus 2 Teilen, zum einen dem Kernel und zum anderen dem Userland, welches mit Buildroot gebaut wird. Das Booten des Raspberry Pis soll über das Netzwerk erfolgen, wobei die notwendigen Daten über einen lokal auf dem Entwicklungsrechner installierten TFTP-Server bezogen werden. Um den Raspberry aus der Ferne administrieren zu können, läuft zusätzlich der SSH-Server dropbear.

2 Testplan

Um die korrekte Funktion der beteiligten Komponenten zu testen, wurden im Laufe des Projektes verschiedene Tests durchgeführt, die im Kapitel 4 näher erläutert werden. Folgende Punkte wurden bei den Tests berücksichtigt.

- Laden von Dateien vom lokal installierten TFTP-Server.
- Booten des Raspberry Pi über das Netzwerk
- SSH-Zugang zum Raspberry Pi
- Verschiedene Benutzeranmeldungen
- Verbinden der Endgeräte mit dem installierten Wlan
- Testen des Treibers durch Lese- und Schreibzugriffe
- Kommunikation mit MQTT testen

3 Installationsanleitung

3.1 Hardware

Für den Nachbau des Systems wird eine Reihe von Hardware benötigt, die nachfolgend aufgelistet ist.

Komponenten
Beliebigen Rechner mit installierten Ubuntu
Raspberry Pi 4 Modell B
1x Ethernet Kabel
1x SD Karte (32Gb)
1x LED
1x 330 Ohm Widerstand
4x Jumper Kabel
1x Breadboard
1x Usb to TTL Seriell Kable
1x Switch

Tabelle 1: Für das Projekt benötigte Hardware Komponenten

3.2 Versuchsaufbau

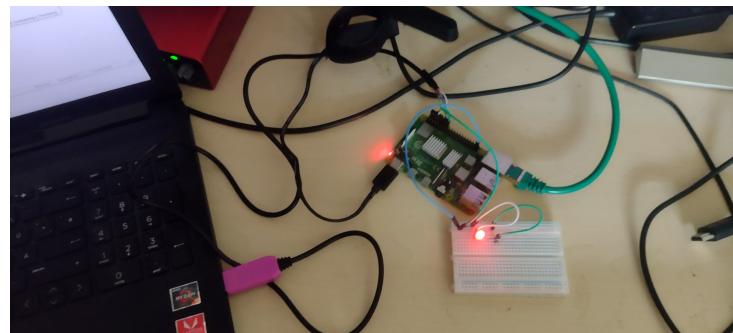


Abbildung 2: Aufbau des Systems

Die Abbildung 3 zeigt den Aufbau des Raspberry Pi. Zu Testzwecken ist er über ein USB-zu-TTL-Seriell-Kabel mit dem Entwicklungsrechner verbunden. Das grüne Ethernetkabel verbindet den Raspberry Pi mit einem Netgear Switch.

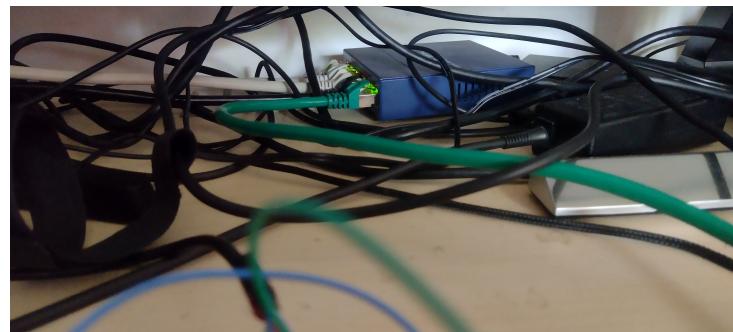


Abbildung 3: Anschluss des Ethernet Kabels am Switch

Die Verdrahtung der LED erfolgt auf dem Breadboard. Abbildung ?? zeigt, wie die Bauteile auf dem Breadboard gesteckt werden müssen und wie der Anschluss an den Raspberry Pi erfolgt.

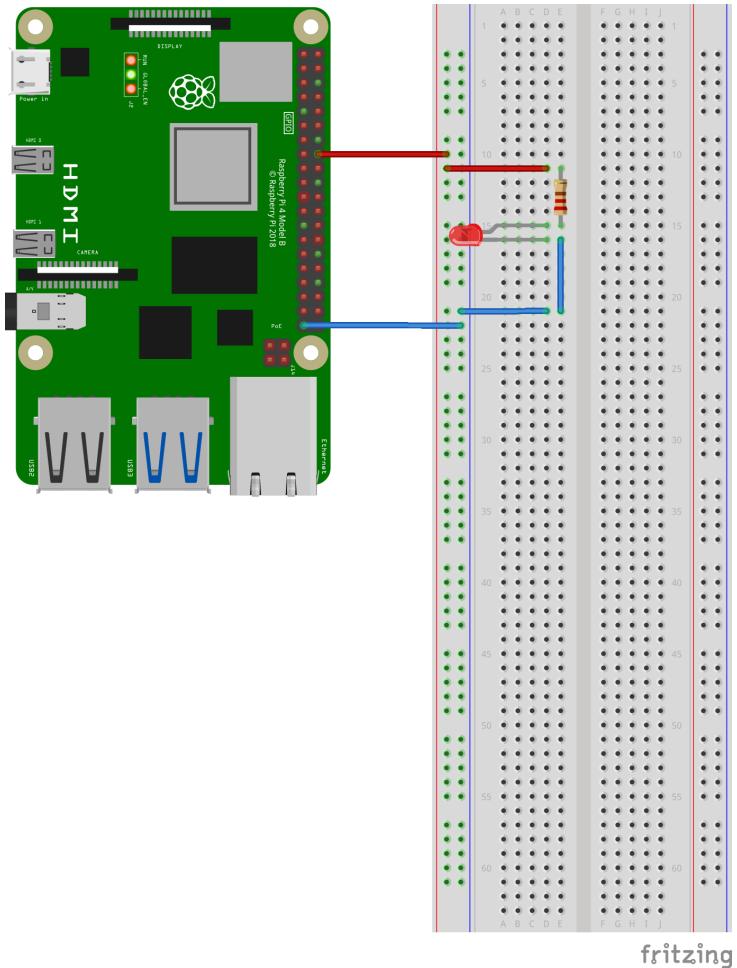


Abbildung 4: Schaltplan für die LED

3.3 Software

Dieses Kapitel gibt einen Überblick über die benötigte Software. Diese wird auf dem Host installiert. Als Entwicklungsrechner wird ein Laptop mit Ubuntu 22.04 verwendet. Für das Projekt wird auch Software wie Git, Gcc und Make benötigt. Die Installation kann einfach über den Package Manager `apt` in Ubuntu durchgeführt werden. Um die Systeme wie Buildroot oder den Linux-Kernel zu konfigurieren, werden unter Umständen noch weitere Pakete benötigt, die auf frisch installierten Systemen oft noch fehlen. Um die SD-Karte für den initialen Raspberry Pi Boot einzurichten, wird zusätzlich die Software `Raspberry Pi imager` benötigt. Nachfolgend sind die notwendigen Kommandos aufgelistet.

```
usr@host:~/ $ sudo apt install build-essential git  
usr@host:~/ $ sudo apt install flex bison libgmp3-dev \  
          libmpc-dev libssl-dev libncurses-dev  
usr@host:~/ $ sudo apt install rpi-imager
```

3.4 Verzeichnisstruktur

Für das Projekt wird empfohlen, Skripte, Konfigurationen sowie Buildroot- und Kernel-Dateien strukturiert in getrennten Verzeichnissen abzulegen. Folgende Verzeichnisstruktur wird empfohlen

```
.  
+-- ~/embedded  
|   +-- Projekt  
|   |   +-- buildroot-2023.02  
|   |   +-- linux  
|   |   +-- driver  
|   |   +-- scripts  
|   |   +-- target  
|   +-- export  
|   |   +-- driver  
|   |   +-- scripts  
|   |   +-- target  
|   |   +-- images  
|   |   +-- configs
```

Das Projektverzeichnis ist das empfohlene Verzeichnis, in dem die Skripte ohne Anpassungen laufen sollten. Das `export` Verzeichnis enthält die Konfigurationsdateien und Startskripte, die mit dem Projekt geliefert werden. Das Verzeichnis `scripts` enthält die relevanten Skripte für das Kopieren von Dateien in das aufgebaute Dateisystem sowie für das Kopieren des Dateisystems in das TFTP-Verzeichnis. Im Verzeichnis `target` werden alle Dateien abgelegt, die für den Betrieb der Dienste auf dem Raspberry Pi relevant sind. Hier befinden sich die Konfigurationsdateien sowie die Skripte, die beim Booten des Raspberry Pi gestartet werden sollen. Das Verzeichnis `driver` enthält den Treibercode, sowie Dateien zum Cross-Compilieren für den Pi. Im Verzeichnis `linux` befindet sich der eigentliche Linux Kernel Code und im Verzeichnis `buildroot2023.02` befindet sich das entpackte Buildroot.

Mit den folgenden Befehlen kopiert man die benötigten Daten in die Projektverzeichnisse.

```
usr@host:~/embedded$ sudo cp -R export/scripts Projekt/scripts  
usr@host:~/embedded$ sudo cp -R export/target Projekt/target  
usr@host:~/embedded$ sudo cp -R export/driver Projekt/driver
```

3.5 TFTP

Der Raspberry Pi holt sich die benötigten Daten wie Kernel und Dateisystem beim Booten über das Netzwerk von einem tftp-Server. Dieser ist auf dem lokalen Entwicklungsrechner installiert und konfiguriert. Der tftp-Server kann über apt mit dem Befehl `sudo apt install tftpd-hpa` installiert werden.

Zur Überwachung der TFTP-Ausgaben muss in der TFTP-Konfiguration das Verbose Logging aktiviert werden. Dazu muss in der Datei `/etc/default/tftpd-hpa` der Eintrag `TFTP_OPTIONS="--secure --verbose"` hinzugefügt werden. Nach dem Neustart des Dienstes mit dem Befehl `service tftpd-hpa restart` kann man sich die Logmeldungen mit dem Befehl `tail -f /var/log/syslog | grep tftp` im Terminal anzeigen lassen.

3.6 Netzwerkboot

Damit der Raspberry Pi seine Daten über das Netzwerk booten kann, muss er zunächst dafür konfiguriert werden. Dies geschieht durch Änderungen im sogenannten **Eeprom** (nicht flüchtiger Speicher). Dazu benötigt man zunächst ein Raspberry Pi Image wie Raspian, das man sich erst erstellen muss, wenn man noch keines hat. Dazu kann man die Software Raspi-Imager verwenden, siehe Kapitel Software 3.3. Im ersten Schritt wird die SD-Karte in den Entwicklungsrechner gesteckt.

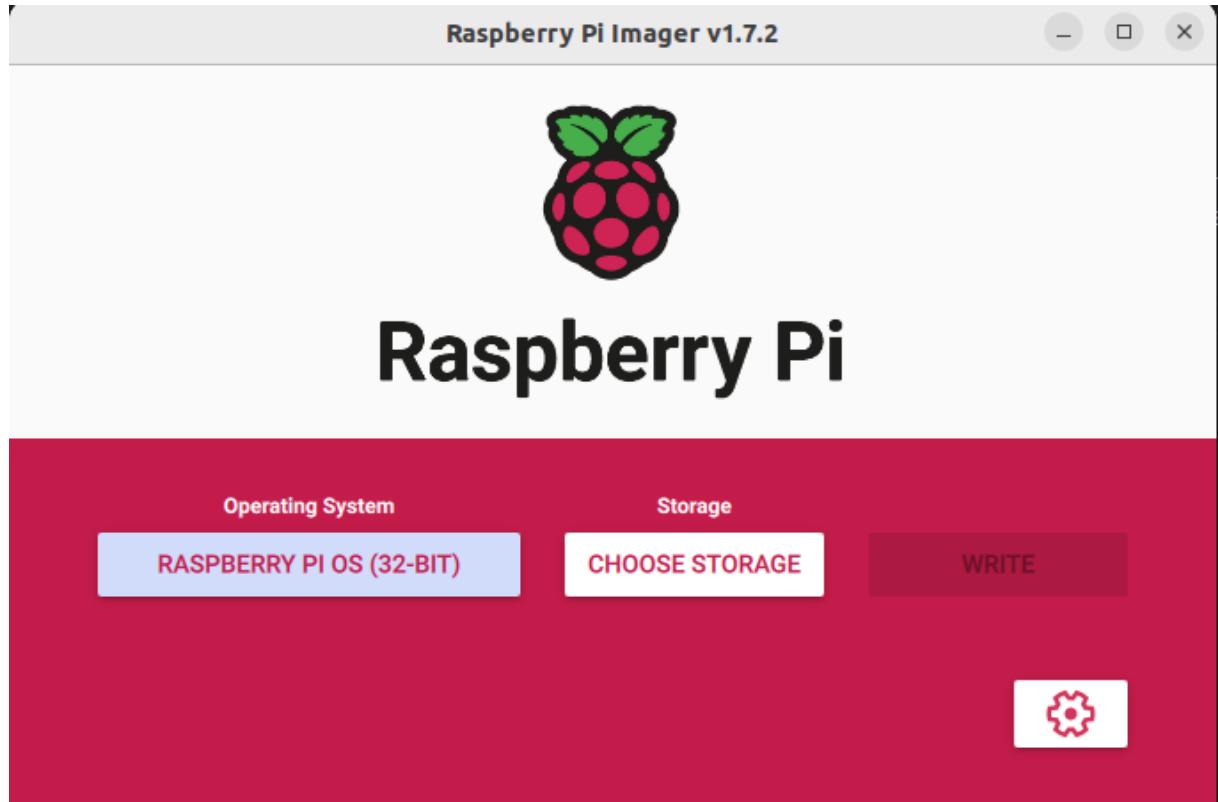


Abbildung 5: Oberfläche des Rpi Imagers

Im Menü wählt man das gewünschte Raspberry Pi OS und das Speichermedium aus. Zusätzlich kann man noch weitere Punkte wie Benutzername, Passwort und Wlan konfigurieren. Standardmäßig kann man sich mit dem Benutzer pi ohne Passwort anmelden, was für den initialen Bootvorgang ausreichend ist.

Nach dem Booten muss zunächst das Paket `rpi-eeprom` installiert werden. Dazu werden die folgenden Befehle nacheinander auf dem Pi ausgeführt.

```
sudo apt update  
sudo apt full-upgrade  
sudo apt install rpi-eeprom
```

Anschließend wird die Konfiguration mit dem Befehl `sudo -E rpi-eeprom-config --edit` bearbeitet.

```

GNU nano 5.4                               /tmp/tmpjg00jplw/boot.conf
[all]
BOOT_UART=1
WAKE_ON_GPIO=1
POWER_OFF_ON_HALT=1
DHCP_TIMEOUT=30000
DHCP_REQ_TIMEOUT=1000
TFTP_FILE_TIMEOUT=20000
TFTP_IP=192.168.0.68
TFTP_PREFIX=1
BOOT_ORDER=0xF21
SD_BOOT_MAX_RETRIES=2
NET_BOOT_MAX_RETRIES=2
CLIENT_IP=192.168.0.168
SUBNET=255.255.255.0
ENABLE_SELF_UPDATE=1
FREEZE_VERSION=0

Fehler: sudo -E rpi-eeprom-config --edit bearbeitet.
Dateien in das Tftp Verzeichnis kopiert werden dazu
[G] Help [D] Delete [W] Write Out [R] Read File [W] Where Is [A] Replace [K] Cut [U] Paste [T] Execute [C] Location
[X] Exit [J] Justify [G] Go To Line

```

Abbildung 6: Eeprom-config

Die Abbildung 6 zeigt die Konfiguration, die für das Booten über das Netzwerk erforderlich ist. Es ist zu beachten, dass die IP-Adressen des TFTP-Servers und des Clients aus dem lokalen Heimnetzwerk stammen. Diese müssen ggf. an das eigene Netzwerk angepasst werden.

3.7 Kernel

Um den Linux-Kernel kompilieren zu können, muss zuerst der Code von Github heruntergeladen werden. Das Projekt benutzt den Kernel 6.3.1.y. Um den Kernelcode zu clonen und anschließend in den richtigen Branch zu wechseln, können die folgenden Befehle verwendet werden.

```

user@host:~/embedded/Projekt$ git clone https://github.com/raspberrypi/linux.git
user@host:~/embedded/Projekt$ cd linux
user@host:~/embedded/Projekt/linux$ git checkout rpi-6.3.1.y

```

Anstatt jeden Schritt für die benötigten Kernelkomponenten einzeln zu erläutern, wird stattdessen erklärt, wie eine bereits vorgefertigte Konfiguration importiert und kompiliert werden kann. Damit der Kernel für den Raspberry Pi kompiliert werden kann, müssen zunächst einige Umgebungsvariablen gesetzt werden. Dazu liegt dem Projekt ein Skript mit dem Namen `export.sh` bei. Dieses wird mittels `source export.sh` ausgeführt und setzt die benötigten Variablen automatisch. Die folgenden Befehle müssen dann in dem Terminal ausgeführt werden, in dem auch das Export-Skript ausgeführt wurde. Dazu wird zunächst eine Standardkonfiguration benötigt. Diese hängt vom verwendeten Raspberry Pi ab. Für den im Projekt verwendeten Raspberry PI 4 ist dies die `bcm2711_defconfig`. Diese wird mit dem Befehl `user@host:~/embedded/Projekt/linux$ make bcm2711_defconfig` erzeugt. Die fertige Kernelkonfiguration mit dem Namen `config.linux` befindet sich im Verzeichnis `~/embedded/export/configs`. Diese muss in das Linux-Verzeichnis kopiert und unter dem Namen `.config` gespeichert werden.

```

user@host: cp ~/embedded/export/configs/config.linux \
          ~/embedded/Projekt/linux/.config

```

Anschließend muss das Menü einmal mit dem Befehl `make menuconfig` geöffnet und einmal ohne Änderungen gespeichert werden. Danach kann der Kernel mit dem Befehl `make -j6 zImage dtbs modules` gebaut werden. Der erste Build kann je nach Hardware auf dem Entwicklungsrechner bis zu einer Stunde und länger dauern. Nach Abschluss des Build-Prozesses muss das `zImage` in das `tftp`-Verzeichnis kopiert werden.

```
user@host:~/embedded/Projekt/linux$ sudo cp arch/arm/boot/zImage \
                                         /srv/tftp/kernel71.img
user@host:~/embedded/Projekt/linux$ sudo cp arch/arm/boot/dts/bcm2711-rpi-4-b.dtb \
                                         /srv/tftp/kernel71.img
```

3.8 Buildroot

Buildroot ist eine Toolbox, die es ermöglicht, mit minimalem Aufwand eigene Linux-Systeme zu bauen [@[Buildroot]]. Um es benutzen zu können, muss es zuerst heruntergeladen und entpackt werden. 3.4. Die für dieses Projekt verwendete Version ist 2023.02.

```
user@host:~/embedded/Projekt$ \
    wget https://buildroot.org/downloads/buildroot-2023.02.tar.gz
user@host:~/embedded/Projekt$ \
    tar xvf https://buildroot.org/downloads/buildroot-2023.02.tar.gz
user@host:~/embedded/Projekt/buildroot2023.02$ make raspberrypi4_defconfig
```

Der letzte Schritt erstellt eine Standardkonfiguration für den Raspberry Pi4 im frisch entpackten buildroot Verzeichnis. Sobald diese vorhanden ist, können die Systeme durch Änderungen in der .config gebaut werden. Dazu wird wieder auf die vorgefertigte Konfiguration aus dem Exportverzeichnis zurückgegriffen. Um die in Busybox konfigurierten Daten verwenden zu können, muss zusätzlich die Busybox-Konfiguration kopiert werden.

```
user@host:~/embedded$ sudo cp export/configs/config.buildroot \
                           Projekt/buildroot2023.02/.config
user@host:~/embedded$ sudo cp export/configs/busybox.config \
                           Projekt/buildroot2023.02/package/busybox/busybox.config
```

Danach kann das System gebaut werden. Dazu wird wieder `make -j6` aufgerufen. Die Dauer hängt wieder von der Hardware des Entwicklungsrechners ab. Wenn das System fertig gebaut ist, muss das Dateisystem noch auf den tftp-Server kopiert werden.

```
user@host:~/embedded/Projekt/buildroot2023.02$ sudo cp output/images/rootfs.cpio \
                                         /srv/tftp
```

3.9 Einrichten Des MQTT-Dashboards

Für das MQTT Dashboard wird die App MQTT Dashboard – IoT and Node – von Vetryu [AndroidApp] verwendet. Diese kann kostenlos im Google Play Store heruntergeladen werden. Um die App mit dem Mqtt Broker zu verbinden, muss das Smartphone im ersten Schritt mit dem Wlan des Raspberry Pi verbunden werden. Danach kann die App eingerichtet werden. Beim ersten Start der MQTT App wird man zunächst aufgefordert den Broker zu konfigurieren. Das Feld **Name** kann frei gewählt werden. Im Feld **Address** muss lediglich die IP des Brokers eingetragen werden. Das Protokoll wird von der Anwendung selbst festgelegt. Sollte das Feld **Port** leer sein, muss dort 1883 eingetragen werden. Anschließend wird die Konfiguration mit einem Klick auf das Diskettensymbol gespeichert. Ist dies geschehen, können Widgets mit einem Klick auf das + Symbol auf dem Dashboard erstellt werden. Für die Subscribed Topics werden in diesem Projekt Textboxen verwendet, um die empfangenen Werte anzuzeigen. Für die Einstellung der Frequenz wird wiederum ein Slider Widget mit einem festen Wertebereich verwendet.

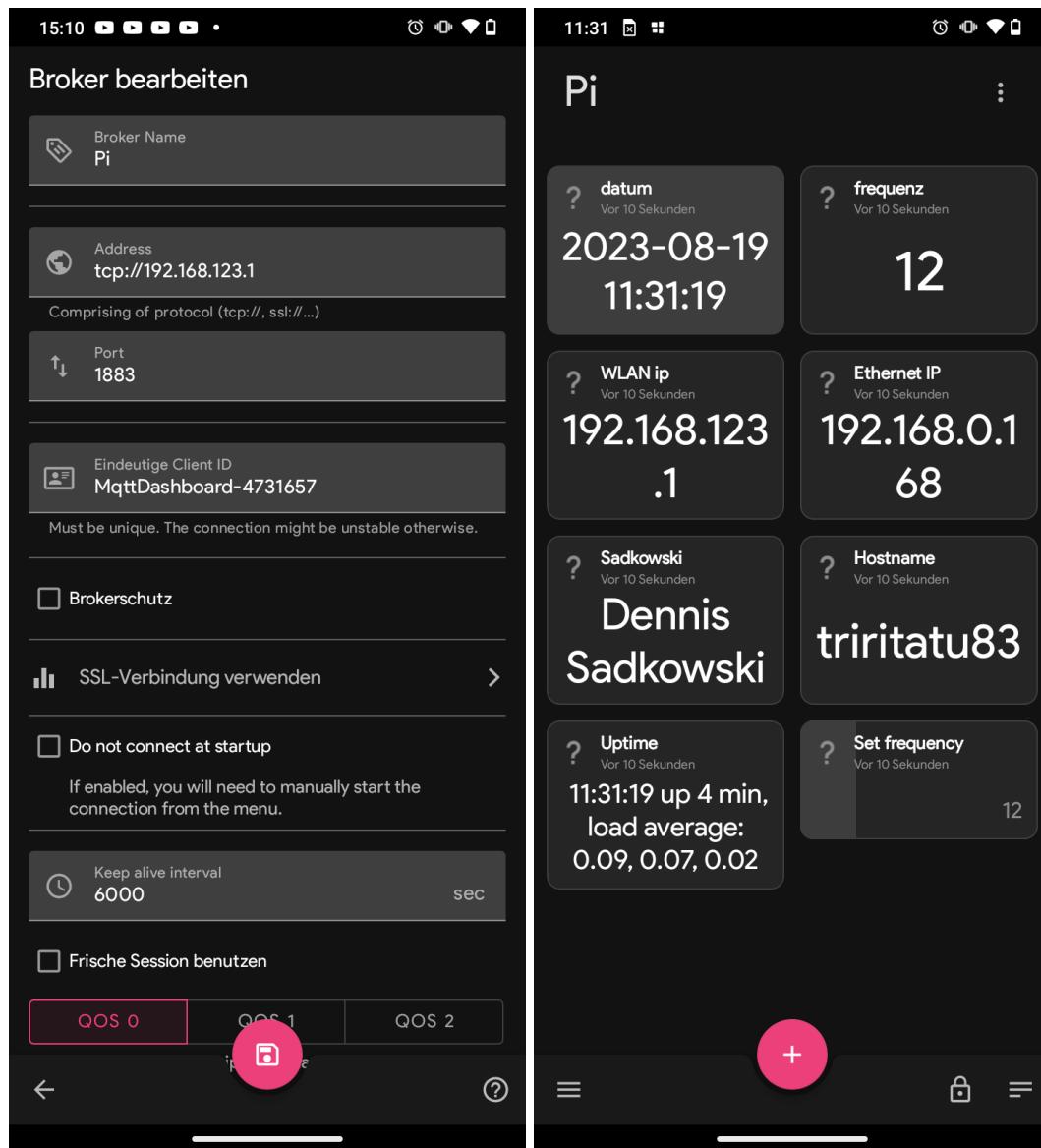


Abbildung 7: Oberfläche der Brokerkonfiguration (links) und der Dashboard Oberfläche (rechts)

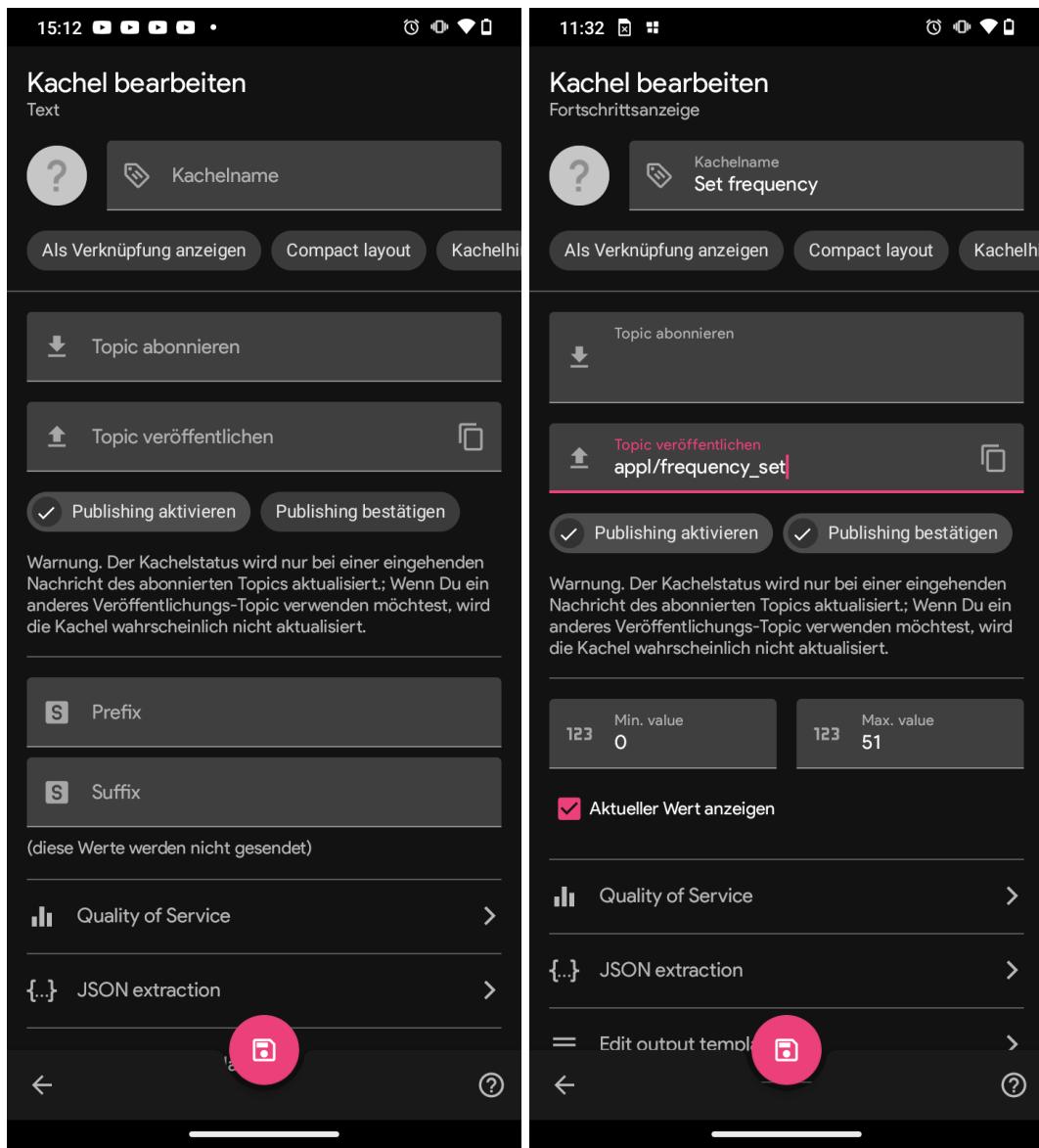


Abbildung 8: Konfiguration der Textkacheln (links) und des Slider Widgets (rechts)

Die Abbildungen zeigen die Oberflächen der entsprechenden Kacheln. Der Name kann frei gewählt werden. Um ein Topic zu subskribieren oder zu publizieren, muss es in das entsprechende Feld eingetragen werden. In der Slider-Kachel kann zusätzlich der Minimal- und Maximalwert eingegeben werden.

4 Systemtest

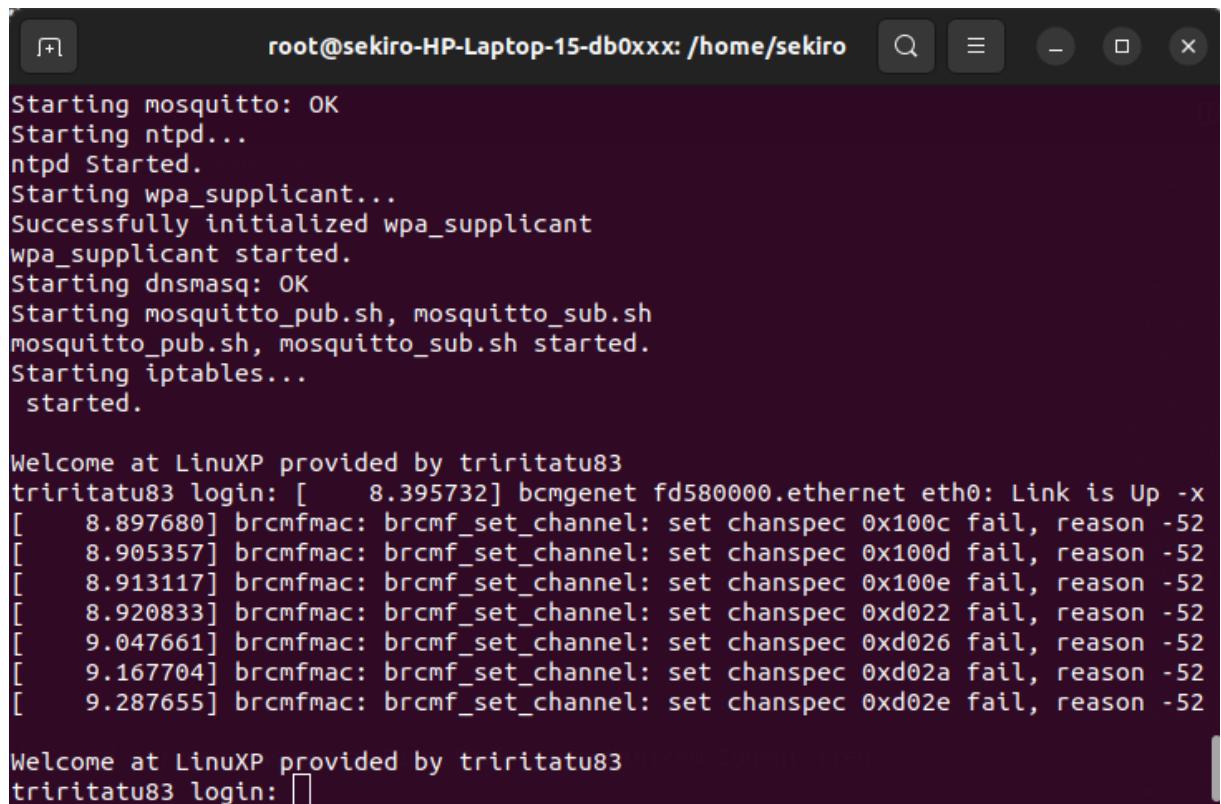
Um sicherzustellen, dass das System fehlerfrei funktioniert, wurden die Komponenten während der Entwicklung auf ihre korrekte Funktion getestet. Dieses Kapitel befasst sich daher mit den Tests, die für die einzelnen Komponenten durchgeführt wurden, sowie mit den Tests, die für das Gesamtsystem durchgeführt wurden.

4.1 Komponententest

Der Aufbau des Testsystems ist identisch mit dem Versuchsaufbau 3. Für die meisten Testzwecke wird über Minicom auf den Raspberry Pi zugegriffen. Mit Hilfe von Minicom können Bootmeldungen des Raspberry Pis über die serielle Schnittstelle ausgegeben werden. Um Minicom nutzen zu können, muss es zunächst mit dem Befehl `sudo apt install minicom` installiert werden. Anschließend kann die Kommunikation mit dem Raspberry Pi mit dem Befehl `sudo minicom -D /dev/ttyUSB0` mitgelesen werden.

4.1.1 Testen des Netzwerkboots

Um das Netzwerk-Booting zu testen, wurde die SD-Karte aus dem Raspberry Pi entfernt und mit Hilfe von Minicom überprüft, ob der Raspberry Pi bis zur Benutzeranmeldung kommt.



```
root@sekiro-HP-Laptop-15-db0xxx: /home/sekiro
Starting mosquitto: OK
Starting ntpd...
ntp Started.
Starting wpa_supplicant...
Successfully initialized wpa_supplicant
wpa_supplicant started.
Starting dnsmasq: OK
Starting mosquitto_pub.sh, mosquitto_sub.sh
mosquitto_pub.sh, mosquitto_sub.sh started.
Starting iptables...
    started.

Welcome at LinuXP provided by triritatu83
triritatu83 login: [  8.395732] brcmgenet fd580000.ethernet eth0: Link is Up -x
[  8.897680] brcmfmac: brcmf_set_channel: set chanspec 0x100c fail, reason -52
[  8.905357] brcmfmac: brcmf_set_channel: set chanspec 0x100d fail, reason -52
[  8.913117] brcmfmac: brcmf_set_channel: set chanspec 0x100e fail, reason -52
[  8.920833] brcmfmac: brcmf_set_channel: set chanspec 0xd022 fail, reason -52
[  9.047661] brcmfmac: brcmf_set_channel: set chanspec 0xd026 fail, reason -52
[  9.167704] brcmfmac: brcmf_set_channel: set chanspec 0xd02a fail, reason -52
[  9.287655] brcmfmac: brcmf_set_channel: set chanspec 0xd02e fail, reason -52

Welcome at LinuXP provided by triritatu83
triritatu83 login: 
```

Abbildung 9: Erfolgreicher Netzwerkboot des Raspberry Pis

4.1.2 Testen der Benutzeranmeldung

Auf dem System gibt es 2 Nutzer deren Anmeldung es zu verifizieren gilt.

Benutzer	Passwort
root	root
Sadkowski	Tbelpxtlq

Tabelle 2: Auf dem Target System verfügbare Benutzer

Das Testen der Logins besteht darin, sich nach dem Booten am Raspberry Pi anzumelden.

```
Welcome at LinuXP provided by triritatu83
triritatu83 login: root
Password:
# who
root          console      00:00   Aug 15 17:36:40
# 
```

Abbildung 10: Anmeldung des Benutzers root

```
Welcome at LinuXP provided by triritatu83
triritatu83 login: Sadkowski
Password:
$ who
Sadkowski      console      00:00   Aug 15 17:41:12
$ 
```

Abbildung 11: Anmeldung des Benutzers Sadkowski

Die Abbildungen ?? und ?? zeigen die erfolgreiche Anmeldung auf dem Raspberry Pi. Der aktuelle Benutzer kann mit dem Befehl `who` erneut bestätigt werden.

4.1.3 Testen des SSH-Zugangs

Auf dem System ist der SSH-Server `dropbear` installiert. Um diesen zu testen, kann man sich mit Hilfe des SSH-Befehls auf dem Raspberry Pi anmelden. Mit dem folgenden Befehl kann man sich von einem Terminal aus über SSH mit einem Ziel verbinden.

```
user@host:~$ ssh root@192.168.0.168
```

Nach dem Einloggen mit ssh wurde das System, auf dem man gerade eingeloggt ist, mit dem Befehl `uname -a` erneut validiert 12.

```

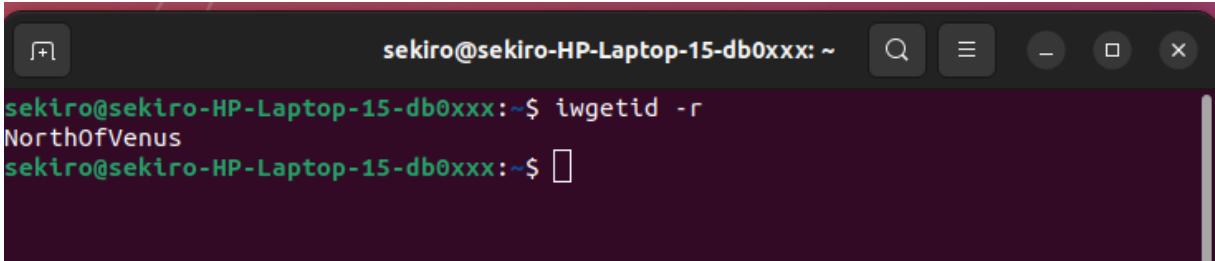
sekiro@sekiro-HP-Laptop-15-db0xxx:~$ ssh root@192.168.0.168
The authenticity of host '192.168.0.168 (192.168.0.168)' can't be established.
ED25519 key fingerprint is SHA256:Osrr/Uz/DbYP7x/qQpiEsfrhMRhBjOXw4xAlQeJprCs.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.168' (ED25519) to the list of known hosts.
root@192.168.0.168's password:
# uname -a
Linux triritatu83 6.3.1-v7l+ #3 SMP Tue Jun 27 15:11:51 CEST 2023 armv7l GNU/Linux

```

Abbildung 12: SSH Verbindung

4.1.4 Testen des Access Pointes

Für die Nutzung des MQTT Brokers ist es notwendig, dass über den Raspberry Pi ein Wlan aufgebaut wird, mit dem sich das Smartphone verbinden kann. Für Testzwecke reicht es aus, ein beliebiges Gerät mit dem Wlan zu verbinden. In diesem Fall wird dafür der Entwicklungsrechner verwendet. Nach dem Hochfahren des Raspberry Pis öffnet man dessen Netzwerkeinstellungen. In der Liste der vorhandenen SSIDs sollte das Wlan **NorthOfVenus** erscheinen. Sobald man nach dem Passwort gefragt wird, sollte man sich mit dem Passwort **xmboqbttxp** verbinden können.



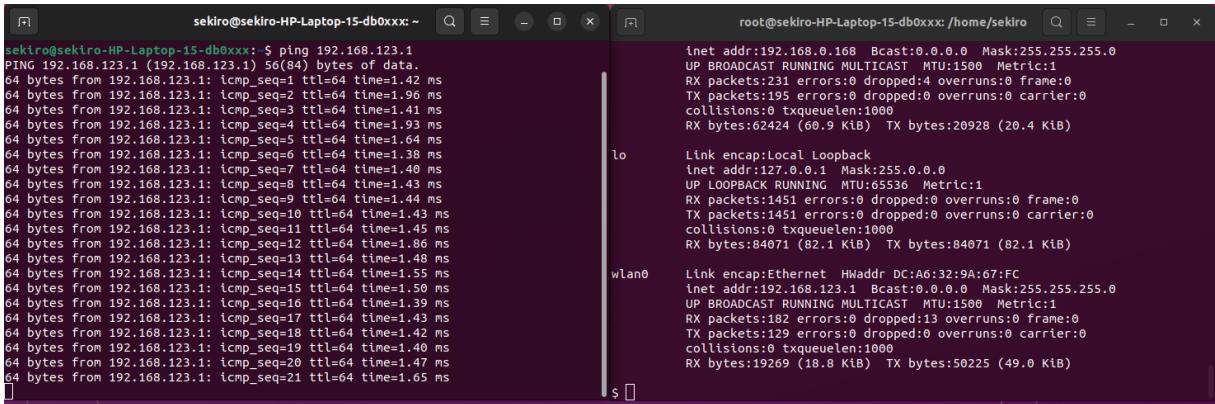
```

sekiro@sekiro-HP-Laptop-15-db0xxx:~$ iwgetid -r
NorthOfVenus
sekiro@sekiro-HP-Laptop-15-db0xxx:~$ 

```

Abbildung 13: Wlan

Um herauszufinden, welches Wlan angeschlossen ist, kann man den Befehl **iwgetid -r** verwenden, wie in Abbildung 13 gezeigt. Um auch die Verbindung zu testen, kann man einmal das Wlan-Interface anpingen. Diese ist unter der Adresse 192.168.123.1 erreichbar. Das Ergebnis sollte wie in der folgenden Abbildung aussehen.



```

sekiro@sekiro-HP-Laptop-15-db0xxx:~$ ping 192.168.123.1
PING 192.168.123.1 (192.168.123.1) 56(84) bytes of data.
64 bytes from 192.168.123.1: icmp_seq=1 ttl=64 time=1.42 ms
64 bytes from 192.168.123.1: icmp_seq=2 ttl=64 time=1.96 ms
64 bytes from 192.168.123.1: icmp_seq=3 ttl=64 time=1.41 ms
64 bytes from 192.168.123.1: icmp_seq=4 ttl=64 time=1.93 ms
64 bytes from 192.168.123.1: icmp_seq=5 ttl=64 time=1.64 ms
64 bytes from 192.168.123.1: icmp_seq=6 ttl=64 time=1.38 ms
64 bytes from 192.168.123.1: icmp_seq=7 ttl=64 time=1.40 ms
64 bytes from 192.168.123.1: icmp_seq=8 ttl=64 time=1.43 ms
64 bytes from 192.168.123.1: icmp_seq=9 ttl=64 time=1.44 ms
64 bytes from 192.168.123.1: icmp_seq=10 ttl=64 time=1.43 ms
64 bytes from 192.168.123.1: icmp_seq=11 ttl=64 time=1.45 ms
64 bytes from 192.168.123.1: icmp_seq=12 ttl=64 time=1.86 ms
64 bytes from 192.168.123.1: icmp_seq=13 ttl=64 time=1.48 ms
64 bytes from 192.168.123.1: icmp_seq=14 ttl=64 time=1.55 ms
64 bytes from 192.168.123.1: icmp_seq=15 ttl=64 time=1.50 ms
64 bytes from 192.168.123.1: icmp_seq=16 ttl=64 time=1.39 ms
64 bytes from 192.168.123.1: icmp_seq=17 ttl=64 time=1.43 ms
64 bytes from 192.168.123.1: icmp_seq=18 ttl=64 time=1.42 ms
64 bytes from 192.168.123.1: icmp_seq=19 ttl=64 time=1.40 ms
64 bytes from 192.168.123.1: icmp_seq=20 ttl=64 time=1.47 ms
64 bytes from 192.168.123.1: icmp_seq=21 ttl=64 time=1.05 ms

root@sekiro-HP-Laptop-15-db0xxx: /home/sekiro
net dev
lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING MTU:1500  Metric:1
        RX packets:231 errors:0 dropped:4 overruns:0 frame:0
        TX packets:195 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:62424 (60.9 KIB)  TX bytes:20928 (20.4 Kib)
wlan0   Link encap:Ethernet HWaddr DC:61:32:9A:67:FC
        inet addr:192.168.123.1  Bcast:0.0.0.0  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1451 errors:0 dropped:0 overruns:0 frame:0
        TX packets:182 errors:0 dropped:13 overruns:0 frame:0
        collisions:0 txqueuelen:1000
        RX bytes:84071 (82.1 Kib)  TX bytes:84071 (82.1 Kib)
```

Abbildung 14: WlanPing

Die Abbildung 14 zeigt den erfolgreichen Ping-Vorgang auf dem Raspberry Pi.

4.1.5 Mosquitto Tests

Um die Funktionalität von `mosquitto_sub` und `mosquitto_pub` zu testen, wurde Mosquitto auf dem Entwicklungsrechner installiert `sudo apt install mosquitto`. Zur Verbindung mit dem Broker wurde der Entwicklungsrechner dann mit dem vom Raspberry Pi bereitgestellten Wlan verbunden. Die erfolgreiche Übertragung der Daten ist in den Abbildungen 15 und 16 dargestellt. Der Raspberry Pi sendet die Nachrichten regelmäßig alle 15 Sekunden.

```
sekiro@sekiro-HP-Laptop-15-db0xxx:~$ mosquitto_pub -h 192.168.123.1 -p 1883 -t appl/frequency_set -m "2"
sekiro@sekiro-HP-Laptop-15-db0xxx:~$ 

root@sekiro-HP-Laptop-15-db0xxx:/home/sekiro$ mosquitto_pub.sh, mosquitto_sub.sh started.
Starting iptables...
started.

Welcome at LinuxXP provided by triritatu83
triritatu83 login: [ 8.315454] bcmenet fd580000.ethernet eth0: Link is Up -x
[ 8.767453] brcmffmac: brcmff set_channel: set chanspec 0x10c fall, reason -52
[ 8.775192] brcmffmac: brcmff set_channel: set chanspec 0x10d fall, reason -52
[ 8.782913] brcmffmac: brcmff set_channel: set chanspec 0x10e fall, reason -52
[ 8.790648] brcmffmac: brcmff set_channel: set chanspec 0xd022 fall, reason -52
[ 8.917382] brcmffmac: brcmff set_channel: set chanspec 0xd026 fall, reason -52
[ 9.037403] brcmffmac: brcmff set_channel: set chanspec 0xd02a fall, reason -52
[ 9.157394] brcmffmac: brcmff_set_channel: set chanspec 0xd02e fall, reason -52

Welcome at LinuxXP provided by triritatu83
triritatu83 login: root
Password:

Login incorrect
triritatu83 login: root
Password:
# mosquitto_sub -h 192.168.123.1 -p 1883 -t appl/frequency_set
2
```

Abbildung 15: Links der `mosquitto_pub` Befehl mit dem entsprechenden `mosquitto_sub` Befehl rechts.

Abbildung 16: Die Abbildung zeigt die Erfolgreiche Ankunft der Daten über die Abonnierten Topics.

4.2 Testen des Treibers

Um den Gerätetreiber zu testen, wird auf diesen lesend und schreibend zugegriffen. Das Schreiben erfolgt mit dem Befehl “echo”. Die LED kann mit einer minimalen Frequenz von 1Hz und einer maximalen Frequenz von 50Hz blinken. Eine Frequenz von 0 schaltet die Led aus und alles darüber schaltet die Led dauerhaft an. Da das Lesen und Schreiben der Frequenz ein kritischer Abschnitt ist, wurde dieser mit dem Datentyp `atomic` geschützt. Zusätzlich werden auch ungültige Eingaben wie negative Werte, Werte über dem Maximum und Werte die keiner Zahl entsprechen wie `blub` oder `k` geprüft. Die Überprüfung der Frequenz erfolgt durch Lesezugriffe auf den Treiber.

geschriebener Wert	erwartete Ausgabe
-	0 (default wert ohne schreiben)
-5	0 (Wert unter Minimum)
99	51 (Wert über Maximum)
12	12 (richtiger Wertebereich)
"2"	2 (richtiger Wertebereich)
"k"	2 (letzte Eingabe ungültiger Wert)
"blub"	2 (letzte Eingabe ungültiger Wert)

Tabelle 3: Eingaben an den Treiber zusammen mit den zu erwartenden Ausgaben

```
# echo $(head -c2 /dev/led_one_ad)
0
# echo -5 > /dev/led_one_ad
# echo $(head -c2 /dev/led_one_ad)
0
# echo 99 > /dev/led_one_ad
# echo $(head -c2 /dev/led_one_ad)
51
# echo 12 > /dev/led_one_ad
# echo $(head -c2 /dev/led_one_ad)
12
# echo "2" > /dev/led_one_ad
# echo $(head -c2 /dev/led_one_ad)
2
# echo "k" > /dev/led_one_ad
sh: write error: Invalid argument
# echo $(head -c2 /dev/led_one_ad)
2
# echo "blub" > /dev/led_one_ad
sh: write error: Invalid argument
# echo $(head -c2 /dev/led_one_ad)
2
# [Richtiges Wertebereich]
```

Abbildung 17: Ein und Ausgaben der Treiberfrequenz

4.3 Test des Gesamtsystems

Das Testen des Gesamtsystems umfasst im Wesentlichen die Komponententests im Zusammenspiel mit dem Smartphone und nicht mit dem Entwicklungsrechner.

4.3.1 Testaufbau

Der Testaufbau entspricht dem Versuchsaufbau siehe ([3.2](#)).

4.3.2 Testdurchführung

Die Durchführung des Systemtests besteht darin, zunächst das Booten des Raspberry Pi mit Hilfe von Minicom zu überwachen. Nach dem Login des Benutzers wird das Smartphone mit dem Wlan des Raspberry Pi verbunden. Sobald dieses verbunden ist, wird als nächstes die MQTT App gestartet und mit dem Broker verbunden. Sobald dieser verbunden ist, wird die MQTT Kommunikation getestet, indem das im Kapitel [3.9](#) eingerichtete Dashboard überwacht wird. Dort sollten sich die Kacheln nach wenigen Sekunden aktualisiert haben. Der Raspberry Pi veröffentlicht die MQTT-Nachrichten der Topics in einem Intervall von 15 Sekunden. Abschließend wird die Funktionalität der LED getestet. Dazu wird der Range Slider auf dem Dashboard auf einen beliebigen Wert gesetzt. Sobald dieser eingestellt ist, fragt die App noch einmal explizit nach, ob der eingestellte Wert wirklich veröffentlicht werden soll. Nach der Bestätigung kann das Verhalten der LED visuell kontrolliert werden.

4.3.3 Testergebnisse

Im Rahmen der durchgeführten Komponenten- und Systemtests wurden die an das Projekt gestellten Anforderungen untersucht. Dabei konnte erfolgreich nachgewiesen werden, dass die Kernfunktionalitäten wie Booten, Login, Wlan-Anmeldung, SSH, MQTT-Kommunikation sowie Lesen und Schreiben auf den Treiber die Anforderungen in vollem Umfang erfüllen. Auch die funktionalen Anforderungen an das User Interface konnten erfolgreich nachgewiesen werden. Dies beinhaltet das erfolgreiche Verbinden der Applikation mit dem Broker sowie das Empfangen und Versenden von MQTT Nachrichten.

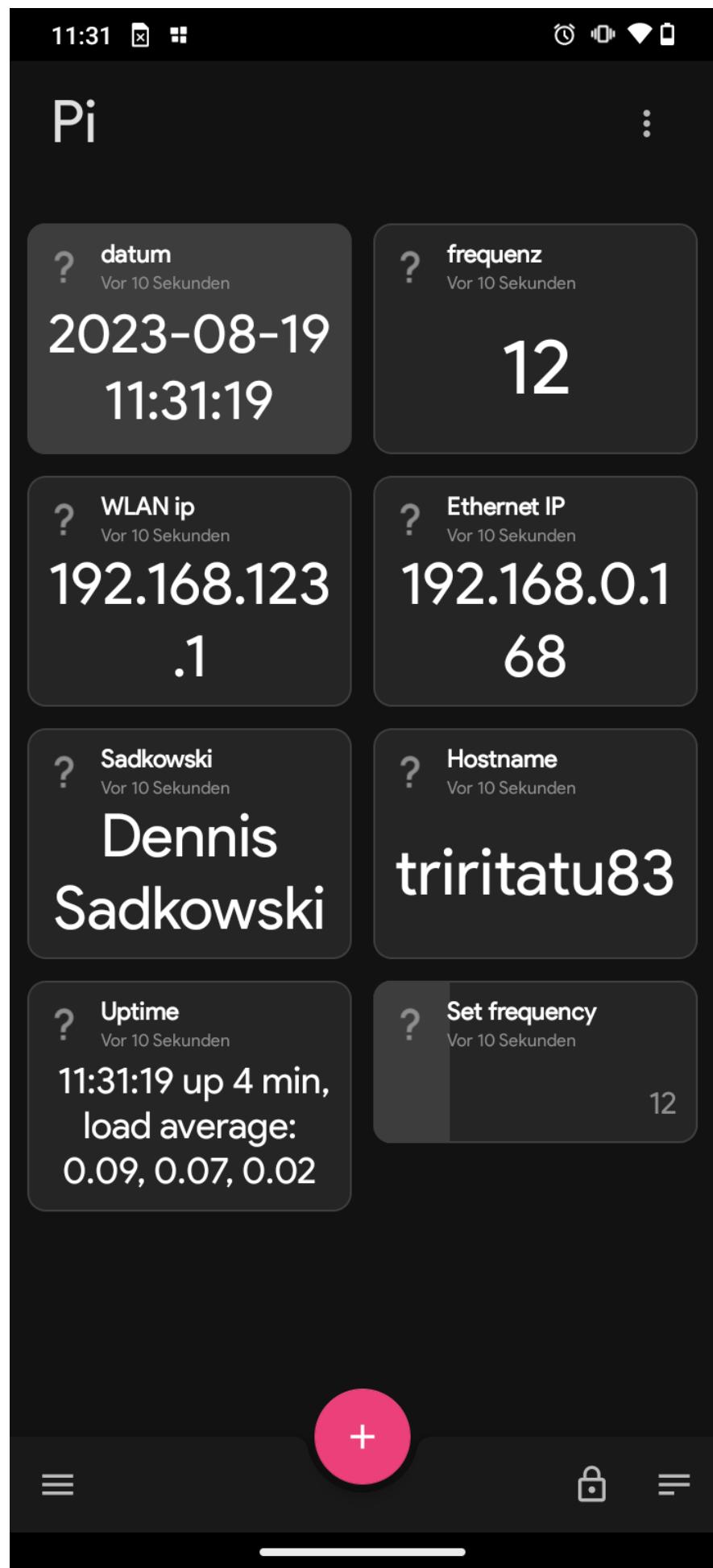


Abbildung 18: Fertiges MQTT Dashboard mit angkommenden Daten
18

5 Zusammenfassung

Das vorliegende Projekt beschäftigte sich mit der Umsetzung einer IoT-Anwendung, bei der eine LED mit Hilfe eines Smartphones gesteuert werden kann. Herzstück des Projekts war ein Raspberry Pi 4, der einerseits einen Wlan Access Point zur Verfügung gestellt hat und andererseits einen MQTT Broker, der die Kommunikation mit dem Smartphone ermöglicht hat. Der Raspberry Pi wurde über einen Switch und eine statische IP-Adresse in das Heimnetzwerk integriert. Dadurch war es möglich, das System über ssh aus der Ferne zu administrieren. Das Projekt hat einen ersten Einblick in die Möglichkeiten von eingebetteten Systemen gegeben. Aufbauend auf den in diesem Projekt gewonnenen Erkenntnissen lassen sich mehr oder weniger sinnvolle Projekte für zu Hause realisieren, wie z.B. ein System, das die Feuchtigkeit der Blumen mittels eines Feuchtigkeitssensors überwacht und über MQTT mitteilt, wann wieder gegossen werden muss. In einer Welt, in der die Vernetzung von Geräten und das Internet der Dinge eine immer wichtigere Rolle im täglichen Leben spielen, hat dieses Projekt einen wertvollen Einblick hinter die Kulissen gegeben und ein besseres Verständnis ermöglicht.

6 Literaturverzeichnis