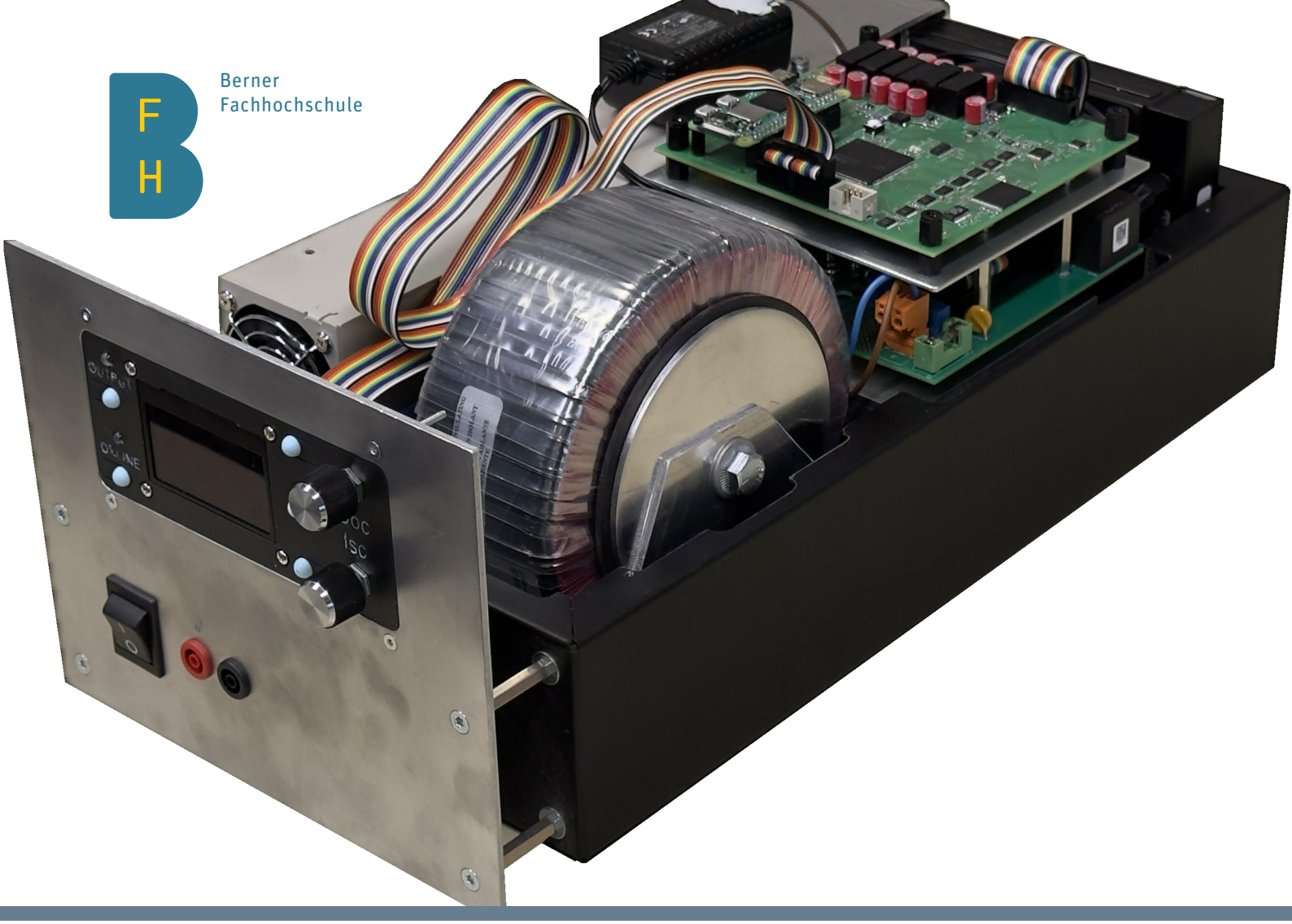
Berner
Fachhochschule



Entwicklung eines programmierbaren PV-Kennliniengenerators

BTE5540 - Bachelor Thesis

Studiengang	Elektrotechnik und Informationstechnologie
Autor	Nicola Borgna und Simon Fiechter
Mitbetreuer	Theo Zwahlen & Luciano Borgna
Experte	Prof. Dr. Christof Bucher

Version 1.1 vom 12. Juni 2025

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Abstract

Der entwickelte Photovoltaik-Modul-Simulator (PVMS) ist ein vielseitiges System zur Nachbildung des elektrischen Verhaltens von Solarmodulen unter verschiedenen, kontrollierbaren Bedingungen. Die Hardwarearchitektur ist in zwei Hauptkomponenten unterteilt: eine Steuerplatine, auf der sowohl ein STM32H757BIT6 [Mat1] als auch ein Raspberry Pi Zero 2 [Mat2] verbaut sind, und eine leistungsstarke Stromquelle, die von der Steuerplatine präzise geregelt wird.

Die Kommunikation zwischen dem STM32 und dem Raspberry Pi erfolgt über eine SPI-Schnittstelle, die eine schnelle und robuste Datenübertragung ermöglicht. Während der STM32 die zeitkritischen Aufgaben und die direkte Steuerung der Stromquelle übernimmt, ist der Raspberry Pi für die Fernsteuerungssoftware zuständig, welche dem Benutzer über ein Netzwerkinterface Zugriff auf alle relevanten Parameter bietet.

Der PVMS erlaubt sowohl eine manuelle Bedienung direkt am Gerät als auch eine Fernkonfiguration über die Softwareoberfläche. Dabei können Kurzschlussstrom (I_{sc}) und Leerlaufspannung (U_{oc}) in beiden Modi (manuell und remote) eingestellt werden. Die Einstrahlung (Irradiation) hingegen ist ausschliesslich über die Fernsteuerung einstellbar.

Ein zentrales Merkmal des Systems ist die Fähigkeit, bis zu 8.192 IV-Kurven (Kennlinien) im Speicher [Mat3] zu halten. Diese können entweder statisch, mit variabler Einstrahlung oder unter dynamischen Verschattungsbedingungen simuliert werden. Dadurch ist es möglich, realitätsnahe Szenarien wie etwa das Verhalten eines Solarmoduls bei vorbeiziehenden Wolken zu emulieren.

Das Gerät unterstützt eine maximale Ausgangsspannung von 60V sowie eine maximale Stromabgabe von 20A, was den Einsatz für eine Vielzahl von Prüf- und Entwicklungsanwendungen ermöglicht insbesondere im Bereich der MPPT-Algorithmen, Wechselrichtertests und Leistungselektronikentwicklung.

Der PVMS stellt somit eine leistungsfähige, präzise und flexibel konfigurierbare Testplattform dar, die reproduzierbare Bedingungen für die Bewertung von Photovoltaiksystemen bietet unabhängig von äusseren Einflüssen wie Wetter oder Tageszeit.

2 Einleitung

Ziel dieses Projekts ist die Entwicklung eines Simulators für Photovoltaik-Module (PVMS), der sowohl lokal als auch aus der Ferne steuerbar ist. Der PVMS soll das Verhalten realer Solarmodule unter verschiedenen Umwelt- und Betriebsbedingungen präzise nachbilden und eine flexible, reproduzierbare sowie sichere Testumgebung für unterschiedliche Anwendungen bieten.

Dieses Dokument beschreibt den Entwicklungsprozess des PVMS, von der Konzeption über die technische Umsetzung bis zu den verfügbaren Funktionen. Es wird erläutert, welche Möglichkeiten der PVMS bietet, wie er intern funktioniert und warum solche Systeme in Forschung, Entwicklung und Qualitätskontrolle von Bedeutung sind.

Ein zentraler Entwicklungsgrund liegt im Bedarf an kontrollierbaren und wiederholbaren Testbedingungen. Gerade bei der Prüfung von Wechselrichtern, MPPT-Algorithmen (Maximum Power Point Tracking) und ähnlichen Systemen ist es entscheidend, dass simulierte Parameter wie Kurzschlussstrom (Isc), Leerlaufspannung (Uoc), IV-Kennlinie, Einstrahlungsstärke oder Teilverschattung präzise und stabil bleiben.

Mit dem PVMS können diese Parameter gezielt eingestellt und verändert werden. So lassen sich Messungen unter konstanten Bedingungen durchführen und zuverlässig vergleichen, unabhängig von Tageszeit, Wetter oder Standort. Dies ist ein klarer Vorteil gegenüber Freifeldtests, bei denen Umwelteinflüsse schwer kontrollierbar sind.

Zwar existieren bereits vergleichbare Systeme am Markt, diese sind jedoch oft sehr teuer und reagieren langsam. Der in diesem Projekt entwickelte PVMS bietet eine kostengünstigere und zugleich leistungsfähigere Alternative. Er soll eine Phasendifferenz zwischen Spannung und Strom von weniger als 20 Grad bei Frequenzen bis 1 kHz einhalten und damit auch dynamische Lasten realistisch simulieren.

Das Dokument beschreibt zudem die Bedienung des PVMS über die lokale Benutzeroberfläche sowie eine speziell entwickelte Fernsteuerungssoftware. Betriebsmodi, Konfigurationsmöglichkeiten und die interne Datenverarbeitung werden detailliert dargestellt.

Der PVMS vereint technische Präzision, Benutzerfreundlichkeit und Anpassungsfähigkeit und stellt eine robuste Plattform für Forschung, Entwicklung und Validierung moderner Photovoltaiksysteme dar.

3 Einführende Theorie

In diesem theoretischen Kapitel werden grundlegende Konzepte und Vorgehensweisen beschrieben, welche in diesem Projekt verwendet wurden. Dieses Kapitel soll als Stütze dienen, wenn Konzepte im Projekt nicht verstanden wurden.

3.1 Lineare Quellen vs. getaktete Quellen

Es gibt zwei Hauptarten von Stromquellen: lineare und getaktete Quellen. Diese unterscheiden sich grundsätzlich in ihrer Funktionsweise und weisen jeweils spezifische Vor- und Nachteile auf.

Bei einer linearen Quelle wird die gewünschte Ausgangsspannung oder -stromstärke kontinuierlich angepasst, typischerweise durch einen steuerbaren Widerstand (z. B. einen linear betriebenen MOSFET). Die überschüssige elektrische Energie wird dabei in Form von Wärme im Regelbauelement (z. B. Transistor) umgewandelt. Dieser Ansatz ermöglicht eine sehr saubere Ausgangsgrösse, ist jedoch energetisch ineffizient.

Eine getaktete Quelle (auch Schaltregler genannt) arbeitet hingegen mit einer schnellen Ein-Aus-Steuerung (PWM). Die Energie wird in kurzen Pulsen übertragen und anschliessend durch Induktivitäten, Kondensatoren und Filterelemente geglättet. Dieses Verfahren ist wesentlich effizienter, erzeugt aber zwangsläufig hochfrequente Störungen (Ripple), die gefiltert werden müssen.

Tabelle 3.1 zeigt die wichtigsten Merkmale beider Typen im Kontext dieses Projekts:

Tabelle 3.1: Vergleich zwischen linearen und getakteten Quellen

Eigenschaft	Lineare Quelle	Getaktete Quelle
Wirkungsgrad	Niedrig	Sehr hoch
Komplexität	Niedrig	Hoch
Ripple / Rauschen	Sehr gering	Hoch (benötigt Filter)
Reaktionsgeschwindigkeit	Sehr hoch	Hoch

Obwohl lineare Quellen einen deutlich geringeren Wirkungsgrad aufweisen, bieten sie Vorteile, die in bestimmten Anwendungen wie dem vorliegenden PVMS entscheidend sind:

- **Ripple und Rauschen:** Lineare Quellen erzeugen nur minimale Restwelligkeit und Störungen. Dies ist besonders wichtig bei empfindlichen Systemen wie Messsystemen oder beim Testen von Wechselrichtern und MPPTs, die durch hochfrequente Störungen beeinträchtigt oder sogar beschädigt werden könnten.

- **Reaktionsgeschwindigkeit:** Lineare Quellen reagieren sehr schnell auf Laständerungen, da sie keine PWM-Erzeugung benötigen. Die PWM-Erstellung wäre ein zusätzlicher Rechenschritt, der bei jeder Spannungsänderung erfolgen müsste und somit das System verlangsamen würde.

Wie bereits erwähnt, ist der geringe Wirkungsgrad der linearen Quelle ein Nachteil. Das bedeutet, dass die zugeführte elektrische Energie deutlich höher ist als die abgegebene. Ein grosser Teil der Energie wird als Wärme abgeführt, was zu erhöhtem Kühlaufwand, grösseren Abmessungen und höheren Kosten führt. Dennoch wurde für den PVMS eine lineare Quelle eingesetzt um die oben genannten Vorteile auszuschöpfen.

3.2 Einfluss der Einstrahlungsstärke auf Solarzellen

Die elektrische Leistung von Solarzellen hängt wesentlich von der auf sie einfallenden Einstrahlungsstärke ab. Änderungen der Einstrahlung führen zu einer nahezu linearen Variation des Kurzschlussstroms I_{sc}, während sich auch die Leerlaufspannung U_{oc} in geringerem Ausmass verändert, dies ist in Abbildung 3.1 zu sehen.

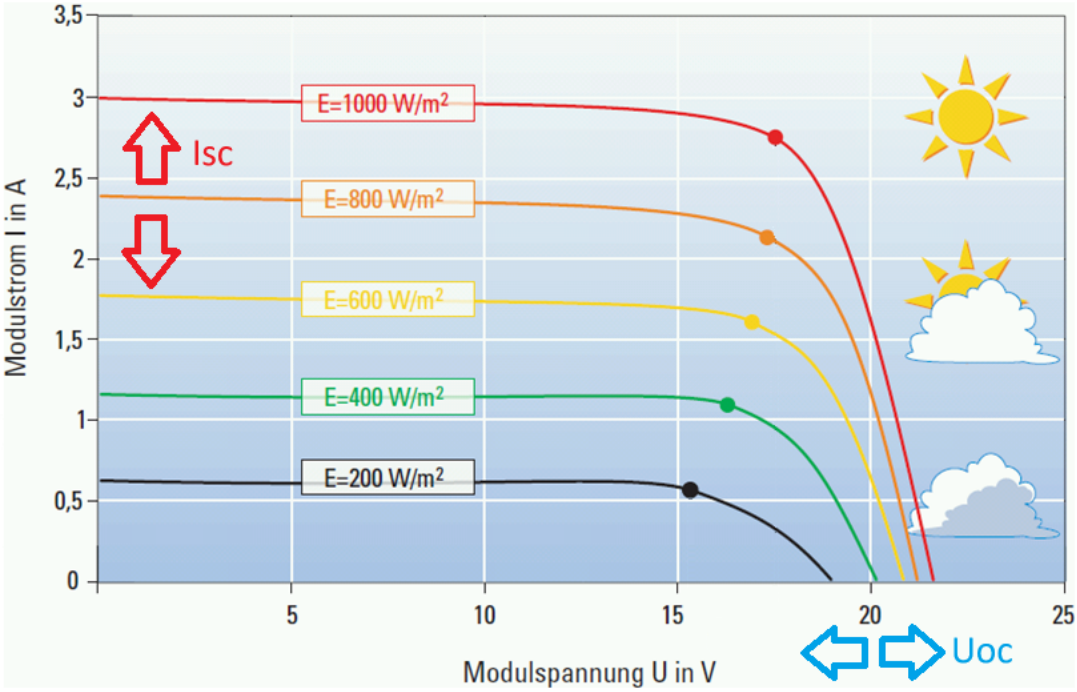


Abbildung 3.1: Abhängigkeit von Kurzschlussstrom und Leerlaufspannung von der Einstrahlungsstärke [Bil1]

Da beide Parameter direkt die Betriebsbedingungen eines photovoltaischen Systems beeinflussen, ist es wichtig, ihre Abhängigkeit von der Einstrahlungsstärke zu berücksichtigen.

3.3 SPI Kommunikation

Die beiden auf der Steuerplatine integrierten Steuergeräte (der STM32 und der Raspberry Pi Zero 2W) kommunizieren über eine SPI-Schnittstelle. In diesem Abschnitt werden die Gründe für diese Wahl erläutert und Beispiele für jede mögliche Art der Kommunikation zwischen den beiden vorgestellt.

3.3.1 Auswahl der Schnittstelle

Für die digitale Kommunikation stehen auf dem Markt zahlreiche Schnittstellen zur Verfügung. Ziel ist es, eine Schnittstelle auszuwählen, die allen Anforderungen die dieses Projekt mit sich bringt abzudecken.

Um eine fundierte Entscheidung zu treffen, müssen zunächst alle Anforderungen an die Kommunikation definiert werden. Die wichtigsten Parameter dabei sind die maximale Datenrate sowie die technische Komplexität der Implementierung. Da in diesem Projekt grosse Datenmengen übertragen werden, ist eine hohe Übertragungsgeschwindigkeit entscheidend, um eine effiziente Kommunikation sicherzustellen.

Die Tabelle 3.2 zeigt einen Vergleich der Schnittstellen, die sowohl vom STM32 als auch vom Raspberry Pi unterstützt werden:

Tabelle 3.2: Vergleich verschiedener Kommunikationsschnittstellen

Schnittstelle	Maximale Datenrate	Hardwarekomplexität
Ethernet[Lit1]	200 bis 400 Gbit/s	Sehr hoch
CAN[Lit2]	Bis 1 Mbit/s	Niedrig
USB[Lit3]	5 bis 20 Gbit/s	Hoch
SPI[Lit4]	10 bis 20 Mbit/s	Niedrig
I²C[Lit5]	400 kbit/s bis 3,4 Mbit/s	Mittel
UART[Lit6]	0.110 bis 115.2 kbit/s	Niedrig

Nach dem quantitativen Vergleich der verfügbaren Schnittstellen ist es ebenso wichtig, die praktischen Aspekte jeder Option zu bewerten. Im Folgenden werden die Vor- und Nachteile der einzelnen Schnittstellen qualitativ betrachtet:

- **Ethernet:** Ein Industriestandard und grundsätzlich eine sehr leistungsfähige Wahl. Allerdings ist die Dokumentation zur Implementierung auf dem STM32 begrenzt, was die Entwicklung deutlich erschwert.
- **CAN:** Weit verbreitet in der Automobilindustrie, jedoch mit relativ niedriger Übertragungsrate. Zudem müsste für die Nutzung auf dem Raspberry Pi ein zusätzliches Hardwaremodul installiert werden.

- **USB:** Bietet sehr hohe Datenraten und ist weit verbreitet. Die Implementierung auf dem STM32 ist jedoch hardwaretechnisch aufwendig und daher komplexer.
- **SPI:** Ermöglicht eine hohe Übertragungsgeschwindigkeit und ist sowohl auf dem STM32 als auch auf dem Raspberry Pi leicht zu implementieren. Damit stellt SPI eine attraktive Lösung dar.
- **I²C:** Im Vergleich zu anderen Schnittstellen eher langsam.
- **UART:** Sehr einfach in der Handhabung und Implementierung, jedoch mit begrenzter Datenrate, wodurch es sich nicht für grosse Datenmengen eignet.

Nach Abwägung aller Optionen fiel die Entscheidung auf die SPI-Schnittstelle. Sie erfüllt die Anforderungen hinsichtlich Geschwindigkeit und Verfügbarkeit und lässt sich auf beiden Plattformen unkompliziert implementieren.

3.3.2 SPI Einstellungen

Damit der SPI auf dem Raspberry Pi wie auch auf dem STM32 funktioniert muss der SPI in den richtigen Modus eingestellt werden. Es gibt 4 Modis beim SPI, mit CPOL und CPHA werden diese Modis beschrieben. CPOL ist die polarisation des Ruhenden clocks und CPHA beschreibt das einlesen der Daten. Genauer ist dies in der Tabelle 3.3 beschrieben.

Tabelle 3.3: Modi des SPI-Bus

SPI-Modus	CPOL	CPHA	Beschreibung
Mode 0	0	0	Takt ruht auf Low, Daten werden bei steigender Flanke übernommen
Mode 1	0	1	Takt ruht auf Low, Daten werden bei fallender Flanke übernommen
Mode 2	1	0	Takt ruht auf High, Daten werden bei fallender Flanke übernommen
Mode 3	1	1	Takt ruht auf High, Daten werden bei steigender Flanke übernommen

Zusätzlich müssen verschiedene Parameter für die SPI-Kommunikation festgelegt werden: Bitreihenfolge, Datenrate, Datenrahmenlänge sowie die Rollenverteilung (Master/Slave).

Die Bitreihenfolge wurde auf MSB-first gesetzt. Die Datenrate wurde vorerst auf 100 kHz begrenzt. Dieser Wert stellt jedoch nicht die maximale Übertragungsgeschwindigkeit dar die endgültige Datenrate kann in späteren Testphasen durch praktische Messungen ermittelt werden.

Die Datenrahmenlänge beträgt 8 Bit, da der verwendete SPI-Treiber [Lit7] auf dem Raspberry Pi (bei Verwendung von Python) derzeit nur 8-Bit-Datenwörter unterstützt.

Der Raspberry Pi wurde als Master definiert, während der STM32 die Rolle des Slave übernimmt.

Der SPI-Bus unterstützt sowohl den Full-Duplex- als auch den Half-Duplex-Modus: Im Full-Duplex-Modus erfolgt das gleichzeitige Senden und Empfangen von Daten über zwei separate Leitungen (MOSI und MISO). Im Half-Duplex-Modus hingegen findet die Datenübertragung abwechselnd statt es wird entweder gesendet oder empfangen, wobei nur eine Datenleitung benötigt wird.

In diesem Projekt fiel die Wahl auf den Full-Duplex-Modus, da dieser die Implementierung auf dem STM32 vereinfacht. Aus technischer Sicht würde jedoch auch der Half-Duplex-Modus ausreichen: Zunächst sendet der Master (Raspberry Pi) Daten an den Slave, anschliessend folgt eine zweite Übertragung, bei der Daten vom Slave gelesen werden. Die Kommunikation erfolgt also sequenziell, wird aber technisch im Full-Duplex-Modus realisiert.

Abbildung 3.2 zeigt schematisch die Datenübertragung im Full-Duplex-Betrieb:

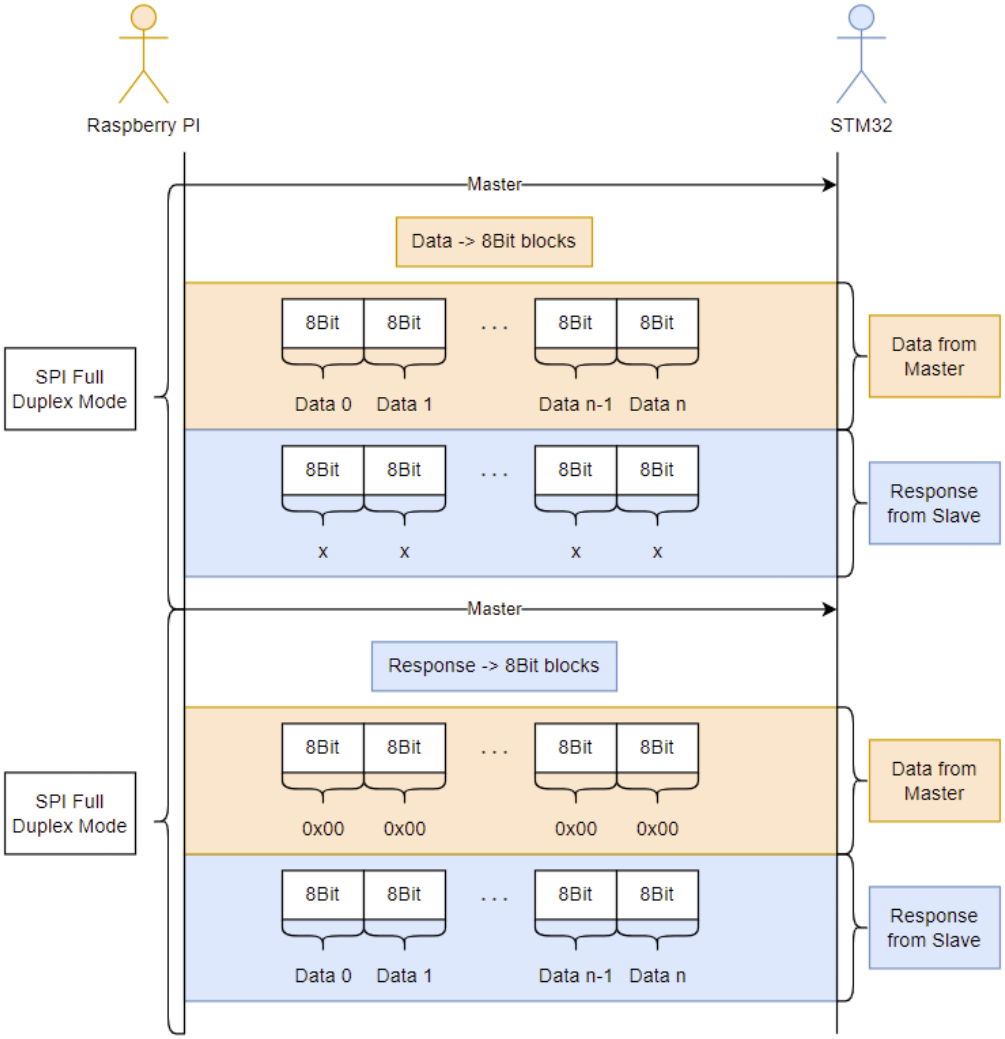


Abbildung 3.2: Full-Duplex-Übertragung im SPI-Modus

In diesem Projekt wurde der SPI-Modus 0 gewählt.

3.4 Masse und Floating-Masse

In diesem Kapitel wird das Verfahren der Anschlussmassen beschrieben. Das system ist galvanisch isoliert und um Masseschleifen zu vermeinden muss ein verfahren der Masseverbindung gewählt werden.

Die Masse, die auf der Steuerkarte und der Steuerplatine verwendet wird, ist nicht identisch mit der Masse des Gesamtsystems. Nun werden die beiden möglichen Ansätze für die Massereferenz in einem Multistring-System erläutert und begründet, weshalb im vorliegenden Design eine Floating-Masse gewählt wurde.

Viele Multistring-Wechselrichter verwenden eine gemeinsame Masseverbindung auf der Negativseite der PV-Strings. Ein solcher Aufbau ist in Abbildung 3.3 dargestellt und stellt eine einfache und etablierte Lösung dar.

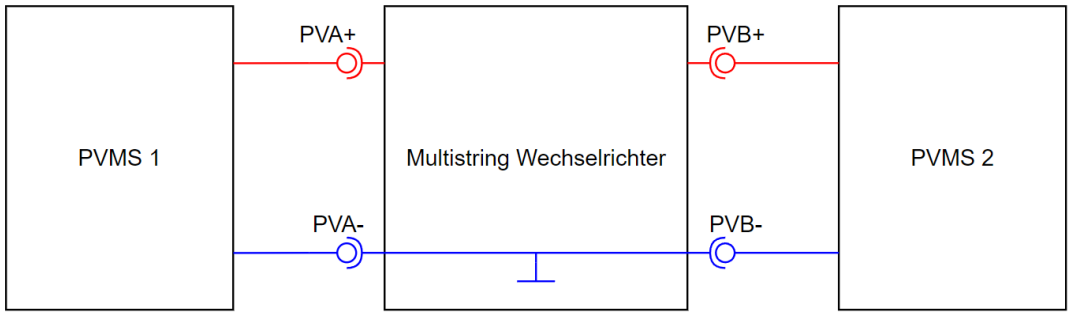


Abbildung 3.3: Multistring-System mit gemeinsamer negativer Masse

Dieser Ansatz führt zu einer Konfiguration wie in Abbildung 3.4 gezeigt. Dabei wird die Stromregelung über eine Stromsenke auf PV- realisiert. Die einzelnen Strings teilen sich eine gemeinsame Masse, was in den meisten Anwendungen gut funktioniert.

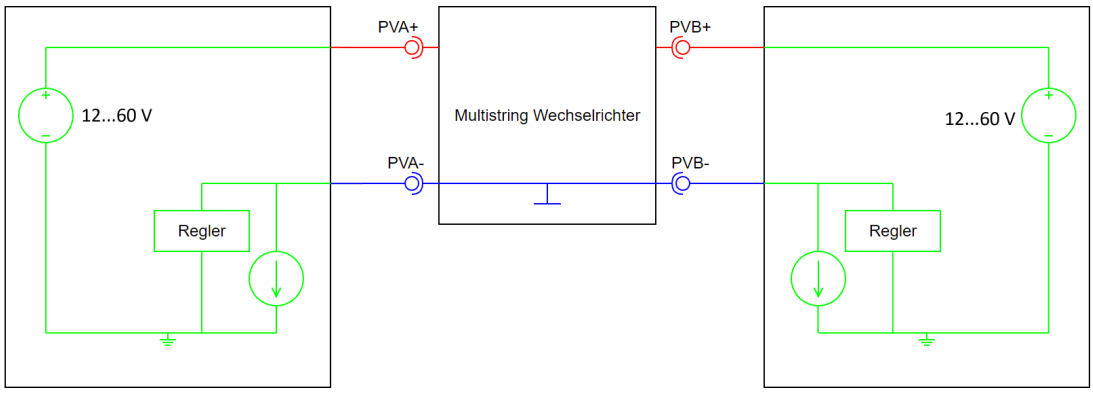


Abbildung 3.4: Topologie mit gemeinsamer Masseverbindung über PV-

Eine alternative Methode besteht darin, jedem PVMS eine eigene, vom Gesamtsystem galvanisch getrennte Masse zuzuweisen, eine sogenannte Floating-Masse. In dieser Konfiguration wird der Strom über eine Quelle an PV+ eingespeist statt über eine Senke aus PV- gezogen. Ein solcher Aufbau ist elektrisch aufwendiger, bietet jedoch vollständige Trennung zwischen den Strings.

Wie in Abbildung 3.5 dargestellt, ist es dabei essenziell, dass die Floating-Masse im PVMS auf PV+ bezogen ist und keinesfalls mit PV- verbunden wird. Ein direkter Masseanschluss würde in diesem Fall zu einem Kurzschluss führen, da zwischen den Bezugspotentialen eine Spannung von bis zu 60 V anliegen kann, was zu erheblichen Schäden am System führen würde.

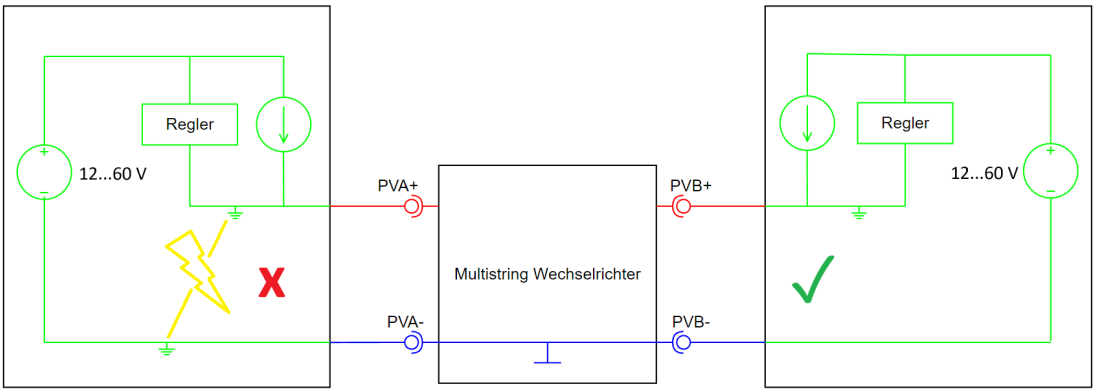


Abbildung 3.5: Multistring-System mit Floating-Masse pro String

Beide Ansätze (gemeinsame Masse oder Floating-Masse) sind grundsätzlich technisch umsetzbar und haben jeweils spezifische Vor- und Nachteile. Die Variante mit gemeinsamer

Masse ist einfacher zu implementieren, während die Floating-Masse eine höhere Unabhängigkeit zwischen den Strings ermöglicht.

Im diese Projekt wurde bewusst die Floating-Masse gewählt, da alle bisherigen Versionen dieses Geräts auf diesem Konzept basierten. Das PV-Labor verfügt daher über umfangreiche Erfahrung und Vertrauen in diese Architektur, obwohl sie elektrisch komplexer ist und keinen unmittelbaren funktionalen Vorteil gegenüber einer gemeinsamen Masse bietet.

3.5 Echtzeitbetriebssystem (RTOS) und FreeRTOS

3.5.1 Was ist ein RTOS

Ein Echtzeitbetriebssystem (Real-Time Operating System, RTOS) ist ein Betriebssystem, das speziell dafür entwickelt wurde, mehrere Aufgaben (Tasks) so zu verwalten, dass bestimmte zeitliche Anforderungen zuverlässig eingehalten werden. Im Gegensatz zu herkömmlichen Betriebssystemen wie Windows oder Linux ist ein RTOS für schnelle und vorhersehbare Reaktionen auf externe Ereignisse optimiert, typischerweise in eingebetteten Systemen mit strengen Echtzeitanforderungen.

Ein zentrales Konzept bei einem RTOS ist die pseudo-parallele Ausführung von Tasks. Das bedeutet, dass bei einem Ein-Kern-System jeweils nur ein Task aktiv ausgeführt wird, der Scheduler des RTOS jedoch so schnell zwischen den Tasks wechselt (Context Switch), dass der Eindruck entsteht, mehrere Tasks würden gleichzeitig laufen. In Systemen mit mehreren Kernen ist echte Parallelität möglich, jedoch bleibt die zeitliche Steuerung durch das RTOS entscheidend.

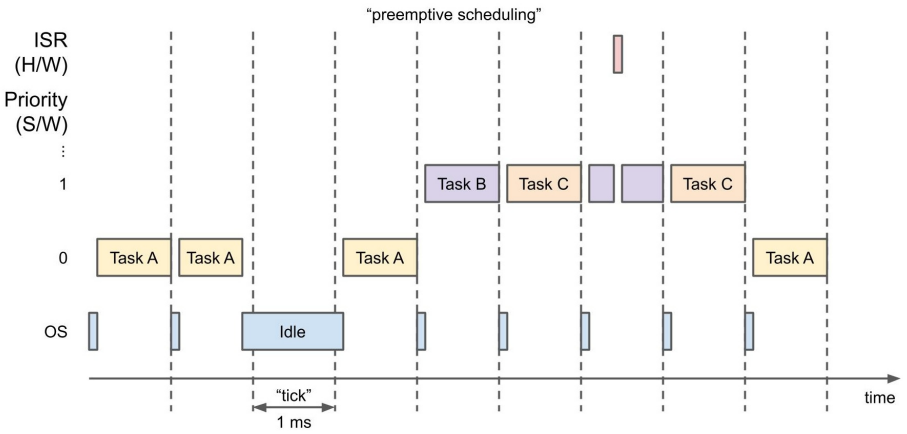


Abbildung 3.6: Pseudo-parallele Ausführung von Tasks in einem RTOS.[Bil2]

Abbildung 3.6 veranschaulicht das Prinzip der pseudo-parallelen Ausführung, bei dem die CPU-Zeit scheinbar gleichzeitig mehreren Tasks zugewiesen wird.

3.5.2 Hauptmerkmale eines RTOS

Ein RTOS stellt in der Regel die folgenden zentralen Funktionen bereit:

- **Scheduler:** Entscheidet, welcher Task als Nächstes ausgeführt wird.
- **Prioritätsverwaltung:** Tasks haben unterschiedliche Prioritäten; höher priorisierte Tasks können andere unterbrechen (Preemption).
- **Synchronisationsmechanismen:** Semaphore, Mutexes und Queues zur Kommunikation und Koordination zwischen Tasks.
- **Zeitverwaltung:** Funktionen für exakte Zeitsteuerung, Delays, Timeouts usw.
- **Determinismus:** Vorhersehbares Verhalten, insbesondere bei Reaktionszeiten.

3.5.3 FreeRTOS

FreeRTOS ist eines der weltweit am häufigsten verwendeten Echtzeitbetriebssysteme im Bereich Embedded Systems. Es handelt sich um ein leichtgewichtiges, portables Open-Source-Betriebssystem mit den folgenden Eigenschaften:

- **Kompakter Kernel:** Geeignet für Mikrocontroller mit begrenztem Speicher (RAM/ROM).
- **Hohe Portabilität:** Unterstützt mehrere Mikrocontroller-Architekturen, z. B. ARM Cortex-M, RISC-V usw.
- **Gute Dokumentation und breite Community.**

Dank seiner einfachen Struktur und Flexibilität lässt sich FreeRTOS leicht in Embedded-Projekte integrieren. Tasks können einfach erstellt, pausiert, gelöscht und synchronisiert werden.

3.5.4 Warum ist FreeRTOS so weit verbreitet

Die Beliebtheit von FreeRTOS lässt sich durch mehrere Faktoren erklären:

- **MIT-Lizenz:** Frei verwendbar, auch in kommerziellen Projekten.
- **Geringe Komplexität:** Der Kernel ist überschaubar und leicht verständlich.
- **Grosse Community und viele Lernressourcen:** Tutorials, Beispielprojekte und Bibliotheken sind weit verbreitet.
- **Gute Integration in gängige Entwicklungsumgebungen:** z. B. STM32CubeIDE, MPLAB X, Atmel Studio usw.

3.5.5 Einsatz in diesem Projekt

Auf dem Markt existieren zahlreiche RTOS-Varianten, darunter Zephyr, ThreadX, embOS, Micrium, ChibiOS, RIOT u. a. In diesem Projekt wurde jedoch FreeRTOS eingesetzt, da es Teil des Ausbildungsprogramms ist und die Studierenden bereits mit seiner Struktur und den zugehörigen API-Funktionen vertraut sind. Diese Entscheidung ermöglicht es, sich stärker auf die inhaltliche Umsetzung des Projekts zu konzentrieren, ohne zusätzliche Einarbeitungszeit für ein neues Betriebssystem einplanen zu müssen.

3.6 Zwischenkernkommunikation

Der Mikrocontroller STM32H757BIT6 ist ein Dual-Core-Baustein, der auf einer heterogenen Multiprozessor-Architektur (AMP) basiert, die zwei Kerne integriert: einen leistungsstarken ARM Cortex-M7 und einen ARM Cortex-M4 mit geringem Stromverbrauch. Eine typische Anforderung an Systeme, die auf dieser Architektur basieren, ist die Fähigkeit, Daten und Befehle zwischen den beiden Kernen auf effiziente, asynchrone und sichere Weise auszutauschen. Zu diesem Zweck bietet STMicroelectronics Unterstützung für das OpenAMP-Framework, das das RPMsg-Protokoll (Remote Processor Messaging) nutzt, um einen hochrangigen Kommunikationsmechanismus zwischen den Kernen zu realisieren.

3.6.1 OpenAMP

OpenAMP (Open Asymmetric Multi-Processing) ist ein Open-Source-Rahmenwerk, das entwickelt wurde, um die Verwaltung der Kommunikation zwischen Kernen in AMP-Architekturen zu erleichtern. Sein Hauptzweck ist die Abstraktion der Komplexität der Kommunikation zwischen Prozessoren durch die Verwendung von standardisierten und portablen Mechanismen. OpenAMP bietet die folgenden Funktionalitäten:

- Inbetriebnahme und Synchronisierung von Sekundärkernen
- Verwaltung der Nachrichtenübermittlung zwischen Kernen über RPMsg
- Unterstützung für VirtIO-Geräte als virtuelle Kommunikationskanäle
- Verwaltung von Interrupts zwischen Kernen

Im STM32-Kontext wird OpenAMP über CMSIS und STM32Cube implementiert und in die Firmware integriert, insbesondere in den Bibliotheken libOpenAMP.a

3.6.2 RPMsg

RPMsg (Remote Processor Messaging) ist ein High-Level-Kommunikationsprotokoll, das im OpenAMP-Framework enthalten ist. Es ermöglicht zwei Prozessoren (in diesem Fall den M7- und M4-Kernen) den asynchronen Austausch von Nachrichten. RPMsg basiert auf VirtIO und dem vring-Konzept (virtueller Ringpuffer) und nutzt gemeinsame Speicherstrukturen und Interrupt-Mechanismen, um die Kommunikation zu gewährleisten.

Ein RPMsg-Kanal wird durch einen logischen Namen identifiziert und erlaubt es, strukturierte Pakete von einem Kern zu einem anderen zu senden. Die Nachrichten können bidirektional sein, und die Infrastruktur verwaltet automatisch Empfangs- und Sendewarteschlangen.

3.6.3 Kommunikationsarchitektur

Die Kommunikation zwischen den beiden Kernen basiert auf der folgenden mehrstufigen Architektur:

- **Gemeinsamer Speicher (Shared RAM):** von beiden Kernen sichtbarer Bereich, der als Austauschraum dient.
- **IPCC (Inter-Processor Communication Controller):** Hardwaremodul, das Interrupts erzeugt, um eine eingehende Nachricht zu melden.
- **VRING (Virtual Ring Buffer):** zwei Ringpuffer (einer in jeder Richtung), die zur Verwaltung der Nachrichtenwarteschlangen verwendet werden.
- **RPMsg:** verarbeitet formatierte Nachrichten.
- **OpenAMP:** bietet eine High-Level-API für das Senden, Empfangen und Registrieren von Remote-Diensten.

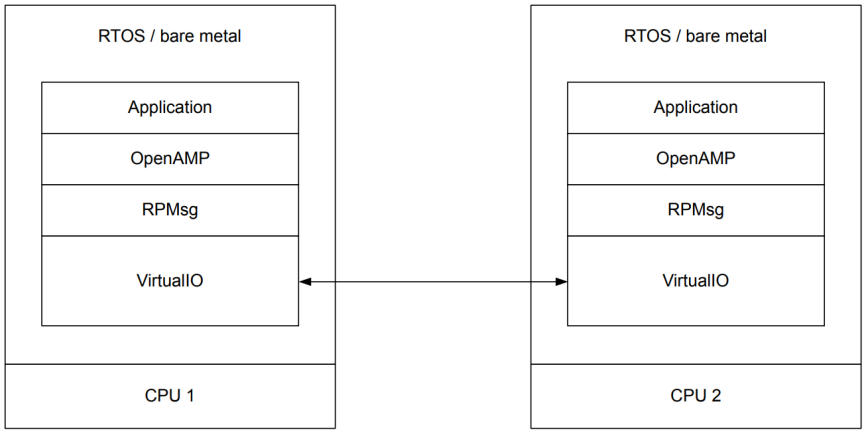


Abbildung 3.7: Diagramm der Kommunikationsarchitektur zwischen den Kernen [Bil3]

3.6.4 Sequenz der Kommunikation

Die Kommunikationssequenz kann gemäss den Anweisungen im STMicroelectronics-Dokument AN5617 [Lit8] realisiert werden. Dieses Dokument ist ein Leitfaden für die Kommunikation zwischen zwei Kernen und enthält alle Informationen, die für die Implementierung eines solchen Systems erforderlich sind. Abbildung 3.8 zeigt ein vereinfachtes Schema des gesamten OpenAMP-Systems mit RPMsg.

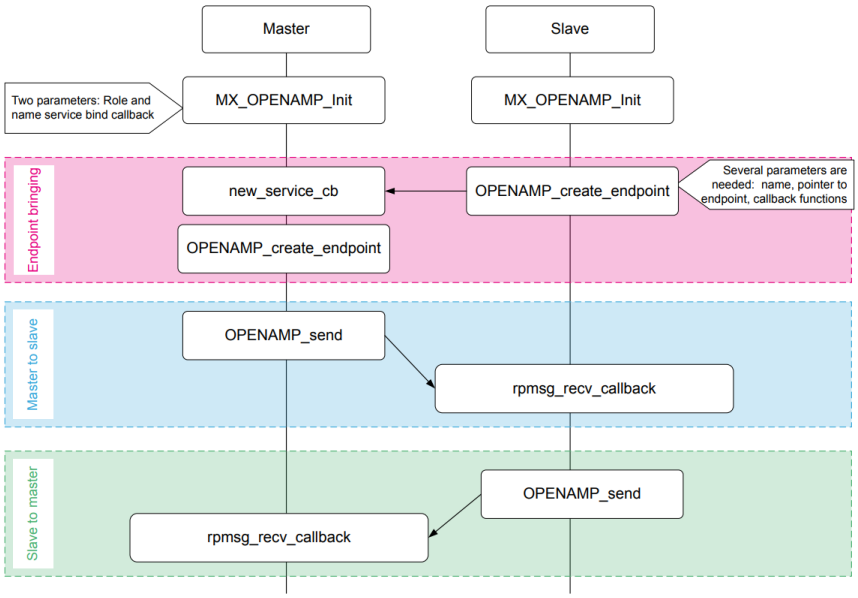


Abbildung 3.8: Verlauf einer typischen Kommunikation zwischen den Kernen [Bil4]

4 Konzept

Das Grundkonzept des PVMS besteht darin, das Verhalten eines Photovoltaikmoduls so realitätsnah wie möglich zu simulieren. Wie in Abbildung 4.1 zu sehen ist, besteht das System aus einem Leistungselektronik-Teil, einer Steuerkarte, die den Strom in Abhängigkeit von der Ausgangsspannung regelt, und einer dritten Platine, über die der Benutzer mit dem Gerät interagieren kann: Sie ermöglicht sowohl die Anpassung der Betriebsparameter als auch die Anzeige der aktuellen Systemwerte. Zusätzlich wird ein programmierbares Netzgerät mit einem Ausgangsbereich von 12 V bis 60 V verwendet, um Verluste zu minimieren, sowie ein 1:1-Transformator zur galvanischen Trennung vom Stromnetz.

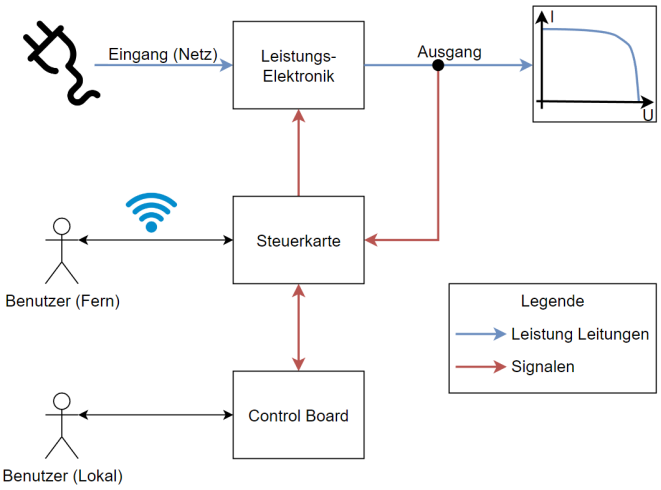


Abbildung 4.1: Überblick über das System

Ein grosser Teil der Hardware sowie das Grundkonzept wurden vom PV-Labor entwickelt. In Tabelle 4.1 ist ersichtlich, welche Hardwarekomponenten von wem umgesetzt wurden.

Tabelle 4.1: Hardwareteilen des PVMS

Teil	Hersteller
Stromquelle	PV-Labor
Steuerkarte	PV-Labor
Control Board	Studierenden

Die Studierenden wurden bei der Hardwareentwicklung weniger beteiligt, sie lieferten jedoch Beistand bei der Wahl des Mikrokontrollers, welcher sie im Anschluss programmierten.

4.1 Speisung des Systems

Das System wird auf zwei Ebenen mit Strom versorgt, die beide über einen Trenntransformator mit dem Netz verbunden sind, um eine galvanische Trennung vom Netz zu ermöglichen. Zum einen ist das der Leitstungsteil und zum anderen ist das der Regelungsteil.

Der Leistungsteil wird anhand eines programmierbaren Netzgerät [Mat4] betrieben. Der Regelungsteil wird über ein 5V AC/DC-Netzteil [Mat5] gespiesen (Kapitel 5.2.1).

4.2 Stromquelle

Die Stromquelle steuert das programmierbare Netzgerät und bestimmt den fliessenden Strom. Diese Printplatte wurde im Rahmen dieses Projektes entwickelt. Zusammen mit der Steuerkarte(Siehe Kapitel 5.1) bildet sie den PVMS.

4.3 Steuerkarte

Die Steuerkarte ist für die Regelung der Stromquelle verantwortlich. Zudem übernimmt sie die Messung der Ausgangsspannung und des Ausgangsstroms. Auf dieser Karte befinden sich ein *STM32H757BIT6* sowie ein *Raspberry Pi Zero 2 W*.

Um dem Benutzer die Interaktion mit der Steuerkarte zu ermöglichen, wurde eine Software entwickelt, die auf dem Raspberry läuft und über eine WLAN-Verbindung erreichbar ist. Zusätzlich ist eine physische Benutzeroberfläche an der Vorderseite des Geräts angebracht.

Die Steuerkarte ist in der Lage, bis zu 8192 Kennlinien zu speichern. Abbildung 4.2 zeigt eine vereinfachte Darstellung des Regelkreises.

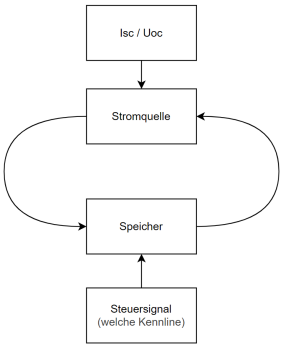


Abbildung 4.2: Vereinfachte Darstellung des Regelkreises

Die Steuerkarte legt fest, welche Kennlinie aus dem Speicher geladen werden soll und welche zugehörigen Werte für Leerlaufspannung (U_{oc}) und Kurzschlussstrom (I_{sc}) gelten (siehe Kapitel 5.1).

4.4 Control Board

Das Control Board erlaubt es den PVMS im Handbetrieb zu steuern. Auch dieses Board wurde im Rahmen dieses Projektes entwickelt. Weitere Details sind im Kapitel 5.3 zu finden.

4.5 Steuerungsmethoden

Das System kann auf zwei Arten gesteuert werden:

- **Über den Raspberry Pi:** In diesem Modus können sowohl statische als auch dynamische Kennlinien simuliert werden, z. B. durch Variation der Bestrahlungsstärke oder durch Wechseln der Kurve während des Betriebs. Ausserdem können über den Raspberry Pi [Mat2] neue Kennlinien in den Speicher geladen werden.
- **Über die Steuerplatine an der Frontseite (Control Board):** Hier ist es möglich, vordefinierte (statische) Kennlinien zu simulieren, die sich bereits im Speicher befinden. Eine Interaktion mit dem Raspberry Pi ist dabei nicht notwendig.

In beiden Modi können der Kurzschlussstrom (I_{sc}) und die Leerlaufspannung (U_{oc}) vom Benutzer frei gewählt werden.

Das Konzept muss nun in Hardware umgesetz werden. In den folgenden Kapiteln wird anschliessend die Umsetztung beschrieben.

5 Hardware

In diesem Kapitel wird die im Projekt eingesetzte Hardware im Detail beschrieben. Ziel ist es, einen umfassenden Überblick über den physikalischen Aufbau des Systems und die wichtigsten verwendeten Komponenten zu geben.

Die im Kapitel dargestellten Schaltungsausschnitte wurden teilweise vereinfacht, indem Messpunkte und andere Teile der Schaltung weggelassen wurden, um die Übersichtlichkeit und das Verständnis zu erleichtern. Die vollständigen Schaltpläne sind im Anhang (2,3, 4) zu finden.

Die Hardware besteht aus drei Leiterplatten (PCBs):

- **Steuerkarte**: Eine Leiterplatte, auf der ein STM32-Mikrocontroller [Mat1] und ein Raspberry Pi Zero 2 W [Mat2] verbaut sind. Diese Karte übernimmt die zentrale Steuerung und Kommunikation des Systems.
- **Stromquelle**: Eine lineare Stromquelle, die den Ausgangsstrom für den Simulator bereitstellt. Aufgrund ihrer linearen Bauweise entstehen dabei hohe Energieverluste und Wärmeentwicklung.
- **Control Board**: Eine Frontplatte mit Bedienelementen, die im vorderen Bereich des Geräts montiert ist. Sie ermöglicht die manuelle Steuerung des Systems durch den Benutzer.

Zur Unterstützung des Systems kommen ausserdem ein 1:1 Isolationstransformator [Mat6] zum Einsatz, der eine galvanische Trennung von 4 KV zwischen dem Netz und dem Simulator gewährleistet und die Kopplungskapazität auf etwa 1 ηF reduziert, sowie ein programmierbares Netzgerät [Mat4], das so geregelt wird, dass der Spannungsabfall über den LeistungsMOSFETs der Stromquelle minimiert wird.

Da es sich bei der Stromquelle um eine lineare Ausführung handelt, sind die Energieverluste erheblich, was zu einer beträchtlichen Wärmeentwicklung führt. Um einen stabilen und sicheren Betrieb des Systems zu gewährleisten, wurde daher eine Wasserkühlung [Mat7] implementiert.

Abbildung 5.1 zeigt eine vereinfachte schematische Darstellung des Gesamtsystems mit den wichtigsten Blöcken.

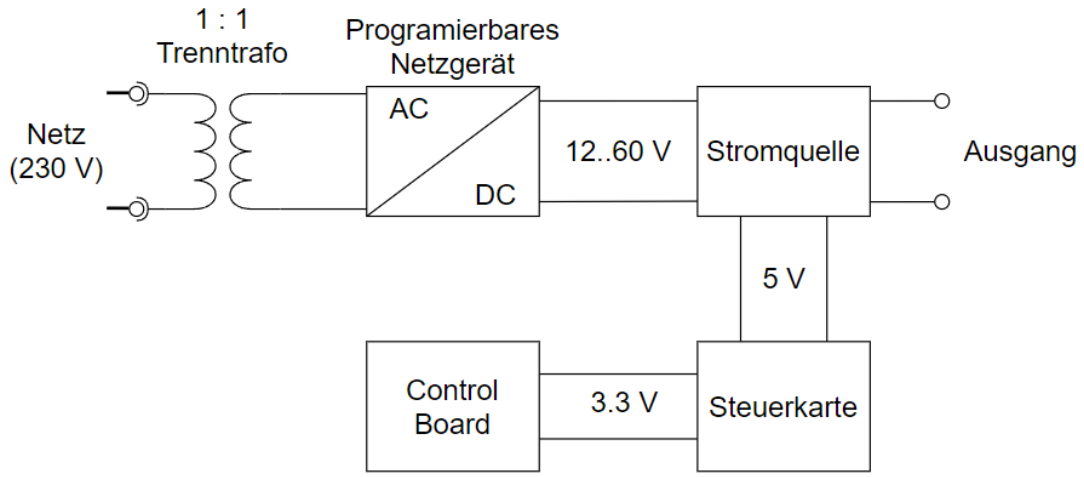


Abbildung 5.1: Vereinfachte schematische Darstellung des Gesamtsystems

5.1 Steuerkarte

In diesem Kapitel wird die Funktionsweise der Steuerkarte beschrieben, welche für die dynamische Regelung der Stromquelle basierend auf der vom Photovoltaic Module Simulator (PVMS) gelieferten Ausgangsspannung verantwortlich ist. Die Ausgangsspannung des PVMS, welche das Verhalten eines photovoltaischen Generators simuliert, wird zunächst auf ein zum Steuersystem kompatibles Niveau skaliert und anschliessend von einem Analog-Digital-Wandler (ADC) digitalisiert.

Die Referenzspannung des ADC wird durch einen Digital-Analog-Wandler (DAC) erzeugt und stellt die Leerlaufspannung U_{OC} des simulierten PV-Generators dar. Der digitalisierte Eingangswert wird genutzt, um eine Speicheradresse zu selektieren: Der parallele Ausgang des ADC bildet die erste Hälfte der Adresse, während die zweite Hälfte, welche die aktuell gewählte Kennlinie (IV-Kurve) identifiziert, vom STM32 gesteuert wird.

Der an der selektierten Speicheradresse abgelegte Wert entspricht dem Strom, den die Stromquelle bei der gemessenen Eingangsspannung liefern soll. Die Kennlinien sind in normierter Form mit einer Auflösung von 12 Bit im Speicher abgelegt, mit Adressen und Werten bis maximal 3932. Um sicherzustellen, dass der Strom bei $U = U_{OC}$ tatsächlich null ist, werden die Werte linear bis zur maximalen Adresse 4096 und zum maximalen Wert 4096 ergänzt, um Bauteiltoleranzen zu kompensieren.

Der aus dem Speicher gelesene Wert wird schliesslich durch einen DAC wieder in ein analoges Signal umgewandelt und dient zur Ansteuerung der Stromquelle. Die Normierung der Kennlinien ermöglicht es, dieselbe Kurve für beliebige Kombinationen von I_{SC} und U_{OC} zu verwenden, indem lediglich die Referenzspannungen der DACs angepasst werden.

Die Hauptkomponenten der Steuerkarte sind in Abbildung 5.2 dargestellt und in Abbildung 5.3 zu sehen. Durch Anklicken der gewünschten Bereiche auf den folgenden Abbildungen können die entsprechenden Kapitel direkt aufgerufen werden. Eine detailliertere Übersicht bietet die Abbildung im Anhang 5.

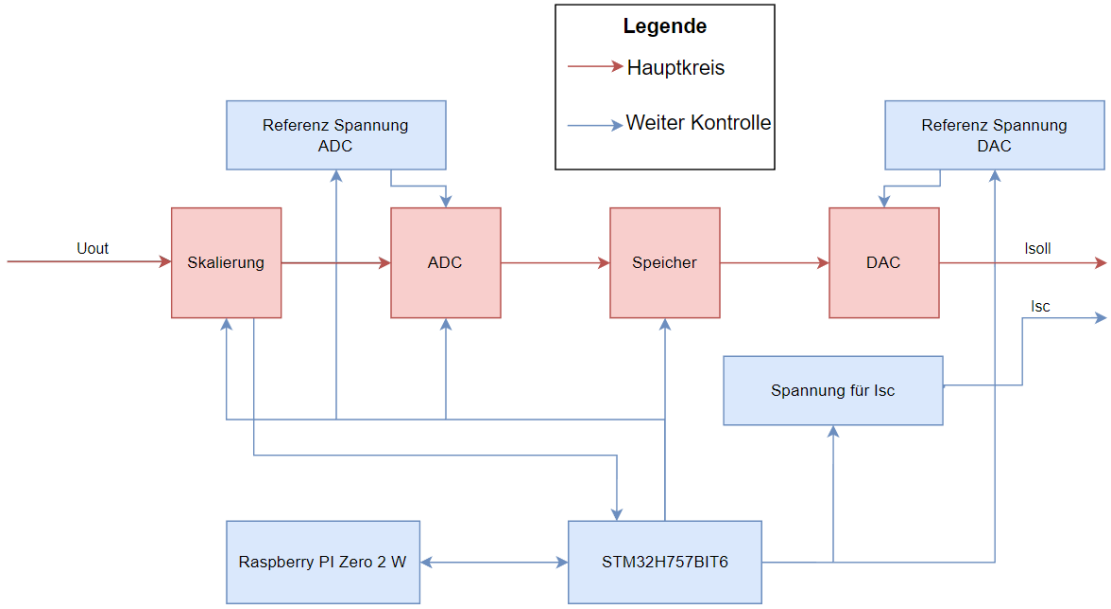


Abbildung 5.2: Vereinfachte schematische Darstellung der Steuerkarte (mit links)

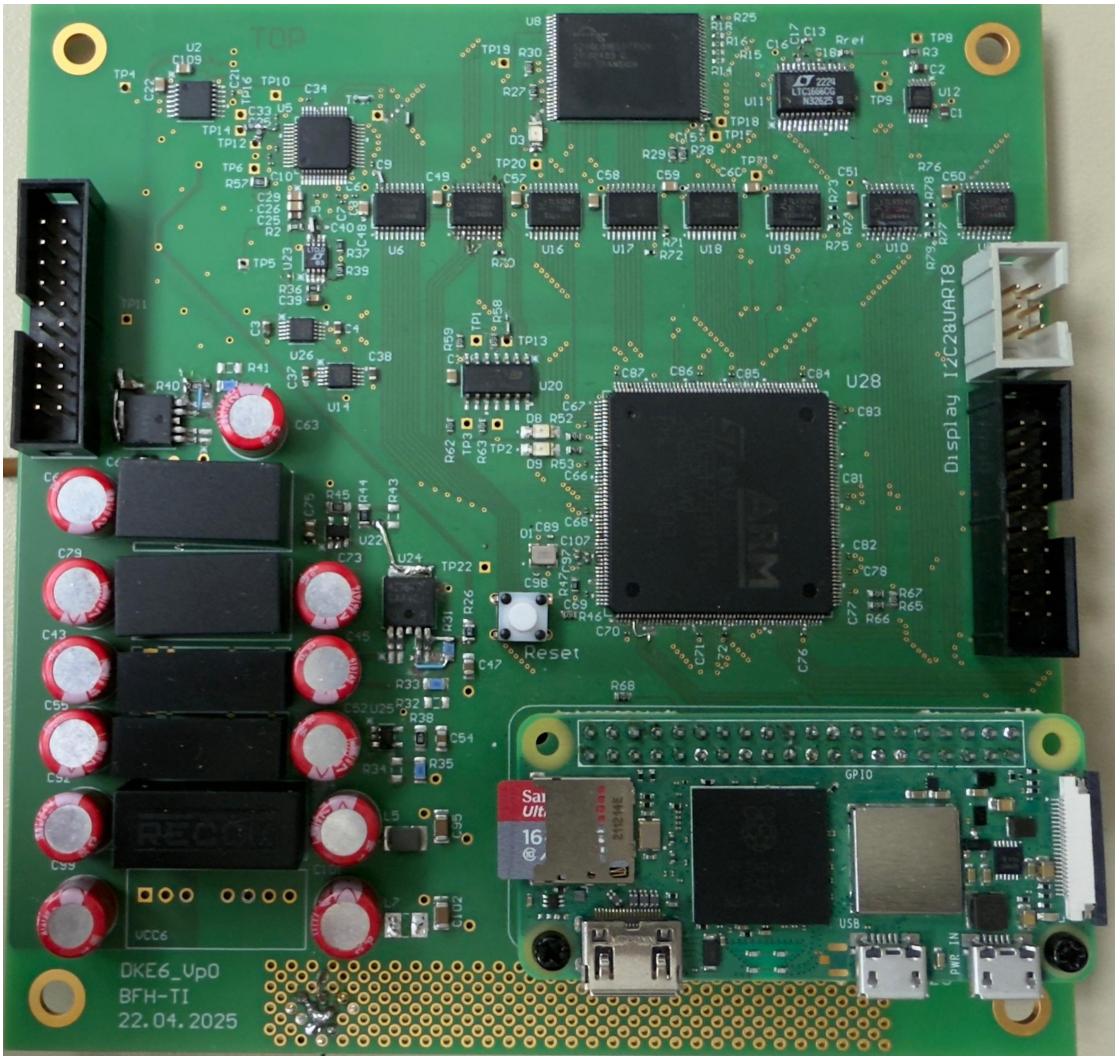


Abbildung 5.3: Echte Steuerkarte (mit links)

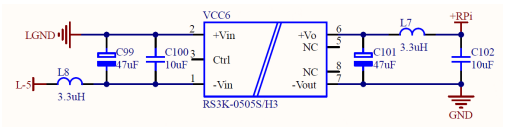
5.1.1 Isolierte DC-DC Wandler

Aus den in Kapitel 3.4 erläuterten Gründen ist es notwendig, dass die Massen gut getrennt sind. Zu diesem Zweck wurden isolierte DC-DC-Wandler verwendet, um die erforderlichen Spannungen für die Steuerkarte zu erzeugen. Auf diese Weise erhält die gesamte Platine eine eigene, von der Masse des Geräteausgangs bzw. der Stromversorgung getrennte Masse.

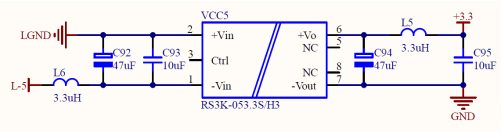
Es wurden insgesamt 6 DC-DC-Wandler verwendet. Diese erzeugen die folgenden Spannungen:

- **+5 V / GND:** Diese Spannungsversorgung wird verwendet, um den Raspberry Pi Zero 2 W mit Energie zu versorgen. (Abbildung 5.4a)
- **+3,3 V / GND:** Diese Spannung ist für die Versorgung des STM32-Mikrocontrollers vorgesehen. (Abbildung 5.4b)
- **+12 V / -12 V:** Dieses symmetrische Spannungsniveau wird zur Versorgung eines Teils der Operationsverstärker (OPAMP) verwendet (Abbildung 5.4c).
- **+5 V / -5 V:** Auch dieses symmetrische Spannungsniveau wird zur Versorgung einiger OPAMPs verwendet. (Abbildung 5.4d)

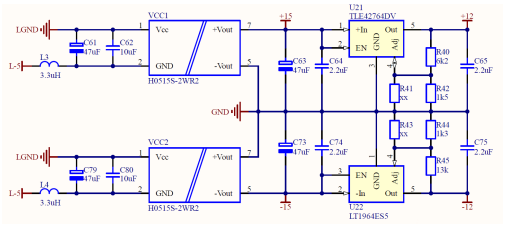
Da es sich bei diesen Wandlern um Pulsweitenmodulation-Wandler (PWM) handelt, ist es notwendig, die Ausgangsspannung zu filtern. Abbildung 5.4 zeigt die verschiedenen Wandler mit dem entsprechenden Filter.



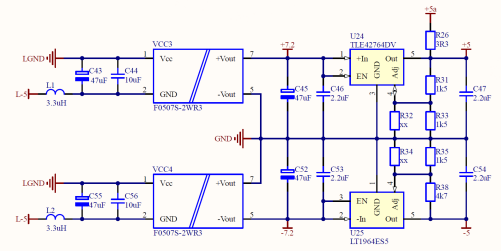
(a) Speisung für den Raspberry Pi



(b) Speisung für den STM32



(c) Simmetrische 12V Speisung



(d) Simmetrische 5V Speisung

Abbildung 5.4: Unterschiedliche DC-DC Wandler auf der Steuerkarte

5.1.2 STM32H757BIT6

Der STM32H757BIT6 ist ein Mikrocontroller (MCU) von STMicroelectronics auf Basis einer Dual-Core-Architektur mit einem ARM Cortex-M7 (bis zu 480 MHz) und einem ARM Cortex-M4 (bis zu 240 MHz). Er bietet eine Vielzahl von Schnittstellen (z. B. SPI, I²C, UART, CAN, Ethernet, USB), verfügt über bis zu 168 GPIOs und ist im 208-Pin-LQFP-Gehäuse untergebracht. Die Wahl dieser MCU wurde massgeblich durch die Dual-Core-Fähigkeit beeinflusst, mit dem Ziel, den Cortex-M4-Kern für die Kommunikation mit dem Raspberry Pi zu verwenden, ohne dabei die Echtzeitverarbeitung des Cortex-M7-Kerns durch z. B. SPI-Interrupts zu beeinträchtigen.

Aufgrund von Herausforderungen bei der Firmwareentwicklung wird in diesem Projekt jedoch vorerst nur ein Kern verwendet. Für zukünftige Projekte oder Erweiterungen bleibt die Möglichkeit bestehen, eine Dual-Core-Version der Firmware zu implementieren.

Der Mikrocontroller übernimmt die Steuerung der auf der Steuerkarte integrierten Peripheriegeräte, die wiederum die Stromquelle regeln. Zu diesen Peripheriegeräten gehören: der Spannungsskalierung (Kapitel 5.1.5), ein ADC (Kapitel 5.1.6), drei I²C-DACs (Kapitel 5.1.8), eine externer Speicher (Kapitel 5.1.9), sechs Busschalter (Kapitel 5.1.10), sowie ein Control Board (Kapitel 5.3). Die Firmware-Architektur zur Ansteuerung dieser Komponenten ist in Kapitel 7 beschrieben.

5.1.3 Raspberry Pi Zero 2 W

Der Raspberry Pi Zero 2 W [Mat2] ist ein kompakter Einplatinencomputer, der in diesem Projekt als zentrales Element für die Fernsteuerung des Systems dient. Er ist leistungsfähig genug für die Ausführung von grafischen Benutzeroberflächen und gleichzeitig platz- und energieeffizient, was ihn für eingebettete Anwendungen ideal macht.

Hardware-Spezifikationen

Das Modell Zero 2 W basiert auf dem Broadcom BCM2710A1 SoC, einem Quad-Core ARM Cortex-A53 Prozessor mit einer Taktfrequenz von bis zu 1 GHz. Der Arbeitsspeicher besteht aus 512 MB LPDDR2 SDRAM, was für typische Steuerungs- und Kommunikationsaufgaben im Embedded-Bereich ausreichend ist. Zur Kommunikation stehen neben WLAN 802.11 b/g/n auch Bluetooth 4.2 sowie eine USB-2.0-Schnittstelle über einen Micro-USB-OTG-Port zur Verfügung. Die Versorgung erfolgt typischerweise über 5V über denselben Micro-USB-Port. Für dieses Projekt war ein eigenes, auf dem Board integriertes 5 V Netzteil geplant (Abbildung 5.4a), das aber aufgrund zu hoher Verbrauchsspitzen nicht ausreichte, so dass man sich entschloss, den Raspberry über die USB-Buchse zu versorgen. Ein Mini-HDMI-Ausgang ermöglicht zudem den direkten Anschluss eines Monitors, beispielsweise zu Debugging-Zwecken.

Systemintegration und Aufgaben

Im Rahmen dieses Projekts übernimmt der Raspberry Pi Zero 2 W die Aufgabe der Fernsteuerung des Gesamtsystems. Er läuft unter einem Linux-basierten Betriebssystem (Raspberry Pi OS) und stellt eine grafische Benutzeroberfläche zur Verfügung, die per Virtual Network Computing (VNC) über das Netzwerk zugänglich ist. Damit kann das Gerät aus der Ferne überwacht und gesteuert werden, ohne dass eine direkte physische Verbindung notwendig ist.

Zur Kommunikation mit dem eigentlichen Steuercontroller (STM32)[Mat1] verfügt der Raspberry Pi über eine SPI-Schnittstelle (Kapitel 3.3). Diese ermöglicht den Datenaustausch zwischen Steuerungssoftware auf dem Raspberry Pi und der Embedded-Firmware des STM32, etwa zur Übergabe von Sollwerten oder zur Protokollierung von Messdaten.

Für Entwicklungs- und Debuggingzwecke stehen drei LEDs zur Verfügung, die direkt über GPIOs angesteuert werden. Eine vierte LED ist optional vorgesehen und kann bei Bedarf ergänzt werden. Diese LEDs dienen zur Signalisierung von Systemzuständen oder Fehlerereignissen und stellen somit ein einfaches, aber effektives Werkzeug zur Laufzeitdiagnose dar.

5.1.4 SPI-Protokoll

In diesem Kapitel wird das für den PVMS entwickelte SPI-Protokoll beschrieben. Um das Protokoll zu definieren, müssen zunächst die wichtigsten Eigenschaften analysiert werden. Das Protokoll muss in der Lage sein, verschiedene Datenarten wie Kennlinien, Strom, Spannung, Kennliniennummern sowie die IP-Adresse vom Master an den Slave zu übertragen.

Hierfür wurde eine einfache Struktur entworfen, die sich zunächst im Wartezustand (Wait State) befindet. Wird ein Startbefehl empfangen, führt der Slave die entsprechende Funktion basierend auf dem Befehlscode aus.

Der Startbefehl folgt stets dem gleichen Schema: Zunächst wird ein 8-Bit-Steuerbefehl übertragen, der festlegt, welche Aktion der Slave ausführen soll. Je nach Befehlsart folgen anschliessend zusätzliche Nutzdaten, beispielsweise für die Übertragung von Kennliniendaten.

Abbildung 5.5 zeigt exemplarisch den Aufbau eines solchen Startbefehls am Beispiel einer Loading-Übertragung:

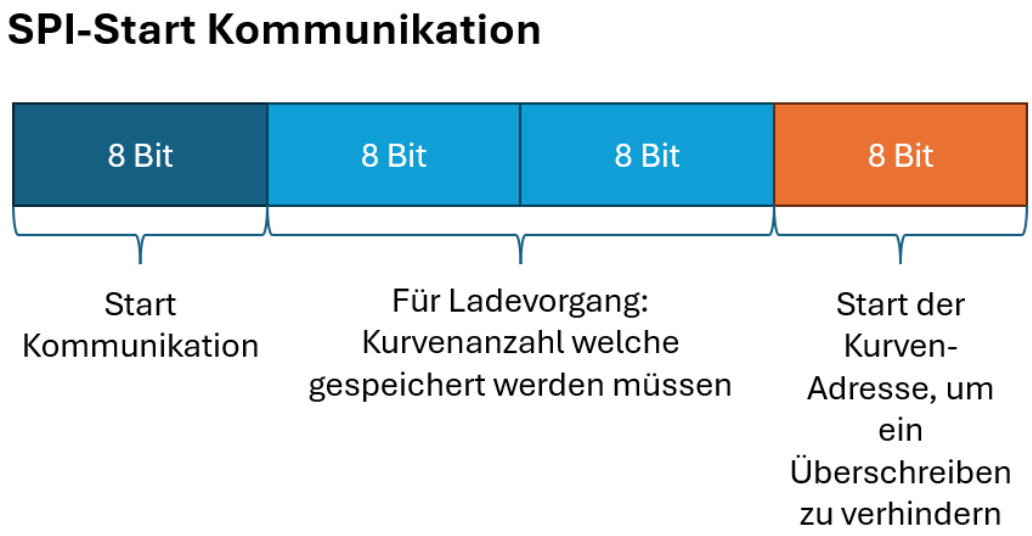


Abbildung 5.5: Startbefehl der SPI-Übertragung für eine Loading-Übertragung

In den folgenden Abschnitten werden die verschiedenen Funktionen des Protokolls im Detail beschrieben.

Connection

Die Funktion *Connection* wird verwendet, um eine Verbindung zum STM32 herzustellen. Dabei sendet der Master ein Startkommando mit dem Zeichen “C”. Der Slave interpretiert dieses Startsignal als Verbindungsanforderung und antwortet mit einem READY-Signal (“R”).

Sobald der Master das READY-Signal empfangen hat, überträgt er seine IP-Adresse an den Slave. Der Slave empfängt die IP-Adresse und sendet diese zur Verifikation erneut an den Master zurück.

Nach erfolgreicher Bestätigung der IP-Adresse ist die Verbindung hergestellt und der Slave kehrt zurück in den Wait-State.

Der Ablauf dieser Kommunikation ist in der Abbildung 5.6 schematisch dargestellt:

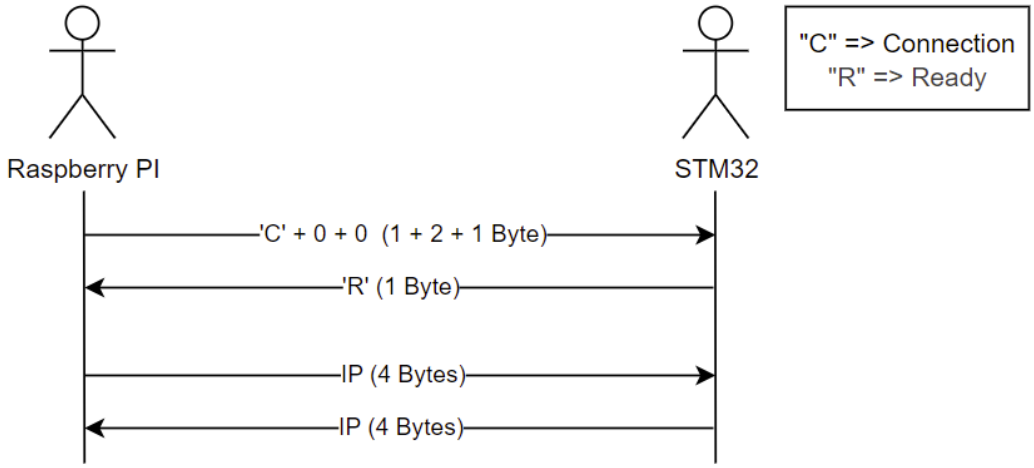


Abbildung 5.6: Verbindungsaufbau via SPI

Loading

Die Funktion *Loading* ist für die Übertragung der Kennlinien vom Master zum Slave verantwortlich. Zu Beginn wird, wie bei allen Befehlen, ein Startsignal gesendet. In diesem Fall ist es das Zeichen “L”. Dieses Signal enthält zusätzliche Informationen, die vom Slave ausgewertet werden:

- Die Anzahl der zu übertragenden Kennlinien
- Die Startadresse, unter der die erste Kennlinie gespeichert werden soll

Anhand dieser Informationen kann der Slave ermitteln, wie viele Speicherblöcke gelöscht werden müssen. Die Angabe der Startadresse dient ausserdem dazu, den Bereich der fest gespeicherten statischen Kennlinien im Speicher zu schützen, dieser darf nicht überschrieben werden.

Die Startadresse ist auf Seiten des Raspberry Pi (Master) konfigurierbar. Wie diese Einstellung vorgenommen wird, wird in einem späteren Abschnitt beschrieben 6.3.8.

Sobald der Slave das Startsignal “L” empfangen hat, antwortet er mit einem WAIT-Signal. Während dieser Wartezeit darf der Master keine weiteren Daten senden, da der Slave mit dem Löschen des Speichers beschäftigt ist. Der Speicher kann nur in Blöcken von 128kB gelöscht werden. Dieser Vorgang benötigt Zeit und wird durch deaktivierte Interrupts abgesichert, wodurch eine SPI-Kommunikation währenddessen nicht möglich ist.

Nach Abschluss des Löschvorgangs sendet der Master das Startsignal erneut. Der Slave antwortet nun mit einem READY-Signal, welches anzeigt, dass er bereit für die Datenübertragung ist.

Nun beginnt die eigentliche Übertragung der Kennliniendaten:

Jede Kennlinie wird zusammen mit zusätzlichen Informationen übertragen:

- Die Zielspeicheradresse
- Ein Status-Byte

Das Status-Byte dient zur Steuerung des Ablaufs. Es zeigt unter anderem an, ob weitere Kennlinien folgen oder ob die Übertragung abgeschlossen ist.

Nach jeder übertragenen Kennlinie sendet der Slave eine Echo-Antwort mit den empfangenen Daten zurück. Der Master vergleicht die Daten zur Verifikation. Stimmen die Daten überein, wird die nächste Kennlinie übertragen. Dieser Ablauf wiederholt sich, bis alle Kennlinien vollständig übertragen wurden.

Um die Funktion Loading ordnungsgemäss zu beenden, sendet der Master im Status-Byte das Zeichen “D” (für Done). Der Slave quittiert den Abschluss mit einem KILL-Signal (“K”) und kehrt anschliessend wieder in den Wartezustand (Wait State) zurück.

Der gesamte Ablauf dieser Datenübertragung ist in der folgenden Abbildung dargestellt:

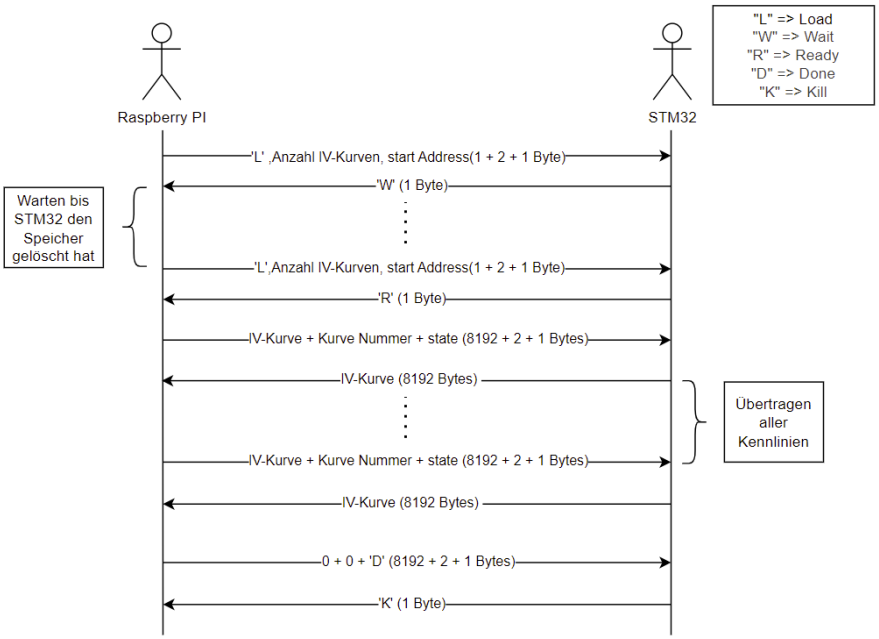


Abbildung 5.7: Datenübertragung mit Loading

Running

Die Funktion Running wird während des Betriebs verwendet und dient der aktiven Steuerung des Systems. Über diese Kommunikation werden folgende Werte vom Master an den Slave übertragen:

- Die aktive Kennliniennummer
- Der momentane Kurzschlussstrom (I_{sc})
- Die momentane Leerlaufspannung (U_{oc})
- Ein State-Byte, das den momentane Zustand oder Kontrollbefehle (z. B. Start, Ende) definiert

Zu Beginn sendet der Master ein Startsignal mit dem Kennzeichen “B” (für Betrieb). Der

Slave erkennt dieses Signal als Startbefehl und aktiviert seinen Ausgang. Daraufhin sendet er ein READY-Signal (“R”) zurück an den Master.

Während der laufenden Übertragung sendet der Master in einem Intervall von 500 ms aktuelle Betriebsdaten: die Kennliniennummer, den Kurzschlussstrom, die Leerlaufspannung sowie ein State-Byte. Der Slave empfängt diese Werte, interpretiert das State-Byte und stellt die entsprechenden Parameter ein.

Zusätzlich antwortet der Slave dem Master mit zwei Messwerten:

- Der aktuell fliessende Strom
- Die aktuell anliegende Spannung

Dieser Übertragungszyklus wird kontinuierlich wiederholt, bis der Master im State-Byte den Status “D” (für Done) setzt. Der Slave erkennt dadurch das Ende der Kommunikation und sendet ein KILL-Signal (“K”) zurück. Anschliessend kehrt der Slave in den Wait-State zurück.

Der Ablauf dieser Kommunikation ist in der folgenden Darstellung schematisch abgebildet:

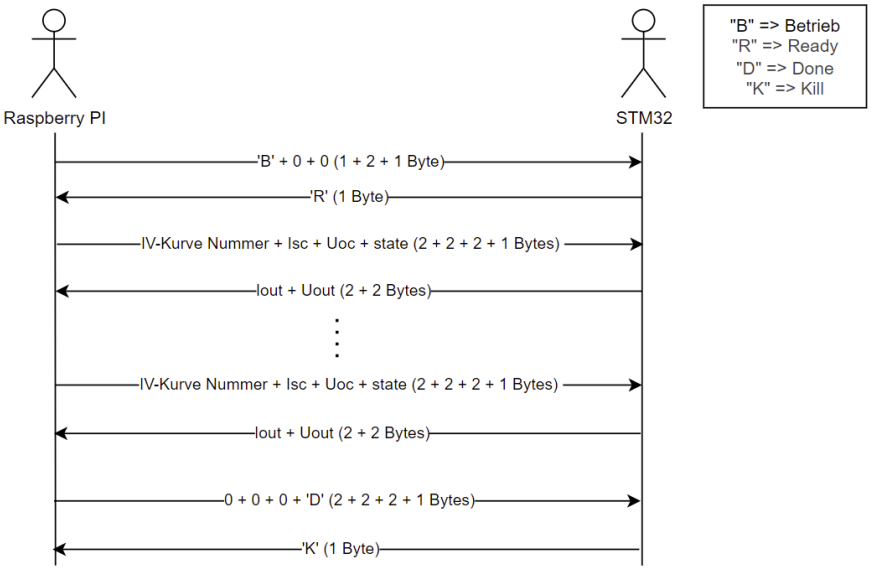


Abbildung 5.8: Datenübertragung im Betriebsmodus

Somit können Raspberry-Pi und STM32 miteinander Kommunizieren. Weiter werden Hardware-Spezifikationen beschrieben.

5.1.5 Spannungsskalierung

Um die Ausgangsspannung des PVMS (bis zu 60 V) mit dem internen ADC der MCU (STM32) messen zu können, muss diese Spannung zunächst skaliert werden. Die skalierung dient zum Schutz des MCUs. Die Spannung wird von 60 V auf 2.4 V maximal herungter skaliert. Angesichts des breiten möglichen Spannungsbereichs wurde ein mehrstufiger Ansatz implementiert:

Zunächst erfolgt eine Grobskalierung über einen invertierenden Verstärker. Diese Wahl ermöglicht es, auch Spannungen korrekt zu verarbeiten, die bei der Messung zwischen PV- und PV+ als Masse sonst negativ wären. Die Verstärkung des Verstärkers ergibt sich aus den Schaltungswerten wie folgt:

$$A_v = \frac{R_1}{R_4 + R_5} = \frac{10\ k\Omega}{620\ \Omega + 61.9\ k\Omega} \approx 0,16 \tag{5.1}$$

Die Spannung wird somit auf etwa 0.16 des Ursprungswertes reduziert. Das bedeutet, dass eine maximale Eingangsspannung von 60 V auf ca. 9,6 V skaliert wird.

Da diese Spannung jedoch weiterhin über dem maximalen Referenzwert des internen ADC (2,5 V) liegt, wird zusätzlich ein mehrstufiger Spannungsteiler eingesetzt. Über einen Multiplexer kann dabei jeweils ein passender Teilerpfad aktiviert werden, wie in Abbildung 5.9 gezeigt.

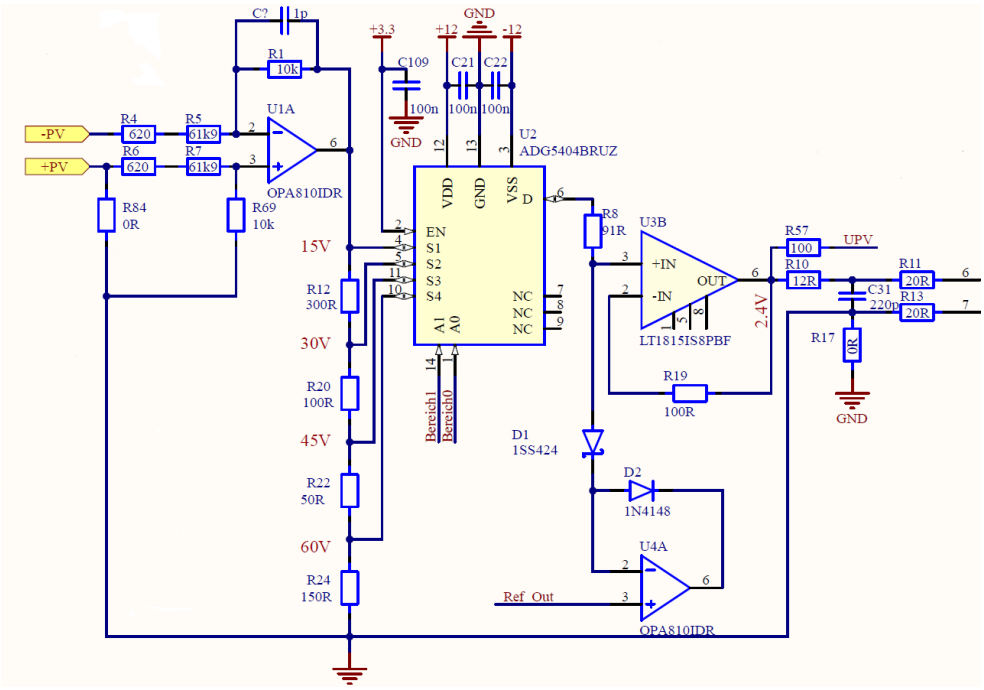


Abbildung 5.9: Mehrstufiger Spannungsteiler mit Multiplexer

Dank dieser Architektur kann die Spannung am Ausgang des Multiplexers stets auf einen Wert zwischen 0 V und 2,4 V begrenzt werden, einfach durch Auswahl des geeigneten Bereichs. Da dieser Bereich bekannt ist, kann die gemessene Spannung korrekt zurückgerechnet (reskaliert) werden.

Ziel dieser Lösung war es, eine möglichst hohe relative Genauigkeit auch bei niedrigen Spannungen zu gewährleisten. Wäre die Spannung lediglich einmalig auf den Maximalwert (60 V) skaliert worden, wären die Spannungsschritte der ADCs (sowohl des internen STM32-ADCs als auch des externen zur Adressierung des Speichers) stets gleich gross geblieben, was bei kleinen Eingangsspannungen zu einem deutlich schlechteren relativen Auflösungsvermögen geführt hätte.

Die verschiedenen Teilbereiche des Spannungsteilers und die daraus resultierenden Spannungsauflösungen sind in Tabelle 5.1 dargestellt.

Bereich	A_0	A_1	**12-Bit ADC Auflösung**	**16-Bit ADC Auflösung**
0-15 V	0	0	3 mV	0,2 mV
15-30 V	0	1	6 mV	0,4 mV
30-45 V	1	0	9 mV	0,6 mV
45-60 V	1	1	12 mV	0,8 mV

Tabelle 5.1: Bereiche des Spannungsteilers

Ohne diese stufenweise Skalierung wären unabhängig von der tatsächlich anliegenden Spannung lediglich Spannungsschritte von 12 mV (für den 12-Bit ADC) bzw. 0,8 mV (für den 16-Bit ADC) möglich gewesen, was die Messgenauigkeit besonders im unteren Spannungsbereich erheblich reduziert hätte.

Es ist jedoch wichtig zu beachten, dass diese Genauigkeitsangaben rein theoretisch sind. In der Praxis wird die effektive Auflösung durch den vorhandenen Signalrauschpegel begrenzt, der in der Regel grösser ist als die idealen Spannungsschritte der ADCs.

Hinsichtlich der Signalverarbeitung sind die Ausgänge der Spannungsteiler wie folgt verbunden:

- Der Signalleiter nach dem Widerstand R57 (UPV) führt zum internen ADC des STM32 (16 Bit).
- Die Ausgänge nach R11 und R13 führen zum positiven bzw. negativen Eingang des externen Differenz-ADCs (12 Bit), welcher zur Bestimmung der Speicheradresse verwendet wird.

5.1.6 AD Wandler

In diesem Project werden zwei Arten von AD-Wandlern verwendet:

- die im STM32 integrierten 16-Bit-Wandler, die zur Messung von I_{out} und U_{out} verwendet werden
- der externe 12-Bit-Wandler, der für die automatische Adressierung des Speichers basierend auf der Ausgangsspannung des Systems zuständig ist

Da der erste Wandler direkt im STM32 integriert ist, wird er nicht als separate Hardwarekomponente betrachtet. Der zweite AD-Wandler ist ein ADS7881 [Mat8] von Texas Instruments mit einer Auflösung von 12 Bit und einer Abtastrate von bis zu 4 MSps. Dieser Wandler erhält die Ausgangsspannung vom im Kapitel 5.1.5 beschriebenen Stufenwahlschaltkreis und dient dazu, die aktuelle Position auf der IV-Kennlinie zu ermitteln. Die Ausgänge dieses Wandlers sind parallel geschaltet und direkt mit den unteren 12 Adressleitungen des Speichers verbunden (siehe Kapitel 5.1.9).

Da jedoch die maximale Spannung Uoc nicht konstant ist, muss auch die Referenzspannung entsprechend angepasst werden. Diese Referenzspannung wird durch einen vom Mikrocontroller gesteuerten DA-Wandler bereitgestellt (siehe Kapitel 5.1.7). Dadurch benötigen die Kennlinien im Speicher stets den gleichen Adressbereich und können für verschiedene Simulationen verwendet werden, bei denen Isc und Uoc beliebig verändert werden.

Der AD-Wandler führt eine Messung durch, sobald eine steigende Flanke am Pin CONVST* erkannt wird, sofern gleichzeitig die Pins RD* und CS* auf Low liegen. In Abbildung 5.10 ist der Schaltplanausschnitt dieses Bauteils dargestellt.

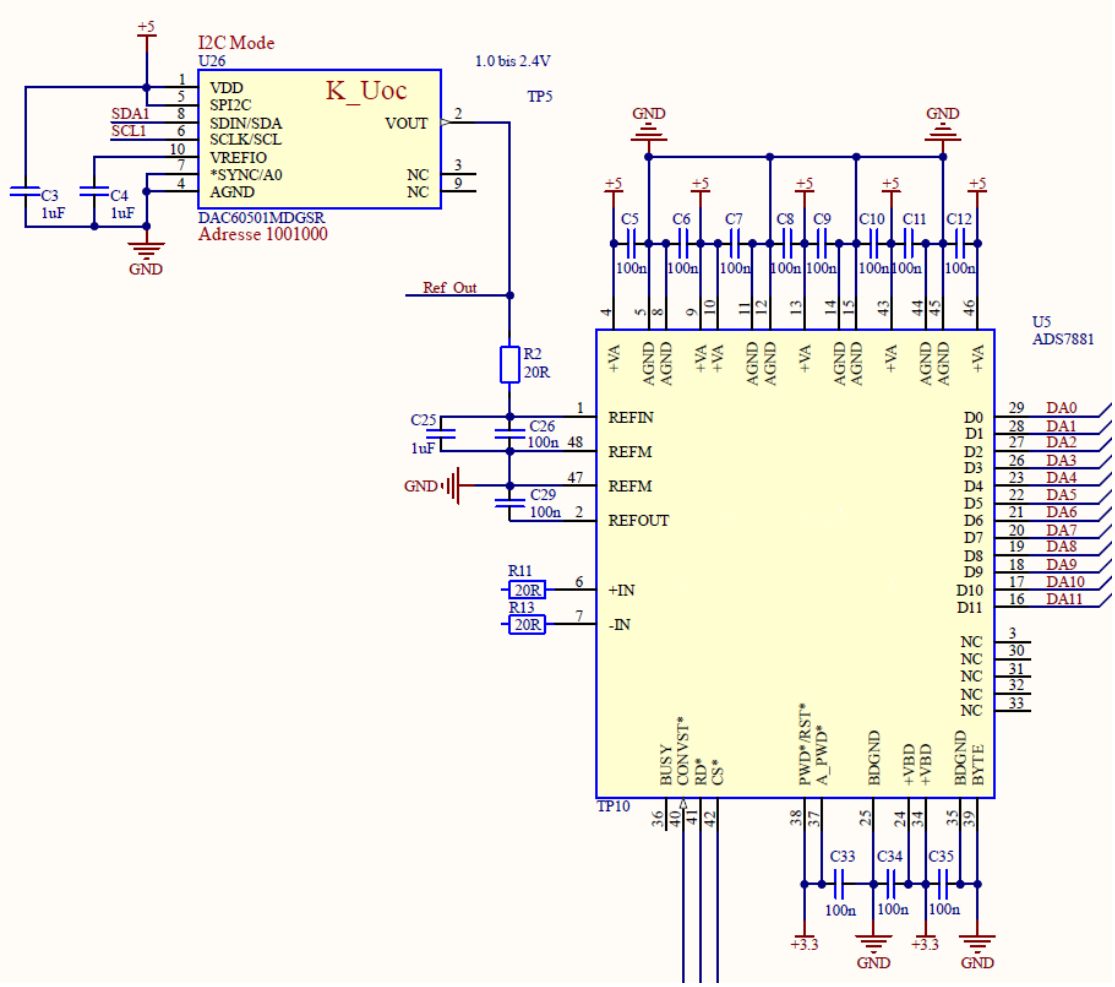


Abbildung 5.10: Schaltplanausschnitt des AD-Wandlers

5.1.7 Speicher DA Wandler

Der am Ausgang des Speichers eingesetzte DA-Wandler hat die Aufgabe, den in der Speicherzelle enthaltenen digitalen Wert in eine analoge Spannung umzuwandeln. Diese Spannung stellt den aktuellen Istwert I_{ist} für den Regelkreis der Stromquelle dar. Für diese Funktion wurde der LTC1666 [Mat9] von Analog Devices gewählt. Dieser DA-Wandler besitzt eine Auflösung von 12 Bit und unterstützt Wandlungsraten von bis zu 50 MSamples/s. Die Steuerung erfolgt über dasselbe Taktsignal wie beim AD-Wandler am Speichereingang, jedoch mit invertierter Polarität. Auch dieser DA-Wandler wird durch eine positive Flanke aktiviert.

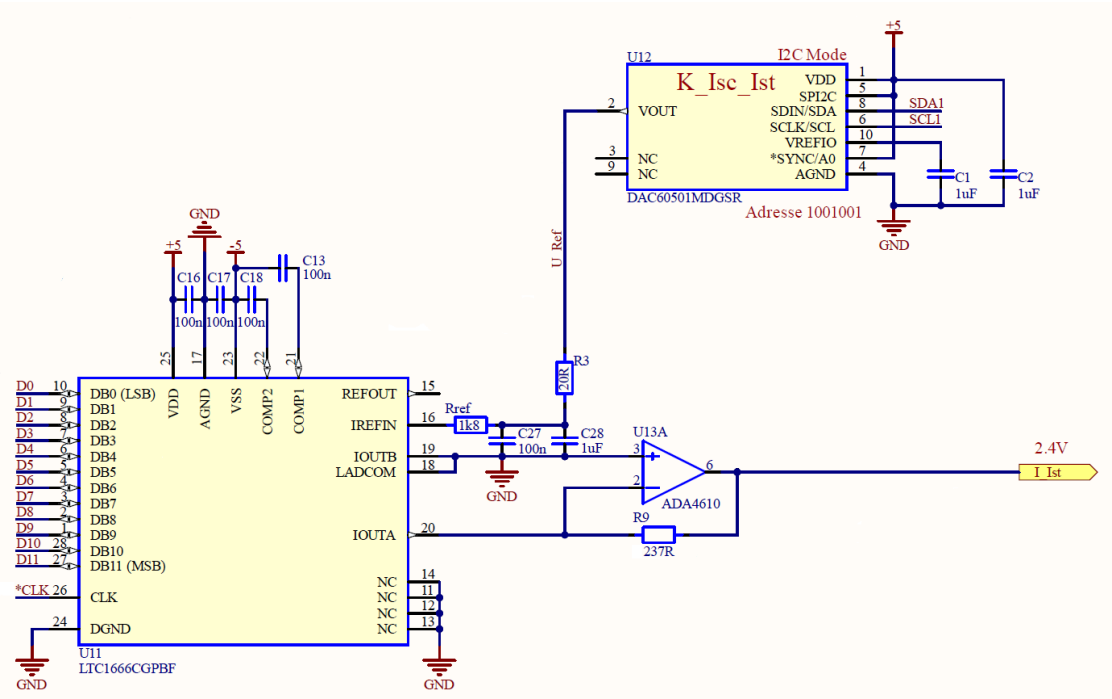


Abbildung 5.11: Schaltplanausschnitt des DA-Wandlers

Eine Besonderheit des LTC1666 ist, dass er nicht mit Spannungen, sondern mit Strömen arbeitet. Der Referenzstrom wird über den Pin “IREFIN” zugeführt. Dieser Strom entsteht durch eine vom Mikrocontroller gesteuerte Referenzspannung U_{Ref}, die über zwei in Serie geschaltete Widerstände fliesst. Die Schaltung ist in Abbildung 5.11 dargestellt. Der resultierende Strom wird nach Formel 5.2 berechnet.

$$IREFIN = \frac{U_{Ref}}{R_{Ref} + R_3} = \frac{U_{Ref}}{1.8\ k\Omega + 20\ \Omega} \approx 5.5 \cdot 10^{-4} \cdot U_{Ref} \tag{5.2}$$

Dieser Referenzstrom wird intern im Baustein mit einem festen Faktor von 8 multipliziert, wodurch sich der maximale Ausgangsstrom $IOUTA_{max}$ ergibt. Mit einer gewählten Referenzspannung von 2.4 V ergibt sich nach Formel 5.2 und folgender Berechnung:

$$IOUTA_{max} = IREFIN \cdot 8 = \frac{2.4\ V}{1.8\ k\Omega + 20\ \Omega} \cdot 8 \approx 10.549\ mA \tag{5.3}$$

Um diesen Strom in die benötigte Spannung von 2.5 V umzuwandeln, wird am Ausgang des Wandlers ein Widerstand R_9 eingesetzt. Dieser wird so dimensioniert, dass bei $IOUTA_{max}$ von ca. 10.549 mA genau 2.5 V entstehen. Die resultierende Spannung wird auf der Stromquellen-Platine (Kapitel 5.2) nochmals um den Faktor 4 verstärkt , um die erforderlichen 10 V zu erreichen, die wiederum einer Ausgangsstromstärke von 20 A entsprechen. Das Verhältnis bleibt dabei konstant bei 2 A/V. Der benötigte Widerstandswert ergibt sich aus der Formel 5.4:

$$U_{max} = 2.5\ V = IOUTA_{max} \cdot R_9 \quad \Rightarrow \quad R_9 = \frac{2.5\ V}{10.549\ mA} \approx 236.98\ \Omega \tag{5.4}$$

5.1.8 I_{sc} U_{oc} und U_{ref} DA Wandler

Wie bereits in den Kapiteln 5.1.6 und 5.1.7 erwähnt, werden für die variablen Referenzwerte zusätzliche DA-Wandler eingesetzt.

Zum Einsatz kommen drei identische DA-Wandler vom Typ DAC60501 [Mat10] der Firma Texas Instruments, die über eine I2C-Schnittstelle vom Mikrocontroller angesteuert werden.

Im aktuellen System liefern zwei dieser DACs die Referenzwerte U_{ref} und I_{sc}, welche derzeit stets identisch sind. Eine Unterscheidung zwischen den beiden wäre nur erforderlich, wenn ein Spannungsabfall von U_{oc} bei abnehmender Einstrahlung berücksichtigt werden soll (Siehe Kapitel 3.2). Diese Funktion ist momentan noch nicht implementiert.

Die drei eingesetzten DACs werden im Folgenden einzeln analysiert und beschrieben.

U_{oc} DA Wandler

Der DA-Wandler für U_{oc} muss in der Lage sein, einen Spannungsbereich von 0 V bis 60 V zu definieren, hat jedoch selbst nur 2.5 V zur Verfügung. Er wird daher als Referenzspannung für den AD-Wandler am Eingang des Speichers verwendet (siehe Kapitel 5.1.6), wobei die zu messende Spannung zuvor bereits auf einen passenden Bereich reduziert wurde.

Dieser DA-Wandler verfügt über keine Verstärkungsstufe in Serie. Der Ausgangswert wird zusätzlich in einem Spannungslimiter nach der Stufenwahl verwendet.

Beim Schreiben eines neuen Werts in diesen DA-Wandler wird gleichzeitig der entsprechende Wert für die Stufenwahl berechnet (siehe Kapitel 5.1.5) und sofort gesetzt.

Ein Auszug aus dem Hauptschaltplan mit dem DA-Wandler und den angeschlossenen Komponenten ist in Abbildung 5.12 dargestellt.

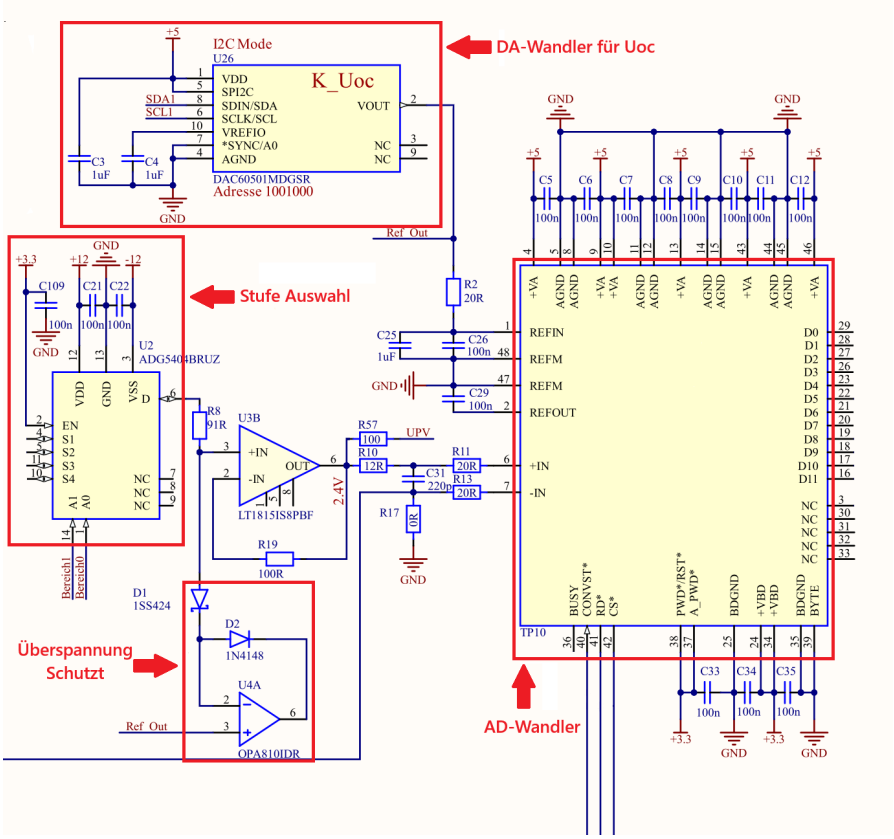


Abbildung 5.12: Schaltplanausschnitt des U_{oc} DA-Wandlers und angeschlossenen Komponenten

I_{sc} DA Wandler

Für den DA-Wandler für I_{sc} ist die Umsetzung deutlich einfacher. Wie bereits bei anderen DA-Wandlern wurde auch hier entschieden, 2.4 V als maximalen Ausgangswert zu verwenden. Allerdings besteht, wie auch bei I_{ist}, ein Verhältnis von 2 A/V zwischen Strom und Spannung. Um also einen gewünschten Maximalwert von 20 A zu erreichen, muss eine Ausgangsspannung von 10 V bereitgestellt werden.

Zu diesem Zweck wurde, wie in Abbildung 5.13 dargestellt, ein Operationsverstärker in invertierender Verstärkerschaltung eingebaut.

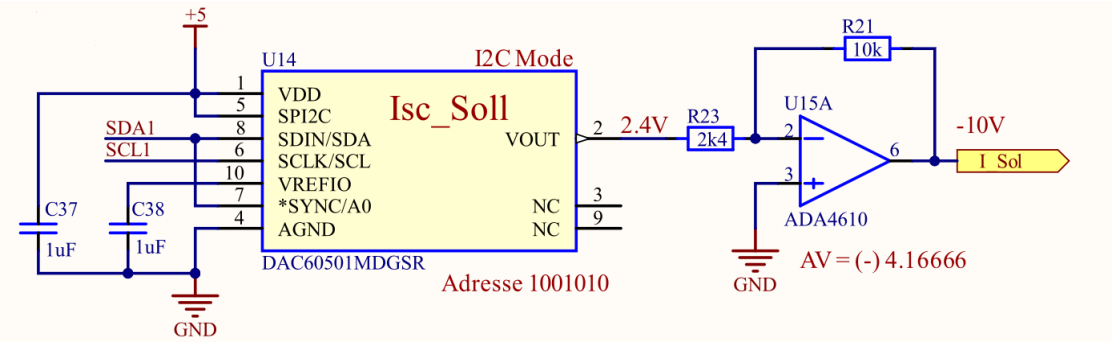


Abbildung 5.13: DA-Wandler für I_{sc} mit nachgeschaltetem Verstärker

Die Verstärkung dieser Schaltung lässt sich mithilfe der Formel 5.5 berechnen:

$$A_v = \frac{R_{21}}{R_{23}} = \frac{10\,\text{k}\Omega}{2.4\,\text{k}\Omega} = 4.1\overline{6} \tag{5.5}$$

Mit diesem Verstärkungsfaktor ergibt sich bei einer Eingangsspannung von 2.4 V eine Ausgangsspannung von 10 V, was genau dem gewünschten Stromwert von 20 A entspricht. Der Zusammenhang zwischen dem Ausgangsstrom I_{sc} und der DAC-Ausgangsspannung U_{DAC} ergibt sich aus Formel 5.6:

$$I_{sc} = U_{out} \cdot 2\,\frac{A}{V} = U_{DAC} \cdot A_v \cdot 2\,\frac{A}{V} = 2.4\ V \cdot 4.1\overline{6} \cdot 2\,\frac{A}{V} = 20\ A \tag{5.6}$$

U_{ref} DA Wandler

Wie bereits zuvor erwähnt, dient dieser DAC dazu, mögliche Schwankungen der Eintrahlungsstärke zu kompensieren, da diese neben einer Änderung von I_{sc} auch eine leichte Variation von U_{oc} zur Folge haben können (siehe Kapitel 3.2).

Wie in Kapitel 5.1.7 beschrieben und in Abbildung 5.14 dargestellt, liefert dieser DAC ausserdem den Referenzstrom für den DAC am Ausgang des Speichers.

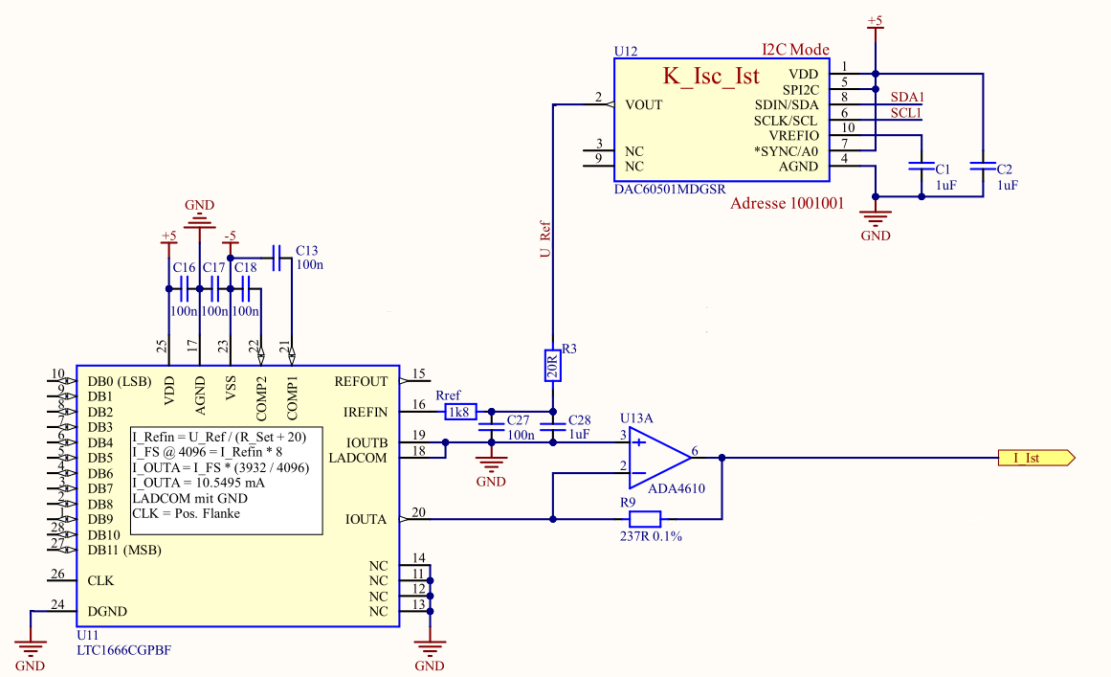


Abbildung 5.14: DA-Wandler für U_{ref} und angeschlossene Speicher DA-Wandler

In der aktuellen Firmware-Version des PVMS ist diese Funktionalität jedoch noch nicht implementiert. Der Wert dieses DACs ist daher stets identisch mit demjenigen des DACs für I_{sc}. Auf diese Weise bleibt U_{oc} unabhängig von der aktuellen Eintrahlungsstärke konstant.

5.1.9 Speicher

Das System verwendet einen parallelen NOR-Flash-Speicher des Typs S29GL01GS von Infineon [Lit27]. Dieser Speicher verfügt über eine Kapazität von 1 Gbit und wurde sowohl aufgrund seiner technischen Eigenschaften, die für dieses Projekt als geeignet erachtet wurden, als auch aus logistischen Gründen ausgewählt: Er war bereits im zuvor vom PV-Labor entwickelten Simulator im Einsatz, und es waren noch einige Exemplare vor Ort verfügbar.

Verschaltung

Während des Schreibvorgangs muss der Speicher über den Flexible Memory Controller (FMC) [Lit9] angesteuert werden. Im Normalbetrieb hingegen wird die Leseadresse teilweise von einem ADC und teilweise von weiteren Pins des STM32 bereitgestellt. Dies erfordert zwei unterschiedliche Adressbusse, die jedoch nicht gleichzeitig aktiv sein dürfen. Aus diesem Grund wurden, wie in Abbildung 5.15 dargestellt, Busswitches in die Leitungen für Adressen und Daten integriert.

Der vom ADC [Mat8] bereitgestellte Adressteil wäre potenziell instabil, wenn der ADC während des Schreibvorgangs weiterhin aktiv wäre. Aus diesem Grund wird der ADC beim Schreiben des Speichers deaktiviert.

Die Busswitches in den Datenleitungen sind ebenfalls von Bedeutung: Ein möglicher Einfluss des STM32 auf den Datenbus, etwa wenn Pins ungewollt aktiviert sind, war nicht genau bekannt. Zudem sollte vermieden werden, dass interne Pull-Up- oder Pull-Down-Widerstände des STM32 den Pegel der Datenleitungen beeinflussen, da diese direkt zwischen Speicher und DAC [Mat9] geschaltet sind. Die Busswitches werden im folgenden Kapitel noch genauer beschrieben.

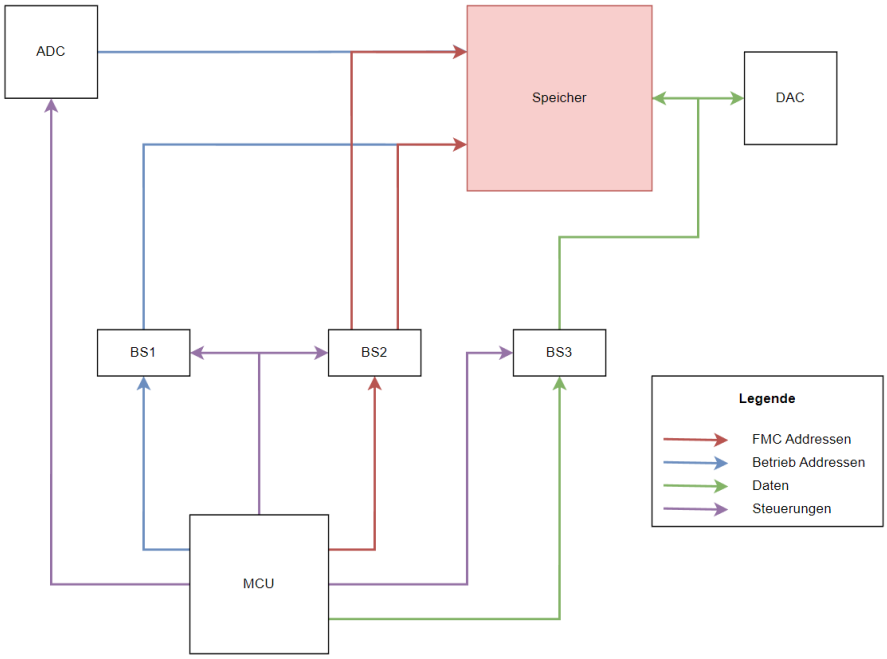


Abbildung 5.15: Verschaltung der Speicheransteuerung über Busswitches

5.1.10 Busswitches

Die bereits im Kapitel 5.1.9 erwähnten Bus-Switches dienen dazu, bestimmte Busleitungen gezielt ein- oder auszuschalten. Ist ein Bus deaktiviert, besteht keine elektrische Verbindung zwischen dem Ein- und dem Ausgangspin, das Signal wird vollständig isoliert.

Im vorliegenden System sind bidirektionale Bus-Switches im Einsatz, die eine Signalübertragung in beide Richtungen ermöglichen. Dies ist entscheidend, nicht nur um Daten in den Speicher zu schreiben, sondern auch, um sie anschliessend auszulesen und die Korrektheit der gespeicherten Informationen zu überprüfen.

Der Einsatz der Bus-Switches ist in diesem Zusammenhang aus mehreren Gründen notwendig:

- Sie ermöglichen die dynamische Umschaltung der Adressleitungen zwischen FMC und ADC,
- Sie stellen sicher, dass die Daten korrekt vom DAC gelesen werden können,
- Sie verhindern elektrische Konflikte zwischen den angeschlossenen Komponenten auf denselben Leitungen.

Ein besonders kritischer Punkt betrifft das Zusammenspiel zwischen dem FMC (einem internen Modul des STM32) und dem ADC. Während des normalen Betriebs wird ein Teil der Speicheradresse vom ADC bereitgestellt. Sollte der ADC jedoch während eines Schreibvorgangs aktiv bleiben, bestünde die Gefahr, dass sowohl FMC als auch ADC gleichzeitig versuchen, dieselben Pins anzusteuern. Dies könnte zu Spannungskonflikten führen, mit unvorhersehbaren Effekten bis hin zur Beschädigung des STM32.

Um solche Risiken auszuschliessen, wird der ADC während der Speicherbeschriftung deaktiviert. Während das Betriebs trennen die Bus-Switches physikalisch die betroffenen Leitungen, um unerwünschte Beeinflussungen zuverlässig zu vermeiden.

5.2 Stromquelle

Die Stromquelle bildet die zentrale Einheit zur Erzeugung eines Ausgangsverhaltens, das den Kenngrössen der eingestellten Kennlinie entspricht. In der unteren Abbildung 5.16 ist die bestückte Leiterplatte zusehen.

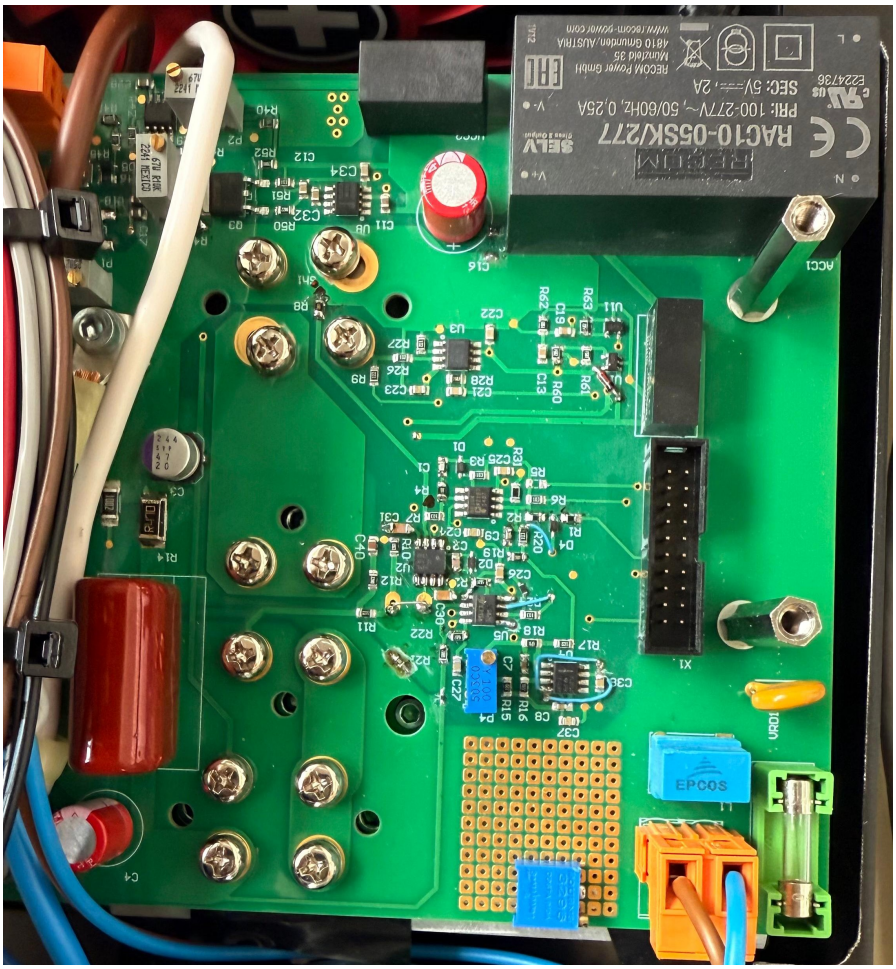


Abbildung 5.16: Stromquelle – Aufbau der Hauptplatine

In diesem Kapitel wird die Funktionsweise dieser Stromquelle beschrieben. Anhand des vorliegenden Schaltplans (siehe Angang 2) wird schrittweise auf die verschiedenen Einheiten und Funktionsblöcke eingegangen, beginnend mit der Spannungsversorgung.

5.2.1 Eingangsspannung

Die Stromquelle wird über einen Netzanschluss (230 V / 50 Hz) mit Energie versorgt. Diese Netzspannung dient ausschliesslich der Versorgung der Steuerelektronik. Die eigentliche Leistungsversorgung des Systems erfolgt über ein im PVMS integriertes Netzgerät [Mat4].

Wie bereits im Kapitel 3.4 erläutert, ist die Versorgung galvanisch getrennt. Diese Trennung ermöglicht die Erzeugung eines virtuellen Massepotentials (*Virtual Ground*), das unabhängig von der eigentlichen Netzversorgung verwendet werden kann. Zur Umsetzung dieser Trennung wird ein isolierter DC/DC-Wandler eingesetzt.

Zunächst kommt die Eingangsspannung von 230V auf ein Trentrafo. Dieser leitet die Versorgung weiter an die Stromquelle. Auf dieser Platine ist ein Spannungwandler der RAC10-K_277 [Mat5] dieser wandelt die Spannung in eine 5V Speisung um. Diese 5V wird anschliessend für die Versorgung der Stromquellenplatine wie auch für die Steuerkarte verwendet. Um jedoch eine Groundleiterschleife zu bilden, wurden auf der Stromquelle weiter galvanisch getrennten Regler eingesetzt.

Der 3S7BE_0515D3U [Mat11] ist ein galvanisch isolierter DC/DC-Wandler, der am Ausgang symmetrische Spannungen von +15 V und −15 V bereitstellt. Diese Spannungen werden mithilfe eines TLP810ADJ-5TR [Mat12] (für +13 V) und eines MIC5270YM5-TR [Mat13] (für −13 V) stabilisiert. Die resultierenden +13 V und −13 V versorgen die empfindlicheren Operationsverstärker der Stromquelle. Die +-15V werden auch für Operationsverstärker verwendet, jedoch sind diese weniger störungsanfällig. Die zugehörige Schaltung ist in Abbildung 5.17 dargestellt.

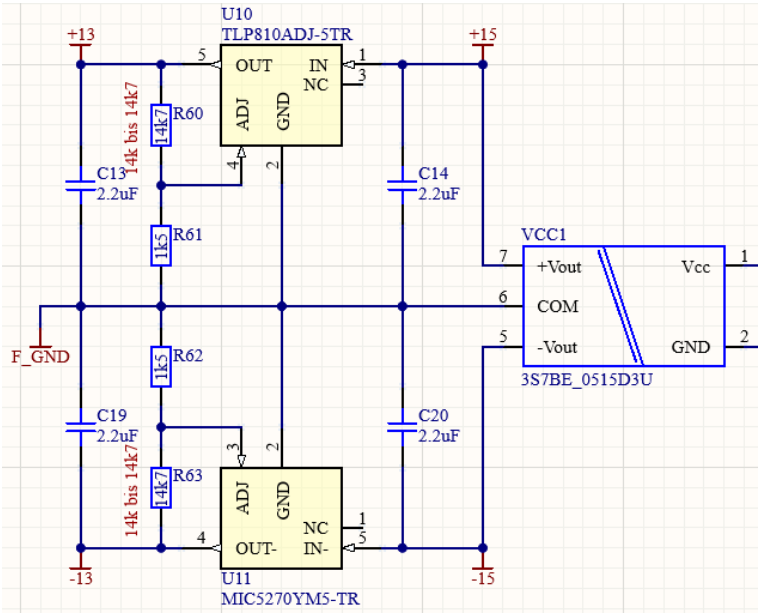


Abbildung 5.17: Erzeugung und Regelung von ±15 V auf ±13 V

Zusätzlich wird eine weitere isolierte Spannung von 9 V benötigt. Diese dient der Kalibrierung der Stromquelle. Sie wird durch den CRV1D0512SC [Mat14] erzeugt. Die zugehörige Schaltung ist in Abbildung 5.18 dargestellt.

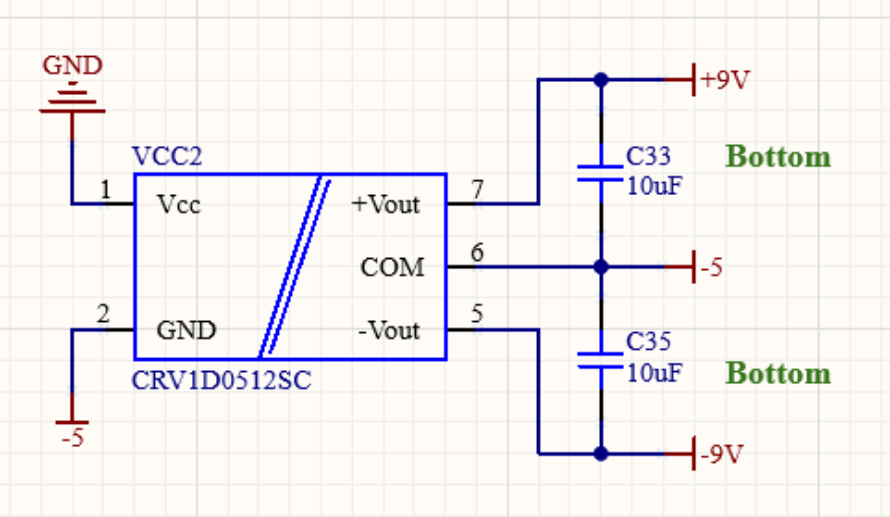


Abbildung 5.18: Erzeugung der isolierten 9 V-Versorgung

5.2.2 Steuerung der Versorgungseinheit

Die Stromquelle wird von einem leistungsstarken Netzgerät mit einer variablen Ausgangsspannung von bis zu $60\,\mathrm{V}$ gespeist. Da unterschiedliche Kennlinien mit verschiedenen Leerlaufspannungen (U_{oc}) simuliert werden müssen, ist eine präzise Regelung der Versorgungsspannung erforderlich.

Die Ansteuerung des Netzgeräts erfolgt über eine Steuerschaltung, die in Abbildung 5.19 dargestellt ist. Diese Schaltung ermöglicht eine Feineinstellung der Ausgangsspannung durch unterschiedliche Massebezugspunkte.

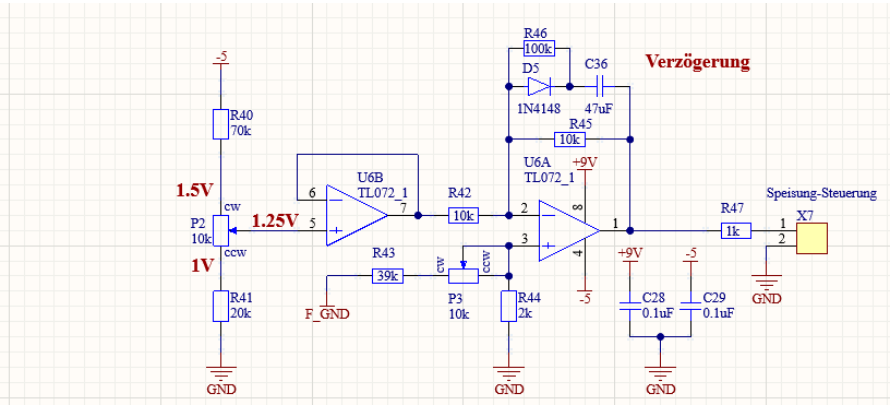


Abbildung 5.19: Schaltung zur Feinanpassung der Versorgungsspannung

Die Schaltung gewährleistet, dass bei einem Spannungsabfall am Ausgang auf $0\,\mathrm{V}$ die Versorgungsspannung automatisch auf ein Minimum reduziert wird. Dieser Schutzmechanismus verhindert, dass die gesamte Leistung über die MOSFETs abfällt, da diese eine derart hohe Belastung nicht dauerhaft tolerieren können (siehe Abschnitt 5.2.4).

Zusätzlich ist die Schaltung so ausgelegt, dass die Spannung *schnell ansteigt*, aber *langsam abfällt*. Dieses dynamische Verhalten ist entscheidend für die korrekte Darstellung der Kennlinien:

- Ein **schneller Spannungsanstieg** stellt sicher, dass das Netzgerät die benötigte Spannung rechtzeitig bereitstellt.
- Ein **kontrolliertes Absenken** vermeidet eine abrupte Überlastung der MOSFETs, die kurzzeitig höhere Leistungen verkraften können.

Durch diese Charakteristik wird eine verzerrungsfreie Abbildung der Strom-Spannungs-Kennlinie gewährleistet, während gleichzeitig die Bauteile vor Schäden geschützt werden.

5.2.3 Stromsenke

Um einen stabilen Betrieb der Stromquelle zu gewährleisten, wird eine Stromsenke implementiert. Diese sorgt dafür, dass die eingesetzten MOSFETs stets im definierten Arbeitspunkt betrieben werden, selbst wenn momentan keine externe Last angeschlossen ist.

Die Stromsenke entzieht der Stromquelle kontinuierlich einen Strom von 20 mA. Dadurch wird sichergestellt, dass die Ausgangs-MOSFETs dauerhaft leitend bleiben und somit im vorgesehenen Arbeitspunkt arbeiten. Dies erhöht die Stabilität und Reproduzierbarkeit des Systems, insbesondere bei der Abbildung dynamischer Kennlinienverläufe.

In Abbildung 5.20 ist die Schaltung der Stromsenke dargestellt.

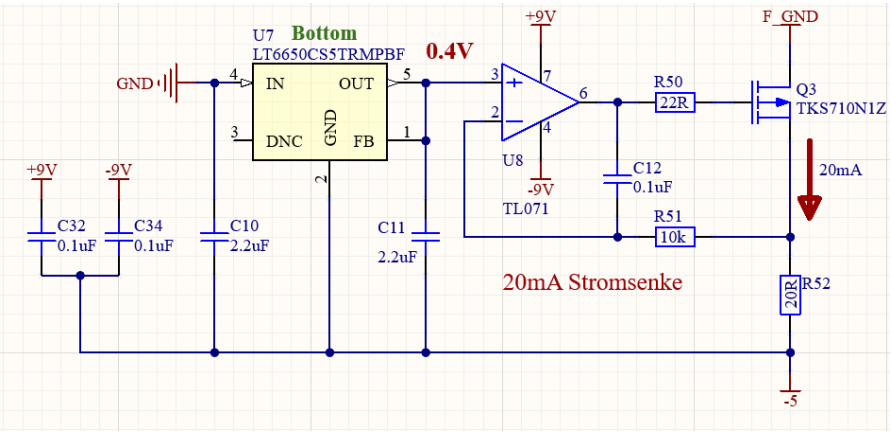


Abbildung 5.20: Stromsenkeschaltung mit konstantem Entnahmestrom von 20 mA

Die Referenzspannung von 0.4 V wird dabei durch einen hochpräzisen Spannungsreferenzbaustein (LT6650CS5TRMPBF)[Mat15] bereitgestellt. Diese Spannung wird einem nichtinvertierenden Verstärker zugeführt, der einen nachgeschalteten MOSFET so ansteuert, dass unabhängig von der Versorgungsspannung ein konstanter Stromfluss von 20 mA gewährleistet ist.

Die Schaltung wirkt wie eine geregelte Stromquelle in umgekehrter Richtung, also als Stromsenke, und stabilisiert das Betriebsverhalten der Ausgangsstufe massgeblich.

5.2.4 Leistungssteuerung

Die Leistungssteuerung bildet das Herzstück der Stromquelle. In diesem Kapitel wird detailliert beschrieben, wie die Leistungselektronik aufgebaut ist und wie sie arbeitet. Ziel ist es, die einzelnen Funktionsblöcke verständlich darzustellen und deren Zusammenspiel im Gesamtsystem zu erläutern.

Da die Leistungsregelung ein komplexes Zusammenspiel mehrerer Komponenten erfordert, wird die Schaltung in sinnvolle Einheiten unterteilt. Jede Einheit übernimmt dabei eine spezifische Aufgabe im Regelprozess der Stromversorgung, angefangen bei der Strommessung über die Gate-Ansteuerung der Leistungstransistoren bis hin zur Rückkopplung und Regelung.

Das vollständige Schaltbild der Leistungselektronik ist in Abbildung 5.21 dargestellt.

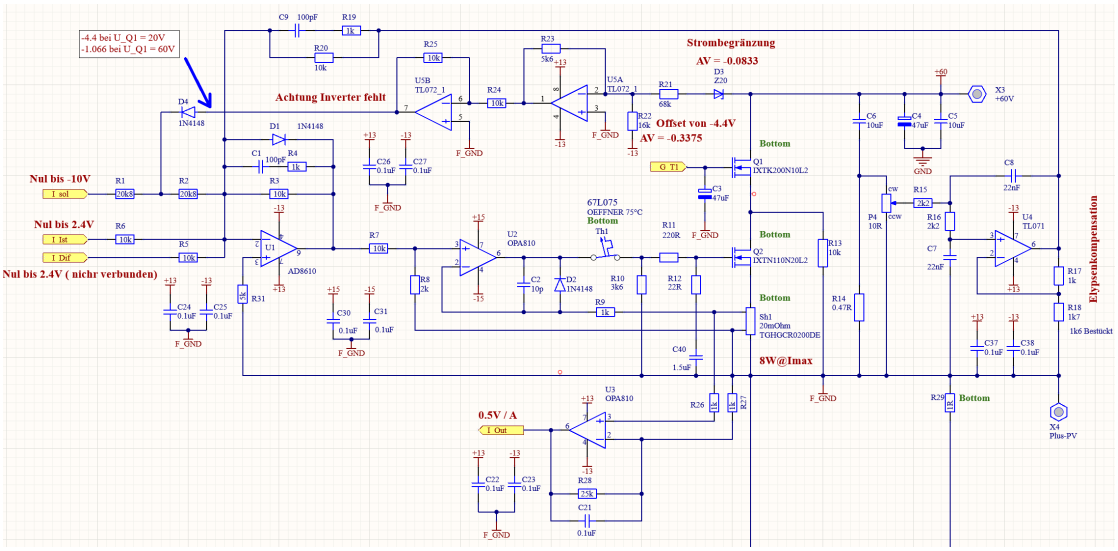


Abbildung 5.21: Schema der Leistungselektronik

Im weiteren Verlauf dieses Kapitels werden die einzelnen Funktionsblöcke der Schaltung näher beschrieben und deren Funktion im Gesamtkontext erläutert.

Strompfad

Abbildung 5.22 zeigt den relevanten Schaltungsausschnitt des Strompfads.

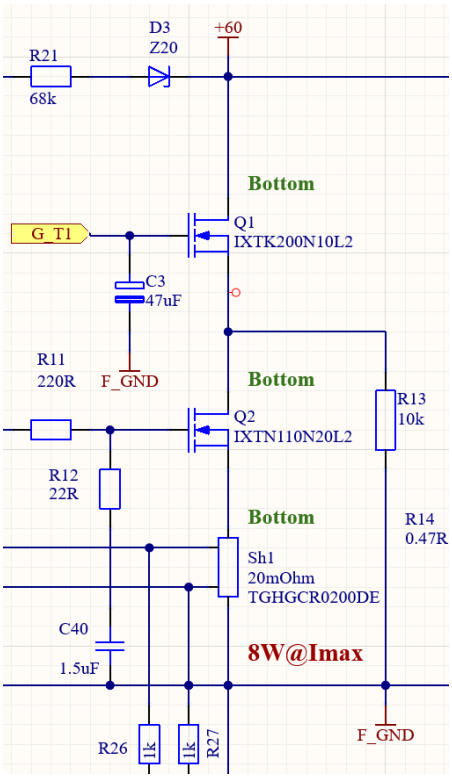


Abbildung 5.22: Strompfad mit MOSFET-Ausgangsstufe

Der Strompfad besteht aus zwei parallel geschalteten MOSFETs (Typ: IXTK200N10L2 [Mat19]), die folgende Eigenschaften aufweisen:

- Maximale Drain-Source-Spannung: $100\,\mathrm{V}$
- Dauerstrombelastbarkeit: $178\,\mathrm{A}$
- Maximale Verlustleistung: $830\,\mathrm{W}$ (unter optimalen Kühlbedingungen)

Bei der maximalen Systemleistung von $1200\,\mathrm{W}$ ($60\,\mathrm{V} \times 20\,\mathrm{A}$) wäre ein einzelner MOSFET überlastet. Durch die Beschaltung wird die Verlustleistung auf beide Transistoren verteilt, wobei jeder etwa die Hälfte der Leistung aufnimmt.

Selbst bei dieser Aufteilung würde die Verlustleistung von ca. $600\,\mathrm{W}$ pro MOSFET die zulässigen Grenzwerte überschreiten. Da keine praktikablen Kühlkörper für diese Leistung verfügbar sind, wurde das in Abschnitt 5.2.2 beschriebene Konzept implementiert: Bei

Überlast reduziert das System die Versorgungsspannung von 60 V auf 12 V, wodurch sich die maximale Verlustleistung auf 240 W für beide MOSFET verringert. Jedoch gilt das nur wenn ein Kurzschluss am Ausgang vorliegt. Das System muss auch bei einer tieferen Versorgungsspannung funktionieren, somit kann es vorkommen, dass ein Spannungsabfall an den beiden MOSFETs von 20 V auftritt. Dies resultiert in einer Verlustleistung von 400 W für beide MOSFET, diese Wärme muss abgeführt werden.

Zur Wärmeabfuhr kommt eine kombinierte Kühlung zum Einsatz:

- Passiver Kühlkörper
- Aktive Wasserkühlung (XILENCE LQ120 [Mat7])

Nun muss die Beschaltung der beiden MOSFETs definiert werden. Diese sind, wie in Abbildung 5.22 dargestellt, nicht parallel geschaltet, sondern in Serie. Folglich muss die Spannung zwischen den beiden MOSFETs aufgeteilt werden, und nicht der Strom wie bei einer Parallelschaltung der MOSFETs. Dies wird in folgenden Kapiteln erläutert.

5.2.5 Feinjustierung des MOSFET-Offsets

Wie im vorherigen Abschnitt beschrieben, wird die Leistung der beiden MOSFETs aufgeteilt. Um dies umzusetzten wurde einer der beiden MOSFETs als Offset-MOSFET definiert. Bedeutet dieser dient nur für die entlastung des 2. MOSFETs. Jedoch muss dieser Offset genau eingestellt werden. In diesem Kapitel wird dies Feinjustierung bedschrieben.

In Abbildung 5.23 ist die Schaltung zur Feinjustierung dargestellt.

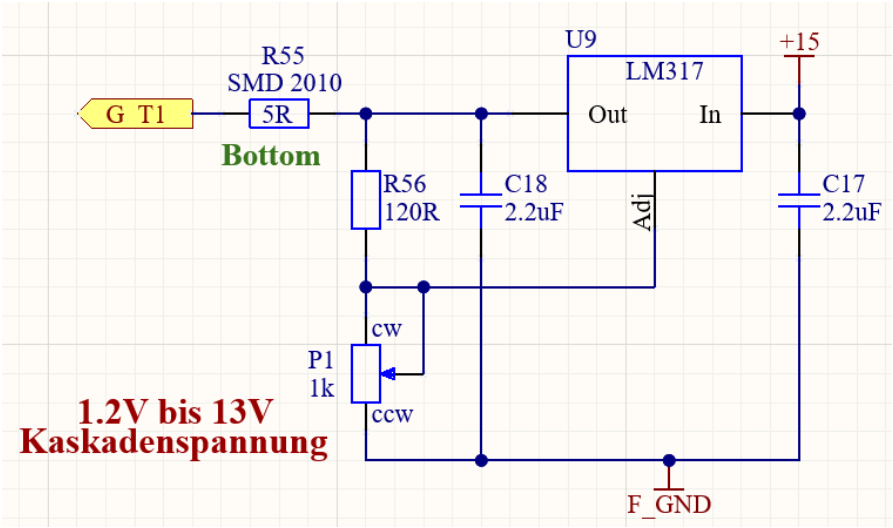


Abbildung 5.23: Schaltung zur Feinjustierung des Offset-MOSFET

Ein Spannungsregler vom Typ LM317 [Mat16] erzeugt eine einstellbare, konstante Ausgangsspannung. Mithilfe des Potentiometers P1 kann diese Spannung im Bereich von etwa 1.2 V bis 13 V justiert werden. Diese Spannung wird zur Ansteuerung des Gates des Offset-MOSFETs Q1 verwendet und bestimmt somit dessen Stromdurchfluss im Ruhezustand.

Durch die gezielte Vorsteuerung des Offset-MOSFETs wird sichergestellt, dass dieser stets in einem leitenden Zustand ist, wodurch eine stabile Ausgangskennlinie auch bei geringen Stromanforderungen gewährleistet werden kann.

Nun ist der erste MOSFET eingestellt, es gilt nun die Funktionsweise des zweiten MOSFETs zu analysieren.

5.2.6 Stromregelung

Um die Stromregelung des zweiten MOSFETs zu realisieren, wird in diesem Abschnitt die entsprechende Schaltung beschrieben. Die unten dargestellte Schaltung zeigt die Regelungseinheit, deren Funktionsweise im Folgenden erläutert wird.

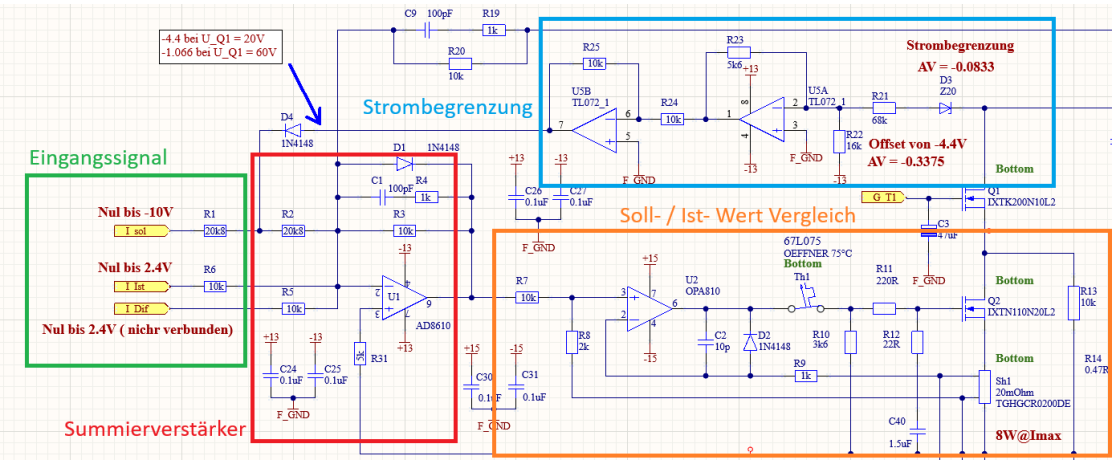


Abbildung 5.24: Schaltung zur Feinjustierung des Offset-MOSFET

Wie in Abbildung 5.24 ersichtlich ist, gliedert sich die Stromsteuerung in vier Teilbereiche. Diese werden im Folgenden im Detail erläutert.

Eingangssignal

Der grün umrahmte Block in Abbildung 5.24 zeigt das Eingangssignal der Regelung. Die Digitalplatine liefert hier eine Spannung im Bereich von $0\,\mathrm{V}$ bis $-10\,\mathrm{V}$ an den isolierten Eingang. Dabei entspricht:

$-10\,\mathrm{V} \rightarrow 20\,\mathrm{A}$ Ausgangsstrom (Kurzschlussstrom I_{SC} der Kennlinie)

Besonderheiten der Schaltung

- Spannungsrückkopplung von den MOSFETs zum Eingang
- Dynamische Strombegrenzung bei plötzlichen Laständerungen

Sicherheitsmechanismen: wie in Abschnitt 5.2.4 beschrieben, erfolgt eine Aufteilung der maximalen Leistung des PVMS. Die in Abschnitt 5.2.2 erläuterte Spannungsreduktion des Netzgeräts bietet jedoch unzureichenden Schutz. Daher wurde ein zusätzliches Schutzsystem implementiert:

Strombegrenzung bei hoher MOSFET-Spannung:

- $U_{MOSFET} > 20\,\mathrm{V}$: Reduktion des Ausgangsstroms
- $U_{MOSFET} \geq 60\,\mathrm{V}$: Hartebegrenzung auf $6\,\mathrm{A}$

Die detaillierte Funktionsbeschreibung dieser Schutzschaltung folgt in Kapitel 5.2.6.

Der $\mathrm{I_{ist}}$-Wert dient zur Einstellung der Sollstromvorgabe. Beim Abfahren der Kennlinie entspricht dieser dem aktuell gewünschten Stromwert. Der $\mathrm{I_{Diff}}$-Eingang ist für zukünftige Anwendungen vorgesehen und ermöglicht die Einspeisung eines Differenzanteils.

Summierverstärker

Der Summierverstärker aggregiert sämtliche Eingangsströme durch mathematische Addition. In Abbildung 5.24 ist dieser in der rot markierten Box dargestellt.

Die RC-Kombination (Kondensator mit Widerstand) dient der Hochfrequenzkompensation:

- Reduzierung der Verstärkung auf $\frac{1}{10}$ bei höheren Frequenzen
- Verbesserung der Stabilität des Operationsverstärkers
- Vermeidung unerwünschter Phasendrehungen

Die antiparallel geschaltete Diode erfüllt zwei Schutzfunktionen:

1. Verhinderung negativer Ausgangsspannungen, die die PVMS-Funktionalität beeinträchtigen würden
2. Schutz der nachgeschalteten MOSFET-Stufe vor Beschädigung durch inverse Spannungen

Im Fehlerfall leitet die Diode und begrenzt damit die Ausgangsspannung des Verstärkers auf $0.7\,\mathrm{V}$ in Durchlassrichtung.

Soll-/Ist-Wert-Vergleich

Im orange umrahmten Block erfolgt der Vergleich von Soll- und Istwert. Über einen Shunt-Widerstand (SH1) mit $20\,\mathrm{m\Omega}$ wird der Stromfluss im Strompfad erfasst. Ein Strom von $1\,\mathrm{A}$ am Shunt entspricht einem Strom von $20\,\mathrm{mV}$.

Durch zusätzliche passive Bauelemente wird die Phasenlage im Operationsverstärker angepasst, um eine stabile Regelung zu gewährleisten. Die Schaltung enthält zudem einen Temperaturschutz: Bei einer Temperatur von $75°\mathrm{C}$ wird eine Sicherung ausgelöst, wodurch der MOSFET abgeschaltet wird. Dieser Teil bildet den P-Anteil im Regelkreis.

Strombegrenzung

Der blau umrahmte Block in Abbildung 5.24 führt die Versorgungsspannung in den Regelkreis zurück. Die Implementierung erfolgt durch:

- Einen Spannungsteiler mit Teilungsfaktor $0{,}082$
- Einen Offset von $4.4\,\mathrm{V}$, erzeugt durch Widerstand R22

Funktionsprinzip Die resultierende Spannung wird mit dem I_{SC}-Signal verrechnet und dient als aktive Strombegrenzung. Die Schaltung arbeitet wie folgt:

1. Bei **keiner Last** (kein Spannungsabfall über den MOSFETs):
   - Erzeugung einer Spannung von $-4.4\,\mathrm{V}$ (vor der Diode)
   - Mit Berücksichtigung der Dioden-Durchlassspannung von $0.6\,\mathrm{V}$ ergibt sich eine Schwellspannung von $-5\,\mathrm{V}$
   - Dies entspricht genau der halben maximalen Eingangsspannung des isolierten Eingangs ($-10\,\mathrm{V} \rightarrow -5\,\mathrm{V}$)

2. Bei **Last** (Spannungsabfall über den MOSFETs):
   - Die Spannung vor der Diode erhöht sich proportional zur MOSFET-Spannung
   - Bei $60\,\text{V}$ MOSFET-Spannung reduziert sich die maximal mögliche Eingangsspannung auf $-1.6\,\text{V}$
   - Dies begrenzt den Ausgangsstrom auf ca. $6\,\text{A}$

Berechnung der Strombegrenzung Der begrenzte Ausgangsstrom ergibt sich aus:

$$I_{out} = I_{max} \cdot \frac{V_{in} + V_{offset}}{V_{in,max}} \tag{5.7}$$

wobei:

- $I_{max} = 20\,\text{A}$ (Maximalstrom)
- $V_{in,max} = -10\,\text{V}$ (Maximale Eingangsspannung)
- $V_{offset} = 4.4\,\text{V}$ (Offsetspannung)

5.2.7 Ellipsenkompensation

Die Ellipsenkompensation dient dem Ausgleich der parasitären Kapazitäten im Ausgangsbereich der Schaltung. Diese umfassen:

- Die intrinsischen Kapazitäten der MOSFETs (C_{gs}, C_{gd}, C_{ds})
- Zusätzliche kapazitive Effekte zur Systemstabilisierung (Phasenrandoptimierung)

Abbildung 5.25 zeigt die parasitären Kapazitäten eines MOSFETs, die zu nichtlinearem Verhalten führen:

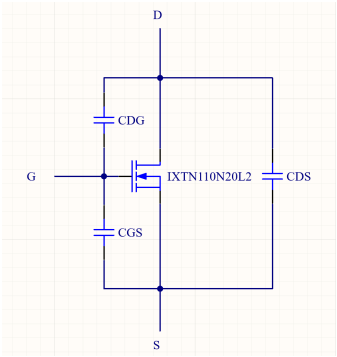


Abbildung 5.25: Parasitäre Kapazitäten eines MOSFETs mit den drei Hauptkomponenten C_{gs}, C_{gd} und C_{ds}

Problemstellung Beim Durchfahren einer Kennlinie entstehen durch die verzögerte Entladung der Kapazitäten charakteristische Abweichungen:

- Stromabweichungen beim Richtungswechsel
- Elliptische Verzerrungen in der Strom-Spannungs-Darstellung
- Abweichung vom idealen, linearen Verhalten

Kompensationsschaltung Die in Abbildung 5.26 dargestellte Schaltung kompensiert diese Effekte:

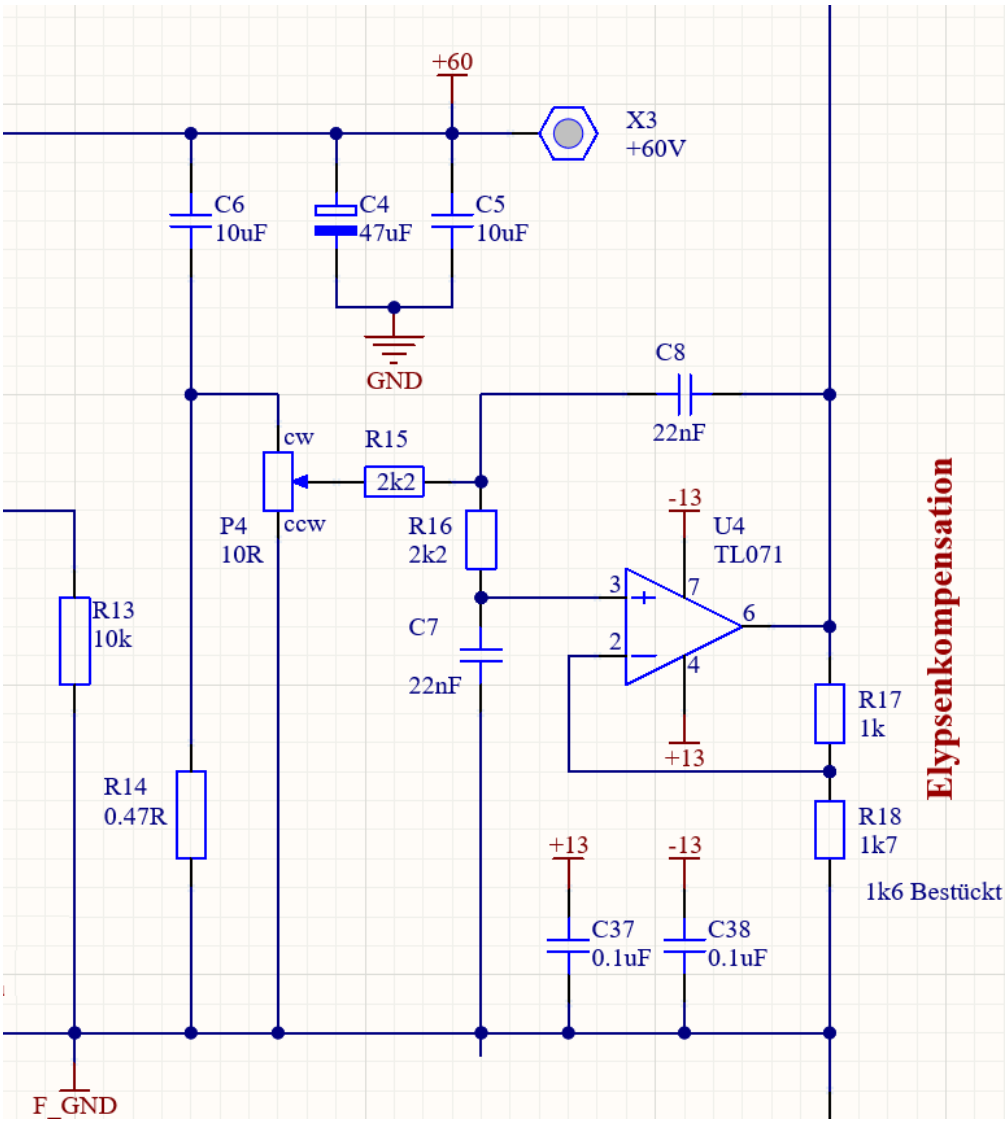


Abbildung 5.26: Aktive Kompensationsschaltung zur Eliminierung der ellipsenförmigen Verzerrungen

Funktionsprinzip

1. Der Kondensator C6 misst den Strom vom virtuellen Masspunkt
2. Bei Lastwechseln (Belastung/Entlastung) regelt die Schaltung:
   - Momentane Erhöhung/Reduktion des Stroms
   - Dynamische Anpassung der MOSFET-Leitfähigkeit
3. Der Widerstand R14 wandelt den Kompensationsstrom in eine Steuerspannung um
4. Diese Spannung wird dem Summierverstärker zugeführt und regelt den Ausgangsstrom

Einstellmöglichkeiten Das Potentiometer *P4* ermöglicht die Feinabstimmung:

- Optimale Anpassung an die Integrierschaltung
- Kompensation der MOSFET-spezifischen Charakteristika
- Einstellung des Kompensationsfaktors

Die implementierte Kompensation erreicht eine Anpassungszeit von weniger als $100\,\mu s$ bei Lastwechseln.

5.3 Control-Board

Das sogenannte *Control-Board* stellt die Human-Machine-Interface für die lokale Bedienung des Geräts dar. Wie im Anhang 3 ersichtlich, ist es über einen 20-poligen Flachbandkabelstecker mit der Steuerkarte verbunden.

Das Board verfügt über folgende Bedienelemente und Anzeigeeinheiten:

- **5 Taster:** Drei der fünf Tasten werden im Polling-Verfahren ausgelesen, während die verbleibenden zwei über Interrupts verarbeitet werden.
- **2 Inkrementalgeber mit integriertem Drucktaster:** Auch die beiden Encoder werden interruptgesteuert ausgelesen [Mat17].
- **2 LEDs:** Diese LED dinen zur Anzeige von Systemzuständen und sind jeweils mit einer der beiden Interrupt-Tasten gekoppelt.
- **1 OLED-Display:** 2.42 Zoll, 128x64 Pixel, SPI-Schnittstelle. Dieses Display dient zur Anzeige von Systeminformationen und Statusmeldungen [Mat18].

Abbildung 5.27 zeigt die physische Ausführung des Control-Boards.

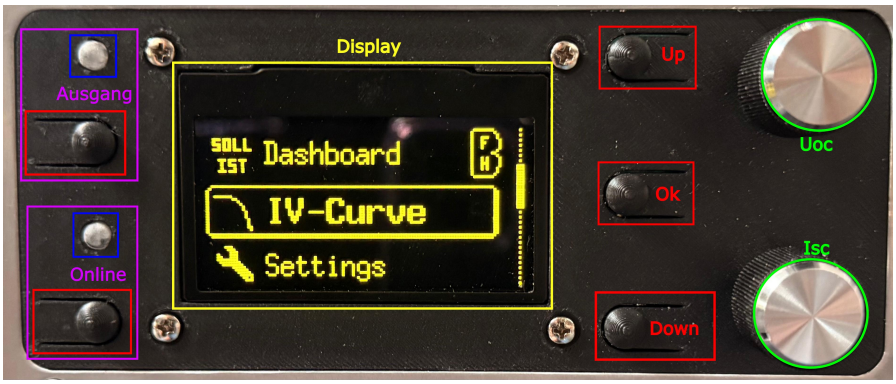


Abbildung 5.27: Control-Board Vorderansicht

Weitere Informationen zu den jeweiligen Funktionen und zur Auslesemethodik sind im Kapitel 7.1.1, 7.1.2 zu finden.

6 Software

In diesem Kapitel wird die Architektur der entwickelten Softwarelösung für das Photovoltaik-Modul-Simulationssystem (PVMS) beschrieben. Ziel ist es, ein System bereitzustellen, das dem Benutzer eine komfortable, intuitive und ortsunabhängige Bedienung über eine grafische Benutzeroberfläche (GUI) ermöglicht. Diese GUI bildet die zentrale Schnittstelle zwischen Mensch und Maschine und dient zur Visualisierung, Steuerung und Kommunikation mit der entwickelten Hardware.

Die Umsetzung erfolgt auf einem Raspberry Pi Zero 2W, der sowohl die Benutzerinteraktion verarbeitet als auch die Kommunikation mit einem angeschlossenen Mikrocontroller übernimmt. Das Gesamtsystem basiert auf einem modularen Client-Server-Prinzip, das sowohl als klassische Desktop-Applikation als auch als WebAssembly-Anwendung genutzt werden kann. Die zentrale Idee: Benutzereingaben werden systematisch über eine Softwarepipeline an die Steuerhardware (STM32) übergeben und ermöglichen so die Echtzeitsimulation verschiedener PV-Kennlinien.

6.1 Übersicht der Architektur

Die Abbildung 6.1 veranschaulicht das übergeordnete Kommunikationskonzept:

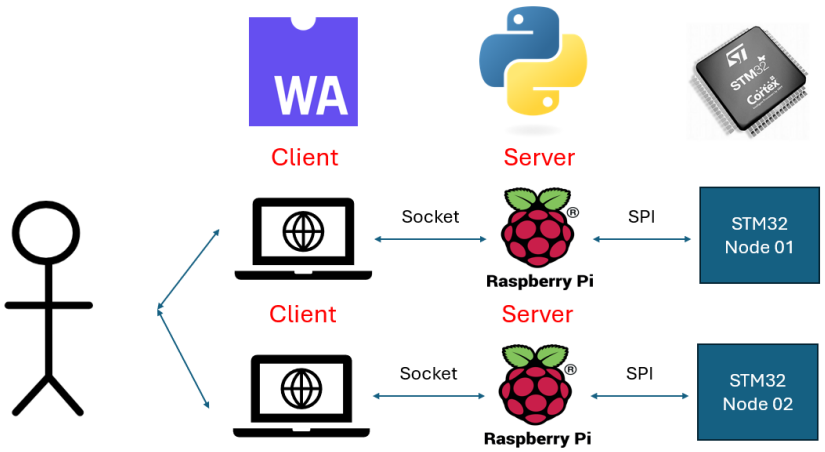


Abbildung 6.1: Konzept der Software- und Kommunikationsarchitektur

Das System ist so aufgebaut, dass ein Benutzer über eine Internetverbindung auf einen Raspberry Pi zugreifen kann. Auf diesem läuft eine benutzerfreundliche Softwarelösung, die eine visuelle Eingabe von Parametern sowie die Steuerung des PVMS ermöglicht. Die vom Benutzer bereitgestellten Daten werden verarbeitet und an einen über SPI angebundenen STM32-Mikrocontroller weitergeleitet. Dieser übernimmt die unterlagerte Ansteuerung der Steuerkarte zur Simulation der PV-Kennlinien.

Dabei entsteht eine strukturierte Datenpipeline, die den Informationsfluss vom Benutzer über den Raspberry Pi bis zum STM32 klar trennt und kontrolliert.

Das zugrunde liegende System basiert auf einer verteilten Client-Server-Architektur. Ein oder mehrere Benutzer interagieren über Desktop- oder Webanwendungen mit dezentralen Raspberry Pi Zero 2W Geräten, die jeweils mit einem STM32-Knoten verbunden sind. Die Kommunikation erfolgt über eine WebSocket- oder TCP-Verbindung.

Der Raspberry Pi agiert dabei als Serverinstanz. Ein in Python implementierter Dienst nimmt Anfragen des Clients entgegen und leitet diese über eine SPI-Schnittstelle an den Mikrocontroller weiter. Jede Einheit ist autark betreibbar und kann unabhängig konfiguriert werden. Die Verbindung zwischen Benutzer und Server erfolgt über WLAN eine galvanische Trennung zwischen Steuer- und Hochspannungselektronik ist dadurch ebenfalls gegeben.

Diese modulare Struktur erlaubt eine einfache Skalierung, z. B. in Testfeldern mit mehreren Simulationspunkten oder für parallele Entwicklungsszenarien.

6.2 Prinzip der Softwarelösung

Das PVMS-System verfolgt den Anspruch, eine möglichst intuitive, performante und flexible Benutzersteuerung zu ermöglichen. Kernkomponente ist dabei eine grafische Benutzeroberfläche, welche die folgenden Simulationsmodi unterstützt:

- **Statische Kennlinie** (konstante PV-Kennlinie),
- **Dynamische Verschattung** (zeitlich variierende Abschattung einzelner Zellbereiche),
- **Dynamische Einstrahlung** (simulierter Tagesverlauf der Sonneneinstrahlung).

Die GUI wurde mit der Python-Bibliothek *PySide6*[Lit10] realisiert. Dieses offizielle Qt-Binding für Python ermöglicht eine performante Entwicklung mit Zugriff auf das vollständige Qt-Framework. Gerade im Umfeld des Raspberry Pi bietet PySide6 Vorteile hinsichtlich Verfügbarkeit, Wartbarkeit und schneller Implementierung.

Für die zukünftige Erweiterung wurde zudem eine Umsetzung als Webanwendung konzipiert. Hierfür kommt *Qt for WebAssembly* zum Einsatz.[Lit11] Dabei wird die bestehende PySide6-Anwendung mit Hilfe des Qt-WebAssembly-Compilers in ein kompaktes Binärformat übersetzt, welches im Browser ausgeführt werden kann, ohne Plugin, rein in der Browser-Sandbox. Die Webanwendung agiert dabei hauptsächlich als Parser und Weiterleiter: Sie verarbeitet Benutzereingaben und kommuniziert bidirektional über WebSockets mit der Desktop-Applikation auf dem Raspberry Pi.

Diese hybride Architektur, bestehend aus Desktop- und Webapplikation, vereint die Vorteile nativer Performance mit der Flexibilität moderner Webtechnologien. Durch die Trennung von GUI, Serverlogik und Mikrocontrollersteuerung ergibt sich ein klar definiertes Schichtenmodell, das die Wartung, Weiterentwicklung und Erweiterung der Software vereinfacht.

Im nächsten Abschnitt werden die einzelnen Softwarekomponenten der Desktop-Applikation im Detail beschrieben.

6.3 Desktop-Applikation auf dem Raspberry Pi

Die grafische Benutzeroberfläche (GUI) des PVMS wurde als Desktop-Applikation auf dem Raspberry Pi implementiert. Sie bildet die zentrale Schnittstelle zwischen Benutzer und System und ermöglicht die Steuerung verschiedener Betriebsmodi, darunter die Auswahl und Simulation von statischen sowie dynamischen Kennlinien.

Die Anwendung läuft direkt auf dem Betriebssystem des Raspberry Pi und nutzt dessen Systemressourcen zur Visualisierung und Interaktion. Da der Raspberry Pi häufig als Embedded-System ohne direkte Peripheriegeräte wie Monitor, Tastatur oder Maus betrieben wird, erfolgt der Zugriff auf die grafische Oberfläche in der Regel über das Netzwerk.

6.3.1 Zugriff und Fernsteuerung

Für eine komfortable Fernsteuerung kommt ein sogenannter VNC-Viewer (Virtual Network Computing) zum Einsatz. Dabei handelt es sich um eine Technologie zur Übertragung des Bildschirminhalts sowie zur Weiterleitung von Maus- und Tastatureingaben über das Netzwerk. Auf dem Raspberry Pi läuft ein VNC-Server, der den grafischen Desktop bereitstellt und Benutzereingaben entgegennimmt.

Ein Benutzer kann sich von einem beliebigen Endgerät (z. B. Laptop, Tablet oder Smartphone) mit dem Raspberry Pi verbinden, indem er einen VNC-Viewer verwendet. Nach erfolgreichem Verbindungsaufbau wird die Desktop-Oberfläche des Raspberry Pi auf dem

entfernten Gerät dargestellt, sodass die Anwendung bedient werden kann, als befände man sich direkt vor Ort.

Diese Lösung hat den Vorteil, dass keine zusätzliche Peripherie benötigt wird und die Bedienung des Systems bequem über bestehende Netzwerke erfolgen kann. Gerade in mobilen oder dezentralen Anwendungsszenarien, wie im Fall des PVMS, stellt dies eine besonders praktische und ressourcenschonende Lösung dar.

6.3.2 Geplante Erweiterung

Ziel der weiteren Entwicklung ist es, die Desktop-Applikation über eine WebSocket-Verbindung mit einem externen Gerät zu koppeln. Dies ermöglicht den direkten Zugriff über die IP-Adresse und den zugehörigen Port des Raspberry Pi-Servers.

Benutzereingaben werden dabei über eine in WebAssembly implementierte grafische Oberfläche an die Desktop-Anwendung übermittelt. Die genaue Umsetzung dieses Kommunikationsprinzips wird in einem späteren Kapitel detailliert beschrieben.

6.3.3 Softwareübersicht

Um einen Überblick über die Struktur der zu entwickelnden Software zu erhalten, wurde ein Klassendiagramm erstellt, das in Abbildung 6.2 dargestellt ist.

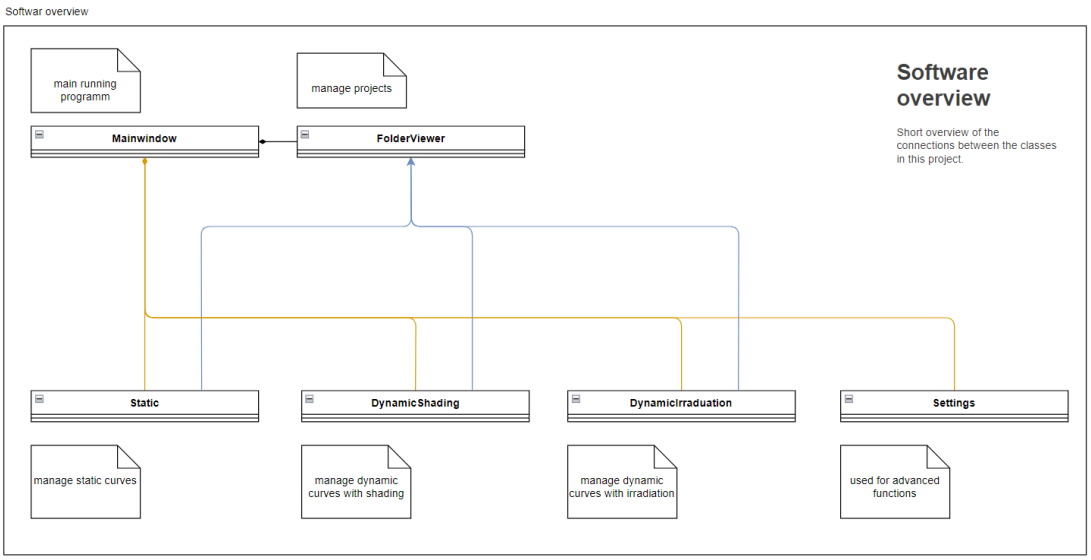


Abbildung 6.2: Übersicht der Desktop-Applikation

Dieses Klassendiagramm veranschaulicht die grundsätzliche Struktur der Software. Auf eine detaillierte Darstellung der Methoden und Attribute wurde bewusst verzichtet, um die Übersichtlichkeit zu wahren.

Zusätzlich zum Klassendiagramm wurden weitere Diagramme erstellt, die die in Abbildung 6.2 dargestellten Unterklassen *Static*, *DynamicShading*, *DynamicIrradiation* und *FolderViewer* im Detail zeigen. Die Klasse *Settings* wird hingegen nicht weiter betrachtet, da sie bisher lediglich als Entwurf für eine spätere Implementierung vorliegt und noch nicht vollständig realisiert wurde. Die zugehörigen Diagramme befinden sich am Ende der entsprechenden Kapitel.

6.3.4 Benutzeroberfläche (GUI)

Nachdem die grundlegende Softwarestruktur definiert wurde, erfolgt im nächsten Schritt die Spezifikation der konkreten Anforderungen und Funktionen der grafischen Benutzeroberfläche. Das GUI wurde mit der Bibliothek *PySide6* entwickelt. In Abbildung 6.3 ist die Struktur der verschiedenen GUI-Seiten dargestellt.

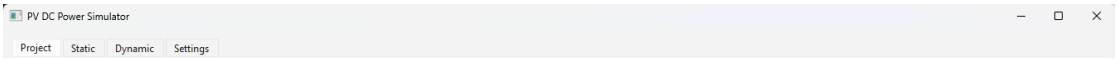


Abbildung 6.3: Seitendarstellung der grafischen Benutzeroberfläche

Die dargestellten Seiten spiegeln den Aufbau sowie die Benutzerführung der Anwendung wider. Die Abbildung zeigt die sogenannte Header-Ansicht des GUIs. Über die verschiedenen Seiten (Pages) kann sich der Nutzer durch die Anwendung navigieren und dabei gezielt Funktionen zur Kennliniendarstellung und -übertragung auswählen.

Die Seiten, die für die Handhabung der Kennlinien verantwortlich sind, heissen *Static* und *Dynamic*. Beide folgen einem einheitlichen Aufbau und bestehen typischerweise aus den folgenden Komponenten:

- einer Eingabemaske zur Parametereinstellung,
- einem Grafikfenster zur Darstellung der Kennlinie,
- einem Bereich *Calculated Values*, in dem berechnete Werte angezeigt werden,
- einem Bereich *Measured Values*, in dem die vom STM32 empfangenen Messdaten dargestellt werden,
- sowie einem *Control Panel*, das die Kommunikation mit dem STM32 steuert.

Diese Auflistung stellt eine grobe Übersicht dar. Die konkrete Funktionsweise sowie der softwaretechnische Aufbau aller Seiten werden in den folgenden Kapiteln im Detail beschrieben.

6.3.5 Projekt

Diese Seite ist auf der Klasse *FolderViewer* aufgebaut. Am Ende dieses Kapitels wird auf das Klassendiagramm des *FolderViewers* eingegangen. Zunächst folgt die Bedienung für den User.

Die Projektseite dient dem Benutzer dazu, ein neues Projekt zu erstellen, ein bestehendes Projekt zu laden oder zu speichern. Um ein Projekt speichern oder öffnen zu können, muss zunächst ein Projekt aus der Liste ausgewählt werden. Ist kein Projekt selektiert, sind die Schaltflächen *Open* und *Save* deaktiviert, wie in Abbildung 6.4 zu sehen ist.

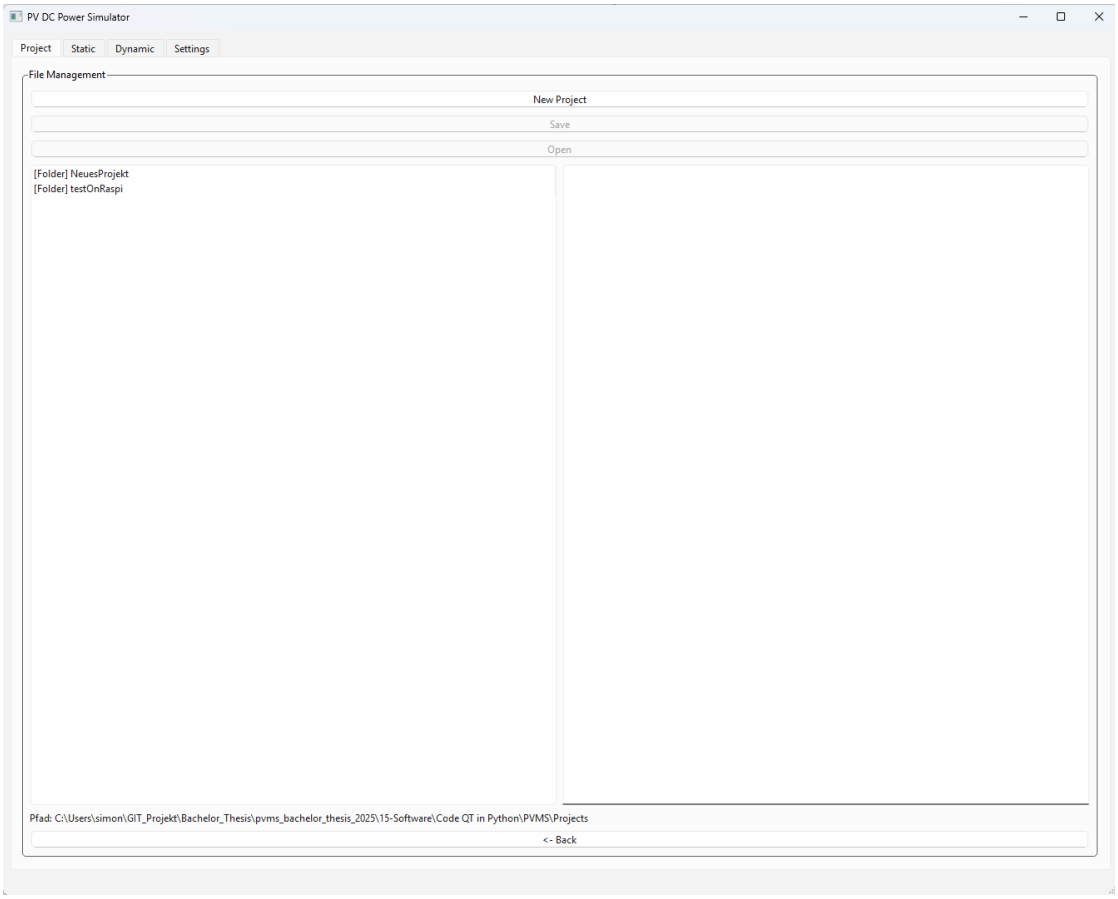


Abbildung 6.4: Startseite des GUIs mit Projektübersicht

Ein Projekt kann durch einen Doppelklick geöffnet werden. Innerhalb eines geöffneten Projekts lassen sich die zugehörigen Daten über die Schaltflächen *Open* und *Save* laden bzw. speichern. Auch innerhalb eines Projekts können die einzelnen Datensätze durch einen Doppelklick angezeigt werden, wie in Abbildung 6.5 dargestellt.

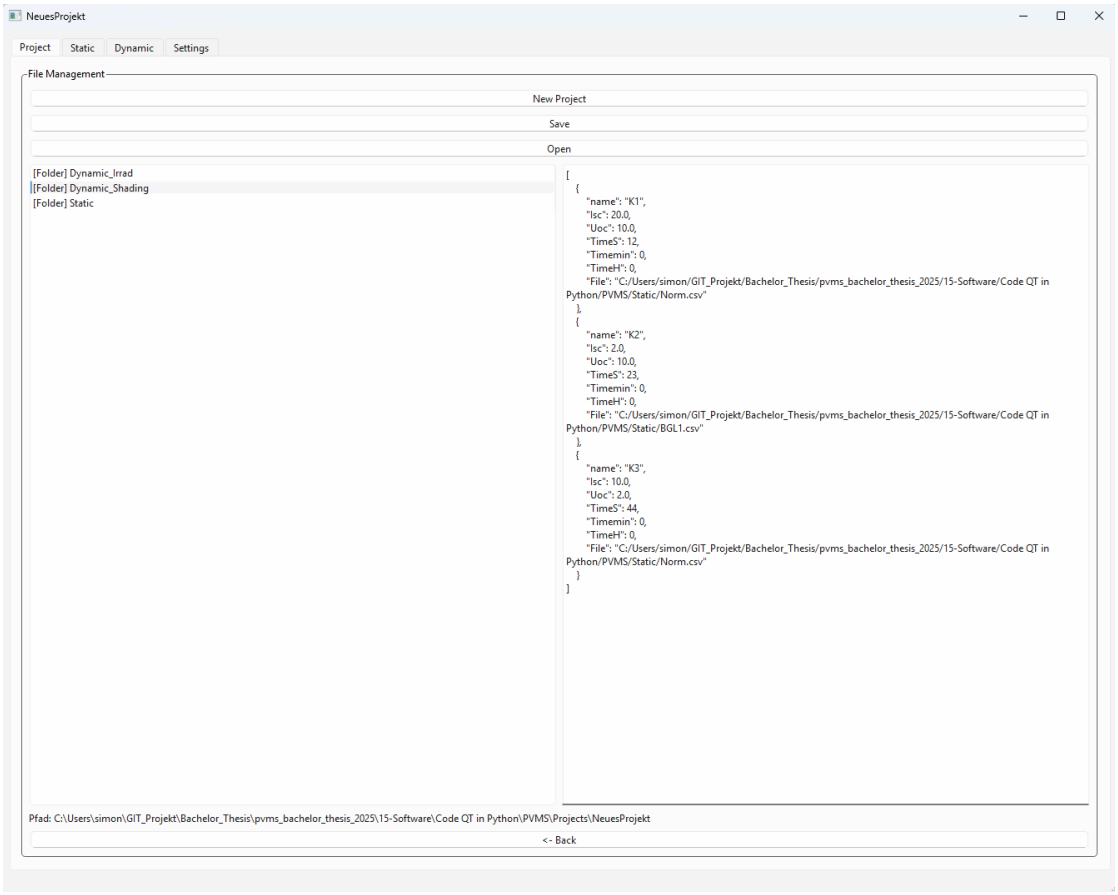


Abbildung 6.5: Projektansicht innerhalb des GUIs

Nach dem erfolgreichen Öffnen eines Projekts wird der Benutzer automatisch auf die nächste Seite des GUIs weitergeleitet, die dem statischen Kennlinienmodus (*Static*) zugeordnet ist. Für genauere Benutzung der Seite Project kann das Manual eingesehen werden.

Nun folgt das Klassendiagramm des *FolderViewer*, welches eine Übersicht über die Funktionen des *FolderViewer* liefert.

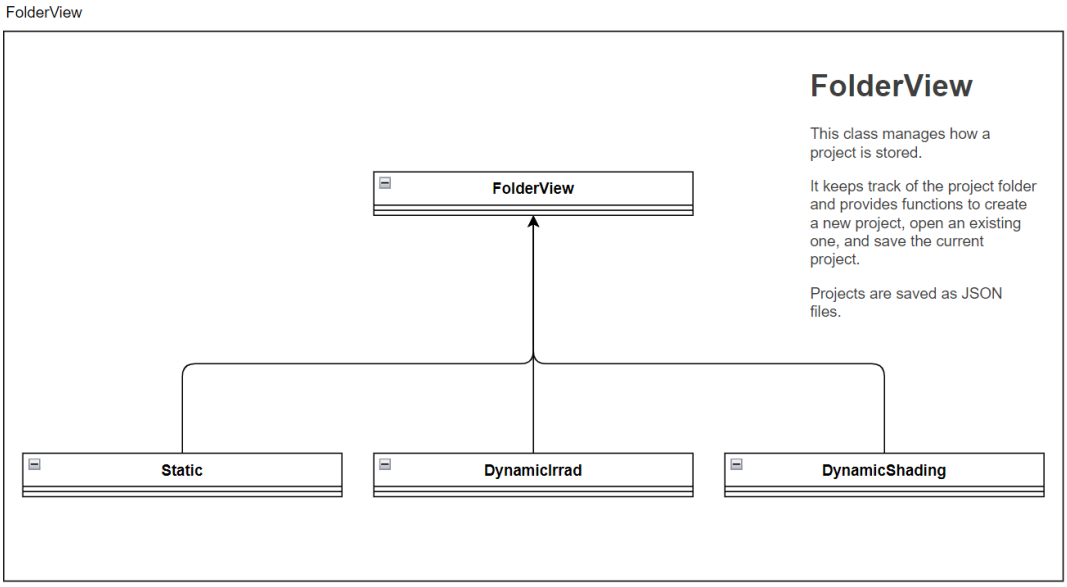


Abbildung 6.6: FolderViewer Klassendiagramm

In Abbildung 6.6 ist ersichtlich, dass alle Unterklassen des GUIs mit dem *FolderViewer* verbunden sind. Dies liegt daran, dass der *FolderViewer* wissen muss, welche Daten gespeichert werden sollen und wie diese später wieder eingelesen werden können.

Die Daten werden vom *FolderViewer* im JSON-Format abgespeichert und können bei Bedarf wieder geladen werden. Die Klassen *Static*, *DynamicIrrad* und *DynamicShading* implementieren jeweils eigene *open()*- und *save()*-Funktionen. Diese sind dafür zuständig, die spezifischen Daten der jeweiligen Klasse zu laden bzw. zu speichern.

Der *FolderViewer* selbst ruft diese Funktionen auf und übernimmt die Organisation der Dateipfade. Er zeigt an, in welchem Ordner man sich befindet und welche Projektdaten dort abgelegt sind. Das eigentliche Speichern und Laden der Inhalte wird von den jeweiligen Klassen übernommen.

6.3.6 Static

Die Seite *Static* dient der Simulation einer einzelnen statischen Kennlinie. In diesem Bereich kann der Benutzer die Kennlinienparameter wie I_{sc}, I_{mpp}, U_{oc} und U_{mpp} einstellen. Optional lässt sich auch die Einstrahlung anpassen, um realistischere Bedingungen zu simulieren. Sobald alle gewünschten Parameter gesetzt und eine geeignete Kurvendatei ausgewählt wurde, kann über den Knopf *Initialize* die Kennlinie initialisiert werden.

Nach erfolgreicher Initialisierung kann das Board ausgewählt und verbunden (*Connected*) werden. Die Datenübertragung zwischen dem Raspberry Pi und dem STM32 erfolgt anschliessend über das SPI-Protokoll 5.1.4 und wird durch die Schaltfläche *Enable/Disable* gestartet / beendet.

Eingehende Messdaten vom STM32 werden in der grafischen Darstellung als Kreuzpunkte visualisiert und gleichzeitig im Bereich *Measured Values* tabellarisch angezeigt. Diese Daten können anschliessend über den Button *Export Data* in das aktuell geöffnete Projekt exportiert werden.

Die Benutzeroberfläche dieser Seite ist in Abbildung 6.7 dargestellt.

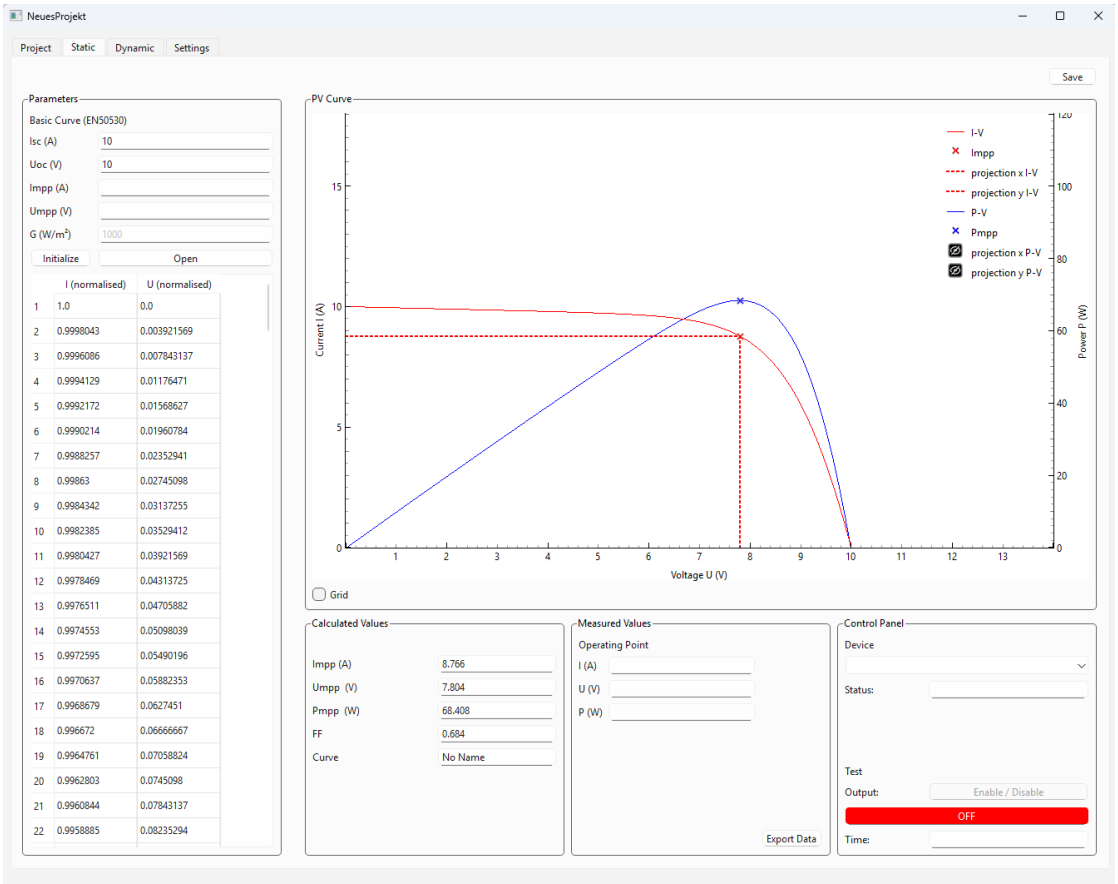


Abbildung 6.7: Ansicht der statischen Simulationsseite im GUI

Die grundlegende Funktionsweise der Seite wurde bereits erläutert. Um nun auch den softwaretechnischen Aufbau zu verstehen, folgt das Klassendiagramm in Abbildung 6.8.

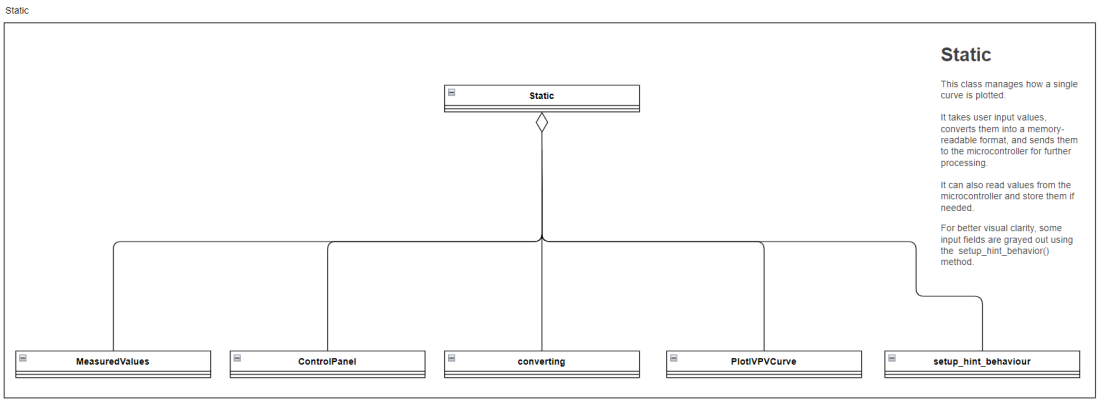


Abbildung 6.8: Klassendiagramm der Seite *Static*

Wie in Abbildung 6.8 ersichtlich, basiert die Seite *Static* auf der gleichnamigen Hauptklasse *Static*, die eine Vielzahl aggregierter Klassen verwendet. Wie bereits zuvor erwähnt, ist der Aufbau dieser Komponenten bei den verschiedenen Seiten grösstenteils identisch.

So enthält jede dieser Seiten z. B. die Klasse *MeasuredValues*, welche die vom Mikrocontroller empfangenen Messdaten verarbeitet und bei Bedarf speichert.

Die Klasse *ControlPanel* ist für den Aufbau und die Verwaltung der Kommunikationsverbindung mit dem Mikrocontroller verantwortlich. Zur weiteren Aufbereitung der PV-Kennliniendaten dient die Klasse *Converting*. Diese sorgt dafür, dass die Daten in ein Format überführt werden, das für die Übertragung und Zwischenspeicherung geeignet ist, sodass der Mikrocontroller die Informationen direkt im Speicher ablegen kann.

Ein zentrales Element der Oberfläche ist selbstverständlich die grafische Darstellung der Kennlinie. Um den GUI-Code übersichtlich und benutzerfreundlich zu gestalten, werden vordefinierte Werte in der Darstellung optisch hervorgehoben z. B. durch graue Farbgebung. Für diese Funktionalität kommt die Klasse *SetupHintBehaviour* zum Einsatz.

Die Architektur der Seite *Static* lässt sich somit modular und nachvollziehbar strukturieren. Gleichzeitig ermöglicht der gemeinsame Aufbau mit den dynamischen Seiten eine Wiederverwendung wesentlicher Komponenten.

6.3.7 Dynamic

Die Seite *Dynamic* bietet zwei zusätzliche Ansichten innerhalb der grafischen Benutzeroberfläche. Wie bereits in Kapitel 6.2 beschrieben, soll die Software in der Lage sein, Kennlinien zu simulieren, deren Parameter sich dynamisch verändern entweder in Abhängigkeit einer zeitlich variierenden Einstrahlung oder infolge einer Verschattung, die die gesamte Kurve beeinflusst.

Um diese beiden Simulationstypen getrennt darstellen und durchführen zu können, wurden zwei eigenständige Unterseiten innerhalb der *Dynamic*-Ansicht implementiert:

- *Dynamic Irradiation*
- *Dynamic Shading*

Diese Seiten ermöglichen es, die entsprechenden Simulationen unabhängig voneinander durchzuführen. Die konkrete Funktionsweise und Umsetzung dieser beiden Komponenten wird in den folgenden Kapiteln näher erläutert.

Dynamic Irradiation

Auf der Seite *Dynamic Irradiation* kann eine Kennlinie parametriert werden. Im Unterschied zur statischen Seite besteht hier zusätzlich die Möglichkeit, den Kurzschlussstrom (I_{sc}) und die Leerlaufspannung (U_{oc}) über ein externes Einstrahlungsprofil dynamisch zu variieren.

Wie bereits bei der statischen Kennlinie muss zunächst eine Kurvendatei importiert und die gewünschten Parameter gesetzt werden. Anschliessend erfolgt die Initialisierung der Kurve über den *Init*-Button.

Zusätzlich muss eine zweite Datei geladen werden, welche das Einstrahlungsprofil enthält. Diese Datei ist extern zu erstellen und im CSV-Format abzulegen. Die Datenstruktur muss dabei dem in Abbildung 6.9 dargestellten Format entsprechen.

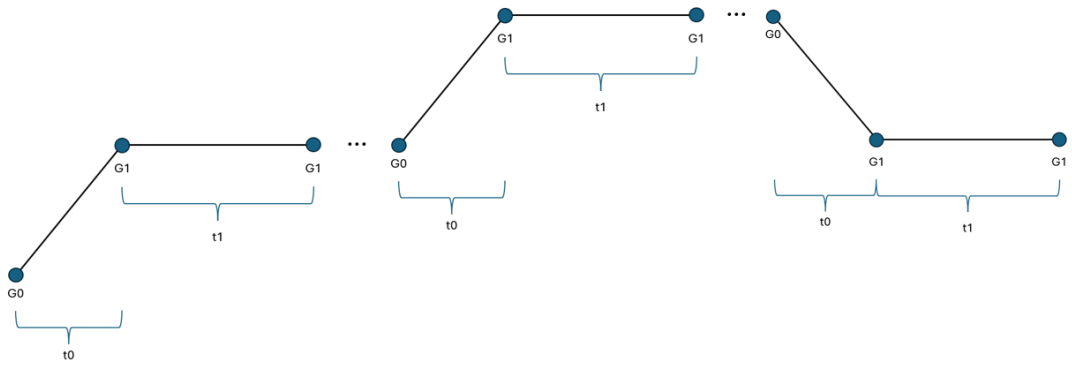


Abbildung 6.9: Datenstruktur des Einstrahlungsprofils im CSV-Format

Sobald beide Dateien erfolgreich geladen und initialisiert wurden, kann die Kommunikation mit dem STM32 über den SPI-Bus aufgebaut werden. Ist die Verbindung aktiv, lässt sich das Einstrahlungsprofil über eine integrierte *Playbar* abspielen.

Während der Simulation wird im Sekundentakt (1 s) der aktuelle I_{sc}-Wert gemäss dem Einstrahlungsprofil berechnet und zusammen mit dem zugehörigen U_{oc} an den STM32 übertragen. Wird die Simulation gestoppt, sendet das System weiterhin den zuletzt berechneten I_{sc}-Wert. Erst bei einem erneuten Start der Simulation (durch Betätigen der *Play*-Schaltfläche) werden neue Werte aus dem Profil übernommen und erneut an den STM32 gesendet.

Das Design der Seite *Dynamic Irradiation* ist in Abbildung 6.10 dargestellt.

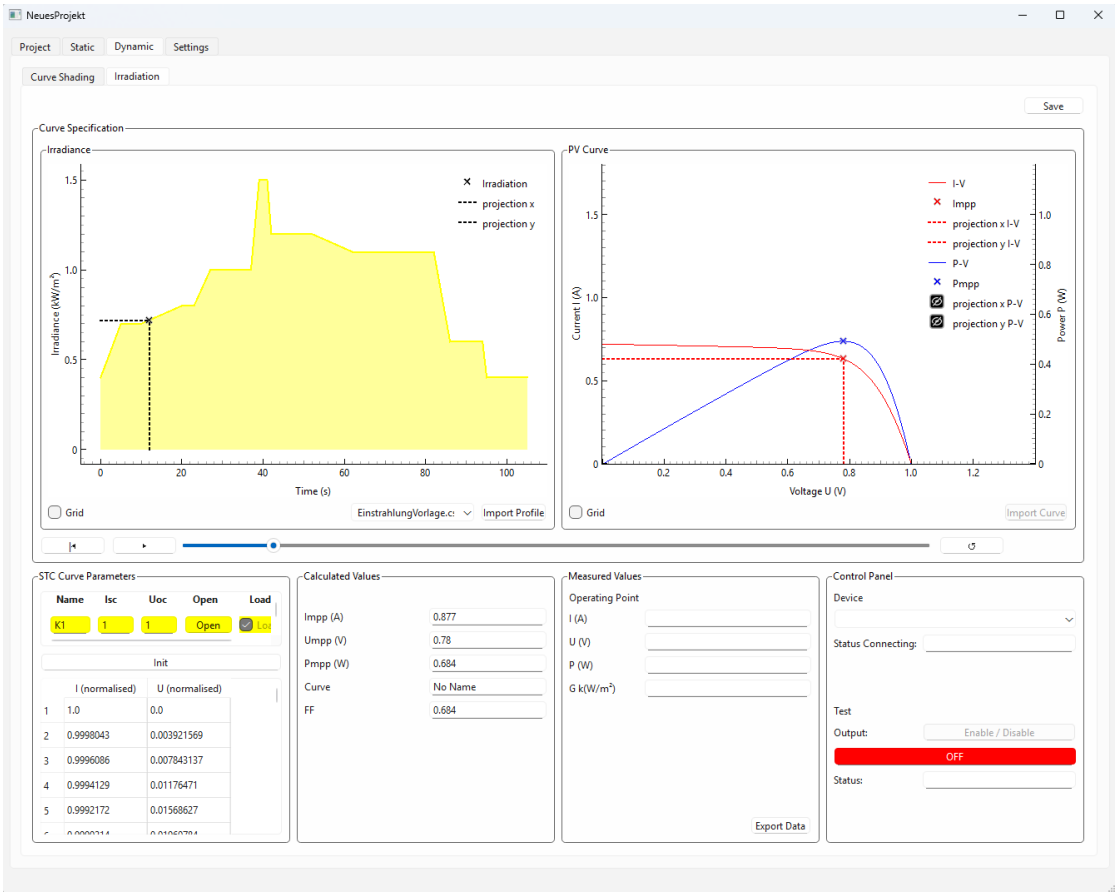


Abbildung 6.10: Design der *Dynamic Irradiation*-Seite im GUI

Aus dem zuvor beschriebenen Funktionsprinzip ergibt sich das in Abbildung 6.11 dargestellte Klassendiagramm, welches die softwaretechnische Umsetzung der Seite *Irradiation* veranschaulicht.

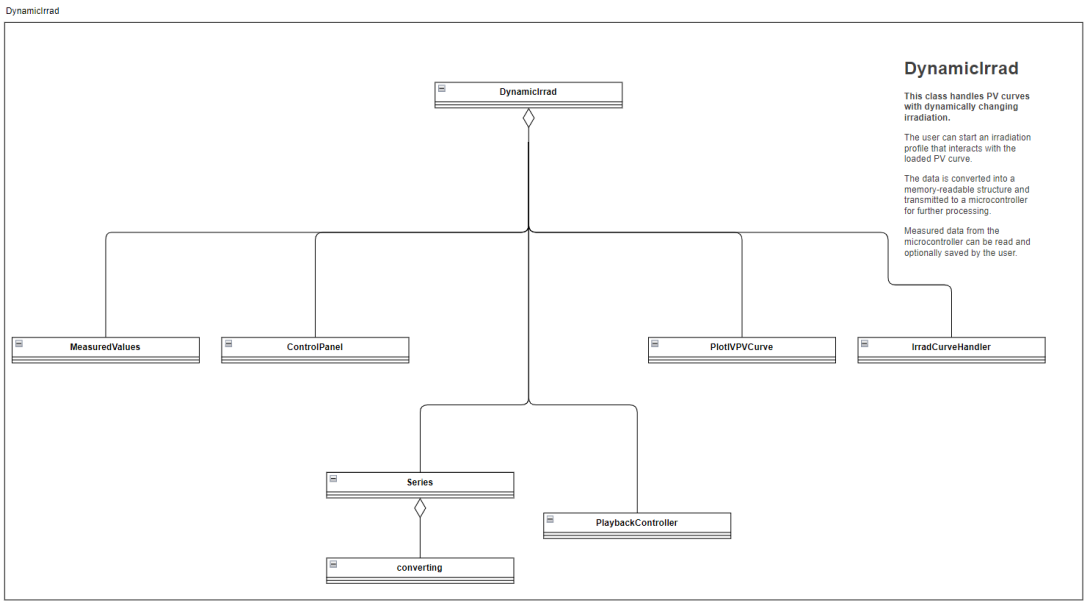


Abbildung 6.11: Klassendiagramm der Seite *Dynamic Irradiation*

Im Vergleich zur statischen Kennliniensimulation umfasst die Seite *Dynamic Irradiation* zusätzliche Klassen, die spezifische Aufgaben im Zusammenhang mit der zeitabhängigen Einstrahlung übernehmen. Neben den bereits bekannten Basisklassen wurden folgende neue Komponenten implementiert:

- **IrradCurveHandler**: Diese Klasse ist verantwortlich für das Laden und die Verwaltung des zeitlich aufgelösten Einstrahlungsprofils. Sie stellt sicher, dass das Profil korrekt geladen, interpretiert und im GUI visualisiert wird. Der *IrradCurveHandler* übernimmt somit die zentrale Rolle bei der Handhabung der Einstrahlungsdaten.

- **Series**: Diese Klasse beinhaltet eine Sammlung an PV-Kennlinien. Für die Seite *Dynamic Irradiation* ist typischerweise nur eine einzelne Kennlinie erforderlich. Dennoch wurde die Klasse *Series* aus Gründen der Wiederverwendbarkeit und einheitlichen Strukturierung integriert, da sie ursprünglich für die Seite *Dynamic Shading* entwickelt wurde.

- **PlaybackController**: Diese Komponente steuert die Abspielfunktion der Zeitleiste. Sie verwaltet die aktuelle Position innerhalb des Einstrahlungsprofils und koordiniert die Synchronisation mit der grafischen Darstellung und den empfangenen Messwerten.

Die übrigen Klassen entsprechen funktional jenen der Seite *Static* und wurden entsprechend wiederverwendet. Durch diese modulare Struktur wird eine konsistente und wartbare Codebasis gewährleistet.

Dynamic Shading

Die Seite *Dynamic Shading* bietet eine besondere Funktionalität: Anders als bei den vorherigen Seiten kann hier nicht nur eine einzelne Kennlinie parametrisiert werden, sondern es besteht die Möglichkeit, mehrere Kennlinien gleichzeitig zu laden und zu initialisieren. Dies ermöglicht die Simulation von Verschattungen, bei denen verschiedene Betriebszustände nacheinander durchlaufen werden.

Das System aktiviert dabei die vorbereiteten Kennlinien sequentiell, wobei keine Übergänge (wie Rampen oder kontinuierliche Steigungen) berücksichtigt werden. Im Gegensatz zur Seite *Dynamic Irradiation*, bei der ein zeitlich verlaufendes Profil simuliert wird, erfolgt die Umschaltung beim Shading direkt und abrupt.

Die jeweils eingestellten Strom- und Spannungswerte (I_{sc}, U_{oc}) werden für die definierte Dauer konstant an den STM32 übertragen. Somit kann eine zeitgesteuerte Abfolge von Verschattungszuständen nachgebildet werden, jedoch ohne gleitenden Übergang.

Das Design und die Struktur der *Dynamic Shading*-Seite sind in Abbildung 6.12 dargestellt.

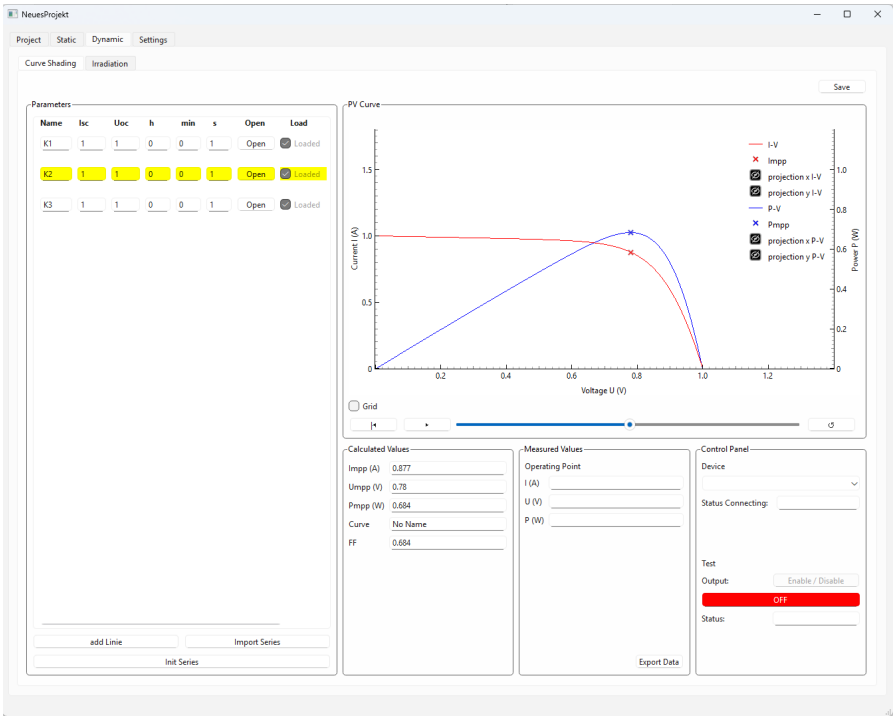


Abbildung 6.12: Design der *Dynamic Shading*-Seite im GUI

Auch für die Seite *Dynamic Shading* wurde ein Klassendiagramm erstellt, welches die Abhängigkeiten und Beziehungen der beteiligten Klassen veranschaulicht.

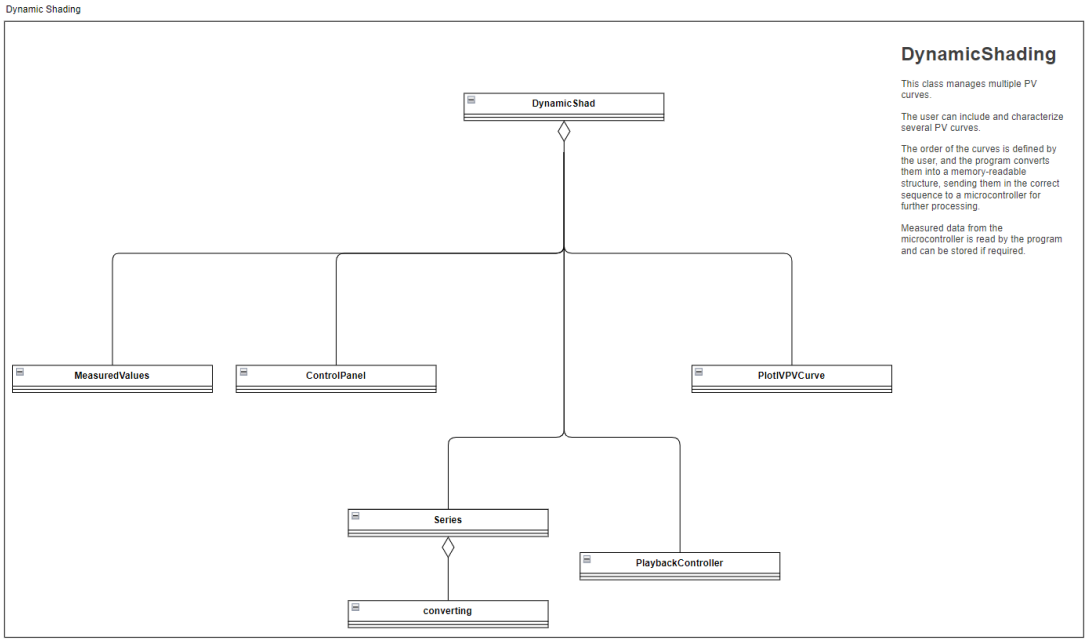


Abbildung 6.13: Klassendiagramm der Seite *Dynamic Shading*

Die Struktur dieser Seite ähnelt in weiten Teilen jener der *Dynamic Irradiation*-Seite (siehe Abschnitt 6.3.7). Der wesentliche Unterschied besteht darin, dass hier keine externe Einstrahlungskurve verwendet wird. Daher entfällt die Klasse *IrradCurveHandler*.

Stattdessen wird die zeitliche Dynamik der Simulation durch benutzerdefinierte Eingaben gesteuert. Die Klasse *Series* übernimmt hierbei die Verwaltung mehrerer PV-Kennlinien, welche individuell geladen und in zeitlicher Reihenfolge durchlaufen werden. Die Simulation basiert somit auf einer Abfolge unterschiedlicher Kennlinien, deren zeitliche Übergänge durch Benutzereingaben bestimmt werden.

Die übrigen Klassen, darunter die grafische Darstellung, der Bereich Measured Values sowie die Kommunikationsschnittstelle, entsprechen funktional jenen der bereits vorgestellten Seiten und wurden dementsprechend wiederverwendet. Durch diese einheitliche Struktur kann der Code effizient gepflegt und bei Bedarf erweitert werden.

6.3.8 Settings

Die Seite *Settings* ist primär für Spezialfälle vorgesehen. Hier kann unter anderem der Startblock definiert werden, in dem die erste Kennlinie gespeichert werden soll. Dies ist insbesondere wichtig, um zu verhindern, dass manuell genutzte Kennlinien (z. B. im Test- oder Entwicklungsbetrieb) versehentlich überschrieben oder gelöscht werden. Auf die genaue Funktionsweise des manuellen Betriebs wird im Kapitel zur Firmware 7 näher eingegangen.

Zusätzlich kann hier festgelegt werden, ob der Ausgang des Boards automatisch deaktiviert werden soll, nachdem ein Profil im Modus *Dynamic Irradiation* oder *Dynamic Shading* vollständig abgearbeitet wurde. Wird diese Option aktiviert, wird der Ausgang nach Abschluss der Simulation automatisch abgeschaltet.

Standardmässig sind diese beiden Einstellungen wie in Abbildung 6.14 gezeigt konfiguriert: Der Speicherblock ist auf 16 gesetzt (die ersten 16 Speicherplätze werden durch die Software nicht überschrieben), und die Option zur automatischen Abschaltung des Ausgangs nach Profilende ist deaktiviert.

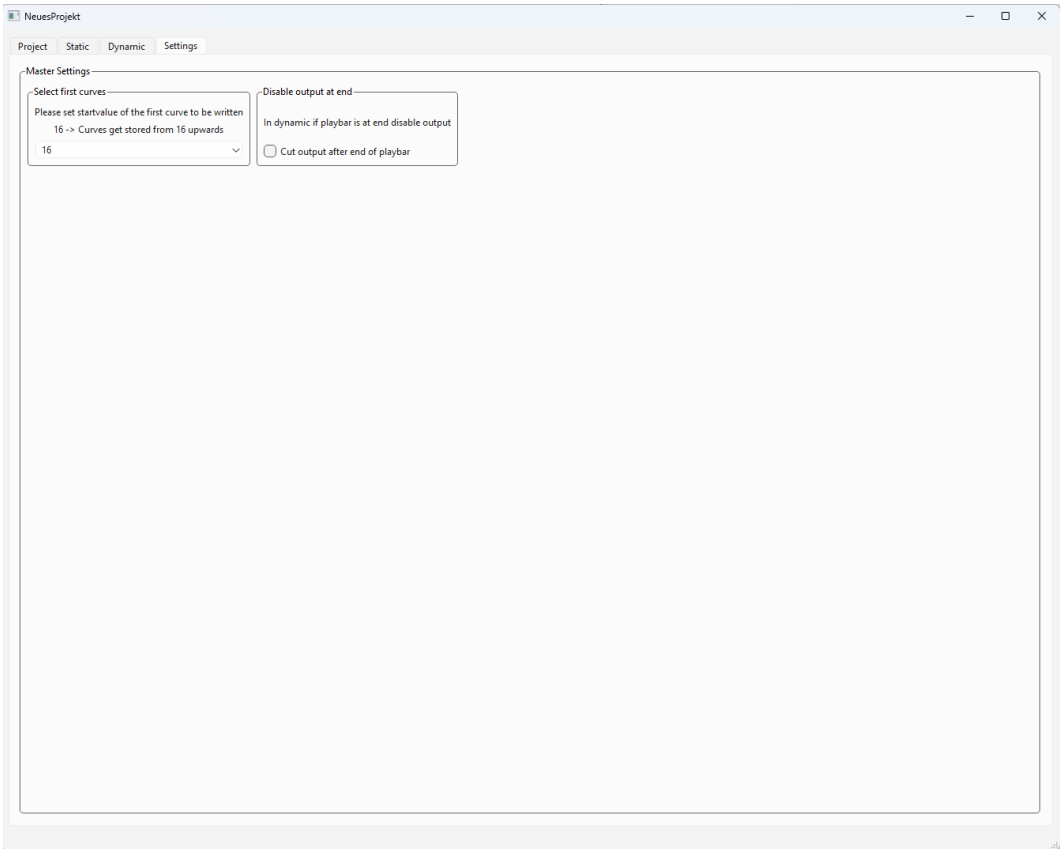


Abbildung 6.14: Design der *Settings*-Seite im GUI

6.4 WebAssembly

In diesem Kapitel wird beschrieben, wie die Software für den webbasierten Einsatz vorbereitet werden kann. Um eine Kommunikation über das Internet zu ermöglichen, ist ein Netzwerkprotokoll erforderlich, das typischerweise auf TCP/IP basiert. Dieses Protokoll erlaubt es einem Client, auf einen Server zuzugreifen.

Der Client stellt in diesem Fall eine Webapplikation dar, die vom Benutzer über einen Webbrowser bedient werden kann. Über einen definierten Socket bzw. Port kann diese Webapplikation mit einem Server kommunizieren, in diesem Projekt übernimmt diese Rolle die Desktopapplikation, die auf dem Raspberry Pi läuft.

Die Webapplikation soll mit *Qt for WebAssembly* kompiliert werden. Sie fungiert dabei hauptsächlich als Parser zwischen Benutzer und Desktopanwendung. Das bedeutet: Auf dem Raspberry Pi läuft ein Server, auf dem die Desktopapplikation aktiv ist. Diese Anwendung steuert sowohl die Hardware (z. B. den SPI-Bus) als auch andere Systemfunktionen. Eingehende Befehle aus der Webapplikation werden über eine WebSocket-Verbindung an die Desktopapplikation weitergeleitet. Ändert sich der Status der Desktopapplikation, so werden diese Änderungen auch an die Webapplikation zurückgespiegelt.

Ein schematisches Beispiel für diese Parserfunktionalität ist in Abbildung 6.15 dargestellt.

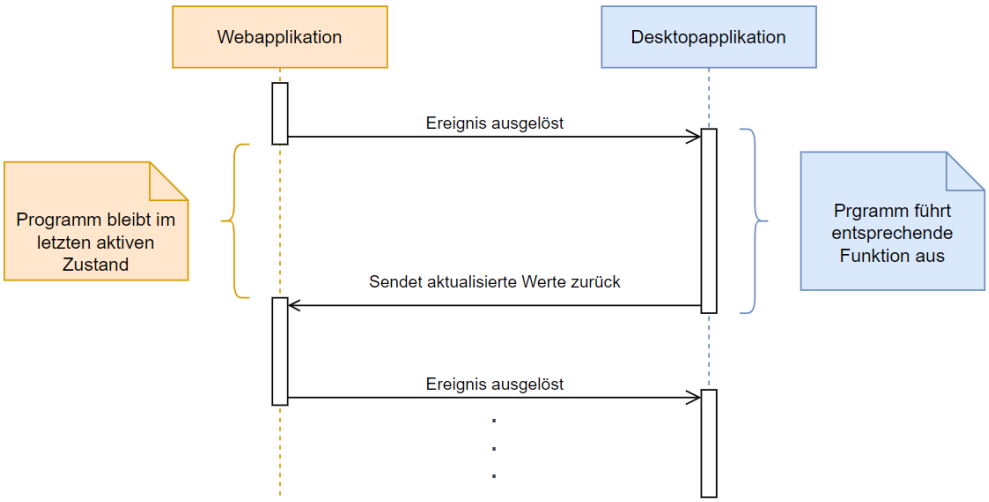


Abbildung 6.15: Kommunikationsstruktur zwischen Webapplikation und Desktopapplikation

Zum aktuellen Stand konnte die Webapplikation im Rahmen dieses Projekts aufgrund des zeitlichen Aufwands noch nicht umgesetzt werden. Eine zukünftige Implementierung ist jedoch vorgesehen und würde eine flexible, plattformunabhängige Steuerung des Systems ermöglichen.

6.5 Ordnerstruktur

In diesem Abschnitt wird die Struktur der Software erläutert, welche Komponente abgelet und wie diese organisiert sind.

Die gesamte Anwendung basiert auf einem Hauptprojekt mit der Datei (PVMS.py), welche als Einstiegspunkt dient und beim Start der Software ausgeführt werden muss. Für die Ausführung wird eine im Git-Repositor [Lit12] enthaltene virtuelle Umgebung (env) verwendet. Zur Inbetriebnahme muss diese Umgebung zunächst aktiviert werden. Anschliessend kann die Software über das aktivierte Terminal gestartet werden. Detaillierte Anweisungen zur Ausführung sind im README.txt im Root-Verzeichnis des Git-Repositories zu finden.

Die Software ist objektorientiert aufgebaut und verwendet Klassen, um wiederkehrende Funktionalitäten modular und wartbar zu gestalten. Jede Klasse ist in einem entsprechenden Ordner abgelegt, der ihrer Funktion innerhalb des Projekts entspricht.

Zusätzliche technische Informationen, insbesondere zu den enthaltenen Funktionen und Methoden, können über die automatisch generierte Doxygen-Dokumentation eingesehen werden.

Für weitere Einzelheiten zur Projektstruktur und zur Verwendung der Software wird auf die README.txt-Datei verwiesen.

7 Firmware

Die Firmware des PVMS ist vollständig in C implementiert. Die Aufgaben der Firmware sind:

- Steuerung der Control-Board (Tasterabfrage, Encoder-Auswertung sowie Ansteuerung von Monitor und LEDs)
- Schreiben des Speichers
- SPI-Kommunikation mit dem Raspberry Pi
- Steuerung der DACs für I_{sc}, U_{oc} und U_{ref}
- Messung der Ausgangsspannung und des Ausgangsstroms
- Verwaltung der Spannungskalierung
- verwaltung des Bus-Switches

Ursprünglich war geplant, beide Kerne des STM32H757BIT6 (Cortex-M7 und Cortex-M4) zu verwenden, um Aufgaben effizient aufzuteilen. Aufgrund von Problemen mit dem OpenAMP-Framework 3.6 wird in der aktuellen Implementierung jedoch nur ein Kern eingesetzt. Das System funktioniert dadurch zwar fehlerfrei, allerdings auf Kosten der Effizienz und strukturellen Klarheit.

Das Kapitel 3.6 hat die Funktionsweise der Kommunikation zwischen den beiden Kernen sowie deren mögliche Implementierung erläutert.

In den folgenden Abschnitten werden die Funktionsblöcke, ihre Verknüpfungen sowie das zeitliche Verhalten des Systems mithilfe von Ablauf- und Zeitdiagrammen nachvollziehbar dargestellt. Ziel ist es, einen umfassenden Überblick über die interne Logik und Struktur der Firmware zu vermitteln.

7.1 FreeRTOS

Das System wurde mit FreeRTOS implementiert und ist in vier Tasks organisiert:

- **DisplayHandlerTask:** Diese Task ist für die Steuerung des OLED-Displays verantwortlich (Kapitel 5.3).
- **BTNsHandlerTask:** Diese Task übernimmt das Auslesen der Taster, welche per Polling abgefragt werden (Kapitel 5.3).
- **OutputhandlerTask:** Diese Task ist für die Speicherverwaltung zuständig (Kapitel 5.1.9).
- **MemoryHandlerTask:** Diese Task steuert die Ausgänge. Sie verwaltet die drei externen DACs für U_{oc}, I_{sc} und U_{ref} sowie das Auslesen der ADCs (Kapitel 5.1.8 und Kapitel 5.1.6).

Die Interaktion und Kommunikation zwischen den einzelnen Tasks sowie externen Komponenten ist in Abbildung 7.1 schematisch dargestellt.

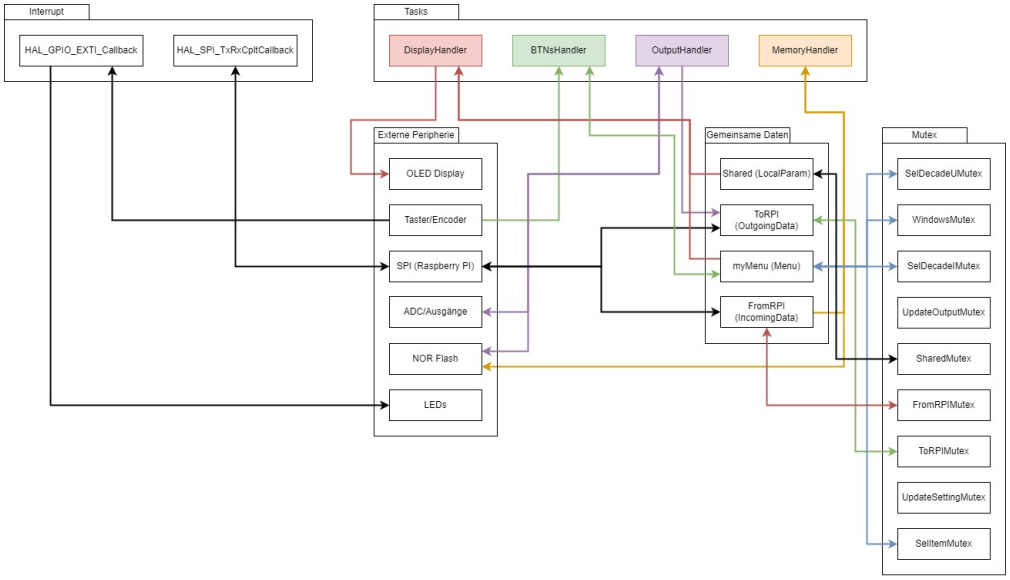


Abbildung 7.1: Struktur des Firmware

In den folgenden Kapiteln werden die einzelnen Tasks und deren Hauptfunktionen im Detail erläutert sowie eine Übersicht über den möglichen Betrieb im manuellen Modus dargestellt. Zusätzlich wird die Kommunikation mit dem Raspberry Pi über SPI beschrieben, die mithilfe von SPI-Interrupts und DMA (Direct Memory Access) umgesetzt wurde.

7.1.1 DisplayHandlerTask

Diese Task ist für die Verwaltung des OLED-Displays [Mat18] zuständig. Je nach aktuellem Menüstatus wird der entsprechende Bildschirm gezeichnet. Solange sich das System im Menü befindet, muss das aktuell ausgewählte Element bekannt sein. Beim Aufrufen eines bestimmten Fensters benötigt die Aufgabe je nach Fenster unterschiedliche Informationen, um den korrekten Inhalt darzustellen.

Ein Ablaufdiagramm der DisplayHandlerTask ist in Abbildung 7.2 dargestellt.

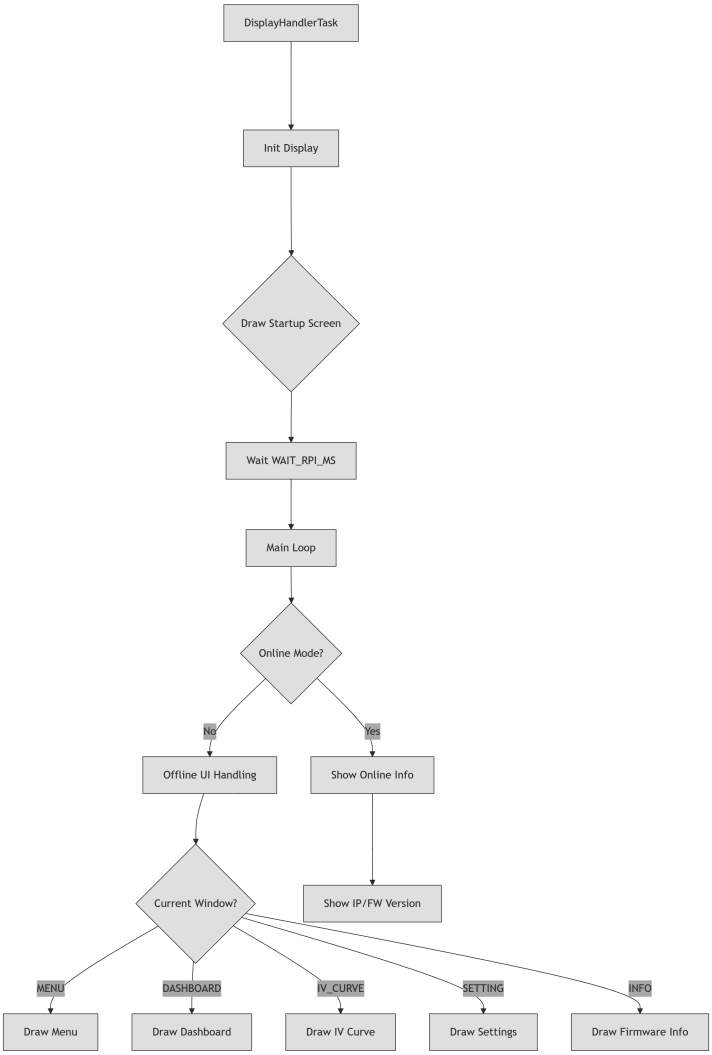


Abbildung 7.2: Ablaufdiagramm der DisplayHandlerTask

7.1.2 BTNsHandlerTask

Diese Task übernimmt das Abfragen der Tasten. Abhängig davon, welche Tasten gedrückt werden, ändert sich der Menüstatus. Die Funktion jeder Taste hängt vom aktuellen Zustand des Menüs ab (Siehe Anhang 7). Wenn eine Taste in einem bestimmten Menü keine definierte Funktion besitzt, wird sie ignoriert.

Ein Ablaufdiagramm der BTNsHandlerTask ist in Abbildung 7.3 dargestellt.

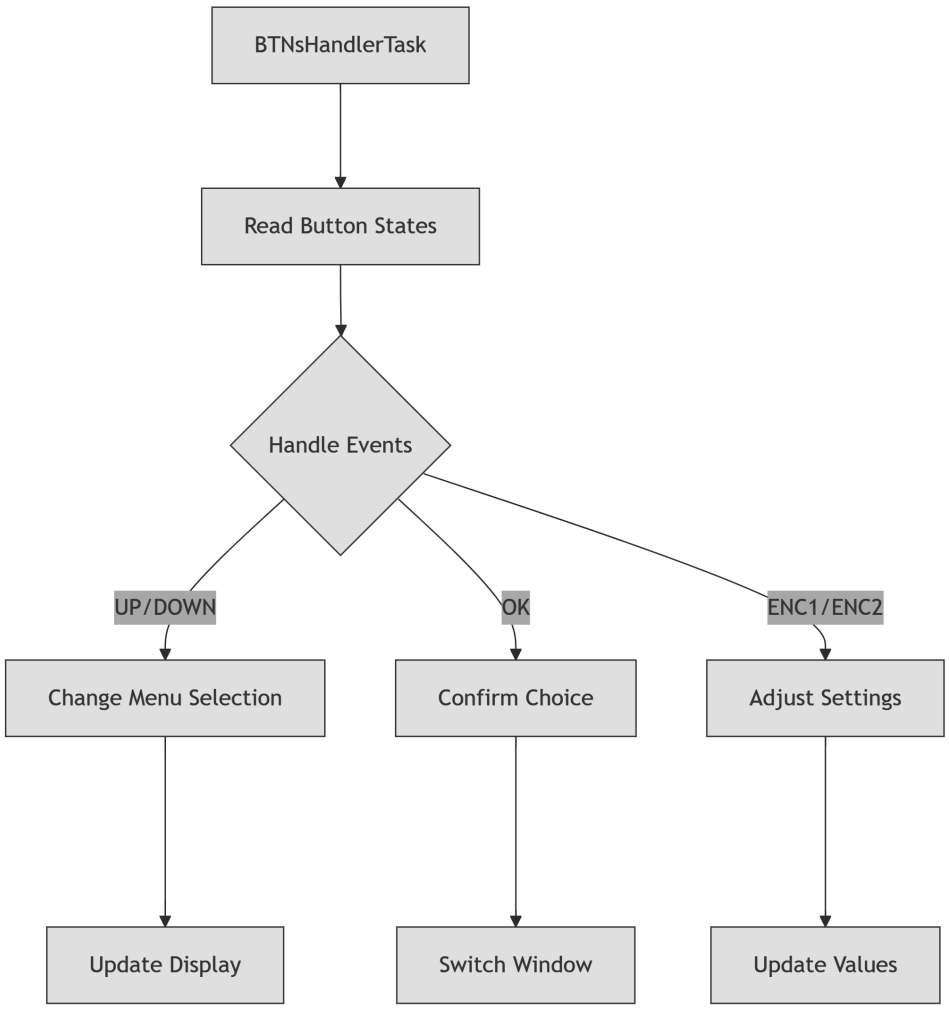


Abbildung 7.3: Ablaufdiagramm der BTNsHandlerTask

7.1.3 OutputHandlerTask

Diese Task steuert die Einstellungen für I_{sc}, U_{oc} und U_{ref}. Ausserdem wählt sie den Speicherblock aus, der die aktuell gewählte Kennlinie enthält. Zusätzlich werden zwei interne ADCs des STM32 ausgelesen, über die I_{out} und U_{out} gemessen werden.

Ein Ablaufdiagramm der OutputHandlerTask ist in Abbildung 7.4 dargestellt.

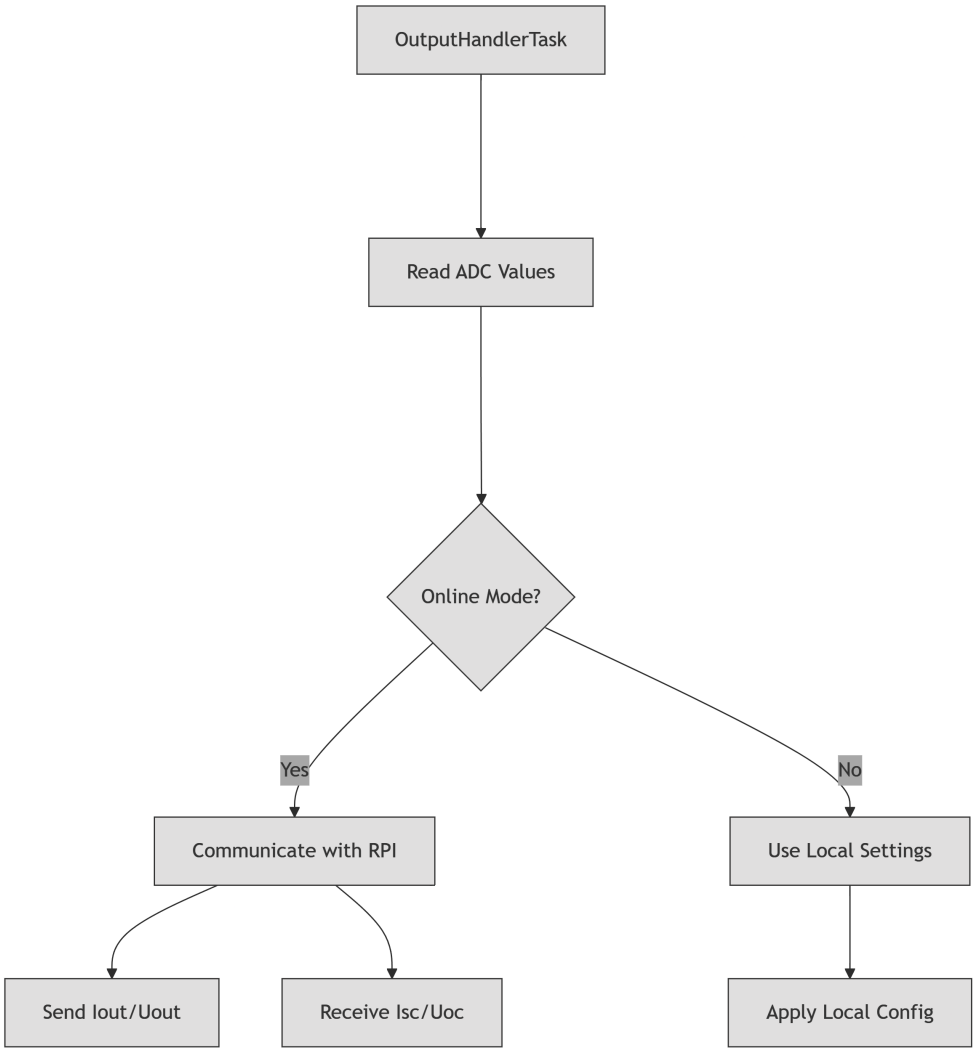


Abbildung 7.4: Ablaufdiagramm der OutputHandlerTask

7.1.4 MemoryHandlerTask

Diese Task ist für das Löschen und Beschreiben des externen Speichers [Mat3] zuständig. Während dieser Vorgänge werden Interrupts deaktiviert, um eine unterbrechungsfreie Durchführung zu gewährleisten. Nach dem Schreiben erfolgt eine Überprüfung, ob die Daten im Speicher mit denen im Schreibpuffer übereinstimmen. Anschliessend erfolgt eine Rückmeldung über die beiden LEDs auf der Steuerkarte: falls das grüne LED leuchtet war das Schreiben erfolgreich.

Ein Ablaufdiagramm der MemoryHandlerTask ist in Abbildung 7.5 dargestellt.

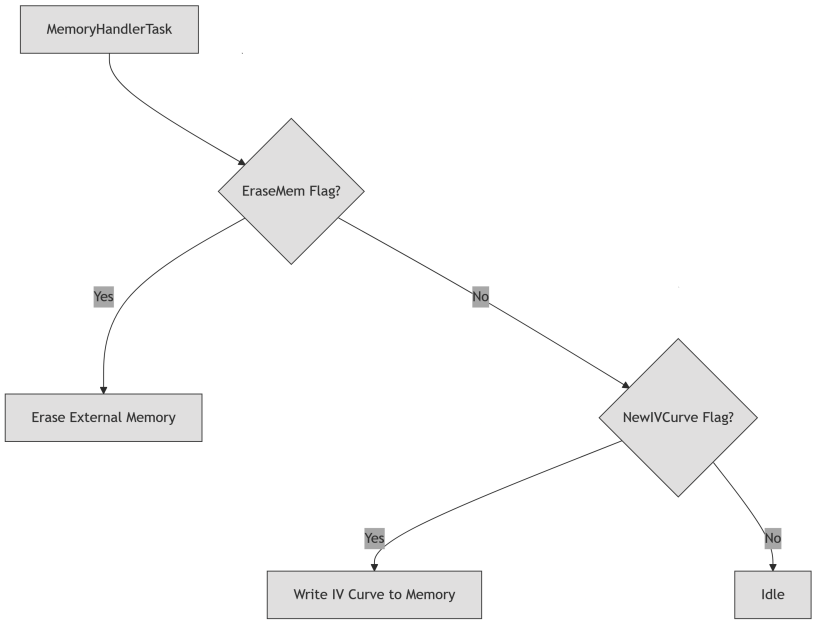


Abbildung 7.5: Ablaufdiagramm der MemoryHandlerTask

7.1.5 Kommunikation mit dem Raspberry Pi

Wie bereits erwähnt, erfolgt die SPI-Kommunikation mit dem Raspberry Pi mithilfe von Interrupts und DMA. Wenn eine neue SPI-Nachricht eintrifft, wird der entsprechende Interrupt ausgelöst, und die empfangenen Daten werden per DMA in einem Puffer abgelegt. Innerhalb des Interrupts wird die Nachricht basierend auf dem aktuellen Zustand der SPI-Zustandsmaschine interpretiert und verarbeitet. Entsprechend dem aktuellen Systemstatus und der empfangenen Nachricht wird eine Antwort generiert. Während des Betriebs überträgt das System die gemessene Spannung und den Strom an den Raspberry Pi, damit diese in der Software angezeigt und in einer Logdatei gespeichert werden können.

Das zustandsmaschine Diagramm ist in Abbildung 7.6 dargestellt.

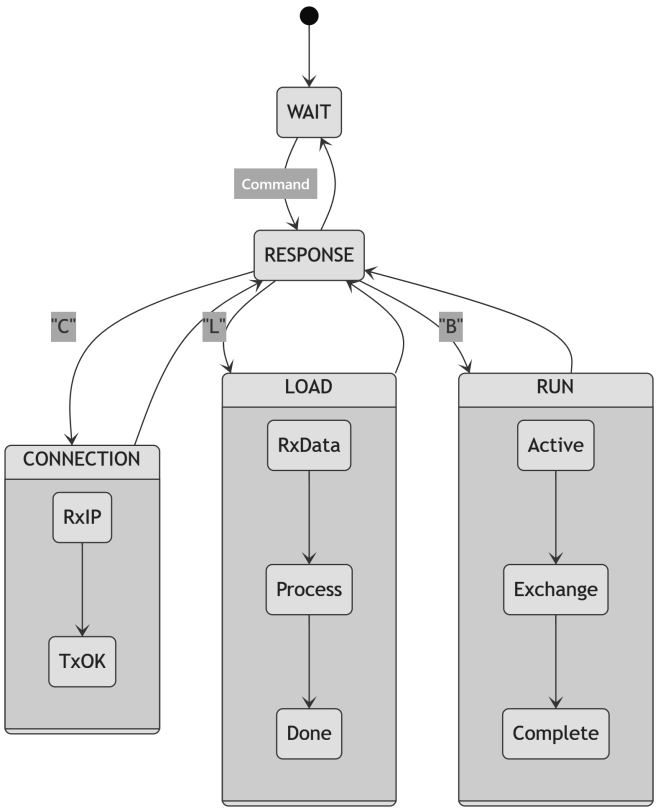


Abbildung 7.6: Zustandsmaschine der SPI-Kommunikation

Weitere Informationen zur SPI-Kommunikation und zu den verwendeten Protokollen sind im Kapitel 3.3 zu finden.

7.2 Manuelle Bedienung des PVMS

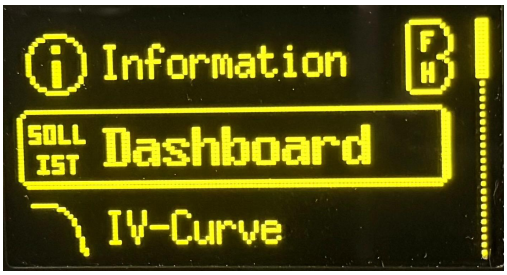
Das PVMS kann über das Control Board (Kapitel 5.3) gesteuert werden. In den folgenden Abschnitten werden die verschiedenen Bildschirme, die auf dem OLED-Display angezeigt werden, die für jeden Bildschirm verfügbaren Funktionen und die Verwendungsmethoden erläutert. Eine vollständigen Bedienungsanleitung kann im Anhang 7 eingesehen werden.

7.2.1 Navigation ins Hauptmenü

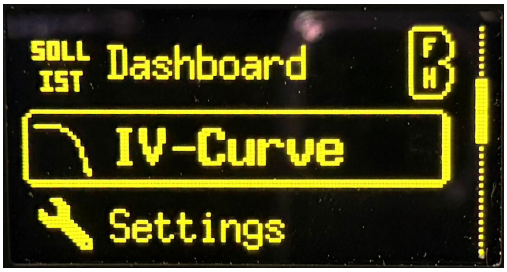
Im Hauptmenü können Sie zwischen 4 Optionen wählen:

- **Dashboard:** Zeigt die aktuellen Messwerte, I_{sc} und U_{oc} an.
- **IV-Curve:** Erlaubt die gewunschte Kennlinie auszuwählen.
- **Settings:** Hier können I_{sc} und U_{oc} angepasst werden.
- **Information:** Zeigt die Firmware-Version und das IP des Raspberry Pi an.

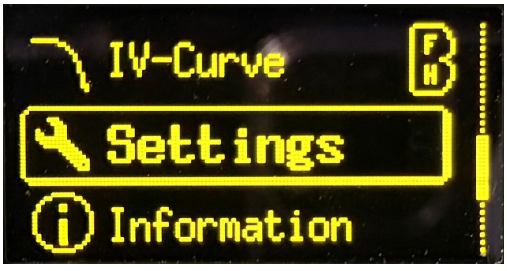
Das ausgewählte Element des PVMS ist immer in der Mitte des Displays sichtbar und ist umrandet. In Abbildung 7.7 sind die 4 möglichen Bildschirme des Menüs zu sehen.



(a) Dashboard ausgewählt



(b) IV-Curve ausgewählt



(c) Setting ausgewählt

(d) Information ausgewählt

Abbildung 7.7: Alle mögliches aussichten des PVMS Menüs

Die Navigation im Menü erfolgt über die Schaltflächen *Up* und *Down*, mit denen man sich zwischen den verschiedenen Elementen bewegen kann. Die Taste *Ok* wird gedrückt, um eine Option zu öffnen (Abbildung 5.27) .

7.2.2 Dashboard

In diesem PVMS-Menü können Sie die aktuelle Ausgangsspannung und den Strom sowie I_{sc} und U_{oc} sehen. Dieses Fenster wird alle 250 ms aktualisiert, und um ein Flackern zu vermeiden, wird nur der alte Wert für die Ausgangsspannung und den Strom gelöscht und der neue geschrieben.

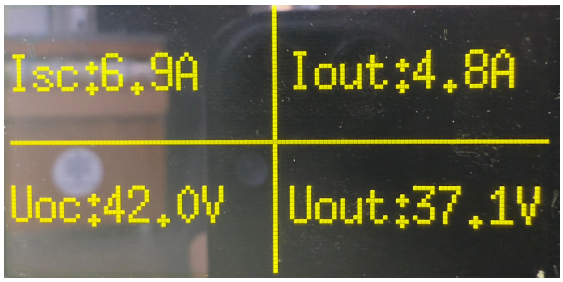


Abbildung 7.8: Fenster *Dashboard* des PVMS

In diesem Menü gibt es nichts zu ändern, daher wird nur die Taste *Ok* (Abbildung 5.27) gelesen und verwendet, um zum Menü zurückzukehren.

7.2.3 IV-Curve

In diesem Fenster können Sie die gewünschte IV-Kurve auswählen. Die Auswahl erfolgt über die Impulse *Up* und *Down*. Erst wenn Sie dieses Fenster über die Schaltfläche *Ok* (Abbildung 5.27) verlassen, wird die ausgewählte IV-Kurve tatsächlich aktiviert.

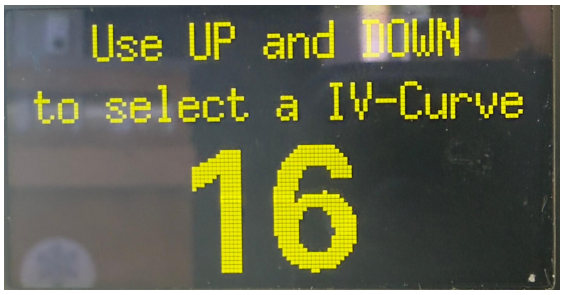


Abbildung 7.9: Fenster *IV-Curve* des PVMS

7.2.4 Settings

Mit *Einstellung* können der Strom I_{sc} und die Spannung U_{oc} geändert werden. Für jeden der beiden Werte gibt es einen Encoder, mit dem der Wert geändert werden kann. Darüber hinaus kann die Schrittweite jedes Encoderschritts durch Drücken der im Encoder integrierten Taste geändert werden und ist im Text durch Unterstreichung gekennzeichnet. Auch hier wird der tatsächliche Wert eingestellt, wenn Sie das Menü durch Drücken der Taste *Ok* (Abbildung 5.27) verlassen.

Abbildung 7.10: Fenster *Settings* des PVMS

7.2.5 Information

In diesem Fenster können Sie die IP-Adresse des Raspberry Pi sehen. Es wird auch ein QR-Code generiert, der einen schnellen Zugriff auf das Programm auf dem Raspberry Pi ermöglicht, wenn Sie sich im Online-Modus befinden. Dieses Fenster ist auch das einzige, das im Onlinemodus angezeigt wird, und alle Schaltflächen werden ignoriert (ausser denen, die über Interrupts gesteuert werden).

Abbildung 7.11: Fenster *Information* des PVMS

7.2.6 Output

Dieser Taster wird über einen Interrupt gesteuert und ermöglicht jederzeit das Deaktivieren des Ausgangs. Wenn der Ausgang aktiv ist, leuchtet die LED über dem Taster. (Abbildung 5.27)

7.2.7 Online-Modus

Auch dieser Knopf wird über einen Interrupt gesteuert und ermöglicht es jederzeit, in den Online-Modus zu wechseln oder ihn zu verlassen, Auch in diesem Fall ist eine LED vorhanden, die den aktuellen Status anzeigt. Wenn die LED aus ist, befindet man sich im Standalone-Modus. Wenn sie hingegen blinkt, ist man im Online-Modus, aber der Raspberry ist nicht verbunden (Kapitel 5.1.4). Wenn die LED leuchtet, ist man online und verbunden. also den Modus, in dem das System aus der Ferne gesteuert wird. (Abbildung 5.27)

8 Messungen

In diesem Kapitel werden die Messungen beschrieben, die im Rahmen dieses Projekts mit dem Photovoltaik-Modul-Simulator (PVMS) durchgeführt wurden. Grundlage der Untersuchungen ist eine normierte Kennlinie mit einem Fill-Factor (FF) von 75 %, welche in den Speicher des Systems geschrieben und anschliessend analysiert wurde. Dabei wurden sowohl statische als auch dynamische Messungen durchgeführt, um insbesondere das Ansprechverhalten der Stromquelle zu evaluieren.

8.1 Analyse der Sprungantwort

Im ersten Schritt wurde das Ansprechverhalten der Stromquelle mittels einer Sprungantwort untersucht. Ziel dieser Messung war es, zu ermitteln, wie das System auf eine plötzliche Stromänderung reagiert. Die Messungen wurden mit einem Oszilloskop des Typs *Rohde & Schwarz RTM3004* [Lit13] durchgeführt.

Die Testbedingungen waren wie folgt:

- Kurzschlussstrom: 10 A
- Leerlaufspannung: 40 V
- Arbeitspunkt (Strom): 8.8 A
- Arbeitspunkt (Spanung): 34.3 V

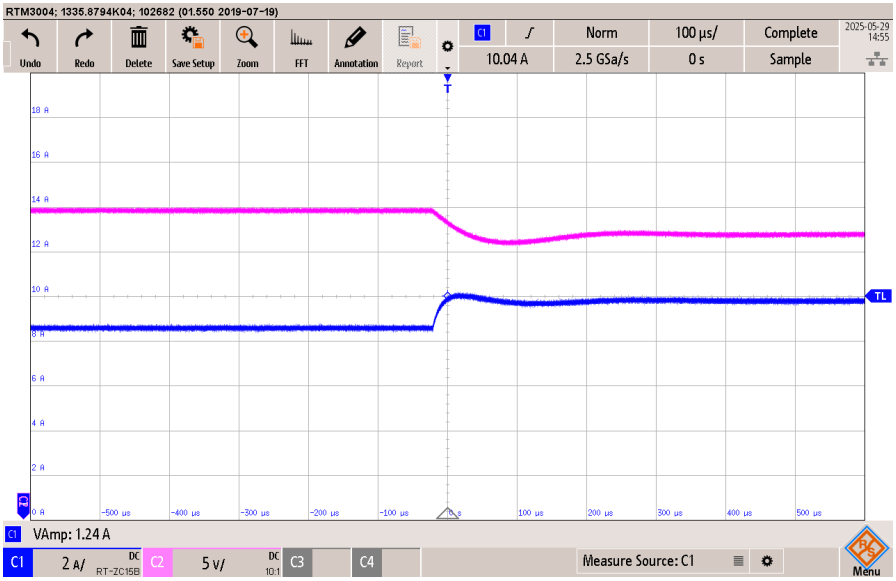


Abbildung 8.1: Sprungantwort des PVMS

Die Kennlinie wurde aus dem internen Speicher geladen. Mittels eines eigens von der Berner Fachhochschule entwickelten unipolaren Schalters wurde ein Stromsprung von 1.6 A erzeugt. Die resultierende Einschwingzeit des Systems betrug etwa 300 µs. Diese Messung wurde mit einer differentiellen Sonde gemessen.

8.2 Thermische Messungen

Im Rahmen der thermischen Messung wurde untersucht, ob der Kühlkörper in Kombination mit einer Wasserkühlung ausreichend dimensioniert ist, um die Verlustleistung der verwendeten MOSFETs zuverlässig abzuführen.

Die Testbedingungen waren wie folgt:

- Kurzschlussstrom: 19 A
- Leerlaufspannung: 40 V
- Arbeitspunkt (Strom): 17 A
- Arbeitspunkt (Spanung): 36 V

Diese Betriebsbedingung wurde über einen Zeitraum von 30 min gehalten. Anschliessend wurde mit einer Infrarotkamera [Lit14] die Oberflächentemperatur der Leistungselektronik erfasst.

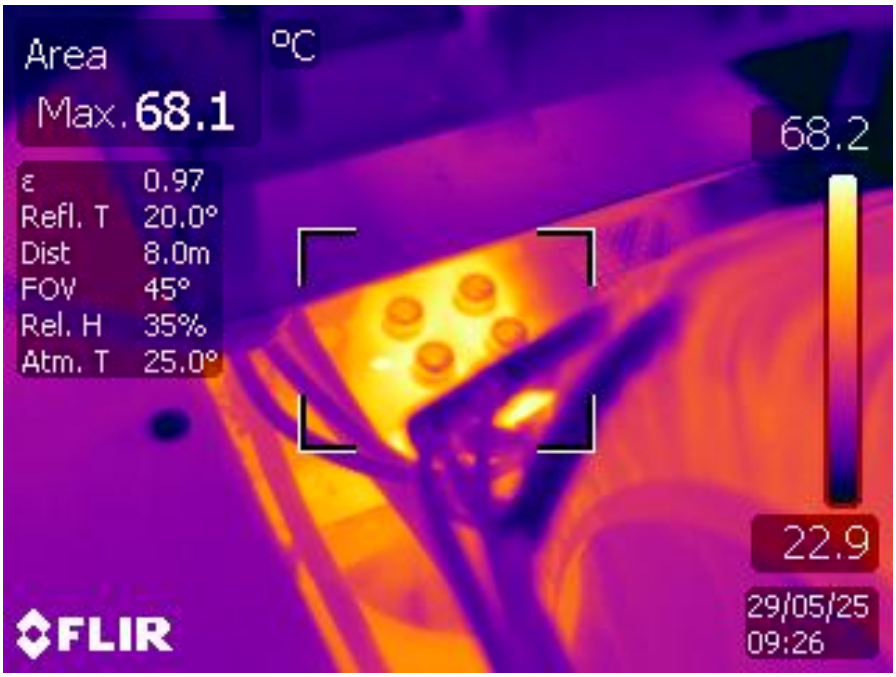


Abbildung 8.2: Infrarotmessung der Leistungstransistoren

Die Ergebnisse zeigen, dass die MOSFETs zwar eine deutliche Erwärmung aufweisen, jedoch ihre thermischen Grenzwerte nicht überschreiten.

8.3 Netzgerät (60 V) – Phasenverschiebungsmessung

In dieser Messreihe wurde das verwendete $60\,\mathrm{V}$-Netzgerät hinsichtlich seiner Phasenlage überprüft. Eine grosse Phasenverschiebung auf der Versorgungsebene kann zu einem instabilen Regelverhalten der Stromquelle führen.

Für diese Analyse wurde eine sinusförmige Spannung mit $10\,\mathrm{V_{RMS}}$ und $6\,\mathrm{V_{pp}}$ bei $500\,\mathrm{Hz}$ am Eingang des Netzteils angelegt. Der Ausgangsstrom wurde gemessen, um die Phasenverschiebung zwischen Strom und Spannung zu bestimmen.

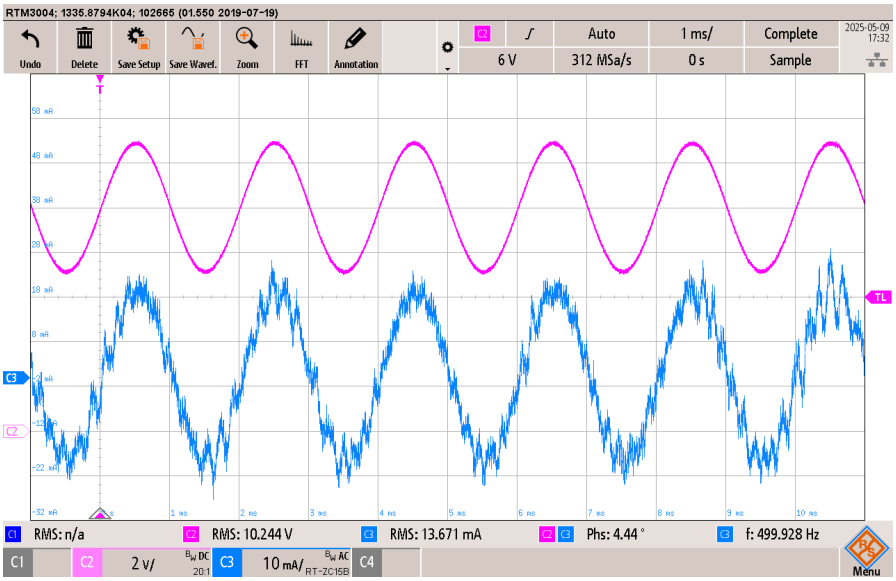


Abbildung 8.3: Phasenverschiebung des $60\,\mathrm{V}$-Netzgeräts

Die Messung ergab eine Phasenverschiebung von $4.44°$, was innerhalb eines akzeptablen Bereichs liegt.

8.4 Statische Kennlinienmessung

Zur Beurteilung der Stabilität des PVMS bei statischer Belastung wurde eine oszillierende Last verwendet. Ziel war es, die maximale Frequenz der Laständerung zu bestimmen, bei der das System noch stabil bleibt.

Die Testbedingungen waren wie folgt:

- Kurzschlussstrom: $10\,\mathrm{A}$
- Leerlaufspannung: $40\,\mathrm{V}$
- Lastamplitude: $2.5\,\mathrm{V_{pp}}$
- Last: elektronische Stromsenke *DM15000/PAS*[Lit15]

Die Messungen wurden mit dem Oszilloskop *Rohde & Schwarz RTM3004* [Lit13], der Stromsonde *RT-ZC15B*[Lit16] sowie der differentiellen Tastkopf *TA057* [Lit17] durchgeführt.

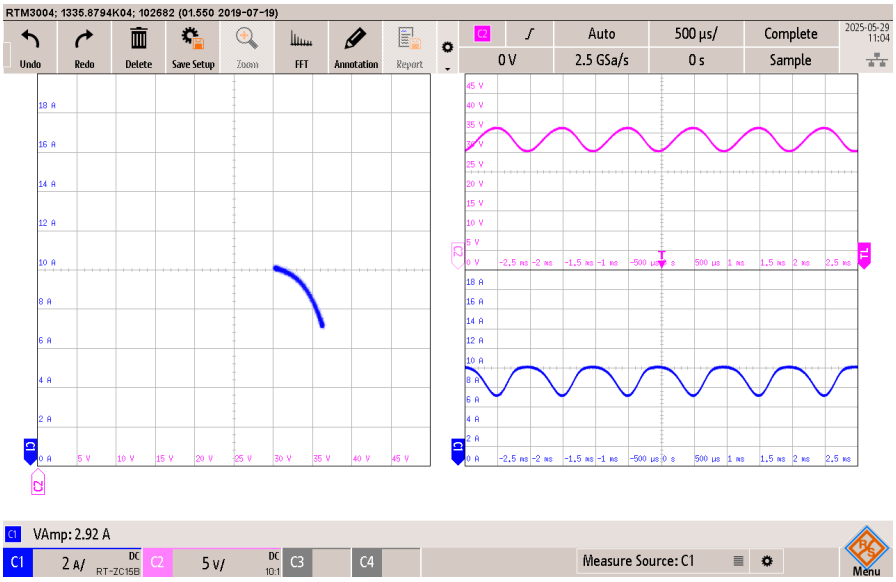


Abbildung 8.4: Lastfrequenz 1 kHz – PVMS stabil

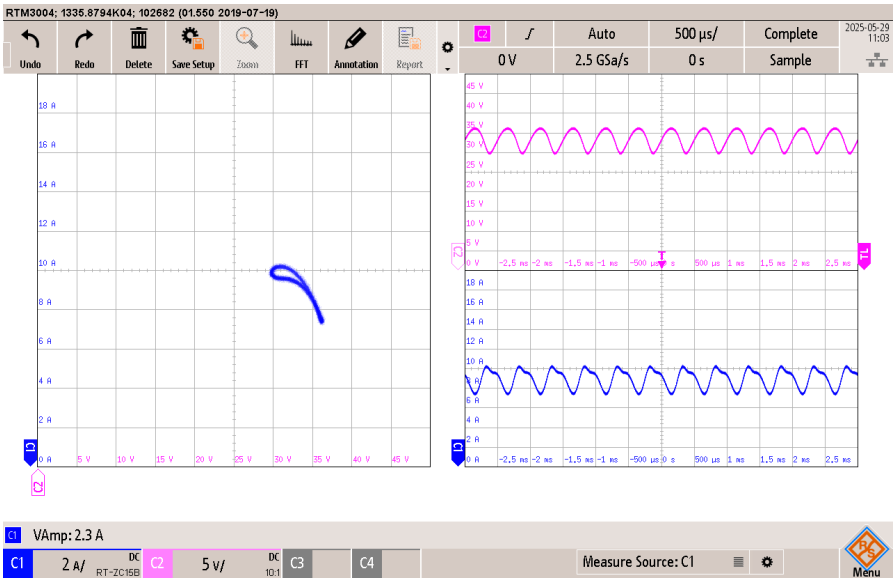


Abbildung 8.5: Lastfrequenz 2 kHz – PVMS instabil

Bei einer Lastfrequenz von 1 kHz arbeitet das PVMS stabil. Wird die Frequenz auf 2 kHz erhöht, beginnt das System instabil zu werden, was auf eine begrenzte Regelbandbreite hinweist.

8.5 Dynamische Kennlinienmessung

Zur Untersuchung des dynamischen Verhaltens wurde die Reaktion des Systems auf eine sich verändernde Bestrahlungsstärke (simuliert) analysiert. Hierfür wurde ein Strahlungsprofil im Modus *Dynamic Irradiation* 6 geladen. Das Profil sah eine Änderung der Einstrahlung von $1100\,\mathrm{W\,m^{-2}}$ auf $400\,\mathrm{W\,m^{-2}}$ über $4\,\mathrm{s}$ vor.

- I_{SC} = 10 A
- U_{OC} = 10 V
- Lastwiderstand: $4\,\Omega$ → $1\,\Omega$ umgeschaltet

Die Ausgangsspannung wurde mit einer differentiellen Sonde [Lit17] gemessen.

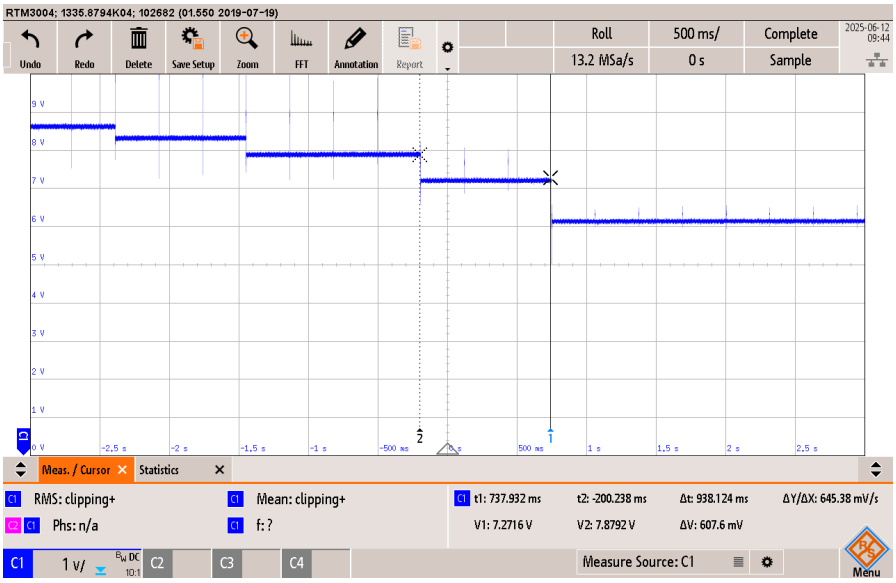


Abbildung 8.6: Dynamisches Verhalten des PVMS unter variablem Bestrahlungsprofil

Die Messung zeigt, dass das Bestrahlungsprofil nicht linear abgefahren wird. Es treten Sprünge in der Ausgangsspannung auf, deren Dauer ebenfalls variiert. Dies ist auf die nicht deterministische Aktualisierung der Steuerparameter zurückzuführen: Während der Mikrocontroller die Sollwerte alle $250\,\mathrm{ms}$ aktualisiert, sendet die Software nur alle $1\,\mathrm{s}$ neue Werte. Diese asynchrone Kommunikation führt zu nichtlinearen Übergängen im Messprofil.

9 Schlusswort

In diesem Kapitel werden die im Rahmen dieser Arbeit umgesetzten sowie die zukünftig geplanten Funktionserweiterungen beschrieben.

9.1 Erreichte Funktionen

9.1.1 Hardware

Die folgenden Hardwarefunktionen wurden erfolgreich umgesetzt:

- **Hohe Ausgangsströme:** Die Stromquelle kann dauerhaft Ströme bis zu 20 A liefern. Das System bleibt dabei bis zu einer Frequenz von 1 kHz stabil.
- **Grosse Kennlinienspeicherung:** Auf der Digitalkarte lassen sich bis zu 8192 Kennlinien speichern.
- **Analoge Speicheradressierung:** Mithilfe des integrierten DA-Wandlers kann der Speicher analog beschrieben werden. Jeder Ausgangswert des Wandlers entspricht einer spezifischen Speicheradresse.

9.1.2 Firmware

Die Firmware des STM32-Mikrocontrollers bietet umfangreiche Funktionen für den autonomen Betrieb:

- **Vollständige Systembedienung:** Sämtliche Funktionen des Systems lassen sich direkt über die Firmware steuern, unabhängig von der externen Software.
- **Kennlinienverwaltung:** Kennlinien aus dem internen Speicher können gezielt ausgewählt werden. Zentrale Betriebsparameter wie Kurzschlussstrom (I_{SC}) und Leerlaufspannung (U_{OC}) sind individuell einstellbar.
- **Ausgangssteuerung:** Der Ausgang des PVMS-Systems kann direkt am Gerät aktiviert oder deaktiviert werden.
- **Bidirektionale Kommunikation im Online-Modus:** Im Online-Modus erfolgt eine Echtzeitkommunikation zwischen Firmware und Software. Dies ermöglicht die gleichzeitige Übertragung von Messwerten und Parametern zur Visualisierung und Überwachung.
- **Lokale Bedienung über Bedienelemente:** Über Drehgeber, Taster und ein integriertes Display kann durch das Menü navigiert und das System lokal gesteuert werden.
- **Speicherung übertragener Kennlinien:** Über die Software können Kennlinien archiviert werden, um sie später erneut zu laden oder auszuwerten.

9.1.3 Software

Die Software verfügt über eine modulare Struktur mit klar abgegrenzten Funktionsblöcken. Ihr Hauptzweck ist die Bereitstellung einer intuitiven und benutzerfreundlichen Bedienoberfläche zur Steuerung des Systems.

Derzeit sind folgende Funktionen implementiert:

- **Visualisierung von Kennlinien:** Bereits vorhandene Kennlinien können eingelesen und grafisch dargestellt werden.
- **Parametersteuerung:** Die Parameter Kurzschlussstrom (I_{sc}) und Leerlaufspannung (U_{oc}) sind individuell einstellbar.
- **Simulation statischer Kennlinien:** Es ist möglich, einzelne statische Kennlinien zu simulieren.
- **Simulation dynamischer Kennlinien mit Profilen:** Vordefinierte Einstrahlungsprofile können verwendet werden, um realitätsnahe dynamische Kennlinien zu simulieren.
- **Dynamische Verschattung:** Zusätzlich ist die Simulation dynamischer Kennlinien mit variabler Verschattung implementiert.
- **Projektmanagement:** Projekte können erstellt, geöffnet und gespeichert werden, um Arbeitsstände effizient zu verwalten.
- **Live-Aktualisierung:** Änderungen an Parametern führen automatisch zu einer Aktualisierung der grafischen Darstellung.
- **Übertragung an den STM32:** Gespeicherte Kennlinien können direkt auf den Mikrocontroller übertragen werden.
- **Darstellung und Archivierung von Messdaten:** Die vom STM32 zurückgelieferten Messwerte werden visualisiert und können optional gespeichert werden.
- **Fernzugriff über Webadresse:** Der Zugriff auf die Software kann über eine Webadresse erfolgen, beispielsweise per VNC-Verbindung.

Diese Funktionen bilden den aktuellen Entwicklungsstand der Software. Weitere technische Details sowie eine ausführliche Beschreibung der Implementierung finden sich im Kapitel 6.

9.2 Geplante Erweiterungen

9.2.1 Hardware

Um die Hardware robuster, kompakter und EMV-optimierter zu gestalten, sind folgende Anpassungen geplant:

- **Neues Leiterplattenlayout:** Eine vollständige Überarbeitung des Leiterplattenlayouts ist vorgesehen. Alle im Rahmen dieses Projekts vorgenommenen Änderungen werden konsolidiert und in das neue Design integriert.
- **Seriewiderstände zur Adressierungsoptimierung:** Im Speicherbereich werden Seriewiderstände integriert, um die Ansprechzeit zu verbessern und Überschwingen zu reduzieren. Dies trägt zu einer stabileren Signalausgabe bei.
- **Verbesserte EMV durch Entkopplungsmassnahmen:** Es wird geprüft, ob zusätzliche Entkopplungselemente die elektromagnetische Verträglichkeit (EMV) weiter optimieren können.
- **Reduzierter Formfaktor:** Eine Verkleinerung des Formfaktors ist angedacht, um die Einsetzbarkeit in kompakteren Systemen zu ermöglichen.

9.2.2 Firmware

Zur Verbesserung der Datenverarbeitung, Systemreaktivität und Benutzerinteraktion sind folgende Erweiterungen vorgesehen:

- **Paketbasierte Datenübertragung:** Anstelle einzelner, synchroner Datenübertragungen werden Messwerte gesammelt, strukturiert und paketweise übermittelt. Dies verbessert die zeitliche Deterministik und erhöht die Robustheit der Kommunikation.
- **Erweiterung des SPI-Protokolls:** Das bestehende SPI-Protokoll wird angepasst, um grössere Datenmengen effizient zu übertragen. Zusätzlich wird die Übertragung von Einstrahlungsprofilen vom Host zum Mikrocontroller ermöglicht.
- **Kontinuierliche Kennlinienverfolgung:** Die Firmware ermöglicht künftig das Nachfahren kontinuierlicher Kennlinienverläufe, anstatt wie bisher nur diskrete Punkte zu nutzen. Damit lassen sich dynamische Verschattungen feiner und realistischer abbilden.
- **Echtzeitvisualisierung im Online-Modus:** Der Online-Modus wird erweitert, sodass der aktuelle Zustand der Ausgangswerte in Echtzeit visualisiert werden kann. Dies erleichtert die Überwachung und die Fehlersuche während des Betriebs.
- **Internes Puffersystem:** Ein interner Zwischenspeicher soll kurzzeitige Lastspitzen beim Datentransfer abfangen und eine gleichmässigere Datenbereitstellung sicherstellen.

9.2.3 Software

Um die Software benutzerfreundlicher, leistungsfähiger und besser wartbar zu gestalten, sind folgende Erweiterungen geplant:

- **Webbasierte Nutzung:** Die Software soll künftig über WebAssembly zugänglich sein, wodurch eine plattformunabhängige Nutzung direkt im Browser ohne lokale Installation ermöglicht wird.
- **Projektimport und -export:** Die Möglichkeit, Projekte zu exportieren und zu importieren, wird implementiert. Dadurch können Projektkonfigurationen einfach zwischen verschiedenen Systemen übertragen oder versioniert archiviert werden.
- **Optimierung der Ordnerstruktur:** Die Darstellung der internen Ordner- und Parameterstruktur wird überarbeitet. Ziel ist eine übersichtlichere und intuitivere Benutzeroberfläche, auf der alle relevanten Inhalte zentral dargestellt werden.
- **Verbesserte zeitliche Auflösung bei dynamischer Verschattung:** Die zeitliche Auflösung bei Verschattungsmessungen soll von derzeit 0,5 s auf bis zu 0,1 s erhöht werden. Dies ermöglicht eine präzisere Erfassung schneller Lichtveränderungen.
- **Sanfte Übergänge bei Verschattungsverläufen:** Übergänge zwischen verschiedenen Verschattungsstufen sollen künftig geglättet werden, um realistischere und kontinuierlichere Kennlinienverläufe darzustellen, insbesondere bei dynamischen Simulationen.
- **Erweiterte Logging-Funktionalität:** Die Software soll künftig über eine umfangreiche Protokollierungsfunktion verfügen. Dazu zählt insbesondere die Möglichkeit, Messdaten als Datenpackete abzuspeichern und diese anschliessend zu exportieren.
- **Kontrollpanel für die SPI-Kommunikation:** Die bestehende Steuerung der SPI-Kommunikation erfolgt derzeit über ein technisches Kontrollpanel, das noch nicht benutzerfreundlich gestaltet ist. Ziel ist ein intuitives, grafisches Interface für die SPI-Kommunikation.
- **Softwaregesteuertes Ausschalten des Ausgangs:** Die Möglichkeit, den Ausgang des PVMS-Systems softwareseitig zu deaktivieren, soll ergänzt werden. Dies erhöht die Sicherheit und Kontrolle im Betrieb.
- **Performanceanalyse und Geschwindigkeitsoptimierung:** In einem weiteren Entwicklungsschritt soll die Performance der Software analysiert werden. Dabei wird geprüft, ob durch schnellere Hardware oder durch Optimierungen im SPI-Datenpfad die Reaktionszeit verbessert werden kann.

Diese Erweiterungen bilden die nächsten geplanten Entwicklungsschritte auf Softwareebene. In Kombination mit den geplanten Hardware- und Firmwareanpassungen sollen sie die Gesamtperformance des Systems erhöhen.

9.3 Weiteres Vorgehen

Das Projekt wird an dieser Stelle nicht beendet, sondern aktiv weiterentwickelt. Es wird an das PV-Labor übergeben, wo die zukünftige Entwicklung auf Basis der zuvor definierten Erweiterungen und Verbesserungen fortgeführt wird.

Die Weiterentwicklung erfolgt im PV-Labor nach dem Scrum-Modell, bei dem Anforderungen in sogenannten Sprints umgesetzt werden. Dieses iterative Vorgehen ermöglicht eine gezielte Planung, Umsetzung und kontinuierliche Optimierung einzelner Systemkomponenten. Auf diese Weise können Änderungen strukturiert vorgenommen und innerhalb klar definierter Zeiträume realisiert werden.

9.4 Erworbene Kenntnisse

Im Verlauf des Projekts konnten vielfältige technische und methodische Kenntnisse erworben und vertieft werden:

- **Ansteuerung elektronischer Stromquellen:** Es wurde ein fundiertes Verständnis für den hardwaretechnischen Aufbau und die elektrische Ansteuerung einer programmierbaren Stromquelle erarbeitet.
- **STM32-Architektur und -Programmierung:** Die Arbeit mit Mikrocontrollern der STM32-Serie ermöglichte eine praxisnahe Auseinandersetzung mit deren Peripheriefunktionen wie ADC (Analog-Digital-Wandler), FMC (Flexible Memory Controller), RTOS (Echtzeitbetriebssystem) sowie Inter-Core-Kommunikation. Diese Funktionen konnten im Projekt angewendet oder konzeptionell nachvollzogen werden.
- **Softwareentwicklung mit Python und PySide6:** Die Entwicklung einer grafischen Benutzeroberfläche vermittelte praktische Erfahrungen mit dem Python-Framework PySide6. Dabei wurden sowohl grundlegende als auch fortgeschrittene Konzepte der GUI-Programmierung erlernt.
- **Fernzugriff mit VNC:** Die Einrichtung und Integration eines VNC-Servers auf dem Raspberry Pi wurde erfolgreich umgesetzt. Durch ein automatisch startendes Skript konnte der Fernzugriff bereits beim Systemstart des Raspberry Pi sichergestellt werden.
- **Systemintegration:** Das Zusammenspiel zwischen Hardware, Firmware und Software wurde im Rahmen dieses Projekts systematisch erprobt, optimiert und dokumentiert.

Insgesamt war das Projekt ein Erfolg. Es bietet eine solide technische Grundlage sowie eine durchdachte Architektur, auf der zukünftige Weiterentwicklungen effizient aufbau-

en können. Die dokumentierten Ergebnisse und der aktuelle Projektstand ermöglichen es nachfolgenden Entwicklerteams, nahtlos anzuknüpfen und das System zielgerichtet weiter auszubauen.

Literatur

[Lit1] Wikipedia, “Ethernet.” [Online]. Available: https://de.wikipedia.org/wiki/Ethernet

[Lit2] ——, “Can.” [Online]. Available: https://de.wikipedia.org/wiki/Controller_Area_Network

[Lit3] ——, “Usb.” [Online]. Available: https://de.wikipedia.org/wiki/Universal_Serial_Bus

[Lit4] ——, “Serial peripheral interface.” [Online]. Available: https://de.wikipedia.org/wiki/Serial_Peripheral_Interface

[Lit5] ——, “I2c.” [Online]. Available: https://de.wikipedia.org/wiki/I%C2%B2C

[Lit6] Altium, “Serielle kommunikationsprotokolle - teil zwei: Uart.” [Online]. Available: https://resources.altium.com/de/p/serial-communications-protocols-part-two-uart

[Lit7] P. S. Foundation, “spidev 3.7.” [Online]. Available: https://pypi.org/project/spidev/

[Lit8] STMicroelectronics, “Stm32h745/755 and stm32h747/757 lines inter-processor communications.” [Online]. Available: https://www.st.com/resource/en/application_note/an5617-stm32h745755-and-stm32h747757-lines-interprocessor-communications-stmicroelectronics.pdf

[Lit9] ——, “Fmc: Flexible memory controller.” [Online]. Available: https://wiki.st.com/stm32mpu/wiki/FMC_internal_peripheral

[Lit10] M. Fitzpatrick, “The complete pyside6 tutorial — create gui applications with python.” [Online]. Available: https://www.pythonguis.com/pyside6-tutorial/

[Lit11] Q. Group, “Qt for webassembly.” [Online]. Available: https://doc.qt.io/qt-6/wasm.html

[Lit12] fiecs5, “R&srtm3000 oszilloskop.” [Online]. Available: https://gitlab.ti.bfh.ch/fiecs5/pvms_bachelor_thesis_2025

[Lit13] R. und Schwarz, “Pvms_bachelor_thesis_2025.” [Online]. Available: https://www.rohde-schwarz.com/ch/produkte/messtechnik/oszilloskope/rs-rtm3000-oszilloskop_63493-427459.html

[Lit14] FLIR, “Flir t335 infrarotkamera.” [Online]. Available: https://www.flir-infrarotkameras.de/FLIR-T335-Infarot-Waermebildkamera

[Lit15] S. Spies, “Aps series of 4-quadrant amplifiers.” [Online]. Available: https://spitzenberger.de/download.ashx?Weblink=1107

[Lit16] R. und Schwarz, “R&srt-zc15b - bedienhandbuch.” [Online]. Available: https://www.rohde-schwarz.com/de/handbuch/r-s-rt-zc15b-bedienhandbuch-handbuecher_78701-161091.html

[Lit17] P. T. Ltd., “Ta057 25 mhz 1400 v differential oscilloscope probe 20:1/200:1.” [Online]. Available: https://www.picotech.com/accessories/active-differential-high-voltage/25-mhz-1400-v-differential-probe

[Lit18] capaciteam, “C++ vs. python: All you need to know.” [Online]. Available: https://capaciteam.com/c-plus-plus-vs-python/

[Lit19] Q. Group, “The future of digital experiences.” [Online]. Available: https://www.qt.io/

[Lit20] mdn, “Webassembly.” [Online]. Available: https://webassembly.org/

[Lit21] Q. Group, “Qt creator.” [Online]. Available: https://www.qt.io/product/development-tools

[Lit22] R. Pi, “Raspberry pi os.” [Online]. Available: https://www.raspberrypi.com/software/

[Lit23] MuyePan, “Crosscompileqtforrpi.” [Online]. Available: https://github.com/MuyePan/CrossCompileQtForRpi

[Lit24] N. B. Simon Fiechter, “pvms_bachelor_thesis_2025.” [Online]. Available: https://github.com/SimonFiechter/pvms

[Lit25] STMicroelectronics, “Stm32h7 internal adc: Internal 16 bit adc.” [Online]. Available: https://www.st.com/resource/en/application_note/an5354-getting-started-with-the-stm32h7-series-mcu-16bit-adc-stmicroelectronics.pdf

[Lit26] RealVNC Ltd., “What is vnc?” [Online]. Available: https://www.realvnc.com/en/connect/what-is-vnc/

[Lit27] I. T. AG, “Datasheet s29glxxxs.” [Online]. Available: https://www.infineon.com/dgdl/Infineon-S29GL01GS_S29GL512S_S29GL256S_S29GL128S_128_Mb_256_Mb_512_Mb_1_Gb_GL-S_MIRRORBIT_TM_Flash_Parallel_3-DataSheet-v22_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0ed07ac14bd5

[Lit28] J. S. AG, “Pid-regler – definition, funktion, einstellung und einsatz.” [Online]. Available: https://www.jumo.ch/web/services/faq/controller/pid-controller

[Lit29] OpenAI, “Chatgpt (gpt-4.5).” [Online]. Available: https://chat.openai.com

Material

[Mat1] STM32, “Stm32h757bit6 dual-core cortex-m7/m4 microcontroller.” [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32h757bi.html#overview

[Mat2] Raspberry, “Raspberry pi zero 2.” [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/

[Mat3] Infineon, “S29gl01gs10tfi020: 1gb parallel nor flash speicher.” [Online]. Available: https://www.mouser.ch/ProductDetail/Infineon-Technologies/S29GL01GS10TFI020?qs=r%2FU9ccuVSsBKe7MFEhuqdw%3D%3D

[Mat4] A. E. . S. Power, “Switching power supplies 60v a. 25a.” [Online]. Available: https://www.mouser.ch/ProductDetail/Advanced-Energy-SL-Power/TF1500A60K?qs=MLItCLRbWswN2DqEHgKLXQ%3D%3D

[Mat5] Recom, “Rac10-k/277.” [Online]. Available: https://www.mouser.ch/datasheet/2/468/RAC10_K_277-1661832.pdf

[Mat6] H. Manufacturing, “Power transformers power transformer, toroid, 1000 va, with 240v @ 4.17a secondary.” [Online]. Available: https://www.mouser.ch/ProductDetail/Hammond-Manufacturing/1182P240?qs=qPaGMJUC%252BStSdINUjVMocw%3D%3D

[Mat7] Xilence, “Xc971 | lq120.” [Online]. Available: https://www.xilence.net/en/water-coolers/139

[Mat8] T. Instruments, “Ads7881ipfbt: 12-bit, 4mhz, high-speed parallel interface sar adw.” [Online]. Available: https://www.mouser.ch/ProductDetail/Texas-Instruments/ADS7881IPFBT?qs=%252BvKcWiXw%252BzR%252BmKmY9%252B9hvw%3D%3D

[Mat9] A. Devices, “Ltc1666cgpbf: Digital to analog converters - dac 12-bit 50msps dac.” [Online]. Available: https://eu.mouser.com/ProductDetail/Analog-Devices/LTC1666CGPBF?qs=hVkxg5c3xu91J8bSMUcIhQ%3D%3D

[Mat10] T. Instruments, “Dacx0501.” [Online]. Available: https://www.ti.com/lit/ds/symlink/dac60501.pdf?ts=1749654121278&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fde-de%252FDAC60501

[Mat11] G.-E. G. Co. KG, “3s7be_3u series.” [Online]. Available: https://gaptec-electronic.com/datenblaetter/3S7BE_3U.pdf

[Mat12] 3peak, “Tpl810 series.” [Online]. Available: https://static.3peak.com/res/doc/ds/Datasheet_TPL810.pdf

[Mat13] M. INC., “Mic5270.” [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/mic5270.pdf

[Mat14] murata, “Crv1 series.” [Online]. Available: https://www.murata.com/products/productdata/8807029506078/kdc-crv1.pdf?1656559813000

[Mat15] L. T. Corporation, “Lt6650.” [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/6650fa.pdf

[Mat16] T. Instruments, “Lm317 3-pin adjustable regulator.” [Online]. Available: https://www.ti.com/lit/ds/symlink/lm317.pdf?ts=1749567085842&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fde-de%252FLM317

[Mat17] Bourns, “Pec11r.” [Online]. Available: https://www.bourns.com/docs/Product-Datasheets/PEC11R.pdf

[Mat18] JOY-IT, “Com 2.42 inch oled display.” [Online]. Available: https://joy-it.net/en/products/COM-OLED2.42

[Mat19] IXYS, “Ixtk200n10l2.” [Online]. Available: https://www.mouser.de/datasheet/2/240/media-3321296.pdf

Bildern Quellen

[Bil1] men at work GmbH, “Alles Über photovoltaik.” [Online]. Available: https://www.work-crew.de/photovoltaik/

[Bil2] digikey, “Introduction to rtos - solution to part 3 (task scheduling).” [Online]. Available: https://www.digikey.ch/en/maker/projects/introduction-to-rtos-solution-to-part-3-task-scheduling/8fbb9e0b0eed4279a2dd698f02ce125f

[Bil3] STMicroelectronics, “Openamp and rpmsg implementation layers.” [Online]. Available: https://www.st.com/resource/en/application_note/an5617-stm32h745755-and-stm32h747757-lines-interprocessor-communications-stmicroelectronics.pdf

[Bil4] ——, “Openamp example.” [Online]. Available: https://www.st.com/resource/en/application_note/an5617-stm32h745755-and-stm32h747757-lines-interprocessor-communications-stmicroelectronics.pdf

Selbständigkeitserklärung

Wir bestätigen, dass wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt haben. Sämtliche Textstellen, die nicht von uns stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Wir bestätige weiterhin, dass wir bei der Erstellung dieser Studienarbeit durchgehend steuernd gearbeitet haben und von einer KI erzeugte Texte bzw. Textfragmente nicht unreflektiert übernommen haben

Ort, Datum: Burgdorf, 12.06.2025

Unterschrift:

Danksagung

Eine spezieller Dank an Theo Zwahlen und Luciano Borgna für ihre Zeit und Hingabe welche sie jede Woche für uns investiert haben.
Ein Dank geht auch an Dr. Prof. Bucher für die Unterstützung und konstruktiven Rückmeldungen welche er uns gab.