

Technical Report

Implementierung einer flexiblen Plattform zur Montage von Hardware im Bereich des Autonomen Fahrens

Tobias Poppe
Matrikel-Nr. 10012931

Stand 28. Oktober 2021

Betreuung: Prof. Dr.-Ing. Anselm Haselhoff / Jan Kronenberger

Institut Informatik

Hochschule Ruhr West

Inhaltsverzeichnis

1	Einleitung	4
2	Mechanik	5
2.1	Trägerplattform	5
2.2	Kameras	7
2.3	LiDAR	9
2.4	GNSS Antennen	10
2.5	Dachkasten	11
2.5.1	Kabeldurchführung	12
2.5.2	Innenausbau	13
3	Elektrik	15
3.1	Stromversorgung	15
3.2	Video und USB-Anschluss	15
3.3	Steuergerät	15
4	Hardware Steuergerät	18
4.1	Batteriemanagment-Platine	18
4.2	Hauptplatine	20
4.2.1	ESP32	22
4.2.2	PWM-IC PCA9685	25
4.2.3	ADC-IC MCP3428	26
4.2.4	MOSFETs	27
4.2.5	Display	27
4.2.6	Restliche Beschaltung	28
5	Software	29
5.1	Bedienung	29
5.2	ROS-Serial-Node	30
5.3	Betriebszustand des Steuergeräts	32
5.4	Programmablaufplan	33
5.5	Auslesen von Messwerten	34
5.5.1	Berechnungen der ADC-Werte	34
5.6	Berechnungen der LTC2943 Werte	35
5.6.1	Berechnung der Spannung	35
5.6.2	Berechnung der Stromstärke	36
5.6.3	Berechnung der Temperatur	36
5.6.4	Berechnung des Ladezustands	36
6	Anhang	38
	Literaturverzeichnis	54
	Abbildungsverzeichnis	56

Abstract

Der Report beschreibt das im Rahmen des Moduls "Projekt 2" und des Projekts "CAMO.NRW" entwickelten modularen Dachträgers zur Entwicklung und Lehre von Assitenzsystemen im KFZ-Bereich.

Es wird sowohl auf den Mechanischen als auch den Elektrischen Aufbau eingegangen.

Weiter wurde ein Steuergerät entwickelt, um die im und am Dachträger verbaute Elektronik überwachen und steuern zu können. Die für dieses Embedded-System benötigte Firmware wurde erstellt und wird ebenfalls in diesem Report erläutert.

Alle erstellten Dateien (Bauteile, Platinen sowie Firmware) sind in einem GitHub-Repository unter <https://github.com/topoppe/camo.nrw.dachbox> veröffentlicht.

1 Einleitung

Im Rahmen des EFRE-geförderten Projektes CAMO.NRW forschen Mitarbeiter der Hochschule Ruhr West zusammen mit Kollegen der Bergischen Universität Wuppertal und der Bergischen Struktur- & Wirtschaftsförderungsgesellschaft an der Zukunft der automatisierten Mobilität. An der Hochschule bestand schon eine Experimentalplattform bestehend aus verschiedenen Kameras, Radarsensoren, LiDAR und mehreren GNSS.

Jene Plattform war im Kofferraum eines PKWs verbaut, jedoch ohne die Beachtung gängiger Sicherheitsvorschriften und Normen.

Durch die komplexe Einbauweise war die gesamte Plattform an einen einzelnen PKW gebunden. Im Rahmen dieser Arbeit wurde die vorhandene Sensortechnik auf einem austauschbaren Geräteträger montiert. Hierbei wurde sowohl auf Sicherheitsvorschriften als auch auf die einfache Montagemöglichkeit auf verschiedenen PKW geachtet.

Der entstandene Geräteträger kann nun sowohl mobil als auch stationär, auf einem beliebigen PKW montiert oder in einer Laborumgebung ohne PKW genutzt werden.

2 Mechanik

2.1 Trägerplattform

Der Geräteträger bietet die Möglichkeit einer Dachmontage. Hierfür muss das Trägerfahrzeug lediglich über die Aufnahme für einen Dachgepäckträger verfügen. Die maximale Dachlast des Trägerfahrzeugs muss für die Montage mindestens 75 kg betragen.

Ausschlaggebender Vorteil eines solchen Systems ist die Möglichkeit der schnellen und unkomplizierten Demontage und erneuter Montage. Die Montage ist auch auf einem anderen PKW möglich.

Nach einer Marktrecherche über verschiedene Trägersysteme fiel die Wahl auf das Modell "Canyon XT" der schwedischen Firma Thule (Abbildung 2.1, Seite 6).

Um eine möglichst große Fläche für Aufbauten zur Verfügung zu haben wurde der Dachträger erweitert mit einer "Extension" der Firma Thule (Abbildung 2.2, Seite 6).

Durch den Einsatz der Erweiterung erhöht sich die Länge um 51 cm auf die Maße in Tabelle 2.1 auf Seite 5.

Um eine einfache und flexible Montage verschiedener Sensoren zu ermöglichen, wurden an den Außenkanten des Dachträgers Vierkantprofile aus Aluminium nach Industriestandard montiert (20 mm x 20 mm, 5 mm Nut, Typ I). Die Länge der Profile beträgt seitlich 100 cm, an der Vorderseite 95 cm und an der Rückseite 90 cm.

	Länge	Breite	Höhe
Außenmaß	178 cm	104 cm	15 cm
Innenmaß	155 cm	99 cm	13 cm

Tabelle 2.1: Maße des Dachkäfigs Thule Canyon XT inkl. Extension.



Abbildung 2.1: Der Dachkäfig Canyon XT von Thule [10].

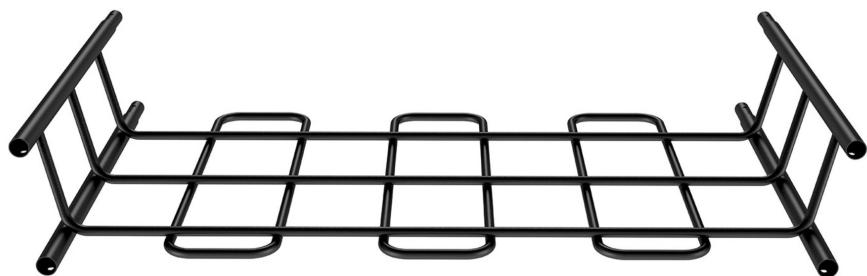


Abbildung 2.2: Die Erweiterung Thule Canyon Extension XT von Thule[9].

2.2 Kameras

Aus dem Vorgängerprojekt waren bereits mehrere Kameras vom Typ BlueFox (3-2, 2024a) der Firma Matrix Vision vorhanden (Abbildung 2.3, Seite 7). Diese Kameras haben eine Auflösung von 1936x1216 Pixeln und kommunizieren mit über einen USB3-Port. Außerdem ist es möglich, über einen Triggereingang mehrere Kameras zu synchronisieren. Die hierfür benötigte Synchronisierungsbox ist im Dachkasten verbaut.



Abbildung 2.3: BlueFOX3-2 Kamera von Matrix Vision[2].

Zur Montage der Kameras an den Vierkantprofilen wurde eine Halterung entworfen und mittels Laser aus 5mm starken POM-Kunststoff gefertigt. Der Halter bietet neben der sicheren Befestigung die Möglichkeit, jede Kamera individuell um die Z-Achse des Fahrzeugs zu drehen. Somit lassen sich schnell und zuverlässig verschiedene Blickwinkel der jeweiligen Kamera einstellen (Abbildung 2.4, Seite 7).

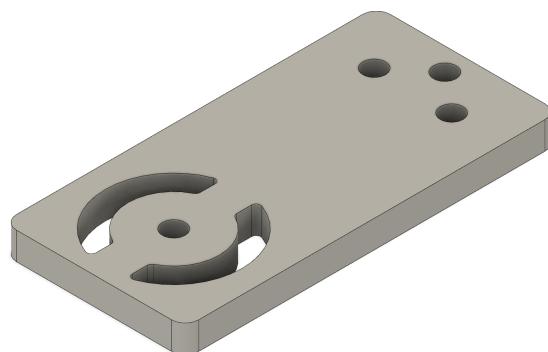


Abbildung 2.4: Kamerahalter aus POM.

Zum Zeitpunkt der Erstellung dieses Dokuments waren 4 Kamerasysteme verbaut, 2 auf dem vorderen Vierkantprofil in Fahrtrichtung und jeweils eine auf den seitlichen Vierkantprofilen in einem Winkel von ca. 45° zur Fahrzeughöngsache.

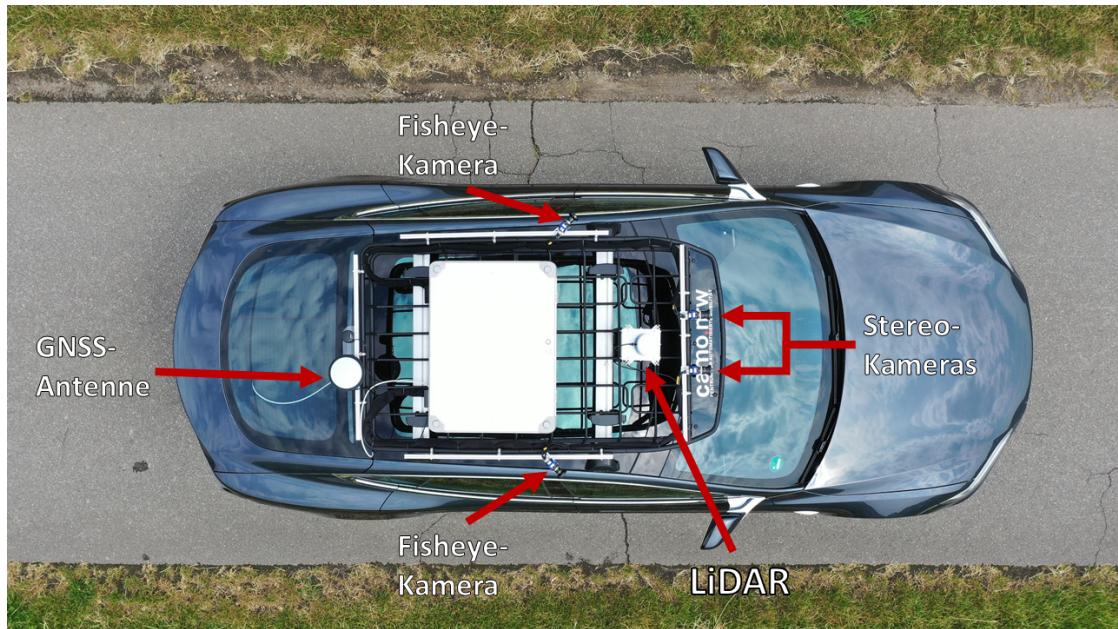


Abbildung 2.5: CAMO-Demonstrator von oben [11].

2.3 LiDAR

Ebenfalls aus einem Vorgängerprojekt war bereits ein LiDAR vom Typ Puck 16 Hi-Res der Firma Velodyne vorhanden. Dieser LiDAR Sensor misst simultan in 16 Ebenen in einem Höhenbereich von 20° um die Rotationsachse.

Um eine Abschattung durch andere Sensoren auf dem Geräteträger zu vermeiden sowie um verschiedene Blickfelder einstellen zu können, wurde der Sensor auf einem Turm mittig auf dem Geräteträger verbaut (Abbildung 2.6, Seite 9). Der Turm ist aus den selben Vierkantprofilen wie die umlaufende Halterung gebaut. Die Einstellung der Neigung zur Fahrzeugquerachse ist mit wenigen Handgriffen möglich. Die Montageplattform auf der Oberseite des Turms wurde aus 5mm starken POM gefertigt. Nuten in der Plattform sichern den Sensor vor ungewolltem Drehen um die Z-Achse.

Der LiDAR-Sensor benötigt neben dem Sensor selbst noch eine GNSS-Antenne. Diese wurde auf einem Halter am rückseitigen Vierkantprofil angebracht.

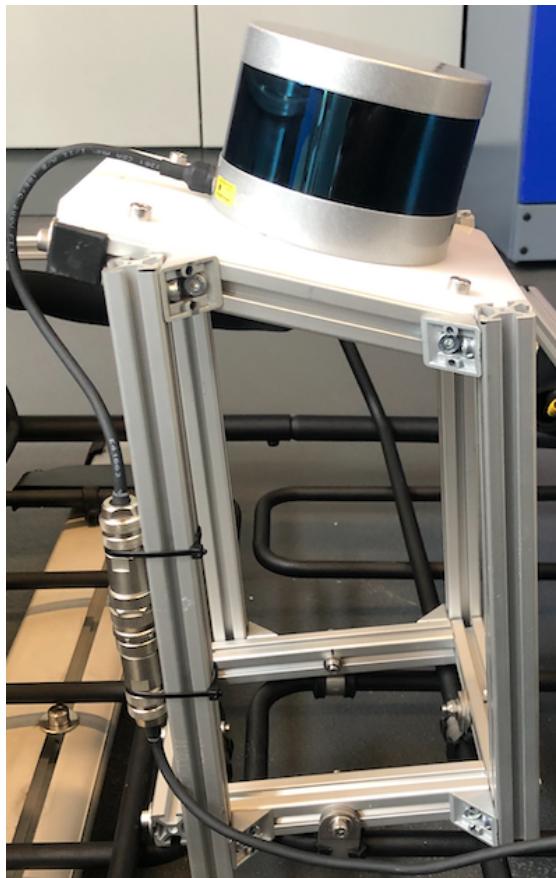


Abbildung 2.6: LiDAR-Sensor montiert auf Turm.

2.4 GNSS Antennen

Um möglichst exakte Daten zur Positionierung des Fahrzeugs erhalten zu können, ist das Gesamtsystem mit einer GNSS-Antenne vom Typ Vexxis GNSS-850 der Firma Novatel ausgestattet.

Die Antenne ist gemeinsam mit der GNSS-Antenne des LiDAR-Sensors auf dem rückseitigen Vierkantprofil angebracht. Hierzu wurde eine Halterung aus Acrylglas gefertigt (Abbildung 2.7, Seite 10).



Abbildung 2.7: GNSS-Antennen.

2.5 Dachkasten

Um Empfindliche Elektronikkomponenten geschützt auf der Trägerplattform unterbringen zu können, wurde auf dem Dachkäfig eine Kiste "K470" der Firma Zarges montiert (Abbildung 2.8, Seite 11).

Diese Kiste ist im Deckelbereich durch eine eingeschäumte Dichtung IP65 geschützt.



Abbildung 2.8: K470 IP65 von der Firma Zarges [13].

	Länge	Breite	Höhe
Außenmaß	60 cm	80 cm	41 cm
Innenmaß	55 cm	75 cm	38 cm

Tabelle 2.2: Maße des Dachkastens K470 von Zarges.

2.5.1 Kabeldurchführung

Im rückseitigen Bereich wurden zwei Kabeldurchführungen eingebaut, die gleichzeitig der Belüftung des Dachkastens dienen (Abbildung 2.9, Seite 12 und Abbildung 2.10, Seite 13).

Die Kabeldurchführungen wurden im Rahmen der vorliegenden Arbeit individuell in einem CAD-System gezeichnet und mittels FDM-Druck hergestellt.

Auf der Innenseite wurde an einer 25mm Öffnung ein Lüfter angebracht. Dieser saugt die Luft oberhalb des eigentlichen Kabelauslasses an und drückt die erwärmte Luft durch die große Öffnung für die Kabeldurchführung wieder nach außen. Auf diese Weise ist gewährleistet, dass beim Betrieb im Regen kein oder nur sehr wenig Regenwasser in das Innere des Dachkastens gelingen kann.

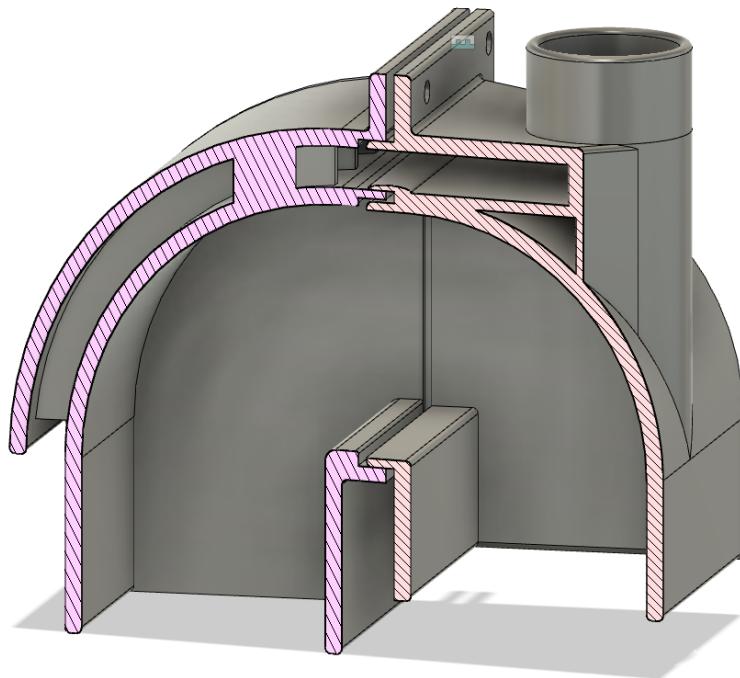


Abbildung 2.9: Querschnitt der Kabeldurchführung.

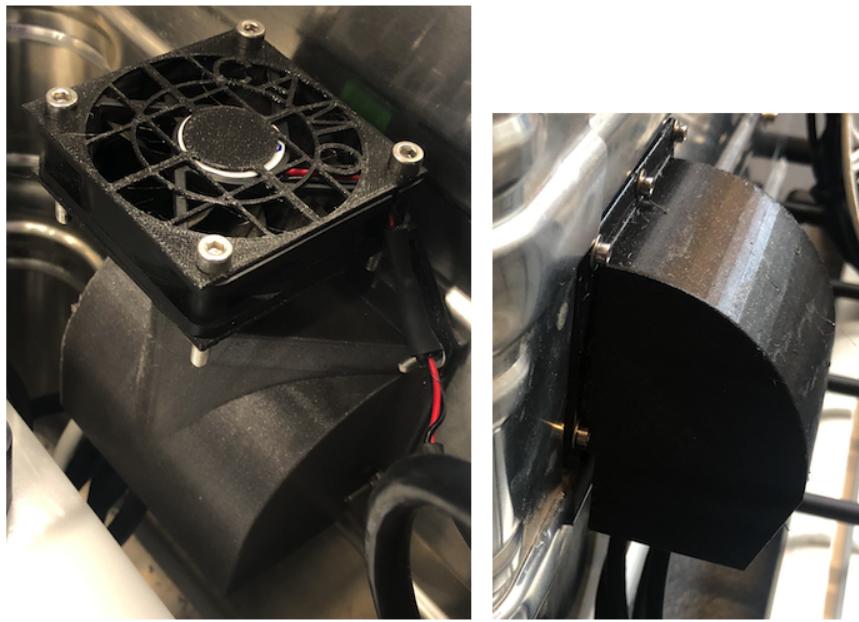


Abbildung 2.10: Kabeldurchführung innen und aussen mit montiertem Lüfter.

2.5.2 Innenausbau

Im Inneren des Dachkastens wurde auf Höhe der unteren Versteifung ein Zwischenboden mittels Vierkantprofilen und lasergeschnittenen POM-Platten eingezogen. Auf diesem Zwischenboden wurden folgende Geräte mit Hilfe von FDM-gedruckten Halterungen montiert:

- Novatel IMU-IGM-A1 Sensorbox
- PPM 40xx GNSS Sensorbox
- Anschlussbox des Velodyne LiDAR
- Synchronisierungbox der Kameras
- HDBaseT HDMI und USB Transceiver
- DC/DC-Wandler Batteriespannung auf 5 V
- DC/DC-Wandler Batteriespannung auf 12 V
- Sicherungskasten
- LiFePo Batterie 13,4 V, 30 AH

- Hauptsicherung 40 A der Batterieversorgung

Weiter ist an der Rückwand des Dachkastens ebenfalls mittels Vierkantprofilen der Hauptrechner vom Typ Nuvo-7160GC befestigt.

Unter dem Zwischenboden ergibt sich ein Stauraum für überschüssige Kabellängen. Der gesamte Innenausbau ist in Abbildung 2.11, Seite 14 dargestellt.

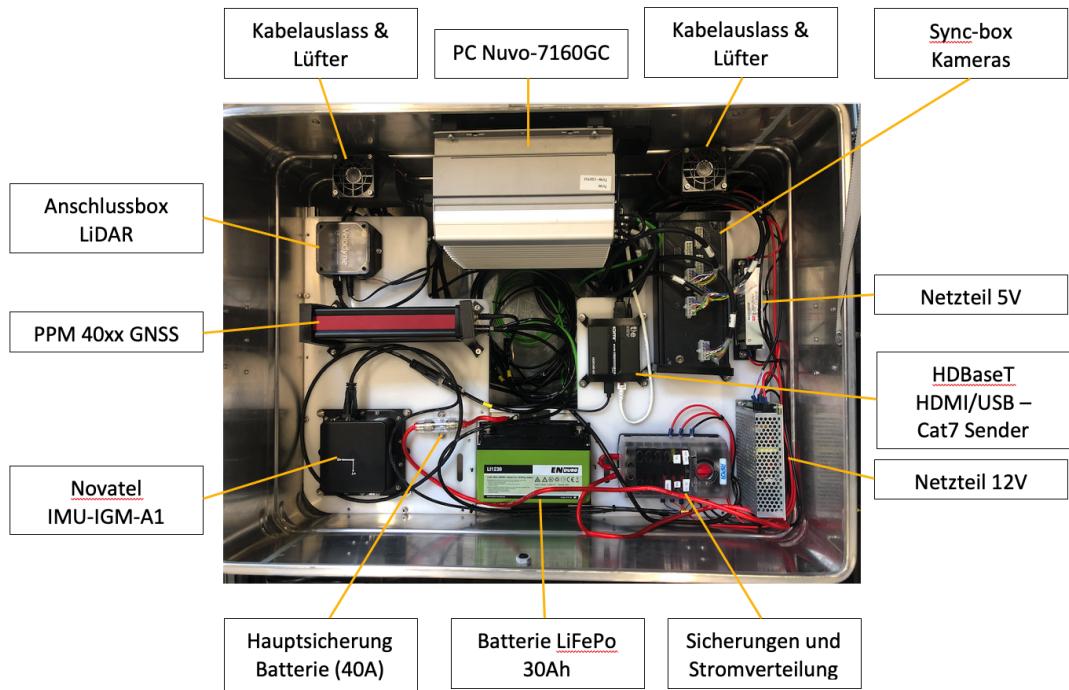


Abbildung 2.11: Innerer Aufbau Dachkasten.

3 Elektrik

3.1 Stromversorgung

Innerhalb der Zarges-Kiste wurde eine LiFePo Batterie mit einer Kapazität von 30 AH verbaut. Durch diese Batterie wird das System während der Fahrt mit Energie versorgt. Die Laufzeit beträgt ca. 4 Stunden.

Die Stromversorgung ist mit einer Hauptsicherung 40 A direkt an der Batterie sowie mit einzelnen Sicherungen für jeden Sensor in einem Sicherungskasten abgesichert. Über externe Anschlüsse ist es möglich, die Batterie zu laden sowie das System über ein externes Netzteil mit Strom zu versorgen. Somit ist ein dauerhafter Betrieb innerhalb einer Laborumgebung oder bei Vorführungen auf Messen möglich. Der Akkuladung ist allerdings nicht möglich.

3.2 Video und USB-Anschluss

Um eine Einführung des Videosignals des PC sowie USB-Leitungen in das Trägerfahrzeug wurde in HDBase-T Transceiver innerhalb des Dachkastens verbaut. Mit diesem Transceiver ist es möglich, ein HDMI-Signal in Full-HD Auflösung sowie 2 USB2.0 Signale über ein einzelnes CAT7 Ethernetkabel zu übertragen.

Das Ethernet-Kabel kann durch die Dichtung der Heckklappe des Trägerfahrzeugs in das Fahrzeugginnere verlegt werden, um so einen Zugriff auf den PC während der Fahrt zu gewährleisten.

Die maximale Länge des Ethernetkabel beträgt ca. 70 Meter, sodass auch bei Vorführungen das Videosignal des PC auf einen Beamer oder ähnliche Präsentationsgeräte übertragen werden kann.

3.3 Steuergerät

Innerhalb der Kiste wurde ein eigens entwickeltes Steuergerät zur Überwachung des Systems sowie zur Steuerung der Stromversorgung und Belüftung verbaut. Der interne Stromlaufplan des Steuergeräts ist in Abbildung 3.1, Seite 16 dargestellt.

Neben den beiden in Kapitel 4 beschriebenen Platinen besteht das Steuergerät aus 3 Hochstrom Relais mit einem Schaltstrom von 80 Ampere sowie aus 2 Anschlussbuchsen. Die Anschlussbuchsen sind von außen erreichbar. Über sie kann der interne Akku geladen werden sowie das komplette System durch eine externe Stromversorgung versorgt werden. Das hierfür verwendete Netzteil muss bei einer Spannung von 12 V DC mindestens einen Strom von 25 Ampere bereitstellen können.

Das Steuergerät selbst ist ebenfalls von Außen erreichbar. Neben einem E-Ink-Display zur Anzeige des Systemstatus sind 5 Taster verbaut, über die zum Beispiel das komplette System von aussen gestartet werden kann. Somit ist es im Normalbetrieb nicht notwendig, die Kiste zu öffnen.

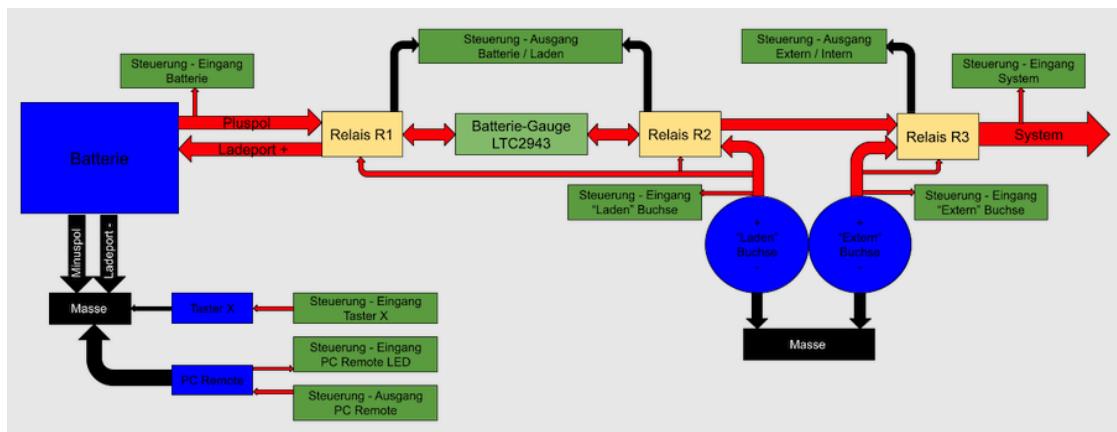


Abbildung 3.1: Stromlaufplan des Steuergeräts.

Anschluss	Relais 1	Relais 2	Relais 3
Klemme 30, Common:	LTC "Batterie"	LTC "System"	System
Klemme 87a, NC	Batterie Plus	Relais 3 Klemme 87	Buchse "Externe Stromversorgung"
Klemme 87, NO	Batterie Ladeanschluss	Buchse "Ladeanschluss" +	Relais 2, Klemme 87a

Tabelle 3.1: Anschlussbeschreibung Relais.

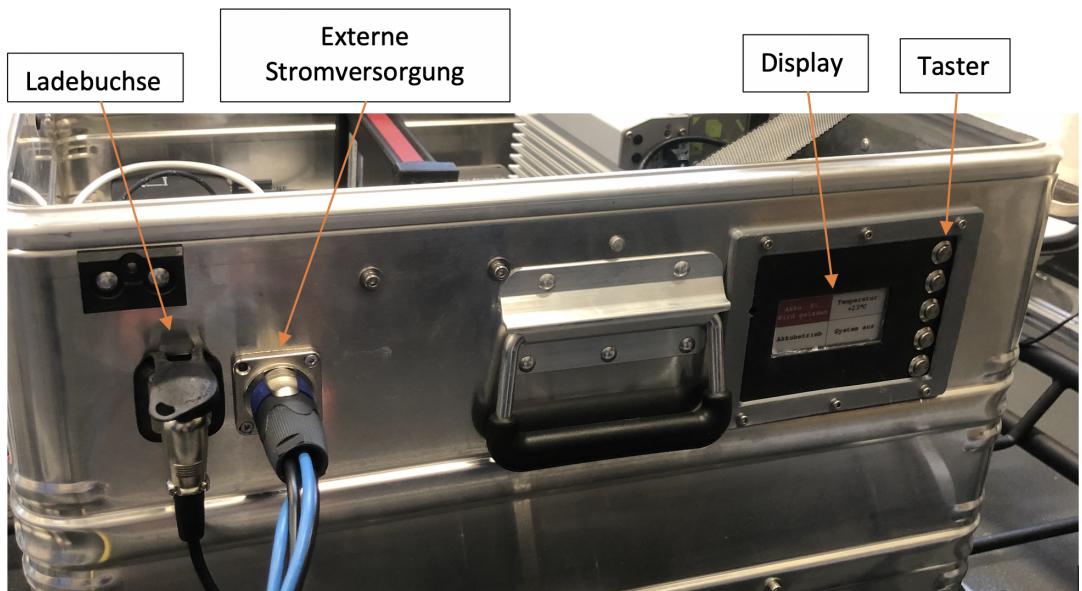


Abbildung 3.2: Außenansicht Steuergerät.

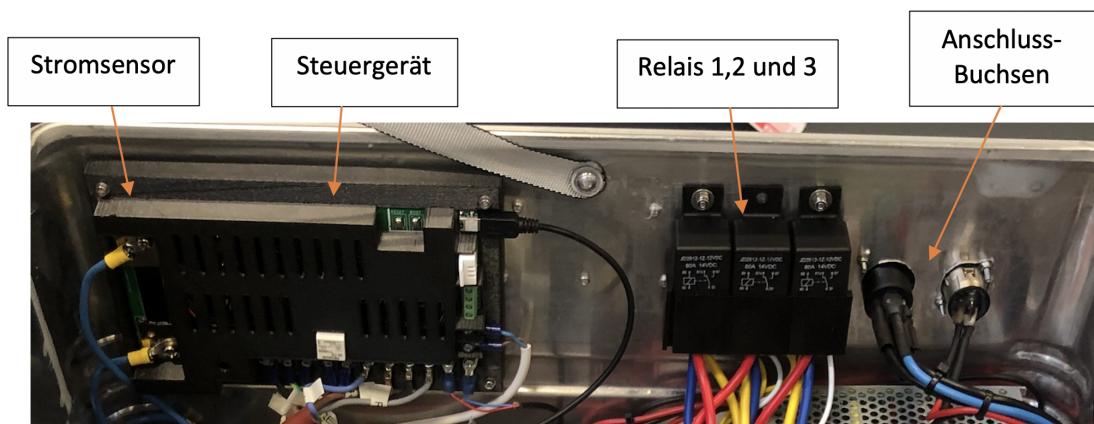


Abbildung 3.3: Innenansicht Steuergerät.

4 Hardware Steuergerät

Das entwickelte Steuergerät besteht aus 2 Platinen:

- einer Batteriemanagment-Platine
- und einer Hauptplatine.

Beide Platinen sind in einem gemeinsamen Gehäuse verbaut und über eine I²C Datenverbinung verbunden.

4.1 Batteriemanagment-Platine

Die Batteriemanagment-Platine überwacht den Zustand des internen Akkus. Da es sich um einen LiFePo-Akku handelt, lässt sich der Ladezustand des Akkus nicht oder nur sehr ungenau über die Ausgangsspannung bestimmen. Daher kommen in Systemen mit LiFePo sogenannte Coulomb-Zähler zum Einsatz. Diese integrieren die dem Akku entnommene oder zugeführte Ladung kontinuierlich. Somit lässt sich jederzeit der genaue Ladezustand des Akkus bestimmen.

Kern der Batteriemanagment-Platine ist das IC "LTC2943" von Analog Devices und ein Messwiderstand. Das IC misst [1]:

- Spannung,
- Strom,
- Integrierte Ladung (Coulomb-Counter),
- Temperatur des IC.

Der verbaute Messwiderstand hat einen Widerstand von 1,5 mOhm, im Zusammenspiel mit dem IC lassen sich Ströme bis 40 Ampere sehr exakt messen.

Problematisch war das Lademanagment des verbauten LiFePo Akkus. Das bereits vorhandene Modell hat eine eingebaute Ladeintelligenz, die über eine eigene Lade-Buchse von einem Netzteil versorgt werden kann. Eine Ladung über die Hauptanschlussklemmen ist leider nicht vorgesehen.

Um die Ladung des Akkus trotzdem präzise messen zu können, ist es daher erforderlich, entweder den abgegebenen Strom an den Hauptanschlussklemmen oder den

zugeführten Strom an der Lade-Buchse zu messen.

Hierzu wurden zwei Hochstromrelais (Relais 1 & 2) verbaut. Mit ihnen wird der Messwiderstand entweder mit den Hauptanschlussklemmen oder mit der Lade-Buchse verbunden. Die Relais werden automatisch durch die Hauptplatine des Steuergeräts angesteuert, sobald das System ausgeschaltet und das externe Ladegerät angeschlossen ist. Die Relais selber werden hierbei vom Ladegerät mit Strom versorgt.

Eine zeitgleiche Strom Entnahme und Zuführung in den Akku ist Systembedingt leider nicht möglich.

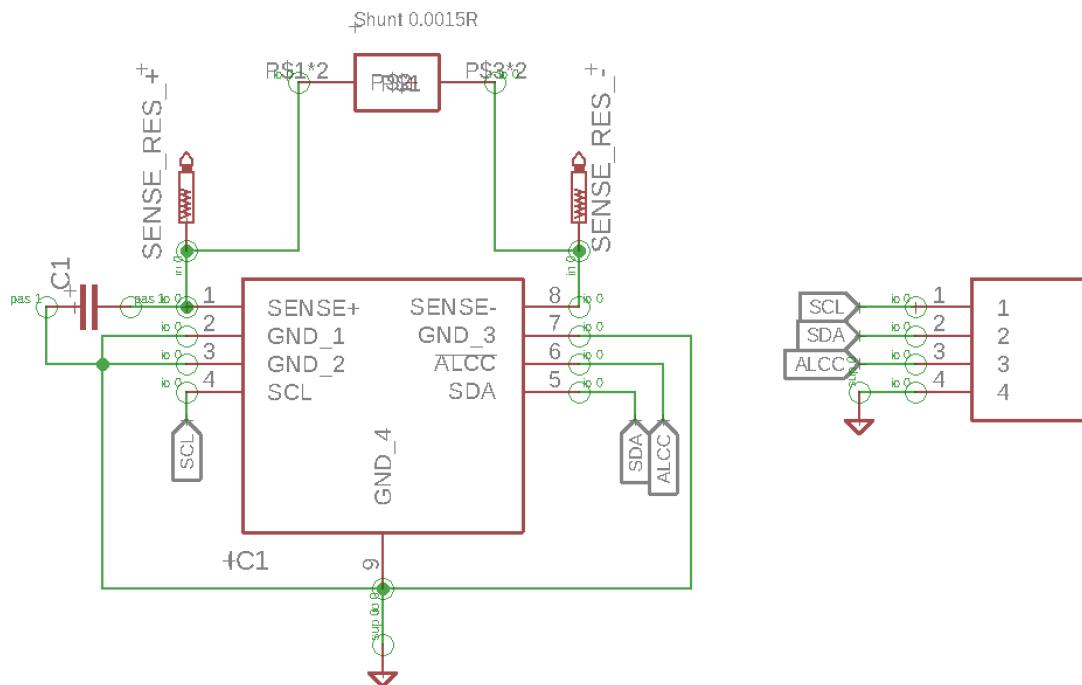


Abbildung 4.1: Schaltplan Batteriemanagment-Platine.

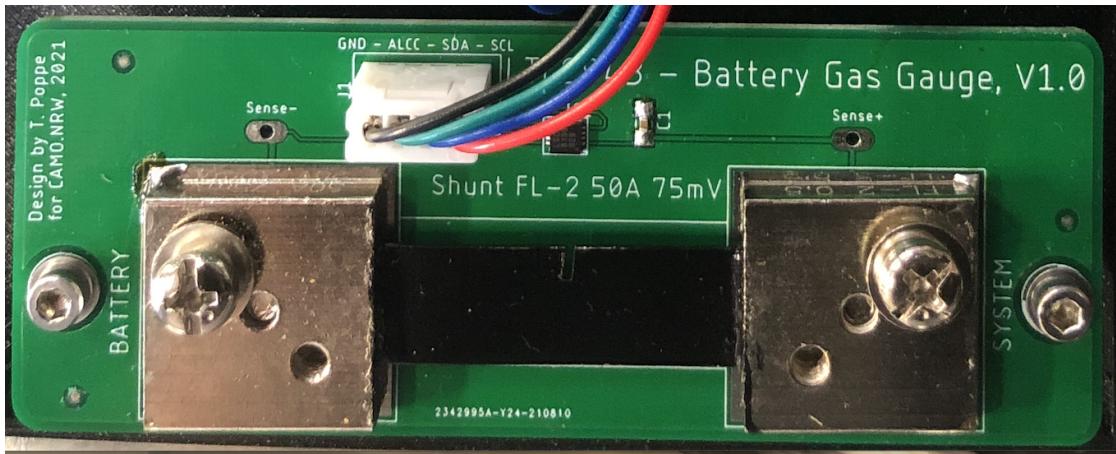


Abbildung 4.2: Batteriemanagment-Platine.

4.2 Hauptplatine

Neben der Batteriemanagment-Platine ist die Hauptplatine des Steuergeräts verbaut. Ihr Funktionsumfang erstreckt sich über:

- ESP32-WROOM-Controller inkl. WIFI und BLE
- Relaissteuerung Batteriebetrieb / Batterieladen
- Relaissteuerung Batterie / Externe Stromversorgung
- Lüftersteuerung Geschwindigkeit und Stromversorgung
- Relaissteuerung der Versorgungsspannung der Lüfter (Akku, Extern, Ladebuchse)
- Hochpräzise Spannungsmessung an der Buche "Externe Stromversorgung"
- Hochpräzise Spannungsmessung an der Buche "Ladeanschluss"
- Hochpräzise Spannungsmessung 0-20 V auf zwei optionalen Kanälen
- Ansteuerung des Powerschalters des PC über MOSFET
- Auslesen der Power-LED des PC
- eingebauter Temperatursensor
- eingebauter Luftfeuchtigkeitssensor

- Anschlussmöglichkeit für einen externen Temperatursensor (DS18B20)
- USB-Anschluss für
 - Programmierung
 - Übergabe aller Messwerte an PC als ROS-Serial-Node
- 5 Taster zur Bedienung
- E-Ink-Display zur Darstellung des Systemzustandes
- RGB-LED zur Darstellung des Systemzustandes
- Anschluss für einen optionalen Dreh-Druck-Schalter
- Zwei Erweiterungsbuchsen mit 5 V, 3,3 V und I²C
- Erzeugung 5 V und 3,3 V aus 9-28 V für alle Komponenten auf der Platine

Die einzelnen Anschlüsse sind in Abbildung 4.3, Seite 21 beschrieben.

Die Hauptplatine ist zusammen mit der Batteriemanagement-Platine in einem im FDM-Verfahren hergestellten Gehäuse an der Seitenwand der Kiste montiert (Abbildung 4.4, Seite 22). Alle Anschlussklemmen sind durch Stege voneinander isoliert.

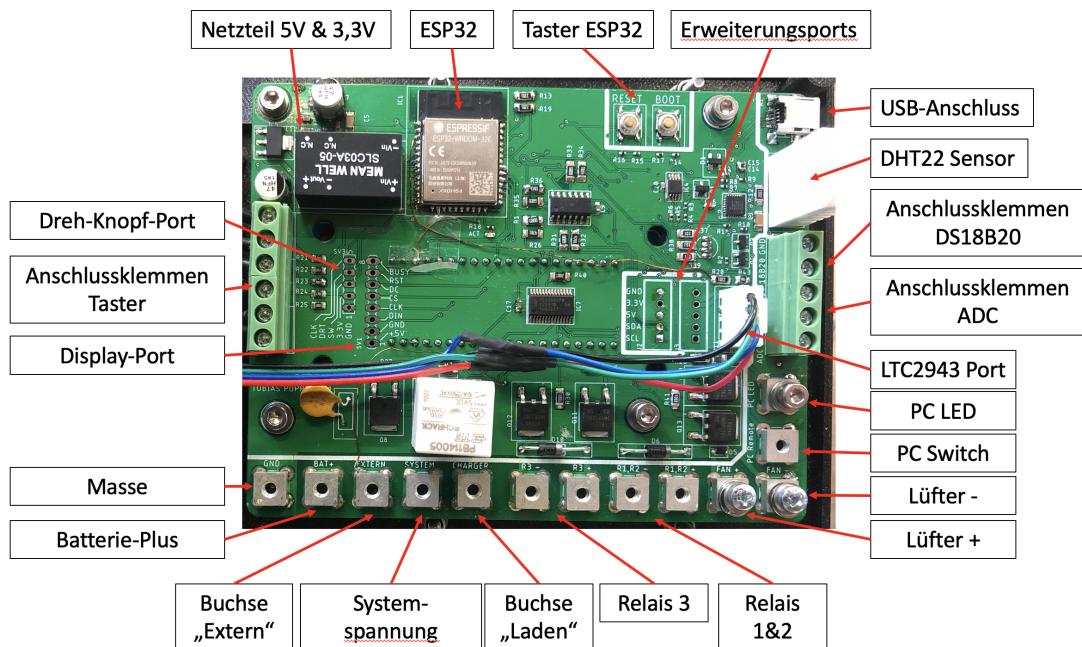


Abbildung 4.3: Hauptplatine Steuergerät.

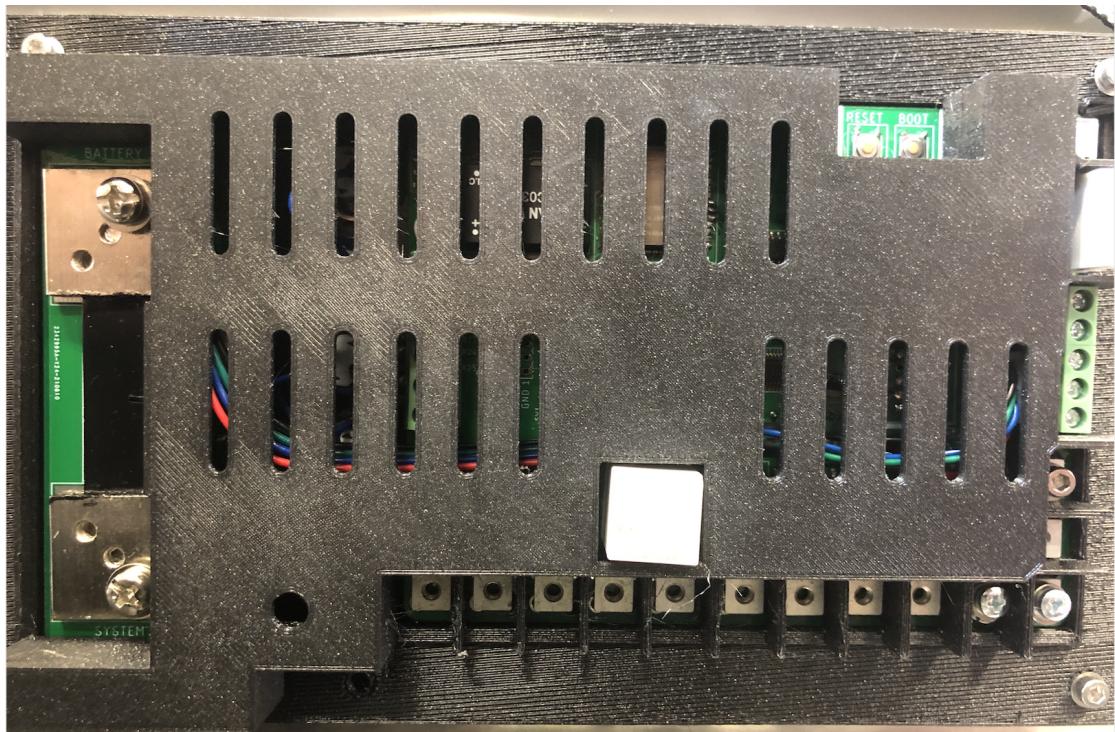


Abbildung 4.4: Gehäuse des Steuergeräts.

4.2.1 ESP32

Kern der Hauptplatine ist ein SoM ESP32-Wroom-32D von Espressif Systems. Das SoM bietet neben einer Dual-Core Xtensa mit einem Takt von 240MHz, 8Mb Flash Speicher und 4Mb Ram Kommunikationsmöglichkeiten über WiFi, Bluetooth sowie Bluetooth-Low-Energy. Die erforderliche Antenne ist ebenfalls direkt auf dem Modul vorhanden.

Außerdem wird das SoM vom Arduino-Framework unterstützt, wodurch eine große Anzahl Libraries für Peripherie zur Verfügung steht. [8]

Das Modul benötigt an äußerer Beschaltung (Abbildung 4.5) lediglich 3 Kondensatoren sowie eine Spannungsversorgung von 3,3 V. Die Programmierung erfolgt über eine USB-to-UART Schnittstelle (Abbildung 4.6) vom Typ CP2104 der Firma SiliconLabs die ebenfalls auf der Hauptplatine verbaut ist. Die selbe Schnittstelle wird für die Datenübertragung als ROS-Serial-Node im laufenden System verwendet. Weiter ist ein Reset des SoM über die DTR-Leitung der Schnittstelle möglich. Die USB-to-UART Schnittstelle bedarf auf Linux-Systemen keinen speziellen Treiber, unter Windows kann eine Installation notwendig sein.

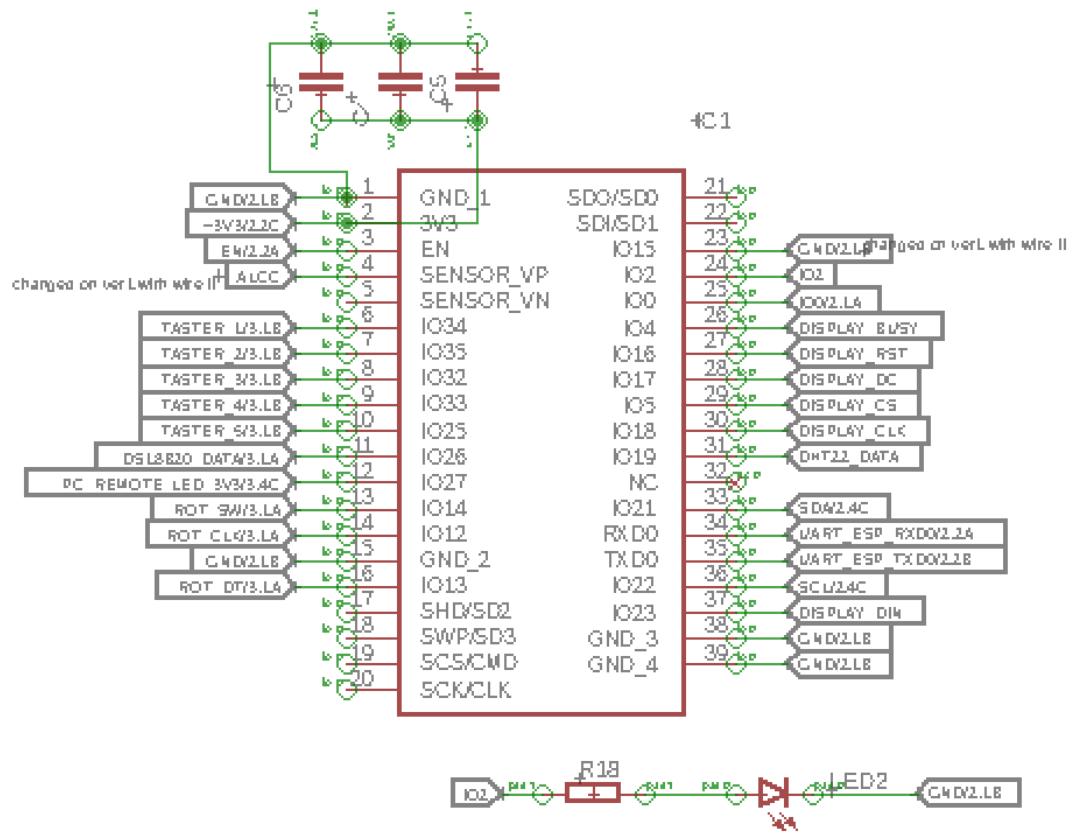


Abbildung 4.5: Beschaltung des ESP32-SoM.

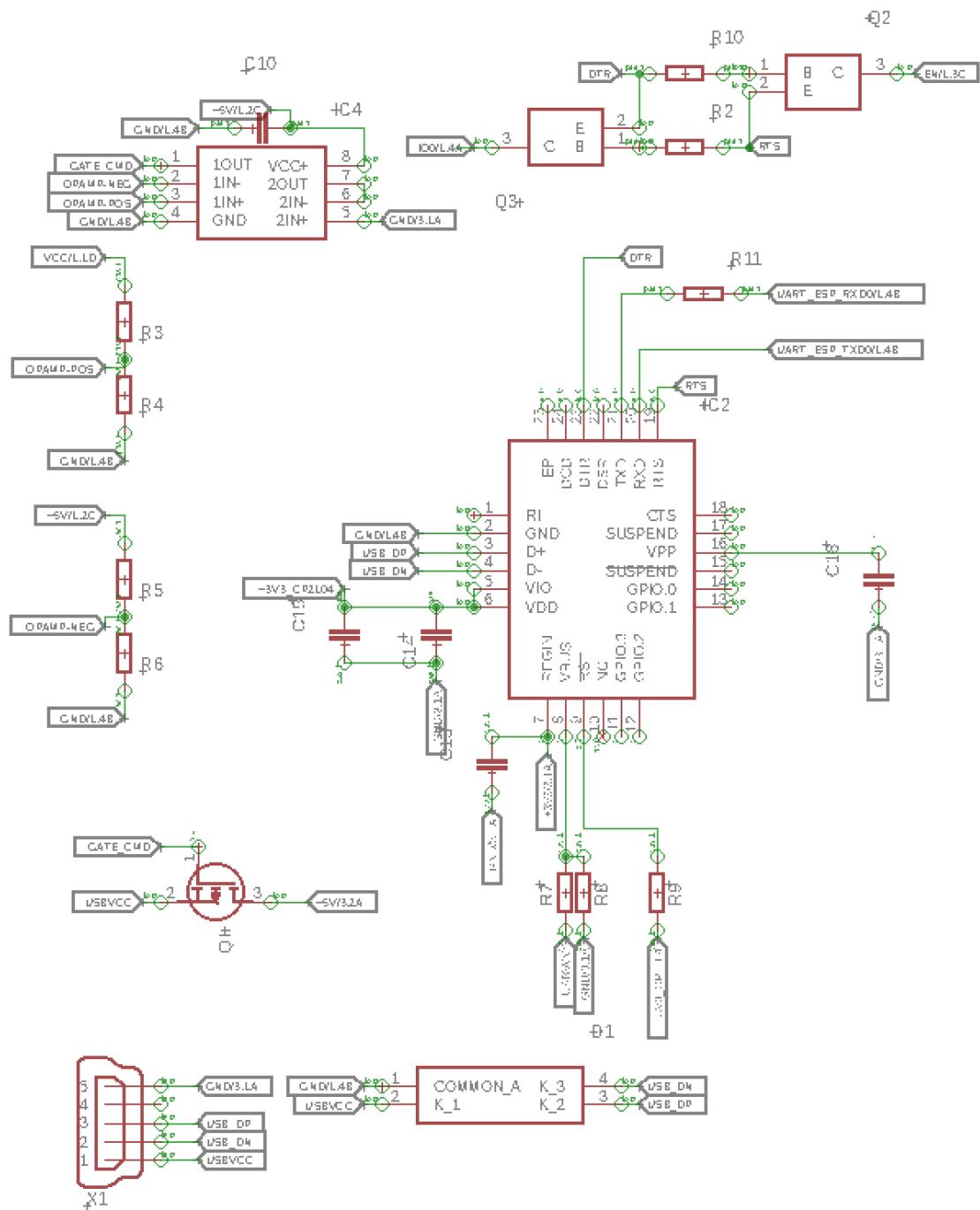


Abbildung 4.6: Beschaltung der USB-to-UART Schnittstelle.

4.2.2 PWM-IC PCA9685

Um die benötigte Anzahl an GPIOs im System zu erzielen, wurde an den ESP32 über die I²C-Schnittstelle ein PCA9685 der Firma NXP Semiconductors. Dieser IC ist eigentlich für das Dimmen mehrerer LEDs konzipiert, lässt sich aber auch für GPIO-Anwendungen nutzen. Er stellt 16 Ausgänge mit einer 12-Bit-PWM zur Verfügung [7].

An ihm sind neben den drei Farbkanälen der RGB-LED die MOSFETs der Relais-Steuerungen, der Lüftersteuerung sowie der PC-Remote-Schaltung angeschlossen.

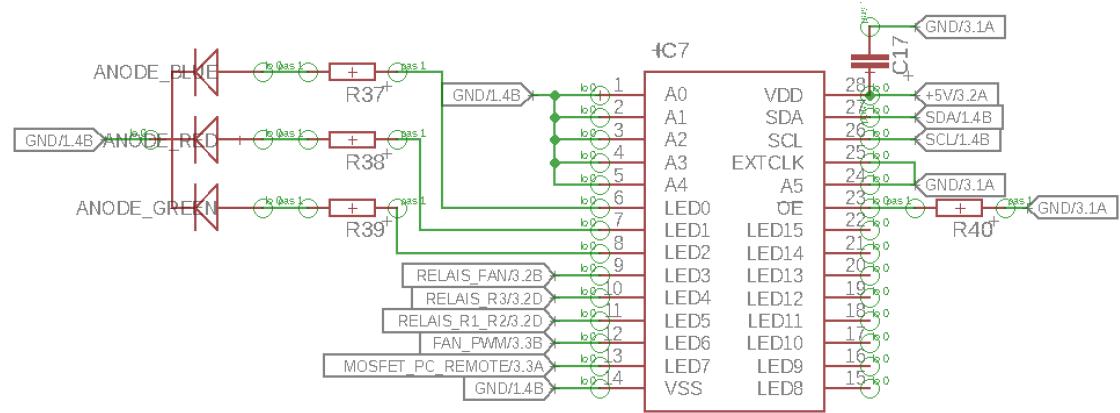


Abbildung 4.7: Beschaltung des PCA9685.

4.2.3 ADC-IC MCP3428

Zum Messen der Spannungen sowohl an der Buchse "Externe Stromversorgung" als auch an der Buchse "Laden" ist ein ADC-IC vom Typ MCP3428 der Firma Microchip Technology Inc. über die I²C-Schnittstelle angebunden. Der MCP3428 bietet auf 4 Kanälen eine A/D-Wandlung mit einer Genauigkeit von 16-Bit an[4]. An jedem Eingang ist zusätzlich ein Spannungsteiler im Verhältniss 1:8 vorgeschaltet. Da die Eingänge des IC nur mit maximal 2,5 V verbunden werden dürfen, ergibt sich somit eine maximale Spannung von 20 V mit einer Auflösung von 5 mV. Die zwei freien Kanäle des IC wurden auf eine externe Anschlussleiste ausgeführt, um eine zukünftige Erweiterung des Systems zu vereinfachen.

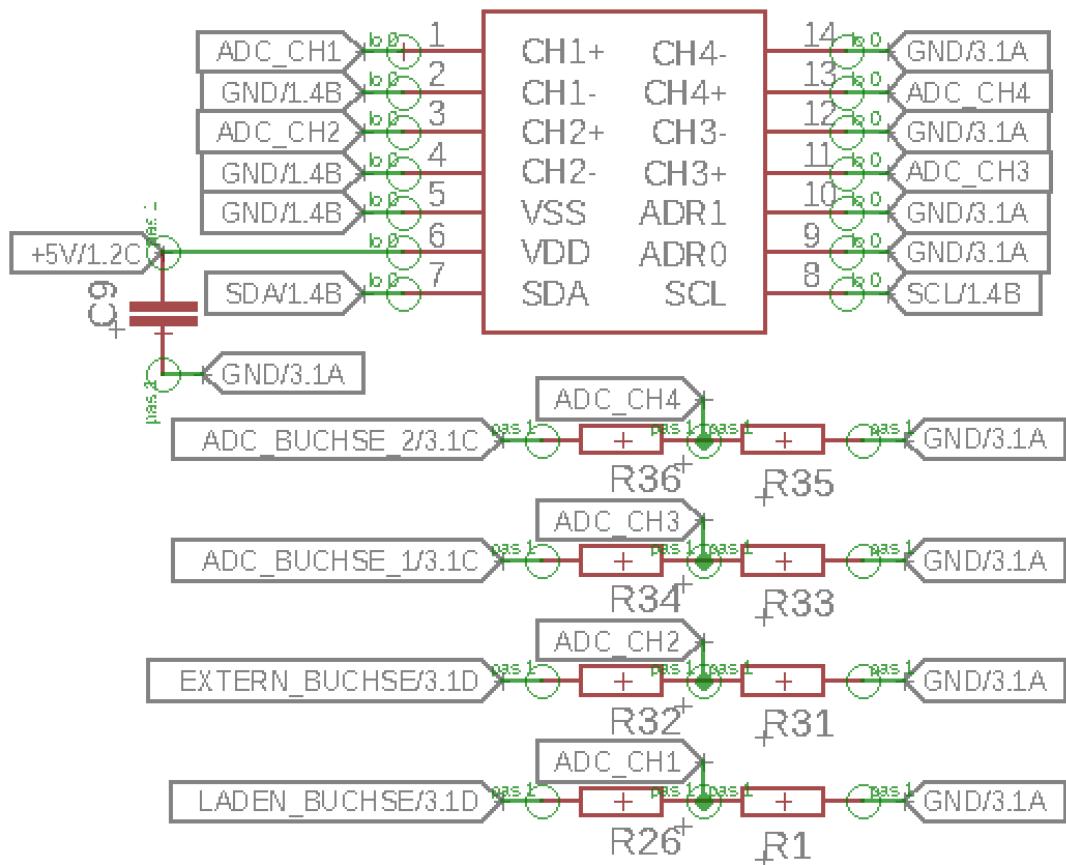


Abbildung 4.8: Beschaltung des MCP3428.

4.2.4 MOSFETs

Zum schalten der externen Relais, des Relais der Versorgungsspannung der Lüfter, der Regelung der Lüfter sowie zum schalten des Remote-Schalteingang des PC wurden MOSFETS vom Typ IRFR024 der Firma Vishay verbaut. Diese MOSFETS schalten bereits bei einer V_{gs} von 3,3 V durch. Der Maximale Schaltstrom liegt ungekühlt bei 9 Ampere [12]. Da alle geschalteten Verbraucher jedoch weit unter 0,5 Ampere benötigen, ist hier eine sehr großzügige Reserve eingeplant. Dadurch konnte auf eine Kühlung der MOSFETS verzichtet werden.

4.2.5 Display

Zur Anzeige der wichtigsten Systemdaten und des Systemzustandes ist ein 3-Farbiges E-Ink-Display der Firma Waveshare verbaut. Das Display bietet bei einer Displaydiagonalen von 2,9" eine Auflösung von 296x128 Pixel [3]. Da die elektrische Ansteuerung des Displays ähnlich einer Vielzahl anderer Displays entspricht, wurde das Display auf einem Sockel auf der Platine montiert. Zusätzlich ist ein Header mit den elektrischen Anschlüssen auf der Platine vorhanden. Somit kann bei Bedarf auf ein anderes Display umgebaut werden.

Da es sich um ein E-Ink-Display handelt, wird nur Strom beim aktuallisieren des Displays benötigt. Allerdings ist die Aktuallisierungsfrequenz aufgrund der 3-Farb-Technik im Display extrem niedrig: Eine Aktuallisierung dauert ca. 20 Sekunden.

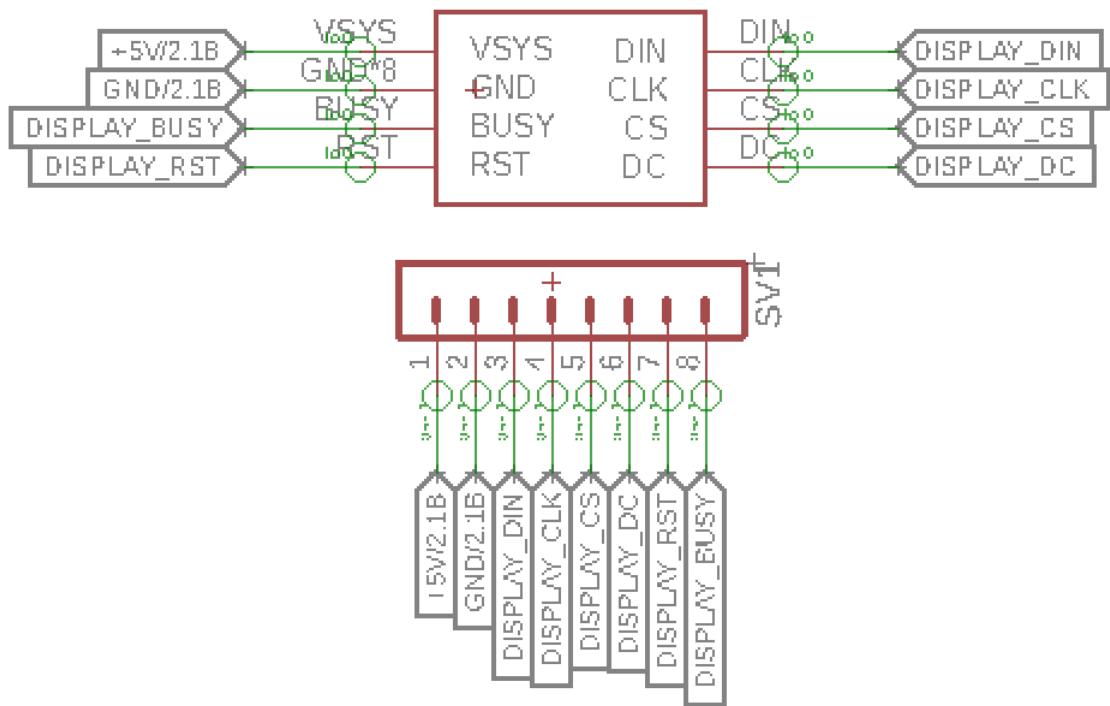


Abbildung 4.9: Beschaltung der Displayanschlüsse.

4.2.6 Restliche Beschaltung

Die weitere Beschaltung der Platine besteht aus der Spannungsversorgung, sowie der E/A-Peripherie. Diese sowie alle anderen Schaltpläne sind im Anhang zu finden.

5 Software

Die Firmware der Hauptplatine wurde mit Hilfe der Software "PlatformIO" [5] geschrieben. "PlatformIO" setzt auf den weit verbreiteten Editor "Visual Studio Code" [6] der Firma Microsoft auf und nutzt unter anderem das Arduino Framework.

Eine Weitergabe des kompletten Software-Projekts inklusive aller benötigten Einstellungen und verwendeter Libraries ist somit sehr leicht möglich.

Da auf der Hardware-ebene bei Peripherie und Speicher noch genügend Reserven übrig sind, ist das Steuergerät daher nachhaltig und zukunftssicher.

5.1 Bedienung

Zur Bedienung des Steuergeräts dienen neben der UART-Schnittstelle vor allem das Display und die daneben angeordneten Taster. Im aktuellen Stand wird lediglich der Taster "1" genutzt.

Der Taster ist hierbei Software seitig an den Power-Switch des PC gekoppelt. Außerdem wird die Stromversorgung aller übriger Hardware eingeschaltet. Bleibt ein Signal des PC über die Power-LED für 30 Sekunden aus, wird die Stromversorgung wieder getrennt.

Die neben dem Display angeordnete RGB-LED leuchtet Grün auf, solange ein Signal der Power-LED des PCs anliegt.

Das Display wurde in 4 Sektoren unterteilt:

- Anzeige des Ladezustandes des eingebauten Akkus,
- Temperatur im inneren der Kiste,
- Art der Stromversorgung,
- Systemzustand.

Die farbgestaltung der einzelnen Sektoren ist abhängig von der aktuellen Anzeige: bei kritischen Werten wird der Hintergrund des Sektors Rot:

- Akku Ladung < 20%,
- Temperatur > 50° Celsius,
- System eingeschaltet.

Achtung: da der LTC2943-IC die abgegebene bzw. zugeführte Ladung des Akkus integriert, muss ihm der aktuelle Stand der Ladung bekannt sein. Nach einem Kaltstart ist dies nicht der Fall! Aus diesem Grund - und um die Langzeitgenauigkeit zu erhöhen - wird der Akkuzustand auf 100% gestellt, sobald ein Ladestrom < 0,2 Ampera gemessen wird.

Nach einem Kaltstart muss daher das Ladegerät angeschlossen werden, bis der Akku voll geladen wurde.

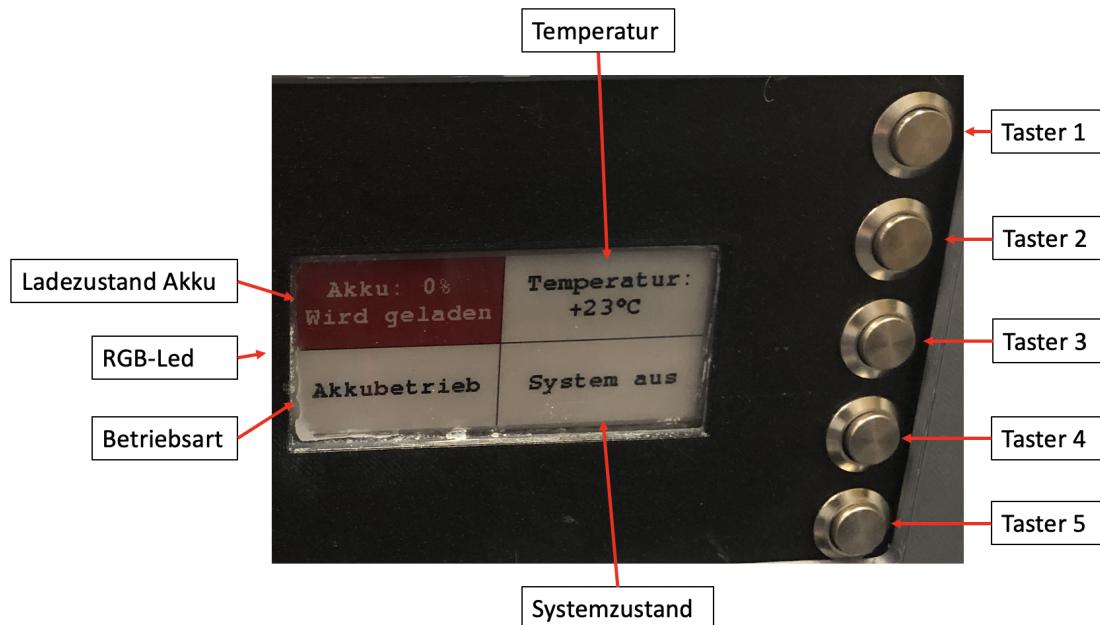


Abbildung 5.1: Bedienoberfläche extern mit LED, Display und Taster.

5.2 ROS-Serial-Node

Im Betrieb stellt das Steuergerät über die USB-Schnittstelle eine ROS-Serial-Node zur Verfügung. Zum einpflegen der Messwerte in ROS muss hierfür die Serial-Node auf dem PC gestartet werden. Anschließend sind folgende Topics in ROS verfügbar:

- dht_temperature → Temperatur im inneren,
- dht_humidity → Luftfeuchtigkeit im inneren,
- ds18b20_temperature → Temperatur des externen Temperaturfühlers,
- ltc2943_spannung → Spannung des Akkus oder des Ladegeräts,
- ltc2943_strom → Strom in (+) oder aus (-) dem Akku,
- ltc2943_temperature → Temperatur des LTC2943 ICs,
- ltc2943_coulomb → integrierte Ladung des Akkus,
- adc_extern_1 → Spannung an der externen ADC-Klemme 1,
- adc_charger → Spannung an der Buchse "Laden",
- adc_externe_stromversorgung → Spannung an der Buchse "Externe Stromversorgung",
- adc_extern_2 → Spannung an der externen ADC-Klemme 2,
- ltc2943_battery_percentage → Ladung des Akkus in %.

```
tobias@ubuntu ~ rosrun rosserial_python serial_node.py /dev/ttyUSB0
[INFO] [1634659844.916645]: ROS Serial Python Node
[INFO] [1634659844.921274]: Connecting to /dev/ttyUSB0 at 57600 baud
[INFO] [1634659847.793986]: Requesting topics...
[INFO] [1634659848.620719]: Note: publish buffer size is 512 bytes
[INFO] [1634659848.621982]: Setup publisher on dht_temperature [std_msgs/Float32]
[INFO] [1634659848.631504]: Setup publisher on dht_humidity [std_msgs/Float32]
[INFO] [1634659848.654013]: Setup publisher on ds18b20_temperature [std_msgs/Float32]
[INFO] [1634659848.664582]: Setup publisher on ltc2943_spannung [std_msgs/Float32]
[INFO] [1634659848.686782]: Setup publisher on ltc2943_strom [std_msgs/Float32]
[INFO] [1634659848.697757]: Setup publisher on ltc2943_temperature [std_msgs/Float32]
[INFO] [1634659848.720280]: Setup publisher on ltc2943_coulomb [std_msgs/Float32]
[INFO] [1634659848.731019]: Setup publisher on adc_extern_1 [std_msgs/Float32]
[INFO] [1634659848.741955]: Setup publisher on adc_charger [std_msgs/Float32]
[INFO] [1634659848.764625]: Setup publisher on adc_externe_stromversorgung [std_msgs/Float32]
[INFO] [1634659848.774380]: Setup publisher on adc_extern_2 [std_msgs/Float32]
```

Abbildung 5.2: Starten der Serial-Node auf dem PC.

```
tobias@ubuntu ~ ➔ rostopic list
adc_charger
adc_extern_1
adc_extern_2
adc_externe_stromversorgung
dht_humidity
dht_temperature
diagnostics
ds18b20_temperature
ltc2943_coulomb
ltc2943_spannung
ltc2943_strom
ltc2943_temperature
rosout
rosout_agg
```

Abbildung 5.3: Anzeige aller Topics.

5.3 Betriebszustand des Steuergeräts

Um den Stromverbrauch zu minimieren, schaltet das Steuergerät sich in einen Deepsleep, sobald der PC länger als 30 Sekunden ausgeschaltet ist. Der Deepsleep wird alle 60 Sekunden unterbrochen. Außerdem weckt die Betätigung von Taster 1 das Steuergerät unverzüglich aus dem Deepsleep.

Durch den Deepsleep konnte der Stromverbrauch des Steuergeräts auf unter 1 mA gesenkt werden. Bei voll geladenem Akku reicht die Kapazität für eine Laufzeit von ca. 2 Jahren.

Durch die Beschaltung der Relais ergeben sich die Zustände aus Tabelle 5.1.

Buchse "Externe Stromversorgung"	Buchse "Laden"	PC Power-LED	Systemspannung	Akku wird geladen
0	0	Aus	Nein	Nein
0	0	An	Ja	Nein
0	1	Aus	Nein	Ja
0	1	An	Ja	Nein
1	0	Aus	Ja	Nein
1	0	An	Ja	Nein
1	1	Aus	Ja	Ja
1	1	An	Ja	Ja

Tabelle 5.1: Elektrische Betriebszustände.

5.4 Programmablaufplan

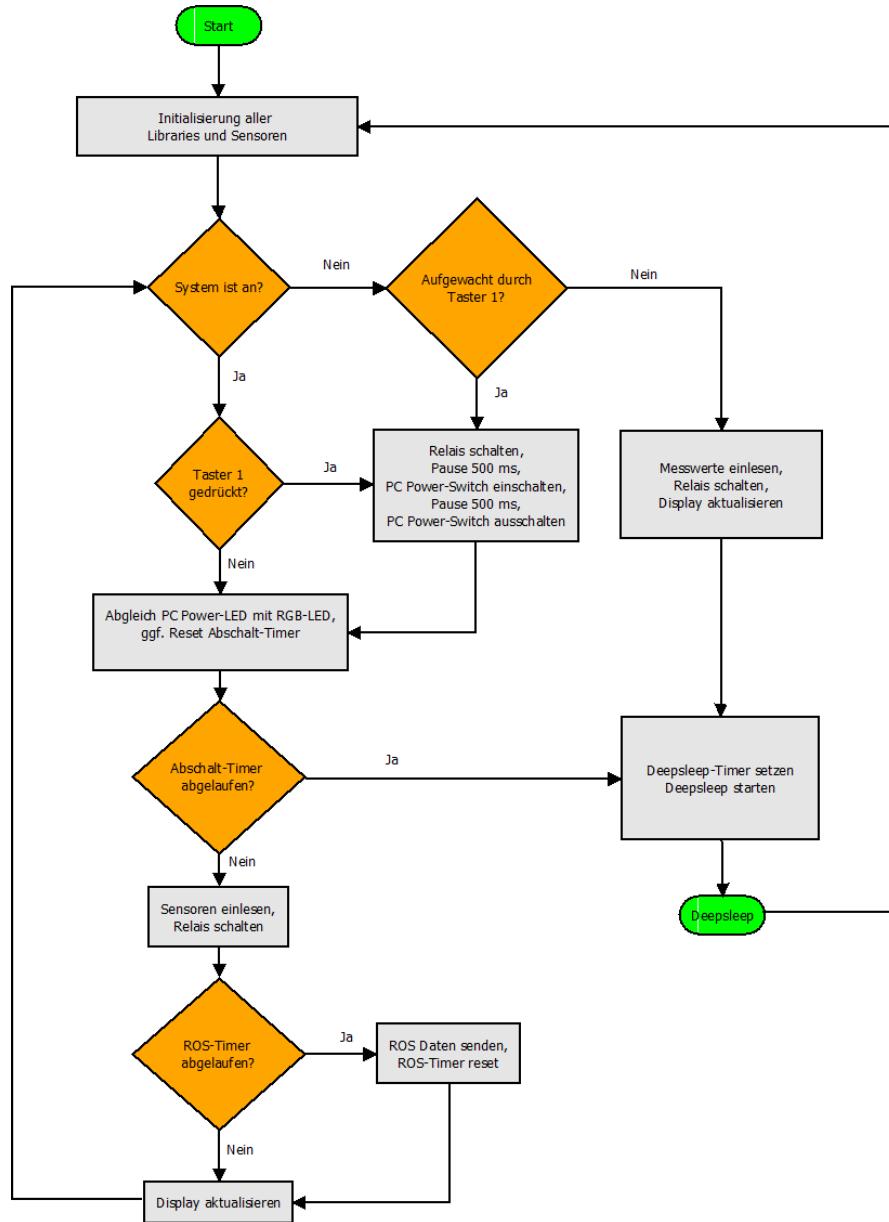


Abbildung 5.4: Programmablaufplan.

Die Dokumentation aller Routinen befindet sich im Anhang.

5.5 Auslesen von Messwerten

Alle Messwerte werden zentral in einem Struct vom Typ "Messwerte" gespeichert. Das Struct besteht aus:

```
struct messwerte
{
    float dht_temperature, dht_humidity;
    float ds18b20_temperature;
    float ltc2943_spennung, ltc2943_strom;
    float ltc2943_temperature, battery_percent;
    uint16_t ltc2943_coulomb;
    float adc_extern_1, adc_charger;
    float adc_externe_stromversorgung, adc_extern_2;
};
```

Zum aktualisieren aller Messwerte wird ein Pointer auf dieses Struct an die Routine "read_all_sensors" übergeben. Diese Routine ruft dann die einzelnen Sensoren ab, speichert die Messwerte in der jeweiligen Variabel und beendet sich anschließend.

Das auslesen der einzelnen Werte geschieht entsprechend der jeweiligen Datenblätter. Umfangreicher ist lediglich das auslesen des ADC-IC und des LTC2943-IC, da hier noch Umrechnungen erfolgen müssen.

5.5.1 Berechnungen der ADC-Werte

Da die ADC-Werte eine Länge von 12 Bits haben, der I²C-Bus jedoch nur 8-Bit pro Speichereinheit überträgt, muss der jeweilige Wert aus zwei Bytes kombiniert werden:

```
long raw_result = (buffer[0] << 8) | buffer[1];
```

Der ADC-IC wird in der Schaltung "single-ended" betrieben. Der Messbereich von -2,048 V bis +2,048 V wird somit beschränkt auf 0 V bis +2,048 V. Daraus ergibt sich ein Wert von 0,001 V für das LSB. Die Umrechnung des Messwertes am ADC-Pin lautet daher:

```
float result = raw_result * 0.001;
```

Da vor dem jeweiligen ADC-Pin noch ein Spannungsteiler verbaut ist, muss aus dem Ergebniss nun noch die Spannung vor dem Spannungsteiler berechnet werden. Die Formel für den Spannungsteiler lautet:

$$U_{out} = \frac{U_{in}}{(R_1 + R_2)} * R1$$

Da wir U_{out} messen interessiert uns U_{in} . Hierzu wird die Formel umgestellt zu:

$$U_{in} = \frac{U_{out} * (R_1 + R_2)}{R_1}$$

Mit den Werten der verbauten Widerständen (4700 Ohm und 680 Ohm) ergibt sich daraus:

$$U_{in} = \frac{U_{out} * (5380)}{680}$$

Im Code entspricht das:

```
result = (result * 5380) / 680;
```

5.6 Berechnungen der LTC2943 Werte

Beim auslesen der Werte des LTC2943-IC müssen ebenfalls einige Berechnungen vorgenommen werden. Grundsätzlich müssen auch hier wie beim ADC-IC erst zwei Bytes zu einem Long-Wert kombiniert werden. Hierzu wird der selbe Code verwendet:

```
Long value =
(buffer[0x_{Adresse MSB}] << 8) | buffer[0x_{Adresse LSB}];
```

5.6.1 Berechnung der Spannung

Das Datenblatt gibt zum Berechnung der gemessenen Spannung folgende Formel an:

$$V_{SENSE-} = 23,6V * \frac{RESULT_h}{FFFF_h} = 23,6V * \frac{RESULT_{DEC}}{65535}$$

Im Code entspricht dies:

```
*spannung = 23.6 * (float(spannung_raw) / 65535);
```

5.6.2 Berechnung der Stromstärke

Zur Berechnung der gemessenen Stromstärke gibt das Datenblatt folgende Formel an:

$$I_{BAT} = \frac{60mV}{R_{SENSE}} * \left(\frac{RESULT_{DEC} - 32767}{32767} \right)$$

Setzt man den verbauten Messwiderstand von 1,5 mOhm ein ergibt sich der Code:

```
*strom = (0.06/0.0015)*((float(strom_raw)-32767)/32767);
```

5.6.3 Berechnung der Temperatur

Die Formel für den Temperaturwert lautet:

$$T = 510K * \frac{RESULT_{DEC}}{65535}$$

Da der IC die Temperatur in Kelvin ausgibt, muss nun noch eine Wandlung in Celsius stattfinden. Dazu wird der Kelvin-Wert von 273,15 subtrahiert.
Daraus ergibt sich der folgende Code:

```
*temperatur = (510*(float(temperatur_raw)/65535))-273.15;
```

5.6.4 Berechnung des Ladezustands

Um aus dem Coulomb-Roh-Wert den Ladezustand berechnen zu können, muss erst die gemessene Ladungsmenge pro LSB berechnet werden.

Das Datenblatt gibt hierzu die folgende Formel an:

$$q_{LSB} = 0,340mAh * \frac{50mOhm}{R_{SENSE}}$$

Daraus ergibt sich für den verbauten Messwiderstand eine Ladung von 0,113333mAh pro LSB.

Die verbaute Batterie hat laut Hersteller eine Kapazität von 30Ah. Um eine Tiefentladung zu vermeiden rechnen wir mit einer Kapazität von 27Ah. Um aus dieser Kapazität einen Hexadezimalen Wert dem Messwert des IC entsprechend zu berechnen, teilen wir die Kapazität durch die Ladungsmenge pro LSB:

$$BAT_{CAPACITY} = \frac{27Ah}{0,0113333}$$

Da wir wissen, das der Coulombzähler vom Programm auf den maximalen Wert gesetzt wird, sobald der Ladestrom geringer als 0,2 A wird, können wir durch subtraktion den minimalen Wert berechnen.

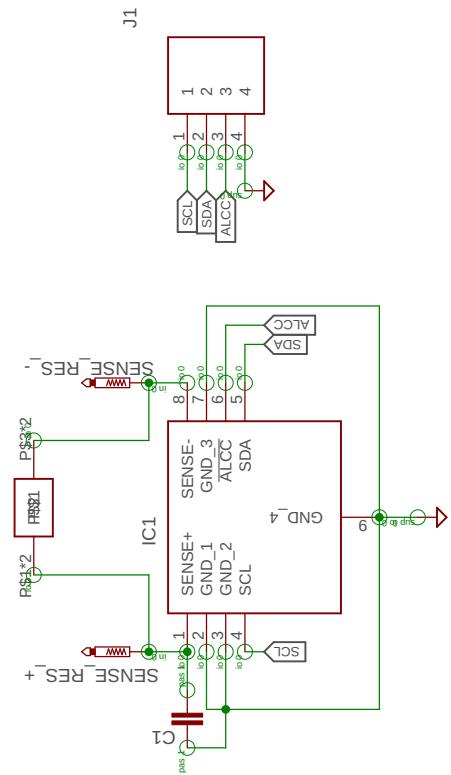
Im Code wird im selben Schritt ein mapping auf den Bereich von 0 - 100 % vorgenommen:

```
float coulomb_bat_empty=65535-(battery_capacity / 0.011333);
float bat_percent_raw =
    map(*coulomb, 65535, int(coulomb_bat_empty), 100, 0);
*battery_percent = bat_percent_raw;
```

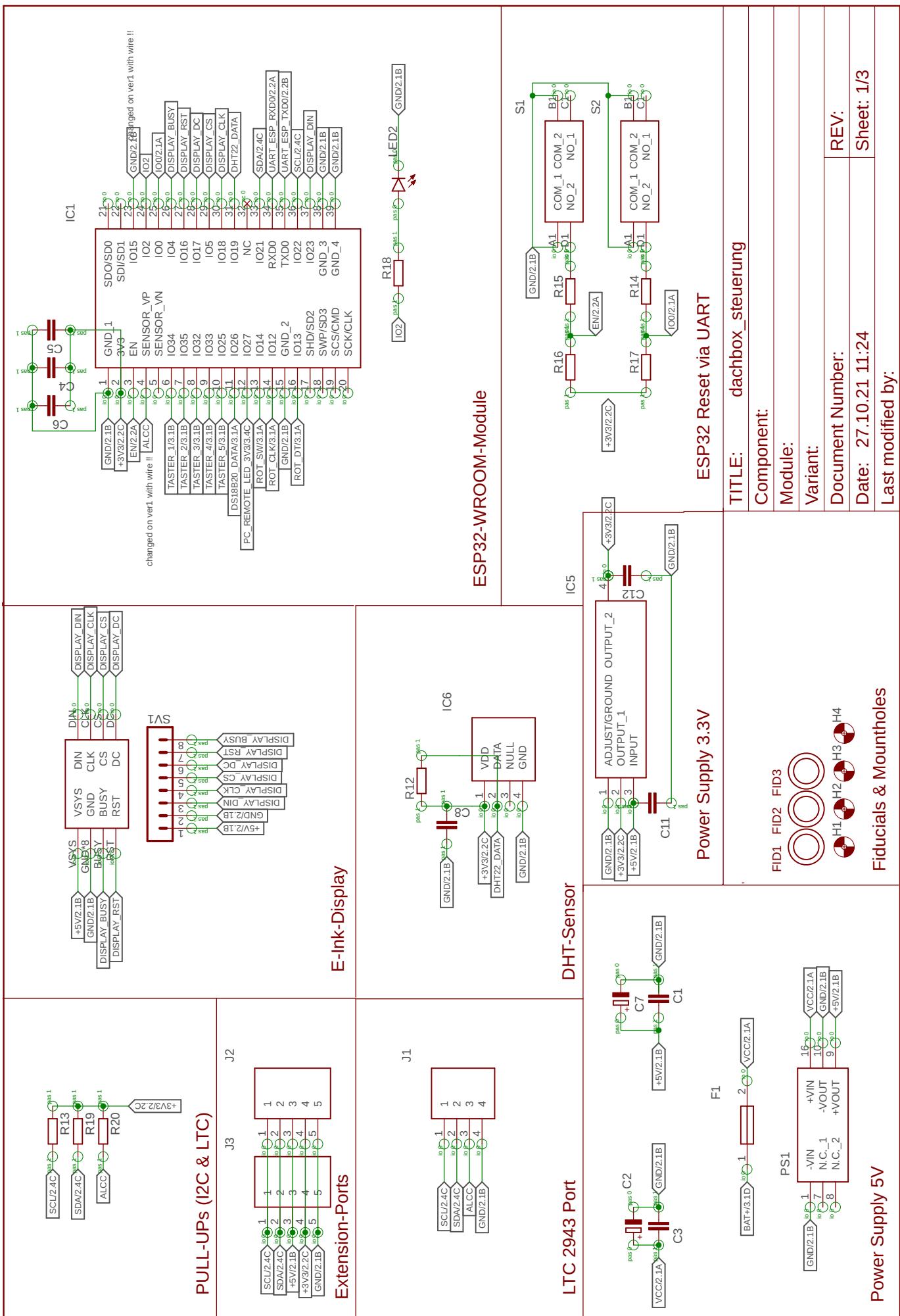
6 Anhang

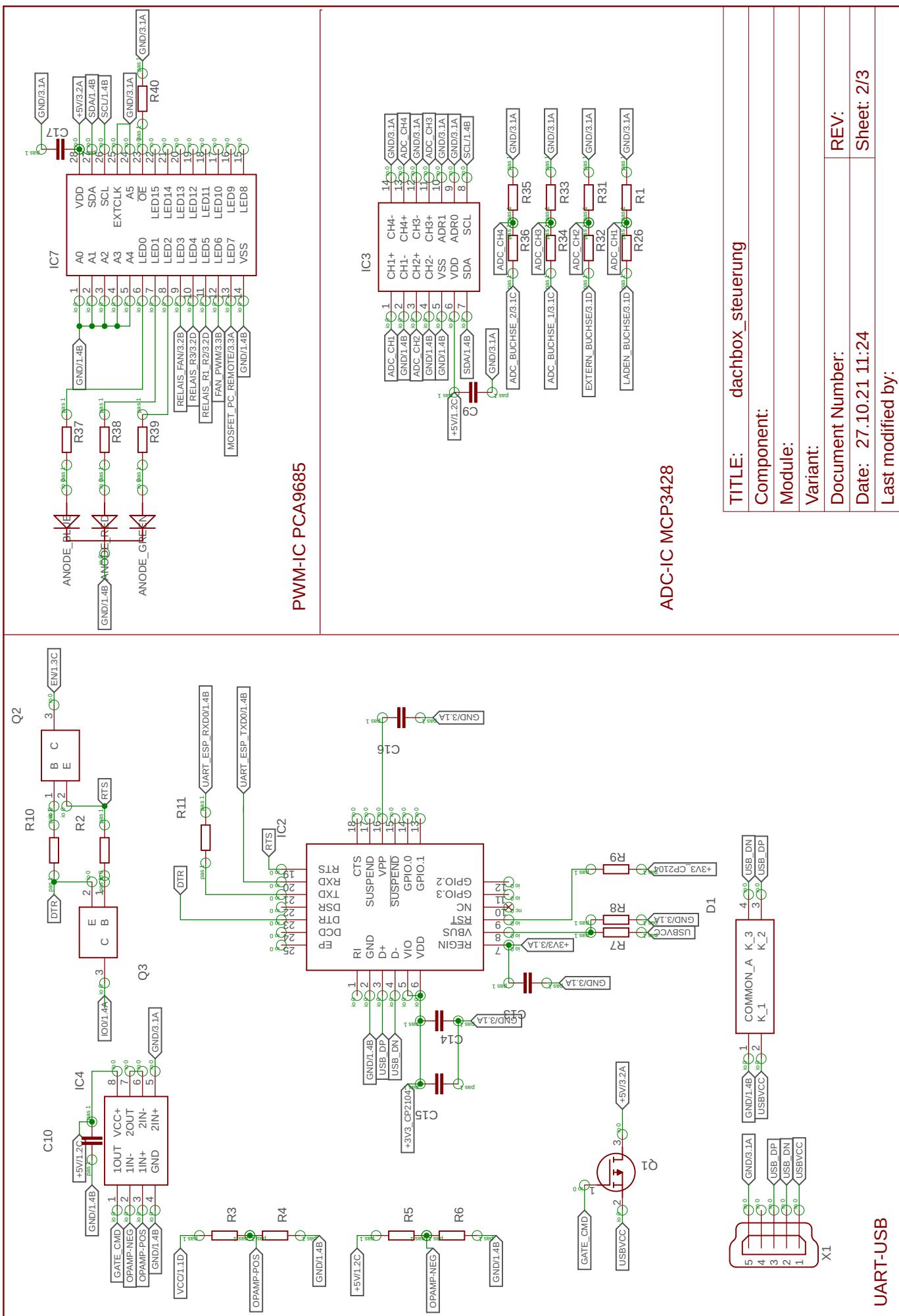
- Schaltplan Platine LTC2934
- Schaltplan Hauptplatine
- Doxygen-Dokumentation Firmware

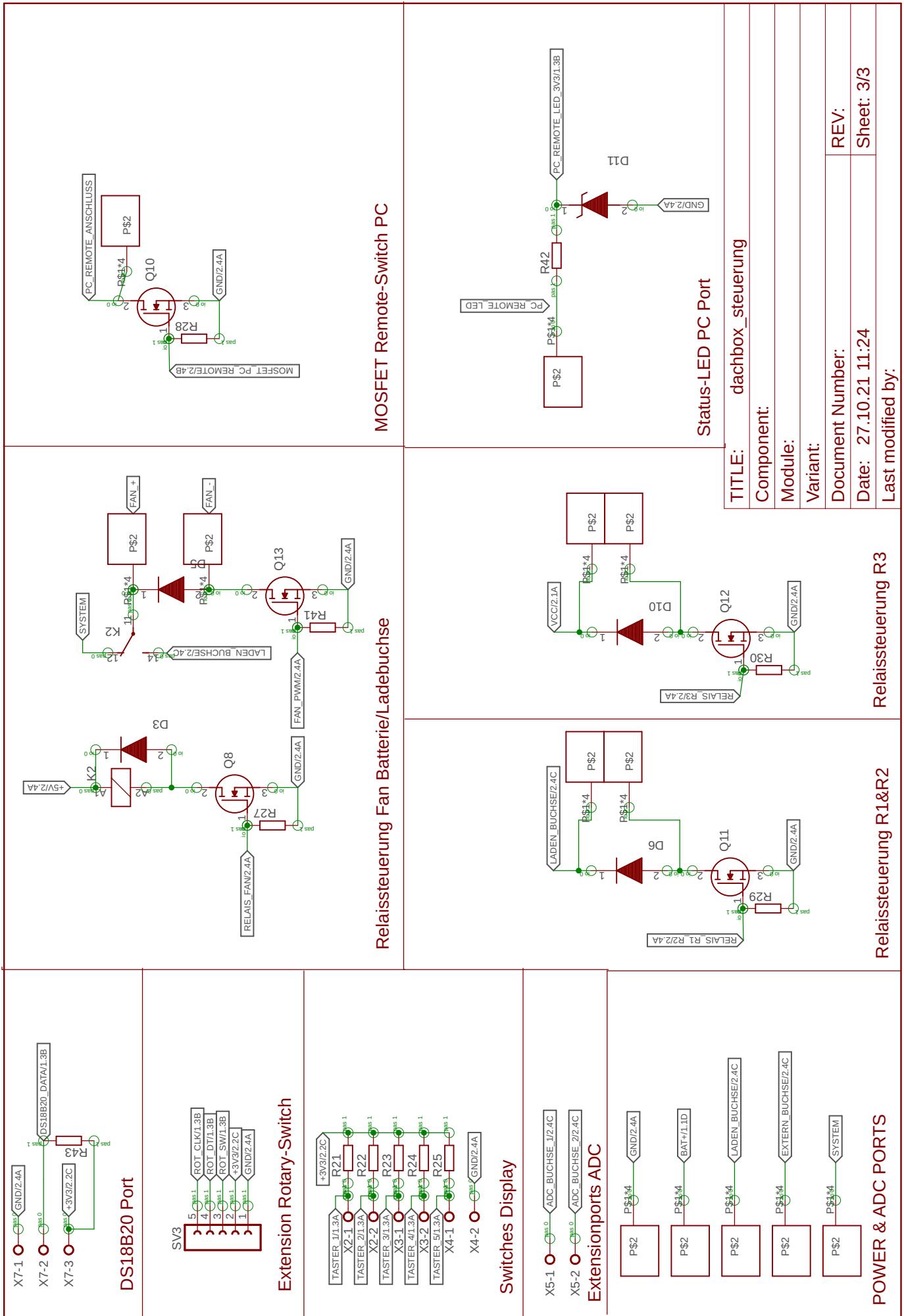
TITLE:	ltc2934_breakout
Component:	
Module:	
Variant:	
Document Number:	
Date:	nicht gespeichert!
Last modified by:	



FID3 FID2 FID1 H2 H1







main.cpp File Reference

Main-File for the CAMO.NRW Firmware. [More...](#)

```
#include <Arduino.h>
#include "PCA9685.h"
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <ros.h>
#include <std_msgs/Float32.h>
#include <GxEPD2_BW.h>
#include <GxEPD2_3C.h>
#include <GxEPD2_7C.h>
#include <Fonts/FreeMonoBold9pt7b.h>
#include <BitmapDisplay.h>
#include <TextDisplay.h>
#include <GxEPD2_display_selection_new_style.h>
```

Classes

```
struct messwerte
struct display_data
```

Macros

```
#define DEBUG false
#define DHTTYPE DHT22
#define DHTPIN 19
#define DS18B20_PIN 26
#define LTC2943_ALCC_PIN 36
#define taster_1_PIN 34
#define taster_2_PIN 35
#define taster_3_PIN 32
#define taster_4_PIN 33
#define taster_5_PIN 25
#define pc_remote_led_PIN 27
#define on_board_led_PIN 2
#define pwm_port_rgb_led_blau 0
#define pwm_port_rgb_led_rot 1
#define pwm_port_rgb_led_gruen 2
#define pwm_port_relais_fan 3
#define pwm_port_relais_r3 4
```

```
#define pwm_port_relais_r1_r2 5
#define pwm_port_fan_pwm 6
#define pwm_port_mosfet_pc_remote 7
#define battery_capacity 27
#define uS_TO_S_FACTOR 1000000 /*! Conversion factor for micro seconds to seconds */
#define TIME_TO_DEEPSLEEP 60 /*! Time ESP32 will go to sleep (in seconds) */
#define ENABLE_GxEPD2_GFX 1
```

Functions

- void **i2c_flush ()**
Routine for flushing the i2c-interface. [More...](#)
- void **pwm_setup ()**
Initialize the pwm-controller. [More...](#)
- void **relais_switch_to_charge (bool on)**
Switch the relais 1 & 2 between using battery for the system or charging the battery. [More...](#)
- void **relais_switch_system_on_bat (bool on)**
Switch the relais 3 between using battery or external power-supply for the system. [More...](#)
- void **luefter_steuerung (float temperatur)**
Set the fan-speed according to the temperature temperature < 40 degree Celsius: Fans off
temperature > 70 degree Celsius: Fans at highest speed temperture between: Fan-Speed
linear to temperature. [More...](#)
- void **fan_relais_extern ()**
Switch fan power-supply to external-connector "Load battery".
- void **fan_relais_batterie ()**
Switch fan power-supply to internal battery.
- void **pc_remote_mosfet (bool on)**
Switch the mosfet to controll the pc's powerswitch on or off. [More...](#)
- void **set_rgb_led (int rot, int gruen, int blau)**
Set the color of the build-in-rgb-led. [More...](#)
- DHT_Unified **dht (DHTPIN, DHTTYPE)**
- void **dht_setup ()**
Init the DHT-Sensor.
- void **get_dht_values (float *dht_temperature, float *dht_humidity)**
Get values from the DHT-Sensor. [More...](#)
- OneWire **ds18b20_oneWire (DS18B20_PIN)**
- void **ds18b20_setup ()**
Init the DS18B20-Sensor.
- void **ds18b20_start_meassurment ()**
Tell the DS18B20-Sensor to start a measurement. The measurements takes about 500 ms, so
comeback later to read out.
- void **get_ds18b20_value (float *temperature)**

Get values from the external DS18B20-Sensor. [More...](#)

void	mcp3428_setup ()	Init the MCP3428 ADC-IC. More...
void	get_adc_values (float *adc_extern_1, float *adc_charger, float *adc_externe_stromversorgung, float *adc_extern_2)	Get values from ADC-IC. More...
void	ltc2943_setup ()	Init the LTC2943 battery gauge IC.
void	ltc_2943_standby (bool state)	Switch the analog front-end of the LTC2943 into standby or vice versa. More...
void	set_ltc_2943_coulomb_full ()	tell the LTC-2943 the battery is fully charged
void	get_ltc2943_values (float *spannung, float *strom, uint16_t *coulomb, float *temperatur, float *battery_percent)	Get values from LTC2943 battery gauge IC. More...
BitmapDisplay	bitmaps (display)	
void	display_setup ()	init the display
void	display_refresh_background_task (void *pvParameters)	background helper task for display-refreshing. More...
void	show_data_on_display (struct display_data data_to_show)	Refresh the display if there are newer values than already displayed. More...
void	gpio_setup ()	init all GPIO ports that are directly connected to the ESP32 More...
void	set_onboard_led (bool on)	set the onboard LED on or off. More...
void	read_all_sensors (struct messwerte *messwerte)	Read all sensors and return their current values. More...
ros::Publisher	pub_dht_temperature ("dht_temperature", &ros_msg_dht_temperature)	
ros::Publisher	pub_dht_humidity ("dht_humidity", &ros_msg_dht_humidity)	
ros::Publisher	pub_ds18b20_temperature ("ds18b20_temperature", &ros_msg_ds18b20_temperature)	
ros::Publisher	pub_ltc2943_spannung ("ltc2943_spannung", &ros_msg_ltc2943_spannung)	
ros::Publisher	pub_ltc2943_strom ("ltc2943_strom", &ros_msg_ltc2943_strom)	
ros::Publisher	pub_ltc2943_temperature ("ltc2943_temperature", &ros_msg_ltc2943_temperature)	
ros::Publisher	pub_ltc2943_coulomb ("ltc2943_coulomb", &ros_msg_ltc2943_coulomb)	
ros::Publisher	pub_ltc2943_battery_percentage ("ltc2943_battery_percentage", &ros_msg_ltc2943_battery_percentage)	
ros::Publisher	pub_adc_extern_1 ("adc_extern_1", &ros_msg_adc_extern_1)	
ros::Publisher	pub_adc_charger ("adc_charger", &ros_msg_adc_charger)	
ros::Publisher	pub_adc_externe_stromversorgung ("adc_externe_stromversorgung", &ros_msg_adc_externe_stromversorgung)	

```
ros::Publisher pub_adc_extern_2 ("adc_extern_2", &ros_msg_adc_extern_2)
void ros_serial_init ()
    init a ROS-Serial-Node and advertise all publishers to it
void ros_serial_send_msg (struct messwerte messwerte)
    write the actual values to all publishers and send it out by uart More...
void setup ()
    setup routine, started after each coldstart or deepsleep-wakeup
void loop ()
    Main Loop.
```

Variables

```
RTC_DATA_ATTR bool system_an
RTC_DATA_ATTR bool first_start = true
RTC_DATA_ATTR uint8_t relais_mode = 0
long switch_sys_off_at_time = 0
RTC_DATA_ATTR PCA9685 pwmController
    Routines for accesing the PCA9685 pwm-controller.

DallasTemperature ds18b20_sensor & ds18b20_oneWire
    DeviceAddress sensor_address
        uint8_t ltc2943_config = 0b11111000
        uint8_t ltc2943_config_standby = 0b11111001
        uint8_t ltc2943_adr = 0x64
    TaskHandle_t Task1
RTC_DATA_ATTR struct display_data data_on_display
    RTC_DATA_ATTR bool refresh_display_busy_flag = false
    std_msgs::Float32 ros_msg_dht_temperature
    std_msgs::Float32 ros_msg_dht_humidity
    std_msgs::Float32 ros_msg_ds18b20_temperature
    std_msgs::Float32 ros_msg_ltc2943_spannung
    std_msgs::Float32 ros_msg_ltc2943_strom
    std_msgs::Float32 ros_msg_ltc2943_temperature
    std_msgs::Float32 ros_msg_ltc2943_coulomb
    std_msgs::Float32 ros_msg_ltc2943_battery_percentage
    std_msgs::Float32 ros_msg_adc_extern_1
    std_msgs::Float32 ros_msg_adc_charger
    std_msgs::Float32 ros_msg_adc_externe_stromversorgung
    std_msgs::Float32 ros_msg_adc_extern_2
    ros::NodeHandle nh
    RTC_DATA_ATTR bool esp_was_in_deepsleep = 0
    int ros_publish_time
```

Detailed Description

Main-File for the CAMO.NRW Firmware.

Author

Tobias Poppe

Date

28.10.2021

This file is the [main.cpp](#) file of the CAMO.NRWs Mainboard Firmware All files and the documentation can be found on <https://github.com/topoppe/camo.nrw.dachbox>

See also

<https://www.camo.nrw>

<https://github.com/topoppe/camo.nrw.dachbox>

Macro Definition Documentation

◆ DEBUG

```
#define DEBUG false
```

Define Debug-State, Pins and bat-capacity

Function Documentation

◆ display_refresh_background_task()

```
void display_refresh_background_task ( void * pvParameters )
```

background helper task for display-refreshing.

Runs on second CPU-Core to avoid blocking while display refreshes

◆ get_adc_values()

```
void get_adc_values ( float * adc_extern_1,  
                      float * adc_charger,  
                      float * adc_externe_stromversorgung,  
                      float * adc_extern_2  
)
```

Get values from ADC-IC.

Calculates values in Volt from the raw-adc-values and stores them into memory

Parameters

adc_extern_1	pointer to a float variable to store the value from adc_extern_1 in Volt into
adc_charger	pointer to a float variable to store the value from the external charger-connector in Volt into
adc_externe_stromversorgung	pointer to a float variable to store the value from the external power-supply-connector in Volt into
adc_extern_2	pointer to a float variable to store the value from adc_extern_2 in Volt into

◆ get_dht_values()

```
void get_dht_values ( float * dht_temperature,  
                      float * dht_humidity  
)
```

Get values from the DHT-Sensor.

Returns the measurement value into memory. Returns NAN if sensor can't be read

Parameters

dht_temperature	pointer to a float variable to store the temperature value into
dht_humidity	pointer to a float variable to store the humidity value into

◆ get_ds18b20_value()

```
void get_ds18b20_value ( float * temperature )
```

Get values from the external DS18B20-Sensor.

Returns the measurement value into memory. Returns NAN if sensor can't be read

Parameters

dht_temperature pointer to a float variable to store the temperature value into

♦ get_ltc2943_values()

```
void get_ltc2943_values ( float * spannung,
                           float * strom,
                           uint16_t * coulomb,
                           float * temperatur,
                           float * battery_percent
                         )
```

Get values from LTC2943 battery gauge IC.

Calculates values from the raw-values and stores them into memory

Parameters

spannung pointer to a float variable to store the voltage value at shunt into
strom pointer to a float variable to store the current value over shunt into
coulomb pointer to a float variable to store the integrated coulomb into
temperatur pointer to a float variable to store the ic's temperature into
battery_percent pointer to a float variable to store the percentage SOC of the battery into

♦ gpio_setup()

```
void gpio_setup ( )
```

init all GPIO ports that are directly connected to the ESP32

This is NOT for the GPIOs on the PWM-IC!

♦ i2c_flush()

```
void i2c_flush ( )
```

Routine for flushing the i2c-interface.

switch of timer

◆ ltc_2943_standby()

```
void ltc_2943_standby ( bool state )
```

Switch the analog front-end of the LTC2943 into standby or vice versa.

To avoid write processes while readout the LTC2943 needs to be shutdown before readout and switch again afterwards

Parameters

state true for switch to standby, false to switch on

◆ luefter_steuerung()

```
void luefter_steuerung ( float temperatur )
```

Set the fan-speed according to the temperature temperature < 40 degree Celsius: Fans off temperature > 70 degree Celsius: Fans at highest speed temperature between: Fan-Speed linear to temperature.

Max Fan-Speed is limited to about 80%, due to higher system-voltage on loading than fans are rated!

Parameters

temperatur temperature as setup for the fans

◆ mcp3428_setup()

```
void mcp3428_setup ( )
```

Init the MCP3428 ADC-IC.

In fact just a general-call reset. Resets also the PWM-IC!

◆ pc_remote_mosfet()

```
void pc_remote_mosfet ( bool on )
```

Switch the mosfet to controll the pc's powerswitch on or off.

Parameters

on bool input for switch the mosfet on (true, switch pressed) or off (false, switch not pressed)

◆ pwm_setup()

```
void pwm_setup ( )
```

Initialize the pwm-controller.

Library using default B000000 (A5-A0) i2c address, and default Wire @400kHz

Resets all PCA9685 devices on i2c line

Initializes module using default totem-pole driver mode, and default disabled phase balancer

Set PWM freq to 1000Hz (default is 200Hz, supports 24Hz to 1526Hz)

◆ read_all_sensors()

```
void read_all_sensors ( struct messwerte * messwerte )
```

Read all sensors and return their current values.

Parameters

messwerte pointer to a struct to save values into

◆ relais_switch_system_on_bat()

```
void relais_switch_system_on_bat ( bool on )
```

Switch the relais 3 between using battery or external power-supply for the system.

Parameters

on bool input for setting the relais to system-battery-powered (true) or external power-supply (false)

◆ relais_switch_to_charge()

```
void relais_switch_to_charge ( bool on )
```

Switch the relais 1 & 2 between using battery for the system or charging the battery.

Parameters

on bool input for setting the relais to system-battery-powered (true) or charge the battery (false)

◆ ros_serial_send_msg()

```
void ros_serial_send_msg ( struct messwerte messwerte )
```

write the actual values to all publishers and send it out by uart

Parameters

messwerte data to send

◆ set_onboard_led()

```
void set_onboard_led ( bool on )
```

set the onboard LED on or off.

Parameters

on true = on, false = off

◆ set_rgb_led()

```
void set_rgb_led ( int rot,  
                   int gruen,  
                   int blau  
                 )
```

Set the color of the build-in-rgb-led.

Parameters

rot value of the red-channel 0-255

gruen value of the green-channel 0-255

blau value of the blue-channel 0-255

- ◆ `show_data_on_display()`

```
void show_data_on_display ( struct display_data data_to_show )
```

Refresh the display if there are newer values than already displayed.

Parameters

data_to_show struct with all data to show on display

Variable Documentation

- ◆ `first_start`

```
RTC_DATA_ATTR bool first_start = true
```

bool if the system is switched on or off, stored in deepsleep-save memory

- ◆ `relais_mode`

```
RTC_DATA_ATTR uint8_t relais_mode = 0
```

bool for init the mcp3428 only at coldstart, stored in deepsleep-save memory

- ◆ `switch_sys_off_at_time`

```
long switch_sys_off_at_time = 0
```

relaismode, stored in deepsleep-save memory

Literaturverzeichnis

- [1] Analog Devices. *LTC 2943 Datasheet*. 27. Okt. 2021. URL: <https://www.analog.com/media/en/technical-documentation/data-sheets/2943fa.pdf>.
- [2] DirectIndustry. *Laborkamera mvBlueFOX3-2*. 18. Aug. 2021. URL: <https://www.directindustry.de/>.
- [3] Waveshare INC. *Pico E-Paper 2,9 Datasheet*. 27. Okt. 2021. URL: <https://www.waveshare.com/product/pico-epaper-2.9-b.htm>.
- [4] Microchip Technology Inc. *MCP3428 Datasheet*. 27. Okt. 2021. URL: <https://www.microchip.com/content/dam/mchp/documents/OTH/ProductDocuments/DataSheets/22226a.pdf>.
- [5] PlatformIO Labs. *PlatformIO*. 27. Okt. 2021. URL: <https://platformio.org/>.
- [6] Microsoft. *Visual Studio Code*. 28. Okt. 2021. URL: <https://code.visualstudio.com/>.
- [7] NXP Semiconductors. *PCA9685 Datasheet*. 27. Okt. 2021. URL: <https://www.nxp.com/docs/en/data-sheet/PCA9685.pdf>.
- [8] Espressif Systems. *ESP32-Wroom Datasheet*. 27. Okt. 2021. URL: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf.
- [9] ThuleGmbH. *Thule official Website - Thule Canyon Extension XT*. 25. März 2020. URL: <https://www.thule.com/de-de/cargo-carrier/roof-basket-accessories/thule-canyon-extension-xt--859101>.
- [10] ThuleGmbH. *Thule official Website - Thule Canyon XT*. 25. März 2020. URL: <https://www.thule.com/de-de/cargo-carrier/roof-baskets/thule-canyon-xt--859002>.
- [11] Kevin Trelenberg. *CAMO-Demonstrator von oben*. 27. Okt. 2021. URL: <https://www.camo.nrw/camo-demonstrator-zur-visualisierung-von-algorithmen-und-sensordaten/>.
- [12] Vishay. *SIHFR023 Datasheet*. 27. Okt. 2021. URL: <https://www.vishay.com/docs/91264/sihfr024.pdf>.

- [13] ZargesGmbH. *Zarges K470 IP65*. 3. Apr. 2020. URL: https://www.zargesshop.de/contents/de/d9_ZARGES_Universalkiste_K470_IP65.html#p24.

Abbildungsverzeichnis

2.1	Der Dachkäfig Canyon XT von Thule [10].	6
2.2	Die Erweiterung Thule Canyon Extension XT von Thule[9].	6
2.3	BlueFOX3-2 Kamera von Matrix Vision[2].	7
2.4	Kamerahalter aus POM.	7
2.5	CAMO-Demonstrator von oben [11].	8
2.6	LiDAR-Sensor montiert auf Turm.	9
2.7	GNSS-Antennen.	10
2.8	K470 IP65 von der Firma Zarges [13].	11
2.9	Querschnitt der Kabeldurchführung.	12
2.10	Kabeldurchführung innen und aussen mit montiertem Lüfter.	13
2.11	Innerer Aufbau Dachkasten.	14
3.1	Stromlaufplan des Steuergeräts.	16
3.2	Außenansicht Steuergerät.	17
3.3	Innenansicht Steuergerät.	17
4.1	Schaltplan Batteriemanagment-Platine.	19
4.2	Batteriemanagment-Platine.	20
4.3	Hauptplatine Steuergerät.	21
4.4	Gehäuse des Steuergeräts.	22
4.5	Beschaltung des ESP32-SoM.	23
4.6	Beschaltung der USB-to-UART Schnittstelle.	24
4.7	Beschaltung des PCA9685.	25
4.8	Beschaltung des MCP3428.	26
4.9	Beschaltung der Displayanschlüsse.	28
5.1	Bedienoberfläche extern mit LED, Display und Taster.	30
5.2	Starten der Serial-Node auf dem PC.	31
5.3	Anzeige alles Topics.	32
5.4	Programmablaufplan.	33