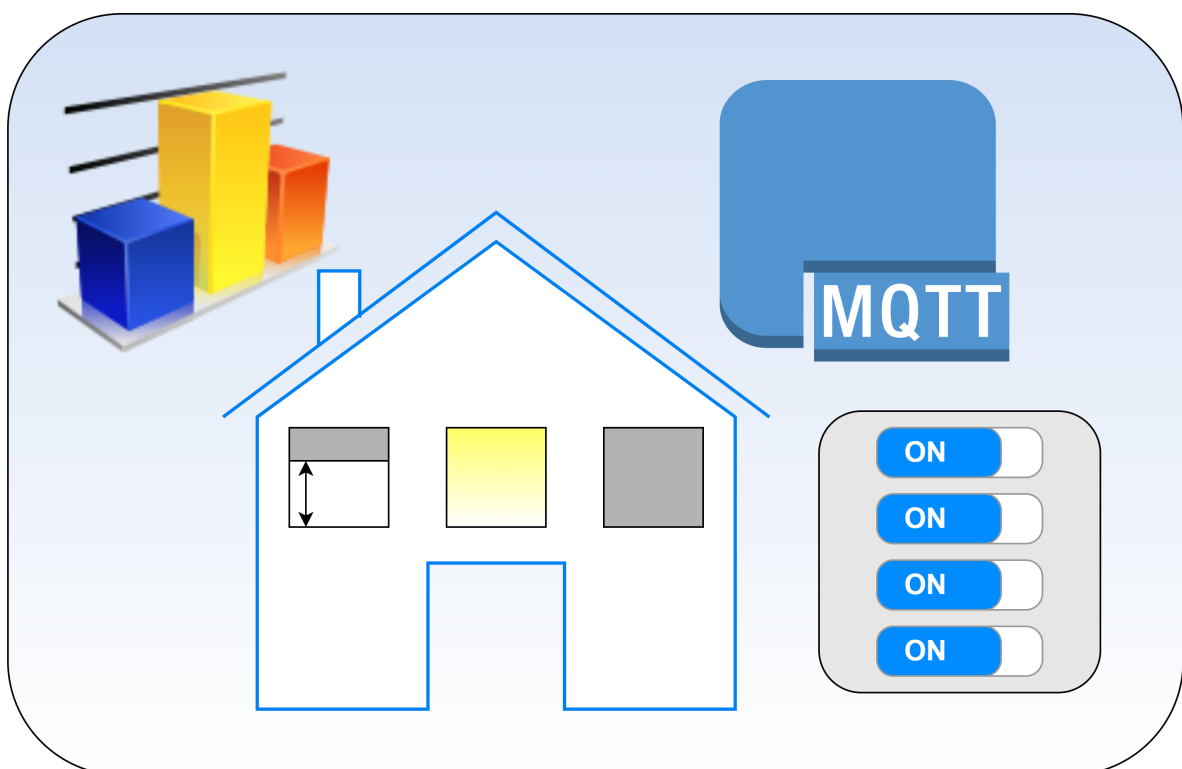


Fachbericht

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	Albert Zihlmann
Auftraggeber	Albert Zihlmann
Version	1.0

Zusammenfassung

Der Softwareteil befasst sich mit verschiedenen Komponenten, so wird ein Framework für den Sensor/Aktorbaustein wie auch das Framework für den Inhouse Server und das MQTT Konzept beschrieben. Das Hardwarekonzept des Sensorbausteins besteht im wesentlichen aus einem WiFi fähigen Mikrocontroller, einem Temperatursensor, Buttons und LEDs. Dabei besteht der Sensorbaustein aus drei physisch getrennten Teilen: der Spannungsversorgung, dem Hauptprint und der Frontprint. Der Frontprint beinhaltet Touchsensoren und LEDs, welche einen individualisierten Einschaltzustand, z.B Lüftung ist an, signalisieren. Das wichtigste im Aktorbaustein ist ebenso ein Mikrocontroller mit WiFi, dazu kommen noch Relais, 10 V Ein-/Ausgänge und LEDs, um den Schaltzustand der Relais zu signalisieren.

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	Architektur	8
4.1.1	OSGI	8
4.1.2	JAR	10
4.1.3	Übersicht Kommunikation Verbindungen	10
5	Software	11
5.1	Mikrocontroller	11
5.1.1	ADC	11
5.2	Recherche	12
5.2.1	ESP32	12
5.2.2	CC3235	14
5.3	Wahl des MCU	15
5.4	Framework	16

5.4.1	Entwicklerumgebung	16
5.4.2	Framework Arduino	17
5.4.3	Framework Zephyr	17
5.4.4	Framework ARM-MBED	17
5.4.5	Framework esp-idf	17
5.4.6	Wahl Framework	17
5.4.7	Smart-Home Plattform	17
5.5	Kommunikation	19
5.5.1	WLAN Konfiguration	19
5.5.2	Lizenzen	24
5.6	Server und Broker	24
6	Hardware	27
6.1	Sensorbaustein	27
6.1.1	Übersicht Frontplatte	27
6.1.2	Übersicht Hauptplatine	28
6.1.3	Schutzbeschaltung	29
6.1.4	Mikrocontroller	29
6.1.5	Temperatursensor	29
6.1.6	Programmierschluss	31
6.1.7	Spannungsversorgung	32
6.1.8	Platzstudie	32
6.2	Aktorbaustein	33
6.2.1	Mikrocontroller	34
6.2.2	Relais	35
6.2.3	Eingänge/Ausgänge 0..10 V	35
6.2.4	Spannungsversorgung	37
6.2.5	Interface	38
7	Schluss	39
7.1	Reflektion	39
7.2	Ausblick	39
8	Ehrlichkeitserklärung	40
	Literatur	41

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. Bisher 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 die Plattform 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

Tabelle 1.1: Ausgangslage

2 Gesamtübersicht

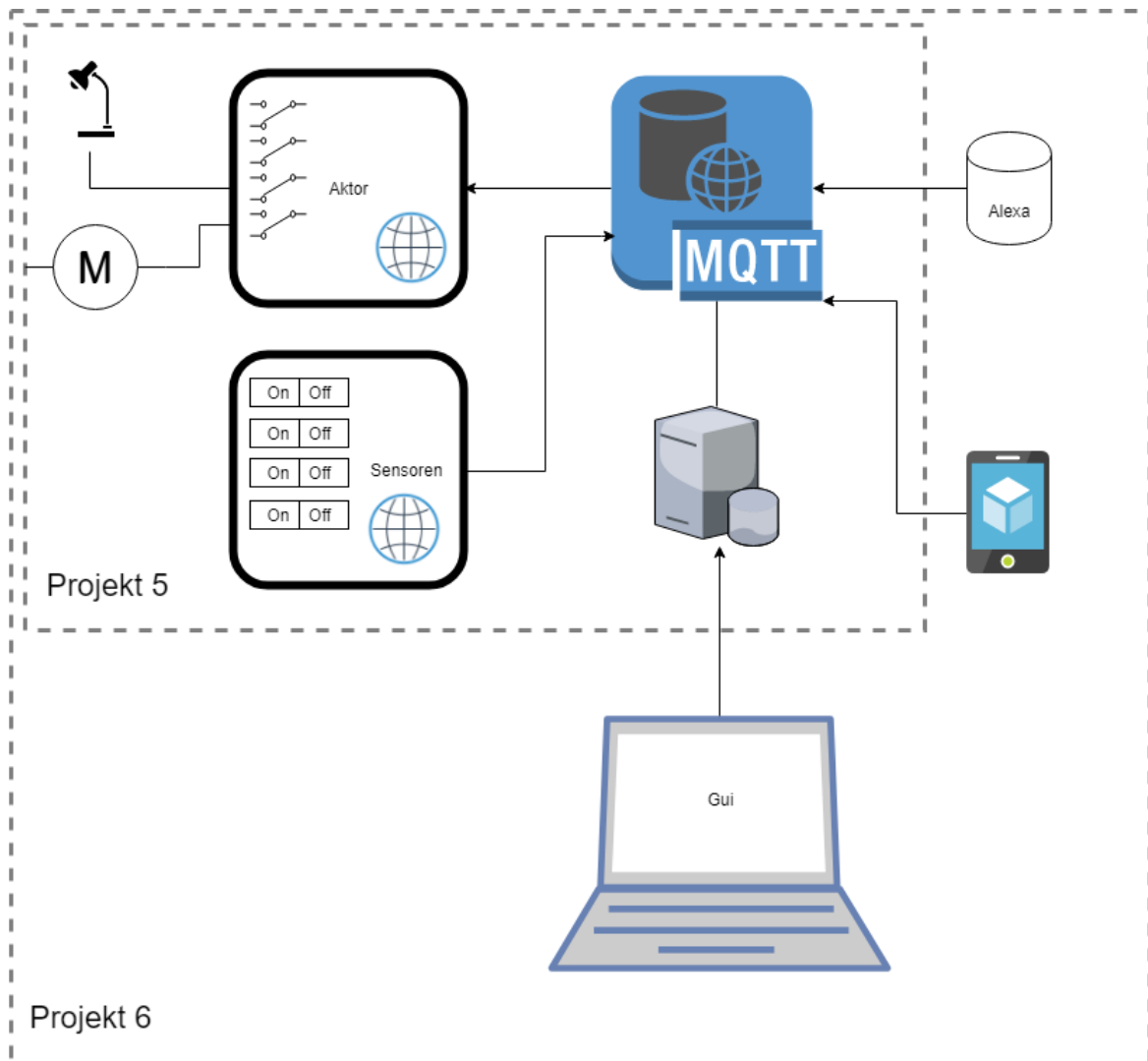


Abbildung 2.1: Gesamtübersicht Projekt 5 und Projekt 6

In der Abbildung 2.1 sind im inneren Quadrat die Komponenten, für welche im Projekt 5 die Evaluationen ausgeführt werden. Dabei ist ein Aktor- wie auch ein Sensor-Print, die Kommunikation mit dem MQTT-Protokoll sowie der Inhouse Server für die Datenerfassung und Systemkonfigurationen enthalten. Im Projekt 6 werden die geplanten Komponenten ausgeführt und erweitert. Im Bereich welcher erweitert werden soll, ist eine Benutzeroberfläche (GUI) wie auch eine Steuerung mittels Smartphone und als Wunschziel Sprachsteuerung inbegriffen.

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 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.

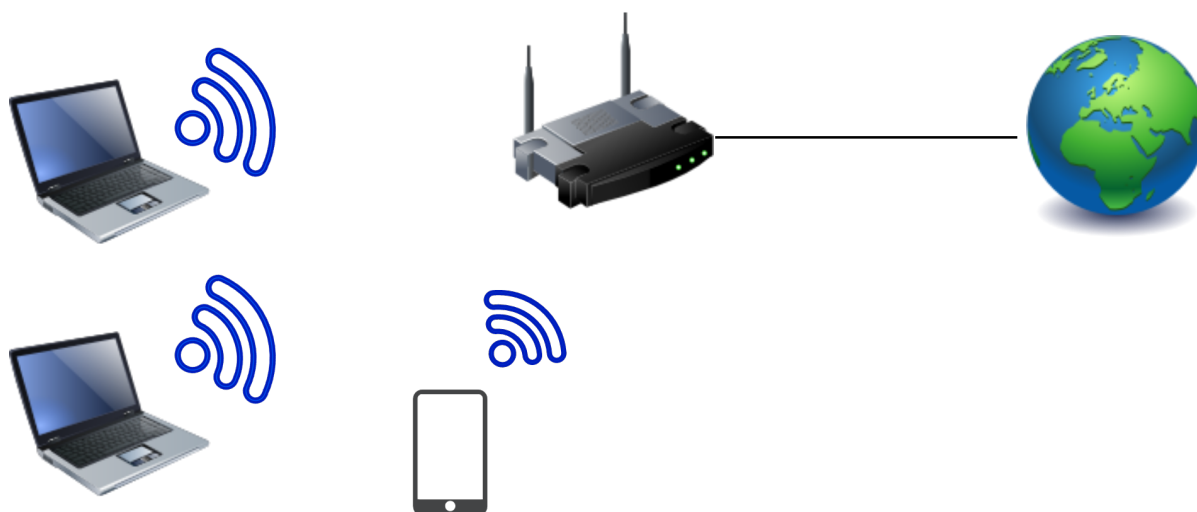


Abbildung 3.1: Übersicht IEEE 802.11

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 überlappende 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 ab 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. Nach dem 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 kennwortbasierte 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 und abonniert und mit dem MQTT-Server 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 [noauthor_mqtt-v5.0.pdf_nodate](#).

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 [noauthor_mqtt-v5.0.pdf_nodate](#). .

3.2.3 Übersicht

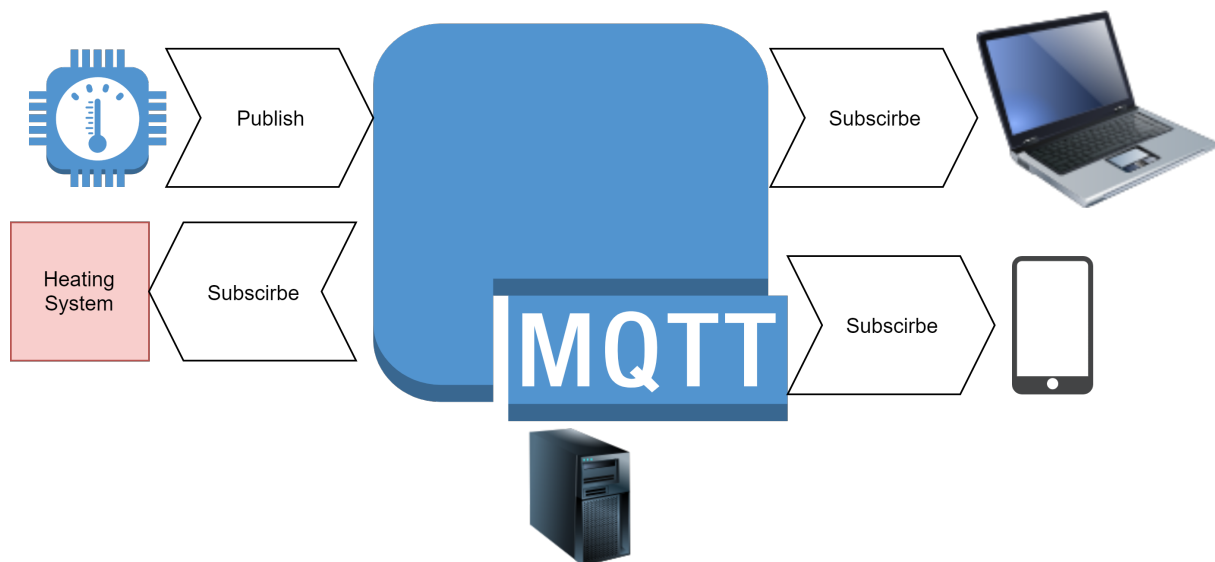


Abbildung 3.2: Übersicht MQTT

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. Ein Client baut eine Verbindung zum Broker auf. Der Client veröffentlicht Nachrichten mit einem Topic und einem Inhalt.

Ist ein Client an bestimmten Daten interessiert abonniert er diesen Topic und empfängt somit den Inhalt.

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 Protokollaus-tausche, 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 **noauthor__mqtt-v5.0.pdf__nodate:**

- 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.

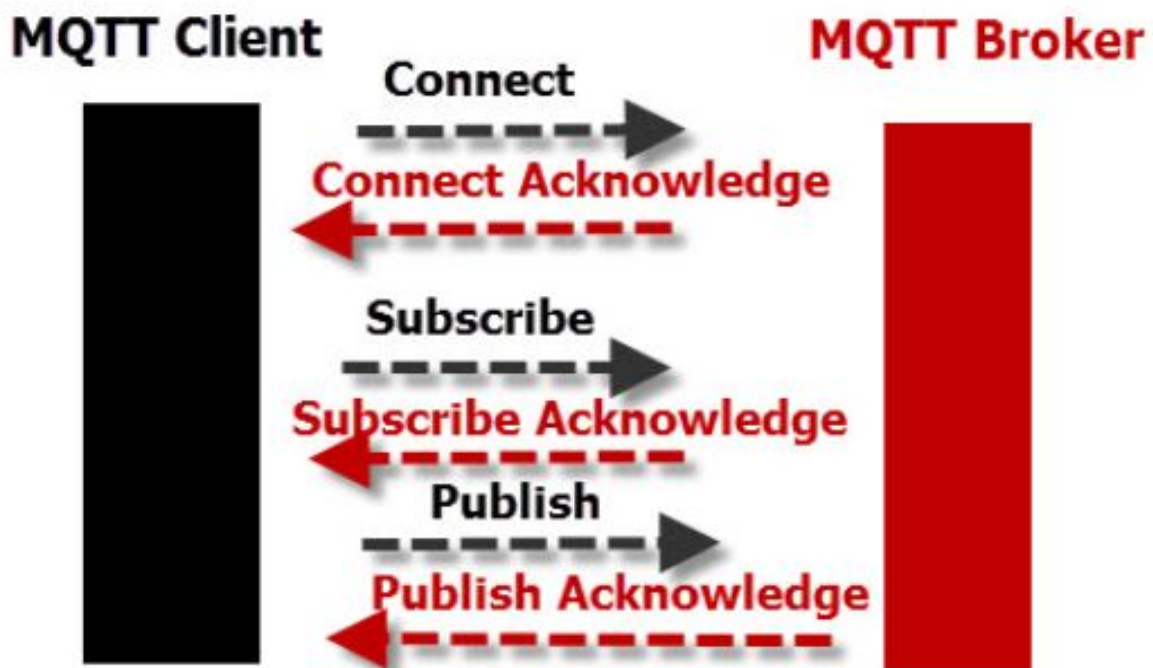


Abbildung 3.3: Protokoll Verbindung Client zu Broker

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

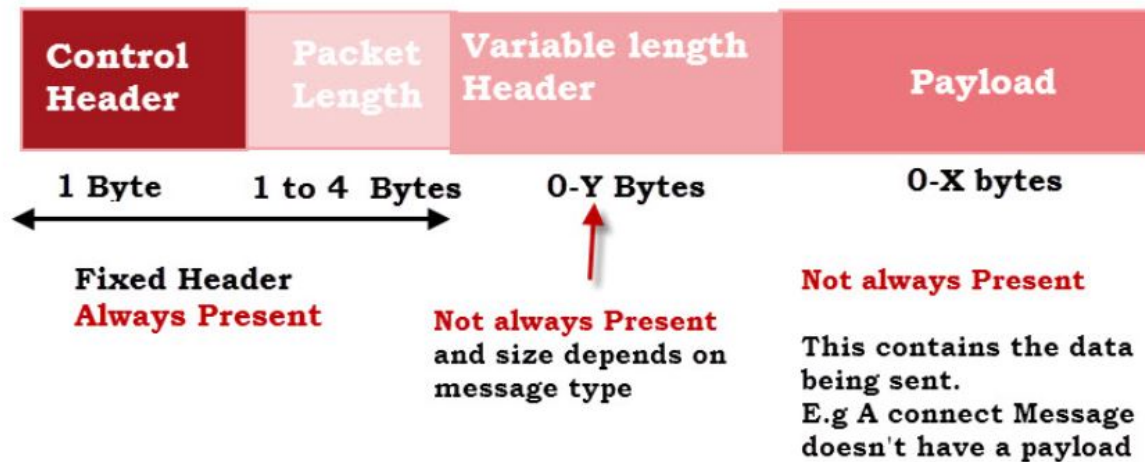


Abbildung 3.4: Standard MQTT Paket Struktur

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 und antworten.

4 OpenHab

In dieser Arbeit wird als Server ein RaspberryPi verwendet. Als Software befindet sich Openhab auf dem Linux Betriebssystem vom RaspberryPi. Openhab ist eine Gebäudeautomation Software und bietet Kontabilität zwischen verschiedenen Smarthome Komponenten wie KNX, Sonos, oder Philips Hue. Der für das Projekt benötigte MQTT Broucker 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 sind wiederverwendbar, sie kommunizieren lokal untereinander. Die OSGI Architektur ermöglicht es den Komponenten die Implementierung vor andern Komponenten zu verbergen, während Dienste kommunizieren über Objekte gerade wenn sie von anderen Komponenten gemeinsam genutzt werden. Dies reduziert die Gesamtkomplexität und ermöglicht eine hohe Zuverlässigkeit, da während laufendem System Wartung und Entwicklungsarbeiten Durchführt werden können.

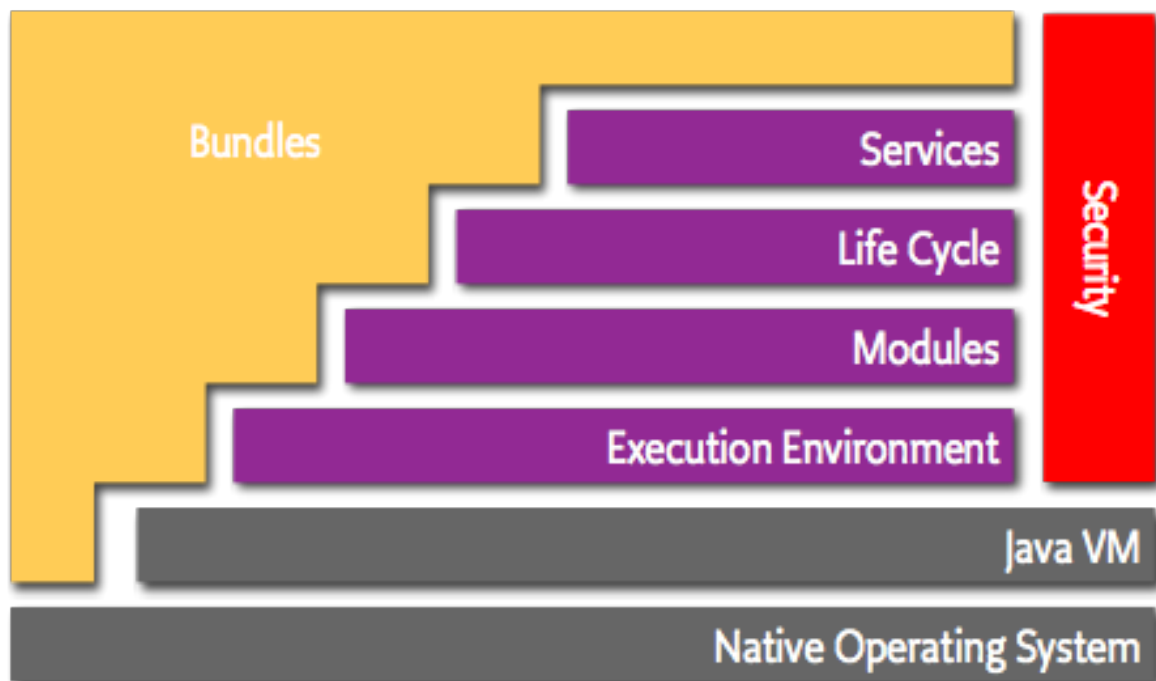


Abbildung 4.1: OSGI Layers <https://www.osgi.org/wp-content/uploads/layering-osgi.png>

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-Standardbibliotheken 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.

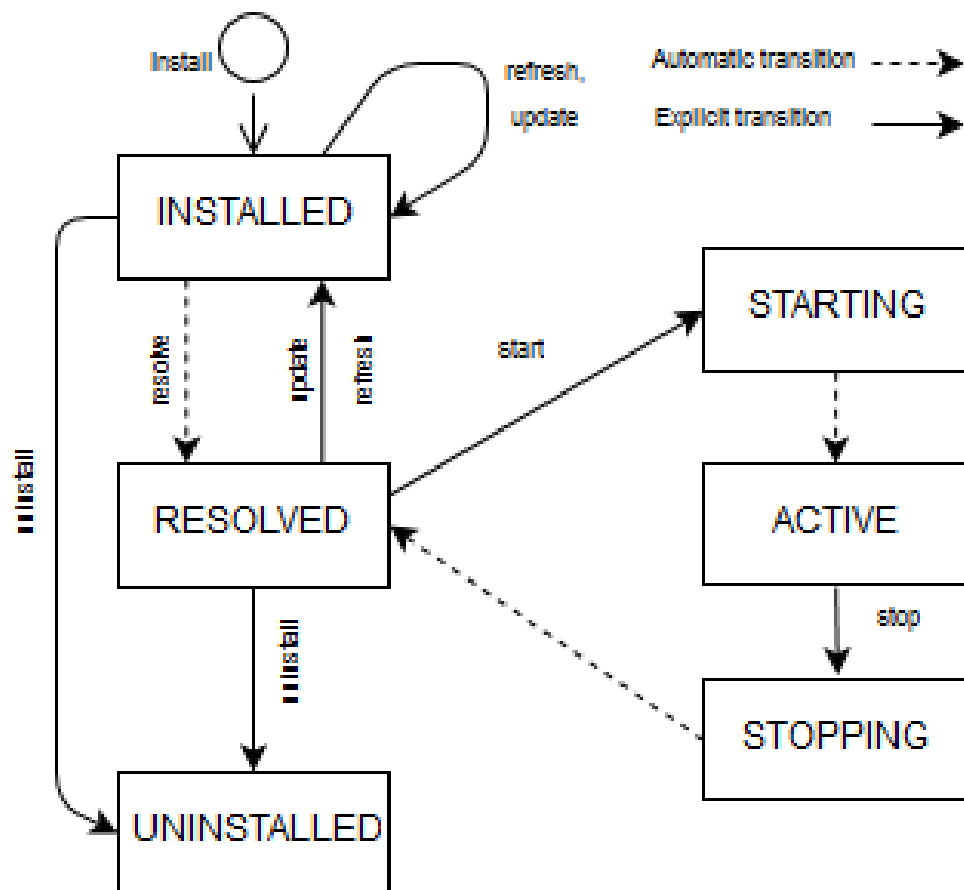


Abbildung 4.2: Bundle State Diagramm

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

4.1.2 JAR

4.1.3 Übersicht Kommunikation Verbindungen

Die Kommunikation zwischen den Komponenten geschieht mittels Event Bus. Alle nicht statusbezogenen Bundels informieren darüber andere Bundles über den Status von Events. Über diesen Bus kommunizieren alle Binding mit einem physischen 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 [3].

<https://www.openhab.org/docs/developer/utis/events.html>

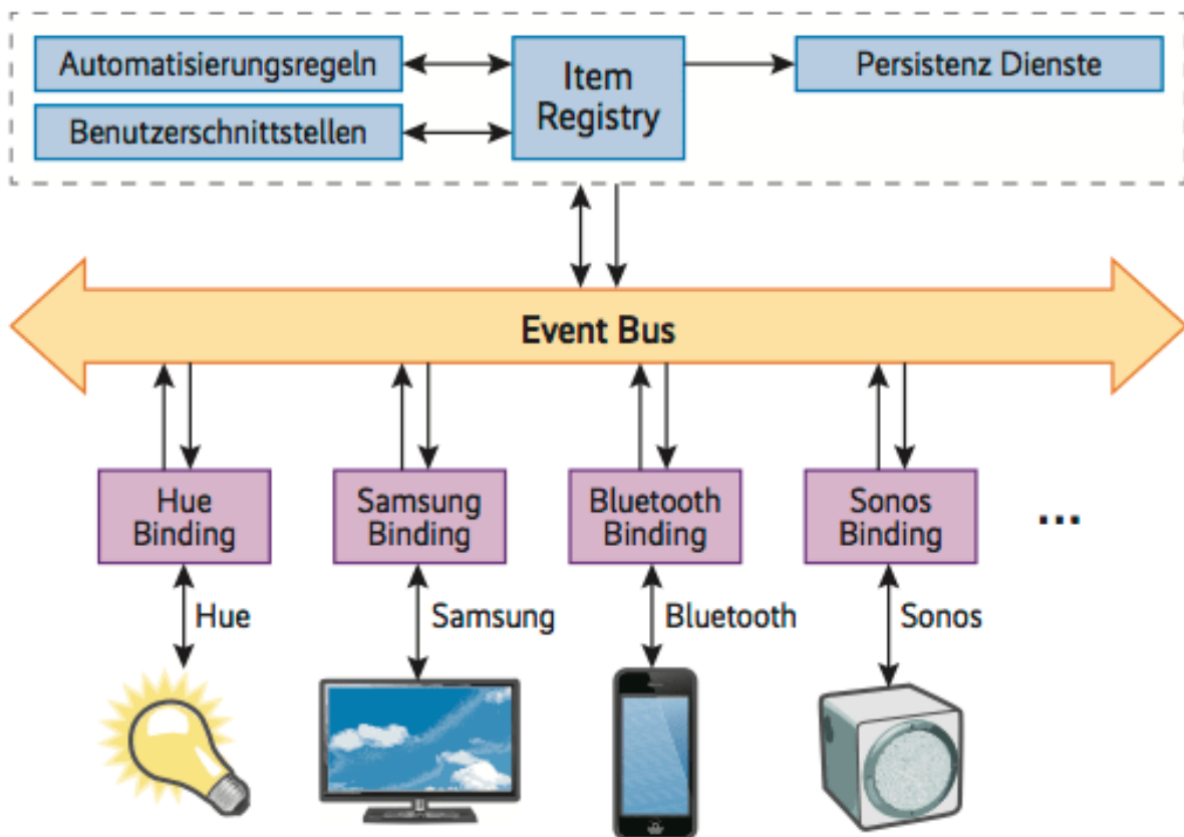


Abbildung 4.3: Eventbus Openhab <https://www.innoq.com/de/articles/2014/11/durchbruch/>

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

Für die Auswahl des Mikrocontrollers, welcher auf dem Sensor wie auch auf dem Aktor-Layout eingebaut wird, werden die nachfolgenden Kriterien in Betracht bezogen. Das erste Kriterium ist die Rechenleistung des Controllers, welche die Aufgaben wie Kommunikation, ADC Wandlungen und einer kleine State Maschine übernehmen muss. Ein weiteres Kriterium ist die Sicherheit, der Mikrocontroller soll den Benutzer schützen, darf somit keinen ungewollten Fremdzugriff auf vorhandene Daten wie auch auf Handlungen oder Schaltvorgänge zulassen. Ein weiteres Kriterium sind die Schnittstellen, welche die Funkverbindung zur Übertragung der Daten während des Betriebes gewährleisten, wie auch die Schnittstelle, bei welcher zu Beginn das Framework geladen wird. Das Pinout des jeweiligen Mikrocontrollers wird in den Unterkapiteln 6.1.4 und 6.2.1 besprochen. Bei der Auswahl wird als letztes Kriterium darauf geachtet, dass die Kosten für den Mikrocontroller nicht wahnsinnig hoch anfallen, sodass sich die Gesamtkosten des Systems in einem Rahmen befindet für eine mögliche Massenproduktion. Die fertig entwickelten Komponenten sollen sich zu einem Konkurrenzfähigem Preis verkaufen lassen.

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 $U = 0.17\text{ V}$ 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: $Wert_{ADC} = 1253.1 \cdot U - 218.54$.

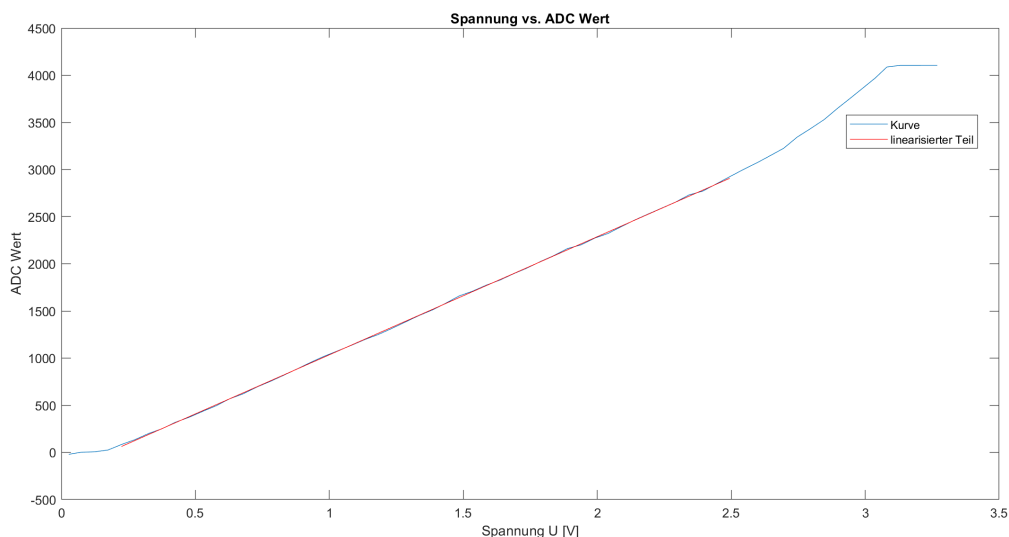


Abbildung 5.1: ESP32 Kurve, Daten aus Grafik von [4]

5.2 Recherche

Es wurden zwei geeignete Mikrocontroller miteinander verglichen, ihre Eigenschaften beschrieben und ausgewertet.

5.2.1 ESP32

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

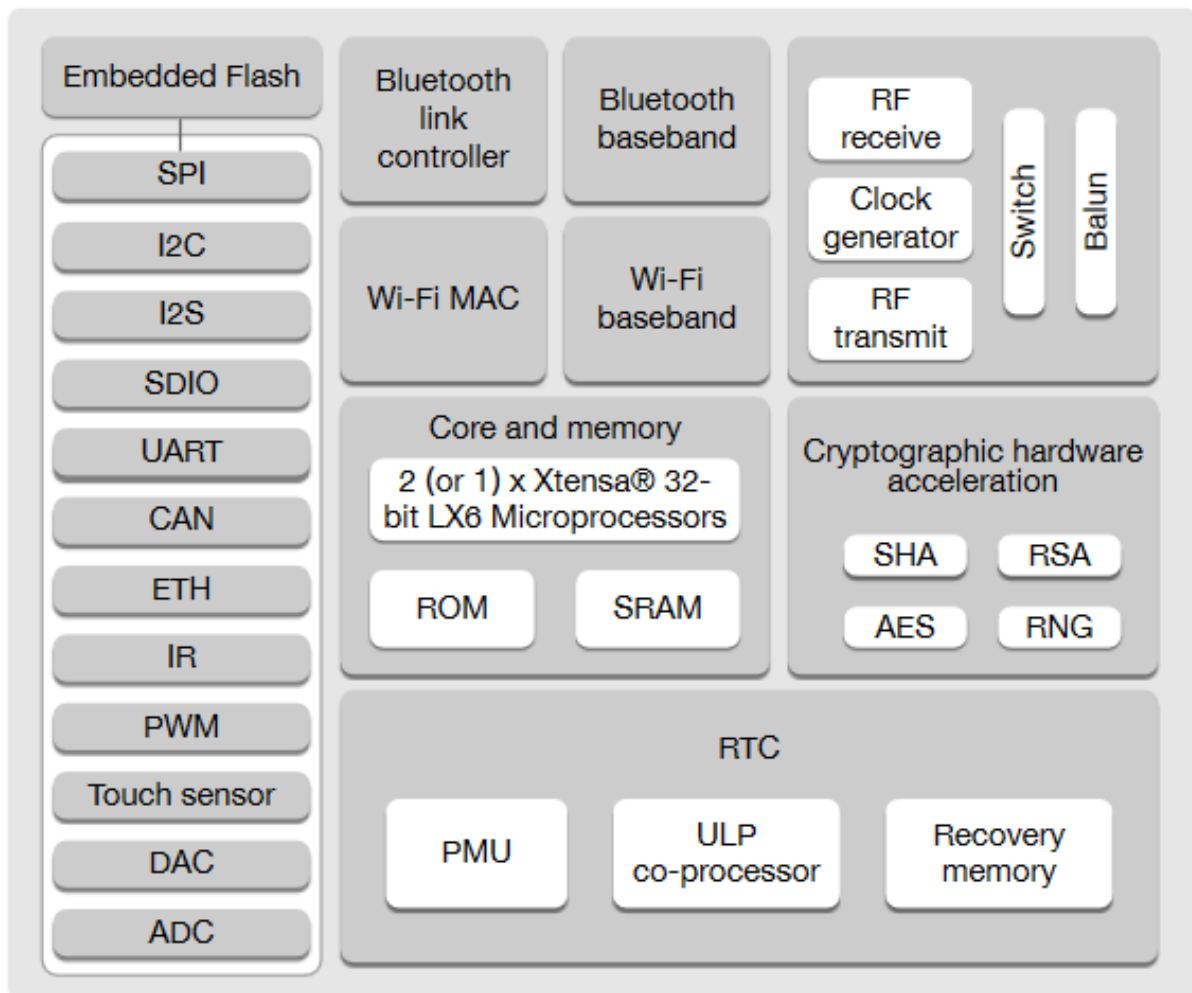


Abbildung 5.2: Blockdiagramm ESP32 [5]

WiFi: Arbeitet mit dem Standard 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 WPA, 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

5.2.2 CC3235

Der CC3235 Controller ist eine Single Chip Lösung von der Firma Texas Instruments. Dieser Chip ist mit einem Arm Cortex M4 MCU ausgestattet. Der CC3235 arbeitet in einer industriellen Umgebung von -40 °C bis 85 °C zuverlässig. Schnittstellen wie UART, I2S, I2C, SPI und I/O Pins sind inbegriffen.

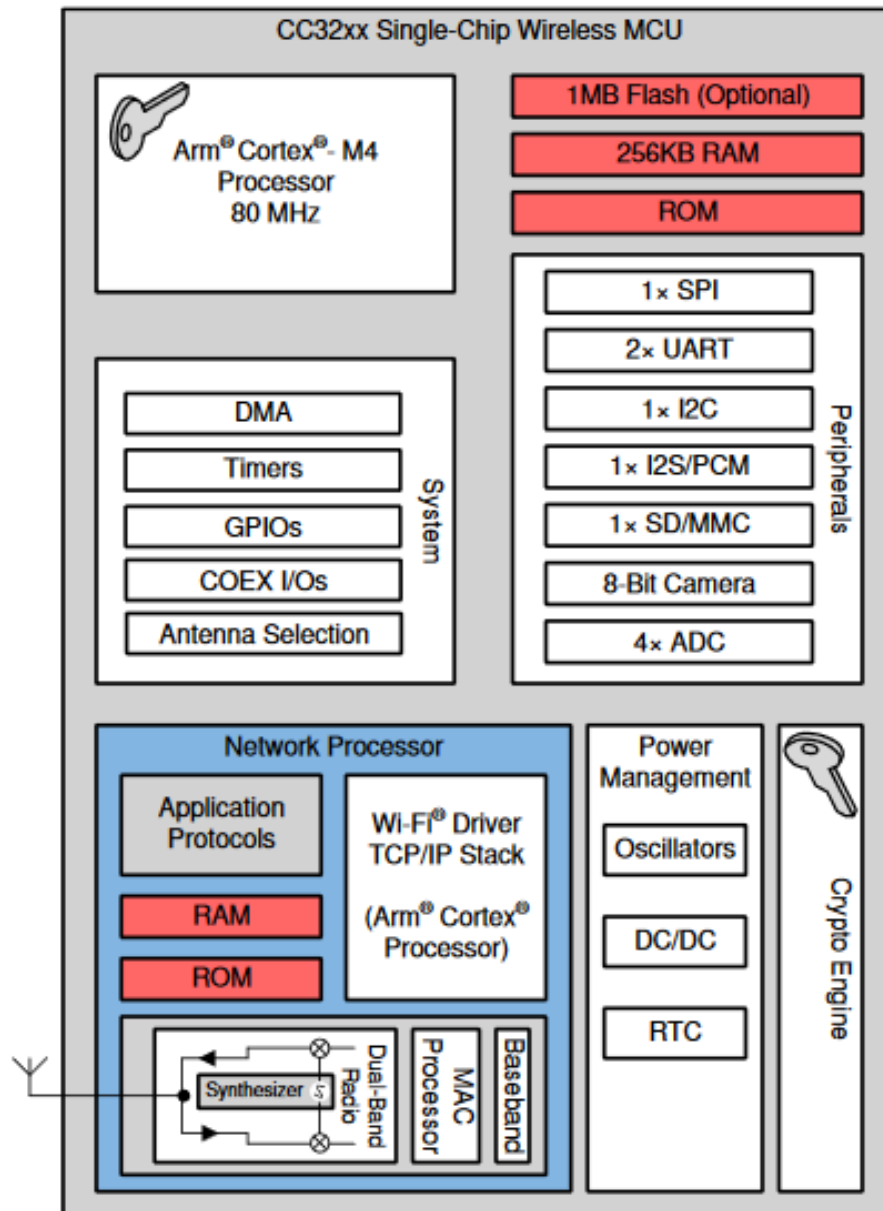


Abbildung 5.3: Blockdiagramm CC3235 noauthor_cc3235s.pdf_nodate

WiFi: Arbeitet mit dem Standard 802.11 a/b/g/n 2.4 GHz und 5 GHz. Mit dem Netzwerkprozessor ist ein AP, Station(STA) sowie WiFi Direct Client inbegriffen.

Bluetooth: Der CC3235x ist für die Unterstützung von BLE/2,4 Funkkoexistenz ausgelegt

CPU und Memory: Modell abhängig bei Betrachtung des Model CC3235SF, 1 MB Flash, 256

KB RAM. Ein 32-bit Arm Cortex-M4 Prozessor

Prozessor: Arm Cortex-M4

Clocks und Timer: Interner 40 MHz Kristal Oszillator, 32,768 kHz Das Generalpurposetimer-module (GPTM) enthält 16- oder 32-Bit-GPTM-Blöcke. Jeder 16- oder 32-Bit Block stellt zwei 16-bit Timer oder Zähler bereit.

Erweiterte Peripherieschnittstellen: 1x SPI, 2x UART, 1x I2C, 1x i2s, 1x SD, 8-Bit Camera, 4x 12-bit ADC und 27 I/O Pins.

Sicherheit: Netzwerk-Passwörter und Zertifikate sind verschlüsselt und signiert. WEP, WPA /WPA2 PSK, WPA2 Enterprise, WPA3 Personal.

Unterstützung bei der Entwicklung: CCS Uniflash ist das Standardwerkzeug zur Programmierung von Texas Instruments MCU.

Kosten bei Mouser: CC3235SF12RGKR 12.91 CHF/Stück

5.3 Wahl des MCU

Im Kapitel 5.2 sind zwei geeignete MCU für diese Projekt gefunden worden. Im Vergleich zwischen dem CC3235 und dem ESP32 hat sich herausgestellt, dass der ESP32 preislich im Vorteil liegt und je nach Modul mit integrierter Antenne und oder SMA Anschluss angeboten wird. Somit fällt die Wahl auf den ESP32 der Firma Espressif Systems.

5.4 Framework

5.4.1 Entwicklerumgebung

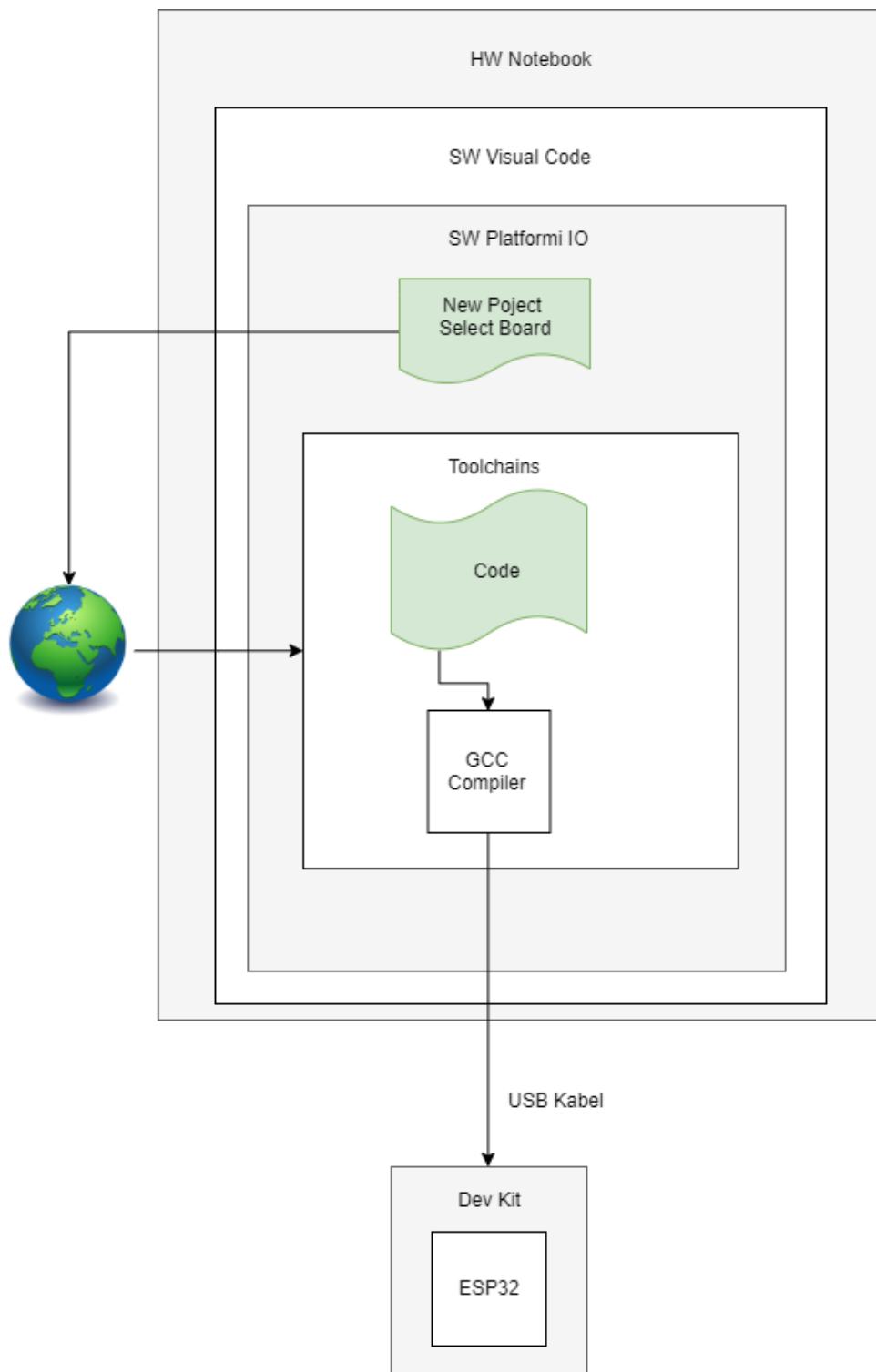


Abbildung 5.4: Entwicklungs Umgebung

In diesem Projekt wurde eine Evaluation mit dem ESP-WROOM-32 Development-Board durchgeführt. Als Entwicklungsumgebung wird Visual Code von der Firma Microsoft verwendet, dies

ist ein freier Quelltext-Editor, mit dem unter anderem Debugging möglich ist [6]. In Visual Code können Extensions eingebunden werden, somit wird PlatformIO IDE [7] kurz PIO Installiert. 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 ESP32 geladen werden, sowie die Serial Port ausgaben können direkt im PIO angezeigt werden. In Abbildung 5.4 ist ein Blockdiagramm auf dem die Verbindungen zwischen Software und Hardware dargestellt sind.

5.4.2 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 [8].

5.4.3 Framework Zephyr

Das Zephyr Os basiert auf einem Kernel mit geringen Platzbedarf. Da das Zephyr modular aufgebaut ist, unterstützt es mehrere Architekturen: Arm Cortex-M, Intel x86, ARC, Xtensa und noch mehr, deswegen ist es sehr universell einsetzbar. Zephyr steht unter der Apache 2.0 Lizenz [9].

5.4.4 Framework ARM-MBED

MBED unterstützt mit mehr als 200'000 Software Entwickler den IoT Markt. Die Grundlegende Hardware eines MBED-Geräts ist ein Arm Mikrocontroller. Ein internes Betriebssystem MBED OS bietet die Möglichkeit Hardware Komponenten anzusteuern oder mit einer Cloud zu verknüpfen. ARM-MBED steht unter der Apache 2.0 Lizenz [10].

5.4.5 Framework esp-idf

ESP-IDF ist das offizielle Farmework für den ESP Chip. ESP-IDF steht unter der Apache Lizenz 2 [11] und dazugehörige Bibliotheken sind auf Github erhältlich.

5.4.6 Wahl Framework

Ein geeignetes Framework für den Mikrocontroller, ist im Umfang des MCU Herstellers und mit einem Prototypen evaluiert worden. Als erste Wahl wurde das Arduino Framework genommen und ein Versuchsaufbau realisiert. Die Wahl des Arduino Framwork 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.

5.4.7 Smart-Home Plattform

Eine Smart-Home Plattform wird benötigt, damit der Benutzer mit dem IOT-Systems interagieren kann. Doch welche Plattform ist für dieses Projekt am besten geeignet? Angedacht ist, dass mithilfe der ausgesuchten Smart-Home-Plattform folgende Bedingungen erfüllt werden: die Plattform...

1. bietet Schnittstellen mit anderen Smart-Home Technologien wie KNX.
2. ist etabliert (grosse Community).
3. ist für Anfänger geeignet.
4. kann als Client auf vielen verschiedenen Geräten laufen.
5. verfügt über ein übersichtliches GUI.
6. hat Keine Darstellungsfehler auf verschiedenen Geräten.
7. Hat gewisse Vorteile gegenüber anderen Plattformen.

Jetzt kommt eine Analyse der Smart-Home-Plattformen. Vorgängig zu vermerken ist, dass es sich als positiv herausgestellt hat, dass alle Plattformen auf einem RasPi installiert werden können.

openHAB / Eclipse Smart-Home:

1. 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 Alexa und KNX sind somit vorhanden. Was ebenso interessant ist, sind konfigurierbare Features wie Dropbox Support, Google Kalender, Text to speech-Implementierung (TTS) und HABDroid. HABDroid kann wie Alexa per Sprachsteuerung Aktionen ausführen. [12]
2. openHaB hat eine sehr grosse Community, wobei die ersten Smart-Home Produkte für den Massenmarkt auf Eclipse Smart-Home basierten. [12]
3. Wie sich herausgestellt hat, ist die Plattform für Anfänger sehr gut zu verstehen.
4. openHAB kann auf jedem Endgerät installiert werden, welches Java Code ausführen kann und genügend Ressourcen hat. [12]
5. Das Gui wird gestaltet durch den Anwender und so kann man sich auch ein Kachelsystem aufbauen, wenn man zu viele Kacheln hat kann es schnell Mal unübersichtlich werden.
6. Auf der Android App von openHab gibt unter dem Menü HABPanel Darstellungsfehler der Kacheln, so dass sie sich überlappen, was unschön ist. Jedoch müssen nicht die Kacheln im HABPanel benutzt werden, sondern es kann eine gute Menüführung mit eigens erstellten Untermenüs direkt auf dem Main Menu verwendet werden.
7. Die MQTT-Anbindung auf einem RasPi gestaltet sich mithilfe von openHab als besonders einfach, denn es kann nach der Installation von openHab direkt einen Broker unter openHab angelegt werden. Hierfür wird als erstes das MQTT Binding unter Add-ons installiert. Danach wird unter Inbox ein neues Gerät (MQTT Thing Binding) hinzugefügt. Das System sucht nun einen Broker, wenn er nichts gefunden hat, kann dann über manuelles Hinzufügen (add manually) einen neuen MQTT Broker hinzugefügt werden. Unter dem Konfigurationsmenü des MQTT Brokers kann unter anderem einen Namen, eine Thing ID, einen Broker Hostname, einen Username, ein Passwort sowie einen Broker Port vergeben werden. Nach der Bestätigung der Konfiguration hat man dann den Broker angelegt. In dem Menu Things kann danach ein neues Gerät hinzugefügt werden. Für das Hinzufügen kann hier MQTT Thing Binding -> manuelles hinzufügen ausgewählt werden. Nun wählt man Generic MQTT THing aus vergibt dort einen Namen und Thing ID, als Bridge wird nun der zuvor angelegte MQTT Broker ausgewählt. Nach der Bestätigung des neuen Gerätes (Thing) kann ein neuer Channel hinzugefügt werden. So kann z.B. ein Gerät welches nur ein oder aus sein muss als Channel Type On/Off Switch hinzugefügt werden. Den Channel ID und das Label können frei gewählt werden. Hier wird ebenso der MQTT state topic (für das lesen) sowie MQTT command topic (für die Befehle) eingegeben werden. Um eine Interaktion zu gewährleisten wird der Kanal mithilfe des Link Channel mit einem Item verknüpft. Unter dem Menü Control sieht man dann eine grafische Darstellung des Items hier z.B der oben genannte On/Off Switch [12].

iobroker:

1. Es sind Schnittstellen vorhanden jedoch weniger als bei openHAB
2. Ist mittlerweile ebenso etabliert, Community ist aber noch ein wenig kleiner als openHAB
3. iobroker bietet, wie openHAB die Möglichkeit auch ohne Programmierkenntnisse bedient zu

werden, jedoch kann auch Javascript angewendet werden, wodurch alle möglichen Wünsche abgedeckt werden können[13]. iobroker benötigt auf den ersten Blick sicherlich mehr Einarbeitungszeit im Gegensatz zu openHAB.

4. Dies ist möglich, jedoch gestaltet sich die CLient Benutzung auf dem Smartphone aufwändiger als bei openHAB

5. Das GUI ist sehr Individualisiert und es können mittels iobroker.vis viele eigene grafische Darstellungen erstellt werden. Jedoch sieht die Oberfläche von iobroker auf den ersten Blick ältermodisch aus.

6. Es gibt soweit keine Darstellungsfehler in iobroker.

7. iobroker bietet mit iobroker.vis ein Visualisierungswerkzeug bereit, welche die Möglichkeit bietet eigene Bedienoberflächen anzulegen. Dabei können die Visualisierung alle Möglichen Komplexitätsgrade annehmen und dazu mit JavaScript verknüpft werden. Beispielsweise kann man ebenso mithilfe von JavaScript eine Schaltfläche in der Vis-Visualisierung ausführen lassen und sich eine Nachricht über Telegram verschicken [14].

Das Fazit der Analyse ist, dass openHAB einfacher zu bedienen ist und eine bessere Anbindung per Smartphone hat als iobroker. Jedoch bietet ioBroker mit iobroker.vis mächtige Werkzeuge, die Frage ist nur, ob das für dieses Projekt benötigt wird.

5.5 Kommunikation

5.5.1 WLAN Konfiguration

Die Funkkommunikation zwischen Sensor, Aktor und MQTT-Broker wird mittels Wireless Local Area Network (WLAN) auf dem OSI layer 1 stattfinden. Um die einzelnen Target also Sensoren und Aktoren in das Lokale Netzwerk zu verbinden, wird beim Starten des ESP32 ein Accespoint eröffnet. Für diesen WiFi-Manager Prozess wird eine Bibliothek eingebunden. Die benötigte Bibliothek steht auf Github zur Verfügung [15].

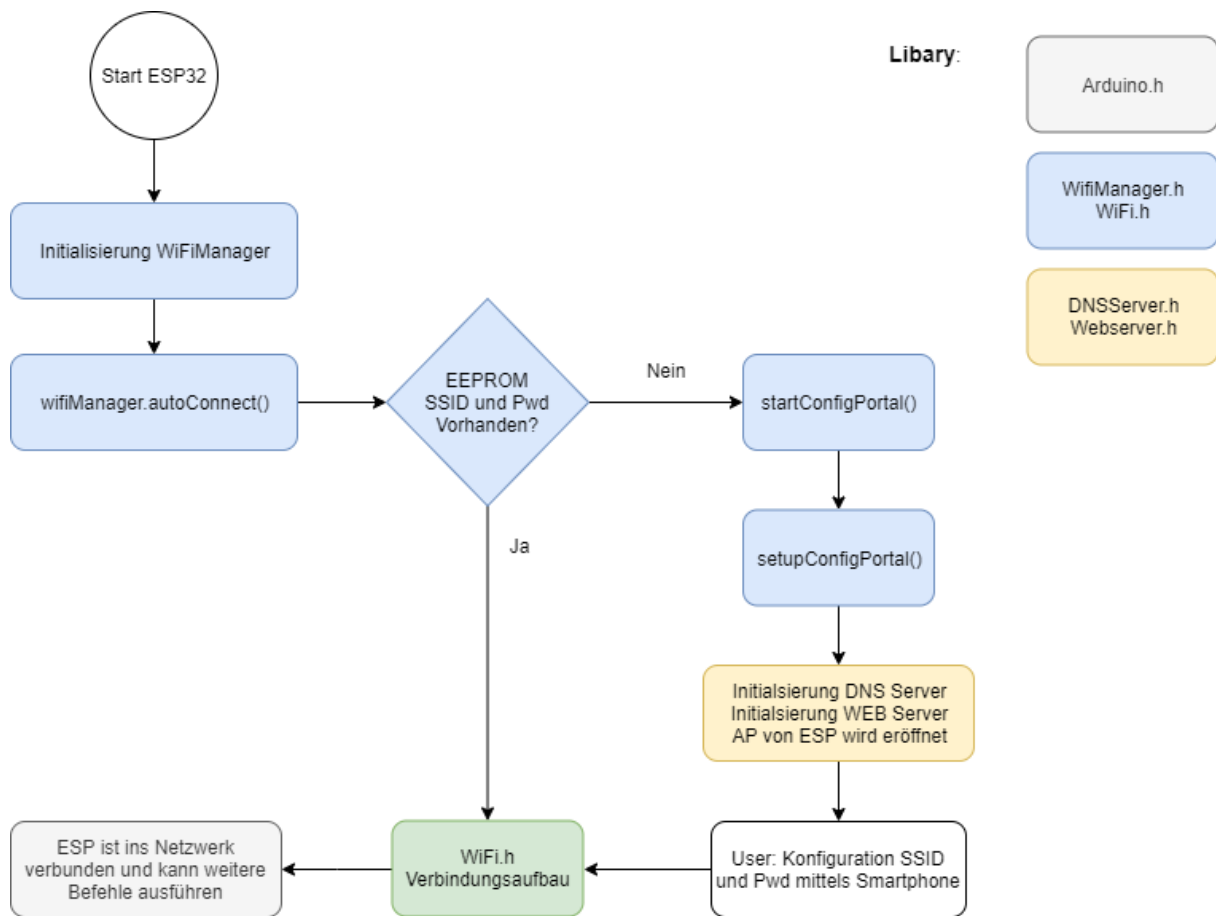


Abbildung 5.5: Statediagramm Verbindungsaufbau ESP ins Netzwerk mit einem AP

Wird die Bibliothek WiFiManager.h eingebunden und initialisiert stehen verschiedene Funktionen zur Verfügung. Die Funktion `autoConnect()` eröffnet nach dem Start des ESP einen Access Point wenn keine Konfigurationsdaten im nichtflüchtigen Speicher vorhanden sind. Falls Konfigurationsdaten vorhanden sind werden sie aus dem EEPROM gelesen und es wird kein Access Point eröffnet. Sollen aber die Konfigurationen bei jedem Start durchgeführt werden, kann die Funktion `startConfigPortal()` aufgerufen werden.

Der Access Point kann bei Bedarf auch passwortgeschützt verwendet werden. In diesem Versuch wird er ohne Passwort genutzt und ist somit mit einem Smartphone oder Notebook ersichtlich, siehe Abbildung 5.6. Wird kein Name zugewiesen, generiert der ESP selber ein Name mit ESP und Chip-ID.

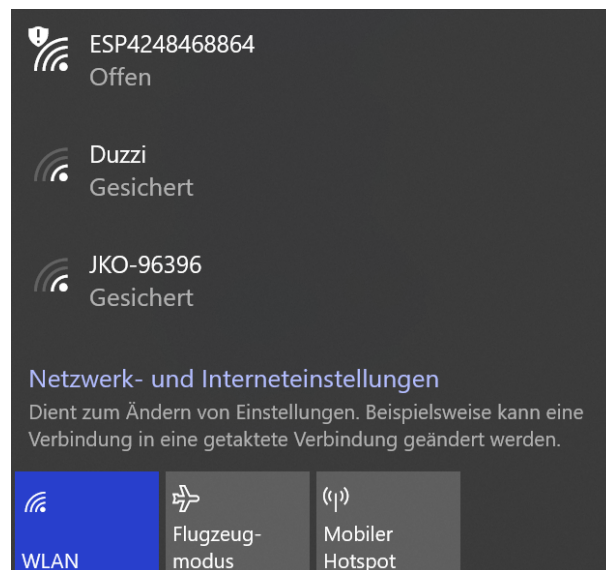


Abbildung 5.6: Esp Acces Point

Sobald "verbinden" mit diesem Netzwerk gewählt wird, startet der Browser eine Konfigurationsseite mit der IP Adresse des Targets. Nun ist ein Menü ersichtlich mit folgenden Wahlmöglichkeiten, siehe Abbildung 5.6:

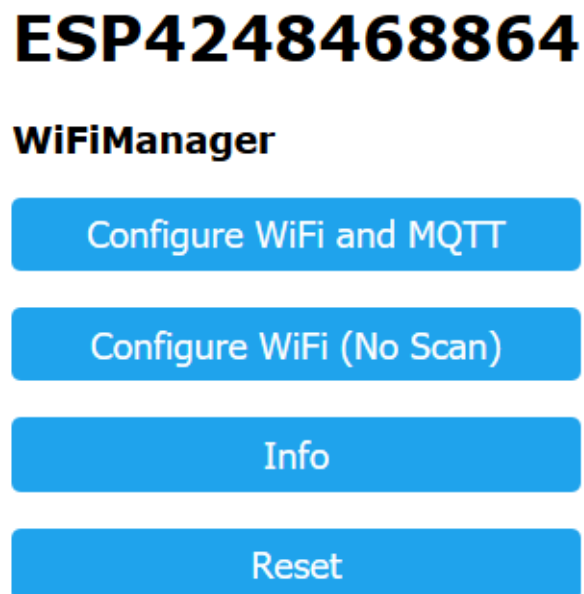


Abbildung 5.7: Esp Konfiguration Home Menu

Wird 'Configure WiFi and MQTT' angewählt erscheint folgende Ansicht in Abbildung 5.8.

QL-81449	🔒 44%
HUALTER	🔒 22%
JKO-96396	🔒 14%

SSID

password

revpi01

ganzeasy

save

[Scan WiFi](#)

Abbildung 5.8: Esp Konfiguration WiFi und MQTT

Netzwerke, welche in der Umgebung gefunden wurden, werden angezeigt, die Parameter die zur WiFi Verbindung notwendig sind, können nun eingegeben werden. Im unteren Teil wird eine Voreinstellung der MQTT-Broker-Adresse angezeigt und dessen Passwort, diese können beliebig geändert werden. Im WiFiManager Hauptmenü können die gleichen Konfigurationen ohne, dass nach vorhandenen Netzen gesucht wird, mit dem 'Configure WiFi(No Scan)' Button, gemacht werden. Mit dem Info Button werden Folgende Informationen angezeigt ChipID, Soft AP IP und Soft AP MAC wie auch Station AP MAC Adresse.

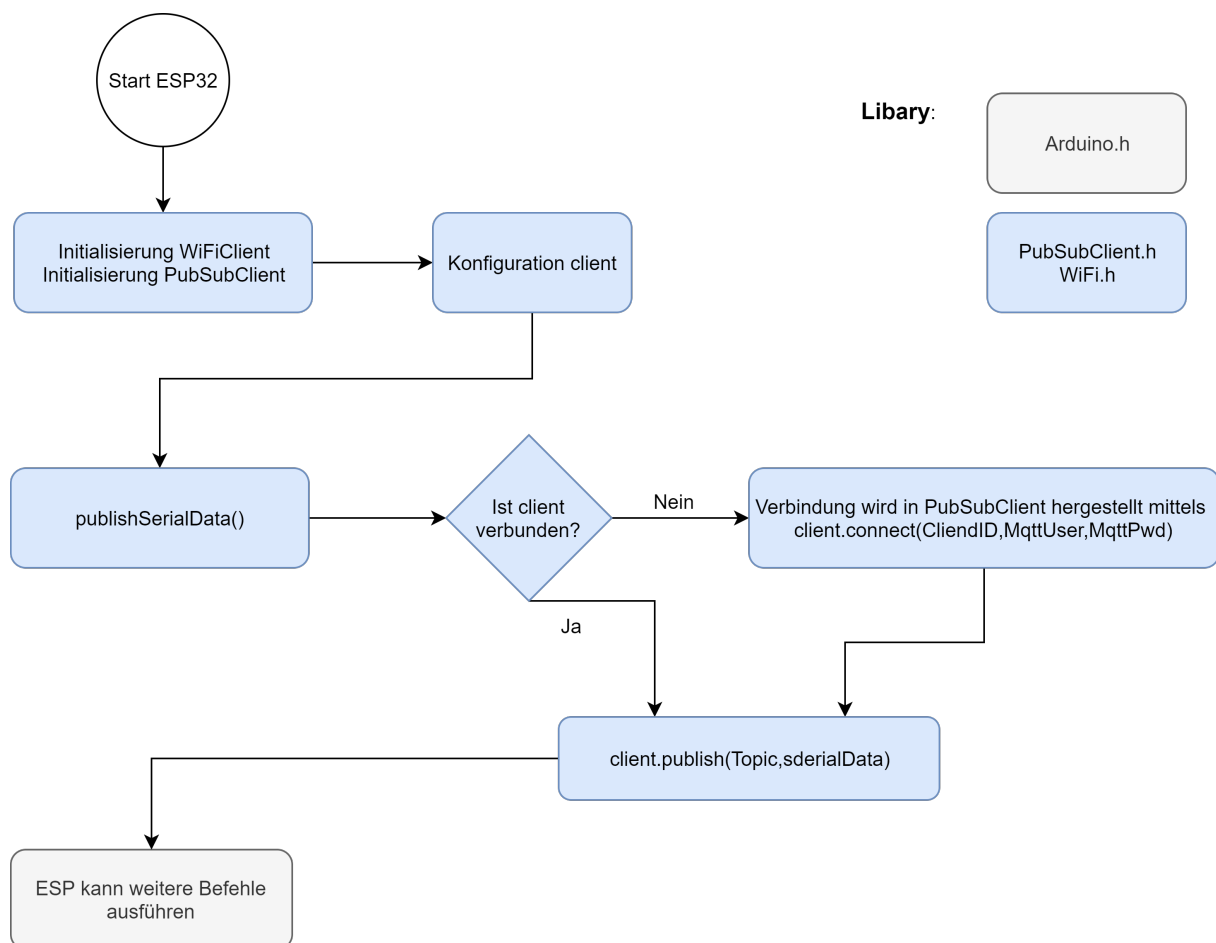


Abbildung 5.9: Esp Info

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

Danach wird der PubSubClient als Client instantiiert, 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 seriellen Daten 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 connect(), hergestellt. Um diesen Prozess erfolgreich durch zu führen werden, CliendID, MQTT-Benutzername und Passwort benötigt.

Ist die Verbindung in Ordnung werden die Daten mit dem Befehl publish() veröffentlicht [16].

5.5.2 Lizenzen

Die Bibliotheken welche in den Kapiteln 5.5.1 , 5.4 erwähnt wurden, sind freie Softwarepakete und können unter der Bedingung der GNU Lesser General Public Lizenz (LGPL) geändert werden. Die LGPL gibt die Freiheit das Programm, für jeden Zweck auszuführen, anzupassen und die Änderungen zu veröffentlichen. **noauthor_gnu.org_nodate**.

5.6 Server und Broker

Als Server wird ein Versuch aufgebaut, in dem ein Raspberry Pi der 4 Generation als inhouse Server verwendet wird. Auf dem Markt sind gibt es unzählige kostenlose Online-MQTT-Broker, bei diesen aber Zuverlässigkeit und Diskretion nicht den Projekt Bedingungen entspricht. Daher kann entweder eine teure Cloud-Server Lösung online gekauft oder ein eigener Broker eingerichtet werden. In diesem Projekt wird auf das Raspberry ein MQTT Broker Installiert, dazu wird der viel verbreitete Mosquitto Broker getestet. Der Mosquitto Broker ist Open Source von Eclipse und befindet sich bereits in den Repositoris der betreffenden Distribution. Möglich ist aber auch direkt den Server von openHAB (folglich: Kapitel 5.4.7) zu verwenden.

Inbetriebnahme Raspi

In diesem Artikel wird erklärt wie das Raspi in Betrieb genommen wird und ins gleiche Subnetz wie der Host verbunden wird. Nach dieser Inbetriebnahme können Befehle einfach mit Secure Shell (ssh) ausgeführt werden. Die Verbindung wird mit dem integrierten Funkchip welcher ab dem Modell Pi 3 vorhanden ist, hergestellt. Als erstes wird die Micro-SD-Karte des RasPis neu formatiert. Das Betriebssystem Raspian Booster Lite steht [17] zur Verfügung es ist ein Java SE Plattform Produkt und kann unter den Oracle Binary Code Licence Bedingungen verwendet werden. Um das Betriebssystem erfolgreich auf die Micro SD Karte zu flashen wird das Programm Etcher [18] verwendet. Um die Secure Shell nach dem Boot Vorgang zu benutzen wird ein leeres Textdokument im Boot Ordner erstellt mit dem Namen `ßsh`", ohne Endung ".txt". Danach können Netzwerkzugangsdaten in einem weiteren Textdokument "wpa_supplicant.conf" mit den lokalen Wlan Zugangsdaten erstellt werden, siehe nachfolgende Abbildung 5.10.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country="your_ISO-3166-1_two-letter_country_code"

network={
    ssid="LA7722"
    psk="490f2490"
}
```

Abbildung 5.10: Konfiguration File wpa supplicant

Im Lokalen Netzwerk kann nun kontrolliert werden ob sich ein Gerät also das RasPi verbunden hat und die IP-Adresse wird angezeigt. Mit dem Konsolenbefehl `ssh pi@192.168. . .` ist ein Login auf das RasPi möglich. Der Benutzername lautet pi und das Passwort raspberry. Die Netzwerkzugangsdaten können mit dem Befehl `'sudo nano /etc/wpa_supplicant/wpa_supplicant.conf'` geändert werden.

Mosquitto

Als erstes, vor einer neuen Installation, soll das System des Rasperrys auf den neusten Stand

gebracht werden, dies kann mit dem Befehl 'sudo apt-get update' geschehen. Mit dem Befehl 'sudo apt-get install mosquitto' wird der Server installiert und ist sofort einsatzbereit [19]. Beim installierten Mosquitto-Broker kann nun in einem Konfigurationsfile, Benutzer und Passwort generiert werden. Standardkonfigurationen sind durch die Installation bereits angelegt worden, wobei zusätzliche Einstellungen als Datei im Verzeichnis unter '/etc/mosquitto/conf.d/' abgelegt werden. Mit dem Befehl 'sudo mosquitto_passwd -c /etc/mosquitto/mqttpasswd systemreader' wird der Benutzer Systemreader erstellt, das Passwort muss anschliessend festgelegt werden. Es wurde noch ein weiterer Benutzer revpi01 erstellt, wobei dann im Befehl zur Benutzererstellung das '-c' weggelassen werden muss sonst wird der erste Benutzer überschrieben. Die erstellten Benutzer können unter dem Befehl 'sudo nano /etc/mosquitto/mqttpasswd' bearbeitet werden siehe Abbildung 5.11.

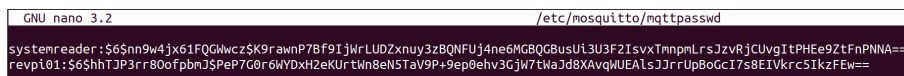


Abbildung 5.11: Konfiguration File Mosquitto

Um Benutzern eine Berechtigung auf bestimmte Topics zu geben, wird eine weitere Datei erstellt. Topics helfen Daten Stream auseinander zu halten. So können zum Beispiel Daten von einem Temperatursensor, der im Wohnzimmer misst, genau von den Daten des Sensors, welcher die Temperatur im Schlafzimmer misst, unterschieden werden. Die Datei wird mit dem Befehl '/etc/mosquitto/user.acl' angelegt. In Abbildung 5.12 ist dem systemreader die Berechtigung gegeben, dass er alles lesen darf und dem Benutzer revpi01, dass er nur in 'temperatursensor1/revpi01/' schreiben und lesen darf.

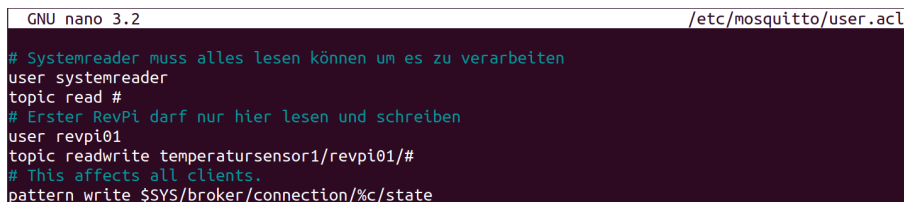


Abbildung 5.12: Konfiguration Topic Berechtigungen

Zu den beiden erstellten Dateien werden nun dem Mosquitto Konfigurationen hinzugefügt, mit dem Befehl 'sudo nano /etc/mosquitto/conf.d/secure.conf', Es öffnet sich ein File, bei welchem der Text wie in Abbildung 5.13 verfasst werden kann.



Abbildung 5.13: Konfigurationen File hinzufügen

Der Mosquitto-Server soll nach den Konfigurationen neu gestartet werden. Mit dem Befehl 'sudo systemctl restart mosquitto' wird ein reboot durchgeführt und anschliessend kann mit dem Befehl 'sudo systemctl status mosquitto' den Status des Mosquitto-Servers geprüft werden. Läuft das System korrekt, ist ein grüner Punkt ersichtlich, folglich Abbildung 5.14.

```
● mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2019-12-29 13:06:20 GMT; 18s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Main PID: 2319 (mosquitto)
    Tasks: 1 (limit: 2200)
   Memory: 628.0K
    CGroup: /system.slice/mosquitto.service
            └─2319 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Dec 29 13:06:20 raspberrypi systemd[1]: Starting Mosquitto MQTT v3.1/v3.1.1 Broker...
Dec 29 13:06:20 raspberrypi mosquitto[2319]: Loading config file /etc/mosquitto/conf.d/secure.conf
Dec 29 13:06:20 raspberrypi systemd[1]: Started Mosquitto MQTT v3.1/v3.1.1 Broker.
```

Abbildung 5.14: Mosquitto Status

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.

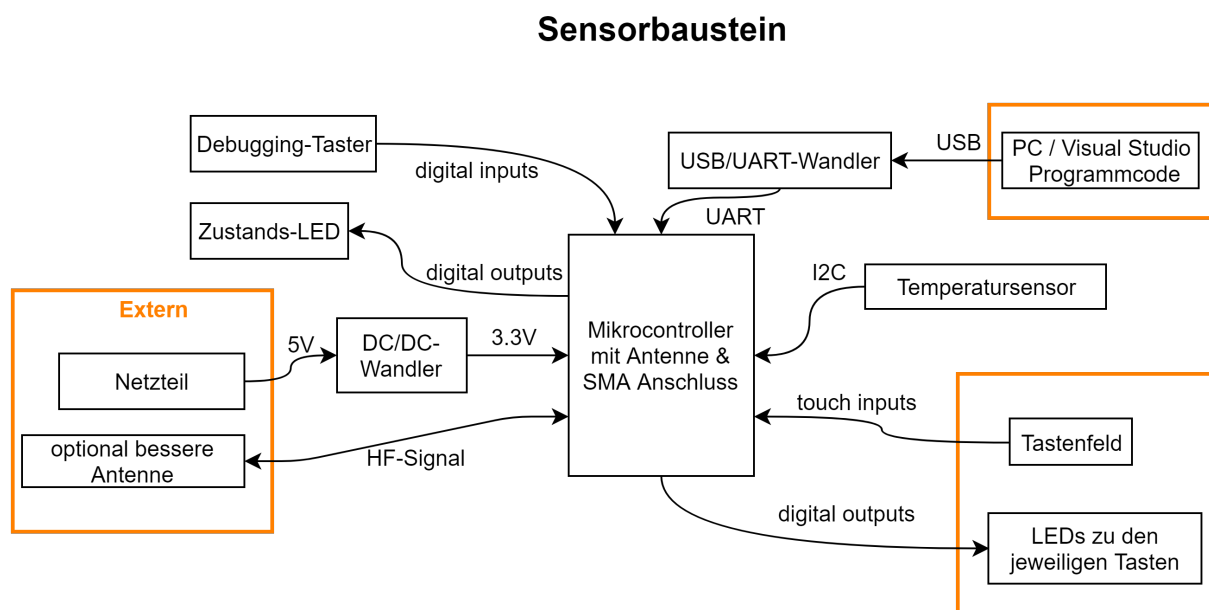


Abbildung 6.1: Übersicht Sensorbaustein

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, welche ein Signal auf die Pins geben und dann vermutlich mithilfe eines Spannungsteilers messen ob, dass Signal grösser oder kleiner wird. 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.

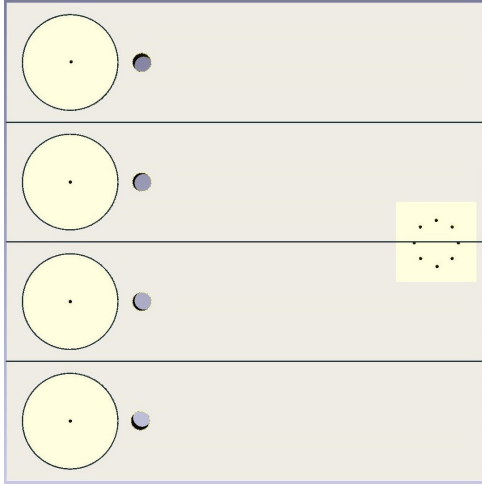


Abbildung 6.2: Frontplatte Vorderansicht

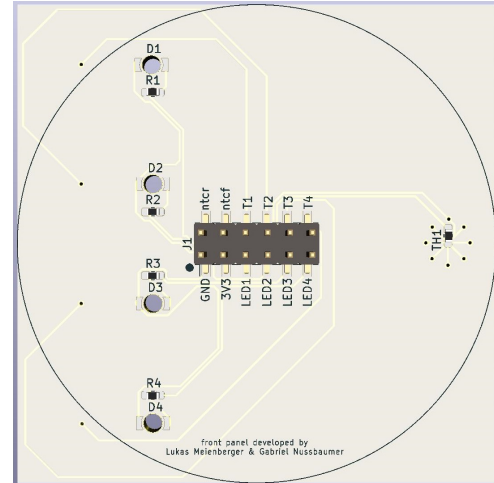


Abbildung 6.3: Frontplatte Rückansicht

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. 6.5) 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.

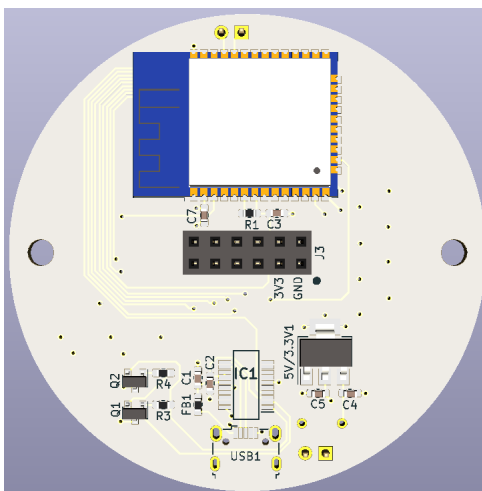


Abbildung 6.4: Hauptplatine Vorderansicht

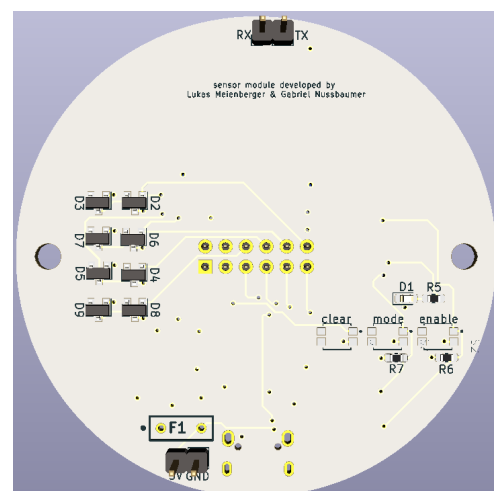


Abbildung 6.5: Hauptplatine Rückansicht

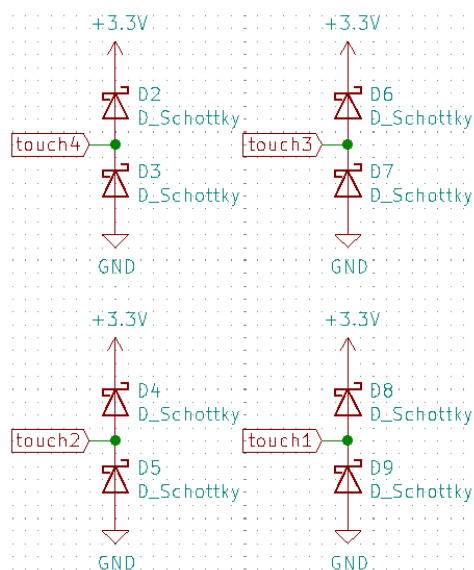


Abbildung 6.6: Shottky Dioden für ESD Schutz

6.1.3 Schutzbeschaltung

6.1.4 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 [5]. Aus folgender Übersicht sind die zur Verfügung gestellten Ein-/Ausgänge des ESP32 zu entnehmen:

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

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. 6.7), welcher im Sensorbausteins verwendet wird, aufgeführt:

6.1.5 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

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

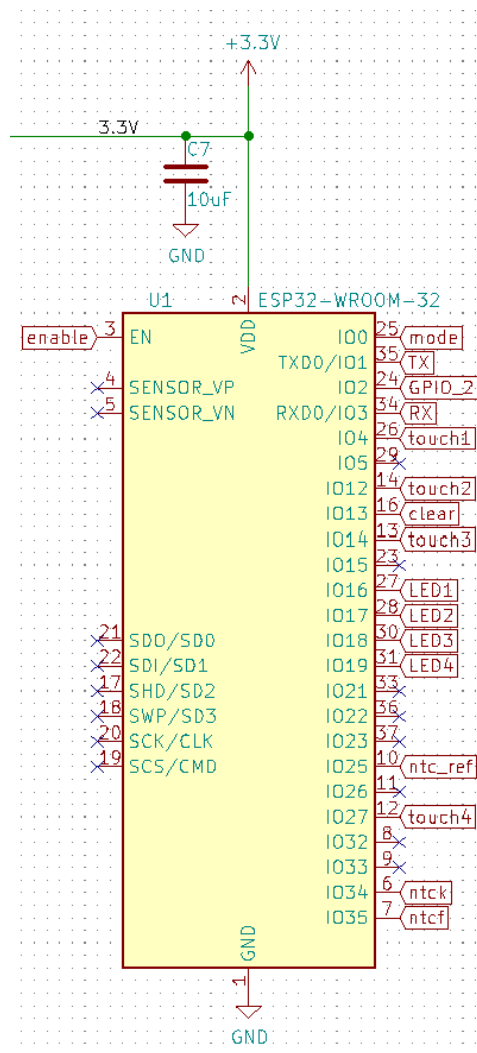


Abbildung 6.7: Mikrocontroller ESP32 im Sensorbaustein

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. 6.8) ist die verwendete Temperaturschaltung abgebildet. 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 ver-

meiden. Die Temperatur wird berechnet indem als erstes der Widerstand des NTCs R_T mit $R_T = \frac{U_{ntc}}{U_{ref} - U_{ntc}}$ berechnet wird. Für U_{ref} wird maximal eine Spannung von 2.5 V verwendet, da der ADC danach nicht mehr linear ansteigt. Dann muss nur noch der ADC Wert mittels einer Geraden in eine Spannung beachtet werden. Die Temperatur in Kelvin ergibt sich dann aus $T = \frac{1}{\frac{1}{T_N} + \frac{1}{B} \cdot \ln(R_T)}$.

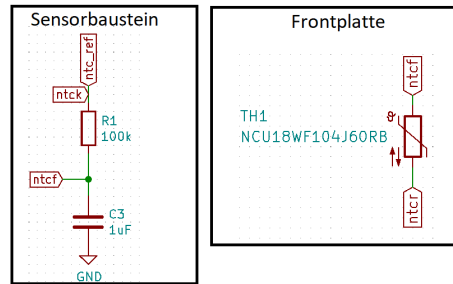


Abbildung 6.8: Schaltung Temperatursensor

6.1.6 Programmieranschluss

Der Mikrocontroller kann mittels eines USB to UART Wandlers programmiert werden. Hierzu wird der FT231XS-U verwendet. In Abbildung 6.9 ist das Schema abgebildet. Auf dem Schema sind dabei zwei Bipolartransistoren zu erkennen, welche es erlauben den ESP32 ohne drücken des Mode- und Enbletasters in den Programmiermodus zu versetzen. Falls es zu Komplikationen kommt steht aber auch die Option mittels des Enable- und Modetaster (Abb. 6.10) sich in den Programmiermodus zu versetzen offen. Dazu muss der Modetaster (auch während des ganzen Flashprozesses) gedrückt bleiben, dann mit dem Enbletaster 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 Button betätigen. Die LED5 zeigt mittels eines kurzen Blinkens das Aufstarten des ESP32 an oder oder dass der ESP32 im Programmiermodus ist.

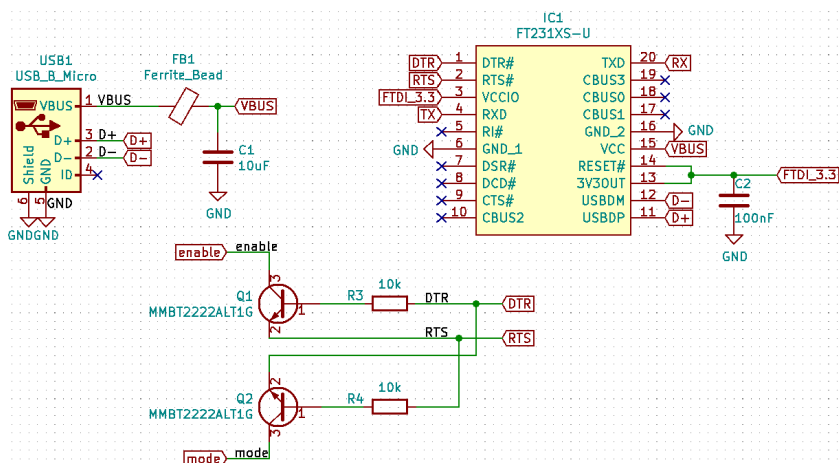


Abbildung 6.9: USB Anschluss

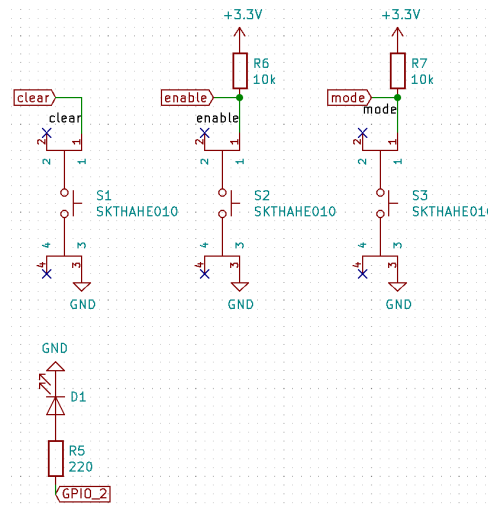


Abbildung 6.10: Buttons und LED auf Sensorbaustein

6.1.7 Spannungsversorgung

Der minimale Strom, welche die Spannungsquelle zur Verfügung stellen muss sind 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 5 V 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 und ist, mit momentan 0.42\$ bei Digi-Key, günstig. Im Schema (Abb. 6.11) 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.

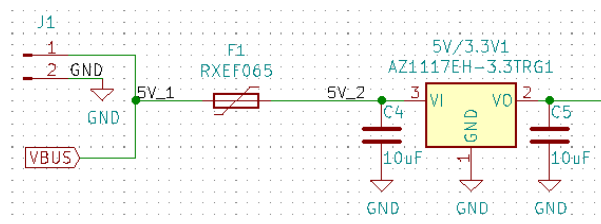


Abbildung 6.11: Spannungswandler Sensorbaustein

6.1.8 Platzstudie

Nun wird mittels einer Platzstudie überprüft ob der Sensorbaustein, mit den verwendeten Komponenten, wie eine Unterputzsteckdose eingebaut werden kann. Im Bild 6.12 ist eine Montageplatte für UP-Steckdosen abgebildet. Der blaue Kreis Zeigt dabei den Radius von 60 mm des Tasters bzw. Kleinkombination. Diese Masse wurden für die Dimensionen der Platine in Abbildung 6.13 übernommen.

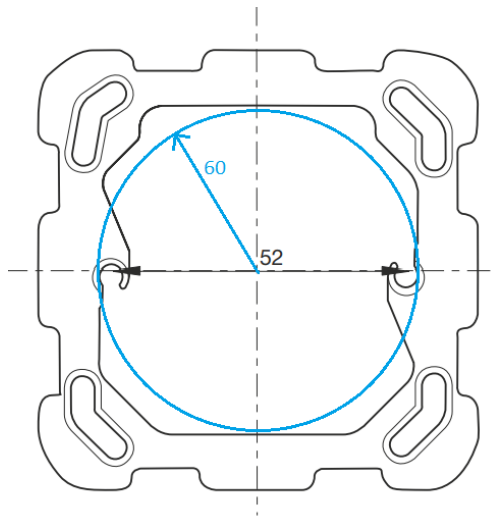


Abbildung 6.12: Montageplatte [20]

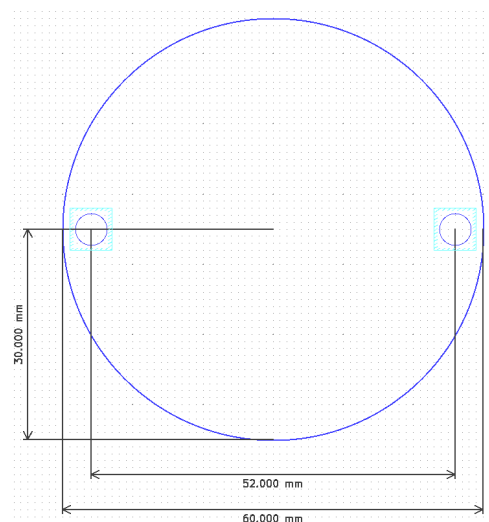


Abbildung 6.13: PCB Sensorbaustein Dimensionen

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.

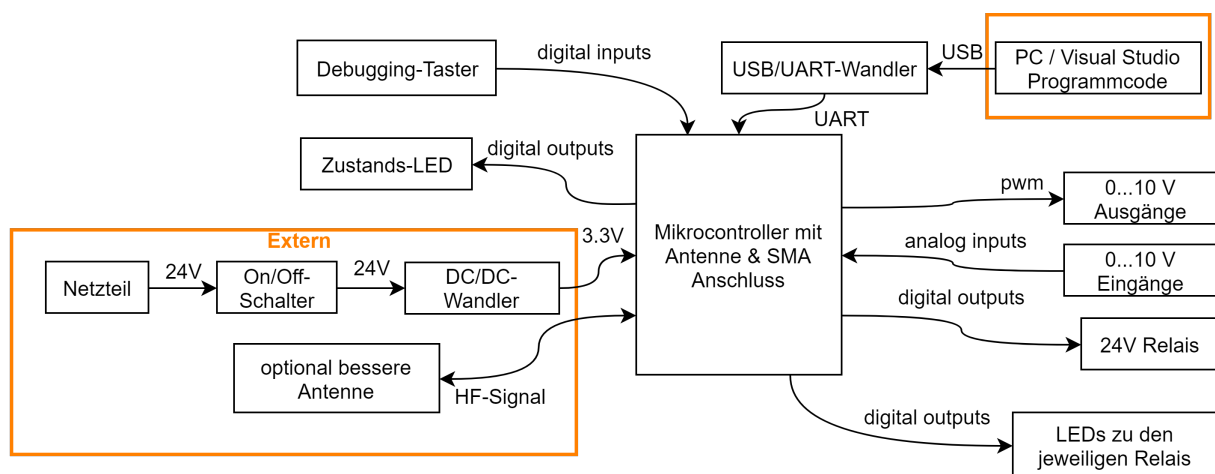


Abbildung 6.14: Übersicht Aktorbaustein

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 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 6.15 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

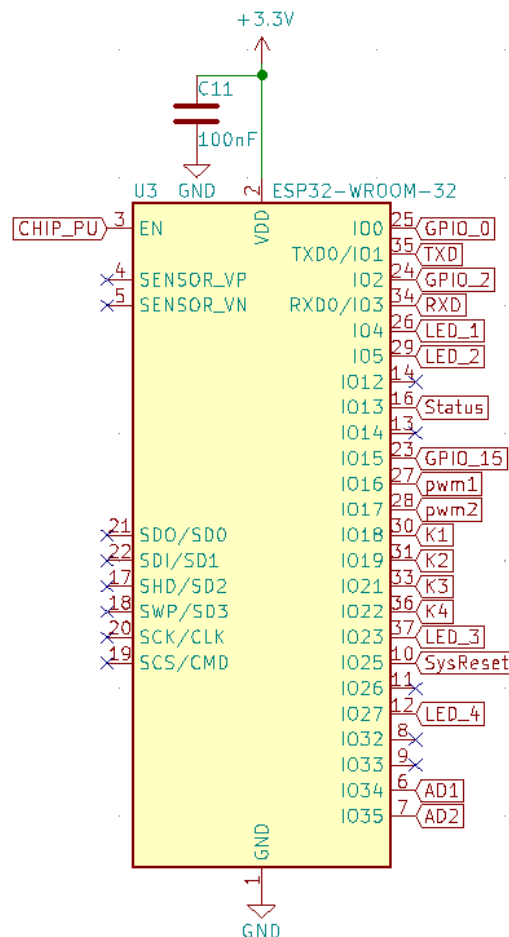


Abbildung 6.15: Mikrocontroller ESP32 im Aktorbaustein

6.2.2 Relais

Die Anforderungen an ein Relais sind, dass es 230 V/AC schalten und die Relaisspule mit 24 V/DC betrieben werden muss. In der Abbildung 6.16 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 **mikrocontroller.net_relais_nodate**. 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.

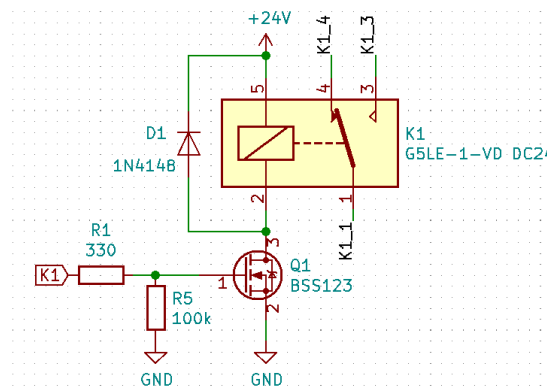


Abbildung 6.16: Relais im Aktorbaustein

6.2.3 Eingänge/Ausgänge 0..10 V

Die 10 V Eingänge werden mittels eines Varistors vor Überspannung geschützt (Abb. 6.17). Das Signal geht dann durch einen Puffer, damit die Signalquelle nicht belastet wird. Als Verstärker wird hierfür ein LM324A eingesetzt, da er sich gut als rail to rail OpAmp eignet. Als letztes wird mittels einer Spannungsteilers die obere Signalstärke von 10 V auf $10 \text{ V} \cdot 22 / (47 + 22) = 3.188 \text{ V}$ beschränkt. Hier muss dann in der Praxis kalibriert werden, da die Widerstände Toleranz behaftet sind. Bei den 10 V Ausgängen wird jeweils ein PWM vom Mikrocontroller herausgegeben und dann das Signal mittels eines Tiefpasses geglättet (Abb. 6.18). 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 kHz PWM \rightarrow Zeitkonstante $\tau = 1000 / 40 \text{ kHz} = 25 \text{ ms} \rightarrow$ Zeit bis zum Endwert $t_{\text{end}} = \tau \cdot \ln(1024) = 173 \text{ ms} \rightarrow$ 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 Trick 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. An dieser Stelle muss auch wieder kalibriert werden, so dass die maximale Ausgangsspannung 10 V beträgt.

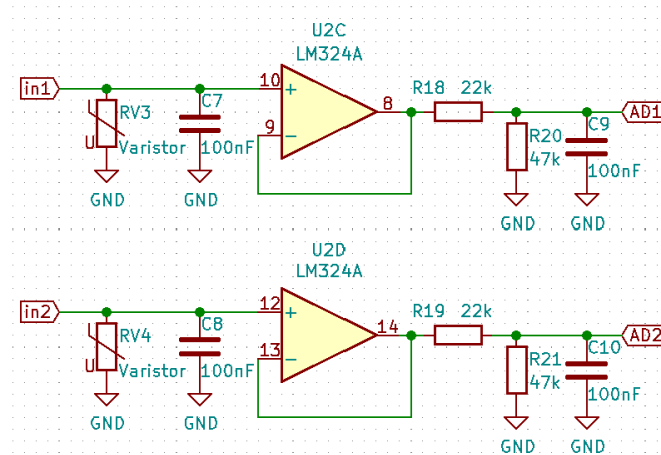


Abbildung 6.17: Eingänge 10 V Aktorbaustein

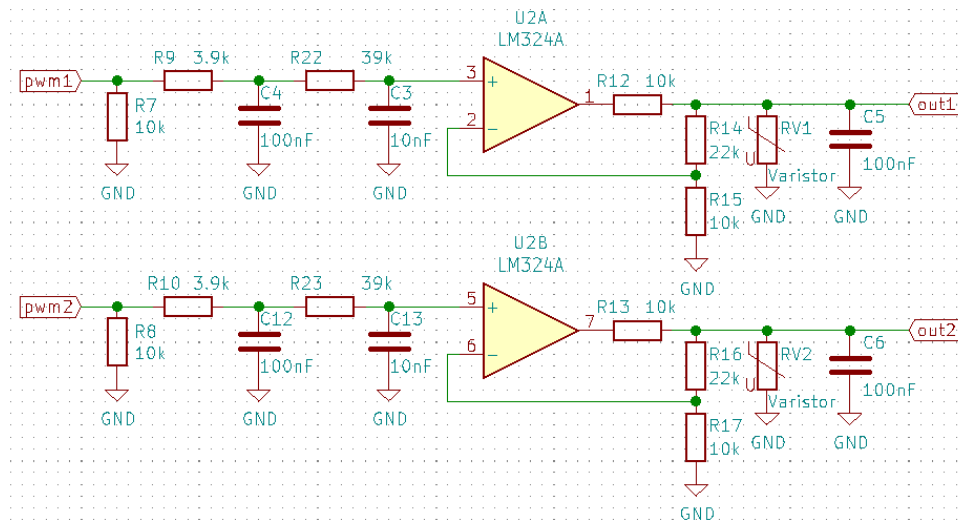


Abbildung 6.18: Ausgänge 10 V Aktorbaustein

6.2.4 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 6.19 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.

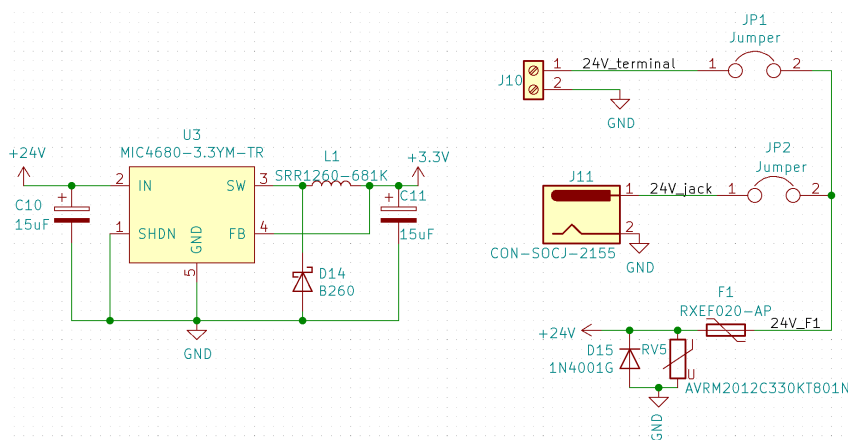


Abbildung 6.19: Spannungsversorgung Aktorbaustein

6.2.5 Interface

Die Ausgänge der Relais sowie die 10 V Ein- und Ausgänge sind auf Schraubklemmen geführt (Abb. 6.20). Es stehen insgesamt 6 LEDs zur Verfügung (Abb. 6.21). 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 6.1.6 Programmieranschluss.

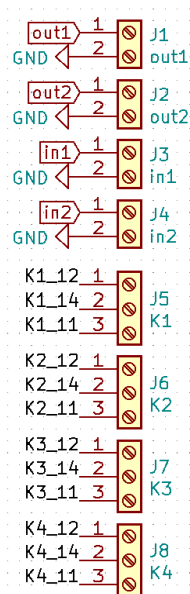


Abbildung 6.20: Klemmenanschlüsse Aktorbaustein

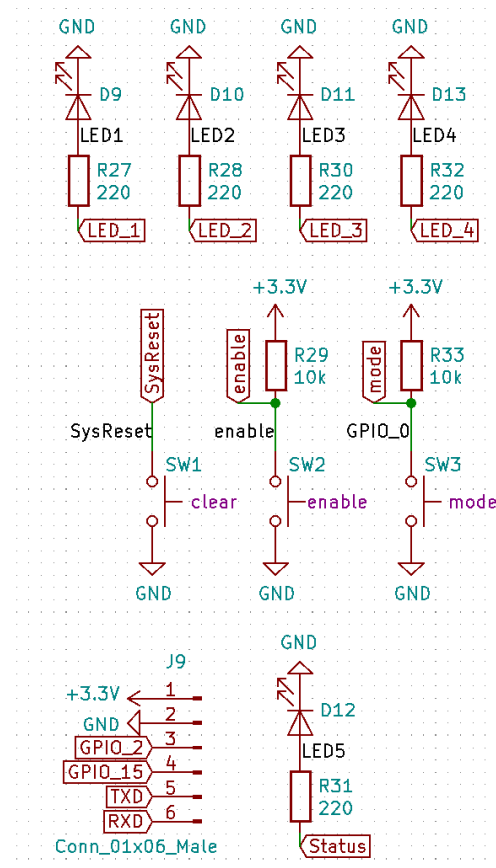


Abbildung 6.21: Buttons und LEDs Aktorbaustein

7 Schluss

7.1 Reflektion

In diesem Kapitel werden die Teilziele vom Pflichtenheft Reflektiert. In der untenstehenden Tabelle 7.1 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 7.1: 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 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 der Terminplan vom Projekt zeitlich nicht eingehalten wurde, so mussten durch Lösen von verschiedenen Problemen konnte in der Projektwoche nicht alle vorgesehenen Arbeiten erledigt werden und so kamen alle nachstehenden Arbeiten in Verzug.

7.2 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 draufgegangen war, wurde vorgenommen vermehrt die Hilfe vom Betreuer beanspruchen, somit sollen mehr Sitzungen stattfinden als im Projekt 5.

8 Ehrlichkeitserklärung

Hiermit erklären wir, dass die vorliegende Arbeit selbständig 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] (). Event Bus, Adresse: <https://www.openhab.org/docs/developer/Utils/events.html> (besucht am 13. Juli 2020).
- [4] 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).
- [5] Espressif, *Esp32-Wroom-32 datasheet_en*, Sep. 2019.
- [6] (). Visual Studio Code - Code Editing. Redefined, Adresse: <https://code.visualstudio.com/> (besucht am 8. Jan. 2020).
- [7] PlatformIO. (). PlatformIO is a new generation ecosystem for embedded development, Adresse: <https://platformio.org> (besucht am 8. Jan. 2020).
- [8] (). Arduino Client for MQTT, Adresse: <https://pubsubclient.knolleary.net/api.html#publish1> (besucht am 18. Jan. 2020).
- [9] (). Introducing Zephyr — Zephyr Project Documentation, Adresse: https://docs.zephyrproject.org/1.12.0/introduction/introducing_zephyr.html (besucht am 19. Jan. 2020).
- [10] (). Mbed Enabled | Mbed, Adresse: <https://www.mbed.com/en/about-mbed/mbed-enabled/introduction/> (besucht am 19. Jan. 2020).
- [11] (). Get Started — ESP-IDF Programming Guide v4.1-Dev-1066-G93a8603c5 Documentation, Adresse: <https://docs.espressif.com/projects/esp-idf/en/latest/get-started/index.html> (besucht am 1. Dez. 2019).
- [12] S. Media, director, *MQTT Binding 2.4 OPENHAB 2 Deutsch / MQTT Broker in OPENHAB 2 Anlegen / OPENHAB 2 Tutorial Deutsch*. Adresse: <https://www.youtube.com/watch?v=-8Wwi6MY0e8> (besucht am 19. Jan. 2020).
- [13] iobroker. (). Einsteiger – Willkommen! – ioBroker, Adresse: [index-5.htm?page_id=6317&lang=de](https://www.iobroker.net/index-5.htm?page_id=6317&lang=de) (besucht am 20. Jan. 2020).
- [14] M. Korte. (11. März 2018). Script über View ausführen, Adresse: <https://www.smarthome-tricks.de/software-iobroker/script-ueber-view-ausfuehren/> (besucht am 20. Jan. 2020).
- [15] zhouhan0126, *Zhouhan0126/WIFIMANAGER-ESP32*, 15. Dez. 2019. Adresse: <https://github.com/zhouhan0126/WIFIMANAGER-ESP32> (besucht am 28. Dez. 2019).
- [16] (). Arduino - AboutUs, Adresse: <https://www.arduino.cc/en/Main/AboutUs> (besucht am 1. Dez. 2019).
- [17] (). Download Raspbian for Raspberry Pi, Adresse: <https://www.raspberrypi.org/downloads/raspbian/> (besucht am 22. Dez. 2019).
- [18] (). balenaEtcher - Home, Adresse: <https://www.balena.io> (besucht am 23. Dez. 2019).
- [19] (). Eigener Cloud-Server mit MQTT – RevPiModIO, Adresse: <https://revpimodio.org/iot-mqtt/> (besucht am 29. Dez. 2019).
- [20] Hager, *Hager Kat1_08 Technik D*, 2019.
- [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).