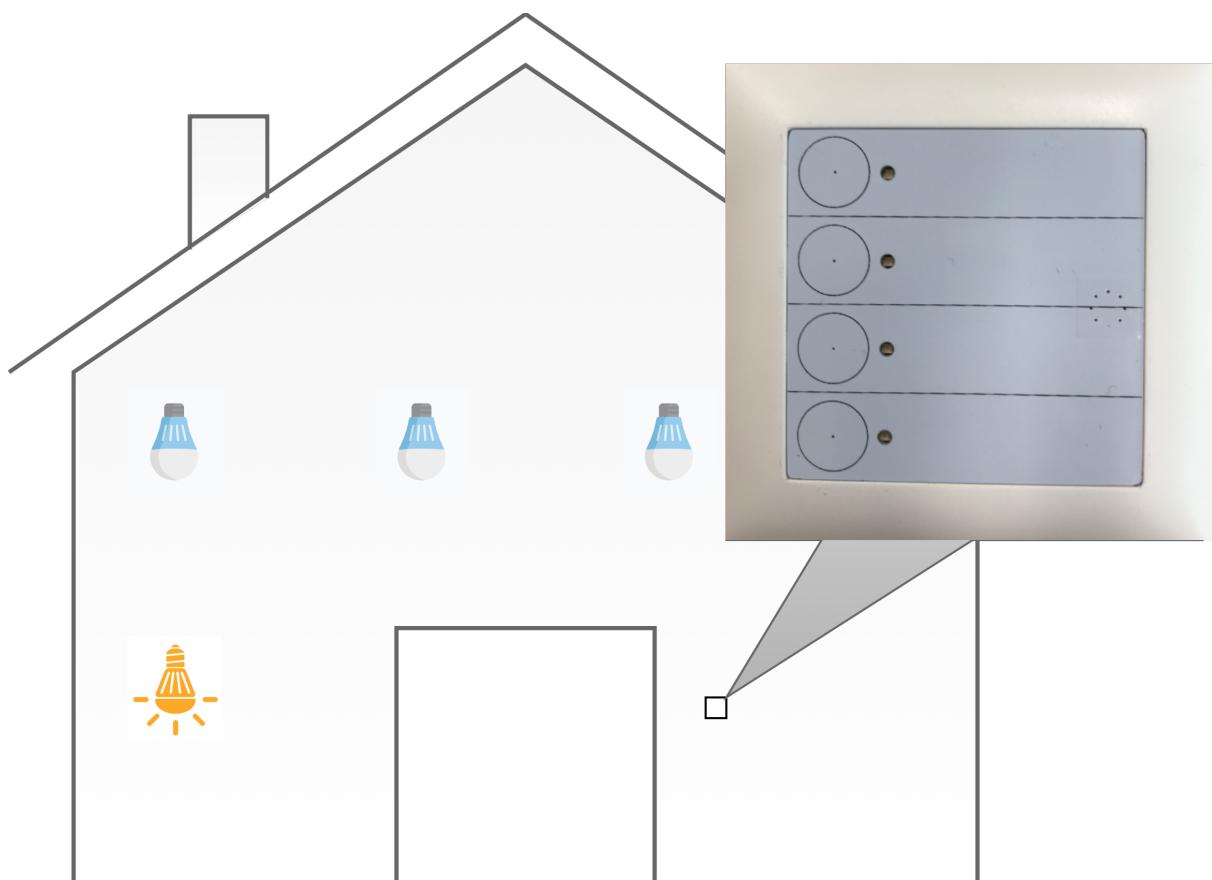


Fachbericht

Integrierte IOT-Raumautomation mit MQTT-Protokoll

Windisch, 14.08.2020



Hochschule

Hochschule für Technik - FHNW

Studiengang

Elektro- und Informationstechnik

Autoren

Lukas Meienberger und Gabriel Nussbaumer

Betreuer und Auftraggeber

Albert Zihlmann

Experte

Remy Bringold

Version

1.0

Abstract

Smart Home is the automation, and data-based control of electronic devices and light energy systems in your own home. This report is about a smart home system solution with Mqtt communication. The development of an actor board with different switching possibilities, where electronic devices are connected, as well as the development of a touch sensor board with temperature measurement are described. A closed system with its own MQTT-broker is completed with the server software. The server software enables a perfect interaction between the components and the integration of a language assistant. The whole system can be integrated with existing Smart Home and offers cross-platform compatibility with other solutions.

Inhaltsverzeichnis

1 Einleitung	1
2 Gesamtübersicht	2
3 Theorie	3
3.1 Wireless Local Area Network (WLAN)	3
3.1.1 Beschreibung	3
3.1.2 Datenrate	3
3.1.3 Sicherheit	4
3.1.4 Funktion WPA2	4
3.1.5 WPA2 im Vergleich zu WPA3	4
3.2 Message Queuing Telemetry Transport (MQTT)	5
3.2.1 Beschreibung	5
3.2.2 Geschichte	5
3.2.3 Überischt	5
3.2.4 MQTT-Server (Broker)	5
3.2.5 Client	5
3.2.6 Eigenschaften MQTT Broker	6
3.3 Mqtt Paket Struktur	6
4 OpenHab	8
4.1 Archidektur	8
4.1.1 OSGI	8
4.1.2 JAR	9
4.1.3 Übersicht Kommunikation Verbindungen	9
5 Software	11
5.1 Mikrocontroller	11
5.1.1 ADC	11
5.2 Framework	13
5.2.1 Entwicklerumgebung	13
5.2.2 Framework Arduino	14
5.2.3 Smart-Home Plattform Openhab	14
5.3 Kommunikation	15

5.3.1	WLAN Konfiguration	15
5.4	Programmcode Sensorbord	17
5.4.1	Übersicht	18
5.4.2	setup()	19
5.4.3	loop()	19
5.4.4	touch()	21
5.4.5	shine()	21
5.4.6	tread()	21
5.4.7	tempf()	21
5.5	Programmcode Aktorbord	25
5.5.1	Übersicht	25
5.5.2	setup()	26
5.5.3	loop()	26
5.5.4	client.loop()	28
5.5.5	callback()	28
5.5.6	publishADC()	28
5.6	Task2code()	28
5.7	Programmcode Openhab	29
5.7.1	Bindings	29
5.7.2	Things	30
5.7.3	Channel	30
5.7.4	Item	30
5.7.5	Rules	30
5.7.6	Sitemaps	30
5.8	Server	30
5.9	Sprachassistent	31
6	Hardware	33
6.1	Sensorbaustein	33
6.1.1	Übersicht Frontplatte	33
6.1.2	Übersicht Hauptplatine	34
6.1.3	Übersicht Spannungsversorgung	34
6.1.4	Schutzbeschaltung	36
6.1.5	Mikrocontroller	36

6.1.6	Temperatursensor	37
6.1.7	Programmieranschluss	37
6.1.8	Spannungsversorgung	39
6.1.9	Platzstudie	40
6.2	Aktorbaustein	42
6.2.1	Mikrocontroller	43
6.2.2	Relais	44
6.2.3	Eingänge	44
6.2.4	Ausgänge	46
6.2.5	Spannungsversorgung	47
6.2.6	Interface	48
7	Validierung	49
7.1	Sensorbaustein	49
7.1.1	ESD Test	49
7.1.2	Energieverbrauch des Sensorbausteins	50
7.1.3	Temperaturmessung	52
7.2	Aktorbaustein	53
7.2.1	Energieverbrauch des Aktorbaustein	53
7.2.2	Wärmeverteilung	55
7.2.3	Reaktion Zeiten	56
8	Lizenzen	59
9	Verbesserungen	60
10	Schluss	62
10.1	Reflektion	62
10.2	Ausblick	62
11	Ehrlichkeitserklärung	63
Literatur		64

1 Einleitung

Für eine integrale Raumautomation müssen alle beteiligten Gewerke geregelt werden, dazu gehören: Beschattung, Licht, Heizung, Lüftung und Klima. Der Auftraggeber möchte eine möglichst preisgünstige Sensor/Aktor-Plattform um jene integrale Raumautomation zu ermöglichen. Bis-her wurde die Idee, verschiedene Sensoren und Aktoren über IOT miteinander auszulesen bzw. zu steuern, in der Raumautomations-Branche verwirklicht, jedoch hat es sich als schwierig erwiesen solch eine anwendungsfreundliche Plattform, wie sich der Auftraggeber gewünscht hat, käuflich zu erwerben. Um die Situation zu verbessern soll eine Plattform auf Basis eines Single-Chip-uC, welcher Ein- und Ausgänge sowie ein Sensor-Baustein beinhalten realisiert werden. Zusätzlich möchte man mithilfe eines MQTT-Brokers die IOT-Geräte verbinden. Die Erstinbetriebnahme der Plattform soll mit Hilfe eines Web-Interface ablaufen, wofür das Gerät erstmal als WiFi Access-Point auf startet. Die folgende Tabelle 1.1 fasst die Anforderungen an das System zusammen.

Aktor Baustein	<ul style="list-style-type: none"> • 4 Relais Schaltspannung 230 V • 2 Ausgänge 0-10 V • 2 Eingänge 0-10 V • Netzwerkschnittstelle • 24 VDC Versorgungsspannung
Sensor Baustein	<ul style="list-style-type: none"> • 4 Taster • Netzwerkschnittstelle • Geometrie: Standard Lichtschalter • 5 VDC Versorgungsspannung
Kommunikation	<ul style="list-style-type: none"> • MQTT Broker • WEB Interface • Sprach Assistent

Tabelle 1.1: Ausgangslage

2 Gesamtübersicht

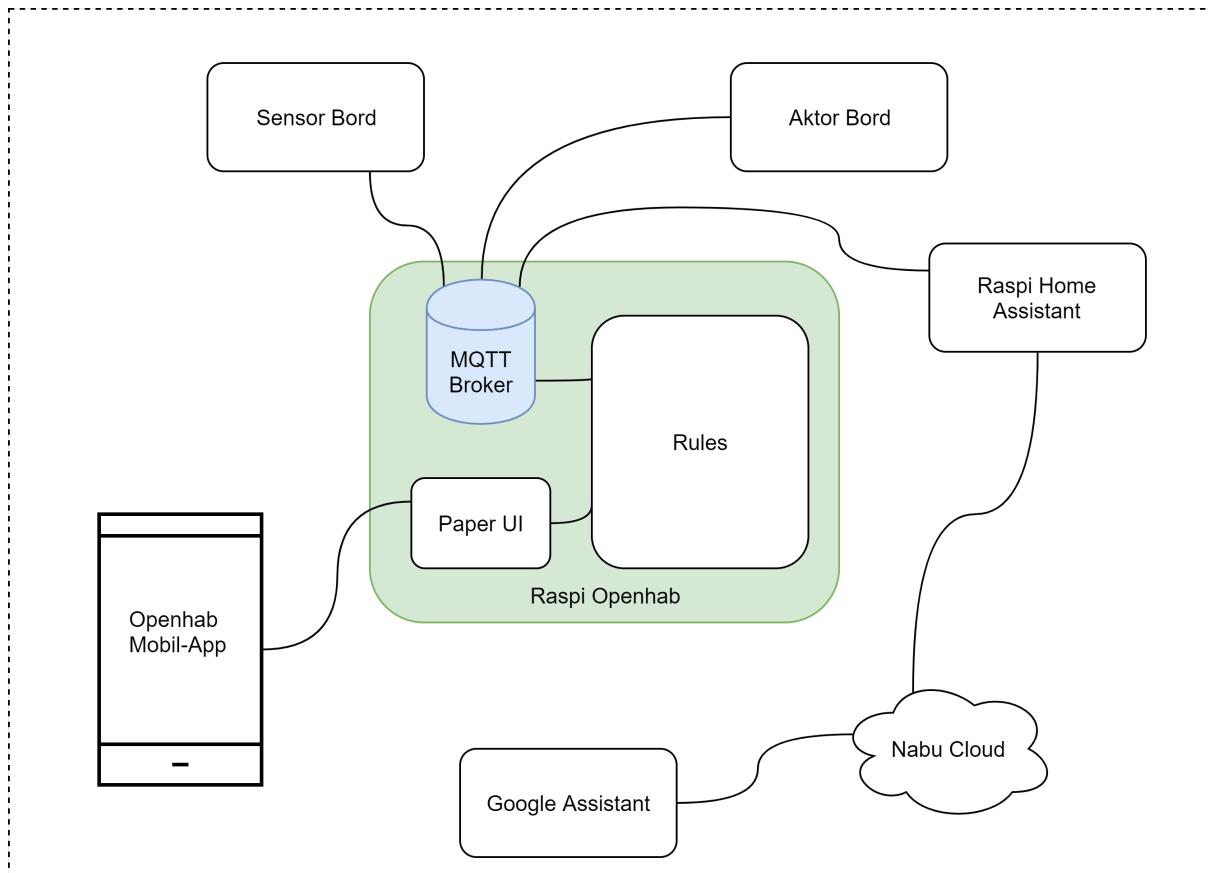


Abbildung 2.1: Gesamtübersicht Komponenten Becholour Thessiss

In der Abbildung 2.1 sind die Geräte und Ihre Verbindungen schematisch abgebildet. Als Herzstück dient der Openhab Server, welcher auf ein Rasperrypi aufgesetzt wurde. Ein eigener Mqtt-Broker, welcher die kommunikations-Messages managt wurde eingebunden. Verschiedene Geräte wie das Aktor- und Sensor Bord, auch der Home Assistant, welcher als Brücke für den Google Assistant dient, gelingt es im eigenen Lokalen Netzwerk zu kommunizieren. Als Web-Interface steht das Paper UI von Openhab wie auch das Mobile-App dem Benutzer, zum Bedienen vom System zur Verfügung. Mit den Rules sind alle Automatisierten Vorgänge in Openhab festgelegt, in diesem Projekt sind dies die Definitionen welchem Befehl welche Schalthandlungen durchgeführt werden müssen. Der Grund warum die Schalthandlungen und die Verknüpfungen in den Ruls definiert werden, liegt daran, dass so keine Programmierung in die Einzelnen Aktor- oder Sensor Bords Durchgeführt werden müssen. Der Benutzer kann die Konfigurationen in Openhab Graphisch im Web-Interface oder als Programmiercode in einem Editor wie Visual Studio Code durchführen, siehe Benutzerhandbuch Anhang.

3 Theorie

3.1 Wireless Local Area Network (WLAN)

3.1.1 Beschreibung

Als lokales Funknetz ist WLAN weit verbreitet. WLAN ist eine Abkürzung für Wireless-Local-Area-Netzwerk, in deutsch ein Drahtloses-Lokales-Areal-Netzwerk. Die Idee der Lokalen Funkkommunikation war, dass in Büros mit mobilen Geräten wie Laptops eine kabellose Verbindung zum Internet hergestellt werden kann. Damit die Anbindung an das standardisierte LAN nicht in jedem Büro unterschiedliche Eigenschaften benötigt, wurde vom IEEE-Komitee entschieden ein Standard einzuführen, das Komitee entwickelte somit den 802.11 Standard für die drahtlosen Verbindungen. IEEE-802.11 Systeme nutzen unlizenzierte Frequenzbänder Bänder wie z.B. 902-928 MHz oder 2.4 - 2.5 GHz. Sämtliche Geräte dürfen dieses Spektrum nutzen, vorausgesetzt sie begrenzen ihre Sendeleistung, damit mehrere Geräte nebeneinander betrieben werden können. IEEE-802.11 Netze bestehen aus Clients wie Laptops und Smartphones sowie einer Infrastruktur, Zugangspunkt AP (Access Point), welcher im Gebäude installiert ist. Der AP ist die Schnittstelle zum verkabelten Netz und ist im Privaten Haushalt auch oft direkt im Modem nebst dem Router integriert siehe Abbildung 3.1.

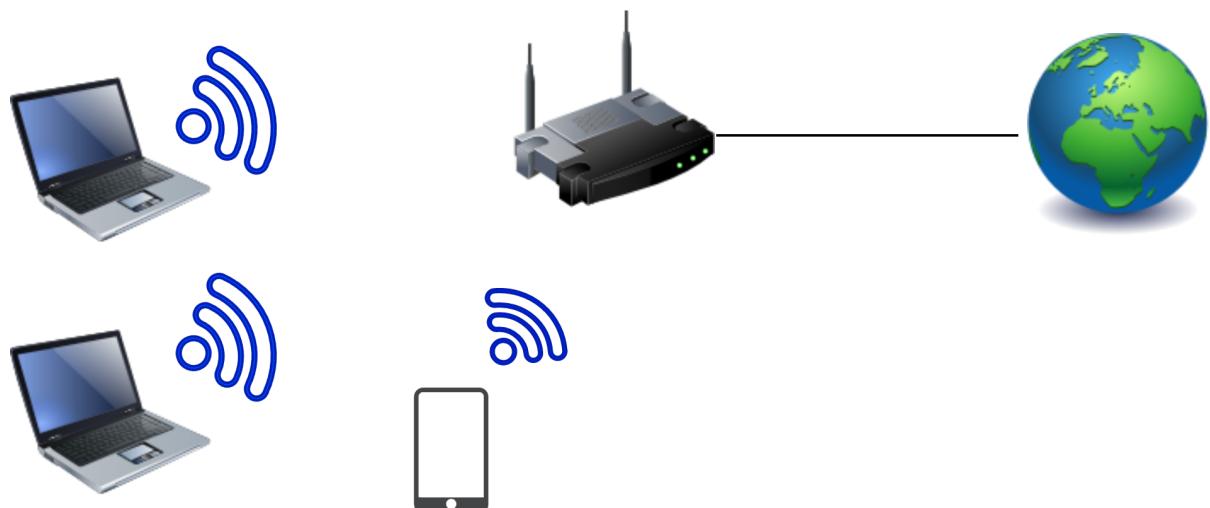


Abbildung 3.1: Übersicht IEEE 802.11

3.1.2 Datenrate

Der IEEE 802.11 Standard definierte 1997 ein drahtloses LAN, das entweder 1 Mbit/s oder 2 Mbit/s sendete und entweder Frequenzen wechselte oder das Signal über das erlaubte Spektrum verstreute. Die Funkverbindung wurde weiterentwickelt, so dass die Geschwindigkeit zunahm, in der nachfolgenden Tabelle ist eine Übersicht.

Standard	Übertragungsleistung	Modulation
IEEE-802.11b	11Mbit/s	Frequenzspreizung
IEEE-802.11a/g	54Mbit/s	OFDM
IEEE-802.11n	450Mbit/s	Mehrere Frequenzbänder

Tabelle 3.1: IEEE 802.11 Standards [1]

Das OFDM-Schema, Orthogonal Frequency Division Multiplexing, verwendet mehrere Trägerfrequenzen. Mehrere eng beieinanderliegende orthogonale Hilfsträgersignale mit überlappenden Spektren übertragen Daten parallel.

3.1.3 Sicherheit

Die kabellosen Übertragungen sind Broadcast-Verbindungen, daher besteht das Problem, dass Informationspakete abgefangen werden können. Um dies zu verhindern ist im IEEE 802.11 Standard das Verschlüsselungsschema, WEP (Wired Equivalent Privacy) enthalten. Das WEP wurde durch WPA (WiFi Protected Access) ersetzt, und schliesslich wurde WPA durch WPA2 ersetzt. Im Juni 2018 wurde von Wi-Fi Alliance WPA3 vorgestellt, die nächste Generation der WiFi Sicherheit.

3.1.4 Funktion WPA2

Die WPA2 Verschlüsselung wird in zwei verschiedenen Szenarien verwendet.

Ein Unternehmen hat ein Authentifizierungsserver mit Benutzernamen und Kennwortdatenbank eingerichtet, mit dem festgelegt wird, ob ein drahtloser Client auf das Netz zugreifen darf. In diesem Fall verwenden Clients Standardprotokolle, um sich zu authentifizieren. Im zweiten Szenario haben alle Clients ein gemeinsames Passwort, die Verbindung wird ohne Authentifizierungsserver aufgebaut. Diese Methode wird oft in privaten Netzwerken angewendet. Der Hauptunterschied ist, dass beim Authentifizierungsserver jeder Client ein Schlüssel zur Datenverschlüsselung bekommt, beim gemeinsamen Passwort wird der Schlüssel bei jedem Client aus dem Passwort abgeleitet und ist daher weniger sicher. Nachdem sich der Client mit dem Passwort ausgewiesen hat, geschieht ein 4-Pakete-Handshake, und somit werden weitere Schlüssel erstellt.

3.1.5 WPA2 im Vergleich zu WPA3

Bei WPA3 gibt es wieder die 2 verschiedenen Betriebsarten.

WPA3-Personal bietet eine stabilere Kennwort basierte Authentifizierung, selbst wenn Benutzer Kennwörter ändern, SAE (Simultaneous Authentication of Equals), schützt den Benutzer stärker gegen Dritte, welche versuchen das Passwort zu erraten.

WPA3 Enterprise bietet kryptografischen Schutz mit der Stärke von 192 Bit, somit geeignet für sensible Daten von einem Finanzinstitut [2].

3.2 Message Queuing Telemetry Transport (MQTT)

3.2.1 Beschreibung

MQTT ist ein Nachrichtentransport Protokoll, mittels Client werden Nachrichten veröffentlicht sprich abonniert und mit dem MQTT-Broker verwaltet. Anwendung findet MQTT im Bereich in eingeschränkten Umgebungen, wie Kommunikation von Maschine zu Maschine (M2M) und Internet der Dinge (IoT). Das Protokoll läuft über TCP/IP oder über andere Netzwerkprotokolle die verlustfreie bidirektionale Verbindungen bieten [3].

3.2.2 Geschichte

MQTT wurde 1999 von dr. Andy Stanford-Clark von IBM und Arlen Nipper von Arcom erfunden. Seit 2013 ist MQTT über die nicht gewinnorientierte Organisation OASIS als Protokoll des Internet der Dinge standardisiert [3].

3.2.3 Übersicht

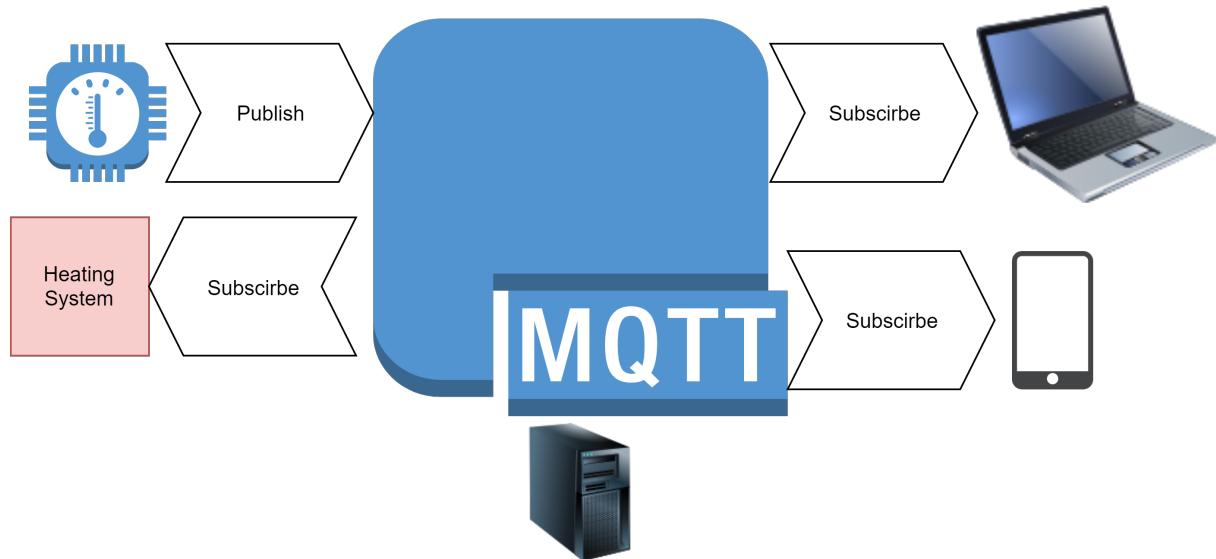


Abbildung 3.2: Übersicht MQTT

In der Abbildung 3.2 werden Daten von einem Thermostat veröffentlicht. Ein Heizsystem, ein Notebook und ein Smartphone haben die Daten abonniert und der MQTT-Broker verwaltet die Nachrichten.

3.2.4 MQTT-Server (Broker)

Der MQTT-Server ist ein Programm oder Gerät, welches als Vermittler zwischen Clients dient und wird auch als MQTT-Broker bezeichnet. Der Broker nimmt Netzwerkverbindungen von Clients an, empfängt so die Nachrichten von Clients und gibt diese Nachrichten an jene Clients weiter, welche ein Abonnement abgeschlossen haben.

3.2.5 Client

Ein Client ist ein Programm oder Gerät, welches MQTT verwendet. Dieser baut eine Verbindung zum Broker auf. Der Client veröffentlicht Nachrichten mit einem Topic und einer Payload. Ist

ein Client an bestimmten Daten interessiert abonniert er diesen Topic und empfängt somit die Payload.

3.2.6 Eigenschaften MQTT Broker

Veröffentlichen, abonnieren oder Publish/Subscribe kann in Form von One-to-many Nachrichten verwendet werden.

MQTT benötigt wenig Zusatzinformationen beim Transport und minimiert so die Protokoll Austausche, somit wird die Belastung des Netzwerkverkehrs minimiert.

Es gibt ein Mechanismus zur Benachrichtigung interessierter Parteien, wenn eine Unterbrechung einer Verbindung auftritt.

Für die Nachrichten Zustellung gibt es drei Service Qualitäten [4]:

- Einmal übertragen, in diesem Fall wird die Nachricht, mit den besten Bemühungen der Betriebsumgebung, übermittelt.
- Mindestens einmal übertragen, wobei sichergestellt wird, dass die Nachricht ankommt, aber Duplikate auftreten können.
- Exakt einmal übermitteln, in diesem Fall wird sichergestellt, dass die Nachricht genau einmal ankommt.

3.3 Mqtt Paket Struktur

MQTT ist ein binärbasiertes Protokoll, bei dem die Steuerelemente Binäre Bytes sind, jedem Befehl ist eine Bestätigung zugeordnet.

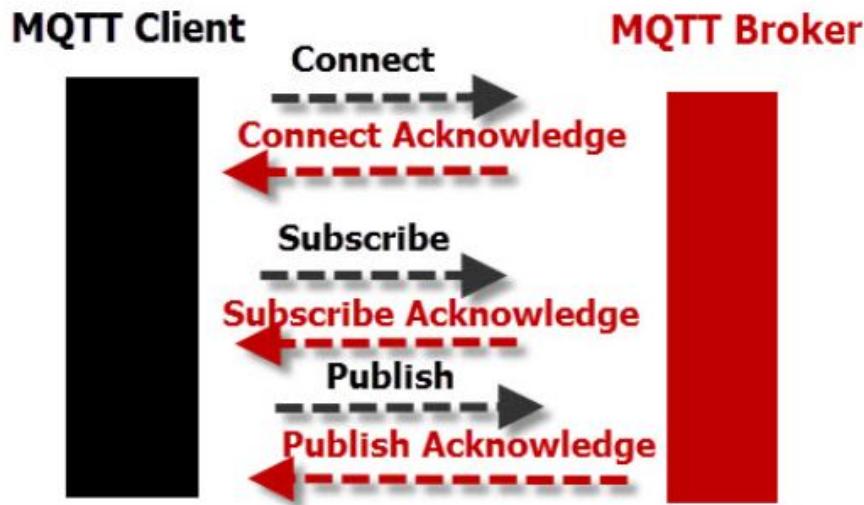


Abbildung 3.3: Protokoll Verbindung Client zu Broker [4]

Topic Namen, Client-ID und Benutzernamen werden als UTF-8-Strings kodiert. Die Payload sind binäre Daten, der Inhalt wie das Format sind anwendungsspezifisch. Das MQTT-Packet besteht aus einem festen 2 Byte-Header und der Payload bei Bedarf kann noch ein variablen Header hinzugefügt werden.

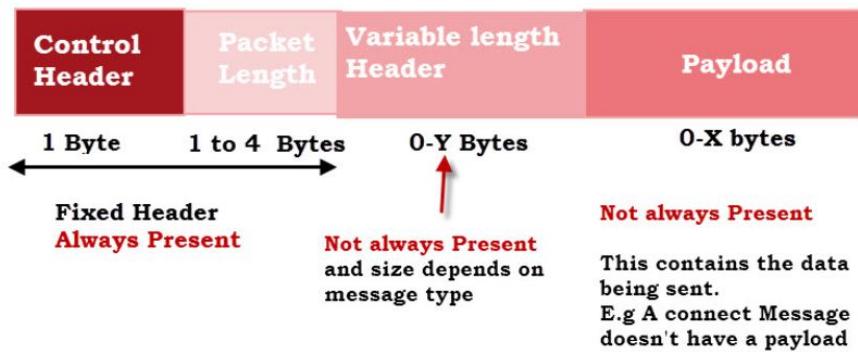


Abbildung 3.4: Standard MQTT Paket Struktur [4]

Es gibt 3 Grundlegende Paket Formate

- Fester Header (Steuerfeld und Länge)
- Fester Header (Steuerfeld und Länge) und variable Payload
- Fester Header (Steuerfeld und Länge) variabler Header und variable Payload

Die Mindestgrösse des Paketlänge Felds beträgt 1 Bit was für Nachrichten die ohne Header kleiner als 127 Byte sind.

Die maximale Paket grösse beträgt 256 MB

Das Steuerfeld ist das erste Byte des Headers es ist in zwei 4-Bit Felder unterteilt und enthält Protokollbefehle wie auch antworten.

4 OpenHab

In dieser Arbeit wird als Server ein RaspberryPi verwendet. Als Software befindet sich Openhab auf dem Linux Betriebssystem. Openhab ist eine Gebäudeautomation Software und bietet Kontabilität zwischen verschiedenen Smarthome Komponenten wie KNX, Sonos, oder Philips Hue. Der für das Projekt benötigte MQTT Broker kann als Binding in Openhab eingebunden werden. Nach den Installationen kann der Funktionserhalt mit den Geräten im selben lokalen Netzwerk gewährleistet werden, auch ohne Verbindung mit dem öffentlichen Internet.

4.1 Archidektur

Openhab basiert auf der modularen OSGI Architektur.

4.1.1 OSGI

Die OSGI Technologie bietet eine Vielzahl von dynamischen Spezifikationen für Java Systemkomponenten. Dies ermöglicht ein Entwicklungsmodell, bei dem eine Anwendung aus mehreren Komponenten entsteht die als Pakete gebündelt sind. Die Komponenten sind Bausteine und wiederverwendbar, sie kommunizieren lokal untereinander. Die OSGI Architektur ermöglicht es den Komponenten die Implementierung vor andern Komponenten zu verbergen. Dies reduziert die Gesamtkomplexität und ermöglicht eine hohe Zuverlässigkeit, da während laufendem System Wartung und Entwicklungsarbeiten Durchführt werden können.

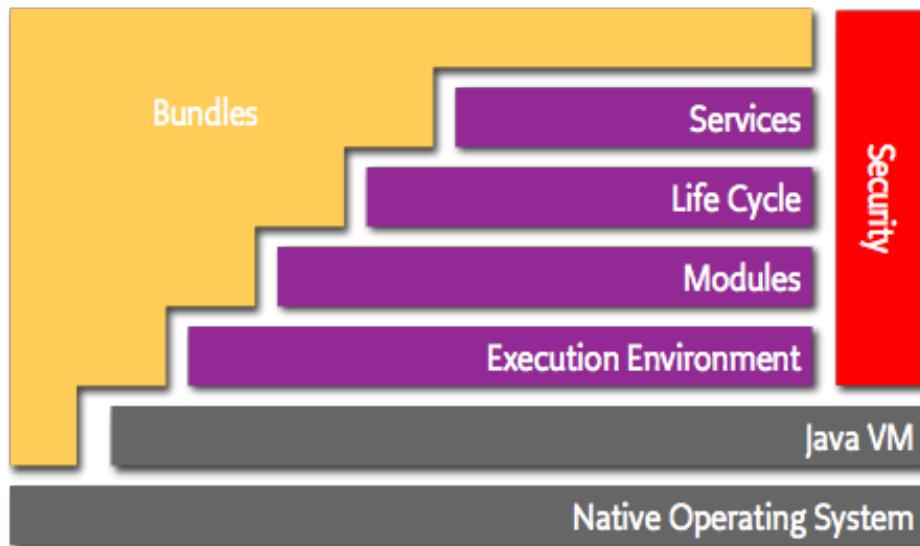


Abbildung 4.1: OSGI Layers [5]

Bundles sind Module sie sind die kleinste Einheit der Modularisierung. Ein Bundle ist eine JAR-Datei mit zusätzlichen Meta-Informationen.

In den Meta-Informationen sind Bundlesabhängigkeits Informationen. Ein Bundle kann von einem andern Bundel oder von einem Paket abhängig sein.

Die OSGi-Runtime verwendet die Informationen über die Abhängigkeiten, um die Bundles zu verdrahten und versteckt alles in dieser JAR, sofern es nicht explizit exportiert wird. Die Abhängigkeiten zu den Java-Standardsbibliotheken werden durch den Bundle-Header verwaltet, so

dass es nicht notwendig ist, die Java-Kernpakete zu importieren.

Bundles werden oft zur Registrierung und zum Konsum von Dienstleistungen verwendet.

Die Bundles können in Laufzeit installiert, deinstalliert und geändert werden. Die Spezifikationen der OSGI Architektur also Abhängigkeiten und der Mechanismus wird mit Hilfe des Lebenszykluskonzepts erreicht. Der Rahmen führt verschiedene Zustände ein und regelt wie sich die vom Bundle exportierten Pakete und Dienste auswirken.

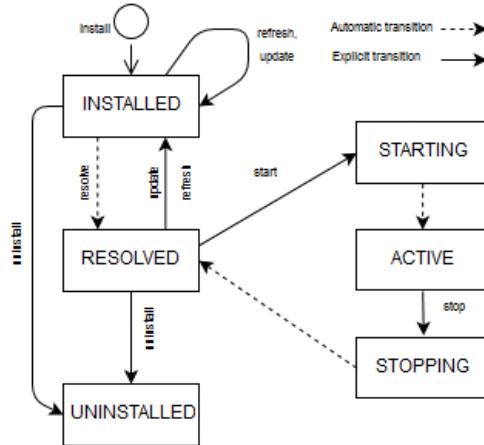


Abbildung 4.2: Bundle State Diagramm [5]

In diesem Diagramm ist ersichtlich, dass Bundles während des Ausführens nicht geändert werden können, sonnst aber jederzeit.

4.1.2 JAR

4.1.3 Übersicht Kommunikation Verbindungen

Die Kommunikation zwischen den Komponenten geschieht mittels Event Bus. Alle nicht Status bezogenen Bundels informieren darüber andere Bundles über den Status von Events. Über diesen Bus Kommunizieren alle Binding mit einem phsikalischen Link zur realen Hardware. Mit dem Events werden auf asynchrone weise in diesem Bus veröffentlicht, durch den EventSubscriber definierte Callback-Schnittstellen werden diese zur entsprechender Funktion vorgesehenen Events wiederum Empfangen. Der EventSubscriber ist als OSGI Dienst registriert [6].

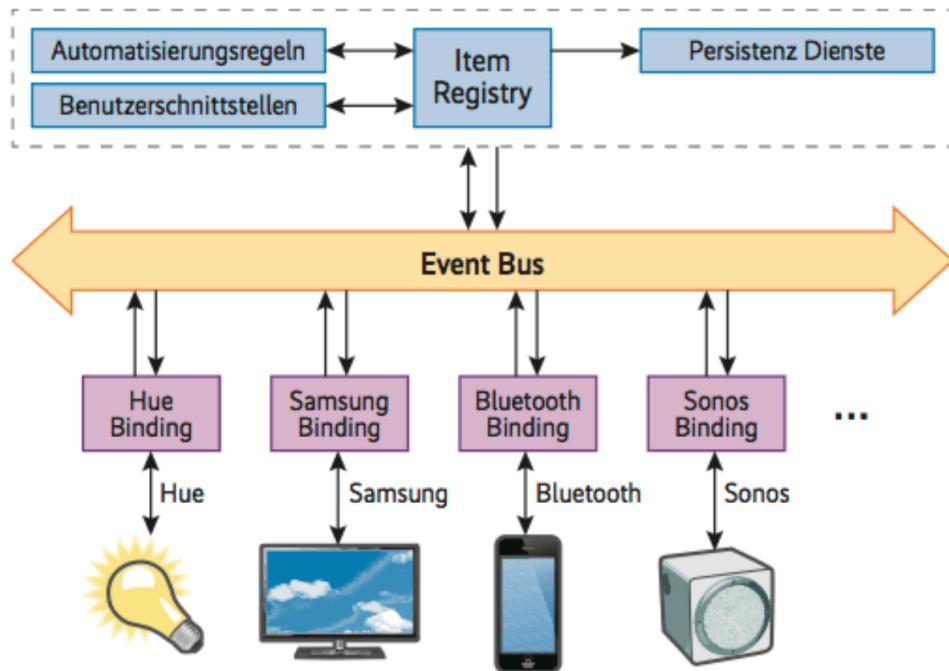


Abbildung 4.3: Eventbus Openhab [7]

In der Abbildung 4.3 ist der Eventbus dargestellt verschiedene Bindings mit Hardwarekomponenten von verschiedenen Smarthome Lösungen empfangen, die für sie vorgesehene Events

5 Software

In diesem Kapitel wird der Softwarebereich beschrieben. Es wird beschrieben wie das Framework für der Mikrocontroller, für den Sensor und Aktorbaustein, evaluiert wurden. Das Framework für den inhouse Server und wie die Kommunikation mit dem MQTT Protokoll realisiert werden.

5.1 Mikrocontroller

Die Auswahl des Mikrocontrollers, ist so ausgefallen, dass dieser für Sensor wie auch für Aktor-Layout geeignet ist. Das Hauptkriterium der Wahl basiert auf der Kommunikation. Der Mikrocontroller von der chinesischen Firma Espressif Systems ist ein Robuster Chip, welcher für industrielle Umgebungen geeignet ist und in Betriebstemperaturen von -40 °C bis +125 °C zuverlässig arbeitet. Der Stromverbrauch kann zwischen verschiedenen Leistungsmodi gewählt werden, so kann während dem Betrieb eine dynamische Leistungsskalierung bewerkstelligt werden. ESP32 Module sind mit integrierten Antennen, Leistungsverstärkern und rauscharmen Empfangsverstärkern hochintegriert. Schnittstellen wie SPI, I2C, UART, WiFi und Bluetooth sind vorhanden.

WiFi: Arbeitet mit dem Standart 802.11b/g/n und 802.11 2,4 GHz bis zu 150 Mbit/s. 4 Virtuelle Wi-Fi interfaces und Soft AP

Bluetooth: Kompatibel mit Bluetooth v4.2 BR/EDR und BLE-Spezifikationen. Sender Klasse 1, Klasse 2 und Klasse 3 ohne externen Leistungsverstärker. Hochgeschwindigkeits-UART HCI, bis zu 4 Mbits/s.

CPU und Memory: Modell abhängig, wobei das Modell ESP-WROOM-32E, mit der Chip Bezeichnung D0WD -V3 betrachtet wird. Dual-Core 32-bit, Flash 4-16 MB, 448 KB ROM, 520 KB SRAM, 16 KB SRAM in RTC.

Prozessor: Tensilica Xtensa LX6 240 MHz

Clocks und Timers: Interner 8 MHz-Oszillator mit Kalibrierung. Externer Quarzoszillator 2 MHz bis 40 MHz. Zwei Timer Gruppen 2x 64-bit Timer mit je einem Haupt Watchdog.

Erweiterte Peripherieschnittstellen: 12-bit SAR ADC bis zu 18 Kanäle, 2x b-bit D/A Wandler, 10x touch Sensoren, Temperatursensor, 4x SPI, 2x I2S, 2x I2C 3x UART.

Sicherheit: Alle unterstützten Sicherheitsfunktionen des IEEE 802.11-Standards, einschließlich WFA, WPA/WPA2 und WAPI

Unterstützung bei der Entwicklung: SDK Firmware für schnelle on-line Programmierung und open Source toolchains basiert auf GCC.

Kosten bei Mouser:

356-ESP32WROOM-32D 3,71 CHF/Stück

5.1.1 ADC

Der ADC des ESP32 verhält sich nicht linear, was zu Problemen führen kann. In der Grafik 5.1 ist zu erkennen, dass der ADC nur bei grösser als U = 0.17 V und kleiner als 3 V überhaupt brauchbar ist, dazu kommt, dass ab 2.5 V der ADC nichtlinear ansteigt. Die Lösung, welche hier verwendet wurde, war eine Linearisierung im Bereich von 0.2 V bis 2.5 V was zu folgender Geradengleichung geführt hat: $Wert_{ADC} = 1253.1 \cdot U - 218.54$. Die Daten stammen aus dem Internet. Diese ist somit mit Vorsicht zu bewerten, es wurden für den Sensor- und Aktorbaustein besser passende Geradengleichungen verwendet, eine für die Temperaturmessung und eine für die analogen Eingänge. Ebenso gibt es noch weitere Probleme mit der ADC-Messung, dazu mehr im Kapitel 9 Verbesserungen.

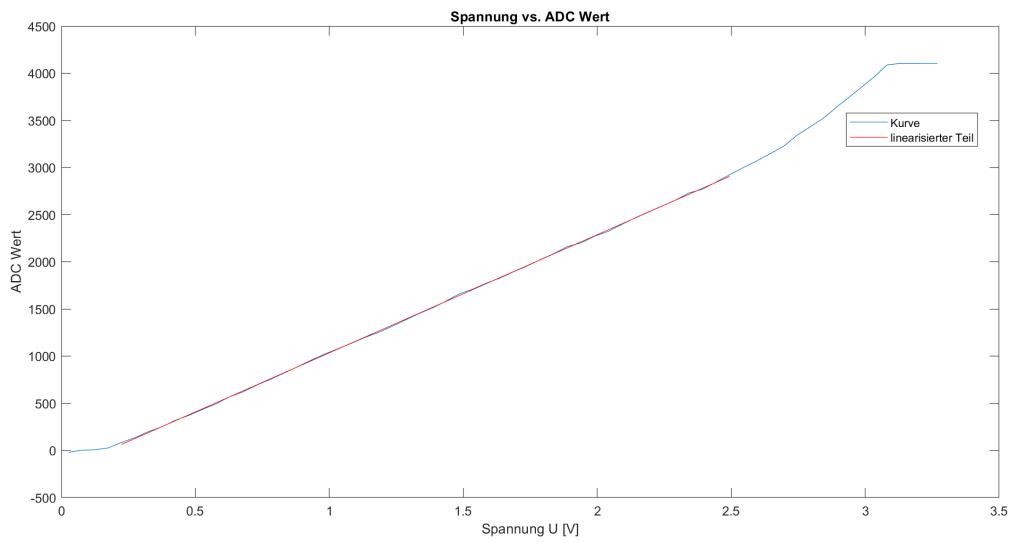


Abbildung 5.1: ESP32 Kurve, Daten aus Grafik von [8]

5.2 Framework

5.2.1 Entwicklerumgebung

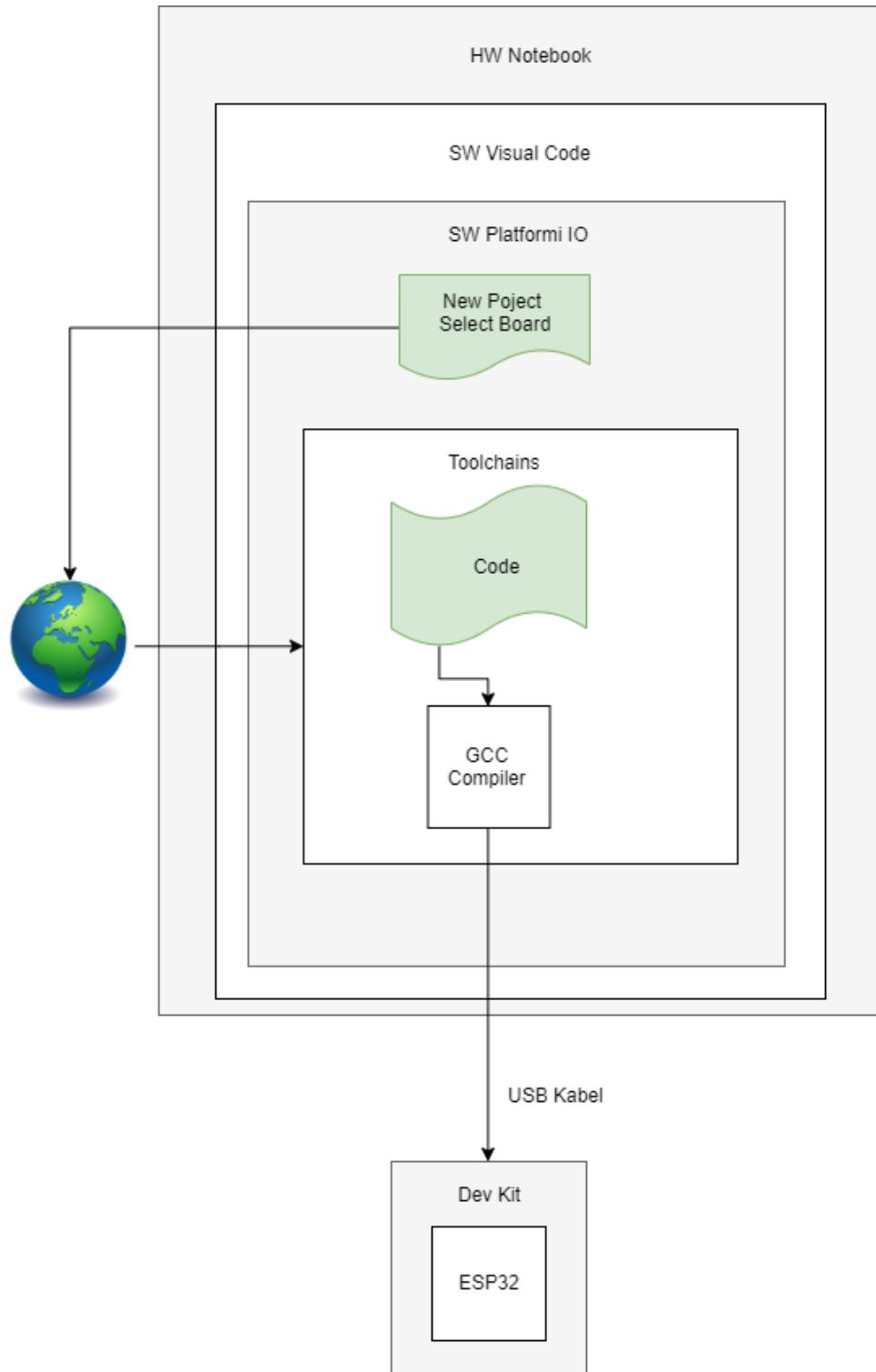


Abbildung 5.2: Entwicklungs Umgebung

Als Entwicklungsumgebung wird Visual Code von der Firma Microsoft verwendet, dies ist ein freier Quelltext-Editor, mit dem unter anderem Debugging möglich ist [9]. In Visual Code kön-

nen Extensions eingebunden werden, somit wird PlatformIO IDE [10] kurz PIO Installiert und Openhab. PIO ist ein plattformübergreifender Code Builder und Bibliotheksmanger. Wird ein neues Projekt eröffnet, muss vom Benutzer zuerst das Entwickler Board gewählt werden, anschliessend kümmert sich PIO um die erforderlichen Toolchains, ladet diese vom Internet herunter und installiert sie. Nun kann das Developmend-Board via USB Kabel angesprochen werden. Programmcode kann kompiliert und auf den Mikrocontroller geladen werden, die Serial Port ausgaben können direkt im PIO angezeigt werden. In Abbildung 5.2 ist ein Blockdiagramm auf dem die Verbindungen zwischen Software und Hardware dargestellt sind.

5.2.2 Framework Arduino

In dem Arduino Framework erfolgt die Programmierung in C und in C++. Das Arduino Framework steht unter der LGPL/GPL Lizenz und ist somit eine freie Software. Ein grosser Vorteil des Arduino Frameworks ist, dass zahlreiche Bibliotheken und sowie Programmcode auf Github erhältlich sind [11]. Die Wahl des Arduino Framework ist auf Grund der Programmiersprache, Verfügbarkeit und bisherigen Erfahrungen erfolgt. Ein Funktionsmuster mit Hotspot, für Grundkonfigurationen und erste MQTT-Nachrichten, konnten Erfolgreich mit dem Arduino Framework realisiert werden.

5.2.3 Smart-Home Plattform Openhab

Eine Smart-Home Plattform wird benötigt, damit der Benutzer mit dem IOT-Systems interagieren kann. Mit der Openhab Smart-Home-Plattform sind folgende Bedingungen erfüllt:

1. Bietet Schnittstellen mit anderen Smart-Home Technologien wie KNX
2. Benutzerfreundlich
3. Client einsetzbar mittels verschiedenen Betriebssystemen
4. Unabhängig vom öffentlichen Netz (Inhouse Server)

Openhab bietet sehr viele Schnittstellen, da durch den modularen Aufbau der Architektur die flexible Anbindung neuer Technologien einfach ist. Die für dieses Projekt wichtige Bindings von MQTT-Broker und KNX sind somit vorhanden. Was ebenso interessant ist, sind konfigurierbare Features wie Dropbox Support, und Google Kalender.

5.3 Kommunikation

5.3.1 WLAN Konfiguration

Die Funkkommunikation zwischen Sensor, Aktor und MQTT-Broker wird mittels Wireless Local Area Network (WLAN) auf dem OSI layer 1 statt finden. Um die einzelnen Target also Sensoren und Aktoren in das Lokale Netzwerk zu verbinden, wird beim Starten des Mikrocontrollers ein Accesspoint eröffnet. Für den Wifi-Configurations Prozess wird die entsprechende Bibliothek eingebunden. Diese steht auf Github zur Verfügung [12].

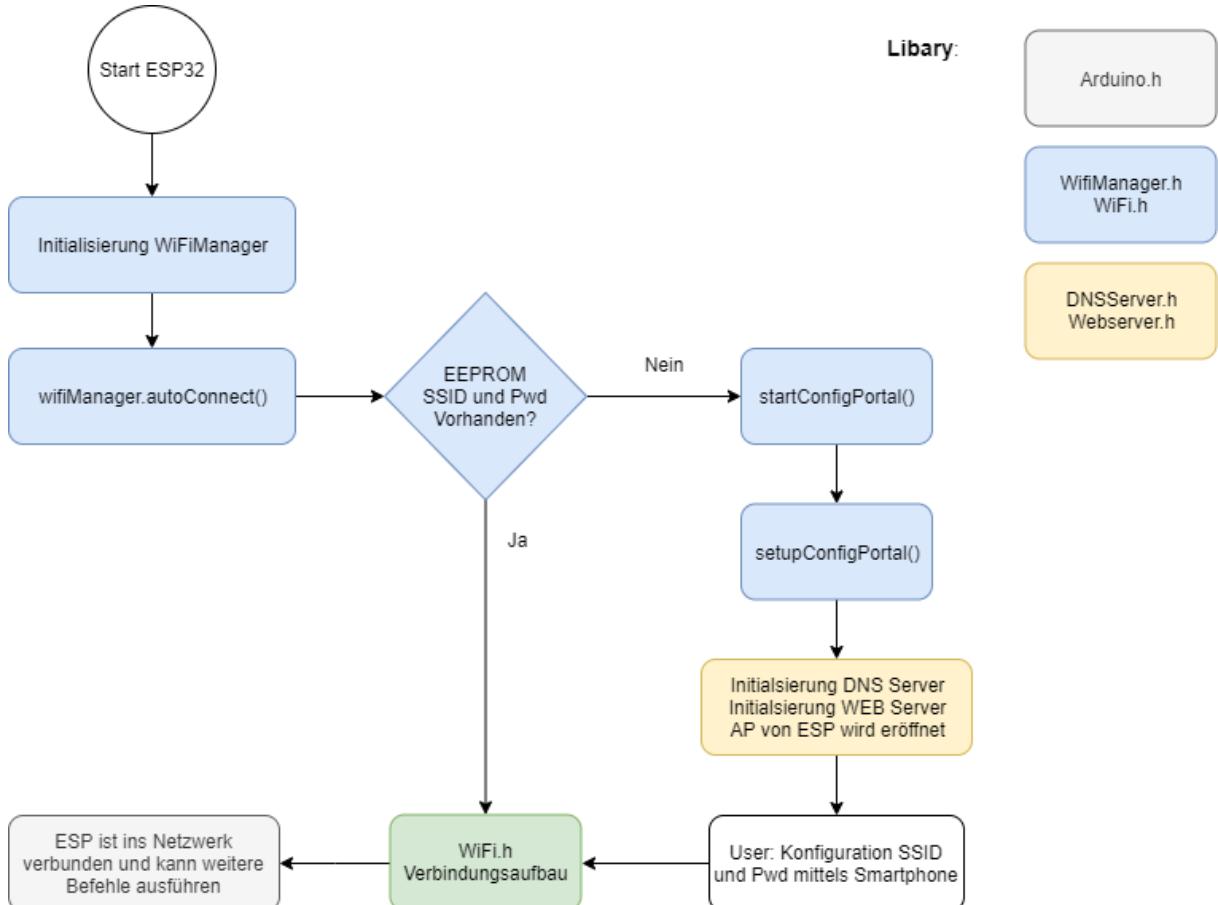


Abbildung 5.3: Statediagram Verbindungsauflaufbau ESP ins Netzwerk mit einem AP

Wird die Bibliothek WiFiManager.h eingebunden und initialisiert stehen verschiedene Funktionen zur Verfügung. Die Funktion autoConnect() eröffnet nach dem Start des Mikrocontrollers einen Access-Point, wenn keine Konfigurationsdaten im nichtflüchtigen Speicher sind. Falls Konfigurationsdaten vorhanden sind, werden sie aus dem EEPROM gelesen und es wird kein Access-Point eröffnet.

Mit einem Gerät beispielsweise Notebook oder Smartphone kann in den Netzwerk Einstellungen der Mikrocontroller mit dem Name 'Aktor' oder 'Sensor' mit anschliessender Chip-ID gefunden werden. Sobald "verbinden" mit diesem Netzwerk gewählt wird, startet der Webbrower eine Konfigurationsseite.

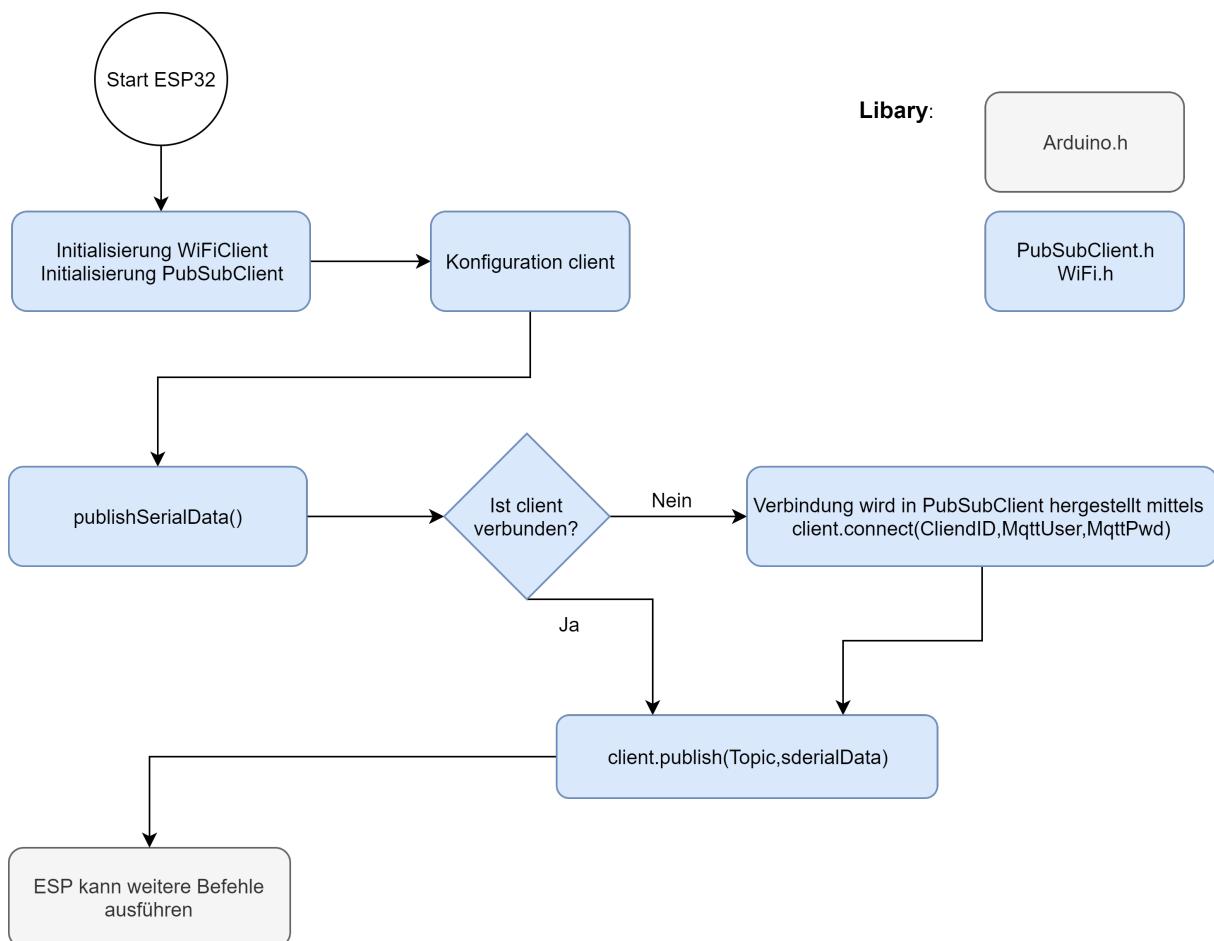
SSID
password
mqtt server
mqtt port
board location

save

[Scan WiFi](#)

Abbildung 5.4: Wifi Konfiguration Portal

Wird 'Scan WiFi' betätigt werden Netzwerke, welche in der Umgebung gefunden wurden, angezeigt, die Parameter die zur WiFi Verbindung notwendig sind, können nun eingegeben werden. Im unteren Teil wird die MQTT-Broker-Adresse und der Port eingegeben. Als 'board-location' soll der Standort des Gerätes eingegeben werden, aus dieser Eingabe werden MQTT-topics generiert.

**Abbildung 5.5:** Esp Info

Die für die MQTT-Kommunikation wird die Libary PubSubClient.h eingebunden. Als erstes wird der WiFiClient initialisiert.

Danach wird der PubSubClinet als Client instanziert, für diesen Vorgang werden folgende Parameter benötigt:

- server: Adresse von MQTT-Server
- port: der Port von dem MQTT-Server
- client: eine Instanz vom Ethernet-Client.

Die Funktion publishSerialData() ermöglicht, eine Nachricht zu veröffentlichen, bei diesem Vorgang wird den Topic und die Payload veröffentlicht. Beim Aufruf dieser Funktion, wird jedes mal überprüft ob die Verbindung zum MQTT-Broker in Ordnung ist.

Ist die Verbindung nicht in Ordnung, wird sie mit der Funktion reconnect() hergestellt.

Ist die Verbindung in Ordnung werden die Daten mit dem Befehl publish() veröffentlicht [13].

5.4 Programmcode Sensorbord

Die Aufgabe welcher der Mikrocontroller vom Sensorboard übernehmen besteht darin, dass er Betätigungen der Touchtasten auf dem Frontprint erkennt. Jeder Touchtaster besitzt ein LED mit dem das Betätigen bestätigt wird. Diese Funktion wurde so initialisiert, damit der User eine sofortige Kenntnis über die ausgelöste Aktion bekommt. Mit dem Sensorboard wird auf dem Frontprint die Umgebungstemperatur gemessen.

5.4.1 Übersicht

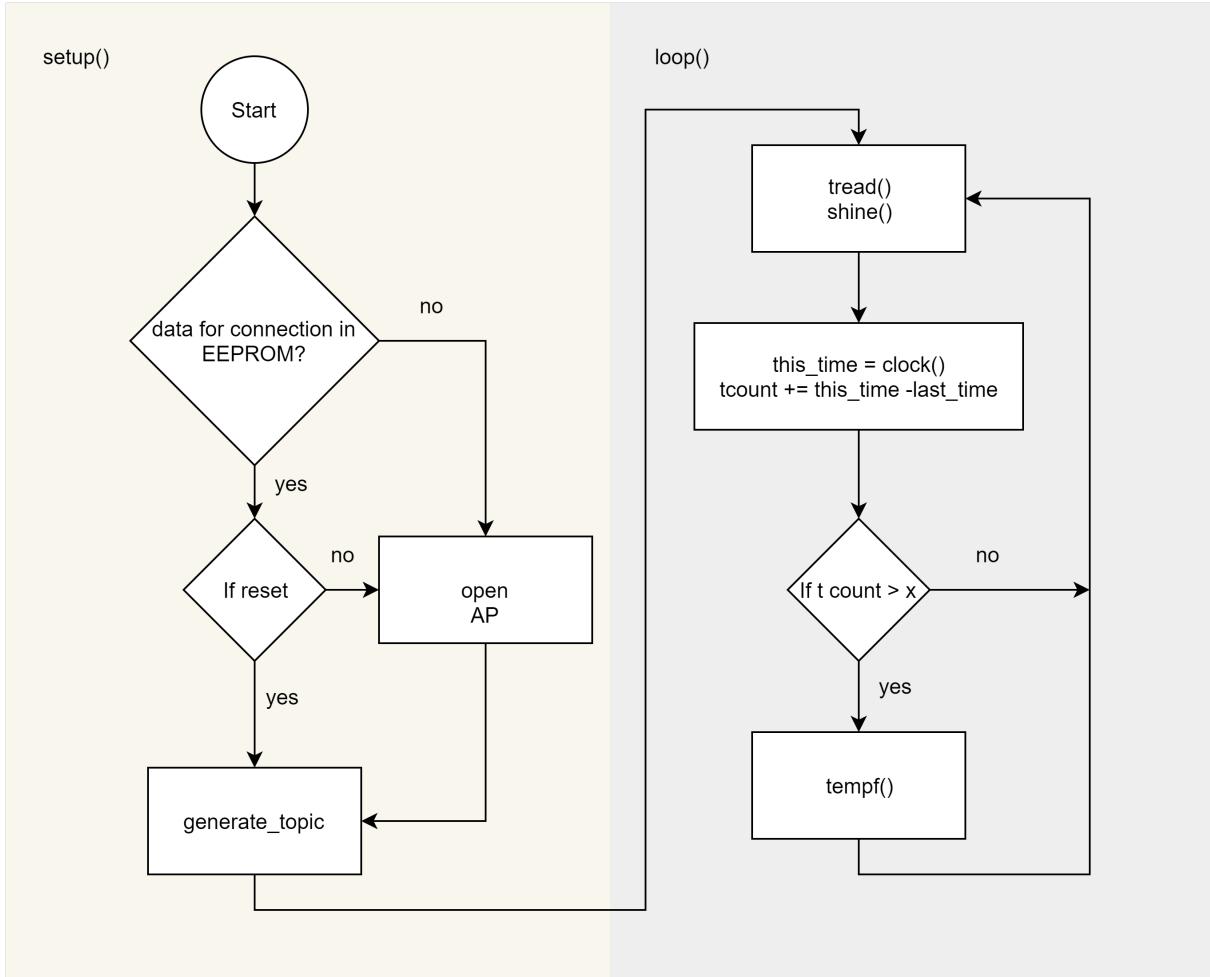


Abbildung 5.6: In dieser Abbildung ist das Statediagramm vom Sensorboard abgebildet

5.4.2 setup()

Wird der Mikrocontroller gestartet so werden als erstes Configurations-Daten im EEPROM gesucht. Werden keine Daten gefunden oder wird die Reset-taste betätigt wird ein Accespoint eröffnet und Configurations Parameter können mittels Browser eingegeben werden. Dieser Vorgang wird im Kapitel Kommunikation genauer beschrieben. Als nächster Schritt werden für die MQTT-Messages die Topics anhand der Bordbezeichnung und der Anwendung generiert. Die Bordbezeichnung wird im Configportal eingegeben und im EEPROM abgespeichert, sie dient dazu, wenn mehrere Boards installiert werden um die empfangenen Nachrichten zu unterscheiden. Eine mögliche Topic für eine MQTT Nachricht wenn beispielsweise der erste Taster betätigt wird kann folgendermassen aussehen: "data/sensorboard/wohnzimmer/S1" in diesem Fall wurde die Bordbezeichnung Wohnzimmer gewählt.

5.4.3 loop()

Im loop() wird als erstes die Funktion tread(), dann shine() aufgerufen, danach wird die Momentane Zeit mit der Funktion clock() geholt. In der Variabel count wird die Zeitdifferenz zwischen den wiederholenden Durchgängen von touch() addiert. Dies passiert solange bis der Wert x erreicht ist, in diesem Fall alle 10 Sekunden. Wenn der Wert x erreicht ist wird die Funktion tempf() aufgerufen, mit dieser wird mittels NTC die Temperatur gemessen. Im gleichen Zyklus wie die Temperaturmessungen Durchgeführt werden, wird die Status LED geschaltet, so kann überprüft werden, ob sich das Bord im normalen Betrieb befindet, sprich der loop() wird gewiss komplett Durchlaufen. Die Funktionen sind in der nachfolgenden Abbildung abgebildet und werden anschliessend erläutert.

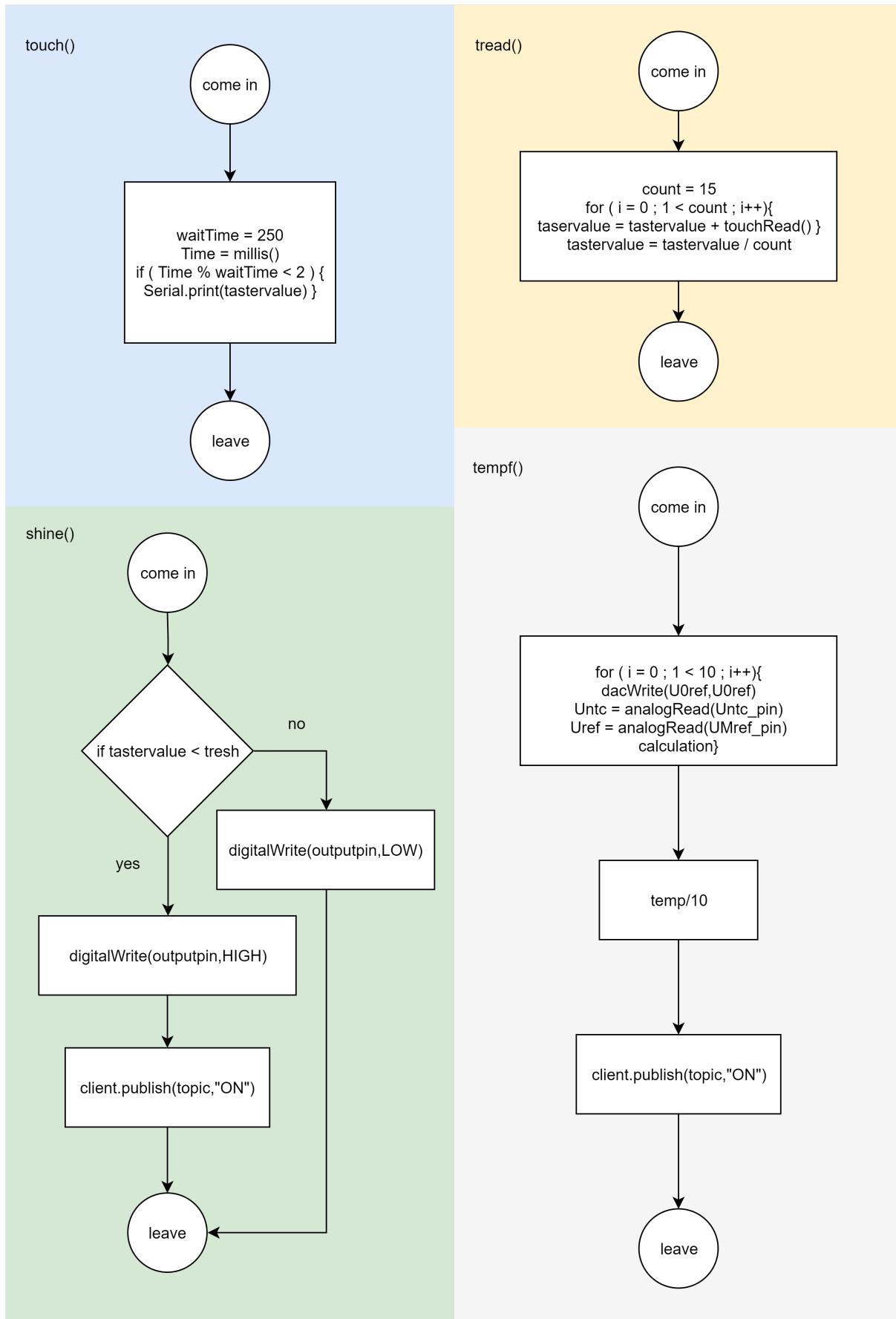


Abbildung 5.7: In dieser Abbildung sind die Funktionen vom Sensorboard abgebildet

5.4.4 touch()

Wie in der Abbildung 5.6 zu erkennen ist, wird die Funktion touch() nicht aufgerufen. Sie dient alleine zum debuggen also um den optimalen thresh Wert zu finden. Dieser Wert definiert ab wann die Touchsensoren als gedrückt erkannt werden. In der Funktion wird die Variablen waitTime initialisiert mit dem Wert 250 weiter wird der Wert millis() in die Variable Time gesetzt. In einer for-wihle werden so lange Time modulo waitTime kleiner als 2 sind, sprich die ersten 2 Millisekunden von 250 Millisekunden, mit Serial.println() der aktuell eingelesene Touchwert ausgegeben. Dieser Wert befindet sich im Ruhezustand um 50-80, wenn eine entsprechende Touch-Taste betätigt wird sinkt dieser Wert auf 10-30.

5.4.5 shine()

In der Funktion shine() wird in einer for-while der Eingelesene Wert das sogenannte taservalue der jeweiligen Touchtasten mit dem Thresholdwert, thresh verglichen. Ist das tastervalue kleiner als thresh, wird eine MQTT-Message mit der entsprechenden Topic und der Payload "ON" published. Ebenso wird die Led der entsprechenden Taste HIGH gesetzt. Trifft der andere Fall zu, wenn das tastervalue grösser als thresh ist wird die entsprechende Led LOW gesetzt.

5.4.6 tread()

In der Funktion tread() werden in einer for-while die aktuellen Werte der vier Touch-Tasten eingelesen und in den int-array tastervalue gesetzt, um Ausreisser zu eliminieren, wird ein Mittelwert erstellt. Es werden jeweils 15 Messungen addiert und anschliessend durch die Anzahl Messungen dividiert.

5.4.7 tempf()

In dieser Funktion wird die Temperatur mit dem NTC ermittelt. So wird als erstes U_{ref} und U_{ntc} als ADC-Wert eingelesen und über 10 Werte gemittelt. Danach wird eine Geradengleichung (Gl.: 5.1) verwendet, um die ADC-Werte in eine Spannung zu konvertieren. Wie im Kapitel 6.1.6 beschrieben und in der Abbildung 6.11 ersichtlich, bildet ein Vorwiderstand von $100\text{ }\Omega$ und der NTC einen Spannungsteiler an dem U_{ref} angelegt wird und am NTC die Spannung U_{ntc} gemessen wird. Aufgrund dessen wird der Widerstand des NTCs R_T mit 5.2 berechnet. Für U_{ref} wird maximal eine Spannung von 2.5 V verwendet, da der ADC danach nicht mehr linear ansteigt, diese Spannung wird mithilfe eines DACs im Setup() ausgegeben. Die Temperatur in Kelvin ergibt sich dann aus der Gleichung 5.3, von dieser Zahl wird danach 273.15 abgezogen, um den Wert in °C zu kriegen. Die angesprochene Gerade (Gl.: 5.1) ist so ausgelegt, dass sie in dem Spannungsbereich der Temperaturmessung funktioniert. Um sie auszulegen und gleichzeitig die Temperatur zu verifizieren wurde folgendes Verfahren angewendet. Als erstes wurde der Temperaturbereich festgelegt, welcher von -5 °C bis +45 °C reicht. Dann wurde mittels eines Klimaschrances 5.9 die Temperaturen in $5\text{ }^{\circ}\text{C} \pm 1\text{ }^{\circ}\text{C}$ Schritten festgelegt. Um die Temperatur genauer zu messen wurde ein NiCr-Ni Temperaturfühlers (Messgerät Therm 2250-1) verwendet. Der Sensorbaustein und der Temperaturfühler werden nun verschiedenen Temperaturen ausgesetzt und der Sensorbaustein gibt seine ADC-Werte weiter (Tabelle: 5.1). Es wurde festgestellt, dass U_{ref} annäherungsweise über die Temperatur konstant bleibt, hier war der ADC Wert 2768 ± 1 . U_{ref} wurde ebenso mittels eines Multimeters gemessen und es ergab sich eine Spannung von $U_{ref} = 2.39\text{ V}$. Die mit dem NiCr-Ni Temperaturfühler gemessenen Temperaturen wurden mithilfe von Gleichung 5.4 und 5.5 in Spannungen verrechnet, welche an dem NTC anliegen, falls die Toleranz klein genug sind. Diese theoretischen Spannungen wurden genommen, entgegen den ADC-Werten gesetzt und einen linearen Fit mit QtiPlot erstellt. Die Gerade (Gl.:

5.1) kam dabei heraus und sie wurde auf die ADC-Werte der Tabelle 5.1 angewendet, um dann mithilfe von den Gleichungen 5.2 und 5.3 die Temperaturen zu berechnen, die der Sensorbaustein schlussendlich anzeigt. Diese berechneten Temperaturen des Sensorbausteins wurden dann den gemessenen Temperaturen des Temperaturfühlers in der Tabelle 5.1 verglichen, die grösste Abweichung war 0.23 K bei 39.2 °C, bei welchen der Sensorbaustein 39.43 °C anzeigen würde. Der Grund dieses Verfahrens ist, dass U_{ref} leicht unterschiedliche Werte annehmen kann, da der DAC nicht genau ist, vereinfacht gesagt wurde im Bereich von $U_{ref} = 2.39$ linearisiert. U_{ref} wird wie U_{ntc} gemessen, was bedeutet dass es keine Rolle spielt ob, $U_{ref} = 2.30$ V oder 2.4 V ist.

Berechnungen zur Temperaturmessung:

$$U = ADC \cdot m_{sensor} + b_{sensor} \quad (5.1)$$

$$\frac{R_T}{R_V} = \frac{U_{ntc}}{U_{ref} - U_{ntc}} \quad (5.2)$$

$$T = \frac{1}{\frac{1}{T_N} + \frac{1}{B} \cdot \ln(\frac{R_T}{R_0})} \hat{=} \frac{1}{\frac{1}{T_N} + \frac{1}{B} \cdot \ln(\frac{R_T}{R_V})} \quad (5.3)$$

$$R_T = R_0 e^{B \left(\frac{1}{T} - \frac{1}{T_0} \right)} \quad (5.4)$$

$$U_{ntc} = \frac{R_T}{(R_V + R_T)} \cdot U_{ref} \quad (5.5)$$

Konstanten zur Temperaturmessung:

$$m_{sensor} = 0.0008155002 \text{ V}$$

$$b_{sensor} = 0.1369856 \text{ V}$$

$$R_V = 100 \text{ k}\Omega \text{ (Vorwiderstand)}$$

$$R_0 = 100 \text{ k}\Omega \text{ (Widerstand des NTCs bei } 25 \text{ }^\circ\text{C})$$

$$B = 4250 \text{ K (B-Konstante des NTCs)}$$

Variablen zur Temperaturmessung:

U_{ntc} (Spannung über NTC)

U_{ref} (Spannung über Vorwiderstand und NTC)

U (Spannung allgemein)

ADC (ADC-Wert allgemein)

T (momentante Temperatur in Kelvin)

Temperatur in °C (Therm 2250-1)	ADC Wert von U_{ntc}	Temperatur in °C (Sensorbaustein)	Abweichungen in °C
-4.6	2254	-4.42	0.18
0.6	2119	0.65	0.05
5.6	1966	5.73	0.13
10.2	1826	9.99	-0.21
14	1687	14.02	0.02
20.2	1474	20.02	-0.18
25.1	1300	24.93	-0.17
29.7	1138	29.64	-0.06
34.3	980	34.50	0.20
39.2	832	39.43	0.23
44	709	43.93	-0.07

Tabelle 5.1: Mit dem Therm 2250-1 aufgenommene Temperaturwerte und die ADC Werte von U_{ntc} , welche zu den Temperaturen des Sensorbausteins berechnet wurden, dazu die Abweichungen der Temperaturen voneinander.

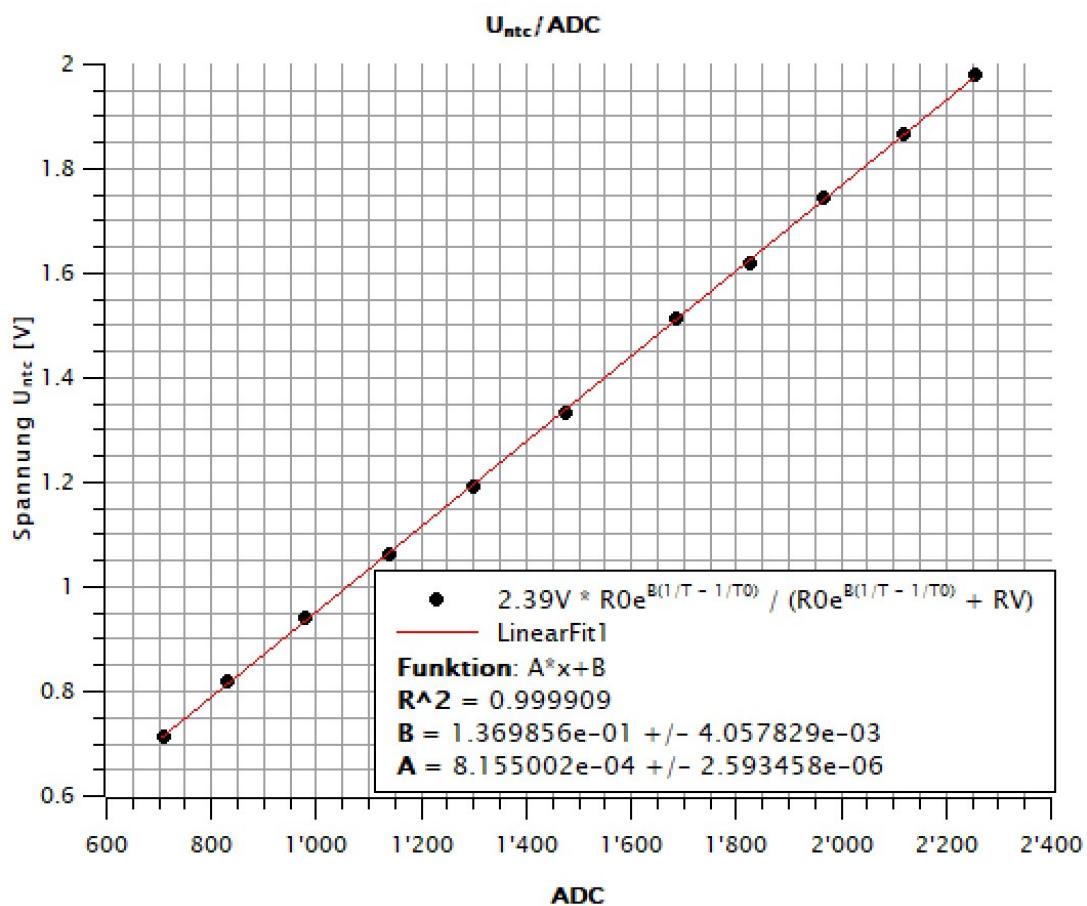


Abbildung 5.8: $U_{ntc,\text{berechnet}}/\text{ADC}$

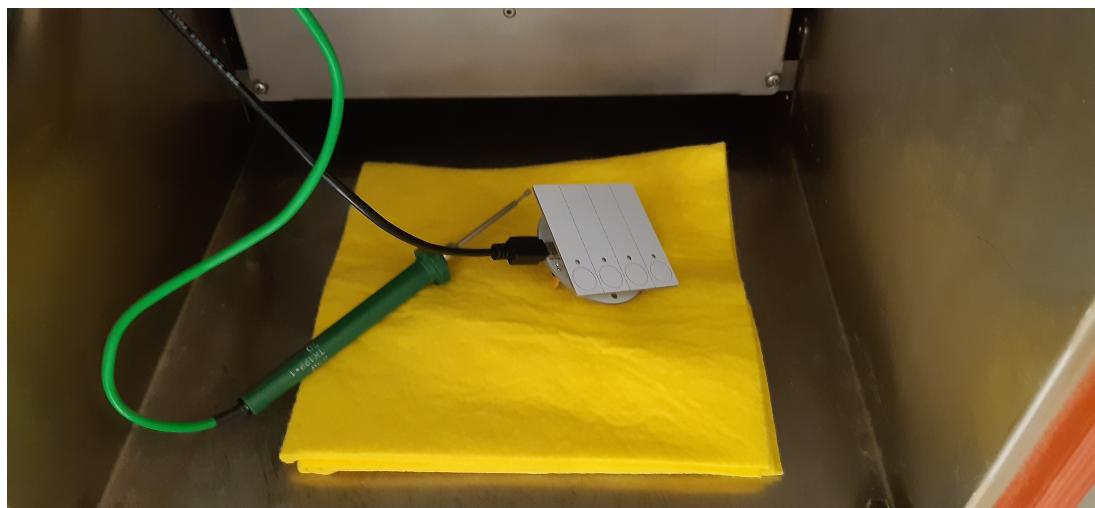


Abbildung 5.9: Temperaturmessung im Klimaschrank

5.5 Programmcode Aktorbord

Die Aufgabe des Aktorbord besteht darin, dass Befehle empfangen, verarbeitet und ausgeführt werden. Es befinden sich vier Relais auf dem Bord, welche 230 Volt schalten können, diese werden mit Digitalen IO Pins angesteuert. Eine weitere Aufgabe ist die Ausgabe von zwei DC-Spannungen 0-10 Volt. Die jeweilige Spannung entsteht durch ein PWM Signal. Eine letzte Aufgabe ist, 0-10 Volt einlesen. Das Eingangssignal gelangt über einen Spannungsteiler an den AD-Wandler. Der Mikrocontroller wertet das Signal aus und generiert eine MQTT-MESSAGE welche in einem Periodischen Zyklus published wird.

5.5.1 Übersicht

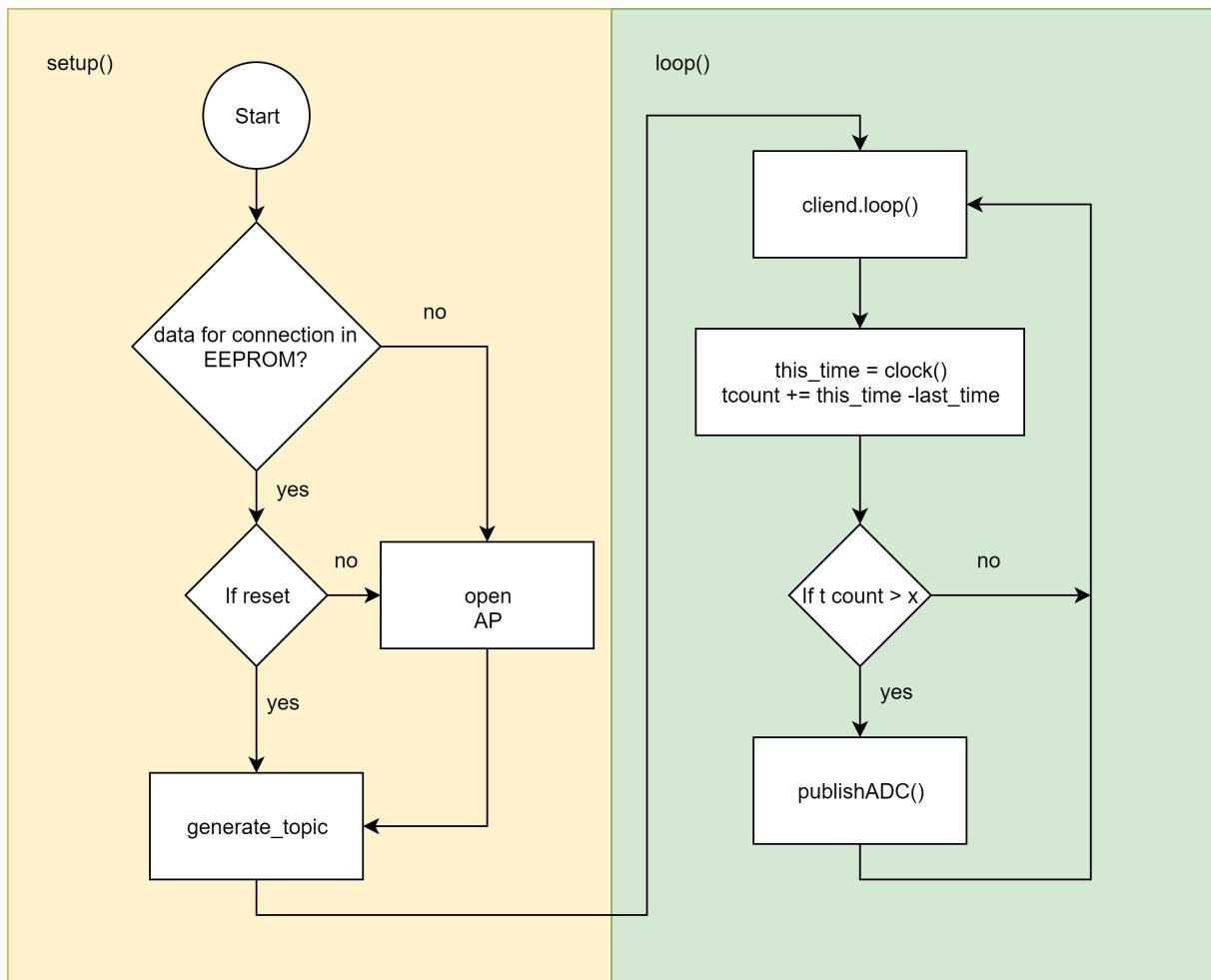


Abbildung 5.10: In dieser Abbildung ist das Statediagramm vom Aktorbord abgebildet

5.5.2 setup()

In der Abbildung ist zu erkennen, dass genau der gleiche Ablauf wie beim Sensorprint 5.4.2 statt findet. Wird aber der Programm Code selber betrachtet werden andere Bezeichnungen wie andere Definitionen von pinMode() usw. im Bezug zum Sensorbord festgestellt werden.

5.5.3 loop()

Im loop() wird als erstes die Funktion clien.loop() aufgerufen. Diese Funktion ist im wesentlichen zuständig, dass MQTT Befehle empfangen und verarbeitet werden, Sie wird anschliessend ausführlich erläutert. In einem weiteren Schritt wird wie im loop() vom Sensorbord beschrieben 5.4.3, Periodischer Zyklus erstellt welche nur alle 10 Sekunden ausgeführt wird. In diesem Zyklus befindet sich die Funktion publishADC, welche die gemessenen Werte von den 0-10 Volt Eingängen als MQTT-Message published.

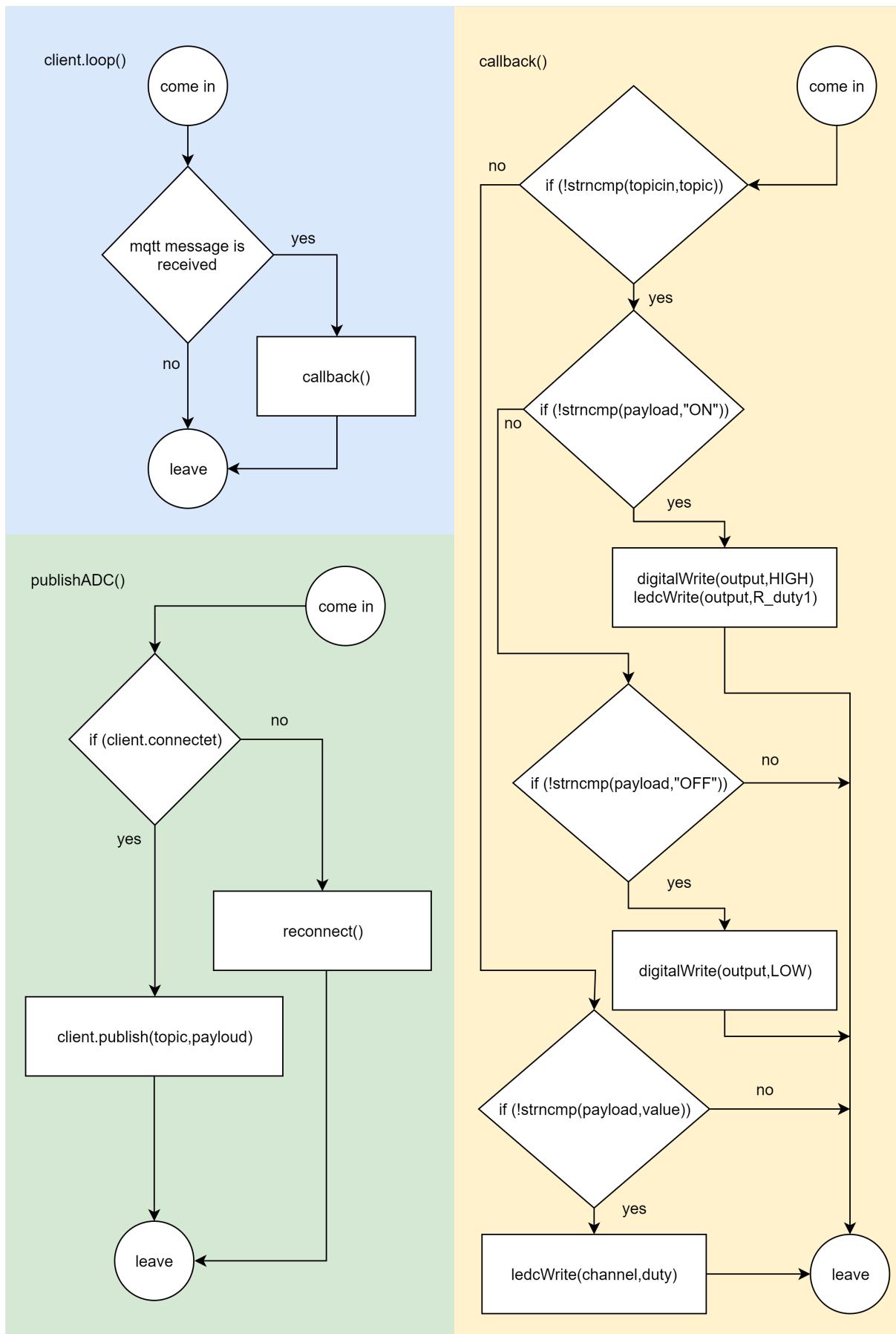


Abbildung 5.11: In dieser Abbildung sind die Funktionen vom Sensorboard abgebildet

5.5.4 client.loop()

Das Objekt *client* wurde aus dem PubSubClient gebaut, wo der `loop()` implementiert wurde. Als erstes wird die Verbindung zum MQTT-Broker überprüft, dann bei Bedarf die MQTT-Nachrichten empfangen und vorbereitet, dass sie in der Funktion `callback()` genutzt werden kann. `callback()` wird aufgerufen.

5.5.5 callback()

In dieser Funktion wird als erstes der Inhalt von der empfangenen Topic überprüft, fällt der Vergleich positiv aus wird der Inhalt der Payload überprüft. Die vier verschiedenen Relais und die zwei Analogausgänge unterscheiden sich an Hand des Topics. Bei den Relais ist die Payload ON oder OFF und das Relais wie auch das LED des entsprechenden Relais wird in den geforderten Zustand gesetzt. Trifft die Topic auf ein Analog Ausgang wird nach Umwandlung der Payload von einem String in ein Float, das PWM-Signal für den entsprechenden Ausgang gesetzt. Den Duty cycle für das PWM-Signal wird berechnet indem die gewünschte Spannung voltage folgender Massen verrechnet wird : $(voltage/3.2 \cdot 4095)/3.25$. 3.2 sind es, weil die Verstärkung 3.2 beträgt und 3.25, weil das die Amplitude des PWM-Signales ist. Leider kann die Amplitude abweichen und ist nicht sehr genau. Mehr zur Genauigkeit in der Validierung.

5.5.6 publishADC()

Mit dieser Funktion werden die 0-10 Volt Eingänge ausgewertet und Resultate als MQTT-Nachricht versendet. Der jeweiligen Messreihe am ADC wird der Mittelwert berechnet, welcher dann von einem float in ein char-Array umgewandelt wird. Als nächstes wird der char-Array und die entsprechende Topic vom PubSubClient veröffentlicht.

5.6 Task2code()

Diese Funktion wird im Gegensatz zum Rest des Programms im zweiten Kernel des ESP32 verwendet. Falls ein Relais in der `Callback()` Funktion gesetzt wird, wird in dieser Funktion die Zeit gemessen und nach 25 ms wird der Sparmodus für das jeweilige Relais gesetzt. Das Setzen des Sparmodus passiert für alle Relais einzeln, es spielt keine Rolle wann ein Relais gesetzt wird, es dauert immer 25 ms $\pm 3\text{ ms}$. Der zweite Kernel ist somit nur damit beschäftigt, zu prüfen ob ein Relais aktiviert wurde und es dann nach einer vordefinierten Zeit in den Sparmodus zu bringen.

5.7 Programmcode Openhab

Der Aufbau des Systems besteht aus verschiedenen Komponenten welche in nachfolgender Tabelle aufgelistet sind

Bezeichnung	Beschreibung
Bindings	Schnittstellen verknüpft verschiedene Dienste miteinander
Things	Definition von Gerät, Verbraucher, Teilnehmer des Systems
Channel	Verbindung zwischen Thing und Item
Item	Repräsentiert Informationen des Gerätes, Schalter, Label usw.
Rules	Festgelegte Regeln für automatische Abläufe
Sitemaps	Benutzeroberfläche, präsentiert Informationen

Tabelle 5.2: Komponenten Openhab Software [14]

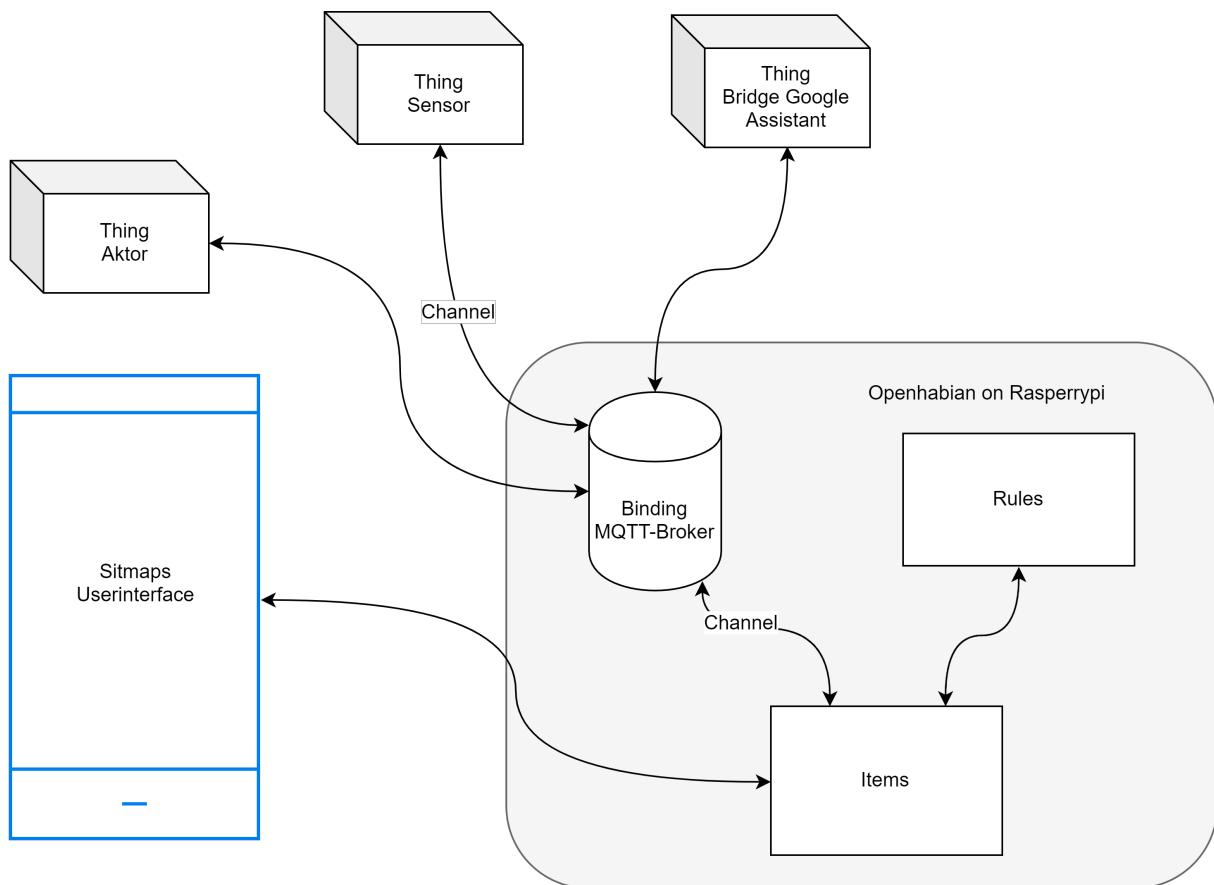


Abbildung 5.12: Komponenten und Verbindungen von Openhab

5.7.1 Bindings

Als Schnittstelle von den Items zu den Things wird ein MQTT-Broker benötigt. Um die Systemzuverlässigkeit zu steigern wurde ein eigener Broker installiert, so kann die Kommunikation auch ohne Netzwerkanschluss ans Öffentliche Netz funktionieren. Ein weiterer Vorteil ist die Sicherheit, es kann auf Passwörter verzichtet werden, da kein Zugriff von ausserhalb des Home-Netzwerk möglich ist.

5.7.2 Things

Als Geräte wurden drei Komponenten hinzugefügt, welche in der Abbildung 5.12 zu erkennen sind. Zum Aktor Thing gelangen im ganzen 8 Channels, um die Relais einzeln anzusprechen wurde pro Relais eine MQTT-Topic erstellt welche je einen Channel verfügt und so mit je einem Item verbunden. Zwei Channel sind für die 0-10 Volt Eingänge und zwei weiter Channel für die 0-10 Volt Ausgänge. Das Sensor Thing verfügt 5 Channel jeder Taster einzeln und ein Channel für die Temperaturmessung.

5.7.3 Channel

Die Channes sind die Verbindungen von den Geräten zu den Items also den Software-Zustände, in diesem Projekt werden alle möglichen Fähigkeiten der Geräte genutzt und mit einem Channel gebunden.

5.7.4 Item

Das Item zeigt den Status des Verbrauchers oder den gemessenen Wert eines Sensors an. Items werden verwendet wenn Regeln definiert werden, ebenso sind sie mit den Channels verlinkt und können einzeln oder als Gruppen auf dem Sitmap genutzt werden. Als anwendung eines Icons können verschiedene Typen gewählt werden. In diesm Projekt wird in erste Linie der normale Switch verwendet welcher in den Zustand ON oder OFF geschalten wird. Für die 0-10 Volt ausgänge werden Dimmer verwendet die als Slider in der Sitmap angezeigt wird und Sie generieren je nach Position einen Wert zwischen 0-1. Um die Messwerte an den 0-10 Volt Eingängen zu verwenden werden die Items als Number genutzt.

5.7.5 Rules

Automatische Prozesse werden mit i Rules definiert. Um das System Benutzerfreundlicher zu Gestalten werden in den Rules die Befehle der Taster verarbeitet und die Entsprechenden Befehle an die Relais generiert. So kann gewährleistet werden, dass der Kunde selber keine Änderungen an den Aktoren und Sensoren selber vornehmen muss. Die Rule Syntax basiert auf Xbase [15]. Um die Aufgaben der Befehls Weiterleitung zu übernehmen bestehen die Ruls im wesentlichen aus when und then. In diesem Fall sind sie im Teil when, auf den Empfang des jeweiligen Update getriggert. Somit wird mit einem Update des Status der Teil then ausgelöst, wo sich eine if Bedingung befinden. In diese Bedingung wird der Stus überprüft

5.7.6 Sitemaps

Als Sitemaps wird das Userinterface bezeichnet. Auf einer Sitemap gibt es die Möglichkeit verschiedene Elemente in Form von Frame, Text oder Schalter darzustellen, diese Elemente präsentieren Status und Informationen vom System. Das Frame ist Beispielsweise die Aufteilung von einem Haus in seine einzelnen Stockwerke. In den einzelnen Stockwerken werden dann Schalter für jeweilige Geräte als Item hinzugefügt. Mehrere Schalter können als Gruppe zusammen gefasst werden, so kann in zentral Schalter ein/aus realisiert werden. Das erstellte Sitmap ist im Webbrower mittels IP-Adresse des Servers oder mit dem Openhab Mobile-App erreichbar.

5.8 Server

Als Server für die Openhab Software wird ein Raspberrypi der 4. Generation mit 2 GB RAM eingesetzt, die Installation und Inbetriebnahme wird im Benutzerhandbuch beschrieben. In Openhab wird ebenfalls ein eigener MQTT-Broker eingebunden. Damit diese Broker von jedem Gerät

im Lokalen Netzwerk erreicht werden kann, wird dem Server eine Statische IP Adresse vergeben.

Um den Sprachassistent einzubinden ist ein weiterer Server mit der Home Assistant Software in das System integriert worden. Dieser Server wurde auf ein Raspberry Pi 4. Generation 2 GB RAM installiert. Der Grund warum nicht beide Server auf dem selben Raspberry Pi installiert wurden, ist einerseits, das System ist physikalisch modular aufgebaut der Sprach Assistent Teil kann als separates Thing (Gerät) betrachtet werden. Andererseits, wenn sich beide Server auf dem Selben Gerät befinden wäre die MQTT-Kommunikation sinnlos.

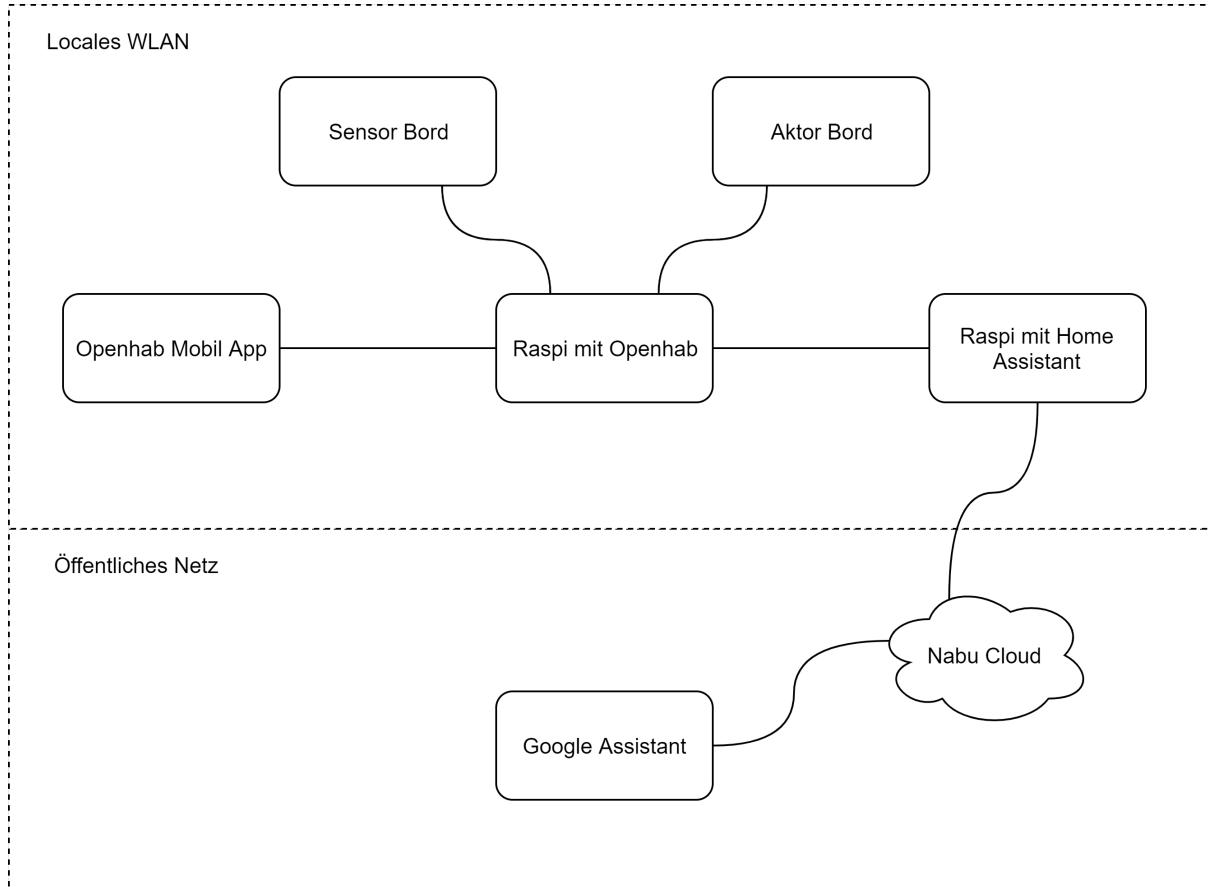


Abbildung 5.13: Systemübersicht

In der Abbildung 5.13 kann erkannt werden, dass sich beide Server im selben Lokalen Netzwerk befinden. Die Kommunikation zwischen den beiden Servern findet über MQTT statt, es besteht die Möglichkeit den Openhab-Server zu umgehen und eine direkte Kommunikation vom Home Assistant zu dem Sensor- oder zum Aktorbord zu realisieren. In diesem Fall müssen aber die Regeln bei welchem Aktion, welche Schalthandlung ausgelöst wird, in den entsprechenden Mikrocontroller Programmiert werden.

5.9 Sprachassistent

Der Home Assistant wird als Brücke für den Google Assistant installiert. Mit der Home Assistant Cloud wird eine Verbindung zum Google Assistant hergestellt. Der Grund warum die Verbindung so aufgebaut wird liegt daran, dass dynamische DNS Adresse, SSL-Zertifikate und das öffnen von Ports auf dem Router so umgangen werden. Leider ist die Cloud nach einer Testphase kostenpflichtig. Wird die Lösung ohne Cloud bevorzugt kann nach Anleitung [16]

gearbeitet werden. Die Cloud ist im /config/configuration.yaml file schon vorinstalliert. Im Webinterface vom Home Assistant kann in den Einstellungen ein Benutzerkonto angelegt werden und schon ist die Cloud aktiv. Das hinzufügen von Schalter um Mqtt-Befehle zu generieren wird im Benutzerhandbuch im Anhang beschrieben. Die Installation auf dem Google Nest, mit einem Smartphone durchgeführt. Mittels Google Home App kann das Gerät 'Home Assistant' hinzugefügt werden.

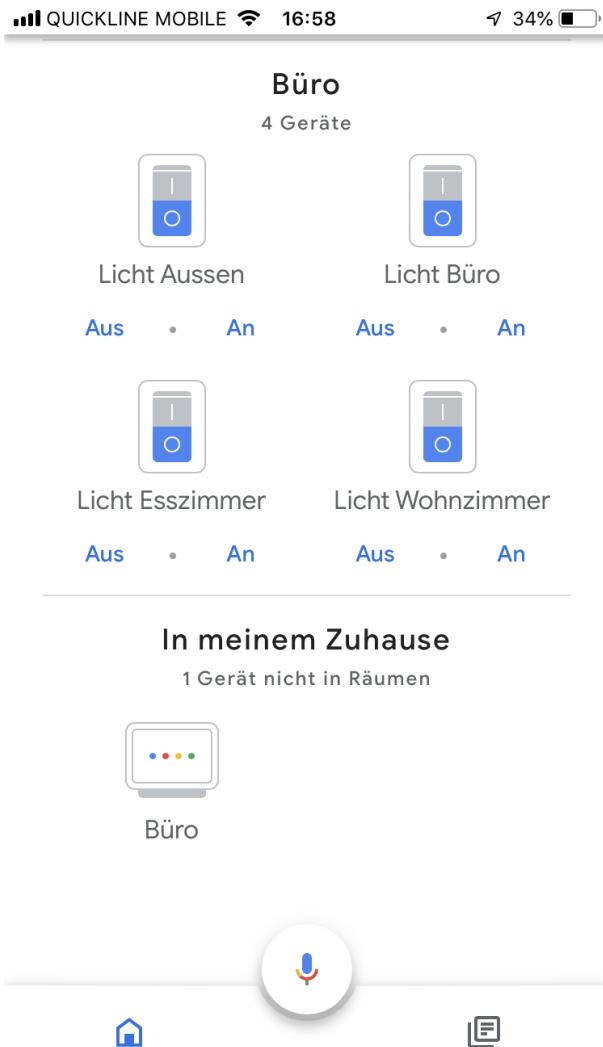


Abbildung 5.14: Google Assistant Mobile-App

In der Abbildung 5.14 können die Schalter, welche Home Assistant definiert wurden erkannt werden. Sie können mit Sprachbefehl geschalten werden in dem die Schalter Bezeichnung und Zustand genannt wird wie Beispielsweise "Licht Büro ein". Die Verschiedenen Geräte können nebst dem Sprachbefehl in dieser Ansicht mit einem Touch Befehl geschalten werden.

6 Hardware

6.1 Sensorbaustein

In diesem Unterkapitel werden die Anforderungen und die Hardware des Sensorbausteins behandelt. Vom Auftraggeber ist ein Sensorbaustein mit 4-Tasten-Feld und Temperaturfühler gefordert. Der Sensorbaustein wird dabei ähnlich einer Unterputzsteckdose montiert und hat eine WLAN-Schnittstelle um die Sensordaten auszulesen. In der Abbildung 6.1 ist die Übersicht des Sensorbausteins dargestellt. Der Baustein wird über ein AC/DC Wandler der an der Hauselektrik angeschlossen werden kann mit 5V Spannung versorgt. Da der Mikrocontroller auf 3.3V läuft, wird mithilfe eines weiteren Spannungswandlers die 5 V zu 3.3 V gewandelt, so kann auch der USB Anschluss als Spannungsversorgung dienen. Um den Mikrocontroller zu programmieren steht eine USB zu UART Verbindung, Taster zum Debuggen, Reseten oder Programmieren und eine Zustands-LED zur Verfügung. Um die Temperatur zu erfassen wird ein NTC verwendet. Da die mitgelieferte Meander-Antenne des ESP32 nur eine kleine Abstrahlleistung hat, kann optional noch eine grössere Antenne, wie z.B. eine zertifizierte grössere Patch-Antenne über RP-SMA angeschlossen werden, falls der ESP32S ausgewählt wird. Der Anwender wird ein Touch-Tastenfeld, sowie dazugehörige LEDs, für die Interaktion mit dem Baustein zur Verfügung haben. Zusammenfassend besteht der Sensorbaustein im wesentlichen aus 3 Teilen, nämlich der Spannungsversorgung, der Hauptplatine und der Frontplatte.

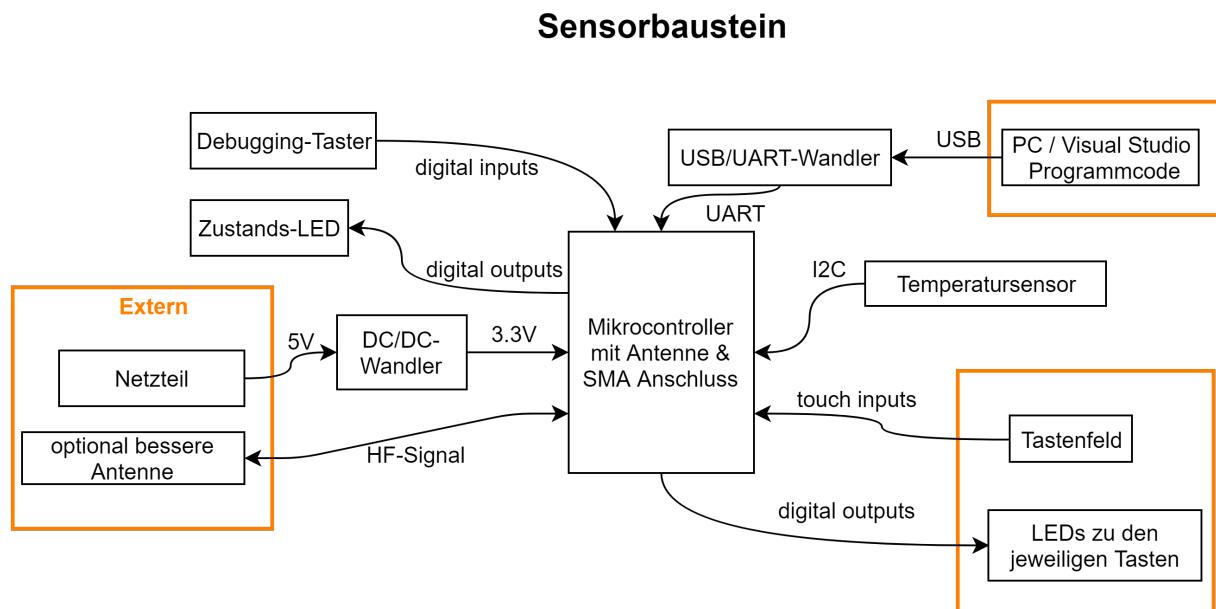


Abbildung 6.1: Übersicht Sensorbaustein

6.1.1 Übersicht Frontplatte

Die Frontplatte besteht aus einem 2-Lagen Print. Auf der Vorderseite (Abb. 6.2) sind vier runde Kupferflächen angebracht, welche als Touch Fläche fungieren. Wenn nun ein Anwender eine dieser Fläche berührt, wird die Kapazität zwischen dieser Fläche und Ground vergrössert. Natürlich befindet sich ein Lötstopplack zwischen Finger und Kupferfläche der Dielektrikum darstellt. Der ESP32 hat intern Touchsensoren verbaut, welches ein Signal V_{refh} auf auf die Pins geben (Abbildung: 6.6). Die vorhandene Kapazität im ESP32 wird sich nun nach einer Zeit mit V_{refh} aufladen, falls die Spannung über der Kapazität V_{refh} erreicht hat, wird das tiefere Signal V_{refl} ausgegeben. Wenn es die Spannung nun wiederum V_{refl} erreicht hat wird wieder

V_{refh} ausgegeben und so weiter, so bildet sich am Touch-Pin ein Dreiecksignal. Falls nun am Touchpin die Kapazität durch berühren vergrössert wird, wird es länger dauern bis sich die gesamte Kapazität aufladet und die Frequenz des Dreiecksignals nimmt ab, bei einem Test mit dem Sensorbaustein nahm die Frequenz von 11 kHz auf 5 kHz ab. Die Periodendauer wird vom ESP gemessen und ein zur Frequenz proportionales Signal ausgegeben. Zu den Tasten gibt es jeweils eine LED, welche von der Rückseite montiert werden. Der NTC ist auf der Rückseite der Frontplatte angebracht. Um die Temperatur der tatsächlichen Raumluft zu messen wurde auf der Frontplatte eine Kupferfläche auf der Aussenseite angebracht, welche die Wärme via Vias zum NTC weitergibt. Als Verbindung zur Hauptplatine des Sensorbausteins ist ein 12-Pin Stecker mit 2.54 mm Rastermaß vorgesehen. Eine Randnotiz: Es wurde darauf geachtet, dass keine visuell störenden Vias, das optische Erscheinungsbild beeinträchtigen.

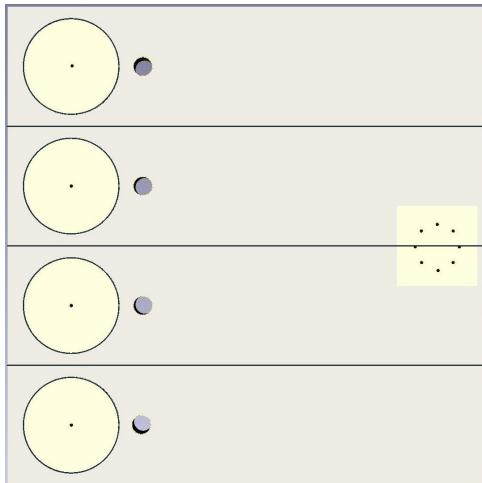


Abbildung 6.2: Frontplatte Vordersicht

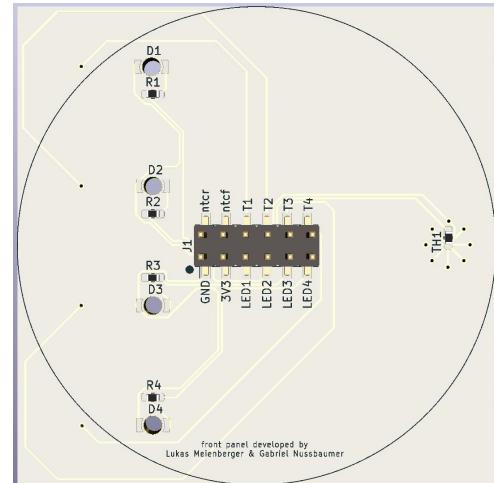


Abbildung 6.3: Frontplatte Rückansicht

6.1.2 Übersicht Hauptplatine

Auf der Vorderseite der Hauptplatine des Sensorbausteins (Abb. 6.4) befindet sich der ESP32, der Programmieranschluss über USB und den 5V/3V Wandler. Über die Buchse kann dann die Frontplatte angeschlossen werden, wobei auf die Richtung geschaut werden muss. Auf der Rückseite (Abb. 6.3) sind Shottky-Dioden, welche bei einem allfälligen ESD auf den Touchflächen greifen. Ebenso befinden sich dort Buttons, eine Status LED sowie 5V und RX/TX Anschlüsse. Randnotiz: Es wurde darauf geachtet, dass die Leiterbahnen zu den Dioden kürzer sind als, die zu den ADC Eingängen.

6.1.3 Übersicht Spannungsversorgung

Die Spannungsversorgung des Sensorbausteins besteht aus einem AC/DC-Leistungsmodul (IRM-02-5) mit einer Leistung von 2 W. Die Eingangsspannung kann 85 VAC bis 305 VAC oder 120 VDC to 430 VDC betragen, ausgegeben werden 5 VDC mit maximal 400 mA.

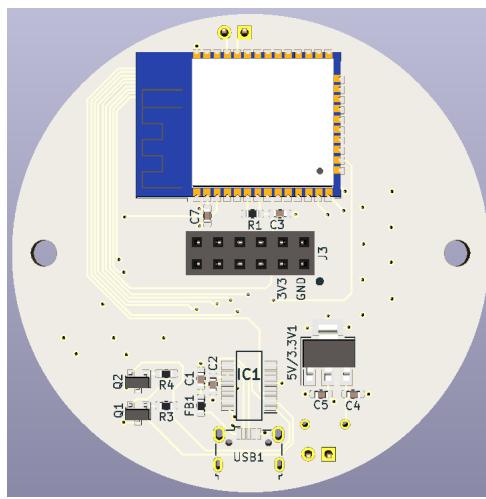


Abbildung 6.4: Hauptplatine Vorderansicht

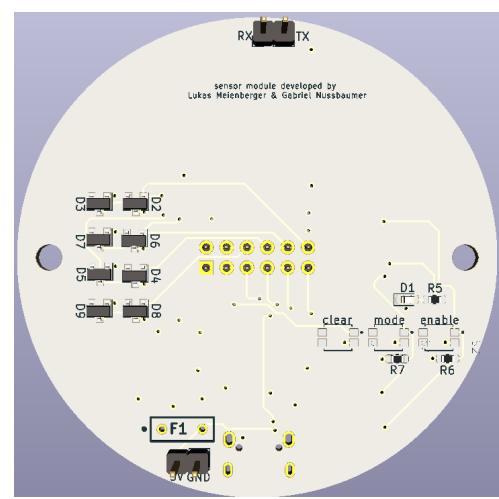


Abbildung 6.5: Hauptplatine Rückansicht

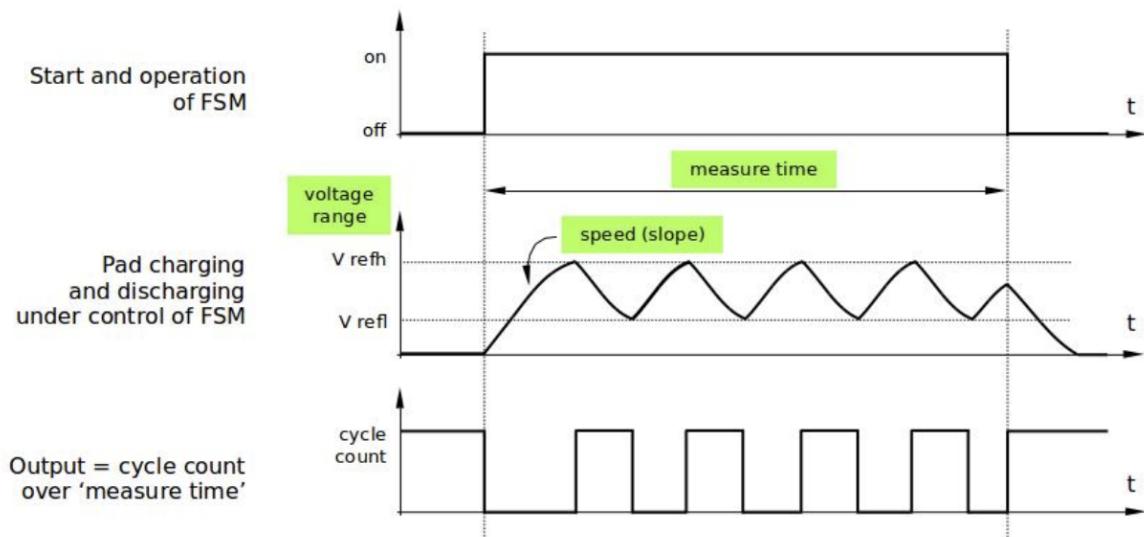


Abbildung 6.6: Funktion des Touches [17]

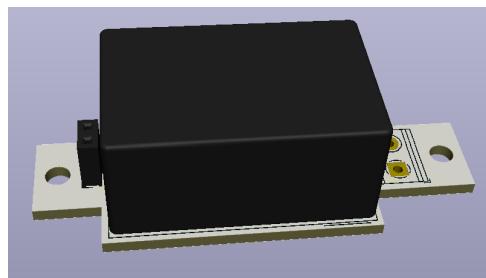


Abbildung 6.7: Übersicht Sensorbaustein

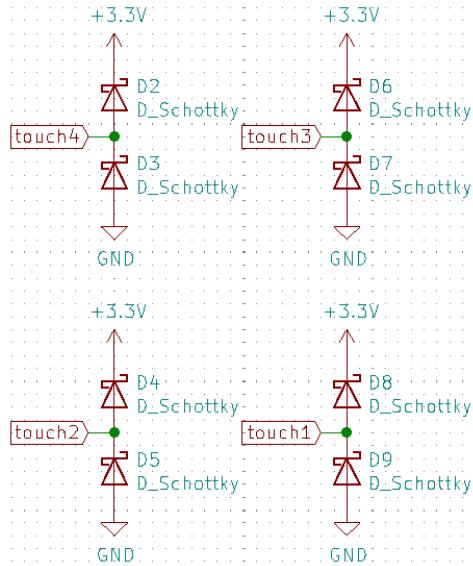


Abbildung 6.8: Shottky Dioden für ESD Schutz

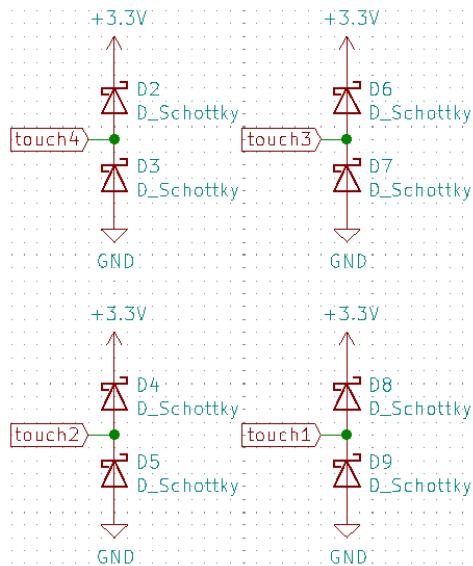


Abbildung 6.9: Shottky Dioden für ESD Schutz

6.1.4 Schutzbeschaltung

6.1.5 Mikrocontroller

Um die Datenkommunikation über WLAN bereitzustellen wird entweder ein ESP32-WROOM oder der ESP32S, welcher im gleichen Formfaktor zur integrierten Antenne auch einen SMA Anschluss bietet, verwendet. Der ESP32 unterstützt Wi-Fi im Bereich von 2.4 GHz bis ca. 2.5 GHz [18]. Aus folgender Übersicht sind die zur Verfügung gestellten Ein-/Ausgänge des ESP32 zu entnehmen:

Der ESP32 verfügt über alle benötigten Schnittstellen, die für den Sensorbaustein relevant sind. Im folgenden (Tabelle 6.2) sind die Inputs/Outputs des Mikrocontrollers (folglich Abb. 6.10), welcher im Sensorbausteins verwendet wird, aufgeführt:

Name	Anzahl
Analog-to-Digital Converter (ADC) channels	18
SPI interfaces	3
UART interfaces	3
I2C interfaces	2
PWM output channels	16
Digital-to-Analog Converters (DAC)	2
I2S interfaces	2
Capacitive sensing GPIOs	10

Tabelle 6.1: I/O Uebersicht

Name	Anzahl
digital outputs für LEDs	5
digital inputs für Buttons	3
Capacitive sensing GPIOs für Touch Buttons	4
UART interface für Programmierung	3
I2C interfaces für Temperatursensor	3

Tabelle 6.2: I/O Sensorbaustein

6.1.6 Temperatursensor

Es wird der NTC NCU18WF104J60RB von Murata Electronics verwendet. Dieser hat den preislichen Vorteil gegenüber anderen Varianten, wie ein Temperaturfühler IC. Der verwendete NTC hat eine B-Konstante von 4250 K im Bereich 25/50 °C mit einer Toleranz von 2%. Der Widerstandswert des NTCs bei 25 °C beträgt 100 K mit einer Toleranz von 5%. Um die Temperatur zu ermitteln wird ein Spannungsteiler (Abb. 6.11) der aus einem 100 K ω Widerstand und dem NTC besteht verwendet. Das Signal ntc_{ref} gibt eine Referenzspannung U_{ref} mithilfe eines DACs aus, da der DAC ungenau ist, wird dieses Signal mithilfe von ntck eingelesen. Die Spannung über dem NTC U_{ntc} wird mit ntcf gemessen und ntcr ist mit Ground verbunden, welches eine separate Rückleitung zur Hauptplatine des Sensorbausteins verfügt, um allfällige Störeinflüsse zu vermeiden.

6.1.7 Programmierananschluss

Der Mikrocontroller kann mittels eines USB to UART Wandlers programmiert werden. Hierzu wird der FT231XS-U verwendet. In Abbildung 6.12 ist das Schema abgebildet. Auf dem Schema sind dabei zwei Bipolartransistoren zu erkennen, welche es erlauben den ESP32 ohne drücken des Mode- und Enabletasters in den Programmiermodus zu versetzen. Falls es zu Komplikationen kommt steht aber auch die Option mittels des Enable- und Modetaster (Abb. 6.13) sich in den Programmiermodus zu versetzen offen. Dazu muss der Modetaster (auch während des ganzen Flashprozesses) gedrückt bleiben, dann mit dem Enabletaster mittels eines kurzen Betätigens den Reset ausgeführt werden und danach kann geflasht werden. Im Aktorbaustein befindet sich die gleiche Schaltung wieder. Falls man einen Systemreset machen möchte kann man den Clear

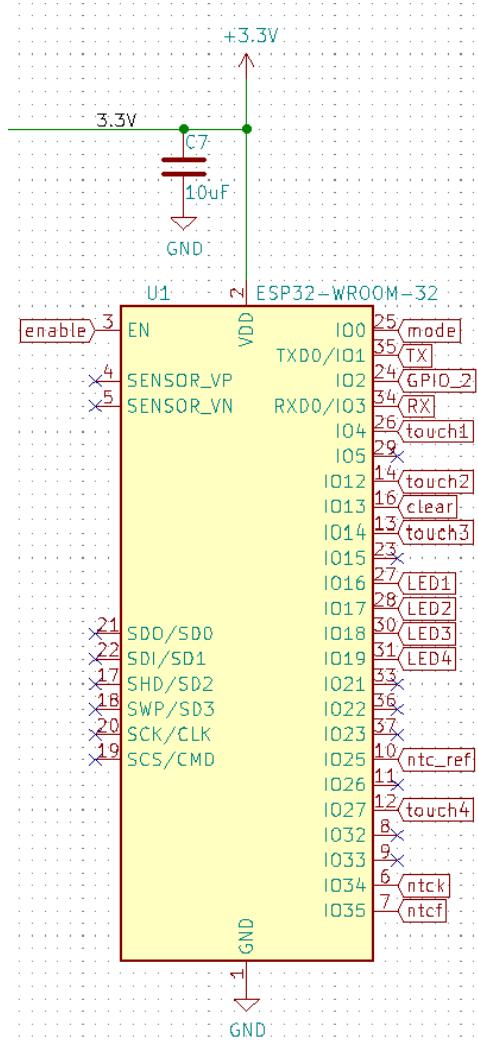


Abbildung 6.10: Mikrocontroller ESP32 im Sensorbaustein

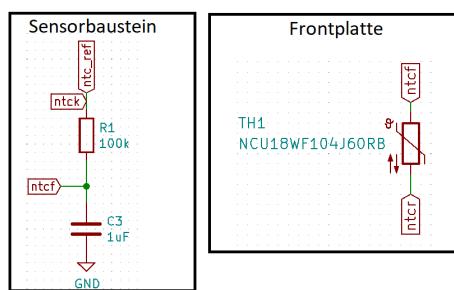


Abbildung 6.11: Schaltung Temperatursensor

Button betätigen. Die LED5 zeigt mittels eines kurzen Blinkens das Aufstarten des ESP32 an oder dass der ESP32 im Programmiermodus ist.

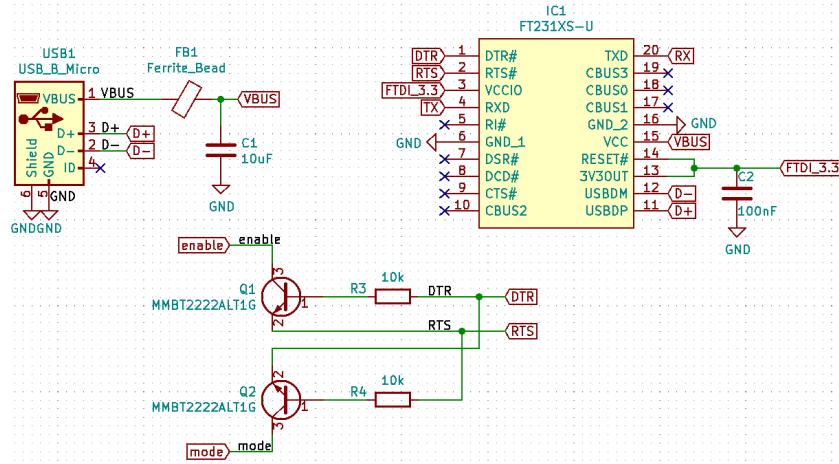


Abbildung 6.12: USB Anschluss

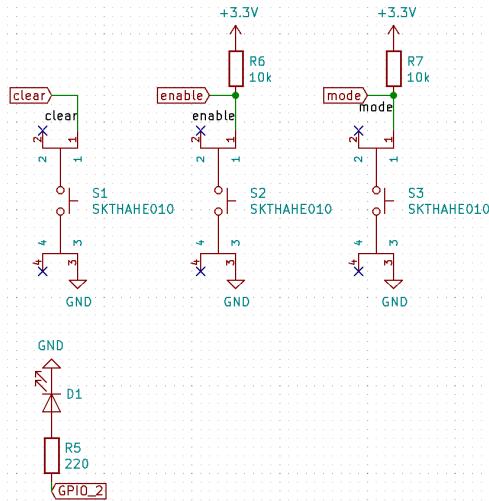


Abbildung 6.13: Buttons und LED auf Sensorbaustein

6.1.8 Spannungsversorgung

Der minimale Strom, welche die Spannungsquelle zur Verfügung stellen muss, ist laut Datenblatt vom ESP32 500 mA, wobei im Normalbetrieb 80 mA und im WiFi Sendebetrieb 160 mA bis 260 mA gebraucht werden. Um den Sensorbaustein mit 5 V zu versorgen, macht es aus Platz gründen Sinn den 230 VAC zu 5 VDC Wandler separat und nicht auf der Hauptplatine zu haben. Dazu wurde der IRM-02-5 verwendet, welcher maximal 2 W zur Verfügung stellt. Der Mikrocontroller benötigt 3.3 V, jedoch soll der Sensorbaustein mit 5V betrieben werden können. Hierfür wird einen Spannungswandler eingesetzt. Der ausgewählte AP1117S33CTR hat einen maximale Input Spannung von 15V, eine maximale Dropoutspannung von 1.2 V bei 800 mA. Im Schema (Abb. 6.14) erkennt man, dass sowohl über USB wie auch über einen Stecker den Print mit 5 V versorgt werden können. Eingebaut ist eine PTC Sicherung sowie eine Diode die verhindern sollen, dass bei einem Kurzschluss oder ähnlichem der Sensorbaustein anfängt grossen Schaden zu nehmen.

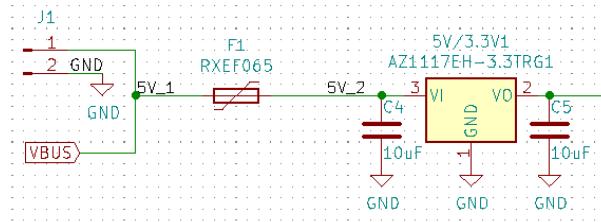


Abbildung 6.14: Spannungswandler Sensorbaustein

6.1.9 Platzstudie

Nun wird mittels einer Platzstudie überprüft ob der Sensorbaustein, mit den verwendeten Komponenten, wie eine Unterputzsteckdose eingebaut werden kann. Im Bild 6.15 ist eine Montageplatte für UP-Steckdosen abgebildet. Der blaue Kreis Zeigt dabei den Radius von 60 mm des Tasters bzw. Kleinkombination. Diese Massen wurden für die Dimensionen der Platine in Abbildung 6.16 übernommen. In Abbildung 6.17 ist zu erkennen wie der Sensorbaustein montiert wird. So wird die Spannungsversorgung mittels Distanzhalter an der Hauptplatine befestigt, welche wiederum an der Montageplatte angeschraubt wird. Die Fronplatte wird mit dem weissen Rahmen an die Hauptplatine gesteckt. Die Richtung des Steckers muss dabei beachtet werden, dazu dient der Markierungspunkt neben Stecker und Buchse. Die Abbildungen 6.18, 6.19 und 6.20 zeigen, wie der Sensorbaustein montiert aussieht.

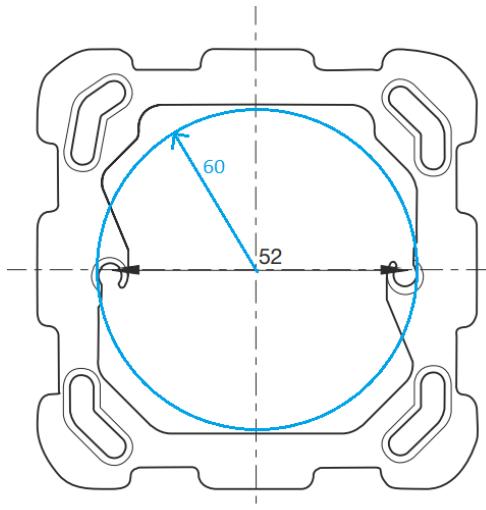


Abbildung 6.15: Montageplatte [19]

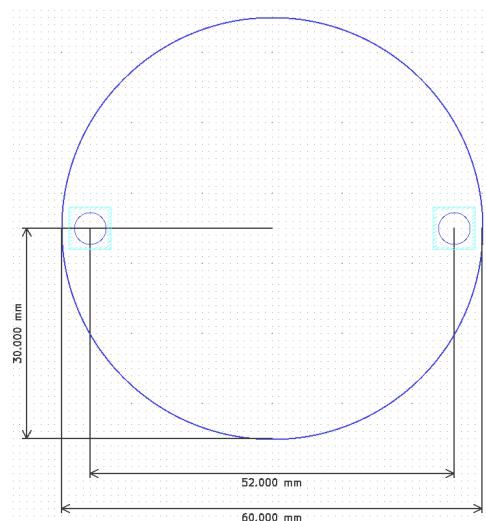


Abbildung 6.16: PCB Sensorbaustein Dimensionen



Abbildung 6.17: Montage des Sensorbausteins



Abbildung 6.18: Montage des Sensors von vorne



Abbildung 6.19: Montage des Sensors von hinten

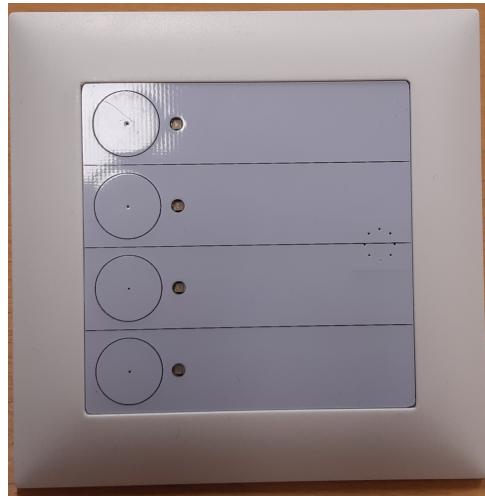


Abbildung 6.20: Frontansicht des fertig montierten Sensorbausteins

6.2 Aktorbaustein

Im folgenden wird die Hardware des Aktorbausteins, ähnlich wie beim Sensorbaustein, beschrieben. Vom Auftraggeber ist ein Aktorbaustein mit 4 Relais, zwei Ausgängen 0..10 V und zwei Eingängen 0..10 V gefordert. Der Aktorbaustein wird über 24V-Netzteil mit Spannung versorgt und mithilfe eines On-Off Schalters kann die Spannungsversorgung getrennt werden. Die Logik wird mit 3.3 V betrieben, dazu braucht es ein DC/DC Wandler der die 24 V in 3.3 V wandelt. Die Programmierschnittstelle ist dabei gleich wie beim Sensorbaustein (siehe Kapitel 6.1). Die Spannung der 0...10 V Ausgänge werden über PWM eingestellt und bei den 0...10 V Eingängen kommt der AD-Wandler des Mikrocontrollers zum Einsatz. Damit man weiß welche Relais geschaltet haben, werden LEDs zum Signalisieren verwendet. Um eine bessere Funkreichweite zu erzielen, kann bei dem Aktorbaustein, wie beim Sensorbaustein, eine Antenne angebracht werden, angedacht ist die Verwendung einer zertifizierten Dipolantenne. In der Abbildung 6.21 erkennt man rechts oben die vier Relais mit den jeweiligen LEDs unten dran. Rechts unten sind die zwei Eingänge und links davon die Ausgänge.

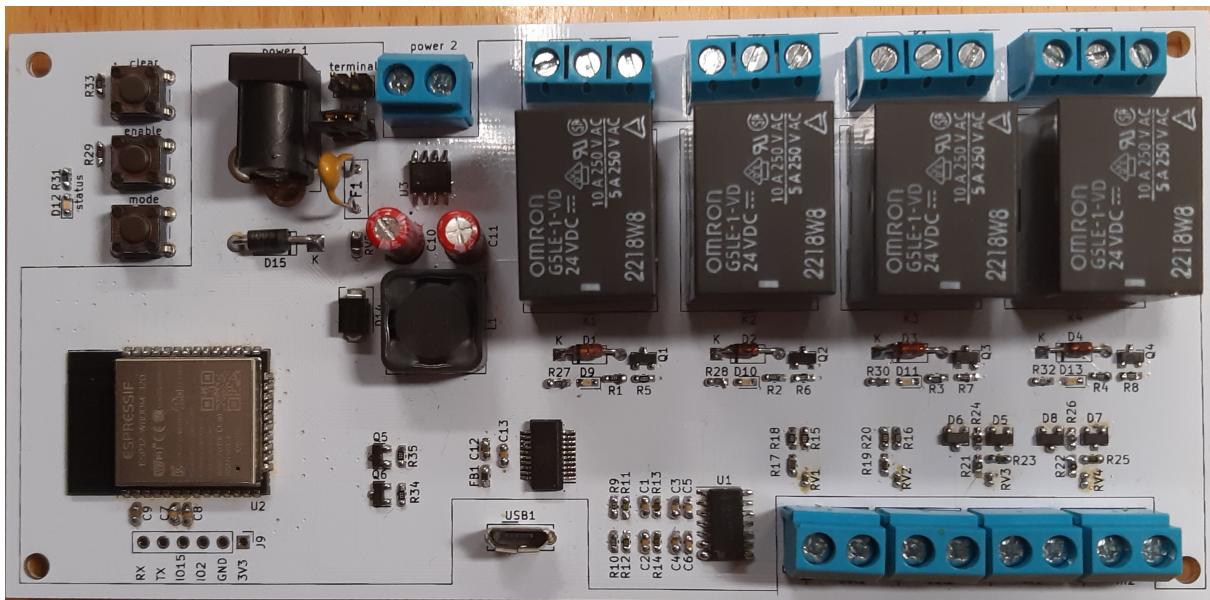


Abbildung 6.21: Frontansicht des Aktorbausteins

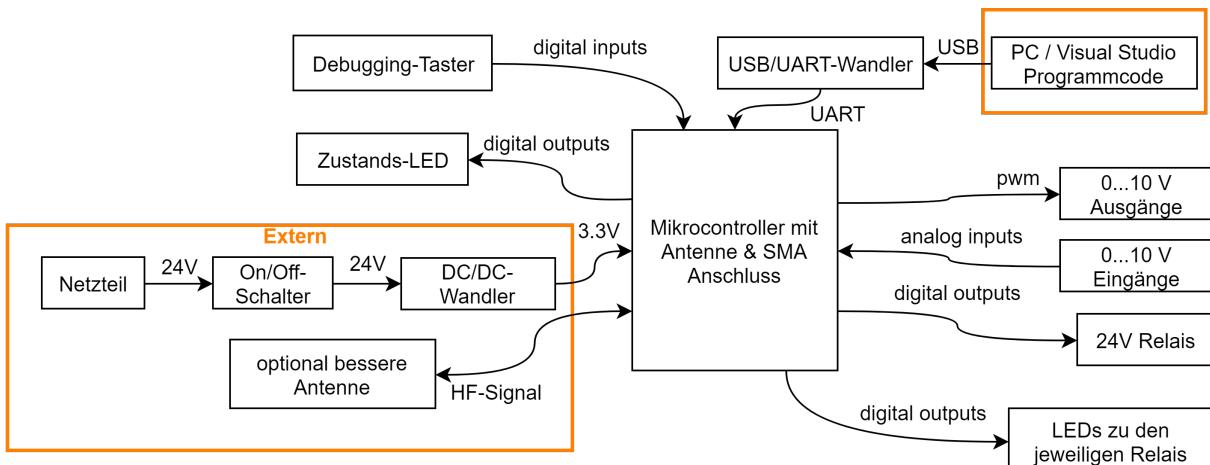


Abbildung 6.22: Übersicht des Aktorbausteins

6.2.1 Mikrocontroller

Es wird wie schon im Sensorbaustein den ESP32 verwendet, dadurch kann garantiert werden, dass die WiFi-Systeme sich sowohl im Aktor- wie auch im Sensorbaustein sich in etwa gleich verhalten, wodurch der Entwicklungsaufwand minimiert wird. Die vom ESP32 benötigten I/Os werden in der Tabelle 6.3 aufgelistet und in der Abbildung 6.23 erkennt man welche I/Os verwendet wurden. Der Programmieranschluss ist wie beim Sensorbaustein und wird nicht nochmals wiederholt.

Name	Anzahl
digital outputs für Relais & LED	10
digital inputs für Buttons	3
ADC inputs für „10 V“ Eingänge	2
PWM outputs für „10 V“ Ausgänge	2
UART Interface für Programmierung	3

Tabelle 6.3: I/O Aktorbaustein

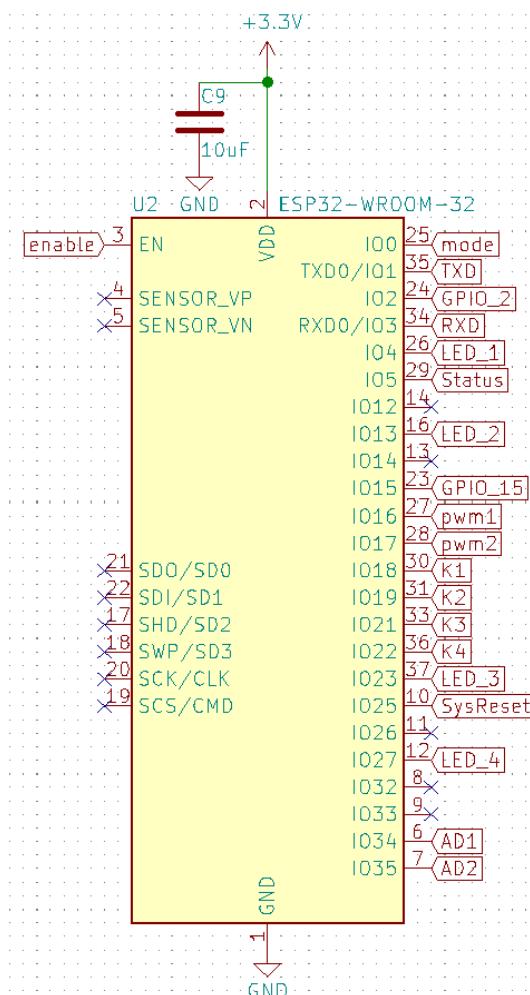


Abbildung 6.23: Mikrocontroller ESP32 im Aktorbaustein

6.2.2 Relais

Die Anforderungen an ein Relais sind, dass es 230 VAC schalten und die Relaisspule mit 24 VDC betrieben werden muss. In der Abbildung 6.24 ist die Schaltung für ein Relais abgebildet. Als Relais wird ein G5LE-1-VD DC24 von Omron Electronics verwendet, welches mit den oben genannten Anforderungen kompatibel ist. Der maximale Schaltstrom beträgt 10 A und die maximale Schaltspannung 250 VAC oder 125 VDC. Die Nennspannung der Spule beträgt 24 VDC, dabei ist garantiert, dass die Spule noch bei 75 % also 18 V der Nennspannung anzieht und bei sicher 10 % der Nennspannung also 2.4 V loslässt. Mit dem Widerstand R_1 wird der Einschaltstrom auf 10 mA begrenzt, mithilfe der Freilaufdiode D1 kann die Selbstinduktionsspannung der Spule kurzgeschlossen werden und R_{24} ist ein Pull-Down-Widerstand, so dass der MOSFET sicher sperrt, falls der Output-Pin nicht angesteuert wird [20]. Als MOSFET wird der BSS123 eingesetzt, welcher sich typischerweise schon mit einer Steuerspannung von 1.7 V durchschaltet und eine Drain-Source-Spannung von bis zu 100 V vertragen kann. Um Energie zu sparen wird nach dem anziehen der Spule mittels eines PWMs die mittlere Spannung über der Spule gesenkt, ein kurzer Test hat ergeben, dass die verwendeten Relais bei 30 % der Nennspannung also 7.2 V sicher noch anziehen, was bedeutet, dass die Leistung von 400 mW auf 36 mW gesenkt wird.

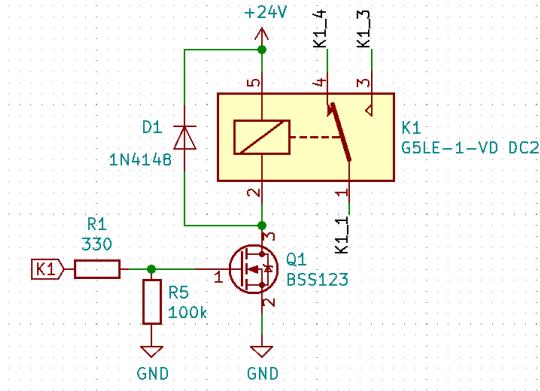


Abbildung 6.24: Relais im Aktorbaustein

6.2.3 Eingänge

Die 10 V Eingänge werden mittels Varistoren (AMCV-0402-260-C180N-T) und Shottky-Dioden vor Überspannung geschützt (Abb. 6.25) Überspannungen können entstehen, wenn eine lange Signalleitung verwendet wird, welche induktiv wirkt und dann abgeschaltet wird. Die Nennspannung des Varistors beträgt 18.4 VAC bzw. 26 VDC, mit einer Varistorspannung von 31 V bis 38 V. Darunter schützen die Shottky-Dioden den ADC-Eingang. Falls der Varistor zerstört wird, wird er leitend und es entsteht ein Kurzschluss am Eingang, was zu falschen Messresultaten führt. Auf Grund dessen, dass der ADC (Abbildung: 5.1), erst ab 170 mV ein Signal erkennt, muss ein Widerstandsnetzwerk verwendet werden. Hierdurch können auch niedrigere Spannungen gemessen werden können. Es wurde in der Gleichung 6.1 die Übertragungsfunktion des Widerstandsnetzwerkes ausgerechnet, durch den Offset ist es nun theoretisch möglich 0 V zu messen. Es wurde eine variable Spannung mit 1 % Toleranz an die 10 V Eingänge des Aktorbausteins gelegt und dann der ADC-Wert aufgenommen (Abbildung 6.26). Die so erstellte Regressionsgerade $U_{AD,measure}$ unterscheidet sich von der theoretisch berechneten Geraden U_{AD1} .

$$U_{AD1} = U_{in1} \cdot \frac{R_{23}||R_{24}}{R_{21} + R_{23}||R_{24}} + 3.3V \cdot \frac{R_{33}||R_{24}}{R_{23} + R_{33}||R_{24}} = U_{in1} \cdot 0.2339 + 0.2123V \quad (6.1)$$

$$ADC = U_{AD1} \cdot 1253.1 - 218.54 = (U_{in1} \cdot 0.2339 + 0.2123V) \cdot 1253.1 \frac{1}{V} - 218.54 \quad (6.2)$$

$$ADC = U_{AD1} \cdot 293.1 \frac{1}{V} + 47.5 \quad (6.3)$$

$$U_{AD1} = 0.003412V \cdot ADC - 0.162V \quad (6.4)$$

$$U_{AD,measure} = 0.003449V \cdot ADC - 0.276V \quad (6.5)$$

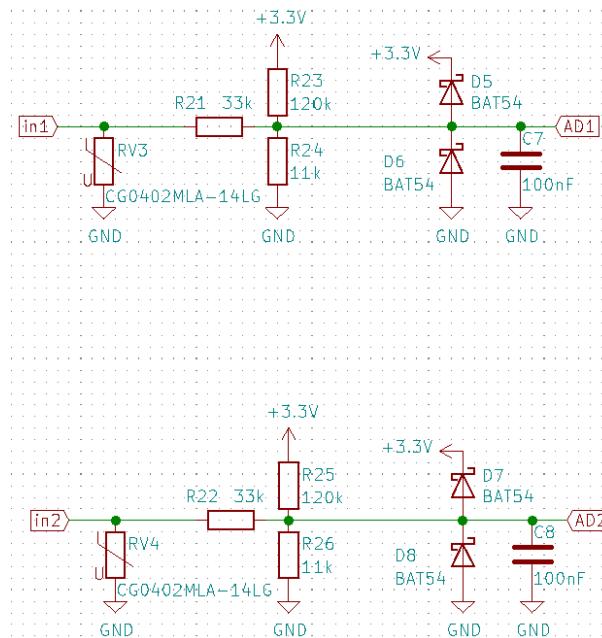


Abbildung 6.25: 10V Eingänge des Aktorbausteins

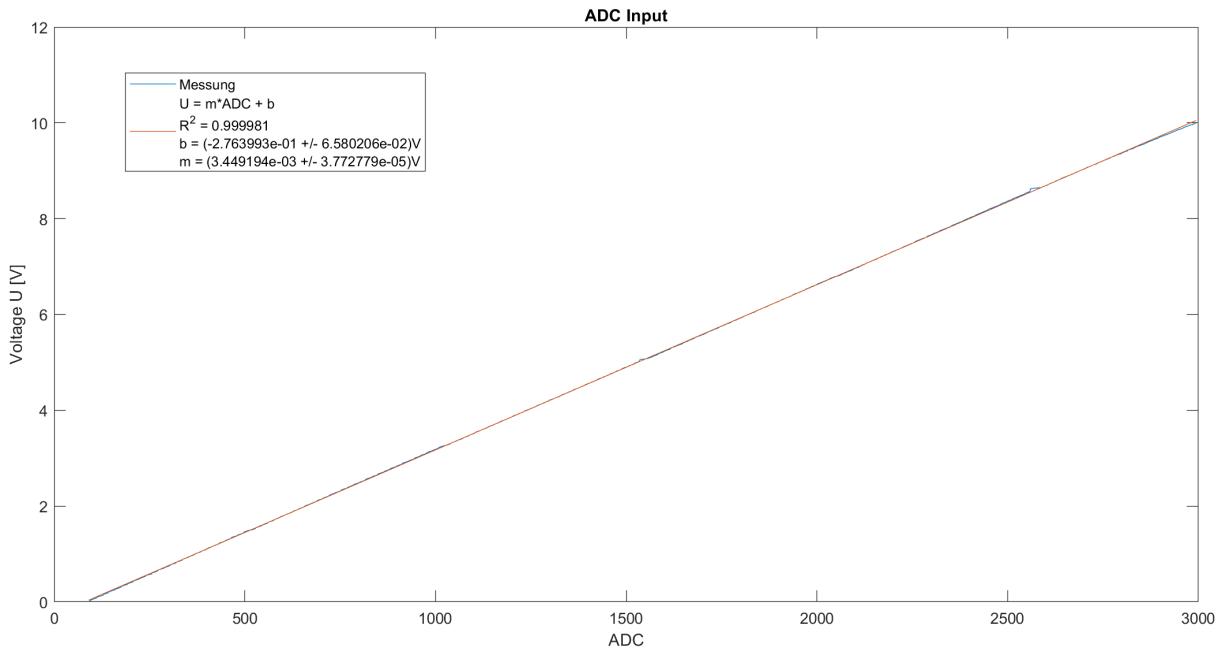


Abbildung 6.26: Spannung gegenüber dem ADC Wert gemessen an beiden 10 V Eingängen und dann gemittelt

6.2.4 Ausgänge

Bei den 10 V Ausgängen wird jeweils ein PWM vom Mikrocontroller herausgegeben und dann das Signal mittels eines Tiefpasses geglättet (Abb. 6.27). Die Polfrequenz sowie die $-3dB$ Grenzfrequenz des Tiefpass erster Ordnung berechnet sich für einen 10 Bit AD Wandler und einen Quarz von 40 MHz , welcher im ESP32 verbaut ist, folgendermassen: 40 MHz Takt $\rightarrow 40 \text{ kHz}$ PWM \rightarrow Zeitkonstante $\tau = 1000/40 \text{ kHz} = 25 \text{ ms}$ \rightarrow Zeit bis zum Endwert $t_{end} = \tau * \ln(1024) = 173 \text{ ms}$ \rightarrow Grenzfrequenz $f_g = 1/\tau = 40 \text{ Hz}$. Die Verzögerungszeit t_{end} ist im Übrigen gleich der Dauer bis die Spannung am Kondensator im Bereich von 1 LSB vom Endwert ist [21]. 173 ms Verzögerungszeit scheinen recht lange zu sein, wenn man diese Zeit verkürzen möchte, könnte man auf einen 8 Bit AD Wandler wechseln, wodurch die Polfrequenz auf 156 kHz angehoben wird. Ein anderer Weg besteht darin eine höhere Filterordnung zu wählen. Um eine Abschwächung von 1024 oder 60 dB des 40 kHz PWM Taktes zu erzielen, kann bei zweiter Filterordnung, welche 40 dB/dec Dämpfung hat, die 40 kHz nicht durch 3 Dekaden (Faktor 1000) sondern nur um 1.5 Dekaden (Faktor 31,6) teilen, um die neue Polfrequenz f_p herauszufinden. Hier macht es aber auch Sinn -80 dB (Faktor 100) bei 40 kHz zu nehmen, um noch ein sauberes Ausgangssignal zu kriegen. Also ist die gewählte Polfrequenz f_p bei 400 Hz , τ bei 2.5 ms und t_{end} bei 17 ms . Die Komponenten der Filterstufen wurden dann wie folgt gewählt: erste Stufe mit $R_1 = 3.9 \text{ k}\Omega$ & $C_1 = 100 \text{ nF}$ und die zweite Stufe mit $R_2 = 39 \text{ k}\Omega$ & $C_2 = 10 \text{ nF}$. Da $R_1 \cdot C_1 = R_2 \cdot C_2$ ist, ist die Polgüte $q_p = 0.5$, woraus resultiert, dass die Dämpfung bei der Polfrequenz $f_p = 400 \text{ Hz}$ einen Wert von -6 dB annimmt. Die -3 dB Grenzfrequenz ist folglich bei $f_g = f_p \cdot \sqrt{2^{1/2} - 1} = 257 \text{ Hz}$ [22] und die Dämpfung bei 40 kHz entspricht nach wie vor den gewünschten -80 dB . Nach dem Tiefpass gelangt das Signal in einen positiven Verstärker welcher die maximale Ausgangsspannung des ESP32 von 3.3 V auf theoretisch $3.3 \text{ V} \cdot (1 + 22/10) = 10.56 \text{ V}$ erhöht..

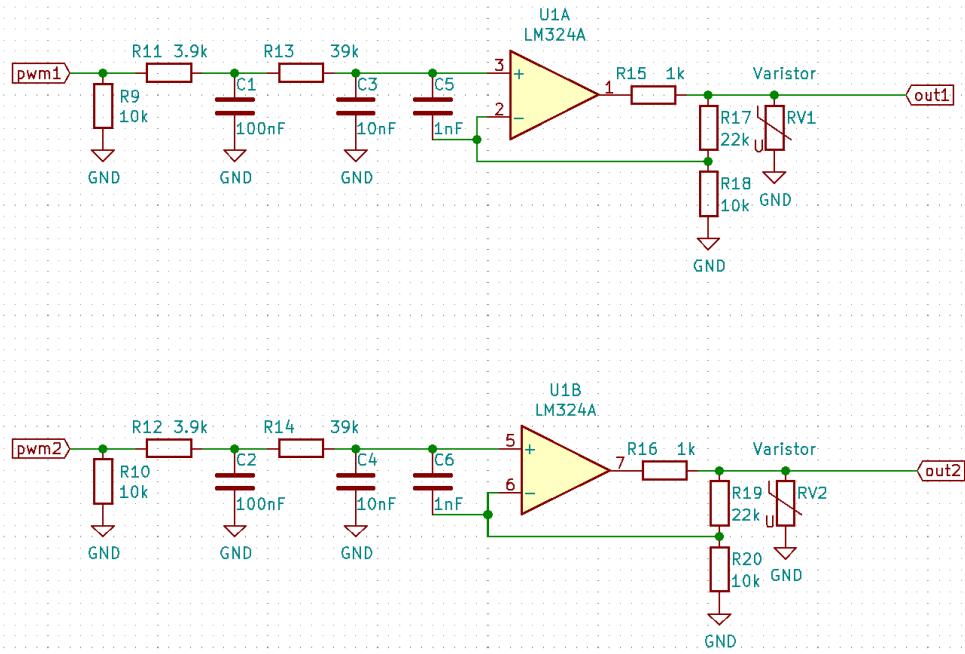
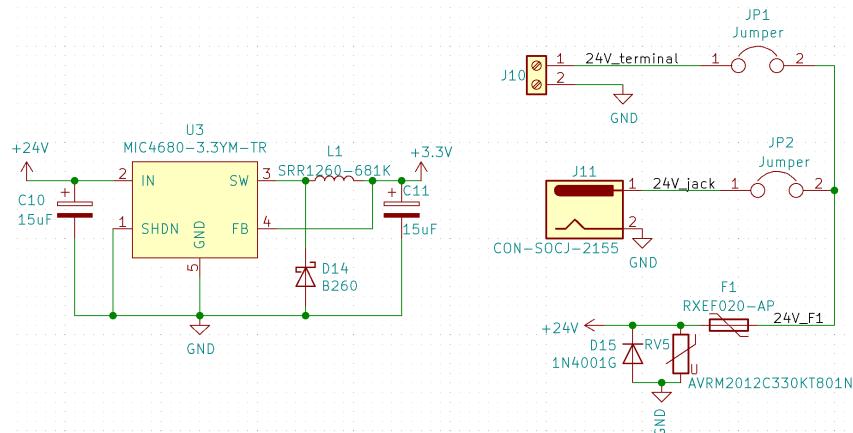


Abbildung 6.27: 10 V Ausgänge des Aktorbausteins

6.2.5 Spannungsversorgung

Mithilfe eines üblichen 24 V Netzgerätes, welches einen Zylinderstecker 2.1 x 5.5 mm (engl. DC Barrel Jack) hat, kann der Aktorbaustein betrieben werden. So gibt z.B von XP-Power ein 18 W 24 V/DC Netzgerät für 11.73 CHF bei mouser. Um die Logik zu betreiben werden jedoch 3.3 V benötigt. Hier kommt der Spannungswandler MIC4680 zum Einsatz, welcher eine Effizienz von bis zu 90 % aufweist. Vom Wandler ausgegeben werden typischerweise 3.3 V, im aller schlimmsten Fall kann die Spannung jedoch zwischen 3.201 V und 3.399 V liegen. In der Abbildung 6.28 ist das Schema der 3.3 V Spannungsversorgung zu erkennen, es wurde hier noch eine Sicherung eingebaut da der Print nicht zerstört wird, falls zu viel Strom fliesst. Der Schraubklemmen sind dafür da, um z.B einen ON-OFF-Wippschalter, welcher am Gehäuse montiert wird, anzuschliessen. Die Beschaltung um den Spannungswandler ist dem Datenblatt des MIC4680 zu entnehmen.



6.2.6 Interface

Die Ausgänge der Relais sowie die 10 V Ein- und Ausgänge sind auf Schraubklemmen geführt (Abb. 6.29). Es stehen insgesamt 6 LEDs zur Verfügung (Abb. 6.30). Die LEDs 1 bis 4 zeigen an ob das jeweilige Relais geschaltet hat. LED5 kann als Status LED verwendet werden. Die Buttons sind die gleichen wie im Sensorbaustein unter Punkt 6.1.7 Programmierananschluss.

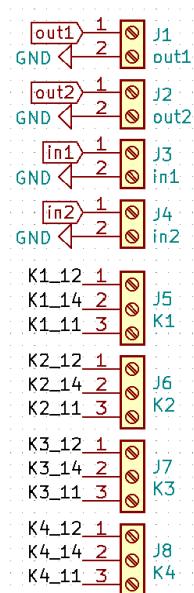


Abbildung 6.29: Klemmenanschlüsse
Aktorbaustein

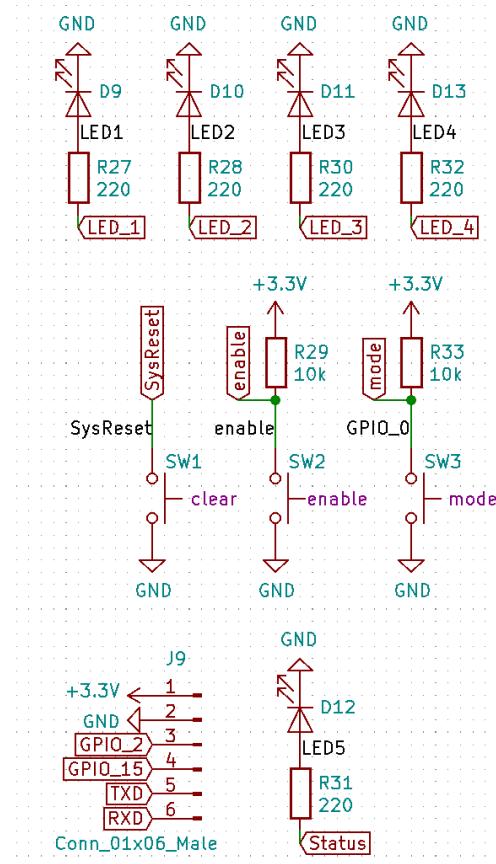


Abbildung 6.30: Buttons und LEDs Ak-
torbaustein

7 Validierung

In diesem Kapitel geht es darum, Anforderungen und Eigenschaften des Aktor- und Sensorbausteins zu testen.

7.1 Sensorbaustein

7.1.1 ESD Test

Der Sensorbaustein wird wie eine Unterputzsteckdose verbaut und die Frontplatte dient als User-Interface. Genauer bedeutet es, dass der Benutzer, die Touchflächen der Frontplatte berühren muss, um mit dem Sensorbaustein zu interagieren. Durch das berühren kann es jedoch zu elektrostatischen Entladungen kommen, die der Sensorbaustein überstehen muss. Deswegen wurde mithilfe eines ESD-Tests, getestet bis zu welchem Prüfschärfegrad, also bis zur welcher Entladespannung der Sensorbaustein funktioniert. In der Tabelle 7.1 sind die Prüfschärfegrade und deren Spannungen abgebildet. Für den Test wurde die Kontaktentladung genommen, mit welcher eine Entladung über einen Finger nachgestellt wird.

Der Testablauf war wie folgt [23]:

- Beginn beim tiefsten Schärfegrad
- 1 Entladung/s
- mind. 10 Entladungen, jeweils auf jeder Touchfläche

	Kontaktentladung	Luftentladung
Prüfschärfegrad	Prüfspannung [kV]	Prüfspannung [kV]
1	2	2
2	4	4
3	6	8
4	8	15
X	Spezial	Spezial

„X“ ist ein offener Prüfschärfegrad. Dieser Prüfschärfegrad muss in der Produktnorm festgelegt werden. Falls höhere Spannungen als die oben angegebenen festgelegt werden, sind möglicherweise besondere Prüfgeräte erforderlich.

Abbildung 7.1: ESD Testablauf [23]

Das Resultat war, dass der Sensorbaustein beim Prüfschärfegrad 4 Schaden nahm. Bei geringeren Spannungen war er funktionstüchtig. Was bedeutet, dass der Sensorbaustein Prüfschärfegrad 3 mit 6 kV erreicht hat. Der Schaden beim Grad 4 war sichtbar und befand sich beim Seriell/UART-Wandler (Abbildung: 7.2), was nicht zu erwarten war. Der betroffene Pin war die Spannungsversorgung des IC's, dazu wurden direkt die 5 V der Eingangsspeisung genommen, die Entweder über USB oder Pins angeschlossen wird. Beide Spannungsversorgungen sind miteinander verbunden, da in der Anwendung nur eines von beidem verwendet wird. Theoretisch wäre es auch möglich gewesen die 3.3 V als Versorgung des Seriell/UART-Wandlers zu nehmen, die Frage ist, ob dann der gleiche Schaden aufgetreten wäre. Die Schutzdiode welche Über- und Unterspannung ableiten sind mit Ground bzw. 3.3 V verbunden und nur der Seriell/UART-Wandler und der DC/DC Wandler sind an der 5 V Speisung angeschlossen. Schlussendlich ist der Strom über den Seriell/UART-Wandler abgeflossen und hat ihn zerstört.



Abbildung 7.2: Der Schaden am Seriell/Uart-Wandler welcher, wegen des ESD Tests mit Prüfschärfegrad 4 zu Stande kam

7.1.2 Energieverbrauch des Sensorbausteins

Der Sensorbaustein wurde unter folgenden Bedingungen gemessen:

1. Ungeschalteter Normalzustand (Abbildung: 7.3)

(Der Sensorbaustein wird nicht bedient und hat eine Verbindung mit einem WLAN Netzwerk)

Der Sensorbaustein hat eine mittlere Leistungsaufnahme von ca. 370 mW. Auffällig ist, dass die Leistung mit einer Periode von 10 s schwankt, was dem zu Programm zugrunde liegt, denn die Temperaturmessung, denn das Status LED toggelt alle 10 s, um anzuzeigen, dass die Temperatur nach 10 s wieder gemessen wird. Unregelmäßige Leistungssprünge von bis zu ca. 400 mW kommen häufiger vor, was wahrscheinlich durch das WLAN ausgelöst wird.

2. Geschalteter Normalzustand (Abbildung: 7.4)

(Der Sensorbaustein wird bedient und hat eine Verbindung mit einem WLAN Netzwerk)
In den Zeiten um ca. 35 s, 60 s und 85 s werden alle Vier Taster betätigt, die Leistungsaufnahme steigt um 10 mW von vorher 370 mW auf ca. 380 mW, was vertretbar ist.

3. Keine Verbindung (Abbildung: 7.5)

(Der Sensorbaustein findet kein zulässiges WLAN Netzwerk)

In diesem Zustand hat der Sensorbaustein eine mittlere Leistungsaufnahme von 930 mW, was beträchtlich ist. Dies ist daran geschuldet, dass der Sensorbaustein vermutlich seine Signalstärke erhöht, um ein Netzwerk zu finden. Deswegen wird nicht empfohlen, den Sensorbaustein, in diesem Zustand zu lassen.

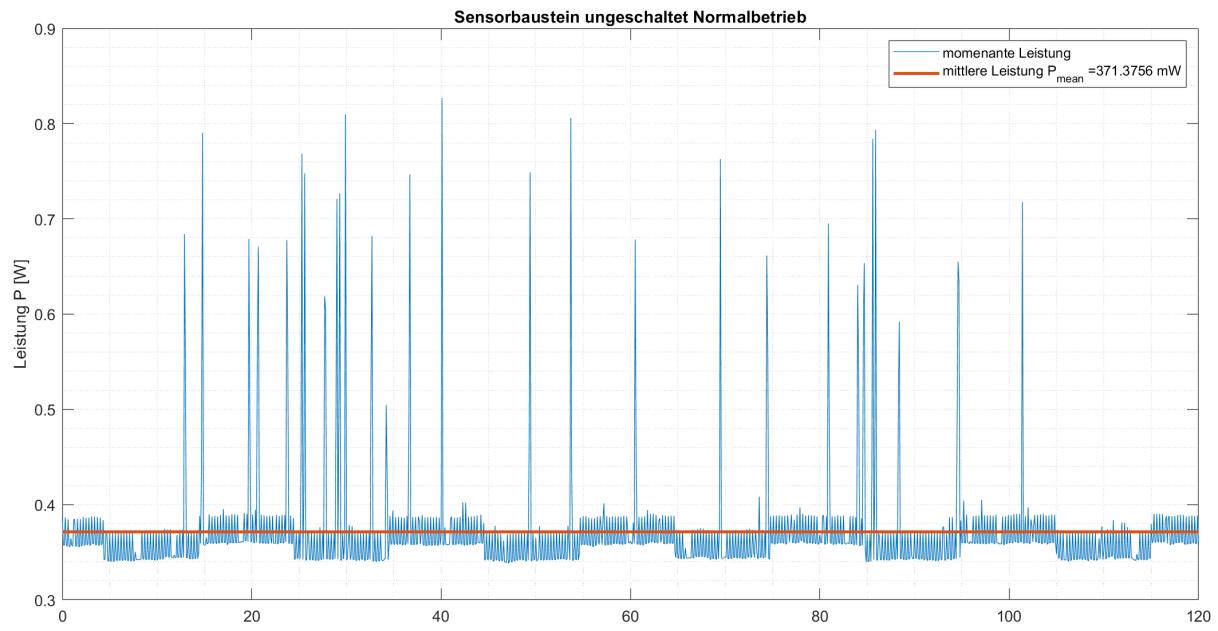


Abbildung 7.3: Die Leistung des Sensorbausteins, wenn er nicht bedient wird und mit dem Netzwerk verbunden ist

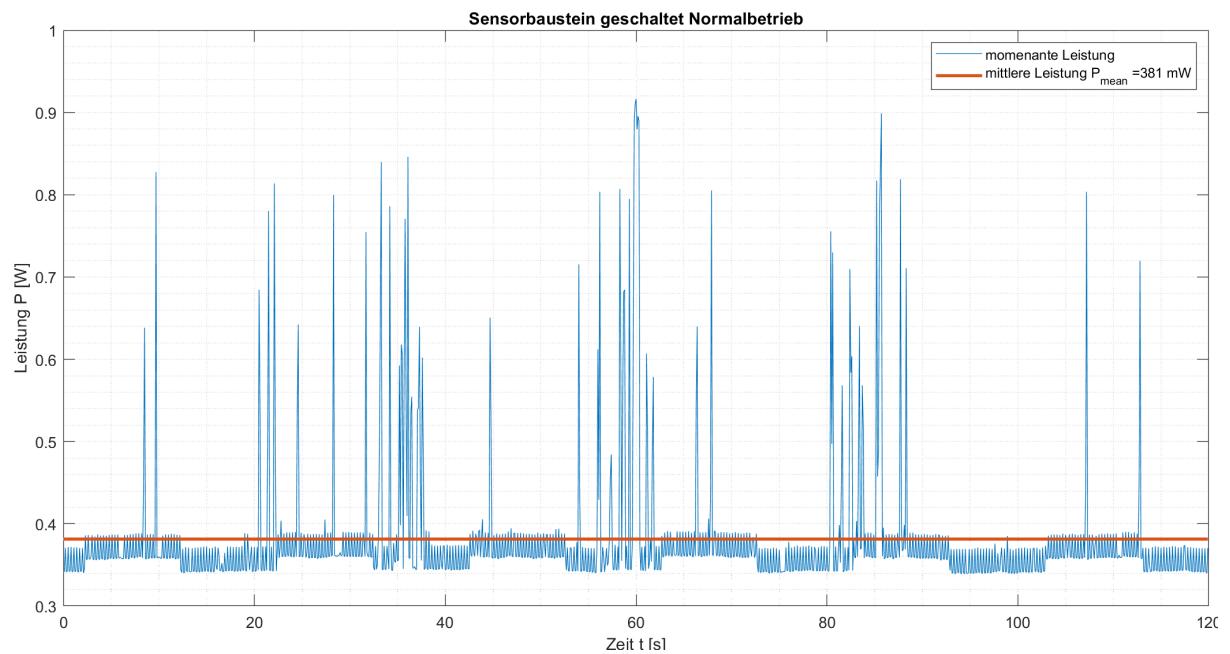


Abbildung 7.4: Die Leistung des Sensorbausteins, wenn er bedient wird und mit dem Netzwerk verbunden ist

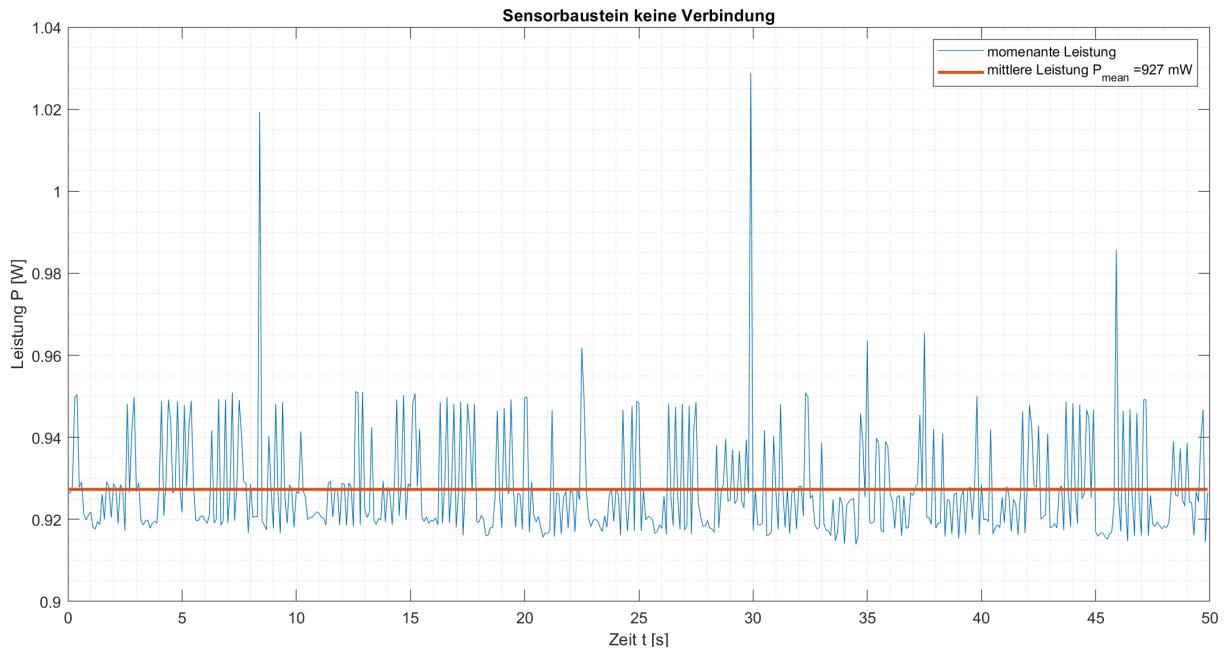


Abbildung 7.5: Die Leistung des Sensorbausteins, wenn er einen Verbindungsauflaufbau versucht, aber kein geeignetes WLAN Netzwerk vorhanden ist

7.1.3 Temperaturmessung

Es wird berechnet, wie gross ungefähr die Abweichung der Temperaturmessung um die Raumtemperatur von Wohngebäuden, was üblicherweise 20°C entspricht, ist. Dazu wurde mit $U_{\text{ref}} = 2.39\text{ V}$ und einer Temperatur von $20.2^\circ\text{C} \approx 20^\circ\text{C}$ gerechnet (folglich Kapitel 5.4.7). Gerechnet wird mit einem Vorwiderstand R_V von $100\text{ k}\Omega$ mit 0.1% Abweichung und einem NTC (NCU18WF104J60RB von Murata) mit einem Widerstand von R_T , bei welchem der Widerstand R_{25} bei 25°C eine Abweichung von 5% hat. Um die Abweichung zu berechnen, wird die anliegende Spannung bei 20.2°C berechnet (Gl.: 7.1). Um $U_{ntc,max}$ zu maximieren, wurde der maximale Widerstand des NTC bei 20.2°C und der kleinste anliegende Wert von dem Vorwiderstand R_V verwendet. Danach wurden mit 7.2 und 7.3 die Temperatur berechnet. Die Abweichung ergab sich aus der Differenz, des zu erwartenden und des zu berechnenden Wertes, was 0.9 K entspricht. Da dieser Wert zu hoch erscheint, grösser als 0.5°C , wäre ein NTC mit geringerer Toleranz besser gewesen. Dazu gibt es Baugleiche NTCs mit den gleichen Parametern, einfach kleinere Toleranzen.

Werte:

$$\begin{aligned}
 R_{T,20.2} &= 126\text{ k}\Omega \\
 R_{T,20.2,max} &= 126\text{ k}\Omega \cdot 1.05 = 132\text{ k}\Omega \\
 R_V &= 100\text{ k}\Omega \\
 R_{V,min} &= 100\Omega \cdot 0.99 = 99.9\text{ k}\Omega \\
 B &= 4250\text{ K} \\
 T_{25} &= 298.15\text{ K}
 \end{aligned}$$

Was als erhöhte Spannung an U_{ntc} bei 20.2 °C anliegt:

$$U_{ntc,max} = U_{ref} \cdot \frac{R_{T,20.2,max}}{R_{T,20.2,max} + R_{V,min}} = 2.39 V \cdot \frac{132 k\Omega}{(132 k\Omega + 99.9 k\Omega)} = 1.36 V \quad (7.1)$$

Temperatur Berechnung bei 20.2 °C im Mikrocontroller:

$$\left(\frac{R_T}{R_V}\right)_{max} = \frac{U_{ntc,max}}{U_{ref} - U_{ntc,max}} = 1.32 \quad (7.2)$$

$$T = \frac{1}{\frac{1}{T_{25}} + \frac{1}{B} \cdot \ln\left(\left(\frac{R_T}{R_V}\right)_{max}\right)} = 292.45 K = 19.3 ^\circ C \quad (7.3)$$

$$\Delta T_{max} = 20.2 ^\circ C - 19.3 ^\circ C = 0.9 K \quad (7.4)$$

7.2 Aktorbaustein

7.2.1 Energieverbrauch des Aktorbaustein

Der Aktorbaustein wurde unter folgenden Bedingungen gemessen:

1. Ungeschalteter Normalzustand (Abbildung: 7.6)

(Der Aktorbaustein wird nicht bedient und hat eine Verbindung mit einem WLAN Netzwerk)

Der Aktorbaustein hat eine mittlere Leistungsaufnahme von ca. 629 mW. Auffällig ist, wie beim Sensorbaustein, dass die Leistung mit einer Periode von 10 s schwankt, was wiederum, wegen des blinkenden Status LED ist. Dazu kommt es alle 10 zu kurzen Leistungssprüngen.

2. Geschalteter Normalzustand (Abbildung: 7.7)

(Der Aktorbaustein wird bedient und hat eine Verbindung mit einem WLAN Netzwerk)

Die Relais werden nacheinander eingeschalten und danach wieder ausgeschaltet, ein eingeschaltetes Relais und ein LED verbrauchen zusammen somit ca. $(900 \text{ mW} - 620 \text{ mW})/4 = 70 \text{ mW}$.

3. Keine Verbindung (Abbildung: 7.8)

(Der Aktorbaustein findet kein zulässiges WLAN Netzwerk)

In diesem Zustand hat der Aktorbaustein eine mittlere Leistungsaufnahme von 1020 mW, was ähnlich viel wie beim Sensorbaustein ist, die Ursachen hierfür werden die gleichen sein.

4. Keine Verbindung (Abbildung: 7.9)

(Der Aktorbaustein schaltet alle Relais einmal mit und einmal ohne Sparmodus)

Zu erwarten ist, dass der Sparmodus ca. 1.5W einspart, denn der eingestellte DutyCycle beträgt 30 %, und ein Relais (G5LE-1-VD DC24) verbraucht 400 mW, wenn es mit 24 V betrieben wird. Somit ergibt sich aus der Gleichung 7.5 eine Ersparnis von 1.5 W. In der Grafik ist im Vergleich dazu eine Ersparnis von 1.7 W (Gl.: 7.6). Der Unterschied ist daher, dass der ausgegebene DutyCycle nicht ganz genau 30 % und die Amplitude nicht ganz genau 24 V beträgt.

$$P_{gespart,\text{theoretisch}} = (1 - 0.3^2) \cdot 4 \cdot 400 \text{ mW} = 1.5 \text{ W} \quad (7.5)$$

$$P_{gespart, \text{gemessen}} = 2680 \text{ mW} - 940 \text{ mW} = 1.7 \text{ W} \quad (7.6)$$

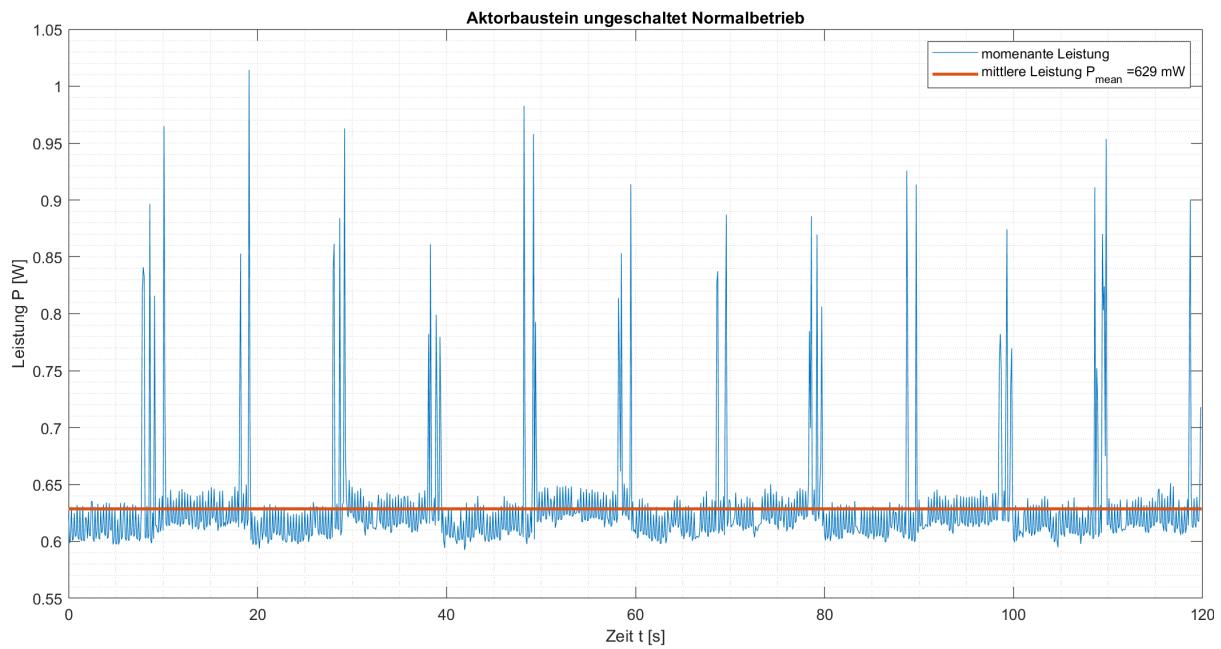


Abbildung 7.6: Die Leistung des Aktorbausteins, wenn er nicht bedient wird und mit dem Netzwerk verbunden ist

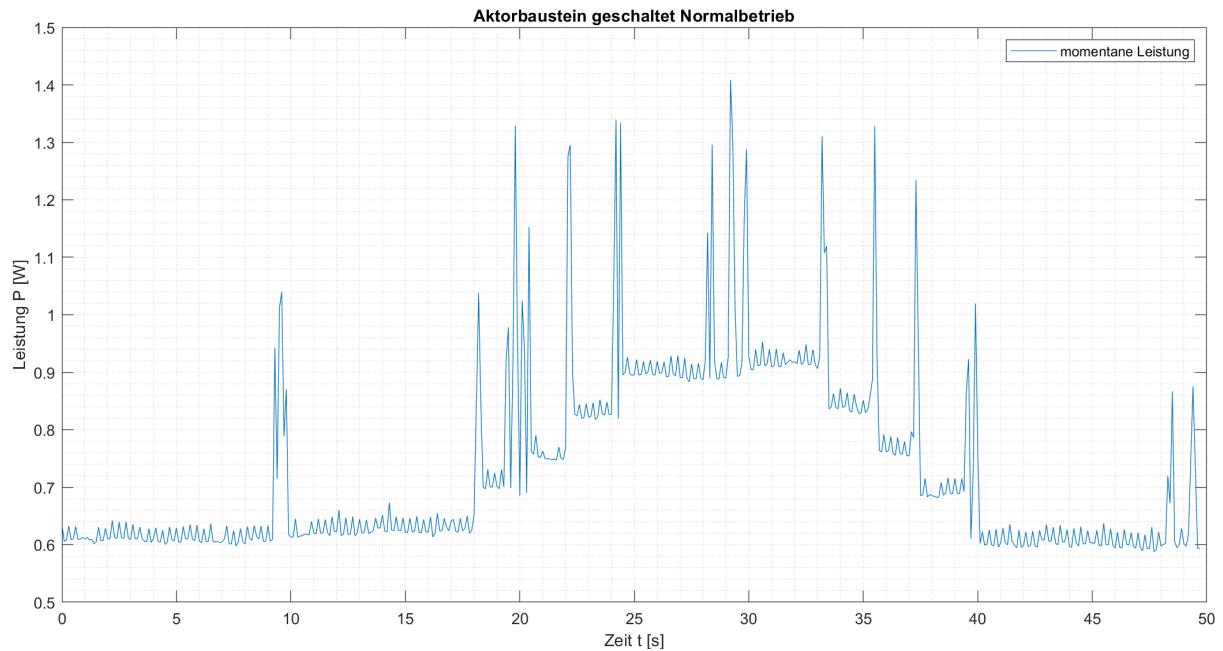


Abbildung 7.7: Die Leistung des Aktorbausteins, wenn er bedient wird und mit dem Netzwerk verbunden ist

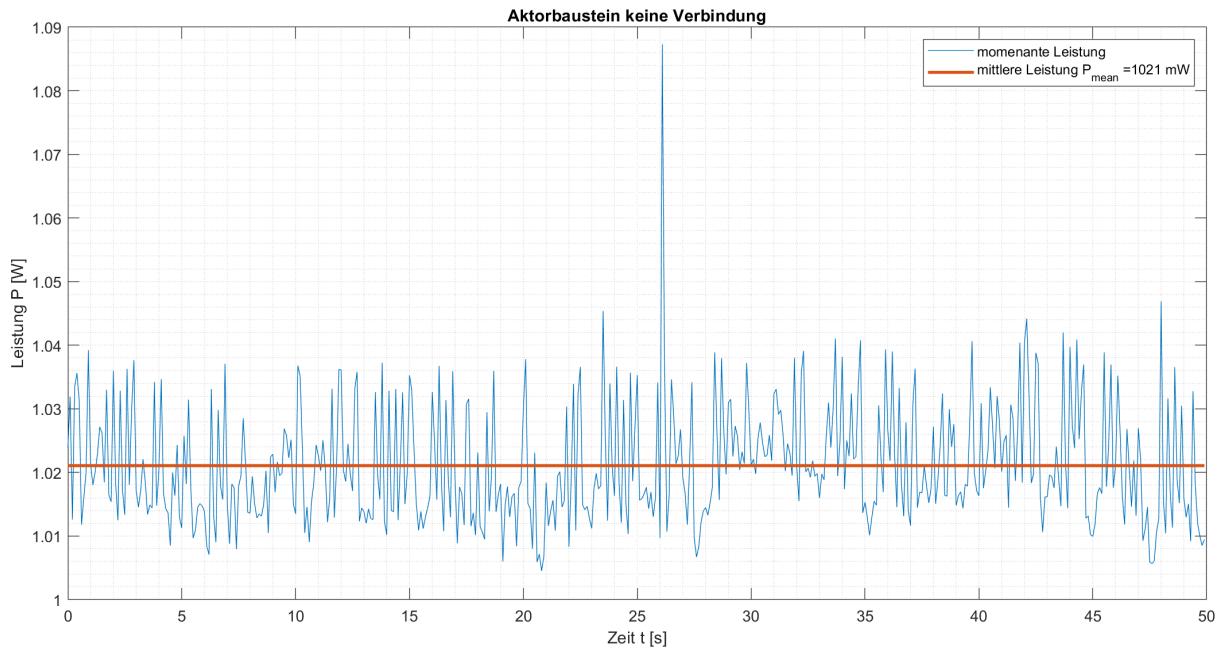


Abbildung 7.8: Die Leistung des Aktorbausteins, wenn er einen Verbindungsauflauf versucht, aber kein geeignetes WLAN Netzwerk vorhanden ist

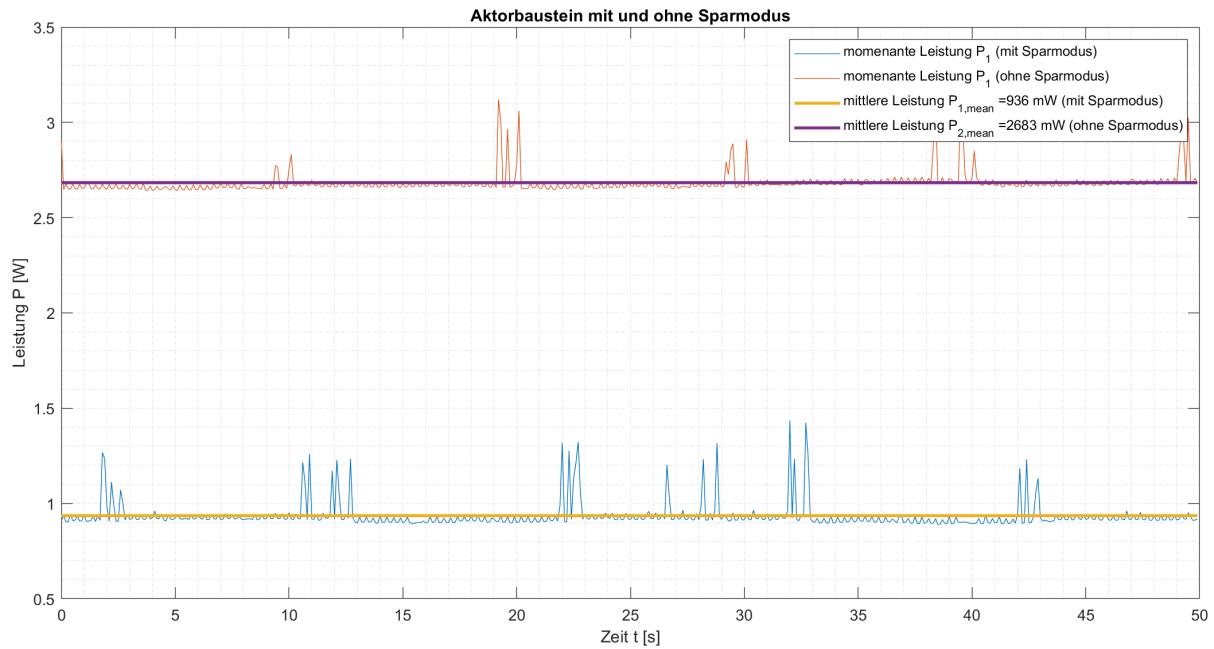


Abbildung 7.9: Die Leistung des Aktorbausteins mit und ohne Sparmodus, wenn er alle Relais schaltet

7.2.2 Wärmeverteilung

Es wurde verifiziert, wie stark sich der Aktorbaustein in Betrieb erwärmt. Hierfür wurde eine Infrarotkamera von Flir verwendet. Erwärmten haben sich bei normalen Betrieb insbesondere die Diode D14 (Abbildung: 7.10), der ESP32 (Abbildung: 7.12) und der DC/DC-Wandler (Abbildung: 7.11). Um zu verifizieren wie stark sich die Relaischaltung erwärmt, wurde eine Last

von 9.2 A 230 VAC über ein Relais geschaltet, laut Infrarot Kamera wurde eine Temperatur von 58 °C gemessen, was mit unbestimmten Unsicherheiten zu verstehen ist.

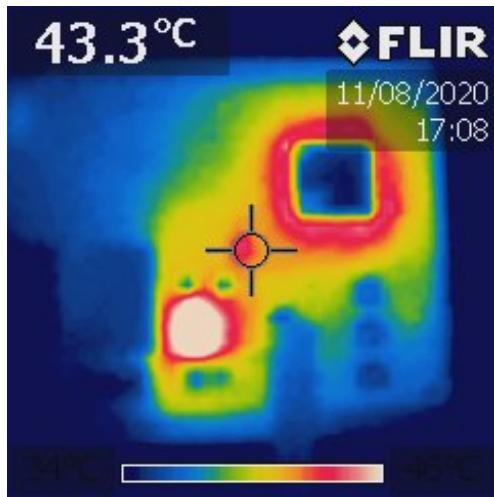


Abbildung 7.10: Wärmeentwicklung über der Diode D14

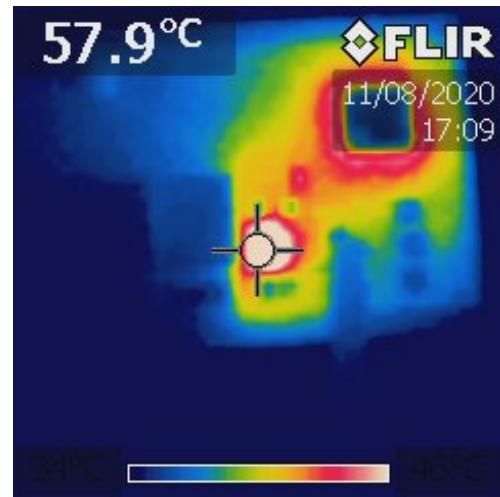


Abbildung 7.11: Wärmeentwicklung über dem DC/DC-Wandler

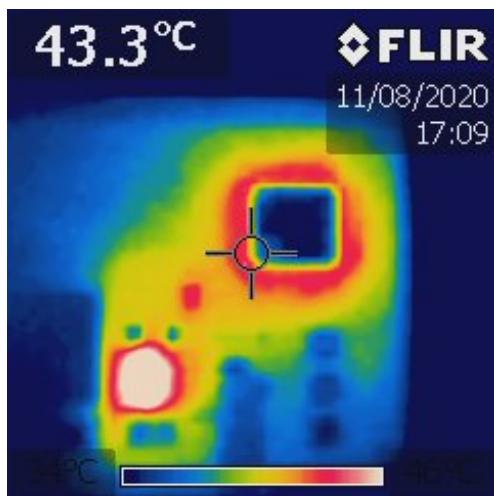


Abbildung 7.12: Wärmeentwicklung über dem ESP32

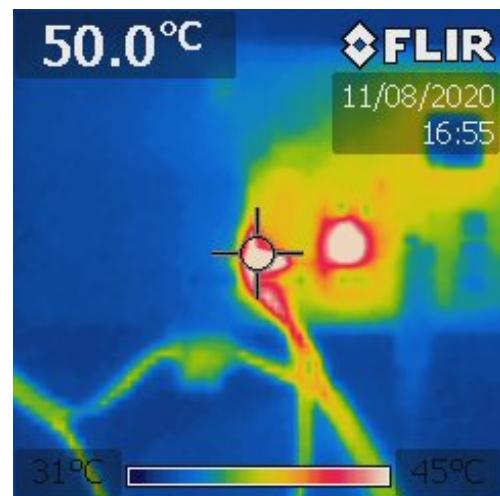


Abbildung 7.13: Wärmeentwicklung über ein Relais welches 9.2 A / 230 VAC schaltet

7.2.3 Reaktion Zeiten

Die Reaktionszeiten von Taster Betätigung auf dem Touch Sensor bis die Schalthandlung am Relais auf dem Aktor Bord Ausgeführt wird, wurde in nachfolgender Messung ermittelt.

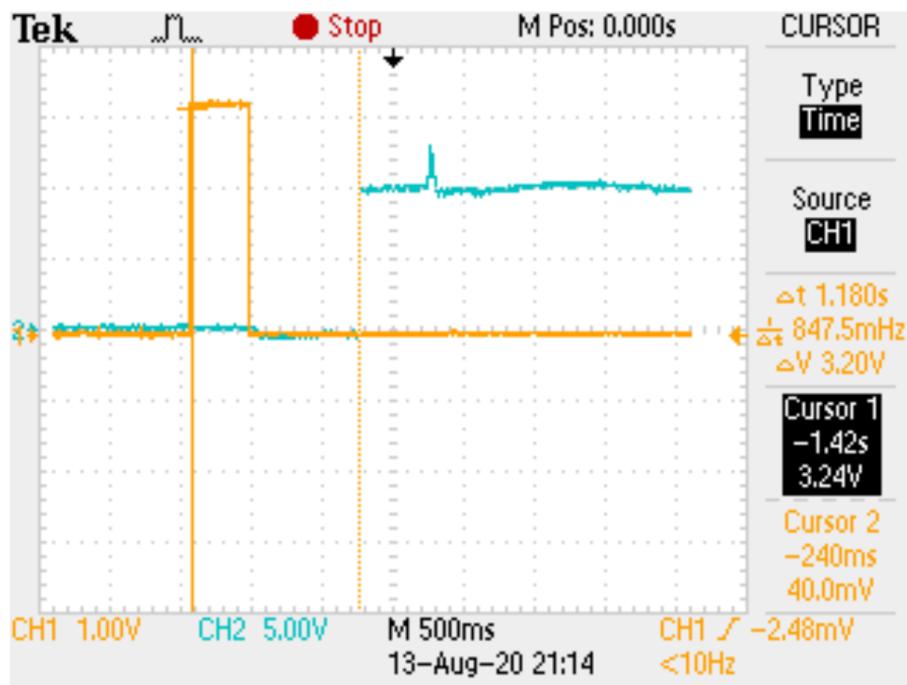


Abbildung 7.14: Signal Taster ist gelb dargestellt, Signal Relais blau dargestellt

In der Abbildung 7.14 kann zwischen dem Auslöse-Signal und dem Schalt-Signal eine Zeitdifferenz von 1.18 Sekunden erkannt werden. Der Grund warum so eine grosse Zeitdifferenz entsteht liegt daran, dass Openhab in den Ruls der Schaltbefehl für den Aktor generiert werden muss.

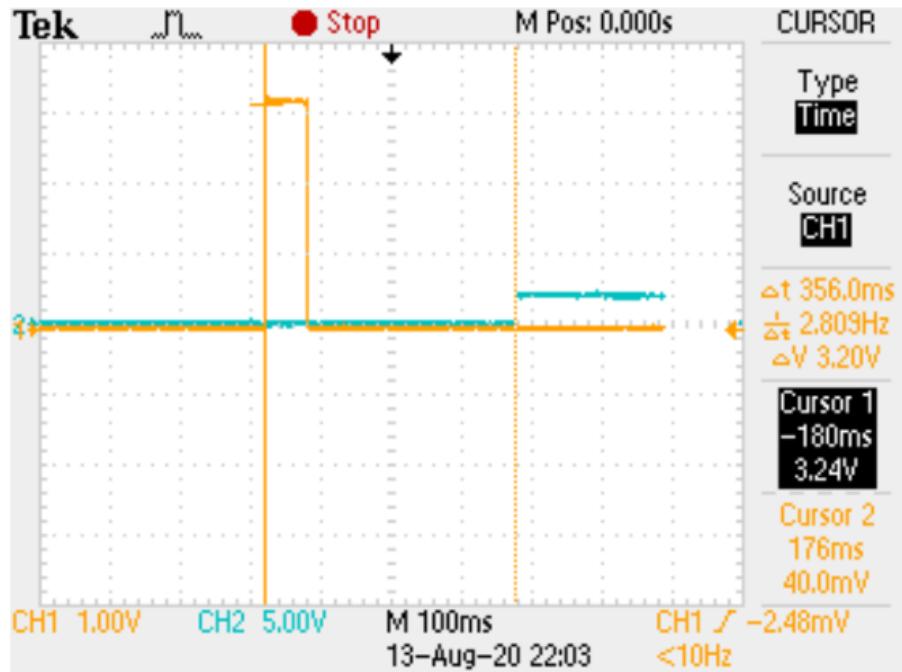


Abbildung 7.15: Zeitmessung ohne Openhab Rules, Signal Taster ist gelb und Signal Relais blau dargestellt

In der Abbildung 7.15 kann zwischen dem Auslöse-Signal und dem Schalt-Signal eine Zeitdifferenz von 356 ms erkannt werden. Bei dieser Messung wurde der Schaltbefehl, also Topic und Payload im Sensor-Programmcode so angepasst, dass direkt eine Schalthandlung ausgelöst wird, sprich Rules von Openhab wurden umgangen.

8 Lizenzen

In diesem Kapitel werden die Lizenzen, welche in der Software genutzt werden, behandelt. In der Tabelle 8.1 sind die Lizenzen der verwendeten Bibliothek aufgeführt und für was sie verwendet wurden. Die Standard GNU GPL erlaubt die Verwendung nur für freie Software und die aufgeführte GNU LGPL auch für proprietäre Software [24]. Iostream ist unter GNU Lizenz, was unsere Software, ebenso zur freien Software macht. Der Grund hierfür ist die strenge Copyleft-Klausel der GNU GPL. Diese sagt aus, dass eine veränderte Fassung des ursprünglichen Werkes, nicht mehr Nutzungseinschränkungen haben darf, als das Original.[25] Die MIT lizenzierte Software kann unter anderen Lizenzen gestellt werden [26]. Die Apache Lizenz erlaubt es die Software für jeglichen Grund zu verwenden und dies ohne Lizenzgebühren [27]. Zusammenfassend ist zu sagen, dass unsere Software, als freie Software angeboten werden müsste.

Bibliothek	Lizenz	Herausgeber	Zweck
Arduino.h	GNU LGPL 2.1 oder höher	Arduino	Für Arduino Funktionen notwendig
Fs.h	GNU LGPL 2.1 oder höher	Ivan Grokhotkov	Ermöglicht Filesystem für den Web-Server
SPIFFS.h	Apache License 2.0	Espressif Systems	Notwendig für das Filesystem des ESP32
iostream	GNU GPL 3.0 oder höher	Free Software Foundation	Braucht es um Daten zu Buffern mit C++
WiFi.h	GNU LGPL 2.1 oder höher	Arduino	Braucht es um ein Wifi einzurichten und ermöglicht SSID und WPA2
esp_wifi.h	Apache License 2.0	Espressif Systems	WIRD benötigt, damit das Wifi unter dem ESP32 läuft
WebServer.h	GNU LGPL 2.1 oder höher	Ivan Grokhotkov	Erstellt den Web-Server
DNSServer.h	Verwendet Bibliothek mit MIT Lizenz	Bjoern Hartmann	Port für den DNS-Server des ESP32
WiFiManager.h	MIT license	github: tzapu	Konfigurieren der WiFi-Zugangsdaten
esp_system.h	Apache License 2.0	Espressif Systems	Konfiguriert die Peripherie des ESP32
stdio.h	Nur Kopieren des Copyright Absatzes	University of California	Beinhaltet grundlegende Funktionen, Makros usw.
time.h	GNU C	-	Makros und Funktionen um mit der Zeit zu rechnen
PubSubClient.h	GNU LGPL 2.1 oder höher	Nick O'Leary	Erstellt einen MQTT Client
driver/adc.h	Apache License 2.0	Espressif Systems	ADC Teiber
ArduinoJson.h	MIT Lizenz	Benoit Blanchon	allgemeine library für C++, wird für den Buffer verwendet

Tabelle 8.1: Verwendete Bibliotheken und deren Lizenzen

9 Verbesserungen

Temperatur

Es wurde ein NTC mit 5 % bei 25 °C Toleranz genommen, besser wäre 1 % gewesen, was nicht viel teurer wäre. Mit der gleichen Berechnung, jedoch mit einer anderen Toleranz, wie unter dem Kapitel 7.1.3, wäre anstatt 19.3 °C 20.0 °C herausgekommen was eine Abweichung von 0.2 °C bei Raumtemperatur ausmachen würde.

Genauigkeit

Der ADC des ESP32 hat einige Tücken. So wurde zwar das Problem, dass Spannungen unter ca. 170 mV nicht gemessen werden können mit einem Widerstandnetzwerk gelöst. Aber es bleibt, das Problem, dass sich verschiedene ESP eine unterschiedliche Spannungen unterschiedlich einlesen. Dies ist deswegen, weil die interne ADC-Referenzspannung des ESP32, zwar als 1100 mV angegeben ist, in Wahrheit aber zwischen 1000 mV und 1200 V schwankt [28]. Dies verdeutlicht die Abbildung 9.1. Industriell lösen lässt sich dieses Problem, wenn man extra vorher kalibrierte ESPs kauft, bei welchen die Referenzspannung in das eFuse Vref gebrannt wurden. Dann kann über eine Funktion, die Referenzspannung ausgelesen werden. Besser wäre es 0 dB Dämpfung für das einlesen einer Spannung zu benutzen, da diese auch wieder Toleranz behaftet ist. Diese interne Dämpfung macht nichts anderes als die angelegte Spannung so stark zu Dämpfen, damit sie kleiner als die Referenzspannung ist. Aktuell ist der Standardwert von 11dB eingestellt. Ein weiteres Problem war, dass die Eingelesenen ADC-Werte sich unterscheiden, je nach dem wie belastet der ESP32 war. So wurde die Kurve 6.26 mithilfe eines Testprogrammes aufgenommen, welches der ESP32 im Gegensatz zur richtigen Software nur gering belastet. Die Ausgabe per PWM, war ebenso Problematisch, denn die obere Amplitude war tiefer, wenn der ESP Belastet wurde, Ursache hierfür kann auch die Versorgungsspannung sein welche um 10m mV bei Belastung tiefer war.

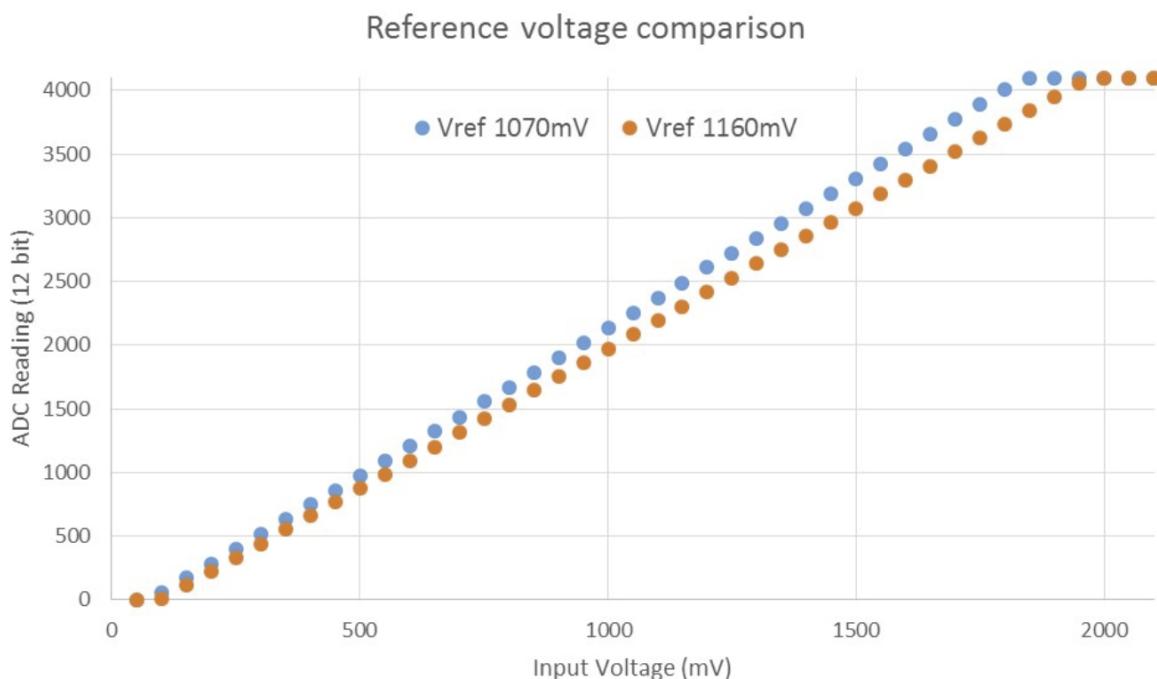


Abbildung 9.1: ADC-Werte unterscheiden sich von ESP zu ESP unterscheiden [28]

Zusammenstecken

Man sollte darauf scheuen, dass die Frontplatte und der Hauptprint des Sensorbausteins richtig

herum zusammengesteckt werden. Es hierzu zwar einen Markierungspunkt und Beschriftungen, die Eindeutig auf die Richtung hinweisen, besser wäre aber wenn es mechanisch nicht anders möglich wäre. Der Sensorbaustein wird durch ein falsches Zusammenstecken nicht beschädigt, nur die Funktion ist beeinträchtigt.

Beschriftung

Beim Aktorbaustein ist die Beschriftung der Taster clear und mode vertauscht.

10 Schluss

10.1 Reflektion

komplett
neu erstel-
len

In diesem Kapitel werden die Teilziele vom Pflichtenheft Reflektiert. In der untenstehenden Tabelle 10.1 sind die zu Beginn des Projekts Definierten Teilziele so wie der Status, in welchem der Fachbericht am Ende verfasst wurde.

Teilziel	Definition	Status
1	Definitive Vereinbarung ist unterschrieben	Erfüllt
2	Analyse und Evaluation Mikrocontroller	Erfüllt
3	Evaluation Funk Verbindung	Erfüllt
4	Evaluation Hardware Komponenten	Erfüllt
5	Analyse von verschiedenen FW-Frameworks	Erfüllt
6	Konzept der HW Infrastruktur erstellt	Erfüllt
7	Funktionsmuster, Fliegender Aufbau, Hotspotfunktion für Grundkonfiguration, MQTT Nachrichten können versendet werden	Erfüllt
8	Evaluation Server Software	Erfüllt
9	Dokumentation abgeschlossen	Erfüllt

Tabelle 10.1: Teilziele

Die Teilziele wurden erreicht, wobei im Verlauf des Projekts die Erkenntnis gewonnen wurde, dass die Evaluationen von Frameworks Teilziel 5 und Evaluation Server Software Teilziel 8, sehr viel Zeit in Beanspruchten. Im Idealfall hätte ein komplettes Funktionsmuster pro Framework aufgebaut werden sollen, wozu aber nicht genügend Zeit vorhanden war. Es konnte nur gerade ein Framework ausführlich getestet werden. Die Evaluation Server Software wurde auf zwei Favoriten beschränkt. Die Meilensteine konnten erreicht werden, wobei die der Termin Plan vom Projekt zeitlich nicht eingehalten wurde, so mussten durch Lösen von verschiedenen Problemen konnte in der Projektwoche nicht alle vorgesehenen arbeiteten erledigt werden und so kamen alle nachstehenden Arbeiten in Verzug.

10.2 Ausblick

Damit nicht die gleichen Probleme wie im Vergangenen Projekt auftauchen, wird ein Fixer Tag für das Projekt vorgesehen, Voraussichtlich ist dies der Samstag. Weil so viel Zeit zum Lösen von Problemen drauf gegangen war, wurde vorgenommen vermehrt die Hilfe vom Betreuer beanspruchen, somit sollen mehr Sitzungen stattfinden als im Projekt 5.

11 Ehrlichkeitserklärung

Hiermit erklären wir, dass die vorliegende Arbeit selbstständig von uns, ohne Hilfe Dritter und nur unter Benutzung der angegebenen Quellen verfasst wurde.

Projekt Team

Lukas Meienberger und Gabriel Nussbaumer

Ort, Datum

Unterschriften

Literatur

- [1] D. Wetherall und A. Tanenbaum, *Computernetzwerke*, 5. Aufl. München: Pearson Deutschland GmbH, 2012, ISBN: 978-3-86894-137-1.
- [2] (). Wi-Fi Alliance® Introduces Wi-Fi CERTIFIED WPA3™ Security | Wi-Fi Alliance, Adresse: <https://www.wi-fi.org/news-events/newsroom/wi-fi-alliance-introduces-wi-fi-certified-wpa3-security> (besucht am 13. Jan. 2020).
- [3] (). Mqtt-v5.0.Pdf, Adresse: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf> (besucht am 12. Jan. 2020).
- [4] steve. (). Understanding the MQTT Protocol Packet Structure, Adresse: <http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/> (besucht am 13. Aug. 2020).
- [5] (). OSGi, Adresse: <https://www.openhab.org/docs/developer/osgi/osgi.html> (besucht am 12. Aug. 2020).
- [6] (). Event Bus, Adresse: <https://www.openhab.org/docs/developer/utils/events.html> (besucht am 13. Juli 2020).
- [7] (). Durchbruch, Adresse: <https://www.innoq.com/de/articles/2014/11/durchbruch/> (besucht am 13. Aug. 2020).
- [8] randomnerdtutorials. (2. Okt. 2019). ESP32 Pinout Reference: Which GPIO pins should you use?, Adresse: <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/> (besucht am 28. Nov. 2019).
- [9] (). Visual Studio Code - Code Editing. Redefined, Adresse: <https://code.visualstudio.com/> (besucht am 8. Jan. 2020).
- [10] PlatformIO. (). PlatformIO is a new generation ecosystem for embedded development, Adresse: <https://platformio.org> (besucht am 8. Jan. 2020).
- [11] (). Arduino - AboutUs, Adresse: <https://www.arduino.cc/en/Main/AboutUs> (besucht am 1. Dez. 2019).
- [12] zhouhan0126, *Zhouhan0126/WIFIMANAGER-ESP32*, 15. Dez. 2019. Adresse: <https://github.com/zhouhan0126/WIFIMANAGER-ESP32> (besucht am 28. Dez. 2019).
- [13] (). Arduino Client for MQTT, Adresse: <https://pubsubclient.knolleary.net/api.html#publish1> (besucht am 18. Jan. 2020).
- [14] (). Introduction, Adresse: <https://www.openhab.org/docs/> (besucht am 6. Aug. 2020).
- [15] (). Xtext - Language Engineering Made Easy!, Adresse: <http://www.eclipse.org/Xtext/#xbase> (besucht am 6. Aug. 2020).
- [16] H. Assistant. (). Google Assistant, Adresse: https://www.home-assistant.io/integrations/google_assistant/ (besucht am 13. Aug. 2020).
- [17] (). Touch Sensor - ESP32-S2 - — ESP-IDF Programming Guide Latest Documentation, Adresse: https://docs.espressif.com/projects/esp-idf/en/latest/esp32s2/api-reference/peripherals/touch_pad.html (besucht am 14. Juli 2020).
- [18] Espressif, *Esp32-Wroom-32_datasheet_en*, Sep. 2019.
- [19] Hager, *Hager_Kat1_08_Technik_D*, 2019.
- [20] Mikrocontroller.net. (). Relais Mit Logik Ansteuern – Mikrocontroller.Net, Adresse: https://www.mikrocontroller.net/articles/Relais_mit_Logik_ansteuern (besucht am 2. Jan. 2020).
- [21] A. Ronald. (15. Dez. 2017). Löcher für Steckdosen bohren | Vorgangsweise, Tipps und Tricks, Adresse: <https://richtig-bohren.net/loecher-fuer-steckdosen-bohren> (besucht am 28. Nov. 2019).
- [22] L. Miller. (). RC-Glied Für PWM, Adresse: <http://www.lothar-miller.de/s9y/categories/13-RC-Glied-fuer-PWM> (besucht am 31. Dez. 2019).
- [23] P. Schleuniger, „EMV_W8“, 8. Apr. 2020.

- [24] (). Gnu.org, Adresse: <https://www.gnu.org/licenses/why-not-lgpl.html> (besucht am 13. Aug. 2020).
- [25] *Copyleft*, in *Wikipedia*, 31. März 2020. Adresse: <https://de.wikipedia.org/w/index.php?title=Copyleft&oldid=198295642> (besucht am 13. Aug. 2020).
- [26] *MIT License*, in *Wikipedia*, 9. Aug. 2020. Adresse: https://en.wikipedia.org/w/index.php?title=MIT_License&oldid=971968362 (besucht am 13. Aug. 2020).
- [27] *Apache License*, in *Wikipedia*, 4. Aug. 2020. Adresse: https://en.wikipedia.org/w/index.php?title=Apache_License&oldid=971184773 (besucht am 13. Aug. 2020).
- [28] (). Analog to Digital Converter - ESP32 - — ESP-IDF Programming Guide Latest Documentation, Adresse: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html> (besucht am 13. Aug. 2020).