

Design und Implementierung eines kabellosen Controllers für IoT- Anwendungen auf Basis des ESP32-Moduls

Studienarbeit T3100
des Studiengangs Elektrotechnik - Automation
an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Meho Kitovnica

Abgabedatum: 19.01.2025

Bearbeitungszeitraum	23.09.2024 – 19.01.2025 (150h)
Matrikelnummer, Kurs	7632804, TEL22GR5
Betreuer	Dipl. Ing (FH) Michael Peschel

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit T3100 mit dem Thema „Design und Implementierung eines kabellosen Controllers für IoT- Anwendungen auf Basis des ESP32-Moduls“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart

19.01.2025

Ort

Datum

Unterschrift



Kurzfassung

Die Studienarbeit beschäftigt sich mit dem Entwurf und der Realisierung eines kabellosen Controllers für IoT-Anwendungen basierend auf dem ESP32-Modul. Ziel ist es, ein benutzerfreundliches Gerät mit integriertem Akku, Lade- und Schutzmechanismen, USB-C- und Pogo-Anschlüssen sowie kabelloser Kommunikation zu entwickeln. Die Arbeit umfasst Konzeption, Schaltungsentwurf, Platinenlayout, Firmware-Programmierung und Funktionstests. Herausforderungen wie Fehler im PCB-Design wurden analysiert und behoben. Der Controller nutzt ESP-NOW für schnelle Peer-to-Peer-Kommunikation, unterstützt UART-Verbindungen und gewährleistet durch robuste Schutzmaßnahmen Sicherheit und Zuverlässigkeit.

Schlagwörter: IoT-Controller, ESP32, kabellose Kommunikation, Akku-Betrieb, PCB-Design, Pogo-Konnektor

Abstract

Keywords: : IoT-Controller, ESP32, Wireless, Battery-Powered, PCB-Design, Pogo-Connector

This study project focuses on the design and implementation of a wireless controller for IoT applications based on the ESP32 module. The goal is to develop a user-friendly device with an integrated battery, charging and protection mechanisms, USB-C and Pogo connectors, as well as wireless communication. The project includes concept development, circuit design, PCB layout, firmware programming, and functional testing. Thanks to its flexible architecture, the controller is suitable for various IoT applications. Challenges, such as errors in the PCB design, were analyzed and resolved. The controller uses ESP-NOW for fast peer-to-peer communication, supports UART connections, and ensures safety and reliability through robust protection measures.

Inhalt

Abbildungen	VI
Tabellen	VII
Codeausschnitt	VII
1 Problemstellung, Ziel und Vorgehensweise der Arbeit	1
2 Zielsetzung der Funktionalitäten	2
3 Schnittstellendefinition	3
4 Auswahl der Bauteile	4
4.1 Akkutechnologie	4
4.2 Ladestandüberwachung	5
4.3 Ladesteuerung und Zellschutz	7
4.4 Spannungsregler	9
5 Schaltungsentwurf	9
5.1 Ladestandüberwachung	9
5.2 Ladesteuerung und Zellschutz	11
5.3 USB-C-Konnektor	12
5.4 POGO-Konnektor	14
5.5 Power-Management zwischen USB-C und Pogo	16
5.6 Mikrocontroller	17
5.7 Weitere Schaltungen	18
6 Layout und Design des Printed-Circuit-Boards	19
6.1 Führung der USB-Datenleitungen	22
7 3D-Modellieren und Drucken vom Gehäuse	23
8 Bestückung und Lötprozess der Leiterplatte	25
9 Durchführung von Funktionstests und Fehlerbehebungen	29
9.1 Funktionstests des Ladeüberwachungs-Bauteils	29
9.2 Funktionstests des Ladesteuerung-Bauteils	30
9.3 Funktionstests der ESP-NOW Verbindung	30
9.4 Funktionstests des Pogo-Konnektors	31
9.5 Funktionstests der Taster	32
10 Firmware Design und Realisierung	33
10.1 Kommunikation mit dem Ladeüberwachung-IC	33
10.2 ESP-NOW Programmierung	38
10.3 UART-Kommunikation über den Pogo-Konnektor	40
10.4 Hauptprogramm	41
11 Anwendungsbeispiel	41
12 Reflexion und Ausblick	44
Literaturverzeichnis	45
Anhang	48

Abbildungen

Abbildung 1: "Grundlegende Schnittstellendefinition anhand der gewünschten Funktionalitäten".....	3
Abbildung 2: "Schaltung des Ladestandüberwachung-Bauteils MAX17055"	9
Abbildung 3: "Schaltung des Ladesteuerung-Bauteils BQ24075".....	12
Abbildung 4: "Schaltbild des USB-C-Konnektors".....	13
Abbildung 5: "Schaltung des Pogo-Konnektors"	14
Abbildung 6: "Schutzschaltung des Pogo-Konnektors".....	15
Abbildung 7: "Schaltung des Power-Management-Bauteils TPS2116DRLR".....	16
Abbildung 8: "Schaltung des Mikrocontroller-Moduls ESP32-S3-WROOM-1-N4"	17
Abbildung 9: "Schaltung eines Tasters"	18
Abbildung 10: "Schaltung des Spannungsreglers"	19
Abbildung 11: "Obere Lage des Printed-Circuit-Boards"	20
Abbildung 12: "Untere Lage des Printed-Circuit-Boards".....	20
Abbildung 13: "Berechnung der Leiterbahnbreite mithilfe von Herstellungsparametern und gewünschter Impedanz mittels eines Online-Kalkulators" [14]	22
Abbildung 14: "3D-Modell des Gehäuses des IoT-Controllers"	24
Abbildung 15: "3D-Modell des IoT-Controllers mit allen Komponenten"	24
Abbildung 16: "Das 3D-gedruckte Gehäuse des IoT-Controllers"	25
Abbildung 17: "Befestigte Leiterplatte"	26
Abbildung 18: "Auftragen der Lötpaste auf die Leiterplatte"	27
Abbildung 19: "Leiterplatte mit aufgetragener Lötpaste"	27
Abbildung 20: "Bestückte Leiterplatte im Reflow-Ofen"	28
Abbildung 21: "Versuchsaufbau für die ESP-NOW Verbindung. Ein IoT-Controller und ein Empfänger mit einem OLED-Display"	31
Abbildung 22: "Versuchsaufbau für die Testung des Pogo-Konnektors"	32
Abbildung 23: "Fehlerhafte (links) und korrigierte (rechts) Taster-Schaltung"	32
Abbildung 24: "I2C Schreib- und Leseprotokoll" [15]	33
Abbildung 25: "IoT-Controller gekoppelt mit einer LED-Matrix"	42
Abbildung 26: "Steuern der LED Matrix mit dem IoT-Controller"	43

Tabellen

Tabelle 1: "Direkter Vergleich der Ladeüberwachung-ICs MAX17055 und BQ27441"	6
Tabelle 2: "Direkter Vergleich der ICs MCP73871T und BQ24075"	8
Tabelle 3: "Einstellungsmodi für den Eingangsstrom des Ladesteuerung-Bauteils BQ24075" [8].....	11

Codeausschnitt

Codeausschnitt 1: "Initialisierung des I2C Buses"	34
Codeausschnitt 2: "Funktion für das Schreiben von Daten in Register des MAX17055-Chips"	36
Codeausschnitt 3: "WLAN Initialisierung für das ESP-NOW-Protokoll"	38
Codeausschnitt 4: "ESP-NOW Initialisierung".....	39

1 Problemstellung, Ziel und Vorgehensweise der Arbeit

Das ESP32 Modul bietet vielseitige drahtlose Kommunikationsmöglichkeiten (WLAN, Bluetooth, ESP-NOW) und ist ideal zur Steuerung von verschiedenen IoT-Anwendungen. Trotz intensiver Recherche existieren keine programmierbaren Controller, die auf ihrer Platine ein vollständig integriertes ESP32-Modul besitzen.

Ziel dieser Studienarbeit ist es, einen Controller mit integriertem ESP32 zu entwickeln. Dieser soll außer Steuerungsknöpfen, einen wiederaufladbaren Akku und ein funktionales Gehäuse besitzen. Die Entwicklung umfasst Konzeption, Platinen-Design, 3D-Modellieren, Programmierung und das Testen. Der Controller soll über mehrere Schnittstellen verfügen und eine benutzerfreundliche Integration in IoT-Projekte bieten.

Die Vorgehensweise der Arbeit beinhaltet:

- Zielsetzung der Funktionalitäten
- Schnittstellendefinition
- Schaltungsentwicklung und Auswahl der Bauteile
- Layout und Design des Printed-Circuit-Boards
- 3D-Modellieren und Drucken vom Gehäuse
- Firmware Design und Realisierung
- Durchführung von Funktionstests

2 Zielsetzung der Funktionalitäten

Die Zielsetzung der Funktionalitäten des IoT-Controllers erfordert eine sorgfältige Abwägung zwischen Anpassungsfähigkeit und Komplexität. Der Controller soll so konzipiert sein, dass er eine breite Kompatibilität mit unterschiedlichen IoT-Anwendungen gewährleistet und dabei eine einfache Implementierung sowie eine intuitive Bedienbarkeit sicherstellt.

Der Grad der Steuerungskomplexität ist abhängig von der jeweiligen Anwendung. Für einfache Aufgaben genügt in der Regel eine geringe Anzahl von Tasten, während komplexere Anwendungen möglicherweise den Einsatz von Joysticks erfordern. Daher besteht die Möglichkeit, dass Joystick-Module als Erweiterung an die Basisplatine angeschlossen werden, sodass der Benutzer den Komplexitätsgrad nach Bedarf anpassen kann. Obwohl eine Joystick-Schnittstelle geplant ist, wird sich diese Arbeit nicht mit der Joystick-Implementation und Testung aus zeitlichen Gründen beschäftigen.

Um den kabellosen Betriebsmodus des Controllers zu ermöglichen, ist die Einbindung eines wiederaufladbaren Akkus notwendig, der durch einen Schaltkreis ergänzt wird, der sowohl das Laden des Akkus als auch die Überwachung seines Ladezustands sowie Schutzmechanismen gewährleistet. Da der ESP32-Modul verschiedene kabellose Kommunikationswege bietet, kann der Bediener zwischen WLAN, Bluetooth-Low-Energy oder ESPNOW-Kommunikation wählen.

Der USB-C-Anschluss stellt den neuesten Standard für Steckverbindungen dar [1], weshalb die Firmware des IoT-Controllers bequem über diesen Anschluss hochgeladen werden soll. Zusätzlich wird ein 4-Pin-Pogo-Stecker verwendet, der schnelles Ein- und Ausstecken ermöglicht, da er mit federgespannten Headern und Magneten ausgestattet ist. Dieser Anschluss kann als UART-Schnittstelle dienen, über die weitere Geräte problemlos mit dem IoT-Controller gekoppelt werden, indem sie ihre MAC-Adressen über die UART-Verbindung austauschen. Nach dem Trennen der physischen Verbindung soll automatisch eine kabellose Verbindung hergestellt werden. Es ist daher vorgesehen, dass der USB-C-Konnektor primär für das Aufspielen von Software und die Durchführung von Fehlersuchen genutzt wird. Der Pogo-Konnektor hingegen ist speziell für das schnelle Laden des Akkus sowie für die Kopplung mit anderen Geräten ausgelegt.

Die konventionelle Ladespannung für moderne, akkubetriebene Kleingeräte (wie etwa Handys, Konsolen-Controller usw.) beträgt 5V. Viele Benutzer besitzen daher nicht nur ein USB-C-Kabel, sondern auch einen 5V-Stecker. Aus diesem Grund eignet sich die 5V-Spannung hervorragend als die Ladespannung.

3 Schnittstellendefinition

Bevor spezifische Bauteile für die Schaltungsentwicklung ausgewählt werden können, lassen sich die Schnittstellen lediglich auf Basis der gewünschten Funktionalitäten grob abschätzen. Dennoch erleichtert eine vorläufige Definition der Schnittstellen die Identifikation der benötigten Komponenten.

Im Kapitel „Zielsetzung der Funktionalitäten“ sind die folgenden Funktionalitäten zusammenfassend festgelegt:

- Möglichkeit, Joysticks optional anzuschließen
- Akkubetrieb mit integriertem Ladesystem, Ladezustandsüberwachung und Schutzmechanismen
- Integriertes ESP32-Modul für kabellose Kommunikation
- USB-C- und Pogo-Anschluss
- 5V-Ladespannung

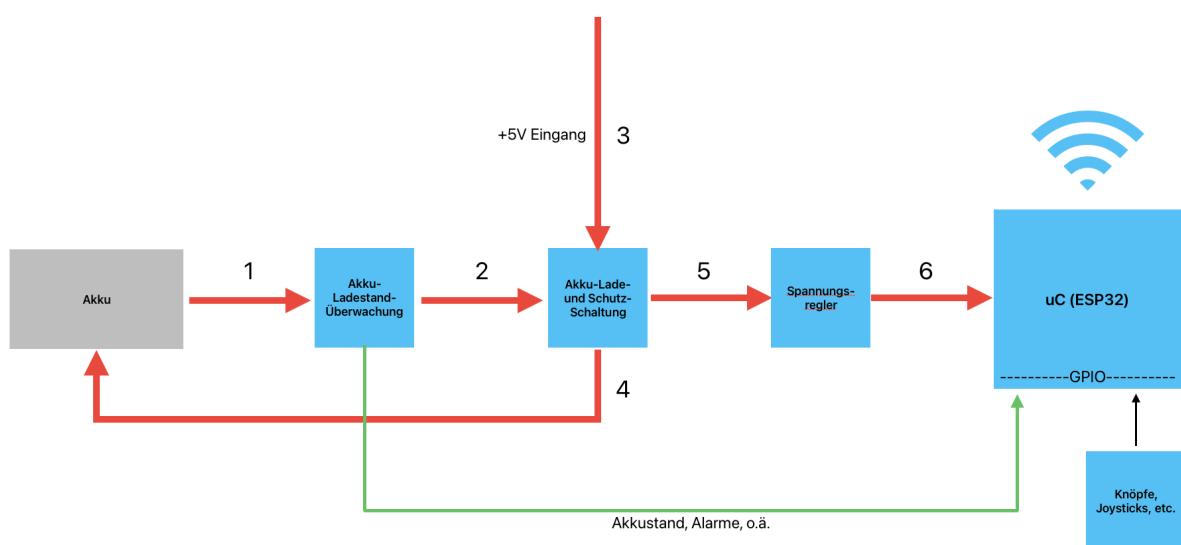


Abbildung 1: "Grundlegende Schnittstellendefinition anhand der gewünschten Funktionalitäten"

Die Abbildung 1 zeigt eine grundlegende Schnittstellendefinition. Die Vierecke sind die Komponenten und die Pfeile stellen den Datenfluss bzw. Stromfluss dar. Der Pfeil Nr. 1 zeigt auf den Ausgang des Akkus, der mit dem „Akku-Ladestand-Überwachung“-Schaltkreis verbunden ist. Bei gewöhnlichen wiederaufladbaren Lithium-Ionen-Akkus können hier Spannungen zwischen 3.7V und 4.2V anliegen. Akku-Ladestand-Überwachungs-ICs benötigen häufig verschiedene Akkufaktoren wie Temperatur, Spannung und Stromfluss, weshalb dieser Schaltkreis unmittelbar am Akku platziert werden muss. Der Schaltkreis ermittelt dann den Akkustand und gibt ggf. Alarne an den Mikrocontroller weiter. Beim Pfeil Nr. 2 wird der indirekte Akkuausgang von der „Akku-Lade- und Schutzschaltung“ aufgenommen. Falls am Pfeil Nr. 3 eine 5V-Spannung anliegt, wird ein Ladestrom am Pfeil Nr. 4 aktiviert. Die Spannung am Ausgang der „Akku-Lade- und Schutzschaltung“ (Pfeil Nr. 5) variiert je nach Vorliegen der 5V am Eingang. Diese Spannung muss daher reguliert werden. Der Spannungsregler sorgt dafür, dass die Spannung auf 3.3V gesenkt wird, da das Mikrocontroller-Modul ESP32 3.3V als Eingangsspannung benötigt. Schließlich liest der Mikrocontroller die Zustände der Knöpfe und Joysticks und überträgt diese über das festgelegte Protokoll.

4 Auswahl der Bauteile

4.1 Akkutechnologie

Zu den Hauptkriterien bei der Akkuauswahl gehören die Chemie des Akkus, Spannung, Kapazität, max. Entladestrom und die Größe des Akkugehäuses. Bei den tragbaren Geräten wie der IoT-Controller stehen zwei übliche Akkutechnologien: Lithium-Ionen (Li-Ion) und Lithium-Polymer (LiPo) Akkus. Obwohl die beiden Akkutechnologien ähnliche Charakteristiken aufweisen, bieten LiPo-Akkus eine höhere Energiedichte, wodurch sie leichter bzw. kleiner und besser für tragbare Geräte geeignet sind [2].

Eine LiPo-Zelle besitzt eine Nominalspannung von 3.7V und eine maximale Ladespannung von 4.2V. Außerdem kann die Kapazität einer LiPo-Zelle zwischen 20mA und 20A liegen [3]. Das bedeutet, dass eine einzelne Zelle vollkommen ausreichend für den Betrieb vom ESP32-Modul ist. Da die Kapazität und Größe des Akkus recht flexibel wählbar sind, werden sie erst nach der Schaltungsentwicklung und Gehäusedesign bestimmt.

4.2 Ladestandüberwachung

Im „Akkuladestand-Überwachung“-Schaltkreis aus der Abbildung 1 müssen mehrere Akku-Parameter gemessen werden, um möglichst genauen Akkuladestand bzw. übrige Kapazität zu bestimmen. Aus diesem Grund ist es zu erwarten, dass dieser Schaltkreis einen komplexeren Schaltungsentwurf als die anderen Schaltkreise erfordert. Daher macht es nur Sinn, den Ladestandüberwachung-ICs als erstes auszuwählen und falls möglich, die restlichen ICs nach dieser zu richten.

Moderne Ladestandüberwachungs-ICs nutzen die sogenannte „Fuel Gauge“-Technologie, die es ermöglicht, das Verhalten eines Akkus anhand verschiedener Messparameter präzise zu charakterisieren. Zum Zeitpunkt der Recherche waren insbesondere zwei Bauteile populär und gut verfügbar: der *MAX17055* und der *BQ27441*. In der folgenden Tabelle sind die für die Auswahl eines Ladestandüberwachungs-ICs relevanten Kriterien vergleichend gegenübergestellt. Sämtliche Angaben basieren auf den technischen Daten aus den jeweiligen Datenblättern der beiden Bauteile.

Kriterien	MAX17055	BQ27441	Vergleich
Technologie	ModelGauge™ m5 EZ Algorithmus, keine Charakterisierung	Impedance Track™ Technologie	Beide haben fortschrittliche Algorithmen, aber der MAX17055 erfordert keine Batteriecharakterisierung.
Stromverbrauch	Niedriger Stromverbrauch (0.7 µA im Standby)	Niedriger Stromverbrauch (0.6 µA im Shutdown)	Beide ICs haben sehr niedrigen Stromverbrauch, mit minimalen Unterschieden im Standby- oder Shutdown-Modus.
Messungen	Misst Spannung, Strom und Temperatur, intern/extern	Misst Spannung, Strom, Temperatur (intern/extern)	Beide bieten umfassende Messungen. Keine signifikanten Unterschiede.
Batterieunterstützung	Unterstützt Li-Ion und LiFePO4	Unterstützt Li-Ion	Der MAX17055 unterstützt mehr Batterietypen
Schnittstelle	I²C-Schnittstelle	I²C-Schnittstelle	Beide verwenden eine I²C-Schnittstelle
State of Charge (SoC)	Prozentualer SoC und Zeit zu leer/voll, Temperaturkompensation	Prozentualer SoC mit Glättungsfilter	Beide bieten SoC-Überwachung

Tabelle 1: "Direkter Vergleich der Ladeüberwachung-ICs MAX17055 und BQ27441"

Aus dem Vergleich lässt sich feststellen, dass beide Bauteile über ähnliche Eigenschaften verfügen. Allerdings bietet der *MAX17055* den Vorteil, dass er eine Implementierung ohne vorherige Batteriecharakterisierung ermöglicht, was den Entwicklungsaufwand erheblich reduziert. Da dieses Projekt zeitlich begrenzt ist, stellt die Möglichkeit, den Ladestandüberwachungs-IC ohne Batteriecharakterisierung zu implementieren, einen entscheidenden Vorteil dar. Aus diesem Grund wurde der *MAX17055* gegenüber dem *BQ27441* für die Integration in das System ausgewählt.

4.3 Ladesteuerung und Zellschutz

Für Anwendungen mit einer einzelnen Lithium-Ionen-Zelle, wie Smartphones, tragbare GPS-Geräte und Media-Player, müssen zwei Batteriemanagement-Funktionen in jeder Systemkonfiguration vorhanden sein: *Ladesteuerung* und *Zellschutz* des Batteriekopfes [4]. Zum Zeitpunkt der Recherche waren insbesondere die ICs *MCP73871T* und *BQ24075* besonders beliebt, da sie nicht nur die Ladesteuerung, sondern auch den Zellschutz innerhalb eines einzigen ICs integrieren. Darüber hinaus bieten beide Bauteile verschiedene Eingänge, die es ermöglichen, das Lade- und Schutzverhalten mithilfe eines Mikrocontrollers anzupassen. In Tabelle 2 sind die beiden ICs *MCP73871T* und *BQ24075* anhand ausgewählter, projektrelevanter Kriterien miteinander verglichen. Sämtliche Angaben basieren auf den technischen Daten aus den jeweiligen Datenblättern der beiden Bauteile.

Kriterien	Microchip <i>MCP73871T</i>	Texas Instruments - <i>BQ24075</i>	Vergleich
Input Power Sources and System Load Management	Dual Input: USB & AC-DC Adapter, 1.8A Gesamtstrom	Dual Input: USB & Adapter, 1.5A Gesamtstrom	<i>MCP73871</i> unterstützt höheren Gesamtstrom (1.8A), nützlich für leistungsintensivere Anwendungen.
Maximaler Ladestrom	Bis zu 1A programmierbar	Bis zu 1.5A programmierbar	<i>BQ24075</i> bietet höheren Ladestrom,

			ideal für schnelleres Laden.
Eingangsspannungsbereich	Bis zu 6V	Bis zu 28V (mit Überspannungsschutz)	BQ24075 unterstützt höhere Eingangsspannungen, nützlich bei Spannungsspitzen.
Thermal Management	Thermische Regulierung, Über- und Unterspannungsschutz	Thermische Regulierung, Kurzschluss- und Überspannungsschutz	Beide bieten thermische Regulierung, aber der BQ24075 hat zusätzlichen Kurzschluss- und Überspannungsschutz.
Batteriemanagement	Automatisches Um- schalten zwischen USB/AC-DC und Batterie	Dynamisches Power-Path-Management, Batterie- Backup	Beide haben ähnliche Power-Path-Management-Systeme
Eingangsüberlastschutz	Eingangs- und Batterieschutz	Überspannungsschutz bis 28V, Kurzschluss-Schutz	BQ24075 bietet umfassenderen Schutz bei hohen Spannungsspitzen und Kurzschlägen.

Tabelle 2: "Direkter Vergleich der ICs MCP73871T und BQ24075"

Obwohl der *MCP73871* einen höheren Gesamtstrom von bis zu 1,8 A unterstützt, bietet der *BQ24075* entscheidende Vorteile, insbesondere durch einen höheren Ladestrom und umfassendere Schutzfunktionen. Aufgrund dieser Vorteile wurde der *BQ24075* für die Implementierung in das System ausgewählt.

4.4 Spannungsregler

Für die Spannungsversorgung des IoT-Controllers wurde der TPAP2112K-3.3TRG1 ausgewählt. Dieser Spannungsregler liefert eine konstante Ausgangsspannung von 3,3 V bei einer maximalen Ausgangstromstärke von 600 mA, wodurch der Energiebedarf des eingesetzten ESP32-Mikrocontrollers abgedeckt wird. Das kompakte SOT-23-5-Gehäuse erlaubt eine Integration in platzbeschränkten Designs. Der Ruhestrom liegt bei 10 µA, wodurch der Spannungsregler auch für Anwendungen mit geringen Energieanforderungen geeignet ist. Die Auswahl des TPAP2112K-3.3TRG1 basiert auf einer Analyse der Systemanforderungen und der technischen Spezifikationen.

5 Schaltungsentwurf

5.1 Ladestandüberwachung

Die Schaltungsentwicklung des Ladestandüberwachung-ICs basiert auf den Empfehlungen des MAX17055-Datenblatts: [5].

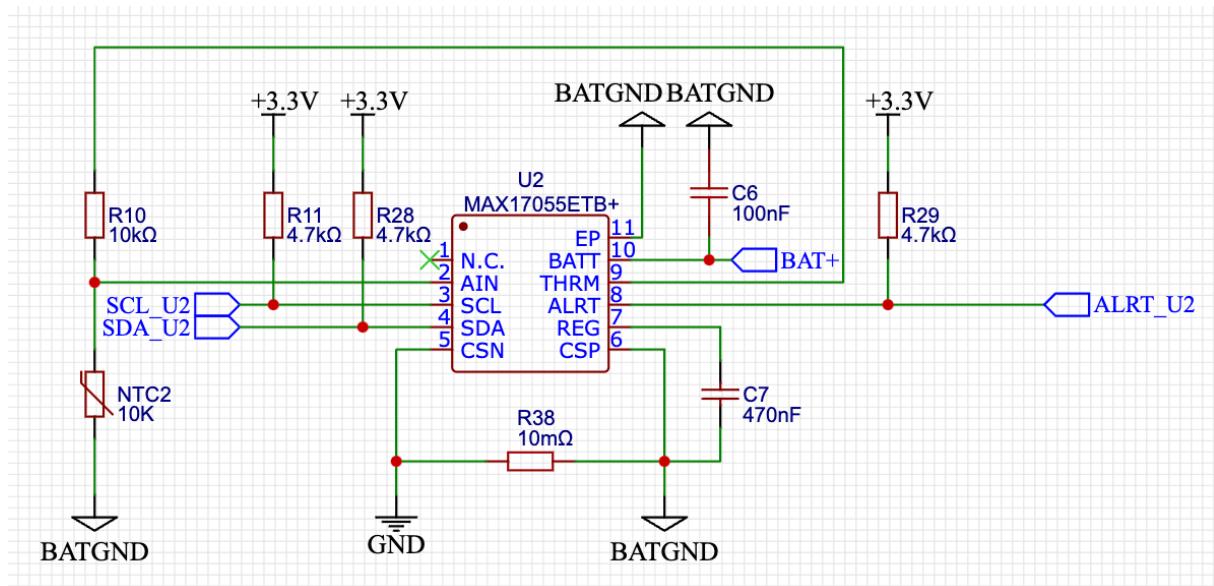


Abbildung 2: "Schaltung des Ladestandüberwachung-Bauteils MAX17055"

Die Abbildung 2 zeigt die Schaltung zur Ladestandüberwachung, die mit dem MAX17055 (in der Abbildung als Bauteil *U2* gekennzeichnet) realisiert wurde. Die Widerstände *R10*, *R38*, *NTC2* sowie die Kondensatoren *C6* und *C7* besitzen die im MAX17055-Datenblatt empfohlenen Werte. Der *ALRT*-Pin ist ein Open-Drain-Ausgang des Bauteils *U2* und mit einem GPIO des Hosts verbunden. Da es sich um einen Open-Drain-Ausgang handelt, ist ein Pull-Up-Widerstand erforderlich (in diesem Fall *R29*), um ein stabiles Verhalten des Ausgangs sicherzustellen.

Die SCL- und SDA-Leitungen sind ebenfalls Open-Drain-I2C-Leitungen, die jeweils Pull-Up-Widerstände benötigen. Der Wert der Pull-Up-Widerstände wurde durch die Berechnung des minimalen und maximalen Widerstands für die I2C-Leitungen ermittelt. Die entsprechenden Formeln und weitere Informationen zur Widerstandsberechnung für I2C-Leitungen finden sich unter [6].

Gleichung 1: "Berechnung den Wert des mindesten Widerstandwerts für die I2C-Leitung" [6]

$$R_P(\min) = \frac{(V_{CC} - V_{OL}(\max))}{I_{OL}}$$

Für V_{CC} , die Betriebsspannung, ist auf 3,3V festgelegt. Die Parameter $V_{OL}(\max)$ (maximale Spannung für einen logischen Low-Pegel) und I_{OL} (Ausgangstromstärke bei einem logischen Low-Pegel) sind im MAX17055-Datenblatt spezifiziert und betragen $V_{OL}(\max) = 0,4$ V und $I_{OL} = 4$ mA ([7]). Durch Einsetzen dieser Werte ergibt sich ein minimaler Pull-Up-Widerstand von $R_P(\min) = 725$ Ω. Anschließend wurde der maximale Widerstand mithilfe der Gleichung 2 berechnet. Hierbei spielt die maximale Aufstiegszeit des Signals (t_r) eine entscheidende Rolle.

Gleichung 2: "Berechnung des maximalen Widerstand für die I2C-Leitung" [6]

$$R_p(\max) = \frac{t_r}{(0.8473 \times C_b)}$$

Die maximale Aufstiegszeit beträgt im „Standard Mode“ 1000 ns, im „Fast Mode“ 300 ns und im „Fast Mode Plus“ 150 ns, [6]. Da der MAX17055 Aufstiegszeiten zwischen 5 ns und 300 ns auf der I2C-Leitung unterstützt (siehe [5]), kommen nur der „Fast Mode“ und der „Fast Mode Plus“ in Frage. Da es sich um keine echtzeitkritische Anwendung handelt und die Übertragungsgeschwindigkeit der Akkudaten nicht besonders hoch sein muss, ist der „Fast Mode“ mit $t_r = 300$ ns als ausreichend betrachtet.

Die kapazitive Last der Busleitung (C_b) kann noch nicht exakt bestimmt werden, wurde jedoch näherungsweise mithilfe [7] auf $C_b = 22,38$ pF geschätzt. Durch Einsetzen der Werte für t_r und C_b ergibt sich ein maximaler Widerstand von $R_p(\max) = 15,82$ kΩ. Der Bereich zwischen $R_p(\min)$ und $R_p(\max)$ ist ziemlich groß. Um den Auswahl zu vereinfachen, ist schließlich der Standard I2C-Pull-Up-Widerstand $R_p = 4.7$ kΩ gewählt.

5.2 Ladesteuerung und Zellschutz

Das Ladesteuerungs-Bauteil „BQ24075“ bietet verschiedene Modi zur Regelung des Eingangsstroms, die über die EN1- und EN2-Pins eingestellt werden können. Tabelle 3 zeigt die möglichen Kombinationen der EN1- und EN2-Pin-Zustände und den jeweiligen aktvierten Modus.

Tabelle 3: "Einstellungsmodi für den Eingangsstrom des Ladesteuerung-Bauteils BQ24075" [8]

EN2	EN1	Maximale Eingangsstrom beim IN Pin
0	0	100mA. USB100 Modus
0	1	500mA. USB500 Modus
1	0	Eingestellt durch externe Widerstände zwischen ILIM zu VSS
1	1	Standby (USB gesperrt Modus)

In der frühen Entwicklungsphase ist es schwierig, den optimalen Modus für den Eingangsstrom des BQ24075 festzulegen. Zudem sind zwei Eingangskonnektoren für den IoT-Controller vorgesehen: USB-C und Pogo. Daher ist beschlossen, die EN2- und EN1-Pins durch den Host-Mikrocontroller ESP32 steuerbar zu gestalten. Zusätzlich sind für die Pins des Sicherheitstimers (TMR), der einstellbaren Ladestromstärke (ILIM) und der einstellbaren Schnell-Ladestromstärke (ISET) Potentiometer vorgesehen. Diese ermöglichen es, während der Entwicklungsphase unterschiedliche Werte zu testen und die Effektivität des Ladens bei Langzeittests zu vergleichen. Im Rahmen dieser Arbeit können jedoch aus zeitlichen Gründen keine umfangreichen Langzeittests durchgeführt werden. Dennoch bietet diese flexible Konfiguration eine Grundlage für zukünftige Weiterentwicklungen und Optimierungen.

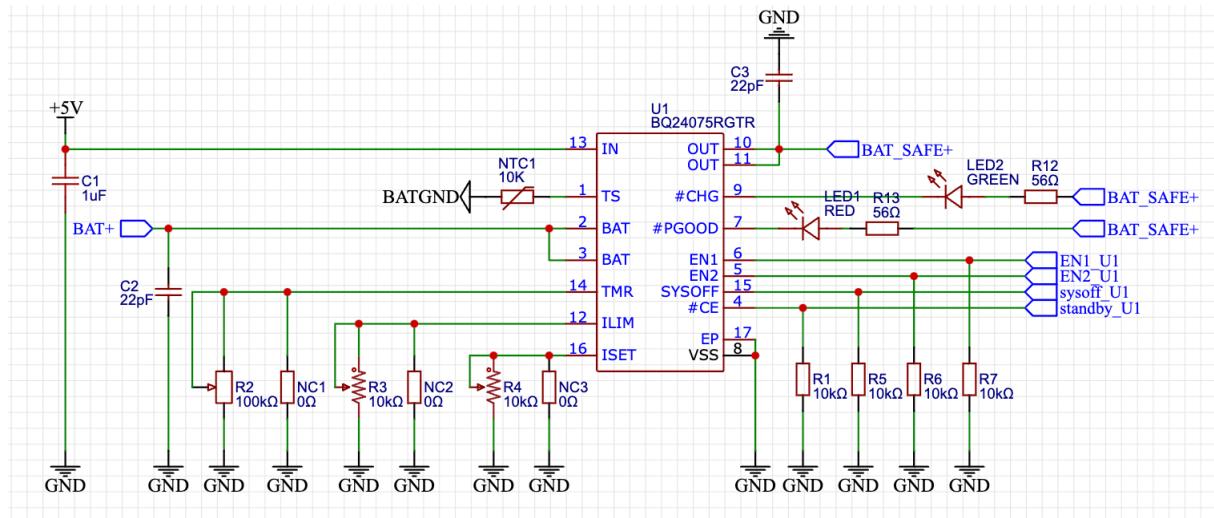


Abbildung 3: "Schaltung des Ladesteuerung-Bauteils BQ24075"

Wie in Abbildung 3 dargestellt, sind die Pins *TMR*, *ILIM* und *ISET* nicht nur mit Potentiometern, sondern parallel dazu auch mit 0- Ω -Widerständen beschaltet. Diese Widerstände können in der Entwicklungsphase unbestückt bleiben, sodass die Einstellung der optimalen Widerstandswerte über die Potentiometer erfolgt. Nach der Festlegung der optimalen Werte können die Potentiometer durch feste Widerstände ersetzt werden. Die Schaltung in Abbildung 3 umfasst zudem Filterkondensatoren: einen 1- μ F-Kondensator am Eingang (IN-Pin) und einen 22-pF-Kondensator am Ausgang (OUT-Pin). Die Pins *#CHG* und *#PGOOD* sind als Open-Drain-Ausgänge ausgeführt und jeweils mit einer LED und einem Vorwiderstand verbunden. Diese Pins schalten durch und bringen die LEDs zum Leuchten, falls der Akku geladen wird (*#CHG*) bzw. falls die Eingangsspannung ordnungsgemäß anliegt (*#PGOOD*). Die Pins *EN1*, *EN2*, *SYS-OFF* (System Enable Input) und *CE* (Charge Enable Input) sind jeweils mit einem 10-k Ω -Pull-Down-Widerstand versehen und mit den GPIOs des Mikrocontrollers verbunden. Zusätzlich verfügt der BQ24075 über einen Pin zur Temperaturüberwachung (*TS*), der mit einem 10-k Ω -NTC-Widerstand beschaltet ist. Diese Temperaturüberwachung ermöglicht es, den Ladevorgang bei zu hohen Temperaturen automatisch zu unterbrechen, um die Sicherheit des Systems zu gewährleisten.

Die Verbesserungen der Schaltung sind im Kapitel 9.2 zu sehen.

5.3 USB-C-Konnektor

Der USB-C-Konnektor dient sowohl als Software-Upload-Schnittstelle als auch Ladequelle des Akkus und die Spannungsversorgung des Systems. Die USB-C-Stromversorgung erwartet, dass ein 5.1k Ω Pull-Down-Widerstand an der CC-Leitung erkannt

wird, bevor sie 5 V @ max 3A auf VBUS bereitstellt. Höhere Spannungen müssen digital ausgehandelt werden, sind jedoch für diese Anwendung nicht notwendig. [9].

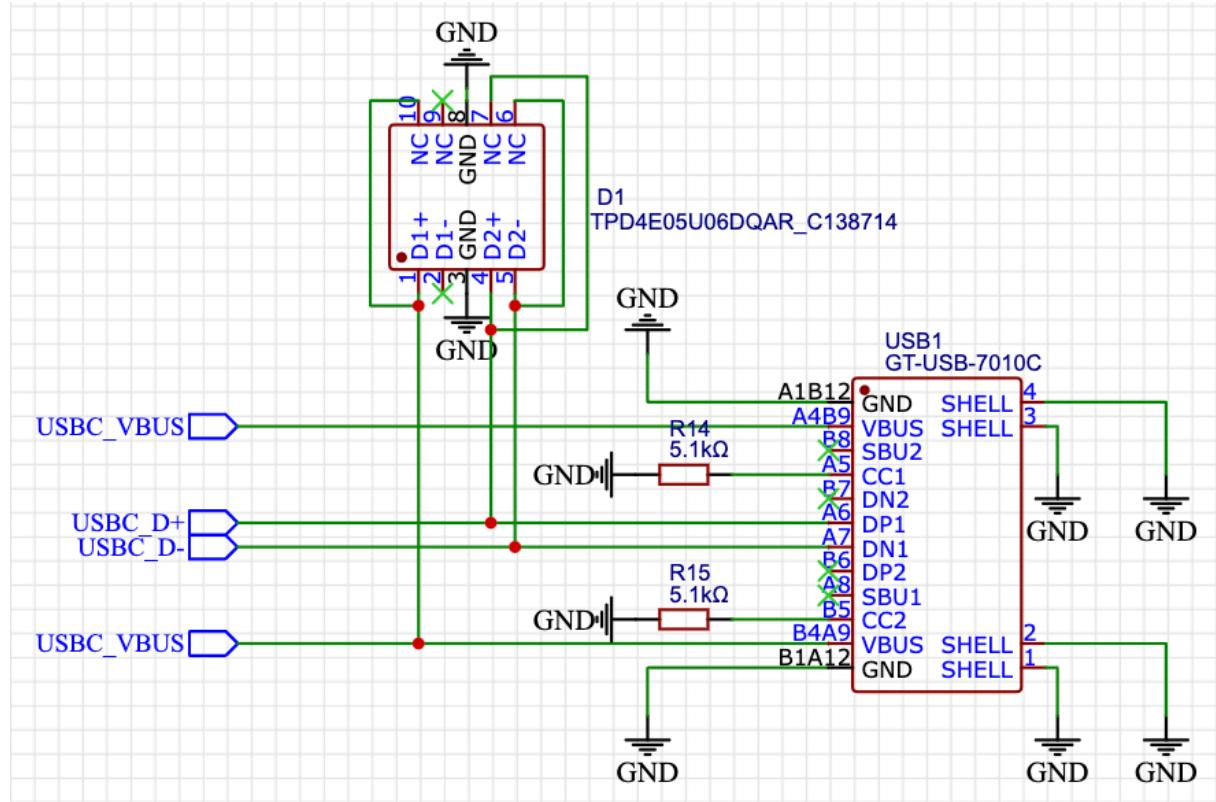


Abbildung 4: "Schaltbild des USB-C-Konnektors"

Abbildung 4 zeigt den USB-C-Konnektor mit USB 2.0-Konnektivität. Für USB 2.0 müssen die CC-Pins des „Upstream Facing Ports“ (der Port eines Geräts, das Strom oder Daten empfängt) gemäß [9] jeweils mit einem 5,1-kΩ-Pull-Down-Widerstand beschaltet werden, um eine ordnungsgemäße Funktion zu gewährleisten. Die Datenleitungen D+ und D- sind direkt mit den GPIO-Pins 20 und 19 des ESP32-S3 verbunden. Die VBUS-Leitung liefert eine Spannung von 5,0 V und ist mit dem Power-Management-Bauteil *TPS2116DRLR* verschaltet, um sicherzustellen, dass bei gleichzeitiger Verbindung beider Spannungsquellen (USB-C und Pogo) mit dem System unerwartetes Verhalten vermieden wird. Zur Absicherung des Systems sind alle relevanten Pins mit der TVS/ESD-Schutzdiode *TPD4E05U06DQAR* ausgestattet, wodurch das System effektiv vor Überspannungen und elektrostatischen Entladungen geschützt wird.

Die Verschaltung des Power-Management-Bauteils *TPS2116DRLR* ist im Kapitel 5.5 „Power-Management zwischen USB-C und Pogo“. Die vollständige Schaltung, inklusive des Power-Management-ICs und des Mikrocontrollers, ist im Anhang zu finden.

5.4 POGO-Konnektor

Die Pins des Pogo-Konnektors sind ungeschützt und könnten bei Kontakt mit einem leitfähigen Material einen Kurzschluss verursachen. Um dies zu verhindern, wird der 5-V-Pin erst dann mit dem System verbunden, wenn der Mikrocontroller ein Freigabesignal sendet, nachdem sichergestellt wurde, dass der Pogo-Stecker ordnungsgemäß mit einem anderen Gerät verbunden ist.

Die Überprüfung erfolgt über die UART-Schnittstelle, die über den Pogo-Stecker bereitgestellt wird. Dabei wird auf eine Bestätigungsnachricht des angeschlossenen Geräts gewartet. Sobald diese Bestätigung vorliegt, sendet der Mikrocontroller das Freigabesignal. Dieser Sicherheitsmechanismus wurde zusätzlich durch den im Kapitel 10.3 beschriebenen Code implementiert.

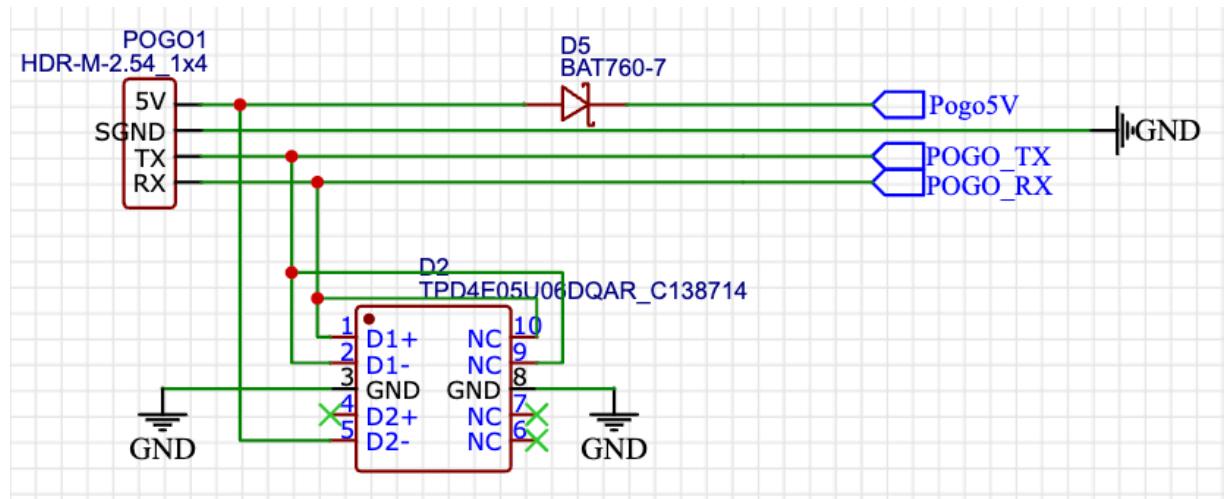


Abbildung 5: "Schaltung des Pogo-Konnektors"

In Abbildung 5 ist die Schaltung des Pogo-Konnektors dargestellt. Diese umfasst den Konnektor (POGO1), TVS/ESD-Schutzdiode (D2) sowie eine Rückstrom-Schutzdiode (D5). Der 5-V-Pin des Pogo-Konnektors ist nicht direkt mit dem System verbunden, sondern wird zunächst über einen Transistor geschaltet. Dieser Transistor leitet den Strom nur dann durch, wenn eine sichere Verbindung zwischen dem Pogo-Stecker und dem Zielgerät sichergestellt wurde. Die Signale POGO_RX und POGO_TX sind mit den GPIO-Pins 4 und 5 des Mikrocontrollers verbunden. Alle Pins des Pogo-Konnektors (außer dem GND-Pin) sind mit TVS/ESD-Schutzdioden ausgestattet, wodurch ein effektiver Schutz vor elektrostatischen Entladungen und Überspannungen gewährleistet wird.

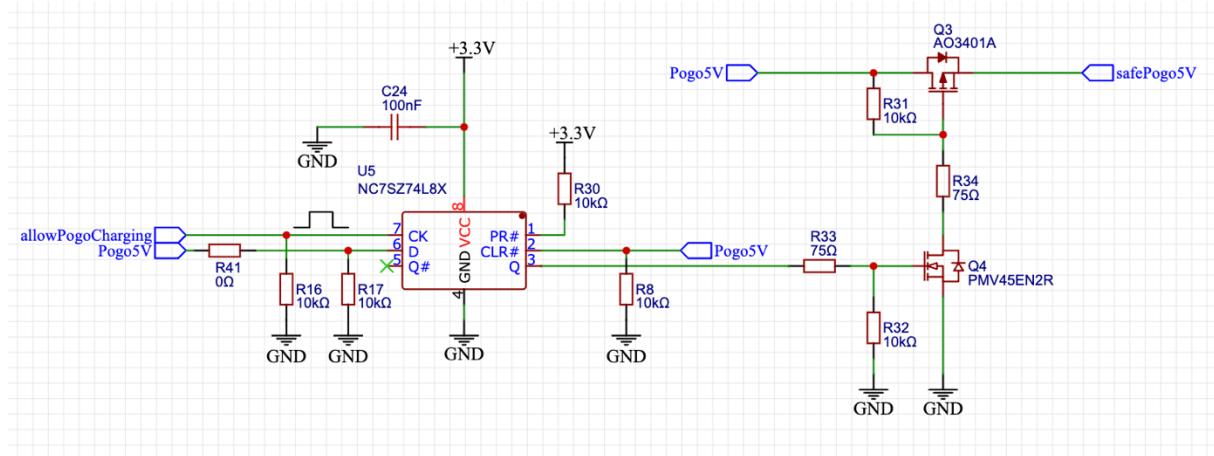


Abbildung 6: "Schutzschaltung des Pogo-Konnektors"

Der Transistor, der den 5V-Pin des Pogo-Konnektors nur unter sicheren Bedingungen mit den 5 V des Systems verbindet, ist in Abbildung 6 als Q3 dargestellt. Q3 ist ein P-Kanal-MOSFET, der im durchgeschalteten Zustand einen sehr niedrigen Drain-Source-Widerstand aufweist ($<50 \text{ m}\Omega$) [10]. Das Gate des P-Kanal-MOSFETs Q3 ist mit dem Drain des N-Kanal-MOSFETs Q4 verbunden. Wenn Q4 durchschaltet, wird das Gate des P-Kanal-MOSFETs Q3 auf ein niedriges Potenzial gezogen, wodurch ein negatives Gate-Source-Spannung (V_{GS}) entsteht. Dies führt dazu, dass Q3 leitend wird und den 5-V-Pin des Pogo-Konnektors mit der 5V-Leitung des Systems verbindet. Ob der N-Kanal-MOSFET Q4 durchschaltet, wird durch ein D-Flipflop (U5) gesteuert. Der Flipflop setzt am Ausgang Q das Potenzial des 5-V-Pins des Pogo-Konnektors auf HIGH, sobald ein Impuls (steigende Flanke) des Mikrocontrollers am CK-Eingang des Flipflops erkannt wird. Das Ausgangssignal Q bleibt so lange auf HIGH, bis entweder eine weitere steigende Flanke am CK-Eingang auftritt oder das hohe Potenzial am 5V-Pin des Pogo-Konnektors verloren geht, was beim Trennen des Konnektors vom Zielgerät geschieht.

Diese Logik gewährleistet eine sichere und kontrollierte Verbindung zwischen dem 5-V-Pin des Pogo-Konnektors und dem System. Sie schützt sowohl den Konnektor als auch das angeschlossene System vor unerwünschten Kurzschlägen oder potenziellen Schäden, die durch unsachgemäße Verbindungen entstehen könnten. Zusätzlich wird die Sicherheit des Systems durch die implementierte Hardware gewährleistet, anstatt ausschließlich von der Software abhängig zu sein, die potenziell Fehler enthalten könnte.

5.5 Power-Management zwischen USB-C und Pogo

Falls gleichzeitig eine Spannungsquelle am USB- und am Pogo-Konnektor besteht, ist es wichtig sicherzustellen, dass beide Quellen entkoppelt sind und feste Prioritäten besitzen. Diese Funktionalitäten lassen sich mithilfe des Power-Management-Bauteils *TPS2116DRLR* realisieren.

Das Power-Management-Bauteil *TPS2116DRLR* unterstützt zwei Betriebsmodi: den *Prioritätsmodus*, bei dem die Spannungsquelle Nr. 1 stets Vorrang vor der Spannungsquelle Nr. 2 hat, und den *manuellen Modus*, bei dem die Auswahl der Spannungsquelle durch ein GPIO-Signal gesteuert werden kann, [11].

Es ist vorgesehen, den USB-C-Konnektor primär für das Update von Software sowie für Fehlersuchen zu nutzen, während der Pogo-Konnektor für schnelles Laden und die Kopplung mit anderen Geräten eingesetzt wird. Da Software-Updates und Fehlersuchen eine höhere Priorität gegenüber dem schnellen Laden und der Geräteverbindung besitzen, wird die Priorität des USB-C-Konnektors gegenüber dem Pogo-Konnektor festgelegt. Außerdem soll die Sicherheit nicht von der Software abhängen, sondern durch Hardware automatisch gewährleistet werden. Daher eignet sich der *Prioritätsmodus* des *TPS2116DRLR* deutlich besser für diese Anwendung als der *manuelle Modus*, da er eine klare, hardwarebasierte Priorisierung ohne Abhängigkeit von Software implementiert.

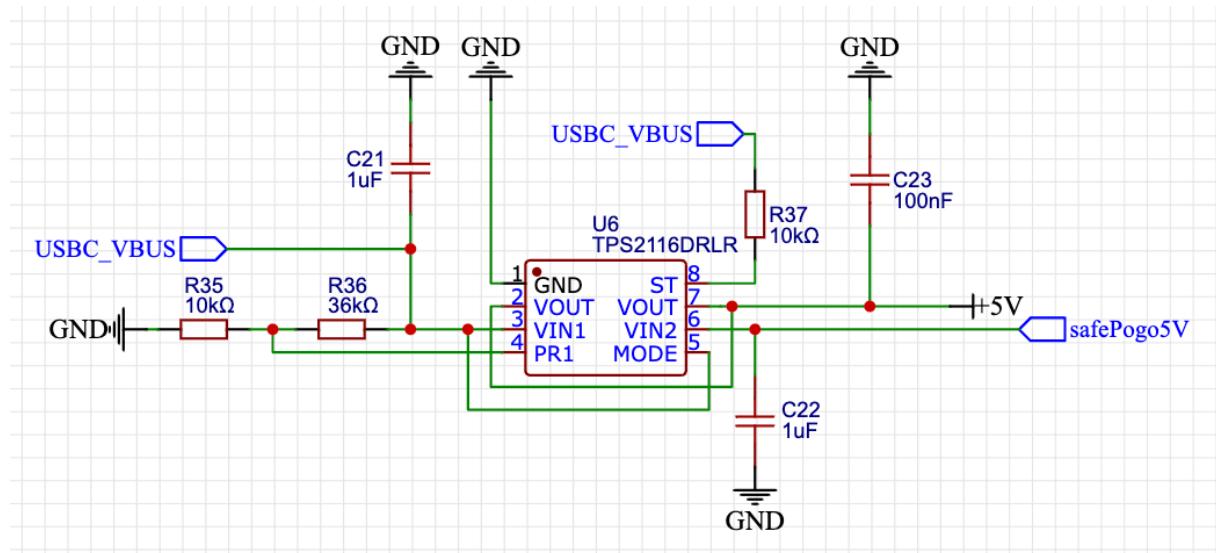


Abbildung 7: "Schaltung des Power-Management-Bauteils *TPS2116DRLR*"

Abbildung 7: "Schaltung des Power-Management-Bauteils *TPS2116DRLR*" Abbildung 7 zeigt die Verschaltung des Power-Management-Bauteils *TPS2116DRLR* (*U6*) im

Prioritätsmodus. In diesem Modus wird die Spannungsquelle $VIN1$ an den Ausgang $VOUT$ geschaltet, wenn am $PR1$ -Pin von $U6$ ein HIGH-Potenzial anliegt. Andernfalls wird die Spannungsquelle $VIN2$ mit dem Ausgang $VOUT$ verbunden. Bei einer anliegenden Spannung am $VBUS$ des USB-C-Konnektors wird ein hohes Potenzial durch den Spannungsteiler, bestehend aus den Widerständen $R36$ und $R35$, an den $PR1$ -Pin geleitet. Dadurch wird $VBUS$ an den Ausgang $VOUT$ weitergegeben. Sollte hingegen der USB-C-Konnektor nicht verbunden sein, aber eine sichere Verbindung am Pogo-Konnektor hergestellt werden, wird die 5V-Spannung des Pogo-Konnektors an $VOUT$ ausgegeben. Um den Prioritätsmodus zu aktivieren, muss außerdem der $MODE$ -Pin des $TPS2116DRLR$ mit $VIN1$ verbunden werden. Die Schaltung zur Konfiguration des Bauteils im Prioritätsmodus wurde aus dem entsprechenden Datenblatt [11] übernommen und an die Anforderungen dieses Systems angepasst.

5.6 Mikrocontroller

Der Mikrocontroller ESP32-S3 bildet das zentrale Steuerungselement des Systems und übernimmt die Kontrolle über die weiteren Bauteile des IoT-Controllers mittels seiner Ein- und Ausgänge. Die Auswahl der geeigneten GPIOs für die benötigten Signale erfolgte unter Berücksichtigung der „Pin Definitions“ -Tabelle im Datenblatt des Mikrocontrollers [12].

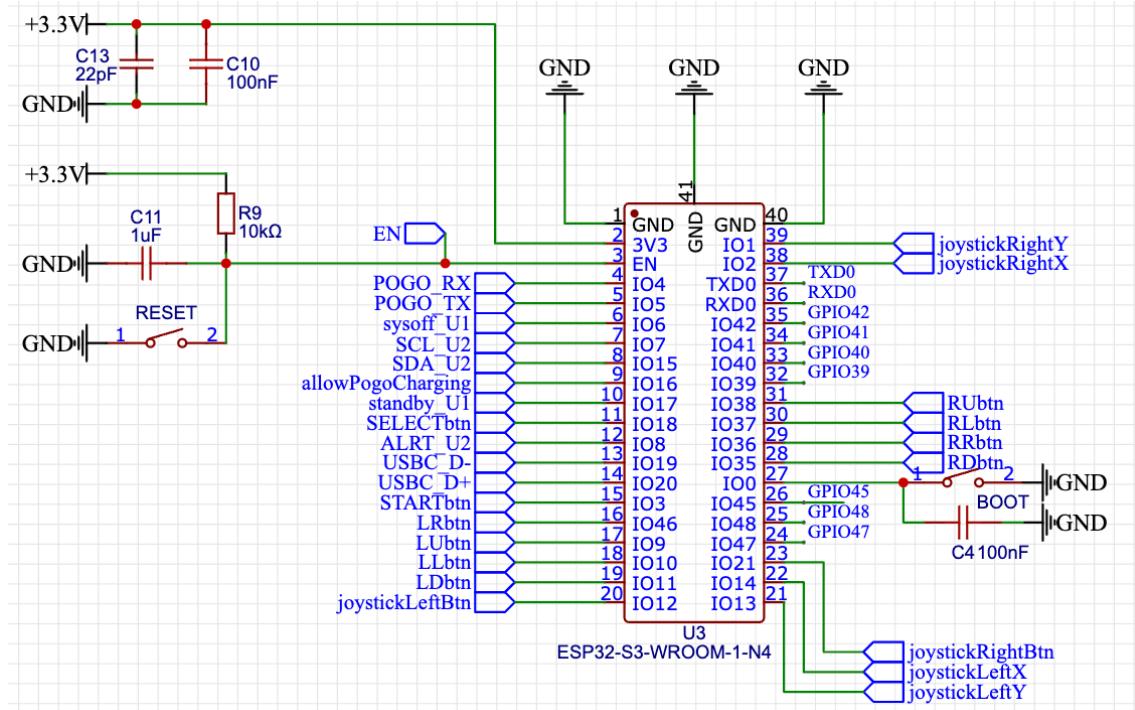


Abbildung 8: "Schaltung des Mikrocontroller-Moduls ESP32-S3-WROOM-1-N4"

Der Download-Modus des ESP32-S3 wird aktiviert, indem der Reset-Taster gedrückt gehalten wird, während der Boot-Taster einmal betätigt und anschließend losgelassen wird. In diesem Modus wird die USB-Serielle/JTAG-Schnittstelle des Mikrocontrollers freigeschaltet, wodurch die Software über diese Verbindung hochgeladen werden kann.

5.7 Weitere Schaltungen

Der IoT-Controller verfügt über insgesamt zwölf Taster, von denen zehn für den Benutzer zugänglich sind. Zusätzlich sind zwei weitere Taster, der Reset-Taster und der Boot-Taster, im Inneren des Gehäuses verbaut.

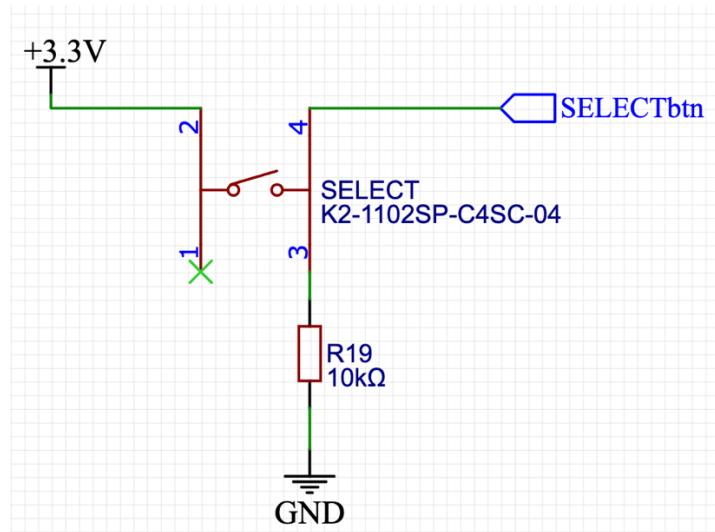


Abbildung 9: "Schaltung eines Tasters"

Die für den Benutzer zugänglichen Taster sind alle identisch verschaltet, wie in Abbildung 9 dargestellt. Wird ein Taster betätigt, kann ein HIGH-Signal am entsprechenden GPIO-Pin des Mikrocontrollers ausgelesen werden. Wenn der Taster nicht betätigt wird, liegt durch den eingebauten Pull-Down-Widerstand ein LOW-Signal am GPIO-Pin an.

Die in Abbildung 9 gezeigte Taster-Schaltung ist eine korrigierte Version, da ein Fehler in der ursprünglichen Schaltung entdeckt wurde, welche auf den gefertigten Platinen implementiert ist. Eine ausführliche Erläuterung des Fehlers sowie der vorgenommenen Anpassungen ist in Kapitel 9.5 zu finden.

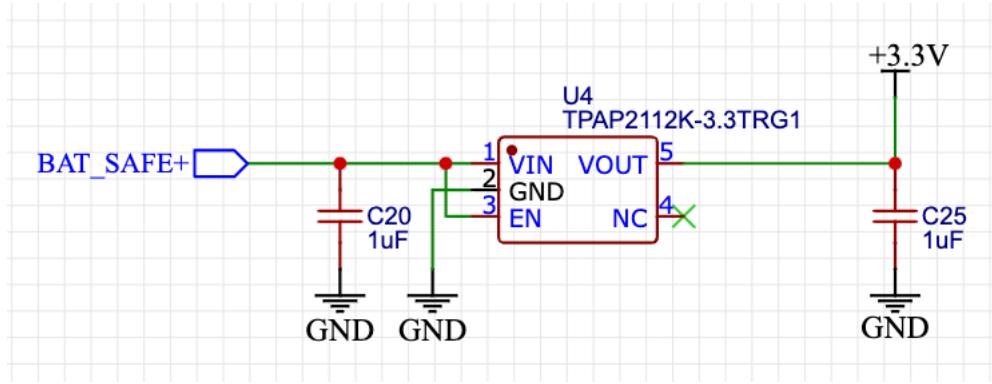


Abbildung 10: "Schaltung des Spannungsreglers"

Der in Abbildung 10 dargestellte Spannungsregler ist mit zwei Abblockkondensatoren ausgestattet, von denen einer am Eingang und einer am Ausgang platziert ist. Diese Kondensatoren dienen dazu, Störungen und Spannungsschwankungen zu minimieren, wodurch eine stabile Spannungsversorgung gewährleistet wird. Der Eingang (VIN) ist mit dem Enable-Pin (EN) des Reglers verbunden, sodass der Spannungsregler automatisch aktiviert wird, sobald das System mit einer Eingangsspannung versorgt wird. In diesem Zustand stellt der Regler die notwendigen 3,3 V für den Betrieb des Mikrocontrollers bereit.

6 Layout und Design des Printed-Circuit-Boards

Das PCB-Layout wurde speziell für einen kompakten IoT-Controller entworfen, mit den Abmessungen von 100mm x 50 mm und abgerundeten Ecken (Radius: 10 mm). Diese Größe bietet eine ergonomische Form, die gut in der Hand liegt, und ist gleichzeitig platzsparend.

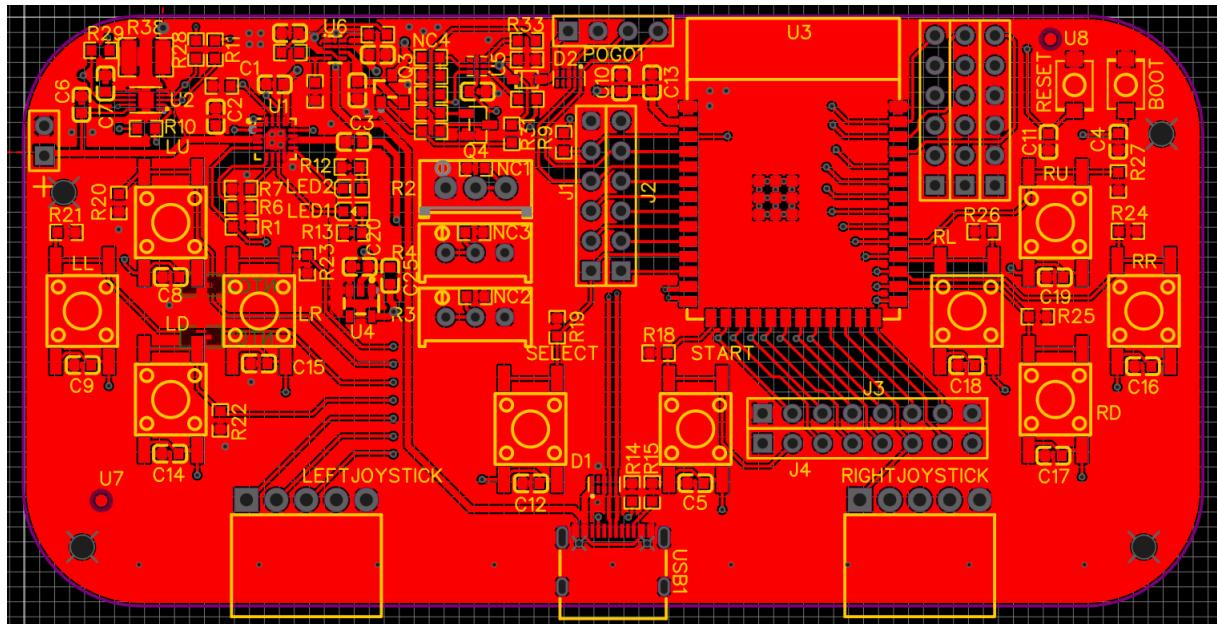


Abbildung 11: "Obere Lage des Printed-Circuit-Boards"

In Abbildung 11 ist ersichtlich, dass die Mehrzahl der Bauteile auf der oberen Lage der Leiterplatte angeordnet ist. Vier Taster befinden sich auf der linken Seite, vier Taster auf der rechten Seite und zwei Taster in zentraler Position. Darüber hinaus sind zwei Joystick-Anschlüsse vorgesehen, die mit herkömmlichen KY-023-Modulen kompatibel sind.

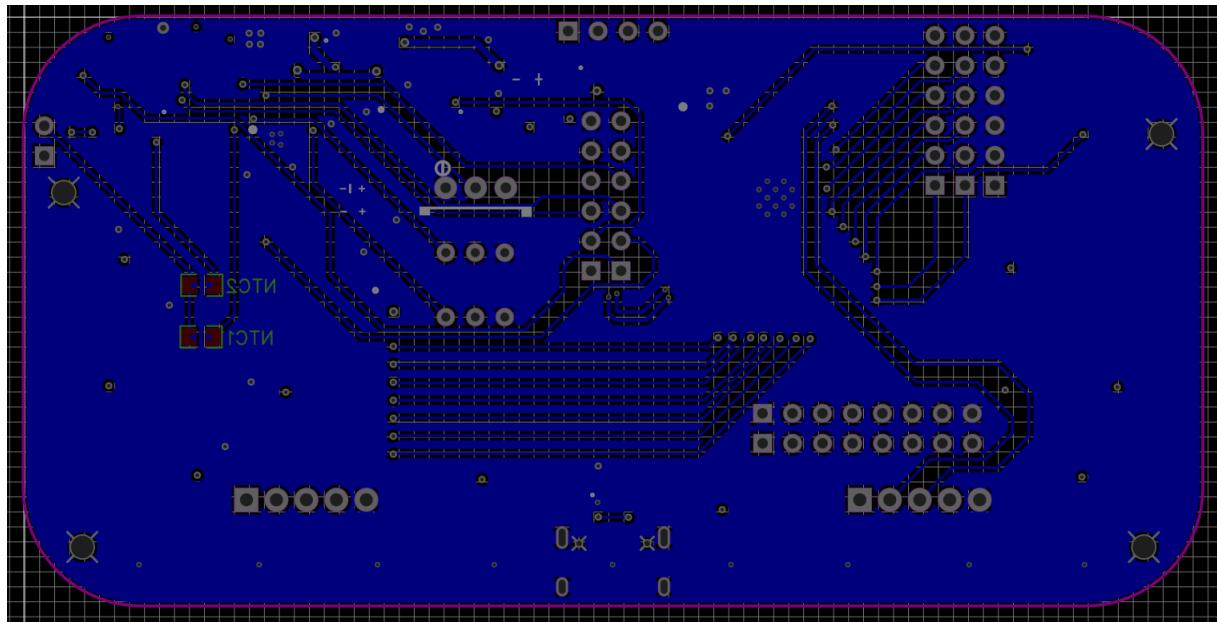


Abbildung 12: "Untere Lage des Printed-Circuit-Boards"

Die untere Schicht der Platine ist mit zwei NTC-Widerständen ausgestattet, die zur Temperaturüberwachung des Akkus dienen.

Das Design nutzt eine 4-Lagen-Platine, um eine saubere Signalführung und optimale elektrische Eigenschaften zu gewährleisten. Die zweite Lage ist vollständig als Massefläche (GND) ausgeführt, während die dritte Lage der 3.3V-Versorgungsspannung dient. Dies minimiert elektromagnetische Störungen und sorgt für eine stabile Spannungsversorgung der empfindlichen Schaltungen. Zusätzlich wurden Abblockkondensatoren mit Werten von 100 nF bis 1 µF so nah wie möglich an den integrierten Bauteilen positioniert, um die Spannungsversorgung zu stabilisieren und hochfrequente Störungen zu minimieren.

Die Designregeln, wie der Leiterbahnabstand und die Größe der Vias, wurden unter Berücksichtigung der Anforderungen und der spezifischen Eigenschaften des Herstellers sorgfältig gewählt. Leitungen, durch die höhere Ströme fließen, wurden mit deutlich breiteren Leiterbahnen ausgestattet als „normale“ Signalleitungen, wie beispielsweise die von den GPIO-Pins. Für die Vias wurden drei unterschiedliche Größen vorgesehen, je nach Verfügbarkeit von Platz und den spezifischen Anforderungen der Schaltung:

- (Durchmesser 0,610 mm; Bohrdurchmesser 0,305 mm): Diese Vias kommen zum Einsatz, wenn ausreichend Platz vorhanden ist.
- (Durchmesser 0,450 mm; Bohrdurchmesser 0,300 mm) mit Epoxyfüllung: Diese Vias werden in besonders platzbegrenzten Bereichen verwendet, wie etwa im mittleren Kupferbereich des BQ24075-Bauteils.

Für die USB-Datenleitungen sind spezielle Designregeln festgelegt, die im Kapitel 6.1 berechnet und beschrieben sind.

Auf dem PCB sind zentrale Komponenten wie der ESP32-Mikrocontroller, der MAX17055-Ladestandüberwachungschip und der BQ24075-Ladecontroller integriert. Die Anordnung der Bauteile wurde so optimiert, dass der MAX17055 in direkter Nähe zum Akku platziert ist, um präzise Spannungs- und Strommessungen zu ermöglichen. Der BQ24075 folgt in der Nähe des MAX17055, um eine effiziente Energieübertragung und Ladeüberwachung sicherzustellen. Die GPIOs des Mikrocontrollers ESP32 sind zusätzlich mit Header-Pins verbunden, um die Fehlersuche und Messungen während der Entwicklungs- und Testphase zu erleichtern.

Für die mechanische Stabilität verfügt die Platine über vier Bohrlöcher, die eine sichere Montage im Gehäuse ermöglichen. Zusätzlich sind Anschlüsse wie ein USB-C-Stecker, ein JST-Konnektor für den Akku und Pogo-Pins für die Verbindung mit externen Geräten integriert. Das Design berücksichtigt begrenzte Platzanforderungen und verwendet die kleinsten möglichen Bauteile, die dennoch händisch auf der Platine platziert werden können. Dies erleichtert die Prototypenfertigung und spätere Reparaturen.

6.1 Führung der USB-Datenleitungen

Die Datenleitungen D+ und D- des USB-Signals müssen als differentielles Paar mit einer Impedanz von $90 \Omega \pm 10\%$ ausgelegt werden [13]. Das Routing eines differentiellen Paars kann mithilfe von E-CAD-Software problemlos durchgeführt werden. Allerdings erfordert dies die Berechnung der Leiterbahnbreite und des Abstands zwischen den Leiterbahnen. Diese Parameter hängen von mehreren Faktoren ab, einschließlich des Typs des PCB-Materials, der angestrebten Impedanz, der Signalebene und der zugehörigen Referenzebene. Die notwendigen Material- und Designparameter können auf der Website des Leiterplattenherstellers, in diesem Fall [JLCPCB.com](#), eingesehen werden. Darüber hinaus stellt der Hersteller einen Impedanzrechner zur Verfügung, der die Auswahl des tatsächlich verwendeten Materials ermöglicht und so eine präzise Anpassung an die Produktionsanforderungen unterstützt.

The screenshot shows the JLCPCB Impedance Configurator interface. At the top, there are dropdown menus for 'Layers' (4), 'PCB Thickness' (1.6), 'Inner Copper Weight' (0.5oz), 'Outer Copper Weight' (1oz), and 'Unit' (mm). Below this is a section titled 'Impedance Configure' with a table:

Impedance (Ω)	Type	Signal Layer	Top Ref	Bottom Ref	Trace Spacing (mm)	Impedance trace to copper (mm)
90	Differential Pair (Non coplanar)	L1	/	L2	0.3	/

Below the table are buttons for '+ New Impedance', 'Duplicate Impedance', and 'Calculate'. The 'Calculate' button is highlighted in blue. At the bottom, there are two tabs: 'Finished thickness 1.56mm±10%' and 'JLC04161H-3313A(Special/Finished thickness 1.58mm±10%)'. The second tab is selected and shows a table of layer properties:

Impedance (Ω)	Type	Signal Layer	Top Ref	Bottom Ref	Trace Width	Trace Spacing	Impedance trace to copper
90	Differential Pair (Non coplanar)	L1	/	L2	0.3343	0.3000	/

Below this table is another table showing the physical properties of each layer:

Layer	Material	Thickness (mil)	Thickness (mm)
L1	Outer Copper Weight 1oz	1.38	0.0350
Prepreg	7628, RC 49%, 8.6 mil	8.28	0.2104
L2	Inner Copper Weight	0.60	0.0152
Core	1.1mm H/HOZ with copper	41.93	1.0650
L3	Inner Copper Weight	0.60	0.0152
Prepreg	7628, RC 49%, 8.6 mil	8.28	0.2104
L4	Outer Copper Weight 1oz	1.38	0.0350

Abbildung 13: "Berechnung der Leiterbahnbreite mithilfe von Herstellungsparametern und gewünschter Impedanz mittels eines Online-Kalkulators" [14]

Abbildung 13 zeigt den Impedanzkalkulator von JLCPCB, der für die Berechnung der Designparameter des differentiellen Paars verwendet wurde. Da es sich um das Routing eines differentiellen Paars handelt, wurde als Typ „Differential Pair“ gewählt, wobei eine gewünschte Impedanz von 90Ω spezifiziert wurde. Wie in Abbildung 4 dargestellt, befindet sich das Signal auf der obersten Lage der Leiterplatte. Daher wurde im Kalkulator unter „Signal Layer“ die Einstellung „L1“ gewählt. Eine obere Referenzebene („Top-Ref“) existiert nicht, da das Signal auf der obersten Lage liegt. Als untere Referenzebene („Bottom-Ref“) wurde die Masseebene (GND) auf der zweiten Lage angegeben, daher wurde „L2“ ausgewählt. Die Datenleitungen des USB-Signals (die Leiterbahnen, die von der USB-Schnittstelle geradlinig bis zum Zentrum der Platine verlaufen) müssen, wie in Abbildung 11 zu erkennen, überkreuzt werden, um die von der ESP32-Spezifikation definierten Pins D+ (GPIO20) und D- (GPIO19) zu erreichen. Für diese Überkreuzung sind Vias erforderlich. Der Abstand zwischen den beiden Leitungen wurde daher so dimensioniert, dass die Vias ausreichend Platz nebeneinander haben. Ein Abstand von 0,3 mm erwies sich dabei als ausreichend. Für die Herstellung wurde das Standard-Material-Stackup „JLC04161H-7628“ ausgewählt. Dieses Stackup besitzt zwar im Vergleich zu anderen verfügbaren Stackups eine hohe dielektrische Konstante, aber für niederfrequente Anwendung wie diese, reicht er völlig aus. Nach Eingabe der Parameter und Auswahl des entsprechenden Materials wurde die Berechnung durch Klicken auf „Calculate“ durchgeführt. Der Impedanzkalkulator gab eine Leiterbahnbreite von 0,3343 mm aus. Dieser Wert wurde anschließend in die Einstellungen der E-CAD-Software für das Routing des differentiellen Paars übernommen. Die Datenleitungen wurden unter Berücksichtigung dieser Parameter bis zu den Vias und weiter zu den entsprechenden Pins des ESP32 geroutet.

7 3D-Modellieren und Drucken vom Gehäuse

Die Entwicklung eines geeigneten Gehäuses für den IoT-Controller ist ein wichtiger Schritt, um die Funktionalität und Benutzerfreundlichkeit des Geräts sicherzustellen. Im Folgenden werden die Herausforderungen und Herangehensweisen beschrieben, die bei der Erstellung des 3D-Modells und beim Druck des Gehäuses berücksichtigt wurden.

Ursprünglich war geplant, die aus der ECAD-Software exportierten 3D-Modelle direkt in die CAD-Software zu importieren, um die Genauigkeit der Platzierungen zu

maximieren. Allerdings war das exportierte 3D-Modell beschädigt und konnte auch mit gängigen Reparaturwerkzeugen nicht wiederhergestellt werden. Dies führte dazu, dass das Design des PCBs und des Pogo-Konnektors manuell in der CAD-Software nachmodelliert werden musste. Aufgrund zeitlicher Einschränkungen wurde entschieden, das Gehäuse ausschließlich ohne Joysticks zu entwickeln. Dieser Ansatz ermöglichte eine Konzentration auf die wesentlichen Funktionen des Geräts und reduzierte die Komplexität der Gehäusegestaltung.

Das Gehäuse wurde so ergonomisch wie möglich gestaltet, um eine komfortable Nutzung zu gewährleisten. Gleichzeitig wurde das Gehäuse so kompakt wie möglich gehalten, um den Anforderungen an Mobilität und Effizienz gerecht zu werden.

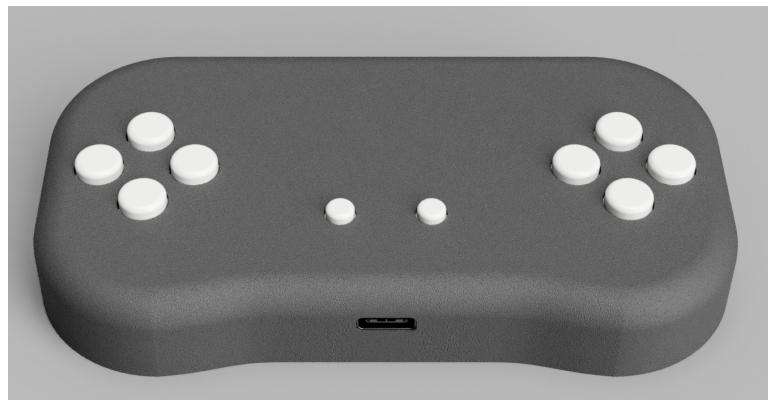


Abbildung 14: "3D-Modell des Gehäuses des IoT-Controllers"

In Abbildung 14 ist das gerenderte 3D-Modell des Gehäuses zu sehen. Es zeigt das externe Erscheinungsbild, das unter ergonomischen und funktionalen Gesichtspunkten gestaltet wurde.



Abbildung 15: "3D-Modell des IoT-Controllers mit allen Komponenten"

Abbildung 15 präsentiert eine 3D-Darstellung des gesamten IoT-Controllers, bei der alle internen Komponenten in ihrer vorgesehenen Position innerhalb des Gehäuses dargestellt sind. Diese Darstellung zeigt, wie die Bauteile aufeinander abgestimmt wurden, um eine kompakte und effiziente Anordnung zu erreichen.



Abbildung 16: "Das 3D-gedruckte Gehäuse des IoT-Controllers"

Das fertige 3D-Modell des Gehäuses wurde mit einem FDM-3D-Drucker ausgedruckt. Als Material wurde PLA verwendet. Nach dem Druck wurden notwendige Nachbearbeitungen vorgenommen, wie das Entfernen von Stützstrukturen und das Glätten der Oberflächen. Die Abbildung 16 zeigt das Endergebnis des Drucks.

8 Bestückung und Lötprozess der Leiterplatte

1. Sortieren der Bauteile

Das Sortieren der Bauteile nach spezifischen Kriterien, wie Art, Größe oder Funktion, ist ein wesentlicher Schritt, um den Bestückungsprozess effizient und zeitsparend zu gestalten. Durch eine systematische Anordnung der Bauteile wird verhindert, dass Zeit mit der Suche nach einzelnen Komponenten verloren geht. Eine gezielte Sortierung trägt zudem dazu bei, Fehlerquellen zu minimieren, da die Bauteile in der benötigten Reihenfolge und schnell zugänglich vorliegen,

2. Befestigung der Leiterplatte auf eine Platform

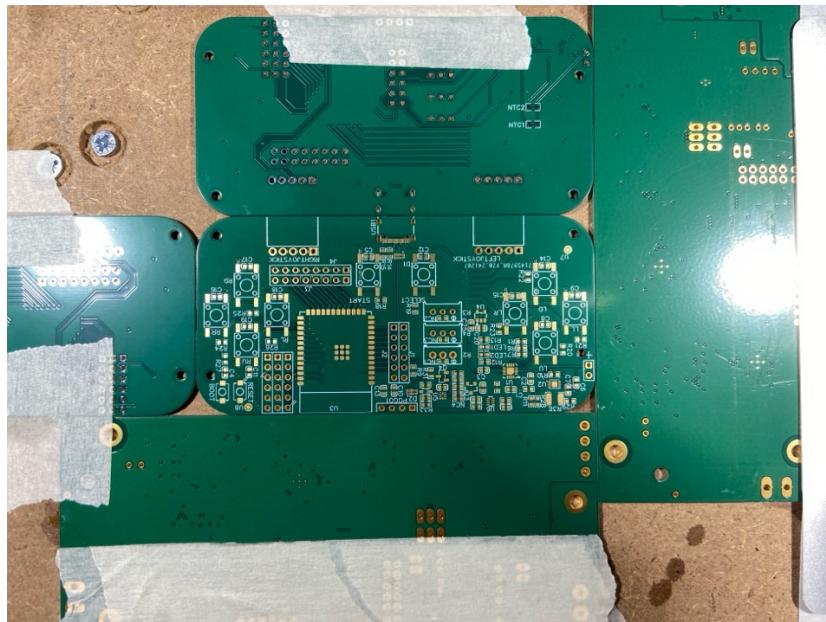


Abbildung 17: "Befestigte Leiterplatte"

Die Befestigung der Leiterplatte auf einer stabilen Plattform gewährleistet eine präzise und sichere Positionierung während des Bestückungs- und Lötprozesses. Durch die Fixierung der Platine wird die Beweglichkeit reduziert, was zu einer höheren Genauigkeit bei der Platzierung der Bauteile und der Durchführung des Lötorgangs führt. Wie in Abbildung 17 dargestellt, ermöglicht diese Methode nach Abschluss des Bestückungsprozesses das einfache Entfernen der fertigen Leiterplatte und das Ersetzen durch eine neue.

3. Reinigung der Leiterplatte und des Stencils von Staub und Fett mit Isopropanol

Die Platine muss vor dem Bestückungs- und Lötprozess gereinigt werden, um sicherzustellen, dass keine Verunreinigungen wie Staub, Öl oder Rückstände von vorherigen Fertigungsschritten die Qualität der Lötverbindungen beeinträchtigen. Solche Verunreinigungen können zu unzuverlässigen Lötstellen, schlechter Leitfähigkeit oder sogar zu Kurzschlüssen führen.

4. Platzieren des Stencils auf die Leiterplatte

Das Stencil dient dazu, die Lötpaste präzise und gleichmäßig auf den vorgesehenen Kontaktflächen der Leiterplatte aufzutragen. Durch das exakte Platzieren des Stencils auf der Platine wird sichergestellt, dass die Paste nur an den spezifischen Lötstellen aufgetragen wird.

5. Auftragen von der Lötpaste



Abbildung 18: "Auftragen der Lötpaste auf die Leiterplatte"

Die Lötpaste ist auf einer Seite des Stencils aufgetragen und anschließend mithilfe eines flachen, dünnen Kunststoffwerkzeugs (ähnlich einer Kreditkarte) gleichmäßig verteilt worden. Dieser Vorgang sorgt dafür, dass die Paste in den Aussparungen des Stencils haften bleibt und eine präzise, gleichmäßige Schicht auf den Lötpads der Leiterplatte aufgebracht wird. Die Abbildung 18 zeigt, wie die Lötpaste verteilt wird.

6. Abziehen des Stencils

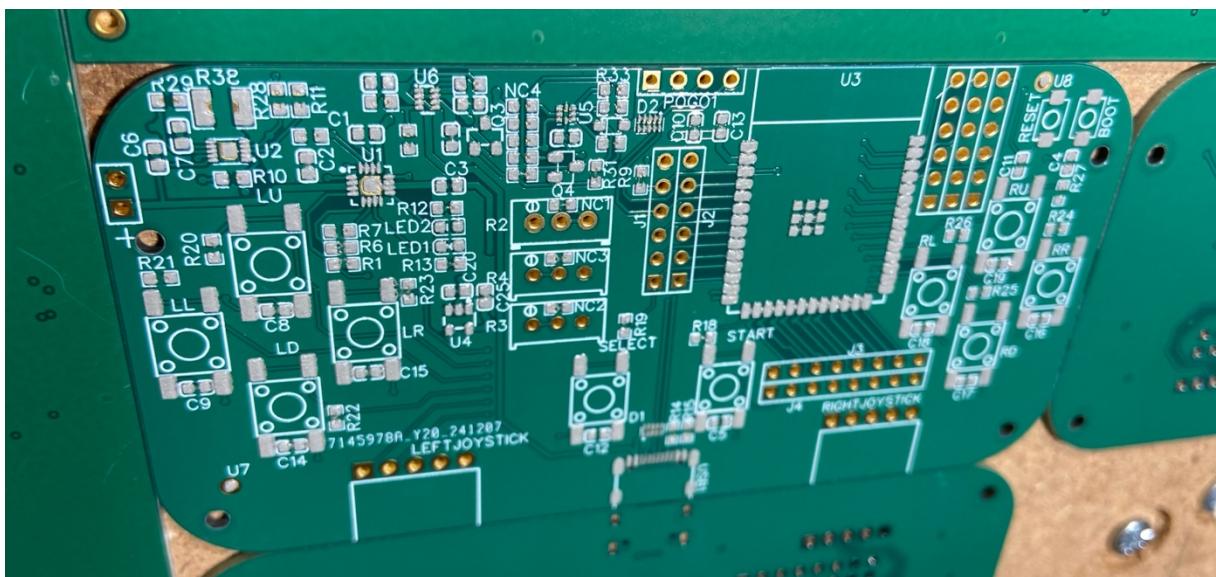


Abbildung 19: "Leiterplatte mit aufgetragener Lötpaste"

Nach dem Abziehen des Stencils ist das Ergebnis in Abbildung 19 dargestellt. Es ist von entscheidender Bedeutung, dass beim Auftragen der Lötpaste eine angemessene Menge verwendet wird, sodass keine überschüssige Paste auf den Leiterbahnen oder benachbarten Pads zurückbleibt. Ebenso sollte darauf geachtet werden, dass sich die Lötpasteflächen nicht übermäßig überkreuzen, da dies zu ungewollten Kurzschlüssen oder fehlerhaften Lötverbindungen führen kann.

7. Bestücken der Leiterplatte

Die Bauteile sind einzeln mit einer feinen Pinzette aufgenommen und sorgfältig auf den vorgesehenen Positionen auf der Leiterplatte platziert. Währenddessen ist das Layout im ECAD-Software verwendet, um die Bauteile zu identifizieren.

8. Reinigung des Stencils und des Arbeitsbereichs nach dem Auftragen der Lötpaste

Das Stencil und der Arbeitsbereich müssen nach dem Auftragen der Lötpaste gründlich gereinigt werden, um überschüssige Paste zu entfernen. Diese Reinigung ist notwendig, um eine saubere und präzise Aufbringung der Lötpaste auf die nächsten Leiterplatten zu gewährleisten und Verunreinigungen zu vermeiden.

9. Leiterplatte in Reflow-Ofen platzieren



Abbildung 20: "Bestückte Leiterplatte im Reflow-Ofen"

Der Reflow-Ofen wird verwendet, um die Lötpaste zu schmelzen und somit eine dauerhafte Verbindung zwischen den Bauteilen und den Leiterplattenspuren herzustellen. Die bestückte Leiterplatte ist vorsichtig in den Ofen eingelegt, wobei die Temperaturprofile des Ofens genau auf die Eigenschaften der Lötpaste und der verwendeten Bauteile abgestimmt sind. Die Abbildung 20 zeigt wie die bestückte Leiterplatte im Reflow-Ofen platziert ist.

9 Durchführung von Funktionstests und Fehlerbehebungen

Nach dem Löten sind die Lötstellen sorgfältig überprüft und getestet worden. Nachdem sichergestellt wurde, dass keine Kurzschlüsse vorliegen und keine unerwarteten Verbindungen gemessen wurden, wurde der IoT-Controller vorsichtshalber zunächst über den USB-Konnektor mit einer Spannungsquelle verbunden. Diese Spannungsquelle war auf eine Stromstärke von 200 mA begrenzt, was der im Normalbetrieb erwarteten Stromstärke entspricht. Sobald bestätigt war, dass sich das System unter diesen Bedingungen normal verhielt, wurde der IoT-Controller an den USB-Port eines PCs angeschlossen, um die Software zu installieren.

Bei der anschließenden Analyse der Platine sowie durch Tests der einzelnen Funktionalitäten stellte sich heraus, dass das Design der Platine Fehler enthält. Die Funktionstests und die Fehler werden in den folgenden Kapiteln beschrieben, zusammen mit Vorschlägen zur Verbesserung für zukünftige Iterationen.

9.1 Funktionstests des Ladeüberwachungs-Bauteils

Die Funktionalität des Ladeüberwachungsbauteils *MAX17055* wurde anhand des im Kapitel 10.1 dargestellten Codes getestet. Zunächst konnte erfolgreich eine I₂C-Verbindung mit dem Bauteil hergestellt werden, die ein zuverlässiges Auslesen und Schreiben von Daten ermöglichte. Der wichtigste auslesbare Parameter des Bauteils ist der „State of Charge“ (Akkuladestand in %), der zusammen mit der verbleibenden Kapazität des Akkus in mAh ermittelt wird.

Um die Genauigkeit des Bauteils zu validieren, wurde der ausgelesene Akkuladestand mit einem kommerziellen Lithium-Ionen-Akkuladegerät verglichen. Diese Tests wurden an vier verschiedenen Akkus mit einer Kapazität von 1100 mAh durchgeführt. Es wurde festgestellt, dass die vom *MAX17055* gemessenen Werte und die vom Ladegerät gemessenen Werte nur eine Abweichung von ± 5 mAh aufwiesen. Dies entspricht

einer Abweichung von etwa $\pm 0,5\%$, was für die geplante Anwendung mehr als ausreichend ist.

9.2 Funktionstests des Ladesteuerung-Bauteils

Beim Anschließen des Systems an eine 5V-Spannungsquelle leuchtet eine helle, rote LED, die über den *PGOOD*-Pin anzeigt, dass eine gültige Eingangsquelle erkannt wurde. Sobald zusätzlich der Akku angeschlossen wird, aktiviert sich eine weitere grüne LED am *CHG*-Pin, die den gestarteten Ladevorgang signalisiert.

Allerdings ergaben die Messungen des Ladeüberwachungsbauteils *MAX17055*, dass die Akkukapazität nur sehr langsam oder gar nicht anstieg. Nach einer detaillierten Analyse des eingestellten Lademodus und des Designs der Platine wurde festgestellt, dass die *EN1*- und *EN2*-Pins des Ladecontrollers jeweils mit Pull-Down-Widerständen beschaltet waren, jedoch keine Verbindung zu den GPIOs des Mikrocontrollers aufwiesen. In Abbildung 3 sind die Leitungen *EN1_U1* und *EN2_U1* zu erkennen, die für die Verbindung zum Mikrocontroller vorgesehen waren. Diese Leitungen waren jedoch im Schaltplan und PCB-Design unvollständig, da keine tatsächliche Verdrahtung oder Zuweisung eines GPIOs vorgenommen wurde.

Obwohl die Pfeile *EN1_U1* und *EN2_U1* im Schaltplan vorhanden waren, generierte die verwendete ECAD-Software keine Fehlermeldung. Dies führte dazu, dass das Routing der Leitungen und die Zuweisung zu GPIOs übersehen wurden.

Verbesserungsmaßnahmen:

- Im Schaltplan geeignete GPIOs des Mikrocontrollers für *EN1* und *EN2* auswählen und zuweisen.
- Das PCB-Design entsprechend anpassen, um die Leitungen der Pins *EN1* und *EN2* korrekt mit ihren jeweiligen GPIOs zu verbinden.

Um die Funktionalität des *BQ24075*-Chips auf der bestehenden Platine weiterhin sicherzustellen, wurden die fehlenden Verbindungen zu den GPIO-Pins mithilfe eines dünnen Kupferdrahtes manuell hergestellt.

9.3 Funktionstests der ESP-NOW Verbindung

Für den Empfang der Controller-Daten wurde ein ESP32-Development-Board mit einem integrierten OLED-Display eingesetzt, das die empfangenen.

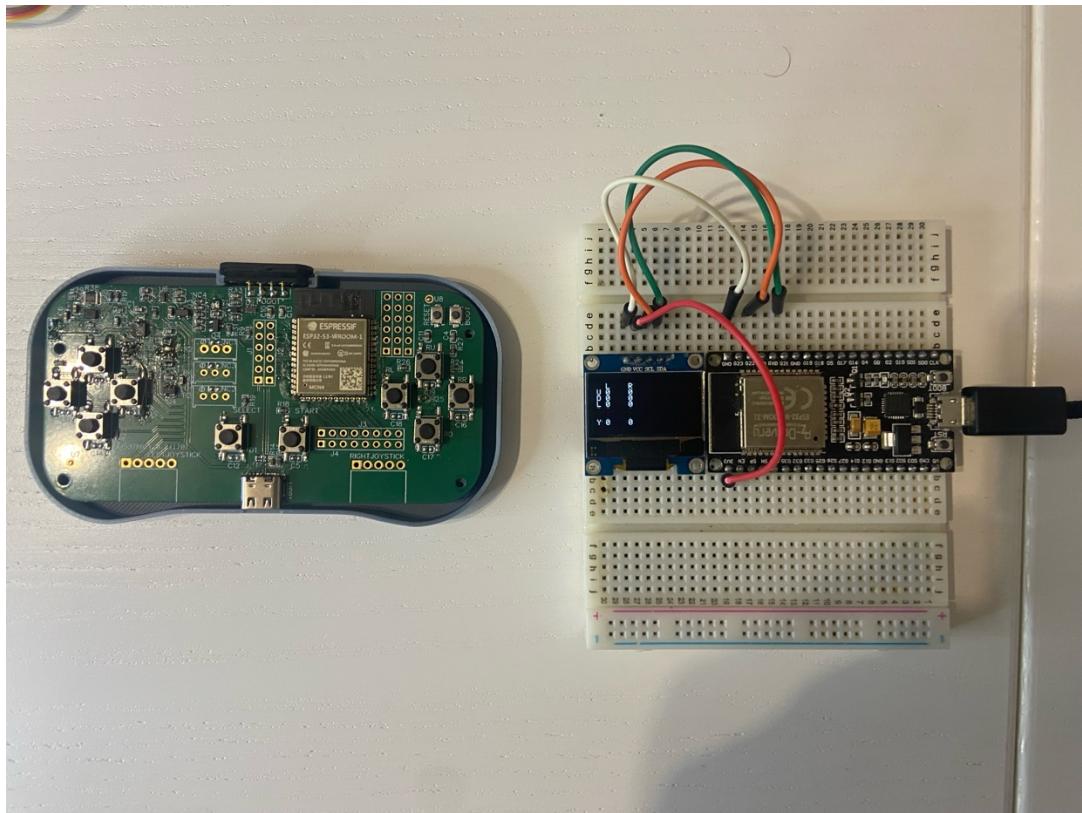


Abbildung 21: "Versuchsaufbau für die ESP-NOW Verbindung. Ein IoT-Controller und ein Empfänger mit einem OLED-Display"

Beim Betätigen eines Tasters am IoT-Controller wird ein Interrupt ausgelöst. Dieser Interrupt ruft eine ESP-NOW-Funktion auf, die die aktuellen Daten des Controllers an den Empfänger sendet. Der Empfänger, der kontinuierlich auf neue Daten hört, empfängt diese und aktualisiert das OLED-Display entsprechend.

Der Testaufbau erwies sich als erfolgreich. Beim Drücken eines Tasters am IoT-Controller wurden die gesendeten Daten ohne spürbare Verzögerung vom Empfänger empfangen und das OLED-Display aktualisierte die entsprechenden Werte sofort.

9.4 Funktionstests des Pogo-Konnektors

Der Ladevorgang über den Pogo-Konnektor wurde durch den Anschluss einer externen Spannungsquelle und die Freigabe des Ladevorgangs durch einen Impuls des Mikrocontrollers getestet. Der Controller wurde dabei zunächst ohne Akku betrieben, um sicherzustellen, dass die Stromversorgung ausschließlich über den Pogo-Konnektor erfolgt.

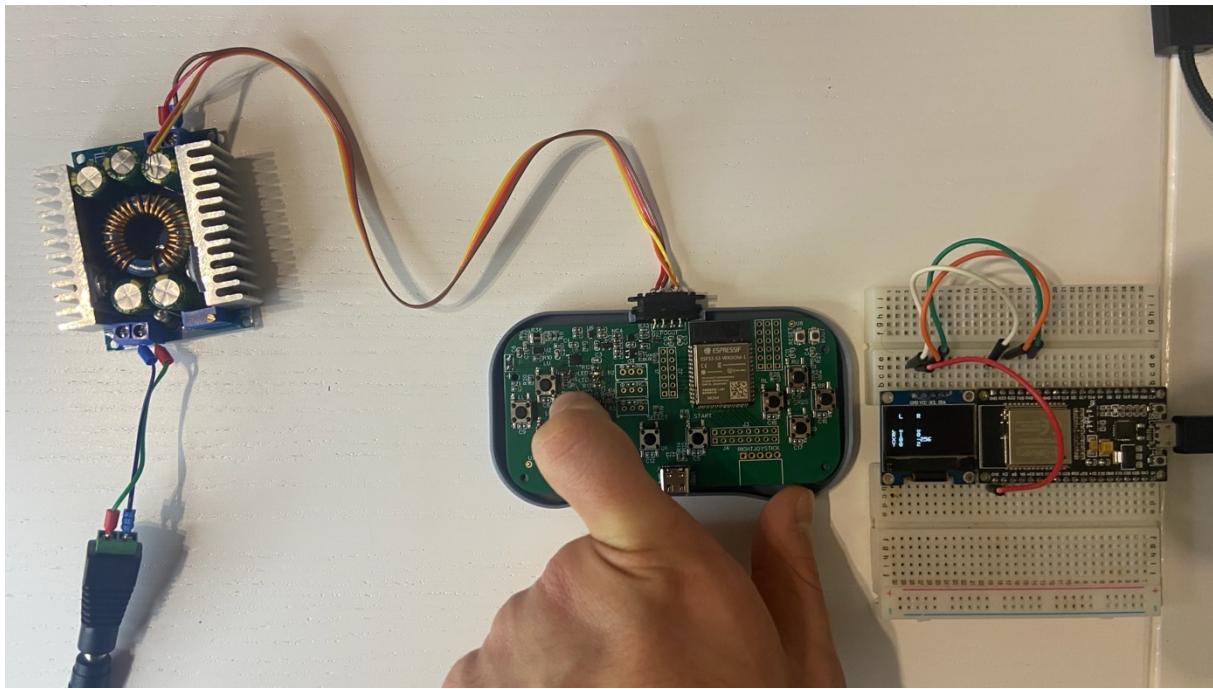


Abbildung 22: "Versuchsaufbau für die Testung des Pogo-Konnektors"

Zur Überprüfung der ordnungsgemäßen Verbindung und Spannungsversorgung wurde der ESP-NOW-Code verwendet, um die Controllerdaten zu senden. Der Test verlief erfolgreich, da der Controller ordnungsgemäß funktionierte und die Daten wie erwartet übertragen wurden.

Im nächsten Schritt wurde ein Akku angeschlossen, woraufhin die grüne LED, die den Ladevorgang signalisiert, aufleuchtete. Dies bestätigte, dass der Ladevorgang über den Pogo-Konnektor korrekt ausgelöst wurde und das System wie geplant funktionierte.

9.5 Funktionstests der Taster

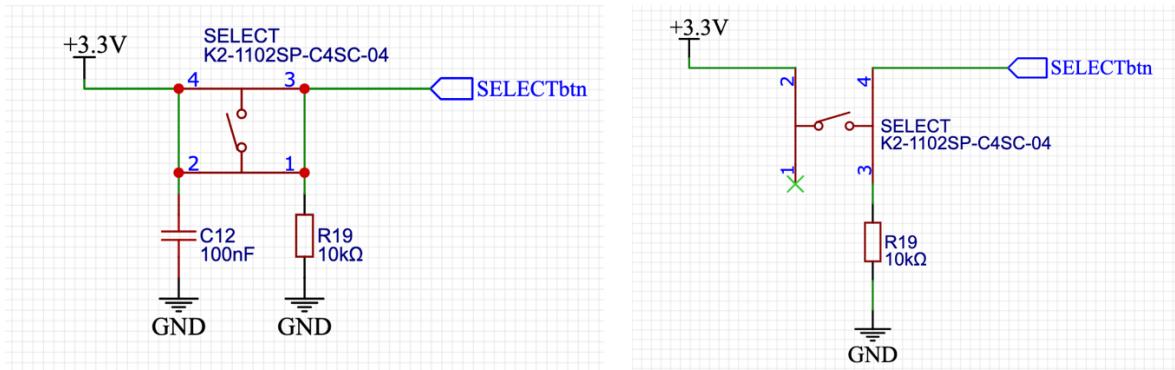


Abbildung 23: "Fehlerhafte (links) und korrigierte (rechts) Taster-Schaltung"

Der erste PCB-Prototyp enthält einen Fehler in den Taster-Schaltungen. Wie in Abbildung 23 dargestellt, ist auf der linken Seite die fehlerhafte Schaltung abgebildet. In dieser Schaltung wurde der Taster irrtümlich kurzgeschlossen, wodurch an den GPIOs der Taster durchgehend ein HIGH-Signal gemessen wurde. Auf der rechten Seite der Abbildung ist die korrigierte Schaltung dargestellt. In dieser Version wird im Normalzustand ein LOW-Signal am GPIO des jeweiligen Tasters gemessen. Erst beim Betätigen des Tasters wird ein HIGH-Signal erzeugt. Zur weiteren Optimierung wurde der ursprünglich vorgesehene Kondensator (in diesem Beispiel C12), der für das Entprellen des Tasters eingeplant war, entfernt. Das Entprellen wird stattdessen softwareseitig umgesetzt, da dies in der Software gelöst werden kann.

Um die Taster auf der bestehenden Platine dennoch verwenden zu können, wurde eine provisorische Anpassung vorgenommen: An jedem Taster wurden zwei Beinchen entfernt, um zu verhindern, dass ein Kurzschluss des Tasters auftritt. Diese Maßnahme stellt eine Übergangslösung dar, bis das Design der Platine entsprechend korrigiert wird, wie im Abschnitt zur Fehleranalyse und Verbesserungsvorschlägen beschrieben.

10 Firmware Design und Realisierung

10.1 Kommunikation mit dem Ladeüberwachung-IC

Der Akkuladestand-Überwachungs-IC MAX17055 nutzt das I²C-Bus-System zur Kommunikation mit dem Host-Mikrocontroller (ESP32). Im Rahmen dieser Kommunikation adressiert der Host ein spezifisches Gerät im Bus über die zugewiesene Slave-Adresse.

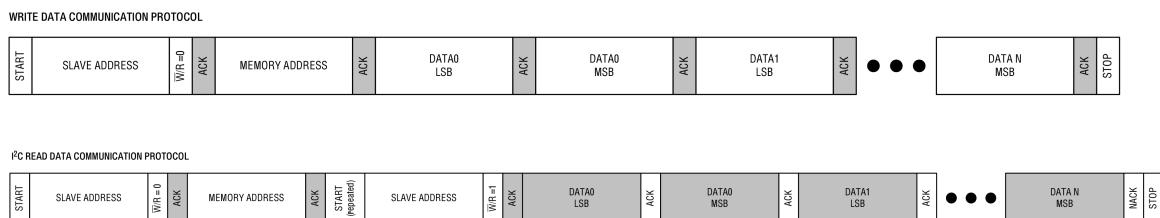


Abbildung 24: "I²C Schreib- und Leseprotokoll" [15]

Die Abbildung 24 illustriert die Unterschiede zwischen dem Schreib- und Leseprotokoll im I²C-Bus-System. Beide Protokolle beginnen mit einem „START“-Signal, gefolgt von der Slave-Adresse sowie einem „nWrite/Read“-Bit, das initial auf Null gesetzt ist. Im nächsten Schritt wird die Ziel-Speicheradresse übertragen. Beim Schreibzugriff folgen

daraufhin die zu schreibenden Daten (beginnend mit dem Least Significant Byte, LSB) an der spezifizierten Speicheradresse. Beim Lesezugriff hingegen erfolgt nach der Speicheradresse ein wiederholtes „START“-Signal, gefolgt von der Slave-Adresse. Dieses Mal ist jedoch das „nWrite/Read“-Bit auf Eins gesetzt. Anschließend antwortet das Slave-Gerät mit den angeforderten Daten. Die Kommunikation erfolgt blockweise und zwischen den Datenblöcken wird ein „ACK“- (Acknowledge-) Signal von dem Datenempfänger gesendet. Dieses Signal dient der gegenseitigen Bestätigung zwischen Host und Slave, dass der vorherige Datenblock erfolgreich übertragen wurde.

```
static esp_err_t init_i2c(void) {
    if (i2c_isinit == false) {
        esp_err_t err_check;

        // Master bus configuration
        i2c_master_bus_config_t i2c_mst_config = {
            .i2c_port = I2C_NUM_0,
            .sda_io_num = SDA_PIN,
            .scl_io_num = SCL_PIN,
            .clk_source = I2C_CLK_SRC_DEFAULT,
            .glitch_ignore_cnt = 7, // typical value is 7
            .intr_priority = 0, // if set to 0, driver will select the default priority (1,2,3)
            .flags.enable_internal_pullup = false,
        };

        // Create new master bus using config and check for error
        err_check = i2c_new_master_bus(&i2c_mst_config, &ret_bus_handle, &bus_handle);
        if (err_check != ESP_OK) return err_check;

        // Device (slave) configuration
        i2c_device_config_t dev_cfg = {
            .dev_addr_length = I2C_ADDR_BIT_LEN_7,
            .device_address = I2C_DEVICE_ADDRESS, // 7MSbit slave address for MAX17055
            .scl_speed_hz = I2C_SCL_SPEED,
            .scl_wait_us = 0,
            .flags.disable_ack_check = false,
        };

        // Add new device on the bus and check for error
        err_check = i2c_master_bus_add_device( bus_handle, &dev_cfg, &ret_handle, &dev_handle );
        if (err_check != ESP_OK) return err_check;

        ESP_LOGI(TAG, "I2C for fuel gauge IC successfully initialized.");

        // Initialize the mutex
        i2c_mutex = xSemaphoreCreateBinary();
        if (i2c_mutex == NULL) {
            ESP_LOGE(TAG, "Failed to create I2C mutex.");
            return ESP_FAIL;
        }
        xSemaphoreGive(i2c_mutex);

        // Set variable true to avoid redundant initialization
        i2c_isinit = true;
    } else {
        ESP_LOGI(TAG, "I2C is already initialized. Skipping initialization.");
    }

    return ESP_OK;
}
```

Codeausschnitt 1: "Initialisierung des I2C Buses"

Um die Kommunikation zwischen dem Host und dem MAX17055-Chip zu ermöglichen, muss die I2C-Funktionalität auf dem Host-Mikrocontroller (ESP32) initialisiert werden.

Die Initialisierung des I2C-Busses sowie des entsprechenden Geräts erfolgt in der Funktion `static esp_err_t init_i2c`, die in Codeausschnitt 1 dargestellt ist.

Der Datentyp `esp_err_t`, welcher aus der Bibliothek `esp_check.h` stammt, wird zur Fehlererkennung eingesetzt. Dieser Datentyp kann verschiedene Werte annehmen, unter anderem:

- **ESP_OK**: Signalisiert eine erfolgreiche Durchführung.
- **ESP_FAIL**: Kennzeichnet einen allgemeinen Fehler.
- **ESP_ERR_NO_MEM**: Gibt an, dass nicht genügend Speicher verfügbar ist.

Weitere Informationen zur Fehlerbehandlung sowie zu weiteren möglichen Rückgabewerten des Datentyps `esp_err_t` können in [16] nachgelesen werden.

In der Funktion `init_i2c` werden sowohl der I2C-Bus als auch das Slave-Gerät mithilfe der Bibliothek `driver/i2c_master.h` konfiguriert und initialisiert. Der Bus nutzt dabei einen der zwei vom ESP32 bereitgestellten I2C-Ports. Im Rahmen der Buskonfiguration werden der SDA-Pin (in diesem Fall GPIO 15) und der SCL-Pin (GPIO 7) spezifiziert, da die Daten- und Takteleitungen auf der Leiterplatte (PCB) fest verdrahtet sind. Für weitere Parameter wie die Clock-Quelle, den Fehlerzähler und die Interrupt-Priorität werden Standardwerte verwendet, da keine spezifischen Anforderungen an die I2C-Kommunikation bestehen.

Im Rahmen der Konfiguration des Slave-Geräts müssen die Slave-Adresse und ihre Länge spezifiziert werden. Für den `MAX17055` ist die Slave-Adresse auf `0x36` festgelegt, mit einer Adresslänge von 7 Bit. Zusätzlich wird die Taktgeschwindigkeit (Clock-Geschwindigkeit) in Hertz definiert, wobei maximal 400 kHz unterstützt werden. Des Weiteren ist die Wartezeit an der SCL-Leitung zu konfigurieren, die im Standardfall auf 0 gesetzt ist. Schließlich wird das Flag `disable_ack_check` auf Null gesetzt, da die ACK-Prüfungen für die Kommunikation mit dem `MAX17055` erforderlich sind (siehe Abbildung 24).

Da der I2C-Bus voraussichtlich von mehreren Funktionen genutzt wird, handelt es sich um eine geteilte Ressource, die vor konkurrierendem Zugriff geschützt werden muss. Um fehlerhafte Zustände zu vermeiden wird der Zugriff durch einen `FreeRTOS-Semaphore` koordiniert. Im Code wird hierfür ein binärer Semaphore erstellt, der den

exklusiven Zugriff auf den Bus sicherstellt. Abschließend wird die globale Variable *i2c_init* auf *true* gesetzt, um sicherzustellen, dass die Initialisierung von I2C-Bus und Slave-Gerät nicht mehrfach durchgeführt wird.

```
565 static esp_err_t writeRegister(uint8_t reg, uint16_t data_wr) {
566     esp_err_t err_check;
567
568     // Array used for splitting data in Low byte and High byte
569     uint8_t data_wr_prep[3];
570     // Save Register to array
571     data_wr_prep[0] = reg;
572     // Low byte
573     data_wr_prep[1] = data_wr & 0xFF;
574     // High byte
575     data_wr_prep[2] = (data_wr >> 8) & 0xFF;
576
577     // Take the mutex
578     if (xSemaphoreTake(i2c_mutex, ( TickType_t ) 10) != pdTRUE) {
579         ESP_LOGE(TAG, "Failed to acquire I2C mutex.");
580         return ESP_FAIL;
581     }
582
583     if (DEBUG) ESP_LOGI(TAG, "Writing 0x%04X to register 0x%02X using writeRegister function.", data_wr, reg);
584     err_check = i2c_master_transmit(i2c_dev: dev_handle, write_buffer: data_wr_prep , write_size: sizeof(uint8_t)*3, xfer_timeout_ms: I2C_WAITTIMEOUT);
585
586     // Release the mutex
587     xSemaphoreGive(i2c_mutex);
588
589     return err_check;
590 }
```

Codeausschnitt 2: "Funktion für das Schreiben von Daten in Register des MAX17055-Chips"

Um auf die Register des MAX17055-Chips zugreifen zu können, wurden die Funktionen *readRegister(Register, Daten)*, *writeRegister(Register, Daten)* und *writeAndVerifyRegister(Register, Daten)* implementiert. Codeausschnitt 2 zeigt die Funktion *writeRegister*, in der die Daten für das Schreibprotokoll (siehe Abbildung 24) vorbereitet und über I2C übertragen werden. Für die Datenvorbereitung wird ein Array mit drei 8-Bit-Elementen erstellt. Das erste Element enthält die Adresse des Registers, das beschrieben werden soll. Das zweite Element entspricht dem Low-Byte der Daten (den ersten 8 Bits), während das dritte Element das High-Byte der Daten (die höheren 8 Bits) enthält. Um den I2C-Bus zu nutzen, muss zunächst der zuvor erstellte FreeRTOS-Semaphore angefordert werden. Sollte dieser nach 10 Ticks weiterhin nicht verfügbar sein, gibt die Funktion den Fehlerwert *ESP_FAIL* zurück. Nach Abschluss der Übertragung wird der Semaphore wieder freigegeben, um anderen Prozessen den Zugriff zu ermöglichen. Treten während der I2C-Übertragung Fehler auf, werden diese entsprechend angezeigt. Andernfalls gibt die Funktion *ESP_OK* zurück, um der übergeordneten Funktion anzuseigen, dass die Übertragung erfolgreich durchgeführt wurde. Die Funktion *readRegister* arbeitet ähnlich wie die Funktion *writeRegister*, wobei anstelle des Schreibens die Daten aus dem angeforderten Register ausgelesen werden. Die Funktion *writeAndVerifyRegister* kombiniert die beiden zuvor beschriebenen Funktionen. Zunächst wird mit der Funktion *writeRegister* ein Schreibvorgang initiiert. Anschließend wird mithilfe der Funktion *readRegister* überprüft, ob die Daten korrekt in das Register geschrieben wurden. Diese Überprüfung stellt sicher, dass der

Schreibvorgang erfolgreich war und die Registerinhalte wie erwartet aktualisiert wurden.

Der MAX17055-Chip muss mithilfe der zuvor implementierten Funktionen initialisiert werden. Laut dem *MAX17055 User-Guide* (siehe [15]) ist es zunächst erforderlich, das Power-On-Reset-Flag im Statusregister zu prüfen. Dieses Flag wird gesetzt, wenn der Chip eine Unterbrechung der Stromversorgung erfährt. Falls das Flag aktiviert ist, müssen alle akkuspezifischen Parameter erneut im Chip gespeichert werden, da der MAX17055 selbst keinen nicht-flüchtigen Speicher enthält.

Zu den wichtigen zu übertragenden Parametern gehören:

- **EXPECTED_BATTERY_CAP**: Die erwartete Kapazität des angeschlossenen Akkus in mAh.
- **END_OF_CHARGE_CURRENT**: Der Stromfluss in mA am Ende eines Laudezyklus. Dieser Wert ist abhängig vom verwendeten Akkulade-IC (in diesem Fall der BQ24075).
- **V_EMPTY_REG_VALUE**: Ein 16-Bit-Wert, der die Spannung des vollständig entladenen Akkus (VEmpty) und den Spannungspegel angibt, bei dem das Empty-Flag wieder gelöscht wird (VRecovery). Dieser Wert muss in einem spezifischen Format übermittelt werden, das im User-Guide definiert ist ([15], S. 21).

Diese Parameter sind im Code als Makros definiert und können projektspezifisch angepasst werden. Die Werte von EXPECTED_BATTERY_CAP und END_OF_CHARGE_CURRENT werden mithilfe der Funktionen *convCapToStFormat* und *convCurrentToStFormat* in die standardisierten Formate des MAX17055-Chips umgerechnet und anschließend übertragen. Der Initialisierungsablauf entspricht den Anforderungen des Software-Implementation-Guides (siehe [17]) und wird in der Funktion *init_fuelGauge* umgesetzt.

Der MAX17055-Chip implementiert den „Model Gauge™ m5 EZ“-Algorithmus, wodurch eine separate Akku-Charakterisierung nicht erforderlich ist [5]. Dieser Algorithmus liefert verschiedene Parameter, die an den Host zurückgegeben werden. Einerseits werden Akkuinformationen bereitgestellt, die direkt dem Benutzer angezeigt werden können, wie etwa der Ladezustand (State of Charge) in Prozent oder die verbleibende

Kapazität in mAh. Andererseits stellt der Algorithmus auch Parameter zur Verfügung, die für die automatische Charakterisierung des Akkus notwendig sind, wie etwa die Anzahl der Ladezyklen, berechnete Gesamtkapazität und andere. Diese Parameter können im Host im nicht-flüchtigen Speicher (NVS) abgelegt werden, sodass sie im Falle eines Power-On-Resets wieder in den Chip geladen werden können. Dieses Verhalten wird durch die Funktionen *loadNVSPParamAfterPOR_fuelGauge()*, die einmal nach dem Boot ausgeführt wird, und *updateNVSPParam_fuelGauge()*, die in regelmäßigen Zeitabständen, zum Beispiel alle 10 Minuten, aufgerufen wird, sichergestellt. Sie sind mit Hilfe der Bibliotheken *nvs.h* und *nvs_flash.h* erstellt. Da die Anzahl der Schreib- und Löschzyklen auf dem NVS Speicher begrenzt ist, wird empfohlen, das zu häufige oder unnötige Schreiben mit der Funktion *updateNVSPParam_fuelGauge()* zu vermeiden.

Die in diesem Dokument beschriebenen Codeausschnitte sowie der vollständige Quellcode sind im Anhang in der Datei *battery_fuelGauge.c* aufgeführt.

10.2 ESP-NOW Programmierung

ESPNOW ermöglicht eine direkte, effiziente Peer-to-Peer-Kommunikation ohne die Notwendigkeit eines Wi-Fi-Access Points. Bevor ESP-NOW verwendet werden kann, muss WLAN initialisiert werden, [18].

```
21 /* WiFi should start before using ESPNOW */
22 void wifi_init(void)
23{
24     ESP_ERROR_CHECK(esp_netif_init());
25     ESP_ERROR_CHECK(esp_event_loop_create_default());
26     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
27     ESP_ERROR_CHECK( esp_wifi_init(config: &cfg) );
28     ESP_ERROR_CHECK( esp_wifi_set_storage(storage: WIFI_STORAGE_RAM) );
29     ESP_ERROR_CHECK( esp_wifi_set_mode(mode: ESPNOW_WIFI_MODE) );
30     ESP_ERROR_CHECK( esp_wifi_start() );
31     ESP_ERROR_CHECK( esp_wifi_set_channel(primary: CONFIG_ESPNOW_CHANNEL, second: WIFI_SECOND_CHAN_NONE) );
32
33 #if CONFIG_ESPNOW_ENABLE_LONG_RANGE
34     ESP_ERROR_CHECK( esp_wifi_set_protocol(ESPNOW_WIFI_IF, WIFI_PROTOCOL_11B|WIFI_PROTOCOL_11G|WIFI_PROTOCOL_11N|WIFI_PROTOCOL_LR) );
35 #endif
36 }
```

Codeausschnitt 3: "WLAN Initialisierung für das ESP-NOW-Protokoll"

Die Funktion *wifi_init* konfiguriert das Wi-Fi-Modul des ESP32. Sie umfasst:

- *esp_netif_init*: Initialisiert das Netzwerkinterface.
- *esp_event_loop_create_default*: Erstellt einen Standard-Ereignisloop.
- *esp_wifi_init*: Initialisiert die Wi-Fi-Bibliothek.
- *esp_wifi_set_mode*: Setzt den Wi-Fi-Modus auf den gewünschten Modus für ESPNOW.

- `esp_wifi_start`: Startet das Wi-Fi-Modul.
- `esp_wifi_set_channel`: Setzt den Wi-Fi-Kanal für die Kommunikation. Die Konfiguration des Kanals ist entscheidend, da ESPNOW nur auf einem bestimmten Kanal arbeitet.

Es besteht die Möglichkeit lange Reichweite zu aktivieren, wenn das entsprechende Konfigurations-Flag gesetzt ist (Zeile 33 bis 35 im Codeausschnitt 3). Dies ermöglicht eine stabilere Kommunikation über größere Entferungen, allerdings auf Kosten von Geschwindigkeit und Bandbreite.

```
43 void espnow_init(void)
44{
45     /* Initialize ESPNOW and register sending and receiving callback function. */
46     ESP_ERROR_CHECK( esp_now_init() );
47
48     ESP_ERROR_CHECK( esp_now_register_send_cb(cb: espnow_send_cb) );
49
50 #if CONFIG_ESPNOW_ENABLE_POWER_SAVE
51     ESP_ERROR_CHECK( esp_now_set_wake_window(CONFIG_ESPNOW_WAKE_WINDOW) );
52     ESP_ERROR_CHECK( esp_wifi_connectionless_module_set_wake_interval(CONFIG_ESPNOW_WAKE_INTERVAL) );
53 #endif
54
55     /* Set primary master key. */
56     //ESP_ERROR_CHECK( esp_now_set_pmk((uint8_t *)CONFIG_ESPNOW_PMK) );
57
58     /* Add broadcast peer information to peer list. */
59     ESP_ERROR_CHECK( espnow_add_peer(mac: BROADCAST_MAC) );
60     ESP_ERROR_CHECK( espnow_add_peer(mac: DEST_MAC) );
61 }
```

Codeausschnitt 4: "ESP-NOW Initialisierung"

In der Funktion `espnow_init` ist die ESP-NOW-Kommunikation eingerichtet. Die Funktion führt folgende Schritte durch:

- `esp_now_init`: Initialisiert die ESPNOW-Kommunikation.
- `esp_now_register_send_cb`: Registriert den Callback für das Senden von Nachrichten. Der Callback `espnow_send_cb` wird aufgerufen, wenn eine Nachricht erfolgreich oder mit einem Fehler gesendet wurde.

Die Callback-Funktion `espnow_send_cb` überprüft den Status der gesendeten Nachricht und protokolliert das Ergebnis.

Das `CONFIG_ESPNOW_ENABLE_POWER_SAVE` Flag (Zeile 50) kann definiert werden, um den Energieverbrauch des Systems zu optimieren. Durch das Setzen dieses Flags wird der sogenannte Energiesparmodus für die ESP-NOW-Kommunikation aktiviert, was den Stromverbrauch während der inaktiven Phasen reduziert. In Zeile 56 wird die Möglichkeit zur Aktivierung der Verschlüsselung der Kommunikation durch Verwendung eines „Primary Master Key“ angesprochen. Diese Funktion ist momentan

jedoch auskommentiert, um die Implementierung zu vereinfachen und den Fokus auf die grundlegenden Kommunikationsmechanismen zu legen.

Um Daten an ein bestimmtes Gerät zu senden, kann nach der Initialisierung die Funktion `esp_now_send` verwendet werden. Diese Funktion ermöglicht es, Nachrichten oder Daten an ein vordefiniertes Gerät innerhalb eines ESP-NOW Netzwerks zu übertragen. Ein Beispiel für die Nutzung dieser Funktion ist in der Datei `gamepad_tools.c` zu finden, speziell in der Funktion `send_gamepad_data`. Diese Funktion wird jedes Mal aufgerufen, wenn ein Taster getätigkt wurde.

Der vollständige ESP-NOW Code ist in der `espnow_handler.c` Datei im Anhang zu finden.

10.3 UART-Kommunikation über den Pogo-Konnektor

Dieser Code implementiert eine UART-Kommunikationsschnittstelle auf einem ESP32-S3-Mikrocontroller mit den Pins GPIO 4 (RX) und GPIO 5 (TX). Es wird regelmäßig geprüft, ob ein Gerät auf UART-Nachrichten antwortet. Wenn eine Antwort erkannt wird, wird ein Interrupt ausgelöst, der einen Impuls an GPIO 16 erzeugt. Der Code beinhaltet drei Funktionen: `init_uart_over_pogo`, `uart_check_task` und `send_pulse_isr`.

Die Funktion `init_uart_over_pogo` konfiguriert:

- UART-Schnittstelle:
 - Baudrate: 115200
 - Datenbits: 8
 - Parität: Keine
 - Stopppbits: 1
 - Flow Control: Deaktiviert
- Interrupt-Pin GPIO 16 als digitale Ausgangsleitung.

Die Hauptlogik wird in der Aufgabe `uart_check_task` implementiert:

1. Testnachricht senden: Es wird eine vordefinierte Nachricht (PING) über UART gesendet.

2. Antwort abwarten: Der Mikrocontroller wartet auf eine Antwort vom UART-Gerät.
Die Antwort wird über den RX-Pin empfangen und im Puffer gespeichert.
3. Erfolgserkennung:
 - Wenn eine Antwort erkannt wird, wird ein Interrupt ausgelöst.
 - Die Funktion `send_pulse_isr` erzeugt dabei einen Impuls am GPIO 16 (HIGH-Signal für 100 ms).
4. Wiederholungsintervall: Wenn keine Antwort erkannt wird, erfolgt ein erneuter Versuch nach 1 Sekunde.

Die Interrupt-Funktion `send_pulse_isr` generiert ein kurzes High-Signal (Übertragungsdauer 100 ms) an GPIO 16. Diese Funktion nutzt die interne Funktion `ets_delay_us` zur genauen Zeitverzögerung.

Der gesamte Code ist in der Datei `uart_over_pogo.c`, die im Anhang zu finden ist.

10.4 Hauptprogramm

Dieses Programm ist für die Initialisierung und den Betrieb eines IoT-Controllers verantwortlich, der verschiedene Komponenten steuert, einschließlich einer Batterieanzeige (Fuel Gauge), der Ladesteuerung, der UART-Kommunikation über den Pogo-Konnektor sowie der ESP-NOW-Kommunikation für die drahtlose Datenübertragung. Hier sind alle oben beschriebene Programme eingefügt und deren Funktionalitäten initialisiert. Der vollständige Code befindet sich in `main.c` Datei im Angang.

11 Anwendungsbeispiel

Als praktisches Anwendungsbeispiel für den IoT-Controller wurde eine 32x32 RGB-LED-Matrix ausgewählt. Diese Matrix ist mit einem 4-Pin-Pogo-Konnektor an ihrer Unterseite ausgestattet, der die Verbindung zum Controller ermöglicht. Mithilfe des Pogo-Konnektors wird nicht nur die Stromversorgung der Matrix, sondern auch die Datenübertragung gewährleistet, wodurch ein nahtloser Betrieb möglich ist.



Abbildung 25: "IoT-Controller gekoppelt mit einer LED-Matrix"

Wie in Abbildung 25 zu sehen ist, kann der IoT-Controller komfortabel über den Pogo-Konnektor mit der LED-Matrix verbunden werden, um den Akku zu laden.

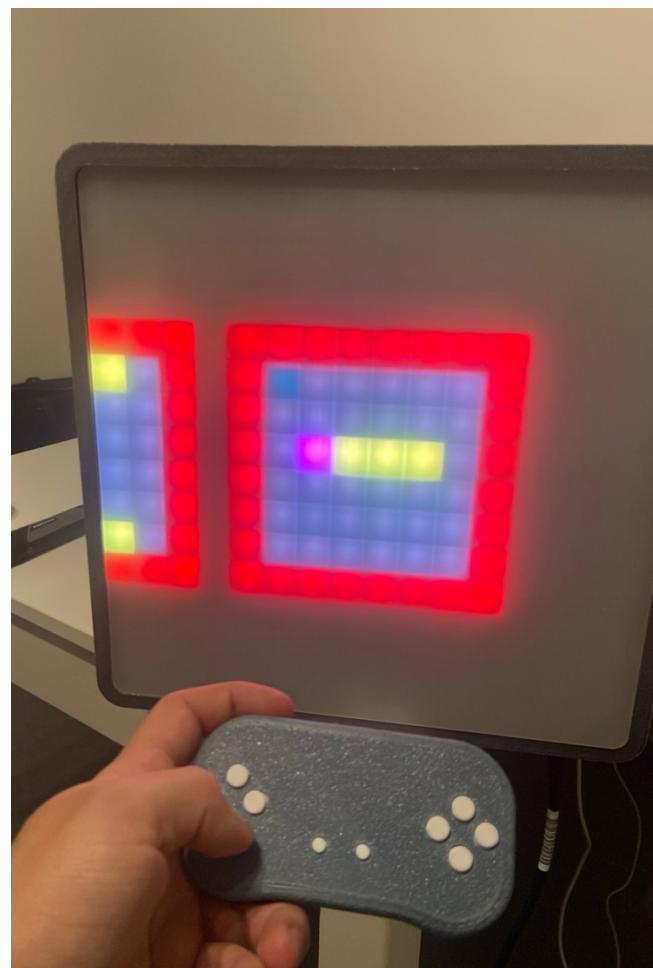


Abbildung 26: "Steuern der LED Matrix mit dem IoT-Controller"

In Abbildung 26 wird demonstriert, wie der IoT-Controller effektiv zur Steuerung der Matrix verwendet wird, indem verschiedene Muster und Animationen angezeigt werden.

12 Reflexion und Ausblick

Die Entwicklung des IoT-Controllers hat gezeigt, dass ein sorgfältiger und iterativer Entwicklungsprozess entscheidend für den Projekterfolg ist. Während die grundlegenden Funktionen wie kabellose Kommunikation, Ladeüberwachung und Spannungsregulierung erfolgreich implementiert wurden, wurden Schwachstellen im PCB-Design identifiziert, die in zukünftigen Iterationen behoben werden sollten. Insbesondere die Verbesserung der Schaltplan-Konsistenz und eine engere Verknüpfung zwischen Software und Hardware bieten Optimierungspotenzial.

Ein wichtiger Aspekt für die Weiterentwicklung wäre die Integration zusätzlicher Funktionen wie die vollständige Unterstützung von Joysticks, eine verbesserte Energieeffizienz durch Low-Power-Modi und erweiterte Sicherheitsfeatures. Langzeittests könnten weitere Erkenntnisse über die Zuverlässigkeit und Belastbarkeit des Systems liefern.

Literaturverzeichnis

- [1 CSL, „USB 4: Universalschnittstelle und Highspeed-Datenkanal,“ 7 Oktober 2024.
] [Online]. Available: <https://www.csl-computer.com/blog/2024/10/07/usb-4-der-highspeed-datenkanal/>. [Zugriff am Dezember 2024].
- [2 robocraze, „Lithium-Ion vs Lithium Polymer Battery,“ [Online]. Available:
] <https://robocraze.com/blogs/post/lithium-ion-vs-lithium-polymer-battery>. [Zugriff am Dezember 2024].
- [3 Padre Electronics, „Lithium polymer battery by capacity,“ [Online]. Available:
] <https://www.batterylipo.com/lithium-polymer-battery-list-by-capacity/#:~:text=Single%20cell%20capacity%20range%3A%2020mA~20000mA.&text=Operating%20temperature%20range%3A%20%2D20°C,20°C~%2B60°C..> [Zugriff am Dezember 2024].
- [4 P. Fundaro, „Impedance Track™ Based Fuel Gauging,“ September 2007.
] [Online]. Available:
https://www.ti.com/lit/wp/sipy002/sipy002.pdf?utm_source=chatgpt.com&ts=1735812686906&ref_url=https%253A%252F%252Fchatgpt.com%252F. [Zugriff am Dezember 2024].
- [5 Maxim Integrated, „MAX17055 Data-sheet,“ [Online]. Available:
] <https://www.mouser.de/datasheet/2/609/MAX17055-3468925.pdf>. [Zugriff am Dezember 2024].
- [6 Texas Instruments, „I2C Bus Pullup Resistor Calculation,“ Februar 2015. [Online].
] Available:
https://www.ti.com/lit/an/slva689/slva689.pdf?ts=1730431601319&ref_url=https%253A%252F%252Fwww.google.com%252F. [Zugriff am November 2024].

- [7 rbts.co, „I²C Related Calculators,“ [Online]. Available:
] <https://rbts.co/tools/calculators/i2c-related-calculators/>. [Zugriff am November 2024].
- [8 Texas Instruments, „Data-sheet BQ2407x Standalone 1-Cell 1.5-A Linear Battery Chargers with Power Path,“ Oktober 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/bq24075.pdf?ts=1736963381905&ref_url=https%253A%252F%252Fpt.mouser.com%252F. [Zugriff am November 2024].
- [9 H. Varya Voronova, „ALL ABOUT USB-C: RESISTORS AND EMARKERS,“ 4 Januar 2023. [Online]. Available: <https://hackaday.com/2023/01/04/all-about-usb-c-resistors-and-emarkers/>. [Zugriff am November 2024].
- [1 Alpha & Omega Semiconductor, „AO3401A Data-sheet,“ September 2018. 0] [Online]. Available: https://www.lcsc.com/datasheet/lcsc_datasheet_2412061733_Alpha---Omega-Semicon-AO3401A_C15127.pdf. [Zugriff am November 2024].
- [1 Texas Instruments, „TPS2116 Data-sheet,“ Januar 2021. [Online]. Available: 1] https://www.ti.com/lit/ds/symlink/tps2116.pdf?ts=1737006329085&ref_url=https%253A%252F%252Fwww.ti.com%252Fpower-management%252Fpower-switches%252Fproducts.html.
- [1 ESPRESSIF, „ESP32-S3-WROOM-1 Data-sheet,“ [Online]. Available: 2] https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf. [Zugriff am November 2024].
- [1 JLCPCB, „PCB Layout Guidelines for USB Type-C,“ 8 Mai 2024. [Online]. 3] Available: <https://jlpcb.com/blog/pcb-layout-guidelines-for-usb-type-c>. [Zugriff am November 2024].
- [1 JLCPCB, „Printed-Circuit-Board Hersteller,“ [Online]. Available: <https://jlpcb.com>. 4] [Zugriff am November 2024].

- [1] Maxim Integrated, „MAX17055 MODELGAUGE M5 EZ USER GUIDE,“
5] Dezember 2016. [Online]. Available: <https://www.analog.com/media/en/technical-documentation/user-guides/max17055-user-guide.pdf>. [Zugriff am Dezember 2024].
- [1] ESPRESSIF, „Error Code and Helper Functions,“ [Online]. Available:
6] https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/esp_err.html. [Zugriff am Dezember 2024].
- [1] Maxim Integrated, „MAX17055 Software-Implementation-Guide,“ [Online].
7] Available: <https://www.analog.com/media/en/technical-documentation/user-guides/max17055-software-implementation-guide.pdf>. [Zugriff am Dezember 2024].
- [1] ESPRESSIF, „ESP_NOW,“ [Online]. Available:
8] https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32s3/api-reference/network/esp_now.html#_CPPv412esp_now_sendPK7uint8_tPK7uint8_t6size_t. [Zugriff am Dezember 2024].
- [1] Nintendo, „Controller im klassischen Design,“ Dezember 2024. [Online]. Available:
9] <https://www.nintendo.com/de-de/Nintendo-Switch-Online/Controller-im-klassischen-Design-1437883.html>.
- [2] Sony, „Playstation Zubehör,“ [Online]. Available: [https://www.playstation.com/de-0\] de/accessories/](https://www.playstation.com/de-0] de/accessories/). [Zugriff am Dezember 2024].
- [2] JLCPCB, „Multilayer high precision PCB's with impedance control,“ [Online].
1] Available: <https://jlcppcb.com/impedance>. [Zugriff am November 2024].
- [2] RAYMING PCB & Assembly, „Full Introduction about Fr4 Dielectric Constant,“
2] [Online]. Available: <https://www.raypcb.com/fr4-dielectric-constant/>. [Zugriff am November 2024].

Anhang

Die Quellcodes, sowie die Schematic und PCB-Design befinden sich aufgrund der Übersichtlichkeit nicht direkt im Anhang, sondern im Ordner mit diesem Dokument. Neure Versionen des Codes und weitere Fortschritte des Projekts werden in <https://github.com/mesha0703/IoTController> gepostet.