

# **Telegram-controlled ultra low power infodisplay**

---

Bachelorarbeit

Name des Studiengangs  
Angewandte Informatik  
**Fachbereich 4**  
vorgelegt von  
Harun Dastekin

Datum:

Berlin, 26.08.2024

Erstgutachter: Prof. Alexander Huhn  
Zweitgutachter: Prof. Thomas Schwotzer

# **Vorwort**

## **0.1 Bericht**

Hintergrund ist die Ablösung traditioneller Papieraushänge zur Darstellung der Raumverfügbarkeit auf energieeffiziente weise.

Das Ziel dieser Abschlussarbeit ist die Frage zu klären:  
„Lohnt sich der Anwendungsfall?“

# Kurzbeschreibung

Das Ziel des Projektes ist ein flacher Prototyp mit 7000 mAh LIPO Akku befestigt zwischen einer Hobbyglaskonstruktion mit Schraubmuttern, eines Telegram controlled ultra low power eink display, mit ausgestelltem e-Paper ESP32 Driver Board kurz angeschlossen an einen E-Ink Display mit Laderegler als Infodisplay an der Wand, neben der Tür eines Raumes der HTW.

**Schlagworte:** E-Ink, Esp32, LiPo, BMS, Telegram, Bitmap, Micropython, MQTT, Node-RED, Grafana

# Inhaltsverzeichnis

0.1	Bericht . . . . .	i
<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Hintergrund der Arbeit . . . . .	1
1.2	Vorstellung der Idee . . . . .	1
1.3	Motivation und Zielsetzung . . . . .	2
1.4	Überblick über die Dokumentation . . . . .	2
<b>2</b>	<b>Grundlagen</b>	<b>4</b>
2.1	Hardwarebestandteile des Infodisplay . . . . .	5
2.1.1	Mikrocontroller ESP32-Driver . . . . .	5
2.1.2	Display . . . . .	10
2.1.3	Spannungssensor und Stromsensor . . . . .	10
2.1.4	Laderegler und Lithium Polymer Batterie . . . . .	11
2.2	Software . . . . .	12
2.2.1	MicroPython . . . . .	12
2.2.2	Micropython Generic Firmware . . . . .	18
2.2.3	Hello World - Blinky - REPL . . . . .	20
<b>3</b>	<b>Analyse</b>	<b>21</b>
3.1	Anforderungen . . . . .	22
3.1.1	MicroPython und Thonny . . . . .	22
3.1.2	Ein- und Ausschalten . . . . .	22
3.1.3	Laufzeit . . . . .	22
3.1.4	Schaltung . . . . .	23
3.1.5	Pinbelegung . . . . .	24
3.1.6	Wartbarkeit und Schneller Einstieg . . . . .	24
3.1.7	Aktualisieren . . . . .	24
3.1.8	Partielles Aktualisieren . . . . .	24

3.1.9	Bitmap an Infodisplay . . . . .	25
3.1.10	Systemarchitektur . . . . .	25
3.1.11	Powerbank . . . . .	25
3.1.12	Leistung . . . . .	25
3.1.13	Tests . . . . .	26
3.1.14	Serielle Schnittstelle . . . . .	26
3.2	Planung der Stromverbrauchs Messung . . . . .	26
<b>4</b>	<b>Konzeption</b>	<b>27</b>
4.1	Technologieauswahl . . . . .	27
4.2	MicroPython und Thonny . . . . .	28
4.3	Laufzeit . . . . .	30
4.4	Schaltung . . . . .	31
4.4.1	Bauteilliste . . . . .	32
4.4.2	Technische Zeichnungen . . . . .	35
4.4.3	Pin-Belegungstabellen . . . . .	36
4.4.4	Strom-, Spannung-, und Leistungskennlinien . . . . .	41
4.5	Pinbelegung . . . . .	41
4.6	Wartbarkeit und Schneller Einstieg . . . . .	41
4.7	Aktualisieren . . . . .	42
4.8	Systemarchitektur . . . . .	42
4.9	Powerbank . . . . .	47
4.10	Leistung . . . . .	47
4.11	Tests . . . . .	47
<b>5</b>	<b>Implementierung</b>	<b>49</b>
5.1	Hardware . . . . .	49
5.1.1	Infodisplay . . . . .	49
5.1.2	Sensor Messung . . . . .	51
5.1.3	Messumgebung . . . . .	52
5.2	Software . . . . .	56
5.2.1	Näheres zum Driver Code . . . . .	59
5.2.2	Display . . . . .	60
5.2.3	Node-Red Flow . . . . .	71
5.2.4	Sensor . . . . .	77
5.2.5	Grafana Dashboard . . . . .	79

<b>6 Test</b>	<b>80</b>
6.1 Testplanung . . . . .	80
6.2 Software Testumgebung . . . . .	84
6.3 Testdurchführung . . . . .	85
6.4 Testergebnisse . . . . .	93
6.5 Ausblick auf die Testlaufbahn des Infodisplay . . . . .	93
<b>7 Evaluation</b>	<b>94</b>
7.1 MicroPython und Thonny . . . . .	94
7.2 Ein- und Ausschalten . . . . .	95
7.3 Laufzeit . . . . .	95
7.4 Schaltung . . . . .	97
7.5 Pinbelegung . . . . .	99
7.6 Wartbarkeit und Schneller Einstieg . . . . .	99
7.7 Aktualisieren . . . . .	101
7.8 Partielles Aktualisieren . . . . .	102
7.9 Bitmap an Infodisplay . . . . .	102
7.10 Systemarchitektur . . . . .	103
7.11 Powerbank . . . . .	103
7.12 Leistung . . . . .	104
7.13 Tests . . . . .	104
7.14 Serielle Schnittstelle . . . . .	104
<b>8 Fazit</b>	<b>105</b>
8.1 Zusammenfassung der Ergebnisse . . . . .	105
8.2 Prägende Entwicklungsphasen des Projektes . . . . .	106
8.3 Ausblick . . . . .	109
<b>Abbildungsverzeichnis</b>	<b>I</b>
<b>Tabellenverzeichnis</b>	<b>IV</b>
<b>Source Code Content</b>	<b>V</b>
<b>Glossar</b>	<b>VII</b>
<b>Literaturverzeichnis</b>	<b>IX</b>
<b>Onlinereferenzen</b>	<b>X</b>

<b>Bildreferenzen</b>	<b>XIII</b>
<b>Anhang A</b>	<b>XIV</b>
A.1  Beispiel . . . . .	XIV
<b>Eigenständigkeitserklärung</b>	<b>XV</b>

# Kapitel 1

## Einleitung

### 1.1 Hintergrund der Arbeit

Diese Arbeit dient der Erlangung des akademischen Grades Bachelor of Science (B. Sc.) des Studiengangs „Angewandte Informatik“ der HTW Berlin.

### 1.2 Vorstellung der Idee

Die Kernidee des Projekts ist die **Ablösung** traditioneller Raumbelegungspläne aus Papier vor Räumen an Hochschulen durch **moderne**, energieeffiziente Infodisplays. Das Infodisplay erfüllt den Zweck, die Vermittlung von Raumbelegungsinformationen dynamischer, zugänglicher und nachhaltiger zu gestalten.

Aktuell nutzt der Fachbereich 4 der HTW Berlin **A4 Papier**, um Raumbelegungspläne darzustellen. Diese Aushänge bieten Studierenden und Lehrbeauftragten einen Überblick über die Belegung der Räume. Einmal ausgedruckt, bleiben diese Informationen für ein halbes Jahr unverändert an den Türen hängen. Dieses *statische* System birgt mehrere Nachteile:

**Dynamische Nutzung:** Lehrbeauftragte haben keine Möglichkeit, ihr Sprechzimmer kurzfristig und flexibel zu nutzen. Ein Infodisplay, das über eine Messaging-Plattform aktualisiert werden kann, ermöglicht, Informationen wie „**Ich befinde mich derzeit im Labor C625**“, „**Bin gleich wieder da!**“, oder **Änderungen der Sprechzeiten** effektiv zu kommunizieren.

**Raumverfügbarkeit:** Studierende verbringen oft Ihre *Zeit* zwischen den Vorlesungen damit, nach einem freien Raum für Gruppenarbeit oder Selbststudium zu suchen. Die aktuelle Praxis erfordert, dass der Belegungsplan manuell auf dem neuesten Stand gehalten wird, was zu Unsicherheiten und ineffizienter Raumnutzung führt.

**Raumnutzung:** Die unterschiedliche Nutzung der Menschen auf dem Campus Wilhelmshof und Treskowallee unterstreichen das Problem der Raumauffindung. Eine effiziente Raumplanung durch Infodisplays könnte klarstellen, ob Räume belegt sind, wodurch eine bessere Auslastung der Lernräume erzielt und Studierenden geholfen wird, geeignete Lernorte zu finden.

Eine Anbindung an das Lehrveranstaltungs- und Informationssystem der HTW Berlin (LSF) wäre für die Raumplanung ideal, fällt jedoch aufgrund von Änderungen in den Zugangsdaten außerhalb des Rahmens dieser Arbeit. Nichtsdestotrotz bietet die Verwendung einer **Display-Lösung**, ähnlich jener in Supermarktketten zur Preisauszeichnung, gepaart mit einer **Messaging-Dienst-Anbindung**, eine sinnvolle Alternative. Besonders der Infodisplay würde sich aufgrund des *geringen Stromverbrauchs* als optimal erweisen, um die Nachhaltigkeit und Effizienz der Informationsübermittlung zu gewährleisten.

## 1.3 Motivation und Zielsetzung

Das Hauptziel ist die Schaffung eines Infodisplay, anhand von Anforderungen des Fachbereich 4 der HTW Berlin.

Somit ist das Ziel dieser Abschlussarbeit: **Ein interaktives Infodisplay mit geringem Stromverbrauch zu bauen um die Frage zu beantworten ob sich der Anwendungsfall für die Hochschule lohnt.**

## 1.4 Überblick über die Dokumentation

Im folgenden werden nun **Grundlagen** und Vertiefungen zu **micropython** beschrieben. Daraufhin folgt die **Analyse** der Anforderungen zum Projekt und die **Konzeptionierung** dieser. Nach der **Implementierung** auf Seite 49, wird eine **Mess-** und **Testumgebung** erzeugt. Diese und die Anforderungen aus dem Kapitel 3 werden in der **Evaluation** 94

überprüft. Und im **Fazit** auf Seite 105 Untersucht und Bewertet.

Dieses Projekt setzt einen guten Baustein zu Abbildung fortschrittlicher Technologien der Industrie zum bildungsbezogenen Umfeld.

# Kapitel 2

## Grundlagen

Der Begriff „resource constrained environment“ (Ressourcenbeschränkte Umgebung) wird verwendet um eine Hardware Entwicklungs-Plattform zu beschreiben, die nur sehr wenig Gestaltungsspielraum zulässt<sup>1</sup>.

Dementsprechend sind Entwickler durch die verfügbaren Ressourcen beschränkt und es werden dadurch oft minimalistische Ansätze gewählt.

Drahtlose Sensornetze wie WSN, RFID oder das IOT sind Beispiele für ressourcenbeschränkte Umgebungen laut [KONG20151].

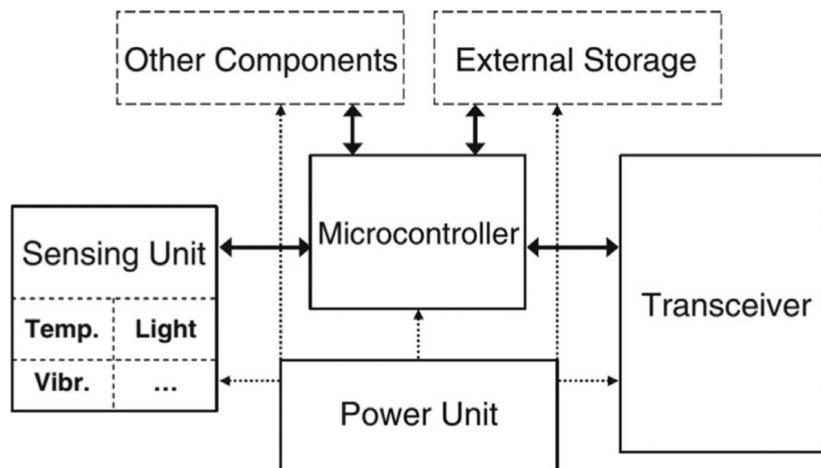


Abbildung 2.1: Eine Microcontroller Systemarchitektur für ein Drahtloses Sensornetz[KONG20151]

<sup>1</sup>Beispiele hierfür sind: Batterilaufzeit, Hardware-Speicher, Rechenleistung, Kommunikations-Bandbreite usw.

In Abbildung 2.1 ist eine Systemarchitektur einer solchen Microcontroller-Entwicklungs-Platform innerhalb eines Drahtlosen Sensornetzwerkes zu sehen.

Der Infodisplay soll Teil eines solchen Drahtlosen Sensornetzwerkes werden. Nämlich dem der HTW und damit auch des Fachbereich 4. Und um dies zu ermöglichen, soll eine Auswertung(im Fazit auf Seite 105) zeigen, ob der Infodisplay, eine ressourcenschonende Anwendung für die Raumbelegung ist.

Dazu sind dem Infodisplay Anforderungen gestellt (siehe dazu Kapitel 3) die es zu erfüllen gilt. Im Kapitel Konzeption werden Lösungsansätze dazu beschrieben. Um diese dann im Kapitel 7 auszuwerten.

Der Grundlagenteil beschäftigt sich mit Hardware- sowie Softwareteil des Infodisplay und seiner Anwendungsumgebung.

Die Hardware des Infodisplay ist zusammengesetzt aus Microcontroller, Display, Laderegler, Akku und einem Schalter S1 zum Ein- und Ausschalten der Anwendung.

Die Software des Infodisplay ist in **micropython** geschrieben. Die User Interface und damit das Frontend, ist ein Telegram-Bot. Das Backend besteht aus einer Node-RED Instanz, mit einer Influx-Datenbank zur Datenpersistierung und einer Grafana Datenvisualisierung der Message Queuing Telemetry Transport-Logs in Tabellen und Messreihen als Graphen. Die drahtlose Kommunikation geschieht im Wireless LAN (Local Area network) über Message Queuing Telemetry Transport.

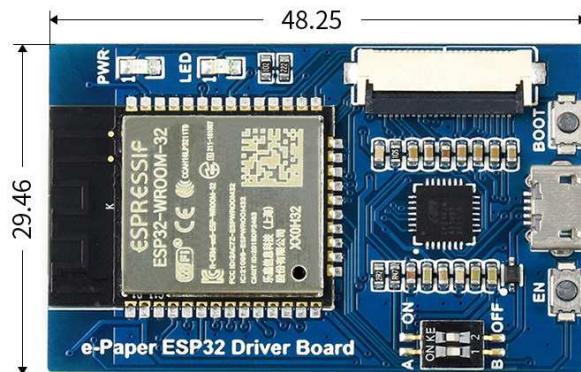
## 2.1 Hardwarebestandteile des Infodisplay

### 2.1.1 Mikrocontroller ESP32-Driver

Der Mikrocontroller für den Infodisplay ist der Esp32-Wroom-32 von Espressif[Sys22a] auf einem **e-Paper ESP32 Driver Board** von Waveshare[Wav24b]

Resistor (Display Config)	Screen
0.47R (A)	2.13inch e-Paper (D), 2.7inch e-Paper, 2.9inch e-Paper (D)
	3.7inch e-Paper, 4.01inch e-Paper (F), 4.2inch e-Paper
	4.2inch e-Paper (B), 4.2inch e-Paper (C), 5.65inch e-Paper (F)
	5.83inch e-Paper, 5.83inch e-Paper (B), 7.3inch e-Paper (G)
	7.3inch e-Paper (F), 7.5inch e-Paper, 7.5inch e-Paper (B)
	1.64inch e-Paper (G), 2.36inch e-Paper (G), 3inch e-Paper (G)
	4.37inch e-Paper (G)
3R (B)	1.54inch e-Paper, 1.54inch e-Paper(B), 2.13inch e-Paper
	2.13inch e-Paper (B), 2.66inch e-Paper, 2.66inch e-Paper (B)
	2.9inch e-Paper, 2.9inch e-Paper (B)

Abbildung 2.3: Schalterstellung zur Display-Art[24b]

Abbildung 2.2: Die Frontansicht auf den **e-Paper ESP32 Driver Board** mit Zwei Schaltern für **A** oder **B**, für die *Displayart* und ON oder OFF, für die Stromversorgung des Serial-Port-Moduls (siehe auch Vorder- und Rückansicht des e-Paper ESP32 Driver Board)

Die Taster EN<sup>2</sup> für einen Neustart des Microcontroller und ein Taster für Boot sind vorhanden, sowie die Schalter für die Display Art und der Ein-/ und Ausschalter für die Stromversorgung<sup>3</sup> des Serial-Ports[24b].

Die Schalterstellung der Display-Art ist auf der Website des Hersteller wie in Abbildung 2.3 gezeigt zu finden.

Einige dieser Peripheriefunktionen des ESP32 nutzt das Infodisplay bereits und die restlichen können dazukommen:

- GPIO (General Purpose Input/Output) - notwendig

<sup>2</sup>Die Defintion für EN aus dem Datenblatte des Esp32-Wroom-32 „High: On; enables the chip Low: Off; the chip shuts down Note: Do not leave the pin floating. „im Datenblatt [Sys23] auf S. 11

<sup>3</sup>Bei Off-Stellung des Schalters, kann kein Programm hochgeladen werden.

- SPI (Serial Peripheral Interface) - notwendig
- ADC (Analog-to-Digital Converter)
- DAC (Digital-to-Analog Converter)
- Touch-Sensoren
- Timer
- LEDC (LED PWM Controller)

Die Spezifikationen des Boards sind laut Herstellerdatenblatt[Wav24b]:

- Wi-Fi protocol: 802.11 b/g/n
- Bluetooth protocol: 4.2, includes traditional BR/EDR, and low power BLE
- Interface: 3-Wire SPI, 4-wire SPI (default)
- Operating voltage: 5V
- Operating current 50mA - 150mA
- Outline dimension: 29.46mm x 48.25mm

Eine Frontansicht des ePaper ESP32 Driver-Board für den Infodisplay ist in Abbildung 2.2 zu sehen.

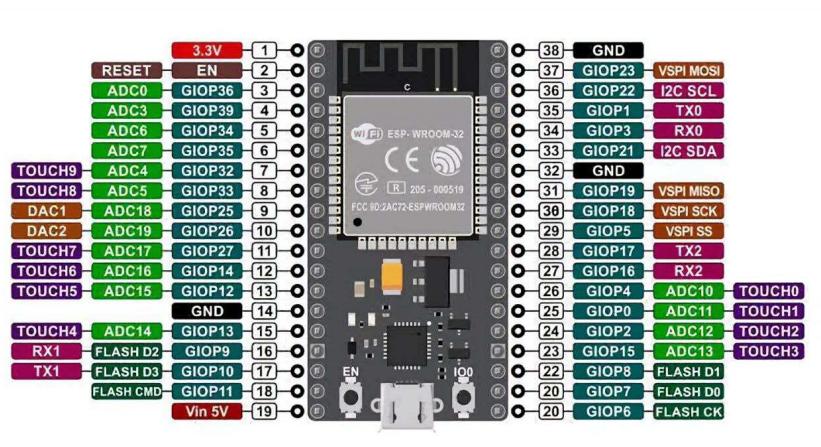
**Pinout des ESP32-Driver** Das e-Paper ESP32 Driver Board ist hardwareseitig mit den Pins des ESP32 verschaltet. Die Display Pins sind intern verlötet. D.h. es ist keine weitere Verdrahtung notwendig.

---

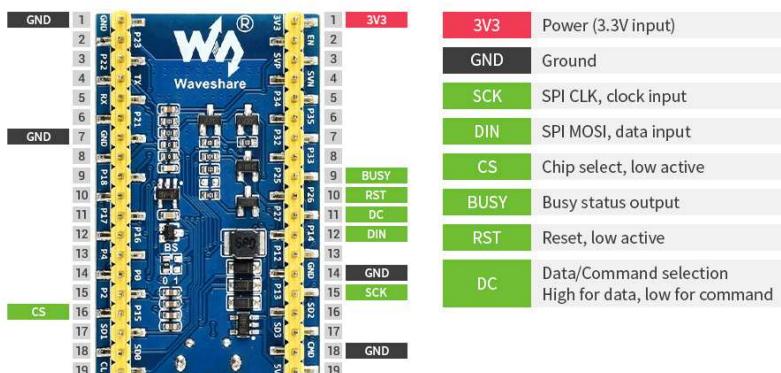
Der Shop[Wav24c], die Website[24b] und die Dokumentation[Wav24b] liefern Informationen zur Internen Verschaltung.

(a)

Pinout aus den FAQ von Waveshare[24a]



(b) Rückansicht mit Hardware Pins des e-Paper ESP32 Driver Board aus dem Shop[Wav24c]



(c) Pinout Definition laut Website[24b]

Pin	ESP32	Description
VCC	3V3	Power input (3.3V)
GND	GND	Ground
DIN	P14	SPI MOSI pin, data input
SCLK	P13	SPI CLK pin, clock signal input
CS	P15	Chip selection, low active
DC	P27	Data/command, low for commands, high for data
RST	P26	Reset, low active
BUSY	P25	Busy status output pin (means busy)

**PS:** The above is the board fixed connection, with no additional operation by the user.

(d) Hardware Pins des e-Paper ESP32 Driver Board laut User Manual[Wav24b]

```
26 /* SPI pin definition -----
27 #define PIN_SPI_SCK 13
28 #define PIN_SPI_DIN 14
29 #define PIN_SPI_CS 15
30 #define PIN_SPI_BUSY 25
31 #define PIN_SPI_RST 26
32 #define PIN_SPI_DC 27
```

Abbildung 2.4: Pinout-Angaben von Waveshare

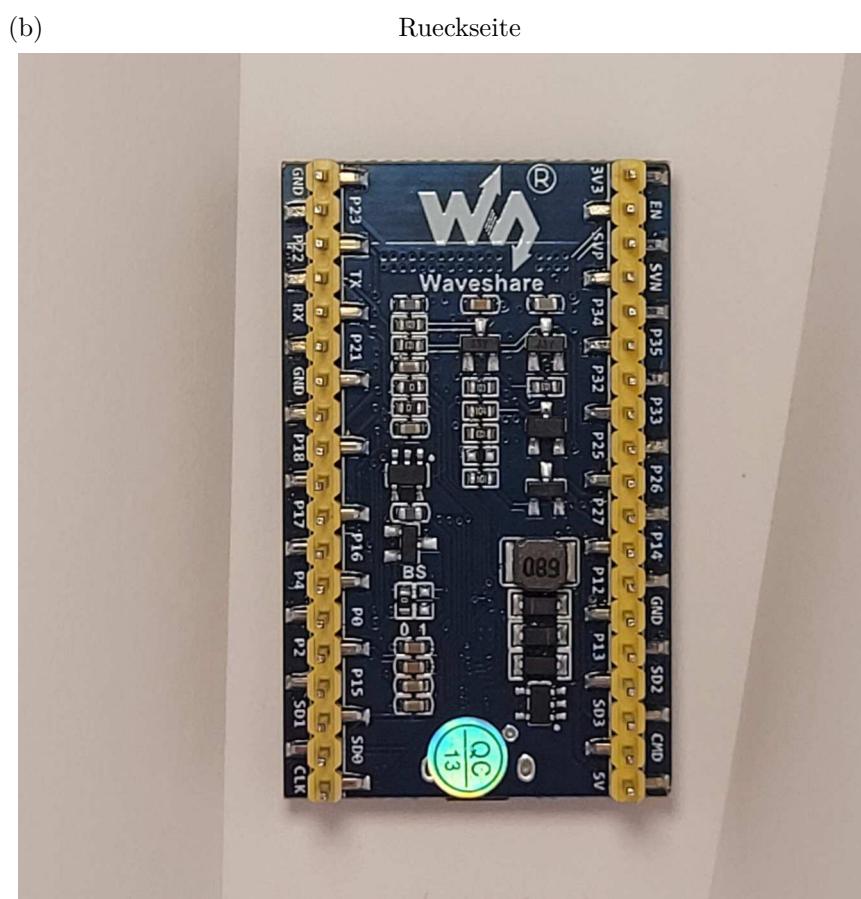


Abbildung 2.5: Vorder- und Rückansicht des e-Paper ESP32 Driver Board

## 2.1.2 Display

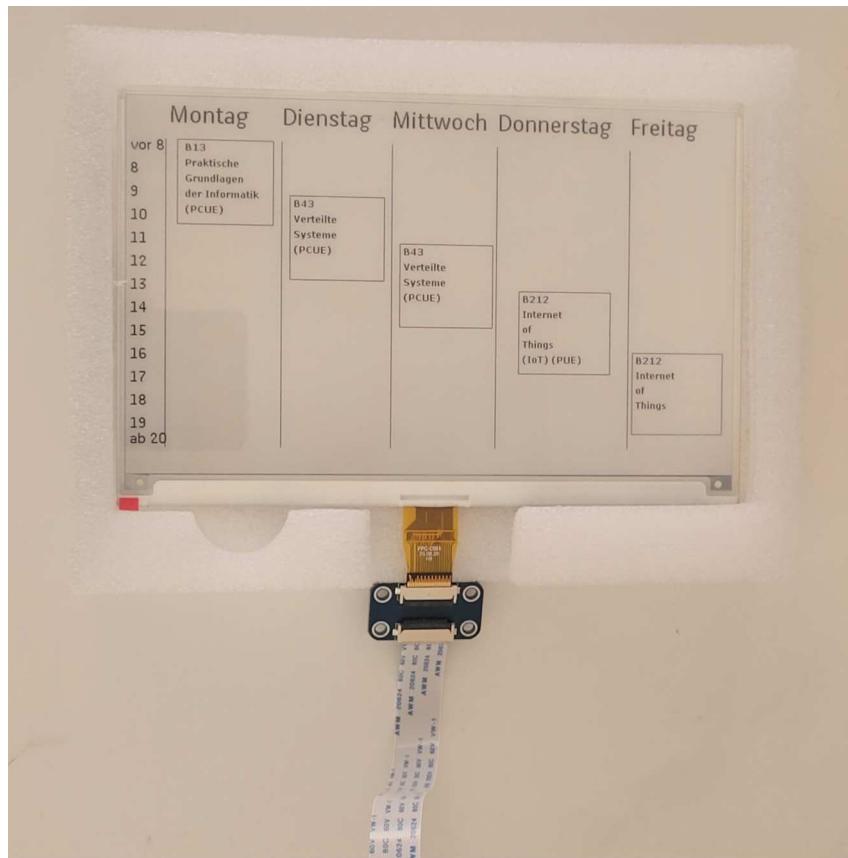


Abbildung 2.6: Der Infodisplay mit

Der 7,5-Zoll-E-Ink-Raw-Display von Waveshare bietet eine SPI-Schnittstelle zur Kommunikation[Gmb24].

Hierbei handelt es sich um elektrophoretisches Papier das einen bistabilen Zustand aufrechterhalten kann, auch ohne Stromversorgung.

## 2.1.3 Spannungssensor und Stromsensor

Der Spannungssensor aus Abbildung 2.7(a) ist ein Hergestellt von mh-electronics[Ele24]

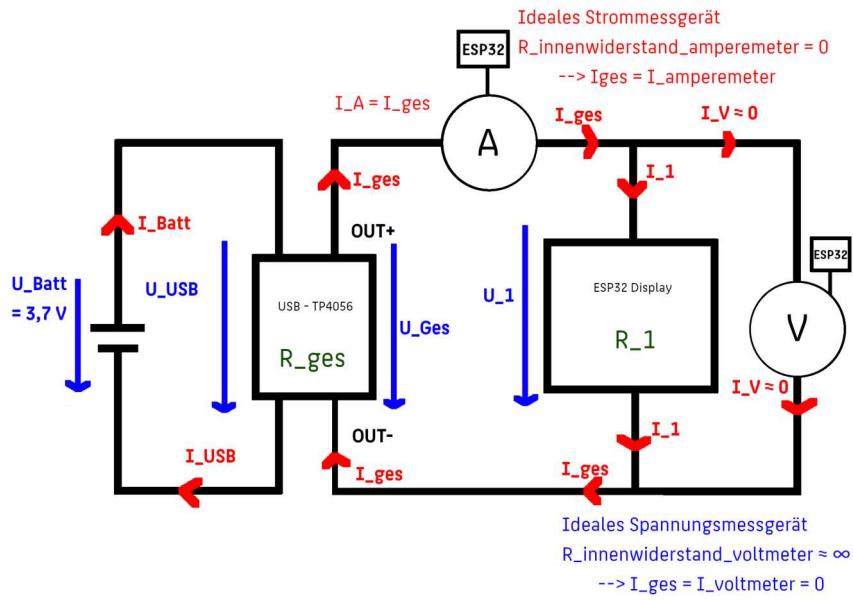


Abbildung 2.7: Schaltskizze Strom- und Spannungsmessung

In der Schaltskizze zur Strom- und Spannungsmessung sind die einzelnen Ströme und Spannungen eingetragen. Die Spannungspfeile stellen die Spannungsabfälle, die Pfeilspitzen den durchfließenden Strom. Die Summe aller Teilströme ist konstant.

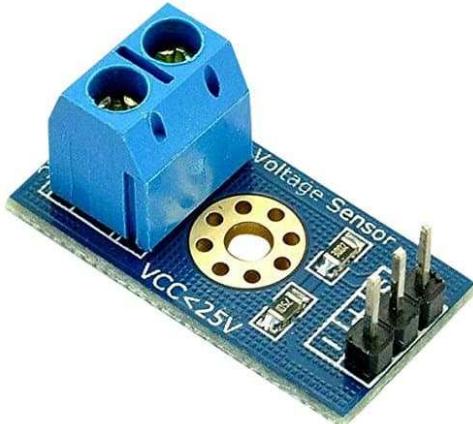
Das Messgerät zur Spannungsmessung ist Parallel zum Display geschaltet und die Stromstärke Messung findet in Reihe zum Display statt. Somit ist eine Berechnung und Messung der elektrischen Kennwerte möglich.

Der Stromsensor aus Abbildung 2.7(b) ist ein ACS712[Mar24b]

#### 2.1.4 Laderegler und Lithium Polymer Batterie

Der Laderegler ist ein TP4056[Sho24] und die Batterie ist eine LIPO mit 3,7 V Leerlaufspannung und 7000 mAh Batteriekapazität in Amperestunden.

(a) Messung der Spannung U in Volt



(b) Der Stromstärke I in Ampere

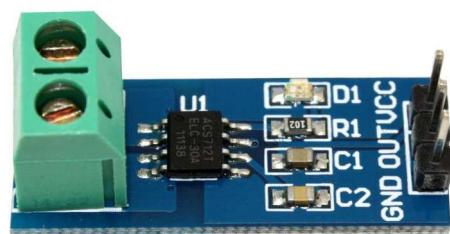


Abbildung 2.8: Messung zur Bestimmung der elektrischen Leistung nach:

$$\mathbf{P} = \mathbf{U} * \mathbf{I}$$

in Watt

## 2.2 Software

### 2.2.1 MicroPython

MicroPython ist eine schlanke Python 3 Implementierung die nur eine Teilmenge der Python Standardbibliothek zur Verfügung stellt.

Sie ist optimiert um auf 'constrained environments' wie dem ESP32 zu entwickeln. Um Anwendungsfälle mit MicroPython auf einem Mikrocontroller in Programmiersprache zu übersetzen ist eine Firmware auf den Mikrocontroller notwendig. Diese micropython-Firmware wird auf den ESP32 *gefashst*.

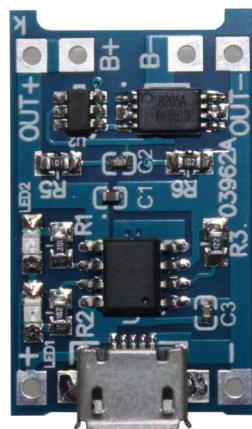
Dazu wird die Adresse der seriellen Schnittstelle (der Port) ermittelt. Über diese Schnittstelle wird der Flasher (in diesem Fall Thonny) später die Firmware auf den ESP32 laden. Damit dieser bei jedem Bootvorgang als erstes die **micropython-Firmware** in den Appspeicher des ESP32 lädt. [Sys24a; Sys24b] **Ermitteln des Port** Bevor der **ESP32-Driver gefashst** werden kann, wird zunächst der Port ermittelt, damit dieser später beim *flashen* auch angegeben werden kann.

Hierzu wird ein Terminal geöffnet und um darin mit dem ls-Befehl alle Seriellen Geräteschnittstellen mit der Zeichenkette **tty\*** aus dem Verzeichnis **/dev** ausgeben.

Das ist in Code-Listing 2.1 dargestellt.

(a)

Der Laderegler TP4056



(b)

Der 7000mAh LIPO



Abbildung 2.9: Batteriemanagementsystem und der Akku

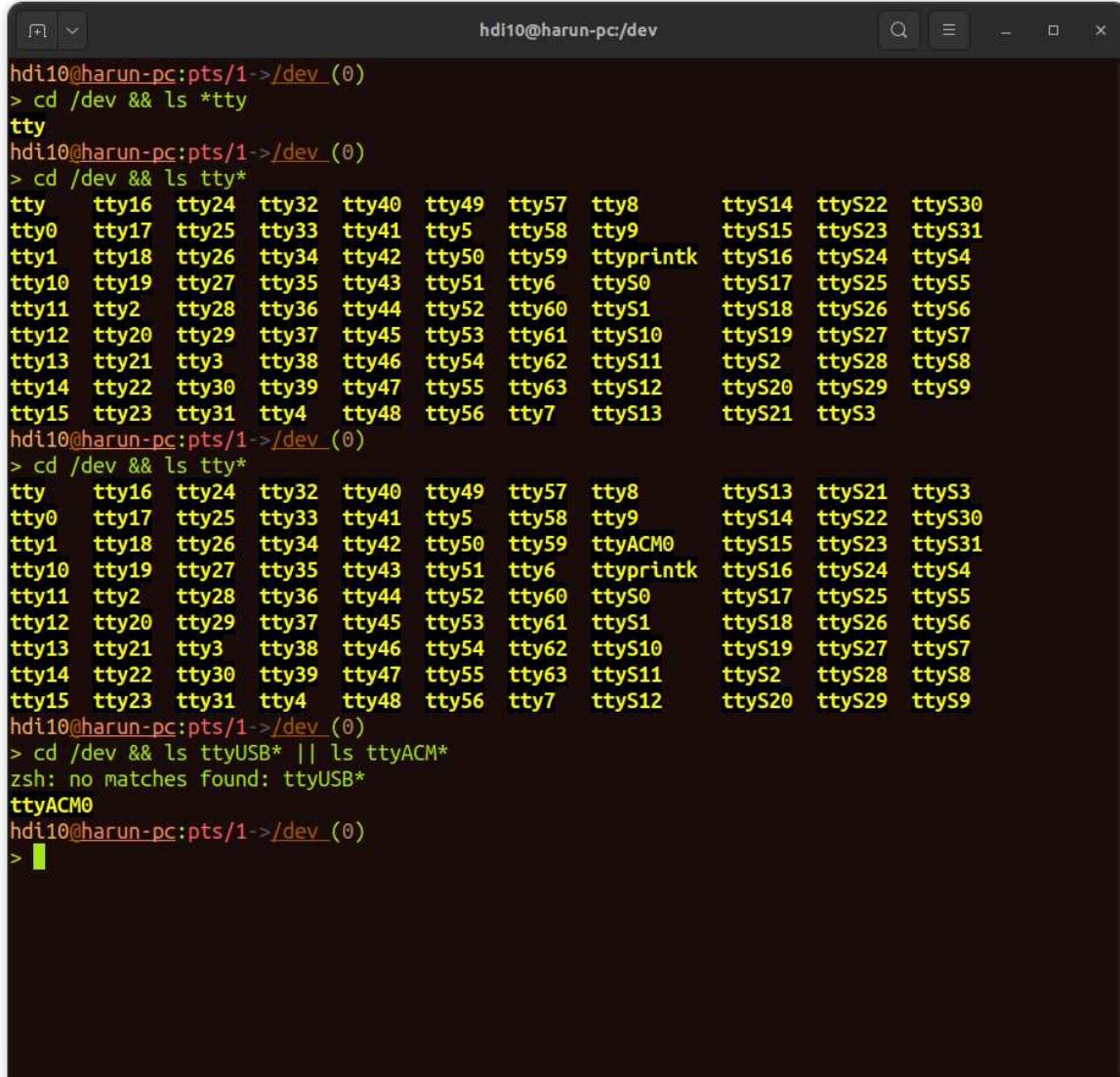
```
1 |         cd /dev && ls tty*
```

Code snippet 2.1: and list directory content, especially files starting with the char-string  
tty]Change Directory to /dev[ices] and list directory content, especially  
files starting with the char-string tty

Nach Anschluss des **ESP-Driver** werden die Zeilen erneut ausgeführt einem Blick auf  
die Einträge wie **ttyUSB** oder **ttyACM0**. Im Vergleich beider Ausgaben wird der Port  
identifiziert. Alternativ gibt dieser Befehl den Port nach Anschluss des **ESP-Driver**  
wieder:

```
1 |         cd /dev && ls tty* ls ttyUSB* || ls ttyACM*
```

Code snippet 2.2: and if succesfull then list directory content starting with char-string  
ttyUSB or ttyACM0]Change Directory to /dev[ices] and if succesfull  
then list directory content starting with char-string ttyUSB or ttyACM0



```
hdi10@harun-pc:pts/1->/dev (0)
> cd /dev && ls *tty
tty
hdi10@harun-pc:pts/1->/dev (0)
> cd /dev && ls tty*
tty  tty16  tty24  tty32  tty40  tty49  tty57  tty8      ttyS14  ttyS22  ttyS30
tty0  tty17  tty25  tty33  tty41  tty5   tty58  tty9      ttyS15  ttyS23  ttyS31
tty1  tty18  tty26  tty34  tty42  tty50  tty59  ttyprintk  ttyS16  ttyS24  ttyS4
tty10  tty19  tty27  tty35  tty43  tty51  tty6   ttyS0      ttyS17  ttyS25  ttyS5
tty11  tty2  tty28  tty36  tty44  tty52  tty60  ttyS1     ttyS18  ttyS26  ttyS6
tty12  tty20  tty29  tty37  tty45  tty53  tty61  ttyS10    ttyS19  ttyS27  ttyS7
tty13  tty21  tty3  tty38  tty46  tty54  tty62  ttyS11    ttyS2  ttyS28  ttyS8
tty14  tty22  tty30  tty39  tty47  tty55  tty63  ttyS12    ttyS20  ttyS29  ttyS9
tty15  tty23  tty31  tty4  tty48  tty56  tty7   ttyS13    ttyS21  ttyS3
hdi10@harun-pc:pts/1->/dev (0)
> cd /dev && ls tty*
tty  tty16  tty24  tty32  tty40  tty49  tty57  tty8      ttyS13  ttyS21  ttyS3
tty0  tty17  tty25  tty33  tty41  tty5   tty58  tty9      ttyS14  ttyS22  ttyS30
tty1  tty18  tty26  tty34  tty42  tty50  tty59  ttyACM0  ttyS15  ttyS23  ttyS31
tty10  tty19  tty27  tty35  tty43  tty51  tty6   ttyprintk  ttyS16  ttyS24  ttyS4
tty11  tty2  tty28  tty36  tty44  tty52  tty60  ttyS0      ttyS17  ttyS25  ttyS5
tty12  tty20  tty29  tty37  tty45  tty53  tty61  ttyS1     ttyS18  ttyS26  ttyS6
tty13  tty21  tty3  tty38  tty46  tty54  tty62  ttyS10    ttyS19  ttyS27  ttyS7
tty14  tty22  tty30  tty39  tty47  tty55  tty63  ttyS11    ttyS2  ttyS28  ttyS8
tty15  tty23  tty31  tty4  tty48  tty56  tty7   ttyS12    ttyS20  ttyS29  ttyS9
hdi10@harun-pc:pts/1->/dev (0)
> cd /dev && ls ttyUSB* || ls ttyACM*
zsh: no matches found: ttyUSB*
ttyACM0
hdi10@harun-pc:pts/1->/dev (0)
>
```

Abbildung 2.10: Screenshot mit Ausgabe der Serielle Schnittstelle auf dem Terminal

Der Port des **ESP-Driver** befindet sich entsprechend der Abbildung 2.10 Port auf PORT **ttyACM0**.

### MicroPython Firmware

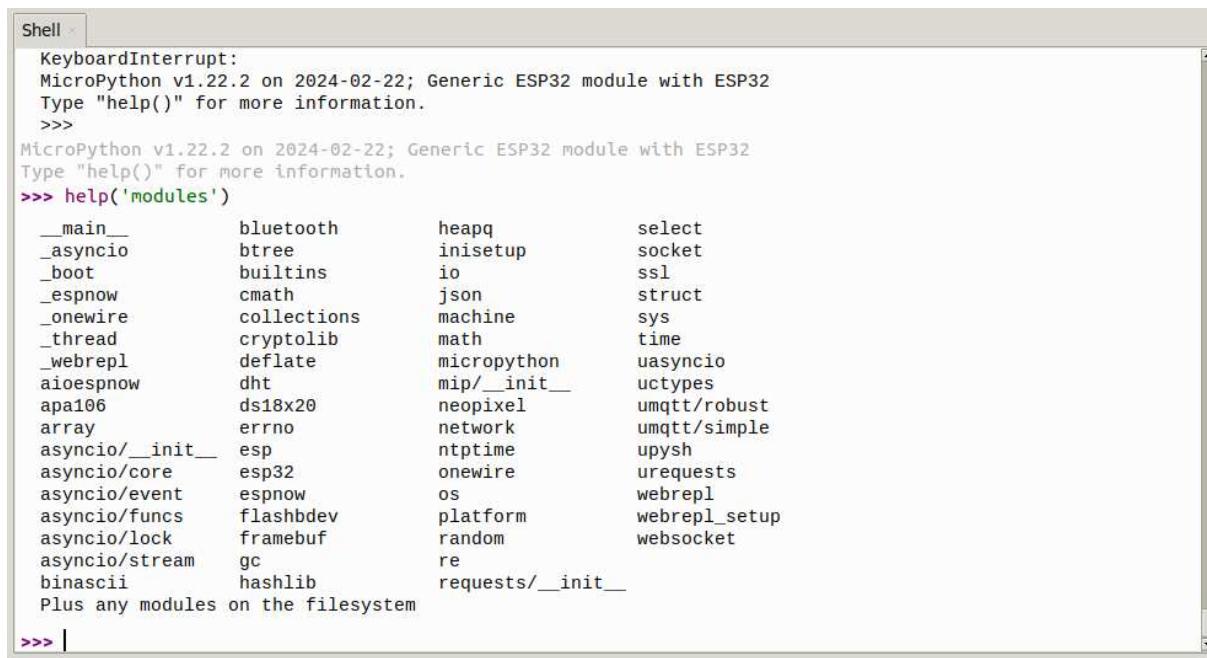
Zum flashen des **ESP-Driver** empfiehlt sich Thonny.

Die **Thonny IDE** für Python und MicroPython bietet benutzerfreundliches programmieren, debuggen und hochladen von Skripten.[Dev24b] Mithilfe der Thonny IDE lässt sich zudem die aktuelle **MicroPython** Firmware *flashen*. **Eigene Firmware**

Diese **Thema** wurde weiter vertieft und diente später im Kapitel 6 dazu eine Testumgebung anzulegen.

Dies sind die Bestandteile des Lösungsansatzes zum *Freezen* von **modules**. Das anbinden zusätzlicher eigener Module auf der Firmware von micropython, wie bspw. den Binary-Files der Images:

1. Firmware in unterschiedlichen Versionen (OTA, No OTA usw.)
2. Eigene Firmware
3. Eigene Firmware Partitionieren
4. Eigener Firmware Module mitgeben (siehe Bild)
5. Eigene Firmware für ein Unix System



The screenshot shows a Windows-style application window titled "Shell". Inside, the MicroPython interpreter is running. It starts with a "KeyboardInterrupt" message, followed by the MicroPython version and build information: "MicroPython v1.22.2 on 2024-02-22; Generic ESP32 module with ESP32". It then prompts for help: "Type "help()" for more information." A command ">>> help('modules')" is entered, and the interpreter lists all available modules. The list includes standard Python modules like \_\_main\_\_, asyncio, boot, espnow, newwire, thread, webrepl, aioespnow, apa106, array, asyncio/\_init\_, asyncio/core, asyncio/event, asyncio/funcs, asyncio/lock, asyncio/stream, binascii, bluetooth, btree, builtins, cmath, collections, cryptolib, deflate, dht, ds18x20, errno, esp, esp32, espnow, flashbdev, framebuffer, gc, hashlib, heapq, json, machine, math, micropython, mip/\_init\_, neopixel, network, nptime, onewire, os, platform, random, re, requests/\_init\_, select, socket, ssl, struct, sys, time, uasyncio, uctypes, umqtt/robust, umqtt/simple, upysh, urequests, webrepl, webrepl\_setup, websocket, and several modules from the filesystem. The output ends with "Plus any modules on the filesystem".

```
KeyboardInterrupt:  
MicroPython v1.22.2 on 2024-02-22; Generic ESP32 module with ESP32  
Type "help()" for more information.  
>>>  
MicroPython v1.22.2 on 2024-02-22; Generic ESP32 module with ESP32  
Type "help()" for more information.  
>>> help('modules')  
__main__      bluetooth      heapq        select  
_asyncio     btree          inisetup      socket  
_boot       builtins       io            ssl  
_espnow      cmath          json          struct  
_newwire    collections    machine       sys  
_thread     cryptolib     math          time  
_webrepl    deflate        micropython  uasyncio  
aioespnow   dht           mip/_init_  uctypes  
apa106      ds18x20       neopixel     umqtt/robust  
array       errno          network      umqtt/simple  
asyncio/_init_ esp          nptime       upysh  
asyncio/core esp32         onewire     urequests  
asyncio/event espnow        os           webrepl  
asyncio/funcs flashbdev    platform    webrepl_setup  
asyncio/lock  framebuffer  random      websocket  
asyncio/stream gc           re            
binascii     hashlib       requests/_init_  
Plus any modules on the filesystem  
>>> |
```

Abbildung 2.11: Screenshot der *Modules* auf dem **ESP32** mit MicroPython v1.22.2

```

hdi10@harun-pc:/run/user/1000/gvfs/smb-share:server=bearliondeer.local,share...
LowPowerInfodisplay/Firmware/CustomFirmware/micropython/ports/esp32
[Aug  8 01:55] .
└── [Aug  8 01:55] boards
    ├── [Aug  8 01:55] ARDUINO_NANO_ESP32
    ├── [Aug  8 01:55] ESP32_GENERIC
    ├── [Aug  8 01:55] ESP32_GENERIC_C3
    ├── [Aug  8 01:55] ESP32_GENERIC_S2
    ├── [Aug  8 01:55] ESP32_GENERIC_S3
    ├── [Aug  8 01:56] LILYGO_TTGO_LORA32
    ├── [Aug  8 01:56] LOLIN_C3_MINI
    ├── [Aug  8 01:56] LOLIN_S2_MINI
    ├── [Aug  8 01:56] LOLIN_S2_PICO
    ├── [Aug  8 01:56] MSSTACK_ATOM
    ├── [Aug  8 01:55] OLIMEX_ESP32_POE
    ├── [Aug  8 01:55] SIL_WESP32
    ├── [Aug  8 01:56] UM_FEATHERS2
    ├── [Aug  8 01:56] UM_FEATHERS2NE0
    ├── [Aug  8 01:56] UM_FEATHERS3
    ├── [Aug  8 01:56] UM_NANOS3
    ├── [Aug  8 01:56] UM_PROS3
    ├── [Aug  8 01:56] UM_TINYPICO
    ├── [Aug  8 01:56] UM_TINYS2
    ├── [Aug  8 01:56] UM_TINYS3
    └── [Aug  8 01:55] UM_TINYWATCH3
[Aug  8 01:56] build-ESP32_GENERIC
├── [Aug  8 01:55] bootloader
├── [Aug  8 01:57] bootloader-prefix
├── [Aug  8 01:56] CMakeFiles
├── [Aug  8 01:55] config
├── [Aug  8 01:56] esp-idf
├── [Aug  8 01:55] genhdr
├── [Aug  8 01:55] log
└── [Aug  8 01:55] partition_table
    └── [Aug  8 01:57] submodules
        ├── [Aug  8 01:54] main_esp32
        ├── [Aug  8 01:54] main_esp32c3
        ├── [Aug  8 01:54] main_esp32s2
        ├── [Aug  8 01:54] main_esp32s3
        ├── [Aug  8 01:55] managed_components
            └── [Aug  8 01:57] espressif_mdns
        └── [Aug  8 01:54] modules
[Aug  8 01:54] hdi10 at harun-pc in /run/user/1000/gvfs/smb-share:server=bearliondeer.local,share=00_b-hdi
[Aug  8 01:54] hdi10 at harun-pc in /run/user/1000/gvfs/smb-share:server=bearliondeer.local,share=00_ba_s05
51006/AUGUST_2024/UltraLowPowerInfodisplay/Firmware/CustomFirmware/micropython/ports/esp32 on
master✓
└─±

```

Abbildung 2.12: Screenshot

Der Schritt eigene Firmware für den **ESP32** zu kompilieren ist leider nicht gelungen. Der erste erfolgreiche Firmware-**build** ist für ein **Unix System**.

Der Ausführbare Firmware *Build* befindet sich unter: '/Firmware/forUnix'.

Beim Startvorgang wird die **micropython**-Firmware vom **Bootloader** des ESP32 aus dem Flash geladen und in den Speicher kopiert. Im weiteren Verlauf führt der **Bootloader** die initialen Setup-Schritte durch. D.h. Auswahl der richtigen Partition und das Laden der Firmware. Im Anschluss übernimmt die Firmware wieder die Kontrolle.[Sys24a; Sys24b]

## 2.2.2 Micropython Generic Firmware

Vorerst wird die Firmware der MicroPython.org Downloadseite genutzt.[Mic24a]

- Im ersten Schritt nach dem Download der Firmware ist in den Thonny Einstellungen, unter dem Reiter *Interpreter*, der Interpreter auf MicroPython zu setzen und der Port `/dev/ttyACM0` für den Upload zu auszuwählen. In diesem Fall ist das der PORT **ACM0**.
- Nach einem Klick auf *install or update firmware* wird erneut der Port **ACM0** verlangt, nach Eingabe kann im nächsten Feld mit der Browse Funktion die heruntergeladene Firmware ausgewählt werden.[Mic24b]
- Und mit folgendem Klick auf *install* startet den flashvorgang auf den Mikrocontroller.
- Und ein *Done!* bestätigt den rfolg.

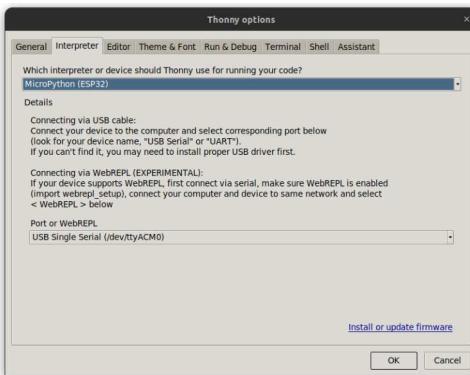
Nach dem Herunterladen der Firmware ist in den Thonny Einstellungen, unter dem Reiter *Interpreter*, der Interpreter auf MicroPython zu setzen und der Port `/dev/ttyACM0` für den Upload auszuwählen. In diesem Fall ist das der PORT **ACM0**. Nach einem Klick auf *install or update firmware* wird erneut der Port ACM0 und den Browse Button zu der Heruntergeladenen Firmware navigiert.[Mic24b].

Diese Schritte sind in Abbildung 2.13 dargestellt. Des weiteren stehen die Möglichkeit den ESP32 mit dem *esptool* zu *flashen*. Hierzu wird erneut die Firmware von MicroPython.org genutzt im Terminal zu *flashen*.[Mic24a; Mic24b]

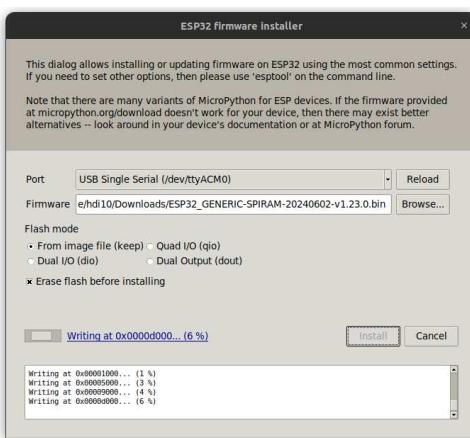
```
1 |     pip install esptool && esptool.py --port /dev/ttyACM0 erase_flash &&
|         esptool.py --chip esp32 --port /dev/ttyACM0 write_flash -z
|             0x1000 ESP32_GENERIC-SPIRAM-20240602-v1.23.0.bin
```

Code snippet 2.3: Flashen der Firmware auf den ESP32 mit dem esptool

- (a) In den Thonny Einstellungen ist der ermittelte Port, in diesem Fall ACM0 und der MicroPython Interpreter zu wählen



- (c) Start des Flashvorgangs an der Bootloader Adresse 0x1000 im Flash[Sys24b]



- (b) Nach dem Klick auf *install or update firmware* müssen nur noch Port und die Firmware als Binärdatei ausgewählt werden



- (d) Ende des Flashvorgangs

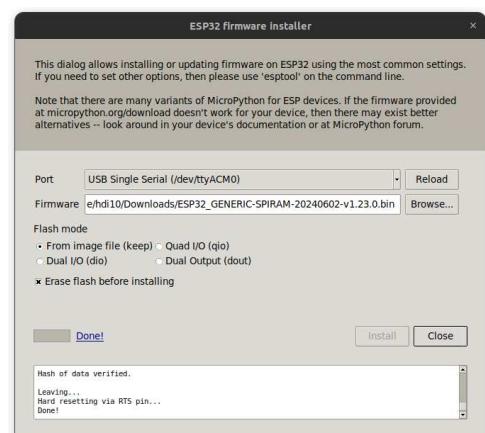


Abbildung 2.13: Der Flashvorgang der Firmware auf den ESP32.

### 2.2.3 Hello World - Blinky - REPL

Die REPL das ist der Name der interaktiven MicroPython prompt die mit dem ESP32 genutzt werden kann.???. Die Verbindung zum interaktiven REPL ist über die UART-Schnittstelle des MicroPython ports oder über eine Wireless LAN (Local Area network) möglich. Hier lassen sich Befehle ausführen oder MicroPython-Scripte hochladen. Mit Ausgabe aller Modul-Bibliotheken bspw. lassen sich alle verfügbaren und integrierten Bibliotheken anzeigen.

Diese *modules* lassen sich dann aus dem persistenten *flashspeicher* mit 'import' importieren.

```
1 |     help('modules')
```

Code snippet 2.4: Ausgabe aller Modul-Bibliotheken

```
Shell
KeyboardInterrupt:
MicroPython v1.22.2 on 2024-02-22; Generic ESP32 module with ESP32
Type "help()" for more information.
>>>
MicroPython v1.22.2 on 2024-02-22; Generic ESP32 module with ESP32
Type "help()" for more information.
>>> help('modules')
__main__      bluetooth    heapq       select
asyncio       btree        inisetup   socket
_boot        builtins    io          ssl
_espnow       cmath        json        struct
_onewire     collections machine   sys
_thread      cryptolib  math        time
_webrepl     deflate     micropython
aioespnow    dht         mip/_init_
apa106       ds18x20    neopixel
array        errno        network
asyncio/_init_ esp         ntptime
asyncio/core esp32       onewire
asyncio/event espnow      os
asyncio/funcs flashbdev  platform
asyncio/lock framebuffer random
asyncio/stream gc          re
binascii      hashlib     requests/_init_
Plus any modules on the filesystem
>>> |
```

Abbildung 2.14: Ausgabe aller integrierten und verfügbaren Bibliotheken des **ESP32-Driver**, die über den Befehl *import* importiert werden können

# Kapitel 3

## Analyse

**Analyse der Anforderungen** mithilfe von *agiler Softwareentwicklung* und *klassischem Projektmanagement*. Es werden beide Herangehensweisen weitgehend versucht abzudecken. Zum einen die **traditionellen Softwareentwicklung**, in der zu Projektbeginn die Anforderungen in einer Definitionsphase möglichst vollständig erfasst und präzise beschrieben werden. Und nach Review, Analyse der Anforderungen und Designspezifikation schließlich Code geschrieben wird, siehe auch Abbildung 3.1.[Pic09]

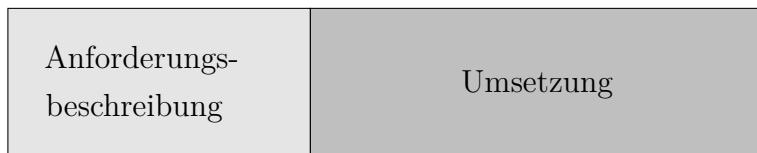


Abbildung 3.1: Definitions- und Umsetzungsphase

Zum anderen von Scrum, mit dem Product-Backlog für den E-Ink Infodisplay, dem Sprint-Backlog für die Abschlussarbeitsphase, dem FB4 als Product Owner und der Form des Product-Backlog.

Zudem werden in Scrum Anforderungen nicht länger zu Projektbeginn vollständig und detailliert erfasst und anschließend in die Entwicklung gegeben.

Denn es existiert keine separate Definitions- und Umsetzungsphase: Anforderungsbeschreibung und Umsetzung erfolgen zeitnah und überlappend, wie in Abbildung 3.1 illustriert [Pic09].

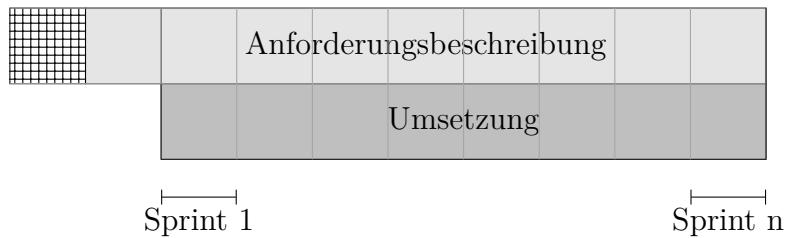


Abbildung 3.2: Anforderungsbeschreibung in Scrum[]

Abbildung 3.1 zeigt diesen Sachverhalt.

## 3.1 Anforderungen

### 3.1.1 MicroPython und Thonny

Der Leser möchte eine kurze Einleitung in die Programmiersprache **MicroPython** von *MicroPython.org* (und der dazugehörigen Firmware) und das Tool Thonny zur Verständnis der Software für den Infodisplay.

Siehe Grundlagen auf Seite 4 oder dem dargelegten Konzept auf Seite 28.

### 3.1.2 Ein- und Ausschalten

Der Nutzer möchte einen Schalter haben um das Infodisplay ein- und auszuschalten. Damit beim Transport oder bei Verwahrung die Batterielebensdauer nicht beeinträchtigt wird. Das Konzept dazu auf Seite 28.

### 3.1.3 Laufzeit

Als Nutzer möchte ich 6-12 Monate die Anwendung nutzen ohne den Akku zu laden. Damit der Aufwand des Aufladens nicht entsteht oder ich keinen Stromnetzanschluss zur

Verfügung habe.<sup>1</sup>

Siehe zu dieser Anforderung den konzeptionellen Ansatz zur Laufzeitverlängerung des Infodisplay auf Seite 30.

### 3.1.4 Schaltung

Als Nutzer möchte ich einen anschaulichen Schaltplan, eine Bauteilliste, die Beschreibung der spezifischen Bauteile, eine Pin-Belegungstabelle, die Spannungs- und Stromwertkennlinien und ein Leistungsdiagramm haben, um weitere Entscheidungen über das Projekt zu treffen.

- anschaulichen Schaltplan,
- eine Bauteilliste,
- eine technische Zeichnung für die Hobbglassplatten in dwg und svg Formaten,
- die Beschreibung der spezifischen Bauteile,
- eine Pin-Belegungstabelle,
- die Spannungs- und Stromwertkennlinien und
- ein Leistungsdiagramm

Konzepte zur Lösung dieser Anforderung auf Seite 31.

---

<sup>1</sup>Ein USB-C BMS würde den Ladevorgang beschleunigen, doch müsste man hier nach dem zulässigen aufladestrom beachten. Das Sparkfun BMS ist hinsichtlich aufladestrom einstellbar mit den Schaltern EN1 und EN2 auf dem PCB. Mithilfe der Schalterstellung laut Tabelle kann das Sparkfun BMS von keinem Ladestrom bis 1500mA Ladestrom im Fast Charge eingestellt werden. Im Suspend Modus ohne Ladefunktion, im Fast Charge mit 1500mA neben dem Laut folgender Tabelle.HIERTABELLE

### 3.1.5 Pinbelegung

Der Nutzer möchte wissen, welche Pins der Infodisplay nutzt und welche in der Messschaltung verwendet wurden.

Siehe dazu Konzept der Pinbelegung<sup>36</sup> oder die Anforderung im Konzeptpapier??.

### 3.1.6 Wartbarkeit und Schneller Einstieg

Der Entwickler möchte einen Überblick über die Projektstruktur der Infodisplay Repository um schnell einen Einstieg zu finden.

Ein Link zur GitLab Repo des Infodisplay ist in der Fußzeile Konzeptanforderung auf der Seite 4.6.

### 3.1.7 Aktualisieren

Der Anwender möchte jeden Tag um 5 Uhr eine Aktualisierung des Infodisplay aus dem LSF um den letzten Stand des Raumbelegungsplans zu haben.

Siehe hierzu die Konzeption der Anforderung auf Seite 42.

### 3.1.8 Partielles Aktualisieren

Der Anwender möchte nur einen teil des Display aktualisiert haben um eine bessere visuelle Darstellung zu erzielen mit weniger Batterieleistung.

### 3.1.9 Bitmap an Infodisplay

Der Nutzer möchte eine Bitmap an den Infodisplay senden damit er ein Bild auf dem Display sehen kann.

Siehe hierzu das **Konzept** der *Anforderung* auf Seite 42 dann die **Evaluation** auf Seite 102 dazu.

### 3.1.10 Systemarchitektur

Der Kunde möchte eine Darstellung der Systemarchitektur um einen Einblick in die Schnittstellen zu gewinnen.

Siehe hierzu die Konzeption einer Systemarchitektur mit Node-RED2 auf der Seite 42.

### 3.1.11 Powerbank

Als Nutzer möchte ich das Infodisplay mit einer Stromquelle oder Powerbank aufladen können um ein schnelles Wiederaufladen zu ermöglichen<sup>2</sup>.

Die realisierte Konzeptidee mit einem TP4056 als Laderegler ist in der Implementierung der Hardware zu sehen 5.1 die Evaluation dazu ist auf Seite 47 zu finden.

### 3.1.12 Leistung

Der Anwender möchte ein Fazit mit Leistungsdiagrammen. Um zu sehen ob ein Energieeinsparpotenzial gegenüber Papier aushängen bemerkbar ist.

Das Konzept zur Leistungsbestimmung befindet sich auf Seite 47 zu finden.

---

<sup>2</sup>Das befüllen der Akkus mit Strom könnte anhand einer Powerbank mobiler gemacht werden.

### 3.1.13 Tests

Der Nutzer möchte einen Testplan in der Test Suite haben um einen Überblick über die Testumgebung, den manuellen und automatisierten Tests zu bekommen.

Die Tests Suite befindet sich auf im Kapitel6.

### 3.1.14 Serielle Schnittstelle

Der Nutzer möchte wissen was ein Bistabiler Display der unter Nutzung der SPI-Schnittstelle monochrome Bilder anzeigen kann[DAL24]

Siehe dazu Kapitel 2

## 3.2 Planung der Stromverbrauchs Messung

(Multimeter zur Strom- und Spannungsmessung) Für die Strommessung wird das Messgerät in Reihe geschaltet. Dazu wird das Multimeter wie in der Abbildung ??img:stromMessschaltung) gezeigt an den Punkten P1 für den positiven Pol und P2 für den Negativen Pol angelegt.

Zur Spannungsmessung wird das Messgerät Parallel geschaltet. Hierzu wird das Multimeter wie in Abbildung ??img:SpannungsMessschaltung) an den Punkten P1 für den positiven Pol und PZ für den negativen Pol angebracht.

# Kapitel 4

## Konzeption

Im Kapitel zur *Konzeption* der Abschlussarbeit wird zunächst auf die Technologieauswahl eingegangen und dann sind Anforderungen aus dem Kapitel 3 aufgegriffen und weiter in die Entwicklung des finalen Produkts geflossen.

So konnte die Planung des Prototypen und des Finalen Produkts nahtlos in die Evaluation 7 übergehen.

### **Konzepte zur Realisierung der Anforderungen**

#### **4.1 Technologieauswahl**

Hier wird die Auswahl spezifischer Technologien und Komponenten für das Projekt aufgeführt.

Als User Interface wurde Telegram gewählt, da es Intuitiv und einfach zu Benutzen ist und eine gute User Experiance Dank der Bot Kommandos.

Zur Umsetzung der Anbindung an Telegram mit MQTT, einer Persistierung mit Influx und einer Visualisierung mit Grafana bietet sich Node-Red hervorragend an.

Dadurch ist eine drahtlose Übertragung auch von Single Buffer Elementen problemlos möglich, falls in Zukunft Bilder über MQTT versand werden sollten. Diese und weitere Schritte sind Bestandteil der Implementierung im Kapitel 5.

MQTT eignet weiter sich, weil jedes ESP (Voltage, Current und Display) eigenständig und unabhängig messen und darstellen kann.

Nach Beendigung der Aufgabe eines ESP wird in den MQTT Pool published, sodass die Subscriber dieses Topic sofort benachrichtigt werden.

Der Esp mit dem Spannungssensor misst die Spannung und published diese unter dem Topic: „B\_EiNk/006/gesamtspannung“.

Der Esp mit dem Strommesser misst die Stromstärke und published diese unter dem Topic: „/EiNK/006/gesamtstrom“.

Der Esp des Infodisplay refreshed und cleared sobald auf den Abonnierten Topic: „B\_EiN-K/006/image“ und „B\_EiNK/006/clear“ etwas gepublished wird. Zudem published er selbst den aktuellen Status des Infodisplay unter dem Topic: „B\_EiNK/006/state“.

## 4.2 MicroPython und Thonny

Aufgrund dieser Anforderung wird im weiteren MicroPython als Programmiersprache genutzt. Nähere Information zu MicroPython und Thonny sind in den Grundlagen 2 oder auf MicroPython.org[Dev24a] oder Thonny.org[Dev24b]

S. dazu Evaluation der Anforderung auf Seite 94.

### **Ein- und Ausschalten**

Ein Ein- und Auschalter als Bauteil S1 soll in die Schaltung eingebaut werden.

Dieser wird Hardwareseitig umgesetzt und als Schalter mit zwei Zuständen (ON-OFF) zwischen der Last und der Spannungsversorgung geschaltet. D.h. der Schalter S1 soll wie in Abbildung 4.1 zwischen ePaper Esp32 Driver-Board und dem AkkuLIPo geschaltet werden.

Die Evaluation der Anforderung nach Implementierung des Hardwareschalters S1 befindet sich auf Seite 95

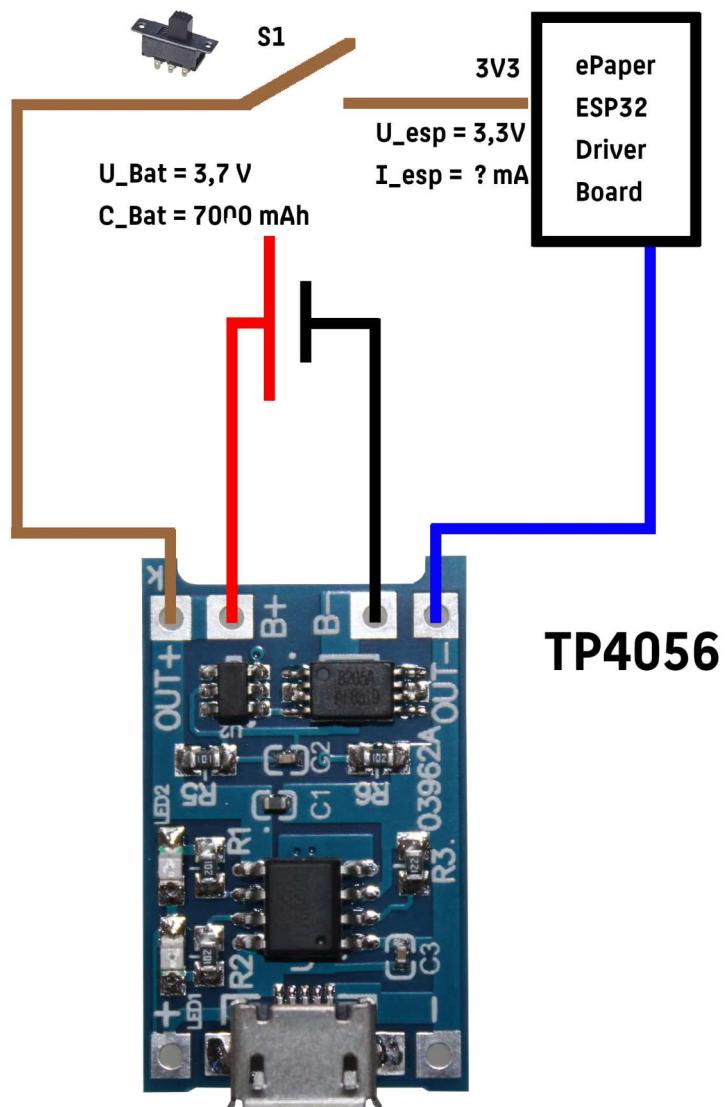


Abbildung 4.1: Schaltbild zur Planung der Positionierung des Schalter S1 als Ein- und Ausschalter

## 4.3 Laufzeit

Die Laufzeit des Akku soll mindestens 6 Monate sein.

Auf der Website des Digikey[TDK24] ist es möglich online die Laufzeit für den Akku zu berechnen:

Dazu wird die

- *Nennkapazität* des Akkus mit **7000 mAh** eingetragen,
- und der **Geräteverbrauch** des Verbrauchers **mA** angegeben.

### Gesamtverbrauch von 80 mA

Mit einer Annahme der Leistungsaufnahme von 80 mA, oder **0,08 A** *Operating Current*[Sys22a]<sup>1</sup> liefert die Webseite des Digikey[TDK24] eine **Laufzeit von 0.122 Monate** dieses Ergebnis erfüllt das die Anforderung nach der Laufzeit nicht.

In der Annahme mit einer Leistungsaufnahme von **80 mA** aus dem Datenblatt[Sys22a], berücksichtigt nicht die Deep-sleep Phasen des Infodisplay. Die sind mehr als 23h pro Tag.

**Gesamtverbrauch von 0,0065 mA** Mit einer Annahme der **Leistungsaufnahme** von **6,5  $\mu$ A** oder **0,00065 mA** bzw. **0,0000065 A** mit derselben Batteriekapazität liefert die Webseite vom Digikey[TDK24] **1495,726 Monate**.

Mit dieser Abschlussarbeit soll die Frage geklärt werden ob mit langen **Deep-sleep** Phasen und unter Nutzung des **ULP Coprocessors** oder andere Konzepte diese Anforderung erfüllt werden kann.

---

<sup>1</sup>Hier werde die vordefinierten Power Modes des Esp32 nicht genutzt. Zu den Power Modes gehören:

- Active, Modem-sleep mit 3-30 mA,
  - Light-sleep mit 800  $\mu$ A,
  - Deep-sleep mit 6,5  $\mu$ A,
  - Hibernation Mode mit 4,5  $\mu$ ,
- laut dem Technical Reference Manual von Espressif Systems[Sys22b]

Um die **Leistungsaufnahme** und somit auch den **Nutzstrom** zu ermitteln ist es notwendig **Messungen** von *Strom* und *Spannung* durchzuführen.

Die *Spannung* des Infodisplay soll abgenommen und überwacht werden. Der durch die Infodisplayschaltung fließende Strom soll bestimmt und überwacht werden. Damit der Leistungsabfall während des *Updatevorgangs* und des *Standby* bestimmt und abgelesen werden kann. Mit *Grafana* sollen diese Messwerte als Spannungs-, Strom-, und Leistungskennlinie dargestellt werden.

## 4.4 Schaltung

Hier werden Konzepte zur Erfüllung der Anforderungen für die Schaltung des Infodisplay aufgeführt.

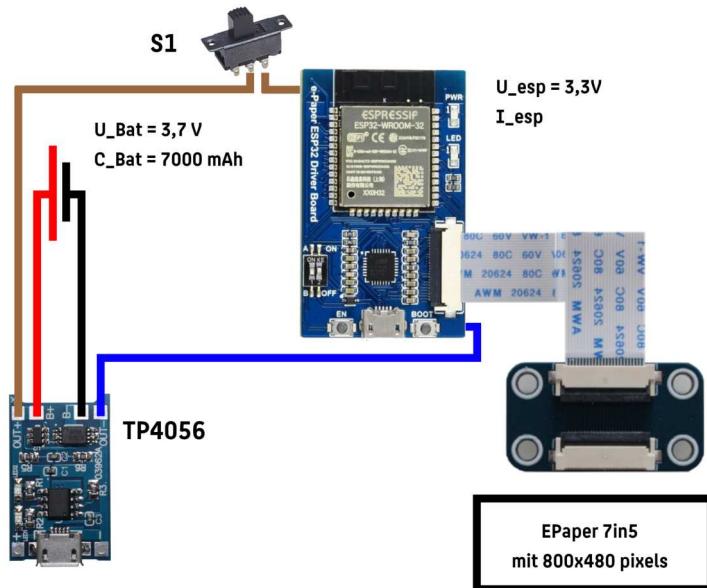


Abbildung 4.2: Schaltbild der Infodisplayschaltung

An der linken unteren Seite des Infodisplay ist der TP4056 mit dem Micro-USB Anschluss zum Laden des Akkus verbaut. Dieser ist mit seinen Anschlüssen Bat + und Bat- mit dem Akku verbunden. Die Anschlüsse Out+ und Out- verlaufen über den Schalter S1 zum 3V Pin und dem GND Pin des Esp32. 4.2

#### 4.4.1 Bauteilliste

Um die Bauteilliste für den Flachen Infodisplay zusammenzustellen werden Bauteilideen des Prototypen weiterverwendet.

Dazu wurde im laufe der Entwicklung des Flachen Infodisplay ein Explorativer Prototyp4.3 gebaut.

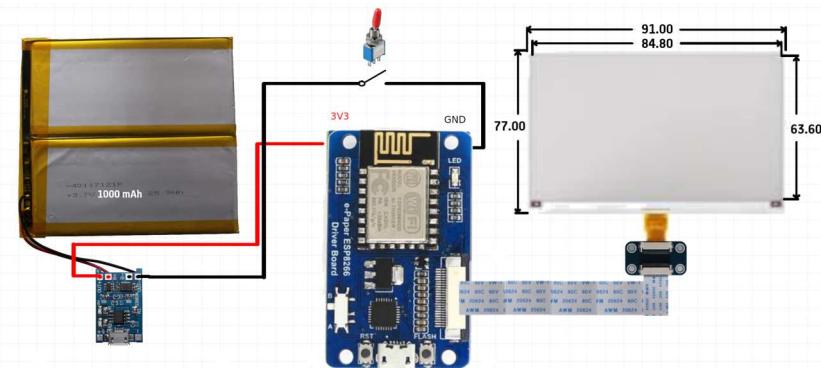


Abbildung 4.3: Schaltplan des Prototypen mit ESP8266 und 4.2 inch Display

Mit dem Prototypen konnten Lösungsmöglichkeiten für Probleme, die während der Hardwareentwicklung entstanden. Und ein vorläufiges Ergebnis für die Software getestet.



Abbildung 4.4: Erster Prototyp ESP8266 1000 mAh Akku und 4.2 inch ePaper der mithilfe zweier Hobbyglas platten steht

Der Prototyp unterscheidet wenigen Bauteilen vom Flachen Infodisplay. Augenmerk ist

es der kleinere Display mit 4.2 inch und 400x300 pixels.<sup>2</sup>

Für den Prototyp-4in2 mit ESP82664.3 wurden folgende Bauteile verwendet:

1. 1x **Waveshare 4.2 inch Display 400x300**
2. 1x **Driver Board ESP8266**
3. 1x LiPo 3V7 1000mAh
4. 1x **TP4056 XD-58A 5V 1A Lithium Charging Board Micro-USB Portable Battery**
5. 4x VEAL-MUTTER M6x30 DIN 7991 - **Abstandhalter**
6. 8x SEKO-SCHR.INNEN-6-KT A4 M6x20 DIN 7991 - **Gewindeschrauben**

Der Flache Infodisplay sollte ähnliche Bauteile aufweisen:

1. 1x **Waveshare 7.5inch E-Ink Display 800x480**
2. 1x Waveshare Universal e-Paper Raw Panel **Driver Board ESP32 WiFi/Bluetooth**
3. 1x LiPo 3,7 V / 7000 mAh
4. 1x TP4056 XD-58A 5V 1A Lithium Charging Board Micro-USB Portable Battery
5. 4x VEAL-MUTTER M6x30 DIN 7991 - Abstandhalter
6. 8x SEKO-SCHR.INNEN-6-KT A4 M6x20 DIN 7991 - Gewindeschrauben
7. zwei Hobbyglasplatten

---

<sup>2</sup>Der 4in2 Display im Shop:<https://www.waveshare.com/4.2inch-e-paper.htm>

#### 4.4.2 Technische Zeichnungen

Die Hobbyglasplatte wird ausgestellt um so eine flache Bauweise des Infodisplay zu gewährleisten. So kann der ESP32 kurz angeschlossen werden.

Um den Infodisplay zu verarbeiten wird der Lasercutter der HTW-Berlin empfohlen. Mit der Vektorgrafik kann im Makers-Lab der Lasercutter unter Anweisung der Studentischen Mitarbeiter der Rahmen und die

**Lasercutter im makers-Lab:**



Abbildung 4.5: Lasercutter Trotec Speedy 400 im Makers Lab der HTW-Berlin

*Vektorgrafik:*

Eine Vektorgrafik als svg für den Lasercutter der HTW. Diese wird über USB an den Trotec Speedy 400 im Makers Lab geschickt und sie aus und lasert die Ausstellungen aus dem Hobbyglas.



Abbildung 4.6: Vektorgrafik einer Hobbyglasplatte 500x500mm für den Lasercutter

#### 4.4.3 Pin-Belegungstabellen

##### Esp32 Driver-Board Pin-Belegung

Der Infodisplay lässt sich mit einem Micro-USB-Kabel an einer USB-Schnittstelle mit bis zu 5V betreiben. Dazu wird das Display an den Esp32-Driver angeschlossen. Im weiteren Verlauf wird die Schaltung in 4.2 mit folgender Pin Belegungstabelle realisiert.

Die Pin-Belegung des Display ist Hardwareseitig intern verdrahtet. D.h. es müssen keine zusätzlichen Pins belegt werden um den Display anzusteuern. Nur eine konstante Stromquelle wie in Form eines USB-Netzteils oder eines LiPo-Akkus ist Voraussetzung.

In der der Display-Klasse<sup>3</sup> werden die Display-maße, die startkoordinaten zur Zeichnung des Bildes und die SPI-Pins für

- serial clock: sck = Pin(13)

<sup>3</sup>[https://gitlab.rz.htw-berlin.de/s0551006/ultralowpowerinfodisplay/-/blob/main/Abgabe/RAW\\_Abgabe\\_7in5\\_Telegram\\_Display\\_ESP32\\_mit\\_epaperBoard/showTable.py](https://gitlab.rz.htw-berlin.de/s0551006/ultralowpowerinfodisplay/-/blob/main/Abgabe/RAW_Abgabe_7in5_Telegram_Display_ESP32_mit_epaperBoard/showTable.py) - Anzumerken ist das dieser spezifische Display-Code austauschbar mit den Klassen aus dem source-folder des Repository sind

- data command: Pin(27)
- chip select: Pin(15)
- beschäftigt: busy = Pin(25)
- reset: rst = Pin(26)
- master out slave in: mosi = Pin(14)

```
1 import epaper7in5_V2
2
3 serial_clock = Pin(13)
4 data_command = Pin(27)
5 chip_select = Pin(15)
6 busy = Pin(25)
7 reset = Pin(26)
8 master_out_slave_in = Pin(14)
9
10 serial_peripheral_interface = SPI(2, baudrate=20000000,
11 polarity=0, phase=0,
12 sck=serial_clock,
13 mosi=master_out_slave_in)
14
15 e = epaper7in5_V2.EPD(serial_peripheral_interface,
16 chip_select, data_command,
17 reset, busy)
18
19 e.init()
20 print('7.5 Display initialized')
21
22 e.clear()
23 print('displayed cleared')
```

Code snippet 4.1: Initialisierung der Pins und Instanzierung des Epaper-Display

### Strom-Sensor Pin-Belegung

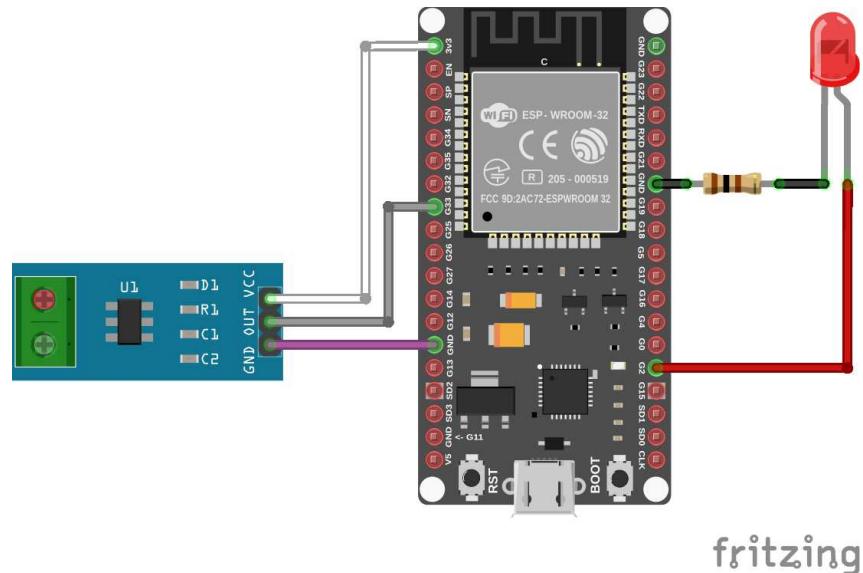


Abbildung 4.7: Breadboard Schaltung der Strommessschaltung

Der Strom Sensor mit Vcc, S und GND sollen mit 3V3, 33 und GND des ESP32 verschaltet werden. LED Kathode an Pin 2 des ESP32 und Anode über einen Widerstand von 100 Ohm auf GND. Wie in Abbildung 4.7 zu sehen.

Spalte 1	Spalte 2	Spalte 3
1	VCC	3V3 Power supply
2	S	Signal
3	GND	Ground

Tabelle 4.1: Pin Belegungstabelle für die Strommessung

Im currentButterfly7.py werden die Pins für die Strommessung und der LED gesetzt, siehe dazu Code-listing 4.2

```

1  from machine import ADC, Pin
2  import time
3
4  class Current:
5      def __init__(self):
6          self.current_s = ADC(Pin(33))
7
8          infoLED = Pin(2, Pin.OUT)
9          infoLED.on()

```

```

10      ...
11      infoLED.off()
12      ...

```

Code snippet 4.2: Initialisierung der Pins im Konstruktor der Klasse für den Strom Sensor in der currentButterfly7.py

### Spannungs-Sensor Pin-Belegung

Der Spannungssensor ist mit Vcc, S und GND mit 3v3 und GND des ESP32 verbunden. Kathode der LED ist mit dem Pin 2 ESP32 verdrahtet und die Anode über einen Widerstand von 100 Ohm auf den GND des Mikrocontroller. Das ist in Abbildung zur Spannungsmessschaltung 4.8 grafisch Dargestellt.

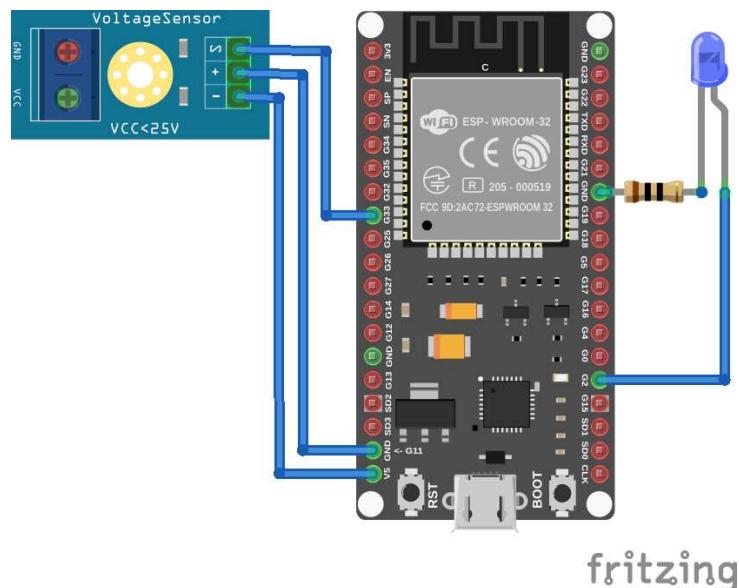


Abbildung 4.8: Grafische Darstellung der Spannungsmessschaltung

Pins	Name	Beschreibung
1	VCC	3V3 Power supply
2	S	Signal
3	GND	Ground

Tabelle 4.2: Pinbelegung Spannungsmessung

Im voltageButterflyV3.py werden die Pins für die Strommessung ähnlich die der Strommessung gesetzt, das ist nochmal im Code-listing 4.2 nochmal aufgeführt.

```
1      from machine import ADC, Pin
2      import time
3
4      class Voltage:
5          def __init__(self):
6              adc_pin = 33
7
8              infoLED = Pin(2, Pin.OUT)
9              infoLED.on()
10             ...
11             infoLED.off()
12             ...
```

Code snippet 4.3: Initialisierung der Pins im Konstruktor der Klasse für den Spannungssensor in der voltageButterflyV3.py

#### 4.4.4 Strom-,Spannung-, und Leistungskennlinien

Für die Ergebnisse der Untersuchung siehe Seite auf Seite 97

### 4.5 Pinbelegung

Siehe hierzu: Pinbelegung

### 4.6 Wartbarkeit und Schneller Einstieg

Zur Wartbarkeit des Codes und für einen schnelle Einstieg in die Entwicklung des Infodisplay wird eine GitLab Repo zur Verfügung gestellt<sup>4</sup>.

Readme's in Markdown unterstützen den Nutzer bei der Orientierung.

---

<sup>4</sup>Link zur Repo: <https://gitlab.rz.htw-berlin.de/s0551006/ultralowpowerinfodisplay>

Siehe dazu auch Seite 7.6

## 4.7 Aktualisieren

Diese Anforderung kann mit Node-RED einer Trigger-Node, die jeden Wochentag um 5 Uhr ein Display-Refresh über das MQTT-Topic: „B\_EiNK/006/image“ auslösen erfüllt werden.

Ein andere Ansatz wäre die Implementierung von MicroCRON eine zeitlich-basiertem Task Anwendung für micropython[fiz20].

Die Evaluation mit dem Node-Red Trigger-node wird auf Seite 101 durchgeführt.

**Partielles Aktualisieren** Noch nicht Implementiert

**Bitmap an Infodisplay** Es sollen Images über einen Node-Red Flow von Telegram empfangen werden und über MQTT als Buffer an den Infodisplay Versand.

Siehe dazu Evaluation auf Seite 102

## 4.8 Systemarchitektur

Ein Node-Red Flow mit den Nodes für Telegram, MQTT und Influx. Die InfluxDB Daten werden dann in einer Grafana Instanz visualisiert.

**Gesamtentwurf** des Systems, einschließlich Hardware- und Softwarearchitektur.

Zur Infodisplay Systemarchitektur s. 7.10

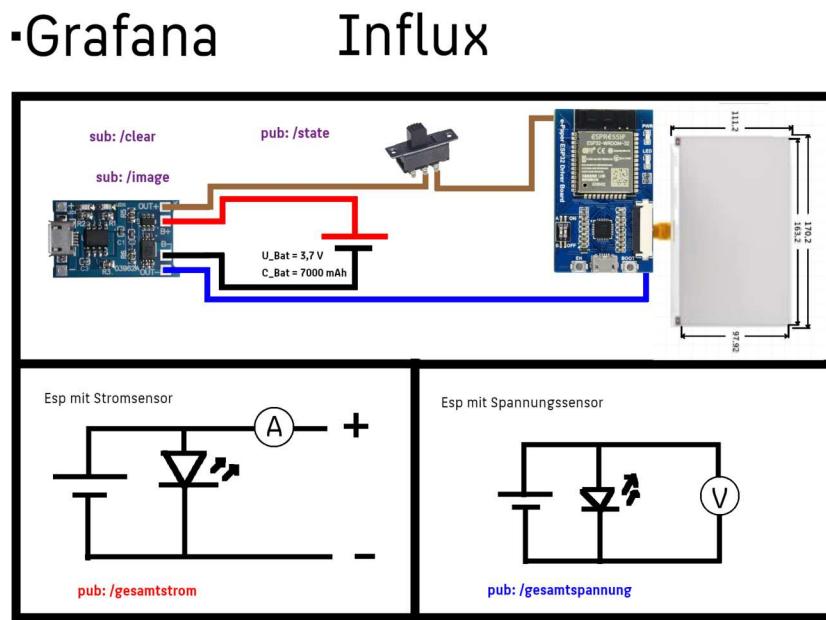
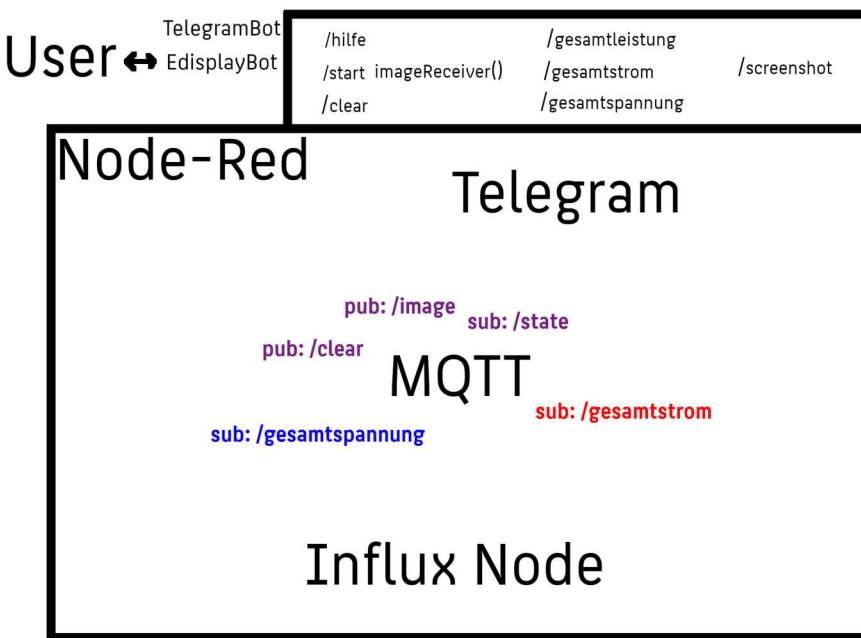


Abbildung 4.9: Erster Gesamtentwurf

Anwendungsfalldiagramm zum Konzept der Integration in eine Messaging-Plattform.

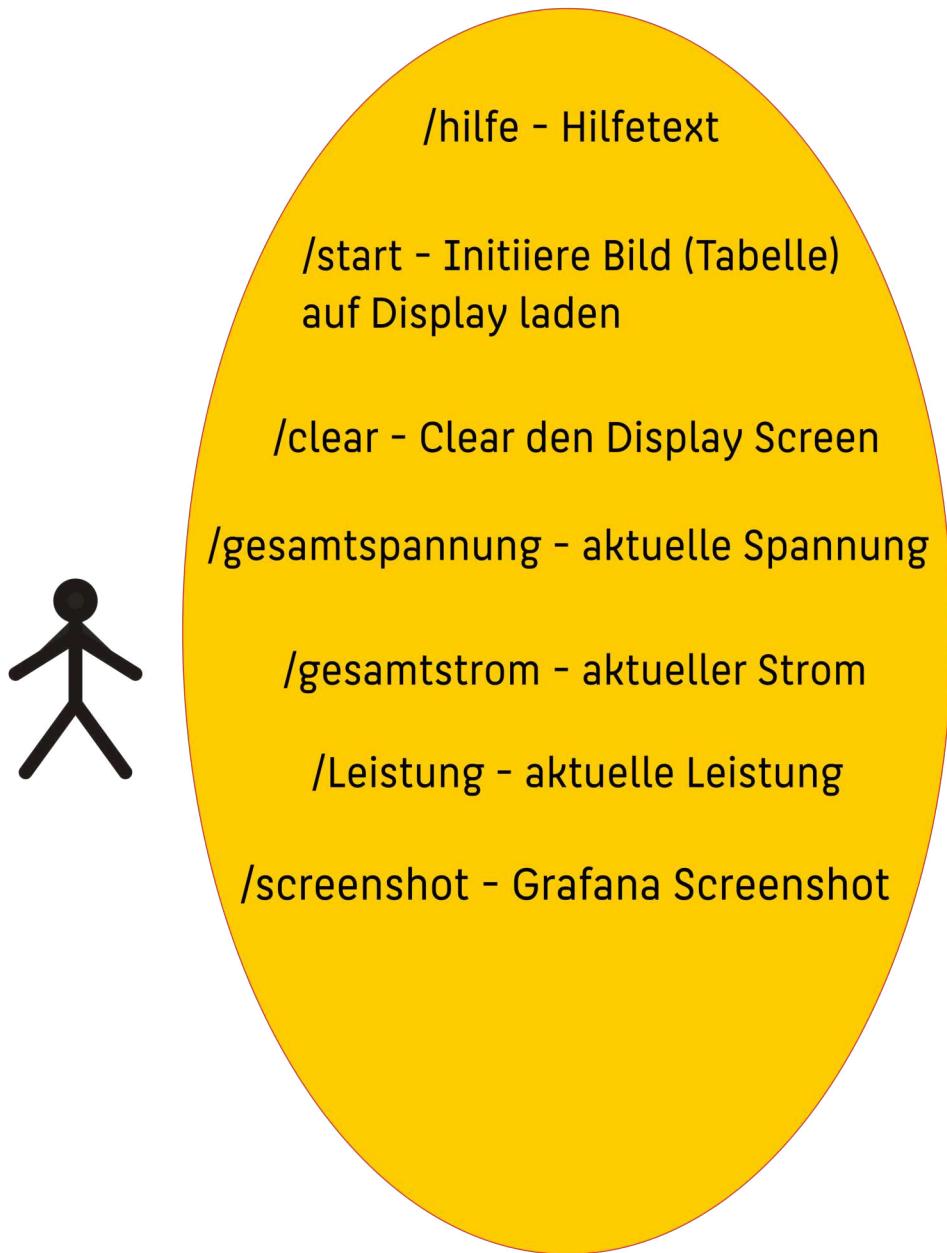


Abbildung 4.10: Anwendungsfalldiagramm: User nutzt den Telegram EDisplayBot

Sequenzdiagramm für den Anwendungsfall: Abfrage der aktuellen Spannung die am Info-display anliegt.

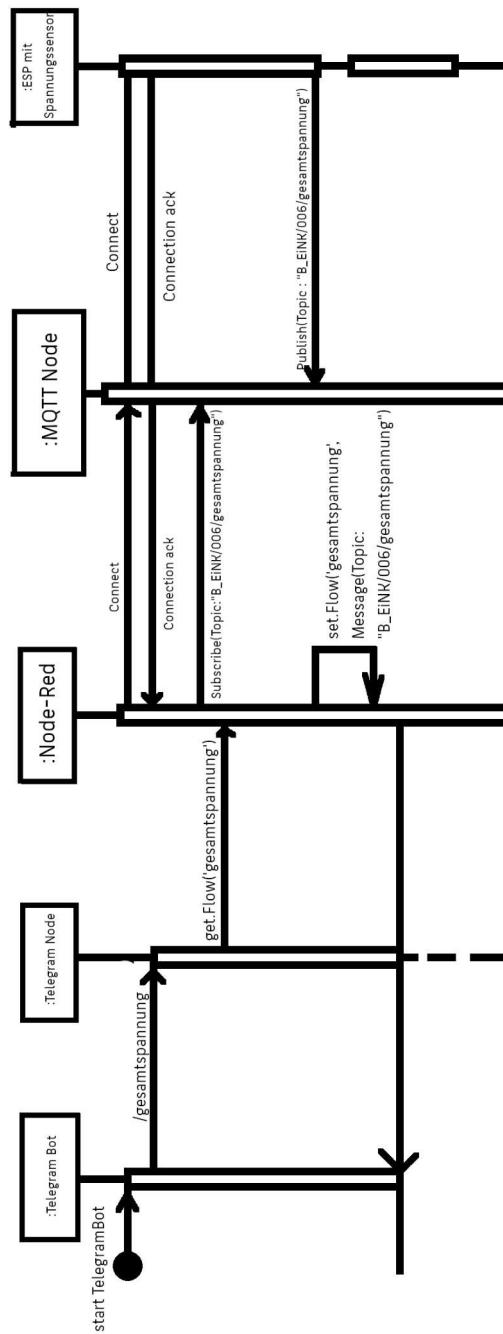


Abbildung 4.11: Sequenzdiagramm: User schickt „/gesamtspannung“ als Text an den TelegramBot:EDisplayBot

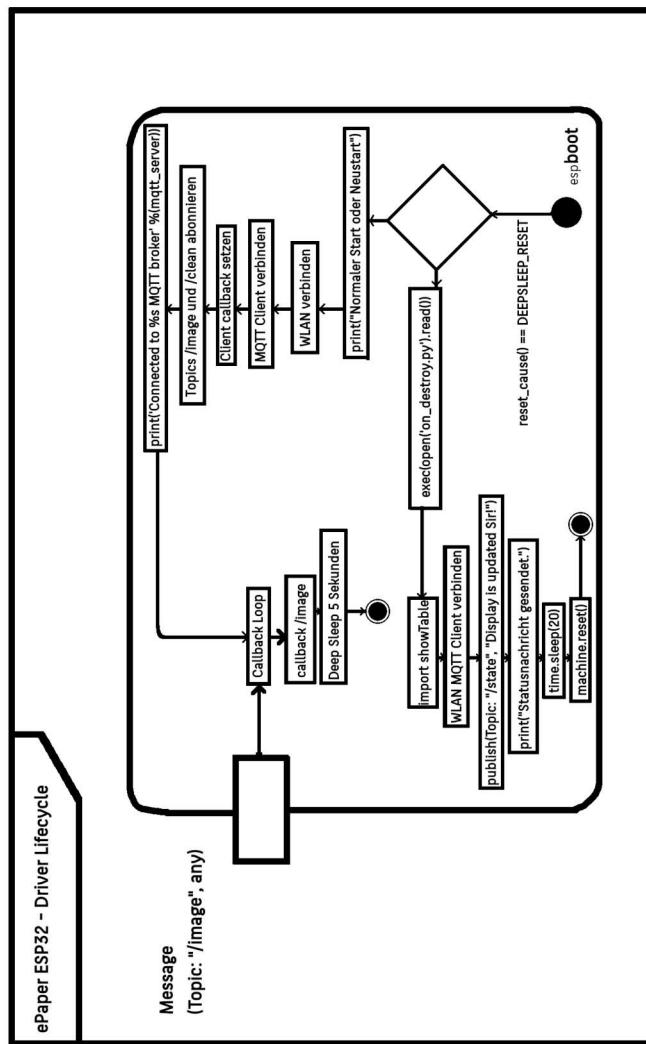


Abbildung 4.12: Activity Diagramm: Display refresh beim erhält einer Nachricht auf dem Topic: „\_EiNK/006/image“,

Siehe auch Evaluation auf Seite 103 oder Fazit.

## 4.9 Powerbank

Durch die Benutzung des TP4056 wird das Laden mit einer Powerbank gewährleistet.

Siehe dazu Evaluation auf Seite 103

## 4.10 Leistung

Die Leistung des Infodisplay soll anhand einer Messumgebung über MQTT berechnet und in Grafana visualisiert werden.

Ein Esp Strom-Messpunkt der die Stromstärke des Infodisplay auf dem Topic /gesamstrom published.

Einem Esp Spannungs-Messpunkt der die Spannung des Infodisplay auf dem Topic /gesamtspannung published.

Um so die Leistung an einer Stelle im System zu berechnen.

Siehe dazu Evaluation auf Seite 104

## 4.11 Tests

Die Anforderung ?? aus der Analyse möchte einen Testplan in der Test Suite haben um somit einen Überblick über die Testumgebung zu bekommen. Die Anforderung verlangt manuelle und automatisierte Test.

D.h. aus den manuellen Testfällen(Anwendungsfälle) können automatisierte Testfälle geschrieben werden.

Mit einem gut strukturierten Testplan, ist es möglich alle Testfälle in Automatisierte Testfälle in der Testumgebung zu überführen.

Wenn alle Automatisierten Testfälle Bestehen, kann eine Aussage zur Überführung in eine Produktivumgebung der HTW getroffen werden. Dazu muss der Fachbereich 4 seine Einwilligung erteilen.

In Ausblick stehen Telegram Tests, MicroPython Test mit unittest, CD/CI-Pipelines und analoge Multimeter Tests.

Manuelle Tests mit dem Multimeter sind in der Abschlussarbeit mit Entwicklung des Infodisplay und einer Messumgebung vollständig automatisiert.

Siehe dazu Evaluation auf Seite 103

# Kapitel 5

## Implementierung

In diesem Kapitel werden die Anforderungen aus der Analyse anhand von Ideen aus der Konzeption implementiert.

### 5.1 Hardware

#### 5.1.1 Infodisplay

**Infodisplayschaltung** Die Hardwareschaltkreis des Infodisplay besteht aus einem ESP32 ePaper Driver-Board mit 7,5 Zoll Display. Diese *Mikrocontroller-Display* Konfiguration ist über einen *mini-slide-switch S1*, angeschlossen an den *USB-Charger TP4056* mit dem **7000 mAh LiPo-Akku**.

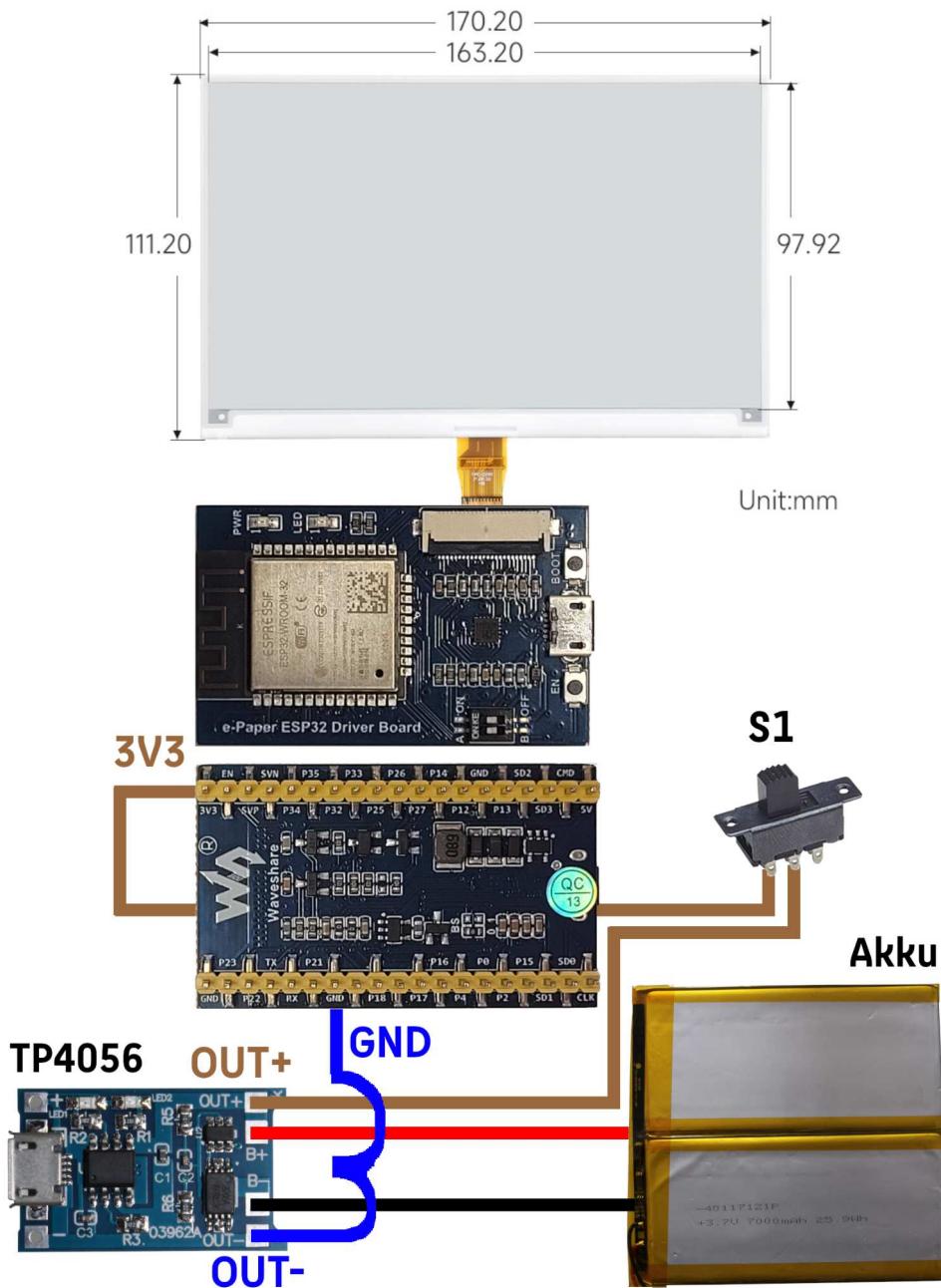


Abbildung 5.1: Hardwareschaltung des Infodisplay

### 5.1.2 Sensor Messung

**Spannungsmessung** Die Spannungsmessung erfolgt mit einem Voltage Sensor von *MH-Electronic* an einem ESP-Wroom-32 und einer LED L1<sup>1</sup> mit einem Vorwiderstand von 100 Ohm.

Die Hardwareschaltung wird auf Basis der Abbildung 4.8 im Kapitel 4 mit der Pin-Belegungstabelle 5.1 zu einer Schaltung 5.2 verschaltet.

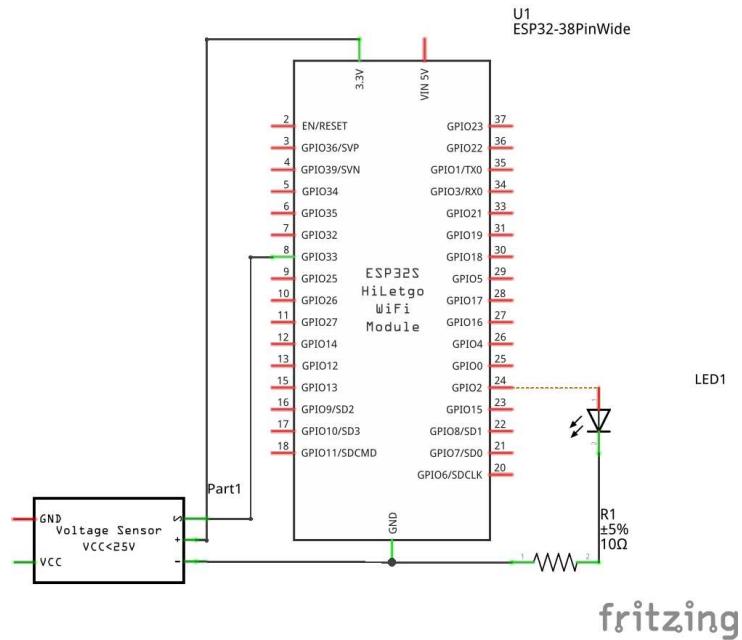


Abbildung 5.2: Pinbelegung der Spannungsmessschaltung

Pins	Name	Beschreibung
1	S	Signal
2	+	3V3 Power supply
3	-	Ground
4	R1	Widerstand 100 Ohm
5	D2	LED-Pin

Tabelle 5.1: Spannungsmessung Pinbelegung

<sup>1</sup>LED: 3 mm blau (1,25 cd-20°) 468 nm klar

**Strommessung** Die Strommessung erfolgt mit ACS-712 Current Sensor an einem ESP-Wroom-32 und einer LED L1<sup>2</sup> mit einem Vorwiderstand von 100 Ohm.

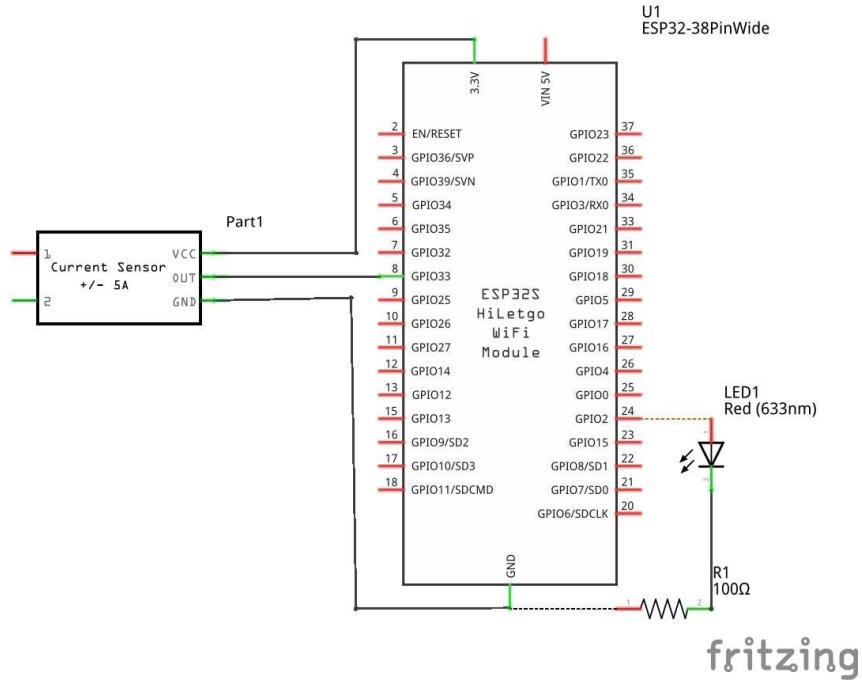


Abbildung 5.3: Pinbelegung der Strommessschaltung

Die Hardwareschaltung wird auf Basis der Abbildung 4.7 im Kapitel 4 mit der Pin-Belegungstabelle 5.2 zu einer Schaltung 5.3 verschaltet.

Pins	Name	Beschreibung
1	VCC	3V3 Power supply
2	OUT	Signal
3	GND	Ground
4	R1	Widerstand 100 Ohm
5	D2	LED-Pin

Tabelle 5.2: Strommessung Pinbelegung

### 5.1.3 Messumgebung

<sup>2</sup>LED: 3 mm rot (3,7 cd-25') 628 nm klar

(a) Breadboard Schaltung Messumgebung

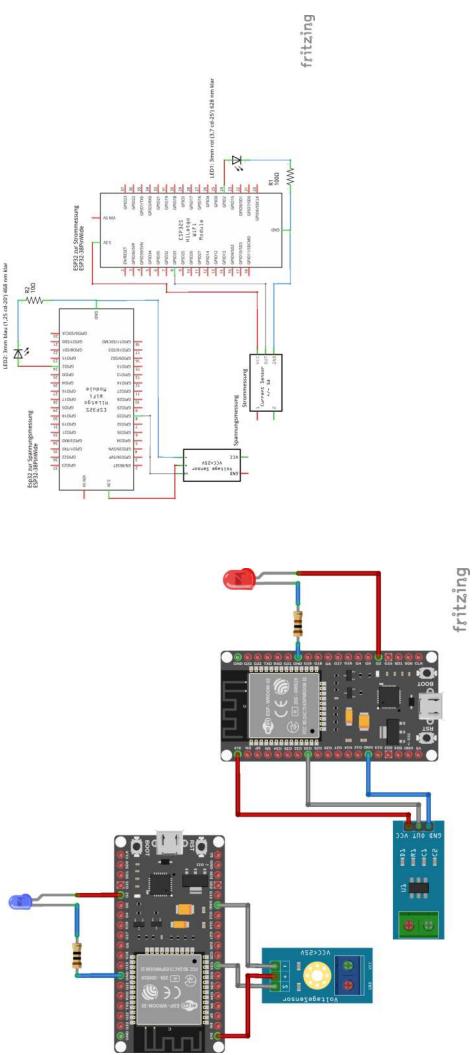


Abbildung 5.4: Leistungsmessschaltung über MQTT und Telegram

Zur Berechnung der Leistung aus den gemessenen Werten für Strom und Spannung wird eine Messschaltung bestehend aus einer Reihenschaltung zur Strommessung und einer Parallelschaltung zur Spannungsmessung aufgebaut.

Der Infodisplay wird während des Messvorgangs nur über den eigenen 7000mAh Akku betrieben.

Für jede Message sollte nun in jeder Sensor-Schaltung, eine Led blinken. Blau für die Spannung und Rot für den Strom.

Nach Kalibrierung der Sensorschaltungen wird der Infodisplay zur Leistungsbestimmung über die Schraubklemmen(Schnittstellen zum Messung) angeschlossen.

Nun sollten schon die ersten Messungen über die Indikator Led's sichtbar sein. Und erste Messwerte sollten den Schnittstellen zur Verfügung gestellt worden sein.(In Grafana; In Node-Red; Mit Telegram; In Influx)

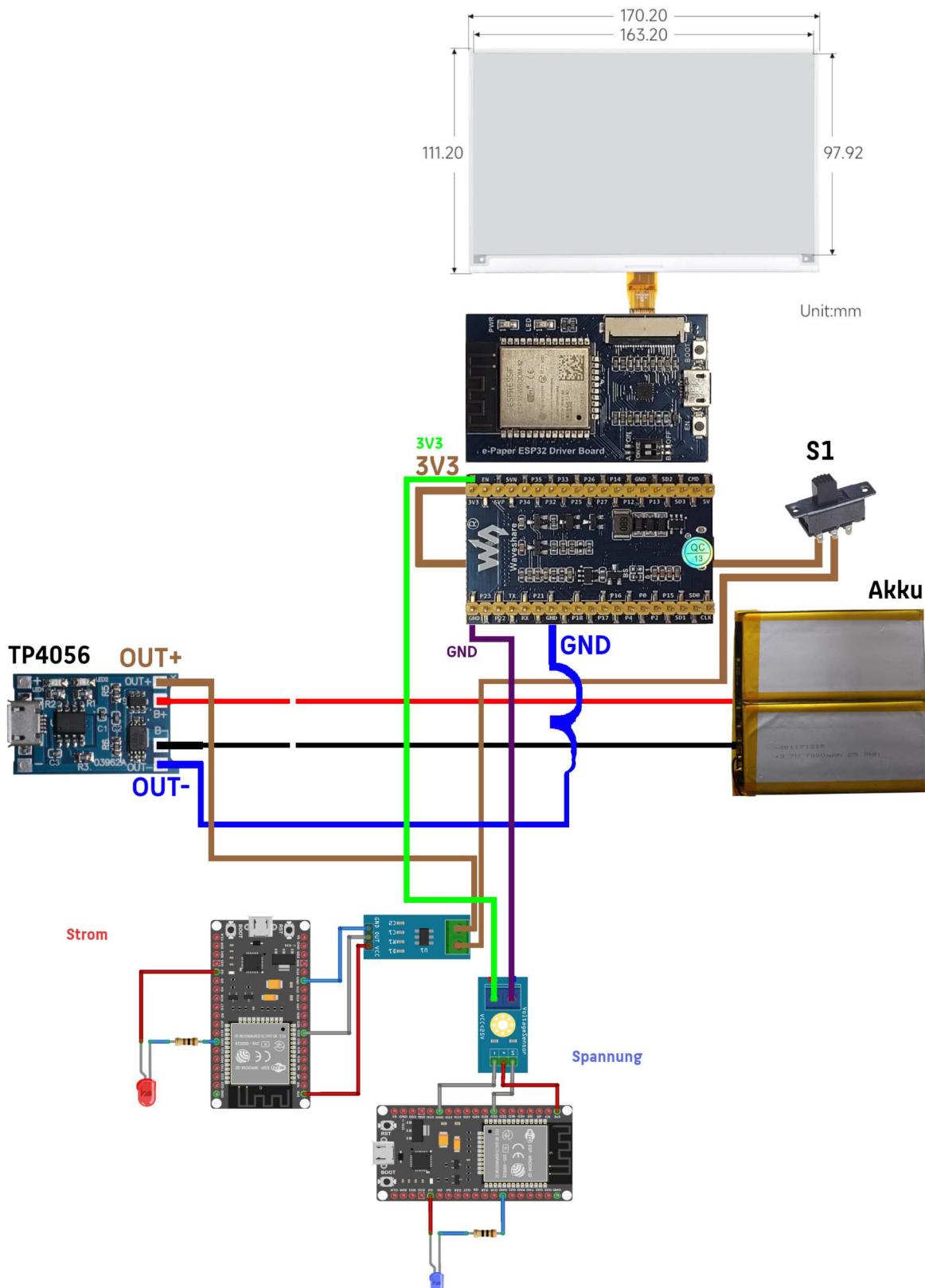
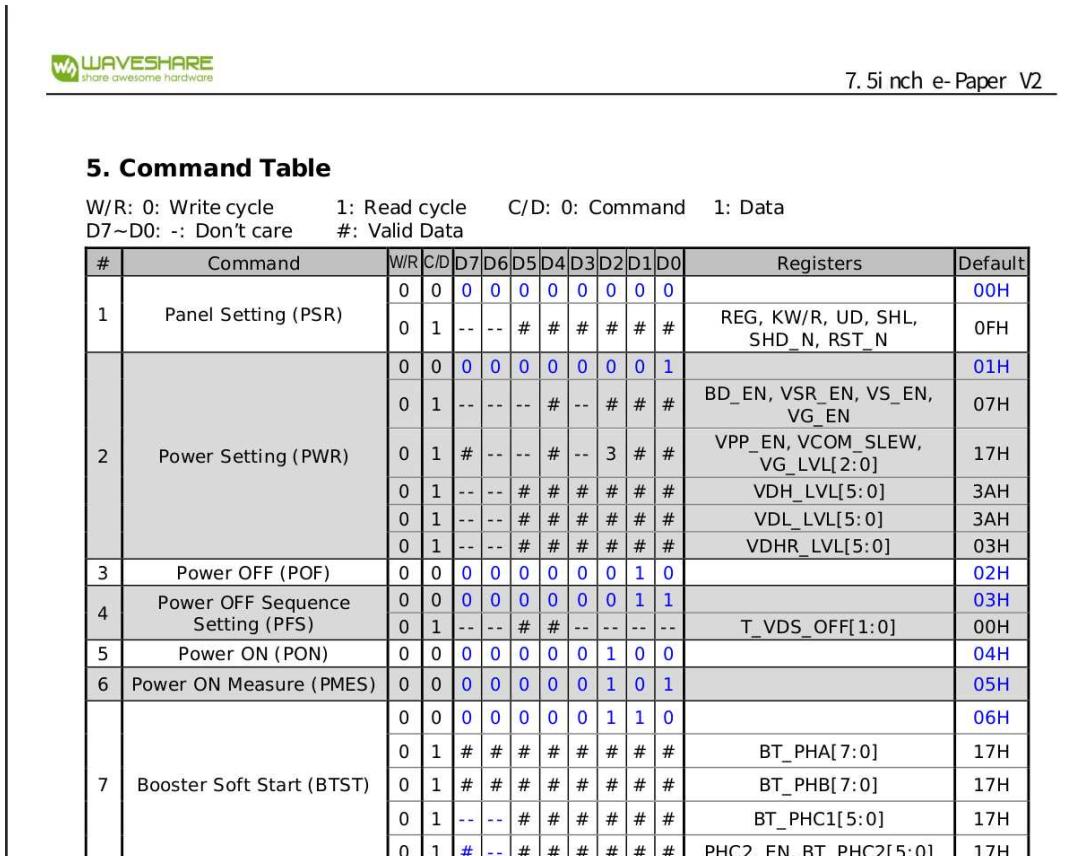


Abbildung 5.5: Gesamte Messschaltung

## 5.2 Software

[https://www.informatik-verstehen.de/?s=treiber&post\\_type=glossary](https://www.informatik-verstehen.de/?s=treiber&post_type=glossary)

Die Datei epaper7in5\_V2.py ist der Displaytreiber. Der sogenannte *driver* oder auch *display-driver* für den 7in5-Eink-Screen.[Wav24d]



The screenshot shows the Waveshare 7.5 inch e-Paper V2 datasheet. At the top left is the Waveshare logo with the tagline "share awesome hardware". At the top right is the model name "7.5i nch e-Paper V2". Below the header is the section title "5. Command Table". Underneath the title is a legend: "W/R: 0: Write cycle" (blue), "1: Read cycle" (blue), "C/D: 0: Command" (blue), "1: Data" (blue), "D7~D0: -: Don't care" (grey), and "#: Valid Data" (blue). The command table itself is a grid where rows represent commands and columns represent bits D0 through D7. The first few rows show the Panel Setting (PSR) command, which consists of two bytes. The second byte has bit D0 set to 1, while all other bits are don't cares. Subsequent rows show other commands like Power Setting (PWR), Power OFF (POF), Power OFF Sequence Setting (PFS), Power ON (PON), Power ON Measure (PMES), and Booster Soft Start (BTST). Each command row includes its hex value in the first column and a list of control parameters in the subsequent columns.

#	Command	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0	Registers	Default
1	Panel Setting (PSR)	0	0	0	0	0	0	0	0	0	0		00H
		0	1	--	--	#	#	#	#	#	#	REG, KW/R, UD, SHL, SHD_N, RST_N	0FH
2	Power Setting (PWR)	0	0	0	0	0	0	0	0	0	1		01H
		0	1	--	--	#	--	#	#	#	#	BD_EN, VSR_EN, VS_EN, VG_EN	07H
3	Power OFF (POF)	0	1	#	--	--	#	--	3	#	#	VPP_EN, VCOM_SLEW, VG_LVL[2:0]	17H
		0	1	--	--	#	#	#	#	#	#	VDH_LVL[5:0]	3AH
4	Power OFF Sequence Setting (PFS)	0	1	--	--	#	#	#	#	#	#	VDL_LVL[5:0]	3AH
		0	1	--	--	#	#	#	#	#	#	VDHR_LVL[5:0]	03H
5	Power ON (PON)	0	0	0	0	0	0	0	1	0	0		02H
6	Power ON Measure (PMES)	0	0	0	0	0	0	0	1	0	1		03H
7	Booster Soft Start (BTST)	0	1	--	--	#	#	--	--	--	--	T_VDS_OFF[1:0]	00H
		0	0	0	0	0	0	0	1	1	0		04H
		0	1	#	#	#	#	#	#	#	#	BT_PHA[7:0]	05H
		0	1	#	#	#	#	#	#	#	#	BT_PHB[7:0]	06H
		0	1	--	--	#	#	#	#	#	#	BT_PHC1[5:0]	17H
		0	1	#	--	#	#	#	#	#	#	PHC2_EN, BT_PHC2[5:0]	17H

Abbildung 5.6: Auszug aus der Command Table im Datenblatt ??

Im Driver-Code werden, zunächst die Display-spezifischen Hexadezimal-Werte aus dem Datenblatt von Waveshare[Wav24b] (s. Abbildung 5.6), in *MicroPython* konstanten gespeichert. Anschließend wird die Klasse **EPD** wie sie in Abbildung 5.7 dargestellt ist implementiert.

```

1      ...
2      lut_vcom0 = [
3          0x00, 0x17, 0x00, 0x00, 0x00, 0x02,
4          0x00, 0x17, 0x17, 0x00, 0x00, 0x02,
```

```
5      0x00, 0x0A, 0x01, 0x00, 0x00, 0x01,
6      0x00, 0x0E, 0x0E, 0x00, 0x00, 0x02,
7      0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
8      0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
9      0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
10     ]
11     ...
```

Code snippet 5.1: lookUpTableVCom0

```
epaper7in5_V2.py
```

EPD_WIDTH	= const(800)	DATA_START_TRANSMISSION_2	= const(0x13)
EPD_HEIGHT	= const(480)	DUAL_SPI	= const(0x15)
PANEL_SETTING	= const(0x00)	PLL_CONTROL	= const(0x30)
POWER_SETTING	= const(0x01)	TEMPERATURE_CALIBRATION	= const(0x40)
POWER_OFF	= const(0x02)	VCOM_AND_DATA_INTERVAL_SETTING	= const(0x50)
POWER_ON	= const(0x04)	TCON_SETTING	= const(0x60)
BOOSTER_SOFT_START	= const(0x06)	RESOLUTION_SETTING	= const(0x61)
DEEP_SLEEP	= const(0x07)	GET_STATUS	= const(0x71)
DATA_START_TRANSMISSION_1	= const(0x10)	VCM_DC_SETTING	= const(0x82)
DISPLAY_REFRESH	= const(0x12)		

**EPD**

---

- lut\_vcom0: List<HEX>
- lut\_ww: List<HEX>
- lut\_bw: List<HEX>
- lut\_wb: List<HEX>
- lut\_bb: List<HEX>

---

- + \_\_init\_\_(spi,cd,dc,rst,busy)
- + reset()
- + \_command(command:HEX)
- + \_data(data:HEX)
- + wait\_until\_idle()
- + clear()
- + sleep()
- + init()
- + set\_lut()
- + display(frame\_buffer:
   
framebuf.FrameBuffer(bytearray(w\*h//8),w,h,framebuf.MONO\_HLSB))
- + display\_frame(frame\_buffer:
   
framebuf.FrameBuffer(bytearray(w\*h//8),w,h,framebuf.MONO\_HLSB))

Abbildung 5.7: Driver-Code mit der EPD-Klasse als Entität mit Typen der Attribute und Signaturen der Operationen

### 5.2.1 Näheres zum Driver Code

**Attribute** Die EPD-Klasse initialisiert zunächst die Look Up Table wie sie im Codelisting 5.1 zu sehen ist. Was diese Look Up Table sind, schreibt Waveshare dazu:

„LUT is the abbreviation of LOOK UP TABLE, and OTP is the abbreviation of ONE TIME PROGRAM. The original intention of LUT is to load waveform files, and the waveform files are divided into OTP and REGISTER. Among them, OTP is the built-in waveform storage method, and REGISTER is the external waveform storage method.“

auf Ihrer Website [https://www.waveshare.com/wiki/4.2inch\\_e-Paper\\_Module\\_Manual](https://www.waveshare.com/wiki/4.2inch_e-Paper_Module_Manual)

Eine andere GitHub-Quelle schreibt dazu:

„The waveform is a look up table with timings for the epaper display controller to determine how to drive the pixels.“

Doch aus Zeit-technischen Gründen wird in dieser Arbeit nicht näher LUT, OTP oder Waveformfiles eingegangen.

### Methode

Alle Methoden der Klasse EPD sind public und stehen somit, mit Instanziierung der EPD-Klasse als ePaper-Objektmethoden zur Verfügung.

Die Python-Methode `__init__` wird bei ePaper-Objektinstanzierung ausgeführt.

Die Überabeparameter der `__init__`-Methode sind zum einen ein Objekt der Klasse SPI. D. h. der Implementierung der Seriellen-Busschnittstelle in Micropython.

class SPI - a Serial Peripheral Interface bus protocol (controller side)[Con24]

.....

Weitere Übergabe Parameter sind laut Datenblatt[wav19], S. 6, Input/Output Terminals, Pin out List

```

1      serial_clock = Pin(13)
2      # Takt der Daten nachdem Master und Slave
3      # synchronisiert werden
4
5      data_command = Pin(27)
6      # Data/Command Signal, zur Steuerung des Modus

```

```

7      # (Daten/Kommandos) bei der Display-Kommunikation
8
9      chip_select = Pin(15)
10     # Chip Select Signal, zur Auswahl des SPI-Slave
11
12     busy = Pin(25)
13     # Signal da anzeigt an, ob das Gerät beschäftigt ist
14
15     reset = Pin(26)
16     # Reset Signal, um das Gerät zurückzusetzen
17
18     master_out_slave_in = Pin(14)
19     # Master Out Slave In, Datenleitung im SPI

```

Code snippet 5.2: Die Pins, die die `__init__`-Methode der EPD-Klasse als Übergabeparameter bekommt

Sie stellt mit einem SPI-Objekt als Übergabeparameter und den Pins für Chip-Select<sup>3</sup>, Data-Command, reset und Busy)zur Serielle Kommunikation(sck und mosi) sowie und die restlichen Pins zur Kommunikation mit dem Display, bei Objektinstanzierung

Im weiteren setzt `__init__` die Initialen GPIO-Werte zur weiteren Kommunikation mit dem Display.

Weitere Methoden sind:

`_command(Byte in Hex)` - Zur Übermittlung von Display Kommandos

`_data(Byte in Hex)` - Zur Übermittlung von Daten an den Display

## 5.2.2 Display

In diesem Unterkapitel Display geht es um den Code der für den Infodisplay geschrieben wurde.

Es beginnt mit einer Minimal Anwendung zur Darstellung eines Bildes auf dem Infodisplay.

---

<sup>3</sup>Command/Data Signal dient zur Steuerung des bei Display-Kommunikation

Der Infodisplay braucht um eine Minimal Anwendung zu starten, zwei Python-Dateien. Einmal den den in vorhergehenden beschriebenen Driver-Code oder Displaytreiber, zur Kommunikation mit dem 7,5 Display. Und einmal einen Display-Code als Ausführbare Datei zur Darstellung von Inhalten.

```
1      import framebuf
2      import epaper7in5_V2
3      ...
4      sck = Pin(13)
5      dc = Pin(27)
6      cs = Pin(15)
7      busy = Pin(25)
8      rst = Pin(26)
9      mosi = Pin(14)
```

Code snippet 5.3: Deklaration der Pins

```
1      ...
2      spi = SPI(2, baudrate=20000000,
3                  polarity=0, phase=0,
4                  sck=serial_clock,
5                  mosi=master_out_slave_in)
6
7      e = epaper7in5_V2.EPD(serial_peripheral_interface,
8                  chip_select, data_command,
9                  reset, busy)
10
11     e.init()
12     print('7.5 Display initialized')
13
14     e.clear()
15     print('displayed cleared')
16     ...
```

Code snippet 5.4: Instanzierung der Serielle Schnittstelle:spi und der ePaperDisplay: e

Die Pins zu Seriellen Kommunikation sind notwendig um eine SPI-Instanz zur erzeugen. Diese nutzt der ePaper-Driver dann zur Initialisierung einer epaper-Instanz. Um so dann den Display zunächst zu initialisieren und clearen.

Der Serial Clock Pin des Esp32 ist der Taktgeber zwischen Display und Esp32. Der MOSI-Pin<sup>4</sup> ist die ausgehende Datenleitung vom Esp32 zum Display.

Die weiteren Pins sind:

- Das Data/Command Signal zur Steuerung des Modus bei der Kommunikation mit dem Display.
- Das Chip Select Signal zur Auswahl des SPI-Slave
- Das Busy Signal zur Gerätestatus abfrage
- Das Reset Signal um den Display zurückzusetzen

Diese Deklaration der Pins und die Instanziierung der Klassen SPI und EPD sind in Code-Listing 5.3 und 5.4 zu sehen.

Im weiteren Verlauf muss ein Bytearray-Buffer für die Darstellung des Bildes erzeugt werden.<sup>5</sup>

Der 7in5-Display hat eine Darstellungsbereich für 800\*480 Pixel. Dafür ist eine Bytearray der Größe von 384.000 Pixel notwendig. Da der Heap-Memory des Esp32 Memory Allocation Fehler wirft [Luc24], wird zunächst nur ein Achtel des Darzustellenden Bereich an ByteArray-Code Buffer erzeugt.

```

1      breite=800
2      hoehe=480
3      ...
4      gc.collect()
5
6      main_buffer = bytearray(breite * hoehe // 8)
7      print('main_buffer created')
8
9      main_fb = framebuffer.FrameBuffer(main_buffer, breite,
10        hoehe, framebuffer.MONO_HMSB)
```

Code snippet 5.5: Der Buffer und FrameBuffer für den Infodisplay

---

<sup>4</sup>Master Out Slave In

<sup>5</sup>An dieser Stelle tritt sehr häufig derselbe Fehler auf, wenn der display-code(main) nicht als einzige Anwendung oder bei Neustart läuft: Memory Allocation Error

Dieser wiederum wird der Framebuffer Klasse übergeben, um, mit dem Buffer, der breite und Höhe des Display sowie der Konstanten zum Farbformat. Um so schlussendlich ein FrameBuffer Objekt zu erzeugen, auf dem gezeichnet werden kann. Wie bspw. ein Rechteck mit breite 50px und der Höhe 125px. Dieser Sachverhalt ist in Code-Listing 5.5 aufgelistet.

Dieser Ansatz ist die Grundlage aller folgenden Programmieranwendungen für den Infodisplay.

### Binary - Variante

*binAufEink- mit image-data.bin für Nachricht1*



Abbildung 5.8: Beispiel-Code für eine Darstellung einer Nachricht anhand einer Binary im App-Speicher

*binAufEink- mit image-data.bin für Tabelle*

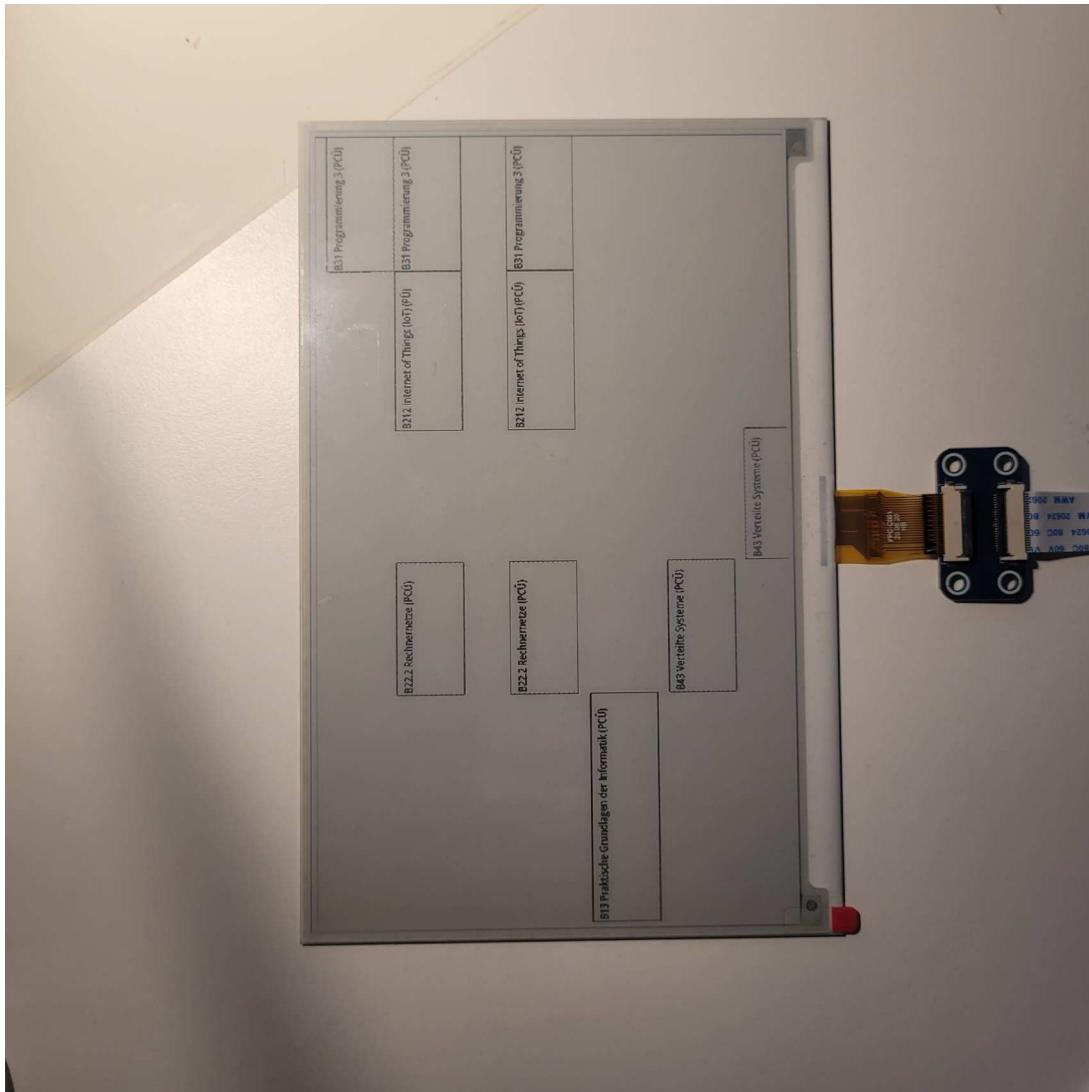


Abbildung 5.9: Beispiel-Code für eine Darstellung einer Tabelle anhand einer Binary im App-Speicher

```
1 import framebuffer
2 from machine import Pin, SPI
3 import epaper7in5_V2
4 ...
5 regulärer display-code (mit e.init() und e.clear())
6 ...
7
8 def display_img():
9     with open("image_bytes.bin", "rb") as f:
```

```
10     chunk_size = 2048 # Size of each chunk, adjust as needed
11     buffer = bytearray(chunk_size)
12
13     index = 0
14     while True:
15         bytes_read = f.readinto(buffer)
16         if bytes_read == 0:
17             break # End of file
18
19         for i in range(bytes_read):
20             if index < len(buf):
21                 buf[index] = buffer[i]
22                 index += 1
23
24         e.display_frame(buf) # Pass the buffer directly to the display
                           function
```

Code snippet 5.6: binAufEink

### Rekursive - Variante: KaroMuster

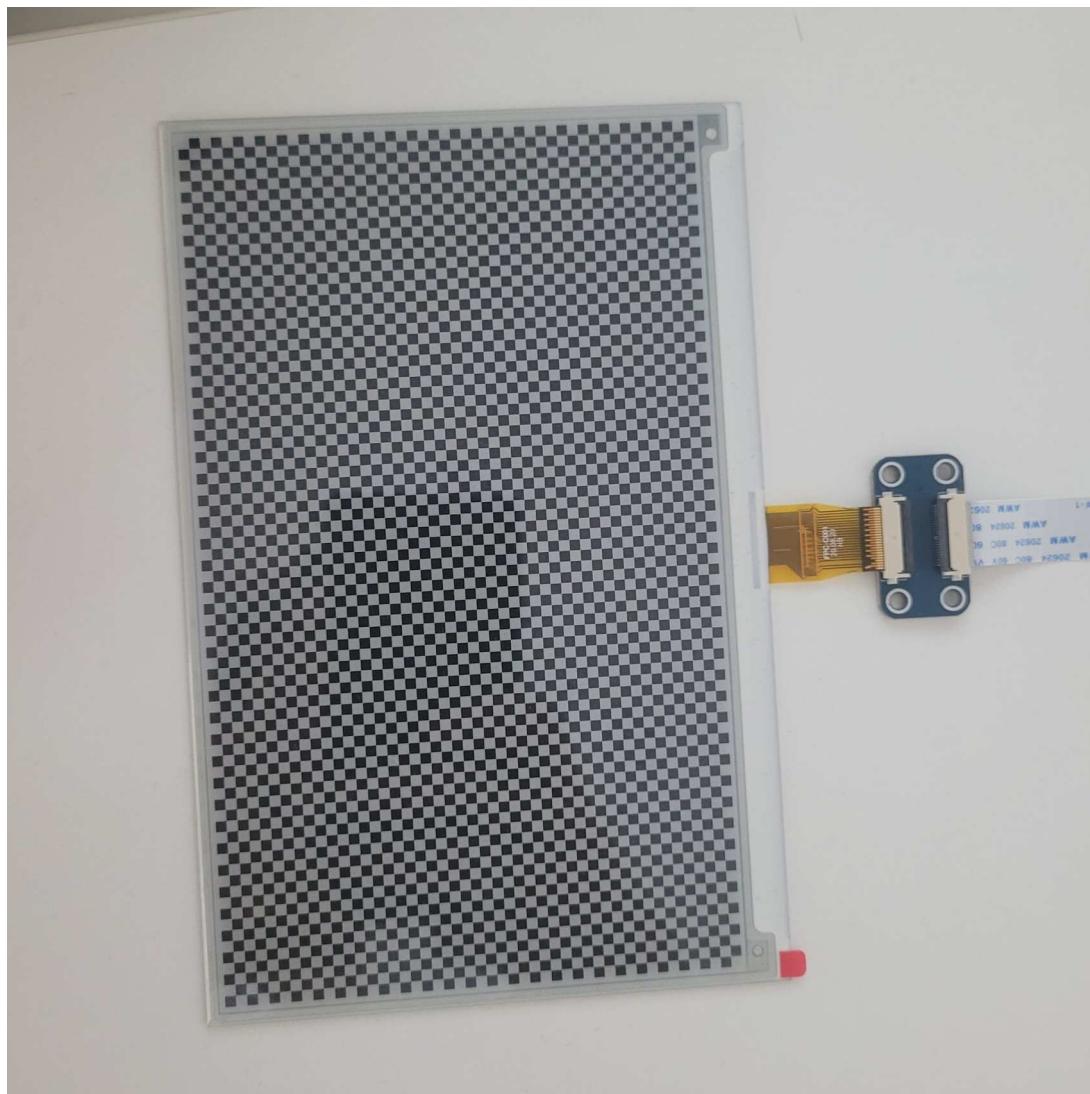


Abbildung 5.10: Rekursiver Aufruf

```
1  for y in range(h):  
2      for x in range(w):  
3          if (x // 10) % 2 == (y // 10) % 2:  
4              fb.pixel(x, y, 0)
```

Code snippet 5.7: Ein KaroMuster erzeugt aus einem Rekursiven Aufruf

### Offline - Variante: OfflineTemplate

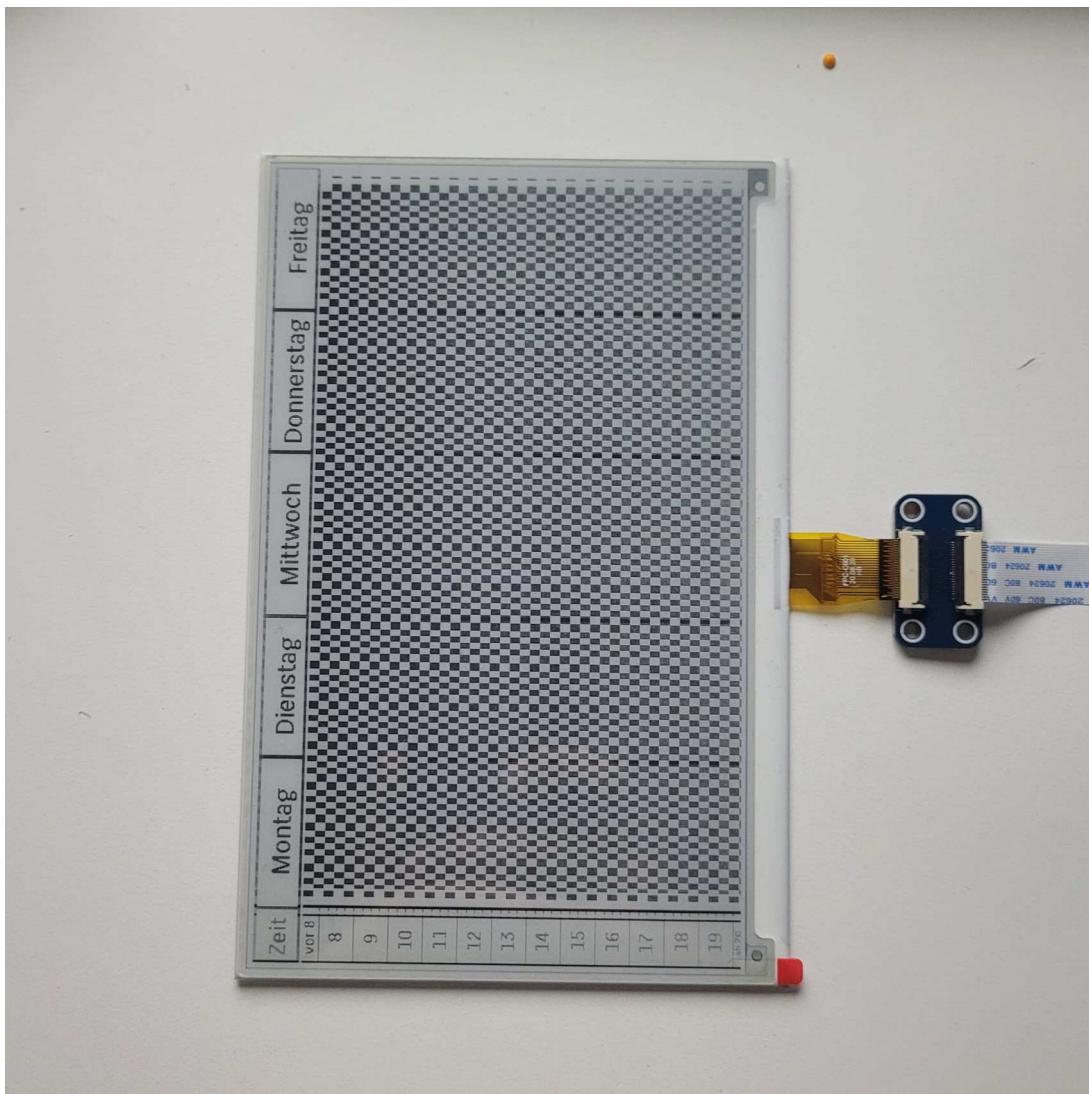


Abbildung 5.11: Hier werden die Zeile-Tage und die Spalte-Zeiten aus einer ByteArray-List.py geladen, der Content wird rekursiv gezeichnet

```
1 obere_zeile = bytearray([0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
2 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
3 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
4 0x00, 0x00, ...]
```

Code snippet 5.8: Byte-Array für die Zeile-Tage

```
1 ...
2
3 from obere_zeile.blit1_invertiert_HTWFont import obere_zeile as
  blit1 # byteArray Mo-Fr
```

```
4
5      from linke_spalte.blit2_invertiert_HTWFont import linke_spalte
6          as blit2 # byteArray 8-20 Uhr
7
8      ...
9
10     def drawTageBlit1():
11         x.blit1 = 0
12         y.blit1 = 0
13         bitmap_data.blit1 = blit1 #import der Lehrtage Zeile
14
15         framebuffer_to_show.blit1 = framebuffer.FrameBuffer(
16             bitmap_data.blit1,
17             801,
18             46,
19             framebuffer.MONO_HLSB)
20
21         main_fb.blit(framebuffer_to_show.blit1, x.blit1, y.blit1)
22         ...
23
24     def drawZeitenBlit2():
25         x.blit2 = 0
26         y.blit2 = 48
27
28         bitmap_data.blit2 = blit2 # import der Lehrzeiten Spalte
29
30         framebuffer_to_show.blit2 = framebuffer.FrameBuffer(
31             bitmap_data.blit2,
32             54,
33             434,
34             framebuffer.MONO_HLSB)
35
36         main_fb.blit(framebuffer_to_show.blit2, x.blit2, y.blit2)
37         ...
38
39     def draw_content(w, h, e):
40         gc.collect()
41         blit3_buffer = bytearray(w * h // 8)
42         sleep_ms(200)
43         framebuffer_to_show.blit3 = framebuffer.FrameBuffer(
44             blit3_buffer,
```

```
41             w,
42             h,
43             framebuffer.MONO_HLSB)
44     framebuffer_to_show.blit3.fill(weiss)
45     x.blit3 = 66
46     y.blit3 = 48
47     for y in range(h):
48         for x in range(w):
49             if (x // 10) % 2 == (y // 10) % 2:
50                 framebuffer_to_show.blit3.pixel(x, y, schwarz)
51
52     main_fb.blit(framebuffer_to_show.blit3, x.blit3, y.blit3)
53
54     ...
55     e.display_frame(main_buffer)
```

Code snippet 5.9: Display-Code: showTable.py welcher Zeile-Tage und Spalte-Zeiten aus einer File.py lädt

**Variante die alles Zeichnet im HTW-Font**

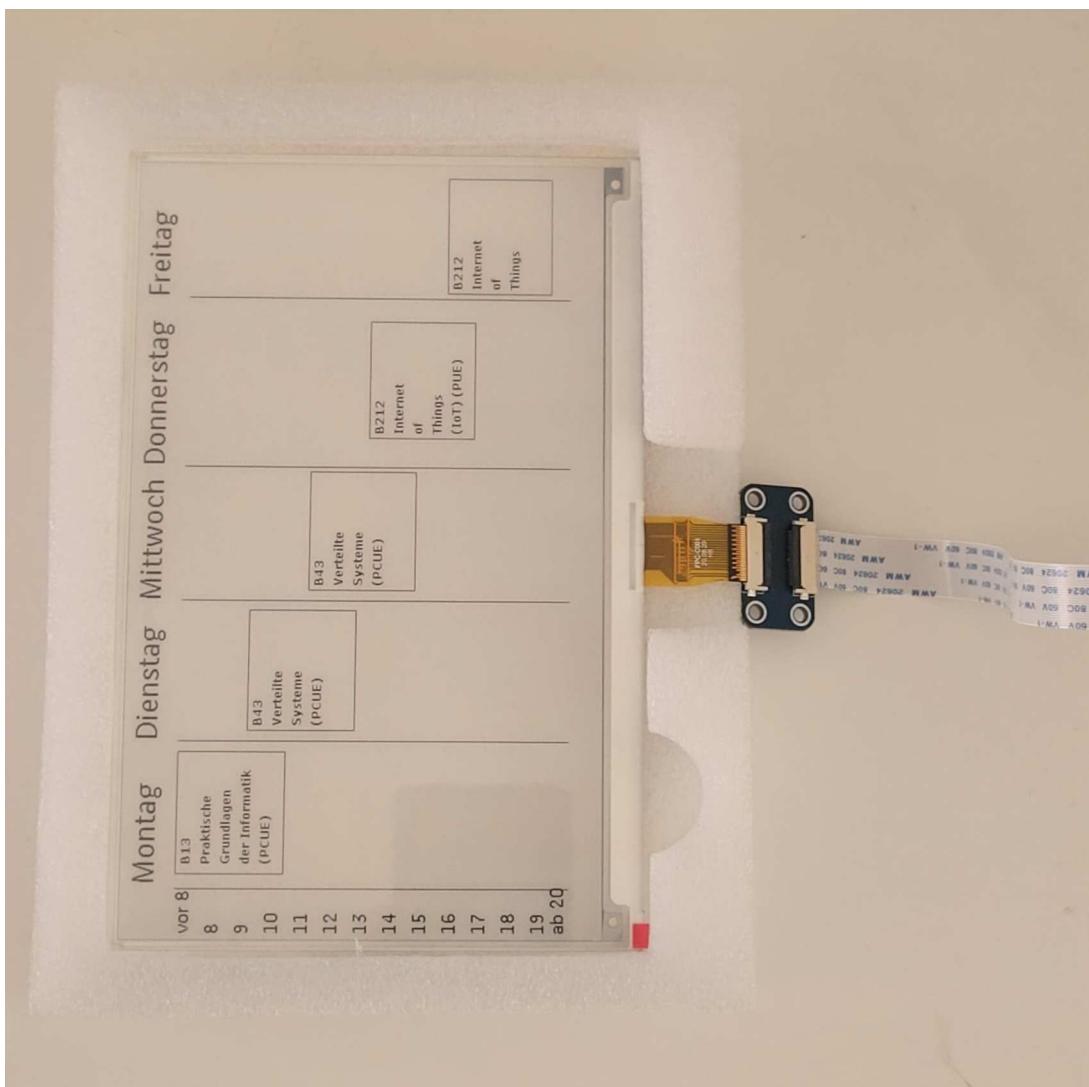


Abbildung 5.12: Hier wird alles mithilfe eines Writer mit dem HTW-Font und der framebuffer Klasse ein Raumbelegungsplan gezeichnet

```

1   ...
2   def draw_line_content():
3   ...
4   #####
5   ##### Donnerstag#####
6   #####
7   main_fb.rect(511, 240, 125, 110, schwarz, False) #Donnerstag
8   modulname3 = 'B212 Internet of Things (IoT) (PUE)'
9   modulname3_0='B212'
10  modulname3_1='Internet'
11  modulname3_2='of'
```

```

12     modulname3_3='Things'
13     modulname3_4='(IoT) (PUE)'
14     display_text(modulname3_0,245,516)
15     display_text(modulname3_1,265,516)
16     display_text(modulname3_2,285,516)
17     display_text(modulname3_3,305,516)
18     display_text(modulname3_4,325,516)
19 ...
20 def alles():
21     main_fb.fill(weiss)
22     draw_line_content()
23     e.display_frame(main_buffer)
24
25 if __name__ == '__main__':
26     main_fb.fill(weiss)
27     draw_line_content()

```

Code snippet 5.10: Byte-Array für die Zeile-Tage

### 5.2.3 Node-Red Flow

Der Node-Red Flow verbindet alle Komponenten des Systems miteinander.

Zu den **Komponenten** gehören *Telegram* mit dem *EDisplayBot*, *MQTT* und die *Topics* auf denen gelauscht oder Veröffentlicht werden kann. Und Schlussendlich auch in die *Influx* Datenpersistenz eingefügt, welche *Grafana* dann zur Visualisierung der Messdaten für Strom und Spannung nutzt um die Leistung auszurechnen und nach der Zeit abzubilden.

**Infodisplay Flow** Dazu besteht der Infodisplay aus Fünf kleineren Flows aus denen einer sich noch in der Entwicklung befindet.

Der sich in der Entwicklung befindliche Teilflow ist:

- Teilflow zur Verarbeitung eines Ankommenden Fotos auf dem Display

Zurzeit speichert der Telegram Photo Receiver anhand der URL des Fotos auf dem Telegramserver, die Datei lokal.

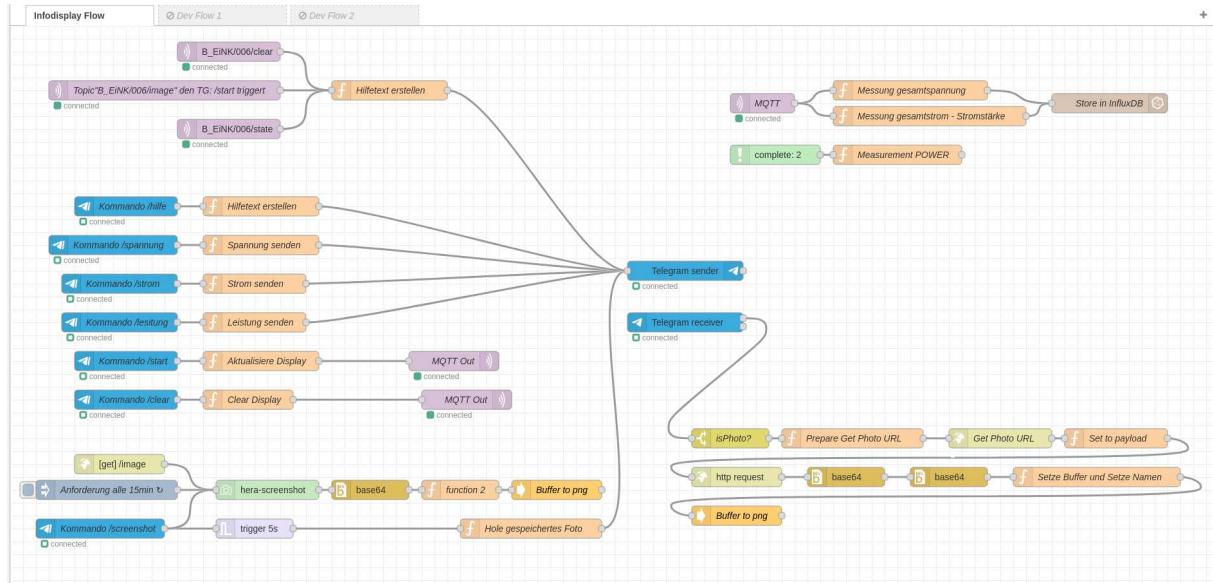


Abbildung 5.13: Der Infodisplay Node-Red Flow im Gesamtbild

In Abbildung 5.13 ist der Gesamte Flow einmal sichtbar. Zunächst wird der Telegram Bot beschrieben und im weiteren Verlauf wird auf alle Teilflows eingegangen. **Telegram**

Der Telegram Bot: EDisplayBot reagiert beim Betreten des Chat mit `/start` mit einer Begrüßung des Nutzers basierend auf seinem angegebenen Vornamen, bezogen aus dem Originale Nachrichten Objekt, aus der darunterliegenden `node-telegram-bot-api lib`.

```
1 |     flow.set('username', msg.originalMessage.from.first_name)
```

Code snippet 5.11: Codezeilen aus dem Funktions Block „Aktualisiere Display“ nach Telegram Kommando Block für `/start`



Abbildung 5.14: Der TelegramBot des Infodisplay: EdisplayBot ( EdisplayBot)

*BotCMD: /start oder /clear*

In diesem Teilflow werden die Kommandos:

1. /hilfe
2. /spannung
3. /Strom
4. /leistung

für den Telegram Sender als Antwort aufbereitet und versendet. Falls notwendig werden Flow variablen gesetzt oder abgerufen.

Die Display Kommandos publishen mit

- /start unter dem Topic: „B\_EiNK/00/image „,
- und /clear unter dem Topic: „B\_EiNK/00/clear „

Durch die Auslagerung des Telegram Bot vom ESP32 auf Node-RED. Unter Nutzung der Drahtlos Kommunikation über MQTT, wird der Flashspeicher entlastet. Somit ist ein Stabiler Codedurchlauf gewährleistet. Zudem sinkt damit auch die Stromauslastung des ESP32-Mikrocontroller. Da dieser weniger Prozessorleistung und Peripherie benötigt und so im Ultra-Low-Power Mode arbeiten kann.

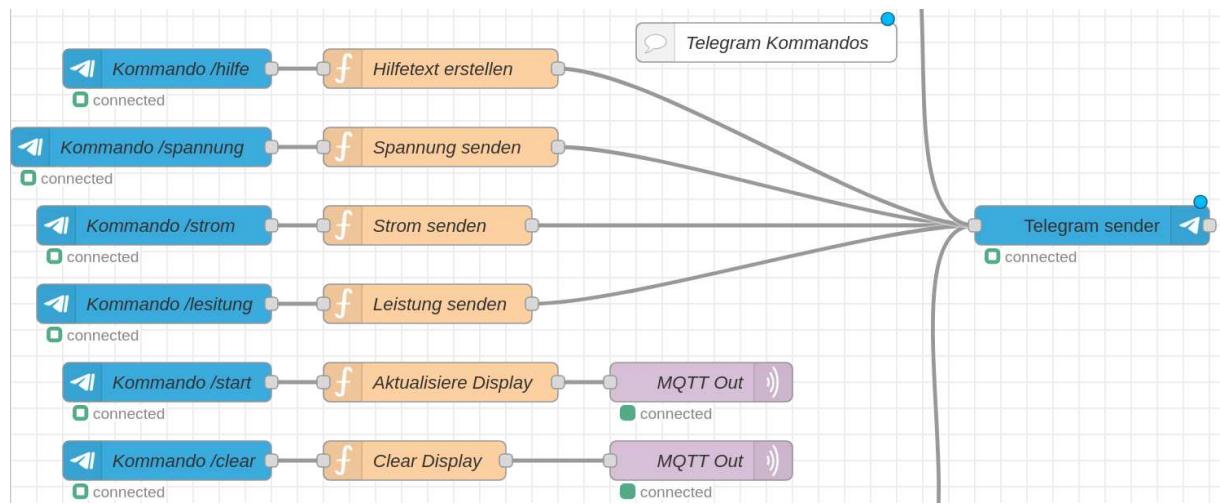


Abbildung 5.15: Bot-Kommandos

*BotCMD: /screenshot*

Zurzeit sendet das Telegramkommando /screenshot ein hera-screenshot des Monitors der lokalen Maschine als Antwort auf das Telegram-Kommando: **/screenshot**

Unter der Bedingung das Grafana auf dem Monitor geöffnet ist, ist es möglich ein Bild von Grafana mit dem Befehl /screenshot zu bekommen.

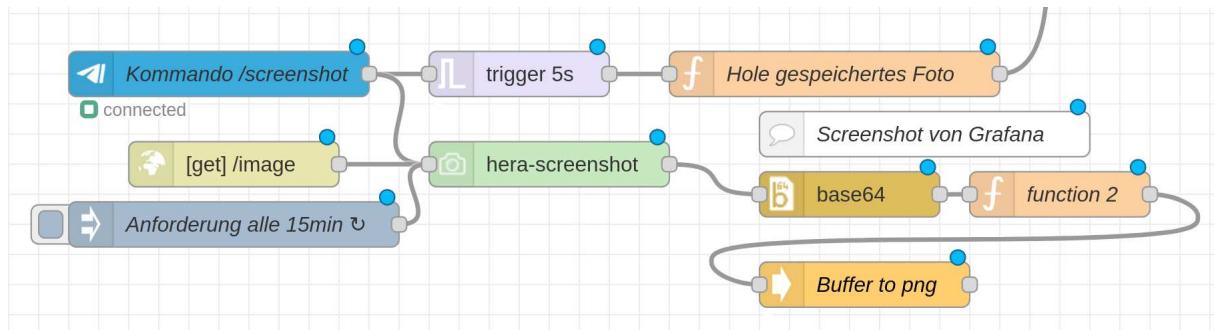


Abbildung 5.16: Screenshot von Grafana zurückschicken an User

*BotCMD: empfängt Photo von User*

Zurzeit empfängt das der EDisplayBot Bilder vom Nutzer und speichert diese Lokal ab. In Zukunft soll der Bot diese Bilder empfangen und verarbeitet als Schwarz-Weiß und skaliert auf 800x400 Pixel soll das Bild als SingleBuffer an den Infodisplay versendet werden.

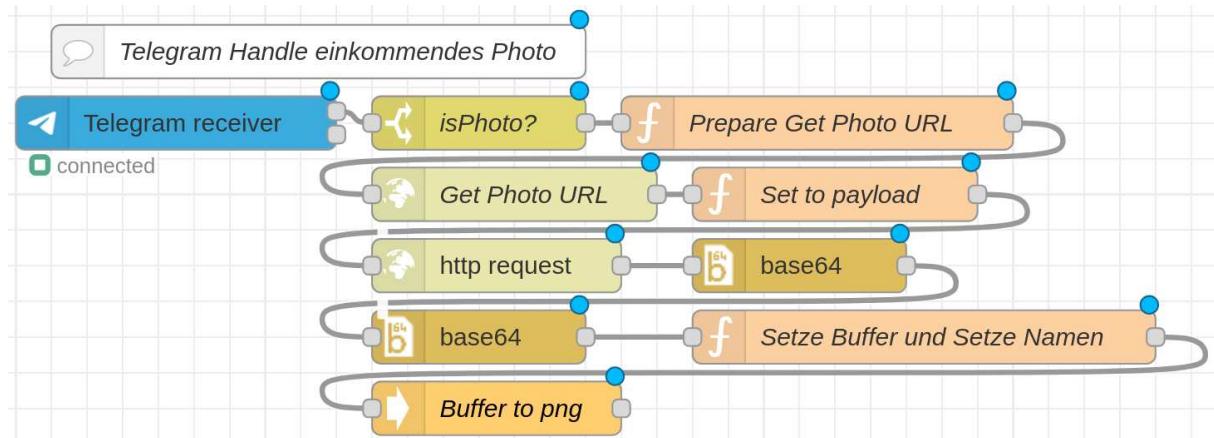


Abbildung 5.17: Der Infodisplay Node-Red Flow GesamtflowInfodisplaybild3

## MQTT

Mit MQTT auf dem Broker HiveMQ lässt sich das Infodisplay ansteuern und gleichzeitig überwachen. Die Strom und Spannungsmesswerte werden empfangen und in der Influx Datenbank gespeichert.

Der Complete-Funktionsblock: 2 triggert nur dann, den Funktion Block „Measurement Power“, wenn die beiden anderen Funktionblöcke „Messung gesamtspannung“ und „Messung gesamtstrom - Stromstärke“ ihre Verarbeitung der Nachricht abgeschlossen haben.

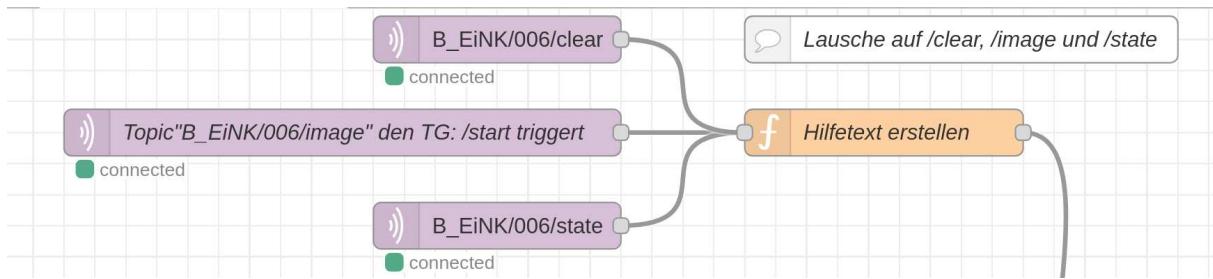


Abbildung 5.18: mqqt

```

1 const metric = "gesamLeistung"
2
3 const gesamtstrom = flow.get('gesamtstrom')
4 const gesamtSpannung = flow.get('gesamtspannung')
5
6
7 var gesamLeistung = gesamtstrom * gesamtSpannung
8
9 flow.set("gesamtLeistung", gesamLeistung)
10
11 msg.payload = {
12     [metric]: gesamLeistung
13 };
14 msg.measurement = "Gesamtleistung1";
15 return msg;
  
```

Code snippet 5.12: Codezeilen aus dem Funktions Block „Measurement POWER“ nach dem Complete Block zur Strom und Spannungsmessung

**Influx** Influx wird zur Datenpersistierung der Messdaten genutzt.

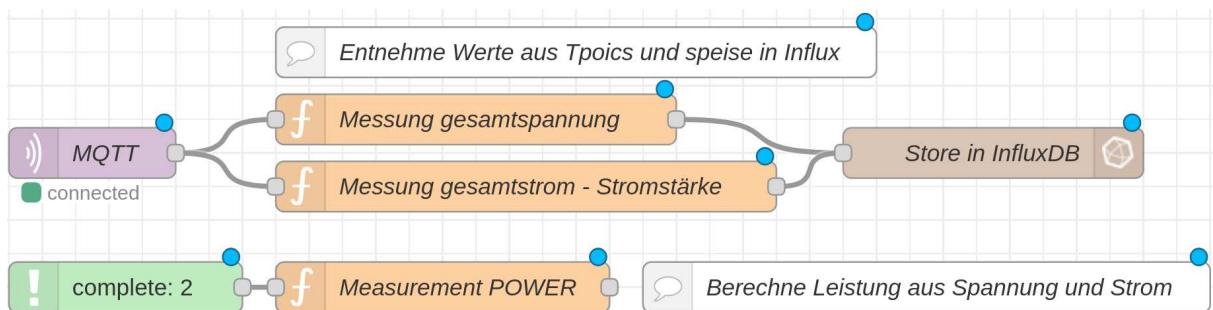


Abbildung 5.19: mqqt

### 5.2.4 Sensor

```

1      from umqttsimple import MQTTClient
2      import machine
3      import time
4      ...
5      from currentButterflyV7 import Current #STROMMESSUNG
6      from config import STROM_TOPIC as strom_topic #STROMMESSUNG
7      from boot import current_sensor # in der boot einmalig
8          Kalibrierung
9      ...
10     from voltageButterflyV3 import Voltage #SPANNUNGSMESSUNG
11     from config import SPANNUNG_TOPIC as spannung_topic
12         #SPANNUNGSMESSUNG
13     ...

```

Code snippet 5.13: Beide Sensoren nutzen MQTT. Sie unterscheiden sich nur in Ihrem Topic und in Ihrer spezifischen Klasse

#### Spannungssensor

```

1      from machine import ADC, Pin
2      import time
3
4      class Voltage:
5          def __init__(self):
6              adc_pin = 33
7
8              infoLED = Pin(2, Pin.OUT)
9              infoLED.on()
10
11             # Umrechnung ADC-Wert in Spannung
12             def no_conf_adc_to_voltage(self):
13
14                 infoLED = Pin(2, Pin.OUT)
15                 infoLED.on()
16
17                 adc_value = self.adc.read() # ADC-Wert lesen
18
19                 # Maximale Spannung = 3.6V (aufgrund der 11dB Attenuation)

```

```
20         max_voltage = 3.6
21
22     # Maximale ADC-Auflösung (12 Bit) = 4095
23     max_adc = 4095
24     voltage = (adc_value / max_adc) * max_voltage
25
26     print("ADC Wert:", adc_value, "Spannung:", voltage, "V")
27     time.sleep(1) # 1 Sekunde warten
28     infoLED.off()
29     return voltage
```

Code snippet 5.14: Byte-Array für die Zeile-Tage

### Stromstärke Sensor

```
1  from machine import ADC, Pin
2  import time
3
4  class Current:
5      def __init__(self):
6          self.current_s = ADC(Pin(33))
7
8          infoLED = Pin(2, Pin.OUT)
9          infoLED.on()
10         ...
11         ...
12      def strommessung(self):
13          infoLED = Pin(2, Pin.OUT)
14          infoLED.on()
15
16          #Anzahl der Messwerte
17          anzahl_samples = 150
18
19          #Summe der Messwerte vom ADC PIN 33
20          summe_aus_adc_werten = 0
21
22          # Sammeln der ADC-Werte
23          for _ in range(anzahl_samples):
24              aktueller_adc_wert = self.current_s.read() # ADC-Wert
25              lesen
```

```
25         summe_aus_adc_werten += aktueller_adc_wert
26         time.sleep(0.003) # 3 ms warten
27
28
29         durchschnitt_adc_value = summe_aus_adc_werten /
30             anzahl_samples
31
32         adc_spannung = (durchschnitt_adc_value /
33             self.max_adc_value) * self.max_voltage
34
35         # Spannungsteiler-Korrektur
36         aktuelleSpannung = adc_spannung * (self.R1 + self.R2) /
37             self.R2
38
39         # Umrechnung des gemessenen Spannungssignals in Strom
40         strom = (aktuelleSpannung - self.zero_point_voltage) /
41             self.sensitivity
42         print("Calculated Current: {:.3f} A".format(strom))

43
44         time.sleep(0.05) # 50 ms warten
45         infoLED.off()
46
47         return strom
```

Code snippet 5.15: Berechnung des Stromwertes

### 5.2.5 Grafana Dashboard

Grafana Version 11.1.1 ist Voraussetzung, da das MQTT-Plugin nur in dieser Version arbeitet

# Kapitel 6

## Test

In diesem Kapitel werden alle Hardware und Softwarebestandteile des Infodisplay getestet. Die Tests wie auch der Produktivcode sind im GitLab Repository der HTW-Berlin zu der Bachelorarbeit zum „Telegram controlled ultra low power infodisplay“ unter der Url des Rechenzentrum<sup>1</sup> zu finden.

### 6.1 Testplanung

Die Testplanung sieht vor, das **Manuelle Testfälle**, welche ein eingreifen einer Testperson benötigen, wie auch **Automatisierte Testfälle** die ohne einen Eingriff ablaufen, in drei weitere Gruppen unterteilt werden.

In **Unit-Tests**, Testfälle die einzelne Komponenten testen. **Component Tests**, Funktionale Tests zur Sicherstellung das Funktionen von Modulen korrekt ausgeführt werden. Und **End to End Tests**, Tests die Sicherstellen das Infodisplay Anwendungsfälle von Anfang bis Ende richtig ablaufen.

Die Ordnerstruktur für die Testplanung sieht dementsprechend folgendermaßen so aus, wie in Abbildung 6.1 zu sehen.

Diese Tests können falls notwendig, weiter Unterteilt werden in **Langzeit- und Kurzeit-tests** je nach Anwendungsfall.

---

<sup>1</sup>Die Url lautet: <https://gitlab.rz.htw-berlin.de/s0551006/ultralowpowerinfodisplay>

- (a) Die Ordnerstruktur für die Automatisierten Testfälle

```
[harun-pc:Automatiserte_Test_Faelle] on dev!
[hdi10]% tree -al 2
.
├── Grafana_mit_logs_MQTT_CAM-Telegram_Node_REPL
│   ├── Infodisplay
│   │   ├── Component-Tests
│   │   ├── E2E-Tests
│   │   └── Unit_Tests
│   ├── Node-Red
│   │   ├── Component-Tests
│   │   ├── E2E-Tests
│   │   └── Unit_Tests
│   ├── SensorMessung
│   │   ├── Component-Tests
│   │   ├── E2E-Tests
│   │   └── Unit_Tests
│   └── Zu_testender_code
│       ├── Infodisplay_framebuf_Tabelle_Refresh_3_v
│       │   └── on_Modulen_Testdauer_5Min
│       ├── Offline_Infodisplay_Run_Example_refresh_
│       │   └── byteArrayImages
│       │       ├── RAW_Abgabe_Spannungssensor
│       │       └── RAW_Abgabe_Stromsensor
│       └── UnterKapitel_Display_Code_sources
└── Unterkapitel_Display_Code_sources

19 directories, 0 files
[harun-pc:Automatiserte_Test_Faelle] on dev!
[hdi10]%
```

- (b) Die Ordnerstruktur der Manuellen Testfälle

```
[harun-pc:Manuelle_Test_Faelle] on dev!
[hdi10]% tree -al 2
.
├── Infodisplay
│   ├── Component-Tests
│   ├── E2E-Tests
│   └── Unit_Tests
├── Messumgebung
│   ├── Component-Tests
│   ├── E2E-Tests
│   └── Unit_Tests
└── SensorMessung
    ├── Component-Tests
    ├── E2E-Tests
    └── Unit_Tests
└── Zu_testender_code
    ├── Infodisplay_framebuf_Tabelle_Refresh_3_v
    │   └── on_Modulen_Testdauer_5Min
    ├── Offline_Infodisplay_Run_Example_refresh_
    │   └── byteArrayImages
    │       ├── RAW_Abgabe_Spannungssensor
    │       └── RAW_Abgabe_Stromsensor
    └── UnterKapitel_Display_Code_sources

18 directories, 0 files
[harun-pc:Manuelle_Test_Faelle] on dev!
[hdi10]%
```

Abbildung 6.1: Die Ordner der Manuellen und Automatisierten Testfälle

Für die Anforderung: 3.1.7 des Fachbereich 4

Wird ein Testfall entwickelt.

Die Hauptkriterien in dieser Abschlussarbeit werden das Bestehen und die Stabilität der Testfälle sein.

Im MicroPython Forum Findet sich ein Eintrag in dem ein User namens djantzen von dem Problem spricht unter einer Ubuntu Entwicklungsumgebung das **machine module** für die *UnitTest* zu Nutzen. So wurden die Klasse machine.py entwickelt um vom *module machine* zu erben und mit diese mit einem PinEvent und einer StateTrackable zu spezialisieren.[dja22]

Das Repo ist Strukturiert nach drei Ordnern. Einem workflows, einem src und einem tests Ordner.

Der **CI-Workflow** startet Tests beim *push* und *pull* auf dem *main-branch*.

Das **Shellscript** *install-micropython.sh*, zur Installation der Testumgebung für die Testfälle in **micropython**.

Die **machine.py** ist Hilfsklasse für die Tests und **run\_tests.py** startet alle Tests.

Mit der Hilfsklasse lassen sich die Testverfahren überschaubarer gestalten. Nun kann auf Pin, PWM und ADC zustände getestet werden..

Um die Testumgebung:

Die Lokale Maschine die eine für den Unix-Port kompilierte Version von der MicroPython-Firmware ausführt zu erzeugen.

Führen wir eine leicht dedizierte Version des *install-micropython.sh* Shellscripts aus.

**Statt diesen Codezeilen:**

```
1 [caption={Mit diesen Codezeilen wird uasyncio mitinstalliert, führt  
allerdings zu fehlern bei der Installation}, captionpos=b,  
label={lst:mit_uasyncio}]  
2 #!/usr/bin/env bash  
3  
4 cd /tmp  
5 git clone https://github.com/micropython/micropython.git  
6 cd micropython/mpy-cross  
7 make  
8  
9 cd ../ports/unix  
10  
11 # Default unix build does not include uasyncio:  
12     https://github.com/micropython/micropython/issues/6109  
13 echo "  
14 freeze(  
15     \"$(MPY_DIR)/extmod\",  
16     ("uasyncio/_init__.py",  
17     "uasyncio/core.py",  
18     "uasyncio/event.py",  
19     "uasyncio/funcs.py",  
20     "uasyncio/lock.py",  
21     "uasyncio/stream.py",  
22     "uasyncio/task.py",  
23 ),  
24     opt=3,  
25 )
```

```

26 " >> variants/manifest.py
27 make submodules
28 make
29 sudo cp micropython /usr/local/bin

```

Nur folgende Zeilen verwenden:

```

1 [caption={In dieser Form ist eine schnelle Aufsetzen der
2   Micropython- Testumgebung gewährleistet}, captionpos=b,
3   label={lst:ohne_uasycio}]
4 #!/usr/bin/env bash
5
6   cd /tmp
7   git clone https://github.com/micropython/micropython.git
8   cd micropython/mpy-cross
9   make
10
11  cd ../ports/unix
12
13  make submodules
14  make
15  sudo cp micropython /usr/local/bin

```

Somit können wir die zusätzliche Installation von uasyncio als als *gefreesetes micropython-module*[Dc24] umgehen. Die notwendigen Pakte können später mit **mip** installiert werden. Innerhalb der interaktiven **micropython** Sitzung mit

```
1 exec(open('scripts/the_script.py').read()).
```

oder außerhalb mit

```
1 micropython run_tests.py
```

Mit den Oberen Zeilen 3 bis 6 aus wird das *GitHub-repository* von *MicroPython* nach **temporary(/tmp) gecloned**, und mit dem *MicroPython-Cross-Compiler* mit *make* für ein Unix kompiliert[mic24].

Eine Ausführbare Firmware *Build* befindet sich unter: '/Firmware/forUnix' im GitLab Repository<sup>2</sup>.

### run\_test.py

```
1 #!/usr/bin/env micropython
2
3 import unittest
4 import sys
5
6 sys.path.append("src")
7
8
9 unittest.main("tests")
```

Die run\_tests.py nutzt die Kompilierte Entwicklungsumgebung für MicroPython. Indem es zu beginn der Datei nach dem *Shebang* die Entwicklungsumgebung **micropython** nutzt und den 'src' sowie 'tests' folder in die Testumgebung einbindet.

## 6.2 Software Testumgebung

Mit Ende der Testplanung ist eine Software-Testumgebung entstanden. Diese Testumgebung besteht aus einem Order 'dieZuTestendeKlasse' und dem Ordner 'hilfsklassen' im 'tests'-folder des rootverzeichnis.

---

<sup>2</sup><https://gitlab.rz.htw-berlin.de/s0551006/ultralowpowerinfodisplay>

## 6.3 Testdurchführung

### Infodisplay Softwaretest - automatisiert

Die Python-Datei **simpler-test.py** führt mit

```
1 |     micropython simpler_test.py
```

Tests zur Instanziierung der SPI-Klasse, des Epaper-Driver<sup>3</sup> als Mock-Klasse, und Tests zur Pin-Wert Änderung durch.

Die Softwaretests zu Pin-Zuständen und Pin-Werten des Infodisplay sind Repository dieser Abschlussarbeit im Ordner 'tests' zu finden.

Der Beginn einer Testsuite 'simpler\_test.py' importiert neben dem unittest -Modul die Hilfsklassen MyTestRunner und das ("von **machine** spezialisierten") **machine-Modul**, mit den Hilfsklassen für die IO-Tests.

```
1 |     import unittest
2 |     from hilfsklasse.runner_test import MyTestRunner
3 |     from hilfsklasse import machine
4 |     from dieZuTestendeKlasse import mock_display
```

Code snippet 6.1: Importzeilen

Die Imports dieses Test: 'simpler\_test.py' sind können dem Anhang entnommen.

Zu der Assistenz Test-Klasse von **machine** schreibt der Autor??:

„PinEvent is a simple container of old and new values. StateTrackable is a base class that records a series of state changes described as PinEvents. Pin, PWM and ADC are mocks that extend StateTrackable.“

Um eine lesbare Ausgabe trotz Modularen Nutzen von Teilen dieser Bibliothek zu Erzeugen, wurde ein Test-Runner entwickelt. Dieser wurde auch während der Entwicklungsphase des Projektes auf Kompilierbarkeit und laufende Tests betrieben.

```
1 |     import unittest
2 |
3 |     class MyTestRunner(unittest.TextTestRunner):
4 |         def __init__(self, *args, **kwargs):
```

<sup>3</sup>e7in5k.py - den 7in5 Display Driver

```

5     super(MyTestRunner, self).__init__(*args, **kwargs)
6
7     def run(self, test):
8         """ Test Logik vor und nach dem Testlauf """
9         print('### Beginn des Testlaufs ###')
10
11    result = super(MyTestRunner, self).run(test)
12
13    print('### Ende des Testlaufs ###')
14    return result

```

Code snippet 6.2: Importzeilen

Der Test-Runner 'MyTestRunner' nutzt das modul unittest um eine MyTestRunner-Klasse zu erzeugen, die es erlaubt Logik vor und nach den Testfällen auszugeben.

Dieser kann einfach in allen weiteren Test importiert und als if \_\_name\_\_ == '\_\_main\_\_': unittest.main(testRunner=MyTestRunner) mitgegeben werden.<sup>4</sup>

```

1     from hilfsklasse.runner_test import MyTestRunner
2     ...
3
4     if __name__ == '__main__':
5         unittest.main(testRunner=MyTestRunner)

```

Code snippet 6.3: Test-Runner in **simpler-test.py**

In 6.3 ist das Code-Listing für den TestRunner aufgeführt, dieser wird an das Ende und den Anfang der 'simple-test.py' eingefügt wurde.

### **Strom- und Spannungsmessung Infodisplay mit Multimeter - Hardwaretest Manuell**

Um zu Testen was der Reale Verbrauch des Infodisplay in Betrieb und beim Display Refresh ist, werden Messungen am Infodisplay mit einem Multimeter gemacht. Die Spannung wird an den Pins 3V3 und GND parallel und der Strom wird zwischen OUT+ und S1. S1 muss in der Schalterstellung „EIN“ gestellt sein um eine Strommessung durchzuführen.

---

<sup>4</sup>Eine der „special variables“ von Python, In Python-Foren Diskutiert[Mar24a].Ein Beitrag auf [Dev10] von 2010, aktualisiert am 23.08.2024

In Abbildung 5.5 sieht man die Messschaltung mit den Sensoren. Dieselben Messpunkte werden auch bei der Messung mit dem Multimeter genutzt.

Hier sollten Multimeter Messungen des Infodisplay beim Display-Refresh folgen. Aber unglücklicherweise ist kurz vor Abgabetermin der ESP32 kaputtgegangen.

Deswegen wird im weiteren auf die Messwerte des vergangenen Zeitraum zurückgegriffen. Dies betrifft den Zeitraum: „Anfang Juli bis 23 Aug“ — **Strom und Spannungsmessung am Infodisplay mit Sensormessung Hardware automatisiert -> messumgebung**

Die Schaltung aus der Abbildung 5.5 Gesamte Messschaltung wird was Messumgebung definiert.

Der Test ist eine Ausgabe der gemessenen und veröffentlichten Strom- oder Spannungswerte.

Abbildung 6.2: Topics des Infodisplay

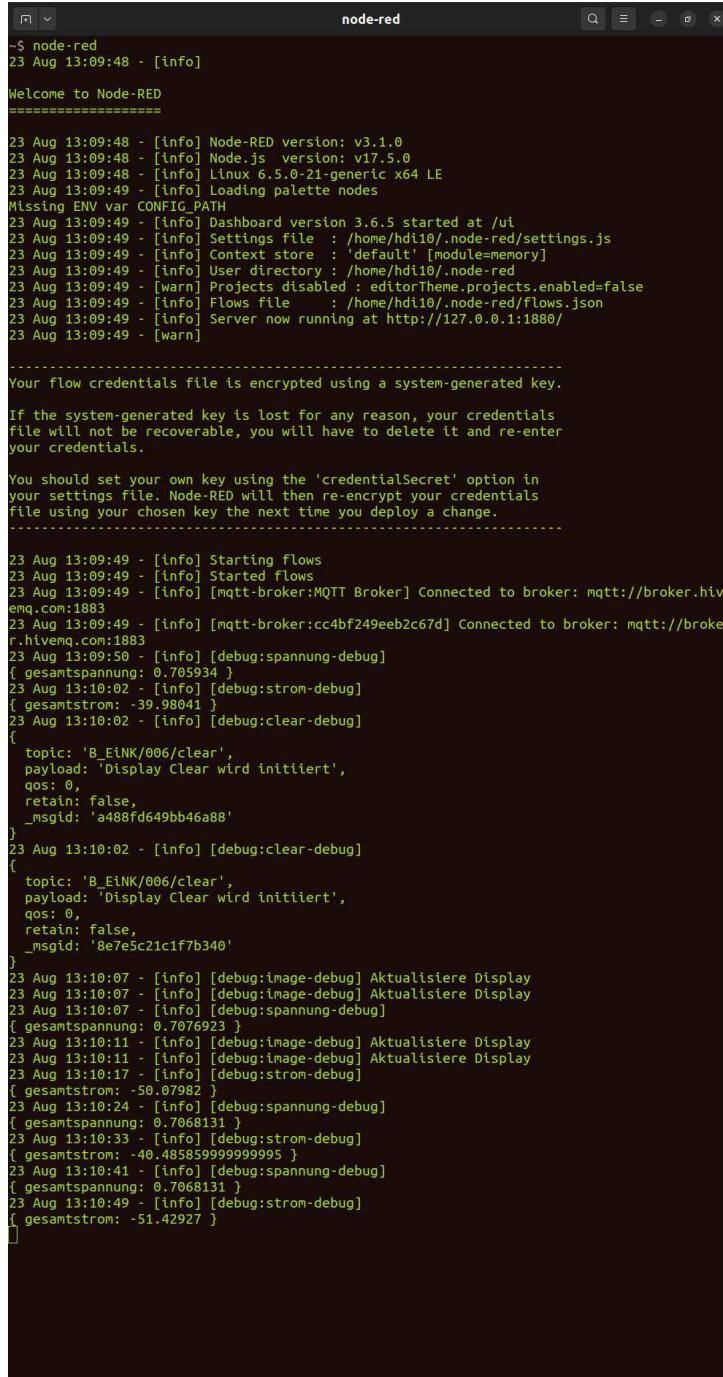
Mit Mosquitto - Befehl:

```
mosquitto_sub -h broker.hivemq.com -p 1883 -t B_EiNK/006/gesamtstrom
```

lässt sich wie auch die anderen Topic: „B\_EiNK/006/gesamtstrom“ oder „B\_EiNK/006/sta-te“ auf dem „Free Public MQTT Broker by HiveMQ“ abonnieren. In Abbildung 6.2



## Node-Red Debug Logs im Terminal und an der Node - Softwaretest manuell



```

$ node-red
23 Aug 13:09:48 - [info] 
Welcome to Node-RED
=====
23 Aug 13:09:48 - [info] Node-RED version: v3.1.0
23 Aug 13:09:48 - [info] Node.js version: v17.5.0
23 Aug 13:09:48 - [info] Linux 6.5.0-21-generic x64 LE
23 Aug 13:09:49 - [info] Loading palette nodes
Missing ENV var CONFIG_PATH
23 Aug 13:09:49 - [info] Dashboard version 3.6.5 started at /ui
23 Aug 13:09:49 - [info] Settings file : /home/hdi10/.node-red/settings.js
23 Aug 13:09:49 - [info] Context store : 'default' [module=memory]
23 Aug 13:09:49 - [info] User directory : /home/hdi10/.node-red
23 Aug 13:09:49 - [warn] Projects disabled: editorTheme.projects.enabled=false
23 Aug 13:09:49 - [info] Flows file : /home/hdi10/.node-red/flows.json
23 Aug 13:09:49 - [info] Server now running at http://127.0.0.1:1880/
23 Aug 13:09:49 - [warn] 

-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.

-----
23 Aug 13:09:49 - [info] Starting flows
23 Aug 13:09:49 - [info] Started Flows
23 Aug 13:09:49 - [info] [mqtt-broker:MQTT Broker] Connected to broker: mqtt://broker.hivemq.com:1883
23 Aug 13:09:49 - [info] [mqtt-broker:cc4bf249eeb2c67d] Connected to broker: mqtt://broker.hivemq.com:1883
23 Aug 13:09:50 - [info] [debug:spannung-debug]
{ gesamtspannung: 0.705934 }
23 Aug 13:10:02 - [info] [debug:strom-debug]
{ gesamtstrom: -39.98041 }
23 Aug 13:10:02 - [info] [debug:clear-debug]
{
  topic: 'B_EINK/006/clear',
  payload: 'Display Clear wird initialert',
  qos: 0,
  retain: false,
  _msgid: 'a488fd649bb46a88'
}
23 Aug 13:10:02 - [info] [debug:clear-debug]
{
  topic: 'B_EINK/006/clear',
  payload: 'Display Clear wird initialert',
  qos: 0,
  retain: false,
  _msgid: '8e7e5c21c1f7b340'
}
23 Aug 13:10:07 - [info] [debug:image-debug] Aktualisiere Display
23 Aug 13:10:07 - [info] [debug:image-debug] Aktualisiere Display
23 Aug 13:10:07 - [info] [debug:spannung-debug]
{ gesamtspannung: 0.7076923 }
23 Aug 13:10:11 - [info] [debug:image-debug] Aktualisiere Display
23 Aug 13:10:11 - [info] [debug:image-debug] Aktualisiere Display
23 Aug 13:10:17 - [info] [debug:strom-debug]
{ gesamtstrom: -50.07982 }
23 Aug 13:10:24 - [info] [debug:spannung-debug]
{ gesamtspannung: 0.7068131 }
23 Aug 13:10:33 - [info] [debug:strom-debug]
{ gesamtstrom: -40.485859999999995 }
23 Aug 13:10:41 - [info] [debug:spannung-debug]
{ gesamtspannung: 0.7068131 }
23 Aug 13:10:49 - [info] [debug:strom-debug]
{ gesamtstrom: -51.42927 }

```

Abbildung 6.3: Der System-Output von Node-Red im Terminal

In Node-Red kann der Debug-Flow des Infodisplay importiert werden. Dieser stellt zu der Ausgabe des node-status am Debug-Node noch eine system console Ausgabe. Diese ist der Abbildung 6.3

**Grafana- Logs - MQTT - Telegram - Softwaretest manuell** In Grafana lassen sich diese Veröffentlichten Messpunkte Abrufen und Visualisieren. Das Grafana Dashboard dazu befindet sich im Repository unter 'Grafana\_DashboardJSON', diese lassen sich in Grafana unter

1 | <http://localhost:3000/dashboards>

mit einem Klick auf den weißen Pfeil im blauen Kästchen und im aufklappenden Menü auf Import.

Im folgenden Bild sehen wie Grafana die Messwerte der Sensoren visuell und textuell darstellt. Visuell anhand der Messdaten aus der Influx Datenbank und Textuell auf Basis eingehender MQTT-Nachrichten auf den abonnierten Topics.



Abbildung 6.4: Ankommende Messwerte als Nachricht über MQTT und als Datenpunkt aus der Influx visualisiert in Grafana

**Ein Kommando über Telegram zum Display Refresh Softwaretest manuell**

Dieser manuelle Softwaretest wurde aus zeitlichen Gründen auch mithilfe Mosquitto und Grafana durchgeführt.

**Ein Kommando über Telegram zum Dsiplay Clear Softwaretest manuell**

Dieser manuelle Softwaretest wurde aus Zeittechnischen Gründen auch mithilfe Mosquitto und Grafana durchgeführt.

## 6.4 Testergebnisse

Die Softwaretest und Hardwaretests wurden zur Ermittlung der Leistung und somit auch der Laufzeit des Infodisplay durchgeführt.

Nun werden die Test- und Messergebnisse mit dem Prototypen in der Evaluation weiterverfahren um zu einem Fazit zu gelangen.

## 6.5 Ausblick auf die Testlaufbahn des Infodisplay

Mithilfe der MicroPython Testumgebung ist nun ein kontinuierliches Testgetriebenes Entwickeln möglich. Solange Konventionen weiter eingehalten werden kann später vielleicht zu anderen Testframework gegriffen werden. Behavior-Driven Development (BDD) mit Cucumber<sup>5</sup>, Testfälle mit Jira Xray<sup>6</sup>, oder mit opentext ehemals microfocus uft<sup>7</sup>.

---

<sup>5</sup><https://cucumber.io/>

<sup>6</sup><https://www.atlassian.com/devops/testing-tutorials/jira-xray-integration-manage-test-cases>

<sup>7</sup><https://www.microfocus.com/documentation/silk-central/200/de/silkcentral-help-de/>

GUID-531809BA-688F-41D5-BDB2-FCE786A284CE.html

# Kapitel 7

## Evaluation

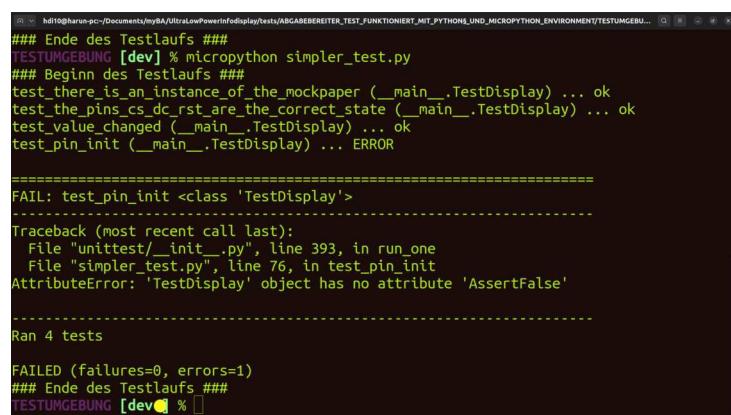
In der Evaluation werden die Anforderungen aus der Analyse mit den Ergebnissen aus der Konzeption, Implementierung und der Tests verglichen. Erfüllen diese Ergebnisse die Anforderungen des Fachbereich 4. Schlussendlich soll dann im Fazit die Frage Beantwortet werden:

„Lohnt sich der Anwendungsfall?“

### 7.1 MicroPython und Thonny

Die Anforderung: 3.1.1

In der Implementierung wurde MicroPython als Programmiersprache genutzt. Des weiteren wurde Thonny zum Flashen der MicroPython Firmware 1.23.0 und des selbst geschriebenen Code genutzt.



```
## Ende des Testlaufs ##
TESTUMGEBUNG [dev] % micropython simpler_test.py
## Beginn des Testlaufs ##
test_there_is_an_instance_of_the_mockpaper (_main_.TestDisplay) ... ok
test_the_pins_cs_dc_rst_are_the_correct_state (_main_.TestDisplay) ... ok
test_value_changed (_main_.TestDisplay) ... ok
test_pin_init (_main_.TestDisplay) ... ERROR

=====
FAIL: test_pin_init <class 'TestDisplay'>
-----
Traceback (most recent call last):
  File "unittest/_init_.py", line 393, in run_one
    File "simpler_test.py", line 76, in test_pin_init
      AttributeError: 'TestDisplay' object has no attribute 'AssertFalse'

Ran 4 tests

FAILED (failures=0, errors=1)
## Ende des Testlaufs ##
TESTUMGEBUNG [dev] %
```

Abbildung 7.1: Mit dem **Test-Runner** und der **Testumgebung** ist **Testgetriebene Entwicklung** möglich

In den Tests wurde die Testumgebung mithilfe der Portierung von micropython auf ein Unix-build portiert. In dieser Testumgebung wurde mit einem eigenen Test-Runner6.2 eine Ausgabe formatiert, wie in Abbildung 7.5 dargestellt. —

## 7.2 Ein- und Ausschalten

Die Anforderung: 3.1.2

Ein Ein- und Ausschalter wurde in Form eines mini-slide-switch angeschlossen zwischen dem 3V3 Pin des ePaper Esp32 Driver-Board, und dem Anschluss OUT+ des TP4056 Laderegler, wie in Abbildung 5.3(a) oder 5.3(b) zu entnehmen, angeschlossen an den Infodisplay.

S1 ist zwischen 3V3 und OUT+ des TP4056 angeschlossen, der Laderegler ist erst mit dem Akku verbunden.

Somit sollte das Infodisplay mit dem Schalter S1 Ein- und Ausschalten. Den Transport oder die Verwahrung des Infodisplay gewährleisten, ohne das die Batterielebensdauer stark beeinträchtigt wird.

## 7.3 Laufzeit

Die Anforderung: 3.1.3

In der Konzeption wurde zur Anforderung der Laufzeit4.3 festgestellt, dass

- **Mit einer Leistungsaufnahme von 80 mA** Liefert die Webseite des Digikey[TDK24] eine Batterielebensdauer von 0,122 Monaten. Dieses Ergebnis erfüllt die Anforderung nach der Laufzeit nicht.
- **Mit einer Leistungsaufnahme von 0,0065 mA** Liefert die Webseite des Digikey[TDK24] 1495,726 Monate, mit derselben Batteriekapazität.

In der Implementierung wurde Grafana zur Visuellen Auswertung der Elektrischen Kennwerte ausgewählt.

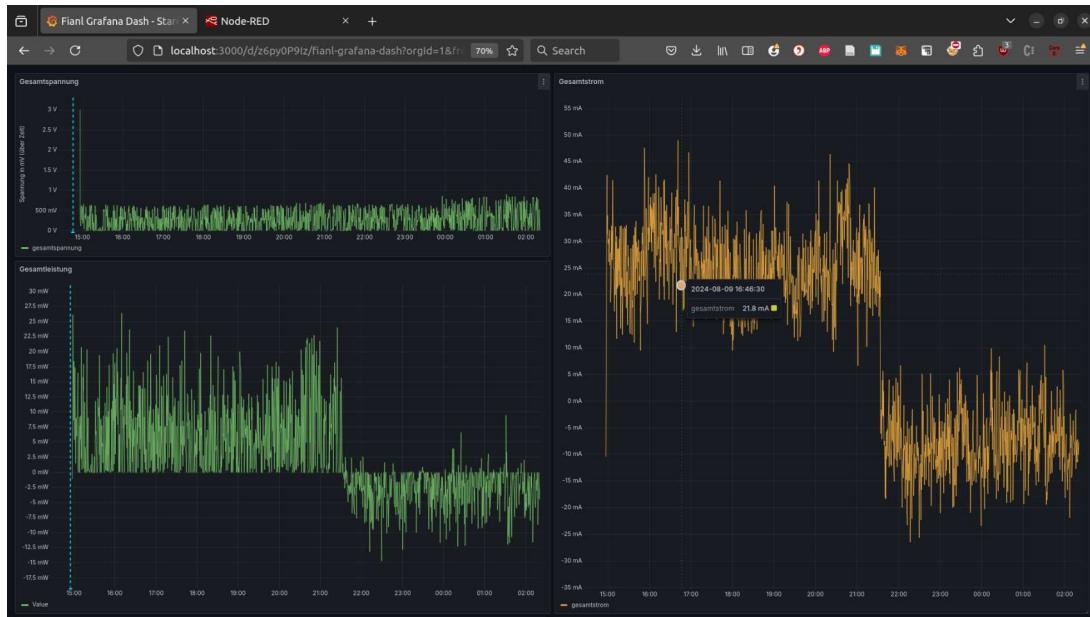


Abbildung 7.2: Die Messreihe vom 09.08.2024 von 15 bis 2 Uhr des Folgetages in Grafana

Die **Messreihe vom 09.08.2024** aus den *Datenreihen der Influx* liefert einen sehr guten Überblick über die **Betriebsspannung**, den **Betriebsstrom** und die **Leistung** an dem Tag. Diese Messreihe ist in Grafana visualisiert und ist in Abbildung 7.2 zu sehen.

Aus der Abbildung 7.2 ist zu entnehmen folgendes zu Entnehmen:

- Die **Betriebsspannung** liegt während des Tages bei durchschnittlich **250 mV**.
- Die **Betriebsstrom** bzw. die Leistungsaufnahme liegt während des Tages durchschnittlich **5-10 mA**.
- Die **Leistung** liegt während des Tages durchschnittlich **5 mW**.

Mit einer durchschnittlichen **Betriebsspannung** von **250 mV** und einem durchschnittlichen **Betriebsstrom** von **7,5 mA**. Ergibt das eine durchschnittliche Leistung von:

$$P = 250 \text{ mV} * 7,5 \text{ mA} = 187,5 \text{ mW}$$

Das sind 187,5 mW Leistung im Durchschnitt.

Mit dem **Betriebsstrom** lässt sich nun auf der Seite des Digikey[TDK24] die Laufzeit des Batteriekapazität berechnen.

**Messung Geräteverbrauch 7,5 mA** Mit der Annahme einer **Leistungsaufnahme** von **7,5 mA** oder **0,005 A** und einer Batteriekapazität von 7000 mAh liefert die Webseite vom Digikey[TDK24] **1296 Monate**. Das übertrifft die Anforderung: Laufzeit auf Seite 22 der 3.

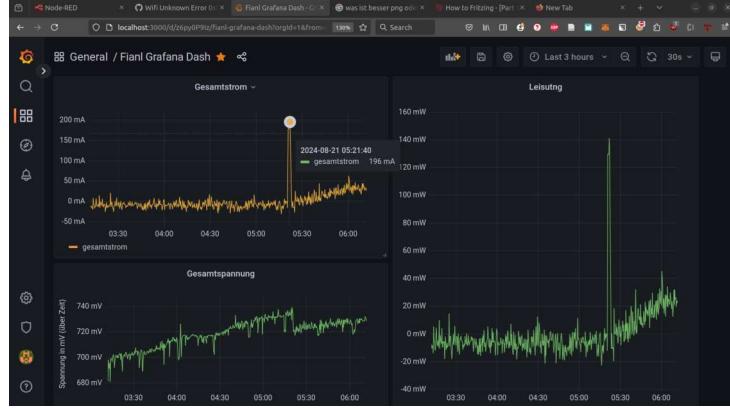
Im nachfolgenden ist ein weiterer Messzeitpunkt aufgeführt. Der Abschluss der Anforderung nach der Laufzeit des Akkus führt zum Fazit auf Seite 8. Die Restlichen Anforderungen in diesem Kapitel 7 werden ungehindert weiter abgehandelt.

## 7.4 Schaltung

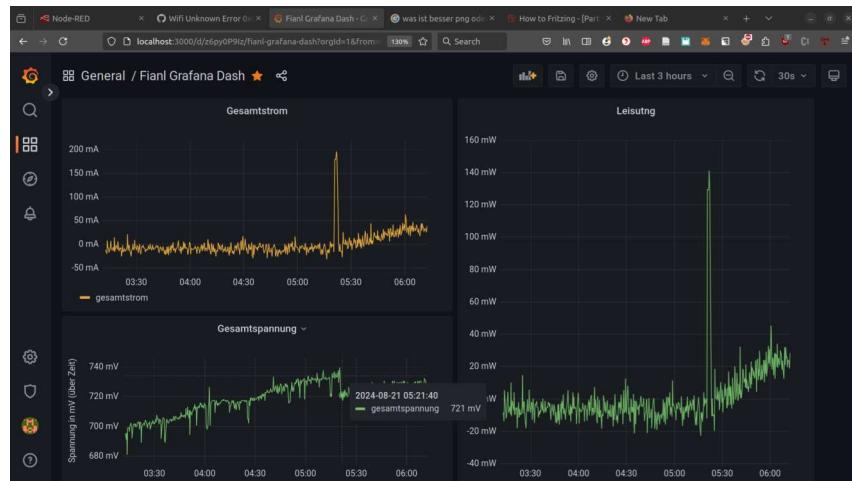
Die Anforderung verlangen: 3.1.4

1. einen anschaulichen Schaltplan wie in Abbildung 4.9 oder 5.1
2. eine Bauteilliste wie in 4.4.1
3. eine technische Zeichnung für die Hobbglassplatten in dwg und svg Formaten wie in 4.6 und im Anhang dieser Abschlussarbeit
4. die Beschreibung der spezifischen Bauteile wie in 2
5. eine Pin-Belegungstabelle wie in 5.1 5.2 und der Abbildung 5.1
6. die Spannungs- und Stromwertkennlinien sind in Abbildung 7.2(a) und 7.2(b) oder 7.2 zu sehen
7. ein Leistungsdiagramm ist 7.2(c) dargestellt

(a) **196 mA** für die **Strom** am **Messpunkt**: 05:21:40 MEZ beim Refresh



(b) **712 mV** für die **Spannung** am **Messpunkt**: 05:21:40 MEZ beim Refresh



(c) **141 mW** beträgt die **Leistung** am **Messpunkt**: 05:21:40 MEZ beim Refresh

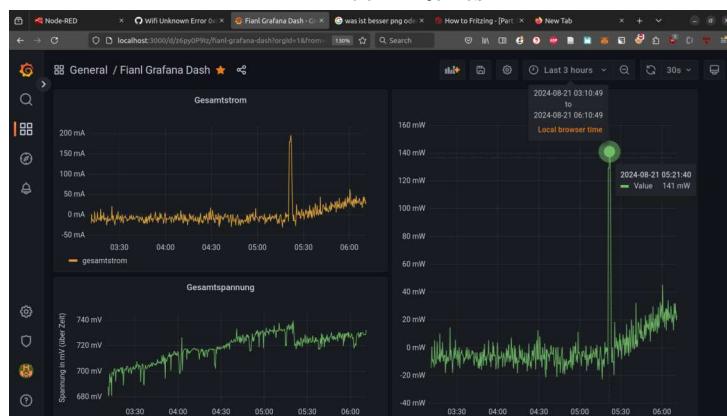


Abbildung 7.3: Messung beim Display-Refresh

## 7.5 Pinbelegung

Die Anforderung: 3.1.5

Die Pin-Belegungstabellen sind im Kapitel 5 den Tabellen 5.1, 5.2 und der Abbildung für die Hardwareschaltung des Infodisplay 5.1 oder dem 2 zu entnehmen.

## 7.6 Wartbarkeit und Schneller Einstieg

Die Anforderung: 3.1.6

Um einen schnellen Einstieg in das Projekt zu finden sind in vielen Ordner **Markdown-Files** als *README.md* verfasst.

Weiterhin ist die Projektstruktur für einen einfachen Einstieg konzipiert.

```

└[hdi10@harun-pc] - [~/Documents/myBA/UltraLowP
owerInfodisplay] - [Sa Aug 24, 03:16]
└[$] <git:(dev*)> tree -caDL 1
[Aug 23 04:15] .
├── [Apr 18 15:01] .editorconfig
├── [Apr 23 16:12] run.py
├── [Jul 12 21:12] TechnischeDoku
├── [Jul 18 12:51] .gitignore
├── [Jul 18 12:51] .gitlab-ci.yml
├── [Jul 18 12:51] micropython.md
├── [Jul 18 12:51] README.md
├── [Aug 6 17:00] MyTex
├── [Aug 8 04:06] MyCode
├── [Aug 13 15:05] Messumgebung
├── [Aug 13 15:05] MyLatestTCode.zip
├── [Aug 13 15:05] Node_red_folder
├── [Aug 13 15:05] StarteAllesOrdner
└── [Aug 17 03:37] Aktuell_und_final_Offline_Di
    splay
    ├── [Aug 17 03:37] MyLatestTCode
    ├── [Aug 21 05:13] MyLatestTexText
    ├── [Aug 21 05:13] MyQuellen
    ├── [Aug 22 20:46] tests
    ├── [Aug 23 02:07] Firmware
    ├── [Aug 23 03:12] Abgabe
    ├── [Aug 23 15:18] .git
    └── [Aug 23 19:07] MyImages

15 directories, 7 files
└[hdi10@harun-pc] - [~/Documents/myBA/UltraLowP
owerInfodisplay] - [~/Documents/myB └[hdi10
└[hdi10@harun-pc] - [~/Documents/myBA/UltraLowP
owerInfodisplay] - [Sa Aug 24, 03:16]
└[$] <git:(dev*)> 

```

Abbildung 7.4: Die GitLab Projektstruktur

In der Abbildung 7.4 ist die Projektstruktur des GitLab Repository im TreeView mit den Dateien bis zu der ersten Ebene im Rootverzeichnis zu sehen:

- .editorconfig - ein template um einen konsistenten programmierstil beizubehalten oder einfach das Unix-style newlines problem zu lösen[Hun24]
- run.py - ein hello-world script
- TechnischeDoku - die Hobbyglasplatte.svg für den Lasercutter
- .gitignore - gitignore
- .gitlab-ci.yml - die CI-Pipe von GitLab

- README.md - readme
- MyTex - Latex Dateien
- MyCode - Alle Micropython Files für den Display
- Messumgebung - alle files für die Sensoren
- Node\_red\_folder - Folder in dem die Noder-Red Flows sind
- StarteAllesOrdner - ein Ordner der Shellscripte beinhaltet zum Starten von Grafana und Node-RED
- MyQuellen - Meine Quellen
- tests - Meine Tests
- Firmware - Die von mir genutzte Firmware und der UnixPort-build
- Abgabe - Meine Zusammgefasste Abgabe
- .git - git logs
- MyImages - Meine genutzten Bilddateien

## 7.7 Aktualisieren

Die Anforderung: 3.1.7

Das Tägliche Aktualisieren an Wochentagen kann wurde mit einer Trigger-Node in Node-RED realisiert.

Der Funktionsblock zur Aktualisierung des Display mit dem Telegram-Kommando '/start' sendet zurzeit eine beliebige Nachricht über MQTT auf dem Topic:'B\_EiNK/006/image' um den Driver-Lifecycle auf dem Infodisplay zu Aktualisierung des Displays, mit dem Import von 'showTable.py' zu starten.

Der Driver-Lifecycle ist in Abbildung 4.12 zu sehen.

## 7.8 Partielles Aktualisieren

Die Anforderung: 3.1.8 Diese Anforderung musste aus Zeitlichen Gründen im Rahmen dieser Abschlussarbeit fallen gelassen werden.

Einen Ausblick hierzu gibt es im Kapitel 4 zu dieser Abschlussarbeit.

## 7.9 Bitmap an Infodisplay

Die Anforderung: 3.1.9 Um eine Bitmap an den Infodisplay zu Übertragen muss ein Buffer über MQTT auf dem TOPIC: „B\_EiNK/006/image“ versendet werden.

```

## Ende des Testlaufs ##
TESTUMGEBUNG [dev] % micropython simpler_test.py
## Beginn des Testlaufs ##
test_there_is_an_instance_of_the_mockpaper (_main_.TestDisplay) ... ok
test_the_pins(cs, dc, rst) are the correct state (_main_.TestDisplay) ... ok
test_value_changed (_main_.TestDisplay) ... ok
test_pin_init (_main_.TestDisplay) ... ERROR
=====
FAIL: test_pin_init <class 'TestDisplay'>
-----
Traceback (most recent call last):
  File "unittest/_init_.py", line 393, in run_one
    File "simpler_test.py", line 76, in test_pin_init
      AttributeError: 'TestDisplay' object has no attribute 'AssertFalse'
-----
Ran 4 tests
FAILED (failures=0, errors=1)
## Ende des Testlaufs ##
TESTUMGEBUNG [dev] %

```

Abbildung 7.5: Mit dem **Test-Runner** und der **Testumgebung** ist **Testgetriebene Entwicklung** möglich

**Nachdem** eine passende Schnittstelle angelegt wurde kann das Bild mithilfe der *display(frame\_buffer)*-methode aus der e7in5k.py - Display-Driver-Datei ein Bild auf dem Display darstellen. Dazu muss die darin befindlichen EPD-Klasse mit einer SPI-Schnittstelle initialisiert werden und es kann ein Bild auf dem Display darstellen.

Diese und andere Display-Anwendungen sind im Kapitel 5 aufgeführt.

Diese Code-Beispiele können aus dem GitLab direkt auf den Infodisplay geladen und ausgeführt werden.

Es handelt sich um Code-Beispiele die verschiedene Lösungsansätze nutzen um ein Bild auf dem Display zu präsentieren.

Diese Code-Beispiele laufen so wie sie auf den Infodisplay geladen werden, robust.

Der Code ist nur geringfügig in Konstellation mit anderen Code-Modulen funktionstüchtig. Da das Flashspeicher-Handling noch aufgefangen werden müsste um die Memory-Allocation Error<sup>1</sup> des Mikrocontroller zu vermeiden.

Als Lösungsschritt wird ein häufiges `machine.reset()` oder das nutzen des Garbage Collectors von micropython empfohlen.

Diese Memory-Allocation Error sind der Hauptgrund für Fehlschlagenden Code.

Doch dazu wird im Kritischen Rückblick des Fazit auf Seite 8 noch weiter eingegangen.

## 7.10 Systemarchitektur

Die Anforderung: 3.1.10 Siehe dazu die **Gesamtsystemarchitektur4.9**

## 7.11 Powerbank

Die Anforderung: 3.1.11

Zur Powerbank anforderung ist zu sagen:

Durch den **Laderegler TP4056** ist es möglich den **Infodisplay** mit einer Stromquelle oder Powerbank mit und ohne Akku zu betreiben.

---

<sup>1</sup>Allein eine Google-Suche zeigt das viele Programmierer auf dasselbe Problem gestoßen sind.

## 7.12 Leistung

Die Anforderung: 3.1.12

Für diese Anforderung werden Leistungsdiagramme angefertigt die das Fazit darstellen sollen.

Damit die Frage nach dem Energieeinsparpotenzial gegenüber Papier aushängen Aussagekräftig beantwortet werden kann.

## 7.13 Tests

Die Anforderung: 3.1.13

Der Nutzer hat eine Test-Umgebung für micropython auf einem UnixPort-build.  
Diese kann für zukünftige Testplanungen mit der Test Suite genutzt werden.

Der Nutzer kann auch den Python zum ausführen der Test nutzen. [cha:test]

Der Nutzer bekommt in der Testumgebung durch die Projektstruktur und den MyTestRunner einen Überblick über die Testdurchführung.

Die Test werden im Kapitel 6 abgehandelt. Die Test und Testumgebung mit dem UnixPort-build sind im GitLab Repository der Abschlussarbeit zu finden.

Auf Seite 8 im Fazit, wird ein Zukunftsanschauung zur Anforderung der Tests gegeben.

(Telegram Test - weitere micropython Test - CD/CI - Multimeter Tests)

## 7.14 Serielle Schnittstelle

Die Anforderung: 3.1.14

# Kapitel 8

## Fazit

### 8.1 Zusammenfassung der Ergebnisse

Im Rahmen dieser Abschlussarbeit wurde ein Explorativer Prototyp des Infodisplay zur Darstellung der Raumbelegung an der HTW entwickelt und getestet, mit dem Ziel, die Effizienz von ePaper Displays gegenüber herkömmlichen Papieraushängen zu evaluieren.

Anhand der durchgeführten Messungen und Berechnungen zeigte sich, dass die Laufzeit des Prototyps bei einer gemessenen Leistungsaufnahme von **7,5 mA** etwa **1 Monat und eine Woche** beträgt.

Dieses Ergebnis liegt deutlich unter der angestrebten Laufzeit von 6 Monaten und erfüllt somit die gestellten Anforderungen nicht.

Im Vergleich dazu wurde jedoch festgestellt, dass bei einer theoretisch geringeren Leistungsaufnahme von **0,0065mA** eine Laufzeit von über 1495 Monaten möglich wäre, was die Anforderung weit übertreffen würde.

Die vorliegenden Ergebnisse verdeutlichen, dass die derzeitige Umsetzung des Prototyps noch nicht die notwendige Energieeffizienz erreicht, um den lagfristigen Betrieb ohne regelmäßigen Batteriewechsel zu gewährleisten.

Für eine praktische Anwendung wären daher Optimierungen im Bereich der Energieeinsparung erforderlich, um eine nachhaltige und kosteneffiziente Lösung für den Austausch von Papieraushängen durch ePaper Displays zu ermöglichen.

## 8.2 Prägende Entwicklungsphasen des Projektes

Prägende Entwicklungsphase des Projektes waren für mich die Agile Arbeitsweise. Der Entwurf und die Entwicklung der Systemarchitektur. Das Aufsetzen der Testumgebung. Der Bau der Prototypen. Das Produkt als Teil eines Drahtlosen Sensornetzes zu sehen. Dem Infodisplay.

Die Systemarchitektur ist in 8.1 und 8.2 Abgebildet.

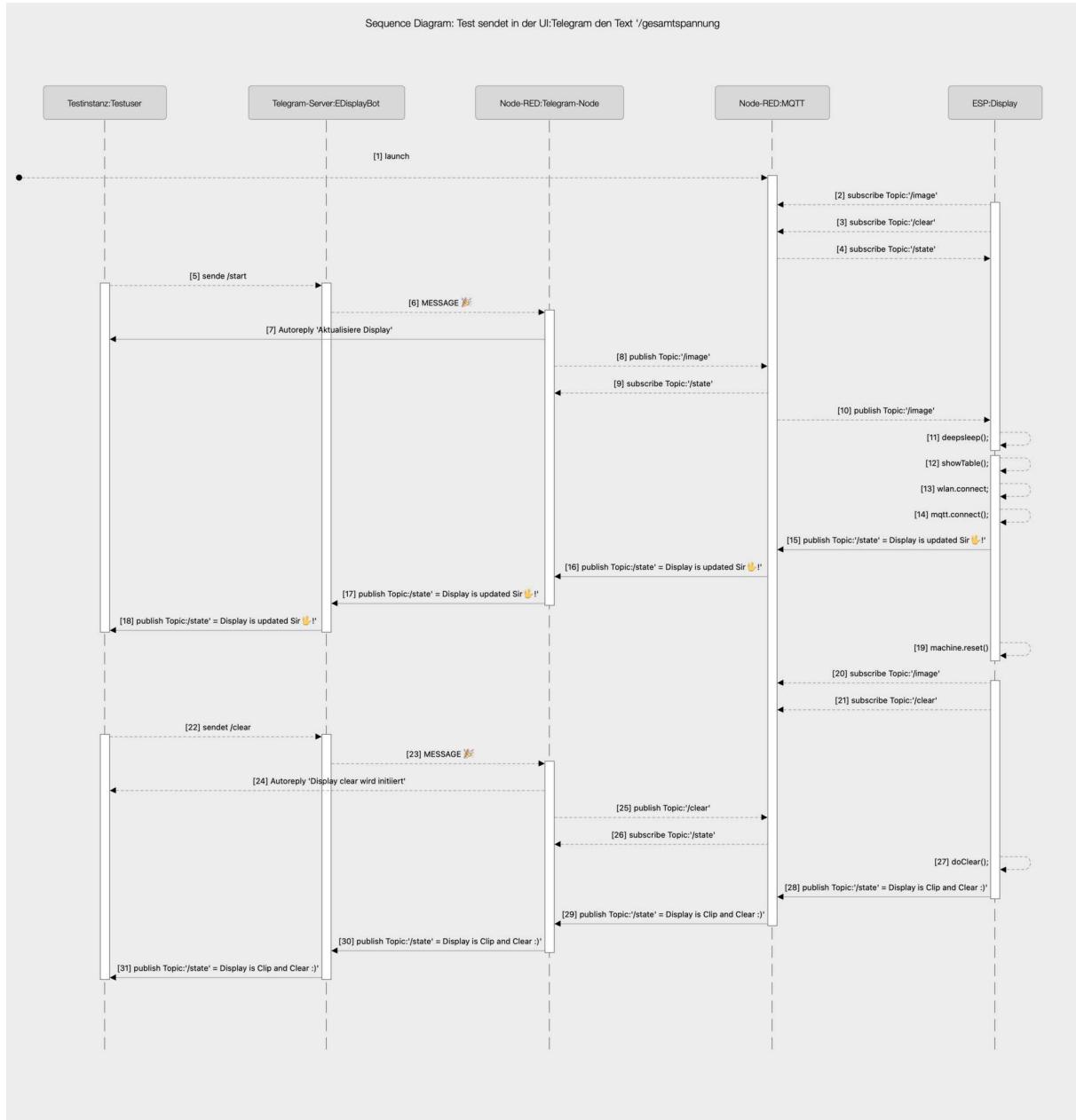


Abbildung 8.1: Sequenzdiagramm für Start und Clear Kommandos

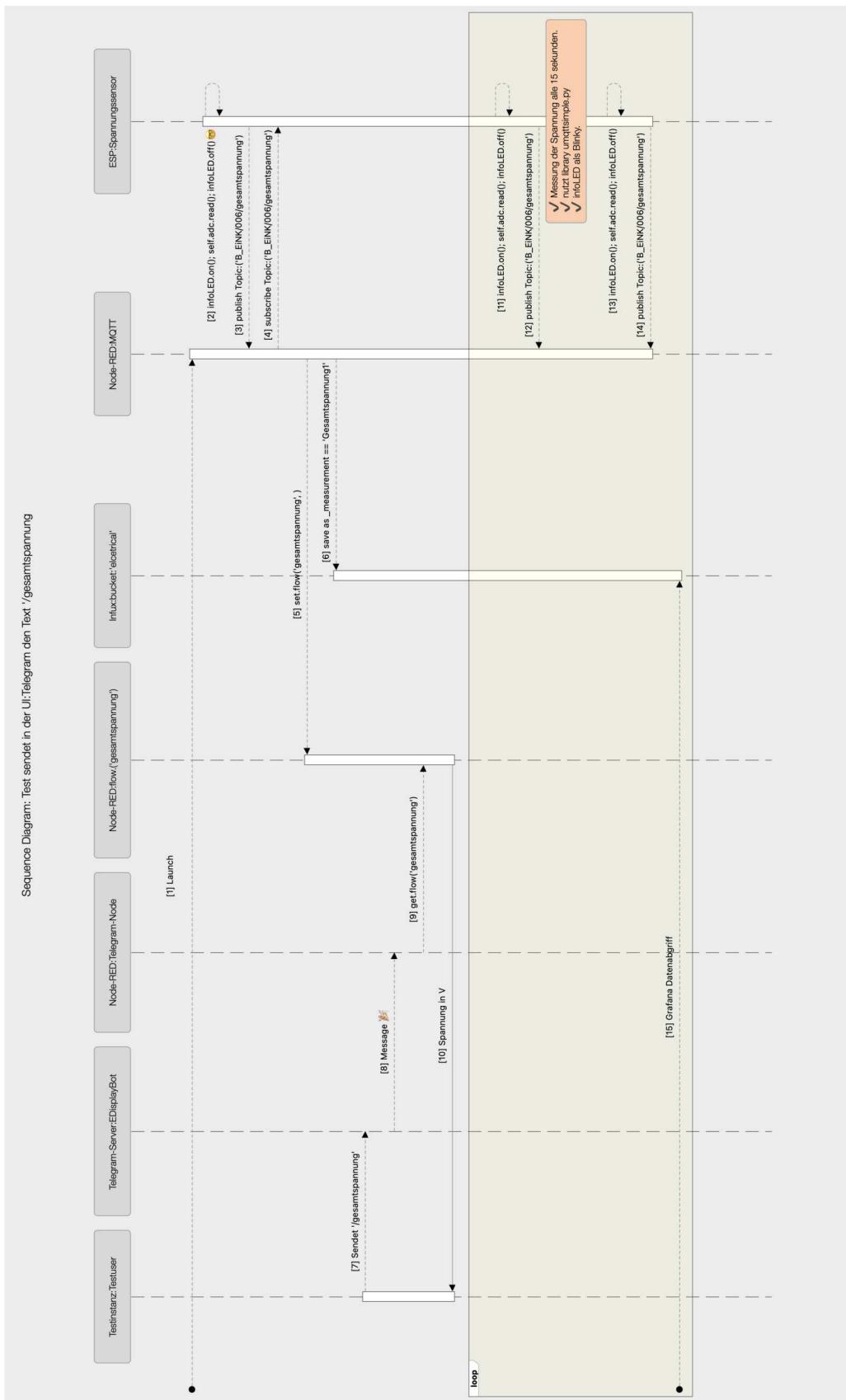


Abbildung 8.2: Sequenzdiagramm zu Spannungsabfrage

## 8.3 Ausblick

Ein Infodisplay zur Raumplanbelegung ist meiner Meinung nach ein Fortschritt in die richtige Richtung. Das ist auch bereits im Sortiment der Händler bemerkbar. Eckstein, der Onlinehandelsshop des Waveshare ePaper Display, bietet aktuell auch Günstigere Varianten des E-Ink Display mit 7in5 Zoll[**eckstein\_neu an**].

Und Waveshare<sup>1</sup>.

Waveshare hat auch bereits Ihre Website aktualisiert.

---

<sup>1</sup>Waveshare hat Ihre Seite angepasst![Wav24a]

# Abbildungsverzeichnis

2.1	Eine Microcontroller Systemarchitektur für ein Drahtloses Sensornetz[ <b>KONG20151</b> ] . . . . .	4
2.3	Schalterstellung zur Display-Art[24b] . . . . .	6
2.2	Die Frontansicht auf den <b>e-Paper ESP32 Driver Board</b> mit Zwei Schaltern für <b>A</b> oder <b>B</b> , für die <i>Displayart</i> und ON oder OFF, für die Stromversorgung des Serial-Port-Moduls (siehe auch Vorder- und Rückansicht des e-Paper ESP32 Driver Board) . . . . .	6
2.4	Pinout-Angaben von Waveshare . . . . .	8
2.5	Vorder- und Rückansicht des e-Paper ESP32 Driver Board . . . . .	9
2.6	Der Infodisplay mit . . . . .	10
2.7	Schaltskizze Strom- und Spannungsmessung . . . . .	11
2.8	Messung zur Bestimmung der elektrischen Leistung nach: <b>P = U * I</b> in <i>Watt</i> . . . . .	12
2.9	Batteriemanagementsystem und der Akku . . . . .	13
2.10	Screenshot mit Ausgabe der Serielle Schnittstelle auf dem Terminal . . . . .	15
2.11	Screenshot der <i>Modules</i> auf dem <b>ESP32</b> mit MicroPython v1.22.2 . . . . .	16
2.12	Screenshot . . . . .	17
2.13	Der Flashvorgang der Firmware auf den ESP32. . . . .	19
2.14	Ausgabe aller integrierten und verfügbaren Bibliotheken des <b>ESP32-Driver</b> , die über den Befehl <i>import</i> importiert werden können . . . . .	20
3.1	Definitions- und Umsetzungsphase . . . . .	21
3.2	Anforderungsbeschreibung in Scrum[] . . . . .	22
4.1	Schaltbild zur Planung der Positionierung des Schalter S1 als Ein- und Ausschalter . . . . .	29
4.2	Schaltbild der Infodisplayschaltung . . . . .	31
4.3	Schaltplan des Prototypen mit ESP8266 und 4.2 inch Display . . . . .	32

4.4	Erster Prototyp ESP8266 1000 mAh Akku und 4.2 inch ePaper der mithilfe zwei Hobbyglas platten steht . . . . .	33
4.5	Lasergravierer Trotec Speedy 400 im Makers Lab der HTW-Berlin . . . . .	35
4.6	Vektorgrafik einer Hobbyglasplatte 500x500mm für den Lasergravierer . . . . .	36
4.7	Breadboard Schaltung der Strommessschaltung . . . . .	39
4.8	Grafische Darstellung der Spannungsmessschaltung . . . . .	40
4.9	Erster Gesamtentwurf . . . . .	43
4.10	Anwendungsfalldiagramm: User nutzt den Telegram EDisplayBot . . . . .	44
4.11	Sequenzdiagramm: User schickt „/gesamtspannung“ als Text an den Tele- gramBot:EDisplayBot . . . . .	45
4.12	Activity Diagramm: Display refresh beim erhalt einer Nachricht auf dem Topic: „_EiNK/006/image, . . . . .	46
5.1	Hardwareschaltung des Infodisplay . . . . .	50
5.2	Pinbelegung der Spannungsmessschaltung . . . . .	51
5.3	Pinbelegung der Strommessschaltung . . . . .	52
5.4	Leistungsmessschaltung über MQTT und Telegram . . . . .	53
5.5	Gesamte Messschaltung . . . . .	55
5.6	Auszug aus der Command Table im Datenblatt ?? . . . . .	56
5.7	Driver-Code mit der EPD-Klasse als Entität mit Typen der Attribute und Signaturen der Operationen . . . . .	58
5.8	Beispiel-Code für eine Darstellung einer Nachricht anhand einer Binary im App-Speicher . . . . .	63
5.9	Beispiel-Code für eine Darstellung einer Tabelle anhand einer Binary im App-Speicher . . . . .	64
5.10	Rekursiver Aufruf . . . . .	66
5.11	Hier werden die Zeile-Tage und die Spalte-Zeiten aus einer ByteArrayList.py geladen, der Content wird rekursiv gezeichnet . . . . .	67
5.12	Hier wird alles mithilfe eines Writer mit dem HTW-Font und der framebuffer- Klasse ein Raumbelegungsplan gezeichnet . . . . .	70
5.13	Der Infodisplay Node-Red Flow im Gesamtbild . . . . .	72
5.14	Der TelegramBot des Infodisplay: EdisplayBot ( EdisplayBot) . . . . .	73
5.15	Bot-Kommandos . . . . .	74
5.16	Screenshot von Grafana zurückschicken an User . . . . .	75
5.17	Der Infodisplay Node-Red Flow GesamtflowInfodisplaybild3 . . . . .	75
5.18	mqqt . . . . .	76
5.19	mqqt . . . . .	76

---

6.1	Die Ordner der Manuellen und Automatisierten Testfälle . . . . .	81
6.2	Topics des Infodisplay . . . . .	87
6.3	Der System-Output von Node-Red im Terminal . . . . .	89
6.4	Ankommende Messwerte als Nachricht über MQTT und als Datenpunkt aus der Influx visualisiert in Grafana . . . . .	91
7.1	Mit dem <b>Test-Runner und der Testumgebung ist Testgetriebene Entwicklung möglich</b> . . . . .	94
7.2	Die Messreihe vom 09.08.2024 von 15 bis 2 Uhr des Folgetages in Grafana .	96
7.3	Messung beim Display-Refresh . . . . .	98
7.4	Die GitLab Projektstruktur . . . . .	100
7.5	Mit dem <b>Test-Runner und der Testumgebung ist Testgetriebene Entwicklung möglich</b> . . . . .	102
8.1	Sequenzdiagramm für Start und Clear Kommandos . . . . .	107
8.2	Sequenzdiagramm zu Spannungsabfrage . . . . .	108
A.1	Ein Foto mit der Canon EOS 550D meiner Ehefrau . . . . .	XIV

# Tabellenverzeichnis

4.1	Pin Belegungstabelle für die Strommessung . . . . .	39
4.2	Pinbelegung Spannungsmessung . . . . .	40
5.1	Spannungsmessung Pinbelegung . . . . .	51
5.2	Strommessung Pinbelegung . . . . .	52

# Source Code Content

2.1	Change Directory to /dev[ices . . . . .	14
2.2	Change Directory to /dev[ices . . . . .	14
2.3	Flashen der Firmware auf den ESP32 mit dem esptool . . . . .	18
2.4	Ausgabe aller Modul-Bibliotheken . . . . .	20
4.1	Initialisierung der Pins und Instanzierung des Epaper-Display . . . . .	38
4.2	Initialisierung der Pins im Konstruktor der Klasse für den Strom Sensor in der currentButterfly7.py . . . . .	39
4.3	Initialisierung der Pins im Konstruktor der Klasse für den Spannungssensor in der voltageButterflyV3.py . . . . .	41
5.1	lookUpTableVCom0 . . . . .	56
5.2	Die Pins, die die <code>__init__</code> -Methode der EPD-Klasse als Übergabepara- meter bekommt . . . . .	59
5.3	Deklaration der Pins . . . . .	61
5.4	Instanzierung der Serielle Schnittstelle:spi und der ePaperDisplay: e . . . . .	61
5.5	Der Buffer und FrameBuffer für den Infodisplay . . . . .	62
5.6	binAufEink . . . . .	64
5.7	Ein KaroMuster erzeugt aus einem Rekursiven Aufruf . . . . .	66
5.8	Byte-Array für die Zeile-Tage . . . . .	67
5.9	Display-Code: showTable.py welcher Zeile-Tage und Spalte-Zeiten aus einer File.py läd . . . . .	67
5.10	Byte-Array für die Zeile-Tage . . . . .	70
5.11	Codezeilen aus dem Funktions Block „Aktualisiere Display“ nach Telegram Kommando Block für /start . . . . .	72
5.12	Codezeilen aus dem Funktions Block „Measurement POWER“ nach dem Complete Block zur Strom und Spannungsmessung . . . . .	76
5.13	Beide Sensoren nutzen MQTT. Sie unterscheiden sich nur in Ihrem Topic und in Ihrer spezifischen Klasse . . . . .	77

5.14	Byte-Array für die Zeile-Tage . . . . .	77
5.15	Berechnung des Stromwertes . . . . .	78
6.1	Importzeilen . . . . .	85
6.2	Importzeilen . . . . .	85
6.3	Test-Runner in <b>simpler-test.py</b> . . . . .	86

# Glossar

**dwg** DWG ist ein CAD-Format (Computer-Aided-Design). Das ursprüngliche Zeichenformat (erkennbar an der Abkürzung „DWG“ für Drawing) enthält heute zwei- und dreidimensionale Vektorgrafiken und wird in den Bereichen Technik, Architektur und Ingenieurswesen für die Entwurfserstellung verwendet.[[AdobeDWGFile2024](#)]. 23, 97

**ePaper Esp32 Driver-Board** ePaper Esp32 Driver-Board von waveshare. 28

**ESL** Electronic Shelf Label. 105

**Esp32** Esp32 von Espressif. 30, 31

**HTW** Hochschule für Technik und Wirtschaft Berlin. ii, 5, 35

**Institute of Electrical and Electronics Engineers** Weltweit größte technische Fachorganisation, die sich für den Fortschritt der Technologie zum Wohle der Menschheit einsetzt . VIII

**IOT** Internet of Things. 4

**Laderegler** Bietet konstante Strom-/Spannungsladung, Übertemperaturschutz und im besten Fall eine Ladezustandsanzeige. 11

**LIP0** Lithium-Ionen-Polymer. ii, 11, 13, 28

**ls** list directory contents. 12

**Message Queuing Telemetry Transport** Protokoll zu drahtlosen Nachrichtenübermittlung. 5

**MicroPython port** MicroPython supports different boards, RTOSes, and OSes, and can be relatively easily adapted to new systems. MicroPython with support for a particular system is called a “port” to that system. Different ports may have widely different functionality. This documentation is intended to be a reference of the generic APIs available across different ports (“MicroPython core”). Note that some ports may still omit some APIs described here (e.g. due to resource constraints). Any such differences, and port-specific extensions beyond the MicroPython core functionality, would be described in the separate port-specific documentation.[**MicroPythonGlossary**]. 20

**REPL** Read-Evaluate-Print-Loop und bezeichnet den interaktiven MicroPython-Prompt auf dem ESP32, der über eine serielle Verbindung oder über WiFi zugänglich ist.[**MicroPythonREPL**]. 20

**RFID** Radiofrequency-Identification. 4

**svg** Das Format Scalable Vector Graphics (SVG)[**AdobeSVGFile2024**]. 23, 35, 97

**User Interface** Grafische Nutzeroberfläche. 5

**Wireless LAN (Local Area network)** Drahtlose Netzwerktechnologie, basierend auf dem standard 802.11 des IEEEInstitute of Electrical and Electronics Engineers. 5, 20

**WSN** Drahtloses Sensornetzwerk oder wireless sensor network. 4

# Literaturverzeichnis

- [Pic09] Roman Pichler. *Scrum - agiles Projektmanagement erfolgreich einsetzen.* 1. Aufl., korrigierter Nachdr. Accessed: 2024-06-12. Heidelberg: dpunkt-Verl., 2009. ISBN: 3898644782.
- [wav19] waveshare. *7.5inch<sub>e</sub> – PaperV2specification.* 2019, S. 22–25.

# Onlinereferenzen

- [24a] *E-Paper Driver HAT Version Description Rev2.3.* Accessed: 2024-04-02. 2024. URL: [https://www.waveshare.com/wiki/E-Paper\\_Driver\\_HAT](https://www.waveshare.com/wiki/E-Paper_Driver_HAT) (besucht am 02. 04. 2024).
- [24b] *E-Paper ESP32 Driver Board.* Accessed: 2024-08-13. 2024. URL: [https://www.waveshare.com/wiki/E-Paper\\_ESP32\\_Driver\\_Board](https://www.waveshare.com/wiki/E-Paper_ESP32_Driver_Board) (besucht am 13. 08. 2024).
- [Con24] MicroPython Contributors. *MicroPython Documentation: machine - functions related to the hardware.* Accessed: 2024-05-26. 2024. URL: <https://docs.micropython.org/en/latest/library/machine.html>.
- [DAL24] LTD. DALIAN GOOD DISPLAY CO. *Bistabil E-Paper Beschilderung,DES 5,83 zoll, geringer Stromverbrauch, hoher Kontrast.* <https://de.good-display.com/product/412.html>. 2024. (Besucht am 24. 08. 2024).
- [Dc24] Paul Sokolovsky Damien P. George und contributors. *MicroPython manifest files.* <https://docs.micropython.org/en/latest/reference/manifest.html>. 2024. (Besucht am 22. 08. 2024).
- [Dev24a] MicroPython Developers. *MicroPython.* <https://micropython.org>. 2024. (Besucht am 23. 05. 2024).
- [Dev24b] Thonny Developers. *Thonny.* <https://thonny.org>. 2024. (Besucht am 23. 05. 2024).
- [dja22] djantzen. *Pico Book.* [https://github.com/djantzen/pico\\_book/tree/master](https://github.com/djantzen/pico_book/tree/master). 2022. (Besucht am 22. 08. 2024).
- [fiz20] fizista. *MicroCRON is a time-based task scheduling program.* <https://github.com/fizista/micropython-mcron>. Accessed: 2024-08-24. 2020. (Besucht am 24. 08. 2024).

- [Gmb24] Eckstein GmbH. *Waveshare 7.5inch E-Ink Raw Display 800x480 for Arduino SPI interface.* <https://eckstein-shop.de/Waveshare800C3974802C75inchE-InkRawDisplaySPIinterfaceArduino>. Accessed: 2024-08-25. 2024. (Besucht am 25.08.2024).
- [Hun24] EditorConfig Python Core: Trey Hunner. *What is EditorConfig?* <https://editorconfig.org/>. Accessed: 2024-08-24. 2024. (Besucht am 21.02.2019).
- [Mic24a] MicroPython. *MicroPython Firmware for ESP32.* [https://micropython.org/download/ESP32\\_GENERIC/](https://micropython.org/download/ESP32_GENERIC/). 2024. (Besucht am 23.05.2024).
- [Mic24b] MicroPython. *MicroPython Firmware for ESP32.* [https://micropython.org/resources/firmware/ESP32\\_GENERIC-20240602-v1.23.0.bin](https://micropython.org/resources/firmware/ESP32_GENERIC-20240602-v1.23.0.bin). 2024. (Besucht am 23.05.2024).
- [mic24] micropython.org. *MicroPython cross compiler.* <https://github.com/micropython/micropython/tree/master/mpy-cross>. 2024. (Besucht am 22.08.2024).
- [Pic09] Roman Pichler. *Scrum - agiles Projektmanagement erfolgreich einsetzen.* 1. Aufl., korrigierter Nachdr. Accessed: 2024-06-12. Heidelberg: dpunkt-Verl., 2009. ISBN: 3898644782.
- [Sho24] Eckstein Shop. *TP4056 Micro USB 5V 1A LiPo Akku Lademodul Lithium Battery Charging Module.* 2024. URL: <https://eckstein-shop.de/TP4056XD-58AMicro-USB5V1ALiPoAkkuLademodulLithiumBatteryChargingModule>.
- [Sys22a] Espressif Systems. *ESP32 Datasheet.* 2022. URL: [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf).
- [Sys22b] Espressif Systems. *ESP32 Technical Reference Manual.* [https://www.espressif.com/sites/default/files/documentation/esp32\\_technical\\_reference\\_manual\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf). Accessed: 2024-08-24. 2022. (Besucht am 13.08.2024).
- [Sys23] Espressif Systems. *ESP32WRROOM32E.* [https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e\\_esp32-wroom-32ue\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf). Accessed: 2024-08-25. 2023. (Besucht am 25.08.2024).
- [Sys24a] Espressif Systems. *Application Startup Flow - ESP32.* <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/startup.html>. 2024. (Besucht am 23.05.2024).
- [Sys24b] Espressif Systems. *Bootloader - ESP32.* <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/bootloader.html>. 2024. (Besucht am 23.05.2024).

- [TDK24] DigiKey by TDK-Lambda. *Rechner für Batterielebensdauer*. <https://www.digikey.de/de/resources/conversion-calculators/conversion-calculator-battery-life>. 2024. (Besucht am 23.08.2024).
- [Wav24b] Waveshare. *e-Paper ESP32 Driver Board User Manual*. 2024. URL: [https://files.waveshare.com/upload/4/4a/E-Paper\\_ESP32\\_Driver\\_Board\\_user\\_manual\\_en.pdf](https://files.waveshare.com/upload/4/4a/E-Paper_ESP32_Driver_Board_user_manual_en.pdf).

# Bildreferenzen

- [Ele24] How to Electronics. *Voltage Sensor Image*. <https://images.app.goo.gl/oZuQP75zaawN8XaQ7>. Accessed: 2024-08-25. 2024. (Besucht am 25.08.2024).
- [Mar24b] Marotronics. *Current Sensor Image*. <https://images.app.goo.gl/3srfjW4CXBsh1T5C6>. Accessed: 2024-08-25. 2024. (Besucht am 25.08.2024).
- [Sys22b] Espressif Systems. *ESP32 Technical Reference Manual*. [https://www.espressif.com/sites/default/files/documentation/esp32\\_technical\\_reference\\_manual\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf). Accessed: 2024-08-24. 2022. (Besucht am 13.08.2024).
- [Wav24c] Waveshare. *Universal e-Paper Raw Panel Driver Board, ESP32 WiFi / Bluetooth Wireless*. <https://www.waveshare.com/e-paper-esp32-driver-board.htm>. Accessed: 2024-08-24. 2024. (Besucht am 13.08.2024).
- [Wav24d] Waveshare. *Universal e-Paper Raw Panel Driver Board, ESP32 WiFi / Bluetooth Wireless*. 2024. URL: <https://www.waveshare.com/e-paper-esp32-driver-board.htm> (besucht am 13.08.2024).

# Anhang A

## A.1 Beispiel



Abbildung A.1: Ein Foto mit der Canon EOS 550D meiner Ehefrau

# Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, den 26.08.2024

Harun Dastekin

Harun Dastekin