Inhaltsverzeichnis

[1. Einleitung 2](#_Toc209344872)

[1.1. Inhalt und Aufbau 2](#_Toc209344873)

[1.2. Ziel der Arbeit 2](#_Toc209344874)

[2. Technische Grundlagen 3](#_Toc209344875)

[2.1. SPI 3](#_Toc209344876)

[2.2. I²C 3](#_Toc209344877)

[2.3. CAN 3](#_Toc209344878)

[2.4. UART 3](#_Toc209344879)

[2.5. Wifi 3](#_Toc209344880)

[2.6. Bluetooth 3](#_Toc209344881)

[3. Systemanalyse und Integration 4](#_Toc209344882)

[3.1. Bestehendes System 4](#_Toc209344883)

[3.2. Anforderungen 8](#_Toc209344884)

[3.3. Integration ins bestehende System 14](#_Toc209344885)

[4. Konzepte 16](#_Toc209344886)

[4.1. Paarvergleiche 16](#_Toc209344887)

[4.2. Nutzwertanalysen 19](#_Toc209344888)

[5. Hardwareentwicklung 20](#_Toc209344889)

[5.1. Schaltungsaufbau 20](#_Toc209344890)

[5.2. PCB-Layout 55](#_Toc209344891)

[6. Software 66](#_Toc209344892)

[7. Test und Inbetriebnahme 79](#_Toc209344893)

[8. Fazit und Ausblick 80](#_Toc209344894)

[8.1. Zusammenfassung der Arbeit 80](#_Toc209344895)

[8.2. Mögliche Weiterentwicklungen 80](#_Toc209344896)

[9. Literaturverzeichnis 81](#_Toc209344897)

[10. Bildverzeichnis 82](#_Toc209344898)

# Einleitung

## Inhalt und Aufbau

## Ziel der Arbeit

# Technische Grundlagen

## SPI

## I²C

## CAN

## UART

## Wifi

## Bluetooth

# Systemanalyse und Integration

## Bestehendes System

Bei dem bestehenden System handelt es sich um ein vollelektrisches Rennsport-Kart für den Einsatz im Jugend Kart Slalom Wettbewerb und Training. Das Kart soll eine Alternative zu vorherrschenden Karts mit Verbrennungsmotor sein und bietet dafür einige Vorteile.

Das Kart ist ausgestattet mit einem kleinen Synchronmotor mit einer Leistung bis 7KW sowie 55 Nm. Diese Leistungsparameter werden dabei auf sehr kleinem Bauraum erzielt, sodass der Motor zentral platziert werden kann, was dem Fahrverhalten sehr zugute kommt. Die Kraftübertragung auf die Hinterachse findet dabei mittels eines Reimens statt, sodass der gesamte Aufbau auch mechanisch simpel bleibt. Angesteuert wird der Motor über CAN, sodass kein Großer Verkabelungsaufwand für Signalleitungen besteht. Versorgt wird das gesamte System von einer 48V Batterie.   
Die Steuerung des Gesamtsystem übernimmt eine Zentrale Steuereinheit (Vehicle Control Unit - VCU). Auf diesem läuft der Code für Leistungsregelung, Kommunikation und Auswertung aller Sensoren. Hierüber ist es möglich das Fahrverhalten des Karts zu steuern über Parameter wie Verteilung des Drehmoments oder die Veränderung von Drehzahlkurven. Hier kommt der große Vorteil des Elektroantriebs zum Tragen, da hierüber verschiedene Leistungsstufen definiert werden können. Wo vorher unterschiedliche Verbrennungsmotoren eingesetzt werden mussten, um unterschiedliche Leistungsklassen zu definieren, ist bei unserem Kart lediglich Softwareänderung notwendig, um jede Leistungsstufe zwischen 3 und 7 KW zu ermöglichen. Über den Code wird zudem sichergestellt, dass über den gesamten Entladezyklus des Akkus dieselbe gleichbleibende Leistung zur Verfügung gestellt wird. Das ist für den Wettbewerb unerlässlich, da hier alle Teilnehmer mit denselben Karts antreten. Die VCU übernimmt zudem sämtliche Kommunikation mit zum Beispiel dem Akku und Auswertung aller Signale wie von den Knöpfen der zentralen Steuereinheit.

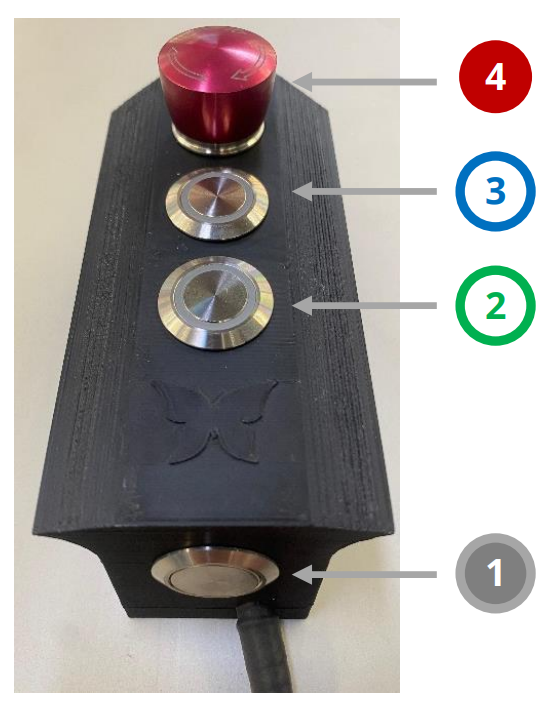
Die Steuerung des Karts erfolgt über eine Zentrale Steuereinheit. Diese besteht aus 3 Knöpfen mit LED-Anzeige und einem Notausknopf. Ein Knopf (Knopf Nr. 1) fungiert als Hauptschalter für das Gesamtsystem. Über ihn wird dem Akku signalisiert den Ausgang aktiv zu schalten und das System mit 48V zu versorgen. Knopf 2 aktiviert den Fahrmodus. Die Aktivierung des Fahrmodus ist dabei an eine Sicherheitsüberprüfung des Systems gekoppelt ohne welche ein Start nicht möglich ist. Dasselbe gilt für den Knopf Nr. 3, jedoch aktiviert dieser den Rückwärtsfahrmodus. Hauptsächlich unterscheiden sich die Knöpfe durch die LED- welche dem Fahrer und Kunden Rückmeldung über den Zustand des Karts geben soll. So zeigen verschiedene Kombinationen aus Leuchtenden und blinkenden LED verschiedene Sati des Karts an wie den Aktivierten Fahrmodus, eine entladene Batterie oder den ausgelösten Notausknopf. Dieser Notausknopf ist das Hauptsicherheitsfeature des Karts. Dabei handelt es sich um einen Normally Closed Druckknopf mit Verriegelung, mit welchem der Fahrer Im Falle eines Notfalls das Kart jederzeit ausschalten kann. Dabei wird nicht nur softwareseitig der Fahrmodus beendet, sondern tatsächlich der Ausgang der Batterie passiv geschalten, sodass das Fahrzeug auch elektrisch in einen sicheren Zustand übergeht, zum Beispiel nach einem Crash, bei dem unter Umständen Kabel gerissen sind oder Kontakte offen liegen. Aus diesem Grund ist der Notaus auch Kabelbruchsicher realisiert, das Bedeutet das sowohl das Drücken des Knopfes als auch eine zerstörte Signalleitung die Aktivierung des Notaus zur Folge hat. Zusätzlich ist diese Steuereinheit mit einem Buzzer ausgestattet, welcher dem Kunden und Fahrer akustische Rückmeldung bei jeder Zustandsänderung des Karts liefert, zum Beispiel bei Aktivierung des Fahrmodus oder einem ausgelösten Notaus.

Abbildung 1: Steuereinheit SMS Revo SL

Zusätzlich zu der Grundfunktion des Karts zum Einsatz im Slalomsport gibt es ein Erweiterungspaket, die Option 1, welche das Kart je nach Wunsch um verschiedene Funktionen erweitert. Die Hauptfunktionalität ist ein RFID-Reader. Gemeinsam mit dem Kauf und Einbau der Option 1 erhält der Kunde RFID-Chips, auf welchen Unterschiedliche Leistungsmodi gespeichert sind. Durch Auslesen des jeweiligen Chips erhält das Steuergerät einen neuen Leistungswert und schaltet in den entsprechenden Modus um. So ist es für den Kunden möglich die Karts in unterschiedlichen Altersgruppen mit unterschiedlichen Leistungsklassen einzusetzen, ohne dass zwischendurch ein Servicetechniker, das Kart flashen muss oder der Kunde für jede Stufe ein eigenes Kart braucht. Der RFID-Reader besitzt dabei ein eigenes CAN-Interface und schickt die übertragenen Daten direkt per CAN2 an die VCU.   
Eine weitere Funktionalität ist eine Statusleuchte. Diese Statusleuchte wird direkt mit dem Steuergerät verbunden, da diese direkt die LED schaltet. Die LED hat dabei verschiedene Funktionalitäten zu erfüllen. Zum einen gibt sie Rückmeldung über verschiedene Stati des Karts. So kann beispielsweise eine State of Charge Abfrage durchgeführt werden. Die VCU gibt dabei den SOC in Form von Lichtimpulsen zurück. Befindet sich der SOC bei Abfrage zwischen 0% und 25% blinkt die LED einmal bis hin zu viermal, wenn sich der Ladestand zwischen 75% und 100% befindet.  
Des Weiteren ist die Status-LED auch teil eines Sicherheitsfeatures. Die LED beginnt zu blinken, sobald und solange das Kart im Fahrbereiten Zustand ist, also der Startknopf gedrückt wurde, aber das Kart noch nicht in Bewegung ist. Das dient zur Visualisierung für Streckenhelfer und Zuschauer, um zu verhindern, dass Leute verletzt werden, da das Kart unerwartet losfährt. Zu guter Letzt zeigt die LED auch den eingestellten Fahrmodus an, sobald dieser geändert wurde. Das dient dazu, um Sabotage und Betrug im Wettbewerb zu verhindern. Die Änderung soll für jeden so sichtbar wie möglich werden, um zu verhindern, dass einzelne Teilnehmer den Fahrmodus verändern und sich einen Vorteil verschaffen.   
Eine weitere Funktionalität der Option1 ist eine Funkfernbedienung, um einzelne Funktionalitäten von außen Steuern zu können. Dazu gehören die Aktivierung des Notaus und das Starten des Karts, also die Aktivierung des Ready-To-Drive Status. Mittels Tastendrucks auf der Fernbedienung kann der Notaus aktiviert werden, was vor allem bei jüngeren Fahrern zur Sicherheit beiträgt. Das Funksignal unterbricht dabei denselben Signalpfad wie der Knopf am Kart. Ein anderer Knopf kann den Ready-To-Drive Modus aktivieren, hier wird lediglich ein Signal an die VCU gesendet, welches dieselbe Funktionalität wie der Knopf am Kart erzeugt. Der Vorteil liegt darin, dass vor allem jüngere Fahrer unterstützt werden können, sollten sie im Wettbewerb nervös werden oder selbst nicht wissen, wie das Kart gestartet wird Der dritte Knopf an der Fernbedienung sendet ein Signal an die VCU, welches die Abfrage und Anzeige des SOC auslöst, welcher anschließend über die LED dargestellt wird. Die letzte taste wird zur Identifikation des Karts verwendet, zu welchem die Fernbedienung gehört. Hierbei leuchtet die Status-LED auf, solange der Knopf gedrückt bleibt. Alle diese Funktionalitäten werden über ein Zugekauftes Funkmodul realisiert, welches aus einem Funkempfänger und 4 Relais auf einer Platine besteht. Die Funktionalitäten der Relais sind dabei vordefiniert. Kanal 1 verhält sich wie ein Flip-Flop und ändert seinen Zustand bei jedem Tastendruck, speichert also seinen Zustand, weshalb es für die Funktionalität des Notaus genutzt wird. Das Relais unterbricht dabei einen kabelbruchsicheren Signalpfad, welche daraufhin als Notaus detektiert wird. Die Relais für Kanal 2 und 3 speichern ihren Zustand nicht und schalten ihren Zustand lediglich für kurze Zeit, sollte eine Funksignal empfangen worden sein. Sie realisieren deshalb Funktionalitäten, für die nur ein kurzes Signal an das Steuergerät gesendet werden soll. Das Relais zieht dabei lediglich eine Signalleitung auf 12V-Pegel, welches direkt mit der VCU verkabelt ist. Dadurch wird dort ein Signal erkannt. Für Kanal 4 ist ein bistabiles Relais auf dem Modul vorhanden, welches zwei Unterschiedliche Zustände annehmen kann, von denen einer immer aktiv ist. Es Verbindet die Status-LED standartmäßig mit dem LED-Signal, welches von der VCU an das Erweiterungssystem geschickt wird. Für die Dauer des Knopfdrucks schaltet das Relais um und aktiviert ein konstantes Leuchten der Status-LED.

## Anforderungen

Die Aufgabe meines Projekts besteht darin alle Erweiterungsoptionen, welche für das Kart verfügbar sind in einem kompakten und günstigen System zu vereinen. Bisher waren alle diese Komponenten voneinander unabhängige Systeme, welche separat zugekauft werden müssen und durch viel Verkabelungsaufwand verbunden und angebunden werden. Dadurch war Vorbereitung kaum möglich, der Verkabelungsaufwand enorm und unübersichtlich, Fehlersuche erschwert.   
Ziel ist Verkabelungsaufwand zu verringern und damit das System Stabiler, günstiger und weniger Fehleranfällig zu gestalten. Es soll möglich sein, sowohl Kabelbaum und Erweiterungsplatine unabhängig voneinander vorbereiten und testen zu können. Dabei sollen alle bisherigen Funktionen erhalten bleiben. Zusätzlich soll das neue System Schnittstellen schaffen, welche sowohl dem Kunden und Fahrer neue Informationen geben sollen als auch für den Entwicklungs- und Wartungstechniker neue Schnittstellen schaffen, um Probleme schnell und einfacher beheben zu können, sowie Anpassungen und Updates einfacher aufspielen zu können. Neben dem Beibehalten der alten Funktionen soll auch die Kompatibilität zu bereits gebauten und verkauften Karts beibehalten werden, um auch diese nachrüsten zu können, ohne Änderungen am Kabelbaum des Karts vornehmen zu müssen.

Aufgabe der Erweiterung bleibt weiterhin die Funktionalität eines RFID-Readers. Die Ausgelesenen Daten werden dabei sowohl auf der Erweiterungsplatine als auch per CAN in der VCU benötigt. Auf der RFID -Karte müssen dafür verschiedene Daten gespeichert sein. Die RFID-Karten werden weiterhin zur Einstellung des vordefinierten Leistungsmodus verwendet. Hierfür wird auf den Karten die ID des Leistungsmodus gespeichert und per CAN an die VCU übermittelt. Um zu verhindern, dass jeder Kunde die Einstellungen aller Karts verändern und so unter Umständen Wettbewerbe verfälschen kann, wird zusätzlich zum Modus auch eine Kundenspezifische Nummer auf der Karte gespeichert und übertragen. Mit dieser Nummer, welche per CAN an das Steuergerät übermittelt werden soll, kann validiert werden, dass die Chips und das Kart zum selben Kunden gehören, sodass jeder Kunde die Chips nur an seine eigenen Karts einsetzen kann. Zusätzlich soll die Kundennummer auch auf der Erweiterungsplatine ausgelesen werden. Eine wichtige Anforderung an den RFID-Reader ist eine hohe Verschlüsselungsstufe. Da die Karts auch im Wettbewerb eingesetzt werden, darf es nicht möglich sein RFID-Chips zu kopieren, zu fälschen oder zu manipulieren.

Mit den bestehenden Funktionen gehört eine Status-Leuchte weiterhin zu den Anforderungen. Diese Statusleuchte muss sowohl von der VCU als auch dem Funkempfänger, welcher auf meiner neuen Platine platziert sein wird, steuerbar sein. Die LED muss in verschiedenen Situationen steuerbar sein:

* Ready-To-Drive-Status:   
  Die LED blinkt kontinuierlich, sobald das Kart in Fahrbereitem Zustand ist, solange das Kart steht. Das Steuersignal dafür sendet die VCU.
* SOC-Abfrage:   
  Die LED blinkt 1- bis 4-mal, je nach aktuellem SOC-Stand. Die Berechnung des SOCs und Übermittlung des LED-Signals übernimmt die VCU
* Identifizierung:  
  Die LED leuchtet kontinuierlich, solange per Funkfernbedienung die Identifizierung des Karts abgefragt wird. Das Signal wird vom Funkempfänger ausgegeben

Um den Aktuellen Verkabelungsaufwand zu reduzieren, soll das Signal nicht mehr als Digitalsignal über eine Leitung direkt vom Steuergerät aus übermittelt werden, sondern als CAN-Message an die Erweiterungsplatine gesendet werden. Um Abwärtskompatibilität beizubehalten, soll zusätzlich die Möglichkeit einer Physischen Übertragung und Auswertung auf der Platine vorgesehen werden.

Eine weitere Funktion, welche beibehalten werden soll, ist die Möglichkeit einige Funktionen des Karts mittels einer Funkfernbedienung zu steuern.   
Dabei soll es möglich sein folgende vier Funktionen mittels der Fernbedienung steuern zu können:

* Notaus - Funktionalität:  
  Mittels einer Taste muss der Notaus ausgelöst werden, welcher sofort die Stromversorgung für das Kart trennt und es zum Stillstand bringt. Die Funktionalität des Notaus ist dabei bereits vorhanden, die Aufgabe der Option 1 besteht lediglich darin ein Signal 12V Signal auszugeben, welches den Notaus aktivieren kann. Zur Sicherheit muss dieses Signal Kabelbruchsicher realisiert werden und soll nicht Softwaregesteuert durch den Mikrocontroller sein.
* Remote Drive - Funktionalität:

Remote Drive bezeichnet das Signal, welches das Kart in den Fahrbereiten Zustand (Ready to Drive) versetzt, in welchem der Fahrer jederzeit in der Lage ist, loszufahren. Die Aktivierung und damit verbundenen Sicherheitschecks übernimmt vollständig die VCU, Aufgabe des Zusatzsystems ist es lediglich ein Signal an das Steuergerät zu senden. Um Verkabelungsaufwand zu minimieren, soll dieses Signal nicht mehr über eine eigene Leitung im kabelbaum realisiert werden, sondern als Signal auf dem CAN-Bus an die VCU gesendet werden. Um die Möglichkeit zu haben alte Karts nachzurüsten, soll aber zusätzlich die Möglichkeit vorgesehen werden, das Signal auch weiterhin über eine eigene physische Leitung an das Steuergerät zu übermitteln, da die alten Kabelbäume keine CAN-Leitungen zur Option 1 führen.

* SOC-Abfrage:  
  Die dritte Funktionalität, welche per Funkfernbedienung zur Verfügung gestellt werden soll, ist die Abfrage des State of Charge. Mit Drücken des Knopfes an der Fernbedienung soll die Status-LED zwischen ein und vier-mal, abhängig vom aktuellen State of Charge, blinken. Die Abfrage des SOC und Berechnung der Anzahl der LED-Signale übernimmt dabei die VCU. Diese Funktionalität soll weiterhin die VCU übernehmen, obwohl der SOC für andere Funktionalitäten bereits an die VCU übermittelt wird, um die Kompatibilität zu alten Kabelbäumen zu wahren, welche noch Keine CAN-Verbindung zur Option 1 vorgesehen haben. Aus demselben Grund soll das Signal auch weiterhin zur Übertragung über eine eigene Leitung zum Steuergerät vorgesehen werden. Primär soll das Signal aber per CAN übermittelt werden, um die Kabelanzahl zu reduzieren und damit Kosten und Aufwand zu verringern.
* Identifikation:  
  Eine weitere Funktionalität der Funkfernbedienung soll es sein, dass Kart zu identifizieren. Hier soll die Status-LED leuchten, solange die entsprechende Taste auf der Fernbedienung gedrückt gehalten wird.

Ziel soll dabei auch sein die Kosten für die Erweiterung zu minimieren und sich von Zukaufteilen zu lösen. Durch das Zugekaufte Modul waren die Funktionen kaum anpassbar du wenig flexibel. Auch die Notwendigkeit einer eigenen Signalleitung für jedes Signal trug zu viel verkabelungsaufwand und höheren Kosten bei. Das neue System soll dahingehend einige Verbesserungen schaffen. So soll es auch möglich sein, dass die Zuweisung einer Fernbedienung auch nach dem Einbau der Option 1 möglich ist. Bisher musste eine Fernbedienung per Tastendruck auf dem Empfängermodul verbunden werden, bevor das Modul verbaut werden konnte und musste somit für den gesamten Fertigungs- und Verkabelungsprozess bei dem System bleiben. Außerdem war es nur mit großem Aufwand möglich defekte Fernbedienungen zu ersetzen oder Ersatz für verlorene Fernbedienungen zu verbinden. Für das neue System soll dieser Verbindungsvorgang softwareseitig ausgelöst werden können, um eine Fernbedienung auch nachträglich ohne großen Aufwand verbinden zu können. Trotz allem soll für Testzwecke oder Sonderwünsche weiterhin eine Aktivierung des Verbindungsmodus per Knopfdruck auf der Platine vorgesehen werden.

Neben diesen bereits in alten variante bereitgestellten Funktionen soll die Erweiterungsplatine auch einige neue Funktionalitäten und Schnittstellen zur Verfügung stellen. Eine dieser Funktionen ist eine kabellose Datenverbindung. Dabei soll es möglich sein, sich sowohl mit Mobilen Endgeräten wie Handy oder Tablet, aber auch mit Laptops mit dem Kart zu verbinden, um verschiedene Funktionalitäten dieser Schnittstelle nutzen zu können:

* Livedaten:  
  Die Hauptfunktionalität, welche die kabellose Schnittstelle bieten soll, ist die Zur Verfügungstellung von Livedaten. Sowohl der Kunde als auch ein Wartungstechniker sollen Livedaten vom Kart empfangen können, welche in einer GUI leicht verständlich und übersichtlich aufbereitet werden. Hierfür ist eine zuverlässige und schnelle Verbindung nötig. Zudem müssen die Daten per CAN empfangen und ausgewertet werden.
* Kunden - Einstellungen:  
  Neben dem Einsehen von Livedaten soll es zudem möglich sein einfache Einstellungen, welche keine neuen Funktionalitäten erlauben oder verbieten, für den Kunden selbst vorzunehmen. Hierbei ist es wichtig, dass diese Einstellungen persistent gespeichert werden, damit die Daten auch nach einem Neustart des Karts erhalten bleiben. Zudem muss optisch dargestellt werden können, welche Einstellungen gerade aktiv oder deaktiviert sind.
* Hersteller – Einstellungen:  
  Neben den Einstellungen, die der Kunde selbst an seinem Kart vornehmen kann, soll es zusätzlich einige Einstellungen geben, welche nur der Hersteller vornehmen kann. Hintergrund ist, dass in jedes Kart dieselbe Platine verbaut und identische Software aufgespielt wird. Je nach Bedarf und Wunsch des Kunden werden nun die Funktionalitäten freigeschalten oder gesperrt. Dadurch sollen hohe kosten durch unterschiedliche Systeme und großer Aufwand beim Auseinanderhalten der Karts mit unterschiedlichen Systemen vermieden werden. Zusätzlich ist es dadurch sehr einfach nachträglich weitere Features freizuschalten, ohne Hardware oder Software am Kart verändern zu müssen. Um zu verhindern das Kunden auf Funktionalitäten Zugriff erhalten, welche nicht gekauft wurden müssen die Einstellungsmöglichkeiten Zugriffsgeschützt werden.

Der Verbindungsaufbau soll dabei so einfach wie möglich sein, muss gleichzeitig aber trotzdem sicher genug sein, dass nur der Besitzer des Karts Verbindung aufbauen kann. Für den Einsatz im Wettbewerb muss es außerdem möglich sein Zugriff auf das Netzwerk zu verhindern, da oft ein Kartclub die Karts für die gesamte Veranstaltung zur Verfügung stellt. Hier darf es nicht möglich sein die Einstellungen während der Veranstaltung zu manipulieren oder Livedaten anderer Fahrer auszulesen.

Eine weitere Schnittstelle für Kunde und Techniker wird über ein Display zur Verfügung gestellt. Dieses Display soll mehrere Seiten besitzen, zwischen welchen einfach gewechselt werden kann. Auch hier steht die Darstellung von Livedaten im Vordergrund, welche auch für den Fahrer live während der Fahrt sichtbar sind. Hier sollen vor allem Daten wie Temperatur und Ladestand angezeigt werden. Dabei ist es wichtig die Daten grafisch und Farblich aufzubereiten, um es dem Fahrer einfacher zu machen, die Werte wahrzunehmen. Des Weiteren sollen auch einzelne Funktionen über das Display veränderbar sein. Aufgrund der Fehlenden Eingabemöglichkeiten soll es sich dabei nur um boolesche Variablen handeln, die verändert werden können. Die Einstellungen sollen einfach und übersichtlich sein. Da es sehr ablenkend für vor allem junge Fahrer sein kann, soll es die Funktion geben, dass sich das Display während der Fahrt ausschaltet. Die Daten sollen wieder sichtbar werden, sobald das Kart zum Stillstand kommt. Neben den Livedaten und den Einstellungen sollen hier auch Fehlercodes angezeigt werden, die während dem Initialisierungsprozess aufgetreten sind und indizieren, dass einzelne Funktionalitäten nicht zur Verfügung stehen.

Zusätzlich soll eine Logdateien Funktionalität vorgesehen werden. Das System soll sowohl Fehler speichern, welche im Betrieb der Erweiterungsplatine auftreten, als auch Fehler im Betrieb der VCU speichern, welche per CAN an die VCU übermittelt werden. Die Speicherung erfolgt dabei mit Timestamp und Fehlercode. Der Fehlerspeicher soll dabei einfach auslesbar sein, ohne sich physisch mit dem System verbinden zu müssen.

Grundsätzlich soll es dabei möglich sein alle Funktionen unabhängig voneinander nutzen zu können, sodass sich jeder Kunde ein eigenes Paket aus Leistungen zusammenstellen kann. Damit muss es auch möglich sein alle Funktionalitäten einzeln zu deaktivieren oder freizuschalten.

## Integration ins bestehende System

Einer der wichtigsten Punkte bei der Entwicklung des neuen Systems ist Kompatibilität mit dem Bereits vorhandenen Gesamtsystem. Dazu gehört, dass sich an anderen Systemen so wenig wie möglich ändern soll. Das beinhaltet sowohl mechanische Anpassungen als auch minimale Anpassungen in der Software oder dem Kabelbaum. Ziel bleibt dabei trotzdem weiterhin möglichst viele Verbesserungen in Bezug auf Kosten und Arbeitsaufwand zu erzielen.   
Vor allem mechanisch soll sich nichts am bisherigen Kart ändern, zum einen, da die Option 1 feste Anbindungspunkte am Kart nutzt, welche sich nur mit sehr großem konstruktionsaufwand für das gesamte Kart verändern lassen, und diesen vorhandenen Platz an den Anbindungspunkte bereits vollständig ausnutzt, zum anderen soll das neue System aber auch in bereits gebauten Karts ohne großen Aufwand nachgerüstet werden können. Diese mechanischen Vorgaben definieren damit die Größe und Form der Platine zu einem sehr großen Teil. Auch die Position der Stecker ist durch die Kabeldurchführungen im Gehäuse sehr beschränkt. Für die Erweiterung mit einem Display soll es ein eigenes Gehäuse geben, sodass für das Display das bestehende Gehäuse nicht bearbeitet werden muss, sondern lediglich die Kabel herausgeführt werden müssen, wofür die bereits vorhandenen Kabeldurchführungen genutzt werden können.   
Auch für den Kabelbaum sollen möglichst wenig Änderungen nötig werden. Da das System bisher aus mehreren losen verkabelten Komponenten bestand gab es nur einen Stecker mit deutlich weniger Pins als bisher. Dieser 6Pin Stecker wird für die Anbindung des RFID-readers und der Statusleuchte übernommen, da bei diesem Stecker die losen, abisolierten Enden der Kabel einzeln in den Stecker auf der Platine gequetscht werden ohne dass es einen Gegenstecker gibt, der vorbereitet werden muss. Das sorgt dafür, dass der RFID-reader und die Statusleuchte weiterhin wie vorher vorbereitet werden können und auch bereits vorbereitete Komponenten weiterverwendet werden können. Für alle Leitungen, die von der VCU zur Option1 geführt werden, wird ein neuer 14-Pin Stecker vorgesehen. Die Steckverbindung besteht dabei aus 2 Bauteilen mit Crimp Kontakten als Pins. Durch die Trennung von externen Bauteilen und Kabelbaumseite kann weiterhin der kabelbaum separat vorbereitet werden, ohne auf optimale Komponenten angewiesen zu sein. Die Verwendung eines neuen Steckers wird durch die höhere Anzahl an Kabeln notwendig. Für die Daten, welche auf Display und Webseite angezeigt werden sollen, muss zusätzlich Can1 zur Erweiterungsplatine geführt werden. CAN1 wird zusätzlich auch für die VCU-Befehle benötigt, welche über die Funkverbindung erhalten werden. Für alte Karts, die keine CAN1 Verbindung vorgesehen haben, sind zusätzlich noch die PINs zur Übertragung per Kabel vorgesehen, sodass zumindest die Funktionalität der Funkfernbedienung erhalten bleibt und die alten Karts ohne Anpassungen die Funktionalitäten des bisherigen Erweiterungspakets ohne Einschränkungen nutzen können, sollte aus Reparaturgründen die Option1 ausgetauscht werden müssen. Für Livedaten und Telemetrie müsste der Kabelbaum jedoch nachgerüstet werden.  
Auch beim Steuergerät steht es im Vordergrund, dass alte Systeme weiterhin zuverlässig laufen, sollte das alte System durch eine neue Platine ausgetauscht werden müssen. So soll sowohl die Auswertung von Digitalsignalen auf der Option1 möglich sein, welche von der VCU gesendet werden, aber auch die Option1 sieht Signale weiterhin vor, sodass sie sowohl als CAN-Signal oder über eine eigene Leitung an die VCU gesendet werden können. Auch die Struktur von Signalen soll gleichbleiben, um in der Auswertung keine Anpassungen vornehmen zu müssen. So werden die Daten, welche per RFID-Reader ausgelesen werden, weiterhin im Selben Aufbau und mit denselben IDs auf CAN2 an die VCU gesendet. Einige Änderungen müssen jedoch vorgenommen werden, diese beschränken sich jedoch auf das Senden und Empfangen von CAN-Nachrichten und deren Verarbeitung, sodass lediglich Erweiterungen, aber keine grundsätzlichen Änderungen in der Funktionsweise der Steuergerätsoftware nötig sind.

# Konzepte

Um diese Anforderungen an das neue System möglichst gut zu erfüllen kann es mehrere Möglichkeiten geben, welche jeweils besondere Aspekte eines Problems oder einer Anforderung besonders erfüllen können, oder aber auch auf Ausgewogenheit abzielen, um jede Anforderung möglichst gleich gut abzudecken. Um eine fundierte Entscheidung für eines dieser verschiedenen Konzepte zu treffen, ist es daher nötig, Anforderungen möglichst spezifisch zu definieren und zu gewichten, aber auch Konzepte möglichst detailreich zu beschreiben und Stärken und Schwächen einzelner Möglichkeiten gezielt herauszuarbeiten. Diese Vorarbeit bietet die Möglichkeit auf einer fundierten Entscheidungsgrundlage die Konzepte zu vergleichen und die bestmögliche Lösung zu identifizieren und eine begründete Entscheidung für eine der Umsetzungsmöglichkeiten für das System zu treffen. Für die Durchführung dieser Gewichtungen, Vergleiche und Abwägungen gibt es verschieden Möglichkeiten, welche in den Ingenieurwissenschaften zur Anwendung kommen. Für diese Arbeit wurde sich für Paarvergleiche zur Gewichtung der Anforderungen, sowie für Nutzwertanalysen zum Vergleich der Konzepte auf Grundlage der Gewichtungen der Anforderungen entschieden. Dieses System wurde gewählt, da es sich hierbei um ein sehr einfaches und schnell umsetzbares System handelt, welches sich für kleinere Systeme sehr gut eignet, ohne unübersichtlich zu werden. Zudem bleibt die Entscheidungsfindung hierbei sehr transparent und nachvollziehbar. Im Folgenden sollen die Paarvergleiche und Nutzwertanalysen kurz beschrieben und erklärt werden. Die Möglichen Konzepte sollen zudem erläutert werden, sowie die Entscheidungen, die getroffen wurden, erklärt werden.

## Paarvergleiche

Beim Paarvergleich geht es darum Anforderungen gegeneinander abzuwägen und zu priorisieren. Der Paarvergleich dient dabei dazu dieses Vorgehen zu vereinfachen und transparenter zu machen, indem man jeden Punkt mit jedem anderen einzeln vergleicht. Am Ende ergibt sich eine sehr genaue Staffelung. Dieses Verfahren soll kurz erklärt werden, bevor die Paarvergleiche für jeden der vorher definierten Anforderungspunkte durchgeführt und erklärt werden.

Bei den zu vergleichenden Anforderungen handelt es sich nicht um die vorher definierten Anforderungen an das System, sondern vielmehr sollen für diese Anforderungen verschiedene Konzepte diskutiert werden. Daher werden verschiedene Kriterien erstellt, die für diese Konzepte relevant sein können, aufgrund derer am Ende die Wahl für eines der Konzepte in der Nutzwertanalyse fällt. Dieses Vorgehen soll beispielhaft an dem Paarvergleich gezeigt werden.  
Für den Paarvergleich wurde eine Excelliste erstellt, welche die Berechnungen automatisch durchführt und dem Anwender einfach aufzeigt, welche Informationen von Ihm benötigt werden.  
Ein Bild, das Reihe, Diagramm, Screenshot, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.

Im Bild kann man die grobe Struktur erkennen. In die gelben Felder sind Eingaben vom Benutzer vorgesehen, in den Restlichen Weißen Feldern werden über Formeln die Eingaben erzeugt. Die Rot Umrandeten Felder sind hier zu einfacheren Erklärung hervorgehoben. In diesen Feldern werden die Kriterien eingetragen, welche miteinander verglichen werden müssen. Um die Ergebnisse durch falsche Eingaben nicht zu verfälschen, muss der Nutzer diese Kriterien nur in die Zeilen Eintragen, in die Spaltenüberschriften werden die Kriterien Namen dann gemäß der Reihenfolge ihrer Eintragungen automatisch kopiert. Damit ergibt sich die Tabelle auf deren Grundlage anschließend die Vergleichsergebnisse eingetragen werden können. In der letzten Spalte sollten zudem Kurze Erläuterungen zu den Kriterien erstellt werden, um es dem Leser oder jemandem, der ebenfalls an dem System arbeiten soll möglichst verständlich zu machen, worauf dieses Kriterium abzielt. Eine Beispielhafte Tabelle könnte damit in etwa so aussehen.

Ein Bild, das Text, Screenshot, Reihe, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Auf Grundlage dieser Tabelle können nun in die gelben Felder zwischen den Kriterien die Vergleichsergebnisse eingetragen werden. Wie bei einer Matrix werden die Vergleichsergebnisse in das Feld eingetragen, in welchem Zeile und Spalte mit dem jeweiligen Namen zusammenstoßen. Die Gewichtung wird durch die Zahlen 0 bis 2 dargestellt, wobei 2 definiert, dass das Kriterium, welches in der Zeile steht, höher zu gewichten ist als das in der der Spalte. Eine 0 hingegen zeigt das Gegenteil an, in diesem Fall wäre die Spalte höher gewichtet als die Zeile. Sollte es Kriterien geben, welche gleich gewichtet werden sollen, kann eine 1 eingetragen werden. Da es jede Kombination aus Kriterien zweimal gibt, da jedes Kriterium als Zeile und Spalte vorkommt, muss vom Nutzer der Tabelle nur die Gewichtung in der oberen Hälfte eingetragen werden. Der Wert der korrespondierenden Zelle wird anhand der Formel

Berechnet, wobei die vom Nutzer eingetragene Gewichtung ist und die Gewichtung der korrespondierenden Zelle. Daraus ergibt sich, dass eine vom Nutzer eingetragene 2 zwangsläufig zu einer 0 in der korrespondierenden Zelle führt, wodurch genau der gewünschte Effekt erzielt wird.

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

Aus den Gewichtungswerten wird nun zeilenweise die Summe gebildet. Dadurch, dass der Wert im Falle einer Höhergewichtung der Zeile gegenüber der Spalte am höchsten ist, ergibt sich nun, dass die Spalte mit der höchsten Summe gegenüber allen anderen Spalten am wichtigsten wiegt. Im oberen Beispiel wäre das die Störungssicherheit. Um nun ein mit anderen Paarvergleichen vergleichbares Ergebnis zu erhalten, werden die Werte auf insgesamt 100% normiert. Das erreicht man, indem man die Summen der einzelnen Zeilen durch die Summe aller Werte teilt. Mit diesen Werten kann nun in einer Nutzwertanalyse zur Bewertung der einzelnen Konzepte genutzt werden. Dieses Verfahren soll nun anhand der realen Konzeptentscheidungen angewendet werden.

## Nutzwertanalysen

blablabla

Ein Bild, das Text, Zahl, Screenshot, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.

# Hardwareentwicklung

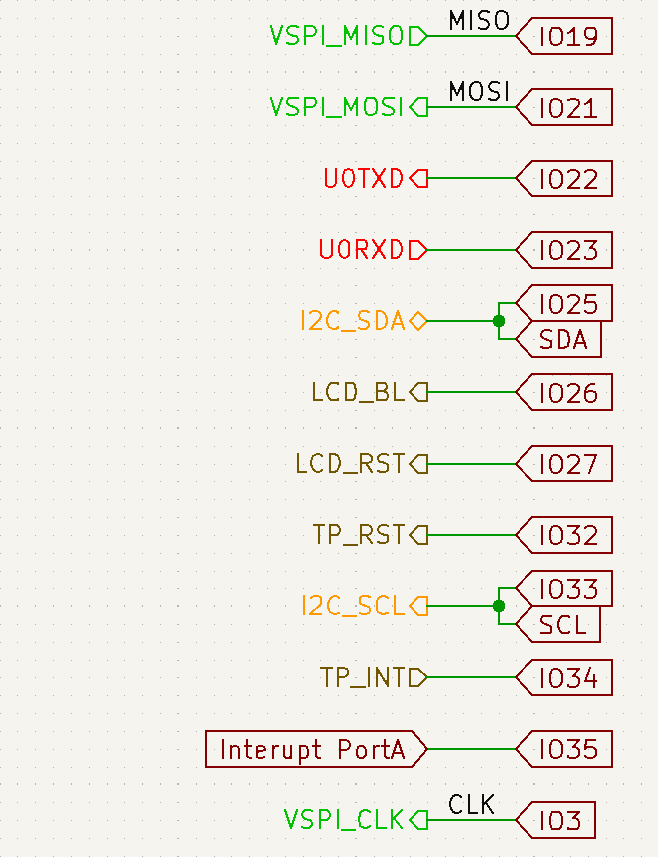
## Schaltungsaufbau

Der erste Entwicklungsschritt für das neue Erweiterungssystem ist der Aufbau geeigneter Schaltungen, welche alle Funktionalitäten abbilden. In diesem Kapitel soll es daher um die Schaltungen, deren Aufbau und Funktionsweiße gehen. Der Schaltungsaufbau ist dabei so organisiert, dass jede Einzelschaltung in einem eigenen Schaltplan realisiert ist und ein Masterblatt diese Systeme alle miteinander verbindet. Dabei wird jede Schaltung einzeln betrachtet und erläutert. Ein Bild, das Text, Diagramm, Plan, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.   
Mikrocontroller  
Zuallererst soll hierbei der Microcontroller als Zentrales Element der Schaltung betrachtet werden. Dabei kommt ein ESP32-Microcontroller zum Einsatz. Dieser übernimmt sämtliche Rechenoperationen und Kommunikationsaufgaben für das System. Der Mikrocontroller wird dabei mit 3.3V versorgt. Dadurch, dass der 3.3V auch Signalpegel für viele Signale ist, ist es sehr wichtig, ein stabiles und rauschfreies Signal am Microcontroller zu erhalten. Dafür sind die Kondensatoren C801 und C802 zwischen VCC und Ground vorgesehen. Bei ihnen handelt es sich um Abblockkondensatoren, welche Hochfrequente Störungen gegen Ground ableiten und kurzfristige, minimale Spannungseinbrüche abpuffern sollen. Außerdem ist es wichtig sicherzustellen, dass die Stromversorgung beim Einschalten des ESP32 stabil ist, um einen sicheren Boot-Vorgang zu gewährleisten ohne Fehler zu erhalten. Das Einschalten passiert dabei über den EN-Pin des Mikrocontrollers. Hierfür wird ein RC-Glied vor den Enable-Pin geschalten. Das Datenblatt empfiehlt dabei mindestens als Wartezeit, bis die Spannungsversorgung am ESP stabil anliegt, bevor der Chip über EN-Pin aktiviert wird. Die Verzögerung des RC-Glied lässt sich dabei folgendermaßen berechnen

Das Datenblatt empfiehlt für , das entspricht einer Verzögerung von .   
Ein Bild, das Text, Diagramm, Plan, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.  
Zusätzlich beim Bootvorgang zu beachten sind die Strapping Pins des Mikrocontrollers. Bei Strapping-Pins handelt es sich um die Pins, welche den Ablauf des Bootvorgangs definieren. So lässt sich beispielsweise der Boot Mode definieren, ob die Software, welche bereits auf dem Gerät gespeichert ist für den Bootvorgang verwendet wird oder ob extern neue Software aufgespielt werden soll. Die Strapping Pins des ESP32 sind dabei GPIO 0, GPIO 2, GPIO 5, GPIO 12 (MTD1), und GPIO 15 (MTD 0). Für diese Pins ist im Datenblatt eine Standartbelegung definiert, bei welcher der normale Bootvorgang abläuft, wobei der Chip aus seinem eigenen Speicher bootet. Die Pegel werden dabei über Pull-Up und Pull-Down Widerstände erzielt, das sind die Widerstände R801 bis R805. Um diesen Bootvorgang nicht zu beeinflussen, wurde daher bewusst darauf verzichtet, die Strapping Pins für weitere Funktionalitäten zu nutzen. Lediglich die Pins GPIO 0 und 2, welche den Bootmodus definieren, werden noch zusätzlich vom USB-Controller gesteuert, um ein Flashen per USB zu ermöglichen.  
Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.  
Ein großer Vorteil des ESP32 ist, dass beinahe alle Pins für alle Funktionen verwendet werden können. Lediglich die Pins GPIO 34, 35, 36 und 39 sind nur als Input verwendbar und die Pins GPIO 6 bis 11 werden intern verwendet und dürfen nicht verwendet werden. Somit bietet der Chip sehr viele Möglichkeiten für Signale oder Kommunikation. Ein verwendetes Kommunikationsprotokoll ist I²C. Es wird sowohl zur Ansteuerung des Displays als auch für die Datenübertragung an eine GPIO Port Erweiterung verwendet. I²C ist ein Protokoll, welches mit zwei Signalleitungen auskommt. Für Datenübertragung (SDA) ist hierbei Port GPIO 25 vorgesehen, für das Clock-Signal wird Pin GPIO 33 verwendet. Die Auswahl der Pins beruht dabei auf praktischen Gesichtspunkten, so sollen die Ports physisch möglichst nebeneinander liegen, um das spätere Layout zu vereinfachen. Für I²C müssen zudem Pull-Up Widerstände vorgesehen werden, da es sich bei I²C um Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.OpenDrain-System handelt, weswegen die Signale auf einen Stabilen Pegel gezogen werden müssen. Bei Übertragung eines Signals wird die Leitung dann auf Ground Potential gezogen. Bei I²C haben sich 4.7 kΩ als Standartwert für die I²C Kommunikation etabliert.   
Ein weiterer sehr verbreiteter Standard zur Kommunikation mit Peripheriegeräten ist SPI. Bei dem aktuellen System werden damit das Display, ein Flashspeicher, sowie der Controller zur Kommunikation auf CAN-Bus 2 gesteuert. Anders als I²C benötigt SPI mindestens 4 Leitungen. Dazu gehört auch wieder ein Clock-Signal, welches jeden Teilnehmer synchronisiert. Dazu kommen anders als bei I²C aber zwei Datenleitungen. Hier wird unterschieden zwischen gesendeten und empfangenen Nachrichten. Jeder Teilnehmer besitzt ein Master In, Slave Out-Signal (MISO) und ein Master Out, Slave In-Signal (MOSI) über welches die Teilnehmer miteinander kommunizieren können. Hinzu kommt ein ChipSelect Signal für jeden Teilnehmer, der auf dem Bus Kommunizieren möchte. Über dieses Signal muss immer genau ein Teilnehmer vom Master freigegeben werden, welcher dann kommunizieren kann. Auch hier werden die Pins so ausgewählt, dass das Layout möglichst vereinfacht werden kann. Eine weitere verbreitete Kommunikationsart, vor allem zur Kommunikation unter Steuergeräten, ist der CAN-Bus. Der Chip besitzt intern einen Controller für Two-Wire Automotive Interface (TWAI), welcher in der Lage ist eine CAN-Kommunikation aufzubauen. Dabei werden 2 physische Leitungen für CAN-High und CAN-Low benötigt. Diese werden nicht am Mikrocontroller angeschlossen, weil zur Kommunikation noch ein CAN-Transceiver benötigt wird. Dieser erhält über UART seine Daten und Befehle vom TWAI-Controller, dabei kommunizieren die Teilnehmer über zwei Leitungen für erhaltene und gesendet Nachrichten. Daher werden auch zwei Pins am Chip dafür verwendet. Da es sich hierbei um ein asynchrones Protokoll handelt, ist hier kein Clocksignal wie bei SPI oder I²C nötig. Im vorliegenden System kommuniziert darüber sowohl der RFID-Reader als auch die Datenübertragung mit dem Steuergerät findet über CAN-Kommunikation statt. Da der Mikrocontroller selbst leider keine zwei CAN-Busse unterstützt muss für den zweiten Bus ein externer CAN-Controller vorgesehen werden, dieser überträgt seine Daten via SPI. Der Controller für Bus 2 bietet allerdings eine Interrupt-Funktionalität. Bei Empfang einer Nachricht wird über diesen Pin ein Signal ausgegeben, welches am ESP32 als Interrupt detektiert werden kann.   
Die Signale U0TXD und U0RXD gehören ebenfalls zu einer UART-Schnittstelle. Diese wird im vorliegenden Beispiel lediglich zur Kommunikation mit einem extern angeschlossenen USB-Gerät verwendet und soll als Schnittstelle zum flashen eines neuen Programms dienen. Auch hier wird mit zwei Leitungen kommuniziert, wobei jeweils zwischen Receive und Transmit unterschieden wird, also eine Signalleitung, zum Empfangen von Nachrichten und eine zum Senden. Auch hier werden die nebeneinanderliegenden Pins GPIO 22 und 23 genutzt. Die Restlichen Pins werden für die Steuerung des Displays genutzt. Zum einen gibt es einen Pin für die Hintergrundbeleuchtung des Displays. Dafür wird der Pin als Ausgang eines PWM-Generators definiert. Über die Frequenz des Pulsweiten Modulierten Signals wird somit die Helligkeit der Hintergrundbeleuchtung bestimmt. Ein weiterer Pin ist der DC-Pin des Displays. Dieser ist ein standartmäßiger Ausgang für ein Digitalsignal. Dieses Signal wird dafür genutzt, um dem Display mitzuteilen, ob ein Befehl übertragen wird oder ob es gerade Daten zum Darstellen erhält. Die Pins GPIO 26 und 27 werden als Digitaler Ausgang genutzt. Hier werden Signale ausgegeben, welche genutzt werden können, um das Display und den Touch Controller zu resetten. GPIO 34 wird ähnlich wie bei CAN genutzt, um einen Interrupt durch den Touch Controller des Displays auszulösen. Der Pin wird auf High gezogen, wenn der Touch Controller eine Berührung erkennt, und der Pin kann im Chip als Interrupt definiert werden, sodass jede Berührung direkt verarbeitet werden kann. Hierfür wird einer der Pins genutzt, welche nur als Eingang fungieren, da es sich nur um einen Einseitigen Interrupt handelt. Lediglich der Touch Controller darf diesen Pin setzen, niemals der Mikrocontroller, dieser wertet das Signal lediglich aus. Dasselbe gilt auch für den Pin GPIO 35. Über diesen ist der Interrupt-Pin des GPIO-Expanders an den Chip angeschlossen. Dieser erzeugt ein Signal an dem Pin, welcher dann als Interrupt im ESP32 definiert werden kann, sobald sich der Wert an einem der als Input definierten Pins ändert. Somit können auch dort ausgelesene Daten zeitnah vom ESP ausgelesen und verarbeitet werden.   
Die bereits angesprochene GPIO Port Erweiterung hat dabei die Funktion alle übrigen Funktionalitäten und Pins zur Verfügung zu stellen, welche am Mikrocontroller keinen Platz mehr gefunden haben. Die Port Erweiterung stellt allerdings lediglich GPIO-Funktionalität zur Verfügung und kann keine Sonderfunktionalitäten wie SPI oder I²C bereitstellen. Daher sind die Pins am Chip für die Kommunikation verwendet und die meisten GPIO-Funktionalitäten werden über die Erweiterung erzielt. Zudem wurde darauf geachtet keine Zeitkritischen Signale über die Erweiterung zu realisieren, da hier jedes Mal per I²C erst Daten abgefragt werden müssen, bevor diese am ESP32 verarbeitet werden können, wodurch die Daten deutlich später zur Verfügung stehen, als wären sie direkt am Chip angeschlossen. Der GPIO-Expander besitzt dabei 2 Register mit jeweils acht Eingängen. Um es übersichtlich zu halten, wurde Register A nur für Eingangssignal und Register B nur für Ausgangssignale verwendet. Das hat außerdem den Vorteil, dass das Signal für Interrupts von Port B ignoriert werden kann, da dort keine Eingehenden Signale erwartet werden, wodurch ein Port mehr für andere Funktionalität direkt am Chip erhalten bleibt. So übernimmt der GPIO-Expander zum Beispiel das Auslesen der vom Funkempfänger gesendeten Signale.   
Ein Bild, das Text, Diagramm, Zahl, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.  
Dazu gehören neben dem Fehlersignal und der Rückgabe des Lernmodus auch die Empfangenen Signale auf Kanal 2, 3 und 4. Durch die Interrupt Funktionalität des Bauteils wird der Mikrocontroller benachrichtigt, sobald sich an diesen Werten etwas ändert. Neben den Signalen liest die Porterweiterung auch das LED-Signal aus, welches von der VCU gesendet wird. Da dieses nur noch in seltenen Fällen verwendet werden wird, nämlich wenn ein altes Kart mit einer neuen Platine aufgerüstet werden soll, hat dieses Signal keine hohe Priorität, weshalb die höhere Bearbeitungszeit, bis es am Mikrocontroller ankommt, vernachlässigbar ist. Gleichzeitig wird über die Erweiterung auch die LED gesteuert. LED\_ON ist dabei das Signal, welches den Transistor für die LED-Schaltet. Auch hier fallen minimal längere Verarbeitungszeiten nicht ins Gewicht. Dasselbe gilt für das Signal Activate\_Lernmodus. Hierrüber wird der Lernmodus des Funkempfängers aktiviert, mit welchem man neue Fernbedienungen einem Funkmodul zuweisen kann. Dieses Signal besteht aus mehreren Impulsen, welche mit Verzögerung gesendet werden, sodass hier genügend Zeit zwischen den Signalen besteht, um einen neuen Impuls über I²C an die Erweiterung zu übermitteln. Zwei weitere Signale, welche über die Port Erweiterung realisiert, werden sind CAN1\_S-Mode und CAN2\_S-Mode. Über diese Signale wird der Silent Mode für die CAN-Transceiver für CAN1 und CAN2 aktiviert. Dieses Signal eignet sich sehr gut, um über die langsamere Expansion realisiert zu werden, da dieses Signal nur sehr selten verändert wird. Bei den Signale TX\_2 und TX\_3 handelt es sich wieder um Signale, welche nur zur Kompatibilität mit alten Systemen vorgesehen sind. TX\_2 realisiert dabei die Aktivierung des Fahrmodus, wenn ein Signal per Funk empfangen wurde, TX\_3 sendet eine SOC-Anfrage zur VCU. Diese Signale kommen aber nur bei Karts alter Generationen zum Einsatz, die keinen CAN-Bus zur Kommunikation zur Option 1 führen, sondern bei denen das Signal noch über eine eigene Signalleitung an die VCU übermittelt wird. Die restlichen beiden Signale sind Steuersignale für einen über SPI gesteuerten RFID-Reader. Diese sind lediglich für testzwecke vorgesehen, um den sehr teuren RFID-Reader, welcher aktuell direkt per CAN angeschlossen ist, zukünftig ersetzen zu können durch ein deutlich günstigeres Bauteil. Deshalb wird hier auch ein Signal wie ein SPI ChipSelect und der Reset für den Controller über den langsamen GPIO-Expander realisiert, obwohl es sich bei Signalen für die SPI-Kommunikation um zeitkritischere Signale handelt. Allerdings werden diese Signale im normalen Betrieb keinerlei Rolle spielen. Um überhaupt Daten übermitteln zu können muss die Adresse des Geräts definiert werden. Das passiert über eine 7 Bit lange Adresse, wobei die letzten Drei Bit über die Eingänge A0 bis A2 definiert werden. Das ist vor allem dann relevant, wenn in einer Schaltung mehrere GPIO-Port Erweiterungen verbaut werden. Im vorliegenden System gibt es keine Überschneidungsgefahr bei den Adressen, weshalb die letzten Adressbits auf 0 0 0 festgelegt werden, definiert durch das Ground-Potenzial. Die Spannungsversorgung erfolgt auch hier mit 3.3V. Auch hier ist ein Abblockkondensator vorgesehen, um eine stabilere Spannungs-versorgung zu gewährleisten. Der Widerstand vor dem Reset Pin sorgt dafür, dass das lowaktive Signal auf einem definierten Pegel liegt und die Port Erweiterung dauerhaft aktiv ist, ohne ungewollt zurückgesetzt zu werden.  
Funkempfänger  
Das Funkmodul, welches die Signale für Aktivierung des Notaus, Aktivierung des Fahrmodus oder Abfrage des State of Charge empfangen wird, wird als Aufsteckplatine realisiert. Dafür wird das Empfangs – und Auswertmodul CX-12 R von SVS verwendet. Dieses ist mit den bisher Ein Bild, das Elektronik, Schaltung, Elektronisches Bauteil, Elektrisches Bauelement enthält.

KI-generierte Inhalte können fehlerhaft sein.verwendeten Funkfernbedienungen kompatibel und bietet alle Funktionalitäten, welche für das System benötigt werden. Das Modul besitzt 16 Pins, welche im Folgenden genauer betrachtet werden sollen, sowie die Beschaltung, welche näher beleuchtet und erklärt wird.

Der erste Pin ist für die Aktivierung eines Sleep Modus vorgesehen. Um den Stromverbrauch zu senken, wird nach einiger zeit eine Taktung aktiv. Diese Taktung tritt 10 Minuten nach Einschalten des Moduls oder dem letzten Datenempfang in Kraft. Ist diese Taktung aktiv, treten allerdings Verzögerungen beim Empfang und dem Verarbeiten der Funksignale auf. Diese Latenz kann bis zu einer Sekunde betragen. Über den Pin lassen sich dafür verschiedene Modi aktivieren. Dabei wird unterschieden, ob die Lötbrücken A und B auf der Platine gesetzt sind:

|  |  |  |
| --- | --- | --- |
|  | **Pin 1 Pegel** | |
| GND | VCC / unverbunden |
| **Lötbrücke B**  gesetzt | Taktung nach 10 Minuten seit Datenempfang | Dauerhaft in Taktung |
| **Lötbrücke B**  nicht gesetzt | Dauerbetrieb ohne Taktung | Taktung nach 10 Minuten seit Datenempfang |

Die Lötbrücke A bewirkt, das Pin 1 dauerhaft auf Ground gezogen wird.   
Die Platine wird standartmäßig ohne gesetzte Lötbrücken geliefert. Um den Arbeitsaufwand zu verringern, soll es nicht nötig sein, jede Platine noch einmal händisch bearbeiten zu müssen, um Lötbrücken zu setzen. Daher soll der Modus nur über den Pegel an Pin 1 bestimmt werden und beide Lötbrücken bleiben ungesetzt. Für das aktuelle System soll der Empfänger dauerhaft aktiv bleiben, somit also Pin 1 auf Groundpotential gezogen werden ohne gesetzte Lötbrücke. Das hängt damit zusammen, dass durch den Notaus auch Sicherheitsrelevante Funktionalitäten über den Funkempfänger realisiert werden, für den 1 Sekunde Latenz zu lange als Verarbeitungszeit sind.

Um Signale empfangen zu können, muss es zuerst möglich sein Fernbedienungen mit dem Empfänger zu verbinden. Dafür bietet das Funkmodul mehrere Modi, welche sich über Pin 2 aktivieren lassen. Die Lernmodi werden dabei durch unterschiedliche Impulse auf Ground Potential aktiviert:

|  |  |
| --- | --- |
| **Modus** | **Aktivierungsmethode** |
| Lernmodus I | GND, 1x kurz (<1s) |
| Lernmodus II | GND, 2x kurz (<1s) |
| Lernmodus III | GND, 3x kurz (<1s) |
| Lernmodus IV | GND, 4x kurz (<1s) |
| Löschmodus I | GND, 1x lang (>3s) |
| Löschmodus II | GND, 2x lang (>3s) |

Wenn ein neuer Sender mit dem Empfängermodul verbunden wird, wird dessen Adresse im Funkempfänger gespeichert. Dieser besitzt bis zu 60 Speicherplätze für Senderadressen. Bei Lernmodus I und III wird eine neue Fernbedienung mit dem Modul verbunden. Hierbei können nun über alle Tasten die unterschiedlichen Funktionalitäten von Kanal 1 bis 4 genutzt werden. Der Unterschied zwischen Modus I und III liegt darin, dass bei Modus I eine Bestätigung an den Sender übermittelt wird und der Nutzer somit an der Fernbedienung eine Rückmeldung über ein eventuelles Fehlschlagen der Verbindung erhält. Mit Modus II und IV lässt sich eine einzelne Taste anlernen. Hier wird nicht die gesamte Fernbedienung mit dem Empfangsmodul gekoppelt, sondern lediglich die jeweilige taste. Auch hier unterscheiden sich die beiden Modi nur in der Rückmeldung an den Sender. Die verbleibenden beiden Modi werden genutzt, um bereits angelernte Fernbedienungen wieder zu lösen. Dabei entfernt Modus I einen Eintrag aus der Liste. Dabei handelt es sich um die Adresse des Senders, von welchem das Modul ein Signal empfängt, solange der Löschmodus aktiv ist. Löschmodus II entfernt hingegen alle gespeicherten Einträge aus der Adressen Speicherliste. Auf dem bisherigen Gesamtmodul wurde diese Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.Funktionalität über einen Knopf auf der Platine realisiert. Das hatte zur Folge, dass eine neue Fernbedienung nur im Ausgebauten Zustand verbunden werden konnte. Mit dem neuen System soll das auch über den Mikrocontroller gesteuert werden können und damit eine Aktivierung über die Netzwerkverbindung oder das Display möglich sein. Nichtsdestotrotz ist weiterhin ein Knopf vorgesehen, um das Funkmodul ohne den Chip testen zu können. Bei dem Knopf handelt sich dabei um einen Drucktaster, welcher den Pin auf Ground zieht. Der Widerstand R412 dient dabei als Pull-Up Widerstand, um bei offenem Schalten einen definierten Spannungspegel am Pin zu garantieren. Der Pin ist zusätzlich mit der GPIO Port Erweiterung am ESP 32 verbunden, über welcher nun durch Schalten des GPIO-Ports gegen Ground ebenfalls der Lernmodus aktiviert werden kann.

Rückmeldung über den Aktivierten Lernmodus gibt Pin 3. Bei dem Pin handelt es sich um einen LED-Ausgang, welcher durch unterschiedliche Frequenz unterschiedliche Modi anzeigt.

|  |  |
| --- | --- |
| **Modus** | **Anzeige** |
| Normaler Betrieb | 1x blinken pro empfangenes Signal |
| Lernmodus I | 1x Blinken pro 2 Sekunden |
| Lernmodus II | 2x Blinken pro 2 Sekunden |
| Lernmodus III | 3x blinken pro 2 Sekunden |
| Lernmodus IV | 4x blinken pro 2 Sekunden |
| Löschmodus I | Blinkt dauerhaft |

Diese Rückmeldung soll für Test- und Debugging Zwecke auch visuell über eine LED dargestellt werden, soll aber auch vom Mikrocontroller ausgewertet werden können, um Rückmeldung per Display oder Netzwerk geben zu können, sobald die Modi über den ESP32 aktiviert wurden. Da es sich bei dem Ausgang um ein Digitalsignal bis maximal 3.3V handelt, kann das Signal einfach an einen der Pins der Porterweiterung des Mikrocontrollers angeschlossen werden. Um einen Stabilen High-Pegel für das Lowaktive Signal zu erzeugen, ist R402 als Pull-Up Widerstand dazwischen geschalten. Die LED wird zwischen den Ausgang des Funkmoduls und ein 3.3V Signal geschalten, da es sich bei dem Signal um ein lowaktives handelt. Als Vorwiderstand dient hier R401. Der Widerstandswert ergibt sich dabei aus dem gewünschten Strom, der durch die LED fließen soll:  
Durch die LED fließt damit ein Strom von 5 mA.

Bei Pin 4 handelt es sich um den ersten digitalen Ausgang des Funkempfängers. Dieser Ausgang schaltet auf High, wenn ein Funksignal empfangen wird, ausgelöst durch den Druck der Taste 4 auf einer der verbundenen Fernbedienungen. Der Ausgang bleibt danach so lange auf 3.3V Spannungspegel bis die Taste losgelassen wird, mindestens aber 1.5 Sekunden. Der Pin wird zur Auswertung direkt mit dem Mikrocontroller verbunden. Zusätzlich wird noch eine LED hinzugefügt, welche den Pegel des Ausgangs anzeigt, um das Debugging zu vereinfachen und die Fehlersuche zu erleichtern. Da es sich hier um ein High-Aktives Signal handelt wird die LED gegen Ground geschalten. Auch hier sollen 5 mA Strom durch die LED fließen, weshalb auch hier wieder 220Ω als Vorwiderstand genutzt werden.   
Um digitale Ausgänge handelt es sich auch bei den Pins 5, 8 und 9 für die Kanäle 3, 2 und 1. Diese verhalten sich genau wie der Digitale Ausgang für Kanal 4 mit der Erweiterung, dass die 3 restlichen Kanäle zusätzlich zum ESP32 auch noch einen Transistor schalten, welcher das Signal zusätzlich bei alten Systemen an die VCU übermitteln soll. Bei dem Signal von Kanal 1 fällt zusätzlich noch die Verbindung zum Mikrocontroller weg, da es sich hier um ein Sicherheitsrelevantes Signal handelt, welches ausfallsicher realisiert werden muss und deshalb nicht softwareseitig verarbeitet werden soll.

Ein weiterer Ausgang für empfangene Signale ist ein Analogausgang auf Pin 6. An diesem Pin wird ein Analogwert zwischen 3.3V und 0V mit 10 Bit Auflösung ausgegeben. Für unser System wird dieser Ausgang jedoch nicht benötigt, weshalb er unverbunden bleibt.

Der nächste Pin wird zur Konfiguration des Moduls benötigt. Das Modul erwartet Signale auf einer Frequenz von 433,62 MHz. Sollte diese Frequenz nicht nutzbar oder gestört sein besteht die Möglichkeit auf eine Alternativfrequenz von 434,22 MHz auszuweichen. Die Auswahl dieser Frequenz wird mit dem Pegel an Pin 7 definiert. Liegt der Pin auf GND, arbeitet das Modul auf Frequenz 2, liegt der PEGEL BEI 3.3V oder wird der Pin nicht beschalten, arbeitet der Empfänger mit Frequenz 1. Da bereits ein ähnliches Modul im Einsatz war, welches mit denselben Frequenzen arbeitete, und dort kaum Probleme auftraten, wird die Frequenz einmalig festgelegt. Zu testzwecken soll trotzdem die Möglichkeit bestehen im fehlerfall auf alternative Frequenzen auszuweichen, weshalb ein Widerstand als Pull-Up auf 3.3V und ein Pull-Down Widerstand gegen Ground vorgesehen sind, wobei der Pull-Down Widerstand standartmäßig unbestückt bleibt.

Die folgenden Pin 10 und 11 definieren die Versorgung und logischen Pegel für den Funkempfänger. Pin 10 ist die Versorgungsspannung, welche auch gleichzeitig den logischen High-Pegel definiert, und zwischen 2.0 und 3.6V liegen muss. Als Versorgungsspannung wird daher 3.3V verwendet. Pin 11 definiert das Groundpotential.

Neben den Eingängen, über welche das Modul Signale empfängt, besitzt das Modul auch mit Pin 12 einen Analogeingang. Dieser wird standartmäßig dafür verwendet, um Batteriespannungswerte zu übermitteln, wenn das Modul nicht mit einer konstanten Spannungsquelle verwendet wird. Das Modul sendet entsprechend des anliegenden Spannungspegels ein Signal, welches von der Funkfernbedienung empfangen wird und mittels einer LED auf der Fernbedienung visualisiert wird.

Dabei gelten folgende Grenzwerte:

|  |  |  |
| --- | --- | --- |
| **Spannung U** | **Rückmeldung** | **Bedeutung** |
| U > 1.2V | LED aus | Ausreichende Versorgungsspannung |
| 1.2V > U > 1.1V | LED blinkt | Abnehmende Versorgungsspannung |
| 1.1V > U | LED blinkt schnell | Zu geringe Versorgungsspannung |

Für das vorliegende System ist die Ausgabe eines Analogen Spannungswerts nicht relevant, weshalb der Pin Dauerhaft auf 3.3V gelegt wird, sodass ein durchgehend ausreichendes Spannungssignal erkannt wird und keine Rückmeldung an die Fernbedienung geschieht.

Zusätzlich zu den LEDs als visuelle Rückgabe gibt es auch die Möglichkeit auditive Rückmeldung vom Modul zu erhalten. Dafür ist am Empfänger der Ausgang für eine Piezoscheibe vorgesehen. Dieser Ausgang gibt vordefinierte Signale bei bestimmten Ereignissen aus, welchem dem Nutzer Rückmeldung über den Status des Systems geben. So gibt die Piezo-Scheibe Tonfolgen aus, wenn die Versorgungsspannung unter einen definierten Schwellwert sinkt. Auch die Aktivierung und Deaktivierung des Lernmodus wird mit je einer individuellen Tonfolge pro Modus ausgegeben. Auch das Einlernen und Löschen eines Senders wird mit je einer Tonfolge quittiert, genauso wie das Löschen der gesamten Speicherliste. Da diese Informationen bereits durch Auswertung der LED-Pins durch den Mikroprozessor erhalten werden und dieser die Informationen auf Display oder Netzwerk visuell zur Verfügung stellt, besteht für dieses System kein Bedarf diese Informationen zusätzlich noch über Tonfolgen auszugeben. Daher ist der Ausgang unbeschalten.

Ein weiterer LED-Ausgang, welcher auch vom Mikrocontroller ausgewertet wird, ist Pin 14. Hier handelt sich um eine visuelle Ausgabe von aufgetretenen Fehlern. Diese Fehler werden auch hier durch unterschiedlich viele Lichtimpulse zurückgegeben. Dabei wird unterschieden zwischen normalem Empfangsbetrieb und Aktiviertem Lösch-/Lernmodus:

|  |  |  |
| --- | --- | --- |
| **Normaler Empfangs-betrieb** | **Fehler** | **Darstellung** |
| Senderversorgungsspannung gering | Blinkt 1x |
| Senderversorgungsspannung kritisch | Blinkt 3x |

|  |  |  |
| --- | --- | --- |
| **Lern- / Lösch-modus** | **Fehler** | **Darstellung** |
| Eintrag konnte nicht gelöscht werden | Blinkt 2x |
| Speicherliste ist voll | Blinkt 3x |
| Sender wurde bereits eingelernt | Blinkt 2x |

Zu Test und Debugging Zwecken ist auch hier weiterhin eine LED zur visuellen Rückmeldung vorgesehen, um die Fehler auch ohne Mikrocontrollerauswertung schnell und einfach identifizieren zu können. Auch hier wird ein gewünschter Strom von 5 mA vorgegeben, sodass sich auch hierwieder ein Vorwieder stand von berechnen lässt. Auch hier handelt es sich wieder um ein Lowaktives Signal, sodass die LED zwischen 3.3V und Pin geschalten werden muss, um bei Aktivierung des Signalpins zu leuchten. Zusätzlich wird der Pin noch mit einem der Eingänge an der GPIO-Porterweiterung des Mikroprozessors verbunden, um die Fehler am ESP32 auswerten und für den Kunden darstellen zu können. Dafür ist auch der Widerstand R407 als Pull-Up vorgesehen, um bei dem Lowaktiven Signal einen definierten High-Pegel zu erzeugen.

Die Pins 15 und 16 sind nun noch als Anschlüsse für eine Antenne vorgesehen. Standartmäßig wird das Modul mit einer einfachen Drahtantenne ausgestattet. Dafür ist Pin 16 vorgesehen. Für die Verwendung einer abgesetzten Antenne lässt sich aber auch ein Koaxialkabel mit einer Impedanz von 50Ω anschließen, in diesem Fall wird Pin 15 für den Anschluss der Schirmung und Pin 16 für den Anschluss des Innenleiters verwendet. Für die Verwendung einer Drahtantenne gibt das Datenblatt eine Länge von 173mm vor. Diese berechnet sich für eine Monopolantenne nach folgender Formel:

VCU-Signale

Ein Bild, das Text, Diagramm, Schrift, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.Wie bereits weiter oben beschrieben sollen die ersten drei Kanäle des Funkmoduls auch ein Signal an die VCU senden. Zu diesem Zweck gibt es drei Transistorschaltungen auf der Platine, welche jeweils ein Signal erzeugen, welches anschließend über eine eigene Signalleitung an die VCU übertragen wird. Diese Signale sind allerdings nur noch aus Kompatibilitätsgründen für ältere Systeme vorgesehen, welche noch keine CAN-Verbindung zur Option1 vorgesehen haben. Einzige Ausnahme bildet der Notaus, welcher über die Funkfernbedienung aktiviert werden kann. Dieser muss aus Sicherheitsgründen Ausfallsicher realisiert werden, weshalb er nicht über den Mikrocontroller geführt werden darf. Dieses Signal ist weiterhin als eigene Leitung von der Option 1 zur VCU vorgesehen. Die Schaltung ist dafür ebenfalls drahtbruchsicher realisiert. Das bedeutet, dass der Notaus sowohl bei einem erkannten Funksignal als auch einer defekten Leitung als aktiviert erkannt wird, sodass ein Schaden am Kabel nicht zu einem Ausfall des Sicherheitssystems führt. Das wird erreicht, indem als Zustand „Notaus aktiviert“ der Stromlose zustand definiert wird. Im Idle Zustand fließt somit dauerhaft ein geringer Strom. Ein Abfall dieses Stromflusses führt zur Erkennung eines aktivierten Notaus. In der Schaltung wird dies durch einen Transistor als Low-Side Schalter realisiert. Das Signal Notaus\_VCU, welches durch über den Stecker zur VCU geführt wird, wird durch den Widerstand R902 dauerhaft auf 12V gezogen. Somit wird im Zustand „Notaus nicht aktiv“ dauerhaft ein 12V Signal erkannt. Liegt nun am Eingang TX\_1, welcher direkt mit dem Pin am Funkempfänger Modul verbunden ist, ein logischer High-Pegel an, Schaltet der Transistor und zieht den Ausgang gegen Ground. Am Steuergerät wird nun Kein Pegel mehr erkannt, was zum Auslösen der Notausprozedur führt. Das ist dasselbe Verhalten, als würde das Kabel beschädigt und den Kontakt öffnen, obwohl der Transistor nicht durchgeschalten ist. Zur Einfacheren Fehlersuche und zu Debugging Zwecken wurde zusätzlich noch eine LED hinzugefügt. Die LED soll leuchten sobald der Notaus durch den Transistor aktiviert wurde, der Ausgang also auf Groundpotential liegt. Die LED wird deshalb zwischen 12V und Ausgang geschalten. Zusätzlich wird ein Vorwiderstand hinzugefügt, um den Strom durch die LED auf 5mA zu begrenzen.   
Durch Verwendung und Verfügbarkeit stehen nur Widerstände der E12 Reihe zur Auswahl. Dort ist der nächste Widerstandswert 1.8 kΩ.  
Damit ergibt sich für den Strom

Bei dem Transistor handelt es sich um einen N-kanal MosFET, welcher als Schalter eingesetzt wird. Der Transistor schaltet wenn , wobei die Threshold Spannung bei 1V liegt. Dadurch, dass Source dauerhaft auf Ground liegt, gilt immer , damit gilt immer

somit schaltet der Transistor wenn gilt

Diese Bedingung ist mit TX\_1 auf 3.3V erfüllt, sodass der Ausgang auf Ground gezogen wird, sobald ein Signal auf Kanal 1 vom Funkempfänger erkannt wird. Der Gatewiderstand R901 ist dabei vorhanden, um die Gateströme zu begrenzen und das Funkmodul zu schützen.

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.Sehr ähnlich zu der Transistorschaltung für das Notaussignal sind auch die Schaltungen für das RemoteDrive-Signal und die SOC-Abfrage. Anders als das Notaussignal müssen diese Signale aber nicht drahtbruchsicher realisiert werden. Stattdessen liegt der Ausgang RemoteDrive\_VCU standartmäßig auf Ground und wird beim Schalten des MosFET auf 12V gezogen. Der Widerstand R1002 dient hier also als Pull-Down Widerstand. Am Schalten des MosFET hat sich im Vergleich zur Schaltung für das Notaus Signal nichts verändert. Dadurch das der Transistor gedreht ist und Source weiterhin auf Ground liegt, schaltet der Transistor, sobald die 3.3V als High-Pegel am Gate des Transistors anliegen. Auch hier soll eine LED hinzugefügt werden, welche optisch den Pegel des Ausgangs anzeigen soll. Diese wurde hier ebenfalls so eingebaut, das sie leuchtet, sobald der Transistor geschalten hat. In diesem Fall liegt der Ausgang auf 12V, sodass die LED zwischen Ground und Ausgang platziert werden muss. Auch wurden als Vorwiderstand 1.8kΩ gewählt, da sich an den Bauteilen zur Notaus-Schaltung nichts verändert hat. Unterschiedlich zur Notaus-Schaltung ist jedoch, dass das Eingangssignal TX\_2 nicht direkt von dem Funkempfänger kommt, sondern vorher vom Mikrocontroller ausgelesen wurde, welche jetzt den Transistor steuert. Das hat den Zweck, dass über Software steuerbar ist, ob die Signale am Ausgang gesetzt werden sollen, da diese nur als Backup für alte Systeme vorgesehen sind. Für die Schaltung zur Erzeugung des State of Charge Signals gelten dieselben Aussagen, da die Schaltungen zur Erzeugung von RemoteDrive-Signal und SOC-Abfrage identisch sind.

CAN-Transceiver

Statt die Signale jeweils über ein eines Datenkabel an die VCU zu übermitteln, sollen sie zukünftig als CAN-Signal übermittelt werden. Dafür benötigt der Mikroprozessor jedoch Zusatzbeschaltungen. Zwar besitzt der ESP32 intern einen TWAI-Controller, für die Kommunikation auf dem CAN-Bus ist jedoch zusätzlich ein CAN-Transceiver notwendig, welcher als Interface zwischen dem CAN-Controller und dem physikalischen CAN-Bus fungiert. Dafür wird hier der TJA1051T-3 als Bauteil eingesetzt. Der Can-Transceiver übernimmt dabei die Umsetzung der vom Controller erzeugten CAN-Frames auf physikalische Spannungspegel auf dem Bus. Andersherum wandelt er genauso erhaltene Buspegel in empfangene Nachrichten um, sodass die Frames schließlich vom CAN-Controller ausgewertet werden können. Damit das Funktioniert muss der Transceiver jedoch auch richtig beschalten werden. Das Bauteil erwartet eine Versorgungsspannung zwischen 4.5V und 5.5V, wodurch sich ein Spannungspegel von 5V hervorragend anbietet. Da das restliche System jedoch auf einem Logischen Spannungslevel von 3.3V arbeitet wird hier die Bauteilvariante „T-3“ eingesetzt. Diese besitzt einen weiteren Spannungseingang , mit welchem der die Versorgungsspannung für den I/O Level Adapter festgelegt wird, also die Definition des Logikpegels für die Eingangssignale. Dieser wird auf 3.3V gelegt und definiert für den CAN-Transceiver somit denselben Logikpegel wie für die restliche Schaltung. Die beiden Kondensatoren C1702 und C1703 sind hierbei wieder als Abblockkondensatoren vorgesehen. Sie sollen Spannungsspitzen abblocken und so das Bauteil schützen und gleichzeitig kurzfristige Spannungseinbrüche abdämpfen. Ein weiter Kontrolleingang ist der Pin S. Über diesen kann der Silent Mode für den CAN-Transceiver aktiviert werden. Im Silent-Mode ist der Sender deaktiviert, die Bus-Pins gehen über in einen rezessiven Zustand. Alle anderen IC-Funktionen, einschließlich des Empfängers, arbeiten weiterhin wie im Normal-Mode. Der Silent-Mode kann verwendet werden, um zu verhindern, dass ein fehlerhafter CAN-Controller Ein Bild, das Text, Diagramm, Screenshot, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.die gesamte Netzwerkkommunikation stört.   
Die restlichen 4 Pins dienen der tatsächlichen Datenübertragung. Bei den Pins TXD und RXD handelt es sich um die serielle Kommunikation mit dem CAN-Controller. Von diesem erhält der Transceiver fertige CAN-Frames und sendet seinerseits wieder empfangene Bussignale, welche er in einen CAN-Frame umgewandelt hat. Die tatsächliche Buskommunikation findet über die Pins CANH und CANL statt. Diese bilden die physikalischen CAN High und CAN Low Leitungen ab. Der Transceiver wandelt die Nachrichten um und sendet diese als dominante und rezessive Pegel auf den Bus. Zwischen CANH und CANL findet sich noch ein Widerstand. Dieser dient als Terminierungswiderstand für den Bus. Er ist an beiden Enden des Busses notwendig, um Störungen zu minimieren und Reflexionen zu vermeiden. Im CAN-Protokoll ist die Impedanz des Busses auf 120Ω definiert, dadurch ergibt sich auch der Widerstandswert. Durch die Verwendung von 2 CAN-Bussen für Daten und den RFID-Reader findet sich diese Schaltung zweimal auf der Platine. Der ESP32 besitzt jedoch nur einen CAN-Controller, weshalb die Kommunikation auf dem zweiten CAN-Bus durch einen Externen Controller realisiert werden muss. Dafür wird die oben gezeigte und erklärte Schaltung um einen CAN-Controller erweitert. Als CAN-Controller kommt hierbei der MCP2515-I/ST zum Einsatz. Dieser bietet die selbe Funktionalität wie der interne TWAI-Controller, mit dem Unterschied, dass die Kommunikation über SPI stattfindet. Die Daten, aus denen der Controller einen CAN-Frame bauen soll, werden dabei per SPI übermittelt und der Controller übernimmt dieselben Aufgaben wie der TWAI-Controller. Dazu gehören das Erstellen der CAN-Frames aus den übermittelten Daten, ID und den berechneten Kontrollfeldern wie CRC oder DLC und die relevanten Bits wie Acknowledge oder End of Frame. Zusätzlich übernimmt der Controller Aufgaben wie Arbitrierung und die damit verbundenen Steuerung, wann die Daten tatsächlich auf den Bus gesendet werden können. Auch die Auswertung der empfangenen Frames übernimmt der Controller. Diese Daten werden wiederum an den Microcontroller übermittelt, welcher die Abarbeitung der Daten im Programmcode durchführen kann. Die SPI-Kommunikation findet dabei über die Pins SO, SI und SCK statt. Bei SCK handelt es sich um das Clocksignal, welches grundlegend für die Synchrone Kommunikation zwischen Master und Slave ist. SO ist das „Slave Out“ Signal, welches im restlichen System als MISO verwendet wird. Über dieses Signal kommuniziert der Slave und sendet seine Nachrichten und Daten an den ESP32. Über den Pin SI kommuniziert der ESP32 mit dem CAN-Controller. Hier empfängt der Slave seine Daten, welche ihm vom Master über die MOSI-Signalleitung übermittelt werden. Damit die Kommunikation zustande kommt muss vorher der CAN-Controller über den ChipSelect ausgewählt werden. Das passiert über den CS-Pin am CAN-Controller. Wird dieser gegen Ground gezogen, wird der Controller informiert, dass er nun mit dem Master kommunizieren soll. Zusätzlich bietet der SPI-Controller im CAN-Controller eine Interuptfunktionalität über den Pin . Über diesen Pin wird ein Signal an den ESP32 gesendet, sobald eine Nachricht empfangen wurde oder ein Fehler aufgetreten ist. Dieser Pin kann im Mikrocontroller als Auslöser für Ein Bild, das Text, Diagramm, Zahl, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.einen Interrupt definiert werden, sodass eingehende Nachrichten Zeitnah bearbeitet werden können. Versorgt wird der Controller mit 3.3V. Auch hier wird mit C306 ein Abblockkondensator zum Schutz des Bauteils und der Sicherstellung der Spannungsversorgung hinzugefügt. Der Reset-Pin wird mit einem Pull-Up Widerstand auf High gezogen, sodass er bei jedem Neustart automatisch aktiviert und bei Shutdown resettet wird.   
Neben dem Interrupt Pin vom SPI-Controller gibt es auch Interrupt Pins, welche direkt vom CAN-Controller selbst gesteuert werden. So indizieren die Pins RX0BF und RX1BF eine empfangene Nachricht im Receive Buffer RXB0 oder RXB1. Auch diese können am ESP32 als Interruptquelle definiert werden. Da bereits die Interruptfunktionalität über SPI genutzt wird, wird die direkte Interrupt Möglichkeit aus dem Buffer des Controllers nicht benötigt und die Pins bleiben unbeschalten. Dasselbe gilt für die Pins TX0RTS, TX1RTS und TX2RTS. Diese Signale bieten die Möglichkeit ein Request To Send Signal direkt an den Buffer im Controller zu senden, woraufhin der Controller seine im Buffer gespeicherten Frames an den Transceiver übermitteln würde. Diese Funktionalität wird ebenfalls über Befehle via SPI realisiert, weswegen die Pins nicht benötigt werden. Als nächstes sollen die Funktionen der Pins OSC1 und OSC2 betrachtet werden. Das Datenblatt empfiehlt für eine Stabile Kommunikation einen externen Oszillator als System Clock für den CAN-Controller. OSC1 ist dabei der Input für den Timer, OSC2 der Output, welcher zum Oszillator zurückgeführt wird. Der ausgewählte Oszillator arbeitet dabei auf einer Frequenz von 8 MHz. Das Datenblatt gibt dabei für den Oszillator die Beschaltung vor mit welcher das Signal möglichst stabil wird und die Start-Up Zeit, welche vom Taktgeber abhängt, möglichst gering bleibt. Die empfohlenen Werte für einen 8MHz Oszillator sind dabei mit 22pF angeben, woraus sich auch die Kondensatoren in der Schaltung ergeben. Das Clock Signal kann zusätzlich über den Ausgang CLKOUTSOF abgegriffen werden. Dieser Ausgang stellt ein Clocksignal zur Verfügung, welches mittels eines Prescalers verändert werden kann, um als Software Timer für genutzt werden zu können. Die Pins TX und RX übernehmen bei dem Controller nun die tatsächliche Kommunikation mit dem Transceiver. Hier werden die tatsächlichen Frames an den CAN-transceiver übermittelt, welcher anschließend die Kommunikation auf dem CAN-Bus übernimmt. TX dient dabei dazu die Frames an den Transceiver zu übermitteln, RX empfängt die Daten des Transceivers.

LED-Treiber

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.Da die Pins des Mikrocontrollers nicht genug Strom liefern und das LED-Modul einen Spannungspegel von 12 V benötigt, wird eine Transistorschaltung als LED-Treiber notwendig. Der Treiber wird hier als Low-Side Switch realisiert, welcher den negativen Anschluss der LEDs gegen Ground verbindet. Als Schalter wird hier ein MosFET eingesetzt, welcher vom ESP32 angesteuert wird, wenn die LED leuchten soll. Die positive Seite der LEDs liegt dabei dauerhaft auf 12V, die Negative Seite der LEDs wird auf Ground gezogen, sobald der Transistor durchschaltet. Der Gatewiderstand ist vorhanden, um den Pin am Mikrocontroller vor Stromspitzen zu schützen. Auch hier gilt dieselbe Rechnung wie bei den Transistorschaltungen für die VCU-Signale. Dadurch, dass Source dauerhaft auf Ground liegt, schaltet der Transistor sobald gilt. Das ist der Fall, sobald der Pin am ESP32 das Gate auf 3.3V zieht.

LED-Signal Auswertung

Ein Bild, das Text, Schrift, Reihe, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.Neben den Signalen, welche für veraltete Systeme noch per Signalleitung an die VCU übermittelt werden, übermitteln diese Karts auch die LED-Signale von der VCU noch mittels einer eigenen Signalleitung. Diese Signale müssen ausgewertet werden, um an die LED-Treiber Schaltung weitergegeben zu werden. Da es sich um ein digitales 12V Signal handelt, reicht es das Signal auf maximal 3.3V zu normieren, sodass das Signal direkt an einen der Pins des ESP32 angeschlossen werden kann. Das Signal vom Steuergerät kommt über den Eingang VCU\_LED direkt vom Stecker. Anschließend wird das Signal von 12V mittels eines Spannungsteilers auf maximal 3.3V am Pin normiert. Die Spannung am Ausgang LED\_Signal, welcher mit dem Pin der GPIO-Erweiterung vom Mikrocontroller verbunden ist, lässt sich folgendermaßen berechnen:  
Daraus ergeben sich die Formel für

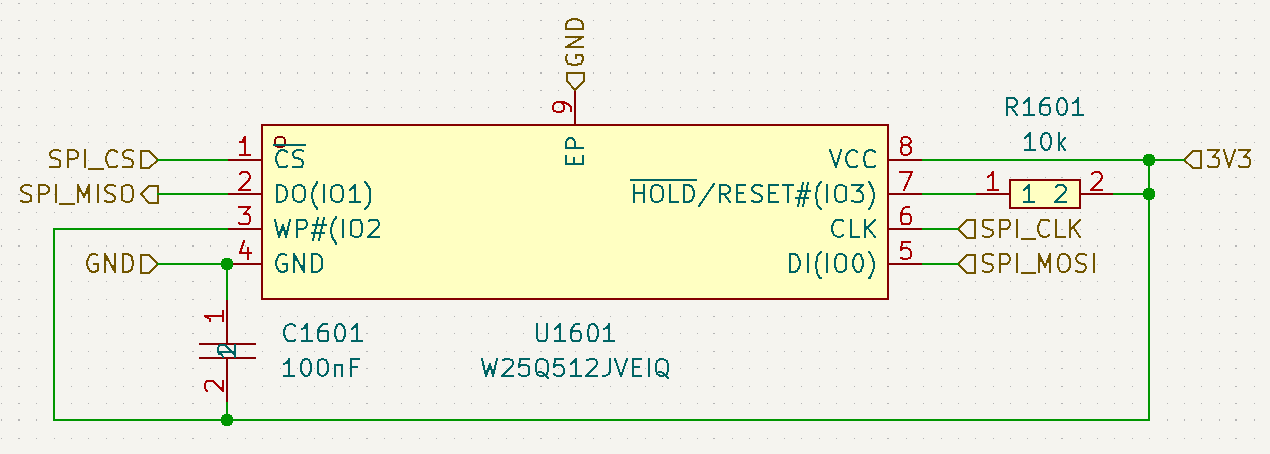
Für ergibt sich damit für

Durch die E-Reihe ergibt sich damit der nächste Wert als 27kΩ. Mit 27kΩ beträgt die maximale Spannung am ESP32-Pin

Damit erkennt der ESP32 an seinem Pin 0V, wenn kein Signal gesendet wird und 3.24V als High Pegel.  
Zum Schutz des Mikrocontrollers ist zusätzlich noch eine Zenerdiode zwischen den Pin und Ground platziert. Die Zenerdiode hat eine Durchbruchspannung bei 3.3V. Das bedeutet, sobald die Spannung am Mikrocontroller 3.3V überschreitet, wird die Diode leitend und hält den Pin bei 3.3V und schützt den Controller so vor Schäden durch Überspannung.

SPI-Flash

Um die Funktionalität eines persistenten Speichers für Logdaten und Einstellungen zu erreichen, ist auf der Platine ein Flash Speicher vorgesehen. Dieser Speicher ist per SPI-Kommunikation adressierbar und beschreibbar. Der Speicher ist dabei persistent, das bedeutet das er seinen Inhalt behält, selbst wenn er nicht mehr bestromt wird. Die Daten sind auch beim nächsten Start-up erhalten. Die Größe des Speichers beträgt 4 MB, für Speicherung von Textdateien und einzelnen Integerwerten ist das völlig ausreichend.   
Der Speicher wird dafür mit 3.3V versorgt. Wie bei allen anderen Bauteilen ist auch hier der Kondensator C1601 als Abblockkondensator vorgesehen. Zusätzlich zum Groundpin GND besitzt der SPI-Flash noch ein großes Lötpad EP unter dem Bauteil, welches ebenfalls auf Ground gezogen wird.



Die SPI-Kommunikation findet bei dem Speicher über die Pins DO und DI statt. DO steht dabei für „Data Out“ und wird daher mit dem MISO-Signal verbunden, dass der Speicher darüber Daten an den Mikroprozessor senden kann. Der Daten empfang über die MOSI-Signalleitung findet am „Data In“ (DI) Pin statt. Hierüber empfängt der Flash befehle und Daten, welche er persistent speichern soll. Als Taktgeber für die SPI-Kommunikation dient der CLK-Pin. Hier erhält der Chip das Clock-Signal vom SPI-Bus. Das letzte SPI-Signal ist der Chip Select für den Speicher. Über diesen wird dem Speicher mitgeteilt, dass nun mit ihm kommuniziert werden soll. Zusätzlich zu den Pins für die SPI-Kommunikation besitzt der Chip noch zwei weitere externe Steuersignale. Eines davon ist Pin 7 für die HOLD-Funktionalität. Der HOLD-Pin ermöglicht es, das Gerät anzuhalten, während es aktiv ausgewählt ist. Wenn HOLD auf Low gesetzt wird, während Chip Select auf Low ist, dass Gerät also gezielt angesprochen wird, wird der Data Out Pin deaktiviert und Signale an den DI- und CLK-Pins werden ignoriert. Sobald der Pegel am HOLD-Pin wieder auf High gesetzt wird, geht der Chip wieder in normalen Betriebsmodus über und nimmt seine Kommunikation wieder auf. Das ist vor allem dann nützlich, wenn sich mehrere Geräte dieselben SPI-Signale teilen, zum Beispiel bei mehreren parallelen Speicherchips. Da dies hier nicht der Fall ist, wird diese Funktionalität nicht benötigt. Weil es sich bei dem Signal um ein lowaktives handelt, wird der Pin mittels eine s Pull-Up Widerstands dauerhaft auf 3.3V gezogen, um ungewollte Störungen durch einen undefinierten Pegel am Pin zu vermeiden. Ein weiteres Steuersignal ist WriteProtect (WP) an Pin 3. Über dieses Signal lässt sich das Schreiben in das Statusregister des Chips verhindern. Zusammen mit Steuerbits im Chip lässt sich damit der gesamte Speicher oder einzelne Bereiche gezielt vor Überschreiben schützen. Da es für das System keine besonders geschützten Speicherbereiche benötigt, wird auch diese Funktionalität nicht genutzt. Der Pin wird dafür dauerhaft auf High gezogen, da es sich auch hier um ein lowaktives Signal handelt.

Power Supply

Wichtig für das System ist die Erzeugung der verschiedenen Spannungslevel. Die Versorgung der Platine findet mit den 48V direkt aus der Batterie des Karts statt. Mit diesem Pegel kann jedoch kein einziges Bauteil auf der Platine arbeiten, dies benötigen 12V, 5V und hauptsächlich 3.3V. Alle diese Spannungspegel müssen daher auf der Platine selbst erzeugt werden. Besonderer Fokus liegt dabei darauf möglichst stabile Versorgungspannungen zu erzeugen, um vor allem Probleme in der Kommunikation zwischen den Komponenten zu verhindern. Auch die Auswahl der Spannungswandler ist wichtig, da diese nur einen bestimmten Strom liefern können. Der Stromverbrauch ist aber bereits durch die Bauteile definiert. Zudem muss darauf geachtet werden Spannungswandler nicht unnötig zu kaskadieren. Durch Kaskadierungen sinkt der Wirkungsgrad der Schaltungen enorm, wodurch die gleichzeitig die Verlustleistung zunimmt. Das hat deutlich mehr Abwärme zur Folge welche Probleme auf der Platine schaffen kann. Als erstes soll daher der Strom bestimmt werden, den die jeweiligen Spannungswandler treiben können müssen. Dazu werden die Maximalströme der Bauteile aus dem Datenblatt betrachtet, um auch den Worst Case abdecken zu können.

|  |  |  |
| --- | --- | --- |
| **Bauteil** | **Versorgungsspannung** | **Stromverbrauch** |
| CAN-Controller | 3.3V | 10 mA |
| Funkreceiver | 3.3V | 18.2 mA |
| GPIO Port Erweiterung | 3.3V | 125 mA |
| Mikrocontroller | 3.3V | 250 mA |
| USB to Serial Wandler | 3.3V | 12 mA |
| Display | 3.3V | 33 mA |
| SPI-Flash | 3.3V | 40 mA |
| RFID-Reader | 3.3V | 26 mA |
| **Gesamt** |  | **515 mA** |

Alle Bauteile, welche mit 3.3V versorgt werden, benötigen im Worst Case 515 mA. Um auf bereits vorhandene Lagerbestände zurückzugreifen und nicht viele verschiedene Bauteile beschaffen zu müssen, sollen möglichst Bauteile eingesetzt werden, welche bereits in anderen Systemen in der Firma zum Einsatz kamen. Für einen Spannungswandler zu 3.3V gibt es daher zwei mögliche Optionen:

1. **Bauteil P7803-500:**

Bei dem Bauteil handelt es sich um einen DCDC-Wandler, welcher direkt aus dem 48V Eingangssignal ein 3.3V Signal erzeugt. Das hätte den Vorteil, dass unnötige Kaskadierungen vermieden werden, wodurch der Wirkungsgrad steigt. Außerdem ist das Spannungssignal stabiler, da nicht die Störeinflüsse mehrerer Spannungswandler weitergegeben werden können. Jedoch ist dieses Bauteil nicht in der Lage den benötigten Strom zu liefern. Der Ausgangsstrom beträgt maximal 500mA, sodass das Bauteil den Maximalstrom nicht liefern kann, was auf Dauer zur Zerstörung des Moduls oder der Schaltung führt.

1. **Bauteil AN\_SY8089A:**bei dem Bauteil handelt es sich um einenSpannungswandler, welcher die 3.3V aus einem 5V Eingangssignal generiert. Dieses Bauteil hat den Vorteil, dass die Ausgangsspannung meist stabiler und weniger Störanfällig ist, je kleiner der Spannungsunterschied zwischen Eingang und Ausgang ist. Nachteile sind die durch die Kaskadierung von 48->5V und von 5V -> 3.3V bedingten höheren Leistungsverluste und Wärmeabgabe. Jedoch ist dieses Bauteil in der Lage bis zu 2A kontinuierlich an Strom zu liefern. Deshalb wird dieses Bauteil in der Schaltung verwendet.

|  |  |  |
| --- | --- | --- |
| **Bauteil** | **Versorgungsspannung** | **Stromverbrauch** |
| CAN-Transceiver 1 | 5V | 50mA |
| CAN-Transceiver 2 | 5V | 50mA |
| 3.3V Spannungswandler | 5V | 365mA |
| **Gesamt** |  | **465mA** |

Alle Bauteile, die mit 5V versorgt werden, benötigen im Worst Case 470 mA. Der Stromverbrauch des 3.3V Spannungswandlers berechnet sich dabei aus dem Worst Case Stromverbrauch der 3.3V Versorgungsspannung.  
Der Wirkungsgrad η ist dem Datenblatt bei einem Laststrom von 515mA entnommen. Damit ergibt sich ein maximaler Eingangsstrom von 365 mA für den maximalen Stromverbrauch aller 3.3V.   
Auch für die Spannungserzeugung des 5V Signals gibt es wieder unterschiedliche Bauteile, welche zur Auswahl stehen.

1. **N7805-500**

Bei diesem Bauteil handelt es sich um einen DCDC, welcher aus 12V ein 5V Signal erzeugt. Die Verwendung dieses Bauteils würde bedeuten, das zuerst aus 48V ein 12V Signal generiert werden muss, aus welchem dann wiederum ein 5V Signal erzeugt wird. Dieses würde wiederum Grundlage zur Generierung des 3.3V Signals sein. Diese Kaskadierung birgt viele Nachteile und sollte deshalb vermieden werden.

1. **P7805-500**Dieses Bauteil erzeugt die 5V direkt aus dem 48V Eingangssignal. Dadurch wird die Kaskadierung aufgebrochen. Zudem ist der Ausgangsstrom mit 500mA ausreichend für die Versorgung aller Bauteile mit 5V Spannungspegel. Die Wahl fällt daher auf dieses Bauteil.

|  |  |  |
| --- | --- | --- |
| **Bauteil** | **Versorgungsspannung** | **Stromverbrauch** |
| RFID-Reader | 12V | 120 mA |
| Status-LED | 12V | 100 mA |
| **Gesamt** |  | **220 mA** |

Alle Bauteile, die mit 5V versorgt werden, benötigen im Worst Case 470 mA.   
Für den 48V zu 12V DCDC steht nur ein Bauteil zur Verfügung, welches in den Lagerbeständen vorrätig ist. Dieses liefert einen Ausgangsstrom von Damit ist die Stromstärke deutlich ausreichend, um alle Bauteile zu versorgen, welche mit der 12V Spannungsversorgung gespeist werden. Hier kann man Zusätzlich noch erkennen, warum die Wahl für den 5V DCDC auf das Bauteil fällt, welches direkt 48V als Eingangsspannung nutzt. Würde zusätzlich noch der 12V->5V DCDC versorgt werden müssen, betrüge die mindestens benötigt Stromstärke

Das würde den Ausgangsstrom des DCDC um 80 mA überschreiten.  
Als zusätzlichen Schutz für die Schaltung ist zusätzlich noch eine Sicherung hinter dem 12V Pfad eingebaut. Diese Sicherung bleibt leitend bis 750 mA und löst bei 1.5A garantiert aus. Diese Sicherung ist dafür da im Falle eines Bauteilschadens die folgenden Bauteile für dem hohen Kurzschlussstrom des DCDC zu schützen. Für den Eingang wird ebenfalls eine Sicherung vorgesehen, zur Auslegung dieser Sicherung muss jedoch der Eingangstrom bekannt sein. Dieser setzt sich zusammen aus dem Eingangsstrom der beiden DCDC-bauteile für 12V und 5V.

Die ausgewählte Sicherung bleibt leitend bis zu einem Strom von 300mA. Garantiert auslösen wird sie lt. Datenblatt ab einem Strom von 650 mA. Die Sicherung eignet sich damit sehr gut, um als Schutz für die Schaltung bei Kurzschlüssen zu fungieren.

Ein Bild, das Text, Diagramm, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Zusätzlich wird als Schutzbauteil noch eine Schottky Diode hinzugefügt. Diese hat die Aufgabe die Schaltung bei Verpolung zu schützen. Wird die Schottky Diode in Sperrrichtung betrieben, wird sie zerstört, wodurch keine Spannung mehr anliegt, welche die restliche Schaltung beschädigen kann. Die Kondensatoren, welche an den Ein und Ausgängen der Spannungswandler platziert sind, dienen der Glättung der Spannungspegel und sollen Spannungsspitzen dämpfen, um Schäden am Bauteil zu verhindern. Die Werte der Kondensatoren sind durch das Datenblatt festgelegt. Zur einfacheren Fehlersuche wird dem Ausgang noch eine LED hinzugefügt. Diese soll leuchten, wenn der Spannungswandler einen Spannungspegel an seinem Ausgang bereitstellt. Der Vorwiderstand berechnet sich dabei entsprechend dieser Formel:

Gemäß der verwendeten e-12 Widerstandsreihe ist der nächstmögliche Wert 1.8kΩ.   
Die Schaltung für den Spannungswandler von 48V zu 5V verhält sich genau wie die vorherige Schaltung, mit der Ausnahme, dass sich die Werte des einen Kondensators ändern, welche im Datenblatt spezifiziert werden. Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.  
Auch die Parameter der Sicherung bleiben die selben, nur das dieses bauteil für den Einsatz bis 6V zugelassen ist, wärend die vorherige Sicherung im Spannungsausgang bis 13.6V eingesetzt werden durfte. Die Ströme, bei welchen die Sicherung auslöst, bleiben identisch. Zusätzlich ändert sich der Wert des Vorwiderstands der LED.

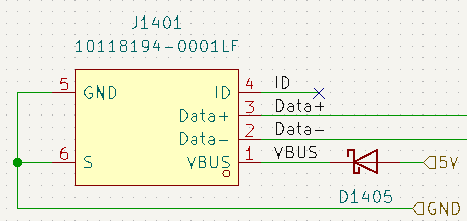
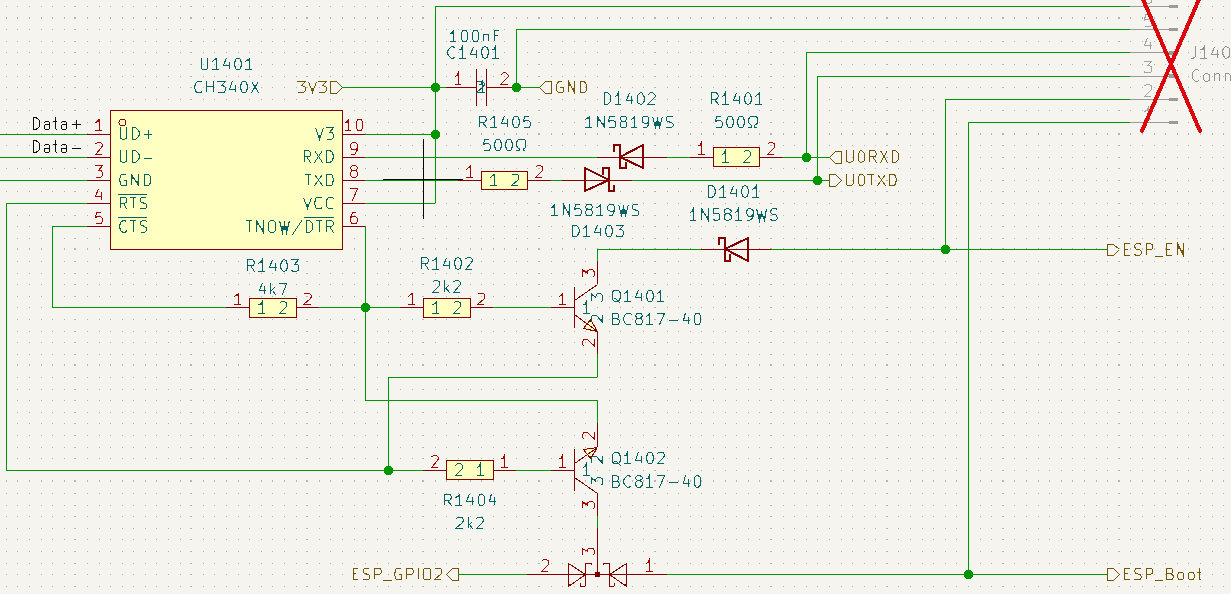
Als letzte Schaltung muss nun noch der Spannungsregler für das 3.3V Signal betrachtet werden. Dieser erzeugt aus dem 5V Eingangssignal ein 3.3V Ausgangssignal. Dieser Spannungsregler eignet sich für mehrere Ausgangsspannungen im Bereich von 1.2V bis 3.3V. Deshalb muss der Chip korrekt beschalten werden, um den korrekten Output zu erzeugen. Die Pins IN und En sind dabei die, welche das Bauteil versorgen. In ist der Spannungseingang, in unserem Fall, dass 5V Signal. Dieses wird auch mit dem EN-Pin verbunden. Dieser aktiviert das Bauteil. Die Festlegung der Ausgangsspannung wird über die Pins FB und LX erreicht. Über FB wird eine Spannung ausgegeben, wobei über den Spannungsteiler die gewünschte Ausgangsspannung erzeugt wird. Über eine Spule wird dieser Spannungswert wieder zum Bauteil zurückgeführt, welches nun auf diese Spannung programmiert wird. Ein Bild, das Text, Diagramm, Screenshot, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.Die Ausgangsspannung lässt sich dabei mit der Formel

berechnen. Damit gilt für :

Mit den Werten und ergibt sich eine Abweichung von 0.7% für die Spannung.  
Sowohl die Bauteilwerte der Spule als auch die der Kondensatoren sind wiederum vom Datenblatt vorgegeben. Auch hier soll wieder eine LED vorgesehen werden, welche einen Spannungspegel am Ausgang visuell anzeigt. Für den Vorwiderstand lässt sich folgender Wert berechnen:

USB-Schaltung

Um diese ganzen Funktionalitäten nutzen zu können bedarf es verschiedener Schnittstellen. Eine dieser Schnittstellen eine Micro-USB Buchse, über welche es möglich sein soll, den Chip mit neuer Software zu bespielen. Da der ESp32 mit USB nicht direkt arbeiten kann, wird noch ein Wandler benötigt, welcher die USB-Kommunikation in eine UART-Kommunikation umwandelt, über welche der Mikrocontroller kommunizieren kann. Die USB-Buchse besitzt dabei 6 Pins. Diese Pins übertragen alle Signale der Physischen USB-Kommunikation. Die Pins GND und S liegen dabei auf Groundpotential. Bei Pin S handelt es sich um den Schirm das Kabels, welches die Signale vor Störungen schützen soll. Hier ist es wichtig, den Schirm auf dasselbe Potenzial wie den Systemground zu legen, da sonst Störungen unter Umständen sogar verstärkt werden können. Über VBUS lässt die Busspannung abgreifen. Diese beträgt bei USB 5V, da der Host immer die Spannungsversorgung für alle Devices übernimmt. Die Spannung wird hier abgegriffen und auf den 5V Versorgungsspannungspfad geführt, um über USB die Platine bestromen zu können. Normale USB-Ports eines PCs können bis zu maximal 500mA an Strom liefern, das ist vollkommen ausreichend, um die CAN-Transceiver und den Spannungswandler auf 3.3V zu betreiben, da der Spannungswandler von 48V auf 5V auch nicht mehr als 500mA liefern kann. So muss die Platine für Testzwecke nicht immer an eine Externe Spannungsquelle angeschlossen sein, welche die 48V Eingangsspannung liefern kann, sondern kann auch mit einem normalen USB-Netzteil betrieben werden oder beim Flashen direkt vom PC aus. Zwar fallen dabei die 12V Funktionalitäten aus, da diese nicht mit Spannung versorgt werden können, dazu gehören jedoch nur der RFID-Reader, die LED und die VCU-Signale, welche für einen grundlegenden Systemtest vernachlässigbar sind, da es auf der Platine dafür optische Indikation über LEDs gibt, ob die Schaltungen funktionieren. Die Schaltung wird dabei von einer TVS-Diode geschützt. Diese Verhindert Spannungsspitzen, welche zum Beispiel ESD-bedingt auftreten können und schützt so die gesamte Schaltung vor Zerstörung. Die tatsächliche Datenübertragung der Bits findet über die Pins Data+ und Data- statt. Die Bits werden dabei über Zwei Signalpegel übertragen, wobei D+ und D- immer 0V oder 3.3V führen. Für den Ruhezustand liegt D+ auf 3.3V, D- auf 0V. Wird nun eine logische 1 übertragen, bleiben die Pegel unverändert bei einer logischen 0 tauschen die beiden Leitungen ihren Zustand. Dieser Wechsel findet bei jeder übertragenen logischen 0 statt. Die Auswertung der Pegelveränderungen übernimmt dabei der USB to Serial Wandler. Im vorli9egenden System übernimmt das der CH340X. Der USB-Wandler hat dabei 2 Pins für Spannungsversorgung. Über VCC wird der Chip mit Spannung versorgt und über V3 wird die Spannung definiert, mit welcher der Chip versorgt wird. Im Vorliegenden System soll der Chip mit 3.3V versorgt werden, da auch alle restlichen Komponenten auf 3.3V arbeiten. Damit das Bauteil korrekt arbeitet muss dafür der Pin V3 ebenfalls auf 3.3V gezogen werden. Würde der Chip mit 5V versorgt, würde der Pin V3 über einen Abblockkondensator gegen Ground verbunden werden. Die beiden Pins DU+ und DU- sind die beiden Pins, welche mit der USB-Buchse verbunden sind. Über diese erhält der Wandler die Bits, welche über die USB-Leitung übermittelt werden und anschließend umgewandelt werden, um über eine UART-Schnittstelle an den ESP32 gesendet zu werden. Die Übermittlung an den ESP32 geschieht über die Pins RXD und TXD. Über diese beiden Pins findet die asynchrone UART-Kommunikation statt. Die beiden Widerstände sind dabei zum Schutz der Bauteile vorgesehen. So sollen diese im Falle eines Kurzschlusses den Strom begrenzen und so den ESP32 und den USB-Wandler schützen. Gleichzeitig dienen die Widerstände zur Dämpfung von Signalflanken, wodurch Störungen vermindert werden soll. Das ist vor allem bei langen Leitungen relevant. Bei den beiden Dioden handelt es sich um TVS-Dioden. Diese Schützt sowohl den Chip als auch den Mikrocontroller vor Überspannung, indem die Diode bei Überspannung durchbricht und die Spannungen auf einen Maximalwert begrenzt. Die TVS-Diode hat dabei gegenüber einer Zenerdiode den Vorteil, dass sie deutlich schneller arbeitet und damit kürzere und höhere Spannungsspitzen abfangen kann. Diese treten vor allem durch ESD-Probleme auf, welche über das Kabel auftreten können. Damit die Software, welche über die USB-Schnittstelle übermittelt wird, auch als neue Software für den Mikrocontroller genutzt wird, muss dieser in den Entsprechenden Bootmodus gebracht werden. Dafür müssen GPIO 0 und 2 auf den korrekten Pegel gezogen werden. Die dafür vorgegebenen Pegel sind Beide Pins gegen Ground zu ziehen. Das wird über die Pins RTS und DTR erreicht. Standartmäßig erfüllt der Pin 6 jedoch nicht die Funktionalität von DTR, sondern TNOW. Dieses Signal ist ein Indikator für eine aktuell ausgeführte Übertragung von Daten über die USB-Schnittstelle. Über einen externen Widerstand lässt sich die Funktion des Pins ändern, sodass er nun die Funktion eines DTR-Pins erfüllt. Dieser Pin zeigt den Zustand „Data Terminal Ready“ an und ist lowaktiv. Dafür muss der TNOW-Pin mittels eines 4.7kΩ zu CTS verbunden werden. CTS ist dabei das Signal für “Clear To Send”. Das weitere Signal, welches zur korrekten Aktivierung des Boot-Modus benötigt wird, ist das Signal RTS. Dieses Signal ist ein Ausgang, welcher als Lowaktives Signal ein „Request To Send“ anzeigt. Um einen korrekten Bootvorgang zu gewährleisten sind verschiedene Phasen nötig, damit die Software auch als neues Programm auf den Mikrocontroller aufgespielt wird. Zuerst muss der Enable-Pin am ESP32 auf Ground gezogen werden, um den Reset einzuleiten. Beim darauffolgenden Start-Up müssen die GPIO-Pins 0 und 2 auf den korrekten Pegeln liegen, damit der korrekte Bootmodus ausgeführt wird. Dieser Ablauf wird über die Schaltung mit den beiden Transistoren automatisch erzeugt, ohne dass ein Button auf der Platine benötigt wird, mit welchem händisch ein Bootmodus aktiviert werden muss, wie bei einigen Entwicklungsboards der Fall ist. Die Transistorschaltung ist dabei dem ESP Devkit V4 entnommen, welches mit demselben Chip arbeitet. Durch diese Schaltung werden abhängig DTR und RTS unterschiedliche Pegel an den Pins GPIO 0 und EN erzeugt:

|  |  |  |  |
| --- | --- | --- | --- |
| **DTR** | **RTS** | **EN** | **GPIO 0 / GPIO 2** |
| 1 | 1 | 1 | 1 / 0 |
| 0 | 0 | 1 | 1 / 0 |
| 1 | 0 | 0 | 1 / 0 |
| 0 | 1 | 1 | 0 / 0 |

Die beiden Transistoren schalten jeweils, wenn die Spannung zwischen Basis und Emitter ist. Das ist der Fall, wenn DTR und RTS unterschiedliche Signalpegel aufweisen. Sind die beiden Pegel an DTR und RTS identisch, dann schalten die Transistoren nicht. In diesem Fall haben die Pins GPIO 0 / 2 und EN die Pegel, welche über die Widerstände am Mikrocontroller definiert sind. Das ist für GPIO 0 High wie für Enable, GPIO 2 wird über einen Pull-Down Widerstand standartmäßig auf Ground gezogen. Wenn DTR auf High liegt und RTS auf Low schaltet der Transistor Q1401. Er zieht damit den Enable-Pin am Mikrocontroller auf den Pegel von RTS, also gegen Ground und löst damit den Reset des Microcontrollers aus. GPIO 0 liegt weiterhin auf High durch den Pull-Up Widerstand am Mikrocontroller. Sind die Pegel vertauscht, liegt also DTR auf LOW und RTS auf High schaltet der Transistor Q1402. Dadurch wird GPIO 0 auf den Pegel von DTR gezogen, welcher zu diesem Zeitpunkt auf Ground liegt. Der Enable-Pin des Mikrocontrollers bleibt dabei auf High durch den Pull-Up Widerstand am Mikrocontroller. Das ist damit der Bootmodus, indem der ESP32 mit den per USB übermittelten Daten startet. Beim Empfang von Daten via USB erzeugt der USB To Serial-Wandler somit automatisch einen Reset am Controller, aktiviert daraufhin den Bootmodus und kehrt anschließend wieder in den normalen Boot- und Betriebsmodus zurück. Auch hier wurden zum Schutz der Bauteile TVS Dioden vor den Pins des Mikrocontrollers vorgesehen, um Zerstörung durch ESD bedingte Spannungsspitzen zu verhindern. Zusätzlich wurde noch ein Programming Pinheader vorgesehen. Über diesen können sowohl Daten übertragen als auch die Pins in den korrekten Bootmodus versetzt werden. Dieser hat den Vorteil, dass im Falle einer defekten oder Fehlerhaften Schaltung trotzdem noch ein Flashen über die Verwendung eines Devkits oder eines ähnlichen Hilfsmittels möglich ist.

Ein Bild, das Text, Diagramm, Zahl, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.Stecker  
Um alle Funktionalitäten des Systems und alle Peripheriegeräte nutzen zu können, benötigt die Platine einige Stecker. Über einen Stecker wird das Display an die Erweiterungsplatine angeschlossen. Das Display benötigt dabei 13 verschiedene Signale. Ein Großteil dieser Signale fallen auf Signalpins für die I²C und die SPI-Kommunikation. Zusätzlich müssen noch Spannungsversorgung und Ground, sowie das PWM-Signal für die Hintergrundbeleuchtung des Displays vorgesehen werden. Sowohl der Stecker als Bauteil als auch die Pinbelegung werden dabei vom Display selbst vorgegeben. Für die Verbindung zwischen Display und Platine soll ein fertiges Flachbandkabel verwendet werden, sodass die Reihenfolge der Pinbelegung am Platinenstecker der des Steckers am Display entsprechen muss.

Ein Bild, das Text, Schrift, Screenshot, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.Für Ein-und Ausgangssignale werden Zwei Stecker vorgesehen. Ein Stecker verbindet den Kabelbaum mit der Platine, am anderen Stecker werden die beiden Zukaufteile RFID-Reader und Status-LED angeschlossen. Diese trennung bringt den Vorteil, dass der gesmate Kabelbaum mit dem passenden Stecker vorbereitet werden kann, ohne nachträglich beim einbau noch weitere Pins hinzufügen zu müssen. Bei dem Stecker für die beiden letzten Bauteile handelt es sich nicht um eine zweiteilige Steckverbindung mit Buchse und Stecker, sondern die Litzen der Kabel werden über eine Quetschverbindung direkt im Stecker befestigt, welcher fest auf der Platine sitzt. So können einzelne externe Bauteile ohne großen Aufwand entfernt und weggelassen werden, sollte der Kunde diese Erweiterung nicht haben wollen. Der RFID-Reader benötigt dabei 4Pins. Bei diesen handelt es sich um die 12V Spannungsversorgung und Ground, sowie die beiden CAN-Leitungen. Für die Status-LED werden die beiden Signal des LED-Treibers nach außen geführt. Für Testzwecke wird zusätzlich noch ein Pinheader vorgesehen, über welchen ein RFID-Reader angeschlossen werden kann, welcher per SPI kommuniziert. Ziel ist es zukünftig den teueren RFID-Reader, welcher aktuell im Einsatz ist, durch eine kleinere und günstigere Variante zu ersetzen. Hierfür wird jedoch kein eigener Stecker vorgesehn, sondern lediglich ein Pinheader, da diese Schnittstelle für den Einsatz im Kart keine Rolle spielt und auf den Platinen weggelassen werden wird.

Ein Bild, das Text, Schrift, Screenshot, Zahl enthält.

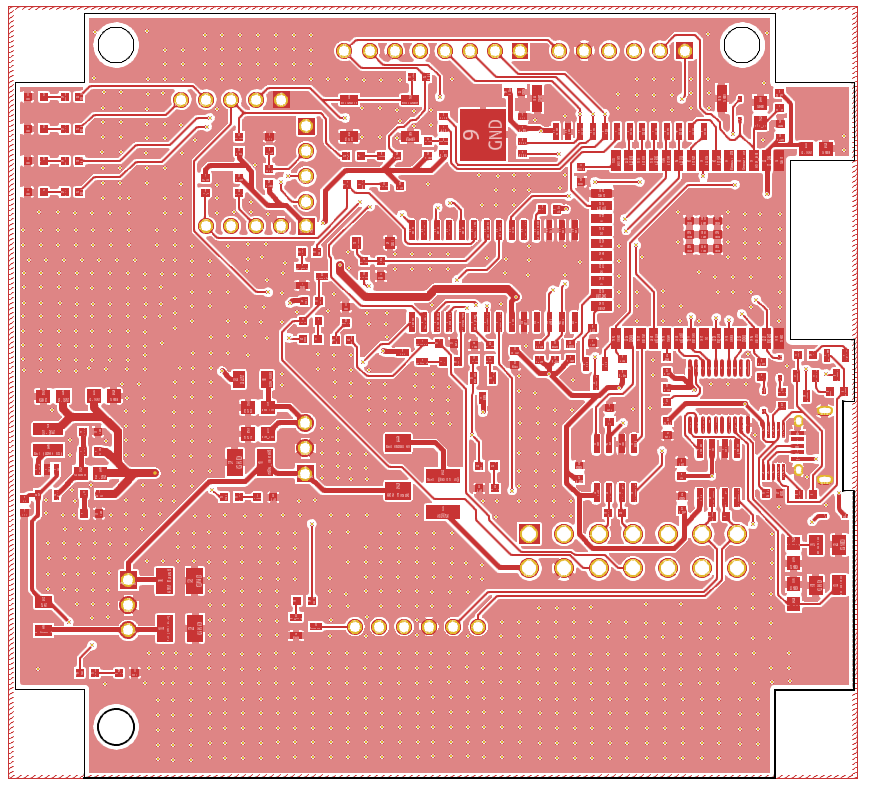
KI-generierte Inhalte können fehlerhaft sein.Der letzte Stecker bietet die Schnittstelle zum Kabelbaum und damit zu allen anderen Komponenten des Karts. Über diesen Stecker findet sowohl die Spannungsversorgung von der Batterie für die Erweiterungsplatine statt als auch die Kommunikation mit dem Steuergerät, sowohl über die beiden CAN-Busse als auch über die Signalleitungen. Die Pinbelegung ist dabei hauptsächlich auf möglichst einfaches Layout ausgelegt. Vor allem die Leiterbahnen für die CAN-Busse sollen so kurz wie möglich sein, weswegen die Pins dafür entsprechend belegt wurden.

## PCB-Layout

Nachdem im vorangegangenen Kapitel der Schaltplan des entwickelten Systems detailliert beschrieben wurde, folgt im nächsten Schritt die Übertragung des Schaltungsentwurfs in ein Leiterplattendesign. Dieser Prozess umfasst nicht nur die reine Umsetzung elektrischer Verbindungen, sondern erfordert auch die Berücksichtigung mechanischer und Elektrischer Vorgaben. Dabei spielen insbesondere die äußeren Abmessungen der Leiterplatte sowie die Positionierung von Befestigungspunkten und Schnittstellen eine wesentliche Rolle, um eine zuverlässige Integration in das Gesamtsystem zu gewährleisten. Aufbauend darauf ist die Platzierung der elektronischen Bauteile entscheidend für die Signalqualität, die thermische Stabilität und die spätere Bestückbarkeit der Platine. Ebenso ist der gewählte Layer-Aufbau maßgeblich für die Leitungsführung, die Abschirmung empfindlicher Signale und die Sicherstellung einer stabilen Versorgungsspannung. Schließlich sind auch hochfrequente Signale gesondert zu betrachten, da deren Führung besondere Anforderungen an Impedanzkontrolle, Längenanpassung und EMV-gerechtes Design stellt. Im Folgenden werden diese Aspekte des PCB-Designs systematisch analysiert und die jeweiligen Designentscheidungen begründet.

Ein Bild, das Text, Diagramm, Zahl, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.Die Mechanischen Vorgaben werden vollständig vom bereits vorhandenen Gehäuse definiert. Durch die Geometrie des Gehäuses werden die Abmaße sowie die Form der Platine bestimmt. Die Abmaße des Gehäuses betragen dabei 85mm x 98mm, wobei in den Ecken des Gehäuses jeweils Aussparungen von 7mm x 7mm vorgesehen werden müssen, da hier Einschmelzgewinde zur Befestigung des Gehäusedeckels eingebracht werden. Im Bild sind diese Bereiche blau markiert. Der vordere Teil des Gehäuses ist zudem für die Kabel vorgesehen, welche seitlich in das Gehäuse eingeführt werden. Somit muss der Bereich zwischen den beiden unteren Anschraubpunkten freigehalten werden. Als Auflagefläche für die Platine dienen dabei die Rot markierten Auflageflächen, welche zusätzlich auch die Anschraubpunkte zur Befestigung der Platine bieten. Für die Platine ergibt sich so eine Größe von 84.50mm x 76.50mm. Differenzen von 0.5mm sind dabei als Toleranz vorgesehen, sodass die Platine ohne Spannungen im Gehäuse sitzt. Von den gegebenen Abmessungen müssen jedoch in jeder Ecke noch die Ausschnitte für die Anschraubpunkte des Deckels entfernt werden.   
Neben den Abmaßen und der Form der Platine werden auch die Anschraubpunkte bereits durch das bestehende Gehäuse definiert. Diese sind mittig in die Auflageflächen eingebracht. Dabei sind Schrauben der Größe M3 zur Befestigung vorgesehen. Zur Durchführung werden daher Löcher mit mindestens 3.6mm Durchmesser benötigt. Wichtig ist aber nicht nur die Löcher für die Schraube vorzusehen, sondern auch den Platz für den Kopf der Schraube, welche nicht mit den Bauteilen kollidieren darf. Ein Bild, das Text, Screenshot, Display, Diagramm enthält.

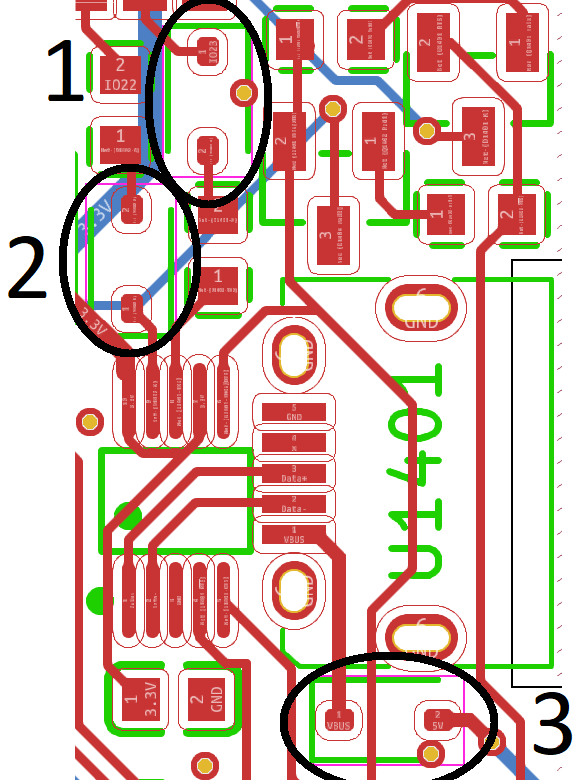
KI-generierte Inhalte können fehlerhaft sein.Dessen Größe beträgt bei einer Zylinderschraube einen Durchmesser von 5.5mm. Dieser Platz ist auf der Platine als Sperrfläche vorgesehen (grün), um versehentliches Platzieren von Bauteilen zu verhindern. Auch Leiterbahnen sollten in diesem Bereich nicht verlaufen, um Störungen zu verhindern. Ebenfalls als Sperrflächen vorgesehen sind die gesamten Auflageflächen auf der Unterseite der Platine (blau), um dort zu verhindern, dass Bauteile mit Durchsteckkontakten vorgesehen werden, sodass die Platine auf den Kontakten aufliegen würde.   
Zu den mechanischen Anforderungen einer Platine zählt auch die Wärmeableitung. Dafür wird bei dieser Platine eine durchgehende Massefläche vorgesehen. Dabei werden zwei große Kupferflächen auf Top- und Bottomlayer hinzugefügt, welche nur die Bereiche von anderen Signalpfaden aussparen. Durch die großen Kupferflächen kann Wärme von heißen Bauteilen sehr gut abgeführt werden und über die gesamte Platine verteilt werden. Dadurch werden Thermal Hotspots vermieden, wodurch Platine und Bauteile geschützt werden. Auch die Verwendung und Anbindung an Kühlkörper wird so vereinfacht, auch wenn das für dieses Projekt eine eher untergeordnete Rolle spielt, da hier weder aktive Kühlelemente wie Lüfter vorgesehen sind, noch passive Kühlung über Kühlkörper oder Ableitung an das Gehäuse einen großen Effekt erzielen würden durch die Verwendung eines Kunststoffgehäuses mit schlechter Wärmeleitfähigkeit. Die Verwendung einer großen Massefläche bietet allerdings nicht nur mechanische, sondern vor allem viele Elektrische Vorteile. So birgt eine Verwendung von großen Masseflächen den Vorteil, dass diese Massefläche eine sehr niedrige Impedanz im Vergleich zu einzelnen Leiterbahnen bietet und damit wenig parasitäre Induktivitäten besitzt. Der Strom hat hier für jedes Signal die Möglichkeit, sich seinen Rückstrompfad selbst zu suchen, wodurch dieser sehr viel effektiver wird. Auch so werden hohe Temperaturen durch hohen Stromfluss an einzelnen Stellen der Platine vermieden. Zudem bieten die beiden Masseflächen auf Bottom und Top Layer einen guten Schutz gegen Elektromagnetische Störungen. Zum einen wirken sie wie ein Schirm schützend für hochfrequente Signalleitungen durch Einflüsse von außen, zum anderen wird durch die breite Massefläche die Erzeugung von EMV-Störungen durch die Rückstrompfad über eventuelle Groundleitungen stark verringert. Diese Störungen können vor allem auch dann auftreten, wenn die Leiterbahnen für den Rückstrom nicht sternförmig geführt werden. Diese Fehlerquelle wird durch die Verwendung von Masseflächen über die gesamte Platine eliminiert.   
Um ähnliche Effekte zu erzielen, wird zusätzlich Via Stitching zwischen den beiden Masseflächen eingesetzt. Dabei handelt es sich um Durchkontaktierungen zwischen oberer und unterer Kupferschicht der Platine, welche engmaschig über die gesamte Platine verteilt werden, um die beiden Masseflächen auf Top und Bottom-Layer thermisch, mechanisch und elektrisch miteinander zu verbinden. Die Vias schaffen dabei einen entstörenden Bereich um hochfrequente Signale, wie zum Beispiel CAN, SPI oder I²C, und verhindern so, dass Störungen von der Platine ausgesendet werden oder Signale auf der Platine gestört werden. Durch die Durchkontaktierung zwischen beiden Massefläche wird zudem eine niederimpedante Verbindung zwischen allen Masseflächen geschaffen und die Bildung von Isolierten Kupferinseln, welche von Leiterbahnen umschlossen sind, verhindert. Der Rückstrompfad ist somit für alle Signale bestmöglich kurz, wodurch Störungen und Reflexionen vermieden werden. Zusätzlich werden die Stabilität und die Signalqualität verbessert durch den Einsatz von Vias, da hierüber Stromschleifen und somit Störspannungen vermieden werden, welche durch Schlechtes Layout von Leiterbahnen bei Verwendung eines Sternpunkt entstehen können. Das kommt vor allem differenziellen Signalen, wie sie auf dieser Platine mit CAN und UART auftreten, zugute. Durch die Vias wird neben den elektrischen Vorteilen auch thermische Eigenschaften verbessert. Durch die Vias ist es möglich, die Wärme über die gesamte Fläche der Platinen auf beide Seiten zu verteilen, ohne dabei vereinzelte Wärme Hotspots zu schaffen. Dadurch wird der Schutz hitzekritischer Bauteile nochmals verbessert und die Langlebigkeit der Platine sichergestellt. Das ist für dieses Projekt nicht unrelevant, da die Platine oft in wärmeren Umgebungen während Rennevents eingesetzt wird und die großen Spannungswandler auf der Platine viel Verlustleitung in Form von Wärme erzeugen. Zusätzlich verbessern die Vias die mechanische Stabilität der Platine. Durch die Vias wird eine stabile Verbindung zwischen den Kupferschichten geschaffen, welche sie durch das Dielektrikum hindurch verbindet und somit Delamination verhindern kann.  
Nach dem Aufbau der Platine soll nun das tatsächliche Layout der Schaltung betrachtet werden. Für die Platzierung der Stecker spielen auch hier wieder mechanische Gegebenheiten eine Rolle. Die Platzierung der Stecker ist dabei so gewählt die Biegeradien der Kabel möglichst gering zu halten. Die Durchführung für die Kabelbaumseite befindet sich dabei auf Ein Bild, das Text, Diagramm, Screenshot, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.der linken Seite des Gehäuses, weshalb die Platzierung des Steckers möglichst weit rechts auf der Platine platziert wird. Dadurch ist genug platz vorhanden, um Entlastung für die Kabel und Biegeradien zu schaffen. Aus diesem Grund ist der Stecker auch auf die Platine eingerückt un nicht direkt an deren kante platziert. Zum einen ergeben sich so Flächen zur darstellung der Pinbelegung auf der Platine. Zum anderen muss zusätzlich der Platz bedacht werden, welchen der kabelbaumseitige Stecker der Steckverbindung benötigt. Mit dieser Platzierung schließt die gesamte Steckverbindung mit der kante der Platine ab. Dasselbe gilt für den Stecker, welcher als Verbindung für RFID-Reader und LED dient. Auch hier sollen Biegeradien möglichst gering gehalten werden. Dadurch, dass die Kabel in dem Stecker in einem 45° Winkel herausgeführt werden, ergeben sich auch keine Probleme durch die Überkreuzung von Eingehenden und Ausgehenden Kabeln, da die Leitungen problemlos übereinander geführt werden können. Die Platzierung des Mikrocontrollers ergibt sich durch die Platzierung des Kabelbaumsteckers und den Anforderungen, welche das Bauteil stellt. Durch die SMD Wlan-Antenne, welche Teil des Chips ist, muss der Mikrocontroller an den Rand der Platine platziert werden. Das ergibt sich dadurch, dass sich unter der Wlan-Antenne keine Kupferschichten befinden dürfen, da diese ansonsten die Signale stören und eine Verbindung unmöglich machen können. Die einfachste Methode, dies sicherzustellen, ist, am Rand der Platine einen Ausschnitt hinzuzufügen und den ESP32 darüber zu platzieren. Wichtig ist hierbei zu beachten, dass die Antenne nicht über die ursprüngliche Platinenkante hinüberragt, um Beschädigungen der Antenne zu vermeiden. Zudem wurde hier darauf geachtet den Mikrocontroller so zu platzieren, dass das Layout der eingehenden CAN-Leitungen möglichst kurz und einfach gehalten werden kann. Deshalb wurde sich hier dafür entschieden, den Chip an die Rechte Seite der Platine zu platzieren.   
Für das Layout von CAN ist Störsicherheit das wichtigste Kriterium, welches es zu beachten gilt. Zu diesem Zweck wurde darauf geachtet, dass die Signale möglichst optimiert geroutet sind. Dazu gehört, dass die Leitungen als Differenzielles Leitungspaar geführt sind. Das bedeutet, dass die Leitungen eng beieinander geführt werden und gleich lang sind. Ziel ist es, dass etwaige Störungen in gleichem Maße auf beide Signalleitungen einwirken, wodurch die Störung durch die Auswertung des Differenziellen CAN-Signals keinen Einfluss auf die Signalqualität hat. Die Reihenfolge der Bauteile, die Ausrichtung des Mikrocontrollers und die Pinbelegung des Steckers sind darauf ausgelegt, die Leitungen so einfach und Störungssicher wie möglich Routen zu können. Zudem sollte auf Knicke und Abzweigungen vermieden werden. Im aktuellen System betragen die Unterschiede der leitungslängenzwischen Stecker und Transceiver 0.05mm und 0.02mm. Auch wurde darauf geachtet die Abschlusswiderstände möglichst nah am Stecker zu platzieren. Auch das Trägt dazu bei Reflexionen und Störungen zu verringern, da es durch die kurzen Abstände Ein Bild, das Text, Schrift, Diagramm, Plan enthält.

KI-generierte Inhalte können fehlerhaft sein.kaum Bereiche gibt, an denen eine Störung in die Schaltung eingreifen kann. Weitere Maßnahmen, um die Störungen zu verringern, sind undurchbrochene Masseflächen unter den Differenziellen Signalen und deren Bauteile. So wurde bei CAN bewusst darauf geachtet, keine anderen Signale im selben Bereich zu routen. Vor allem Kreuzende Signalleitungen würden ein sehr hohes Störungspotenzial bietet, was unter allen Umständen vermieden werden soll. Neben anderen Signalen wurde im unmittelbaren Bereich der CAN-Schaltungen auch auf Masse-Vias verzichtet, um hier keine Störungen durch andere Signale in den CAN-Aufbau einzubringen. Auch die CAN-Signale selbst wurden durchgehend auf einem Layer geroutet, um die Leitungsimpedanzen so gering wie möglich zu halten. Des Weiteren wird auch darauf geachtet, alle Kondensatoren, welche hier als Abblockkondensatoren wirken sollen, so nah wie möglich an den jeweiligen Pins zu platzieren. Bei Can2 muss zusätzlich noch der CAN-Controller platziert werden. Auch hier wurde darauf geachtet, die Leitungsbahnen für die SPI-Kommunikation so kurz wie möglich zu halten und keine Leitungen zu kreuzen.

Viele der eben genannten Maßnahmen lassen sich dabei auf die gesamte Platine anwenden. So sollten alle Hochfrequenten, zusammengehörigen Leitungspaare, welche meist für Kommunikation genutzt werden, möglichst nahe beieinander geführt werden, um Störungen durch die Leitungen zu minimieren. Zudem wird durch eng beieinander liegende Leitungen eine in etwa gleiche Leitungslänge erreicht, welche vor allem bei Synchronen Kommunikationsprotokollen relevant ist. Außerdem bleibt es so leichter andere Signale zu routen, da bei hochfrequenten Signalen darauf geachtet werden sollte möglichst keine kreuzenden Leitungen über oder unter den Signalen entlangzuführen. Auch sollten die leitungslängen so kurz wie möglich gehalten werden. Leider hat das beim vorliegenden System nicht immer funktioniert und die Signalleitungen für SPI und I²C kreuzen sich untereinander und mit anderen Signalleitungen. In solchen Fällen sollte darauf geachtet werden, den Winkel zwischen den kreuzenden Leitungsbahnen möglichst nah an 90° zu bringen. Damit ist der Bereich, in dem die Leitungen sich überschneiden, minimal und ein Übersprechen der Signale am unwahrscheinlichsten.  
Auch für alle Kondensatoren, welche als Filterschaltung oder zur Verbesserung des Signalqualität dienen gelten für alle Schaltungen dieselben Regeln, wie sie bereits bei CAN genannt wurden. Es ist wichtig die Leiterbahnen möglichst kurz zu halten und die Kondensatoren möglichst nah an dem Bauteil zu platzieren, welches geschützt werden soll oder dessen Spannung gefiltert werden soll. Das schützt davor, dass das bereits gefilterte Signal erneut gestört wird über die Leiterbahn zwischen Kondensator und Bauteil. Am wichtigsten ist die Platzierung der Abblockkondensatoren als Filter hochfrequenter Spannungsspitzen direkt Ein Bild, das Diagramm, Farbigkeit, Reihe, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.an den Ausgänger der Spannungswandler. Alle hier erzeugten Störungen breiten sich über die Versorgungsspannung über die gesamte Platine aus.   
genauso wichtig ist es sich mit der korrekten Platzierung von Schutzbauteilen zu Beschäftigen. Dabei handelt es sich zum einen um die Sicherungen nach den Spannungswandlern auf 12V und 5V, aber auch um die Schutzdioden, welche vor allem bei der USB-Schaltung zum Einsatz kommen. Die Sicherrungen sollen die Schaltungen vor Zerstörung durch hohe Ströme, zum Beispiel bei einem Kurzschluss in der Schaltung, schützen. Die Gefahr besteht dabei darin, das durch zu hohen Eingangstrom Bauteile zerstört werden oder Leiterbahnen sich enorm erhitzen. Es ist daher besonders darauf zu achten, dass Sicherungen besonders nah der Quelle der Spannung platziert wird, um als erstes auszulösen, bevor sich bereits andere Bauteile oder Leiterbahnen erhitzen konnten. In der Vorliegenden Schaltung bedeutet das die Sicherungen Ein Bild, das Diagramm, Reihe, Screenshot, Plan enthält.

KI-generierte Inhalte können fehlerhaft sein.möglichst nah an den Ausgangspins der Spannungswandler zu platzieren, oder im Falle der Eingangssicherung für die 48V Versorgungsspannung, möglichst nah am Eingangsstecker. Außerdem müssen die Thermischen Eigenschaften der Sicherung beachtet werden, da sich diese Im Betrieb stark erhitzen können. Auf der Platine sind deshalb rund um die Sicherungen große Abstände und unter den Sicherungen durchgehende Kupferflächen vorgesehen, um die Wärme besser ableiten zu können und keine hitzeempfindlichen Bauteile zu Beschädigen. All das kann man sehr gut anhand der Eingangssicherung betrachten.   
Ähnliches gilt für die Schutzdioden. Die Dioden sollen im Falle eines Spannungsimpulses, zum Beispiel bedingt durch ESD-Impulse, die Überspannung ableiten und so die Schaltung schützen. Dafür ist es wichtig, dass die Dioden so nah wie möglich an der Quelle der Störung, also meist einem Stecker, über welchen die Störung von außen eingebracht wird, sitzen. Die Diode muss dabei sehr schnell reagieren, das kann sie nur, wenn der Pfad zu ihr so kurz wie möglich ist. Außerdem wird so die Leitung verkürzt, von welcher eine Störung in die restliche Schaltung ausstrahlen kann. An den beiden Schutzdioden 1 und 2 kann man die beiden genannten Prinzipien sehr gut verdeutlichen. Diode 2 soll den Spannungsspitzen am Ausgang des USB-Wandlers abfangen und ist deshalb möglichst nah am IC platziert. Diode 1 ist in den Signalpfad eingebracht mit Signalen, die vom Mikrocontroller an den ESP geschickt werden und ist daher nahe am Ausgangspin des ESP32 platziert. Diode 3 ist die Schutzdiode, welche das 5V Signal vom USB-Host zur Versorgung der Schaltung abgreift, ist das kritischste Signal, da hier die Spannungsspitzen den größten Schaden anrichten können. Auch hier wurde darauf geachtet die Diode nah am Stecker zu positionieren, um Störungen frühzeitig zu filtern.

Ein weiterer Wichtiger Aspekt zum Schutz der Schaltung ist die korrekte Auslegung der Leiterbahnen für die Versorgungsspannung und die berechneten Ströme. Werden die Leiterbahnen zu dünn ausgelegt, ist die Erhitzung bei den fliesenden Strömen unter Umständen zu hoch und die Leiterbahnen und die Bauteile können Schaden nehmen.   
Die mindestens benötigte Leiterbahnbreite berechnet sich dabei gemäß IPC-2221 folgendermaßen:

Mit:

Daraus ergibt sich für W:

Bei den zu bestellenden Platinen beträgt die Leiterbahndicke   
Die Temperaturerhöhung wird auf festgelegt. Dieser Wert wird recht hoch gewählt, da das Kart vor allem im Hochsommer auf Außenplätzen verwendet wird, wo das Kart durchgehend der Sonne ausgesetzt ist.

Für die 12V und 5V führenden Spannungspfade wird ein Strom von festgelegt, da das der Wert ist, ab welchem die Sicherung auslöst, sodass dadurch keine höheren Ströme langfristig auf der leiterbahn fließen können. Damit ergibt sich für 5V und 12V eine Leiterbahnbreite von

Als Sicherheitsfaktor wird die Leiterbahn auf 0.2mm gewählt, um die leiterbahnen nicht am Limit zu betreiben.  
  
Für den Spannungspfad der 3.3V werden 2A als Maximalstrom angesetzt. Das ist der Wert, welchen der Spannungswandler kontinuierlich liefern kann. Damit ergibt sich folgenden Leiterbahnbreite.

Da der tatsächliche Stromverbrauch der Bauteile deutlich unterhalb der 2A liegt, wird die Leiterbahnbreite ohne große Sicherheit auf 0.51mm festgelegt. Die 2A treten nur im Falle eines Fehlers auf.

# Software

Aufbauend auf dem Schaltplan und dem Layout, welche definieren wie die Signale den Mikrocontroller erreichen, soll nun die Verarbeitung dieser Signale in der Software betrachtet werden. Dabei soll für alle für alle Komponenten zuerst um den Aufbau und den Ablauf der Software gehen, ohne die tatsächliche Software zu betrachten. Anschließend sollen diese Abläufe in Programmcode umgesetzt und näher betrachtet werden erläutert werden.

Ein Bild, das Diagramm, Text, Reihe, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.Aufgabe der Software ist es die Signale, welche dem Mikrocontroller zur Verfügung gestellt werden gemäß den Anforderungen, welche in früheren Kapiteln an das System gestellt wurden, zu verarbeiten. Dieser Ablauf soll mittels Flussdiagramme betrachtet werden. Dabei geht es darum Abläufe schematisch und übersichtlich darzustellen. Es werden große Abläufe in einzelne kleine Prozessschritte unterteilt und deren zeitlicher und logischer Ablauf grafisch dargestellt. Die Abläufe sind dabei von Funktionen und Variablen losgelöst, dass ist erst Aufgabe der Software den Ablauf, welcher hier definiert wird, tatsächlich in Programmcode umzusetzen. Diese Ablaufdiagramme sollen nun für Alle relevanten Abläufe erstellt und erklärt werden. Diese Betrachtung soll dabei für alle Komponenten und Kommunikationsabläufe betrachtet werden.   
Anschließend soll die Umsetzung dieser Abläufe in Software erklärt werden. Dabei gibt es einige Grundlegende Voraussetzungen, welchen der gesamte Code folgen soll.   
Im Vordergrund soll die schnelle Verarbeitung aller Signale stehen. Es soll daher vermieden werden, die Eingangspins von Signalen via Polling, also das zyklische Auslesen des Eingangsregisters, abzufragen. Stattdessen sollen die Eingänge einen Interrupt auslösen, sodass die Abfrage eines Signals nur durchgeführt wird, wenn tatsächlich eine Veränderung am Pin erkannt wurde. Dabei ist darauf zu achten, die Interrupt Service Routinen (ISR) so knapp wie möglich zu halten, da diese die Abarbeitung des restlichen Codes blockieren. Vor allem auf Delays, Wartefunktionen oder Kommunikation mit anderen Bauteilen muss in der ISR verzichtet werden. Stattdessen setzt jede ISR lediglich ein Flag, welches im normalen Programmcode abgearbeitet werden kann. Diese Abarbeitung geschieht in der Loop Funktion. Das ist die Funktion, welche als Dauerschleife vom Mikrocontroller ausgeführt wird. Hier werden alle Prozesse abgearbeitet, wenn das jeweilige Flag gesetzt wurde. Ist kein Flag gesetzt, findet keine Abarbeitung statt. Damit die Abarbeitung alle Funktionen zügig geht, ohne dass sich Abläufe gegenseitig blockieren, soll im gesamten Code auf Delays durch aktives Warten verzichtet werden. Es soll möglich sein, dass während einzelne Funktionen blockiert sind, andere weiterhin abgearbeitet werden können. Neben der Loop Funktion gibt es die Funktion Setup(). Diese wird einmal zu Beginn des Programmablaufs ausgeführt und soll alle Initialisierungen und Konfigurationen der Bauteile und Pins vornehmen.   
  
Zu diesen im Setup ausgeführten Funktionen gehört die Funktion init\_ports(), welche immer zu Beginn eines jeden Programmstarts abläuft. Ein Bild, das Text, Screenshot, Software, Display enthält.

KI-generierte Inhalte können fehlerhaft sein.Diese legt die Funktionalität aller Pins als Input, Output oder Alternate Funktion zur Verwendung für Busse oder Interrupts fest. Zur besseren Lesbarkeit wurden Defines als Pinnamen definiert, hinter welchen sich der tatsächliche Pin verbirgt. Die Funktion PinMode legt dabei einen Pin als klassischen GPIO-Input oder Output fest zur Verwendung als klassischer Digitaler Pin. Die Pins für SPI, I²C und UART- Kommunikation werden über die Funktionen begin() festgelegt, welche in den jeweiligen Librarys definiert sind. So bezeichnet Wire das Objekt, über welches die I²C Kommunikation gesteuert wird. Über die Funktion analogWrite() wird der Pin für das PWM-Signal für die Hintergrundbeleuchtung des Displays definiert und konfiguriert. Der übergebene Parameter regelt dabei den Duty Cycle, also das Verhältnis zwischen High und Low. Der Parameter wird dabei als 8.Bit Wert übergeben, wobei 255 einem Duty Cycle von 100% entspräche. Bei dem übergebenen Pin beträgt der Duty Cycle .   
Die Interupts werden zur Übersichtlichkeit in einer eigenen Funktion definiert, welche ebenfalls im Setup aufgerufen wird. Dabei handelt es sich um die 3 Pins, welche bereits als Input definiert wurden.   
Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.  
Diese werden mit der Funktion attachInterrupt als Interruptquelle definiert. Zusätzlich wird die Interrupt Service Routine übergeben, die Funktion, Ein Bild, das Text, Screenshot, Schrift, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.welche ausgeführt wird, sobald ein Interrupt eintritt.

Die Signale werden jedoch nicht alle direkt vom Mikrocontroller selbst eingelesen. Einige werden über die Porterweiterung erfasst und per I²C vom Mikrocontroller abgefragt. Auch diese Pins müssen dafür entsprechend konfiguriert werden, was ebenfalls per I²C Kommunikation vom ESP 32 übernommen wird. Dabei gibt das Datenblatt die Form der Kommunikation vor. Das Bauteil erwartet ein Startbit mit seiner Adresse und dem Read/Write-Bit. Das ist die Adressierung, um zu bestimmen welcher Chip angesprochen werden soll. Der Chip antwortet daraufhin mit einem Acknowledge-Bit. Wird dieses Bit nicht gesendet, bedeutet das, dass die Adressierung nicht erfolgreich war, entweder durch eine Falsche Adresse oder Fehlgeschlagene Kommunikation. In diesem Fall soll ein Fehler zurückgegeben werden und die Funktion beendet werden. War die Adressierung erfolgreich, wird als nächstes die Adresse des zu beschreibenden Registers übermittelt. Auch hierauf antwortet der Chip mit einem Acknowledge. Bleibt dieses aus, zum Beispiel wegen einer falschen Registeradresse, soll auch hier wieder ein Fehler zurückgegeben und die Ein Bild, das Text, Screenshot, Software, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.Funktion beendet werden. Im Code wird diese Funktionalität in der Funktion GPIO\_Exp\_WriteRegister() umgesetzt. Dabei werden zuerst die an die Funktion übergebenen Parameter überprüft. Dabei handelt es sich um die Registeradresse und die Daten. Die Daten dürfen dabei eine Größe eines Bytes nicht überschreiben. Die Registeradressen müssen sich dabei in einem definierten Bereich zwischen 0x10 – 0x1A oder 0x20 -0x2A befinden. Hier soll die Abarbeitung der Funktion beendet werden, bevor ein Fehler während der Kommunikation auftreten kann. Um die Kommunikation aufzubauen, wird zuerst mit der Funktion Ein Bild, das Text, Screenshot enthält.

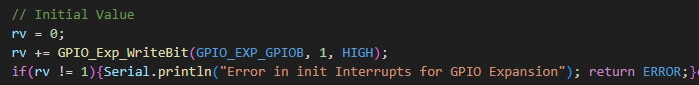
KI-generierte Inhalte können fehlerhaft sein.beginTransmission() die Adresse geschrieben. Die Funktion fügt dabei automatisch das Startbit an die Adresse hinzu. Die geschriebenen Adressen und Daten werden dabei nicht direkt übermittelt, sondern zuerst in einen Buffer geschrieben. Dasselbe gilt für die Funktion write(). Hier werden zuerst die Registeradresse und anschließend die Daten in den Buffer geschrieben. Die Funktion gibt dabei zurück, wie viele Bytes erfolgreich in den Buffer geschrieben wurden. Ein Rückgabewert von 0 bedeutet entsprechend, dass keine Daten in den Buffer geschrieben wurden. In diesem Fall wird die Funktion unter Rückgabe eines Fehlers beendet. Die Serielle Ausgabe dient dabei zum Debugging, während man per USB mit dem System verbunden ist, um so genauere Fehlerbeschreibungen zu erhalten. Mit der Funktion endTransmission() wird der Inhalt des Buffers nun gesendet. Wird ein gerät unter der angegebenen Adresse gefunden, antwortet dieses mit einem Acknowledge. Der Empfang des Acknowledge-Bit wird vom Sendenden Chip empfangen. Die erfolgreiche Kommunikation wird durch die Rückgabe der Funktion endTransmission() erkannt. Ist dieser Wert 0, ist kein Fehler aufgetreten. Neben der Funktion GPIO\_Exp\_WriteRegister() gibt es auch die Funktion GPIO\_Exp\_WriteBit(). Diese Funktion funktioniert genau wie die Funktion WriteRegister(), mit dem Unterschied, dass sie nur ein Bit im Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.adressierten Register verändert. Da es nur möglich ist das gesamte Register zu adressieren, liest die Funktion als erstes das entsprechende Register und übergibt diesen Wert mit dem einen modifizierten Bit an die Funktion WriteRegister(), welche das Register neu beschreibt. Dafür wird zusätzlich zu den Schreibfunktionen für die Porterweiterung auch eine Lesefunktion benötigt, welche die adressierten Register ausliest und zurückgibt. Anders als bei der Schreibfunktion wird hier jedoch schon nach der Registeradresse gesendet und auf ein Acknowledge-Bit gewartet. Hier wird jedoch anders als bei dem Schreibbefehl noch kein Stoppbit gesendet. Wird das ACK-Bit erhalten, wird der Wert aus dem Register gelesen und von der Funktion zurückgegeben. Im Code wird der Anfang identisch zur Schreibfunktion umgesetzt. Als erstes wird geprüft, dass sich die Registeradresse in einem gültigen Adressbereich befindet. Der erste Unterschied lieg darin, dass statt einer 0 eine 1 an die Adresse als R/W-Bit angehängt wird, um eine Leseoperation zu kennzeichnen. Im Anschluss wird auch hier hier die Registeradresse in den Buffer geschrieben. Schlägt Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.eine dieser Schreibfunktionen fehl, wird die Funktion mit einer Fehlerrückgabe beendet. Anders als bei der Schreibfunktion werden hier keine Daten mehr geschrieben, sondern es folgt endTransmission zum Senden den Buffers. Das ist nötig, damit der Chip für die folgende Leseanfrage weiß, für welches Register diese gilt. Daher ist auch darauf zu achten mit dem Senden der Daten kein Stoppbit zu schicken, da die Kommunikation ahne gelesene Daten noch nicht abgeschlossen ist. Das wird durch den Parameter „false“ erreicht, welcher der Funktion endTransmission() übergeben wird. Auch hier wird wieder auf eine korrekte Übertragung und ein erhaltenes Acknowledge-Bit geprüft. Über die Funktion requestFrom() wird nun der Lesevorgang eingeleitet. Mit dieser Funktion stellt der Mikrocontroller eine Leseanfrage an die Porterweiterung. Dabei werden wieder die Geräteadresse und die Anzahl an erforderten Bytes übergeben. Auf diese Anfrage antwortet der Chip mit dem Inhalt des Registers. Dieser wird im Eingangsbuffer gespeichert und kann dort mit Wire.read() ausgelesen werden. Vorher wird mittels der Funktion available() abgefragt, ob sich neue Daten im Buffer befinden. Sollte diese Abfrage fehlschlagen und der ESP32 keine neuen Daten erhalten haben, wird ein Fehler zurückgegeben, andernfalls beendet sich die Funktion mit der Rückgabe der erhaltenen Daten. Ähnlich wie bei den Schreibfunktionen gibt es auch hier eine Funktion, welche nur ein Bit zurückgibt. Das es nur möglich ist ganze Register Ein Bild, das Text, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.auszulesen, nutzt die Funktion GPIO\_Exp\_ReadBit() die ReadRegister() Funktion und maskiert am Ende den erhaltenen Wert, um einen Bitwert zurückgeben zu können. Die Funktionen werden nun auch genutzt, um das Bauteil entsprechend der Anforderungen zu konfigurieren. Dazu gehört die Konfigurierung der Pins als Input/Output, sowie anderer Register. Dies findet wie die Konfigurierung der Mikrocontroller Pins in der Setup-Funktion statt. Dort wird die Funktion init\_GPIO\_Expansion() aufgerufen. In dieser Funktion wird als erstes das IO-Control Register beschrieben. In diesem Register kann die Adressierung der Register definiert werden. Durch das Setzen von Bit7 des Registers werden die Adressen für Bank A und Bank B blockweise unterteilt, sodass die Adressen 00-0A für die Register von GPIOA genutzt werden. So können nun die entsprechenden Register korrekt adressiert werden. Als nächstes sollen die Ports als Ein und Ausgänge definiert werden. Dafür müssen die Register IO\_Direction A (IODIRA) und IO-Direction B (IODIRB) beschrieben werden.   
Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.  
Pin 0 entspricht dabei Bit 0. Die Ports werden nun entsprechend dem Schaltplan definiert. Jede Aufgerufene Funktion gibt dabei ihren Rückgabewert aus, welcher bei erfolgreicher Bearbeitung 1 betragen muss. Damit nicht jeder Funktionsaufruf einzeln kontrolliert werden muss, werden die Rückgabewerte summiert und blockweise überprüft.  
Um manche Signale korrekt auswerten zu können, bietet das Bauteil die Option aus der Software heraus Pull-Up Widerstände zu aktivieren. Diese werden hier nicht benötigt, da alle Pull-Up Widerstände bereits im Schaltplan zur Bestückung vorgesehen sind. Um keine Signale zu verfälschen, werden die Pull-Up Widerstände für alle Pins deaktiviert. Als letztes müssen die Interruptquellen definiert werden. Dafür werden zuerst alle Eingangspins als Interruptquelle freigegeben. Das passiert über das Register GP\_Interrupt\_Enable\_A (GPINTENA). Hier werden alle benötigten Bits auf 1 gesetzt. Das wird über den Wert 0xAF für die Bits 7, 5, 3, 2, 1 und 0 erreicht. Zur Auslosung von Interrupts gibt es 2 Varianten. Entweder wird ein Interrupt Ausgelöst, wenn der Wert dem Invertierten Wert entspricht, welcher im Default Value Register (DEFVALA) definiert ist oder der Chip löst einen Interrupt bei jeder Änderung der Pin-Levels aus. Das Interruptverhalten wird dabei im Interrupt Controll register (INTCONA) festgelegt. Soll der Wert gegen das DEFVAL-Register vergleicht werden, muss das entsprechnde Bitim INTCONA auf 1 gesetzt werden. Das ist der Fall für die Pins 5, 3, 2 und 0. Für diese Pins muss entsprechend auch der Wert im DEFVALA\_Register gesetzt werden. Bei den Signalen an Pin 0 und 2 handelt es sich um lowactive Signale, weshalb der Wert hierfür auf High gesetzt werden muss, da der Interrupt immer beim invertierten Signalpegel ausgelöst wird. Die Werte für Pins 3 und 5 werden auf Low gesetzt.Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.  
Die Initialisierungen der Interrupts ergeben sich durch die Funktionalitäten der an den Pins angeschlossenen Signale. Die Pins 0 und 2 lesen die beiden lowaktiven LEDs vom Funkmodul ein und sollen die Impulse zählen, welche zur Fehler oder Modusindifizierung ausgesendet werden. Dafür sollen sie auf jeden Low-Pegel reagieren. Ähnliches gilt für Pins 3 und 5. Hier liegen die Signale an, welche das Funkmodul für den Pegel der Kanäle 2 und 3 verwendet. Diese sind Highaktiv, weshalb der Mikrocontroller bei jedem Highpegel über das Eintreffen eines Funksignals informiert werden soll. Anders sieht es bei den Signalen 1 und 7 aus. Diese schalten die Status-LED. Das bedeutet, das bei jedem High-pegel die LED eingeschalten werden soll, bei jedem Low-Pegel aber auch wieder aus. Daher wird hier immer gegen den vorherigen Wert am Pin verglichen und nicht gegen einen fest definierten wert. Zuletzt soll über IOCAN1 noch die Polarität des Interrupt-Signals an den ESP32 gesetzt werden, welches hier als Highaktiv definiert wird.   
Zuletzt wird noch einer der Ausgänge beschalten. Dort ist an Pin1 ein lowaktives Signal angeschlossen, welches standartmäßig nicht aktiv sein soll. Dieses wird daher auf High gezogen. Dafür wird das GPIOB-Register beschrieben.  


Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.Wird ein Interrupt von der GPIO-Erweiterung an den Mikrocontroller gesendet, reagiert der Chip auf die Steigende Flanke und fügt die Interrupt Service Routine aus. Das ist für die Porterweiterung die Funktion ISR\_GPIO\_Expansion(). Diese Funktion macht nichts anderes als ein Flag zu setzen, welches dafür sorgt, dass der Interrupt im nächsten durchlauf des Loop abgearbeitet wird. Die Abarbeitung des Interrupts erfolgt in der Funktion process\_ISR\_GPIO\_Expansion(). In dieser wird als erstes das Interrupt\_Flag-Register (INTFA) ausgelesen. In diesem Register ist das Bit aktiv, für welches ein Interrupt generiert wurde. Dieses wird nun maskiert und für jedes Bit gecheckt und entsprechend verarbeitet. Die Verarbeitung soll für die jeweiligen Signale gesondert betrachtet werden. Dabei wird das Register durch das Auslesen Automatisch zurückgesetzt.

Die Betrachtung für die Verarbeitung der Interrupt soll anhand der Pins 3 und 5 direkt betrachtet werden. Dabei handelt es sich um die empfangenen Ein Bild, das Text, Diagramm, Entwurf enthält.

KI-generierte Inhalte können fehlerhaft sein.Signale für Kanal 2 und 3 des Funkmoduls. Nachdem das Signal am Mikrocontroller empfangen wurde, soll als erstes gecheckt werden, welche Einstellungen aktiv sind. Standartmäßig soll das Signal als Can-Message auf CAN-Bus 1 an die VCU gesendet werden. Es gibt jedoch auch alte Systeme, welche noch nicht über CAN-Kommunikation verfügen. Für diese ist es Möglich die Signale über einzelne Signalleitungen zu übertragen. In diesem Fall muss dafür der entsprechende Transistor geschalten werden, welcher das Signal erzeugt. Dieser muss nach einer kurzen Wartezeit auch wieder deaktiviert werden. Programmiertechnisch soll die Umsetzung beispielhaft an Kanal2 gezeigt werden. In der Interrupt Serviceroutine des GPIO-Expanders wird ein Flag ISR\_RX\_2 gesetzt, wenn der entsprechende Pin für das Funksignal als Interruptquelle identifiziert wurde. Dieses Flag wird im Loop zyklisch abgefragt und bei Bedarf die Funktion send\_RemoteDrive\_Request() aufgerufen. Diese Funktion prüft als erstes, ob das Signal analog oder per Can gesendet werden soll. Die Einstellungen dafür sind im Non-Volatile Storage (NVS) des ESP32 gespeichert, einem Speicherbereich, welcher auch nach Reboot erhalten bleibt. Wenn das Signal analog gesendet werden soll, wird nun der Transistor geschalten. Dieser wird über Pin 4 der Porterweiterung gesteuert, weswegen dort ein Bit über die bereits betrachteten Funktionen gesetzt wird. Zusätzlich wird der aktuelle Zeitpunkt zuzüglich einer Wartezeit gespeichert. Das Flag wird noch nicht zurückgesetzt. Das liegt an dem Delay, welches zwischen ein und Ausschalten abgewartet werden muss.Ein Bild, das Text, Screenshot, Software, Betriebssystem enthält.

KI-generierte Inhalte können fehlerhaft sein. Um nicht die Abarbeitung aller anderen Funktionen zu blockieren, wird hier nicht die Funktion delay() genutzt. Stattdessen wird die Funktion zyklisch aufgerufen. Die weitere Abarbeitung ist jedoch so lange unterbrochen, bis der aktuelle Zeitpunkt vor dem liegt, welcher zu beginn, zuzüglich der Wartezeit, gespeichert wurde. Auf diese Weise unterbricht die Funktion keine anderen Prozesse. Damit der Ausgang der Porterweiterung nicht bei jedem Durchlauf erneut beschrieben wird, wird der erste teil der Funktion durch die Variable Signal\_active verriegelt, welche erst bei einem Fehler oder nach Ablauf des Delays zurückgesetzt wird. Nach Ablauf dieser Wartefunktion wird der Transistor wieder zurückgesetzt, sodass an der VCU ein Signalimpuls erkannt wird. Zu diesem Zeitpunkt wird auch das Flag aus dem Loop zurückgesetzt, um die weitere Abarbeitung zu verhindern. Ist die Einstellung, die Signale via Can zu schicken aktiv, so wird die Funktion send\_CAN1\_Message() mit entsprechenden Parametern aufgerufen. Diese Funktion soll später genauer betrachtet werden. Alle hier getätigten Aussagen gelten auch für empfangene Signale von Kanal 2, lediglich mit Änderung der Signalnamen.   
Für empfangene Signale auf Kanal 4 gibt es jedoch einige Unterschiede. Hierbei handelt es sich um das Signal zur Identifizierung des Karts über die Statusleuchte. Schon bei der Abfrage der Interrupt Flags unterscheidet sich dieser Kanal von den anderen. Anders als bei Kanal 2 und 3 wird jedes Mal, wenn sich der zustand des Pins ändert ein Interrupt gesendet.

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.Das hängt damit zusammen, dass die LED so lange Leuchten soll, wie der Knopf auf der Fernbedienung gedrückt wird. Das Flag wird daher nur von der ISR des Port Expanders verändert. Die Auswertung wiederum findet wie bei den anderen Kanälen im Loop statt. Hier muss jedoch zwischen den Unterschiedlichen Stati der LED unterschieden werden. So soll die LED nur geschalten werden, wenn sie aktuell aus ist und das Flag gesetzt ist. Andersherum soll sie nur ausgeschalten werden, wenn das Flag nicht gesetzt ist, die LED aber noch leuchtet. Zu diesem Zweck wird der Zustand der LED in einer Variable ID\_cur\_state gespeichert. Dabei handelt es sich nicht um den realen Status der LED, sondern um einen pseudozustand, welcher jedes Mal verändert wird, wenn das Flag abgearbeitet wird. Dabei wird davon ausgegangen, dass die LED zu Beginn immer ausgeschalten ist. Über die Funktionen Status\_LED\_ON() und Status\_LED\_OFF() wird dann der Transistor, welcher als Treiber für die LED fungiert, geschalten oder zurückgesetzt. Dieser wird über Pin 0 an der Porterweiterung geschalten.  
Die LED wird jedoch nicht nur von dem Funkempfänger gesteuert, es muss auch möglich sein ihren Zustand von der VCU aus zu verändern. Hierbei gibt es zwei Varianten wie dieses Signal an die Erweiterungsplatine übermittelt werden soll. Für alte Signale wird das Signal über die bestehende Signalleitung im Kabelbaum übermittelt. Das Signal wird dann über die Porterweiterung ausgelesen. Dabei löst das Signal einen Interrupt aus. Genau wie Kanal 4 des Funkempfängers, wird bei jeder Veränderung am Pins ein Interrupt generiert. Dieser setzt und deaktiviert ein Flag, welches im Loop ausgewertet wird und dort die LED schaltet. Wichtig ist dabei, dass diese Abarbeitung des Flag nur stattfindet, wenn die Funktionalität der LED auch vorhanden ist. Auch hier wird darauf geachtet, dass die LED nur geschalten wird, wenn ihr aktueller Zustand das auch erlaubt. Dafür ist die variable LED\_cur\_state vorgesehen, welche ebenso wie die Variable ID\_cur\_state einen Pseudozustand der LED speichert. Zusätzlich wird auch der Status von ID\_cur\_state abgefragt, also ob aktuell ein Kart über die Fernbedienung identifiziert wird, da das Funksignal Priorität gegenüber dem VCU Signal besitzt. Das hat den Hintergrund, dass die LED zu blinken beginnt, sobald das Kart in eine nReady-To-Drive Zustand versetzt wird. Auch in diesem Modus soll es noch Möglich sein das Kart über die Fernbedienung zu identifizieren.

Ein Bild, das Text, Screenshot, Software, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.  
Die Zweite Variante, wie der Mikrocontroller Information zur Status-LED erhält ist per CAN-Message. Diese Option soll zu einem Späteren Zeitpunkt betrachtet werden.

Ein Bild, das Diagramm, Text, Plan, technische Zeichnung enthält.

KI-generierte Inhalte können fehlerhaft sein.Damit der Funkempfänger Signale empfangen kann, muss zuerst eine Fernbedienung verbunden werden können. Dafür sendet der Mikrocontroller ein Signal an das Funkmodul, welches daraufhin in einen Pairing-Modus eintritt. Dabei gibt es verschiedene Lernmodi, in welche das Modul versetzt werden kann. Die Unterschiedlichen Modi werden dabei durch Unterschiedliche Anzahl und Länge der Signalimpulse definiert. Für den Eintritt in einen der Lernmodi, wird das Signal mit einer Signaldauer von weniger als einer Sekunde gesendet. Die Anzahl der gesendeten Signale bestimmt dabei die Art des Lernmodus. Für Senden eines Löschsignals muss die Signallänge mindestens 3 Sekunden betragen.

Ein Bild, das Text, Schrift, Screenshot, Grafiken enthält.

KI-generierte Inhalte können fehlerhaft sein.Durch diese langen Signalzeiten ist es hierbei besonders wichtig auf blockierende Wartefunktionen im Code zu verzichten. Auch hier wird deshalb wieder die Funktion so lange pausiert, bis der Timout der Funktion abgelaufen ist. Sie wird zwar in dieser zeit zyklisch aufgerufen, es findet aber keine Verarbeitung statt, da sie direkt wieder verlassen wird.

Ein Bild, das Text, Screenshot, Software, Betriebssystem enthält.

KI-generierte Inhalte können fehlerhaft sein.Mit Aufruf der Funktion gibt es noch keine Timerbedinungen, weshalb bei erstem Eintritt in die Funktion keine Wartezeit abgewartet werden muss. Stattdessen wird das Signal aktiviert, indem der zugehörige Ausgang an der Porterweiterung auf Low gesetzt wird, da es sich hier um ein lowaktives Signal handelt. Hier wird auch, abhängig vom übergebenen Modus die Wartezeit festgelegt, für welche das Signal aktiv sein soll. Damit der Ausgang nicht bei jedem zyklischen Aufruf neu beschrieben wird, wird das Setzen des Signals über die Variable Signal\_active verriegelt. Dieses wird nur im Falle eines Fehlers in der I²C-Kommunikation oder beim zurücksetzen des Signals wieder freigegeben. Zusätzlich wird im Falle eines Fehlers der Timer resettet, der Signalcounter ctr wieder zurückgesetzt und die Funktion mit einer Fehlerrückgabe verlassen. Bei Erfolg beendet sich die Funktion mit Returnwert 0, welcher als Pausierte Funktion interpretiert wird.

Ist die Wartezeit abgelaufen, soll das Signal wieder zurückgesetzt werden. Das soll nur geschehen, wenn das Signal vorher aktiv war, um unnötige Schreibprozesse auf die Porterweiterung zu vermeiden. Hier wird als erstes das Signal wieder auf inaktiv gesetzt, um bei erneutem Aufruf der Funktion wieder am Anfang der Funktion starten zu können. Auch wird hier der Ein Bild, das Text, Screenshot, Software, Betriebssystem enthält.

KI-generierte Inhalte können fehlerhaft sein.Signalcounter erhöht, da ein weiterer Signalimpuls erfolgreich gesendet wurde. Dieses erfolgreiche Senden wird durch Rücksetzen des Pins auf High erreicht. Im Falle eines Fehlers bei der Kommunikation wird die Funktion mit Rückgabe eines Fehlers verlassen und alle Variablen auf den Initialisierungswert zurückgesetzt. Als letztes wird gecheckt, ob das Senden des erwarteten Signal abgeschlossen ist. Die Anzahl der Impulse entspricht bei den Lernmodi dem Wert des Modus, die Löschmodi benötigen jedoch nur einen oder zwei Signalimpulse und werden vorher normiert. Wenn die Anzahl der bisher gesendeten Signale noch unter der erwarteten Anzahl gesendeter Signale liegt, wird der Timer neu konfiguriert und die Funktion mit Rückgabewert 0 verlassen. Dadurch wird die Funktion im nächsten Durchlauf mit aktivierter Wartezeit von vorne begonnen. Ist das Signal vollständig, wird die Funktion erfolgreich verlassen.  
Den Aufruf der Funktion übernimmt dabei der loop. Dieser überprüft, ob das Flag ISR\_Learn\_RF\_Flag gesetzt ist. Solange das der Fall ist, wird die Funktion zyklisch aufgerufen. Sobald das Signal vollständig gesendet wurde, wird die Funktion check\_RF\_Acknowledge() aufgerufen. Diese soll anhand des LED-Signals, welches vom Funkmodul ausgegeben und vom Mikrocontroller eingelesen wird, erkennen, ob der Modus erfolgreich aktiviert wurde. Ist dies der Fall, wird das Flag ersetzt und der Aufruf der Funktionen zur Aktivierung der Lernmodi beendet. Die Funktion zur Auswertung der Rückgabewerte erfolgt dabei durch Ermittlung der Frequenz der gesendeten Signale. Die Acknowledge Signale für die Ein Bild, das Text, Screenshot, Software, Betriebssystem enthält.

KI-generierte Inhalte können fehlerhaft sein.einzelnen Modi unterschieden sich nur in der Anzahl ihrer gesendeten Signalimpulse. Dabei bezieht sich die Anzahl der gesendeten Impulse immer auf 2 Sekunden. Die Funktion setzt also zu Beginn der Funktion den Counter auf 0 zurück und startet einen 2 Sekunden-Timer. Nach Ablauf dieses Timers wird wieder der Zähler ausgelesen und die gezählten Impulse mit den erwarteten verglichen. Entspricht die Anzahl der gezählten Impulse denen des Modus, kehrt die Funktion erfolgreich zurück. Um zu verhindern, dass durch Timingprobleme oder ungünstige Signalflanken das erste Signal verfälscht ist, wird die Funktion zyklisch aufgerufen. Die Deaktivierung der Funktion nach einer bestimmten Anzahl versuche übernimmt der loop. Das Zählen der Signale und erhöhen des Counters übernimmt dabei die Interrupt Service Routine des GPIO Port Expanders. Dieser reagiert mit einem Interrupt auf jeden Lowpegel am entsprechenden Pin und erhöht den Counter in seiner ISR.  
So ähnlich läuft auch die Verarbeitung von aufgetretenen Fehlern ab, welche von dem Funkempfänger an den ESP32 gemeldet werden. In der Auswertungsfunktion der Interrupts der Port Erweiterung wird ein Counter erhöht, für jedes Fehlersignal, dass vom Funkempfänger gesendet wird. Auch hier wird das Signal über die Anzahl der gesendeten Impulse definiert. Wird ein Fehlersignal registriert, wird im Loop die Funktion check\_RF\_Error() ausgeführt. In dieser Funktion wird als erstes eine Wartefunktion aufgerufen. Diese hat die Aufgabe nach dem ersten Impuls, welcher zum Aufruf der Funktion geführt hat, auf weitere Signalimpulse zu warten. Danach werden die erhaltenen Fehlerinformationen ausgewertet. Zwei Impulse haben dabei mehrere Bedeutungen, je nach aktuell aktiviertem Lernmodus. Im Zusammenhang mit einem aktiven Löschmodus, bedeutet Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.diese Sequenz, dass der gewünschte Eintrag nicht gelöscht werden konnte. In Zusammenhang mit einem der Lernmodi weist dieser Fehler darauf hin, dass die Fernbedienung bereits verbunden wurde. Drei Impulse hingegen bedeuten, dass der Adressspeicher des Moduls bereits gefüllt ist und keine weiteren Fernbedienungen mehr hinzugefügt werden können. Tritt einer dieser Fehler auf, soll das auf der Webseite für den Nutzer dargestellt werden. Zum Abschluss der Funktion wird der Fehlerzähler resettet, um für die nächsten Auftretenden Fehler keine verfälschten Daten zu erhalten.

Als nächstes soll die Software betrachtet werden, welche für die Kommunikation über den CAN1 Bus benötigt wird. Can wurde bereits für die Übermittlung der Funksignale an das Steuergerät und den Erhalt von Befehlen zum Schalten der LED kurz genannt. Diese Funktionalitäten sollen nun im Folgenden betrachtet und erläutert werden, dabei soll es nicht nur um senden und empfangen von Nachrichten gehen, sondern auch die Verarbeitung der erhaltenen Daten betrachtet werden.

Die dafür notwendige Initialisierung findet auch für CAN wieder in der Funktion Setup() statt, welche immer einmal nach dem Bootvorgang abgearbeitet wird. Die Initialisierung für CAN ist dabei zweiteilig. Als erstes findet die Definition der Pins und des Can-Objekts statt. Für die Can-Kommunikation werden die Librarys <ESP32CAN.h> und <CAN\_config.h> genutzt. Bei dieser wird ein Objekt vom Typ CAN\_device\_t mit Namen CAN\_cfg erzeugt. Dieses speichert alle zur Kommunikation nötigen Einstellungen wie Pins und Busrate und ermöglicht den Zugriff auf empfangene und zu Sendende Nachrichten. So werden die beiden Pins für die Kommunikation auf die bereits vorher definierten Pins des ESP32 festgelegt. Über das Attribut speed wird die Busrate festgelegt, als mit Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.welcher Geschwindigkeit der Bus kommuniziert. Diese beträgt beim vorliegenden System 500 Mbit/s. Um Nachrichten empfangen zu können muss ein Buffer für Nachrichten definiert werden. Das passiert über das Attribut rx\_queue und die Funktion xQueueCreate(). Diese alloziert Speicherplatz, in welchem ankommende Nachrichten gespeichert werden können und erwartet dabei als Attribute die Anzahl an Speichereinheiten und die Größe der zu reservierenden Speichereinheiten. Die Größe der Speichereinheiten wird dabei als die Größe eines CAN-Nachrichten Objekts festgelegt, wovon maximal 40 im Buffer gespeichert werden sollen. Danach wird die bereits von der Library zur Verfügung gestellte Funktion CANInit aufgerufen, welche die Kommunikation für den internen TWAI-Controller aufbaut und ihn entsprechend konfiguriert. Ist diese Funktion erfolgreich, wird dies in Form der Variable CAN1\_active gespeichert. Diese dient dazu, um keine Fehler durch Versuche auf einen inaktiven CAN-Bus zu senden zu erzeugen.   
Der zweite Teil der Initialisierungsfunktion betrifft nicht die Konfiguration für TWAI-Controller oder ESP32, sondern definiert die CN-Nachrichten und Signale gemäß der vorhandenen DBC-File. Da es sich nur um wenige Nachrichten und Signale handelt, werden diese händisch im C-Code definiert. Dafür wurden in einer eigenen Headerdatei Strukturen für Nachrichten und Signale definiert, welche die Eigenschaften der Signale Ein Bild, das Text, Screenshot, Schrift, Design enthält.

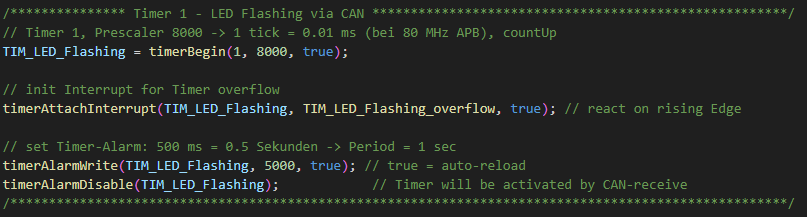
KI-generierte Inhalte können fehlerhaft sein.und Nachrichten im C-Code abbilden. Zu den zu speichernden Daten für die Signale gehört dabei die Länge in Bits, die Polarität des gespeicherten Wertes, also ob dieser Wert als negativer Wert interpretiert werden kann. Zusätzlich wird ein Faktor gespeichert, mit welchem die ausgelesenen Daten multipliziert werden sollen, um zum Beispiel Kommazahlen übertragen zu können. Der Offset Wert wird verwendet, um den Wertebereich des Signals zu verschieben, da dieser auf den erhaltenen Wert aufaddiert wird. Zuletzt wird für jedes Signal noch sein Startbit innerhalb der Nachricht gespeichert. Die Nachricht hingegen speichert nur die Signale, welche der Nachricht zugeordnet wurden, sowie deren Anzahl. Die Nachrichten werden dabei als Zeiger auf ein Objekt gespeichert, um keine unnötigen Kopien der Objekte zu erzeugen. Die wichtigste Eigenschaft, welche eine Nachricht besitzt, ist ihre ID. Über diese wird jede Nachricht identifiziert. Innerhalb der Funktion define\_CAN1\_Messages() werden nun alle benötigten Signale und Nachrichten erstellt und die Signale den Nachrichten zugewiesen. Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.Beispielhaft kann man das an dem Signal Overall\_Voltage und der Nachricht Battery\_Voltage erkennen. Mit den Übergebenen Parametern ergibt sich für das Signal ein Wertebereich von 0 bis 255 V, welcher mit 8 Bits darstellbar ist. Durch den Faktor von 0.1 wird der Wertebereich zugunsten der Genauigkeit auf 0 – 25.5V verkleinert. Um den geforderten Wertebereich, in welchem sich die Akkuspannung bewegen kann, abzubilden, wird nun noch ein Offset von 40V hinzugefügt, damit können Werte zwischen 40V und 65.V abgebildet werden. Dieses Signal wird anschließend der Nachricht Battery\_Voltage hinzugefügt. Diese besitzt die ID 32 oder 0x20. Neben der Gesamtspannung des Akkus überträgt diese Nachricht noch drei weitere Signale, welche die Minimale, Maximale und Durchschnittliche Zellspannung abbilden.   
Diese Informationen können genutzt werden, um CAN-Frames zu bauen, welche an das Steuergerät gesendet werden sollen. Das wird hier am Beispiel des SOC-Request Signals gezeigt, welches vom Mikrocontroller an das Steuergerät gesendet wird. Zum Senden wird ein Array festgelegt, wobei jeder Eintrag ein Datenbyte enthält. Die Position der Datenbits innerhalb des Bytes wird über das Startbit definiert, welches im entsprechenden Ein Bild, das Text, Screenshot, Schrift enthält.

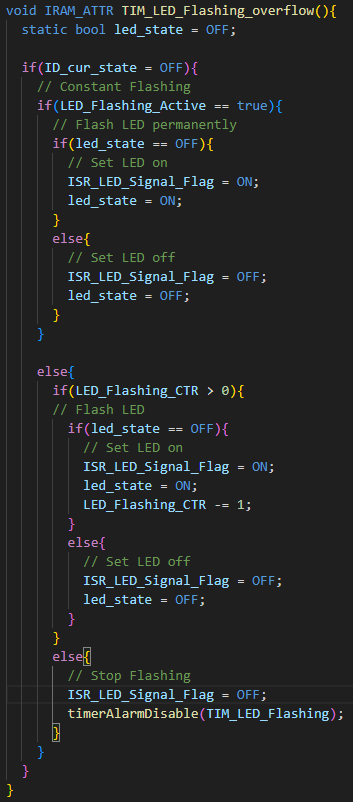
KI-generierte Inhalte können fehlerhaft sein.Signalobjekt gespeichert ist. Dadurch, dass hier nur ein einzelnes Bit gesendet wird, können die Faktoren Offset und Faktor vernachlässigt werden. Das tatsächliche Senden der Nachricht passiert in der Funktion send\_CAN1\_Message(). Diese erwartet neben dem gerade erstellten Datenarray die ID der zu sendenden Nachricht und die Anzahl der zu Sendenden Bytes. Innerhalb der Funktion wird zuerst die Korrektheit der Eingaben überprüft. Sollte hierbei ein Fehler auftreten, wird die Funktion Ein Bild, das Text, Screenshot, Software, Betriebssystem enthält.

KI-generierte Inhalte können fehlerhaft sein.mit einem Fehler verlassen. Darauf folgt der Aufbau des CAN-Frame, welcher durch die Library definiert wird und an die VCU gesendet werden soll. Hier werden die an die Funktion übergebenen Parameter an das CAN-Objekt übergeben. Dafür wird zuerst das Format der Nachricht definiert. In diesem Fall handelt es sich um eine Nachricht im Standartformat, ohne Extended ID oder andere Sonderformate. Als nächstes werden Identifier und Datenlänge übergeben, bevor die Daten aus dem übergebenen Array in den Frame übertragen werden. Über die Funktion CANWriteFrame() wird nun die Nachricht gebaut, in den Sendebuffer geschrieben und schließlich vom CAN-transceiver auf den Bus gesendet. Ist diese Funktion erfolgreich, wird die Funktion erfolgreich verlassen, in andern Fällen unter Rückgabe eines Fehlers beendet.  
Der Empfang und die Auswertung empfangener Nachrichten ist dagegen in mehrere Abschnitte aufgeteilt. Den Empfang der Nachrichten übernimmt dabei der Controller selbstständig und schreibt alle empfangenen Nachrichten in einen Empfangsbuffer, welcher bei der Initialisierung des CAN-Objekts definiert wurde. Um Nachrichten verarbeiten zu können, muss nun lediglich der Speicher zyklisch ausgelesen werden. Das passiert innerhalb der Loopfunktion. Der Mikrocontroller reagiert dabei nicht auf einen Interrupt oder ähnliches, sondern versucht bei jedem Durchlauf der Loop-Funktion den Speicher auszulesen. Dafür wird die Funktion process\_CAN1() aufgerufen. Diese liest mit der Funktion xQueueReceive den Buffer des CAN-Controllers aus. Dabei muss ihr dieser Speicherbereich übergeben werden zusammen mit einem CAN-Frame Objekt, in welches der Inhalt der ausgelesenen Nachricht gespeichert werden soll. Zudem wird ein Timeout in Ticks hinzugefügt, welches angibt, wie lange die Funktion auf neue Nachrichten warten soll, bevor sie eine Ein Bild, das Text, Screenshot, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.Nachricht ausliest. Wurde im Buffer eine Nachricht empfangen, gibt die Funktion pdTRUE zurück und die Verarbeitung des erhaltenen CAN-Frame kann beginnen. Dafür wird lediglich die ID der Nachricht aus dem Objekt ausgelesen. Diese wird dann mit IDs der Nachrichten verglichen, welche verarbeitet werden sollen, um jede Nachricht entsprechend ihrer Anforderungen und enthaltenen Signale auswerten zu können. Stimmt die ausgelesene ID mit einer der erwarteten überein, wird für diese Auswertungsfunktion aufgerufen. Diese Auswertung soll beispielhaft anhand einer erhaltenen Battery\_Voltage Nachricht gezeigt werden. In diesem Fall würde die Funktion process\_CAN\_BatteryVoltage() aufgerufen werden. Diese erhält für alle benötigten Signale aus einer Nachricht den Wert anhand der für jedes Signal spezifizierten Einstellungen über die Funktion decodeSignal() und verarbeitet diesen Wert entsprechend. Im Falle der Nachricht Battery\_Voltage ist nur das Signal Overall\_Voltage interessant. Dessen Wert wird auf dem Display und auf der Livedatenseite dargestellt. Dafür wird der Wert in eine globale Variable geschrieben, welche nach einer Aktualisierung von Webseite und Display dargestellt wird. Die Funktion decodeSignal erwartet für die Auswertung neben dem erhaltenen CAN-Frame auch das Signalobjekt, welches aus der Nachricht ausgelesen werden soll. Diese Funktion erzeugt zuerst aus den Datenbytes, welche in einem Array gespeichert sind, einen großen Wert. Das ist sehr hilfreich für Signale, welche länger als ein Byte sind oder Daten, die über Bytegrenzen hinweg gespeichert sind. Um das zu erreichen, werden die Bytes gemäß ihrer Position immer um Vielfache von 8 bitweise geschoben und mit den Bytes vorher verbunden. Um nur die Bits zu betrachten, welche auch zum passenden Signal gehören, wird als erstes eine Maske generiert. Ein Bild, das Text, Screenshot, Software, Betriebssystem enthält.

KI-generierte Inhalte können fehlerhaft sein.Diese Maske besitzt nur in dem Bereich Einsen, in denen das Signal gespeichert ist. Diese Maske wird über die gespeicherte Signallänge definiert. Damit wird das Signal maskiert, sodass alles Bits gelöscht werden, die nicht zu den Daten des Signal gehören und die Daten um den Wert des Startbits zurückgeschoben, wodurch der Wert der Daten nun dem Wert des Signals entspricht. Um den korrekten Wert des Signals zu erhalten, wird es noch mit dem definierten Faktor multipliziert und anschließend der Offset-Faktor aufaddiert. Dieser Wert wird zurückgegeben. Dieser Ablauf ist für alle erhaltenen Nachrichten derselbe bis auf die Nachricht Option1\_Commands. Alle anderen Nachrichten enthalten nur Signale, deren Daten visuell auf dem Display oder der Webseite dargestellt werden sollen. Die Nachricht Option1\_Commands enthält hingegen Befehle vom Steuergerät an die Erweiterungsplatine, welche entsprechend verarbeitet werden müssen. Dabei handelt es sich um die Befehle zur Steuerung der Status-LED. Dabei wird zwischen zwei Signalen unterschieden. Das erste der beiden Signale übermittelt einen Wert, welcher Angibt, wie oft die LED blinken soll. Das wird genutzt, wenn die VCU mit einem exakten Wert auf die SOC-Anfrage antwortet. Dieser Wert wird ebenfalls in eine Globale Variable geschrieben, welche als Zähler fungiert. Das zweite Signal ist Enable-Bit, um die LED so lange blinken zu lassen, bis die VCU das Blinken wieder beendet. Die Bearbeitungsfunktion setzt hierfür ein Flag, welches das Blinken für andere Funktionen indiziert. Zusätzlich aktivieren beide Signale einen Timer. Dieser Timer dient als Taktgeber für das Blinken der LED. Dieser Timer wird bereits zu Beginn des Programmablaufs im Setup initialisiert und gestartet, jedoch noch nicht aktiviert. Der Timer mit seinen Einstellungen wird über die Funktion timerBegin() gestartet und initialisiert. Der Takt des Timer ergibt sich über den Prescaler und den Wert im Auto Reload Register (ARR).   
Über den Prescaler wird die Timerauflösung festgelegt und darüber die Zählgeschwindigkeit. Dabei handelt es sich lediglich um einen Vorteiler, welcher die Eingangsfrequenz verkleinert. Das ARR legt den Wert fest, bei dem der Zähler resettet wird und eine Funktion ausführt. Das bestimmt die Laufzeit des Timers. Die beiden Werte lassen sich dabei über folgende Formel berechnen:

Für die Frequenz, welche der Timer ausgeben soll, wird 2 Hz festgelegt. Für den Prescaler bietet es sich an Vielfache des Clock-Taktes zu wählen, um gerade Werte für die Eingangsfrequenz zu erhalten.

Der Wert des Prescalers wird direkt beim Start des Timers festgelegt. Der Wert des ARR wird über die Funktion timerAlarmWrite() festgelegt. Hier wird neben dem Timerobjekt der Wert des ARR übergeben, bei dem der Timer einen Alarm auslösen soll. Dieser Alarm löst einen Interrupt aus. Dieser wird über die Funktion timerAttachinterrupt() konfiguriert. Hier wird festgelegt, welche Funktion als ISR bei Erreichen des ARR ausgeführt wird.   
Das ist die Funktion TIM\_LED\_Flashing\_overflow(). Die Funktion übernimmt dabei sowohl das dauerhafte Blinken als auch die Kontrolle über eine Bestimmte Anzahl an Impulsen für die LED. Zuerst wird wieder die Priorisierung der LED-Signalquellen beachtet. So soll die LED nur blinken, wenn das Signal gerade nicht durch die Fernbedienung überschrieben wird. Ist das Flag für das dauerhafte Blinken gesetzt wird der Counter zur Bestimmung der Anzahl der Impulse missachtet und der Zustand der LED wird bei jedem erreichen des Timer-ARR gewechselt. Ist das Flag nicht gesetzt, prüft die Funktion, ob der Counter noch Impulse der LED erwartet. Ist das der Fall wird auch hier mit jedem Aufruf der Funktion der Zustand der LED gewechselt. Zusätzlich wird jedoch der CTR bei jedem Leuchten der LED dekrementiert. Ist die korrekte Anzahl an Impulsen abgearbeitet, wird der Timeralarm deaktiviert, sodass der Timer keinen Interrupt mehr auslöst und die Funktion nicht mehr aufgerufen wird, bis wieder ein Signal via CAN erhalten wird. Da es sich hier um einen Funktionsaufruf innerhalb einer Interrupt Service Routine handelt wird nicht direkt die Funktion Status\_LED\_ON() oder Status\_LED\_OFF() aufgerufen, welche den Transistor des LED-Treibers schaltet, da diese die I²C Kommunikation zur GPIO-Porterweiterung übernimmt. Diese ist innerhalb einer ISR nicht zulässig, da sie für die Dauer der Kommunikation alle anderen Funktionen blockieren würde. Stattdessen wird hier das ISR\_LED\_Signal\_Flag gesetzt, welches regelmäßig im Loop abgefragt wird.   
  
Neben CAN-Bus 1wird auch CAN-Bus 2 für Kommunikation genutzt. Da hierfür ein eigener externer CAN-Controller nötig wurde, unterscheiden sich die Abläufe zum Empfangen und Senden von Nachrichten etwas und auch die Konfiguration des Controllers fällt umfangreicher aus als bei CAN1. Auch hier findet die Initialisierung im Setup() statt. Die Funktion init\_can2() übernimmt dabei die Konfiguration des CAN-Controllers. Die Kommunikation mit dem Controller findet dabei über SPI statt. Um die Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.Initialisierung des Kontroller vornehmen zu können, muss er zuerst in den Config Modus versetzt werden. Das passiert, indem in das CAN\_Control-Register (CANCTRL) der Wert 0x80 geschrieben wird. Der Chip quittiert diesen Config Modus, sobald er eingetreten ist, indem er im Register CAN\_Status (CANSTAT) die letzten drei Bits auf den Wert 0x4 setzt. Die Konfigurierung wird nur fortgesetzt, wenn sich der Chip im korrekten Modus befindet. Das wird über die Variable connected für den weiteren Ablauf der Initialisierung gespeichert. Um sicherzustellen, dass der Chip genug zeit hat sich in den Modus zu versetzen, wird in delay eingesetzt. Dieses stellt hier kein Problem dar, da diese Funktion nur einmal zu Beginn des Programmablaufs aufgerufen wird und daher die zyklischen Abläufe im loop nicht blockiert. Wurde der Chip erfolgreich in den korrekten Modus versetzt werden die entsprechenden Initialisierungen vorgenommen. Die Bitwerte, welche in den Registern Configuration 1 bis 3 (CNF1 … 3) festgelegt werden, bestimmen das Timing des Busses und der Kommunikation auf Bitebene. Eng damit verbunden ist die Synchronisation der einzelnen Busknoten, da bei CAN kein Clocksignal übermittelt wird. Die Kommunikation ist dabei in einzelne Segmente aufgeteilt. Ein Bild, das Text, Reihe, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Die Gesamtzeit ist die Nominal Bit Time . Diese wird durch die Busrate festgelegt. Für den vorliegenden CAN-Bus beträgt die Baudrate

Damit berechnet sich für :

setzt sich dabei zusammen aus den Einzelzeiten der Segmente, welche für eine fehlerfreie Kommunikation nicht überschreiten dürfen.

Die Definition der Längen der einzelnen Segmente findet dabei in Vielfachen des Zeitabschnitts statt. Das ist die kleinste Zeiteinheit, mit welcher der Controller arbeiten kann. Der Wert von ergibt sich dabei aus der Frequenz des Oszillators .

BRP ist dabei der Baud Rate Prescaler, welcher die Auflösung von TQ und dauer von bestimmt. Für BRP = 0 ergibt sich für folgender Wert:

Für die Gesamtlänge ergibt sich somit

Das erste Segment, welches innerhalb dieser 8 stattfinden muss, ist das Synchronization Segment. Um Phasenverschiebungen zwischen den Oszillatorfrequenzen der einzelnen Knoten auf dem Bus auszugleichen, muss jeder CAN-Controller in der Lage sein, sich mit der relevanten Signalphase des eingehenden Signals zu synchronisieren. Dieses Segment ist auf die Länge von 1 festgelegt.  
Das Segment, dessen Länge als nächstes festgelegt wird, ist Phase Segment 2. Dieses ist vorhanden, um Edge Phase Errors auf dem Bus zu auszugleichen. Dieses Segment beginnt immer mit dem Sample Point. Das ist der Punkt an dem der Logikpegel gelesen und ausgewertet wird. Dieser sollte immer bei etwa 70% der Gesamtzeit liegen. Damit ergibt sich für :

Da nur als ganzzahliges Vielfaches auftreten darf und mindestens betragen muss, wird festgelegt.

Für die Festlegungen der Längen von Propagation Segment und Phase Segment müssen zudem eine Regel erfüllt sein:

Mit und bleiben für die restlichen beiden Segmente noch

Damit ist die Bedingung ebenfalls erfüllt.

Zuletzt müssen noch die Zeiten für Propagation und Phase Segment 1 festgelegt werden. Diese beiden müssen zusammen 4 betragen, um die Gesamtzeit auszunutzen. Da die Laufzeit für Phase Segment 1 meist größer gewählt wird als die vom Propagationsegment, ergeben sich die Zeiten zu

Zusätzlich zu den Segmentlängen muss die Synchronization Jump Width (SJW) festgelegt werden. Diese passt den Bit-Takt nach Bedarf um an, um die Synchronisation mit der übertragenen Nachricht aufrechtzuerhalten. Zusätzlich gilt für SJW die Bedingung:. Typischerweise wird SJW bei stabiler Taktquelle auf festgelegt.

Die Initialisierung des Chips mit diesen Werten findet in den Registern Configuration 1 bis 3 (CNF 1… 3) statt. In CNF1 werden die Bits 6 und 7 zur Speicherung des Wertes von SJW verwendet. Dabei entspricht der Wert im Register SJW – 1. Für einen Wert von 1 werden die Bits mit 00 beschrieben. Die Bits 0 bis 5 speichern den Prescaler BRP. Dieser wurde für die obere Rechnung als 0 festgelegt. Damit ergibt sich der Registerwert zu 0x00.

In CNF2 werden die Werte für Propagation Segment und Phase Segment festgelegt. Auch hierbei gilt, dass der im Register gespeichert Wert   
 und entspricht. Die Länge des Propagation Segment wird dabei in die Bits 0 bis 2 eingetragen, die des Phase Segment 1 in die Bits 3 – 5. Bit 6 wird als Konfigurationsbit für den Sample Point genutzt. Wird dieses Bit auf Eins gesetzt, wird am Sample Point Drei mal abgetastet, liegt das Bit auf 0 nur einmal. Diese häufigere Abtastung hat den Vorteil das Fehler minimiert werden, da bis zu ein Fehler ignoriert werden kann. Bit 7 wird verwendet, um die Festlegung des Zeitwertes für Phase Segment 2 zu bestimmen. Wird dieses Bit auf 0 gesetzt, wird festgelegt. Mit dem Bit auf 1 wird der Wert verwendet, welcher für die Länge des Phase Segment 2 im CNF3 – Register festgelegt wird. Für die aktuelle Variante soll der Wert über das CNF3 Register festgelegt werden, sodass der Wert auf 1 gesetzt wird. So ergibt sich folgender Wert für das Register:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.Wie bereits angesprochen muss nun noch der Wert für Phase Segment 2 in CNF3 gespeichert werden. Dafür werden die Bits 0 bis 2 beschrieben. Auch hier wird als Registerwert eingetragen.

Zusätzlich zu den Konfigurationen für Timing und Synchronisation müssen auch die Einstellungen für Interrupt Erzeugung übergeben werden. Dabei wird über das Register CAN\_Interrupt\_Enable (CANINTE) verschieden Interruptquellen aktiviert. Für das vorliegenden Beispiel sollen Interrupts für Receive Buffer 0 Full, Receive Buffer 1 Full, Error Interrupt und Message Error Interrupt freigeschalten werden. Als letztes muss der Controller für normalen Betrieb wieder in den Normal Mode zurückgesetzt werden. Dafür wird wie bei der Aktivierung des Config Modus das CANCTRL-Register beschrieben, wobei der Wert 00 den korrekten Modus aktiviert.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein. Um diese Konfigurierungen der Register vornehmen zu können werden Funktionen benötigt, welche per SPI-Daten in die Register schreiben oder daraus lesen. Die dafür geschriebenen Funktionen sind write\_SPI\_Register() und read\_SPI\_Register(). Die Funktionen nutzen dabei die bereits vorhandenen Funktionen aus der SPI-Library zur Kommunikation auf dem Bus. So beginnt die Kommunikation immer mit dem Aktivieren des entsprechenden Chips über den Chip Select. Das übernimmt die Funktion SPI\_select(), welche den übergebenen Chip über die passende CS-leitung aktiviert. Als erstes wird der Befehl für Lesen oder Schreiben übermittelt. Das passiert über die Funktion transfer() aus der SPI-Library. Auf den Befehl folgt die zu bearbeitende Registeradresse. Ab hier unterschieden sich beide Funktionen. Während in der Schreibfunktion nun der Registerwert gesendet wird, schickt die Lesefunktion ein Dummy Byte, woraufhin der Chip mit dem Inhalt des abgefragten Registers antwortet. Als letztes muss der Chip Select wieder zurückgenommen werden, sodass eine korrekte Kommunikation für den nächsten Chip sichergestellt werden kann.

Zusätzlich müssen im Rahmen der Initialisierung, wie bei CAN1, die benötigten Signale und Nachrichten definiert werden. Da CAN2 nur zur Kommunikation mit dem RFID-Reader verwendet wird, benötigt die Option 1 lediglich eine Nachricht mit zwei Signalen. Dabei handelt es sich um die auf der Karte gespeicherte Kundennummer und den Fahrmodus, welcher mit der Karte aktiviert werden soll.   
Der Ablauf zum Auslesen von Daten ist dabei unterschiedlich zu CAN1. Zum einen wird hier nicht jeden Durchlauf der Empfangsbuffer auf neue Nachrichten geprüft, sondern der CAN-Controller sendet ein Signal an den Mikrocontroller, welches dort einen Interrupt auslöst. In der dazugehörigen ISR wird dann ein CAN2-Flag gesetzt. Auf dieses reagiert der Loop mit dem Aufruf der Funktion process\_CAN2(). Diese übernimmt die Auslesen und die Verarbeitung der erhaltenen Daten. Das Auslesen von Nachrichten funktioniert dabei genau wie die Konfigurierung über SPI. Als erstes wird geprüft, ob der aufgetretene Interrupt durch eine Nachricht im Receive Buffer ausgelöst wurde. Dafür wird das CAN\_Interrupt\_Flag-Register (CANINTF) ausgelesen. Der Interrupt für RX0 Buffer 0 wird durch Bit 0 indiziert, weswegen der Registerwert entsprechend maskiert wird. Ist ein Interrupt aufgetreten, welcher auf eine neue Nachricht im Speicher hinweist, muss der Interrupt wieder zurückgesetzt werden, um bei einem erneuten Aufruf den Inhalt nicht zu verfälschen.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Anders als bei CAN1 gibt es hier leider kein CAN-Frame Objekt, aus welchem man ID und Daten auslesen kann, sondern diese Informationen müssen via SPI aus den Registern ausgelesen werden. Stattdessen wird die ID in den Registern Receive Buffer 0 Standard Identifier High (RXB0SIDH) und Receive Buffer 0 Standard Identifier Low (RXB0SIDL) gespeichert. RXB0SIDH speichert dabei die ersten Bits 10 - 3 des 11 Bit langen Identifiers, RXB0SIDL die Bits 2 – 0. Um diese nun zu einem Wert zusammenzuführen, wird der Wert aus RXB0SIDH um 3 Bits verschoben und mit den 3 fehlenden Bits verbunden. Diese Bits sind im Register RXB0SIDL an den Positionen 7 – 5 gespeichert, weswegen der Registerwert zuerst auf die Ersten 3 Bits verschoben werden muss. Diese ausgewertete ID kann nun mit der erwarteten Nachricht verglichen werden. Ist die ID korrekt, wird mit der Auswertung und Verarbeitung der Daten begonnen. Dafür wird zuerst der DLC-Wert ausgelesen, welcher angibt, wie viele Bytes in der Nachricht übertragen wurden. Diesen Wert findet man im Receive Buffer 0 Data Length Code Register (RXB0DLC). Um den korrekten DLC zu erhalten, dürfen nur die ersten vier Bit betrachtet werden, sodass der Registerwert entsprechend maskiert wird. Mit diesem DLC-Wert können nun die Daten aus dem Speicher ausgelesen werden. Diese sind byteweise in den Registern Receive Buffer 0 Data 0 bis 7 (RXB0D0…7) gespeichert. Um aus diesen Bytes einen großen Wert zu erhalten, welcher anschließend ausgewertet werden kann, werden nacheinander die Datenbytes Ein Bild, das Text, Screenshot, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.ausgelesen und höherwertige Bytes vor den bisher bereits ausgelesenen positioniert. Das wird durch Verschiebung in Bytes erreicht. Die Register bieten dabei den Vorteil, dass sie konsekutive Adressen besitzen, sodass die Startadresse inkrementiert werden kann, um die Folgebytes auszulesen. Um die einzelnen Signale aus den Daten auszulesen, wird eine Maske generiert mit welcher die Daten anschließend maskiert werden. Der korrekte Wert wird nun durch Verschiebung um das Startbit nach hinten erreicht. Für die Option 1 ist dabei nur die Kundennummer interessant. Diese wird überprüft, ob es sich dabei um eine Herstellernummer oder einen Kunden handelt. Handelt es sich um eine Herstellernummer wird ein Master Modus aktiviert, in welchem es möglich ist, vertiefte Einstellungen und Freischaltungen vorzunehmen. Dieser ist bis zum nächsten Neustart aktiv.  
Sendefunktionen sind für CAN2 keine vorgesehen, da stand jetzt nur der RFID-Reader auf CAN2 sendet.

Die meisten Daten, die über CAN erhalten werden, sind dafür vorgesehen als visuelle Daten für den Kunden dargestellt zu werden. Die erste Variante zur Darstellung von Daten ist über eine Webseite. Im Folgenden soll daher aufgezeigt werden, wie ein Webserver gehostet wird und welche Funktionalitäten dieser bietet. Der ESP32 bietet zwei Unterschiedliche Wlan-Funktionen. Zum einen kann er als Client betrieben werden sich mit einem Wlan-Netzwerk verbinden. Zum anderen ist er auch in der Lage selbst ein eigenes Netzwerk aufzuspannen und so als Accesspoint für andere Wlanfähige Geräte zu dienen. Dieser Modus wird für dieses System genutzt. Dafür wird im Setup die Initialisierung des Wlan-Chips durchgeführt. Über die Funktion softAPConfig() aus der WiFi.h-Library wird der ESP als Accesspoint gestartet. Die übergebenen Parameter legen dabei die IP-Adresse und alle damit verbundenen Informationen fest. Das dient dazu, dass der Mikrocontroller bei jedem Neustart dieselbe IP-Adresse behält. Neben den IP-Informationen müssen auch die Verbindungsinformationen wie SSID und das Passwort festgelegt werden. Diese werden Initial für jedes Kart identisch im Persistenten Speicher des ESP abgelegt, sind aber später individuell änderbar.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.  
Damit ist die Konfiguration des Wlanmoduls abgeschlossen und der Webserver kann initialisiert werden. Für diesen werden die einzelnen Seiten festgelegt, bevor er gestartet werden kann. Jede Seite benötigt dabei eine eigene URL und einen definierten handler, welcher sowohl die graphische Oberfläche als auch die Funktionalität zur Verfügung stellt. Über die Funktion begin() wird der Webserver gestartet. Zusätzlich wurde für zwei Seiten das HTTP-Protokoll definiert.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Als nächstes sollen diese Seiten, deren Aufbau und Funktionen näher betrachtet werden. Dabei soll es weniger um den Aufbau der HTML-Seiten gehen als um die Umsetzung der Funktionen, welche über die Webseiten realisiert werden.   
Jede Seite besitzt eine Abfrage, ob die Funktionalität des Wlanmoduls freigegeben ist. Diese erzeugt für jede URL eine Webseite ohne Funktionen. Hintergrund ist, dass der ESP immer als Access Point arbeitet und den Webserver hostet, damit es für den Hersteller möglich ist auf Einstellungen zuzugreifen, auch wenn die Funktion für den Kunden deaktiviert ist. Dadurch, dass tatsächlich eine Seite ohne Funktionen erstellt wird, ist es auch nicht möglich über Umwege auf Funktionalitäten, welche eigentlich gesperrt sein sollten zuzugreifen. Ein Bild, das Text, Screenshot, Schrift enthält.

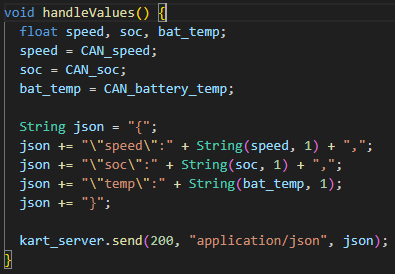
KI-generierte Inhalte können fehlerhaft sein.

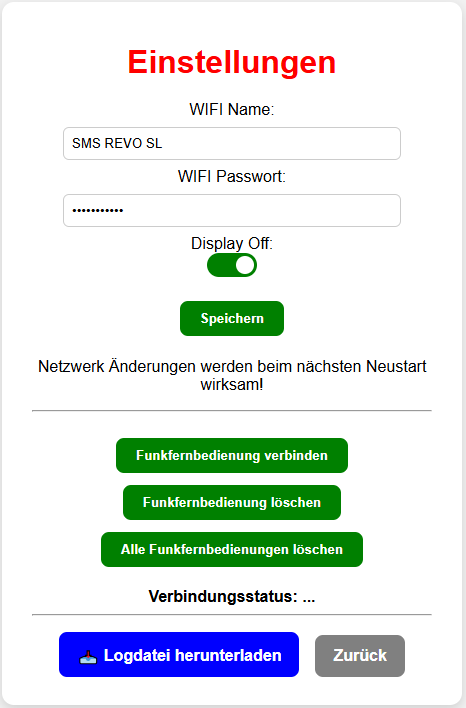
Ist die Funktionalität freigegeben, ist die erste Seite, welche der Kunde betritt, die Root Seite. Diese stellt noch keine Funktion zur Verfügung, welche in den Anforderungen definiert wurde, sondern liefert lediglich eine Übersicht und ein Menü, um die beiden anderen Seiten mit Funktionen zu erreichen. Anhand dieser Seite soll die allgemeine Funktionsweise gezeigt werden. Die Seite wird über HTML aufgebaut. Darüber werden im Abschnitt <style> das Aussehen der Seite und der Elemente auf der Seite definiert. Im Abschnitt <body> werden die Elemente auf der Seite hinzugefügt und mit entsprechenden Funktionen versehen. Im Beispiel der Hauptseite besteht die Seite aus einer Überschrift im Style <h1>, welche den Namen des Karts zeigt. Darunter sind zwei Buttons platziert vom Typ a mit entsprechenden Farben platziert. Die Funktion des Knopfes wird über den Parameter href definiert, indem dort die URL angegeben wird, auf die bei Drücken des Knopfes verlinkt wird. So führt Drücken des Grünen Knopfes auf die Seite /livedaten, der Blaue Knopf verlinkt auf Seite /einstellungen. Die Beschriftung des Knopfes wird hinter dem Link angegeben. Die Darstellung auf der Seite passiert über die Funktion send() des Webserver Objekts. Diese sendet eine Antwort auf eine HTTP-Anfrage des Webservers. Dabei werden der Funktion verschiedene Parameter übergeben. Der erste Parameter ist der HTTP-Status Code, in diesem Fall 200 für OK, danach wird der Typ des Inhalts angegeben, im Fall dieser Seite wird ein HTML-Text gesendet und schließlich der eigentliche Inhalt der Seite als HTML-Text.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

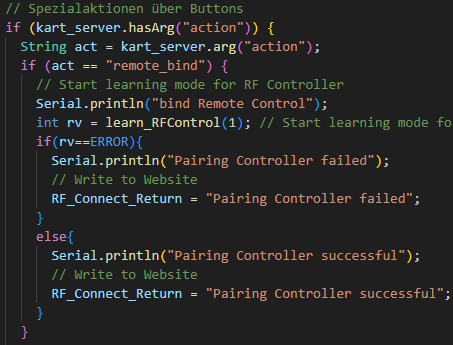
Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.Das Verarbeiten des Webservers und das Stellen der HTTP-Anfragen an den Handler, übernimmt der Loop. Dort wird zyklisch die Funktion kart\_server.handleClient() aufgerufen, welche nach offenen HTTP-Requests sucht und diese an die Handler weiterreicht, welche für die jeweilige URL definiert wurden.  
Die Seite /livedaten, welche über die Hauptseite erreicht werden kann, stellt die Daten vom Kart, welche per CAN an die Erweiterungsplatine gesendet werden, visuell für den Nutzer dar. Im aktuelle Stand der Webseite sind lediglich 3 Werte dargestellt, um die Funktionsweise und das Konzept zu zeigen. Die Werte sollen sich automatisch aktualisieren, ohne dass der Kunde dies händisch einleiten muss. Die Webseite hat dafür ein in den HTML-Text eingefügten Skript-Teil. Dort läuft ein JavaScript Programm, welches alle 200ms die Werte vom ESP32 abfragt und in den Feldern auf der Webseite darstellt. Das Skript beinhaltet eine asynchrone Funktion, welche die Werte von der Seite /values mit der Funktion fetch() abfragt. Bei der Seite /values handelt es sich um eine json-Application, welche die Werte für die darzustellenden Werte über die Funktion response.json zurückgibt. Aus diesem json-Datensatz können nun die Werte für Geschwindigkeit, SOC und Temperatur ausgelesen werden und mit der entsprechenden Einheit versehen werden. Diese Werte werden im normalen HTML-Body als Inhalt der Datenboxen dargestellt.  
  
Damit das Auslesen der Werte von der Seite /values funktioniert, muss diese eine json-Application zur Verfügung stellen. Diese liest als erstes die Variablen vom ESP. Dabei handelt es sich um globale Variablen, welche von den Auswertungsfunktionen von CAN beschrieben werden und von allen Funktionen, die diese Daten darstellen wollen, ausgelesen werden können. Diese Daten werden in einen json-String verpackt und als Applikation an der Webserver gesendet.

Eine weitere Funktionalität, welche über die WLAN-Schnittstelle und den Webserver geschaffen wird, ist die Änderung von Einstellungen sowohl für den Kunden als auch den Hersteller. Dafür ist die Webseite /einstellungen vorgesehen. Über diese soll es möglich sein alle wichtigen Einstellungen vorzunehmen, ohne dass der Kunde darüber neue Funktionen freischalten kann. Dazu gehört zum Beispiel das Festlegen eines WLAN-Namens und eines neuen Passworts. Das ist vor allem für Vereine wichtig, welche mehrere Karts besitzen, um diese anhand des Netzwerknamens auseinanderhalten zu können. Auch das Passwort kann über die Einstellungen geändert werde. So kann jeder Verein sein eigenes Passwort festlegen, um zu verhindern, dass unbefugte mit dem Standardpasswort, welches bei allen Karts voreingestellt ist, auf fremde Karts zugreifen können. Diese Änderungen sollen nicht automatisiert übernommen werden, sondern erst durch Drücken des Speichern Knopfes in den Persistenten Speicher geschrieben werden. Aufgrund der bereits bestehenden Wlan-Verbindung werden diese Änderungen erst nach einem Neustart wirksam. Anders als bei den bisherigen Seiten wird der Aufbau der Seite nicht als HTML-Body, sondern als HTML-Formular realisiert. Dieses sendet einen HTTP-Post, sobald Daten aus dem Formular, wie zum Beispiel die Eingaben für Wlan SSIO und Passwort, vom Client an den Server übertragen werden sollen. Dieser HTTP-Post wird im Mikrocontroller vom Handler abgefragt über die Funktion method(). Wenn in den Feldern Daten eingetragen sind, werden diese im Non-Volatile Storage des ESP32 gespeichert.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Dasselbe gilt für den Schalter für Display Off. Dieser soll einen Modus aktivieren, in welchem das Display ausgeht, sobald das Kart losfährt, um zu verhindern, dass es den Fahrer irritiert oder verunsichert. Auch dieser Schalter wird bei einem HTTP-Post ausgelesen und der Wert entsprechend gespeichert. Der HTTP-Post wird dabei durch Drücken des Speicherknopfes gesendet.  
Neben der Änderung von Einstellungswerten soll es auch möglich sein Fernbedienungen mit dem Kart zu verbinden. Dafür wurden drei Buttons zum Verbinden und Löschen einzelner Fernbedienungen und dem Löschen aller Fernbedienungen vorgesehen. Die Funktionsweise dieser Knöpfe soll anhand der Verbindung einer Fernbedienung mit dem System erklärt werden. Jeder Knopf wird bei Senden eines HTTP-Post ausgewertet. Über das Übertragene Argument kann der gedrückte Knopf identifiziert werden. Darüber ist nun bekannt, welcher Lern oder Löschmodus aktiviert werden soll. Für den ersten Knopf zum Anlernen einer Fernbedienung ist das Lernmodus 1, für die beiden Löschmodi sind die Modi 5 und 6 vorgesehen. Die Aktivierung des Lernmodus findet dabei über die bereits betrachtete Funktion learn\_RFControl() statt.   
Um dem Nutzer eine Rückmeldung über den Status des Verbindungsversuchs zu geben, wird unterhalb der Knöpfe die Variable RF\_Connect\_Return angezeigt. Tritt während dem Versuch eines Verbindungsaufbaus ein Fehler auf, wird diese Variable beschrieben. Ebenso wird darüber ein erfolgreicher Abschluss der Funktion angezeigt. Die Anzeige dieser Variable passiert wieder über ein in den HTML-Code eingefügtes Java Skript. Dieses fragt den Wert von der Webseite /RF\_Connect\_Return ab und stellt ihn der Seite zur Anzeige zur Verfügung. Dabei handelt es sich bei der URL /RF\_Connect\_Return nicht wie bei der Livedaten Seite um eine json-Application, sondern hier reicht es aus, den Wert als reinen Text zu übertragen, da es nur um den Wert einer Variablen geht.

Die letzte Funktionalität, welche über diese Webseite realisiert wird, ist der Download der Fehler-Logdatei. Der dafür vorgesehene Button funktioniert genau wie die beiden Knöpfe auf der Hauptseite und verlinkt lediglich zur Webseite /DownloadLog. Diese Seite überprüft automatisch beim Aufruf der Seite, ob die Logdatei vorhanden ist und beginnt den Download, wenn die Datei gefunden wurde. Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Aktuell wird als Speicherort für die Logdatei der Persistente Speicher des Mikrocontrollers genutzt. Zukünftig soll dafür der auf der Platine vorgesehene Flashspeicher genutzt werden, da hier die Daten auch nach dem Flashen eines neuen Softwarestands erhalten bleiben. Dasselbe gilt für das Speichern der Einstellungen. Dort kommt vor allem der Vorteil zum Tragen, dass kritische Daten wie das Wlanpasswort im SPI-Flash verschlüsselt abgelegt werden, während sie im NVS des ESP als Klartext gespeichert sind. Leider war zum Zeitpunkt der Bestellung der Platine das Bauteil nicht mehr verfügbar, weswegen in der ersten Version der Platine die Speicherung der Persistenten Daten noch im ESP selbst stattfindet.

Die Eingabe der gerade angesprochenen Einstellungen findet dabei über die Seite /hersteller statt. Hierrüber soll es für den Hersteller möglich sein alle Hauptfunktionen der Option 1 zu aktivieren oder zu deaktivieren. Daher darf es nicht möglich sein, dass Kunden diesen Bereich erreichen können. Zu diesem Zweck wird der Master Modus genutzt, welcher nur über die Verwendung der RFID-Karte mit Herstellerkundennummer erreicht werden kann. Daher ist diese Seite auch die einzige, welche immer aktiv ist, selbst wenn die Funktionalität der Wlan-Schnittstelle für den Kunden deaktiviert wurde. Für jede große Einstellung ist ein Schiebebutton wie auf der /einstellungen-Seite vorgesehen, welcher zusätzlich die aktuelle Einstellung anzeigt. Anders als auf der vorherigen Seite, sollen die Werte aber gespeichert werden, sobald der Knopf verändert wird. Dafür wird ein AJAX (Asynchronous JavaScript And XML) Post Request verwendet. Damit findet die Datenaustausch wie bei einem normale Post Request statt, nur mit dem Unterschied, dass die Seite nicht neu geladen werden muss. Der Ablauf dieses Post Request und das Speichern der Daten soll im Code anhand zweier Einstellungen gezeigt werden. Der Post Request übergibt dabei zwei Argumente. Key ist der Name der Einstellung, beispielsweise Display\_Enable, zur Freischaltung der Display-Funktion. State ist der Wert des Schalters. Dieser Wert wird nun der entsprechenden Einstellung übergeben und im NVS des ESP gespeichert.

Ein Bild, das Text, Screenshot, Display, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Die Darstellung der Schalter übernimmt wie bei den vorherigen Seiten ein Formular. Dabei wird die Anzeige über die Variablen displayEnable und rfEnable sichergestellt. Diese Anzeigeoption sorgt dafür, dass der Schalter Grün hinterlegt wird, sobald der Wert im Speicher des ESP auf True gesetzt wird. Diese Variablen werden dafür bei jedem laden der Webseite auf den aktuellen Wert des Speichers gesetzt.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Für die Funktionalität des Speicherns wird auch hier wieder ein JavaScript im HTML-Code eingesetzt. Dieses besteht aus zwei Abschnitten. Zum einen wird hier die Funktion sendToggle() realisiert, welche den HTTP-Post mit den entsprechenden Argumenten an den Server sendet. Zum anderen wird für jeden Schalter ein Event Listener hinzugefügt, welcher auf Änderungen des jeweiligen Schalters wartet. Entdeckt einer dieser Listener eine Veränderung bei seinem Schalter, ruft er die Funktion sendToggle mit den passenden Argumenten auf und sendet damit den HTTP-Post für den jeweiligen Schalter an den Server.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.Damit die Schalter ihre Einstellungen speichern können, muss dieser Speicher vorher definiert und die entsprechenden Variablen angelegt werden. Dafür wird im Setup die Funktion init\_storage() aufgerufen. Diese Funktion definiert den Speicherbereich für die Einstellungen und ermöglicht den Zugriff auf den Speicher über ein Objekt. Das passiert über die Funktion begin(). Diese Initialisiert den Speicherbereich „settings“. Durch den Parameter False wird er als beschreibbar angelegt. Sollte der Speicher bereits existieren, wird der Speicherbereich geöffnet. Danach legt die Funktion alle benötigten Variablen im Speicher an. Das wird über die Funktion put…() mit dem entsprechenden Datentyp erreicht, welcher angelegt werden soll. Der Funktion wird dabei ein Key-Value Paar übergeben. Der Wert wird so abgespeichert, dass er über den Schlüssel wieder ausgelesen werden kann. Damit die Variable nicht überschrieben wird, sollte sie bereits existieren, wird vorher aus dem Speicherbereich gelesen. Das passiert über die Funktion get…(), in Kombination mit dem auszulesenden Datentyp. Dabei wird der Key definiert, unter welchem die Variable abgespeichert wurde. Der zusätzlich angegebene Wert ist der Rückgabewert, sollte die Variable im Speicher noch nicht existieren. Die Variable soll also nur beschrieben werden, wenn der Rückgabewert der Funktion dem Wert -1 entspricht. Da es sich bei den zu speichernden Variablen um boolesche Werte handelt, kann der gespeicherte Wert nie den Wert -1 annehmen, außer die Variable existiert noch nicht. Auf diese Weise werden für folgende Einstellungen Variablen angelegt:

|  |  |
| --- | --- |
| RF\_enable | Aktivierung für Funkempfänger Funktionen |
| Display\_enable | Aktivierung für Display Funktionen |
| WiFi\_enable | Aktivierung für WiFi Funktionen |
| RFID\_enable | Aktivierung für RFID-Funktionen |
| LED\_enable | Aktivierung für Status-LED Funktionen |
| RF\_CAN\_en | Aktivierung für Funksignale via CAN (Kompatibilität) |
| Display\_off | Aktivierung Display Aus Modus beim Fahren |
| WIFI\_Name | Netzwerkname für Wlan |
| WIFI\_Password | Netzwerkpasswort für Wlan |

Display erklärung

# Test und Inbetriebnahme

# Fazit und Ausblick

## Zusammenfassung der Arbeit

## Mögliche Weiterentwicklungen

# Literaturverzeichnis

# Bildverzeichnis