

# Projektaufgabe Embedded Systems SS2024

Tim Krüger, 1197031

13.08.2024

## Inhaltsverzeichnis

<b>1 Allgemeines</b>	<b>2</b>
1.1 Persönliche Angaben . . . . .	2
1.2 Eigenständigkeitserklärung . . . . .	2
<b>2 Einleitung</b>	<b>3</b>
2.1 Motivation . . . . .	3
2.2 Aufgabenstellung . . . . .	3
<b>3 Installationsanleitung</b>	<b>4</b>
3.1 Projektstruktur . . . . .	4
3.2 Netzwerkboot . . . . .	4
3.3 Buildroot . . . . .	5
3.3.1 Installationshinweise . . . . .	5
3.4 Kernel . . . . .	5
3.4.1 Installationshinweise . . . . .	5
3.5 WLAN-AP . . . . .	5
3.6 Hardwareaufbau . . . . .	6
3.7 Gerätetreiber . . . . .	7
3.8 MQTT-Konfiguration . . . . .	7
3.9 Anderes . . . . .	7
<b>4 Systemtest</b>	<b>7</b>
4.1 Testplan . . . . .	7
4.2 Komponententest . . . . .	7
4.2.1 Boot und Entwicklung . . . . .	7
4.2.2 LED-Treiber . . . . .	7
4.2.3 WLAN . . . . .	7
4.2.4 MQTT . . . . .	8
4.3 Test des Gesamtsystems . . . . .	9
<b>5 Zusammenfassung</b>	<b>9</b>

# 1 Allgemeines

## 1.1 Persönliche Angaben

Name: Tim Krüger

Matrikelnummer: 1197031

Studiengang: Master Informatik

Datum: 13.08.2024

## 1.2 Eigenständigkeitserklärung

Eidesstattliche Erklärung  
zur Projektarbeit Eingebettete Systeme im SS2024

Name:

Matrikelnummer:

Ich versichere durch meine Unterschrift, dass die vorgelegte Arbeit ausschließlich von mir erstellt und verfasst wurde. Es wurden keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt.

Krefeld, 13.08.24

Ort, Datum

Krüger

Unterschrift

Abbildung 1: Eigenständigkeitserklärung

## **2 Einleitung**

### **2.1 Motivation**

### **2.2 Aufgabenstellung**

### 3 Installationsanleitung

#### 3.1 Projektstruktur

Als Teil dieses Projekts werden alle notwendigen Dateien (Dokumentation, Configs, Skripte etc.) mitgeliefert. Die Struktur des Dateibaums ist im folgenden dargestellt:

```
Dev/ESY/
|-- buildroot/
|-- configs/
|-- devicetree/
|-- docs/
|   |-- res/
|-- kernel/
|-- modules/
|-- scripts/
|-- target/
`-- txts/
```

- *Dev/ESY/*: Das Root-Verzeichnis des Projekts
  - Bei einer anderen Verzeichnisstruktur müssen die Skripte *post-build.sh* und *post-image.sh* entsprechend angepasst werden
- *buildroot/*: Das Buildroot-Verzeichnis.
  - Genaue Hinweise folgen in Kapitel 3.3
- *configs/*: Verzeichnis für die Buildroot- und Kernel-Konfigurationsdateien
- *devicetree/*: Verzeichnis für den Device-Tree-Blob zur Ansteuerung der GPIOs
- *docs/*: Dokumentation des Projekts inkl. dem Abbildungsverzeichnis *res/*
- *kernel/*: Bereits vorkompilierte Kernel
  - Genaue Hinweise folgen in Kapitel 3.4
- *modules/*: Der Treiber bzw. das Kernelmodul zur Ansteuerung der LED
  - Genaue Hinweise folgen in Kapitel 3.7
- *scripts/*: Verschiedene Skripte (Teil des Entwicklungsprozesses)
- *target/*: Verschiedenen Skripte und Konfigurationsdateien, welche auf den Raspberry Pi kopiert werden
- *txt/*: Verschiedene Textdateien, hauptsächlich zur initialen TFTP-Konfiguration

#### 3.2 Netzwerkboot

Für einen schnellen Entwicklungsprozess (Iterationsgeschwindigkeit + Deployment) wird Netzwerkboot via TFTP verwendet. Der Raspberry Pi zieht sich hierbei, bei korrekter Konfiguration, alle notwendigen Dateien des Bootprozesses von einem TFTP-Server, welcher auf dem Host-Rechner läuft.

Ein solches Setup kann mit den folgenden Schritten ans Laufen gebracht werden:

- Installation und Starten eines TFTP-Servers auf der Host-Maschine
  - Monitoren des Outputs bspw. via: *tail -f /var/log/syslog | grep tftp*
  - Empfehlung: Extensives Logging aktivieren
- Originale Pi4-Bootfiles herunterladen und ins TFTP-Verzeichnis kopieren
- Bootloader des Raspberry Pi's **muss** für Netzwerkboot angepasst werden
  - Dafür muss der Pi entsprechend geflashed werden
- Anpassung der Netzwerkschnittstellen
  - Zuweisung einer statischen IP ans Ethernet-Interface des Host-PC
  - Zuweisung und Konfiguration des Ethernet-Interface auf dem Pi (erfolgt über die Datei */target/interfaces*)

Es ist außerdem empfehlenswert das serielle Interface des Pi's für Output und Debugging zu verwenden. Die notwendigen Konfigurationsdateien werden bereitgestellt:

- Kopieren der Konfigurationsdateien
  - */txt/cmdline.txt* nach */srv/tftp/*
  - */txt/config.txt* nach */srv/tftp/*
- Installation eines beliebigen Terminalemulators, z.B. *Minicom*
  - Starten bspw. via: *sudo minicom -D /dev/ttyUSB0*

### 3.3 Buildroot

Als Systembuilder wird in diesem Projekt *Buildroot* verwendet. *Buildroot* ist im Endeffekt eine Sammlung von Makefiles, welche einem in einem kernelähnlichen Interface verschiedene Konfigurationsoptionen bieten. Buildroot wurde in diesem Projekt ausschließlich zum Bau des Userlands verwendet, der Kernel wurde *per Hand* kompiliert. Die exakten Konfigurationsschritte werden an dieser Stelle ausgelassen, da die Buildroot-Config zur Verfügung gestellt wird.

#### 3.3.1 Installationshinweise

- Buildroot installieren
  - Git-Repository nach *buildroot/* clonen
  - *config/config.buildroot* nach *buildroot/* kopieren
  - *make menuconfig* aufrufen und Auswahl speichern
- Durch ein anschließendes Aufrufen von *make* wird das Userland gebaut
- Die Skripte *post-build.sh* und *post-image.sh* klinken sich in diesen Prozess ein und kopieren die notwendigen Dateien an die richtigen Stellen im Image des TFTP-Servers
- **Achtung:** Auf korrekte Pfade achten
  - Die Pfade zu den Skripten können per *make menuconfig* geändert werden
  - Die Pfade innerhalb der Skripte müssen natürlich ebenfalls stimmen (Root-Verzeichnis oben im Skript als Variable hinterlegt)
  - Extensives Logging ist aktiviert und sollte an dieser Stelle schnell Auskunft geben können

### 3.4 Kernel

Ein bereits crosskompilierter Kernel, welcher WLAN- und Modulefähig ist, findet sich im Verzeichnis *kernel/*. Es handel sich dabei um einen Kernel mit der Version 6.6.4. Dieser Kernel lief in meinen Tests am besten. Speziell die neueren Kernelversionen  $\geq 6.9$  haben bei mir Probleme gemacht. Bspw. lief der Arbeitsspeicher auf meiner Maschine beim Kompilieren von Kernel 6.9.1 voll, was schließlich zu einem Crash führte.

#### 3.4.1 Installationshinweise

Falls Sie einen eigenen Kernel für dieses Projekt kompilieren wollen, stehen alle notwendigen Dateien zur Verfügung.

- Clonen des entsprechenden Kernels von Git
- Hinterlegen der notwendigen Umgebungsvariablen durch: *source /scripts(exports).sh*
- *config/config.linux* ins Kernel-Verzeichnis kopieren
- Kompilieren durch: *time make -j ihr\_name dtbs modules*
- Der neue Kernel muss dann noch namentlich im *post-image.sh* Skript hinterlegt werden
- Beim nächsten Aufruf von *make* innerhalb von *Buildroot* wird dieser dann kopiert

### 3.5 WLAN-AP

Ein Teil der Aufgabenstellung war die Konfiguration des Raspberry Pi's als WLAN Accesspoint. Der Kernel und das Userland wurden dafür entsprechend präpariert. Aufgespannt wird das WLAN *VimIsTheBest*. Skripte und Konfigurationsdateien finden sich im Verzeichnis *target/*. Alle Dateien werden nach *init.d* kopiert und automatisch beim Bootvorgang gestartet.

Die WLAN-AP-Konfiguration umfasst:

- WPA
- DNS und DHCP
- Firewall mit NAT und IP-Forwarding

Unter Umständen taucht beim Booten die Fehlermeldung *brcmfmac: brcmf\_set\_channel: set chanspec 0x100d fail, reason -52* auf. Diese wird von der WLAN-Hardware ausgelöst.

- Dabei handelt es sich um einen bekannten Bug, bisher ohne richtigen Fix:
  - Siehe z.B. [<https://github.com/raspberrypi/linux/issues/6049>]
- Dieser betrifft sowohl Raspberry Pi's der Version 4 als auch der Version 5
- Er tritt, laut meiner Recherche, seit den Kernelversionen  $\geq 6.3$  auf
- Der Bug hat aktuell keinen Effekt auf uns, sollte aber beobachtet werden

### 3.6 Hardwareaufbau

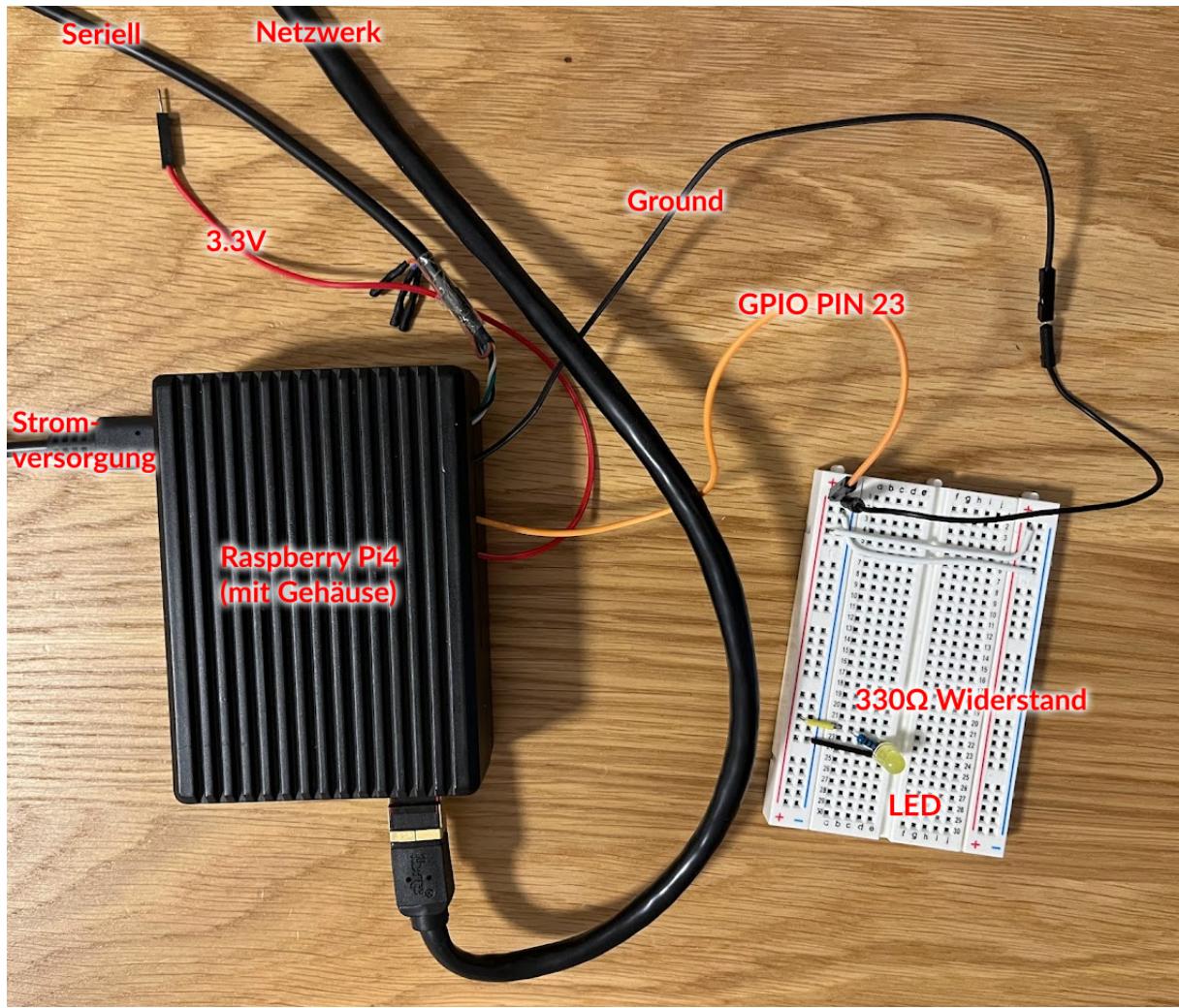


Abbildung 2: Hardwareaufbau

Der Hardware- bzw. Schaltungsaufbau kann *Abbildung 2* entnommen werden. Es handelt sich dabei um eine sehr einfache Schaltung welche Demonstrationszwecken dient. Folgende Aspekte können durch den Aufbau demonstriert werden:

- Logging via serieller Schnittstelle
- Booting via Netzwerk (TFTP)
- Ansteuerung eines GPIO Pin's mittels des Device-Tree's
  - Später: Integration in Kernelmodul und Fernsteuerung per MQTT
- Der Pi ist in meinem Fall noch mit einem optionalen Gehäuse ausgestattet, welches Schutz- und Kühlungszwecken dient

### 3.7 Gerätetreiber

### 3.8 MQTT-Konfiguration

### 3.9 Anderes

## 4 Systemtest

### 4.1 Testplan

### 4.2 Komponententest

#### 4.2.1 Boot und Entwicklung

#### 4.2.2 LED-Treiber

#### 4.2.3 WLAN

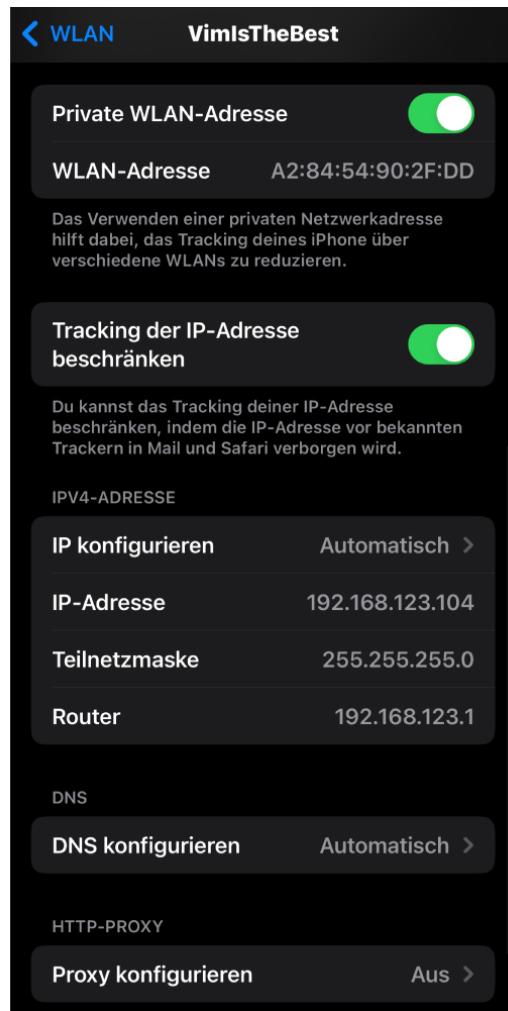


Abbildung 3: WLAN-Test

#### 4.2.4 MQTT

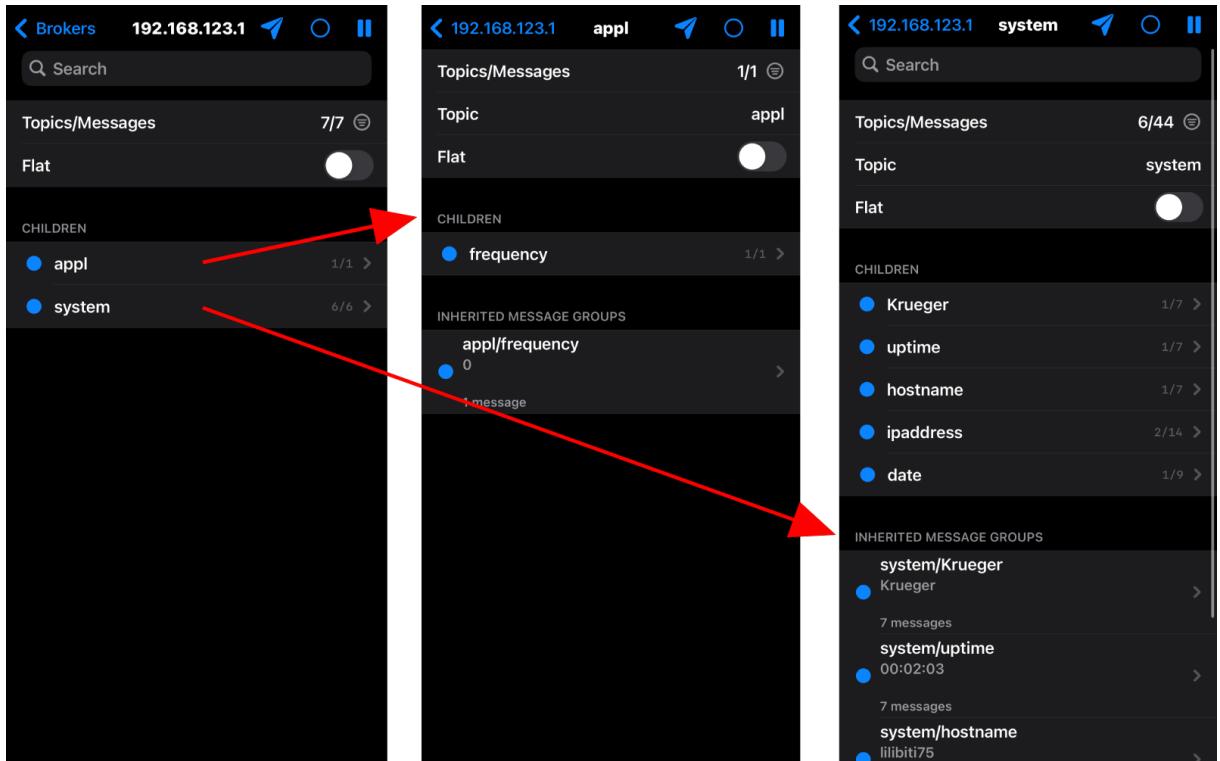
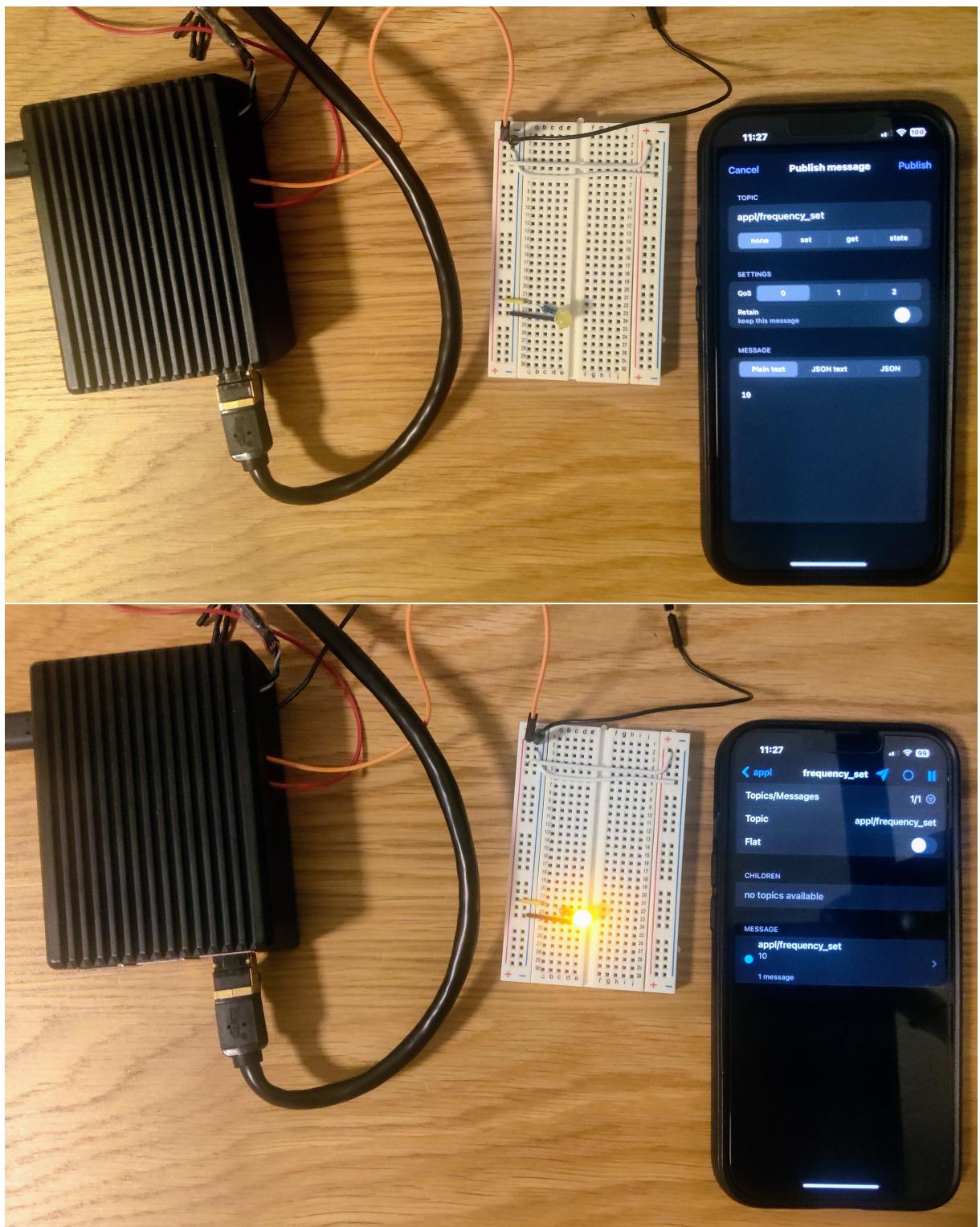


Abbildung 4: MQTT-Test

#### 4.3 Test des Gesamtsystems



#### 5 Zusammenfassung