HTBLuVA Rankweil

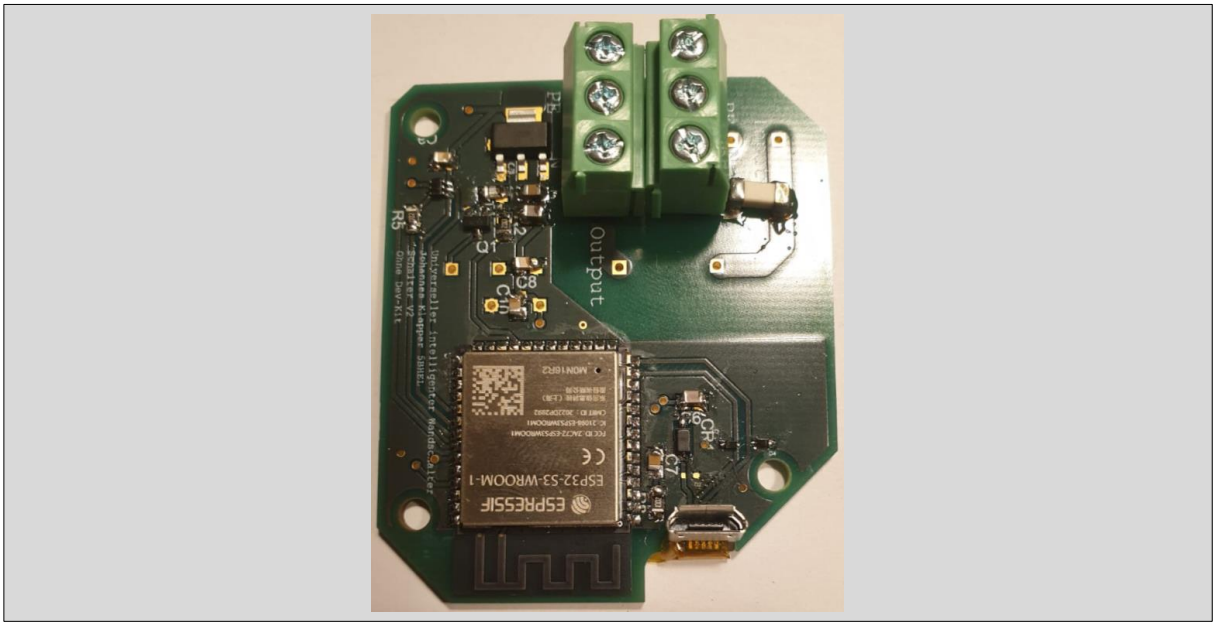
Höhere Lehranstalt für Elektronik

und Technische Informatik

DIPLOMARBEIT

Gesamtprojekt

Universeller Intelligenter Wandschalter



Ausgeführt im Schuljahr 2022|2023 von: Betreuer/in:

Johannes Klapper	5BHEL	Dipl. Ing. Gerald Bischof
Tim Kicker	5BHEL	Dipl. Ing. Gerald Bischof

Rankweil, am 21.03.2023

Abgabevermerk:

DA original, am 30.03.2023 Dipl. Ing. Gerald Bischof

DA digital, am 30.03.2023 AV Dipl.-Ing. Leopold Moosbrugger

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Rankweil, am 05.04.2023

.....

Johannes Klapper

.....

Tim Kicker

Abkürzung und Hinweise

DA	Diplomarbeit
Client	Programm, welches auf dem Endgerät des Nutzers ausgeführt wird
Bridge	Bindungsglied und Kontroll-Element des Systems
Smart-Switch	Intelligenter Schalter anzusteuerndes Gerät
Last	Vom Smart-Switch geschaltetes Gerät
BLE	Bluetooth Low Energy
ESP	Mikrocontroller ESP32-S3
DevKit	Development Kit für Mikrocontroller ESP32-S3
ROM	Read Only Memory
GPIO	General Purpose Input/Output
SPI	Serial Port Interface
JTAG	Joint Test Action Group
UART	Universal Asynchronus Receiver Transmitter
VS-Code	Visual Studio Code

Inhaltsverzeichnis

1 DIPLOMARBEIT DOKUMENTATION

Namen der Verfasser	Johannes Klapper, Tim Kicker
Jahrgang | Schuljahr	5BHEL | 2022|2023
THEMA der Diplomarbeit	**Universeller intelligenter Wandschalter**
Kooperationspartner	b2 electronics

Aufgabenstellung

Es wird ein intelligenter Wandschalter entwickelt. Dessen Anwendungsbereiche inkludieren Licht- und Heizungssteuerung. Die Steuerung erfolgt über ein Endgerät in Form verschiedener individuell einstellbarer Modis oder über herkömmliches mechanisches Umschalten. Der Einbau des Smart-Switches soll sich von dem, eines herkömmlichen Kippschalters nicht unterscheiden.

Individuelle Themenstellung im Rahmen des Gesamtprojektes

Johannes Klapper	**Smart-Switch & Kommunikation** Projektleitung, Erstellung der Smart-Switch Hardware, Firmware für ESP, BLE-Kommunikation
Tim Kicker	**Client, Bridge & Kommunikation** Erstellung der Client- und Bridge-Applikationen, Entwurf und Implementation der Protokolle, Allgemeine Kommunikation inklusive BLE
Realisierung	Die Realisierung erfolgt in Form eines funktionalen Aufbaus des Gesamtsystems mittels einer nicht größengerechten Versuchsplatine, sowie ein Prototyp in der geplanten Originalgröße. Ebenfalls wird die Firmware auf allen Komponenten implementiert.
Ergebnisse	Es wurde die komplette Soft- bzw. Firmware für Client, Bridge und Switch fertiggestellt. Die Hardware für den Smart-Switch wurde erfolgreich umgesetzt. Das Gesamtsystem wurde erfolgreich aufgebaut und ist funktional. Die finale Form des Smart-Switches konnte zwar aufgebaut, aber aufgrund einiger zeitlicher Verzögerungen nicht mehr in Betrieb genommen werden.
Einsichtnahmen**)	Archiv der HTL Rankweil, www.diplomarbeiten.berufsbildendeschulen.at

2 DIPLOMA THESIS DOCUMENTATION

Author(s)	Johannes Klapper, Tim Kicker
Form | Academic year	5.Klasse | 2023
Diploma Thesis Topic	**Universal Intelligent Switch**
Cooperation Partners	b2 electronics
Assignment of Tasks	
Johannes Klapper	**Switch & Communication** Project lead, implementation of switch hardware, firmware for ESP, BLE communication
Tim Kicker	**Client, Bridge & Communication** creation of Client- and Bridge application, design and implementation of communication protocols, general communication
Idea Scheme Design	The implementation takes place in form of a functional prototype of the whole system in form of a larger experimental board and in form of a prototype in size of the final, planned product.
Construction Materiality	The total soft- and firmware for Client, Bridge and Switch has been implemented. The hardware for the switch was successfully implemented as well. The whole system was built and is functional. The original size hardware was built but could not be taken into service.

3 ZUSAMMENFASSUNG

A Motivation

Ein Problem, welches heutzutage eine immer größer werdende Rolle spielt, ist zweifelslos die Energiekrise. Es gibt bereits viele Lösungsansätze im Smart Home Bereich, welche gute Arbeit leisten, allerdings benötigen die meisten solcher Systeme großräumige Installationen und sind deswegen auch recht teuer. Deshalb möchten wir die Idee eines „Universellen Intelligenten Wandschalters“ in die Tat umsetzen, mit welchem bei minimalem Installationsaufwand bereits viel Energie gespart werden kann. Ebenfalls ist die automatische Steuerung äußerst praktisch.

B Vorwissen

Es wurde Wissen aus allen fachspezifischen Unterrichtsgegenständen benötigt. Aus DIC kam die Erfahrung mit Schnittstellen, Mikrocontrollern und deren Programmierung. Aus FSST kam Vorwissen in C# und VS-Code zum Einsatz. Aus MTRS waren die nötigen Berechnungen bei der Hardware bekannt und Wissen aus KSN wurde bei den Kommunikationen über BLE und WLAN eingesetzt.

C Umsetzung

Hardwareseitig wurde ein herkömmlicher Wechselschalter so nachgestellt, dass er nicht nur mechanisch, sondern auch per Mikroprozessor geschaltet werden kann. Auch wurde noch ein Temperatursensor mit dem Mikroprozessor verbunden, um Richtwerte für die Heizungssteuerung zu erhalten. Abschließend wurde die Energieversorgung des Smart-Switchs über das Stromnetz sichergestellt, wobei Größe und Abwärme der Bauteile beachtet werden mussten.

Die Software beinhaltet die Kommunikation zwischen uC und Bridge, sowohl als auch die zwischen Client und Hub. Die einzelnen Teile benötigten jeweils eine eigene Firmware, diese wurden entwickelt und implementiert. Ebenfalls musste eine grafische Oberfläche zur benutzerfreundlichen Verwendung der Software erstellt werden.

4 ABSTRACT

A Motivation

The energy crisis without a doubt is a major problem nowadays. Resources are running low and energy preserving technologies are on the rise. A quite popular field with a lot of innovative ideas are Smart Homes. A downside of most of these systems is that for their implementation they require a high scale rewiring of the houses. Our System, the “Universal intelligent Switch” solves this issue. Even with its easy installation, it can already save a lot of energy and is practical as well.

B Previous Knowledge

To realize the project, we had to apply prior knowledge from all electronic specific subjects. DIC provided us the Know-how to deal with and program Interfaces and Microcontrollers. From FSST, we were already familiar with the development environments C# and VS-Code to some extent. MTRS provided us with the knowledge to do Hardware and Temperature specific calculations and last but not least through KSN we had previously gained a basic understanding of wireless communication, especially Wi-Fi.

C Implementation

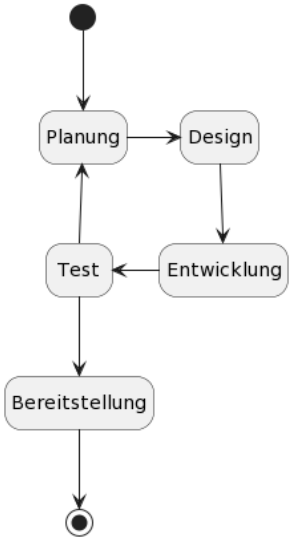
From a Hardware viewpoint, the task was to recreate a commonly known light switch, with the added capability of being controlled by the Microprocessor. The temperature values were also read to give a reference for the heating regulation. Lastly, the power supply had to be provided via the grid. Temperature as well as size restrictions had to be taken into account.

On the Software side, the communications between uC and Bridge as well as between Bridge and Client have to be set up. Furthermore, the Firmware for all of these components needs to be created and implemented.

5 PLANUNGSGRUNDLAGEN

5.1 Planung im Voraus

5.1.1 Extreme Programming



Extreme Programming ist eine Methode die in jeder Produktentwicklung Anwendung findet. Die links stehenden Schritte werden dabei so lange ausgeführt, bis ein funktionierendes Endprodukt zustande kommt. Diese Methode wurde bei Hard- wie Software angewandt. Bei der Hardware musste auf Design und Entwicklung etwas mehr Fokus gelegt werden, da die Testung erst nach dem Bestellen und Bestücken einer Platine möglich ist. In der Software waren hingegen Planung und Testung im Vordergrund.

5.1.2 Meilensteintrendanalyse

Bei der Meilensteintrendanalyse werden im Voraus fixe Zwischenziele mit Datum festgelegt die erreicht werden sollen. Meilensteine bestehen wiederum aus mehreren Arbeitsschritten, welche alle erledigt werden müssen. Dies hilft das Projekt zu unterteilen, die Übersicht zu fördern und bietet eine zeitliche Reverenz, an der man sich orientieren kann.

Software:
1. Kommunikation zwischen App u. Mikroprozessor funktioniert (18.11.2022)
2. Firmware fertiggestellt (18.1.2023)
3. Client-Applikation (V2) für Laptop fertiggestellt (28.2.2023)
4. Gesamtsystem fertiggestellt (31.3.2023)

Hardware:
1. Energieversorgung für Mikroprozessor fertiggestellt (2.11.2022)
2. Smart-Switch physisch wie über Mikrocontroller Schaltbar (23.12.2022)
3. Gehäuse und Wärmetechnik fertiggestellt (28.2.2023)
4. Gesamtsystem fertiggestellt (31.3.2023)

Vorgang	Anfang	Ende
Diplomarbeit anmelden	26.09.22	30.09.22
uC recherche Bestellung	26.09.22	04.10.22
Kommunikation zwischen Bridge und Software	05.10.22	17.11.22
Kommunikation Bridge/Client Funktioniert	18.11.22	18.11.22
Kommunikation Bridge und uC	18.11.22	17.01.23
Kommunikation Bridge/uC Funktioniert	18.01.23	18.01.23
Erstellung Desktop-Applikation	18.01.23	27.02.23
Lichtschalter auftreiben	26.09.22	27.09.22
Netzteil für uC fertigstellen	05.10.22	24.10.22
uC auf funktionalität getestet	25.10.22	02.11.22
Wechselschalter durch uC steuern lassen	03.11.22	02.12.22
Schalter auf phsisches Schalten und uC reagiere...	05.12.22	22.12.22
Temperatursensor an uC anbringen	23.12.22	12.01.23
Funktionsfähigkeit des Hardware-Gesamtsystem...	13.01.23	23.01.23
Wärmeprobleme des Gesmatsystems beheben	24.01.23	06.02.23
Gehäuse erstellen	07.02.23	01.03.23
Gesamtsystem funktionsfähig machen	02.03.23	30.03.23

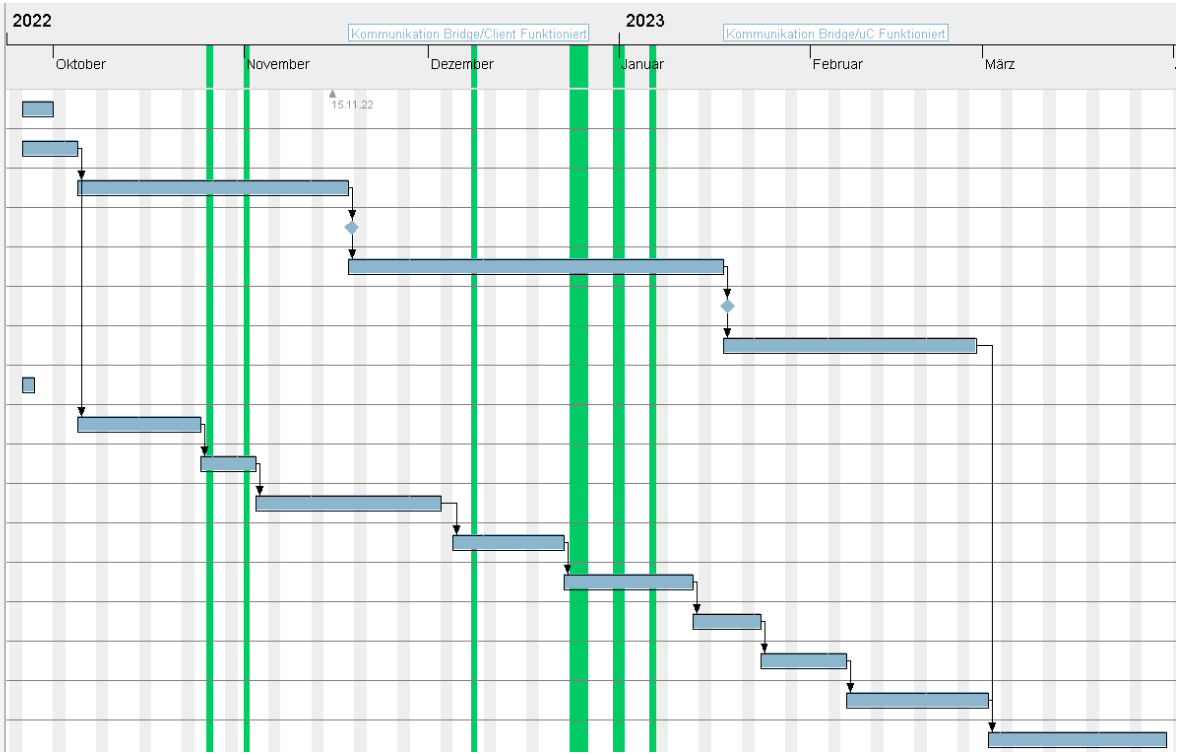


Abbildung 1: Projektplan Gantt Chart

5.2 Erkenntnisse und Abweichungen

Eine Erkenntnis, die sich bereits früh einstellte, ist es, dass manche Arbeitsschritte sich aufgrund unvorhergesehener Fehler oder Probleme extrem verlängern. Im Kontrast dazu kann bei einem fehlerfreien Ablauf wiederum viel Zeit eingespart werden. Die vorausgehende Planung hat also eine sehr hohe Relevanz, um schnell voranzukommen. Zur Planung wurde weitgehendst die Meilensteintrendanalyse angewendet, ebenfalls wurde bei der Umsetzung die Extreme Programming Methode verwendet, auf diese hätte im Vorfeld allerdings mehr Fokus gelegt werden sollen. Speziell die Planungsphase kam nämlich etwas zu kurz.

5.2.1 Hardware

Der grundsätzliche Plan konnte umgesetzt werden, allerdings gibt es einigen Raum für Verbesserung in Hinsicht auf zukünftige Projekte.

Der wohl größte Fehler war die mangelnde Vorbereitung bei der Bauteilauswahl. Genauere Recherche im Vorfeld hätte zweifellos eine große Zeitersparnis eingebracht, da jedes nicht optimal gewählte Bauteil mit einem viel größeren Aufwand in die Schaltung integriert werden muss und bei der Programmierung erneut für Probleme sorgen kann. Ein Beispiel für ein solches Bauteil ist der Temperatursensor. Um dies zu vermeiden hätte Extreme Programming weitgehender eingesetzt werden müssen.

Bei der Fertigstellung der Schaltung sollte also bereits die genaue Programmierung für jedes Bauteil feststehen. Die Leistungsanforderungen der Bauteile müssen selbstverständlich ebenfalls im Voraus berechnet bzw. berücksichtigt werden.[1]

5.2.2 Software

Der geplante Ablauf konnte anfangs gut verfolgt werden. Jedoch verschob sich der Zeitplan durch den Framework-Wechsel nach hinten (Siehe Software->MAUI).

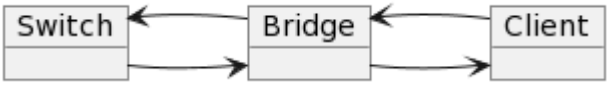
Die gelernte Erkenntnis liegt darin, immer nur bereits getestete UI-Frameworks zu verwenden.

[1] Vgl. Asana.

6 GRUNDLAGEN

6.1 Gesamtsystem

Das System ist folgendermaßen aufgebaut:



Der Client, beispielsweise ein Laptop, ist das Endgerät, das der Benutzer unseres Systems bedient. Die Bridge dient als zentrales Rechenzentrum, welches die Anfragen an die einzelnen Smart-Switches weiterleitet. Der Switch selbst ist für das Schalten der Last und das Auslesen der Temperatur zuständig.

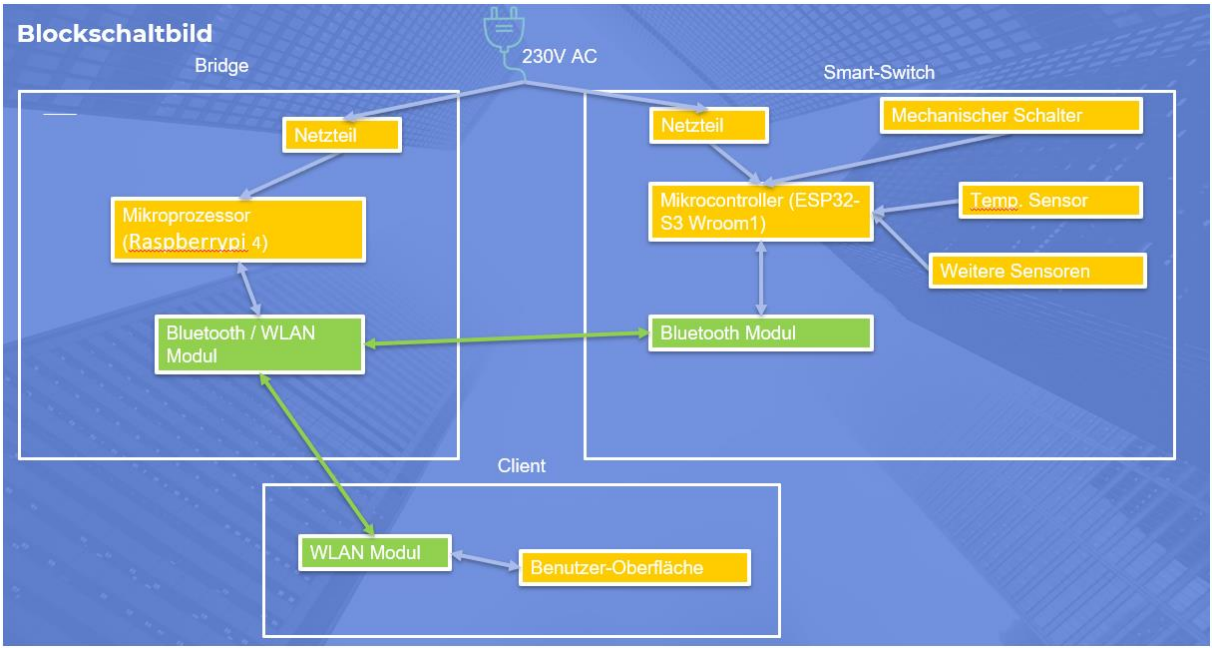


Abbildung 2: Gesamtsystem Blockschaltbild detailliert

Die Funksignale, Wlan und BLE sind im Blockschaltbild grün gekennzeichnet. Die einzelnen Teile unter Bridge und Client werden fertig gekauft und müssen nicht selbst gebaut werden. Die weiteren Sensoren sind eine Erweiterungsmöglichkeit für das Projekt, die während der Diplomarbeit noch nicht umgesetzt werden.

7 FINANZIELLER ASPEKT

7.1 Ausgaben

Um das Projekt realisieren zu können, mussten Bauteile und Leiterplatten gekauft werden. Der Großteil des benötigten Geldes wurde von der Partnerfirma finanziert, der Rest privat bezahlt. Zu guter Letzt hat die HTL Rankweil ebenfalls einige Bauteile gesponsort. Nicht vorrätige Bauteile wurden aufgrund der Lieferzeiten selbst bestellt und in Folge auch selbst bezahlt. Dabei wurde gelernt, dass der Preis der meisten Einzelbauteile sehr gering ist, der genaue Preis spielt deshalb bei kleinen Stückzahlen fast keine Rolle. Der Fokus sollte auf einzelne teure Bauteile gelegt werden (z.B. Relais). Ebenfalls ist der Preis von fast allen Bauteilen recht billig im Vergleich zu den Kosten eines Fehlers in einem Layout oder eines schlecht gewählten Bauteils. Dabei muss man nämlich mit preislichen als auch zeitlichen Bußen rechnen.

8 SMART-SWITCH

8.1 Allgemeines

Unter „Smart-Switch“ verstehen wir in unserem Projekt die Komponente, die in eine Elektroinstallation statt einem Lichtschalter verbaut werden kann. Der Smart-Switch selbst muss einige Aufgaben und Kriterien erfüllen. Er soll vor Ort mittels eines Tasters bedient werden oder über Bluetooth ferngesteuert werden können. Er muss die Temperatur aus einem Sensor auslesen und diese an die Bridge weiterleiten. Ebenfalls muss er als BLE-Server fungieren, damit er mit der Bridge kommunizieren kann. Um eine angeschlossene Last wie eine Beleuchtung ein- oder ausschalten zu können, sollte er hohe Ströme aushalten damit möglichst viele unterschiedliche Lasten gesteuert werden können. Das Schalten selbst geschieht dabei je nach Programmierung des Clients. Übergeordnet kann aber der Smart-Switch natürlich auch mechanisch, wie bei einem herkömmlichen Lichtschalter, geschalten werden. Zu guter Letzt muss der Smart-Switch in eine Buchse, in Normgröße der EU, Platz haben und möglichst einfach mit einem normalen Schalter austauschbar sein.[2]

8.2 Konzept

8.2.1 Vorgehensweise

Es wurden zwei unterschiedliche Smart-Switches gebaut. Die erste Version diente für Testzwecke, deshalb musste auf die Größe nicht geachtet werden. Die erste Schaltung wurde über die USB-Programmierschnittstelle versorgt, daher konnte auf ein Netzteil noch völlig verzichtet werden. Es wurden viele Messpunkte und Ausgabemöglichkeiten angebracht, um die Funktionalität möglichst einfach überprüfen zu können.

Die zweite Version des Smart-Switches ist dann schon die finale Form, es wurden nur noch minimale Testmöglichkeiten, in Form einer LED und 4 Test Pins, vorgesehen. Ebenfalls wurden einige Bauteile durch leistungstechnisch effizientere Alternativen ersetzt. Es war geplant, dass der grundsätzliche Aufbau gleich ist, und so sollte das Gesamtsystem nach der erfolgreichen Inbetriebnahme der 1. Version, problemlos mit der 2. Version austauschbar sein. Allerdings gab es hier noch Probleme durch fehlerhaft umgesetzte Bauteile. Mit dieser Vorgehensweise sollte ebenfalls verhindert werden, dass wichtige Spezifikationen übersehen werden, da das System ansonsten nicht funktioniert.

[2] Vgl. Elektrotechnische Normung in Österreich.

8.3 ESP32-S3-Wroom-1 und ESP32-S3-DevKitC-1-N8R2

8.3.1 Entwicklungsplatine

Für das Projekt wird die ESP32-S3-DevKitC-1-N8R2 Entwicklungsplatine verwendet. Der darauf verbaute uC ist der ESP32-S3 Wroom1. Dieser benötigt eine 3,3V Spannungsversorgung. Weiters verfügt die Platine über eine 5V Spannung, welche von einigen Peripherie Bauteilen wie dem Relais benötigt wird. Die 5V Spannung wird über die USB- und UART-Schnittstellen geliefert, welche ebenfalls für das Debugging und das Programmieren verwendet werden. Ein Vorteil der Entwicklungsplatine sind die vielen Kontrolloptionen z.B. eine rote Power-On LED, eine RGB LED und zwei Tasten. Auch sehr praktisch sind die mit Füßen versehenen GPIO-Pins, sie sind bei Testungen einfach zugänglich und übersichtlich angeordnet.

Abbildung 3: ESP DevKit

8.3.2 Mikrocontroller

Der Mikrocontroller selbst, hat weder das DevKit noch die Bluetooth Antenne angeschlossen. Wird später vom Mikrocontroller oder dem ESP gesprochen, ist also eigentlich die Version ESP32-S3-DevKitC-1-N16R2 gemeint.

8.3.2.1 Auswahlverfahren

	ESP32-S3	ESP32-C3	ESP32-8266	WiPy-3.0
Größe [mm]	18x25,5	16,6x13,2	5x5	15,3x20
I2C bzw. SPI	Beides	beides	beides	beides
BLE-Version	BLE (5.2)	BLE (5.0)	-	BLE (4.2)
Anzahl GPIOs	36	22	17	25
Antenne inkl.	Ja	Ja	-	Nein
Rom vorhanden	Ja	Ja	Ja	Nein

Der ESP8266 ist der einzige Baustein bei dem Vorwissen vorhanden ist. Da dieser aber kein Bluetooth unterstützt, war dieser für unser Projekt leider unbrauchbar. Der WiPy kommuniziert auf dem älteren BLE-Standard und darüber hinaus hätte die Antenne separat bestellt und angebracht werden müssen.[3] Da er ansonsten auch keine signifikanten Vorteile bringt wurde er ebenfalls nicht gewählt. ESP32-S3 bzw. C3 erfüllen alle nötigen Spezifikationen. Es wurde der S3 aufgrund des neueren BLE-Standards gewählt. Allerdings hätte der doch entscheidende Aspekt der Größe bei der Auswahl genauer untersucht werden können.[4] [5]

8.3.2.2 Benötigte Funktionen

Auf der Entwicklungsplatine sind einige dieser Funktionen (siehe unten) bereits implementiert (C). Ebenfalls gibt es interne Funktionen, abgesehen von der mehr als ausreichenden grundsätzlichen Rechenleistung, welche für das Projekt essenziell sind (I). Diese Funktionen sind bereits programmiert und es müssen keine Änderungen vorgenommen werden. Zu guter Letzt gibt es noch die Peripherie-Funktionen, diese benötigen zusätzliche GPIO-Pins.

Funktionen

- Kommunikation mit dem Temperatur Sensor über SPI (5 GPIOs)
- Einlesen des Signals des Tasters (1GPIO)
- Steuerung der Relais (1-2 GPIOs)
- Anschluss mindestens einer Programmier-Schnittstelle (2 GPIOs, C)
- Energieversorgung mit nötiger Beschaltung (C)
- Pulldown des Enable Pins (C)
- Ground (C)
- ROM (I)
- BLE (I)

Ein ROM (Flash) muss unbedingt vorhanden sein, da ansonsten die bestehende Programmierung verloren geht, sobald der ESP vom Programmiergerät abgeschlossen wird. Dadurch wäre der Smart-Switch komplett nutzlos, da er natürlich nicht programmiert werden darf, während er an der Netzspannung hängt. BLE ist für die Kommunikation mit der Bridge ebenfalls unabdingbar. Siehe Kapitel 11.3.3.1.1.

[3] Vgl. WiPy 3.0.
[4] Vgl. ESP8266 – Mikrocontroller.net.
[5] Vgl. Helmut (2021).

8.4 Smart-Switch Version 1

Für die erste Generation der Leiterplatte wurde ein Entwicklungsboard des ESP-Prozessors (ESP Devkit) eingesetzt, um in einem frühen Stadium des Projekts Erfahrung mit den Komponenten sammeln zu können.

Die folgende Erläuterung der Schaltung fällt stellenweise etwas kürzer aus, weil manche Komponenten auch in der Version 2 vorkommen und im entsprechenden Teil ausführlicher erklärt sind.

8.4.1 Schaltung

Alle in der Schaltung (siehe Kapitel 15.1.1) verbauten Kondensatoren dienen der Spannungsstabilisierung. Dazu wurde mit Ausnahme von C6 und C7 eine Kapazität von 100nF verwendet. Diese wiederum dienen als Schutz gegen größere Störungen bei den Eingangsspannungen und sind deshalb von 3,3V und 5V auf Ground geschaltet. Sie haben einen Wert von 100uF.

8.4.2 ESP-Devkit Beschaltung

Die Widerstände R1 bis R4 werden den LEDs jeweils vorgeschalten, um den Stromfluss weit genug zu reduzieren, sodass er im Anwendungsbereich der LEDs liegt. Bei LEDs 2 bis 4 hätte das theoretisch auch über die Software geregelt werden können, so kann aber auch bei falschen Softwareeinstellungen (zu hoher Duty-Cycle) nichts passieren.

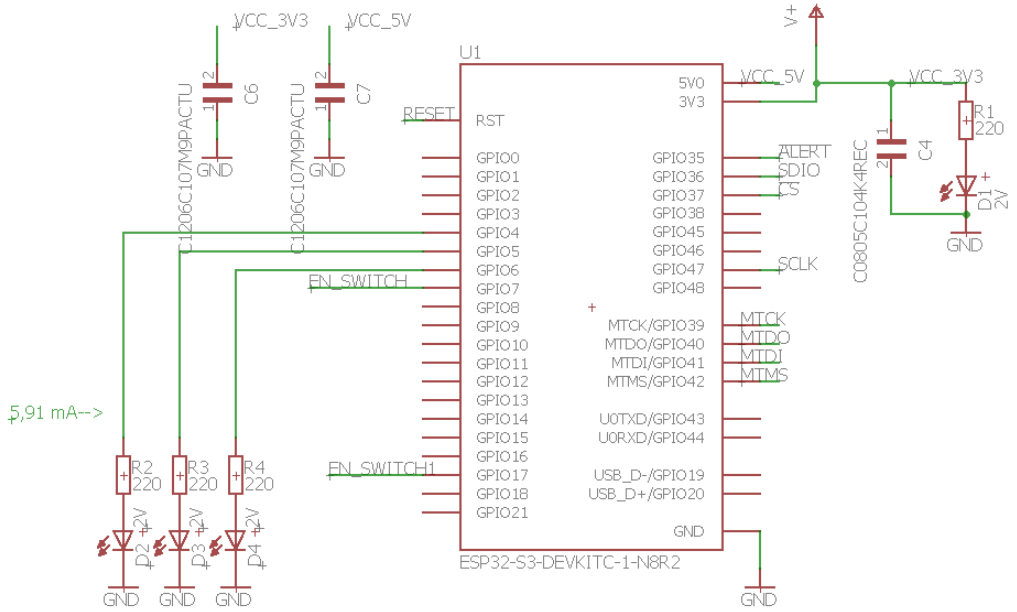


Abbildung 4: ESP DevKit Anschluss V1

6	42	I/O/T	MTMS, GPIO42	11	37	I/O/T	SPIDQS, GPIO37, FSPIQ, SUBSPIQ
7	41	I/O/T	MTDI, GPIO41, CLK_OUT1	12	36	I/O/T	SPIIO7, GPIO36, FSPICLK, SUBSPICLK
8	40	I/O/T	MTDO, GPIO40, CLK_OUT2	13	35	I/O/T	SPIIO6, GPIO35, FSPID, SUBSPID
9	39	I/O/T	MTCK, GPIO39, CLK_OUT3, SUBSPICS1	17	47	I/O/T	GPIO47, SPICLK_P, SUBSPICLK_P_DIFF

Abbildung 5: ESP SPI und JTAG-Schnittstellen

In Abbildung 5 sind die Anschlüsse aller Datenleitungen an das Devkit zu sehen. Die Switch Enables sowie die LEDs 2 bis 4 wurden dabei auf beliebige GPIOs gelegt, die keine spezielle benötigte bereits definierte Funktion haben. Die 4 MTxx Signale sind an der vordefinierten JTAG-Schnittstelle des DevKits angeschlossen, die GPIOs 39 bis 42. Das gleiche gilt für die 4 Signale der SPI, hier werden GPIOs 35 bis 37 als Datenleitungen verwendet und GPIO 47 als Clock.[6]

8.4.3 Temperatursensor

Für die Messung der Temperatur ist ein Sensor erforderlich.

Auswahlverfahren

Der Sensor sollte für die Anwendung möglichst genau sein. In der folgenden Tabelle ist eine Übersicht einiger Möglichkeiten aufgelistet.

	TMP126	TMP114	ADT 7411	MCP98224
Größe [mm]	1,85x1,80	0,76x0,76	4,9x6	2x3
Temperatur Bereich [°C]	20 bis 30 -20 bis 80	-10 bis 80	-40 bis 120	75 bis 95 -40 bis 120
Genauigkeit [°C]	+-0,25 +-0,3	+-0,3	+-0,5	+-0,2 +-1
Schnittstelle	SPI (3 Pin)	I2C	SPI und I2C	I2C
Hersteller	TI	TI	Analog Devices	Microchip Tech

Der MCP98224 ist für andere Anwendungen bei hoher Wärme konzipiert, deshalb sind seine Toleranzen erst bei höheren Temperaturen gut. Der ADT 7411 hat mehrere Schnittstellen was praktisch sein kann, falls zusätzliche Sensoren hinzugefügt werden, er ist jedoch nicht allzu genau. Die TMPs 114 und 126 sind sehr ähnlich, allerdings hat der Tmp126 einen etwas größeren Footprint was bei diesen kleinen Größen das Anlöten erleichtert. Ebenfalls ist seine Toleranz in unserem Messbereich noch eine Spur besser, deshalb wurde er gewählt.[7]

[6] Vgl. ESP32-S3-DevKitC-1 v1.1 - ESP32-S3 - — ESP-IDF Programming Guide latest documentation.
[7] Vgl. TMP126NDCKR Texas Instruments | Mouser.

8.4.3.1 Anschluss

In Abbildung 6 ist die Beschaltung des Temperatursensors zu sehen. Die Signale der Schnittstelle können an den Test Pins abgegriffen werden. So können bei Schwierigkeiten bei der Inbetriebnahme schnell die Fehler ausfindig gemacht werden.

Bei der Auswahl des Sensors wurde nicht berücksichtigt, dass es nur eine Datenleitung gibt, was die SPI-Übertragung etwas komplizierter als notwendig macht.

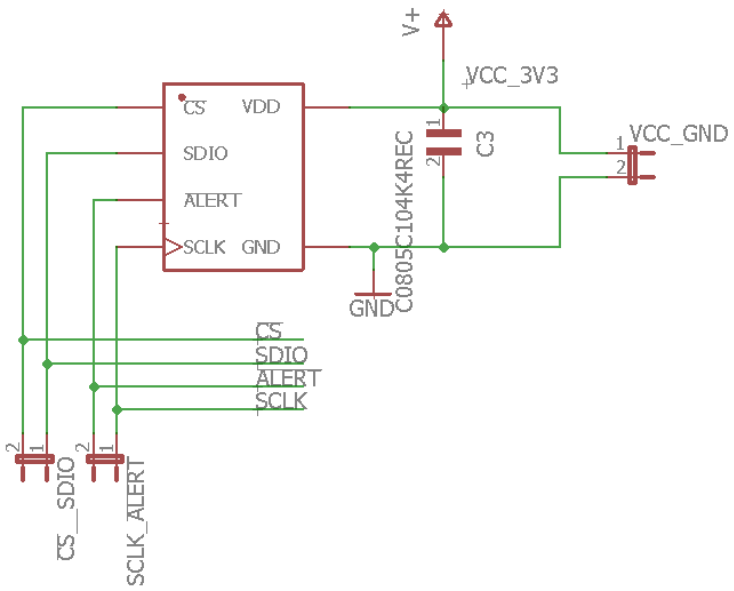


Abbildung 6: Temperatur Sensor Anschluss V1

Der Temperatur Sensor besitzt eine Sonderform einer SPI . Daten Ein- und Ausgang sind dabei auf denselben Pin geschaltet, es handelt sich also um eine 3 Pin SPI. Die standardmäßige, in den ESP-Bibliotheken vordefinierte, Programmierung für die SPI kann deshalb nicht angewendet werden. Stattdessen werden die Pins manuell angesteuert.[8]

[8] Vgl. SPI - Arduino Reference.

8.4.3.2 Programmierung

In der untenstehenden Grafik ist die Befehlsstruktur abgebildet, mit welcher der Temperatur Sensor angesteuert werden muss.

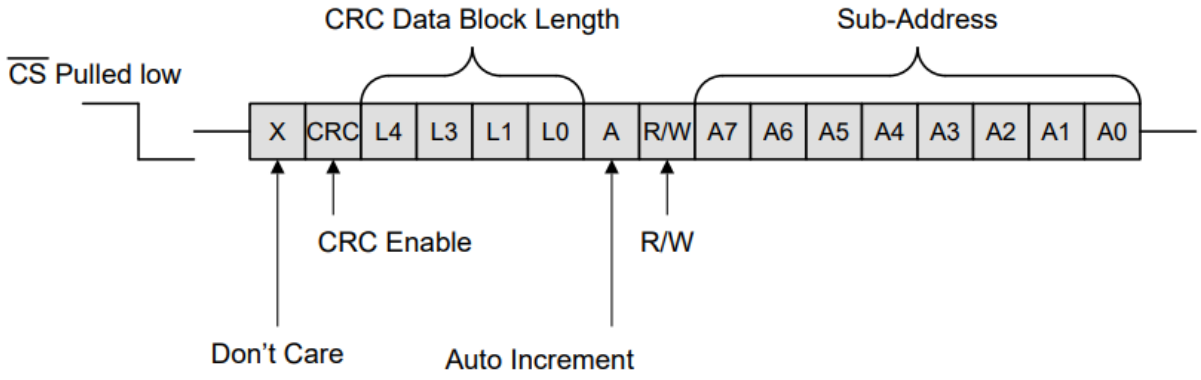


Abbildung 7: Befehlsstruktur Temperatur Sensor

Der CRC (Cyclic Redundancy Check) ist optional und wird mit dem CRC Enable Bit ein- oder ausgeschaltet. Da die Kommunikation über einen möglichst langen Zeitraum fehlerfrei ablaufen muss, macht es Sinn diesen Check durchzuführen. Die Länge des CRC-Checks ist minimal eingestellt. Mit dem Auto Increment Bit wird eingestellt, ob das folgende Kommando kontinuierlich oder einmalig ausgeführt werden soll. Das Read/Write Bit ist selbsterklärend und die Sub-Adressen können im Datenblatt des Tmp126 unter Register nachgeschaut werden.

Verwendete Register

Bei den Read Befehlen stehen in der Datenspalte, die Positionen der für uns Relevanten Informationen.

	Sub-Adresse	Read/Write (Bit)	Auto Increment (Bit)	Daten
Konfiguration	03h	Write (0)	Ein (1)	0086h
Temperatur Obergrenze	06h	Write (0)	Ein (1)	1E80h
Alert enable	04h	Write (0)	Ein (1)	0014h
Temperatur lesen	00h	Read (1)	Aus (0)	FFFCh
Alert lesen	02h	Read (1)	Ein (1)	0090h

Programmablauf

Der T-High Alert wird ausgegeben, sobald die gemessene Temperatur eine kritische Höhe erreicht (Siehe Kapitel 8.5.3.2). Wird ein T-High Alert ausgelesen muss der ESP in den Schlafmodus versetzt werden. Ansonsten könnte der Smart-Switch aufgrund einer Überhitzung kaputt gehen.

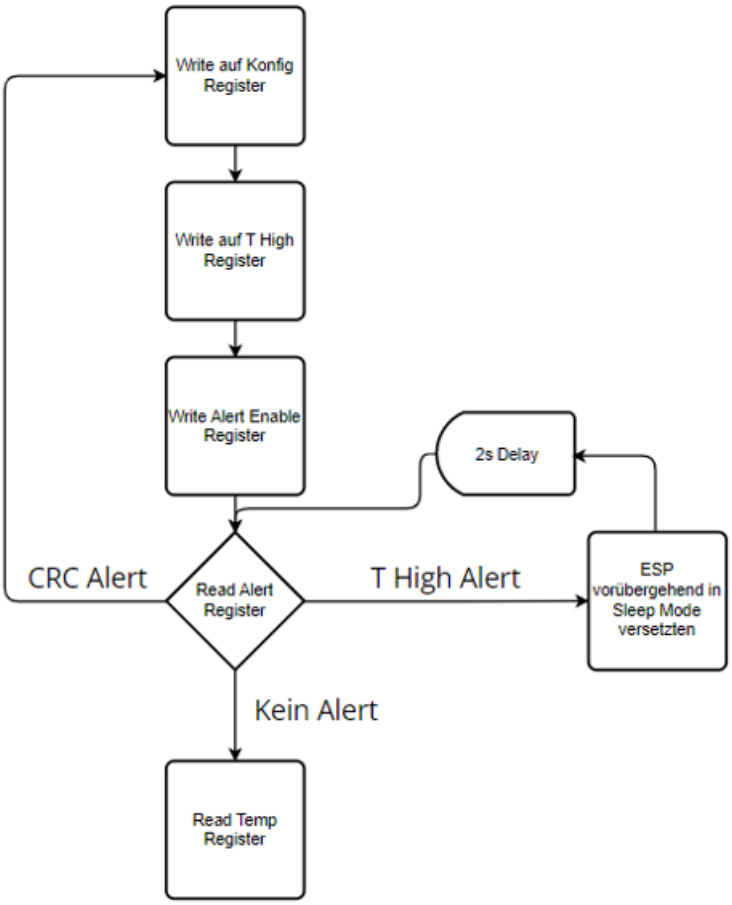


Abbildung 8: Flussdiagramm Temperatur Sensor

Dabei muss natürlich beachtet werden, dass dadurch die Last vorübergehend keinen Strom bekommt. Das T High muss also möglichst hoch angesetzt werden, damit eine solche Ausnahme wirklich nur im Notfall eintritt.

$T_{GMax} = 60°C$ … Maximale Ambient Temperatur

Beispielbefehl

```
#define Simple_Read 0b0100110100000000
```

Das ist der kontinuierliche Temperatur Lese Befehl, dabei wird hier ein 3-stelliges CRC-Bit mitversandt. Sollte es also zu Kommunikationsschwierigkeiten kommen, so wird beim Alert Pin sofort der CRC Alert geworfen, wird dieser ausgelesen wird eine neue Übertragung initialisiert.

8.4.3.3 Temperaturberechnungen

Aufgrund der möglichst einfachen Installation befindet sich der Temperatur Sensor ebenfalls auf der Leiterplatte. Da einige Teile sich doch recht stark erwärmen ist in davon auszugehen, dass die Umgebungstemperatur im Gehäuse um etwas höher ist als die Außentemperatur. Um nun als von der gemessen Temperatur auf die Umgebungstemperatur zu gelangen, müssen die thermischen Berechnungen berücksichtigt werden (Siehe Kapitel 8.5.3.2.1).

Um die Temperaturen unterscheiden zu können wird die Temperatur im Gehäuse als Ambient definiert und die außerhalb als Umgebung.

$\Delta T_{U,A} = 13{,}51\ K$ … maximale Temperaturänderung Umgebung zu Ambient

Die Temperatur im Gehäuse kann bei maximalem Stromfluss über den Netzspannungsbereich, bis zu 13,5 °C höher werden als die der Umgebung. Da es keine Messmöglichkeit für die Netzspannung gibt muss eine momentane Erwärmung im Gehäuse approximiert werden.

$\Delta T_{U,A\_Norm} = 4{,}54\ K$ … gewöhnliche Temperaturänderung Umgebung zu Ambient

Von der gemessenen Temperatur müssen also 4,54 °C abgezogen werden, um auf die Umgebungstemperatur zu gelangen.

Für ein finales Produkt wäre es zweckmäßig im Gehäuse gute Lüftungsöffnungen anzubringen, um die Erwärmung im Gehäuse gegenüber der Umgebung möglichst gering zu halten.

Weiterführend erfolgt die Ausgabe der Temperatur nicht in °C sondern das LSB (14tes Bit) entspricht 0,03125°C. Somit entspricht das 9. Bit des Datenworts genau einem Grad. Unten die Rechnung von Bits zu °C.

```
  Temperatur=Temperatur+Bit;  //Bit in Temperaturwert einfügen
  Temperatur=Temperatur*2;  //Bits eine Stelle nach links schieben
  count++;
}

Temperatur_Grad=Temperatur;    //Temp_Grad ist ein float, für Kommastellen benötigt
Temperatur_Grad=Temperatur_Grad/64; //Umrechnung auf °C
Temperatur_Grad=Temperatur_Grad-4.54; //Erwärmung im Gehäuse abziehen
```

8.4.4 Relais

Allgemeines

Relais können als elektrisch gesteuerte Schalter verwendet werden, um den Stromfluss in einer Schaltung zu erlauben oder zu unterbrechen. Ein Relais besteht aus Schaltkontakt, der durch eine Spule betätigt wird. Wenn ein Strom durch die Spule fließt, erzeugt dies ein Magnetfeld, das den Schaltkontakt anzieht und den Stromkreis schließt. Schaltet man das Schaltsignal ab öffnet sich der Kontakt wieder. Ein Problem mit dem mechanischen Schaltkontakt ist es, dass er empfindlich auf hohe Leistungen reagieren kann, es kann passieren, dass er geschlossen hängen bleibt.[9]

Anwendung

In der ersten Version wurde das Relais V23057 verwendet.[10] Die schaltbaren Anschlüsse wurden dabei auf 3 Pins geführt (rechte Seite). Zur Testung können ein Netzteil und eine Last, beispielsweise eine LED verwendet oder auch einfach ein Ohmmeter verwendet werden. Zusätzlich ist beim Schalten des Relais ein Klicken zu hören. Die Freilaufdiode D6 muss angebracht werden damit der Strom, der nach dem Abschalten aufgrund der Induktivität des Relais noch fließt, nicht abrupt unterbrochen wird, sondern bis zur Entladung der Spule weiter fließen kann. Ansonsten würde sich die Spannung am Transistor erhöhen, bis es zu einem Durchbruch käme. Der Transistor wird vom ESP über den Vorwiderstand R5 angesteuert. Die unten dargestellte Schaltung ist auf dem Board zwei Mal angebracht. Dadurch kann auch das Verhalten eines Doppelwippenschalters nachgestellt werden.

.

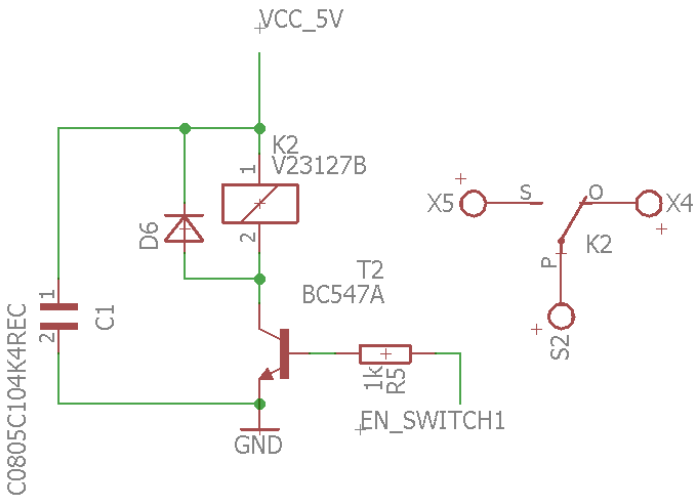


Abbildung 9: Kartenrelais V1

[9] Vgl. Relais • Was ist ein Relais? Wie funktioniert ein Relais?
[10] Vgl. 3-1393215-5 . - Leistungsrelais, SPDT, 5 VDC, 8 A, V23057, Durchsteckmontage.

8.4.5 JTAG-Schnittstelle

Die JTAG-Schnittstelle wird angebracht, um mögliche Alternativen zur ESP-Programmierung zu erforschen. Weiterführend kann sie ansonsten auch zum Debuggen verwendet werden.

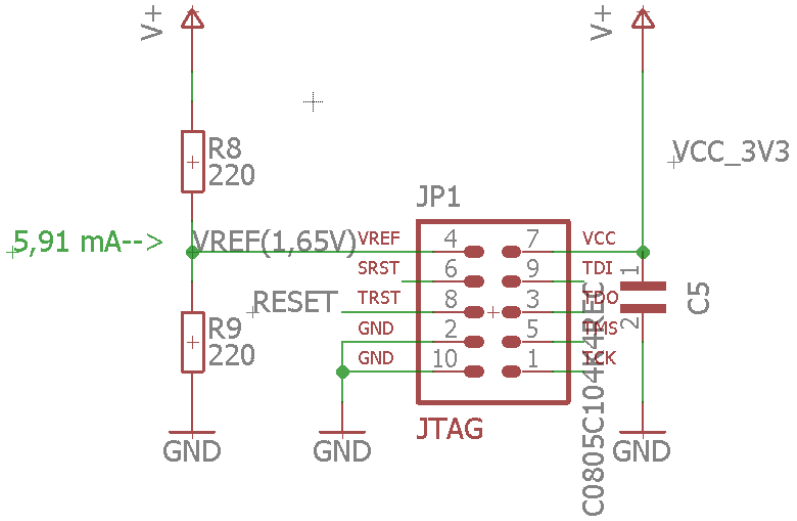


Abbildung 10: JTAG Schnittstelle V1

Nach einigen Tests und Recherche stellt sich die USB-Schnittstelle aber als einfachere Variante heraus. Zum Debuggen eignet sich ebenfalls eine andere Schnittstelle nämlich die UART besser.[11]

[11] Vgl. JTAG Debugging - ESP32-S3 - — ESP-IDF Programming Guide latest documentation.

8.4.6 Layout

Das Layout des Versuchsboards hat sehr gut funktioniert. Es konnten alle Schaltungsteile erfolgreich in Betrieb genommen werden.

Ein Fehler des Layouts war lediglich, dass die Abstände zwischen Netzspannung und den zum Prozessorteil führenden Leiterbahnen zu gering waren. Ebenfalls darf die Leiterbahn keinesfalls durch den Sperrbereich verlaufen. Die Fehler sind in Abbildung 12 gekennzeichnet. Da kein Platzproblem vorhanden war, wäre die Lösung einfach gewesen in dem man die Relais‘ weiter nach oben versetzt hätte.

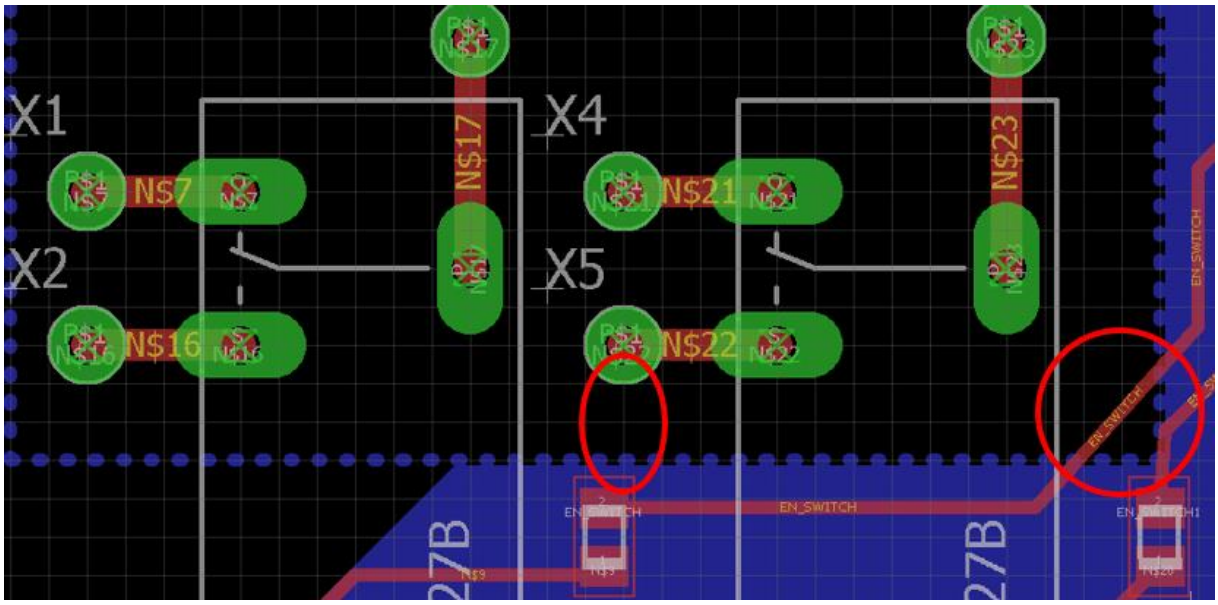


Abbildung 11: Netzspannungsbereich Verletzung V1

Weiters hätte die !Alert Leiterbahn auf der Rückseite geführt werden müssen (Durchkontaktierungen). Hier, in Abbildung 12, nicht schön gelöst.

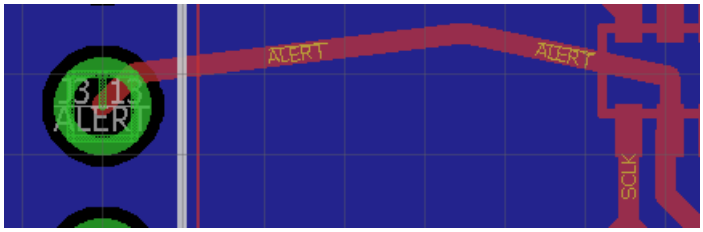


Abbildung 12: Temperatur Sensor suboptimaler Anschluss V1

8.4.7 Inbetriebnahme

8.4.7.1 Platine

Bis auf einige Probleme mit den Relais Lieferungen und anfänglichen Schwierigkeiten mit dem Temperatursensor, lief die Inbetriebnahme der Platine ziemlich reibungslos ab.

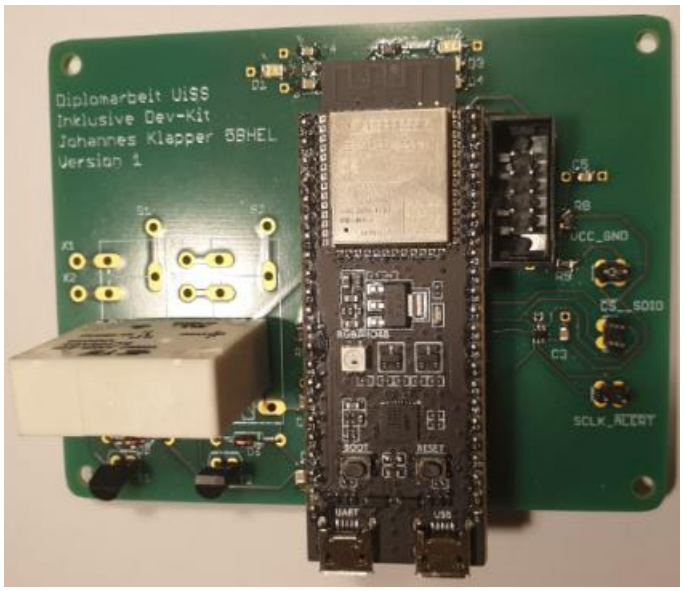


Abbildung 13: Schalter V1

Zuerst wurden Relais für das Schalten der Netzspannung bestellt, welche für die finale Anwendung notwendig sind. Für die Testung sind sie allerdings ungeeignet. Bei der Bestellung der Relais niederer Spannung gab es ein Lieferproblem und es wurde eine Version der Relais mit falschem Footprint geliefert (in Abbildung 13 zu sehen). Die Tests wurden deshalb mit diesem Relais durchgeführt. Details zum Relais kann man im dazugehörigen Datenblatt finden, siehe CD.

8.4.7.2 Messungen

Die Funktionalität der Platine konnte mithilfe folgender Messungen und Tests bestätigt werden. Für die Tests wurden meistens die LEDs zur Ausgabe von Signalen genutzt. Die Messungen wurden mit einem Multimeter und einem Analog Discovery durchgeführt.

Testung der Grundfunktionen

Dass die Spannungsversorgung funktioniert, konnte mithilfe der Power-On LED (D1) bestätigt werden. Mithilfe des kleinen nachfolgenden Testprogramms werden nacheinander die verschiedenen LEDs ein- und ausgeschaltet. Ein Relais wird ebenfalls im Sekundentakt umgeschaltet. Beide dieser Funktionen haben fehlerlos funktioniert.

```
void setup() {
    pinMode(4,OUTPUT);
    pinMode(5,OUTPUT);
    pinMode(6,OUTPUT);
    pinMode(7,OUTPUT);
}

void loop() {

  digitalWrite(4,HIGH);
  delay(1000);
  digitalWrite(4,LOW);
  digitalWrite(5,HIGH);
  delay(1000);
  digitalWrite(5,LOW);
  digitalWrite(6,HIGH);
  digitalWrite(7,HIGH);
  delay(1000);
  digitalWrite(6,LOW);
  digitalWrite(7,LOW);
}
```

Strommessung

Die Platine wird über den USB-Anschluss versorgt. Die 5V Leitung der USB-Verbindung wird am Kabel aufgetrennt. Dazwischen wird ein Amperemeter geschalten. Die Platine benötigte während der Programmierung 121,2 mA (rechtes Bild). Während des laufenden Programms, Datenaustausch via BLE, variiert der Strom recht stark in einem Bereich von 65 – 95 mA. Das liegt unter anderem an dem wiederholten Advertisen und der Kommunikation. Es kann näherungsweise ein linearer Strom von 80 mA angenommen werden. Der Grund für den niedrigen Stromverbrauch, ist sowohl die geringe Umgebungstemperatur als auch die nicht vollständige Auslastung der Platine. Damit ist gemeint, dass nicht alle benötigten Funktionen verwendet werden.

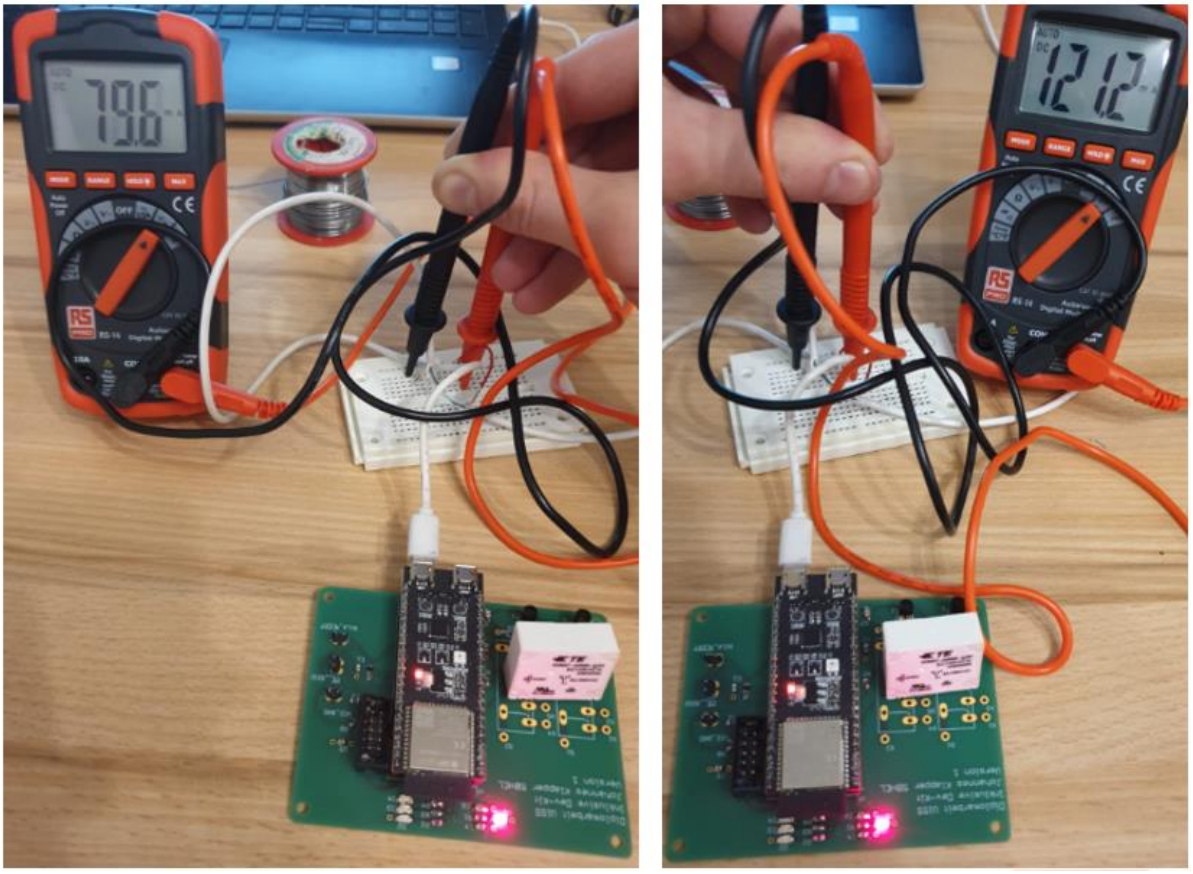


Abbildung 14: Stormmessung DevKit BLE

SPI-Schnittstelle ausmessen

Abbildung 15: Messung mit Analog Discovery

Die SPI-Schnittstelle wird mit einer Baudrate von 9600 Hz betrieben. Unten stehen die dabei ausgelesenen Signale.

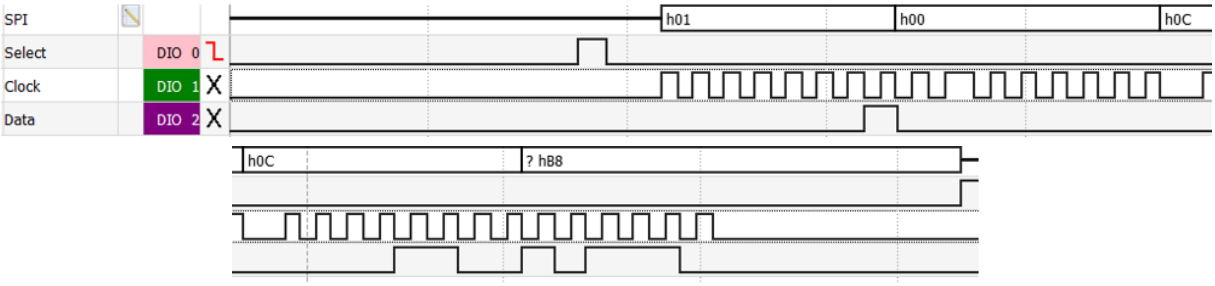


Abbildung 16: SPI-Signal

Die Ausgabe im oberen Bild stellt den Write Befehl dar. Unten werden die Temperaturdaten ausgelesen. Aufgrund der 14 Bit Ausgabe der Temperatur werden die letzten zwei Bits nicht ausgelesen. Deshalb wird beim letzten Byte ein „?“ angezeigt.

Die Temperatur wird in Hex Zahlen übertragen. Die Umwandlung in eine binäre Temperatur erfolgt nach dem Auslesen der Schnittstelle im Code (Siehe Kapitel 8.4.3.3).

Ausgabe im Terminal: `Temp = 1628 25.44`

Die erste Zahl ist der Hex-Wert, die zweite der Dezimal-Wert.
Die Temperatur beträgt 25,44°C.

8.5 Smart-Switch Version 2

8.5.1 Spezifikationen

Die zweite, finale Version des Smart-Switches wurde so gebaut, dass sie in eine Wandinstallations-Einbaudose von EU-Normgröße passt. Weiters erfolgt die Stromversorgung nicht mehr über ein Mikro USB Kabel, sondern über einen AC/DC Wandler direkt von der Netzspannung. Der Netzspannungsbereich muss galvanisch komplett vom Rest der Schaltung getrennt sein. Luft- und Kriechstrecken sind zu beachten.

Abbildung 17: Smart-Switch Version 2 in Buchse

8.5.2 Schaltung

Die Schaltung weicht aufgrund der anderen Spezifikationen von der Version 1 ab. Es werden einige Bauteile durch alternativen ausgetauscht (z.B. Relais), weiters werden einige für die Versorgung relevante Schaltungsteile hinzugefügt, wie der Linearregler inklusive Beschaltung.

8.5.2.1 Versorgung

Die Versorgung des Niederspannungsteils wird anschließend anhand des Netzspannungsanteils und des Linearreglers im Detail erklärt. Anschließend folgt eine Übersicht der Implementierung der einzelnen Bauteile.

8.5.2.1.1 Netzeil

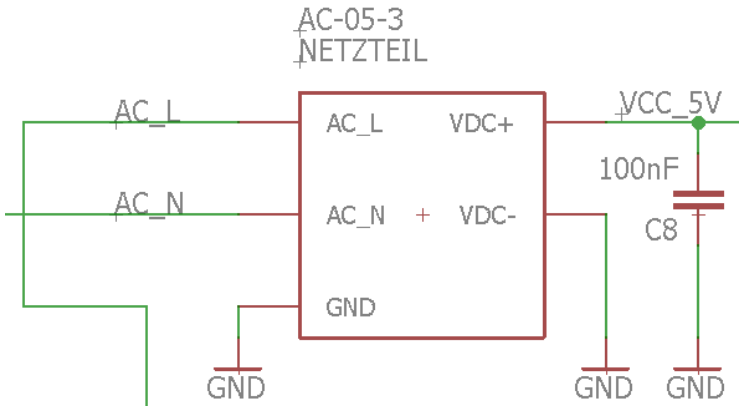
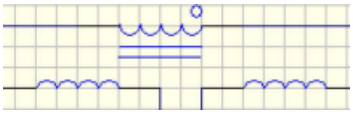
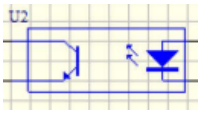


Abbildung 18: Netzteil V2

Die Aufgabe des Mini-Netzteils ist es als Spannungsversorgung für die restliche Platine zu fungieren. Dabei sollte es möglichst präzise eine 5V Ausgangsspannung liefern. Dafür ist dieses Netzteil mit der sehr guten Toleranz von 2,5% und dem Temperaturkoeffizienten von nur 0,03%/°C ideal geeignet. Es wurde immerhin auch für die Versorgung von ESPs oder Raspberry-Pis entworfen. Der Kondensator dient zur weiteren Glättung der Spannung.





Im Netzteil werden die AC und DC-Teile galvanisch komplett getrennt. Das bedeutet, dass trotz einer Energieübertragung, die unterschiedlichen Schaltungsteile nicht direkt miteinander verbunden sind. Zudem bietet der AC/DC Wandler einen guten Schutz des Niederspannungsteils gegen Störungen von außen. [12]

[12] Vgl. Galvanische Trennung - was ist das?

8.5.2.1.2 Linearregler

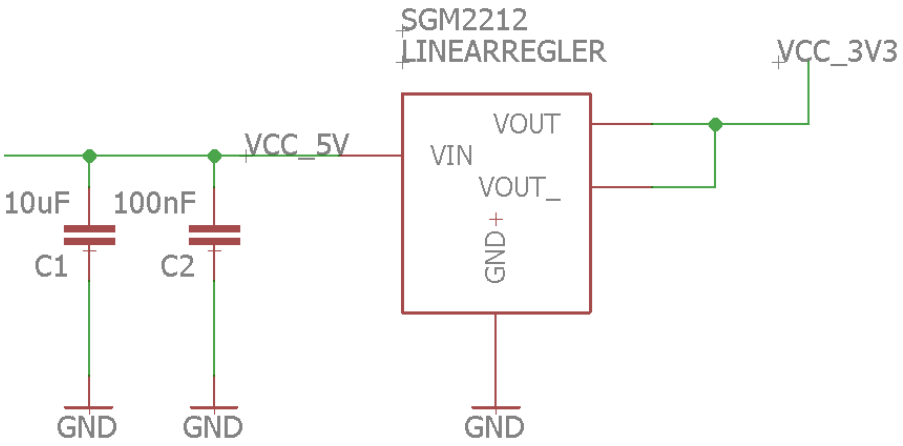


Abbildung 19: Linearregler V2

Der Linearregler wandelt die 5V nach einer weiteren Glättung durch die Kondensatoren, auf die vom ESP benötigte, 3,3V Versorgungsspannung um. Das Vout_-Pad dient zur Kühlung des Linearreglers und es muss mit Vout verbunden werden. Durch die Spannungswandlung und den möglichst gleichbleibenden Strom wird einiges an Leistung in Wärme umgewandelt, welche abgeführt werden muss.[13]

8.5.2.1.3 Sicherung

Die Sicherung dient zum Schutz des Relais im Fehlerfall.

Es wurde eine SMD-Sicherung ausgewählt, um Platz zu sparen. Die Sicherung hat einen Nennwert von 5A, das bedeutet, dass sie bei diesem Strom sicher nicht auslöst. Der Strom ist für die vorgesehene Anwendung viel zu hoch. Allerdings ist es so möglich, mit dem Smart-Switch große Lasten zu steuern, wie die Beleuchtung einer Lagerhalle. Erst bei höheren Strömen über längere Zeit oder sehr viel höheren Strömen nach kurzer Zeit wird sie durchbrennen und damit die Schaltung schützen. Dieser Wert ist der Größte, der für eine Sicherung mit diesem Footprint verfügbar ist. Die Leiterbahnen und die anderen Bauteile währen in der Lage noch mehr Strom zu leiten bei einer Überlastung, durch einen zu großen Stromverbrauch der Last, brennt also zuerst die Sicherung durch.

Abbildung 20: Sicherung V2

[13] Vgl. Ewald, W. (2020).

8.5.2.1.4 Solid State Relais

Allgemeines

Ein Solid State Relais (SSR) ist ein Bauelement, das verwendet wird, um elektrische Lasten ohne die Verwendung beweglicher mechanischer Teile zu schalten. SSRs bestehen aus einem Sensor, einer Steuerschaltung und einem elektronischen Schalter (z.B. ein Thyristor).

Wenn ein Steuersignal an den Eingang des SSR angelegt wird, aktiviert die Steuerschaltung das Schaltgerät, das dann den Stromfluss durch den Ausgangskreis ermöglicht. SSRs bieten mehrere Vorteile gegenüber herkömmlichen Relais, wie z. B. schnellere Schaltgeschwindigkeit, längere Lebensdauer, kleinere Größe, geringeres Rauschen und höhere Zuverlässigkeit. Ebenfalls sind sie resistenter gegen große Leistungen, die bei uns aufgrund der unbekannten Last auftreten könnten.[14]

Anwendung

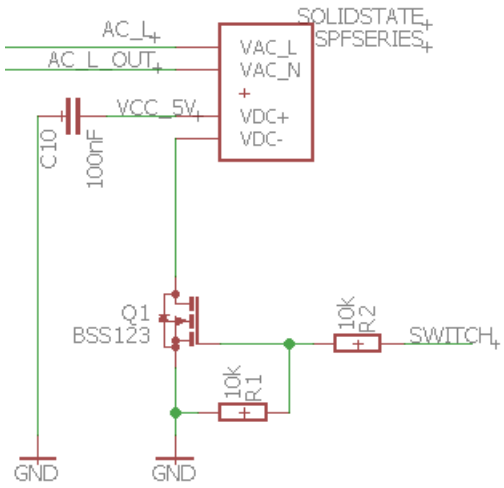


Abbildung 21: Solid State Relais V2

Beim zweiten Relais verhält es sich ähnlich wie beim Smart-Switch V1, es wird allerdings das Solid-State Relais SPF240D25 verwendet und der bipolare Transistor wurde durch den FET Q1 ersetzt. Das Relais wird benötigt, um die Netzspannung von 230V AC mit dem Mikrocontroller schalten zu können. Der Pulldown des Gate Anschlusses des FETs ist nötig, um definierte Verhältnisse zu schaffen (Gate auf Ground ziehen), da beim Einschalten des ESPs der State des Pins Switch (GPIO 21) hochohmig ist.

[14] Vgl. Sicheres und effizientes Schalten von Strom oder Spannung mit Hilfe von Halbleiterrelais.

8.5.2.2 ESP-Anschluss

8.5.2.2.1 Temperatursensor (JTAG-Anschluss)

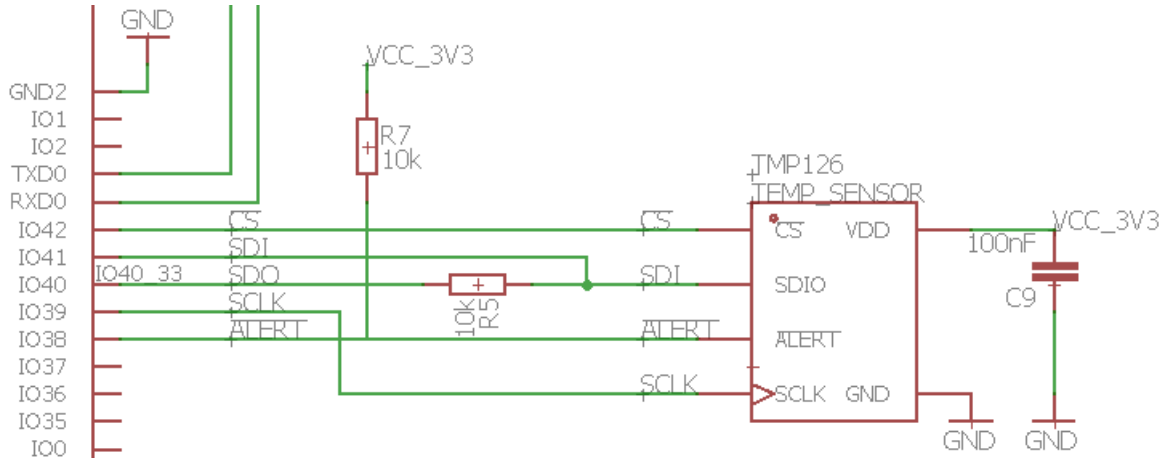


Abbildung 22: Temperatursensor V2

Der Alert Pin sollte mit der Versorgungsspannung verbunden werden, damit unabhängig vom CS Zustand Benachrichtigungen gesendet werden können. Der Widerstand R7 wird hierbei benötigt damit kein zu großem Strom über den Alert Pin fließt. Der Kondensator dient zur Glättung der Versorgungsspannung.

Auf der Seite des Mikrocontrollers müssen SDI und SDO über einer 10 kOhm Widerstand zusammengeschalten werden. Dies muss gemacht werden, damit die 3 Pin SPI korrekt angesteuert werden kann.

Programmierung

Im Gegensatz zum Anschluss der 1. Version kann hier, aufgrund der korrekten Pin-Beschaltung, das vordefinierte SPI-Protokoll der Arduino IDE eingesetzt werden. Es werden zwar nicht die klassischen SPI-Pins verwendet, diese Anpassung kann im Code aber sehr einfach vorgenommen werden. Der Alert Pin kann nun ebenfalls unabhängig vom CS-Pin ausgelesen werden, wodurch sich der Code um einiges erleichtert. [15]

```
SPI.beginTransaction(SPISettings(9600, MSBFIRST, SPI_MODE0));
```

[15] Vgl. Arduino & Serial Peripheral Interface (SPI) | Arduino Documentation.

8.5.2.2.2 USB-Schnittstelle und Test/UART Pins

Der primäre Programmier-Anschluss ist die USB-Schnittstelle. Sie kann exklusiv verwendet werden. Die Pins dienen primär zur Testung, könnten aber mit einer Vorbeschaltung auch zur Programmierung verwendet werden. Dies ist aber nur eine Sicherheitsmaßnahme, dass im Falle von unvorhersehbaren Problemen mit der USB-Schnittstelle eine andere Möglichkeit gibt.[16]

USB-Schnittstelle

Der ID-Pin muss nicht angeschlossen werden, er dient zur Unterscheidung zwischen dem Host (Typ A) und der Peripherie (Typ B), bei der Kommunikation über USB 2.0. Der Pin bleibt offen, was den ESP zum Host macht, auf der Peripherieseite wir der Pin mit dem Ground verbunden.[17]

Die Datenleitungen werden direkt auf die GPIOs 19 und 20 des ESP geführt. Dabei werden sie jeweils über eine bidirektionale Diode mit dem Ground verbunden. Die Dioden dienen zum Schutz der Datenleitungen vor Störungen durch beispielsweise ESD. Sie müssen bidirektional sein, da die Datenleitung auch negative Spannungen haben können und die Dioden deshalb in beide Richtungen leiten müssen.

Während der Programmierung des ESPs erfolgt die Spannungsversorgung über den Usb-Stecker und die Diode CR1. Werden später die benötigten 5V vom Netzteil geliefert, verhindert die Diode, dass ein externer Verbraucher versorgt wird. [18] [19]

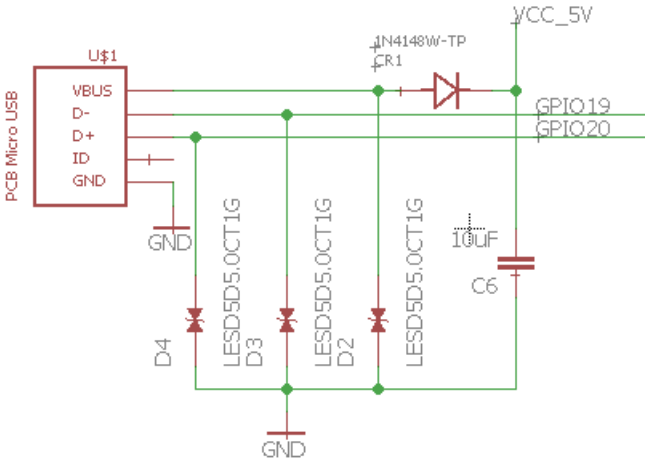


Abbildung 23: USB Anschluss V2

[16] Vgl. Eine Einführung in Mini-USB: Definition, Funktionen und Verwendung (2021).
[17] Vgl. Glaser, O. (2012).
[18] Vgl. Eine Einführung in Mini-USB: Definition, Funktionen und Verwendung (2021).
[19] Vgl. USB: Pinbelegung von USB A, B, C und Micro-USB (2022).

Pins

Die USB-Schnittstelle ist funktional, deshalb können die Pins wie geplant zur Testung eingesetzt werden. Die Ausgabe findet an 4 Pins statt. Die Datenleitungen werden direkt an den ESP angeschlossen. Ebenfalls werden Vcc wie Ground angeschlossen. Um den Anschluss als „Universal Asynchronous Receiver Transmitter“ oder UART müssen die Datenleitungen der USB-Schnittstelle an Txd und Rxd angeschlossen werden.

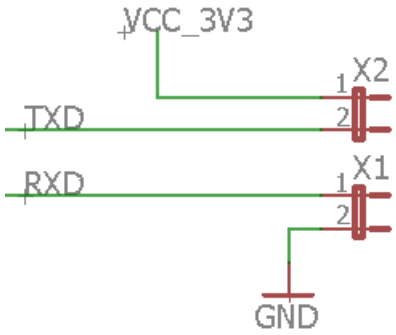


Abbildung 24: Testpin V2

8.5.2.2.3 Taster

Der mechanische Taster ist standardmäßig im Leerlauf, es fließt kein Strom. Wird er betätigt so schließt sich der Kontakt und die Taster_V- Spannung liegt am GPIO 14 an. Die dem Taster folgende Beschaltung ist eine Entprellung mit einer Dauer von 46,90 ms. Damit wird vorausgesetzt, dass der Taster mindestens für 1/20 Sekunde betätigt werden muss, damit er registriert wird.

$$t = \sqrt{R * C} = 46{,}90\ ms$$

Der Tiefpass, der aus R6 und C3 gebildet wird, ist für diese Entprellung verantwortlich. Der R4 Pulldown wird benötigt, damit bei nicht betätigtem Taster der IO14 Pin nicht undefiniert ist (er wird auf „LOW“ gezogen).

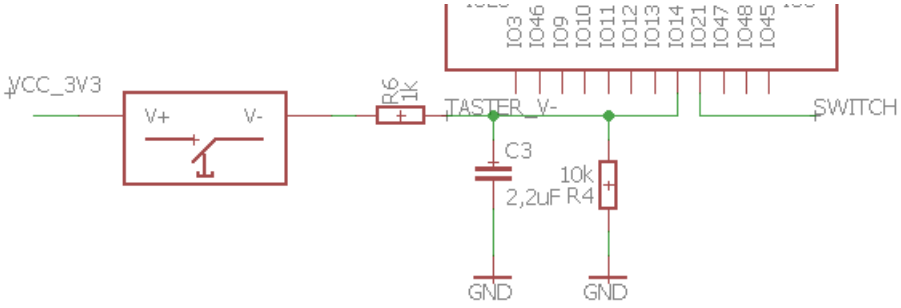


Abbildung 25: Tasteranschluss V2

8.5.2.2.4 Spannungsversorgung

Die 3,3V kommen vom Ausgang des Spannungsreglers, sie werden auf der Seite des ESP sicherheitshalber nochmals, mit Hilfe der Kondensatoren geglättet. Der Enable Pin benötigt einen externen Pullup auf 3,3V damit der ESP gestartet wird. Bekommt der Enable Pin keine Spannung wird der ESP ausgeschalten. Die Diode dient wie beim Smart-Switch Version 1 der Testung, diesmal wurde allerdings ein etwas größerer Widerstand verwendet, um den Stromverbrauch etwas zu senken.

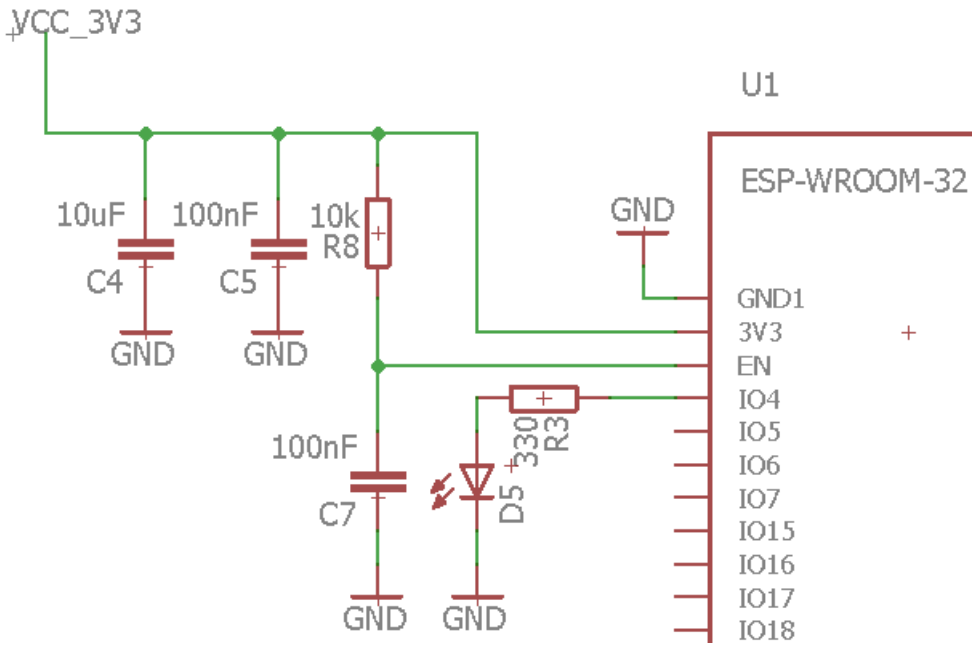


Abbildung 26: Spannungsversorgung ESP V2

8.5.3 Layout

Beim Entwurf des Layouts mussten einige wichtige Schritte durchgeführt werden. Erstens um die Funktionalität der Schaltung mit höchstmöglicher Sicherheit zu garantieren. Zweitens um alle folgenden Anforderungen zu erfüllen.

Anforderungen

- Der Durchmesser einer genormten Schalterbuchse beträgt 68 mm, die Tiefe variiert zwischen 40 und 60 mm.[20]
- Die Ambient Temperatur im Gehäuse, muss so nahe wie möglich an der Außentemperatur liegen
- Das thermisch gesehen kritischste Bauteil darf nicht überhitzen
- Die verbrauchte Leistung hat so klein wie möglich zu sein

8.5.3.1 Größenoptimierung

Beim Design des finalen Smart-Switches wurde versucht, aufgrund der Produktionskosten die Bauteile nur auf der Top Seite zu platzieren. Dies stellte sich aus Platzgründen allerdings als unmöglich heraus. Deshalb wurden Bauteile auf „TOP“ und „BOTTOM“ Seite platziert, das verbesserte ebenfalls das Handling.

Platzsparende Maßnahmen:

- Minimale Bauteil- und Leiterbahnabstände ausnutzen (Bauteilnorm 0805)
- Einige Bauteile auf der Rückseite der Platine anbringen
- Größte Diagonale der Platine minimieren

Es wurden das Netzteil, die LED, die Test Pins und der Taster mit der jeweiligen Beschaltung auf der Rückseite platziert. So sind alle Testungsmöglichkeiten, auf einer Seite platziert. Ebenfalls ist der Taster so gut zugänglich. Das Netzteil wurde aus reinen Platzgründen auf der Bottom Seite angebracht.

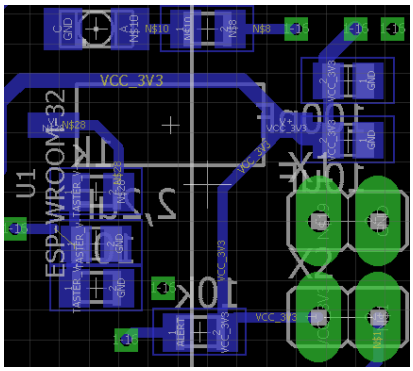


Abbildung 27: Bottom Bauteile Layout V2

[20] Vgl. Feli (2021).

Mit der größten Diagonale der Platine ist der größte Abstand von zwei Punkten auf der Platine gemeint. Der Abstand muss kleiner als der Durchmesser einer Buchse sein, damit die Leiterplatte hineinpasst. Die Diagonale konnte auf 64,91 mm verkürzt werden, damit ist die Anforderung erfüllt.

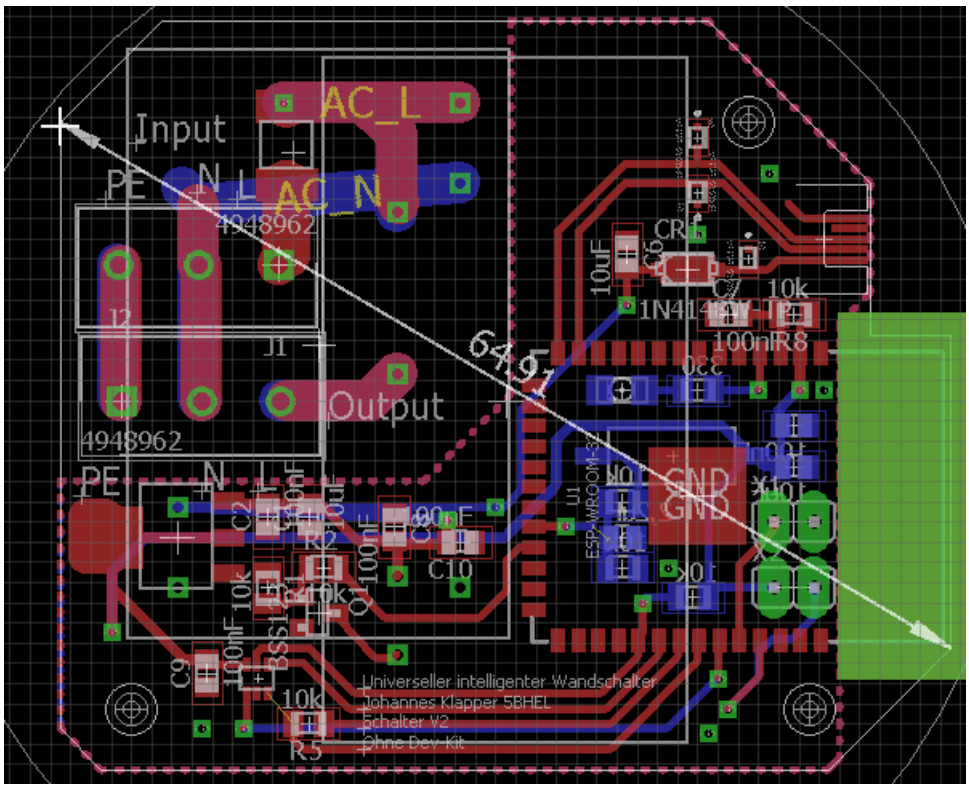


Abbildung 28: Größte Diagonale

8.5.3.2 Thermische Berechnungen

Für die thermische Berechnung wird näherungsweise angenommen, dass die Aluminiumplatte, an der Innenwand des Gehäuses die Temperatur im Gehäuse perfekt gleichmäßig verteilt. Die Temperatur im Gehäuse wird als Ambient bezeichnet. In der Aluplatte werden Lüftungsschlitze angebracht, um die Kühlung zu verbessern und um für eine erhöhte BLE-Reichweite zu sorgen.

8.5.3.2.1 Linearspannungsregler

Das kritischste Bauteil beim Thema Abwärme ist der Linearspannungsregler zwischen 5V und 3,3V, da hier die Spannungsdifferenz in Abhängigkeit vom Strom direkt in Wärme umgewandelt wird. Da das Bauteil aber sehr klein ist, könnte es auch leicht zu einer Überhitzung führen, die maximale Ambient Temperatur muss deshalb ebenfalls anhand des Linearreglers geprüft werden. Das Bauteil wird einerseits über seine Pins über die Leiterplatte gekühlt, andererseits direkt durch die Ambient-Luft.

Die thermischen Widerstände des Bauteils können dem Datenblatt entnommen werden.

$\theta_{J,A} = 117\ K/W$ … Thermischer Widertand Junction zu Ambient (Luft)

$\theta_{J,P} = 29\ K/W$ … Thermischer Widertand Junction zu Pin groß

Der Thermische Widerstand zu den kleinen Pins kann ignoriert werden, da sich die Kühlfläche auf der Platine direkt am großen Pin befindet, an die kleinen Pins aber nur vergleichsweise dünne Leiterbahnen angeschlossen sind.

Der thermische Widerstand der Kühlfläche wurde von einem Fachmann von b2 abgeschätzt.

$\theta_{J,Co} = 50\ K/W$ … Thermischer Widertand Junction zu Copper (Kühlfläche)

Thermische Schaltung

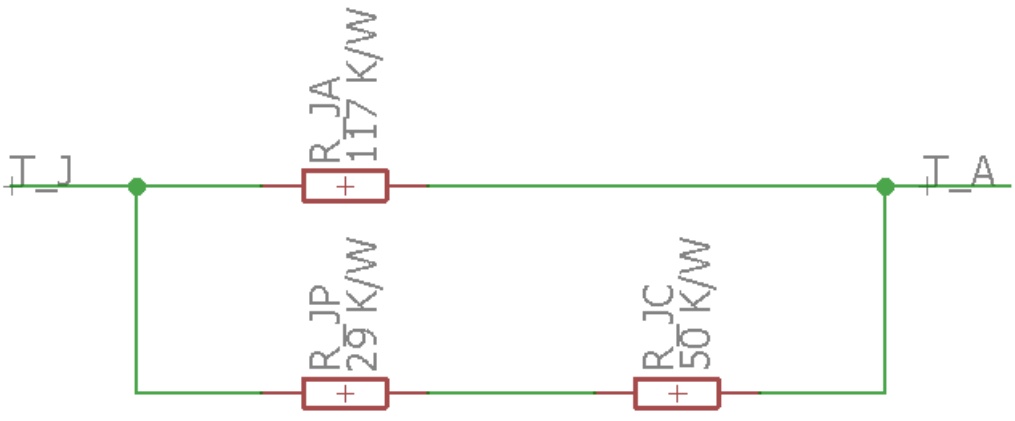


Abbildung 29: Thermische Schaltung V2

Junction zum Pin und Pin zur Leiterbahn liegen logischerweise in Serie. Die Kühlungen über die Leiterbahn und die Luft sind parallel.

$\theta_{LGes} = \frac{(\theta_{J,P}+\theta_{J,Co})*\theta_{J,A}}{(\theta_{J,P}+\theta_{J,Co})+\theta_{J,A}} = 47{,}16\ K/W$ … Thermischer Gesamtwiderstand Linearregler

$T_{JMax} = 150\ °C$ … Maximale Junction Temperatur

$T_{GMax} = 60\ °C$ …Maximale Ambient Temperatur, also Temperatur im Innenraum des Gehäuses

$P_{LMax} = \frac{T_{JMax}-T_{GMax}}{\theta_{LGes}} = 1{,}91\ W$ … Maximale Leistung Linearregler

$I_{LMax} = \frac{P_{LMax}}{U_{Diff}} = 1{,}12\ A$ … Der Linearregler könnte also einen wesentlich größeren Strom treiben als die von uns maximal benötigten 400 mA.

$I_{Real} = 400\ mA$ … Realer Stromverbrauch DC-Schaltungsteil

$P_{LReal} = U_{Diff} * I_{Real} = 680\ mW$ … Reale in Wärme umgewandelte Leistung

$\Delta T_{L,A} = P_{LReal} * \theta_{LGes} = 32{,}07\ K$ … Realer Temperaturunterschied zwischen Linearregler und Ambient

$T_L = T_{GMax} + \Delta T_{L,A} = 92{,}07\ °C$ … Die maximale Temperatur des Linerarreglers

Da die maximale Temperatur des kritischsten Bauteils weit unter dem maximalen Temperaturwert liegt, den der Linearregler erreichen darf, haben wir Wärmetechnisch kein Problem.

8.5.3.2.2 Relais

$I_{ACMax} = 5\ A$ … Maximaler AC-Strom

$U_{AC} = 230\ V$ … AC-Spannung

$U_{MaxDP} = 1{,}6\ V$ … Maximum On Voltage Drop Peak

$U_{MaxD} = \frac{U_{MaxDP}}{\sqrt{2}} = 1{,}13\ V$ … Maximum On Voltage Drop

$P_{ACMax\_Verlust} = I_{ACMax} * U_{MaxD} = 5{,}66\ W$ … Maximale Verlustleistung am elektronischen Schaltkontakt

$P_{ACMax} = I_{ACMax} * U_{AC} - P_{ACMax_{Verlust}} = 1143{,}34\ W$ … Maximaler Leistungsverbrauch der Last

8.5.3.2.3 Gesamtes Gehäuse

Das Gehäuse muss die Temperatur entstehen durch die gesamte Verlustleistung aller Komponenten im Gehäuse abführen. Im Wesentlichen setzt sich die Leistung aus zwei Komponenten zusammen.

$U_{DCMax} = 5\,V$ … Maximale DC-Spannung

$P_{DCReal} = I_{Real} * U_{DCMax} = 2\,W$ … Maximale Leistung des DC-Anteils

$P_{Ges} = P_{ACMax\_Verlust} + P_{DCReal} = 7{,}66\,W$ …Maximale Leistung des Relais

$C_{VS} = 200\,\frac{K*cm^3}{W}$… Constant Vertical Surface ist eine gängige Konstante wie sie von Entwicklern zur annähernden Kühlwirkung von senkrechten Flächen verwendet wird.

Es ist sehr relevant, dass die Platine vertikal in der Buchse verbaut wird, es wirkt dann nämlich die Höhe der Kühlfläche quadratisch, durch die Abwärme entstehende Konvektion der Luft.

$h = 4{,}7\,cm$ … Platinen Höhe

$b = 5{,}59\,cm$ … Platinen Breite

$\theta_{U,A} = \frac{C_{VS}}{h^2*b} = 1{,}62\,\mathrm{K/W}$ … Thermischer Widerstand Umgebung zu Ambient

$\Delta T_{U,A} = P_{Ges} * \theta_{U,A} = 13{,}51\,K$ … maximale Temperaturänderung Ambient zur Umgebung

$T_{UMax} = T_{GMax} - \Delta T_{U,A} = 46{,}49\,°C$ … maximale Umgebungstemperatur, bei der die Platine garantiert betrieben werden kann

Um eine normale thermische Änderung zwischen Umgebung und Ambient festzustellen müssen gewöhnliche Verlustleistungen festgelegt werden.

$I_{ACNorm} = 1\,A$ … Gewöhnlicher AC-Strom (Wahlwert)

$P_{DCNorm} = P_{DCReal}/1{,}2 = 1{,}67\,W$ … Gewöhnliche Leistung des DC-Anteils (ohne 20% Puffer)

$P_{ACNorm\_Verlust} = I_{ACNorm} * U_{MaxD} = 1{,}13\,W$… Gewöhnliche Verlustleistung am elektronischen Schaltkontakt des Relais

$P_{Ges} = P_{ACNorm\_Verlust} + P_{DCReal} = 2{,}80\,W$ …Gewöhnliche Gesamtleistung

$\Delta T_{U,A\_Norm} = P_{Ges\_Norm} * \theta_{U,A} = 4{,}54\,K$ … Gewöhnliche Temperaturänderung Umgebung zu Ambient

Die definierten Bauteilwerte wurden den Datenblättern entnommen, siehe CD.

8.5.3.3 Leistungsoptimierung bzw. Management

Um den Leistungsverbrauch des Smart-Switches niedrig zu halten, muss auf einen möglichst niedrigen Stromverbrauch aller Bauteile geachtet werden. Weiters sollte bei ICs darauf geachtet werden, dass sie über einen Standby Modus verfügen. Ein Problem dabei ist die nicht definierte Last, die mit unserem Smart-Switch geschalten wird. So ist es unklar wie viel Strom über die Platine fließen wird. Als Resultat daraus mussten Bauteile mit Toleranzen für hohe Ströme verwendet werden, die einen etwas höheren Verbrauch haben. Ebenfalls benötigen die Leiterbahnen eine entsprechende Breite. Diese Maßnahmen sind speziell bei der Anwendung des Smart-Switches im Haus relevant. Ersetzt man beispielsweise alle Lichtschalter mit UiWs so summieren sich diese Maßnahmen natürlich und sparen auf längere Zeit einiges an Geld.

8.5.3.3.1 Low-Power Mode

Der Low-Power Mode des ESP wird bei jeder Möglichkeit eingeschaltet. Der ESP muss ihn nur verlassen, wenn er ein Eingangssignal von der BLE-Schnittstelle, dem Taster, der USB-Schnittstelle oder dem Alert Pin bekommt. Da die USB-SS nur zur Programmierung benötigt wird, kann dieser Fall während der Verwendung ausgeschlossen werden.

Table 4-8. Current Consumption in Low-Power Modes

Work mode	Description	Typ (μA)
Light-sleep	VDD_SPI and Wi-Fi are powered down, and all GPIOs are high-impedance.	240[1]
Deep-sleep	RTC memory and RTC peripherals are powered on.	8
	RTC memory is powered on. RTC peripherals are powered off.	7
Power off	CHIP_PU is set to low level. The chip is powered off.	1

Abbildung 30: ESP Low Power Modis

BLE

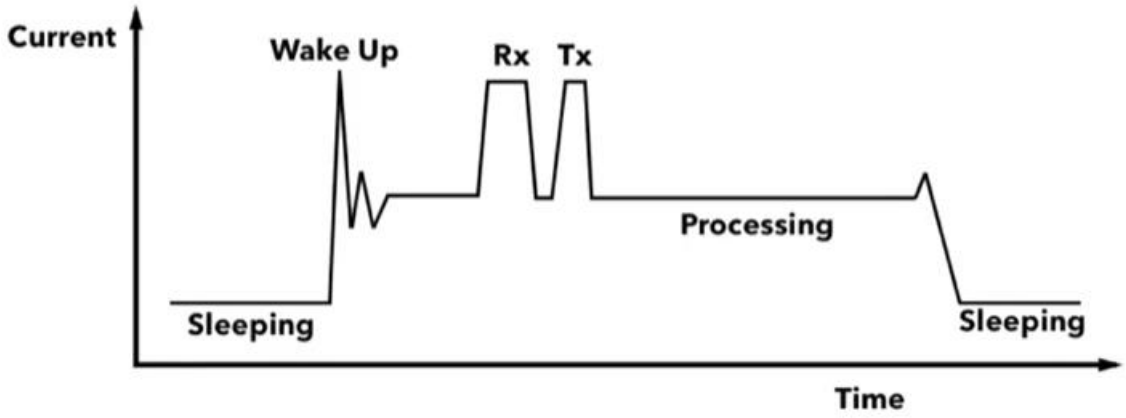


Abbildung 31: BLE-Stromverbrauch

Ein großer Teil der benötigten Leistung wird von der BLE-Kommunikation benötigt. Da nur an bestimmten Zeitpunkten Daten versendet werden, kann hier schon viel Energie gespart werden. Da BLE in den Sleeping Mous umschaltet.

8.5.3.3.2 Absolute Maximum Ratings

Der maximale Verbrauch ist von allen Leiterbahnen und Bauteilen abhängig. Dabei ist die einzige relevante Information die Spezifikationen des schwächsten Glieds. Die Spezifikationen werden zwischen Hoch- und Niederspannungsbereich unterschieden.

Der Hochspannungsteil gibt an wie viel Last Geschalten werden kann. Bei der Verwendung des Smart-Switches ist auf diese Limitierung unbedingt zu achten. Bei Überlastung brennt die Sicherung nämlich durch und muss ausgetauscht werden. Die Last darf maximal 1143,34W an Leistung verbrauchen. Das sollte für alle vorgesehenen Einsatzbereiche mehr als genügen.

Beim Niederspannungsteil muss primär darauf geachtet werden wie viel Strom von den Geräten kollektiv benötigt wird und ob die Versorgung diesen Strom liefern kann. Der ESP selbst verbraucht bei voller Kommunikation über alle angeschlossenen Schnittstellen und die eine Ausgabe von 100% auf alle GPIOs etwa 400 mA. Das hängt wiederum natürlich vom Stromverbrauch der angeschlossenen Bauteile ab. Speziell für das Solid State Relais kann der Stromverbrauch nur approximiert werden. Deshalb sind in die 400 mA etwa ein 20% Puffer eingerechnet.

Leiterbahnen

Es gibt nur eine Leiterbahn, die einen größeren Strom von 5A führt, und zwar die von den Relais Schaltkontakten. Dauerhaft werden größere Ströme durch die Sicherung verhindert.

Auf der Platine wird exklusiv die Normstärke von 35um für die Leiterbahnenverwendet.

Leiterbahnbreiten

Schichtstärke / Kupferendstärke	Leiterbahnbreite		Max. Strom in Abhängigkeit zur Temperaturerhöhung				
	metric	imperial	10°K	20°K	30°K	45°K	60°K
35µm	0.25mm	10mil	0.5A	0.8A	1.0A	1.3A	1.6A
	0.50mm	20mil	1.0A	1.6A	2.0A	2.5A	3.0A
	1.00mm	40mil	2.2A	3.0A	3.6A	4.2A	4.8A
	1.50mm	60mil	3.0A	3.8A	4.6A	5.3A	6.5A
	2.00mm	80mil	3.8A	5.0A	6.5A	7.5A	8.5A
	3.00mm	120mil	4.5A	6.5A	8.0A	9.5A	11.0A

Die verhältnismäßig schmälste Leiterbahn befindet sich hierbei zwischen der Eingangsphase und der Sicherung. Die anderen AC Leiterbahnen sind alle Doppelseitig verlegt und stellen deshalb kein Problem dar. Aber selbst diese Leiterbahn sollte sich mit der Breite von 3 mm

Abbildung 32: Sicherungsanschluss

und einem maximalen Strom von 5 A nicht um mehr als 12,5 K erwärmen. Wenn das Verhältnis zwischen Strom und Temperatur im Bereich von 4,5 – 6,5 A näherungsweise als linear angenommen wird.

8.6 Software

8.6.1 Espressif IDF

Diese Option für die Software-Implementierung wurde zunächst gewählt, da das Programm von der Herstellerfirma des Mikrocontrollers stammt. Die Kompatibilität mit dem Board ist ebenfalls dokumentiert, somit können dort keine Probleme auftreten. Allerdings sind die allgemeine Qualität und Dokumentation des Frameworks schlecht im Vergleich mit Arduino IDE.

Installation

Die Installation für Espressif „IoT Development Framwork“ sollte keinesfalls durchgeführt werden, ohne zuvor eine geeignete „Integrated Development Environment“ oder IDE aufgesetzt zu haben. Die Verwendung ist zwar trotzdem theoretisch möglich, muss aber im Terminal durchgeführt werden und ist dadurch unheimlich aufwendig. Weiterführend muss das Framework komplett und korrekt wieder deinstalliert werden, bevor die Installation in der IDE gestartet wird.[21]

Die Installation mit der IDE Visual Studio Code ist wiederum recht einfach. Es ist in VS-Code möglich Extensionen für alle möglichen Zwecke herunterzuladen, auch Espressif IDF. Ein ausführliches Tutorial zur Installation ist unter der folgenden Quelle zu finden.[22]

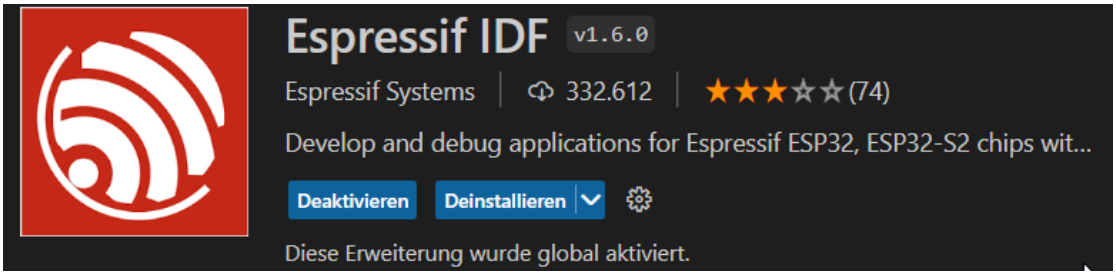


Abbildung 33: Espressif IDF in VS Code

Anschließend muss die Erweiterung aktiviert werden. Mithilfe der der Kommando Eingabe und der Toolleiste können dann alle Funktionen der Erweiterung verwendet werden. Die Navigation ist recht komfortabel, allerdings müssen Fehleinstellungen beim Speichern oder Öffnen von Dateien vermieden werden, diese können ansonsten zu schwer behebbaren Fehlern führen.[23]

Für den ESP32-S3 und die dazugehörigen Dev-Kits gibt es bereits vordefinierte Librarys und der Verbindungsaufbau klappt ebenfalls reibungslos.

[21] Vgl. Get Started - ESP32 - — ESP-IDF Programming Guide latest documentation.
[22] Vgl. ESP-IDF VS Code Extension (2023).
[23] Vgl. IoT Development Framework I Espressif Systems.

Fehlerbeschreibung

Zunächst können die Bibliotheken, die mit der Erweiterung zusammen installiert wurden, nicht gefunden werden.

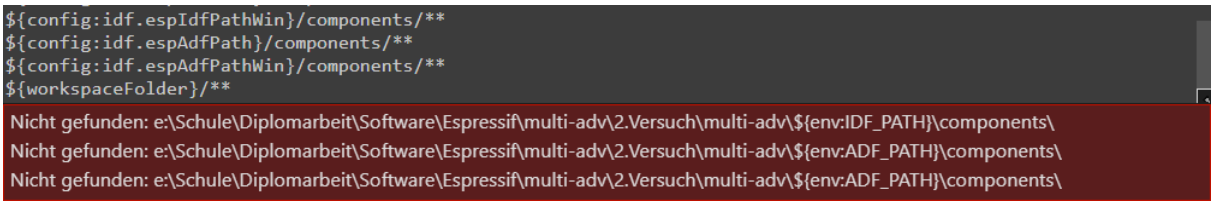


Abbildung 34: CMake Speichpfad Fehler

Die Bibliotheken erneut zu installieren bzw. falls sie installiert wurden, manuell den korrekten Pfad einzugeben behebt dieses Problem.

CMake, eine Erweiterung zur Speicherung und Optimierung von Code, kann dann allerdings die benötigten Dateien nicht erstellen.

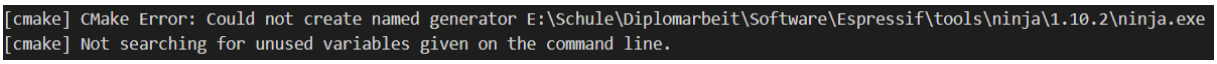


Abbildung 35: CMake Ninja Fehlermeldung

Das Problem mit CMake ist ebenfalls nicht die Behebung dieses Fehlers, sondern die Folgefehler, die daraus entstehen. Nach einigen Stunden der Fehlersuche ist das schlussendlich auch der Grund für das Aufgeben der Espressif IDF in VS-Code als Entwicklungsumgebung.

8.6.2 Arduino IDE

Programmiersprache

Die Entwicklungsumgebung Arduino IDE funktioniert mit einer eigenen Programmiersprache, welche auf C++ basiert. Sie ist speziell auf das Programmieren von Arduino Boards ausgelegt, andere Mikrocontroller lassen sich aber ebenfalls verwenden. Die Sprache ist Benutzerfreundlicher als C++ und kommt mit vielen vordefinierten Bibliotheken. Dadurch wird der Arbeitsaufwand um einiges reduziert, auch wenn die Bibliotheken nicht immer anwendbar sind.[24]

Installation

Die Installation der Arduino IDE ist überaus einfach, es gibt nicht übermäßig viel zu erklären.[25] Die ESP32 Platinen sind allerdings noch nicht eingebunden. Diese können im Library Manager von Github heruntergeladen werden.[26] Im Board Manager können der ESP32-S3 oder das DevKit zur Programmierung ausgewählt werden.

Verwendung

Die Entwicklungsumgebung bietet eine sehr simple Steuerung an. Der Haken dient zur Verifizierung der Syntax des Codes. Der Pfeil dient zum Uploaden des Programms auf den

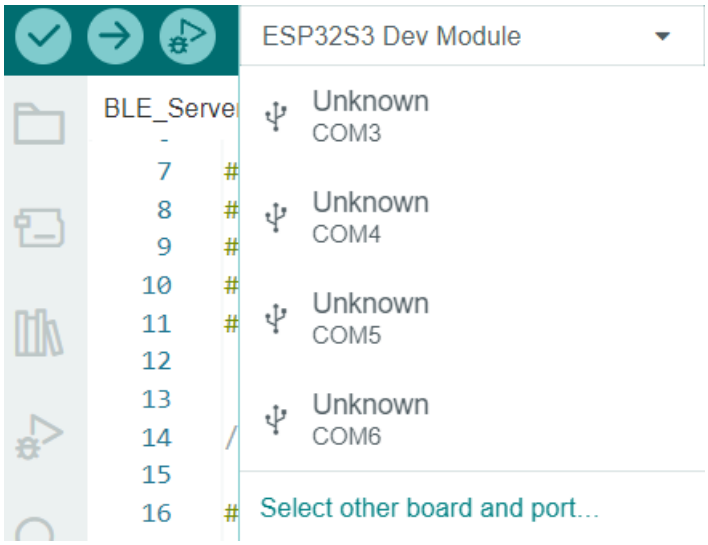


Abbildung 36: Arduino IDE Funktionen

ESP. Das dritte Symbol dient dem Debuggen, dazu wird noch ein zusätzlicher Anschluss über die UART benötigt. In der Liste rechts kann zu guter Letzt das verbundene Board ausgewählt werden und den COM-Port, über den es verbunden ist. Hier ist das Board nicht angeschlossen.

[24] Vgl. moicapnhap.
[25] Vgl. Downloading and installing the Arduino IDE 2.0 | Arduino Documentation.
[26] Vgl. Arduino_ESP32_OTA (2023).

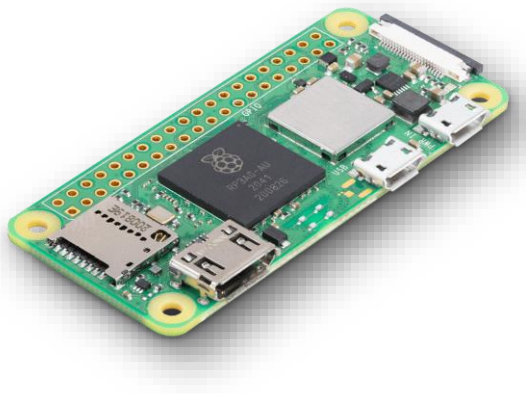
9 BRIDGE

9.1 Allgemeines

Die Bridge stellt das Bindeglied zwischen den einzelnen Smart-Switches und den Clients dar. Sie übernimmt auch die Steuerung der Smart-Switches, damit deren Programme minimal gehalten werden können und somit auch die Kommunikation aller Teilnehmer einfacher gestaltet wird.

9.2 Hardware

Der physische Computerbestandteil besteht hierbei aus einem „Raspberry-Pi“. Dies ist ein kompakter Computer, welcher in Form eines Moduls geliefert wird. Aufgrund des Fokus auf Kompaktheit, fällt die Wahl des Modells hierbei auf das Model Zero 2W, welches mit einer Größe von 65mm x 30mm eines der kompakteren Geräte der Marke ist. Diese Variante verfügt über einen Quad-Core 64-bit ARM-Microchip, welcher mit einer Frequenz von einem Gigahertz und einem Arbeitsspeicher von 512MB arbeitet. Dieser beinhaltet sowohl ein WLAN- als auch ein Bluetooth-Modul, welche beide für die Kommunikation benötigt werden. [27]

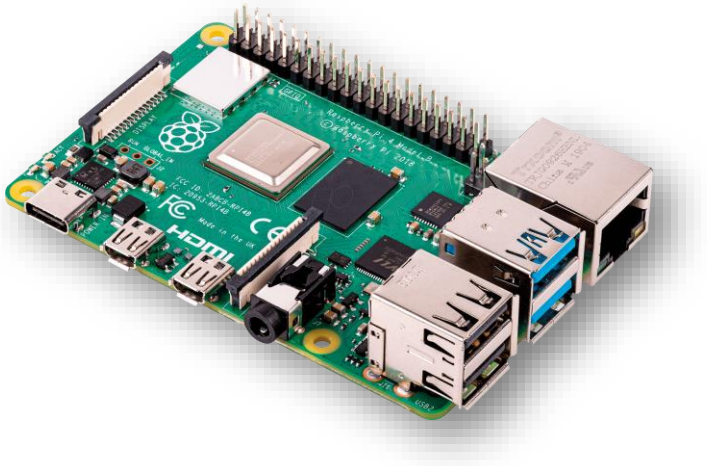


[28]

Abbildung 37: Raspberry-Pi Zero

Durch die derzeitigen schlechten Marktverhältnisse musste auf ein Modell zurückgegriffen werden, welches bereits im Besitz eines Team-Mitgliedes war. Hierbei handelt es sich um das Model 4B in der Variante mit 8GB Arbeitsspeicher. Dieser verfügt über diverse weitere Schnittstellen, wie beispielsweise mehrere USB-A Ports, einem Ethernet-Port oder mehrere micro-HDMI Ports. [29]

[27] Vgl. Ltd, R. P.
[28] Vgl. Abbildung: Ltd, R. P.
[29] Vgl. Ltd, R. P.



[30]

Abbildung 38: Raspberry-Pi 4B

Zum Schutz des Gerätes vor Staub oder Stößen wird noch eine vorgefertigte Hülle verwendet, welche mittels 4 Schrauben befestigt wird.

Abbildung 39: Raspberry-Pi 4B Case

Der Computer wird über die USB-C Schnittstelle mit 5V versorgt. Als Netzteil wird hierbei das mitgelieferte verwendet.

9.3 Betriebssystem

9.3.1 Allgemeines

Das Raspberry-Pi verwendet als primäres Speichermedium eine Micro-SD Karte. Auf diese muss zuvor das verwendete Betriebssystem installiert werden

9.3.2 Raspbian

Raspbian (auch Raspberry Pi OS genannt) ist das meistverwendete Betriebssystem für alle Raspberry-Pi Modelle und wird auch vom offiziellen Anbieter entwickelt. Dieses System basiert

[30] Ltd, R. P.

auf dem Open-Source Betriebssystem GNU/Linux, einem quelloffenen Projekt, welches von jeder Person und jedem Unternehmen frei verwendet werden darf.[31]

Für die Bridge wurde die „Headless“-Variante von Raspbian verwendet. Diese verfügt über kein grafisches Benutzerinterface, da dieses für die Funktionen der Bridge nicht essenziell ist. Mit dieser Methode werden auch Ressourcen gespart.[32]

9.3.3 Installation

Für die Installation des Betriebssystems wird das Programm Raspberry-Pi Imager verwendet, welches von der Raspberry-Pi Foundation bereitgestellt wird. Vor der Beschreibung der SD-Karte können auch noch mehrere Einstellungen vorgenommen werden. Hierbei wird der Hostname der Bridge auf „uiw-Bridge“ gesetzt und ein Passwort für den Root-Benutzer vergeben. Zudem wird noch SSH aktiviert, um eine direkte Entwicklung auf der Bridge zu ermöglichen. [33]

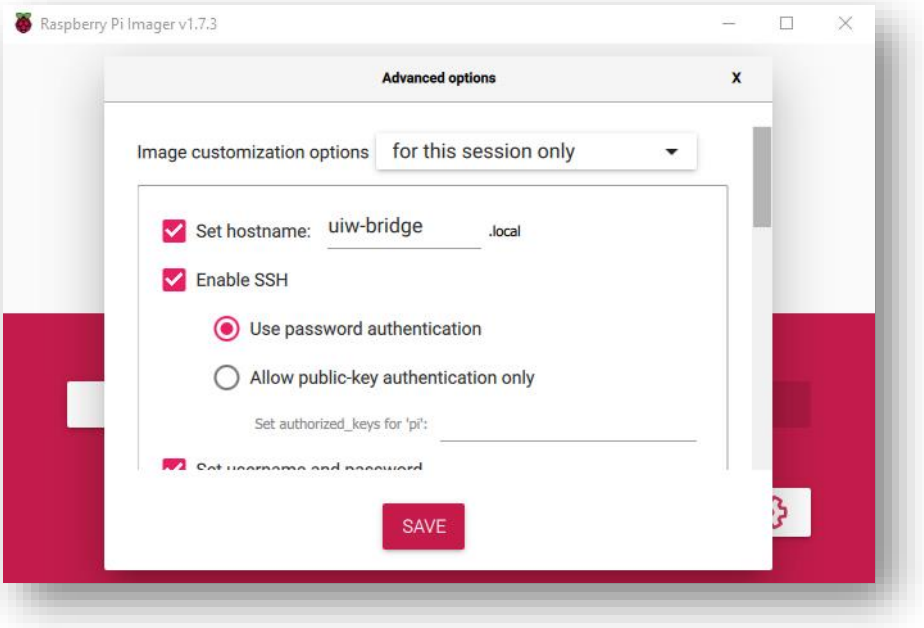


Abbildung 40: Raspberry-Pi Imager

9.4 Software

Für die Erfüllung der Funktionen muss eine individuelle Software für die Bridge erstellt werden. Dieses Programm soll das Gerät beim Hochfahren einrichten, die Kommunikation regeln und die einzelnen Modis abprüfen und anschließend die Smart-Switches steuern.

[31] Vgl. Das GNU-System und Linux - GNU-Projekt - Free Software Foundation.
[32] Vgl. FrontPage - Raspbian.
[33] Vgl. Ltd Raspberry-Pi.

9.4.1 Programmiersprache

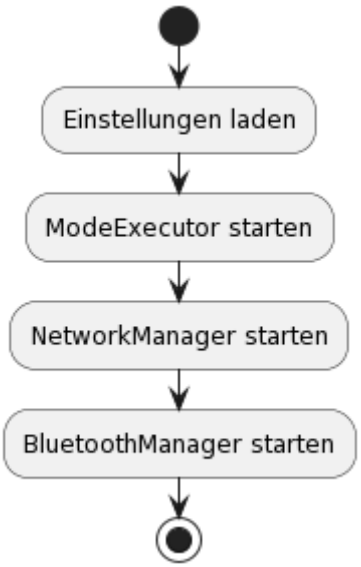
Die Applikation muss in einer Sprache geschrieben werden, welche vom Betriebssystem unterstützt wird. Für das Für die Kombination von Raspbian und dem Modell 4B stehen eine große Auswahl von Programmiersprachen zur Verfügung. Zur Treffung einer Entscheidung werden die beliebtesten Möglichkeiten verglichen (Siehe Anhang). Da es sich bei diesem Projekt um den Bereich des Internets der Dinge handelt, fällt die Wahl auf Python, da diese Sprache mehrere geeignete Bibliotheken für die Entwicklung bietet. Zusätzlich ist es relativ einfach mit ihr umzugehen.

9.5 Funktion

Die Arbeitsweise der Software ist durch einzelne Klassen geregelt, welche jeweils durch einen bestimmten Funktionsbereich eingegrenzt sind. Somit wird das Projekt übersichtlich gehalten und es ist einfacher, Fehler aufzufinden und zu beheben.

9.5.1 Start der Software (Main-Methode)

Der Ausgangspunkt der Software stellt die „main“-Klasse dar. Diese wird beim Start automatisch ausgeführt. Sie initialisiert die Haupt-Klassen des Projekts und startet die nötigen Threads und Tasks.



9.5.2 Globale Elemente

Komponenten, welche global von allen anderen Elementen verwendet werden, finden ihren Platz in der „Global-States“ Klasse. In ihr befinden sich die Manager der Verbindungen, die derzeitigen Einstellungen und auch diverse Funktionen zur Aktualisierung anderer Objekte.

Diagramm

Erklärungen

WriteLock: Dieser Wert wird auf Wahr gesetzt, wenn an den Einstellungen der Smart-Switches und Modis eine Änderung durchgeführt wird. Somit wird verhindert, dass auf eine Datei von mehreren Threads zugegriffen wird. Sobald diese Variable wieder vom derzeitigen Leser/Schreiber auf Falsch gesetzt wird, kann die nächste Zugriffoperation auf diese Datei erfolgen.

SendMessage: Diese Variable gibt an, ob eine neue Nachricht an den Client gesendet werden soll. Dies wird von der Klasse „Network-Manager“ abgefragt.

modeExecutorRunning: Dieser Wert gibt Aussage darüber, ob die Modis derzeitig abgeprüft werden.

switchList: In dieser Liste werden alle derzeitig aktiven Smart-Switch gespeichert.

modeMan: Dies ist die derzeitig laufende Instanz des „Mode-Managers“

init(): Diese Funktion ist zuständig, bei der Initialisierung des Objektes derzeitige Einstellungen anzuwenden und das Objekt aufzusetzen. Dabei wird beispielsweise die derzeitige Startzeit gesetzt.

UpdateSwitchState(): Dieses Methode wird verwendet, um den Switches einen neuen Wert zuzuweisen.

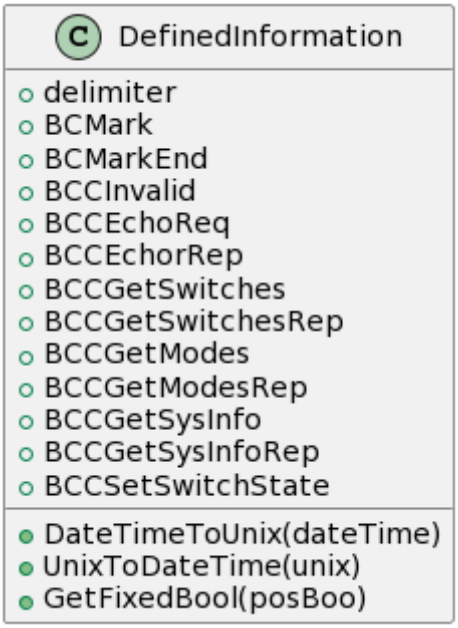
UpdateTime(): Hierbei können Zeit-Variablen von Switches gelesen und manipuliert werden.

AddTestSwitch(): Diese Klasse wird nur zur Testung und zur ersten Initialisierung der Klasse verwendet.

9.5.3 Vordefinierte Werte

Vor dem Start des Programmes müssen einige vordefinierte Werte festgelegt werden, welche von Protokollen, Codierungen und Standards definiert wurden. Hierbei kommt die Klasse „DefinedInformation“ zum Einsatz.

Diagramm



Erklärungen

BC: Hierbei handelt es sich um die Werte der Struktur der Kommandos. Diese werden einem Festwert zugewiesen, um eine Übereinstimmung zwischen den Verhaltensweisen bezüglich Strukturen bei Client und Bridge anzuwenden

DateTimeTOUnix(dateTime): Diese Funktion übernimmt als Parameter ein DateTime-Objekt und wandelt diese in das genormte Unix-Time Format um.

Dieses Format trägt als Startpunkt den 1. Januar 1970 und wird häufig in Computersystemen verwendet, um Zeitpunkte zu speichern. Es hat den entscheidenden Vorteil, dass Zeitpunkte somit als eine einzige Zahl dargestellt werden können.[34]

UnixToDateTime(unix): Dies ist das Gegenstück zur obigen Funktion. Hierbei werden UnixTime-Werte wieder in DateTime-Objekte umgewandelt.

9.5.4 Verwaltung der Einstellungen

Die derzeitigen Einstellungen werden an einem zentralen Ort abgespeichert. Diese müssen neben den Objekten und Variablen auch als feste Datei abgespeichert werden, um die Daten nach einem Neustart wiederzuverwenden. Dies sollen simple Textdateien sein, welche leicht eingelesen werden können.

[34] Vgl. Unixtime.org.

9.5.4.1 Vergleich der Lösungswege

Zur Abspeicherung von Daten bieten sich sehr viele Möglichkeiten. Hierbei sollen die Daten in reinem Text abgespeichert werden, um diese für einen Menschen leicht lesbar zu machen.

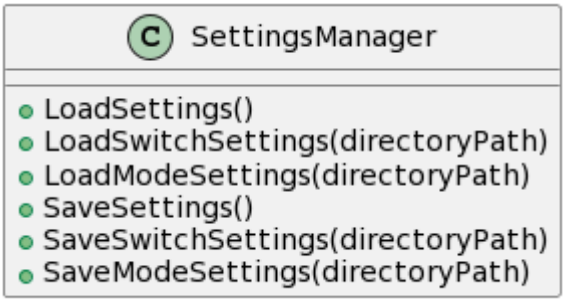
Um eine gute Lösung für das System zu finden, werden die verschiedenen Lösungsarten diskutiert (Siehe Anhang).

Schlussendlich wird das XML-Format verwendet, da es einfach zu lesen und bzw. zu schreiben ist, und ebenso auch von vielen Applikationen verwendet wird. Für Python gibt es beispielsweise eine populäre Bibliothek, welche einen großen Umfang an Funktionen bieten.

9.5.4.2 Realisierung

Das Speichern der Daten wird in einer eigener Klasse namens „SettingsManager“ geregelt. Mithilfe dieser werden die gespeicherten Modis und Smart-Switch aufbewahrt

Diagramm



Erklärungen

LoadSettings(): Diese Methode ruft einfach die beiden Funktionen beider Typen zur Ladung auf. Ihr Wert liegt also nur in der Vereinfachung des Ablaufs.

LoadSwitchSettings(directoryPath): Diese Funktion wird zum Laden der Schalterzustände verwendet. Als Parameter übernimmt diese den allgemeinen Pfad der Einstellungen. Falls dieser jedoch nicht existiert, wird eine neue Datei erstellt.

LoadModeSettings(directoryPath): Diese Methode führt die gleichen Arbeitsschritte durch, als die obige, jedoch auf die Modis bezogen.

SaveSettings(): Diese Funktion übernimmt die gleiche Rolle wie die „LoadSettings()“ Methode, jedoch wird hierbei gespeichert.

SaveSwitchSettings(directoryPath): Diese Funktion ist zum Speichern der Smart-Switch zuständig. Als Parameter übernimmt diese den allgemeinen Pfad der Einstellungen. Falls dieser jedoch nicht existiert, wird eine neue Datei erstellt., ansonsten überschrieben.

SaveModeSettings (directoryPath): Diese Methode führt die gleichen Arbeitsschritte durch, als die obige, jedoch auf die Modis bezogen.

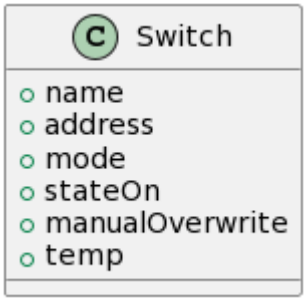
9.5.5 Schalterbezogene Elemente

Alle Elemente, welche sich auf den Smart-Switch beziehen werden als („Switch“-)Objekt dargestellt. Das gilt auch für den Schalter an sich.

9.5.6 Schalter

Das „Switch“-Objekt verfügt über alle benötigten Variablen eines Schalters, jedoch besitzt diese Klasse abgesehen des Konstruktors keine Funktionen.

Diagramm



Erklärungen

Name: Dies ist der Anzeigename des Schalters.

Address: Dies ist die Adresse des Schalters mit einer Größe von 8 Bit.

stateOn: Der physikalische Zustand des Schalters

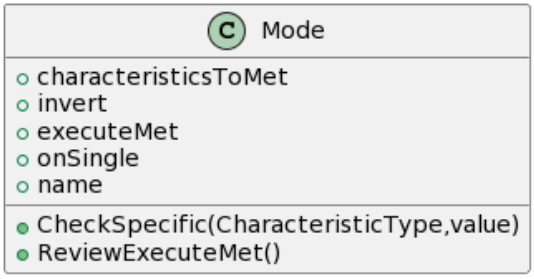
init(address) (): Der Konstruktor legt die Adresse fest und generiert einen zufälligen Namen. Dies geschieht über eine eingebundene Klasse namens „Haikunator“. [35]Dabei wird ein Name ausgewählt, welcher aus einem Adjektiv und einem Nomen besteht. Mit einem Bindestrich werden die beiden Wörter zusammengeführt. Durch einen solchen lesbaren Namen, kann der Benutzer den Schalter besser erkennen, als beispielsweise durch die Adresse.

9.5.7 Modus

Für die einzelnen Modis gibt es wie bei den Schaltern eine eigene Klasse, da mit dem Verfahren der objektorientierten Programmierung (Siehe Anhang) effizienter Gearbeitet werden kann.

Diagramm

[35] Vgl. Atrox (2023).



Erklärungen

characteristicsToMet: Eine Liste bestehend aus allen Charakteristiken des Modis.

Invert: Gibt an, ob „executeMet“ invertiert werden soll

executeMet: Diese Variable gibt an, ob die Bedingungen des Modis erfüllt wurde.

onSingle: Gibt an, ob ein Modus schon bei Erfüllung einer Charakteristik ausführbar ist.

Name: Der nutzerdefinierte Name

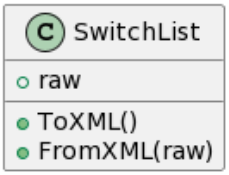
CheckSpecific(CharacteristicType,value): Überprüft, ob eine Charakteristik einen bestimmten Wert erfüllt hat.

ReviewExecuteMet(): Prüft alle einzelnen Charakteristiken ab und gibt an, ob der Modus zur Ausführung bereit ist.

9.5.8 Aufbewahrung der Schalter

Um die verschiedenen Smart-Switches zu speichern, werden eigene Klassen für diese Listen erstellt

Diagramm



Erklärungen

Raw: Ein Objekt einer Liste. Hier werden die einzelnen Smart-Switches gespeichert

ToXML(): Diese Funktion wandelt die Liste mittels der XML Bibliothek von Etree in einen string um und gibt diesen zurück. Ein Beispiel für einen solchen XML-String könnte so aussehen:[36]

[36] Vgl. xml.etree.ElementTree — The ElementTree XML API.

```
<switchList>
    <switch name="switchy"
            address="2"
            stateOn="True"
            lastContacted="1672677319.0"
            mode="Default" manualOverwrite="False" />
</switchList>
```

FromXML(raw): Diese Methode fungiert als Gegenstück der Vorigen. Sie entnimmt als Eingangsparameter einen XML-String und überschreibt mit dessen Inhalt ihre eigenen Werte

9.5.9 Verwaltung der Modis

Für die Durchführung der Modis ist die Datei „ModeExecutor“ zuständig. Dabei werden die einzelnen Modis mittels eines Threads überprüft. Dieser kann mittels der Funktion „Start()“ begonnen werden.

Der Thread wartet anfangs, bis der Schreibschutz gelöscht wurde, damit keine Modis überprüft werden, während diese sich gerade aktualisieren. Anschließend wird jeder gespeicherte Smart-Switch durchgegangen und der gespeicherte Modus überprüft. Sollte dieser die Voraussetzungen für eine Ausführung erfüllen, wird dies gemacht. Dabei wird auch die Invertierung berücksichtigt. Die Benachrichtigung des jeweiligen Smart-Switches erfolgt dabei wieder über den Bluetooth Abteil. (Siehe „BluetoothManager“)

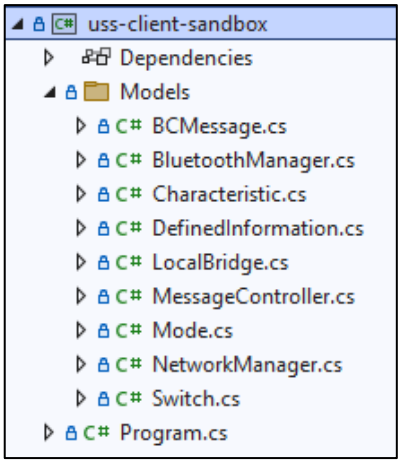
10 CLIENT

10.1 Grundidee

Um die Einstellungen der Bridge zu manipulieren, braucht es eine Applikation, welche auf einem weiteren Gerät installiert wurde. Diese Software bildet die Schnittstelle zum Benutzer selbst. Die verwendeten Komponenten sind im Anhang zu finden.

10.2 Konsolen-Applikation

Vor der Entwicklung einer grafischen Oberfläche, wird ein Konsolenprogramm erstellt, welches alle nötigen Funktionen zur Manipulation und Austausch von Daten enthält. Dies ermöglicht eine einfache Testung. Anbei eine Übersicht der Projektstruktur

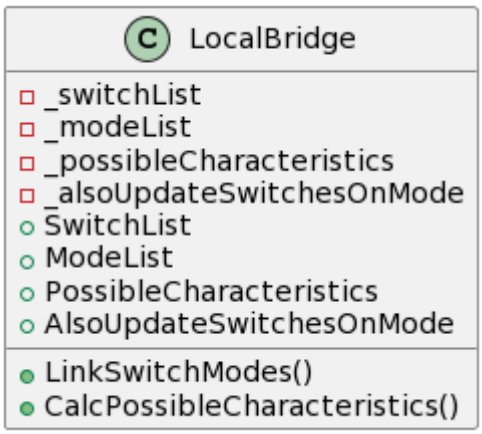


Die Funktion des Datenaustausches wird im Kapitel 13 Kommunikation behandelt.

10.2.1 Bridge aus Client-Sicht

Eine Bridge wird von dem Client als ein Objekt gesehen, welche als die Klasse „LocalBridge“ definiert wird. Diese beinhaltet alle Daten, welche die Applikation braucht, um mit dieser zu kommunizieren und Einstellungen zu verwalten.

Diagramm



switchList: Dieses Objekt beinhaltet alle derzeit verfügbaren Schalter.

modeList: Dieses Element enthält alle auf der Bridge gespeicherten Modis

possibleCharacteristics: Dies ist wiederum eine Liste, welche alle möglichen Sorten von Characteristiken enthält, welche von den einzelnen Instanzen als Typ angenommen werden können

alsoUpdateSwitchesOnMode: Falls ein Modus gelöscht wird, welcher von einem Schalter derzeitig verwendet wird, kann dieser bei Aktivierung dieser Variable durch den Standart-Modus ersetzt werden.

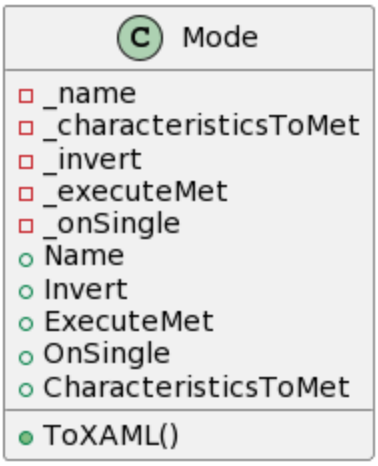
CalcPossibleCharacteristics(): Diese Methode berechnet die Anzahl der möglichen Charakteristiken.

Weitere: Die weiteren Variablen und Methoden sind zuständig, um die obigen privaten Variablen für den Zugriff von außen durchzugeben. Dabei können auch die Zugriffsrechte beschränkt werden.

10.2.2 Mode als Objekt

Wie auch bei der Bridge wird hier ein Mode als eigenes Objekt angesehen. Die Variablen und Methoden bleiben bis auf die XAML-Formatierung identisch zur Python-Implementierung der Bridge. Genaueres kann im entsprechenden Kapitel gefunden werden.

Diagramm



Erklärungen

ToXAML(): Formatiert den Modus als XAML-Text und gibt diesen zurück.

10.2.3 Schalter als Objekt

Wie auch beim Modus werden hier nur die Ergänzungen der Klasse genauer erklärt. Mehr zur Implementierung kann im Kapitel 11 Bridge gefunden werden

Diagramm

Erklärungen

VirtMode: Da der Name des Modus als Text gespeichert wird, muss noch eine Verbindung zum richtigen Objekt des Modus erzeugt werden. Dies wird durch dieses dargestellt. Bei einer Änderung wird auch der Text-Name umgestellt.

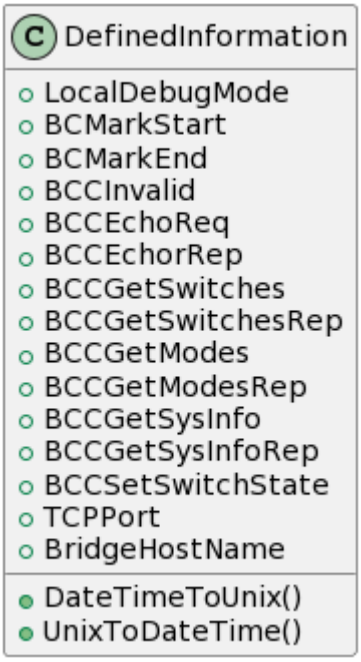
SetStateCommand: Kommandos werden genauer im Kapitel 12.5 Client V2 beschrieben. Hierbei handelt es sich um denjenigen, welcher eine Änderung des Schalters hervorruft.

Weiteres: Wie auch zuvor dienen die weiteren Variablen als Zugriffs-Operatoren

10.2.4 Vordefinierte Einstellungen

Neben den Definitionen der Kommandos und den Unix-Umrechnungen befinden sich auch weitere Informationen zur Kommunikation zwischen Client und Bridge in der Defined-Information Klasse.

Diagramm



Erklärungen

LocalDebugMode: Hiermit kann der Debug-Modus eingeschalten werden. Bei der Aktivierung verbindet sich der Client mit einer Loop-Back-Adresse, um sich mit der simulierten Bridge zu verbinden

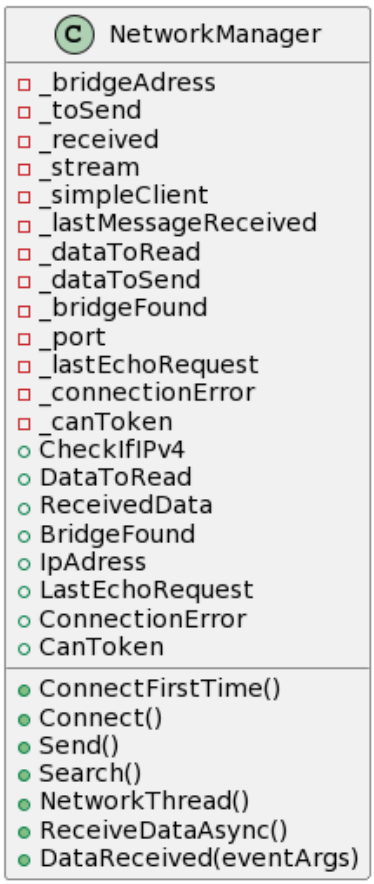
TCPPort: Diese Konstante enthält die Port-Nummer, unter welcher der TCP-Server der Bridge läuft.

BridgeHostName: Dieser Bestandteil der Klasse enthält den Hostnamen der Bridge

10.2.5 Management der Kommunikation

Dies wird von der Klasse „NetworkManager“ geregelt, welche im genauen im Kapitel 13 Kommunikation behandelt wird. Im Folgenden sind Ergänzungen zu finden. Zum Datenaustausch bedarf es einem TCP-Client, welcher durch das Nuget-Paket „SimpleTCP“ bereitgestellt wird.

Diagramm



Erklärungen

bridgeAdress: Enthält die derzeitige IP-v4 Adresse der Bridge

toSend: Gibt an, ob eine neue Nachricht gesendet werden soll

received: Gibt an, ob eine neue Nachricht empfangen wurde

simpleClient: Objekt des TCP-Clients

lastMessageReceived: Enthält die letzte empfangene Nachricht

dataToRead: Enthält die neuen Bytes, welche gelesen werden sollen

dataToSend: Enthält die neuen Bytes, welche als nächstes versendet werden sollen

bridgeFound: Ein Boolean-Wert, welcher angibt, ob die Bridge gefunden wurde

connectionError: Gibt an, ob beim Verbindungsaufbau ein Fehler aufgetreten ist

canToken: Enthält einen Token, welcher bei Aktivierung alle Threads beendet.

Weiteres: Alle weiteren Funktionen und Variablen, welche nicht im Kapitel 13 Kommunikation behandelt werden, dienen als Zugriffs-Operatoren für die privaten Variablen.

10.2.6 Speichern der Modis & Smart-Switches

Auch auf der Client-Seite müssen diese einzelnen Objekte in einer Liste abgelegt werden Hierzu werden auch eigene Typen als Klasse definiert, diese erben von der Klasse „ObservableCollection“. Dieser Datentyp benachrichtigt zusätzlich auch das UI bei einer Aktualisierung.

Unter einer Vererbung versteht man, dass eine Klasse (Kindklasse oder Unterklasse) die Eigenschaften und Verhaltensweisen einer anderen Klasse (Basisklasse oder Überklasse) übernehmen kann. Dadurch kann eine neue Klasse von einer bestehenden Klasse abgeleitet werden, ohne dass der Code der Basisklasse erneut geschrieben werden muss.

In C# wird eine Vererbung durch ein „:“ neben der Definition der Klasse dargestellt.

```
6 references
public class ModeList : ObservableCollection<Mode>
{
```

Diagramme

Der grundsätzliche Aufbau der Listen ist abgesehen vom Typ. Sie beinhalten beide auch Funktionen zur Serialisierung als XML

Erklärungen

GetStringList(): Diese Funktion gibt alle Namen der Modis als weitere Liste zurück. Dies ist für die Anzeige im UI notwendig.

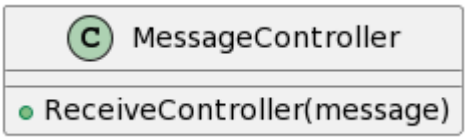
10.2.7 Umgang der empfangenen Nachricht

Nach dem Empfang einer Nachricht muss entschieden werden, welche Aktion der Client als nächstes durchführen soll. Dies wird von der Funktion „ReceiveController“ in der statischen Klasse „MessageController“ übernommen.

Eine statische Funktion in C# ist eine Funktion, die zu einer Klasse und nicht zu einem bestimmten Objekt gehört. Diese kann direkt über den Klassennamen aufgerufen werden, ohne dass eine Instanz der Klasse erstellt werden muss. Um ein Element als statisch zu markieren, wird das Wort „static“ nach dem Zugriffs-Operator eingefügt.[37]

```
1 reference
public static class MessageController
{
    1 reference
    public static void ReceiveController(BCMessage message)
    {
```

Diagramm



Erklärung

ReceiveController(message): In dieser Funktion befindet sich ein switch-case, welches je nach Art der Nachricht eine andere Aktion durchführt.

Ein switch-case-Statement in C# ist ein Kontrollstruktur, die verwendet wird, um aus einer Reihe von möglichen Werten einen bestimmten Abschnitt des Codes auszuführen. Es enthält dabei eine Ausdrucksangabe, die einen bestimmten Wert berechnet, und eine Reihe von case-Anweisungen, die den Wert mit den unterschiedlichen Fällen vergleichen. Wenn ein Fall gefunden wird, der dem Wert entspricht, werden die Anweisungen innerhalb dieses Falls ausgeführt. Wenn kein Fall gefunden wird, wird die default-Anweisung ausgeführt, falls vorhanden. [38]Anbei ein Beispiel, welches ausgeführt wird, sobald es sich bei der eimpfangenen Nachricht um eine Liste von Modis handelt.

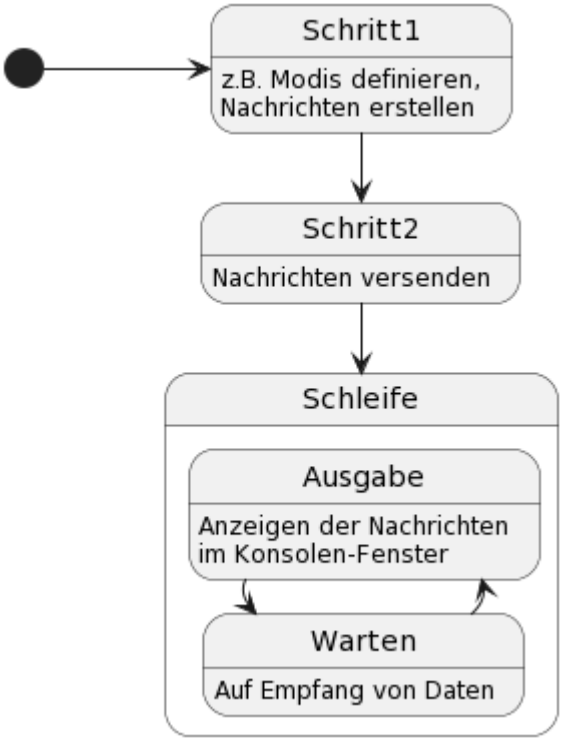
```
case BCCommand.GetModesRep:
    LocalBridge.ModeList.FromXML(message.DataString);
    Console.WriteLine("[Bridge:] (ModeList) " + LocalBridge.ModeList.Count);

    LocalBridge.LinkSwitchModes();

    break;
```

Zuerst wird die neue Liste übernommen und abgespeichert, dabei wird eine Meldung im Konsolen-Fenster ausgegeben. Anschließend werden die neuen Modis in den Schaltern aktualisiert.

[37] Vgl. Was ist eine statische Funktion in C? | Referenz (2021).
[38] Vgl. BillWagner (2023b).

10.2.8 Durchführung von Testungen

Um Verhaltensweisen mittels des Konsolenprogramms zu testen, werden die entsprechenden Arbeitsschritte in die „main“-Methode der Klasse „Programm“ eingefügt. Bei der Ausführung des Programmes werden die Nachrichten versendet und auf eine Antwort der Bridge gewartet.



Beispiel

Im folgenden Programm werden zwei Modis erstellt, welche anschließend zur Bridge gesendet werden.

```
// Modes erstellen
var md = new Mode("einModus", new Characteristic(CharacteristicType.Temperature, 13, false));
md.OnSingle = true;
var md1 = new Mode("zweiterModus", new Characteristic(CharacteristicType.Date, 33, false));

// Modes zur ModeList hinzufügen
LocalBridge.ModeList.Add(md);
LocalBridge.ModeList.Add(md1);

var mdl = new BCMessage(BCCommand.GetModesRep, LocalBridge.ModeList.ToXML(), 0);
    // Nachricht erstellen

NetworkManager.Send(mdl);   // Modes senden
```

Das Programm wird aus Testgründen im Debug-Mode ausgeführt. An der Ausgabe der Konsolen lässt sich erkennen, dass der Test erfolgreich war und die Daten übertragen wurden.

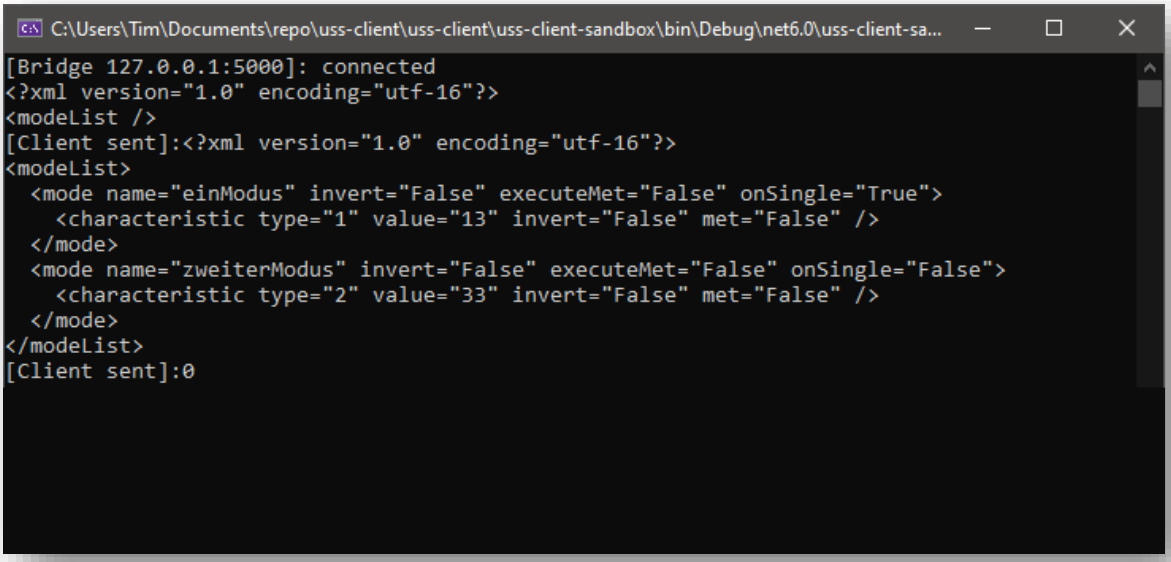
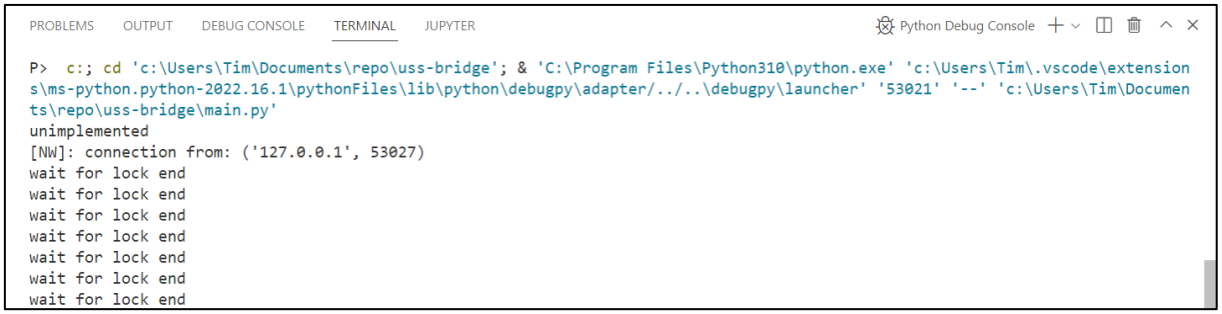


Abbildung 41: CLI-Client

Hier im Client-Log kann nachvollzogen werden, dass sich die Bridge zuerst mit dem Client verbindet, anschließend die XML-Daten ausgegeben und schließend auch versendet werden.



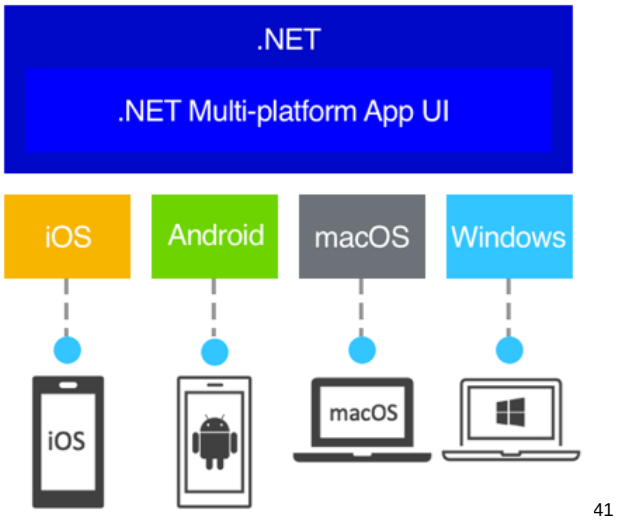
Im Konsolen-Fenster der Bridge kann dies auch verifiziert werden. Der Verbindungsaufbau ist zu erkennen, sowohl als auch der Schreibprozess der Modis.

10.3 Client-Applikation (Version 1)

10.3.1 MAUI-Framework

Für die Erstellung von einer grafischen Benutzeroberfläche benötigt es ein UI-Framework. Hierbei fiel die Entscheidung auf .NET MAUI (Multi-platform App UI). Dies ist ein Open-Source-Framework für die Erstellung von plattformübergreifenden Anwendungen, die auf iOS, Android, macOS, Windows und anderen Plattformen laufen können. Es ist eine Erweiterung von Xamarin und wurde von auch Microsoft entwickelt. Im Kern von .NET MAUI stehen XAML-basierte Views, die es Entwicklern ermöglichen, plattformübergreifende Benutzeroberflächen

zu erstellen. Es enthält auch eine umfangreiche Sammlung von Controls und Layouts, die genau für eine solche Entwicklung optimiert sind.[39][40]



[41]

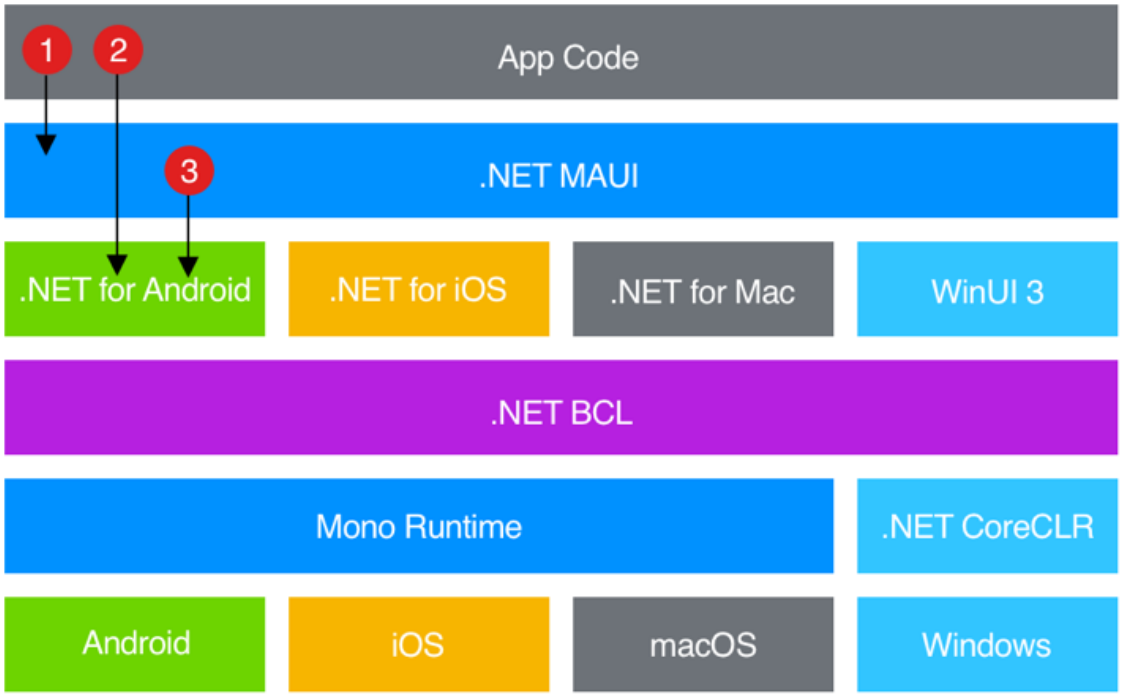
Abbildung 42: .NET MAUI Layers

Dieses System ermöglicht es, eine einheitliche API für die Entwicklung plattformübergreifender Anwendungen zu verwenden, indem es die Android-, iOS-, macOS- und Windows-APIs in einer API vereint. Auf diese Weise kann der Code einmal geschrieben und auf verschiedenen Plattformen ausgeführt werden, während gleichzeitig ein umfassender Zugriff auf alle Funktionen der jeweiligen Plattformen gewährleistet wird.

Im Rahmen von .NET 6 werden verschiedene plattformspezifische Frameworks zur Erstellung von Apps bereitgestellt, darunter .NET für Android, .NET für iOS, .NET für macOS und Windows UI 3 (WinUI 3). Diese Frameworks greifen auf die gleiche .NET Base Class Library (BCL) zu, die die Details der zugrunde liegenden Plattform abstrahiert.[42]

Obwohl die BCL es ermöglicht, eine gemeinsame Geschäftslogik für plattformübergreifende Anwendungen zu teilen, bieten die verschiedenen Plattformen unterschiedliche Modelle für die Definition von Benutzeroberflächen und für die Interaktion zwischen den Elementen der Benutzeroberfläche. Diese kann für jede Plattform separat mit dem entsprechenden plattformspezifischen Framework erstellt werden. Dies erfordert jedoch die Verwaltung einer Codebasis für jede Gerätefamilie separat.[43]

[39] Vgl. Was ist .Net? – Erläuterung zu Dotnet – AWS.
[40] Vgl. davidbritch (2023a).
[41] Abbildung: davidbritch (2023b).
[42] Vgl. .NET MAUI: Paradiesische App-Entwicklung.
[43] Vgl. davidbritch (2023a).



[44]

Abbildung 43: .NET MAUI Layers (Specific)

10.3.2 Designmuster

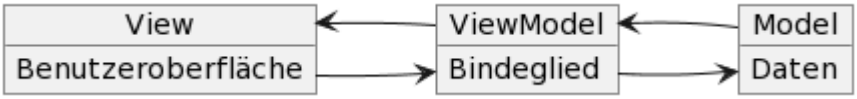
Um die Applikation möglichst effizient und übersichtlich zu gestalten, bedarf es an einem Design-Muster, nach welchem die verschiedenen Komponenten ausgelegt sind. Es gibt viele verschiedene Design-Muster, die bei der Entwicklung von Software eingesetzt werden können, weshalb eine Entscheidung getroffen werden muss.[45]

10.3.2.1 Vergleich

Model-View-ViewModel (MVVM) ist das beliebteste Muster, welches am öftesten eingesetzt wird. Es teilt die Anwendung in drei Komponenten: Das Model, die View und das ViewModel. Das Model enthält die eigentlichen Daten, die von der Anwendung verwendet werden, während die View die Benutzeroberfläche darstellt, die der Benutzer sieht und mit der er interagiert. [46]Das ViewModel fungiert als Vermittler zwischen dem Model und der View, indem es die Daten aus dem Model abruft und in eine Form transformiert, die von der View angezeigt werden kann. Es ermöglicht auch, dass Änderungen an der Benutzeroberfläche vom Benutzer erfasst werden und diese Änderungen an das Model weitergeleitet werden.[47]

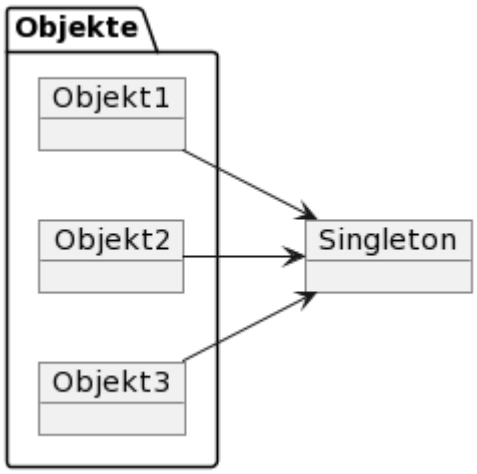
[44] Abbildung: davidbritch (2023b).
[45] Vgl. Design Patterns – schneller und sicherer programmieren (2020).
[46] Vgl. Augsten, S. (2022).
[47] Vgl. michaelstonis (2022).

Durch die Verwendung von MVVM wird die Trennung von Geschäftslogik und Benutzeroberfläche erleichtert. Der Code, der für die View und das ViewModel geschrieben wird, ist in der Regel unabhängig von der tatsächlichen Implementierung der Benutzeroberfläche und des Models. Dies erleichtert die Tests und Wartung des Codes, insbesondere in komplexen Anwendungen.



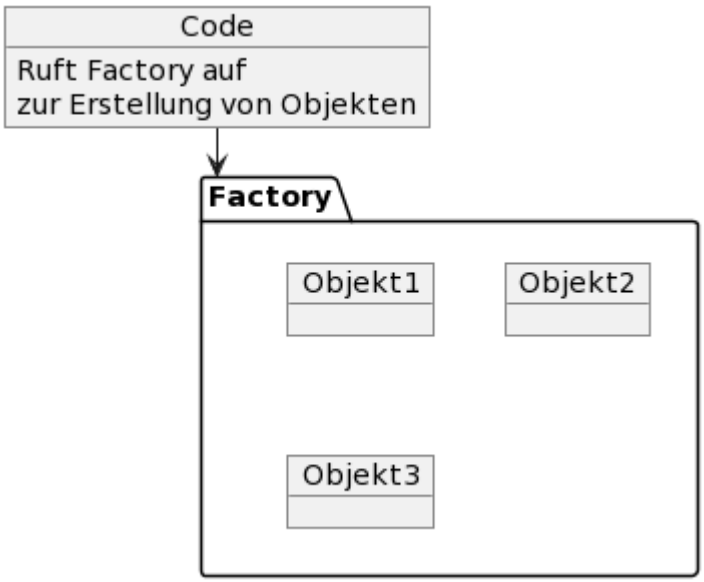
Model-View-Controller (MVC) ist ein Designmuster, das ähnlich wie MVVM die Anwendung in drei Komponenten aufteilt, jedoch ohne das ViewModel. Das Model enthält die Daten, die View enthält die Benutzeroberfläche und der Controller dient als Vermittler zwischen dem Model und der View.[48]

Beim Singleton-Muster wird sichergestellt, dass es nur eine einzige Instanz einer Klasse gibt und dass diese global verfügbar ist. Dies ist nützlich, wenn nur eine einzige Instanz einer bestimmten Klasse benötigt wird, z.B. für den Zugriff auf eine gemeinsame Ressource.[49]



Auch ist es möglich, mittels des Factory-Systems separate Klassen für die Erstellung von Objekten zu verwenden. Dadurch wird der Code, der für die Erstellung der Objekte zuständig ist, von dem Code getrennt, der die Objekte tatsächlich verwendet.[50]

[48] Vgl. StephenWalther (2022); The Model View Controller Pattern – MVC Architecture and Frameworks Explained (2021).
[49] Vgl. Singleton Design Pattern: Das Singleton-Entwurfsmuster kurz erklärt - IONOS.
[50] Vgl. Factory Pattern: Alle Informationen zum Factory Method Pattern (2020).



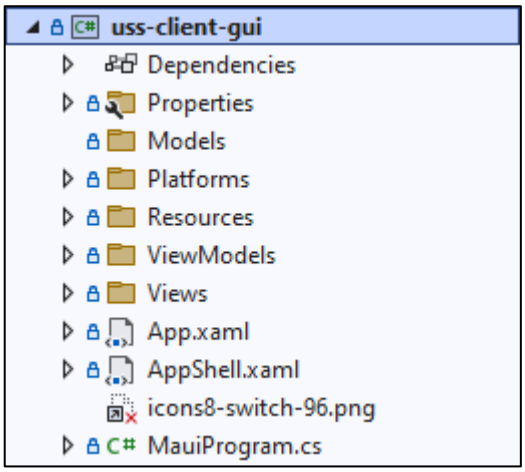
10.3.2.2 Entscheidung

Schlussendlich fällt die Entscheidung auf das Model-View-ViewModel Design-Muster, da es von der Industrie am meisten verwendet wird und auch mehrere Bibliotheken bereitstehen, um den Umgang mit MVVM zu vereinfachen. In diesem Projekt wird das Community-Toolkit verwendet. Dies ist eine Open-Source-Bibliothek, die von der .NET-Community entwickelt wird und neben der MVVM-Automatisierung eine Sammlung von Steuerelementen, Erweiterungen und Dienstprogrammen für die Entwicklung von Cross-Platform-Apps mit .NET MAUI bietet. Das Toolkit wurde entwickelt, um Entwicklern dabei zu helfen, schneller und effizienter plattformübergreifende Anwendungen zu erstellen, indem es häufig verwendete Funktionen und Benutzeroberflächenelemente bereitstellt. Das Toolkit enthält dazu auch eine Vielzahl von Komponenten, darunter Steuerelemente für die Benutzeroberfläche (z.B. Diagramme, Schaltflächen, Listen und Schieberegler), Erweiterungen für Datenbindung, Navigation, Animationen und vieles mehr.[51]

10.3.3 Aufbau

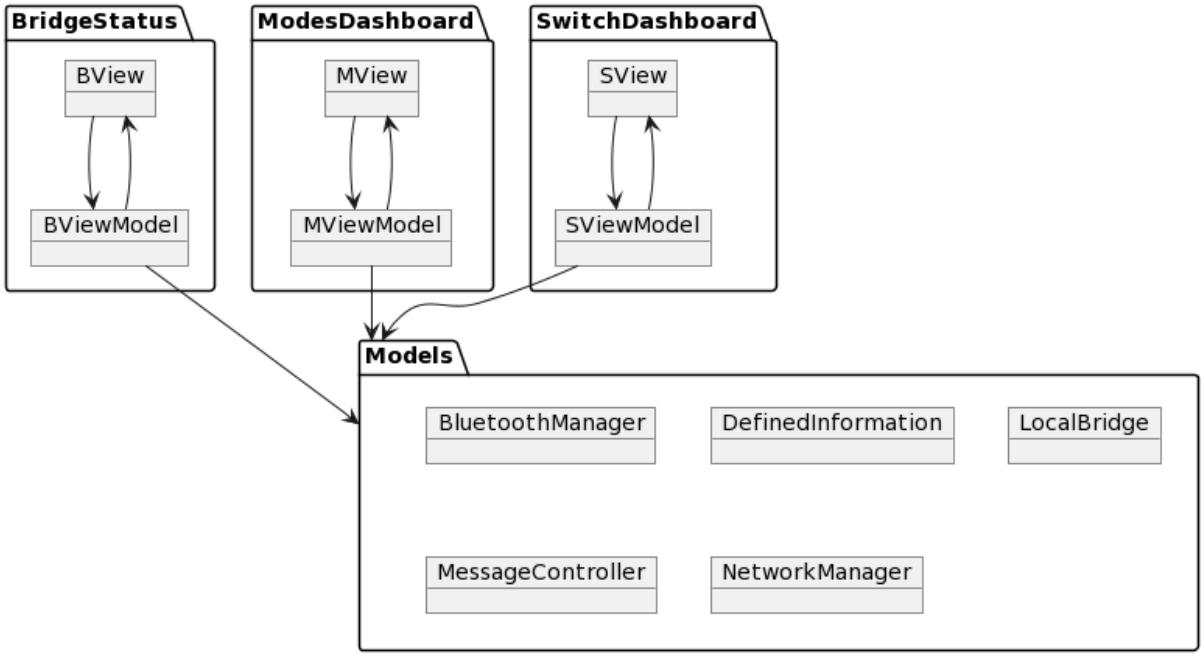
Die einzelnen Dateien des Projektes werden in einer hierarchischen Ordnerstruktur angelegt. Dabei sind diese auch nach Funktionen getrennt.

[51] Vgl. bijington (2022).



Neben den Ordnern des MVVM-Musters finden wird auch den „Platforms“-Ordner, welcher die Konfigurationen für die einzelnen Plattformen beinhaltet.

Die komplette Applikation ist in 3 Abschnitte gegliedert: BridgeStatus, ModesDashboard und SwitchDashboard. Diese Teile erhalten alle ihe eigene View, sowohl als auch ein ViewModel. Letztere greifen auf die Models der Applikation zu.



10.3.4 Ablauf nach Ausführung

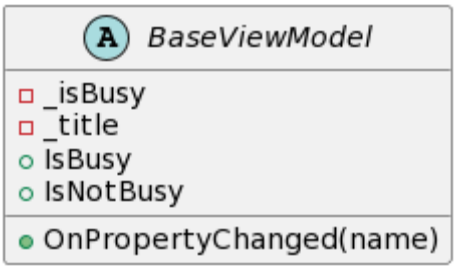
Während dem Startvorgang der Applikation wird versucht, eine Verbindung mit der Bridge herzustellen. Dies geschieht in der Klasse „App“. Sollte der Verbindungsversuch erfolgreich sein, werden alle Tabs im UI freigeschaltet.

```
public App()
{
    NetworkManager.ConnectFirstTime();  // Verbindungsversuch
    // Tabs freischalten
    OtherTabsVisible = !NetworkManager.ConnectionError
    InitializeComponent();  // UI starten
    MainPage = new AppShell();
}
```

10.3.5 Implementierung Model-View-ViewModel

10.3.5.1 BaseViewModel

Um das MVVM-Muster umzusetzen, wird eine abstrakte Klasse namens „BaseViewModel“ erstellt, von welcher alle ViewModels erben.



Diese abstrakte Klasse beinhaltet mehrere Variablen, welche mit der „OnPropertyChanged“ Methode versehen sind. Diese dient dazu, Änderungen der Datenfelder der View weiterzugeben.

```
private string _title;  // private Variable

public string Title // Datenfeld
{
    get { return Title; }
    set
    {
        _title = value;
        OnPropertyChanged("Title"); // Methode zum Aufruf des PropertyChanged Events
    }
}

public event PropertyChangedEventHandler PropertyChanged;   // Event definition


protected virtual void OnPropertyChanged(string propertyName)   // Handler
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

Mit Hilfe des Community-Toolkits kann dieser Code-Abschnitt durch zwei Zeilen vereinfacht werden:

```
[ObservableProperty]
private string title;
```

10.3.5.2 Anzeige der Views

Die Elemente, welche im Hauptfenster dargestellt werden, können in der „AppShell“ definiert werden. Diese XAML-Datei wird so modifiziert, dass zum derzeitigem ViewModel immer das richtige View angezeigt wird.

```
<TabBar>
    <ShellContent Title="Bridges"
                  ContentTemplate="{DataTemplate local:BridgeDashbaordView}" />
    <ShellContent Title="Switches"
                  ContentTemplate="{DataTemplate views:SwitchDashboardView}"
                   IsEnabled="{Binding OtherTabsVisible}"
                   IsVisible="{Binding OtherTabsBisible}"/>
    <ShellContent Title="Modes"
                  ContentTemplate="{DataTemplate views:ModesDashboardView}"
                   IsEnabled="{Binding OtherTabsVisible}"
                   IsVisible="{Binding OtherTabsBisible}"/>
</TabBar>
```

10.3.6 Informationen zur Bridge

Die BridgeStatusView hat zur Aufgabe, eine Möglichkeit zur Verbindung mit der Bridge anzubieten, als auch Informationen diesbezüglich anzuzeigen. Falls ein Verbindungsversuch beim Start der Anwendung missglückt ist, so kann ein erneuter Verbindungsversuch mit dem Knopf „Try Again“ gestartet werden. Erfolgt dieser, wird die IP-Adresse der Bridge angezeigt.

Das User Interface (View) ist unvollständig, da dieses vor dem UI-Framework Wechsel nicht vollendet wurde.

Abbildung 44: Bridge-Dashboard (V1)

Die „Connect“-Funktion wird durch einen asynchronen Task geregelt. In C# bezieht sich ein asynchroner Task auf einen Arbeitsschritt, der auf einem separaten Thread ausgeführt wird und parallel zur Hauptanwendung ausgeführt wird. Im Gegensatz zu synchronen Tasks, bei denen die Anwendung blockiert wird, bis der Task abgeschlossen ist, ermöglichen asynchrone Tasks der Anwendung, während des Ausführungsprozesses weiterhin reaktiv zu bleiben. Sie werden üblicherweise mit dem async/await-Schlüsselwort in C# implementiert. Das async-Schlüsselwort wird verwendet, um eine Methode als asynchron zu markieren, während das

await-Schlüsselwort verwendet wird, um die Steuerung an die Hauptanwendung zurückzugeben, während der asynchrone Task im Hintergrund ausgeführt wird.[52]

Die Verwendung von asynchronen Tasks kann dazu beitragen, die Leistung und Reaktionsfähigkeit der Anwendung zu verbessern, insbesondere bei Anwendungen mit hohem Datenvolumen oder wenn auf Ressourcen zugegriffen wird, die längere Zeit benötigen, um eine Antwort zu liefern, wie z.B. bei Netzwerkanforderungen oder Datenbankabfragen.[53]

Auch hierbei wird die Implementation durch das Community-Toolkit erleichtert, da eine asynchrone Funktion durch den Text „[RelayCommand]“ zu einem Command wird.[54]

Bei der Ausführung des Commands wird die „isBusy“ Variable auf Wahr gesetzt. Dies hat den Vorteil, dass nicht zu viele asynchrone Operationen auf einmal abgearbeitet werden, da alle diese auf die Freigabe dieser Variable warten.

Auch wird alles innerhalb des Commands in einem „Try-Catch“-Block durchgeführt. Dies ist eine Struktur, die verwendet wird, um Fehler abzufangen, die während der Ausführung des Codes auftreten können. Der Try-Block enthält den Code, der möglicherweise einen Fehler auslösen kann, während der Catch-Block den Code enthält, der ausgeführt wird, wenn ein Fehler auftritt.[55]

```
[RelayCommand]
2 references
async Task TryConnectAsync()
{
    if (IsBusy) return;

    try
    {
        IsBusy = true;
        NetworkManager.Connect();

        if (!NetworkManager.ConnectionError)
        {
            NetworkManager.Send(new BCMessage(BCCommand.GetSwitches, "null", 0));
            NetworkManager.Send(new BCMessage(BCCommand.GetModes, "null", 0));
        }
    }
    catch (Exception) {  }
    finally
    {
        IsBusy = false;
        OnPropertyChanged(nameof(NoConnectionVisible));
        OnPropertyChanged(nameof(ConnectionVisible));
    }
}
```

[52] Vgl. Asynchronous programming - C# | Microsoft Learn.
[53] Vgl. dotnet-bot.
[54] Vgl. bijington (2022).
[55] Vgl. BillWagner (2023a).

Der „finally“-Block wird nach dem Try-Catch ausgeführt und benachrichtigt über das PropertyChanged-Event das UI.

10.3.7 Einstellungen der Schalter

Um verschiedene Einstellungen an den Schaltern vorzunehmen, bedarf es einer eigenen Oberfläche. Auf dieser sollen alle Schalter aufgelistet werden. Dies wird im Abteil „SwitchDashboard“ geregelt. Für jedes Schalterelement in der Liste stehen dabei mehrere Elemente zur Modifikation der Einstellungen.

Abbildung 45: Devices-Overview

Für jeden Schalter ist die ID, der Name, der Modus und der derzeitige Zustand zu sehen, welche nach Belieben verändert werden können. Wenn die Einstellungen übernommen werden sollen, wird der Button „Save Settings“ betätigt, welcher wie schon zuvor einen asynchronen Task ausführt.

10.3.8 Modifikation der Modis

Um die Funktionen der Schalter zu automatisieren, bedarf es dem Abteil „ModeDashboard“. In dieser Ansicht können neue Modis hinzugefügt und gelöscht werden, sowie auch deren Einstellungen manipuliert werden. Dabei werden wie auch beim SwitchDashboard die Charakteristiken der Reihe nach aufgelistet.

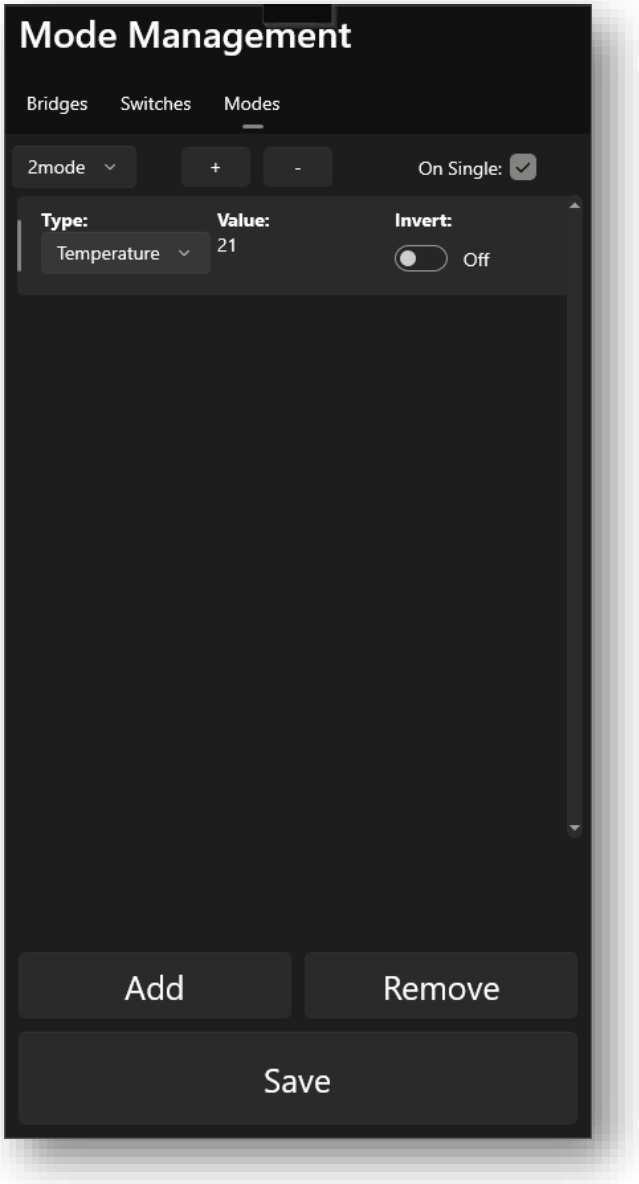


Abbildung 46: Mode-Management (V1)

Am unteren Bildschirmrand können wieder die Einstellungen gespeichert werden. Zudem befinden sich daneben zwei weitere Buttons, welche dafür da sind, Charakteristiken zu löschen oder hinzuzufügen.

10.3.9 Probleme

Während der Erstellung der MAUI-Applikation treten immer mehr Fehler des MAUI-Systems zum Vorschein. Dies liegt vor allem daran, dass dieses System sich derzeit in einer „Preview"-Phase befindet und noch nicht vollständig für die Produktion veröffentlicht wurde.[56]

Der große Vorteil, welcher MAUI bieten soll, ist die einfache plattformübergreifende Entwicklung. Dies konnte jedoch nicht zur Vollständigkeit ausgenutzt werden, da der Androide-Debugger unter den verwendeten Betriebssystemen derzeitig nicht verwendbar ist.

Auch sind viele wichtige Funktionen noch nicht implementiert, wie zum Beispiel mehrere UI-Controls oder Events. Dazu befinden sich sehr viele Fehler im MAUI-Framework, welche das Arbeiten so erschwert, dass das die Applikation nur noch schwer umsetzbar ist.

Aufgrund dieser Komplikationen wird das UI-Framework auf WPF gewechselt.

10.4 Client-Applikation (Version 2)

10.4.1 WPF-Framework

Windows Presentation Foundation (WPF) ist ein von Microsoft entwickeltes Framework zur Erstellung von grafischen Benutzeroberflächen (GUI) in Windows-Anwendungen. Es wurde erstmals mit dem .NET-Framework 3.0 veröffentlicht und ist seitdem ein wichtiger Bestandteil des .NET-Frameworks.

WPF ermöglicht die Erstellung von Desktop-Anwendungen mit modernen und interaktiven Benutzeroberflächen, die auf verschiedenen Windows-Plattformen laufen. Es bietet eine XAML (Extensible Application Markup Language) genannte deklarative Sprache, mit der Benutzeroberflächen einfach und schnell entworfen werden können.[57] Darüber hinaus unterstützt WPF auch eine Reihe von Grafik- und Multimedia-Funktionen, einschließlich 2D- und 3D-Grafiken, Animationen und Video.[58]

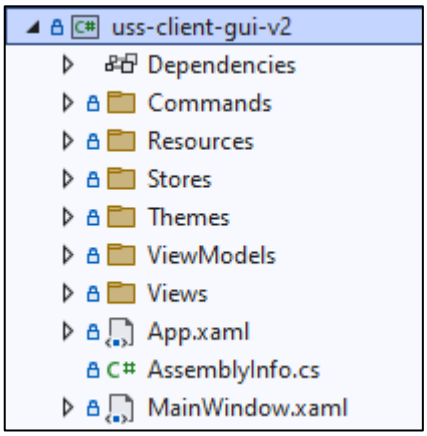
Ein weiterer Vorteil von WPF ist die Trennung von Design und Code. Mit dem Framework können Entwickler die Design-Elemente und den Code für die Anwendungslogik getrennt voneinander erstellen, wodurch die Entwicklung und Wartung von Anwendungen einfacher und effizienter wird.[59]

Die Wahl fiel nach dem Framework-Wechsel auf WPF, da dieses schon lange in der Industrie verwendet wird und bereit für ein Production-Environment ist.

[56] Vgl. Ramel, B. D./09/29/2022.
[57] Vgl. jwmsft (2022).
[58] Vgl. Windows Presentation Foundation | WPF und .NET.
[59] Vgl. What is WPF? - The complete WPF tutorial.

10.4.2 Aufbau

Der grundlegende Aufbau des WPF-Projekts bleibt im Grunde gleich. Jedoch wird hierbei nicht das Community-Toolkit verwendet, wodurch mehrere Funktionen selbst implementiert werden müssen.



Somit werden weitere Ordner für die Organisation dieser Funktionen benötigt

10.4.3 Design

Um der Applikation eine moderne Benutzeroberfläche zu verleihen, wird ein vorgefertigtes „Theme“ verwendet. Dabei handelt es sich um das „DarkTheme“ von „AngryCarrot789“ welches frei verwendet werden kann. In der folgenden Abbildung sind die einzelnen Designs der UI-Elemente abgebildet.[60]

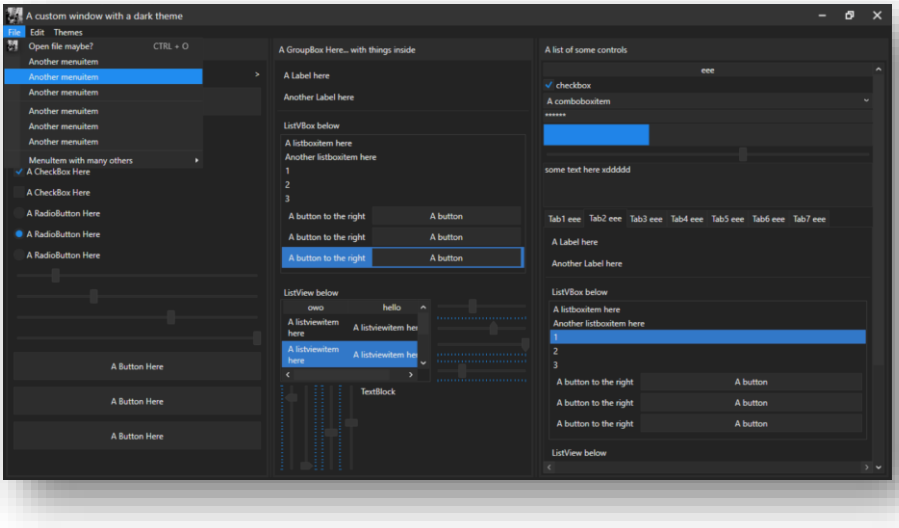


Abbildung 47: WPFDarkTheme

Um das Theme zu aktivieren werden die benötigten Dateien im entsprechenden Ordner „Themes“ abgelegt. Anschließend müssen diese noch im „App.xaml“ aktiviert werden. Dies ist

[60] Vgl. REghZy (2023).

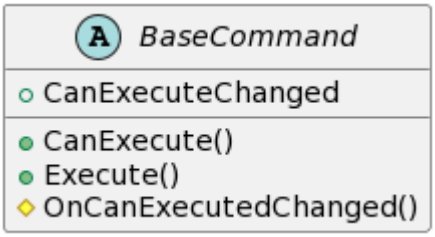
eine spezielle Datei, welche die Anwendungsressourcen und das globale Verhalten der Anwendung definiert. Sie wird automatisch beim Erstellen eines neuen WPF-Projekts in Visual Studio erstellt und definiert die Standard-Startklasse der Anwendung und die Standardfarben, Schriftarten und Stile, die in der Anwendung verwendet werden sollen.[61]

Unter dem Knotenpunkt „Resourcen“ kann nun der Pfad für das entsprechende Theme festgelegt werden.

```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="/Themes/Themes/ColourfulDarkTheme.xaml"/>
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Application.Resources>
```

10.4.4 Implementation der Commands

Für die Implementation der Commands wird hierbei eine Basisklasse erstellt, von welcher die einzelnen Commands erben



Die CanExecute-Funktion gibt an, wann ein Command ausgeführt werden kann. Dieser ist standardmäßig immer auf „Wahr“ gesetzt. Die wirklichen Anweisungen eines Commands befinden sich in der Execute-Methode, welche von den einzelnen Instanzen überschrieben werden können.

Beispiel:

Im folgendem Code-Abschnitt wird ein Command erstellt, welcher bei Aktivierung einen neuen Modus zur bestehenden Liste hinzufügt und anschließend das UI benachrichtigt.

[61] Vgl. Working with App.xaml - The complete WPF tutorial.

```
2 references
internal class AddModeCommand : BaseCommand
{
    private ModesDashboardViewModel vm;

    1 reference
    public AddModeCommand(ModesDashboardViewModel vm)
    {
        this.vm = vm;
    }

    1 reference
    public override void Execute(object parameter)
    {
        // add mdoe
        LocalBridge.ModeList.Add(new ussclientsandbox.Models.Mode("Mode " + (LocalBridge.ModeList.Count + 1).ToString()));
        vm.UpdateEntireUI();    // update ui of viewmodel
    }
}
```

10.4.5 Implementierung MVVM

Die Umsetzung des Model-View-ViewModel-Musters bleibt zur vorherigen Version nahezu identisch. Auch hier wird im Hauptfenster definiert, welches View zu welchem ViewModel angezeigt wird.

```
<DataTemplate DataType="{x:Type viewmodels:BridgeStatusViewModel}">
    <view:BridgeStatusView/>
</DataTemplate>

<DataTemplate DataType="{x:Type viewmodels:ModesDashboardViewModel}">
    <view:ModesDashboardView/>
</DataTemplate>

<DataTemplate DataType="{x:Type viewmodels:SwitchDashboardViewModel}">
    <view:SwitchDashboardView/>
</DataTemplate>
```

Dies geschieht bei dieser Umsetzung mittels „DataTemplates“. Dies sind Vorlagen, welche definieren, wie bestimmte Elemente dargestellt werden sollen.

Anschließend wird festgelegt, wo das derzeitige ViewModel zu finden ist. Dabei wird auf das „CurrentViewModel“ der Klasse „MainViewModel“ gebunden.

```
<!-- Use the current viewmodel as content control-->
<ContentControl Content="{Binding CurrentViewModel}"
                Grid.Row="1" Grid.ColumnSpan="3"/>
```

Das MainViewModel enthält immer das derzeitige ViewModel und wird aktualisiert, sobald sich dieses ändert. Ein Wechsel der ViewModels kann durch den „NavigationStore“ erfolgen. Dies ist eine statische Klasse und trägt dies als ihre einzige Aufgabe.

10.4.6 Informationen zur Bridge

Das Prinzip der BridgeStatusView bleibt gleich zur vorherigen Version. Zusätzlich wird noch eine Abbildung eingefügt, welche die Verbindung symbolisiert.

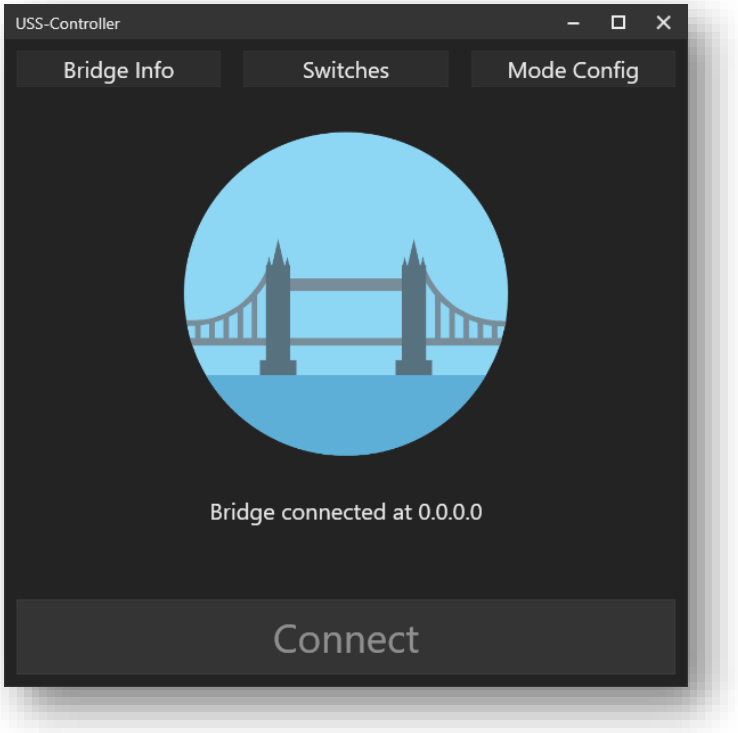


Abbildung 48: Bridge-Dashboard (Connected)

In dieser Abbildung wird keine valide IP-Adresse angezeigt, da hier die Applikation im Debug-Modus ausgeführt wird. Sollte keine Verbindung bestehen, so erscheint die Abbildung grau und der „Connect“-Button ist aktiviert.

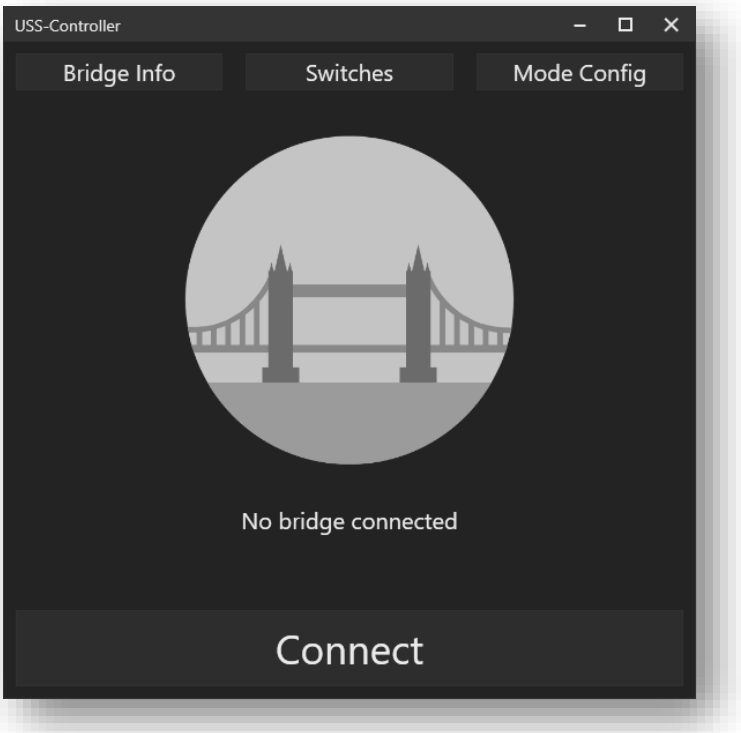


Abbildung 49: Bridge-Dashboard (Disconnected)

10.4.7 Einstellungen der Schalter

Im Gegensatz zur ursprünglichen Version werden hier Elemente entfernt, welche als nicht-notwendig eingestuft wurden. Dabei handelt es sich beispielsweise um die ID des Smart-Switches. Das grundlegende User-Interface bleibt jedoch gleich.

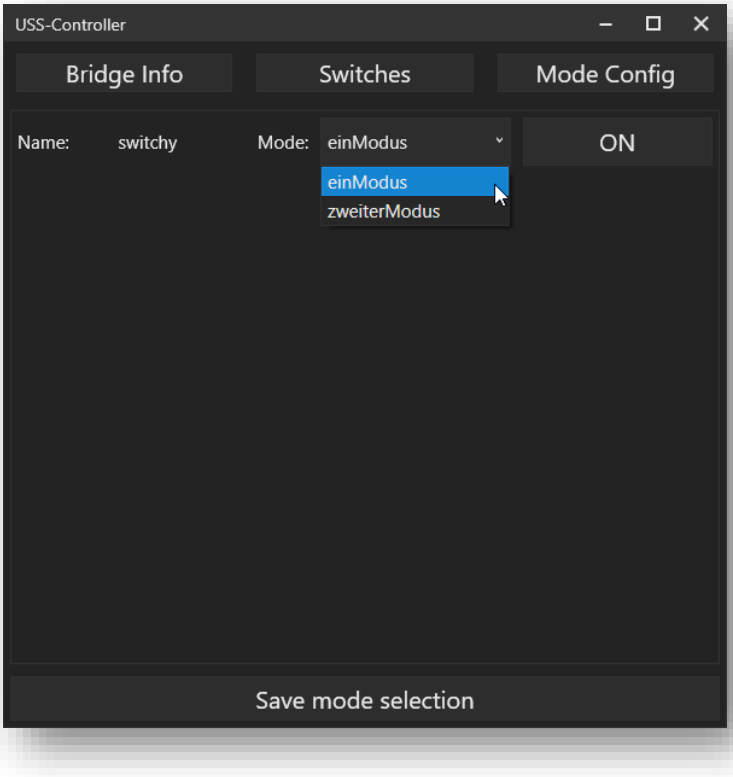


Abbildung 50: Mode-Config

In der obigen Abbildung ist der derzeitig gespeicherte Smart-Switch mit dem Namen „switchy“ abgebildet. Dieser befindet sich im aktiven Zustand und der eingestellte Modus kann frei verändert werden.

10.4.8 Konfiguration der Modis

Das Interface dieses Abteils ist so abgesichert, dass es fehlerhafte Eingaben des Benutzers korrigiert. Beispielsweise wird eine Uhrzeit-Charakteristik auf 00:00 zurückgesetzt, sollte der eingegebene Wert die möglichen Uhrzeiten eines Tages überschreiten.

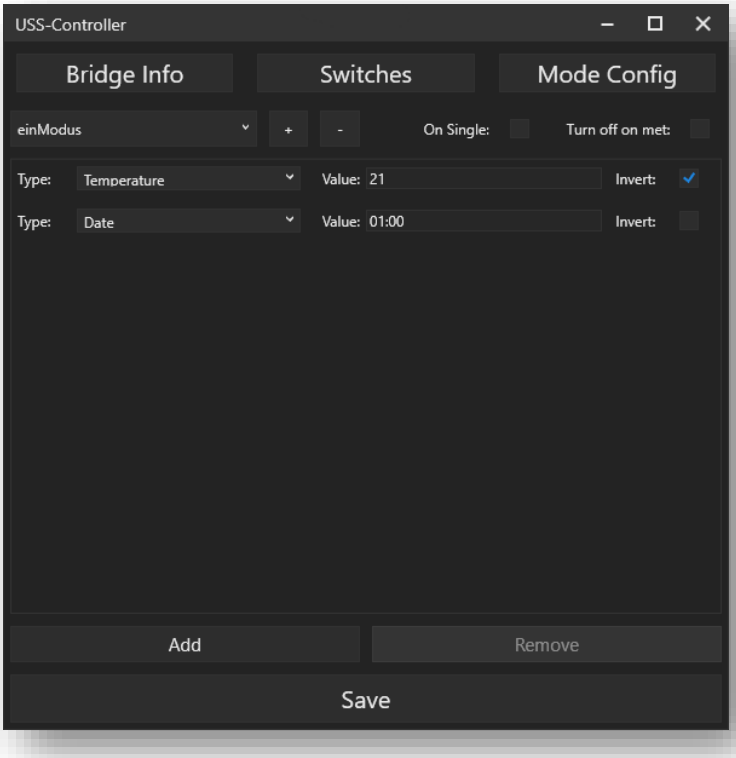


Abbildung 51: Switch-Config

Hier abgebildet sind die Einstellungen des Modus „einModus“, welcher zwei Characterisiken besitzt. Der Modus wird aktiviert, wenn es sich um die Uhrzeit 1:00 handelt und die Temperatur nicht 21 Grad Celsius beträgt.

11 KOMMUNIKATION

11.1 Grundidee

Um einen Austausch von Informationen zu ermöglichen, bedarf es einer Kommunikation zwischen den einzelnen Komponenten. Dabei werden standardisierte Kommunikationsarten verwendet.

11.2 Zwischen Bridge und Client

11.2.1 Nutzen

Damit die Bridge die nötigen Einstellungen erhält, um den Schalter zu steuern, bedarf es an Informationen, welche vom Benutzer (Client) versendet werden.

11.2.2 Wahl der Kommunikationsart

Die Art der Kommunikation muss so gewählt werden, dass diese auch über eine lange Distanz noch aufrecht bleibt. Hierbei fällt die Wahl auf das Open Systems Interconnection Modell (OSI).

Dieses Modell wurde 1984 eingeführt und beschreibt Regeln und Standards, wie Anwendungen und Netzwerkinfrastrukturen aussehen sollen. Genauere Erklärungen können im Anhang gefunden werden

11.2.3 Codierung

Um die Kommunikation zu beschleunigen, wird die Größe einer Nachricht komprimiert. Dies entsteht durch eine Vereinbarung der beiden Parteien, welche in einer Session kommunizieren. Deshalb wurde hierbei eine eigene Codierung entwickelt.

11.2.3.1 Vergleich der Lösungswege

Eine Möglichkeit zur Codierung wäre die Daten in klarem Text zu übertragen. Dabei würden die Zeichen durch ein Semicolon „;“ abgegrenzt.

Beispiel: „ID:2212,GETSTATE“

Die andere Möglichkeit wäre, den einzelnen Bits eine spezielle Rolle zuzuteilen. Falls dennoch Text übertragen werden soll, kann dieser dennoch in den vorgesehenen Feldern abgespeichert werden

Beispiel: „00111001101“ -> Erste 2 Bits: Markierung, 3->6 Zeitstempel, …

Direkte Übertragung der Daten in Text	Wertigkeit auf Bits
Direkt lesbar /Schreibbar	Bits müssen nach je nach Rolle den richtigen Bits zugewiesen werden
Höherer Speicherplatz benötigt (Durch Text)	Durch spezielle Zuweisung der Bits wird der Platzverbrauch minimal gehalten

Da der Fokus der Kommunikation auf Schnelligkeit und Effizienz beruht, wird hierbei die Möglichkeit 2 verwendet.

11.2.3.2 Definition

Eine Nachricht wird hierbei in sieben Abschnitte eingeteilt, wobei die ersten 3 Nachrichten als „Header“ angesehen werden.

Type:	StartMark	Length	CheckSum	ID	Command	Data	EndMark
Size:	2 Bytes	3 Bytes	1 Byte	1 Byte	1 Byte	Dependent on Length	3 Bytes
	Used to recognize message in stream of data	Length of data	Check if message was sent/received correctly = Bits after CS % 8	ID of Switch (Legacy)	Information of message type	Raw data	Used to recognize message in stream of data

Markierung: Beide benötigten Protokolle werden jeweils mit einer Start- und Endmarkierung versehen, welche anfangs 2 Bytes und am Ende 3 Bytes besitzen. Diese Felder sind notwendig, um die Daten in einem Stream zu erkennen und exportieren zu können.

Länge: In diesem Feld wird die Länge angegeben, welche der Nicht-Header-Anteil besitzt. Dies ist notwendig, da Anzahl der gesendeten Daten variiert.

Checksumme: Zur Verifizierung und Integrität der Nachricht wird eine Checksumme im Header eingefügt. Diese berechnet sich ausfolgender aus der Summe aller Bytes % 4.

Bei diesem Verfahren werden alle Bits des „Data“- Anteiles der Nachricht (Gegenstück des Headers) summiert und anschließend durch 8 (Anzahl der Bits eines Bytes) dividiert. Der dadurch entstehende Rest stellt nun die gesuchte Checksumme dar. Tritt bei der Überprüfung der Checksumme ein Fehler auf, wird das „Command“-Feld der Nachricht auf „Invalid“ (0001) gesetzt.

ID: Das ID-Feld besteht nur aus Legacy-Gründen. Da dieses Protokoll ursprünglich auch zur Kommunikation zwischen Bridge und Schalter genutzt werden sollte, musste die ID des Smart-Switches in der Nachricht abgespeichert werden. In der derzeitigen Version des Systems trägt dieses Feld keine Bedeutung.

Data: In diesem Feld werden echten Daten untergebracht. Dies können wieder Bits sein, welche eine eigene Bedeutung tragen oder auch einfacher Text.

Art der Nachricht: Zur Identifikation der Nachricht ist das „Command“-Feld zuständig, welches mittels einer 4-Bit Nummer die Nachricht genau identifiziert.

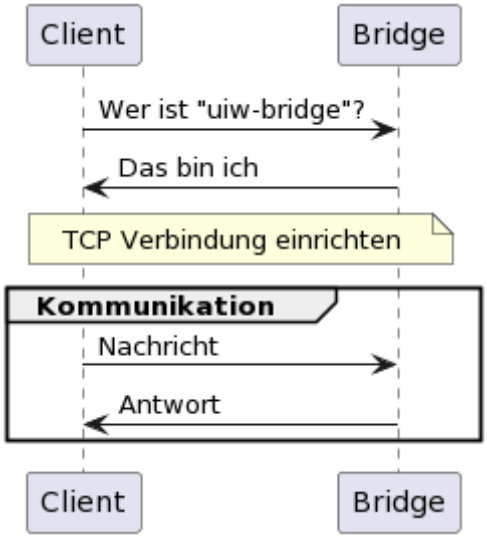
Type:	Invalid	EchoReq	EchoRep	GetSwitches	GetSwitchesRep	GetModes	GetModesRep	GetSysInfo	GetSysInfoRep	SetSwitchState
Send by:	-	-	-	Client	Client/Bridge	Client	Bridge	Client	Bridge	Client
In Bits:	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010
	Only obtained if error occurred	Echo Request	<- Reply	Request data of all switches	<- Reply	Request data of all modes	<- Reply	Request system data	<- Reply	Set state of switch

In der obigen Abbildung sind 10 verschiedene Kommando-Typen abgebildet. Jede Nachricht (mit Ausnahme von „Invalid“ und „SetSwitchState“) erhält dabei jeweils ein „Reply“-Gegenstück, welches die Antwort auf die gesendete Nachricht kennzeichnet. Soll vom Client ein Wert überschrieben werden, so wird dieser Typ wiederverwendet. Somit muss für diese Nachricht kein eigener Wert verwendet werden.

11.2.4 Anwendung

11.2.4.1 Beschreibung

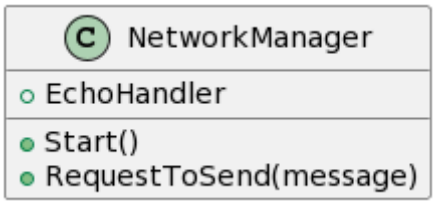
Die Bridge stellt bei dieser Kommunikation den Server dar, mit welchem sich ein Client verbinden kann. Die Bridge besitzt dabei den vordefinierten Hostname „usw-bridge“, damit diese in einem Netzwerk gefunden werden kann. Über diese Methode erhält der Client die IP-Adresse der Bridge, welche für die Verbindung notwendig ist. Der Server der Bridge läuft dabei auf dem vordefiniertem Port 5000. Dies ist eine Art Tor, welche einer Anwendung zugeteilt wurde.



11.2.4.2 Implementation Bridge

11.2.4.2.1 Network-Manager (Bridge)

Der grundsätzliche Austausch von Daten zwischen dem Client und der Bridge wird dabei in der Klasse „NetworkManager“ abgearbeitet.



Zur Implementation wird auf die Standard-Bibliothek „TCPServer“ zurückgegriffen, welche alle benötigten Funktionen zur Implementierung bereits inkludiert.

Die folgende Methode startet einen neuen TCP-Server, welcher bis zur Beendung des Programmes laufen soll und teilt ihm die Methode „EchoHandler“ als Handler zu.

```
def Start():
    server = TCPServer(('', DefinedInformation.TCPPort), EchoHandler)
    server.serve_forever()
```

Ein Handler ist dabei eine Methode, welche aufgerufen wird, wenn ein spezielles Event auftritt. In diesem Falle wäre dies ein Verbindungsversuch ausgehend von einem Client.

Dieser Handler wartet auf den Empfang einer neuen Nachricht, welche in der Variable „msg“ abgespeichert wird. Anschließend wird diese mittels der Funktion „BCValidateMessage“ Klasse „MessageController“ decoded. Wenn eine Flagge für das Versenden einer Nachricht aktiviert wurde, wird diese Nachricht abgesendet und die Flagge anschließend wieder deaktiviert.

```
class EchoHandler(BaseRequestHandler):
    def handle(self):
        print('[NW]: connection from:', self.client_address)
        while True:

            msg = self.request.recv(16384) #8192
            MessageController.BCValidateMessage(msg)
            if not msg:
                break
            try:
                if (GlobalStates.sendMessage):
                    self.request.send(bytes(GlobalStates.messageToSend.fullMessage))
                    GlobalStates.sendMessage = False
            except AttributeError:
                print('[NM]: Attribute Error')
```

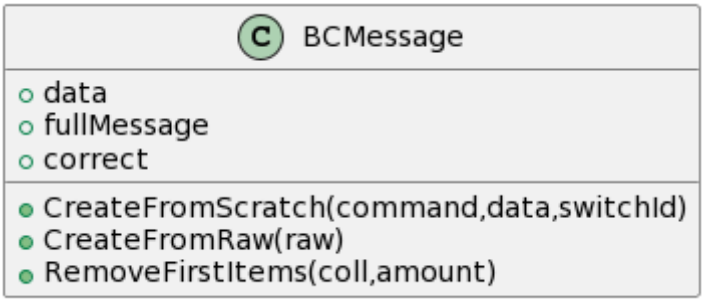
Um eine Nachricht zu senden, wird die Funktion „RequestToSend“ aufgerufen, und dieser die Nachricht als Parameter übergeben. Anschließend wird die Nachricht in der Klasse für globale Objekte und Variablen („GlobalStates“) abgespeichert. Um das Versenden zu ermöglichen, wird dabei die vorher besprochene Flagge gesetzt.

```
def RequestToSend(message):
    GlobalStates.messageToSend = message
    GlobalStates.sendMessage = True
    print('[NM]: Sent (' + str(message.data) + ')')
```

11.2.4.2.2 Message-Manager

Dieses Abteil des Programmes ist für die Codierung und Decodierung von Nachrichten zuständig.

BCMessage**:** Diese Klasse stellt eine Nachricht dar, welche für die Kommunikation verwendet werden kann. Ein Objekt dieser Klasse beinhaltet neben den einzelnen Felder der Codierung lediglich eine Liste aller Bytes („fullMessage“), eine Liste des Daten-Anteils („data“), eine Flagge, welche die Zuverlässigkeit der Nachricht beinhaltet („correct“) und das Kommando als Enumeration. Letzteres ist ein Datentyp, welcher einen Wert einer vordefinierten Aufzählung annehmen kann. Die Aufzählung wird hierbei durch die 10 Arten der Kommandos gebildet.



Zur Erstellung einer Nachricht bedarf es der Funktion „CrateFromScratch“. Deren Parameter sind die ID, das Kommando als Enum und die Daten als Bytes. Zuerst werden alle Parameter abgespeichert und überprüft, ob es sich bei den Daten um einen akzeptierbaren Wert handelt.

```
test = type(data)

self.command = command
self.commandRaw = command.value
self.dataString = data

if(type(data) == str):
    self.data = bytes(data, encoding='utf-8')
else:
    self.data = data
```

Anschließend werden alle Felder der Codierung nacheinander in die Liste eingefügt. Hierbei ein Beispiel anhand der Endmarkierung:

```
# add endmark bytes
for item in bytes(DefinedInformation.BCMarkEnd, 'utf-8'):
    self.fullMessage.append(item)
```

Für die Berechnung der Checksumme wird wiederum eine weitere Methode („CalcCheckSum“) zur Hilfe genommen. Diese wurde als eigene implementiert, um diese bei der Decodierung wiederzuverwenden.

```
def CalcCheckSum(self, dataToCalc ):
    fullVal = 0
    dataBytes = []

    if(type(dataToCalc) == str):
        dataBytes = bytes(dataToCalc, encoding='utf-8')
    else:
        dataBytes = dataToCalc

    datLen = dataBytes.count
    for item in dataBytes:
        fullVal += item

    while(fullVal > 8):
        fullVal = fullVal / 8

    # to byte
    fullVal = int(fullVal)
    fullValList = fullVal.to_bytes(1, 'big')
    fullVal = fullValList[0]

    return fullVal
```

Für die Decodierung einer Nachricht ist die Methode „CreateFromRaw“ zuständig, dessen einziger Parameter die empfangene Nachricht ist, welche an die Liste der gesamten Daten angehängt wird. Anschließend werden alle Felder aus der Nachricht gelesen und entfernt. Die erhaltene Checksumme wird mit der Berechneten abgeglichen. Sollten diese Werte nicht übereinstimmen, so wird die Flagge, sowohl als auch das Kommando auf „Invalid/False“ gestellt.

11.2.4.3 Implementation Client

11.2.4.3.1 Network-Manager (Client)

Auch hier übernimmt eine gleichnamige Klasse „NetworkManager“ die Aufgabe der Kommunikation.

In der Methode „Connect“ wird versucht, eine Verbindung mit der Bridge aufzubauen. Zuerst ruft diese eine Weitere Methode „Search“ auf, um die Bridge im Netzwerk zu suchen.

Hierbei wird die erste IPv4-Adresse mit dem passenden Hostname ausgewählt. Wenn keine dieser Adressen passt, wird angenommen, dass die Bridge nicht auffindbar ist und die Funktion gibt den Werte „false „ (Falsch) zurück. Dies geschieht auch, wenn ein Fehler bei der Suche auftritt.

```
public static void Search()
{
    try
    {
        if (!DefinedInformation.LocalDebugMode)
        {
            IPAddress[] addresslist = Dns.GetHostAddresses(DefinedInformation.BridgeHostName);
            bridgeAdress = new IPAddress(addresslist[0].GetAddressBytes());


            foreach (var item in addresslist)
            {
                if (CheckIfIPv4(item.ToString()))
                {
                    bridgeAdress = new IPAddress(item.GetAddressBytes());
                    break;
                }
            }
        }
        else { bridgeAdress = new IPAddress(0); }
        bridgeFound = true;
    }
    catch (Exception e)
    {
        Console.WriteLine("[NM]: {0}", e.Message);
        bridgeFound = false;
    }
}
```

Die Überprüfung, ob eine IP-Adresse auch eine gültige IPv4 Adresse ist, wird in der Methode „CheckIfIPv4“ abgearbeitet.

Hierbei werden „Regular Expressions“ (RegEx) verwendet. Dies sind Zeichenketten zur Beschreibung von Mustern innerhalb eines Strings. Ein RegEx besteht aus einer Kombination aus Literal- und Metazeichen. Literalzeichen beschreiben genau die Zeichen, welche gefunden werden sollen, während Metazeichen bestimmte Arten von Mustern beschreiben. Beispiele für Metazeichen sind Sterne, Punkte und Fragezeichen. Die dabei verwendete Zeichenkette wurde dabei von einer Internet-Quelle übernommen.

Wenn die Voraussetzungen für eine IPv4 Adresse stimmen, erhält der Rückgabewert den Zustand „true“, ansonsten „false“. Wie auch bei der Suchfunktion wird dieser Wert auch zurückgegeben, wenn ein Fehler auftreten sollte.

```
public static bool CheckIfIPv4(string IpAddress)
{
    try
    {
        Regex regex = new Regex(@"((^\s*((([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9]
        bool flag = false;
        string IPv = string.Empty;
        if (!string.IsNullOrEmpty(IpAddress))
        {
            if (IpAddress.Count(c => c == '.') == 3)
            {
                flag = regex.IsMatch(IpAddress);
                IPv = "IPv4";
            }
            else
            {
                IPv = "Version of";
                flag = false;
            }
        }

        if (flag)
        {
            return true;
        }
        else
        {
            Console.WriteLine("{0} is not a valid {1} IP address", IpAddress, IPv);
            return false;
        }
    }
    catch (Exception) { return false; }
}
```

Kann die Bridge erfolgreich gefunden werden, so wird ein neues Objekt der Klasse TCP-Client erstellt, welche durch das „Simple-TCP“ Nuget Packet bereitgestellt wird.

Anschließend werden diesem Objekt mehrere Methoden angebunden, welche bei Auftreten verschiedener Events aufgerufen werden. Zum Abschluss wird der TCP-Client gestartet. Sollte hierbei ein Fehler auftreten, wird dieser von dem Try-Catch-Block aufgefangen und die entsprechende Flagge gesetzt.

Für das Empfangen einer Nachricht wird ein Task verwendet. Ein Task stellt in C# eine asynchrone Operation dar. Diese läuft parallel zum Haupt-Thread des Programmes ab. Beim Empfang einer Nachricht wird diese in eine Liste eingefügt. Sollte dieser Ein (End-) Markierung enthalten, wird angenommen, dass es sich hierbei um eine echte Nachricht handelt. Diese wird anschließend decodiert und dem „Message-Controller“ übergeben.

Der Ablauf zur Decodierung einer Nachricht ist nahezu identisch mit dem der Bridge. Die einzigen Unterschiede stellen hierbei zumal der Syntax als auch die Properties dar.

11.3 Zwischen Bridge und Schalter

11.3.1 Allgemeines

Die Kommunikation zwischen der Bridge und den Schaltern muss natürlich einigen Anforderungen entsprechen. Die Bridge sollte eine möglichst hohe Anzahl an Schaltern verwalten können. Die zu übertragenden Datenpakete sind allerdings sehr klein, zeitlich stellt dies deshalb kein großes Problem dar. Die Reichweite, also die Entfernung, in der sich die Schalter von der Bridge befinden, ist allerdings kritisch. Soll beispielsweise ein großes Haus ausgestattet werden, so ist es möglich, dass suboptimaler Weise mehrere Bridges benötigt werden. Die Kommunikation muss weiters in beide Richtungen möglich sein. Bridge zu Schalter, um Kommandos versenden zu können und in die andere Richtung um Temperaturdaten zu Übertragen. Ebenfalls muss die Verbindung zwischen den Geräten mit dem Anschließen des Schalters ans Stromnetz automatisch geschehen.

11.3.2 Mögliche Übertragungsarten

Bluetooth (BLE)

Bluetooth bietet uns eine mehr als genügende Datenrate. Die nicht allzu große Reichweite stellt allerdings ein Problem dar, diese hängt unter anderem von der Version der Bluetooth Kommunikation ab. Für größere Reichweiten werden allerdings Versionsunabhängig sehr hohe Energiemengen benötigt (Siehe Kapitel 8.4.7.2). Eine mögliche Lösung dafür wäre ein Mesh-Netzwerk. Mit Bluetooth Versionen ab 4.0 und den dazugehörigen Low-Energy Modes kann auch eine energieeffiziente Übertragung erreicht werden. Ebenfalls hat unsere Partnerfirma bereits Erfahrung mit Bluetooth.[62]

BLE (Bluetooth Low Energy) ist der ideale Bluetooth Modus für uns. Er eignet sich speziell für nicht frequente kurze Kommunikationen mit geringer Datenmenge, was genau unseren Anforderungen entspricht. Es gibt auch bereits viele Dokumentationen für die Implementation von BLE.

Zigbee

Zigbee ist eine Kommunikationsmöglichkeit, die ähnlich wie Bluetooth funktioniert, allerdings ist die Reichweite etwas höher und die Datenrate ist etwas niedriger. Das macht es für uns zu einer attraktiven Möglichkeit. Allerdings haben wir kein Vorwissen und keine Erfahrung mit der Technologie und die Dokumentation ist ebenfalls schlechter als beispielsweise BLE. Deshalb wird BLE gewählt.

[62] Intro zu Bluetooth Stromverbrauch | Bluetooth® Technologie Website (2017).

Powerline Kommunikation

Bei der Powerline Kommunikation werden die Daten direkt über die Stromleitung versandt. Da der Anschluss sowieso benötigt wird ist dies eine elegante Lösung. Reichweitetechnisch wäre sie auch ideal für uns, da wir problemlos durch das ganze Haus kommunizieren könnten. Die Datenrate wäre ebenfalls mehr als hoch genug, allerdings ist das Modul, das für diese Kommunikation in jeden Schalter eingebaut werden müsste, relativ groß. Da es sowieso schon mit Platzproblemen zu kämpfen ist, ist dies kritisch. Die Technologie ist auch noch nicht so ausgereift wie die Alternativen. Das macht die Implementation sehr aufwendig und kompliziert, oft entstehen Datenlecks oder dergleichen, aufgrund der rauschenden Netzspannung. Deshalb wurde diese Übertragungsart nicht gewählt.

Wlan

Wlan wäre neben Bluetooth die uns am besten vertraute Art der Übertragung. Bluetooth bietet allerdings eine größere Reichweite für den gleichen Energieverbrauch an, was für unseren Verwendungszweck einen hohen Stellenwert hat. Weiters müsste ein eigenes WLAN-Netzwerk aufgespannt werden, um die automatische Verbindung zwischen Bridge und Schalter zu ermöglichen, dies benötigt erneut eine größere Menge an Energie.

11.3.3 Bluetooth Low Energy (BLE)

11.3.3.1 Theorie

Die gewählte Kommunikationsart BLE ist im Vergleich zu herkömmlichem Bluetooth recht aufwendig. Da der Aufbau um einiges komplizierter ist. Grundsätzlich handelt es sich aber auch bei BLE um eine serielle Schnittstelle, über welche die Daten gesendet werden.

11.3.3.1.1 Verbindungsarten

Punkt zu Punkt

Zwei Geräte verbinden sich direkt miteinander, die Bridge verbindet sich also mit jedem Schalter direkt, um den Datenaustausch zu ermöglichen. Die übliche Datenrate von 1,4 Mb/s sinkt zwar stark aufgrund der multiplen Verbindungen, das ist für unsere winzigen Datenpakete aber wie gesagt irrelevant. Das ist auch die verwendete Verbindungsvariante.

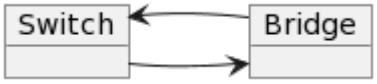


Abbildung 52: BLE-Point to Point

Mesh Netzwerk

Bei einem Mesh Netzwerk können viele unterschiedliche Geräte miteinander verbunden werden. Jedes Gerät kann selbst Nachrichten verschicken oder weiterleiten, so ist es möglich das Problem der Reichweite quasi zu eliminieren. Der Datenverkehr wird so natürlich deutlich erhöht, mit der Datenrate von 3,45 kbit/s ist das allerdings kein Problem. Aufgrund des höheren Implementationsaufwands wurde diese Variante noch nicht realisiert, bietet aber eine gute Erweiterungsmöglichkeit.

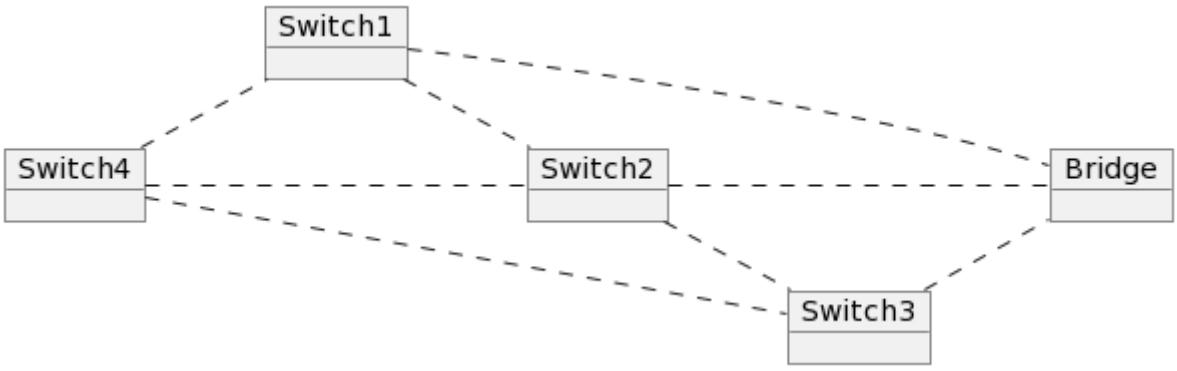


Abbildung 53: BLE Mesh Netzwerk

11.3.3.1.2 GAP

Die wichtigste Definition im Generic Access Profile (GAP) ist der Aufbau einer Verbindung zwischen zwei Geräten. Dabei wird bei der BLE Punkt zu Punkt Verbindungen zwischen dem Zentrum und der Peripherie unterschieden. Das Zentrum nimmt die Master Rolle ein und verbraucht gleichzeitig den Großteil der Rechenleistung, diese Rolle nimmt bei uns deshalb die Bridge ein. Die Peripherie, bei uns der Schalter, übernimmt die Slave Rolle.

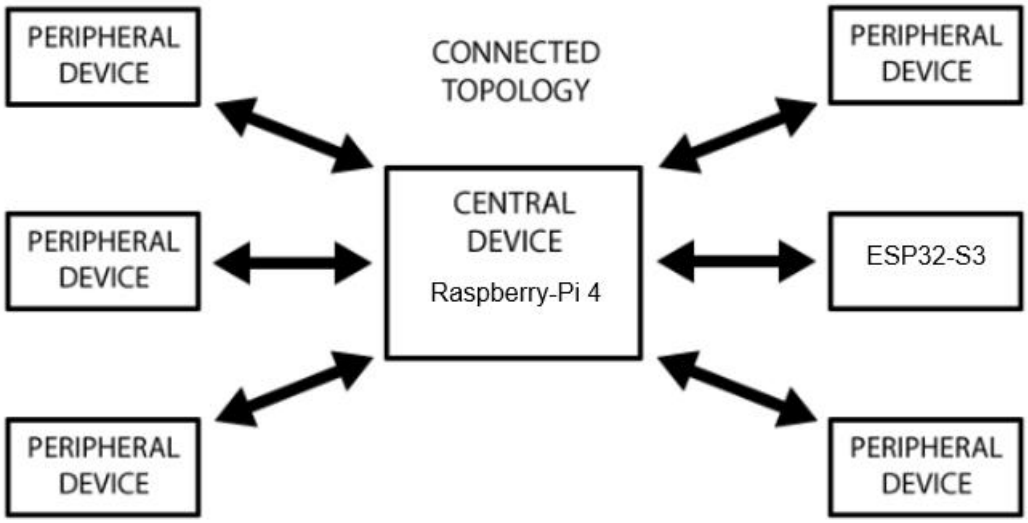


Abbildung 54: GAP Funktionalität

Die Peripheriegeräte können sich nur mit einem Zentrum verbinden. Müssen also Daten von einem Schalter zum anderen geschickt werden geschieht dies über das Zentrum.

Um von der Bridge für einen Verbindungsaufbau erkannt werden zu können, muss der Schalter in den General Discoverable Mode gesetzt werden.

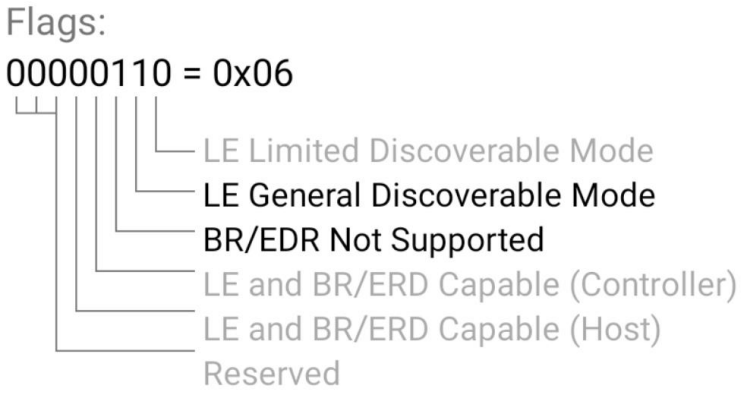


Abbildung 55: BLE Advertising Flag

Weiters muss die Peripherie als Directed Connectable operiert werden. Dies ermöglicht bei einem Verbindungsverlust eine sehr schnelle Wiederverbindung und da es nur ein Zentrum gibt ist das Verbinden mit anderen Geräten irrelevant.

Um schlussendlich eine Verbindung aufzubauen muss der korrekte Advertising Modus ausgewählt werden, in unserem Fall: [63]

ADV_DIRECT_IND	Legacy 4.x	Connection-oriented communication	Connectable directed advertisement.

Das GAP verfügt ebenfalls über hervorragende Sicherheitsprotokolle, diese werden in den vordefinierten Bibliotheken bereits automatisch eingebunden. Beim Schalter wurde beispielsweise die Arduino BLE Library verwendet.[64]

11.3.3.1.3 GATT

Allgemeines

Das Generic Attribute Profile (GATT) ist für den tatsächlichen Datenaustausch zwischen Schalter und Bridge verantwortlich. Auch wenn durch GAP mehrere „gleichzeitige“ Verbindungen hergestellt werden können. Kann man immer nur zwischen 2 Punkt zu Punkt verbundenen Geräten kommuniziert werden. Das heißt die Bridge muss sich mit der Kommunikation mit den verschiedenen Schaltern abwechseln.

Das Zentrum heißt hier GATT-Client und die Peripheriegeräte GATT-Server. Der Datenaustausch funktioniert in dem ein GATT-Server (der Schalter) eine Anfrage an den Client (die Bridge) sendet. Diese antwortet dann mit einer entsprechenden Antwort. Was für Daten in was für einem Format genau versendet werden, ist in der Attribut Protokoll Tabelle (ATT) vordefiniert und wird Profil genannt. Man kann der Tabelle auch eigene Profile hinzufügen. [65]

Protokoll

Type:	StartMark	CheckSum	ID	Command	Data	EndMark
Size:	2Byte	1Byte	1Byte	1Byte	1Byte	2Byte
	Used to recognize message in stream of data	Check if message was sent/received correctly = Bits after CS % 8	Unique device ID	Information of messag type	Raw data	Used to recognize message in stream of data

[63] Vgl. Lesson 2 – BLE profiles, services, characteristics, device roles and network topology (2019).
[64] Vgl. ArduinoBLE (2023).
[65] Vgl. Afaneh, M. (2020).

Kommando Typen

Type:	Invalid	GetState	GetStateRep	GetTemp	GetTempRep	EchoRequest	EchoReply
Send by:	-	Bridge	Bridge/Switch	Bridge	Switch	-	
In Bits:	0001	0010	0011	0100	0101	0110	0111
	Only obtained if error occurred	Request value of current switch state	Real value of state Can act as->1. Reply of 0010 2. Setting state	Get temperature value of sensor	Response to 0011		

Aufbau der Profile

Ein solches Profil besteht aus Services und den diesen untergeordneten Charakteristiken.

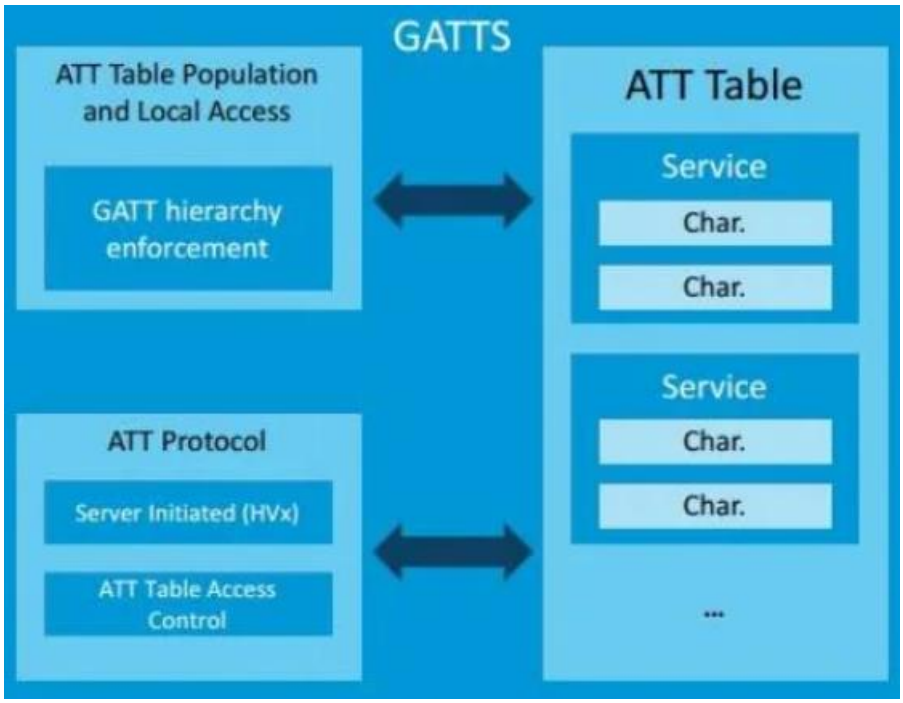


Abbildung 56: GATT-Tabelle

Diese sind jeweils mit einer UUID (Universal unique identifier) gekennzeichnet und können mit dieser aufgerufen werden. Dabei werden UUIDs zwischen 16 Bit und 128 Bit an Länge unterschieden. Die 16 Bit IDs sind bereits vordefiniert, während die 128 Bit IDs selbst definiert werden können, was wir in unserem Fall auch getan haben. Das gilt sowohl für die Services als auch die Charakteristiken. [66]

[66] about-ble-server-profile-20-1024.jpg (1024×768).

Ein Service ist also eine Einheit, die erstellt werden muss, um eine gewisse Aufgabe zu erfüllen, in unserem Fall die Kommunikation zwischen Schalter und Bridge. Untergeordnet können mehrere Charakteristiken erstellt werden, die die genauen Spezifikationen einer Nachricht enthalten Für unsere Zwecke wird nur eine einzige Charakteristik benötigt, welche das Kommunikationsprotokoll enthält. Es wird deshalb jeweils eine selbst definierte UUID vergeben. Diese haben wir über einen Generator erhalten.[67]

```
16  #define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914a"
17  #define CHARACTERISTIC_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
18  #define DESCRIPTOR_UUID     "4fafc201-1fb5-459e-8fcc-c5c9c331914c"
```

Es gibt auch noch weitere Teile, die in einem Profil enthalten sind, beispielsweise Descriptor, die verwendet werden, um zusätzliche Informationen über ein Profil, einen Service oder eine Charakteristik zu bieten. Diese wurden aufgrund der Übersicht ebenfalls hinzugefügt, um sie kennzeichnen zu können haben sie ebenfalls eine UUID.[68] [69]

11.3.3.2 Praxis

Die Bluetooth-Verbindung wird gestartet und alle nötigen Parameter werden definiert.

```
void setup() {
  Serial.begin(115200); //BLE Interface einschalten
  BLEDevice::init("Smartswitch-1234");  //Gezeigter Name meines Servers
  BLEServer *pServer = BLEDevice::createServer();  // Als BLE Server advertisen
  pServer->setCallbacks(new MyServerCallbacks()); //Checken ob man Verbunden ist

  BLEService *pService = pServer->createService(SERVICE_UUID); //Service erstellen

  BLECharacteristic *pCharacteristic = pService->createCharacteristic(  //Properties der Charakteristik festlegen
    CHARACTERISTIC_UUID,
    BLECharacteristic::PROPERTY_READ |     //Read klar
    BLECharacteristic::PROPERTY_WRITE |    //Write man kann bestimmte Zeilen schicken
    BLECharacteristic::PROPERTY_NOTIFY    //Notify bei jeder Änderung z.B. des Temp Sensor wird der neue Wert gesendet
  );
```

```
  BLEDescriptor *UselessDescriptor = pCharacteristic->createDescriptor(DESCRIPTOR_UUID);
  //Muss hinzugefügt werden damit Notify funktioniert
  BLEDescriptor *UselessDescriptor(DESCRIPTOR_UUID);  //Descriptor bietet zusätzliche Daten ist optional
  VariableDescriptor.setValue("Do kunnt d Temperatur");
  pCharacteristic.addDescriptor(&VariableDescriptor);

  pCharacteristic->setValue(Temp_Sensor());   //Wert für die Charakteristik setzen
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();   //Advertising definieren
  pService->start();    //Service starten

  pAdvertising->addServiceUUID(SERVICE_UUID); //Service dem Advertisting hinzufügen
  pAdvertising->setScanResponse(true);    //Kann gefunden werden
  pAdvertising->setMinPreferred(0x12);

  BLEDevice::startAdvertising(); //Advertising starten
}
```

[67] Online UUID Generator Tool.
[68] Lesson 2 – BLE profiles, services, characteristics, device roles and network topology (2019).
[69] Introduction to Bluetooth Low Energy.

Unten zu sehen sind die Daten, welche vom Schalter bei der BLE-Übertragung für den Client sichtbar sind. Aufgrund eines Problems mit dem Descriptor, haben sowohl Service als auch Charakteristik keinen Namen. Die grafische Datendarstellung wurde mithilfe des NrF-Connect am Smartphone gemacht. Dabei handelt es sich um eine App welche Bluetooth Verbindungen aufbauen kann und die entsprechenden Daten ausgibt.

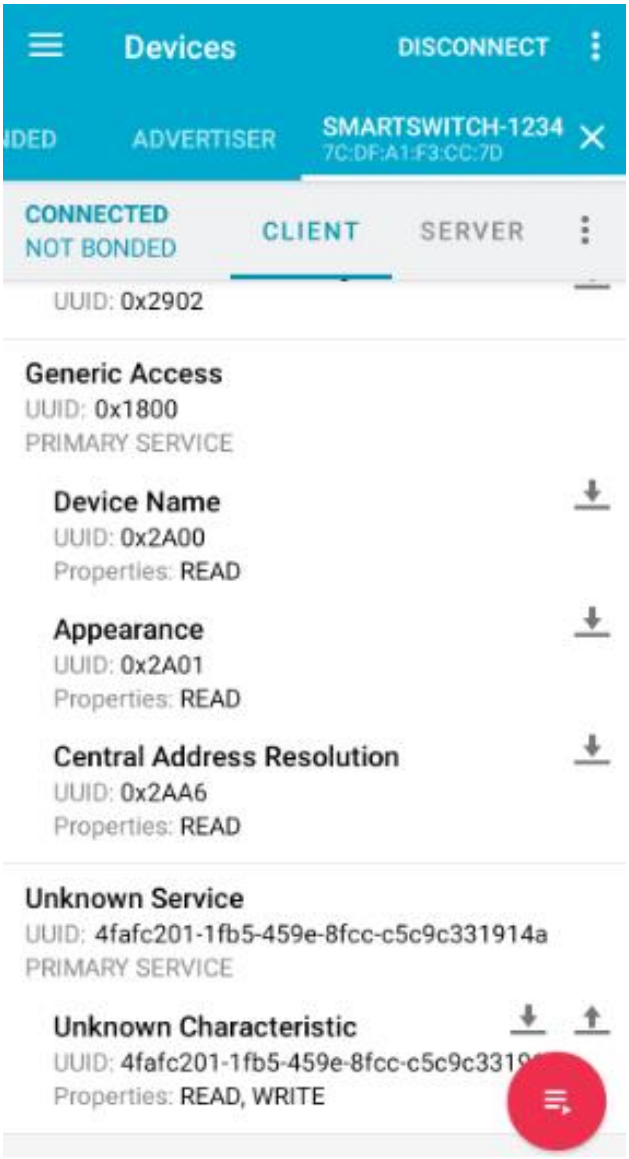


Abbildung 57: BLE Verbindung Handy

12 INSTALLATIONSANLEITUNG

Das Ziel ist es die Installation so einfach wie möglich zu gestalten, aber trotzdem möglichst universell anwendbar zu sein. Die gewählte Lösung war es hierbei Ein- und Ausgänge für Phase, Nullleiter und Erdung anzubringen. Ist nur ein Nullleiter und oder eine Erdung vorhanden, so kann jeweils ein beliebiger der beiden Pins angeschlossen werden. Die Erdung ist zwar für die Funktionalität nicht relevant, muss aber aus Sicherheitsgründen angeschlossen werden. Wird der Schalter nach der Installation gegen Berührungen vollständig geschützt ist dürfen die Erdungsanschlüsse offengelassen werden. Zusätzlich ist hier empfohlen die Erdung des Niederspannungsanteils über den vorgesehenen Pin (Siehe Kapitel 8.5.2.2.2) an dem Gehäuse zu befestigen.

Der Anschluss erfolgt über zwei 3-polige Schraubklemmen Ein- und Ausgang müssen dabei korrekt angeschlossen werden (Siehe Kapitel 15.2).

WICHTIG

Während der Installation muss der Strom im entsprechenden Raum unbedingt abgestellt werden. Dabei sind die 5 klassischen Sicherheitsregeln zu beachten. [70]

[70] Adam, B. (2020).

12 DANKESWORTE

An dieser Stelle möchten wir uns herzlich bei alle denjenigen bedanken, die uns geholfen haben, diese Diplomarbeit zu verfassen. Zum einen ist unser Betreuungslehrer Herr Bischof, unserem Firmenbetreuer Joseph Kruijen und zum andern Erika Kicker, die uns unsere Texte Korrektur gelesen hat.

Vielen Dank.

13 ABBILDUNGSVERZEICHNIS

14 FORTSCHRITTSBERICHT

14.1 Teammitglied 1 (Johannes Klapper)

DATUM	WAS	STUNDEN
19.09.2022	Recherche existierende Wandschalter	4
26.09.2022	Details zur Umsetzung festlegen	4
03.10.2022	Planung Projekt	3
10.10.2022	Planung Projekt Recherche Umsetzbarkeit	4
17.10.2022	Auswahl der Übertragungsart	4
19.10.2022	Konzept festlegen Mikrocontroller Recherche	3.5
24.10.2022	Schaltplan Schalter v1 erstellt	4
07.11.2022	Temperatur Sensor und Mikrocontroller Auswahl Dev-Kit bestellt Schaltung überarbeitet	4
09.11.2022	Bauteil Librarys und Layout erstellt	3
13.11.2022	Layout finalisiert	2
14.11.2022	Gerber Files erstellt Platine bestellt	4
17.11.2022	Einstieg in ESP-IDF	5
21.11.2022	Fehlerbehebung Cmake Testprogramm DevKit	6
28.11.2022	Erneute Fehlerbehebung Cmake Testprogramm Relais	4
05.12.2022	Einstieg Arduino IDE Testprogramme erneut (+Temperatur Sensor)	4.5
11.12.2022	Bestückung Schalter v1 Testung	2.5
12.12.2022	Temperatur Sensor Programmänderung	4

16.12.2022	Testung Schalter v1 gesamt	5
19.12.2022	Planung Schalter v2	4
08.01.2023	Bauteilauswahl Librarys erstellen	5
09.01.2023	Librarys erstellen und Schaltung planen	3
16.01.2023	Schaltung erstellen BLE-Implementierung	4
18.01.2023	BLE Testung/Fehlerbehebung	2.5
19.01.2023	BLE Testung/Fehlerbehebung Verbindungsaufbau Handy erfolgreich	2
06.02.2023	Schaltplan fertigstellen Layout beginnen	4
09.02.2023	Layout Neustart (Größenvorgaben)	3
16.02.2023	Implementation BLE	5
20.02.2023	Layout anpassen	3
21.02.2023	Layout und Schaltplan anpassen Bauteile bestellen	4.5
27.02.2023	Layout finalisieren	4
02.03.2023	SPI-Programmierung	2.5
03.03.2023	Platine Schalter v2 bestellen	2
06.03.2023	SPI-Programmierung Bauteile nachbestellen	4
07.03.2023	Dokumentation beginne	5
08.03.2023	Programmanpassung Temperatur Sensor Berechnungen Temperatur Sensor	5.5
13.03.2023	Dokumentation	4
14.03.2023	Gehäuse erstellen	4.5
16.03.2023	Dokumentation	2
18.03.2023	Dokumentation finalisieren	5.5
19.03.2023	Dokumentation beenden	4.5

14.2 Teammitglied 2 (Tim Kicker)

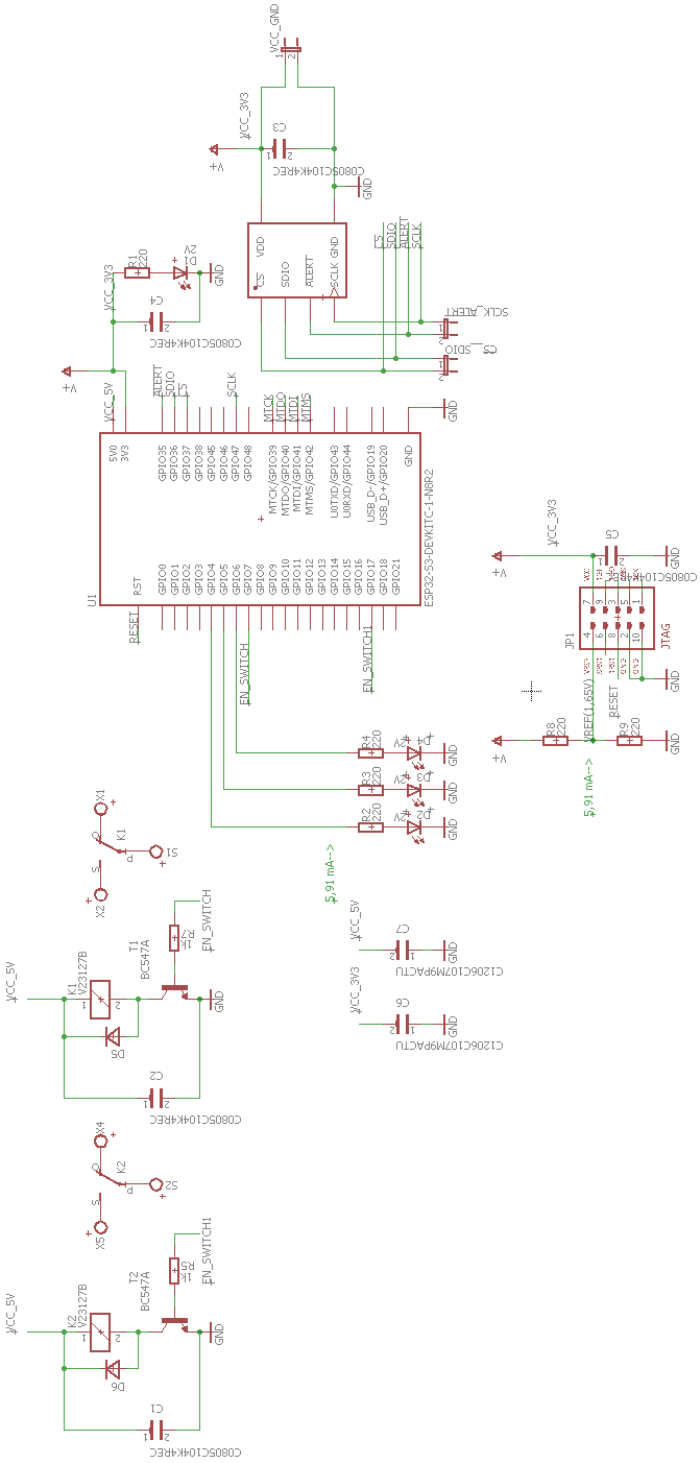
DATUM	WAS	STUNDEN
19.09.2022	Recherche Wandschalter etc.	4
26.09.2022	Planung Projekt	4
03.10.2022	Recherche/Planung Frameworks etc. Aufstellung Struktur (Klassen etc.)	4
10.10.2022	Grundstrukturen (MVVM, etc.). Erstellung Git Repos Aufsetzung Entwicklungsumgebung (VS,VSC über SSH	4
13.10.2022	Raspi aufsetzen + Einrichtung	2
14.10.2022	Erstellung DefinedInformation, GlobalStates, MessageManager und ModeManager	3
16.10.2022	Erstellen und Lesen von Nachrichten (Client & Bridge)	7
17.10.2022	Weiterarbeitung Nachrichten	4
24.10.2022	Implementation Klassen Modes, Switches, etc. Checksumme/Überprüfung von Nachrichten	4
07.11.2022	Abprüfen von Modis, Beginn Kommunikation	4
14.11.2022	Kommunikation (Client <-> Bridge)	4
20.11.2022	Kommunikation (Client <-> Bridge)	5
21.11.2022	Autofind + Connect (Client <-> Bridge)	4
23.11.2022	Bugfixing Checksumme	3
28.11.2022	Serialize Modes/Switches zu XML, Deserialize Modes	4
01.12.2022	Read/Load XML von Messages	3.5
05.12.2022	Install-Script erstellen (Bash), Bugfixing TCP-Server/Client	4
10.12.2022	Recherche .NET MAUI	3
11.12.2022	Demoprojekt + Übung MAUI	4
12.12.2022	Erstellung MAUI-Projekt, Implementierung MVVM + Grundstruktur	4
18.12.2022	Planung + Erstellung UI-MAUI	5

19.12.2022	Verknüpfung Basis- und MAUI-Projekt	4
28.12.2022	Bugfixing .NET MAUI	7
04.01.2023	Recherche Alternative MAUI	2
06.01.2023	Besprechung Framework-Wechstel	1.5
09.01.2023	Beginn Neuimplementation in WPF	4
14.01.2023	Neuimplementation in WPF	3
16.12.2023	Mode-Executor (Bridge)	4
23.01.2023	Weiterführung WPF-Implementation	4
28.01.2023	Planung Bluetooth-Verbindung	3
30.01.2023	Implementation Bluetooth-Verbindung	4
04.06.2023	Implementation Bluetooth-Verbindung	4.5
06.02.2023	Testung Bluetooth-Verbindung mittels Handy-Simulation	4
16.02.2023	Implementation Bluetooth-Verbindung	6
18.02.2023	Dokumentation	5
27.02.2023	Dokumentation	4
23.02.2023	Bugfixing (Client)	4
06.03.2023	Dokumentation + Bugfixing (Bridge)	4
13.03.2023	Dokumentation	4
20.03.2023	Dokumentation	4
27.03.2023	Dokumentation	4
28.03.2023	Dokumentation	5
29.03.2023	Dokumentation	3.4

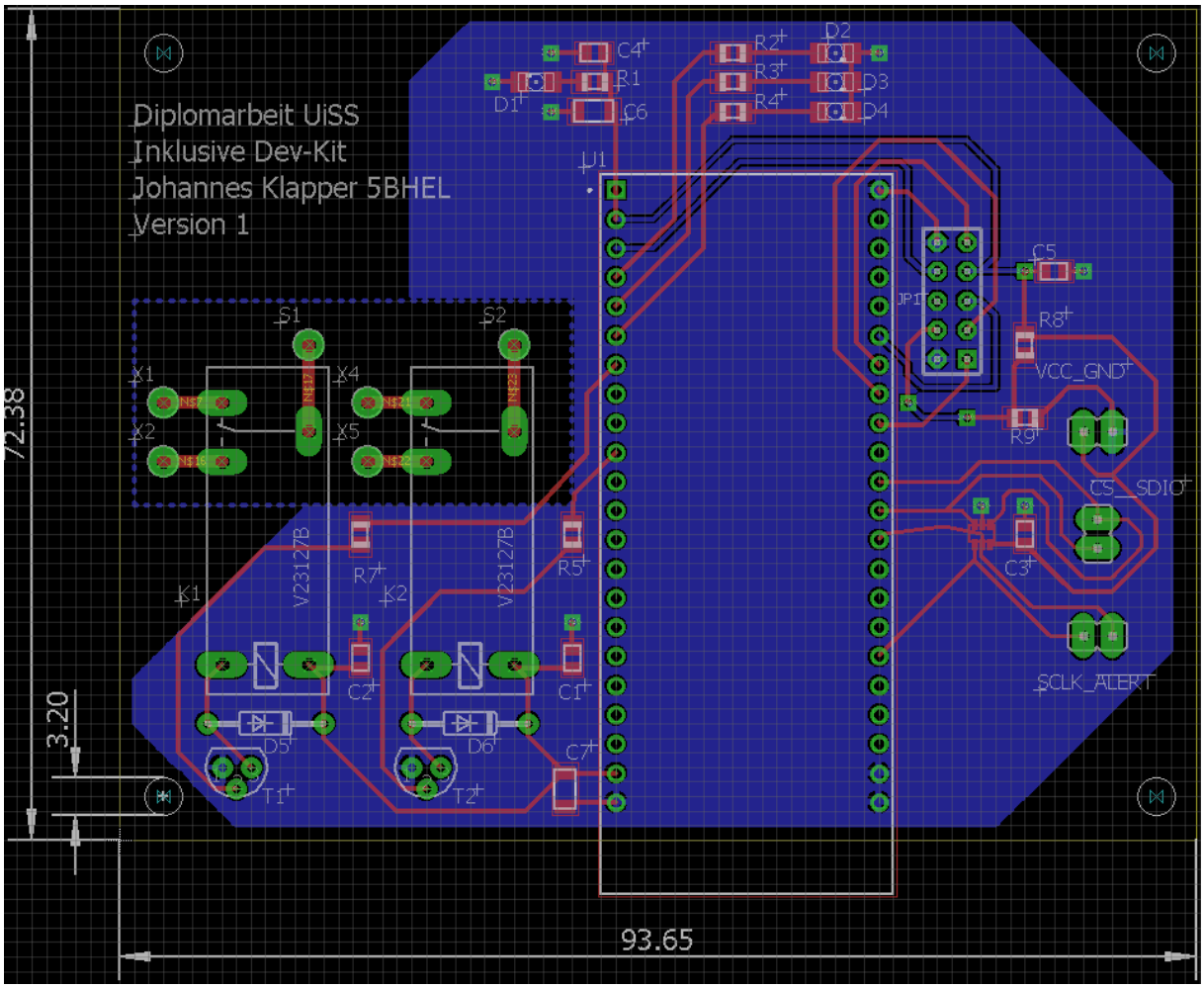
15 ANHANG

15.1 Schalter V1

15.1.1 Schaltplan



15.1.2 Layout

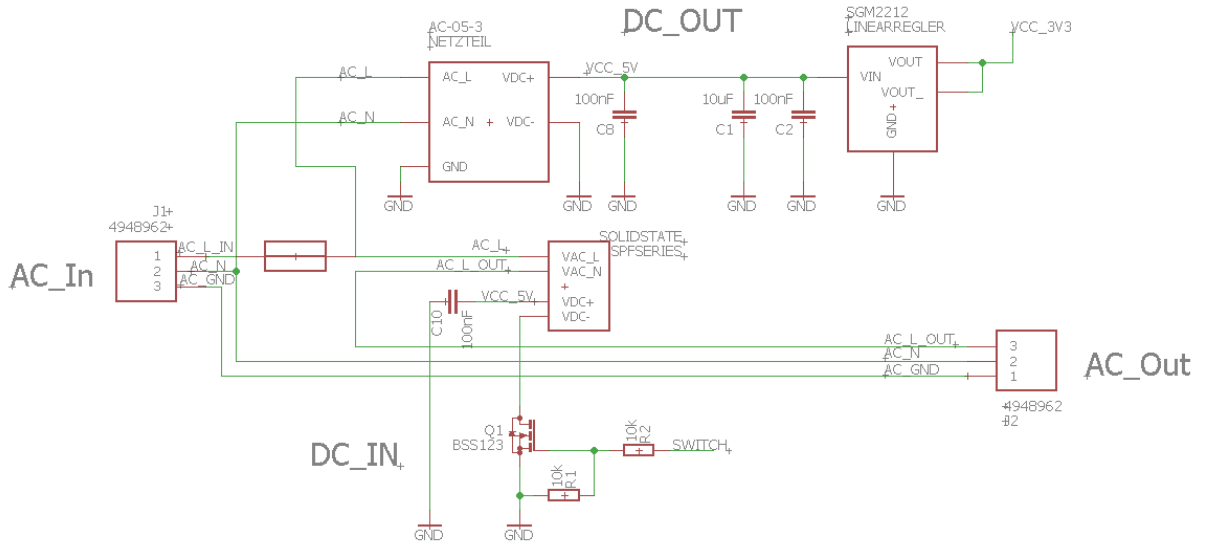


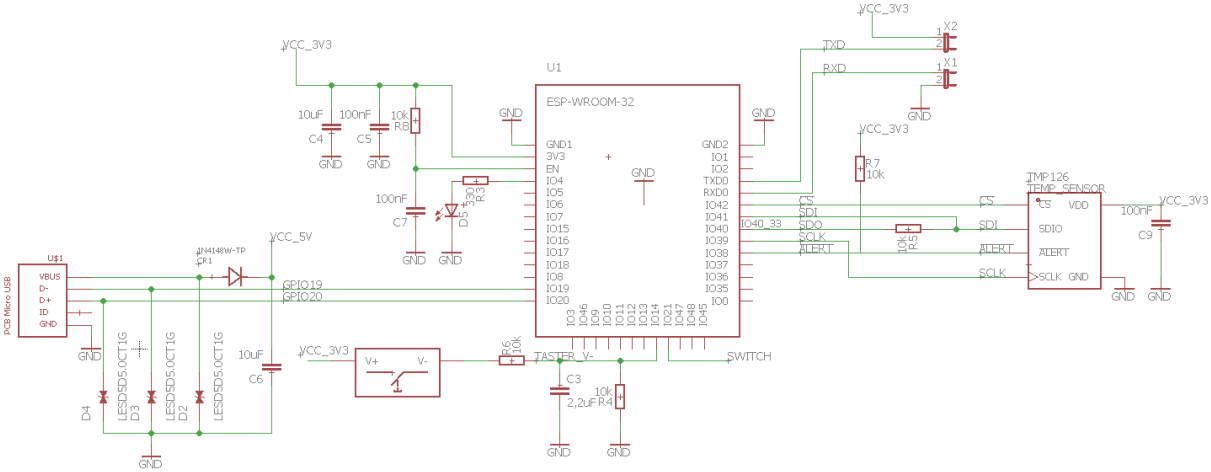
15.1.3 Stückliste

Bauteil	Bauteilbezeichnung	Anzahl
Testpins	2 polig	3
JTAG-SS	10 polig	1
Relais	V231127	2
Transistor	BSS123	2
LED	QBLP650-S2	4
Diode (Draht)	1N4148W	2
Temperatur Sensor	TMP126	1
ESP-Devkit	ESP32-S3-Wroom-1B	1
Kondensator	C-1206	2
Kondensator	C-0805	5
Widerstände	R0805	8
Bauteil	Bauteilbezeichnung	Anzahl

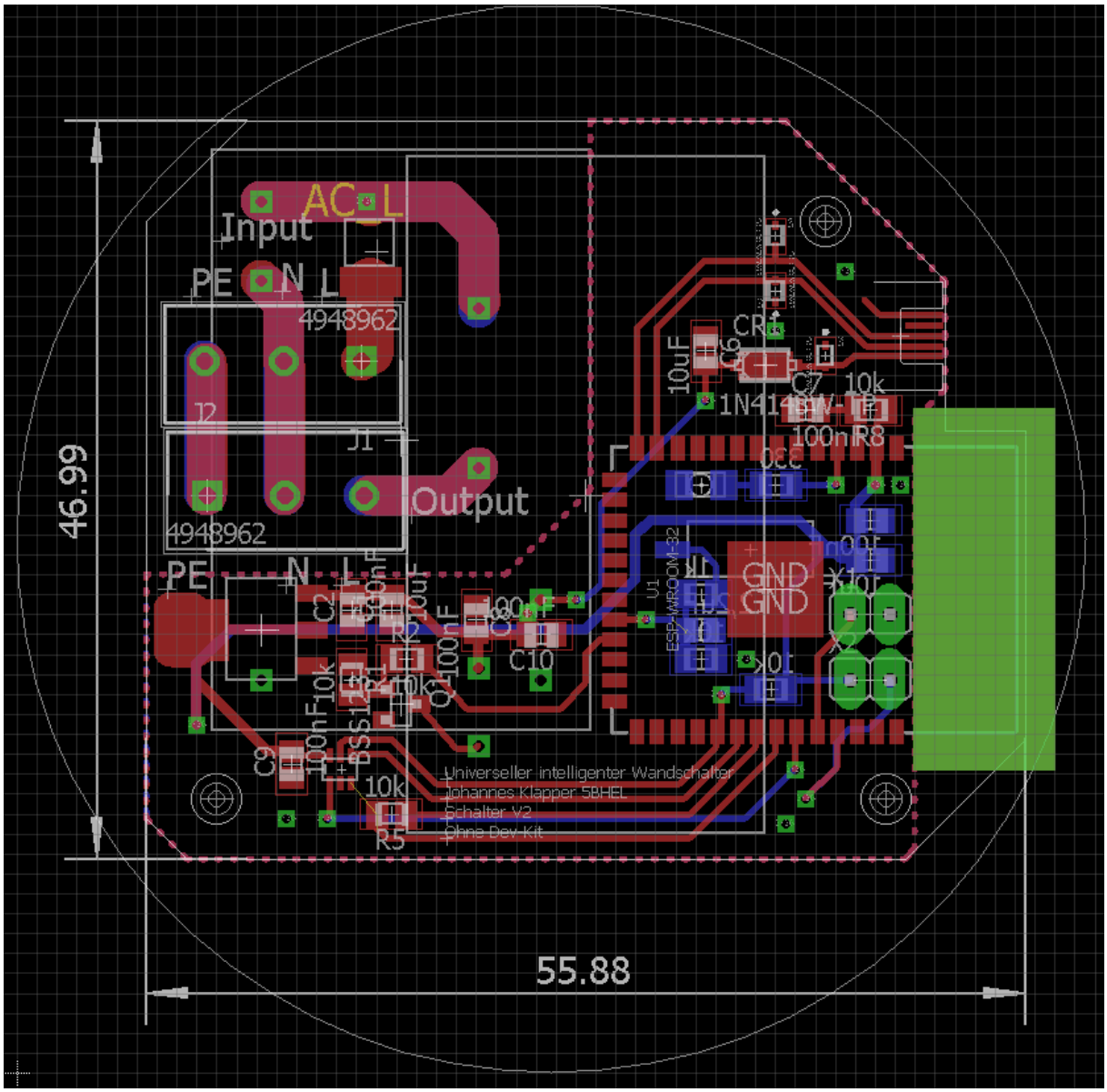
15.2 Schalter V2

15.2.1 Schaltplan





15.2.2 Layout


AC L
Input
PE N L
4948962
J2
J1
Output
4948962
PE N L
46.99
55.88
Universeller intelligenter Wandschalter
Johannes Klapper 5BHEL
Schalter V2
Ohne Dev-Kit
GND
GND
ESP-WROOM-32
10uF
100nF
10k
R5
C10

15.2.3 Stückliste

Bauteil	Bauteilbezeichnung	Anzahl
Sicherung	NBK240818-JP1021B	1
Schraubklemme	RS Stock 494-8962	1
Taster	PHAP3361	1
Solid-State	SPF240D25	1
Netzteil	AC-05-3	1
FET	BSS123	1
LED	QBLP650-S2	1
Diode	1N4148W	1
Temperatur Sensor	TMP126	1
Linearregler	SGM2212	1
USB-SS	USB3080	1
Bi-Diode	PESD5V0S1B	3
ESP	ESP32-S3-Wroom-1B	1
Kondensator	C-0805 100nF	6
Kondensator	C-0805 10uF	3
Kondensator	C-0805 2,2uF	1
Widerstände	R0805 10kOhm	5
Widerstand	R0805 1kOhm	2

15.2.4 USB-Schnittstelle

Etwas genauer gesagt handelt es sich um eine USB 2.0 Mini-A. Die „Universal Serial Bus“ Schnittstelle im (Volksmund Mikro-Usb) wird noch sehr oft eingesetzt, obwohl die Version 3.0 (Volksmund Usb-C) mit einer weitaus ausgereiften Kommunikation viele Vorteile bringt.

Es ist ein Anschluss mit 5 Pins. Die Pinbelegung lautet wie folgt: [71]

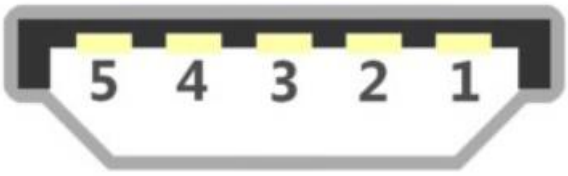


Abbildung 58: USB Pinbelegung

Pin	Name	Beschreibung	Adern Farbe
1	VBUS	5 Volt	Rot
2	D-	Datenleitungen	Weiß
3	D+	Datenleitungen	Grün
4	ID	Identifikation	kein Kabel
5	GND	Masse	Schwarz

Die Datenleitungen können mit bis zu 480 Mbit/s kommunizieren, da der ESP dies auch beherrscht kann er auch mit voller Datenrate beschrieben und ausgelesen werden.

[71] USB: Pinbelegung von USB A, B, C und Micro-USB (2022).

15.3 Bridge

15.3.1 Programmiersprache

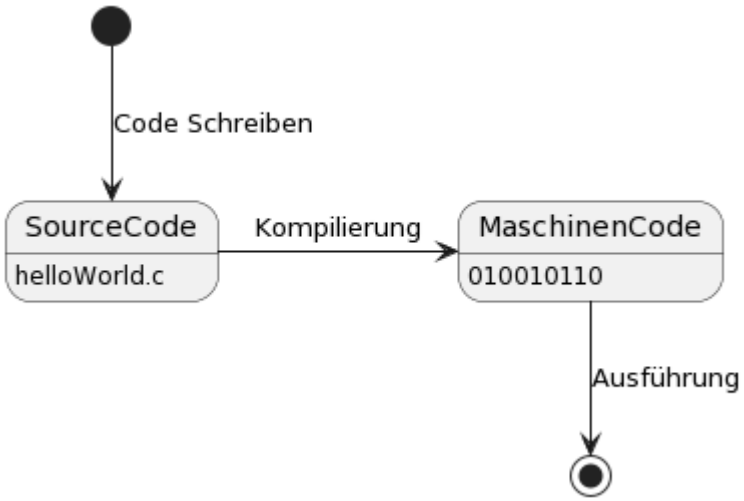
	C	C++	Javascript	Java	Pyton
Objektorientierung	Nein	Möglich	Möglich	Ja	Möglich
Meiste Verwendung	Hardware	Hardware	Web	Allzweck	Allzweck
Geschwindigkeit	Schnell	Schnell	Langsam	Moderat	Langsam
Schwierigkeitsgrad	Schwierig	Schwierig	Moderat	Einfach	Einfach
Typ	Kompiliert	Kompiliert	Interpretiert	Kompiliert	Interpretiert

Interpretiert vs. Kompiliert

Interpretierte und kompilierte Sprachen unterscheiden sich in ihrer Art der Ausführung.

Kompilierte Sprachen werden in Maschinensprache übersetzt, bevor diese ausgeführt werden. Dieser Prozess wird auch als „Kompilieren“ bezeichnet. Diese Programmiersprachen sind dadurch oft sehr schnell.[72]

Interpretierte Sprachen werden direkt ausgeführt, ohne dass diese zuerst übersetzt werden. Stattdessen wird das Programm Zeile für Zeile ausgeführt, während es gelesen (interpretiert) wird. Diese Sprachen sind oft einfacher zu schreiben, was jedoch eine geringere Geschwindigkeit zur Folge hat.[73]



[72] Vgl. Difference between Compiled and Interpreted Language (2020).
[73] Vgl. Compiler und Interpreter.

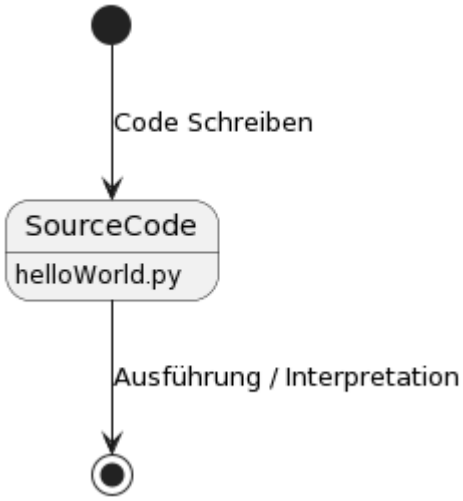
Objektorientiert vs. Funktional

Diese zwei Kategorien unterscheiden sich in ihrem Ansatz zur Lösung von Einteilungen.

Objektorientierte Sprachen sind auf die Verwendung von „Objekten“ ausgerichtet, welche einzelne Daten und Funktionen zusammenfassen. Dies sind Instanzen von „Klassen“, welche bestimmte Funktionalitäten definieren. Sprachen, welche Objektorientierung unterstützen, beinhalten oft Konzepte wie Vererbung, Polymorphie und Kapselung.[74]

Funktionale Sprachen sind auf die Verwendung von Funktionen ausgerichtet. Dies sind unabhängige Elemente, welche bestimmte Abläufe ausführen. Diese Programmiersprachen legen den Fokus auf die Verarbeitung der Daten anstatt auf die Veränderung.[75]

Zusammenfassend lässt sich sagen, dass objektorientierte Sprachen eine bessere Wiederverwendbarkeit von Codes zur Folge haben, während funktionale eine einfachere Modellierung von Prozessen und ein besseres Arbeiten mit parallelem Code ermöglichen[76]



Vergleich

Sowohl C als auch C++ sind kompilierte Sprachen, welche oft für die Programmierung von Systemen verwendet wird, welche sich auf einer niedrigen Ebene befinden. Dabei sind diese zwei Sprachen zwar sehr leistungsstark, jedoch auch sehr komplex und fordern ein gewisses Maß an Fähigkeiten und Erfahrung.[77][78]

[74] Vgl. Objektorientiertes Programmieren I - einfach erklärt!
[75] Vgl. Funktionale Programmierung: Erklärung & Beispiel (2020).
[76] Vgl. Objektorientierte, Prozedurale und Funktionale Programmierung.
[77] Vgl. Übersicht über die Programmiersprachen (2022) (2021).
[78] Vgl. W3Schools C++.

JavaScript (JS) ist eine interpretierte Skriptsprache, welche oft für Webanwendungen verwendet wird. Dabei ist diese einfacher zu erlernen als die oben genannten zwei Sprachen, jedoch auch leistungsärmer.[79]

Python ist wie JS eine interpretierte Hochlevel-Skriptsprache, auch einfach zu verwenden ist. Dabei verfügt diese über eine Vielzahl von Tools und Bibliotheken.[80]

Java ist eine objektorientierte Sprache, welche meist für die Entwicklung von Enterprise- und Android- Anwendungen verwendet wird. Ein Nachteil der Sprache ist ihre hohe Resourcenanforderung.

15.3.2 Installation der benötigten Komponenten

Für die Erstellung der Applikation werden mehrere einzelne Software-Komponenten benötigt. Dabei handelt es sich um systemabhängige „Dependencies „und Bibliotheken. In diesem Kapitel werden die Komponenten nur installiert, der Umgang und die Erklärung sind in den entsprechenden Kapitel zu finden.

Dependencies

Unter Dependencies werden Abhängigkeiten zwischen verschiedenen Software-Komponenten in einem Unix bzw. Linux-System verstanden. Wenn eine Anwendung ausgeführt werden soll, kann es sein, dass diese auf andere Elemente angewiesen ist, welche als Abhängigkeiten bezeichnet werden.[81]

Raspbian verwendet ein sogenanntes Paketverwaltungssystem. Dies ermöglicht es neben der Installation von Paketen, die Dependencies automatisch zu verwalten. Hier speziell wird das System „Advanced Package Tool“ (APT) verwendet.[82]

Mittels APT kann ein Programm in der Konsole ganz einfach installiert werden, indem man den Befehl *sudo apt-get install <Packet-Name>* verwendet:

```
sudo apt-get install bluetooth bluez libbluetooth-dev
sudo python3 -m pip install pybluez
```

Bibliotheken

Unter einer Bibliothek versteht man eine Ansammlung von Programmcode, welcher für einen bestimmten Zweck geschrieben wurde und von Entwicklern verwendet werden kann, um ihre eigene Software zu erstellen. Diese enthält normalerweise Funktionen, Klassen und andere Code-Elemente, die häufig verwendet werden, um Zeit und Aufwand zu ersparen. Die meisten

[79] Vgl. W3Schools Javascript.
[80] Vgl. Welcome to Python.org.
[81] Vgl. InstallingSoftware - Community Help Wiki.
[82] Vgl. apt › apt › Wiki › ubuntuusers.de.

Programmiersprachen enthalten bereits eine Standardbibliothek, welche diverse Grundfunktionen enthält.

Für die Programmiersprache Python wird auch ein Package-Manager namens „Pip Installs Packages“ (PIP) verwendet. Um hierbei ein Paket zu installieren, wird der Befehl *pip install <Packet-Name>* verwendet:[83]

```
pip install haikunator
```

Achtung: Je nach Version kann dieser Befehl variieren (Ex. V3 -> „pip3“)

Vereinfachung

Für eine vereinfachte Installation wird ein „Bash“-Skript geschrieben, welches bei der Ausführung alle verwendeten Komponenten installiert.

Ein Bash-Skript ist eine Textdatei, welches einzelne Unix- bzw. Linux-Befehle enthält. Es kann dabei verwendet werden, um Aufgaben automatisch auszuführen, anstatt diese manuell einzugeben.[84]

Das verwendete Bash-Skript sieht dabei wie folgend aus:

```
1   #!/bin/bash
2
3   installBluetoothApt=`sudo apt-get install bluetooth bluez libbluetooth-dev`
4   installBluetoothPIP=`sudo python3 -m pip install pybluez`
5   installHaikunator=`pip install haikunator`
6
7   echo $installBluetoothApt
8   echo $installBluetoothPIP
9   echo $installHaikunator
```

15.3.3 Entwicklungsumgebung

15.3.3.1 Verwendete Tools

Eine Entwicklungsumgebung (auch IDE genannt, ist eine integrierte Anwendung, welche Entwickler bei der Erstellung, Verwaltung und Debugging von Programmen unterstützt. Für die Bridge wird dabei der Compiler und den Text-Editor Visual Studio Code benötigt.

Visual Studio Code

VSCode ist eine plattformübergreifende und Open-Source IDE von Microsoft. Hiermit ist es möglich, mit einer großen Auswahl an Programmiersprachen zu arbeiten, darunter auch Python. Diese IDE bietet eine benutzerfreundliche grafische Oberfläche. Um die Funktionen

[83] Vgl. ThePip.
[84] Vgl. Bash-Skripting-Guide für Anfänger › Shell › Wiki › ubuntuusers.de.

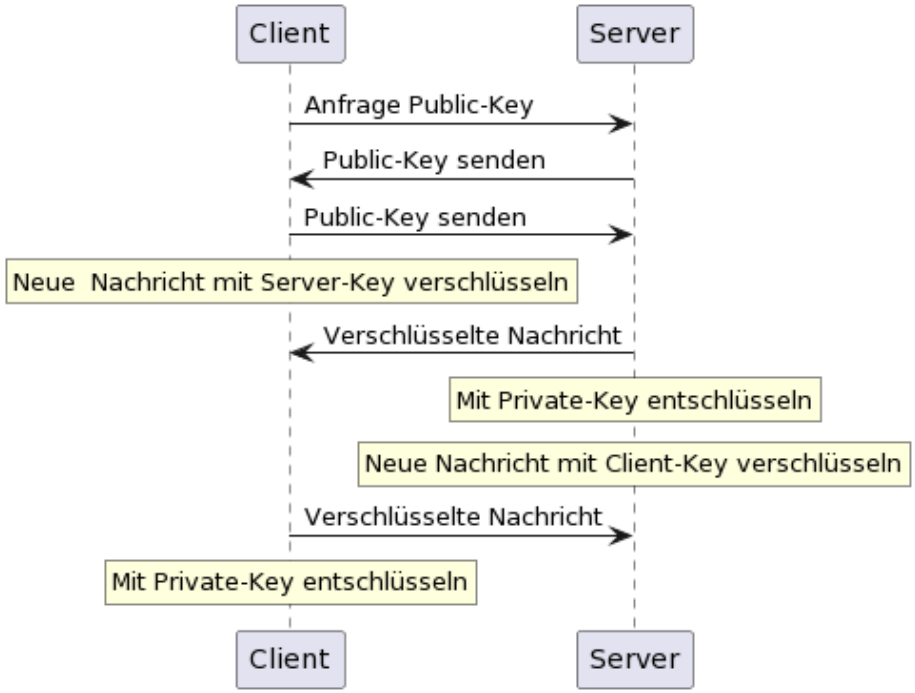
der IDE zu erweitern, werden sogenannte Plugins installiert, welche aus dem „Store“ entnommen werden können.[85]

Dieses Programm wird auf dem anderen Rechner installiert, wobei „remote“ auf die Bridge zugegriffen wird, um auf ihr zu entwickeln. Hierbei wird SSH verwendet.

SSH

Secure Shell (SSH) ist ein Netzwerkprotokoll, welches eine sichere Kommunikation zwischen zwei Geräten ermöglicht. Hierbei wird es verwendet, um von einem Rechner auf die Bridge zuzugreifen. Zur Sicherung der Verbindung kommen Authentifizierungen, als auch Verschlüsselung zum Einsatz. [86]

Zur Verschlüsselung dient ein Verfahren namens „Public/Private Key“. Dies ist ein spezielles Kryptographie-System, welches einen öffentlichen und einen privaten Schlüssel verwendet, welche jeder Teilnehmer individuell erhält. Der öffentliche Key wird frei verteilt und dient zur Verschlüsselung der Nachrichten. Der private Schlüssel wird hingegen geheim gehalten und wird für die Entschlüsselung der Nachricht verwendet. Wenn nun ein Gerät einem Anderen eine Nachricht senden will, holt dieser den öffentlichen Schlüssel des Empfängers und verschlüsselt die Nachricht. Nach Empfang der Nachricht beim anderen Gerät, wird die Nachricht nun mit dem privaten Schlüssel lesbar gemacht[87].[88]



[85] Vgl. Visual Studio Code - Code Editing. Redefined.
[86] Vgl. Lonvick, C. M./Ylonen, T. (2006); Aleksic, M. (2021).
[87] Vgl. Kryptographie - einfach erklärt (2018).
[88] Vgl. „Public Key“ oder „Private Key“: Unterschiede der Verfahren zur Daten-Verschlüsselung - PSW GROUP Blog.

Um sich zu authentifizieren, wird nun ein Passwort verwendet, welches zuerst auf der Bridge festgelegt wird.

GIT

Git ist ein freies und quelloffenes Versionskontrollsystem, das es Entwicklern ermöglicht, ihren Code und andere Dateien zu verwalten und zu verfolgen. Somit ist es möglich eine Entwicklung sehr einfach parallel auszuführen. Es ist auch jederzeit möglich, eine vorherige Version des Projekts wiederherzustellen.

Dieses Tool ist eines der meist verwendeten Systeme dieser Art und ist auch für seine Geschwindigkeit und Flexibilität bekannt.[89]

Alle Daten des Projektes werden bei Git in einem sogenannten „Repository“ abgespeichert. Dieses sollte jederzeit erreichbar sein, weshalb ein externer Service namens „GitHub“ verwendet wird, welcher diesen Dienst anbietet.[90]

Änderungen an dem Projekt können mittels „Commits“ erstellt werden. Diese speichern Datensätze an Änderungen und eine Nachricht, welches diese beschreibt.

Die parallele Entwicklung wird mit Hilfe von Zweigen („Branches“) geregelt. Der Zweig, welcher für die Produktion bereit ist, wird als „Main-“ oder „Masterbranch“ bezeichnet. Alle anderen Branches „zweigen“ von diesem ab und werden für die Entwicklung von neuen Funktionen verwendet.

Die erneute Zusammenführung der Branches mit einem anderem wird als „Merging“ bezeichnet.[91]

15.3.3.2 Einrichtung

Um die Entwicklung erst möglich zu machen, müssen die einzelnen Werkzeuge eingerichtet werden.

Git/Github

Für die Nutzung von Git muss ein globaler Nutzername, sowohl als auch eine E-Mail-Adresse angegeben werden. Hierbei werden die Daten von Git-Hub verwendet

```
git config --global user.name <user>
git config --global user.email <mail>
```

[89] Vgl. Atlassian.
[90] Vgl. universal-smart-switch/uss-bridge.
[91] Vgl. Git.

Auch wird ein spezieller Schlüssel SSH-Schlüssel von Github verwendet, welche unter den „SSH/GPG“-Einstellungen eingefügt werden kann. Diesem werden aus Sicherheitsgründen jedoch nur die benötigten Berechtigungen erteilt. Zur Erstellung wird in der Kommandozeilge der Befehl *gh auth* verwendet.

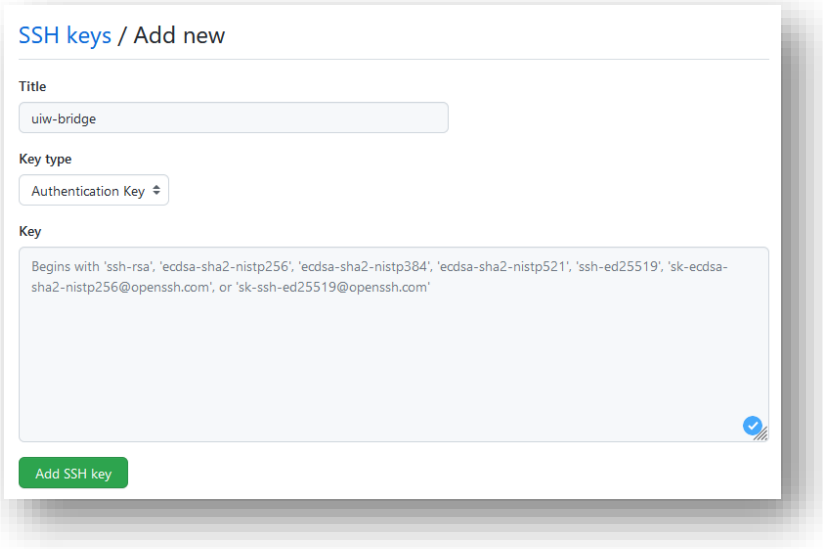


Abbildung 59: GitHub Key

VSCode

Um sich von einem Remote-Gerät mit der Bridge zu verbinden, wird der „Connect“-Button (unten links) ausgewählt. Anschließend wird die IP-Adresse der Bridge eingegeben.

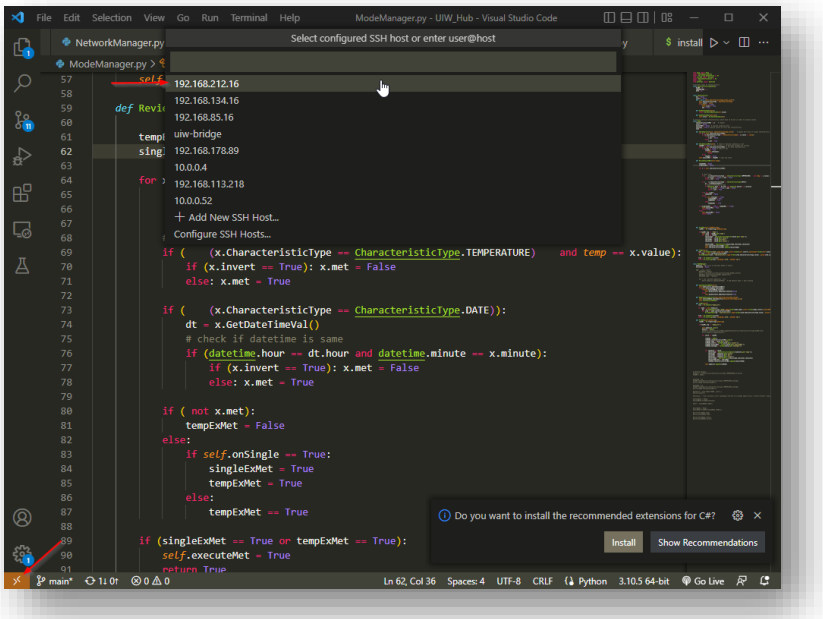


Abbildung 60: VSCode

15.3.4 Verwaltung der Einstellungen: Lösungswege

15.3.4.1 XML

Die „Extended Markup Language“ (auch XML genannt), ist eine flexible, textbasierte Markup-Sprache, die für die Datenbeschreibung und -übertragung verwendet wird. Im Gegensatz zu HTML, das für die Formatierung und Darstellung von Websites verwendet wird, ist XML eine generische Markup-Sprache, die für eine Vielzahl von Anwendungen verwendet werden kann.[92]

Ein XML-Dokument besteht aus den folgenden grundlegenden Teilen:

1. Prolog: Ein optionaler Teil, der Meta-Informationen wie die Verwendung von Zeichencodierungen oder DTDs (Document Type Definitions) enthält.
2. Wurzelelement: Dieses beschreibt das Dokument selbst und enthält alle untergeordneten Elemente.
3. Elemente: Die Hauptbausteine von XML, die Daten beschreiben. Elemente bestehen aus einem Namen, einem Inhalt und ggf. Attributen.
4. Attribute: Zusätzliche Informationen, die einem Element hinzugefügt werden, um weitere Einzelheiten über das Element bereitzustellen.
5. Inhalt: Der tatsächliche Inhalt des Elements, der aus Text, weiteren Elementen oder beidem bestehen kann.[93]

Ein XML-Dokument sollte auch eine gültige Struktur haben, bei der sich jedes Element schließt und jedes Element eindeutig definiert ist.[94]

Beispiel:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
    <year published="1925"></year>
  </book>
  <book>
    <title>To Kill a Mockingbird</title>
    <author>Harper Lee</author>
    <year published="1960"></year>
  </book>
</bookstore>
```

[92] Vgl. XML Tutorial.
[93] Vgl. XML Syntax.
[94] Vgl. Extensible Markup Language (XML) 1.0 (2006).

In diesem Beispiel enthält das Wurzelelement "bookstore" zwei untergeordnete Elemente "book". Jedes "book"-Element enthält weitere Elemente "title", "author" und "year" mit Attributen "published".

15.3.4.2 JSON

Die „JavaScript Object Notation“ (JSON) ist wie XML ein offenes Datenformat, das hauptsächlich zur Übertragung und Speicherung von Daten verwendet wird. Es wurde aus JavaScript abgeleitet und ist ein einfaches, menschenlesbares Datenformat, das jedoch eine sehr klare und strukturierte Syntax hat.

Ein JSON-Dokument besteht aus Key-Value-Paaren, die mithilfe von geschweiften Klammern `{}` gruppiert werden. Jeder Key ist ein String, während der zugehörige Value ein anderes JSON-Objekt, ein Array, ein Boolean, eine Zahl oder ein String sein kann. [95]

Beispiel:

Abgebildet ist das obige Beispiel, diesmal jedoch in JSON.

```json
{
  "name": "John Doe",
  "age": 35,
  "isMarried": true,
  "address": {
    "street": "123 Main St",
    "city": "New York",
    "state": "NY"
  },
  "hobbies": ["reading", "traveling", "cooking"]
}
```

15.3.4.3 CSV

Die dritte Möglichkeit, Daten im Klartext abzuspeichern ist das CSV-Format („Coma-Seperated Values“). In einer CSV-Datei werden die Daten in Zeilen organisiert, wobei jede Zeile einen Datensatz darstellt. Die Felder innerhalb eines Datensatzes werden durch Kommas voneinander getrennt[96].

Beispiel:

```
title, author, published
The Great Gatsby, F. Scott Fitzgerald, 1925
To Kill a Mockingbird, Harper Lee, 1960
```

[95] Vgl. JSON Introduction.
[96] Vgl. Hoffman, C. (2022).

15.3.4.4 YAML

Dieses Format unterscheidet sich von XML und JSON darin, dass diese sehr einfach zu lesen und zu schreiben ist.[97]

In YAML werden Daten durch Einrückungen organisiert, um die Hierarchie von Daten zu zeigen. Schlüssel-Wert-Paare werden durch Doppeltpunkte getrennt. Einfache Datentypen wie Zahlen und Strings können direkt ausgezeichnet werden, während komplexere Datenstrukturen wie Listen und „dictionaries“ verwendet werden können, um mehrere Werte zu speichern.[98]

Beispiel:

```yaml
- book: The Great Gatsby
  author: F. Scott Fitzgerald
  date: 1925

- book: To Kill a Mockingbird
  author: Harper Le
  date: 1960
```

15.4 Client

15.4.1 Programmiersprache

Wie auch schon für die Bridge-Applikation muss eine Programmiersprache für die Client-Software ausgewählt werden. Die einzelnen Arten und Unterteilungen dieser Sprachen sind im gleichnamigen Kapitel der Bridge zu finden. Die Client-Applikation soll mindestens auf Windows 10 ausführbar sein und zugleich auch einfach zu erstellen. Die Beschreibungen der Sprachen werden im gleichnamigen Kapitel der Bridge genauer erklärt.

	C#	C++	VB	Python
Objektorientierung	Ja	Möglich	Ja	Ja
Meiste Verwendung	Desktop	Hardware	Desktop	Allzweck
Geschwindigkeit	Moderat	Schnell	Moderat	Langsam
Schwierigkeitsgrad	Einfach	Schwierig	Einfach	Einfach
Typ	Kompiliert	Kompiliert	Kompiliert	Interpretiert

[97] Vgl. The Official YAML Web Site.
[98] Vgl. Velimirovic, A. (2020).

Vergleich

C# ist eine moderne, einfach zu erlernende Sprache, die in die .NET-Plattform integriert ist. Es bietet eine reiche Bibliothek an Tools und Bibliotheken, einschließlich einer modernen grafischen Benutzeroberfläche[99].[100]

C++ ist eine leistungsstarke Sprache, die für anspruchsvolle Anwendungen geeignet ist, einschließlich Desktop-Anwendungen. Es hat jedoch eine steile Lernkurve und eine syntaktisch komplexere Sprache als C# oder Python.[101]

VB ist eine einfach zu erlernende Sprache, die auf der .NET-Plattform basiert. Es ist eine gute Wahl für die Entwicklung von einfachen Desktop-Anwendungen mit einer einfachen grafischen Benutzeroberfläche.[102]

Python ist eine einfach zu erlernende, aber mächtige Sprache, die oft für die Erstellung von Skripten und kleinen Anwendungen verwendet wird. Es gibt auch spezielle Bibliotheken wie PyQt, die es ermöglichen, Desktop-Anwendungen mit einer einfachen und intuitiven Schnittstelle zu erstellen.[103]

Schlussendlich fällt die Entscheidung auf die Sprache C#, da diese neben den genannten Vorteilen auch von den Team-Mitgliedern bis zu einem gewissen Grad beherrscht wird.

15.4.2 Plattform

Eine Entwicklungs-Plattform ist eine umfassende Sammlung von Tools, Technologien und Bibliotheken, die Entwicklern dabei hilft, Anwendungen für eine bestimmte Plattform oder Gerätefamilie zu erstellen. Sie bietet normalerweise eine integrierte Umgebung, in der Entwickler ihre Anwendungen schreiben, debuggen, testen und bereitstellen können.

Eine gute Plattform sollte einfach zu verwenden sein, über eine breite Palette von Funktionen und Bibliotheken verfügen, die Entwicklern helfen, ihre Anwendungen zu erstellen, und eine starke Community haben, die bei Fragen und Problemen unterstützt.[104]

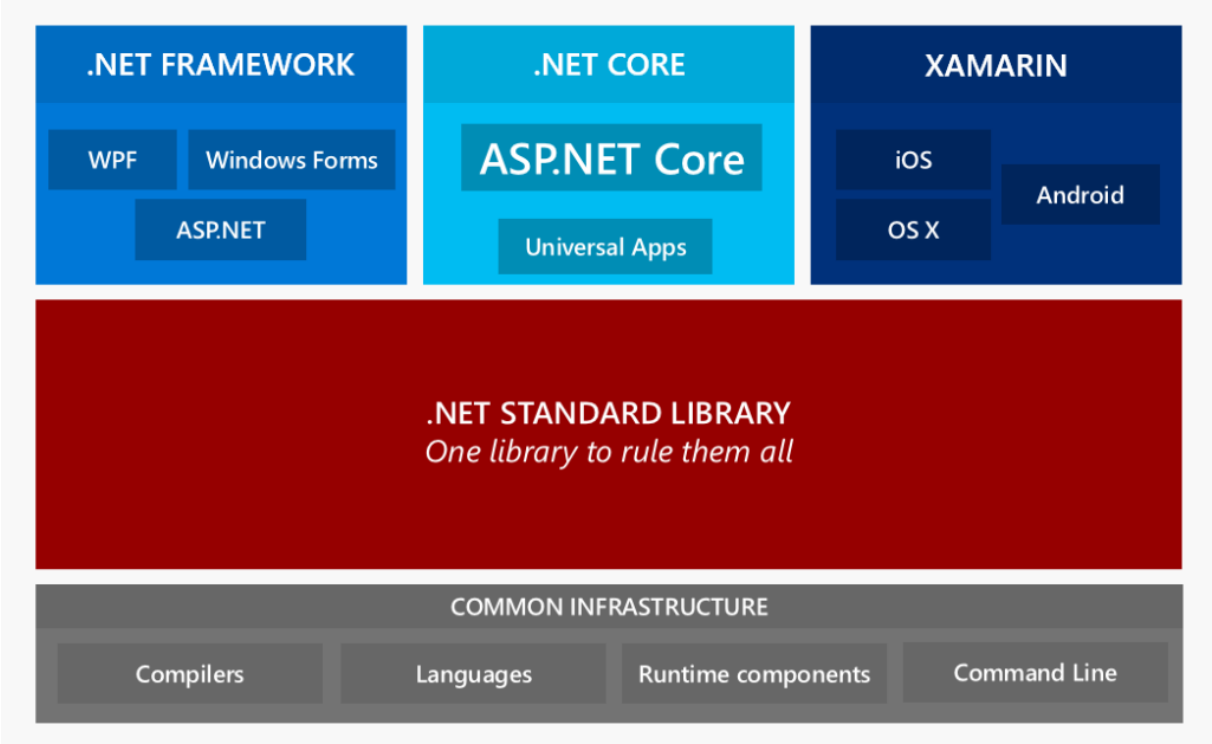
C# ist hierbei in die .NET-Plattform von Microsoft integriert. Neben diesen Sprachen unterstützt diese auch eine Vielzahl von Programmiersprachen, darunter auch die besprochene VB. Ein großer Vorteil dieser Plattform ist es, dass sie sehr viele Bibliotheken für die Entwicklung bereitstellt.[105]

[99] Vgl. What is .NET? An Overview of the Platform.
[100] Vgl. BillWagner.
[101] Vgl. BillWagner; ISO.
[102] Vgl. dotnet-bot.
[103] Vgl. Welcome to Python.org; Qt | Cross-platform Software Design and Development Tools.
[104] Vgl. CONET.
[105] Vgl. .NET Framework: Was ist das, wer braucht das? | NETZWELT.

15.4.2.1 Funktionsweise

.NET funktioniert durch die Verwendung einer Common Language Runtime (CLR), der als Herzstück des .NET-Frameworks bezeichnet wird. Diese CLR ist dabei für die sichere Ausführung von Anwendungen verantwortlich. Dabei verwaltet diese die Ressourcen, Threads und Speicher, welche von einer Anwendung verwendet werden.[106]

Wenn eine .NET-Anwendung ausgeführt wird, wird der Code von einem Compiler in die Microsoft Intermediate Language (MSIL) übersetzt, die auch als CIL (Common Intermediate Language) bezeichnet wird. Dies ist eine maschinenunabhängige Sprache, die von jedem Computer verarbeitet werden kann. MSIL-Code wird erst zur Ausführungszeit in maschinenabhängigen Code übersetzt, der auf dem Zielcomputer ausgeführt werden kann. Dies ermöglicht es, dass die Anwendungen auf einer Vielzahl von Plattformen ausgeführt werden können, ohne dass sie für jede Plattform neu kompiliert werden müssen.[107]



[108]

Abbildung 61: .NET Layers

[106] Vgl. .NET | Free. Cross-platform. Open Source.
[107] Vgl. CIL or MSIL | Microsoft Intermediate Language or Common Intermediate Language (2019).
[108] Abbildung: Strahl, C. M., EPS Software Corp ,. <a href="/People/Bio/Rick Strahl">Rick.

15.4.2.2 App-Modelle

Um eine Anwendung mittels .NET zu entwickeln, bedarf es jedoch noch an einem App-Model. Diese befinden über den oben genannten Schichten und sind verschiedene Implementierungen von .NET. Sie sind auch in der obigen Abbildung zu sehen.

Vergleich

Das .NET-Framework ist die ursprüngliche Implementierung des .NET-Ökosystems und wurde für den Einsatz auf Windows-Systemen entwickelt.[109]

.NET Core ist eine moderne, offene und modular aufgebaute Implementierung des .NET-Ökosystems, die für den Einsatz auf einer Vielzahl von Plattformen, einschließlich Windows, Linux und MacOS, entwickelt wurde. .NET Core unterstützt eine kleinere Auswahl an Programmiersprachen als das .NET-Framework, aber es ist schneller, leichter und skalierbarer.

Xamarin ist ebenso Implementierung des .NET-Systems, die speziell für die Entwicklung mobiler Anwendungen für iOS, Android und Windows entwickelt wurde. Es ermöglicht Entwicklern, Anwendungen in C# zu schreiben und diese auf mehreren Plattformen zu veröffentlichen, wodurch die Zeit und Kosten für die Entwicklung mobiler Anwendungen erheblich reduziert werden.[110][111]

Fazit

Durch die mögliche Unterstützung mehrerer Plattformen fällt der erste Entschluss auf Xamarin. Jedoch wird die Implementierung im Verlauf des Projektes aufgrund von Problemen gewechselt, welche im entsprechenden Kapitel zu finden sind.

15.4.3 Installation der benötigten Komponenten

Die Entwicklung des Clients erfordert die Installation von mehreren wichtigen Komponenten, wie beispielsweise die Plattform oder die IDE.

.**NET**

Die Plattform kann von der offiziellen Website heruntergeladen und installiert werden. Wichtig dabei ist, dass die Prozessor-Architektur x86 ausgewählt wird. Diese beschreibt die grundlegenden technischen Merkmale und Funktionen eines Computerprozessors, welcher einer der wichtigsten Komponenten eines Computers darstellt und verantwortlich für die Ausführung von Anweisungen aus Programmen und der Verarbeitung von Daten ist.

Visual Studio

[109] Vgl. Was ist .NET Framework? – einfach erklärt (2022).
[110] Vgl. Was ist Xamarin? - Xamarin | Microsoft Learn.
[111] Vgl. Plattformübergreifende mobile App-Entwicklung mit Xamarin (2019).

Visual Studio ist eine IDE von Microsoft, die für die Entwicklung von Anwendungen auf verschiedenen Plattformen wie Windows, MacOS und Linux verwendet wird. Es unterstützt neben C# auch eine Vielzahl von Programmiersprachen, darunter C++, Visual Basic, Python, F# und viele andere.

Mithilfe dieser Umgebung können Entwickler ihre Projekte schnell und einfach organisieren, indem sie Code schreiben, Fehler beheben, Tests ausführen und Anwendungen bereitstellen. Es bietet auch viele Tools, um die Entwicklung zu beschleunigen, einschließlich einer visuellen Oberfläche für die Entwicklung von Benutzeroberflächen, Debugging-Tools, einer Codebibliothek und vielem mehr.

Wie auch .NET kann diese von der offiziellen Website installiert werden. Wichtig bei dieser ist jedoch, dass die benötigten Komponenten ausgewählt werden. Unter dem Reiter „Workloads“ können diese selektiert werden.[112]

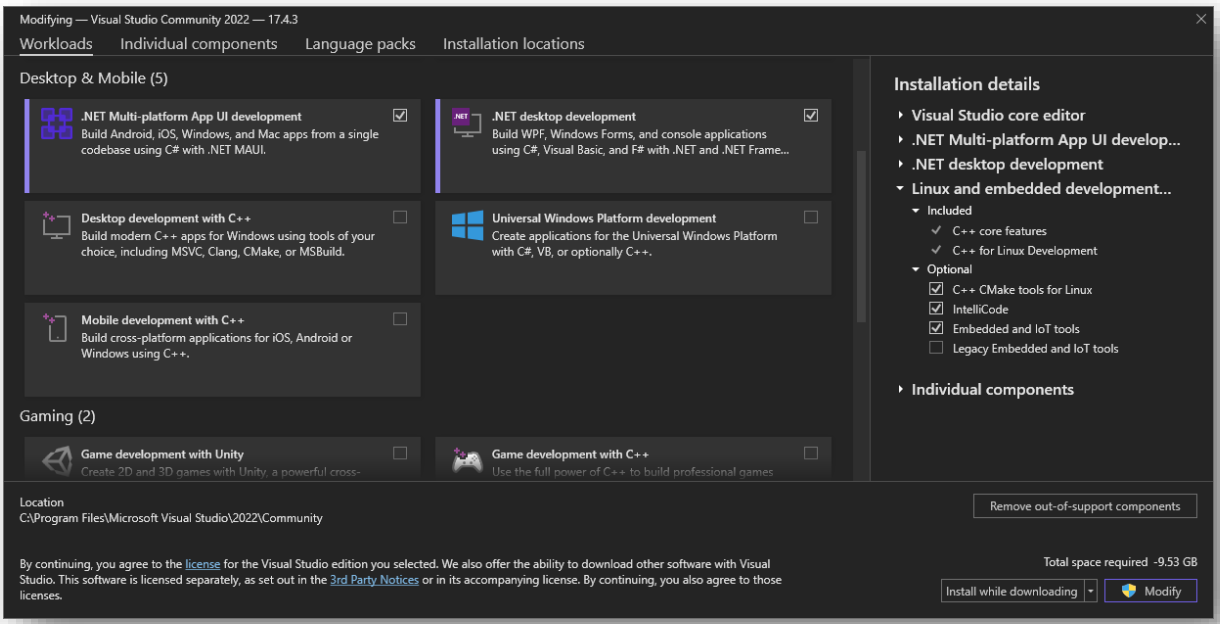


Abbildung 62: Visual-Studio Installer

GitHub-Destkop

Wie auch bei der Bridge wird zur Versionsverwaltung das Werkzeug git verwendet. Für Windows gibt es dabei eine nützliche Software von Github, welche diese Arbeitsweise deutlich erleichtert.

[112] Vgl. Visual Studio: IDE and Code Editor for Software Developers and Teams.

Die Anwendung unterstützt auch die Verwaltung von Zweigen, die Überwachung von Konflikten bei der Zusammenarbeit an Projekten und die Integration mit anderen Tools wie Visual Studio Code.[113]

Innerhalb der Anwendung kann das Repository der Clients per Button gecloned werden.

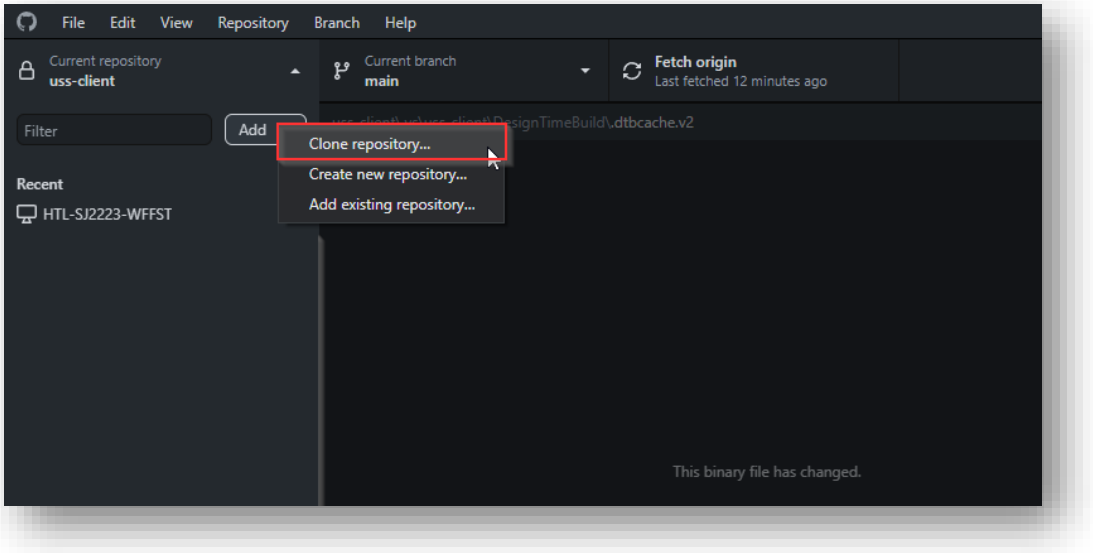


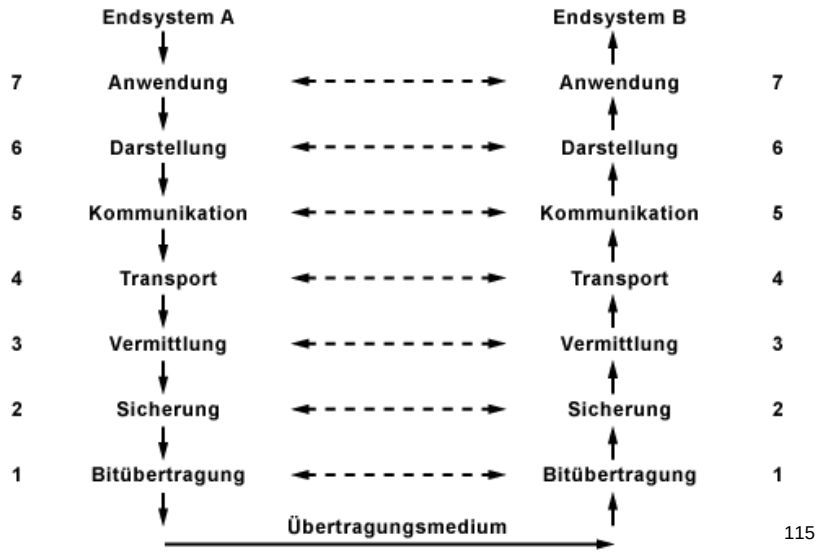
Abbildung 63: GitHub-Desktop

[113] Vgl. GitHub Desktop.

15.5 Kommunikation

15.5.1 Aufteilung des OSI-Modells

Das OSI-Modell ist in 7 aufeinanderfolgende Schichten gegliedert, welche jeweils einen eng definierten Aufgabenbereich zugeteilt sind.[114]



[115]

Abbildung 64: OSI-Modell

Schicht 1: Bitübertragungsschicht

Dieses Abteil stellt eine elektrische oder mechanische Schnittstelle zum Medium der Übertragung dar. Die Protokolle auf dieser Schicht unterscheiden sich lediglich durch die Art des Mediums selbst[116][117]

Schicht 2: Sicherungsschicht

Diese Schicht (auch Data Link Layer genannt) ist zuständig für eine zuverlässige und funktionierende Verbindung zwischen dem Übertragungsmedium und dem Endgerät. Zudem enthält dieses Abteil auch Methoden zur Fehlerbehebung und zur Datenflusskontrolle. [118]

Schicht 3: Vermittlungsschicht

Die Vermittlungsschicht ist verantwortlich, für die logisch und zeitlich getrennte Kommunikation zwischen den Teilnehmern. Dies ist das erste Abteil, auf welchem eine logische Adressierung der Geräte erfolgt.[119]

[114] Vgl. OSI Model Reference Chart.
[115] Abbildung: OSI-Schichtenmodell.
[116] Vgl. OSI-Schichtenmodell.
[117] Vgl. Physical Layer.
[118] Vgl. Data Link Layer.
[119] Vgl. Network Layer.

Schicht 4: Transportschicht

Hierbei werden die einzelnen Pakete einer Anwendung zugeteilt. Diese Schicht dient somit als Übersetzer zwischen den anwendungs- und transportorientierten Schichten.[120]

Schicht 5: Kommunikationsschicht

Dieses Fach organisiert die Verbindung zwischen den Systemen mittels Kontroll- und Steuerungsmechanismen.

Schicht 6: Darstellungsschicht

Dieses Abteil stellt die Daten in unterschiedlichen Formaten dar, welche vom Nutzer verstanden werden können.

Schicht 7: Anwendungsschicht

Diese Ebene stellt die Daten endgültig für den Anwender dar. Auf der Anwendungsschicht findet auch die Datenein- und Ausgabe statt.[121]

15.5.1.1 Speziell: Transmission Control Protocol

Das Transmission Control Protocol (TCP) ist ein zustandsorientiertes Netzwerkprotokoll, das für die Übertragung von Daten in Computernetzen verwendet wird. [122]

TCP arbeitet auf der Transportschicht des OSI-Modells und ist für die Übertragung von Datenströmen verantwortlich. Es sorgt dabei für eine Verbindungsaufnahme, Übertragung und Beendigung von Datenströmen, Überprüfung auf Fehler und dessen Korrektur.[123][124]

Der entscheidende Vorteil von TCP ist, dass Daten, die von einer Anwendung gesendet werden, vollständig und in der korrekten Reihenfolge beim Empfänger ankommen. Es ist ein zuverlässiges Protokoll und eignet sich daher für die Übertragung von wichtigen Daten, bei denen eine zuverlässige Übertragung erforderlich ist, wie beispielsweise für Webseiten, E-Mail oder Dateiübertragungen.[125]

[120] Vgl. Transport Layer.
[121] Vgl. Application Layer.
[122] Vgl. Was ist TCP (Transmission Control Protocol)? - Definition von WhatIs.com.
[123] Vgl. TCP - Transmission Control Protocol.
[124] Vgl. Protocol Suites (TCP/IP).
[125] Vgl. TCP (Transmission Control Protocol) – das Transportprotokoll im Porträt (2020).

16 QUELLENVERZEICHNIS

Adam, B. (2020): Die fünf Sicherheitsregeln der Elektrotechnik, WEKA Media - Der Fachverlag für Ihren beruflichen Erfolg, in: https://www.weka.de/elektrosicherheit/die-fuenf-sicherheitsregeln-der-elektrotechnik/ (Zugriff am 29.3.2023).

Afaneh, M. (2020): How Bluetooth Low Energy Works: Advertisements (Part 1), Novel Bits, in: https://novelbits.io/bluetooth-low-energy-advertisements-part-1/ (Zugriff am 15.3.2023).

Aleksic, M. (2021): What Is SSH (Secure Shell) And How Does It Work?, Knowledge Base by phoenixNAP, in: https://phoenixnap.com/kb/what-is-ssh (Zugriff am 9.2.2023).

Asana: Projektmanagement-Methoden im Überblick: 13 Modelle im Vergleich • Asana, Asana, in: https://asana.com/de/resources/project-management-methodologies (Zugriff am 13.3.2023).

Atlassian: Was ist Git? | Atlassian Git Tutorial, Atlassian, in: https://www.atlassian.com/de/git/tutorials/what-is-git (Zugriff am 9.2.2023).

Atrox (2023): HaikunatorPY, .

Augsten, S. (2022): Was bedeutet MVVM?, in: https://www.dev-insider.de/was-bedeutet-mvvm-a-1103448/ (Zugriff am 16.2.2023).

bijington (2022): .NET MAUI Community Toolkit documentation - .NET Community Toolkit, in: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/ (Zugriff am 16.2.2023).

BillWagner: C#-Dokumentation: Einstieg, Tutorials, Referenz., in: https://learn.microsoft.com/de-de/dotnet/csharp/ (Zugriff am 11.2.2023).

BillWagner (2023a): try-catch - C# Reference, in: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/try-catch (Zugriff am 16.2.2023).

BillWagner (2023b): if- und switch-Anweisungen: Wählen Sie den Ausführungspfad zwischen Denkverzweigungen aus., in: https://learn.microsoft.com/de-de/dotnet/csharp/language-reference/statements/selection-statements (Zugriff am 17.2.2023).

CONET: Microsoft .NET, CONET | IT-Systemhaus & IT-Beratung, in: https://www.conet.de/DE/loesungen/microsoft/microsoft-net (Zugriff am 11.2.2023).

davidbritch (2023a): Was ist .NET MAUI? - .NET MAUI, in: https://learn.microsoft.com/de-de/dotnet/maui/what-is-maui (Zugriff am 16.2.2023).

davidbritch (2023b): What is .NET MAUI? - .NET MAUI, in: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui (Zugriff am 17.2.2023).

dotnet-bot: Visual Basic docs - get started, tutorials, reference., in: https://learn.microsoft.com/en-us/dotnet/visual-basic/ (Zugriff am 11.2.2023a).

dotnet-bot: Task Class (System.Threading.Tasks), in: https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.task?view=net-7.0 (Zugriff am 16.2.2023b).

Ewald, W. (2020): Spannungsversorgung - Linear- und Schaltregler • Wolles Elektronikkiste, Wolles Elektronikkiste, in: https://wolles-elektronikkiste.de/spannungsversorgung-linear-und-schaltregler (Zugriff am 18.3.2023).

Feli (2021): Lichtschalter und Steckdosen: Höhe und Maße, Talu.de, in: https://www.talu.de/lichtschalter-und-steckdosen/ (Zugriff am 19.3.2023).

Glaser, O. (2012): Answer to „Why does micro USB 2.0 have 5 pins, when the A-type only has 4?“, Electrical Engineering Stack Exchange, in: https://electronics.stackexchange.com/a/35468 (Zugriff am 19.3.2023).

Helmut (2021): Die ESP32-Evolution: S2, S3, C3, Arduino-Hannover, in: https://arduino-hannover.de/2021/12/09/die-esp32-evolution-s2-s3-c3/ (Zugriff am 18.3.2023).

Hoffman, C. (2022): What Is a CSV File, and How Do I Open It?, How-To Geek, in: https://www.howtogeek.com/348960/what-is-a-csv-file-and-how-do-i-open-it/ (Zugriff am 9.2.2023).

ISO: ISO/IEC 14882:2020, ISO, in: https://www.iso.org/standard/79358.html (Zugriff am 11.2.2023).

jwmsft (2022): Übersicht über XAML - UWP applications, in: https://learn.microsoft.com/de-de/windows/uwp/xaml-platform/xaml-overview (Zugriff am 22.2.2023).

Lonvick, C. M./Ylonen, T. (2006): The Secure Shell (SSH) Protocol Architecture, .

Ltd, R. P.: Buy a Raspberry Pi Zero, Raspberry Pi, in: https://www.raspberrypi.com/products/raspberry-pi-zero/ (Zugriff am 9.2.2023a).

Ltd, R. P.: Buy a Raspberry Pi 4 Model B, Raspberry Pi, in: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (Zugriff am 9.2.2023b).

Ltd Raspberry-Pi: Raspberry Pi OS, Raspberry Pi, in: https://www.raspberrypi.com/software/ (Zugriff am 9.2.2023).

michaelstonis (2022): Model-View-ViewModel, in: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm (Zugriff am 16.2.2023).

moicapnhap: Was ist der unterschied von c++ und arduino ide, moicapnhap, in: https://de.moicapnhap.com/post/was-ist-der-unterschied-von-c-und-arduino-ide (Zugriff am 19.3.2023).

O.V. (2006): Extensible Markup Language (XML) 1.0, in: https://web.archive.org/web/20060615212726/http://www.w3.org/TR/1998/REC-xml-19980210 (Zugriff am 9.2.2023).

O.V. (2017): Intro zu Bluetooth Stromverbrauch | Bluetooth® Technologie Website, Website zur Bluetooth®-Technologie, in: https://www.bluetooth.com/de/bluetooth-resources/intro-to-bluetooth-power-consumption/ (Zugriff am 29.3.2023).

O.V. (2018): Kryptographie - einfach erklärt, in: https://praxistipps.chip.de/kryptographie-einfach-erklaert_102083 (Zugriff am 22.2.2023).

O.V. (2019): CIL or MSIL | Microsoft Intermediate Language or Common Intermediate Language, GeeksforGeeks, in: https://www.geeksforgeeks.org/cil-or-msil-microsoft-intermediate-language-or-common-intermediate-language/ (Zugriff am 11.2.2023).

O.V. (2019): Lesson 2 – BLE profiles, services, characteristics, device roles and network topology, Embedded Centric, in: https://embeddedcentric.com/lesson-2-ble-profiles-services-characteristics-device-roles-and-network-topology/ (Zugriff am 15.3.2023).

O.V. (2019): Plattformübergreifende mobile App-Entwicklung mit Xamarin, Marsner Technologies, in: https://marsner.com/de/blog/plattformubergreifende-mobile-app-entwicklung-mit-xamarin/ (Zugriff am 22.2.2023).

O.V. (2020): Funktionale Programmierung: Erklärung & Beispiel, IONOS Digital Guide, in: https://www.ionos.de/digitalguide/websites/web-entwicklung/funktionale-programmierung/ (Zugriff am 9.2.2023).

O.V. (2020): Difference between Compiled and Interpreted Language, GeeksforGeeks, in: https://www.geeksforgeeks.org/difference-between-compiled-and-interpreted-language/ (Zugriff am 9.2.2023).

O.V. (2020): TCP (Transmission Control Protocol) – das Transportprotokoll im Porträt, IONOS Digital Guide, in: https://www.ionos.de/digitalguide/server/knowhow/tcp-vorgestellt/ (Zugriff am 9.2.2023).

O.V. (2020): Design Patterns – schneller und sicherer programmieren, IONOS Digital Guide, in: https://www.ionos.de/digitalguide/websites/web-entwicklung/was-sind-design-patterns/ (Zugriff am 16.2.2023).

O.V. (2020): Factory Pattern: Alle Informationen zum Factory Method Pattern, IONOS Digital Guide, in: https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-das-factory-pattern/ (Zugriff am 16.2.2023).

O.V. (2021): Was ist eine statische Funktion in C? | Referenz, in: https://juttadolle.com/was-ist-eine-statische-funktion-in-c/ (Zugriff am 22.2.2023).

O.V. (2021): Übersicht über die Programmiersprachen (2022), in: https://lerneprogrammieren.de/uebersicht-ueber-die-programmiersprachen/ (Zugriff am 9.2.2023).

O.V. (2021): The Model View Controller Pattern – MVC Architecture and Frameworks Explained, freeCodeCamp.org, in: https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/ (Zugriff am 16.2.2023).

O.V. (2021): Eine Einführung in Mini-USB: Definition, Funktionen und Verwendung, MiniTool, in: https://de.minitool.com/bib/mini-usb.html (Zugriff am 18.3.2023).

O.V. (2022): Was ist .NET Framework? – einfach erklärt, GIGA, in: https://www.giga.de/downloads/windows-10/specials/was-ist-net-framework-einfach-erklaert/ (Zugriff am 11.2.2023).

O.V. (2022): USB: Pinbelegung von USB A, B, C und Micro-USB, GIGA, in: https://www.giga.de/tipp/usb-pinbelegung-a-b-c-micro/ (Zugriff am 19.3.2023).

O.V.: Das GNU-System und Linux - GNU-Projekt - Free Software Foundation, in: https://www.gnu.org/gnu/linux-and-gnu.de.html (Zugriff am 9.2.2023).

O.V.: FrontPage - Raspbian, in: https://www.raspbian.org/ (Zugriff am 9.2.2023).

O.V.: Compiler und Interpreter, in: https://www.elektronik-kompendium.de/sites/com/1705231.htm (Zugriff am 9.2.2023).

O.V.: Objektorientiertes Programmieren I - einfach erklärt!, Studyflix, in: https://studyflix.de/informatik/objektorientiertes-programmieren-i-423 (Zugriff am 9.2.2023).

O.V.: W3Schools C++, in: https://www.w3schools.com/cpp/default.asp (Zugriff am 9.2.2023).

O.V.: W3Schools Javascript, in: https://www.w3schools.com/js/DEFAULT.asp (Zugriff am 9.2.2023).

O.V.: Welcome to Python.org, in: https://www.python.org/ (Zugriff am 9.2.2023).

O.V.: InstallingSoftware - Community Help Wiki, in: https://help.ubuntu.com/community/InstallingSoftware#Package_Dependencies (Zugriff am 9.2.2023).

O.V.: apt › apt › Wiki › ubuntuusers.de, in: https://wiki.ubuntuusers.de/apt/apt/ (Zugriff am 9.2.2023).

O.V.: Bash-Skripting-Guide für Anfänger › Shell › Wiki › ubuntuusers.de, in: https://wiki.ubuntuusers.de/Shell/Bash-Skripting-Guide_f%C3%BCr_Anf%C3%A4nger/ (Zugriff am 9.2.2023).

O.V.: Visual Studio Code - Code Editing. Redefined, in: https://code.visualstudio.com/ (Zugriff am 9.2.2023).

O.V.: „Public Key“ oder „Private Key“: Unterschiede der Verfahren zur Daten-Verschlüsselung - PSW GROUP Blog, in: https://www.psw-group.de/blog/%E2%80%9Epublic-key%E2%80%9C-oder-%E2%80%9Eprivate-key%E2%80%9C-unterschiede-der-verfahren-zur-daten-verschlusselung/591 (Zugriff am 9.2.2023).

O.V.: universal-smart-switch/uss-bridge, GitHub, in: https://github.com/universal-smart-switch/uss-bridge (Zugriff am 9.2.2023).

O.V.: Git, in: https://git-scm.com/ (Zugriff am 9.2.2023).

O.V.: XML Tutorial, in: https://www.w3schools.com/xml/ (Zugriff am 9.2.2023).

O.V.: JSON Introduction, in: https://www.w3schools.com/js/js_json_intro.asp (Zugriff am 9.2.2023).

O.V.: The Official YAML Web Site, in: https://yaml.org/ (Zugriff am 9.2.2023).

O.V.: xml.etree.ElementTree — The ElementTree XML API, Python documentation, in: https://docs.python.org/3/library/xml.etree.elementtree.html (Zugriff am 9.2.2023).

O.V.: OSI Model Reference Chart, in: https://learningnetwork.cisco.com/s/article/osi-model-reference-chart (Zugriff am 9.2.2023).

O.V.: Physical Layer, in: https://contenthub.netacad.com/itn-dl/4.0.1 (Zugriff am 9.2.2023).

O.V.: Data Link Layer, in: https://contenthub.netacad.com/itn-dl/6.0.1 (Zugriff am 9.2.2023).

O.V.: Network Layer, in: https://contenthub.netacad.com/itn-dl/8.0.1 (Zugriff am 9.2.2023).

O.V.: Transport Layer, in: https://contenthub.netacad.com/itn-dl/14.0.1 (Zugriff am 9.2.2023).

O.V.: Application Layer, in: https://contenthub.netacad.com/itn-dl/15.0.1 (Zugriff am 9.2.2023).

O.V.: TCP - Transmission Control Protocol, in: https://www.elektronik-kompendium.de/sites/net/0812271.htm (Zugriff am 9.2.2023).

O.V.: Protocol Suites (TCP/IP), in: https://contenthub.netacad.com/itn-dl/3.3.3 (Zugriff am 9.2.2023).

O.V.: Objektorientierte, Prozedurale und Funktionale Programmierung, in: https://dev-supp.de/programmierung/objektorientierte-programmierung-vs-prozedurale-programmierung-vs-funktionale-programmierung (Zugriff am 10.2.2023).

O.V.: XML Syntax, in: https://www.w3schools.com/XML/xml_syntax.asp (Zugriff am 10.2.2023).

O.V.: Qt | Cross-platform Software Design and Development Tools, in: https://www.qt.io (Zugriff am 11.2.2023).

O.V.: .NET Framework: Was ist das, wer braucht das? | NETZWELT, in: https://www.netzwelt.de/news/153991-net-framework-braucht-.html (Zugriff am 11.2.2023).

O.V.: .NET | Free. Cross-platform. Open Source., Microsoft, in: https://dotnet.microsoft.com/en-us/ (Zugriff am 11.2.2023).

O.V.: Was ist Xamarin? - Xamarin | Microsoft Learn, in: https://learn.microsoft.com/de-de/xamarin/get-started/what-is-xamarin (Zugriff am 11.2.2023).

O.V.: Visual Studio: IDE and Code Editor for Software Developers and Teams, in: https://visualstudio.microsoft.com/ (Zugriff am 11.2.2023).

O.V.: GitHub Desktop, GitHub Desktop, in: https://desktop.github.com/ (Zugriff am 11.2.2023).

O.V.: Singleton Design Pattern: Das Singleton-Entwurfsmuster kurz erklärt - IONOS, in: https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-das-singleton-pattern/ (Zugriff am 16.2.2023).

O.V.: Asynchronous programming - C# | Microsoft Learn, in: https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/async-scenarios (Zugriff am 16.2.2023).

O.V.: Windows Presentation Foundation | WPF und .NET, Visual Studio, in: https://visualstudio.microsoft.com/de/vs/features/wpf/ (Zugriff am 16.2.2023).

O.V.: What is WPF? - The complete WPF tutorial, in: https://wpf-tutorial.com/about-wpf/what-is-wpf/ (Zugriff am 16.2.2023).

O.V.: .NET MAUI: Paradiesische App-Entwicklung, in: https://entwickler.de/dotnet/dotnet-maui-cross-plattform-framework (Zugriff am 17.2.2023).

O.V.: Working with App.xaml - The complete WPF tutorial, in: https://wpf-tutorial.com/wpf-application/working-with-app-xaml/ (Zugriff am 17.2.2023).

O.V.: What is .NET? An Overview of the Platform, Auth0 - Blog, in: https://auth0.com/blog/what-is-dotnet-platform-overview/ (Zugriff am 22.2.2023).

O.V.: Was ist .Net? – Erläuterung zu Dotnet – AWS, Amazon Web Services, Inc., in: https://aws.amazon.com/de/what-is/net/ (Zugriff am 22.2.2023).

O.V.: Was ist TCP (Transmission Control Protocol)? - Definition von WhatIs.com, ComputerWeekly.de, in: https://www.computerweekly.com/de/definition/TCP-Transmission-Control-Protocol (Zugriff am 22.2.2023).

O.V. (2023): ArduinoBLE, .

O.V.: about-ble-server-profile-20-1024.jpg (1024×768), in: https://image.slidesharecdn.com/nordicframeworkserver-131016111555-phpapp02/95/about-ble-server-profile-20-1024.jpg?cb=1381922171 (Zugriff am 15.3.2023).

O.V.: Introduction to Bluetooth Low Energy, Adafruit Learning System, in: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt (Zugriff am 15.3.2023).

O.V.: Galvanische Trennung - was ist das?, in: https://www.elektrikervergleich.ch/ratgeber/galvanische-trennung-was-ist-das-c:225500 (Zugriff am 16.3.2023).

O.V. (2023): ESP-IDF VS Code Extension, .

O.V. (2023): Arduino_ESP32_OTA, .

O.V.: ESP8266 – Mikrocontroller.net, in: https://www.mikrocontroller.net/articles/ESP8266 (Zugriff am 18.3.2023).

O.V.: ESP32-S3-DevKitC-1 v1.1 - ESP32-S3 - — ESP-IDF Programming Guide latest documentation, in: https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/hw-reference/esp32s3/user-guide-devkitc-1.html (Zugriff am 18.3.2023).

O.V.: Relais • Was ist ein Relais? Wie funktioniert ein Relais?, Studyflix, in: https://studyflix.de/elektrotechnik/relais-5057 (Zugriff am 18.3.2023).

O.V.: OSI-Schichtenmodell, in: https://www.elektronik-kompendium.de/sites/kom/0301201.htm (Zugriff am 18.3.2023).

O.V.: SPI - Arduino Reference, in: https://www.arduino.cc/reference/en/language/functions/communication/spi/ (Zugriff am 19.3.2023).

O.V.: JTAG Debugging - ESP32-S3 - — ESP-IDF Programming Guide latest documentation, in: https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/api-guides/jtag-debugging/index.html (Zugriff am 19.3.2023).

O.V.: Get Started - ESP32 - — ESP-IDF Programming Guide latest documentation, in: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html (Zugriff am 19.3.2023).

O.V.: IoT Development Framework I Espressif Systems, in: https://www.espressif.com/en/products/sdks/esp-idf (Zugriff am 19.3.2023).

O.V.: Downloading and installing the Arduino IDE 2.0 | Arduino Documentation, in: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started/ide-v2-downloading-and-installing (Zugriff am 19.3.2023).

O.V.: Elektrotechnische Normung in Österreich, in: https://www.ove.at/ove-standardization/elektrotechnische-normung-oesterreich/ (Zugriff am 27.3.2023).

O.V.: WiPy 3.0, in: https://docs.pycom.io/datasheets/development/wipy3/ (Zugriff am 27.3.2023).

O.V.: TMP126NDCKR Texas Instruments | Mouser, Mouser Electronics, in: https://www.mouser.at/ProductDetail/595-TMP126NDCKR (Zugriff am 27.3.2023).

O.V.: 3-1393215-5 . - Leistungsrelais, SPDT, 5 VDC, 8 A, V23057, Durchsteckmontage, in: https://at.farnell.com/schrack-te-connectivity/3-1393215-5/leistungsrelais-spdt-5vdc-8a-tht/dp/2974811 (Zugriff am 27.3.2023).

O.V.: Sicheres und effizientes Schalten von Strom oder Spannung mit Hilfe von Halbleiterrelais, Digi-Key Electronics, in: https://www.digikey.at/de/articles/how-to-safely-and-efficiently-switch-current-or-voltage-using-ssrs (Zugriff am 28.3.2023).

O.V.: Arduino & Serial Peripheral Interface (SPI) | Arduino Documentation, in: https://docs.arduino.cc/learn/communication/spi (Zugriff am 29.3.2023).

O.V.: Online UUID Generator Tool, in: https://www.uuidgenerator.net/ (Zugriff am 29.3.2023).

Ramel, B. D./09/29/2022: Did .NET MAUI Ship Too Soon? Devs Sound Off on „Massive Mistake“ -, Visual Studio Magazine, in: https://visualstudiomagazine.com/articles/2022/09/29/net-maui-complaints.aspx (Zugriff am 17.2.2023).

REghZy (2023): The Dark Theme app, .

StephenWalther (2022): Understanding Models, Views, and Controllers (C#), in: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/understanding-models-views-and-controllers-cs (Zugriff am 16.2.2023).

Strahl, C. M., EPS Software Corp ,. <a href="/People/Bio/Rick Strahl">Rick: Getting to the ASP.NET Core, in: https://www.codemag.com/article/1609071/Getting-to-the-ASP.NET-Core (Zugriff am 19.3.2023).

ThePip: pip: The PyPA recommended tool for installing Python packages., .

Unixtime.org: Unix TimeStamp - Epoch Converter - TimeStamp Converter, in: https://unixtime.org (Zugriff am 9.2.2023).

Velimirovic, A. (2020): What is YAML? Explained With Examples, phoenixNAP Blog, in: https://phoenixnap.com/blog/what-is-yaml-with-examples (Zugriff am 9.2.2023).