Fachhochschule Westküste
Hochschule für Wirtschaft & Technik

HAW HAMBURG

Masterarbeit

Denise Wendlandt

Design und Programmierung einer mikrokontrollergesteuerten Überwachungsplatine für die Magnetstromversorgung und Hot-Swap-Steuerung in Elektronenbeschleunigern

Fachhochschule Westküste
Fachbereich Technik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Hamburg University of Applied Sciences
Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Denise Wendlandt

Design und Programmierung einer mikrokontrollergesteuerten Überwachungsplatine für die Magnetstromversorgung und Hot-Swap-Steuerung in Elektronenbeschleunigern

Masterarbeit eingereicht im Rahmen der Masterprüfung
im gemeinsamen Masterstudiengang Mikroelektronische Systeme
am Fachbereich Technik
der Fachhochschule Westküste
und
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Heike Neumann
Zweitgutachter: Prof. Dr. Sönke Appel

Eingereicht am: 20. Februar 2025

Denise Wendlandt

Thema der Arbeit

Design und Programmierung einer mikrokontrollergesteuerten Überwachungsplatine für die Magnetstromversorgung und Hot-Swap-Steuerung in Elektronenbeschleunigern

Stichworte

Mikrokontroller, Platine, Hot-Swap-Matrix, Petra IV, Eagle, galvanische Trennung, Programmierung, C, UART

Kurzzusammenfassung

In dieser Arbeit werden Hardware und Software für ein Überwachungssystem entwickelt und implementiert. Dieses System soll die Funktion der Hot-Swap-Matrix sowie die Spannungsversorgung der Magnete eines Elektronenbeschleunigers überwachen. Dabei gewährleisten die Magnete die Stabilität und Aufrechterhaltung des Elektronenstrahls und spielen eine zentrale Rolle im Betrieb des Beschleunigers.

Denise Wendlandt

Title of Thesis

Design and programming of a microcontroller-driven monitoring board for magnetic power supply and hot-swap control in electron accelerators

Keywords

Microcontroller, circuit board, hot-swap matrix, Petra IV, Eagle, galvanic isolation, Programming, C, UART

Abstract

In this thesis, hardware and software for a monitoring system are developed and implemented. This system is intended to monitor the function of the hot-swap matrix and the power supply to the magnets of an electron accelerator. The magnets ensure the stability and maintenance of the electron beam and play a central role in the operation of the accelerator.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

I^2C Inter-integrated circuit.

ADU Analog-Digital-Umsetzer.

CPU Central processing unit.

CRC Cyclic redundancy check.

DAU Digital-Analog-Umsetzer.

Desy Deutsche Elektronen-Synchrotron.

DMA Direct Memory Access.

EMI elektromagnetische Interferenz.

ESD Elektrostatische Entladungen.

GPIO General Purpose Input/Output.

IC integrierte Schaltkreis.

ISR Interrupt service routine.

LDO Low Drop Out.

LED Leuchtdiode.

MOSFET Metall-Oxid-Halbleiter-Feldeffekttransistor.

.

PWM Pulsweitenmodulation.

RAM Random access memory.

SCK Serial Clock.

SCL Serial Clock.

SDA Serial Data.

SPI Serial Peripheral Interface.

UART Universal Asynchronous Receiver Transmitter.

1 Einleitung

Das Deutsche Elektronen-Synchrotron (Desy) ist eines der weltweit führenden Unternehmen im Bereich der Beschleunigerphysik und Synchrostrahlung. Als Teil der Helmholtz-Gesellschaft liegt der Forschungsschwerpunkt im Bereich der grundlegenden Struktur der Materie und der Prozesse im Universum. Es wird unter anderem Forschung in den Fachbereichen der Physik, Chemie, Biologie und der Materialwissenschaft ermöglicht. Mithilfe von hochmodernen Teilchenbeschleunigern bietet das Desy Wissenschaftlern aus der ganzen Welt die Möglichkeit unter anderem komplexe Strukturen und ultraschnelle Prozesse auf atomarer Ebene zu untersuchen [21].
Eine zentrale Rolle bei diesem wissenschaftlichen Erfolg trägt die PETRA III Anlage. Mit ihrer außergewöhnlichen Brillanz als Speicherring für Synchrotronstrahlung hat sie über viele Jahre hinweg entscheidende Fortschritte in der Forschung ermöglicht. Doch trotz der Leistungsfähigkeit kommt die Petra III Anlage aus technischer und wissenschaftlicher Sicht immer weiter an ihre Grenzen. Um den steigenden Anforderungen innerhalb der Wissenschaft gerecht zu werden, wird das Projekt Petra IV als Nachfolger entwickelt. Hierbei soll Petra IV als hochmodernes 3D-Röntgenmikroskop eingesetzt werden, das dreidimensionale Strukturbilder vom Millimeterbereich bis hin zur atomaren Auflösung liefert. Technisch gesehen basiert Petra IV auf einem Synchrotron, in dem Elektronen nahezu auf Lichtgeschwindigkeit beschleunigt werden. Die Anlage soll die Petra III Anlage um das 500- bis 1000-fache in der Leistungsfähigkeit übertreffen. Somit bietet die neue Petra IV Anlage eine um ein Vielfaches höhere Strahlhelligkeit und Auflösung. Mit dieser Verbesserung kann die Forschung unter anderem in der Entwicklung nachhaltiger Energietechnologien, in der Untersuchung biologischer Strukturen für medizinische Fortschritte und in der Verbesserung von Werkstoffen für die Industrie verbessert werden [9].

Für die präzise Steuerung und für die Stabilisierung der Elektronen innerhalb des Rings der Petra IV Anlage werden verschiedene Magnete verwendet. Eine stabile Spannungsversorgung der Magnete ist für den Beschleunigerbetrieb essentiell, da ein Ausfall der Spannungsversorgung an den Magneten zu einem Verlust des Elektronenstrahls führen

kann. Dieser Verlust würde im ungünstigsten Fall die Wiederholung eines wissenschaftlichen Versuchs nach sich ziehen. Dies wiederum führt unter Umständen zu hohen Kosten, zu einem erheblichen gesteigerten Zeitaufwand für die Forschenden und zu einer Reduzierung des guten Rufs des Desy. Aus diesen Gründen ist der Verlust des Elektronenstrahls aufgrund einer instabilen Magnetstromversorgung unbedingt zu vermeiden.
In dieser Masterarbeit ist ein Prototyp zu entwickeln, welcher die Status der Spannungsversorgungen der Magneten und einer Hot-Swap-Matrix überprüft und die gewonnen Daten an ein übergeordnetes System weiterleitet. Mithilfe von dem übergeordneten System können anschließend Vorhersagen über einen möglichen Ausfall der Magnetstromversorgung getroffen und gegen wirkende Maßnahmen eingeleitet werden. In dieser Arbeit ist eine Platine zu entwickeln, die verschiedene Signale überwacht und mithilfe eines Mikrokontrollers auswertet und weiterleitet. Die Software zur Verarbeitung der Daten auf dem Mikrokontroller ist ebenfalls Bestandteil dieser Arbeit.

An dieser Stelle möchte ich mich herzlich beim Desy sowie bei meinen Kollegen und Kolleginnen für ihre wertvolle Unterstützung und die Möglichkeit, diese Arbeit zu verfassen, bedanken. Besonderem Dank gilt hierbei meinem Betreuer Christian Putscher.

2 Grundlagen

In diesem Kapitel werden Grundlagen vorgestellt, welche zum Verständnis dieser Arbeit vorausgesetzt werden. Hierbei wird zunächst die Hot-Swap-Matrix beschrieben. Diese Matrix wurde bereits im Vorfeld dieser Arbeit entwickelt und ist hinsichtlich ihrer Funktion mit geeigneten Mechanismen zu überwachen. Darauf erfolgt eine kurze Vorstellung des zu verwendenden Mikrokontrollers.

2.1 Die Hot-Swap-Matrix

In diesem Kapitel wird die Hot-Swap-Matrix vorgestellt. Mithilfe der Hot-Swap-Matrix wird eine verzugsfreie Umschaltung der Stromversorgung eines Magneten auf eine Reservestromversorgung ermöglicht. Dies kann einen Verlust des Elektronenstrahls verhindern, was für den Betrieb der zukünftigen Petra IV Anlage essenziell ist. Die Hot-Swap-Matrix stellt dabei eine Halbleiter-Matrix bestehend aus N-Metall-Oxid-Halbleiter-Feldeffekttransistoren (MOSFETs) dar. Die MOSFETs agieren als Schalter und ermöglichen das Umschalten zwischen den Spannungsversorgungen.
In der Abbildung 2.1 ist ein MOSFET-Schalter abgebildet.

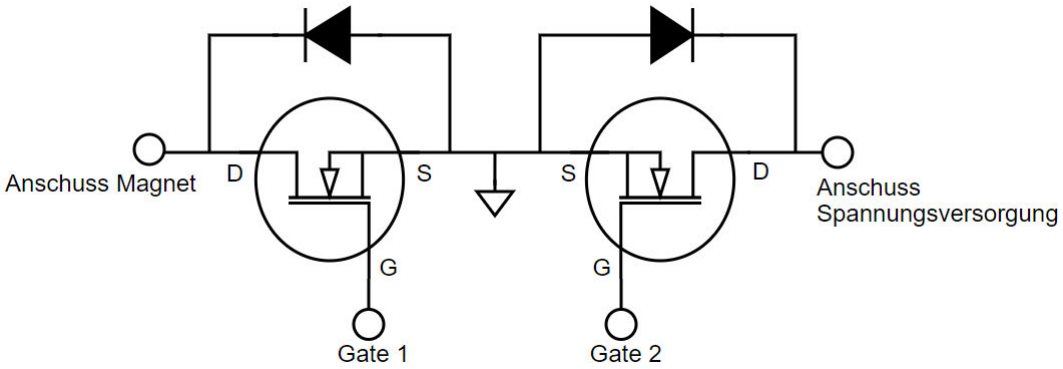


Abbildung 2.1: Darstellung eines MOSFET-Schalters

Zu erkennen sind zwei N-MOSFETs, welche Back-to-Back (Rücken an Rücken) über die Source-Anschlüsse (S) miteinander verbunden sind. Parallel zu den MOSFETs ist jeweils eine Diode geschaltet, die in der Richtung von Source nach Drain des Transistors leitfähig ist. An den Drain-Pin (D) des linken MOSFETs wird der Magnet, welche mit der Spannungsversorgung betrieben werden soll, anschlossen. An den Drain-Pin des rechten Transistors wird die zu verwendende Spannungsversorgung angelegt. Die Gate-Spannungen werden von einem Gate-Treiber bereitgestellt, wobei die Spannungen aus dem gleichen Signal erzeugt werden. Diese Schaltung wird als bidirektionaler MOSFET-Schalter verwendet. Liegt eine zu geringe Spannung an den Gates (G) an, sodass die MOSFET nicht durchschalten, wird ein Stromfluss durch die Schaltung durch die beiden Dioden verhindert. Der Magnet ist somit von der Spannungsversorgung getrennt. Wird hingegen eine ausreichend positive Gate-Spannung angelegt, leiten beide Dioden und der entsprechende Magnet kann durch die angeschlossene Spannungsversorgung betrieben werden.
Nachdem die Funktion eines einzelnen Schalters erläutert wurde, ist die Gesamtfunktion der Hot-Swap-Matrix zu erklären. Hierfür ist in der Abbildung 2.2 die Hot-Swap-Matrix, ein Magnetstromkreis und die Reservespannungsversorgung dargestellt.

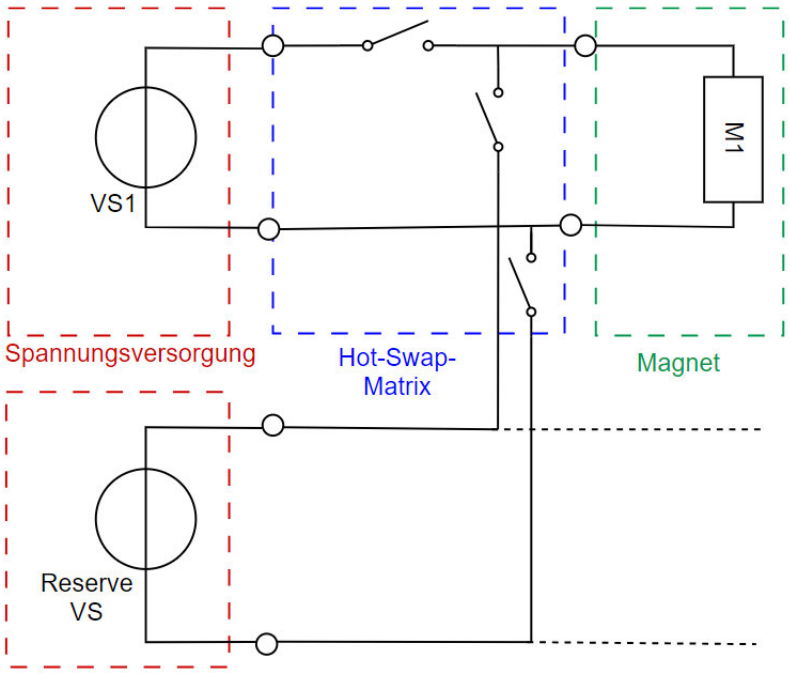


Abbildung 2.2: Darstellung der Hot-Swap-Matrix für einen Magnetstromkreis

Die Hot-Swap-Matrix (blau) stellt das Verbindungsglied zwischen der Spannungsversorgung, dem Magneten und der Reservespannungsversorgung dar. Die eingezeichneten Schalter stehen dabei für die vorgestellte MOSFET-Struktur. Im oberen Kreislauf ist ein Magnet (grün) zu erkennen, der durch die Spannungsversorgung VS1 (rot) betrieben werden kann. Sollte diese Spannungsversorgung ausfallen, ist es möglich über die Schalter in der Hot-Swap-Matrix die Reservespannungsversorgung (rot) hinzu zu schalten.
Diese Abbildung stellt eine stark vereinfachte Version dar. Im realen Betrieb sind mehr als 4000 Magnete zur Aufrechterhaltung des Betriebs geplant [22]. Dabei sollen eine große Anzahl an Magnete mit der Hot-Swap-Matrix ausgestattet werden. Hierbei wird eine Reserve-Spannungsversorgung voraussichtlich an acht verschiedene Magnetstromkreise angeschlossen werden. Im realen Betrieb sind somit mehrere Hot-Swap-Systeme zu implementieren.

2.2 Der Mikrokontroller

In diesem Kapitel wird der in diesem Projekt zu verwendende Mikrokontroller vorgestellt. Es wird der STM32F072RBT6 mit dem Board NUCLEO-F072RB implementiert. In der Abbildung 2.3 ist das Board dargestellt.

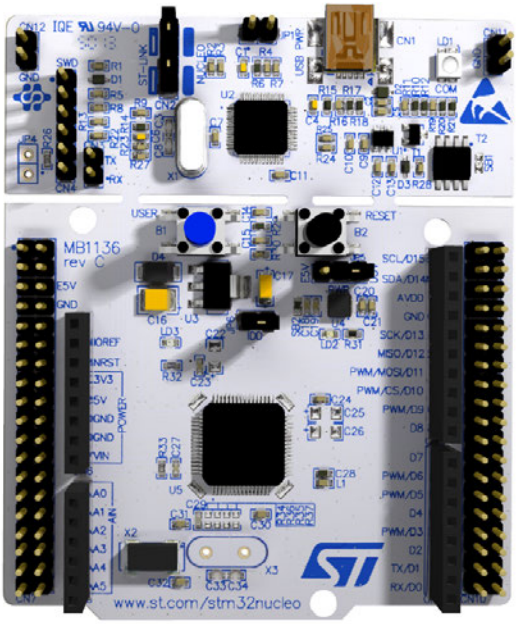


Abbildung 2.3: STM32 Nucleo-64 board

In der Tabelle 2.1 werden einige Eckdaten des Mikrokontrollers aufgeführt. Die Daten sind dem Datenblatt des Mikrokontrollers, als auch dem User Manual des Boards entnommen. Beide Quellen sind in dem digitalen Anhang vorhanden.

Tabelle 2.1: Eckdaten des Mikrokontrollers

Hersteller	STMicroelectronics
Kern	Arm 32-bit Cortex-M0 Central processing unit (CPU)
Programmspeichergröße	128 kB
Random access memory (RAM) Datengröße	16 kB
Maximale Taktfrequenz	48 MHz
Digitale und I/O Spannungsversorgung	$2 - 3.6$ V
Anzahl der Inputs/Outputs	bis zu 87
Peripherie	Direct Memory Access (DMA) Controller mit 7 Kanälen 12-Bit Analog-Digital-Umsetzer (ADU) mit 16 Kanälen 12-Bit Digital-Analog-Umsetzer (DAU) mit 2 Kanälen Zwölf Timer
Kommunikation Interfaces	Zwei Inter-integrated circuit (I^2C)-Anbindungen Vier Universal Asynchronous Receiver Transmitter (UART)-Anbindungen Zwei Serial Peripheral Interface (SPI) Anbindungen Eine Controller Area Network (CAN) Anbindung Ein Universal Serial Bus (USB) 2.0 Interface

Durch die Verwendung des NUCLEO-Boards sind die gesamten Peripherien über Steckverbinder zu erreichen. Ebenfalls stehen eine User-Leuchtdiode (LED) und ein User-Button zur freien Programmierung zur Verfügung. Die Programmierung des Mikrokontrollers wird durch einen USB-C Zugang ermöglicht. Es sind mehrere Möglichkeiten vorhanden das Board über Spannungen zu versorgen. Zum einen kann das Board über die USB-C Schnittstelle versorgt werden, zum anderen gibt es externe Spannungspins, an denen eine Spannung von 3.3 V oder 5 V angelegt werden kann.

3 Rahmenbedingung

Innerhalb dieses Kapitels werden die Rahmenbedingungen des Projekts beschrieben. Hierbei wird zunächst die bisherige Ausgangssituation genauer betrachtet. Anschließend werden die einzelnen gestellten Anforderungen an dieses Projekt aufgeführt und analysiert.

3.1 Ausgangssituation

Innerhalb von diesem Unterkapitel werden die zu entwickelnden Komponenten in das bereits gegebene System eingeordnet. Das Ziel ist es einen Überblick über das gesamte System zu erhalten. Die Abbildung 3.1 dient zur visuellen Unterstützung und zeigt die einzelnen Komponenten des gesamten Systems.

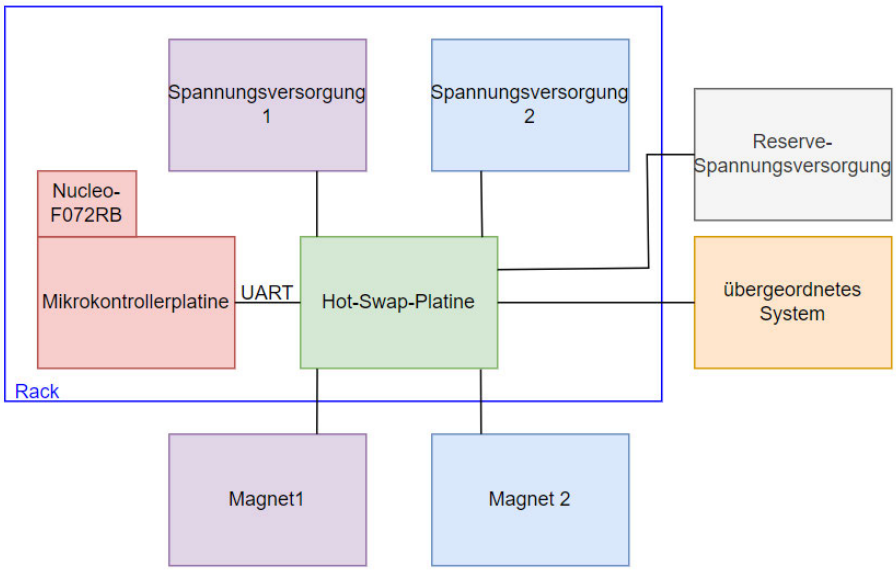


Abbildung 3.1: Überblick über das Gesamtsystem

Zentral in grün ist die Hot-Swap-Platine dargestellt. Die Hot-Swap-Platine enthält die in dem Grundlagenkapitel 2.1 vorgestellte Hot-Swap-Matrix. Anders wie in der Vorstellung in dem Grundlagenkapitel, wird diese Hot-Swap-Matrix für zwei Magnetstromkreise ausgelegt. Dies bedeutet, dass an die Hot-Swap-Matrix zwei unterschiedliche Versorgungsspannungen und zwei Magnete angeschlossen werden können. Beide Versorgungskreise weisen dabei zueinander unterschiedliche Potentiale auf und sind strikt voneinander zu trennen. Die Versorgungsspannungen werden von einer Spannungsquelle mit zwei separaten Ausgängen bereitgestellt. In der Abbildung sind die zwei Versorgungskreise in lila und in blau dargestellt. Der Prototyp dieser Hot-Swap-Platine wird zeitgleich zu dieser Arbeit entwickelt.
In grau ist die Reserve-Spannungsversorgung dargestellt. Sollte ein Problem in einer der Spannungsversorgungen auftreten, kann die Reserve-Spannungsversorgung durch die MOSFET Schalter an einen der Versorgungskreise zugeschaltet werden. Dies soll den Ausfall eines Magneten aufgrund einer fehlerhaften Versorgungsspannung entgegenwirken. Kontrolliert wird dieser Vorgang durch ein übergeordnetes System, welches in der Grafik in orange dargestellt ist.
In rot ist die in dieser Arbeit zu entwickelnde Erweiterung des Gesamtsystems zu erkennen. Die Erweiterung umfasst eine Platine, welche mit einem Mikrokontroller ausgestattet ist. Die genauen gestellten Anforderungen an diese Mikrokontrollerplatine sind im nächsten Unterkapitel einzusehen.
Die Spannungsquelle, die Hot-Swap-Platine und die zu entwickelnde Erweiterung befinden sich zusammen in einem Einschubfach eines Racks, welches voraussichtlich nach Realisierung des Projektes in der Halle der Petra IV Anlage aufzufinden ist.

3.2 Anforderungen

Es folgt die Vorstellung der Anforderungen an das in dieser Arbeit zu entwickelnde System. Hierbei werden die Anforderungen in einen hardware-spezifischen und einen software-spezifischen Teil unterteilt. Der hardware-spezifische Teil umfasst dabei die Anforderungen an die Konzeptionierung und an die Entwicklung der Mikrokontrollerplatine. Innerhalb des Softwareteils werden die Anforderungen an die Signalauswertung durch den Mikrokontroller thematisiert.

Innerhalb des Hardwareteils ist zunächst eine Konzeptfindung durchzuführen. Hierbei sind geeignete Verfahren zur Überwachung der Funktion der Hot-Swap-Platine und der

Magnete zu ermitteln. Das Ziel ist es, mit diesem Verfahren eine mögliche Fehlfunktionen innerhalb der oben genannten Komponenten frühzeitig zu erkennen. Anschließend sind diese Informationen möglichst zeitnah an das übergeordnete System weiterzuleiten. Um dies zu ermöglichen, sind bei der Entwicklung der Verfahren die kritischen Signale und Spannungen, welche zu einem Ausfall des Elektronenstrahls führen können, besonders zu berücksichtigen.
Anschließend müssen für die gefundenen Überwachungsverfahren effiziente Umsetzungsmöglichkeiten ermittelt werden. Hierbei sind geeignete Schaltungen zu entwickeln und passende Bauteile auszuwählen. Der Schaltplan muss mit dem Entwicklungstool Eagle erstellt werden, da dieses Tool im Unternehmen für die Platinenentwicklung eingesetzt wird und daher unter anderem die benötigten Bibliotheken bereits vorhanden sind. Besondere Beachtung bei der Erstellung des Schaltplans liegt dabei auf dem Vorhandensein von verschiedenen Potentialen. In dieser Anwendung sind insgesamt drei verschiedene Potentiale vorhanden. Jeweils ein Potential für jeden Magnetstromkreis und ein weiteres Potential für die Mikrokontrollerplatine. Eine Vermischung dieser Potentiale führt zu unerwünschten Nebeneffekten. Bei der Auswahl der Bauteile ist zu berücksichtigen, dass bevorzugt standardisierte Bauteile verwendet werden. Dadurch soll eine höhere Flexibilität bei der Beschaffung, eine Unabhängigkeit von bestimmten Herstellern und eine geringere Gefahr für potentielle Engpässe innerhalb der Lieferkette erreicht werden. Gleichzeitig können Kosten bei der Bauteilebeschaffung reduziert werden.
Bei der Entwicklung der Schaltungen ist zu berücksichtigen, dass alle Signale, welche überwacht werden sollen, eine Verbindung zu dem Mikrokontroller aufweisen müssen. Hierbei ist es erforderlich das STM32 Nucleo-64-Board NUCLEO-F072RB in den Schaltplan zu integrieren. Dieses Board enthält den Mikrokontroller STM32F072RB. Beide Komponenten wurden bereits im Grundlagenkapitel 2.2 vorgestellt.
Ebenso ist vorgegeben, dass die Mikrokontrollerplatine mit der Hot-Swap-Platine über eine UART-Verbindung kommunizieren können muss. Hierüber erfolgt der Austausch von Daten und von allgemeinen Anweisungen. Um dies zu realisieren, müssen beide Platinen über Steckverbinder miteinander verbunden werden. Über die Steckverbinder muss zusätzlich zum Austausch der Daten über UART der gesamte Signalaustausch erfolgen. Dies umfasst die Übertragung von Versorgungsspannungen und die Übertragung der zu überwachenden Signale. Damit die Steckverbinder später miteinander genau in der Position übereinstimmen, ist eine enge Zusammenarbeit bei der Entwicklung der Prototypen der Hot-Swap-Platine und der Mikrokontrollerplatine erforderlich. Schon bei geringen positionellen Abweichungen der Steckverbinder können die Platinen nicht mehr korrekt verbunden werden.

Nach der Implementierung der Schaltungen im Schaltplan ist das Layout der Platine zu erstellen. Dies erfolgt ebenfalls in dem Entwicklungstool Eagle. Die Platine muss dabei vier Lagen umfassen. Bei dem Erstellen des Layouts ist unter anderem darauf zu achten, dass die Platzierung der Bauteile auf der Platine so erfolgt, dass die entstehenden Verbindungen zwischen den Bauteilen möglichst kurz sind. Dies bedeutet, dass keine Leiterbahn von einem Ende zu dem anderen Ende der Platine gezogen werden darf. Vor allem bei störanfälligen Signalen muss auf eine kurze Anbindung geachtet werden. Des Weiteren sollen die Mikrokontrollerplatine und die Hot-Swap-Platine beide zusammen über ein Einschubfach direkt in das Rack implementiert werden. Aus diesem Grund sind vorgegebene Maße einzuhalten, welche die Platinen zusammen nicht überschreiten dürfen. Der Einschub ist dabei 100 mm breit, 35 mm hoch und 160 mm tief. Bei einer Überschreitung dieser Maße kann nicht garantiert werden, dass die Platinen in das Rack passen.
Ebenfalls zu beachten ist, dass die Platine nur einen geringen Stromverbrauch aufweisen darf, da die vorgesehene Anbindung zur Spannungsversorgung der beiden Platinen nur eine begrenzte Menge an Leistung zur Verfügung stellen kann. Es muss eine maximale Leistung von 1 W eingehalten werden.
In der Tabelle 3.1 sind die einzelnen Anforderungen an die Hardwareentwicklung nochmals zusammengefasst.

Tabelle 3.1: Anforderungen an den Hardwareteil

	Anforderung
1.	Konzeptfindung für geeignete Überwachungsverfahren, um eine Fehlfunktion an der Spannungsversorgung der Magnete oder an der Hot-Swap-Matrix frühzeitig zu erkennen
2.	Entwicklung von Schaltungen zur Umsetzung der Überwachungsverfahren
3.	Verwendung des STM32 Nucleo-64 Boards NUCLEO-F072RB mit dem Mikrokontroller STM32F072RB
4.	Entwicklung einer vierlagigen Platine mit Eagle
5.	Maximaler erlaubter Leitungsverbrauch von 1W

Es folgt die Vorstellung der Anforderungen an den Softwareteil.
Zunächst ist ein geeignetes Softwarekonzept zu erstellen. Hierbei ist zu ermitteln, wie die eingehenden Signale empfangen und ausgewertet werden müssen. Dabei sind geeignete Verfahren und Mechanismen zum Datenempfang durch einen Mikrokontroller für die entsprechenden Signale zu bewerten. Anschließend muss für die Signale eine Priorisierung festgelegt werden. Hierbei sind kritische Signale höher zu priorisieren als weniger kritische Signale. Dabei gilt es zu entscheiden, welche der übertragenden Signale als kritisch

eingestuft werden müssen.
Die Weiterleitung der Daten an die Hot-Swap-Platine wird durch eine UART-Verbindung ermöglicht. Hierbei soll im Rahmen des Softwarekonzeptes ein geeignetes Fehlererkennungsverfahren gefunden werden. Mithilfe von diesem Verfahren muss eine fehlerhafte Kommunikation zwischen dem Mikrokontroller und der Hot-Swap-Platine festgestellt werden können. Ebenfalls muss ein Konzept für die Datenübermittlung über die UART-Verbindung erstellt werden. Dabei ist erneut die Priorisierung der einzelnen Signale zu beachten.
Besonders zu berücksichtigen ist die Konzeptionierung des Timings des gesamten Softwarecodes. Hierbei müssen die Datenerfassung, die Datenverarbeitung und die Datenübermittlung genau aufeinander abgestimmt werden. Es dürfen keine Informationen durch das zu langsame Verarbeiten der eingehenden Signale durch die Software verloren gehen. Ebenso muss auf eine konstante Datenübermittlung mittels UART geachtet werden. Durch die konstante Übermittlung sollen mögliche Fehlkommunikationen erkannt werden. Auch wird vermieden, dass ein Datenstau entsteht, welcher zu einer zeitlichen Verzögerung der Datenübermittlung führt.
Anschließend ist die Software auf dem Mikrokontroller zu implementieren. Hierbei muss die Entwicklungsumgebung STMCubeIDE verwendet werden. Als Programmiersprache ist C vorgesehen. Des Weiteren ist bei der Implementierung drauf zu achten, dass das System möglichst autark verwendet werden soll. Dies bedeutet, dass das Softwaresystem Programmfehler, wie beispielsweise das Aufhängen des Systems an einer Stelle des Programmcodes, selbst erkennen und beheben muss.
In der Tabelle 3.2 werden die Anforderungen an die Software nochmals zusammengefasst tabellarisch dargestellt.

Tabelle 3.2: Anforderungen an den Softwareteil

	Anforderung
1.	Konzepterstellung zum Empfang und zur Auswertung der zu überwachenden Signale
2.	Konzepterstellung für Fehlererkennungsverfahren, um eine inkorrekte UART-Übertragung zu ermitteln
3.	Konzepterstellung für Übermittlungsmöglichkeiten der Daten via UART an die Hot-Swap-Platine
4.	Konzepterstellung für das Timing zwischen Datenerfassung, Datenauswertung und Datenübermittlung
5.	Implementierung des Software in der STMCubeIDE unter Verwendung der Programmiersprache C

4 Konzept

In diesem Kapitel wird das Konzept dieser Arbeit vorgestellt. Hierbei gliedert sich das Konzept in einen Hardwareteil und einen Softwareteil.
Innerhalb des Hardwareteils werden Konzepte zur Erstellung der Platine getroffen. Im Softwareteil wird die Auswertung der einzelnen Signale durch den Mikrokontroller und der Aufbau des gesamten Softwarecodes behandelt.

4.1 Hardware

Es folgt die Konzeptvorstellung des Hardwareteils. Zunächst werden die Signale, welche von der Hot-Swap-Platine an die Mikrokontrollerplatine übergeben werden, vorgestellt. Anschließend werden die daraus resultierenden Konsequenzen für die vorzunehmende Hardwareentwicklung beschrieben. Es folgt die Vorstellung von weiteren möglichen Überwachungsmöglichkeiten, deren Implementierung auf der Mikrokontrollerplatine vorzunehmen ist. Anschließend wird der Anschlussplan der Signale an die Pins des Mikrokontrollers vorgestellt.

4.1.1 Überblick über die von der Hot-Swap-Platine übertragenen Signale

In diesem Teil des Hardwarekonzeptes werden zunächst alle Signale beschrieben, welche von der Hot-Swap-Platine an die Mikrokontrollerplatine übermittelt werden. Die zugehörigen Hardwareschaltungen zur Erzeugung der Signale befinden sich auf der Hot-Swap-Platine und werden in dieser Arbeit nicht tiefer gehend erläutert.

Überwachung der Hot-Swap-Matrix

Mithilfe der ersten implementierten Überwachungsmethode wird die Hot-Swap-Matrix überwacht. Es soll die Schalterstellung der verschiedenen MOSFET-Schalter durch digitale Signale erkenntlich gemacht werden. Hierfür sind für jeden Magnetstromkreis zwei digitale Signale vorhanden. Das eine digitale Signal zeigt mit seinem Pegel die aktuelle Ansteuerung des Schalters, welcher sich zwischen Spannungsversorgung und Magnet befindet, an. Ein High-Pegel bedeutet, dass der Schalter geschlossen ist. Ein Low-Pegel zeigt die geöffnete Schalterstellung an. Dieses Signal heißt Matrix_1_1. Ein Weiteres digitales Signal zeigt die Ansteuerung des Schalters, welcher zwischen Reserve-Spannungsversorgung und Magneten liegt, an. Dieses Signal heißt Matrix_1_2. Für den zweiten Magnetstromkreis auf der Hot-Swap-Platine werden die gleichen Schalter mit digitalen Signalen überwacht. Diese Signale heißten dann Matrix_2_1 für den Schalter zwischen Spannungsversorgung und Magnet und Matrix_2_2 für den Schalter zwischen Reserve-Spannungsversorgung und Magnet. Die Abbildung 4.1 soll dies grafisch verdeutlichen. In rot werden die Schalter, welche für die jeweiligen Matrix-Signale verwendet werden, farblich hervorgehoben. Beide Magnetstromkreise sind über eine Hot-Swap-Matrix verbunden.

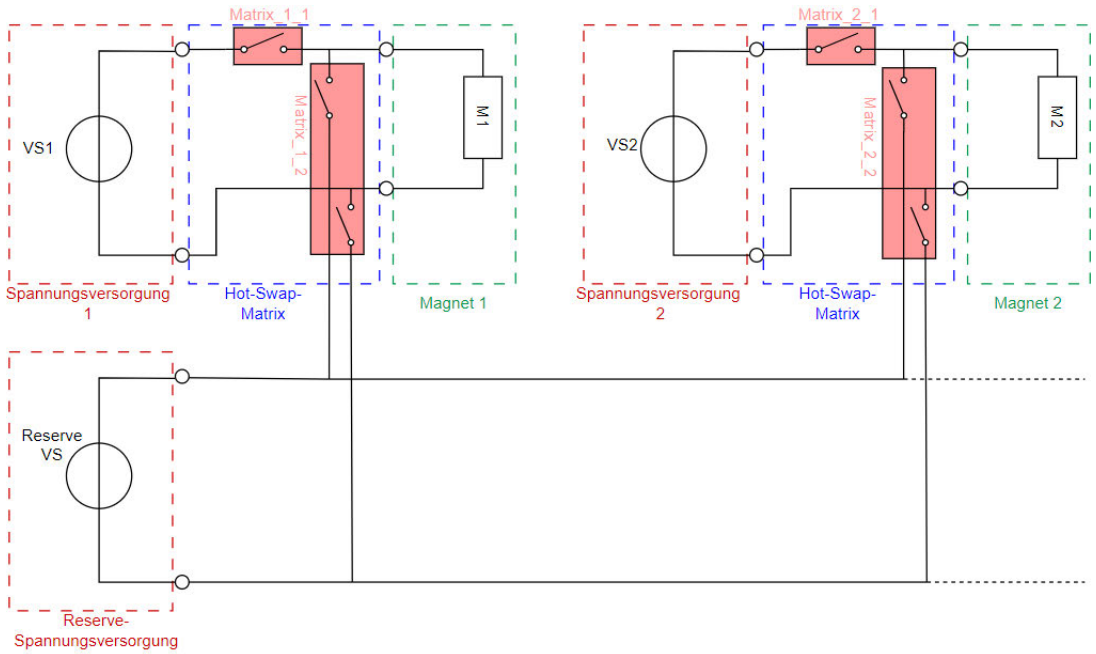


Abbildung 4.1: Übersicht mit zwei Magnetstromkreisen und einer Reservespannungsversorgung

Mithilfe von dieser Überwachungsmethode soll ein vereinfachter Überblick über die verwendeten Spannungsversorgungen bei den einzelnen Magneten ermöglicht werden. Durch das übergeordnete System kann so nachvollzogen werden, welche Spannungsversorgung welchem Magneten zugeordnet ist. Zudem können Fehler, wie das unbeabsichtigte Zusammenschalten verschiedener Magnetstromkreise durch eine falsche Schalterstellung, reduziert werden.

Überwachung der Magnetspannungen

Eine weitere implementierte Überwachungsmöglichkeit stellt die Überwachung der Magnetspannungen dar. Mithilfe dieser Überwachung können Anstiege beziehungsweise Abfälle in der Spannungsversorgung der beiden Magneten festgestellt werden. Es werden digitale Signale eingesetzt. Mithilfe der Abbildung 4.2 soll die Bedeutung der digitalen Signale genauer erklärt werden.

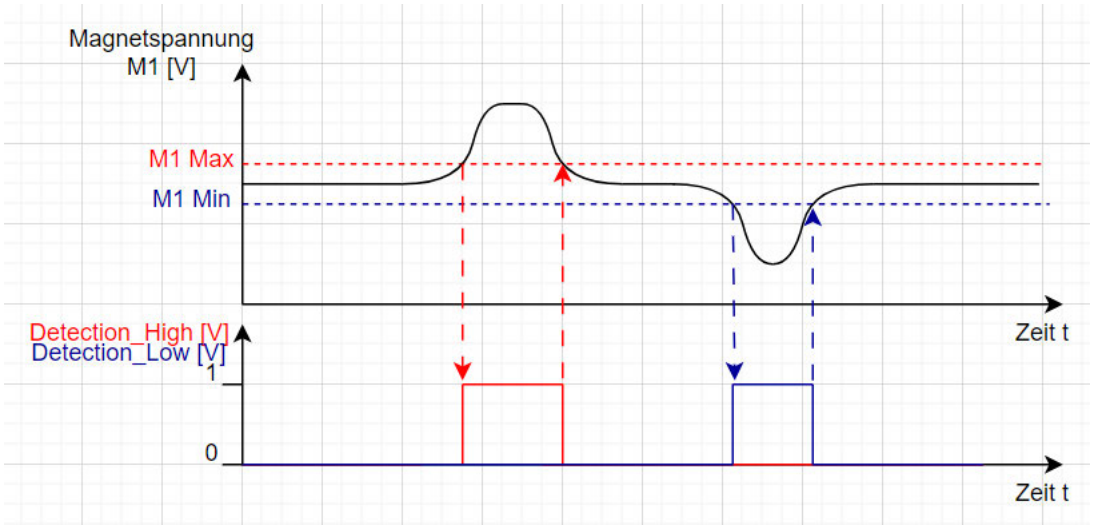


Abbildung 4.2: Entstehung der digitalen Signale zur Überwachung der Magnetspannung

Innerhalb des oberen Graphen ist in schwarz ein möglicher Verlauf der Magnetspannung an dem Magneten M1 zu erkennen. Die gestrichelte rote Linie gibt den maximalen Schwellwert der Magnetspannung und die blaue gestrichelte Linie den minimalen Schwellwert der Magnetspannung an. Sollte die Magnetspannung die rote gestrichelte Linie für den maximalen Schwellwert übersteigen, wird das Detection_1_High-Signal auf eins gesetzt. Dieses Signal wird in rot innerhalb des unteren Graphen dargestellt.

Solange die Magnetspannung über den Schwellwert liegt, bleibt das digitale Signal auf einem High-Pegel. Sollte die Magnetspannung hingegen unter die blau gestichelte Linie, welche den minimalen Schwellwert angibt, fallen, wird das Detection_1_Low Signal auf einen High-Pegel gesetzt. Dieses Signal bleibt solange auf einem High-Pegel, bis die Magnetspannung den unteren Schwellwert wieder übersteigt. Die gleiche Anordnung liegt auch für die Spannung an dem Magneten M2 vor. Hier heißen die digitalen Signale Detection_2_High und Detection_2_Low. Somit sind insgesamt vier digitale Signale für die Überwachung der Magnetspannung zuständig.
Mithilfe dieser digitalen Signale kann eine erhöhte beziehungsweise eine verringerte Versorgungsspannung an den Magneten erkannt werden. Die jeweiligen Schwellwerte sind dabei sehr entscheidend. Da die Spannung direkt die Magnetkraft beeinflusst, ist eine stabile Spannungsversorgung innerhalb eines vorgesehenen Bereichs essentiell. Die Stärke des durch den Magneten erzeugten elektrischen Felds nimmt bei einer zu geringen Spannungsversorgung ab. Bei einer zu hohen Spannungsversorgung nimmt die Stärke hingegen zu. Beides führt dazu, dass der Elektronenstrahl nicht mehr korrekt gesteuert werden kann, was wiederum Ungenauigkeiten und den eventuellen Verlust des Elektronenstrahls begünstigt. Mithilfe dieser Signale ist zusammenfassend eine Änderung in der Spannungsversorgung der Magnete festzustellen. Anhand der übermittelten Daten können geeignete Maßnahmen ergriffen werden. Dazu zählt unter anderem das Umschalten auf die Reserve-Spannungsversorgung.

Überwachung der Dump-Schaltung

Eine weitere Überwachungsmöglichkeit bezieht sich auf die Überwachung der Dump-Schaltung. Hierfür wird zunächst die Funktion der Dump-Schaltung erklärt.
Im Allgemeinen ist eine Dump-Schaltung eine Hardwareschaltung, welche dazu dient eine gespeicherte elektrische Ladung oder Energie sicher abzuleiten oder abzuführen (zu dumpen). In Elektromagneten wird während des Betriebs Energie in Form eines Magnetfelds gespeichert. Beim Abschalten des Stroms kann diese gespeicherte Energie eine hohe Spannung erzeugen, welche Schäden an elektrischen Bauteilen verursacht. Die Ursache für diese entstehenden Spannungsspitzen liegt im Induktionsgesetz [36].

$$\mathrm{U_L} = -\mathrm{L} \cdot \frac{\delta \mathrm{I}}{\delta \mathrm{t}} \tag{4.1}$$

Wenn durch eine Spule ein Strom fließt, wird ein Magnetfeld erzeugt, dessen Stärke proportional zum Strom ist. Wenn der Stromfluss plötzlich unterbrochen wird, steigt die Änderung des Stromflusses ($\frac{\delta I}{\delta t}$) stark an, was wiederum zu einer hohen negativen Spannung (U_L) führt. Diese Spannung kann so hoch werden, dass Bauteile, wie die MOSFETs der Hot-Swap-Matrix, zerstört werden.
Die Dump-Schaltung soll dies verhindern, indem die vorliegende Spannung innerhalb der Schaltung überwacht. Sobald diese Spannung einen Schwellenwert übersteigt, wird ein niederohmiger Pfad über einen Widerstand freigegeben. Über diesen Widerstand wird die gespeicherte Energie gezielt abgeleitet und in Wärme umgewandelt. So können gefährliche Spannungsspitzen vermieden werden. Fließt in der Dump-Schaltung ein bestimmter Strom, schaltet ein Optokoppler durch und erzeugt dabei ein digitales Signal. Dieses digitale Signal wird an die Mikrokontrollerplatine übermittelt und zeigt mit einem High-Signal die Aktivierung der Dump-Schaltung an. Dies kann zum einen als Kontrolle genutzt werden, ob die Schaltung bei einer bewussten Abschaltung der Spannungsversorgung der Magneten korrekt funktioniert. Zum anderen kann über das unvorhergesehene Auftreten eines High-Signals auf ein mögliches Problem mit der Spannungsversorgung des Magneten geschlossen werden. Da jede Hot-Swap-Matrix mit zwei Magnetstromkreisen verbunden werden kann, sind ebenso zwei Dump-Schaltungen vorhanden. Es werden somit zwei digitale Signale übermittelt. Dump_1 für den Magnetstromkreis 1 und Dump_2 für den Magnetstromkreis 2.

Adresssignale

Als sogenannte Adresssignale dienen vier digitale Signale. Diese Signale sind nicht für eine bestimmte Überwachungsmöglichkeit zuständig, sondern sollen Informationen über die verwendete Hot-Swap-Platine übermitteln. Die Signale übermitteln mit einem High-Pegel eine binäre Eins und mit einem Low-Pegel eine binäre Null. Beispielsweise kann so eine Revisionsnummer der Platine angegeben werden. Mithilfe der Signale ist festzustellen, welche Platine wo verbaut ist und welche Aktualität diese Platine aufweist. Dies ermöglicht einen leichteren Überblick über die verwendete Hardware innerhalb der Anlage.

PWM-Signal

Des Weiteren wird ein Pulsweitenmodulation (PWM)-Signal zur Informationsübermittlung übertragen. Die Übertragung des PWM-Signals bezieht sich auf eine Überwachungsmethode, welche auf der Mikrokontrollerplatine zu implementieren ist. Somit wird die genaue Bedeutung von diesem Signal erst unter dem Kapitel 4.1.3 Umsetzung weiterer Überwachungsmöglichkeiten auf der Mikrokontrollerplatine im Unterkapitel Auswertung des PWM Signals genauer erläutert werden.

Spannungssignale

Innerhalb von diesem Kapitel werden verschiedene relevante Versorgungsspannungen vorgestellt.
Die Mikrokontrollplatine wird über die Hot-Swap-Platine mit einer 12 V-Spannung versorgt. Ebenso wird ein 5 V-VCC-Spannungssignals, welches für die Spannungsversorgung verschiedener Bauteile auf der Hot-Swap-Platine zuständig ist, an die Mikrokontrollerplatine übergeben. Des Weiteren erfolgt die Übermittlung einer zweiten 12 V-Spannung. Diese Spannung wird von der Spannungsquelle der Magnete erzeugt und ist durch die Mikrokontrollerplatine zu überwachen. Hierbei muss auf eine galvanische Trennung dieser Spannung zu der Mikroplatine geachtet werden. Des Weiteren werden noch zwei Magnetspannungen an die Mikrokontrollerplatine übermittelt. Diese Spannungen zeigen an, wie die Magnete aktuell mit Spannung versorgt werden. Die Magnetspannungen sind ebenfalls galvanisch von der Mikrokontrollerplatine zu trennen.

4.1.2 Konsequenzen für die Konzeptionierung der Hardware

Im folgenden werden nun die Konsequenzen, welche sich aus den vorgestellten Signalen ergeben, für die weitere Hardwarekonzeptionierung der Mikrokontrollerplatine erläutert. Die erste Konsequenz befasst sich mit der Bewerkstelligung der Spannungsversorgung der Mikrokontrollerplatine. Anschließend wird in der zweiten Konsequenz eine erforderliche Spannungsreduktion der übertragenen digitalen Signale konzeptioniert.

Gewährleistung der Spannungsversorgung der Mikrokontrollerplatine

Innerhalb von diesem Kapitel werden verschiedene Methoden zur Bewerkstelligung der Spannungsversorgung der Mikrokontrollerplatine beschrieben. Die Mikrokontrollerplatine wird über eine 12 V Spannung durch die Hot-Swap-Platine versorgt. Das verwendete Mikrokontrollerboard erlaubt eine Spannungsversorgung von 3.3 V oder 5 V. Da noch weitere Bauteile auf der Mikrokontrollerplatine mit einer Versorgungsspannung betrieben werden müssen, wird sich für die Versorgung mit 5 V entschieden. Es muss somit eine Hardwareschaltung entwickelt werden, die die Eingangsspannung auf ein geeignetes Spannungslevel reduziert. Hierfür werden zunächst verschiedene Methoden vorgestellt.

Die erste Methode beinhaltet die Verwendung eines Low Drop Outs (LDOs). Der LDO arbeitet wie ein aktiver linearer Regler, der überschüssige Spannung in Form von Wärme ableitet. Er kann vereinfacht mit einem variablen Widerstand verglichen werden, jedoch steuert er die Spannung durch eine präzise Rückkopplungsschleife [38].
Vorteilhaft an dieser Methode ist die unkomplizierte Umsetzung. Es muss eine einfache Schaltung mit wenigen externen Komponenten entwickelt werden. Typischerweise wird hier nur der LDO mit einem Eingangs- und einem Ausgangskondensator benötigt [24]. Dies führt unter anderem zu einer geringen Fehleranfälligkeit innerhalb des Entwicklungsprozesses. Durch die Verwendung von einer niedrigen Anzahl an Bauteilen kann die Schaltung platzsparend auf der Platine implementiert werden. Ein weiterer Vorteil liegt in der sehr stabilen Ausgangsspannung des LDOs. Die Restwelligkeit und der Ausgangsrauschpegel sind sehr gering, was eine zuverlässige Versorgung des Mikrokontrollerboards mit einer stabilen Eingangsspannung gewährleistet. Durch das geringe Rauschen wird zudem die Wahrscheinlichkeit minimiert, dass andere störungsempfindliche Schaltungen oder Signale auf der Platine beeinträchtigt werden [35].
Hingegen ist nachteilig zu betrachten, dass ein hoher Energieverlust durch die Abgabe der überschüssigen Energie in Form von Wärme entsteht. Besonders bei höheren Spannungsunterschieden ist dies sehr ineffizient [35]. Resultierend nimmt bei einer hohen Spannungsdifferenz von Eingangsspannung zu Ausgangsspannung der Wirkungsgrad stark ab [12]. Auch der Bereich der übertragbaren Ströme ist auf kleine bis mittlere Ströme begrenzt, da hohe Ströme eine erhöhte Wärmeproduktion bedeuten. Unter Umständen ist es somit notwendig Kühlkörper einzusetzen, welche den LDO herunterkühlen, um einen Ausfall des Bauteils zu vermeiden [34].

Die zweite mögliche Methode besteht darin, eine Zenerdiode (Z-Diode) mit Vorwider-

stand zu verwenden. Die Zenerdiode wird dabei parallel zur Last in Sperrrichtung betrieben, sodass die Spannung über der Last immer gleich der Spannung über der Z-Diode entspricht. Auf diese Weise wird eine konstante Ausgangsspannung gewährleistet. Die überschüssige Spannung, also die Differenz zwischen Eingangsspannung und Zenerspannung, fällt über den Vorwiderstand ab. Des Weiteren begrenzt der Vorwiderstand den Stromfluss durch die Zenerdiode und schützt diese vor Überlastung [27].
Ein Vorteil liegt in der einfachen Implementierung der Schaltung. Für eine korrekte Schaltungsfunkion müssen die Zenerdiode und der Vorwiderstand passend ausgelegt werden [27]. Die aufzubringende Entwicklungsarbeit ist somit gering. Die geringe Anzahl an Bauteilen reduziert sowohl den Platzbedarf auf der Platine als auch die aufzubringenden Bauteilkosten.
Ein wesentlicher Nachteil dieser Methode ist ihre geringe Effizienz. Bei höheren Strömen wird der Vorwiderstand zunehmend ineffizient und erzeugt erhebliche Wärme. Ebenso führt eine große Spannungsdifferenz dazu, dass die Schaltung ähnlich viel Wärme wie ein LDO abgibt. Jedoch ist bei dieser Schaltung die Regelgenauigkeit im Vergleich zum LDO deutlich geringer. Dies liegt daran, dass die Genauigkeit von dem Widerstand der Last abhängig ist. Somit können Änderungen in der Last eine deutlich veränderte Ausgangsspannung bewirken [27].

Als dritte Methode wird die Verwendung eines Abwärtswandlers, auch Buck-Converter genannt, aufgeführt. Ein Abwärtswandler zählt zu den DC-DC-Wandlern. Der Wandler wandelt die Spannung durch ein Schaltprinzip, bspw. PWM, und durch Energiezwischenspeicherung in Spulen und Kondensatoren um [18]. Durch verschiedene Bauteilformen kann sowohl eine Spannungsreduktion, als auch eine Spannungserhöhung ermöglicht werden [42].
Ein besonders positiver Aspekt ist die hohe Effizienz [42]. Diese hohe Effizienz resultiert aus der Verwendung von Induktivitäten und Kondensatoren zur Energiespeicherung. Dadurch kann die Energie besonders bei großen Spannungsdifferenzen effizient umgewandelt werden. Somit wird ein hoher Wirkungsgrad erreicht, welcher bei niedrigen und hohen Spannungsdifferenzen einen gleichbleibenden hohen Wert aufzeigt [12]. Im Gegensatz zum LDO oder zur Z-Diode entsteht keine große Wärmeabgabe. Die gewünschte Ausgangsspannung lässt sich durch die externe Beschaltung des Abwärtswandlers mit Widerständen einstellen. Dies ermöglicht eine hohe Flexibilität in der Entwicklungsphase. Durch das Vorhandensein einer Feedback-Schleife wird zudem eine gute Stabilität der Ausgangsspannung erreicht [44] [39] [11].
Ein Nachteil liegt in der höheren Komplexität des Entwicklungsaufwands [42]. Es wer-

den mehrere externe Bauteile, wie Widerstände, Kondensatoren und Spulen benötigt. Auch die aufzubringenden Kosten sind höher, als bei den anderen bisher vorgestellten Methoden. Ein ebenfalls wichtiger Nachteil ist die Erzeugung von hochfrequenten Störungen, wie elektromagnetische Interferenz (EMI), die auf benachbarte Schaltungen oder störungsempfindliche Signale negative Auswirkungen haben können [42]. Hierfür werden unter Umständen zusätzliche Filter benötigt. Innerhalb der Tabelle 4.1 werden nochmals die Vor- und Nachteile aller vorgestellten Methoden aufgeführt.

Tabelle 4.1: Vor- und Nachteile der Methoden zur Gewährleistung der Spannungsversorgung

	LDO	Zenerdiode	Abwärtswandler
Wirkungsgrad	bei geringer Spannungsdifferenz nahezu 100%, nimmt bei Zunahme der Differenz stark ab [24]	gering, nimmt bei Zunahme der Differenz stark ab	hoch
Komplexität	gering	sehr gering	hoch
Kosten	niedrig	sehr niedrig	mittel
Vorteile	geringe Restwelligkeit und geringes Rauschen	sehr einfaches und kostengünstiges Design	hohe Effizienz
	einfaches und kostengünstiges Design	geringer Platzbedarf auf der Platine	flexibel
	kompakt und leicht integrierbar		geeignet für große Spannungs-differenzen
Nachteile	hohe Verlustleistung	Ausgangsspannung von Last abhängig	komplexer Schaltungsaufbau
	begrenzte Stromkapazität	gegebenenfalls hohe Wärmeentwicklung	hohe EMI
	Wirkungsgrad nimmt bei Zunahme der Spannungsdifferenz ab	begrenzte Spannungsreglung	höhere Kosten

Es ist im folgenden eine Entscheidung über die zu verwendende Methode zu treffen. Innerhalb des Projekts ist eine Spannungsreduktion von 12 V auf 5 V vorzunehmen. Somit wird ein Spannungsabfall von insgesamt 7 V über dem zu verwendenden Bauteil erwartet. Ein erheblicher Teil der Energie wird bei der Z-Diode in Form von Wärme abgegeben, was den Wirkungsgrad der Schaltung stark reduziert. Zusätzlich ist die Ausgangsspannung

bei der Z-Diode stark von der Last abhängig. Bei größeren Änderungen der Last oder des Laststroms kann die Spannung über der Z-Diode schwanken, was zu einer ungenauen Spannungsregulierung führt. Dies bedeutet, dass die Z-Diode in vielen Fällen keine präzise und stabile Reduzierung der Spannung liefert. Aus diesen Gründen stell die Z-Diode in dieser Anwendung keine geeignete Lösung dar.
Im Vergleich zur Z-Diode weist der LDO eine deutlich bessere Regelgenauigkeit auf. Jedoch wird auch bei dem LDO ein großer Teil der Energie in Wärme abgegeben. Der Wirkungsgrad des LDOs kann bei der Annahme, dass der Eingangsstrom gleich dem Ausgangsstrom ist, mit der folgenden Formel vorgenommen werden [45].

$$\eta = \frac{\mathrm{V_{OUT}}}{\mathrm{V_{In}}} \tag{4.2}$$

Wird eine Ausgangsspannung von 5 V und eine Eingangsspannung von 12 V angenommen, ergibt sich folgende Rechnung.

$$\begin{aligned} \eta &= \frac{5\,\mathrm{V}}{12\,\mathrm{V}} \\ &= 0.416 \end{aligned}$$

Es ergibt sich ein Wirkungsgrad von 41.6 %. Dieser Wert ist sehr gering. Aus diesem Grund wird die Verwendung eines LDOs ausgeschlossen.
Der Abwärtswandler bietet im Vergleich zur Z-Diode und zum LDO einen gleichbleibenden hohen Wirkungsgrad und ist somit in dieser Anwendung eine geeignete Lösung. Es wird eine präzise Spannungsumwandlung ermöglicht, wodurch die Spannungsversorgung der Bauteile auf der Mikrokontrollerplatine gesichert gewährleistet ist. Die höheren Kosten und der gesteigerte Entwicklungsaufwand werden für eine gesicherte Spannungsversorgung hingenommen.

Spannungsreduktion von digitalen 5 V-Signalen auf 3.3 V

In diesem Kapitel wird die zweite Konsequenz für die Hardwareentwicklung beschrieben. Alle digitalen Signale, welche von der Hot-Swap-Platine an die Mikrokontrollerplatine übergeben werden, liegen auf einem 5 V-Pegel. Die General Purpose Input/Output (GPIO)-Eingänge des zu verwendenden Mikrokontrollers können maximal 3.3 V verarbeiten. Andernfalls würde eine zu hohe Spannung an den GPIO Pins Schäden an dem Mikrokontroller und dessen Beschaltung herbeiführen. Um dies zu verhindern, sind die

digitalen Signale von einem 5 V-Pegel auf einen 3.3 V-Pegel zu reduzieren. Hierbei sind geeignete Bauteile zu finden, welche diese Reduktion durchführen können. Im folgenden werden verschiedene Methoden vorgestellt.

Die erste Option sieht die Verwendung eines Spannungsteilers vor. Hierbei wird die gewünschte Ausgangsspannung über zwei in Reihe geschalteten Widerstände eingestellt. Vorteilhaft an dieser Methode ist die Einfachheit in der Implementierung. Dies erspart Entwicklungsarbeit, da die Auslegung der Widerstände durch eine einfache Formelberechnung möglich ist. Auch können Kosten für das Material gespart werden, da Widerstände im Vergleich zu anderen möglichen Bauteilen kostengünstig sind. Ebenfalls wird eine Unabhängigkeit von bestimmten Herstellern erreicht, da Widerstände vielfältig zu erhalten sind.
Negativ zu betrachten ist die Ungenauigkeit der Signalübertragung, welche die Signalqualität des Ausgangssignals beeinflusst. Diese Ungenauigkeit entsteht unter anderem durch die vorhandenen Toleranzen der einzelnen Widerstände [33]. Zudem ändern sich die realen Widerstandswerte mit der Temperatur. Somit können Widerstände mit hohem Temperaturkoeffizienten bei einer schwankenden Umgebungstemperatur die Ausgangsspannung des Spannungsteilers beeinflussen [33]. Zudem ist es möglich, dass externe Störungen, wie elektromagnetische Interferenzen, über ungeschirmte Leitungen oder die Widerstände in den Spannungsteiler einkoppeln und somit die Signalqualität des Ausgangssignals reduzieren [26]. Ein weiterer Nachteil ist, dass bei hohen Frequenzen und schnellen Signalen eine verstärke Signalverzerrung auftreten kann. Dies ist unter anderem auf die parasitären Kapazitäten der realen Widerstände und der kapazitiven Kopplung der Verbindungsleitungen zur Masse zurückzuführen. Durch die Verwendung von mehreren Widerständen erzeugen die parasitären Kapazitäten ein Tiefpass-Verhalten. Somit werden hohe Frequenzen stärker gedämpft als niedrige Frequenzen. Dies führt insbesondere bei steilen Signalflanken zu einer Verzerrung des Signals [14][26]. Des Weiteren ändert sich bei hohen Frequenzen die Impedanz der Schaltung. Dies bedeutet, dass die Frequenzabhängigkeit der Impedanz das Verhältnis des Spannungsteilers verändert, wodurch eine abweichende Ausgangsspannung festzustellen ist [14].

Als zweite Methode wird die Verwendung eines MOSFET-basierenden Wandlers vorgestellt. Hierbei wird ein n-Kanal-MOSFET-Transistor so verschaltet, dass seine Source mit dem niedrigeren Spannungspegel verbunden ist. Der Gate-Eingang wird über einen Widerstand auf diese niedrigere Spannung festgelegt. Der Drain-Pin des MOSFETs ist über einen Pull-Up-Widerstand mit der höheren Spannung verbunden.Liegt am Drain-

Pin das höhere Spannungssignal an, bleibt der MOSFET gesperrt. In diesem Zustand sorgt der Pull-Up-Widerstand dafür, dass die Spannung am Drain bei 5 V gehalten wird, während die Source durch einen weiteren Pull-Up-Widerstand auf 3.3 V fixiert bleibt. Sinkt das 5 V-Signal auf 0 V, wird der MOSFET leitend. Dadurch wird die Source auf 0 V gezogen, sodass auch am Mikrocontroller-Eingang 0 V anliegt. [25].
Eine geringe Verzögerung und Verzerrung des Ausgangssignals wird durch eine theoretisch sehr hohe Geschwindigkeit des Schaltvorgangs des MOSFETs erreicht. Diese beruht auf der Beschaffenheit des Kanals. Da es sich um ein unipolares Bauelement handelt, müssen Ladungsträger bei einem Schaltvorgang nicht rekombinieren, sondern lediglich zufließen oder abfließen. Dies führt zu einer hohen Beweglichkeit der Ladungsträger und ermöglicht Schaltfrequenzen im MHz-Bereich [48]. Ebenfalls vorteilhaft zu erwähnen ist, dass MOSFETs sehr weit verbreitet sind. Sie sind somit von mehreren unabhängigen Herstellern zu beziehen [41]. Des Weiteren kann diese Schaltung bei Bedarf auf eine bidirektionale Pegelwandlung ausgelegt werden. Anschließend ist eine Pegelwandlung in beide Richtungen möglich [25].
Nachteilig an dieser Methode sind folgende Punkte zu betrachten. Die Schaltgeschwindigkeit der MOSFETs ist abhängig von der Auslegung der Pull-Up-Widerstände. Dies liegt an den Gate-Kapazitäten, die beim Schalten geladen oder entladen werden müssen [48]. Die Pull-Up-Widerstände bilden mit der Ladekapazität eine RC-Zeitkonstante $\tau = R \cdot C$ [16]. Je größer somit die Widerstandswerte gewählt werden, desto länger dauert das Laden der Kondensatoren und desto langsamer ist die Schaltzeit der MOSFETs. Eine langsame Schaltzeit kann wiederum zu Verzögerungen und Verzerrungen im Ausgangssignal führen. Somit ist für eine gute Signalqualität die korrekte Dimensionierung der Schaltung essentiell. Ein weiterer Nachteil liegt in der Anzahl der zu überwachenden Signale. Insgesamt werden 15 digitale Signale übertragen, welche alle eine Pegelwandlung benötigen. Die MOSFET-Schaltung muss für jedes einzelne digitale Signal implementiert werden. Dies führt wiederum zu einem höheren Aufwand, zu gesteigerten Kosten und zu einem erhöhten Platzbedarf auf der Platine. Ein letzter zu erwähnender Nachteil ist die vorhandene Temperaturabhängigkeit der MOSFETs [20].

In der dritten Methode wird die Verwendung eines Pegelwandlungs-ICs vorgestellt. Dieser integrierte Schaltkreis (IC) enthält die in Methode drei vorgestellte MOSFET-Variante. Zusätzlich sind in diesem IC bereits Pull-Up-Widerstände, Verstärker, um die Signale präziser zu formen, und Schutzmechanismen integriert worden[32].
Zunächst sollen die Vorteile dieser Methode genannt werden. Mithilfe von diesen ICs kann eine hohe Geschwindigkeit und Signalqualität erreicht werden. Die Schaltung ist auf das

Problem der Spannungsreduktion spezialisiert und optimiert worden. Ebenso müssen die kritischen Pull-Up-Widerstände bei der Schaltungsentwicklung nicht manuell ausgelegt werden [32]. Dies erspart viel Arbeit und Ungenauigkeiten, da die optimierte Auslegung bereits innerhalb des ICs integriert ist. Die Beschaltung der ICs ist ebenfalls einfach vorzunehmen. Ein weiterer Vorteil sind die bereits in dem IC integrierten Schutzmechanismen. Hier kann ein Schutz vor elektrostatischer Entladung (ESDs), ein Kurzschlussschutz oder ein Überspannungsschutz integriert sein, welcher die Zuverlässigkeit der Schaltung erhöht [32]. Durch die Optimierung der Schaltung ist die Energieeffizienz von diesem IC sehr hoch. Dadurch wird ein geringer Stromverbrauch garantiert. Die ICs werden von mehreren Herstellern angeboten. Dabei gibt es IC Varianten mit mehreren Anschlüssen für verschiedene Eingangssignale. Dies führt zu einer Reduktion des Platzbedarfs, da eine Schaltung für mehrere Signale ausgelegt werden kann [31].
Nachteilig zu erwähnen sind die aufzubringenden Kosten für die Bauteilbeschaffung. Diese ICs sind meist teurer als die anderen vorgestellten Methoden. Für den Betrieb von diesem IC werden auf beiden Spannungspegelseiten unabhängige Spannungsquellen benötigt. Dies erhöht die Spannungskomplexität auf der Platine. Bei extremen Temperaturen kann sich zudem die Performance der ICs verschlechtern, da die internen Transistoren und Schutzmechanismen temperaturabhängig sind [31]. In der Tabelle 4.2 werden nochmals alle Methoden und die wichtigsten Aspekte gegeneinander über gestellt.

Tabelle 4.2: Vergleich von verschiedenen Methoden zur Spannungspegelreduktion

	Spannungsteiler	MOSFET-basierte Pegelwandlung	Pegelwandlungs-IC
Kosten	sehr günstig	günstig/moderat	moderat
Schaltungs-aufwand	sehr gering	moderat durch Pull-Up-Widerstände	höher, aber einfache Integration
Signalqualität	beeinträchtigt bei schellen Signalen	gut	sehr gut
Geschwindigkeit	begrenzt, von Widerständen abhängig	hoch	sehr hoch
Bidirektionalität	nein	ja	ja
Temperatur-abhängigkeit	Widerstandswerte können temperatur-abhängig sein	vorhanden	gering
Überspannungs-schutz	keiner	keiner	ja

Anschließend wird eine Methode ausgewählt, welche bei der Entwicklung der Platine umzusetzen ist. Das wichtigste Kriterium ist das Beibehalten einer guten Signalqualität, damit der Mikrokontroller das Signal korrekt auswerten kann. Sollte eine zu starkes Rauschen auftreten, könnte das Signal falsch interpretiert werden, was wiederum zu einem Informationsverlust oder einer Falschinformation führt. Bei einer Verzerrung des Ausgangssignals kann es dazu zu einer zeitlichen Verzögerung kommen, bis der vorhandene Signalpegel von dem Mikrokontroller erkannt wird. Diese zeitliche Verzögerung kann zu einer späten Informationsübermittlung führen.
Wird die Methode des Spannungsteilers genauer betrachtet, gibt es bei dieser Methoden deutliche Vorteile in der Einfachheit der Entwicklung und in den geringen Kosten. Die negativen Auswirkungen auf die Signalqualität und Geschwindigkeit sind jedoch erheblich. Diese Methode eignet sich daher besser zur Reduzierung von Spannungssignalen, ist jedoch aufgrund der geringen Präzision für digitale Signale nicht ideal und wird somit für diese Anwendung nicht gewählt.
Die Methoden zwei und drei bieten beide eine hohe Signalqualität und eine hohe Geschwindigkeit bei der Signalreduzierung. Der größte Unterschied der Methoden liegt im Bereich des Entwicklungsaufwands und der Kosten. Das Pegelwandlungs-IC benötigt nur wenig Entwicklungsaufwand, jedoch sind die aufzubringenden Bauteilkosten höher anzusetzen. Die MOSFET-basierte Methode benötigt einen höheren Entwicklungsaufwand, aber die Kosten sind niedriger. Jedoch muss bei dieser Methode sehr genau auf die Auslegung der Pull-Up-Widerstände geachtet und zusätzliche Schutzmechanismen müssen extra in die Schaltung integriert werden. Dies steigert unter Umständen wiederum die Kosten. Aus diesem Grund wird das Pegelwandlungs-IC gewählt. Dieses IC ist genau auf diese Anwendungen optimiert und besitzt somit die meisten Vorteile.

4.1.3 Umsetzung weiterer Überwachungsmöglichkeiten auf der Mikrokontrollerplatine

Innerhalb von diesem Kapitel werden weitere mögliche Überwachungsmethoden, welche auf der Mikrokontrollerplatine zu implementieren sind, beschrieben. Hierbei wird unter anderem die Umsetzung einer Überwachung der Umgebungstemperatur und eine weitere Beobachtung der Magnetspannungen diskutiert. Auch wird die Überwachung von verschiedenen Spannungsversorgungen konzeptioniert.

Überwachung der Versorgungsspannungen der Hot-Swap-Platine

Innerhalb von diesem Kapitel wird die Überwachung von verschiedenen Versorgungsspannungen der Platinen diskutiert. Zu überwachen sind hier unter anderem die 12 V Versorgungsspannung der Mikrokontrollerplatine und die 5 V VCC-Spannung der ICs der Hot-Swap-Platine. Beide Spannungen sind für einen korrekten Betrieb der Hot-Swap-Platine und der Mikrokontrollerplatine essentiell.
Die 12 V Spannung versorgt die Mikrokontrollerplatine mit Spannung. Sollte diese Spannung einen falschen Wert oder starke Schwankungen aufweisen, wird die Funktion der Mikrokontrollerplatine stark beeinträchtigt beziehungsweise es erfolgt ein Ausfall der Platine. Um dies zu vermeiden, ist somit diese Spannung mithilfe eines ADUs des Mikrokontrollers zu überwachen.
Die 5 V VCC-Spannung versorgt verschiedene ICs der Hot-Swap-Platine mit Spannung. Eine inkorrekte Spannung führt unter Umständen dazu, dass die ICs nicht korrekt arbeiten oder zerstört werden. In diesen Fällen sind die ermittelten Daten und die Funktion der Platine unbrauchbar. Um falsche Rückschlüsse durch fehlerhafte Daten zu vermeiden, ist diese Spannungsversorgung essentiell für den Überwachungsbetrieb. Aus diesem Grund ist diese Spannung ebenfalls mithilfe eines ADUs des Mikrokontrollers zu überwachen.
Um die Spannungen auf einen für den Mikrokontroller passenden Wert zu reduzieren, werden Spannungsteiler verwendet. Diese sind ausreichend, da keine schnellen Signaländerungen, wie bei einem digitalen Signal erwartet werden, und keine galvanische Trennung von Nöten ist.

Temperaturüberwachung

In diesem Kapitel wird eine Temperaturüberwachung der Hot-Swap-Platine und der Mikrokontrollerplatine konzeptioniert. Bei einer zu hohen Umgebungstemperatur können einige ICs und verwendete Bauteile auf der Platine eine ungenaue Funktion aufweisen oder Schaden nehmen. Ebenso kann durch die Überwachung der Umgebungstemperatur indirekt die korrekte Funktion des Lüftersystems verifiziert werden. Bei einem Ausfall oder einer Fehlfunktion des Systems können möglicherweise durch das übergeordneten System steigende Temperaturwerte festgestellt werden.

Im folgenden werden einige für dieses Projekt wichtige Auswahlkriterien eines Temperatursensors aufgeführt.
Zu dem jetzigen Entwicklungszeitpunkt ist die vorliegende Temperatur an dem Einsatzort der Platinen nicht bekannt. Aus diesem Grund sollen die Temperatursensoren einen möglichst großen Bereich der Temperaturskala abdecken. Es ist jedoch davon auszugehen, dass der Einsatzort der Platinen im Inneren liegt. Aus diesem Grund ist der positive Temperaturbereich umfassender abzudecken, als der negative Temperaturbereich.
Um die vorliegende Temperatur mit einem möglichst genauen Wert feststellen zu können, muss eine hohe Genauigkeit gewählt werden. Eine zu hohe Abweichung der Messdaten von dem Sollwert kann zu fehlerhaften Daten führen.
Des Weiteren muss ein Kommunikationsprotokoll zur Übertragung der Daten ausgewählt werden. Mögliche Optionen sind hier die Übertragung mittels One-Wire, I^2C oder SPI. Im folgenden werden die Methoden kurz vorgestellt und deren jeweiligen Vor- und Nachteile gegenübergestellt. Die Tabelle 4.3 vergleicht die Methoden hinsichtlich allgemeiner Fakten.

Tabelle 4.3: Vergleich von One-Wire, I^2C und SPI

	One-Wire	I^2C	SPI
Anzahl der Drähte	1 (plus Masse)	2	mindestens 4
Geschwindigkeit	Standard: 9.6 kBits/s Overdrive: 125 kBits/s [3]	Standard: 100 kBits/s High-Speed: 3.4 MBits/s [10]	Bis zu 60 Mbits/s [10]
Maximale Anzahl an Geräten	Begrenzt auf 255 [41]	7 Bit: 127 10 Bit: 1023	Von Anzahl an SS-Pins abhängig
Master-Slave-Model	Ja	Ja	Ja
Datenrichtung	Halbduplex	Halbdplex	Vollduplex
Synchronisation	asynchron	synchron	synchron
Komplexität	gering	moderat (benötigt Pull-Up-Widerstände)	moderat
Fehlersicherheit	niedrig	moderat Überprüfung durch ACK-Bit	niedrig
Implementierungs-kosten	sehr niedrig	niedrig	etwas höher durch mehrere Leitungen

One-Wire ist ein serielles Kommunikationsprotokoll. Hierbei wird für die Datenkommunikation nur eine Datenleitung benötigt. Zusätzlich zur Datenleitung wird eine gemeinsame Masseverbindung vorausgesetzt. Somit erfolgt die Datenübertragung asynchron. Dieses Kommunikationsprotokoll unterstützt zudem das Master-Slave-Model [15]. Der Vorteil dieses Kommunikationsmodels liegt in dem geringen aufzubringenden Arbeitsaufwand bei der Implementierung. Ebenso werden nur geringe Materialkosten erwartet, wodurch diese Methode als kostengünstig einzustufen ist. Nachteilig zu betrachten ist die sehr geringe Übertragungsgeschwindigkeit der Daten. Dies ermöglicht einen nur langsamen Datenaustausch zwischen Temperatursensor und Mikrokontroller [15]. Ein weiterer Nachteil besteht darin, dass kein Fehlerüberwachungssystem vorhanden ist. Daher lässt sich nicht feststellen, ob die empfangenen Daten korrekt sind.
I^2C ist ein serielles und synchrones Kommunikationsprotokoll. Die Datenübertragung erfolgt mithilfe einer Datenleitung Serial Data (SDA) und einer Taktleitung Serial Clock (SCL). Es wird das Master-Slave-Modell unterstützt. Jeder Slave erhält dabei eine 7- oder 10-Bit Adresse. Über diese Adresse ist es dem Master möglich einen bestimmten Slave anzusprechen. Bei einer 7-Bit Adressierung können bis zu 127 Slaves mit einem Master verbunden werden [7]. Ein Vorteil der Verwendung von I^2C ist, dass mehrere Geräte auf der selben Leitung angeschlossen werden können. Somit eignet sich I^2C gut für Anwendungen, wo viele Geräte mit einer minimalen Verkabelung verbunden werden müssen. Ebenso wird eine einfache Erweiterung um neue Slaves erreicht, da diese nur an die entsprechenden SDA und SCL Leitung angebunden werden müssen. Es können moderate Übertragungsgeschwindigkeiten erreicht werden, welche meist für die Erfassung von Sensordaten ausreichend sind. Durch die geringe Anzahl an Leitungen und die niedrigen bis moderaten Übertragungsgeschwindigkeiten tritt in der Regel ein geringerer Stromverbrauch auf. Des Weiteren wird bei der Kommunikation zwischen Master und Slave jedes empfangene Byte vom Empfänger mit einem Acknowledge Bit (ACK-Bit) bestätigt. Durch das Ausbleiben dieses Bits können eventuelle Fehler aufgedeckt werden. Nachteilig zu betrachten ist, dass nur eine Halbduplex-Kommunikation möglich ist. Es kann somit nur gesendet oder empfangen werden. Beides gleichzeitig ist nicht möglich [7].
SPI ist ebenfalls ein synchrones und serielles Kommunikationsprotokoll. Hierbei werden zwischen einem Master und einem Slave besonders hohe Übertragungsgeschwindigkeiten erreicht. Für die Übertragung werden mindestens vier Leitungen benötigt. Die erste Leitung ist die Taktleitung Serial Clock (SCK), die zweite Leitung dient zur Datenübertragung von dem Master zum Slave (MOSI), die dritte Leitung stellt die Datenverbindung vom Slave zum Master her (MISO) und die vierte Leitung dient zur Auswahl des Slaves,

mit dem aktuell kommuniziert werden soll (SS). Ein Vorteil dieses Protokolls ist, dass eine Vollduplex-Kommunikation ermöglicht wird. Hier können Daten gleichzeitig gesendet und empfangen werden. Ebenso können sehr hohe Datenraten erreicht werden [7]. Bei einer korrekten Verdrahtung wird eine geringe Fehleranfälligkeit erreicht. Nachteilig zu betrachten ist der deutlich höhere Implementierungsaufwand und die komplexere Verdrahtung. Da jeder Slave eine eigene Anbindung an den Master benötigt, wird unter Umständen eine hohe Anzahl an anzuschließenden Pins benötigt oder die Anzahl der Slaves ist stark begrenzt. Dies erhöht zusätzlich den Aufwand, wenn das System um zusätzliche Slaves erweitert werden soll. Ebenso ist im Vergleich zu I^2C keine Fehlerüberprüfung implementiert. Dies muss bei Bedarf durch eine geeignete externe Logik durchgeführt werden [7]. In der Tabelle 4.3 werden die drei Übertragungsarten miteinander auf allgemeine Fakten verglichen.

Nach Einbezug der gegebenen technischen Daten, sowie der Vor- und Nachteile wird im folgenden nun ein passendes Kommunikationsprotokoll ausgewählt. In diesem Projekt wird I^2C ausgewählt, da die moderaten Übertragungsgeschwindigkeiten für die Übertragung der Umgebungstemperatur ausreichend sind. Zudem sollen mehrere Sensoren implementiert werden, was bei SPI einen deutlich höheren Aufwand verursachen würde. Ein weiterer Vorteil von I^2C ist das Vorhandensein einer Fehlerüberwachung, was diese Methode im Vergleich zu anderen Methoden positiv hervorhebt.
All die genannten Aspekte sind bei der Auswahl des Temperatursensors zu berücksichtigen.

Signalüberwachung mit Potentialtrennung

Wie bereits beschrieben, ist die Überwachung der Magnetspannungen essentiell für den Beschleunigerbetrieb. Von der Hot-Swap-Platine wird bereits ein digitales Signal zur Überwachung der Magnetspannung an die Mikrokontrollerplatine übermittelt. Um die Informationen dieses Signals verifizieren zu können, ist es sinnvoll eine zweite Überwachungsmöglichkeit der Magnetspannungen auf der Mikrokontrollerplatine zu implementieren. Dies soll Fehler innerhalb der Deutung der aktuellen Magnetspannung verhindern. Hierbei ist es sinnvoll die Magnetspannungen mithilfe eines ADUs des Mikrokontrollers auszuwerten und anschließend mit den digitalen Detektionssignalen zu vergleichen. Das Signal wird dann sowohl durch eine Hardwareschaltung als auch durch die Software separat voneinander überwacht. Um die Magnetspannung an den Mikrokontroller anlegen zu

können, muss die galvanische Trennung der Potentiale der Magnetspannungen und des Mikrokontrollers beachtet werden. Innerhalb von diesem Kapitel werden somit verschiedene Möglichkeiten aufgeführt, mit denen eine Potentialtrennung der Signale ermöglicht wird. Hierbei ist eine hohe Signalintegrität und eine geringes Rauschen wichtig, da die Signale möglichst genau durch den ADU des Mikrokontrollers ausgewertet werden sollen. Es folgt nun die Vorstellung von drei verschiedenen möglichen Umsetzungsmethoden.

Zuerst wird der Trennverstärker, auch Isolationsverstärker genannt, vorgestellt. Es handelt sich hierbei um einen Verstärker, der zwischen Eingangs- und Ausgangsschaltkreis galvanisch isoliert ist. Teilweise wird ebenfalls die Spannungsversorgung des Bauteils galvanisch isoliert übermittelt. Die Übertragung des Signals kann durch verschiedene Kopplungstechniken erfolgen. Es wird zwischen optischen, kapazitiven und magnetischen Techniken unterschieden. Der Eingang des Verstärkers verarbeitet das analoge Signal und überträgt es anschließend durch eine Isolationseinheit. Diese Isolationseinheit stellt die physikalische Sicherung sicher und beinhaltet eine der Kopplungstechniken. Am Ausgang wird das Signal zunächst verstärkt und anschließend ausgegeben [47].
Ein wesentlicher Vorteil des Trennverstärkers liegt in der optimierten internen Schaltung des ICs, die eine galvanische Trennung des Eingangs- und des Ausgangssignals ermöglicht. Diese galvanische Trennung schützt vor sogenannten Masseschleifen und die damit verbundenen Störungen und Signalverzerrungen [37]. Somit trägt der Trennverstärker zu einer stabilen und unverfälschten Signalübertragung mit einer hohen Linearität bei [23]. Dies führt dazu, dass Trennverstärker besonders in Anwendungen, wo eine hohe Signalintegrität wichtig ist, verwendet werden [4]. Ein weiterer Vorteil stellt der Schutz gegenüber von EMI und Hochspannungsereignissen mit hohen Spannungsspitzen dar [37]. Der Trennverstärker verhindert somit Beschädigungen an empfindlichen nachfolgenden elektronischen Bauteilen. Zudem wird das Übertragen von bipolaren Signalen, welche sowohl positive als auch negative Spannungswerte aufweisen, ermöglicht [37]. Auch die kompakte Bauform des ICs stellt einen Vorteil dar. Diese Bauweise erleichtert die Implementierung und minimiert den benötigten Platzbedarf auf der Platine.
Nachteilig zu betrachten sind die entstehenden Bauteilkosten, da die ICs teurer als andere Methoden zur galvanischen Trennung sind [4].

Ein alternatives Verfahren stellt die Verwendung eines Optokopplers dar. Bei einem Optokoppler wird ein elektrisches Signal mithilfe von Licht übertragen, um eine galvanische Trennung zwischen Eingang und Ausgang herzustellen. Der Eingang des Optokopplers enthält dabei eine LED, die durch das angelegte elektrische Signal angesteuert wird.

Ist das elektrische Signal stark genug, emittiert die LED Licht, welches durch einen lichtempfindlichen Empfänger, wie einen Phototransistor oder eine Photodiode, auf der Ausgangsseite des Optokopplers erfasst wird. Anschließend erzeugt der Empfänger ein elektrisches Signal [19].
Der größte Vorteil dieser Methode liegt in der vollständigen galvanischen Trennung zwischen Eingangs- und Ausgangskreis. Somit ist auch bei dieser Anwendung ein Schutz gegenüber von Masseschleifen vorhanden [6]. Ein weiterer Vorteil liegt in der einfachen Implementierung der Optokoppler. Hier ist meist nur das IC selbst und keine weiteren Bauteile notwendig [41]. Des Weiteren bieten Optokoppler einen Schutz gegenüber von hohen Spannungen und Spannungsspitzen, da die optische Trennung keine elektrische Verbindung zwischen den zwei Schaltkreisen benötigt [6].
Nachteilig zu betrachten ist die begrenzte Linearität des Optokopplers, welche zu Verzerrungen im Ausgangssignal führen. Hervorgerufen wird dies unter anderem durch die Nichtlinearität des Photodetektors und der LED. Die LED erzeugt in Abhängigkeit des Stroms, der durch sie hindurchfließt, Licht. Diese Beziehung ist nicht vollkommen linear. Ebenfalls der Photodetektor weist eine nichtlineare Kennlinie auf. Auch kann die Übertragungscharakteristik von Optokopplern durch die Temperatur beeinflusst werden. Dies kann zu weiteren Verzerrungen in der Signalübertragung führen [19]. Die Übertragung des bipolaren Signals stellt ebenfalls einen Nachteil dar. Die verwendete LED innerhalb des Optokopplers benötigt einen definierten Vorwärtsstrom, um Licht zu emittieren. Bei negativen Spannungswerten wird diese LED in Sperrrichtung betrieben, was dazu führt, dass kein Licht durch diese LED emittiert wird [5]. Um trotzdem positive als auch negative Signale übertragen zu können, sind Schaltungserweiterungen vorzunehmen. Hier kann beispielsweise ein Level shifting unternommen werden, wo die negativen Spannungswerte in einen positiven Bereich verschoben werden oder es werden zwei Optokoppler eingesetzt. Ebenfalls zu erwähnen ist, dass die Sendedioden des Optokopplers im Alterungsprozess an Leistung und Effizienz verlieren. Dies führt zu einem Herabsetzen des Übertragungsverhältnisses [46].

Das dritte Verfahren beinhaltet eine Kombination aus ADU und DAU. Hierbei wird das analoge Spannungssignal zunächst mithilfe eines ADUs in ein digitales Signal umgewandelt. Anschließend kann dieses digitale Signal beispielsweise über einen digitalen Isolator übertragen werden. Der digitale Isolator stellt dabei die galvanische Trennung zwischen den Stromkreisen sicher. Nach der galvanischen Trennung wird das Signal auf der Empfängerseite durch einen ADU wieder in ein analoges Spannungssignal umgewandelt.

Ein Vorteil ist, dass durch den Einsatz des digitalen Isolators eine komplette galvanische Trennung des Eingangs- und des Ausgangskreises erreicht wird [13]. Durch die Digitalisierung des Signals können Methoden der digitalen Signalverarbeitung angewendet werden, um das Signal weiterzuverarbeiten oder zu optimieren. Beispielsweise können digitale Filter angewendet werden. Somit ist eine hohe Flexibilität in der Signalverarbeitung gegeben.
Die Signalintegrität ist stark von der Präzision des gewählten ADUs und des DAUs abhängig. Je höher die Auflösung ist, desto genauer kann das Eingangssignal abgebildet werden. Ebenso wird durch eine höhere Auflösung der Quantisierungsfehler reduziert [43]. Hieraus kann der Nachteil abgeleitet werden, dass die Bauteile sehr sorgfältig ausgewählt werden müssen. Dementsprechend wird ein höherer Implementierungsaufwand erwartet. Ebenso kommt es durch den Umwandlungsprozess vom anlogen in ein digitales Signal und anschließend wieder zurück zu einem erhöhten Zeitbedarf, welcher eine Verzögerung in der Datenbereitstellung bedeutet. Durch das Verwenden von mehreren Komponenten ist ebenfalls der Energieverbrauch höher, als bei den anderen vorgestellten Optionen.

In der Tabelle 4.4 werden nochmals alle drei Verfahren gegenübergestellt.

Tabelle 4.4: Vergleich verschiedener Methoden zur Potentialtrennung

	Trennverstärker	Optokoppler	ADU-DAU-Verfahren
Galvanische Trennung	sehr gut	sehr gut	sehr gut
Präzision und Linearität	hoch	eingeschränkt, Verzerrungen möglich	eingeschränkt, Quantisierungsfehler möglich
Latenz	sehr gering	mäßig	hoch
Komplexität	einfach	einfach	hoch
Kosten	hoch	niedrig	hoch viele Bauteile benötigt
Energieverbrauch	niedrig	niedrig	höher

Es folgt nun die Auswahl einer der beschriebenen Möglichkeiten. Mithilfe der galvanischen Trennung sind unter anderem die zwei Magnetspannungen zu überwachen. Es ist erforderlich, dass selbst geringe Abweichungen von dem Sollwert der Magnetspannung erkennbar sind. Aus diesem Grund ist bei der galvanischen Trennung ein besonderes Augenmerk auf die Signalintegrität zu legen. Dies bedeutet das möglichst kein Rauschen und

keine Verzerrungen durch das Bauteil hervorgerufen werden sollen. Hierbei weist der Optokoppler deutliche Nachteile im Vergleich zu dem Trennverstärker und dem ADU-DAU-Verfahren auf. Aus diesem Grund wird der Optokoppler nicht in für diese Anwendung ausgewählt. Im Vergleich zwischen dem Trennverstärker und dem ADU-DAU-Verfahren besteht ein wesentlicher Unterschied im Implementierungsaufwand. Während bei dem ADU-DAU-Verfahren viele verschiedene aufeinander abgestimmte Komponenten benötigt werden, wird bei dem Trennverstärker nur ein kompaktes Bauteil benötigt. Gerade im Hinblick auf den Platzverbrauch auf der Platine und den aufzubringenden Kosten bei der Bauteilbestellung weist somit der Trennverstärker Vorzüge auf. Auch entsteht bei dem ADU-DAU-Verfahren durch das Verwenden mehrerer Bauteile eine gewisse Latenz. Diese Latenz führt zu einer Verzögerung innerhalb der Datenübermittlung. Aus diesen Gründen wird sich in diesem Projekt für den Trennverstärker als zu implementierendes Bauteil entschieden.

Auswertung des PWM Signals

Um die Daten der zwei verschiedenen Methoden zur Überwachung der Magnetspannung miteinander vergleichen zu können, muss für die softwarebasierte Auswertung ebenfalls ein Schwellwert einstellbar sein. Die Größe des Schwellwerts wird mithilfe des PWM Signals von der Hot-Swap-Platine an die Mikrokontrollerplatine übermittelt. Im folgenden wird nun die Auswertung des Signals konzeptioniert.
Hierbei werden zwei mögliche Ansätze vorgestellt. Der erste Ansatz wertet das PWM-Signal auf dem Mikrocontroller softwareseitig anhand seines Duty-Cycles aus. Beim zweiten Ansatz wird das PWM-Signal zunächst durch einen Tiefpassfilter gemittelt und anschließend über einen ADU an den Mikrocontroller weitergegeben.

Zunächst soll die softwarebasierende Methode vorgestellt werden. Bei dieser Methode wird ein Timer so eingestellt, dass bei jeder Flanke des PWM Signals ein Interrupt ausgelöst wird. Dies kann durch die Aktivierung des Input-Capture-Modus erfolgen [17]. Bei einer steigenden Flanke wird der aktuelle Zählerstand erfasst und gespeichert. Bei der nächsten fallenden Flanke wird erneut der Zählerstand erfasst. Die Differenz zwischen diesen beiden Werten entspricht der Dauer des High-Pegels. Der Duty Cycle wird berechnet, indem die Dauer des High-Pegels durch die Gesamtperiode des Signals geteilt wird.
Mithilfe dieser Methode wird eine hohe Präzision erreicht, da durch die direkte Messung

der Zeiten mithilfe eines Timers sehr genaue Ergebnisse erzielt werden können. Ebenso ist diese Methode an verschiedene PWM-Frequenzen und PWM-Signale ohne zusätzliche Hardware anpassbar. Dadurch entsteht eine hohe Flexibilität. Ein weiterer Vorteil stellt die effiziente Ressourcennutzung des Mikrokontrollers dar. Es wird der bereits vorhandene Timer des Mikrokontrollers benutzt, sodass keine weiteren Bauelemente zur Signalauswertung auf der Platine implementiert werden müssen.
Ein bedeutender Nachteil ist jedoch die hohe CPU-Auslastung. Bei der Erfassung der Interrupts und der Berechnung der Zeiten ist die CPU erforderlich. Je nach Frequenz des PWM-Signals kann hier eine hohe Belastung entstehen, welche die Ausführung von anderen CPU-Aufgaben blockiert. Des Weiteren wird eine sorgfältige Implementierung in der Software benötigt, um die genaue Messung sicherzustellen.

Es folgt nun die Vorstellung des hardwarebasierten Ansatzes. Hierbei wird ein Tiefpassfilter implementiert, welcher den Mittelwert des PWM-Signals bestimmt. Der resultierende Spannungswert kann anschließend mit einem ADU ausgelesen werden [8].
Vorteilhaft bei dieser Methode ist die geringe benötigte CPU-Last. Die CPU muss nur den ADU-Wert auslesen und im Gegensatz zur softwarebasierten Methode keine großen Berechnungen durchführen. Wenn die PWM-Frequenz bekannt ist, kann der Tiefpass-Filter sehr einfach entworfen werden. Dafür wird lediglich ein Widerstand und ein Kondensator benötigt [8]. Die Änderungen innerhalb des Duty-Cycles können kontinuierlich und ohne Latenz überwacht werden.
Ein Nachteil ist, dass der Tiefpass-Filter auf die Frequenz des PWM-Signals abgestimmt werden muss. Dies kann bei variablen Frequenzen problematisch sein. Die Genauigkeit, mit der der Mittelwert bestimmt wird, hängt von der Qualität des Filters und der Auflösung des ADUs ab. Des Weiteren werden zusätzliche Hardwarebauteile benötigt, welche auf der Platine implementiert werden müssen [8].

Für dieses Projekt wird sich für die hardwarebasierte Methode entschieden. Der ausschlaggebende Grund stellt die CPU-Auslastung dar. Diese ist in der softwarebasierten Methode sehr hoch, was dazu führen kann, dass kritische Signale mit Verzögerungen oder sogar gar nicht verarbeitet werden. Das PWM-Signal wird dauerhaft gesendet, sodass dauerhaft CPU-Kapazität für das Verarbeiten der Interrupts und für das Bestimmen der Zeiten benötigt wird. Dies ist nicht mit den anderen Anforderungen an die Software vereinbar.

4.1.4 Anschlussplan an den Mikrokontroller

Innerhalb von diesem Unterkapitel soll ein Anschlussplan der in den vorherigen Unterkapiteln vorgestellten Signale erstellt werden. Hierfür werden zunächst nochmals die benötigten Schnittstellen und die Signale, welche über die Schnittstellen an den Mikrokontroller weitergegebene werden, aufgeführt. Das Interrupt-Signal für den Temperatursensor und das enable-Signal für die Pegelwandlungs-ICs ergeben sich aus den jeweils gewählten Bauteilen und werden innerhalb der Implementierung genauer beschrieben.

1. GPIO-Pins
    a. Vier Detection-Signale
    b. Zwei Dump-Signale
    c. Vier Matrix-Signale
    d. Ein Interrupt-Signal des Temperatursensors
    e. Ein PWM-Signal
    f. Ein Enable-Signal für die Pegelwandlungs-ICs
    e. Vier Adress-Signale
2. I^2C-Verbindung zum Auslesen der Temperaturdaten
3. UART-Verbindung zur Hot-Swap-Platine
4. ADUs
    a. Magnetspannung M1
    b. Magnetspannung M2
    c. 12V- Spannung der Spannungsquelle der Magnete
    d. 12V-Versorgungsspannung der Mikrokontrollerplatine
    e. 5V VCC-Spannung
    f. PWM-Signal

Zusammengefasst werden somit 17 GPIO-Pins und sechs ADU Anbindungen an den Mikrokontroller benötigt. Des Weiteren benötigt I^2C zwei Pins für die SDA- und SCL-Leitung, sowie UART zwei Pins für die RX- und die TX-Anbindung. Der gesamte Anschlussplan mit der genauen Pinbelegung kann in dem Anhang unter dem Kapitel A.2 angesehen werden.

4.2 Konzeptionierung der Software

In diesem Kapitel erfolgt die Konzeptionierung der Software. Zuerst wird die Priorisierung der Signale festgelegt. Anschließend erfolgt die Konzeptionierung des Empfangs der verschiedenen Signale und die Auswahl eines Fehlererkennungsverfahrens für die UART-Übertragung. Des Weiteren werden verschiedene Möglichkeiten, den über UART zu übertragenden Frame zu gestalten, diskutiert. Zuletzt werden einige Timinganforderungen konzeptioniert.

4.2.1 Priorisierung der Signale

Innerhalb von diesem Kapitel erfolgt die Priorisierung der einzelnen zu überwachenden Signale. Hierbei müssen kritische Signale höher priorisiert werden, um eine zeitnahe Abfrage und Verarbeitung der Daten sicherzustellen. Das Kapitel beginnt mit den am höchsten priorisierten Signalen. Anschließend werden die weiteren Signale in absteigender Priorität beschrieben.
Einen besonderen Stellenwert nimmt die Überwachung der Magnetspannungen ein. Wie bereits mehrfach betont, ist ein störungsfreier Betrieb der Magnete von entscheidender Bedeutung für den Beschleunigerbetrieb. Daher werden die Signale zur Überwachung der Versorgungsspannungen der Magnete als besonders kritisch eingestuft. Aus diesem Grund besitzt die Überwachung der vier digitalen Detektionssignale sowie der Magnetspannungen an den ADUs höchste Priorität.
Die vier digitalen Dump-Signale weisen eine ähnlich hohe Priorität auf. Wenn eines dieser Signale ein High-Signal zeigt, deutet dies auf ein Problem mit der Spannungsversorgung der Magnete hin. Aus diesem Grund ist ein Pegelwechsel dieser Signale umgehend zu erkennen und dem übergeordneten System mitzuteilen.
Es folgt die Überwachung der verschiedenen Versorgungsspannungen innerhalb der Prioritätenreihenfolge. Sie dienen zur Überprüfung, ob das Überwachungssystem selbst funk-

tionsfähig ist. Andernfalls können die gesammelten Daten Fehlinformationen beinhalten, die zu einer falschen Reaktion durch das übergeordnete System führen können.
Etwas niedriger in der Priorität ist die Temperaturüberwachung einzuordnen. Eine abfallende Spannungsversorgung ist zunächst kritischer zu bewerten, da ihr Toleranzbereich kleiner ist und ein Funktionsausfall schneller eintritt als bei einer Temperaturänderung. Aus diesem Grund ist die Überwachung der Temperatur weniger kritisch als die Überwachung der Spannungen und wird daher unter der Spannungsüberwachung in der Priorität angeordnet.
Die Priorität des PWM-Signals ist geringfügig niedriger anzuordnen. Dieses Signal dient zur Festlegung eines neuen Schwellenwerts für die Überprüfung der Über- oder Unterschreitung der Grenzwerte der Magnetspannung. Die Grenzen der Schwellenwerte werden nur selten und langsam angepasst. Daher hat diese Änderung eine eher niedrige Priorität.
Es folgt die Überwachung der Matrix-Signale und der Adress-Signale. Beide Signale werden vor allem für ein Status-Update der verbauten Hardware und dem aktuellen Schaltungszustand der Anlage verwendet. Aus diesem Grund ist die Priorität von diesen Signalen am geringsten anzusehen.

4.2.2 Empfang und Auswertung der eingehenden Signale

Dieses Kapitel dient dazu verschiedene Mechanismen zur Verarbeitung der empfangenen Signale vorzustellen. Es stehen die Optionen Abfrage der Daten durch Polling, Empfang der Daten mittels Interrupts sowie Übertragung der Daten über DMA zur Verfügung. Zuerst sollen diese Möglichkeiten vorgestellt und miteinander verglichen werden. Anschließend wird eine passende Methode für die verwendeten Schnittstellen ausgewählt.

Vergleich Polling, Interrupt und DMA

Beim Polling werden die Daten regelmäßig in einer Schleife durch die CPU abgefragt. Dies kann bei häufigem Abfragen der Daten allem sehr ineffizient sein, da die CPU durch die Abfrage voll ausgelastet ist und keine parallelen Aufgaben ausführen kann. Positiv zu betrachten ist hingegen die einfache Implementierung. Ebenso kann das Zeitintervall der Abfrage manuell voreingestellt werden, sodass ein vorhersehbares Zeitverhalten entsteht. Nachteilig an diesen festgelegten Abfrageintervallen ist, dass spontan auftretende Änderungen, die zwischen zwei Abtastintervallen vorkommen, unter Umständen nicht erfasst werden können [28].

Bei dem Verwenden von Interrupts wird ein Ereignis, beispielsweise die Pegeländerung an einem GPIO-Pin, als Hardware-Interrupt registriert. Dies führt dazu, dass die CPU sofort reagiert und eine Interrupt service routine (ISR) ausgeführt wird. Somit kann eine Änderung an dem Dateneingang sofort erkannt und verarbeitet werden. Nach Beendigung der ISR wird das unterbrochene Programm fortgeführt. In der Zeit zwischen den Interrupts kann die CPU andere Aufgaben erfüllen. Somit ist eine Parallelität der Aufgaben möglich. Bei der Verwendung von mehreren verschiedenen Interrupts können verschiedene Prioritäten vergeben werden [28]. Nachteilig zu nennen, ist die komplexere Implementierung im Vergleich zum Polling. Auch kann beim Auftreten von mehreren Interrupts zur gleichen Zeit eine Verzögerung bei der Datenerfassung entstehen, da immer zuerst der höher priorisierte oder zuerst ausgelöste Interrupt verarbeitet wird.
Mithilfe des DMA-Controllers können Daten direkt von den Peripheriegeräten in den Speicher geschrieben werden, ohne dass die CPU eingreifen muss. Somit geschieht diese Datenerfassung komplett unabhängig von der CPU. Die CPU kann in dieser Zeit andere Aufgaben erfüllen [28]. Nachteilig ist, dass die Implementierung sehr komplex ist. Auch gibt es nur eine begrenzte Anzahl von DMA-Kanälen. Der DMA reagiert ebenfalls nicht auf schnelle Veränderungen der Eingangsdaten. Somit ist der DMA-Controller für kontinuierliche Datenströme geeignet, welche effizient verarbeitet werden müssen. In der Tabelle 4.5 werden alle drei Methoden miteinander verglichen.

Tabelle 4.5: Vergleich Polling, Interrupt und DMA

	Polling	Interrupt	DMA
Reaktionszeit	Langsam abhängig vom Intervall	Sehr schnell	Schnell aber nicht „Echtzeit“-fähig für Einzelereignisse
Effizienz	Ineffizient benötigt CPU-Zeit	Effizient CPU arbeitet nur bei Bedarf	Sehr effizient CPU kaum beansprucht
Implementierung	Einfach	Mittel	Hoch
Parallelität	Keine CPU voll ausgelastet	Unterstützt parallele Aufgaben	Vollständig parallelisierbar CPU und DMA arbeiten unabhängig
Beständigkeit	Änderungen können verpasst werden	Sehr zuverlässig reagiert sofort auf Änderungen	Sehr beständig für kontinuierliche Datenübertragungen

GPIO-Pins

Die Datenerfassung der GPIO-Pins soll in gleichmäßigen und regelmäßigen Intervallen erfolgen, damit die aktuellen Daten an die Hot-Swap-Platine übertragen werden können. Somit ist eine Datenerfassung via Polling sinnvoll. Hier kann ein festes Zeitintervall vorgegeben werden, indem die Daten immer neu abgefragt werden. Um zu verhindern, dass Datenänderungen zwischen den Abfrageintervallen verloren gehen, ist es zudem zielführend besonders die Pins mit den kritischen Signalen zusätzlich über Interrupts zu überwachen. Somit kann gewährleistet werden, dass immer die aktuellsten Daten über UART versendet werden und keine Datenänderung verloren geht. Eine Erfassung der Daten mithilfe von DMA ist nicht möglich, da der verwendete Mikrokontroller die GPIO-Pins nicht als Eingangsperipherie für den DMA-Controller unterstützt.

I^2C

Über das Kommunikationsprotokoll I^2C sind die aktuellen Temperaturdaten der Temperatursensoren abgefragt werden. Hier ist ebenfalls eine kontinuierliche Abfrage der Daten erwünscht, damit die aktuellsten Temperaturdaten übermittelt werden können.
Dabei muss sowohl beim Einsatz von Polling als auch beim Einsatz von Interrupts ein Befehl zum Abrufen der Daten aufgerufen werden. Der entscheidende Unterschied liegt jedoch darin, dass beim Polling der I^2C-Bus kontinuierlich durch die CPU abgefragt werden muss, während beim Interrupt eine Callback-Funktion ausgelöst wird, sobald die Daten vom Slave gesendet wurden. In der Zeit zwischen dem Start des Interrupts und dem Aufrufen der Callback-Funktion kann die CPU anderen Aufgaben nachgehen. Aus diesem Grund wird die Interrupt-Methode vor der Polling-Methode präferiert. Da kein kontinuierlicher Datenfluss vorhanden ist, sondern nur 16 Bit an Temperaturwerten pro Abfrage übertragen werden, ist von der Verwendung eines DMA-Controllers abzusehen. Dementsprechend wird sich bei dem Empfang der Temperaturdaten für die Interrupt-Methode entschieden.

ADUs

Die Datenabfrage von den ADUs soll ebenfalls kontinuierlich in einem festen Zeitintervall erfolgen. Es werden insgesamt sechs verschiedene Signale, welche über sechs Kanäle eines ADUs übertragen werden, überwacht.

Polling weist denselben Nachteil wie die I^2C-Übertragung auf. Die CPU wird während der gesamten Datenabfrage kontinuierlich beansprucht. Aus diesem Grund wird Polling im weiteren Vergleich nicht weiter betrachtet. Es wird sich somit zwischen Interrupts oder DMA-Controller entschieden. Um die sechs ADU-Kanäle gleichzeitig auszulesen, werden alle zugehörigen Interrupts zeitgleich aktiviert. Dies birgt das Risiko, dass ein Interrupt übersehen wird, wodurch die aktuellsten Daten nicht ausgelesen werden. Zudem führt das gleichzeitige Auftreten der Interrupts zu einer gewissen Latenz beim Auslesen der Daten. Da sechs ADU Kanäle mit einer jeweiligen Auflösung von 12 Bits gleichzeitig abgefragt werden, entsteht ein Datenstrom von 72 Bits. Um die Effizienz zu steigern, ist der Einsatz eines DMA-Controllers sinnvoll. Dieser ermöglicht es, den Datenstrom direkt in einen Speicher zu schreiben, ohne die CPU zu belasten. Nach Abschluss der Übertragung wird eine Callback-Funktion aufgerufen, in der die Daten aus dem Speicher weiterverarbeitet werden können.

UART

Über das Kommunikationsprotokoll UART sind regelmäßig die aktuellen Daten an die Hot-Swap-Platine zu senden. Da eine Datenübertragung aus mehreren Datenpaketen besteht, ist ein großer Datenstrom gegeben. Bei der Übertragung durch Interrupts wird die CPU nach jedem übertragenden bzw. empfangenem Byte aktiviert, um die Daten zu verarbeiten. Dies kann bei hohem Datenaufkommen belastend sein, da die CPU kontinuierlich die anderen auszuführenden Aufgaben unterbrechen muss. Aus diesem Grund wird bei der UART-Übertagung der DMA-Controller verwendet. Hierbei werden die Daten dauerhaft durch den DMA-Controller gesendet und empfangen, ohne dass die CPU ihre Aufgaben unterbrechen muss. Es wird erst eine Callback-Routine ausgelöst, wenn alle Daten aus dem vorhergesehenen Speicherbereich gesendet wurden oder der vorhergesehene Speicherbereich mit neu empfangenen Daten voll ist.

4.2.3 Fehlererkennungsverfahren für die UART-Übertragung

In diesem Kapitel sollen verschiedene Verfahren, mit denen eine fehlerhafte Übertragung der Daten über UART festgestellt werden kann, vorgestellt werden. Hierfür werden die Verfahren Checksumme, Fletcher-Checksumme, Cyclic redundancy check (CRC)-Check und Hamming-Code betrachtet. In der Tabelle 4.6 werden die Methoden hinsichtlich einiger Aspekte verglichen.

Tabelle 4.6: Vergleich von Fehlererkennungsverfahren

	Checksumme	Fletcher-Checksumme	CRC-Check	Hamming-Code
Fehler-erkennung	Grundlegende Fehlererkennung (Einzelbitfehler)	Besser als einfache Checksumme (Einzel- und Mehrfachfehler)	Sehr gut (Einzel- und Mehrfachbitfehler)	Gut (Einzelbitfehlern)
Fehler-korrektur	Nein	Nein	Nein	Ja (Einzelbitfehler)
Komplexität	Sehr gering	Gering	Mittel	Mittel
Rechnungs-aufwand	Sehr gering	Gering bis Mittel	Gering bis Mittel	Gering
Effektivität bei UART	Einfach und schnell, aber anfällig für Fehler	Gute Erkennung von Einzel- und Mehrfachfehlern, effizient für UART	Sehr gut für einfache und robuste Fehlererkennung, effizient	Gut für fehleranfällige Übertragungen, ermöglicht Fehler-korrektur

Bei der Verwendung einer Checksumme werden alle Daten aufsummiert, um eine Summe zu erhalten. Diese Summe wird anschließend als Checksumme zusammen mit den Daten übertragen. Dadurch, dass nur eine einfache Addition der Daten ausgeführt werden muss, ist sowohl die Komplexität, als auch der Rechenaufwand mit sehr gering zu bewerten. Mithilfe von dieser Methode können Einzelbitfehler erkannt werden. Bei dem Auftreten von mehreren Fehlern oder dem Vertauschen von einzelnen Bits kann keine zuverlässige Fehlererkennung erwartet werden. Dies resultiert daraus, dass jeder Summand der Summe gleich behandelt wird. Sollten sich zwei Fehler gegeneinander aufheben, ist die Checksumme die Gleiche, obwohl bei der Übertragung zwei Fehler entstanden sind. Aus diesem Grund kann die Fehlererkennung von mehreren Fehlern nicht garantiert werden. Die Checksumme ist somit eine sehr einfache und schnelle Methode, welche für UART Verbindungen, die eine niedrige Wahrscheinlichkeit von Übertragungsfehlern aufweisen,

geeignet ist. Jedoch ist die Zuverlässigkeit der Fehlererkennung stark begrenzt [30].

Die Fletcher-Checksumme ist eine Verbesserung zu der eben vorgestellt Checksumme. Hierbei werden zwei Prüfsummen erstellt. Die erste Prüfsumme wird wie bei der Checksumme durch einfach Addition der zu übertragenden Daten bestimmt. Die zweite Checksumme wird durch die Addition der Ergebnisse der ersten Checksumme berechnet. Um die Summen möglichst klein zuhalten, wird anschließend eine Modulo-Operation durchgeführt. Beispielweise wird bei der Fletcher-16-Checksumme modulo 255 gerechnet. Durch diese Methode entsteht die Möglichkeit neben Einfachfehlern auch Mehrfachfehler bestimmen zu können. Dadurch, dass weiterhin einfache Additionen durchgeführt werden, ist die Komplexität und der Rechnungsaufwand weiterhin mit gering zu bewerten. Somit stellt diese Methode eine effiziente Möglichkeit zur Fehlererkennung dar, wo keine aufwändigere Implementierung gewünscht ist [30].

Als nächstes wird der CRC-Check vorgestellt. Der CRC-Check basiert auf der binären Division der zu übertragenden Daten durch ein vorher festgelegtes Generatorpolynom. Der Rest dieser Division stellt die CRC-Checksumme dar, die zusammen mit den Daten übermittelt wird. Auf der Empfangsseite wird die Division mit dem gleichen Polynom und den übertragenden Daten mit angehängter CRC-Checksumme durchgeführt. Sollte die Übertragung fehlerfrei durchgeführt worden sein, ist hier der Rest der Division Null. Andernfalls ist ein Fehler innerhalb der Datenübertragung aufgetreten. Es gibt verschiedene Arten des CRC-Checks, wobei der Name durch die Länge des Generatorpolynoms bestimmt wird. Beim CRC-8 ist das Polynom acht Bit lang, während beim CRC-16 das Polynom 16 Bit aufweist. Mithilfe eines längeren Generatorpolynoms können sowohl Einzelbitfehler als auch Mehrfachbitfehler sicherer erkannt werden. Jedoch erhöht sich die Komplexität und der Rechnungsaufwand mit einem größeren Generatorpolynom [40].

Zuletzt wird die Methode des Hamming-Codes vorgestellt. Mithilfe dieser Methode ist neben der Fehlererkennung auch die Fehlerkorrektur von Einzelbitfehlern möglich. Bei dieser Methode werden sogenannte Paritätsbits in die Datenbits eingefügt. Berechnet werden die Paritätsbits aus den zu sendenden Datenbits. Dabei werden alle Positionen der Daten, wo sich eine eins befindet, miteinander addiert. Die Anzahl der benötigten Paritätsbits bestimmt sich aus der Anzahl der zu übertragenden Bits. Bei 2^n Bits müssen n+1 Bits als Paritätsbits vorgesehen werden. Die Komplexität der Berechnung bestimmt sich somit aus der Anzahl der Datenbits. Die Paritätsbits sollten dabei sorgfältig berechnet werden. Der Rechenaufwand hingegen ist gering, da bei dieser Methode Bits addiert

werden. Jedoch wird bei der Addition der Übertrag nicht beachtet. Der besondere Vorteil liegt darin, dass bei dem Auftreten eines einzelnen Bitfehlers die Position des Fehlers auf der Empfangsseite berechnet werden kann. Somit ist eine nachträgliche Korrektur möglich. Jedoch ist es eine genau Fehlervorhersage bei Mehrfachfehlern, die direkt hintereinander auftreten, nicht garantiert [29].

Nachdem mehrere Methoden vorgestellt wurden, soll eine Methode ausgewählt werden. Da die UART-Verbindung die aktuellsten Messdaten überträgt, sollte die verwendete Methode eine hohe Zuverlässigkeit in der Übertragung bieten. Auch ist es sinnvoll, dass nicht nur Einzelbitfehler, sondern auch Mehrfachbitfehler erkannt werden können. Aus diesem Grund ist die Checksumme keine geeignete Methode für diese Anwendung. Auch der Hamming-Code ist in der Erkennung von Mehrfachbitfehlern nicht die zuverlässigste Methode. Es wird sich somit gegen den Hamming-Code entschieden, obwohl die Korrektur von Einzelbitfehlern vorteilhaft ist. Diese Methode sollte jedoch für die zukünftige Weiterentwicklung des Projekts in Betracht gezogen werden. Der wesentliche Vorteil des CRC-Checks gegenüber der Fletcher-Checksumme liegt in der flexiblen Anpassbarkeit der Zuverlässigkeit der Fehlererkennung durch die Wahl der Länge des Generatorpolynoms. Gleichzeitig kann der Rechenaufwand durch die Verwendung eines kürzeren Generatorpolynoms verringert werden. Aus diesen Gründen fällt die Wahl auf den CRC-Check.

4.2.4 UART Frame

Innerhalb von diesem Kapitel werden verschiedene Möglichkeiten zum Aufbau des UART-Frames miteinander verglichen. Dieser UART-Frame wird an die Hot-Swap-Platine gesendet und soll alle relevanten Daten enthalten. Hierbei ist die in dem Kapitel 4.2.1 festgelegte Priorität der Signale zu beachten.

Die erste Möglichkeit, um den Frame zu gestalten, liegt darin eine feste Struktur vorzugeben, die bei jeder Datenübermittlung eingehalten wird. Dabei werden alle zu versendenden Daten hintereinander verschickt. Die Vorteile dieser Methode liegen in der Einfachheit der Umsetzung und der geringen Fehleranfälligkeit bei der Interpretation der Daten. Da immer die gleichen Daten in der selben Reihenfolge geschickt werden, ist eine Fehlinterpretation relativ gering. Durch eventuelle Start- und Endbytes am Anfang und Ende der UART-Übertragung können Fehler durch verschobene Daten einfacher erkannt werden. Nachteilig zu nennen, ist die geringe Flexibilität beim Versenden der Daten. Soll-

te sich die Anzahl der Datenpunkte in der Zukunft ändern, ist eine manuelle Anpassung notwendig. Auch können keine Daten priorisiert werden, da alle im gleichen Zeitintervall versendet werden.

Die zweite Möglichkeit sieht es vor, zwei verschiedene Frames zu versenden. Die Frames können beispielsweise durch die Wahl eines unterschiedlichen Startbytes voneinander unterschieden werden. Der erste Frame kann dabei die priorisierten Daten enthalten. Der zweite Frame enthält anschließend die restlichen Daten, welche eine geringere Priorität aufweisen. Somit ermöglicht diese Methode die höher priorisierten Daten häufiger und unabhängig von den anderen Daten zu versenden. Dadurch wird eine hohe Flexibilität ermöglicht, bei der die Daten entsprechend ihrer Priorität behandelt werden können. Auch die Fehlertoleranz ist im Vergleich zur ersten Methode höher. Sollte ein Frame aus unterschiedlichen Gründen beim Senden verloren gehen, kann ein wichtiges Frame schneller erneut verschickt werden. Nachteilig ist der erhöhte Aufwand zu betrachten. Sowohl der Sender, als auch der Empfänger müssen anhand der Startbytes die Daten richtig interpretieren können. Außerdem wird mehr Overhead durch die Verwaltung von zwei Frame-Typen benötigt.

Die dritte Methode teilt den Frame in zwei Abschnitte ein. In der ersten Hälfte des Frames werden die Daten mit der höchsten Priorität, die Magnetspannungen, bei jeder Übertragung gesendet, sodass die Daten mit einer möglichst geringen Latenz immer aktuell übermittelt werden. Die zweite Hälfte des Frames dient der Übertragung weniger priorisierter Daten, wobei eine zyklische Reihenfolge angewendet wird. Um die eindeutige Zuordnung der übertragenen Informationen zu gewährleisten, wird jedem Datenpaket ein Identifier hinzugefügt, der angibt, welche Variable übertragen wird. Der Identifier 0x01 könnte beispielsweise signalisieren, dass die VCC-Spannung übermittelt wird. Durch diese Struktur wird eine klare Trennung zwischen priorisierten kritischen und weniger kritischen Daten erreicht. Die Methode stellt sicher, dass die wichtigsten Daten bei jeder Übertragung enthalten sind, während weniger dringliche Informationen durch den zyklischen Mechanismus regelmäßig gesendet werden. Die Identifier ermöglichen es, die empfangenen Datenpakete eindeutig zu interpretieren und den entsprechenden Variablen zuzuordnen. Nachteilig zu betrachten ist, dass der zusätzliche Identifier den Kommunikationsaufwand erhöht. Dies führt zu einem gesteigerten Overhead. Zudem kann es bei weniger priorisierten Daten durch die zyklische Übertragung zu Verzögerungen kommen. Darüber hinaus erhöht die Implementierung der zyklischen Reihenfolge und der Verwaltung der Identifier die Komplexität des Codes.

Nachdem nun mehrere Möglichkeiten aufgeführt wurden, wird sich innerhalb von diesem Absatz für eine Methode zur Fehlerkennung innerhalb der UART Übertragung entschieden. Die erste Möglichkeit beachtet nicht die Priorisierung der verschiedenen Signale. Dadurch, dass immer alle Daten versendet werden, entsteht ein langer Frame mit vielen Datenpakten. Die Übertragung erfordert viel Zeit, was dazu führt, dass zwischen den neuen Daten der kritischen Signale eine längere Latenz besteht. Somit wird sich nicht für die erste Methode entschieden. Der Unterschied zwischen der zweiten und der dritten Methode liegt in der Anordnung der Daten innerhalb der Frame. Bei der zweiten Methode wird ein separater Frame ausschließlich für die Übermittlung kritischer Signale verwendet. Dies bedeutet, dass unkritische Daten in separaten Frames und zeitlich getrennt gesendet werden. Im Gegensatz dazu kombiniert die dritte Methode kritische und unkritische Daten innerhalb eines einzigen Frames. Die kritischen Signale werden dabei bei jeder Übertragung gesendet, während die unkritischen Daten zyklisch ergänzt werden. Ein entscheidender Vorteil der dritten Methode besteht darin, dass die Frames die gleiche Anzahl an Datenpaketen enthalten und daher stets das gleiche Zeitintervall zur Übermittlung benötigen. Dadurch entsteht eine konstante und gleichmäßige Übertragung. Dies ermöglicht es dem Empfänger, die Übertragungszyklen zu überwachen, indem die Zeitabstände zwischen den Frames überprüft werden. Wenn die Übertragungen nicht mehr regelmäßig erfolgen, kann dies auf mögliche Fehler in der Datenübertragung hinweisen. Somit bietet die dritte Methode zusätzliche Sicherheit zur Überprüfung der korrekten Funktion des Systems. Aufgrund der eben genannten Aspekte wird die dritte Methode bevorzugt. Sie vereint die zuverlässige Priorisierung kritischer Signale mit der regelmäßigen Übertragung unkritischer Daten und ermöglicht gleichzeitig eine Überwachbarkeit der korrekten Funktion der Software.
Für dieses Projekt bedeutet dies, dass ein Frame mit zehn Datenpaketen gestaltet wird. Jedes Datenpaket weist eine Größe von 8-Bit auf. Das erste Datenpaket enthält einen spezifischen Header, der die Art des zu sendenden Datenpakets identifiziert. Für die Übermittlung der gesammelten Messdaten wird der Header 0x01 verwendet. Anschließend werden in den nächsten vier Datenpaketen die kritischen Werte der aktuellen Magnetspannungen übermittelt. Es folgt das Senden des Identifiers. Mit dessen Unterstützung werden die weniger kritischen Signale zyklisch übermittelt. Der Identifier gibt dabei an, welche Daten in den folgenden zwei Datenpaketen übertragen werden.

Der folgenden Auflistung kann entnommen werden, wie der Identifier den zu übertragenden Daten zugeordnet ist.

Identifier 0: Temperaturdaten Sensor 1

Identifier 1: Temperaturdaten Sensor 2

Identifier 2: Temperaturdaten Sensor 3

Identifier 3: 12 V Signal Power-Supplies

Identifier 4: 12 V Spannungsversorgung

Identifier 5: 5 V VCC- Spannung

Identifier 6: digitale Signale (Detektions-, Matrix-, Dump- und Adresssignale)

Zuletzt erfolgt das Senden der CRC-16-Checksumme in den letzten zwei Datenpaketen. In der Abbildung 4.3 ist die Zusammensetzung des Frames nochmals grafisch dargestellt. Jeder Kasten beinhaltet ein 8-Bit Paket.

Header	Magnet-spannung M1	Magent-spannung M1	Magnet-spannung M2	Magnet-spannung M2	Identifier	Daten	Daten	CRC	CRC

Abbildung 4.3: Zusammensetzung des über UART zu übertragenden Frames

4.2.5 Konzeptionierung der Timinganforderungen

Eine Anforderung an das System ist es die Daten innerhalb eines vorgegebenen gleichbleibenden Intervalls zu sammeln und anschließend an das Hauptsystem mittels UART zu übermitteln. Um das feste Zeitintervall zu realisieren, ist es sinnvoll Timer zu verwenden, die nach Ablauf einer einstellbaren Zeit einen Interrupt auslösen. Ein Vorteil dieser Methode ist die Genauigkeit in der Ausführung. Hardware-Timer bieten meist eine sehr präzise Zeitsteuerung, da sie unabhängig von der CPU oder weiteren Systemaufgaben arbeiten. Dadurch, dass die Zeitmessung ohne CPU-Anspruch durchgeführt werden kann, werden die Ressourcen effektiv genutzt und die CPU nicht blockiert. Während der Counter des Timers abläuft, können dementsprechend andere Aufgaben durch die CPU ausgeführt werden. Eine Alternative zu dem Hardware-Timer wäre das Verwenden von Polling, wo in einer Endlosschleife zwischen Datenabfrage und aktiven Warten gewechselt

wird. Dies ist im Gegensatz den Timern ressourcenintensiv, da die CPU dauerhaft mit dem Warten und Abfragen der Daten beschäftigt ist. Ebenso ist keine gute Genauigkeit der Zeitpunkte der Datenabfrage gegeben, da diese durch die Prozessorauslastung beeinträchtigt werden kann. Somit ist von dieser Methode abzusehen.

Um die verschiedenen Daten unabhängig voneinander und der Priorisierung entsprechend abzufragen, werden vier verschiedene Timer benötigt. Davon dienen drei Timer zur Datenerfassung und ein Timer zur regelmäßigen Datenübermittlung.
Zunächst wird der Timer für die konstante UART-Übertragung konzeptioniert. Hierfür muss zunächst die Zeit bestimmt werden, welche für eine UART-Übertragung benötigt wird. Es werden insgesamt elf Datenpakete bei einer UART-Übertragung übermittelt. Als Baudrate wird die 230400 Bits pro Sekunde gewählt. Es wird ein Stopbit und eine gerade Parität bei der Übertragung verwendet. Zuerst wird die benötigte Zeit für die Übertragung eines Bits bestimmt.

$$\begin{aligned} \text{Baud} &= \frac{1}{\text{Baudrate}} \\ &= \frac{1}{230400\,\frac{\text{Bit}}{\text{s}}} \\ &= 4.34\,\mu\text{s}\,\text{pro}\,\text{Bit} \end{aligned}$$

Pro Datenpaket werden ein Startbit, acht Datenbits, ein Paritätsbit und ein Stopbit übertragen. Dies ergibt insgesamt elf Bits pro Datenpaket. Bei einer Gesamtanzahl von elf Datenpaketen pro Übertragung ergibt sich die benötigte Zeit für die Übertragung eines Frames wie folgt.

$$\begin{aligned} \text{t}_{\text{Ges}} &= \text{Baud} \cdot 11\,\text{Bit} \cdot 11 \\ &= 4.34\,\frac{\mu\text{s}}{\text{Bit}} \cdot 11\,\text{Bit} \cdot 11 \\ &= 525.17\,\mu\text{s} \end{aligned}$$

Die Übertragung von elf Datenpaketen benötigt somit 525.17 μs. Für die Timereinstellung wird ein Intervall von 1 ms für das Senden der UART-Daten gewählt. Dieser Abstand ermöglicht es, die vorherige UART-Übertragung komplett abzuschließen und die neuen Daten zur Übertragung bereitzustellen. Ebenso wird ein Puffer für Zeitverzögerungen implementiert, die durch die GPIO-Interrupts oder das Empfangen von UART-Daten entstehen. Der Empfänger dieser UART-Daten kann dementsprechend ein konstantes

Empfangsintervall von 1 ms erwarten. Es folgt die Bestimmung der Timereinstellungen, um dieses Intervall zu ermöglichen. Der Timer wird so eingestellt, dass jede Mikrosekunde ein Zählschritt ausgeführt wird. Es wird ein 16-Bit Timer verwendet, welcher mit einer Frequenz von 48 MHz betrieben wird.

$$\begin{aligned} f_{\text{Timer}} &= \frac{1}{1\,\mu\text{s}} \\ &= 1\,\text{MHz} \end{aligned} \qquad \begin{aligned} \text{Prescaler} &= \frac{f_{\text{Takt}}}{f_{\text{Timer}}} \\ &= \frac{48\,\text{MHz}}{1\,\text{MHz}} \\ &= 48 \end{aligned}$$

Mit einem Prescaler von 48 zählt der Timer jede Mikrosekunde einen Zählschritt nach oben. Bei einem gewünschten Zählintervall von 1 ms muss somit die Zählvariable einen Wert von 1000 erhalten.

Es folgt die Timer Bestimmung für die ADU-Auswertung. Zunächst wird bestimmt, wie viel Zeit für das Auslesen der sechs Kanäle des ADUs benötigt wird. Der ADU weist eine Auflösung von 12 Bit auf. Hieraus ergibt sich eine benötigte Wandlungszeit von 12.5 Takten. Diese Zeit wird benötigt, um die analogen Werte des Signals in einen digitalen Wert zu wandeln. Zudem wird eine Variable Sample-and-Hold-Zeit genutzt. Diese Variable ist die Zeitverzögerung an, bis eine stabile Spannung am Kondensator des ADUs anliegt. Es wird eine Zeit von 7.5 Takten verwendet. Die ADU-Frequenz beträgt 14 MHz

$$\begin{aligned} t_{\text{ADU}} &= \frac{\text{Wandlungszeit} + \text{Sample-and-Hold-Zeit}}{f_{\text{ADU}}} \cdot 6 \\ &= \frac{12.5 + 7.5}{14\,\text{MHz}} \cdot 6 \\ &= 8.57\,\mu\text{s} \end{aligned}$$

Zum Auslesen der sechs Kanäle werden insgesamt 8.57 μs benötigt. Da zu diesem Zeitpunkt noch nicht bekannt ist, wie lange die Auswertung der Daten der ADUs benötigt, wird ein Zeitpuffer integriert. Die berechnete Zeit stellt das minimale Intervall für die Abfrage der ADU-Werte dar. Wird das Abfrageintervall zu kurz gewählt, ist die CPU vollständig mit der Verarbeitung und Abfrage der ADU-Werte ausgelastet und kann keine

weiteren Aufgaben ausführen. Um dieses Problem zu vermeiden, wird der hinzugefügte Puffer bewusst großzügig ausgelegt. In der Implementierung muss das hier festgelegte Zeitintervall jedoch erneut überprüft und gegebenenfalls angepasst werden. Das Intervall soll 70 μs betragen. Es wird wie bei der UART-Übertragung eine Zählzeit von einer Mikrosekunde pro Schritt festgelegt. Dementsprechend erhält der Prescaler ebenfalls einen Wert von 48. Die Zählvariable muss bei einem Intervall von 70 μs den Wert 70 aufweisen.

Die Abfrage der Temperaturwerte soll in einem deutlich längeren Intervall erfolgen. Eine Temperaturänderung wird voraussichtlich langsamer erfolgen als die Veränderung der Magnetspannung. Aus diesem Grund wird Timer so eingestellt, dass die Temperatursensoren alle 1.5 s abgefragt werden. Bei insgesamt drei Timer ist es sinnvoll, die Abfrage eines Timers immer nach 500 ms zu starten und den abzufragenden Timer zyklisch zu ändern. Es wird eine Zählzeit für einen Schritt von einer Millisekunde festgelegt.

$$\begin{aligned} f_{\text{Timer}} &= \frac{1}{1\,\text{ms}} \\ &= 1\,\text{kHz} \end{aligned} \qquad \begin{aligned} \text{Prescaler} &= \frac{f_{\text{Takt}}}{f_{\text{Timer}}} \\ &= \frac{48\,\text{MHz}}{1\,\text{kHz}} \\ &= 48000 \end{aligned}$$

Der Prescaler muss einen Wert von 48000 und die Zählvariable von 500 aufweisen.

Zuletzt ist der Timer für die GPIO Auswertung via Polling zu bestimmen. Hier wird ebenfalls wieder eine Zählzeit von 1 μs verwendet. Das Abfrageintervall beträgt 300 μs. Im Vergleich zu der Auswertung der Magnetspannung durch die ADUs werden hier eher unkritische Signale abgefragt. Die kritischen GPIO-Signale sind mit einem Interrupt ausgestattet, sodass hier keine Änderung verpasst wird. Aus diesem Grund ist das längere Abfrageintervall ausreichend. Um dieses Abtastintervall zu erhalten, muss der Prescaler einen Wert von 48 und die Zählvariable einen Wert von 300 annehmen.

5 Implementierung

In diesem Kapitel wird die Umsetzung der Anforderungen und der Konzepte behandelt. Hierfür wird zunächst innerhalb des Unterkapitels Hardware die Erstellung der Schaltpläne und der Platine mit Hilfe des Entwicklungstools Eagle beschrieben. Anschließend erfolgt in dem Unterkapitel Software die Programmierung des Mikrokontrollers. Es ist anzumerken, dass die nachfolgende Beschreibung der Implementierung der Hardware auf einen zweiten, bereits erweiterten Prototypen basiert. Dieser zweite Prototyp konnte bis zur Abgabe dieser Arbeit aus Zuliefergründen nicht vollständig fertiggestellt werden. Da die Softwareentwicklung parallel erfolgt, ist die Software auf den ersten Prototypen ausgelegt.

5.1 Hardware

Es erfolgt zunächst die Implementierung der Hardware. Dieses Kapitel ist in die Erstellung des Schaltplans und Erstellung der Platine unterteilt. Im ersten Teil werden die einzelnen implementierten Schaltungen vorgestellt und beschrieben. Im zweiten Teil erfolgt anschließend die Vorstellung der Platine, welche die Schaltungen aus dem ersten Teil enthält.

5.1.1 Erstellung der Schaltpläne in Eagle

Es folgt die Vorstellung von einzelnen Auszügen aus dem Schaltplan. Hierbei wird die einzelne vorgenommene Schaltung und die verwendeten Komponenten vorgestellt. Der gesamte Schaltplan ist innerhalb des Anhangs unter dem Kapitel A.4 einzusehen.

Verbindungsstecker zwischen den Platinen

Zunächst werden die Verbindungsstecker zwischen den Platinen vorgestellt. Es sind insgesamt sechs Stecker zu implementieren. Die Anzahl und die Signalbelegung wird durch die Hot-Swap-Platine vorgegeben. In der Abbildung 5.1 sind die sechs Stecker zu erkennen.

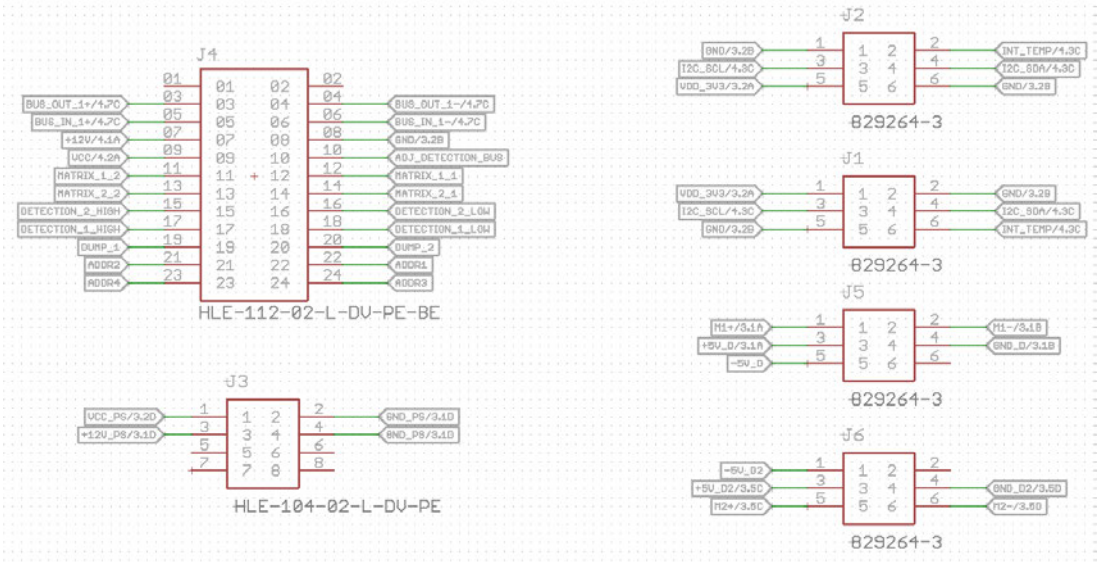


Abbildung 5.1: Auszug aus dem Schaltplan: Verbindungsstecker

Es folgt nun eine Beschreibung der einzelnen Stecker.

Zuerst wird der Stecker J4 beschrieben. Die ersten beiden Pins weisen keine Belegung auf. Über Pin drei bis sechs wird ein differentielles Bussignal übertragen. Dieses Bussignal wird durch eine RS485-Schaltung in das UART-Signal umgewandelt. Die Vorstellung dieser Schaltung erfolgt innerhalb des Kapitels RS485 Empfänger im hinteren Teil der Hardwareimplementierung. Über Pin sieben und Pin acht wird die 12 V-Versorgungsspannung der Mikrokontrollerplatine und die GND-Anbindung übertragen. Über Pin neun wird das 5 V VCC-Signal übermittelt. Auf dem Pin zehn ist das PWM-Signal ADJ_Detection_Bus gelegt. An Pin elf bis Pin vierzehn liegen die Matrix-Signale an. Es folgt an den Pins 15 bis 18 die Übertragung der Detection-Signale. Anschließend werden an Pin 19 und Pin 20 die Dump-Signale übertragen. Über die Pins 21 bis 24 werden die vier Adress-Signale übergeben.
Der Stecker J3 dient zur Übertragung der 12 V-Versorgungsspannung sowie des GND-Signals der Spannungsversorgungen. Zusätzlich wird das 5 V-VCC-Signal übertragen.

Die Stecker J1 und J2 sind nahezu identisch aufgebaut. Beide Stecker werden dazu verwendet, um die zwei Temperatursensoren auf der Hot-Swap-Platine zu betreiben. Da diese Sensoren auf der Hot-Swap-Platine räumlich getrennt angeordnet sind und die entsprechenden Leitungen nicht über die ganze Platine gezogen werden sollen, ist für jeden Sensor ein Stecker vorgesehen. An beiden Steckern wird die Vorsorgungsspannung von 3.3 V, zwei GND-Anbindungen, die Signale SCL und SDA für die I^2C Kommunikation und ein Interruptsignal übermittelt.
Die Stecker J5 und J6 dienen der Übertragung der Magnetspannungen. Dabei repräsentieren M1+ und M1- die Magnetspannung des ersten Magneten, während M2+ und M2- die Magnetspannung des zweiten Magneten übertragen. Zusätzlich wird für jede Magnetspannung ein +5 V- und ein −5 V-Signal auf demselben Potential der jeweiligen Spannung übermittelt.

Versorgungsspannung des Mikrokontrollers

Innerhalb von diesem Kapitel wird die implementierte Schaltung zur Bewerkstelligung der 5 V-Versorgungsspannung des Mikrokontrollers erläutert. Von dem ersten Prototypen konnte der Entschluss gefasst werden, dass ein LDO aufgrund der hohen Differenz von Eingangs- zu Ausgangsspannung der damit verbundenen Wärmeentwicklung an seinem absoluten Maximum arbeitet. Aus diesem Grund wird auf dem zweiten Prototypen ein Abwärtswandler implementiert. Um einen späteren Vergleich der beiden Schaltungen innerhalb des Betriebs ausführen zu können, ist eine im Vergleich zum ersten Prototypen verbesserte LDO-Schaltung weiterhin auf der Platine vorhanden. Im folgenden wird zunächst die LDO-Schaltung und anschließend die Schaltung des Abwärtswandlers erläutert.
In der Abbildung 5.2 ist der Schaltplanauszug des LDOs zu erkennen.

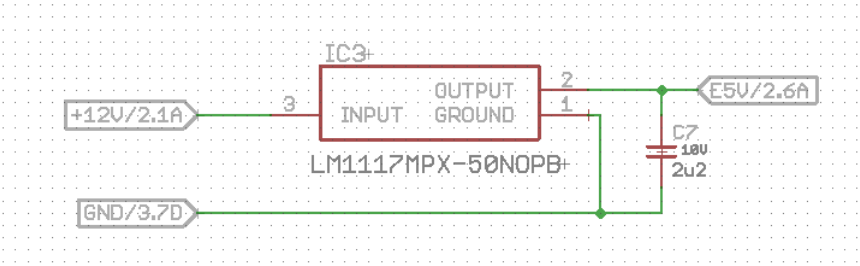


Abbildung 5.2: Auszug aus dem Schaltplan: LDO

Es wird das 12 V-Eingangsspannungssignal an den Input-Pin und das GND-Potential an den GND-Pin angelegt. Am Ausgang der LDOs liegen die benötigten 5 V an, welche für die Spannungsversorgung des Mikrokontrollers und für weitere ICs, wie die Pegelwandler für die digitalen Signale, verwendet werden.

Es folgt die Beschreibung der Schaltung zur Reduktion der 12 V-Spannung mithilfe des Abwärtswandlers. Der entsprechende Auszug aus dem Schaltplan ist innerhalb der Abbildung 5.3 zu erkennen.

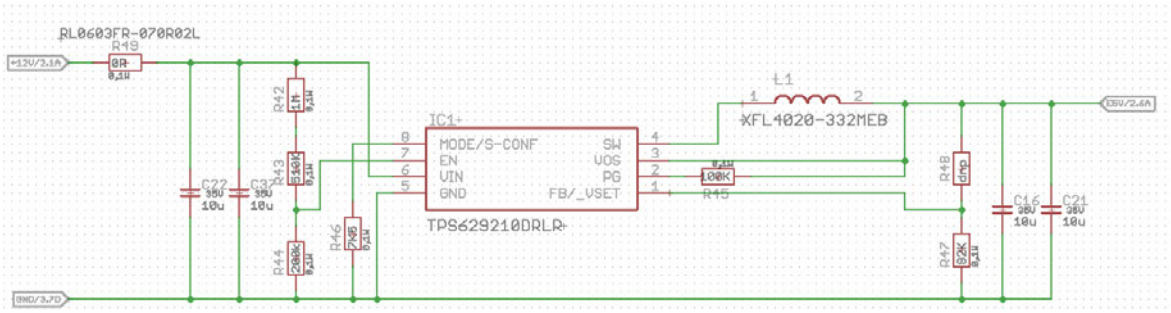


Abbildung 5.3: Auszug aus dem Schaltplan: Abwärtswandler

Es wird der Buck-Converter TPS629210DRLR zur Spannungsreduktion verwendet. An den Pin sechs wird die 12 V-Eingangsspannung dem IC zugeführt. Über die Kondensatoren C22 und C32 werden Spannungsschwankungen innerhalb der 12 V-Spannung herausgefiltert und das Signal geglättet. Durch den Spannungsteiler der Widerstände R42, R43 und R44 wird der enable-Pin (Pin 7) mit der 12 V-Spannung verbunden. Bei einem High-Signal am Enable-Pin wird das IC aktiviert und in den Betriebsmodus versetzt. Liegt hingegen ein Low-Signal an, bleibt das IC deaktiviert und außer Betrieb. Der Spannungsteiler wird verwendet, damit das IC erst bei einer ausreichend hohen Eingangsspannung aktiviert wird. Aus dem Datenblatt des TPS629210DRLR ist zu entnehmen, dass das IC ab einer Spannung von 1 V an dem enable-Pin in den Betriebsmodus versetzt wird. Wäre der enable-Pin direkt an die 12 V-Eingangsspannung angeschlossen, würde das IC aktiviert werden, sobald die Eingangsspannung den Wert von 1 V übersteigt. In der folgenden Rechnung wird bestimmt, welchen Wert die Eingangspannung annehmen muss, damit das IC nach dem Spannungsteiler in den Betriebsmodus versetzt wird. Es wird die

allgemeine Formel des Spannungsteilers angewendet.

$$\begin{aligned} \mathrm{V_{IN\_min}} &= \mathrm{V_{en\_min}} \cdot \frac{R42 + R43 + R44}{R44} \\ &= 1\mathrm{V} \cdot \frac{1\,\mathrm{M\Omega} + 510\,\mathrm{k\Omega} + 200\,\mathrm{k\Omega}}{200\,\mathrm{k\Omega}} \\ &= 8.55\,\mathrm{V} \end{aligned} \tag{5.1}$$

Die minimale Eingangsspannung, ab der das IC in den Betriebsmodus versetzt wird, liegt bei einem Wert von 8.55 V.
Mithilfe des Pin acht des ICs kann der Betriebsmodus des Buck-Converters eingestellt werden. Hier ist sowohl durch den Anschluss des Pins auf GND, als auch über den Widerstandswert des Widerstands R46 der Auto Pulsfrequenzmodulation (PFM)/PWM Modus ausgewählt. Dieser Modus ermöglicht eine automatische Optimierung der Effizienz bei unterschiedlichen Lastbedingungen.
An dem Pin vier liegt der Ausgang der Schaltfrequenz an, welche die Spule L1 ansteuert. Die Ausgangsspannung der Spule wird an den Pin VOS zurück gekoppelt, um die Spannung zu stabilisieren. Mithilfe des Feedbackpins FB/_VSet (Pin 1) kann die Ausgangsspannung über den folgenden Spannungsteiler von R48 und R47 eingestellt werden. Der Widerstand R48 wird nicht bestückt. Somit ist diese Leitung als offen anzusehen. Der Widerstand R47 weist einen Wert von 82 kΩ auf. Der Tabelle 8-2 aus dem Datenblatt des Bauteils ist zu entnehmen, dass dieser Widerstandswert eine Ausgangsspannung von 5 V erzeugt. Die Ausgangskondensatoren C16 und C21 dienen zur Glättung der Ausgangsspannung.

Spannungsteilung für die Eingangsspannungsüberwachung

In diesem Unterkapitel werden die Spannungsteiler von 12 V und von 5 V auf 3 V beschrieben. Die Spannungsteilung ist notwendig, um die Signale mithilfe eines ADUs des Mikrokontroller überwachen zu können.

Der hierfür relevante Schaltplanauszug ist in der Abbildung 5.4 zu erkennen. Es müssen die Widerstände R29 und R30 sowie R34 und R35 bestimmt werden.

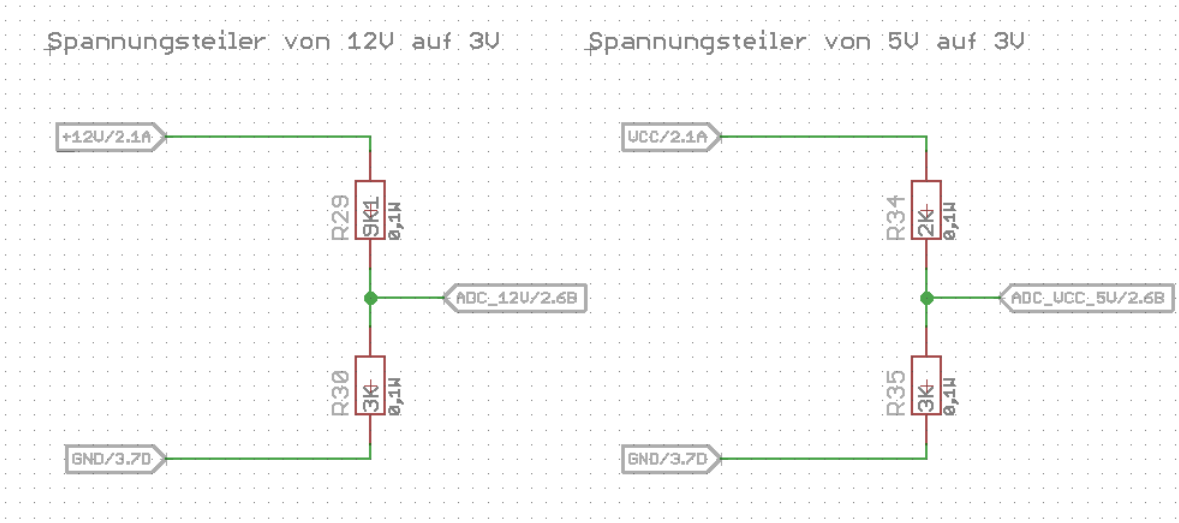


Abbildung 5.4: Auszug aus dem Schaltplan: Spannungsteiler

Zunächst wird die Berechnung des Spannungsteilers von 12 V auf 3 V durchgeführt. Es soll ein maximaler Stromfluss von 1 mA möglich sein. Zuerst wird der benötigte Gesamtwiderstand R_{Ges} bestimmt.

$$\begin{aligned} \mathrm{R_{Ges}} &= \frac{\mathrm{U}}{\mathrm{I}} \\ &= \frac{12\,\mathrm{V}}{1\,\mathrm{mA}} \\ &= 12\,\mathrm{k\Omega} \end{aligned} \tag{5.2}$$

Anschließend erfolgt die Bestimmung der Widerstandsgrößen R_{29} und R_{30}.

$$\begin{aligned} \mathrm{R_{29}} &= \frac{\mathrm{U_{R29}}}{\mathrm{U_{Ges}}} \cdot \mathrm{R_{Ges}} \\ &= \frac{9\,\mathrm{V}}{12\,\mathrm{V}} \cdot 12\,\mathrm{k\Omega} \\ &= 9\,\mathrm{k\Omega} \end{aligned} \tag{5.3}$$

$$\begin{aligned} \mathrm{R_{30}} &= \frac{\mathrm{U_{R30}}}{\mathrm{U_{Ges}}} \cdot \mathrm{R_{Ges}} \\ &= \frac{3\,\mathrm{V}}{12\,\mathrm{V}} \cdot 12\,\mathrm{k\Omega} \\ &= 3\,\mathrm{k\Omega} \end{aligned} \tag{5.4}$$

Es folgt die Berechnung, welche Leistung über den Spannungsteiler abfällt. Diese Leistung ist bei der Auswahl der Widerstände zu berücksichtigen.

$$\begin{aligned} P &= U \cdot I \\ &= 12\,V \cdot 1\,mA \\ &= 12\,mW \end{aligned} \tag{5.5}$$

Mit den ausgerechneten Werten können geeignete Widerstände für R_{29} und R_{30} ausgewählt werden. Für R_{29} wird ein Widerstand mit 9.1 kΩ und für R_{30} ein Widerstand mit 3 kΩ ausgewählt. Beide Widerstände sollten eine Leistung von mindestens 0.1 W vertragen können. Es werden SMD 0603 Widerstände mit einer Toleranz von ±1 % ausgewählt. Es folgt die Berechnung des 5 V Spannungsteilers. Hier erfolgt ebenfalls eine Strombegrenzung auf 1 mA.

$$\begin{aligned} R_{Ges} &= \frac{U}{I} \\ &= \frac{5\,V}{1\,mA} \\ &= 5\,k\Omega \end{aligned} \tag{5.6}$$

Anschließend erfolgt die Bestimmung der Widerstandsgrößen R_{35} und R_{34}.

$$\begin{aligned} R_{34} &= \frac{U_{R34}}{U_{Ges}} \cdot R_{Ges} \\ &= \frac{2\,V}{5\,V} \cdot 5\,k\Omega \\ &= 2\,k\Omega \end{aligned} \tag{5.7}$$

$$\begin{aligned} R_{35} &= \frac{U_{R35}}{U_{Ges}} \cdot R_{Ges} \\ &= \frac{3\,V}{5\,V} \cdot 5\,k\Omega \\ &= 3\,k\Omega \end{aligned} \tag{5.8}$$

Anschließend wird der Leistungsabfall über dem Spannungsteiler bestimmt.

$$\begin{aligned} P &= U \cdot I \\ &= 5\,V \cdot 1\,mA \\ &= 5\,mW \end{aligned} \tag{5.9}$$

Mit den ausgerechneten Werten können geeignete Widerstände für R_{34} und R_{35} ausgewählt werden. Für R_{34} wird ein Widerstand mit 2 kΩ und für R_{35} ein Widerstand mit

3 kΩ ausgewählt. Beide Widerstände sollten mindestens eine Leistung von 0.1 W umsetzen können. Auch hier werden SMD 0603 Widerstände mit einer Toleranz von ±1% gewählt.

Spannungsumsetzung der digitalen Signale

Es folgt die Umsetzung der Spannungsreduktion der digitalen Signale mithilfe des Pegelwandlungs-ICs TXS0104ED. In der Abbildung 5.5 ist ein Ausschnitt des Schaltplans zu erkennen. Es wird die implementierte Schaltung einmal vorgestellt, da die Wandlung bei allen digitalen Signalen auf identische Weise erfolgt.

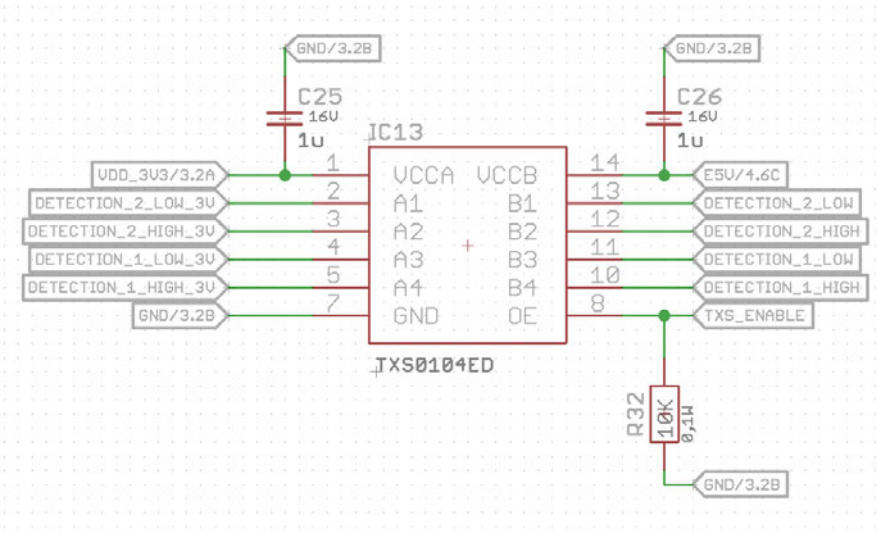


Abbildung 5.5: Auszug aus dem Schaltplan: Spannungsumsetzung der digitalen Signale

Auf der linken Seite des Bauteils TXS0104E werden die Signale, die mit dem Mikrocontroller verbunden werden sollen, an die Pins A1, A2, A3 und A4 angeschlossen. Die Spannung, auf die die Signale nach der Spannungsumsetzung geregelt werden sollen, wird an VCCA angelegt. In diesem Fall beträgt sie 3.3 V. Pin 7 wird mit GND verbunden. Auf der rechten Seite des Bauteils werden die 5 V-Signale, die über die Steckverbindungen von der Hot-Swap-Platine übermittelt werden, an die Pins B1, B2, B3 und B4 angeschlossen. An VCCB werden 5 V angelegt. Das Bauteil wird über den Pin OE (Output Enable) durch den Mikrokontroller gesteuert. Liegt ein Low-Signal am OE-Pin an, ist das Bauteil deaktiviert. Alle Ein- und Ausgänge befinden sich im hochohmigen Zustand. Wird

hingegen eine Eingangsspannung von 3.3 V angelegt, erhält der OE-Pin ein High-Signal, wodurch das Bauteil aktiviert wird.

Galvanische Trennung der Magnetspannungen

In diesem Abschnitt wird die Umsetzung der galvanischen Trennung zur Messung der Magnetspannung beschrieben. Der entsprechende Schaltplanauszug ist in der Abbildung 5.6 zu erkennen. Es wird nur der Schaltplanauszug für die Magnetspannung M1 beschrieben. Der Schaltplan für die galvanische Trennung des zweiten Magneten ist nahezu identisch und kann im Anhang unter dem Kapitel A.4 eingesehen werden.

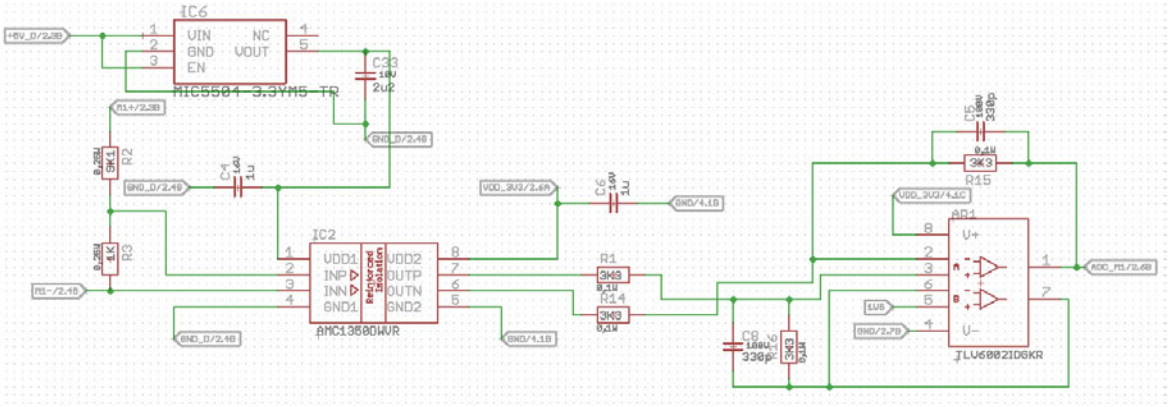


Abbildung 5.6: Auszug aus dem Schaltplan: Galvanische Trennung Magnetspannung

Der Baustein AMC1350DWVR (IC2) ist der Trennverstärker, welcher für die galvanische Trennung der Magnetspannung verantwortlich ist. Auf der linken Seite des Bauteils ist der Anschluss für den Magneten mit dem zugehörigen Potential angeordnet, während auf der rechten Seite das Potential des Mikrokontrollers anliegt.
Zunächst soll die linke Seite des Trennverstärkers betrachtet werden. An VDD1 wird eine 3.3 V-Versorgungsspannung angeschlossen. Diese Spannung wird mithilfe eines LDOs generiert. Der LDO ist in diesem Schaltplanauszug unter dem IC6 zu erkennen. Hier wird die 5 V Eingangsspannung, welche durch den Buck-Converter erzeugt wurde, auf 3.3 V reduziert. Der Enable-Pin (Pin 3) ist direkt mit der 5 V-Spannung verbunden. Dadurch ist das Bauteil permanent aktiv, sobald eine Spannung anliegt. Dieser LDO wird benötigt, damit an dem Trennverstärker eine konstante Spannungsversorgung anliegt, um einen störungsfreien Betrieb der Schaltung zu gewährleisten.
An dem Pin GND1 des Trennverstärkers wird das zur Magnetspannung M1 zugehörige

GND-Potential angeschlossen.
An die Pins INP und INN werden Plus und Minus der Magnetspannung angelegt. Da der Trennverstärker maximal eine Spannung von ± 5 V verarbeiten kann, jedoch deutlich höhere Magnetspannungen auftreten können, ist die Verwendung eines Spannungsteilers erforderlich. Mithilfe von diesem Spannungsteiler soll die Magnetspannung reduziert werden, damit der Trennverstärker korrekt arbeiten kann. Um den Spannungsteiler genau auslegen zu können, muss zuerst die rechte Seite vom Trennverstärker betrachtet werden.

Auf rechten Seite werden an den VDD2-Pin die 3.3 V-Versorgungsspannung, welche vom Mikrokontrollerboard generiert werden, angeschlossen. Der GND2-Pin wird mit dem GND-Potential des Mikrokontrollers verbunden.
An OUTP und OUTN liegt die Magnetspannung nach der galvanischen Trennung an. Zu beachten ist hierbei, dass dieses Signal differentiell ist. Diese differentielle Spannung kann maximale Werte von ±2 V in Bezug auf eine Ausgangs-Gleichtaktspannung (V_{CMout}) annehmen. Dieser Wert ist dem entsprechendem Datenblatt im digitalen Anhang zu entnehmen.
Um das differentielle Signal in ein single-ended Signal umzuwandeln, wird ein Operationsverstärker (AR1) verwendet. Dieser Operationsverstärker agiert als Addierer und besitzt zwei voneinander unabhängige Kanäle. An Pin acht wird die Versorgungsspannung VDD_3V3 angeschlossen. Diese Spannung weist einen Wert von 3.3 V auf. Somit kann der OP in einem Bereich von 0 V bis maximal 3.3 V arbeiten, wobei im realen Zustand meist nicht die 3.3 V erreicht werden können.
An dem Kanal A des OPs wird das differentielle Signal angeschlossen (Pin zwei und Pin drei). Da das differentielle Signal Werte von ± 2 V erreichen kann, muss es mit einer Referenzspannung angehoben werden. Dies ist erforderlich, um sicherzustellen, dass der gesamte Spannungsbereich vom Operationsverstärker verarbeitet werden kann. Die Referenzspannung wird mithilfe eines LDOs erzeugt. Der zugehörige Schaltplanauszug ist in der Abbildung 5.7 zu erkennen.

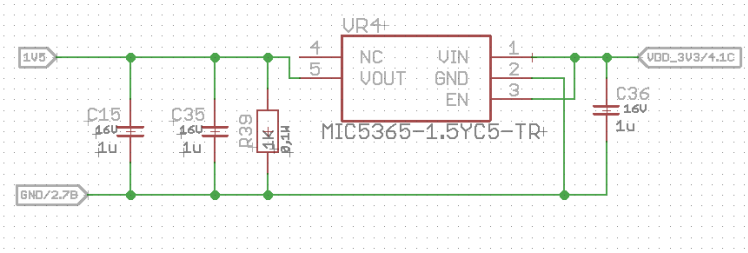


Abbildung 5.7: Auszug aus dem Schaltplan: Referenzspannung 1.5 V

Durch den LDO wird aus der 3.3 V-Spannung eine 1.5 V-Spannung erzeugt. Der Grund, warum gerade eine 1.5 V-Spannung als Referenzspannung verwendet wird, wird zu einem späteren Zeitpunkt erläutert. Der Enable-Pin (Pin drei) ist bei diesem LDO mit der Versorgungsspannung VDD_3V3 verbunden. Somit ergibt sich an dem Enable-Pin ein High-Signal, welches das Bauteil freischaltet.

Bezugnehmend auf die Abbildung 5.6 werden die vom LDO erzeugten 1.5 V an den Eingang B des Operationsverstärkers angeschlossen. Am Pin eins des OPs liegt anschließend das single-ended Ausgangssignal vor. Dieses Signal wird an einen ADU des Mikrokontrollers angeschlossen.

Es folgt nun die Auslegung der Schaltung. An den ADU des Mikrokontrollers darf maximal eine Spannung von 3.3 V angelegt werden. Hieraus kann auf die Spannungswerte für das differentielle Signal und der Referenzspannung geschlossen werden. Die Differenz des differentiellen Signals sollte genau die Hälfte der 3.3 V betragen, um sowohl die positiven Spannungswerte als auch die negativen Spannungswerte der Magnetspannung gleichmäßig abbilden zu können. Dies bedeutet bei einer maximalen Mikrokontroller Spannung von 3.3 V, dass das differentielle Signal ± 1.6 V aufweisen sollte. Die Referenzspannung sollte dementsprechend ebenfalls 1.6 V betragen, um die -1.6 V komplett auf einen positiven Wert anheben zu können. In der Implementierung ist ein LDO mit einer Ausgangsspannung 1.5 V verbaut worden, da die 1.5 V-Ausgangsspannung einen Standardwert darstellen. Somit werden auf der negativen Spannungsseite 0.1 V abgeschnitten.
Mithilfe von diesen Größen kann nun der Spannungsteiler auf der linken Seite des Trennverstärkers ausgelegt werden. Hierfür wird der positive maximale Wert von 1.6 V für das differentielle Signal (U_{OUT}) betrachtet. Der Trennverstärker besitzt nah dem Datenblatt einen Verstärkungsfaktor von 0.4 $\frac{\mathrm{V}}{\mathrm{V}}$. Es soll nun in der folgenden Rechnung der maximale positive Wert bestimmt werden, welcher an den Eingängen des Trennverstärkers anliegen darf.

$$\begin{aligned} \mathrm{U_{IN}} &= \frac{\mathrm{U_{OUT}}}{\mathrm{Gain}} \\ &= \frac{1.6\,\mathrm{V}}{0.4\,\frac{\mathrm{V}}{\mathrm{V}}} \\ &= 4\,\mathrm{V} \end{aligned} \tag{5.10}$$

Anschließend gilt es den Spannungsteiler so einzustellen, dass an dem Eingang des Trennverstärkers maximal die berechneten 4 V anliegen. Hierbei wird vorgegeben, dass eine Magnetspannung von 40 V als maximaler Wert messbar sein sollen. Somit werden die 40 V bei der Bestimmung des Spannungsteilers als U_{Ges} angenommen. Es soll ein maximaler Stromfluss von 4 mA auftreten. Zuerst wird die Größe des Gesamtwiderstands des Spannungsteilers bestimmt.

$$\begin{aligned} R_{Ges} &= \frac{U_{Ges}}{I_{Ges}} \\ &= \frac{40\,V}{4\,mA} \\ &= 10\,k\Omega \end{aligned} \tag{5.11}$$

Mithilfe des Gesamtwiderstands kann die Größe des Widerstandes R3, unterer Widerstand des Spannungsteilers, bestimmt werden. Dabei gilt, dass $U_{IN} = U_{R3}$.

$$\begin{aligned} R3 &= \frac{U_{R3}}{U_{Ges}} \cdot R_{Ges} \\ &= \frac{4\,V}{40\,V} \cdot 10\,k\Omega \\ &= 1\,k\Omega \end{aligned} \tag{5.12}$$

Es folgt die Bestimmung der Größe des Widerstandes R2.

$$\begin{aligned} R2 &= R_{Ges} - R3 \\ &= 10\,k\Omega - 1\,k\Omega \\ &= 9\,k\Omega \end{aligned} \tag{5.13}$$

Die berechneten Werte werden mit den verfügbaren Werten in der vorhandenen Bibliothek von Eagle abgeglichen. Dabei muss insbesondere der maximale Leistungsverbrauch

berücksichtigt werden.

$$P = U \cdot I \tag{5.14}$$

$$P = 40\,V \cdot 4\,mA$$

$$P = 0.16\,W$$

Die einzubauenden Widerstände müssen somit mindestens eine Leistung von 0.16 W vertragen können. Somit wird für R3 ein Widerstand mit 1 kΩ und für R2 ein Widerstand mit 9.1 kΩ verwendet. Beide Widerstände sind 1206 SMD-Widerstände mit einer Toleranz von ± 1% und können eine Leistung bis zu 0.25 W vertragen. Durch den leicht veränderten Widerstandswert von R2 ergibt sich eine maximale positive anliegende Spannung von 3.96 V an dem Trennverstärker.

Galvanische Trennung der 12 V-Versorgungsspannung

In diesem Abschnitt erfolgt die Umsetzung der galvanischen Trennung für die Messung der 12 V-Spannung. Diese Schaltung ist nahezu identisch zu der Schaltung der galvanischen Trennung der Magnetspannung. Nur die zu überwachende Eingangsspannung kann andere Werte annehmen (0 V – 12 V). In der Abbildung 5.8 ist der entsprechende Auszug aus dem Schaltplan zu erkennen.

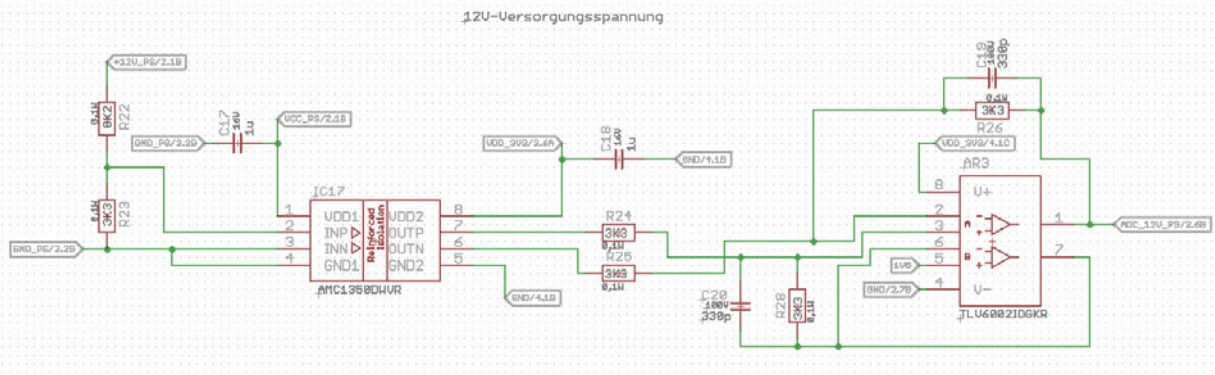


Abbildung 5.8: Auszug aus dem Schaltplan: Galvanische Trennung 12 V-Versorgungsspannung

Die durchzuführende Berechnung zur Auslegung des Schaltung ist zu dem vorherigen Kapitel identisch. Aus diesem Grund werden hier nur die Ergebnisse angegeben. Die

vollständige Berechnung kann im Anhang unter dem Kapitel A.3 eingesehen werden. Für den Widerstand R22 wird ein Widerstand mit 8.2 kΩ und für R23 ein Widerstand mit 3.3 kΩ ausgewählt. Auch dies sind beides 1206 SMD-Widerstände mit einer Toleranz von ± 1%. Es ergibt sich eine Eingangsspannung (U_{R23}) an dem Trennverstärker von 3.3V bei einer angelegten Spannung von 12 V. Am Ausgang des Operationsverstärkers liegt bei einer Eingangsspannung von 12 V eine Spannung von 3 V an.

Temperatursensor

Es wird der Temperatursensor STLM75DS2F von STMicroelectronics verwendet. Innerhalb der Abbildung 5.9 ist der zugehörige Ausschnitt aus dem Schaltplan zu erkennen.

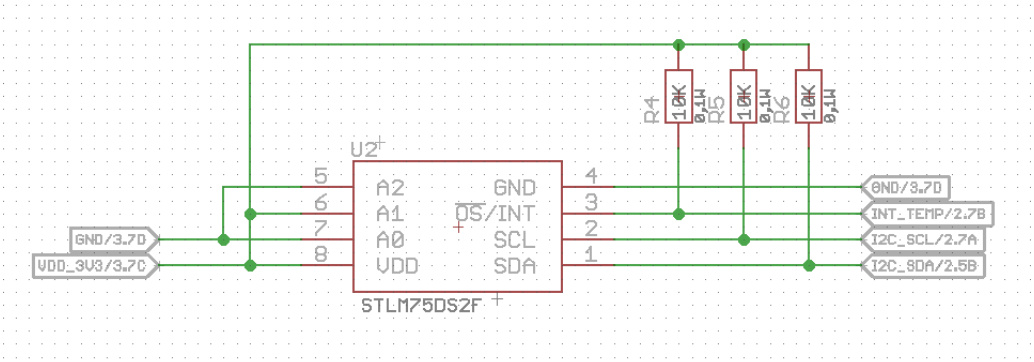


Abbildung 5.9: Auszug aus dem Schaltplan: Temperatursensor

Der Sensor wird mit der Versorgungsspannung VDD 3.3 V betrieben. Mithilfe der Eingänge A0, A1 und A2 kann die individuelle Adresse des Sensor eingestellt werden, indem die einzelnen Eingänge mit VDD oder auf GND verbunden werden. In diesem Fall sind A0 und A2 auf Low, also auf GND, und A1 auf High gelegt. Nach dem Datenblatt des Sensors ergibt sich eine Dezimaladresse von 74. In binär ist das eine 1001010, wobei die letzten drei Bits A2, A1 und A0 darstellen. Die beiden Eingänge SCL und SDA stellen die Anbindung an den I^2C-Bus dar. Hierbei ist zu beachten, dass sowohl SCL als auch SDA jeweils einen Pull-Up-Widerstand auf VDD benötigen. Dafür werden hier die Widerstände R5 und R6 mit jeweils 10 kΩ verwendet. Des Weiteren bietet dieser Sensor die Möglichkeit ein Interruptsignal zu konfigurieren und zu verwenden. Der Interrupt wird bei dem Über- oder Unterschreiten eines festzulegenden Schwellwerts ausgelöst. Das Signal wird durch den Ausgang $\overline{OS}$/INT übertragen. Hier ist ebenfalls durch den Widerstand R4 ein Pull-Up-Widerstand integriert.

RS485 Empfänger

Die hier vorzustellende Schaltung ist von der Hot-Swap-Platine übernommen worden. Jedoch soll die Funktion der Schaltung zur Vollständigkeit erklärt werden.
Mithilfe von dieser Schaltung wird das differentielle Bussignal zu einem UART-Signal umgewandelt, so dass die zu sendenden Daten über eine größere Entfernung störungsarm übertragen werden können. In der Abbildung 5.10 ist der zugehörige Schaltplanauszug zu erkennen.

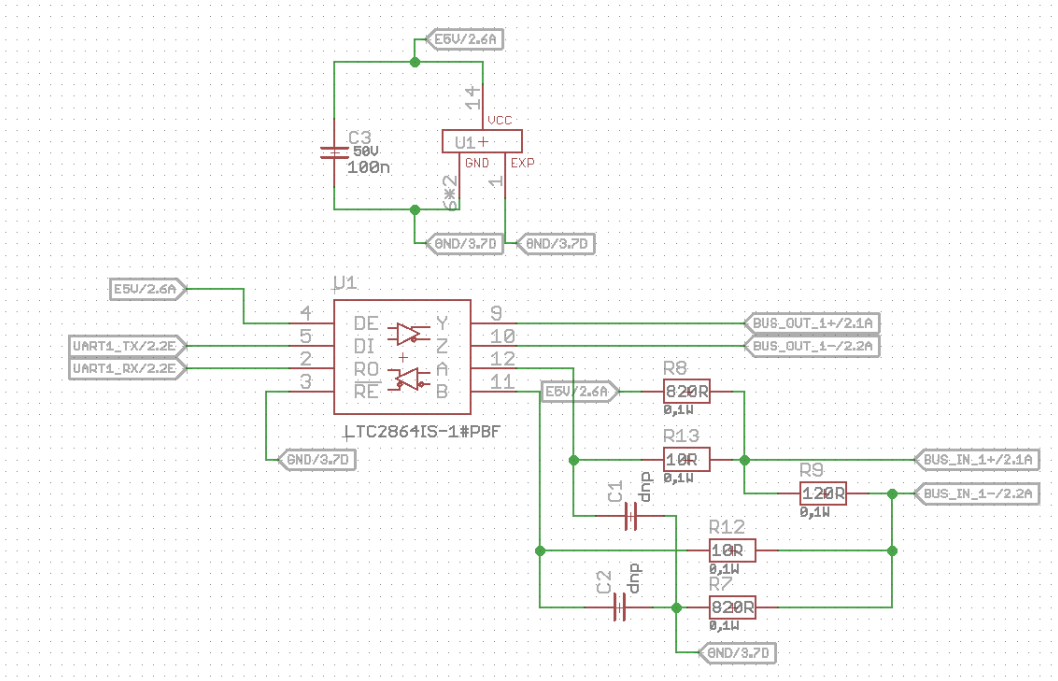


Abbildung 5.10: Auszug aus dem Schaltplan: RS485 Empfänger

Das Bauteil LTC2864IS ist ein RS485/RS422 Transceiver. Auf der rechten Seite des Bauteils wird das differentielle Signal an die Pins Y und Z für das Ausgangssignal, sowie an A und B für das Eingangssignal angelegt. Auf der linken Seite des Bauteils werden an den Pins DI und RO die UART Rx- und TX-Pins angeschlossen. Der Pin drei und der Pin vier sind zur Freigabe des Sendens und Empfanges vorhanden. Mithilfe von dem Pin drei (Receiver Enable (RE)) wird der Empfänger und durch den Pin vier (Driver Enable (DE)) der Sendetreiber aktiviert. Sobald an dem DE Pin ein High-Signal anliegt, sendet der Transceiver über die Leitungen Y und Z Daten. Liegt an dem RE Pin ein Low-Signal an, empfängt der Transceiver über die Leitungen A und B Daten und gibt diese an die

RO Leitung weiter. In dieser Schaltung ist sowohl das Senden, als auch das Empfangen der Daten freigeschaltet.
Der Widerstand R9 stellt den Abschlusswiderstand, welche Reflexionen vermeiden soll, dar. Die Widerstände R7, R8, R12 und R13 dienen zur Strombegrenzung und Rauschunterdrückung. Mithilfe der Kondensatoren C1 und C2 kann in Zukunft noch eine zusätzliche Filterung der Signale vorgenommen werden. In dieser Schaltung sind die Kondensatoren zunächst nicht zu bestücken.

Mikrokontroller

In der Abbildung 5.11 ist der Schaltplan des Mikrokontrollers abgebildet. Die genaue Pinbelegung ist dem Anschlussplan zu entnehmen, welcher in dem Anhang unter dem Kapitel A.2 einzusehen ist.

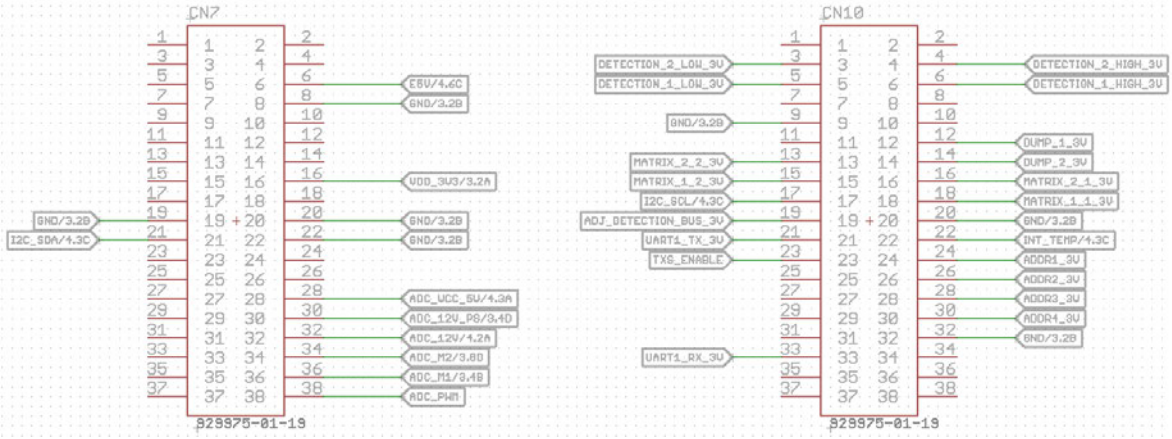


Abbildung 5.11: Auszug aus dem Schaltplan: Mikrokontroller

5.1.2 Erstellung des Platinendesigns in Eagle

Nach der Fertigstellung der Schaltpläne erfolgt das Design der Platine. Auf der folgenden Seite ist in der Abbildung 5.12 das gesamte Design des zweiten Prototyps zu erkennen.

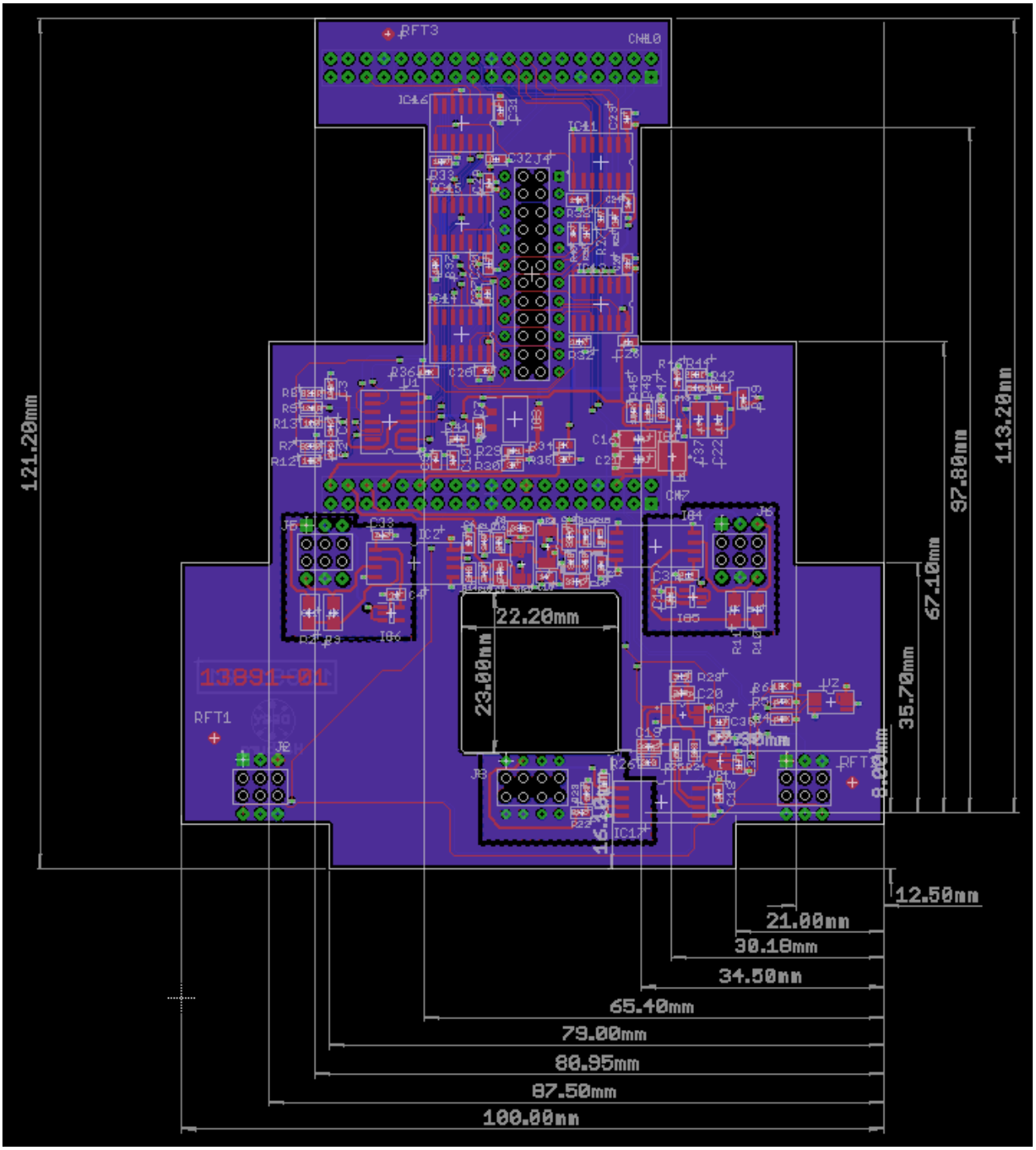
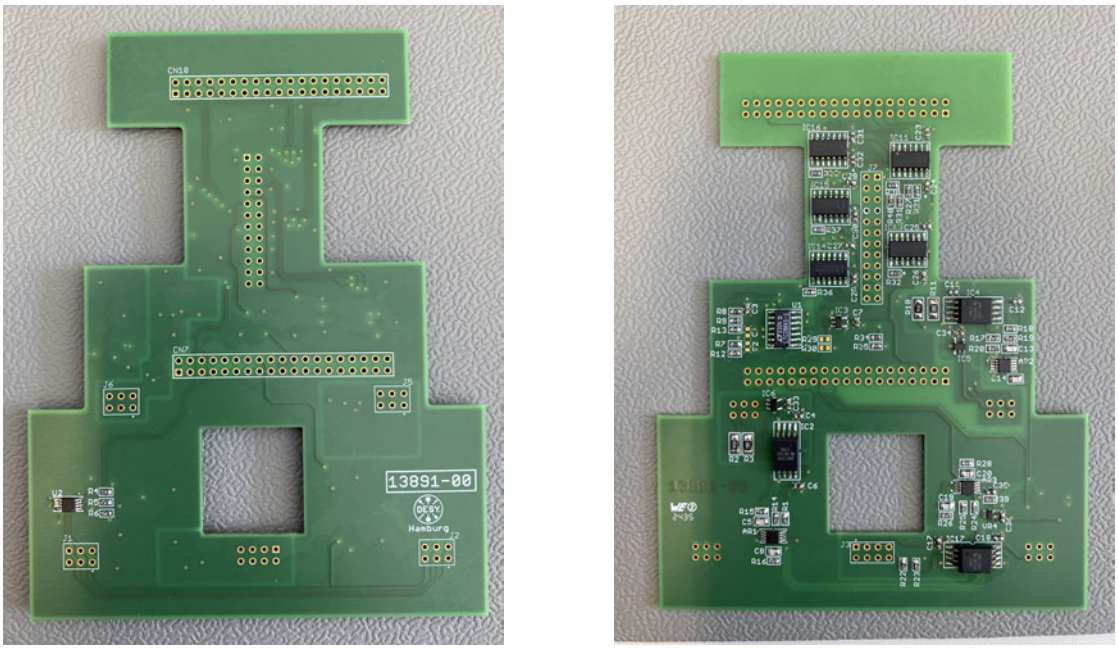


Abbildung 5.12: Finale Platinendesign in Eagle

Die Platine weist vier Lagen auf. Der Platinenumriss besitzt eine maximale Breite von 100 nm und eine maximale Länge von 121.20 nm und erfüllt somit die Anforderung an die Maße der Platine. Die Anforderung an die Tiefe muss zusammen mit der Hot-Swap-Platine und dem aufgesteckten Mikrokontrollerboard betrachtet werden. Die Umrisse und Aussparungen in dem Layout der Mikrokontrollerplatine sind durch die Hot-Swap-Platine und der darauf verbauten Komponenten vorgegeben. Dort wo die Bauteile höher sind, als die Steckverbindung der Platinen zueinander, müssen auf der Mikrokontroller-Platine Aussparungen eingeplant werden. Die Positionen der Steckverbinder sind ebenfalls durch die Hot-Swap-Platine vorgegeben.Alle Bauteile wurden auf dem Top-Layer platziert, um die Handhabung zu erleichtern. Beim Zusammenbau der Platinen befinden sich die Bauteile jedoch auch auf der Unterseite. Die Bauteile sind möglichst nahe an den Steckverbindern platziert, wo die entsprechenden Signale für die Schaltung übertragen werden. Aufgrund der vorgegebenen Positionierung der Steckverbinder müssen einige Leiterbahnen länger gestaltet werden. Um Störungen zu vermeiden sind hier Kondensatoren möglichst nah an den Eingangspins der Bauteile zu platzieren.

In den Abbildungen 5.13 ist die fertig bestückte Platine des ersten Prototyps zu erkennen.



(a) Vorderseite (b) Rückseite

Abbildung 5.13: Bestückte Mikrokontrollerplatine des ersten Prototyps

5.2 Software

In diesem Kapitel wird die Umsetzung der Software beschrieben. Hierbei sollen einige markante Implementierungspunkte erklärt werden. Der gesamte Softwarecode ist im Anhang unter dem Punkt A.5 einzusehen.

5.2.1 Übersicht über die Programmstruktur

In diesem Kapitel wird eine Übersicht über die komplette Programmstruktur gegeben. In der Abbildung 5.14 wird die Struktur grafisch dargestellt.

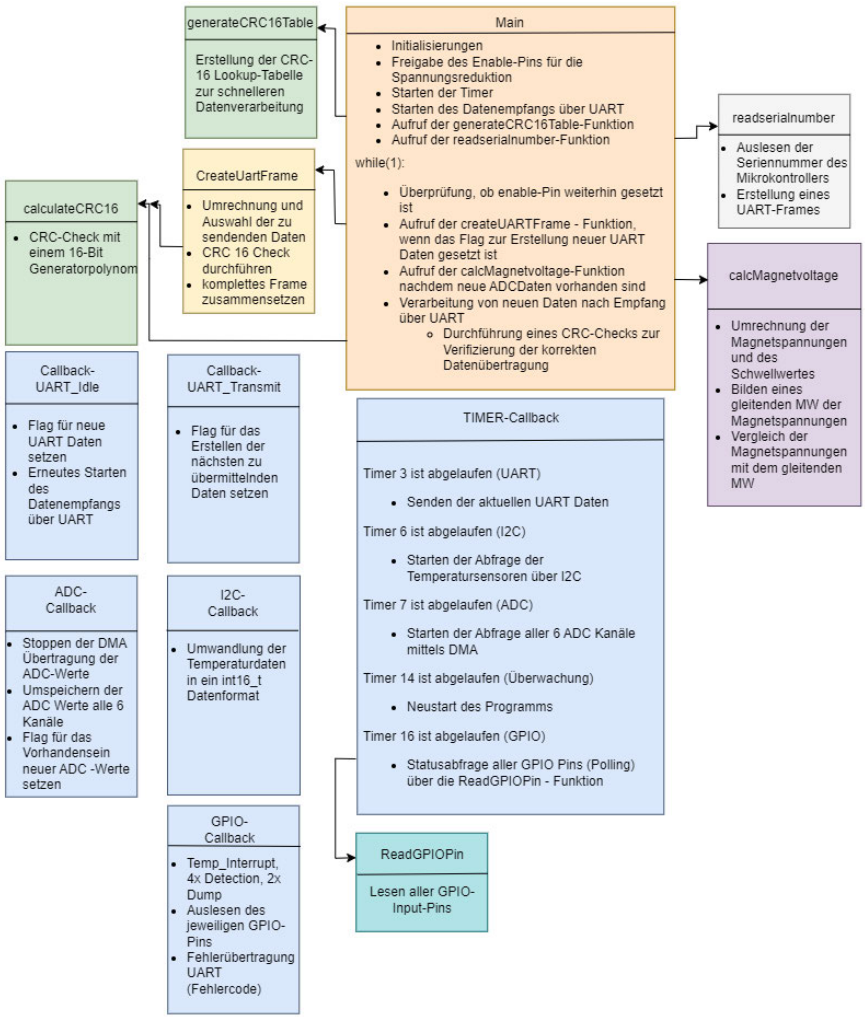


Abbildung 5.14: Übersicht über die Softwarestruktur

Zuerst wird in diesem Abschnitt die Aufgabe der Main-Funktion vorgestellt. Die Main ist in der Abbildung in orange dargestellt. Innerhalb der Main werden zunächst alle Initialisierungen durchgeführt. Anschließend wird der enable-Pin für die Pegelkonverter-Bauteile auf High gesetzt und die vorhandenen Timer gestartet. Es erfolgt das Erstellen der CRC-Lookup-Tabelle (grün) und das Abfragen der Seriennummer des Mikrokontrollers (grau).
Innerhalb der while(1)-Schleife werden verschiedene Flags abgefragt. Zunächst folgt die Abfrage, ob neue ADU-Werte zur Verarbeitung bereit stehen. Bei positiver Abfrage wird die Funktion calcMagnetvoltage() aufgerufen (lila). Innerhalb dieser Funktion werden die ADU Werte umgerechnet und der Schwellwertvergleich der Magnetspannungen durchgeführt.
Ein weiteres Flag dient zum Überprüfen, ob ein neuer UART-Frame erstellt werden kann. Hier wird bei einem Wert von Eins des Flag die Funktion CreateUartFrame() aufgerufen (gelb). Innerhalb von dieser Funktion werden die über UART zu versendenden Daten formatiert und in ein Array gespeichert.
Zuletzt wird das Flag für das Vorhandensein von neuen über UART empfangenen Daten überprüft. Liegen neue Daten zur Verarbeitung vor, wird zunächst ein CRC-Check durchgeführt. Ist das Ergebnis Null und der Check somit positiv, werden die Daten entsprechend ihrer Funktion verarbeitet. Ist das Ergebnis des Checks hingegen ungleich Null, wird ein Fehlerbit gesetzt, welches bei der nächsten UART-Übertragung zu versenden ist.

Wie bereits im Konzept und dem Kapitel 4.2.5 beschrieben, wird die Datenabfrage durch verschiedene Timer durchgeführt. Nach Ablauf des Counters eines Timers, wird ein Interrupt ausgelöst, welcher wiederum den Timer-Callback (blau) aufruft. Dieser Callback wird von allen aktiven General Purpose Timern aufgerufen. Somit muss innerhalb des Callbacks bei der Verwendung von mehreren Timern zunächst abgefragt werden, von welchem Timer der Interrupt ausgelöst wurde.
Zum Erfassen der verschiedenen Daten sind drei unterschiedliche Timer vorgesehen. Diese Timer sind bereits in dem Kapitel 4.2.5 konzeptioniert worden. Mithilfe des Timers sechs werden die aktuellen Temperaturdaten in zyklischer Reihenfolge von den drei Sensoren abgeholt. Durch den Timer sieben wird eine neue DMA Übertragung gestartet, um die aktuellen ADU Werte aller sechs Kanäle zu erhalten. Der Timer 16 führt beim Auftreten des Timer-Callbacks die Funktion ReadGPIOPin (türkis) aus. In dieser Funktion werden mithilfe von Polling alle vorhandenen GPIO-Input-Pins auf ihren aktuellen Status abfragt.
Zusätzlich zu den drei Timern zur regelmäßigen Datenabfrage werden zwei weitere Timer

benötigt. Der Timer 3 wird verwendet, um regelmäßig nach dessen Ablauf das Senden eines neuen Datenframes über UART zu garantieren. So kann eine regelmäßige Übermittlung der Daten ermöglicht werden. Der Timer 14 dient als Überwachungstimer und wird nach jedem Senden der UART Daten zurückgesetzt. Sollte dieser Timer nicht vor seinem Ablauf zurückgesetzt werden, erfolgt ein Systemreset, da ein annehmbarer Fehler innerhalb der Software vorliegt.

Sowohl bei Interrupts als auch bei der Nutzung des DMA ist es möglich, Callbacks beim Abschluss bestimmter Ereignisse auszuführen. Alle Callbacks werden in der Abbildung in blau dargestellt. Bei I^2C wird der I^2C Callback nach Beendigung der Datenübermittlung vom Slave zum Master aufgerufen. Innerhalb des Callbacks sind die ausgelesenen Temperaturdaten der Sensoren in ein geeignetes Datenformat umzuwandeln.
Der ADU-Callback wird aufgerufen, wenn alle Kanäle des ADUs abgefragt und die Werte über den DMA in den vorgesehenen Speicher abgelegt sind. Innerhalb des Callbacks muss die DMA Übertragung gestoppt werden, da diese ansonsten dauerhaft neue Daten übermitteln würde, bevor die alten Daten verarbeitet sind. Ebenso müssen die Werte für die weitere Nutzung in ein weiteres Array um gespeichert werden. Um die Callbackroutine möglichst kurz zu halten, erfolgt die Verarbeitung der Daten innerhalb der Main. Um zu signalisieren, dass neue ADU Daten vorhanden sind und ausgewertet werden können, wird ein Flag auf High gesetzt.
Für die hoch priorisierten digitalen Signale, welche an GPIO-Pins anliegen, sind ebenfalls Interrupts vorgesehen. Bei einer steigenden oder fallenden Flanke wird die GPIO-Callbackroutine ausgelöst. In dieser Routine wird zunächst der Interrupt auslösende Pin bestimmt. Anschließend wird bei der nächsten UART Übertragung ein abweichender Frame versendet, welcher die Änderungen in den digitalen Signalen mitteilt. Bei der Implementierung des ersten Prototyps der Platine unterlief der Fehler, dass die zu überwachenden Signale über GPIO-Interrupts an Pins angeschlossen wurden, die nicht alle Interrupt-Funktionalität unterstützen. Jeder Interrupt-Kanal kann nur einem Pin zugewiesen werden, sodass beispielsweise PA6 und PB6 nicht gleichzeitig Interrupts auslösen können, wenn sie denselben Interrupt-Kanal nutzen. Dieser Umstand ist bei der Erstellung des ersten Prototyps nicht bedacht worden und wurde im zweiten Prototypen behoben. Da der zweite Prototyp jedoch zum Zeitpunkt des Abschlusses dieser Arbeit noch nicht verfügbar war und die Software nur mit dem ersten Prototypen getestet werden konnte, wurden die GPIO-Interrupts eingeplant, aber nicht vollständig optimiert.
Zuletzt sind noch zwei Callbackroutinen für die UART Übertragung vorhanden. Die Callbackroutine UART_Transmit wird ausgelöst, wenn alle Daten aus dem vorgesehe-

nen Buffer übertragen wurden. Innerhalb der Routine wird ein Flag gesetzt. Dieses Flag zeigt an, dass die Übertragung erfolgreich abgeschlossen worden ist und die nächsten Daten für die darauffolgende UART-Übertragung in den Buffer gespeichert werden können. Der zweite UART Callback dient zum Empfang von Daten über die UART-Verbindung. Er wird durch den Aufruf der Funktion HAL_UARTEx_ReceiveToIdle_DMA ermöglicht. Mithilfe von dieser Funktion wird der Callback UART_Idle aufgerufen, sobald in der UART-Übertragung eine Pause von der Länge eines einzelnen Zeichens auftritt. Dies ermöglicht die sofortige Verarbeitung der empfangenen Daten. Innerhalb des Callbacks werden die empfangenen Daten in ein weiteres Array gespeichert, der Inhalt des Empfangsarrays gelöscht und der Empfang für neue UART-Daten erneut gestartet. Es wird ein Flag gesetzt, dass neue empfangene UART-Daten zur Auswertung bereit stehen. Die Auswertung erfolgt innerhalb der Main.

5.2.2 Auslesen der Temperaturdaten

Innerhalb von diesem Unterkapitel wird erläutert, wie die Temperatursensoren nacheinander abzufragen und die Daten anschließend auszuwerten sind.
Die Datenabfrage der Sensoren über I^2C wird in dem Callbacks des Timers sechs gestartet. Innerhalb des I^2C-Callbacks werden die empfangenen Temperaturdaten verarbeitet. In der Abbildung 5.15 ist ein Ausschnitt aus dem Datenblatt des Temperatursensors erkennbar. Es wird das Format des Temperaturregisters dargestellt, aus dem die Temperaturdaten zur Übermittlung ausgelesen werden. Die Temperaturdaten werden in dem Register in zwei 8-Bit Blöcke unterteilt. Dabei enthalten die niedrigsten sieben Bits keine relevante Information und sind bei der Auswertung der Temperaturdaten vernachlässigbar.

Table 7. Temperature register format

Bytes	HS byte								LS byte							
Bits	MSB	TMSB							TLSB							LSB
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STLM75	TD8 (Sign)	TD7 (TMSB)	TD 6	TD 5	TD 4	TD 3	TD 2	TD 1	TD0 (TLSB)	0	0	0	0	0	0	0

Abbildung 5.15: Auszug aus dem Datenblatt: Temperatur Register

In dem Auszug aus dem Softwarecode 5.1 ist die Codezeile für die Verarbeitung der Temperaturdaten des Sensors 1 dargestellt. Die aus dem Register ausgelesenen Temperaturdaten sind in dem Array temperature_data gespeichert. Es ist zu erkennen, wie der erste 8 Bit-Block um acht Stellen nach links verschoben wird und anschließend der zweite 8-Bit Block rechts an den ersten-Block angefügt wird. Es folgt ein Shiften um sieben Stellen nach rechts, um die sieben unrelevanten Bits zu verwerfen.

```
1      temp_sensor1 = (temperature_data[0][0] << 8 | temperature_data[0][1])
    >> 7;
```

Quellcode 5.1: Umrechnung des Temperaturwerts

5.2.3 Umrechnung und Auswertung der ADU Werte

In diesem Abschnitt wird die Auswertung der von dem ADU ausgelesenen Daten erläutert. Hierbei werden die Daten der Magnetspannungen, sowie der drei zu überwachenden Versorgungsspannungen innerhalb der Funktion calcMagnetvoltage() verarbeitet.

Um eine möglichst hohe Ausführungsgeschwindigkeit bei der Datenverarbeitung zu erhalten, werden bei der Auswertung der ADU-Werte nur Ganzzahloperationen durchgeführt. Dies hat zur Folge, dass Nachkommastellen bei der Berechnung wegfallen. Da die Magnetspannungen möglichst genau ausgewertet und übertragen werden sollen, ist dies fatal. Aus diesem Grund wird eine Skalierung der Daten mithilfe von Zweierpotenzen vorgenommen. Jeder Variablen wird eine Bezeichnung angehängt, die den Skalierungsgrad angibt (P10, P12, P14....). Die Bezeichnung steht dabei für die entsprechenden Zweipotenzen, mit denen die Variable skaliert wird. Beispielsweise steht P10 für eine Skalierung mit 2^{10}.
Um ein Beispiel zu geben, wird eine empfangene Magnetspannung von 12.45 V angenommen. Diese Spannung soll nun beispielsweise mit einem Faktor von 3.2 multipliziert werden. Dies würde eine Spannung von 39.84 V ergeben. Da in dem Programm nur mit Ganzzahlen gerechnet wird, würde die Multiplikation von 12 V mit 3 einen Wert von 36 V ergeben. Dies wäre eine Abweichung von 3.84 V vom korrekten Ergebnis. Zum Vergleich wird nun die Berechnung mit einer P10 Skalierung durchgeführt. Hierfür werden die Werte zunächst in eine P10 Skalierung umgewandelt, anschließend wird die Berechnung mit

Ganzzahlen durchgeführt und das Ergebnis wieder in eine P1 Skalierung zurückgeführt.

$$\text{Magnetspannung\_P10} = 12.45\,\text{V} \cdot 2^{10} = 12748.8\,\text{V}$$

$$\text{Faktor\_P10} = 3.2 \cdot 2^{10} = 3276.8$$

$$\begin{aligned}\text{Ergebnis\_P10} &= \frac{\text{Magnetspannung\_P10} \cdot \text{Faktor\_P10}}{2^{10}} \\ &= \frac{12748\,\text{V} \cdot 3276}{2^{10}} = 40783.64\,\text{V}\end{aligned}$$

$$\begin{aligned}\text{Ergebnis\_P1} &= \frac{\text{Ergebnis\_P10}}{2^{10}} \\ &= \frac{40783\,\text{V}}{2^{10}} = 39.82\,\text{V}\end{aligned}$$

Es ist zu erkennen, dass der mit der P10 Skalierung berechnete Wert nur 0.02 V von dem korrekten Wert abweicht. Somit kann gezeigt werden, dass durch die Anwendung der Skalierung eine viel höhere Genauigkeit in der Berechnung erreicht wird. Ein weiterer Vorteil stellt das Teilen durch die Zweierpotenzen innerhalb der Softwareimplementierung dar. Diese kann durch einfache Shift-Operationen durchgeführt werden.

Innerhalb des Quellcodeausschnitts 5.2 ist ein Teil der Funktion calcMagnetvoltage() dargestellt. Da der ADU eine Auflösung von 12 Bit besitzt, weisen alle ausgelesenen ADU-Werte bereits eine P12 Skalierung auf. Um die jeweiligen Skalierungsfaktoren zu den Spannungen hinzuzurechnen, sind die Faktoren innerhalb von Variablen mit einer P12 Skalierung vorberechnet.

```
1  int Faktor34_P12 = 13926; //3.4 * 2^12
2  int RefSpannung15_P12 = 6144; //1.5 * 2^12
3  int SkalierungMagnet_P10 = 25600; // (10/0.4)*2^10
4  int SkalierungPWM_P12 = 1987; // (5/3.3) * 1/10 *3.4 * 2^12
5  int SkalierungVCC_P12 = 22282;  //16/10 * 3.4 * 2^12
6  int Skalierung12V_P12 = 55705; // 4* 3.4 * 2^12
7  int Skalierung12VPS_P10 = 8960; //(3.5/0.4)*2^10
8
9  ADC_M1_P12 = ((ADC_M1 * Faktor34_P12) >> 12) - RefSpannung15_P12;
10 ADC_M1_P10 = ((ADC_M1_P12 >> 2) * SkalierungMagnet_P10) >> 10;
11 ADC_M2_P12 = ((ADC_M2 * Faktor34_P12) >> 12) - RefSpannung15_P12;
12 ADC_M2_P10 = ((ADC_M2_P12 >> 2) * SkalierungMagnet_P10) >> 10;
```

Quellcode 5.2: Skalierung der Magnetspannungen

Die Formeln zur Berechnung der einzelnen Spannungen werden aus der Implementierung der Schaltungen bestimmt. Hierbei sind die verschiedenen Verstärkungsfaktoren und Referenzspannungen zu berücksichtigen. Die Mittelwertbildung der Magnetspannungen für den Vergleich mit dem Schwellwert wird durch eine exponentielle Glättung durchgeführt. Hierbei wird der aktuelle gemessene Spannungswert zu dem vorherigen Wert hinzu addiert. Beide Werte sind mit einem Faktor Alpha versehen. Mithilfe dieses Faktors kann die Einflussstärke des neuen Werts zum neuen Mittelwert definiert werden. In der Implementierung ist der Faktor Alpha mit 0.5 gewichtet. Der betreffende Codeausschnitt ist im Quellcodeausschnitt 5.3 zu erkennen.

```
1  ADC_M2_MW_P10 = (ADC_M2_P10 >> 1) + (ADC_M2_MW_P10 >> 1);
```

Quellcode 5.3: Ermittlung des gleitenden Mittelwertes

Die Bestimmung der Schwellwerte erfolgt durch Addition bzw. Subtraktion des aus dem PWM-Signal bestimmten Schwellwerts von dem aktuellen Mittelwert. Wenn einer der Vergleiche des aktuellen Wertes mit den berechneten Schwellwerten positiv ausfällt, wird mit der nächsten UART-Übertragung eine Fehlermeldung übermittelt. Diese Meldung beinhaltet den Header 0x02 und die Werte der zwei Magnetspannungen. Im übergeordneten System sind die übertragenen Spannungen auf Auffälligkeiten zu überprüfen.

5.2.4 Erstellung des UART Frames

In diesem Kapitel wird der geschriebene Softwarecode zur Erstellung des UART Frames genauer beschrieben.
Um eine reibungslose Übertragung der UART-Daten zu ermöglichen, werden zwei verschiedene Datenbuffer erstellt. Solange die Daten des einen Datenbuffers versendet werden, wird der zweite Datenbuffer mit neuen Daten gefüllt. Mithilfe von dieser Technik soll der Abstand zwischen zwei UART-Übertragung möglichst kurz gehalten werden. Umgesetzt wird die Erstellung des UART-Frames innerhalb der Funktion CreateUART-Frame().
Zunächst werden die P10 skalierten Magnetspannungen auf eine P9 Skalierung umgerechnet. Dies ist notwendig, da für die Übertragung der Magnetspannungen zwei 8-Bit Datenpakete vorgesehen sind. Innerhalb der Implementierung wurde festgestellt, dass die Magnetspannungen in P10 Skalierung den darstellbaren Zahlenbereich von 16-Bit übersteigen. Eine Magnetspannung von 30 V würde eine P10 Darstellung von 35840 V benötigen. Der darstellbare Zahlenbereich einer signed 16-Bit int-Zahl kann nur positive

Zahlen bis $2^{15} - 1$ darstellen. Dies ergibt eine maximale positive Zahl von 32767. Somit liegen die 35840 außerhalb des darstellbaren Bereichs. In der P9 Skalierung können alle erwartbaren Magnetspannungen dargestellt werden.
Anschließend erfolgt das Zusammensetzen des Frames. Innerhalb des Quellcodeausschnitts 5.4 wird die Zuweisung der Daten zu den ersten sechs Datenpaketen dargestellt. Bei dem Aufteilen der Magnetspannungen in zwei 8-Bit-Blöcke wird das MSB dem ersten Block zugeordnet.

```
1    // Header fuer normale UART Datenuebertragung
2    dataBufferUARTTx1[0] = 0x01;
3    // Einfuegen der Magnetspannungen
4    // MSB zuerst
5    dataBufferUARTTx1[1] = (uint8_t) ((ADC_M1_P9 >> 8) & 0xFF);
6    dataBufferUARTTx1[2] = (uint8_t) (ADC_M1_P9 & 0xFF);
7    dataBufferUARTTx1[3] = (uint8_t) ((ADC_M2_P9 >> 8) & 0xFF);
8    dataBufferUARTTx1[4] = (uint8_t) (ADC_M2_P9 & 0xFF);
9    // Fehlermeldungen
10   dataBufferUARTTx1[5] = 0x00;
```

Quellcode 5.4: Erstellung des UART-Frames

Die Auswahl der weiteren zu übertragenden Daten wird durch einen Identifier vorgenommen. Die Auswahl erfolgt mithilfe einer switch-case Anweisung. Die Zuweisung des Identifiers zu den Daten kann in dem Konzept unter dem Kapitel 4.2.4 nachgelesen werden. In dem Quellcodeausschnitt 5.5 wird der Code beispielhaft für den Identifier 2 dargestellt. Bei dem Identifier 2 werden die Temperaturdaten einer der Sensoren übertragen. Das MSB der Daten wird in dem ersten 8-Bit-Paket übermittelt.

```
1    //Temperatur Sensor 3
2    case 2:
3      dataBufferUARTTx1[6] = identifier[i];
4      dataBufferUARTTx1[7] = (uint8_t) ((temp_sensor3 >> 8) & 0xFF);
5      dataBufferUARTTx1[8] = (uint8_t) (temp_sensor3 & 0xFF);
6      i++;
7      break;
```

Quellcode 5.5: Erstellung des UART-Frames: Auswahl der variablen Daten

Nach der Erstellung des Frames wird der CRC Check durchgeführt. Die Checksumme wird anschließend an die Stellen 9 und 10 des Datenbuffers eingefügt. Der CRC-Check wird im folgenden Kapitel erläutert.

5.2.5 Implementierung des CRC-Checks

Im folgenden wird die Implementierung des CRC-Checks genauer betrachtet.
Es wird der CRC-16-CCITT-Check zur Überwachung der UART-Daten implementiert. Dieser Check verwendet das Generatorpolynom $x^{16} + x^{12} + x^5 + 1$, was hexadezimal 0x1021 entspricht. Als Initialwert wird 0xFFFF verwendet. Es wird keine Reflexion vorgenommen. Die Bitreihenfolge wird somit nicht verändert.
Der CRC-Check wird in zwei Funktionen aufgeteilt. Um eine höhere Geschwindigkeit bei der Bestimmung des CRCs zu erreichen, wird zu Beginn des Programm eine Lookup-Tabelle erstellt. Innerhalb dieser Tabelle werden die CRC- Werte für alle 256 möglichen Eingangsbytes gespeichert. Dies spart bei der Verwendung Zeit, da nur ein Speicherzugriff und eine XOR-Operation bei der Berechnung auszuführen sind. Der verwendete Code für die Erstellung der Lookup-Tabelle ist in dem Quellcodeausschnitt5.8 zu erkennen. Der Code basiert auf der Logik der Quelle [1].

```
1  void generateCRC16Table(void) {
2    for (int i = 0; i < 256; i++) {
3      uint16_t crc = i << 8;
4      for (int j = 0; j < 8; j++) {
5        if (crc & 0x8000) {
6          crc = (crc << 1) ^ CRC_POLYNOM;
7        } else {
8          crc <<= 1;
9        }
10     }
11     CRC16_Table[i] = crc;
12   }
13 }
```

Quellcode 5.6: Funktion zur Erzeugung der CRC-Tabelle

In der äußersten for-Schleife (Zeile 2) wird über alle möglichen 8-Bit Eingangsbytes iteriert (0 bis 255). Jedes Eingangsbyte wird in die oberen 8 Bits der 16-Bit CRC-Variablen geschoben. Die unteren 8 Bits der CRC-Variablen werden dabei mit Nullen aufgefüllt. In der inneren for- Schleife (Zeile 4) wird die Polynomdivision durchgeführt, indem die Bits des aktuellen Eingangsbytes nacheinander verarbeitet werden. In jeder Iteration wird geprüft, ob das MSB von CRC auf 1 gesetzt ist (Zeile 5). Wenn das MSB 1 ist, wird die CRC-Variable nach links geschoben und mit dem CRC-Generatorpolynom 0x1021 (CRC_POLYNOM) XOR-verknüpft. Ist das MSB 0, wird nur ein Linksshift ausgeführt.

Nach 8 Iterationen der inneren Schleife ist das aktuelle Eingangsbyte vollständig verarbeitet worden und die CRC-Variable enthält den berechneten CRC-Wert für dieses Byte. Dieser Wert wird in der Lookup-Tabelle (CRC16_Table) gespeichert. Der gesamte Vorgang wird in der äußeren Schleife für das nächste Eingangsbyte wiederholt.
In einer zweiten Funktion wird der tatsächliche CRC-Check für die entsprechenden Eingangsdaten durchgeführt. Die Funktion ist innerhalb des Quellcodeausschnitts 5.7 dargestellt.

```
1  uint16_t calculateCRC16(uint8_t *data, uint16_t length) {
2
3    uint16_t crc = 0xFFFF;  // Initialwert
4
5    for (uint16_t i = 0; i < length; i++) {
6      uint8_t byte = data[i];
7      crc = (crc << 8) ^ CRC16_Table[(crc >> 8) ^ byte];
8    }
9
10   return crc;
11 }
```

Quellcode 5.7: Funktion zur Durchführung des CRC-Checks

Die Funktion erhält als Übergabewerte die zu verarbeitenden Daten und die Anzahl der 8-Bit-Pakete. In der Zeile 3 wird zunächst der Startwert initiiert. Anschließend wird eine for-Schleife über die angegebene Anzahl an Datenpaketen ausgeführt. Hierbei wird das aktuelle Datenpaket in die lokale Variable byte gespeichert. Anschließend wird der CRC-Wert aktualisiert, indem die oberen 8 Bits des aktuellen CRC-Werts mit dem aktuellen Datenpaket XOR-verknüpft werden. Dieser Wert dient als Index, um den entsprechenden Eintrag in der Lookup-Tabelle abzurufen. Der CRC-Wert wird um 8 Bits nach links verschoben und mit dem Tabellenwert XOR gerechnet, um das Ergebnis zu aktualisieren. Am Ende der Schleife enthält die Variable crc die berechnete Prüfsumme, die von der Funktion zurückgegeben wird.

5.2.6 Senden der UART Daten

Das Senden der UART-Daten wird innerhalb des Callbacks des Timers 3 gestartet. Hierbei wird mithilfe einer if-Abfrage unterschieden, welche UART-Daten versendet werden sollen. Innerhalb des Quellcodeausschnitts 5.8 ist der entsprechende Ausschnitt aus dem Timer-Callback zu erkennen.

```
1  if (htim->Instance == TIM3) {
2    if (UARTTxBufferSelection == 0x02 && sendSerialnumber != 1
3        && MagnetProblem != 1) {
4      HAL_UART_Transmit_DMA(&huart1, (uint8_t*) dataBufferUARTTx1, 11);
5    }
6    else if (UARTTxBufferSelection == 0x01 && sendSerialnumber != 1
7        && MagnetProblem != 1) {
8      HAL_UART_Transmit_DMA(&huart1, (uint8_t*) dataBufferUARTTx2, 11);
9    }
10   else if (sendSerialnumber == 1 && MagnetProblem != 1) {
11     HAL_UART_Transmit_DMA(&huart1, (uint8_t*) serialNumberUART, 13);
12   }
13   else if (MagnetProblem == 1) {
14     HAL_UART_Transmit_DMA(&huart1, (uint8_t*) MagnetUART, 5);
15   }
16 }
```

Quellcode 5.8: Auswahl der zu sendenden UART-Frames

Innerhalb der Zeile 2 und der Zeile 6 erfolgt die Abfrage für das Senden der Messdaten an die Hot-Swap-Platine. In der Zeile 2 wird dabei der Datenbuffer 1 und in Zeile 6 der Datenbuffer 2 gesendet. Der Nutzen der beiden Buffer wurde bereits beschrieben. Gesendet werden dürfen die Daten nur, wenn kein Fehler innerhalb der Magnetspannungen gefunden wurde und die Seriennummer nicht übermittelt werden soll. Zur Abfrage dienen die Flags sendSerialnumber und MagnetProblem. Das Senden der Seriennummer wird in der Abfrage in Zeile 10 ermöglicht, jedoch nur, wenn die Magnetspannungen den Schwellwert nicht über- oder unterschritten haben. In diesem Fall wird eine Fehlermeldung an die Hot-Swap-Platine gesendet (Zeile 13), welche die höchste Sendepriorität aufweist.

6 Verifikation der Implementierung

In diesem Kapitel werden einige grundlegende Tests durchgeführt, um die korrekte und funktionsfähige Implementierung der Hardware und der Software zu verifizieren.

6.1 Hardware

Zuerst werden in diesem Kapitel einige Test durchgeführt, um die Funktion der implementierten Hardwareschaltungen zu bestätigen. Hierbei wird zunächst die Schaltung zur galvanischen Trennung, die Schaltung zur UART-Übertragung und anschließend die I^2C Übertragung zur Datenabfrage der Temperatursensoren getestet. Des Weiteren sind alle vorhanden Spannungen auf der Platine mithilfe eines Multimeters zu verifizieren. Alle Oszilloskop Bilder sind mithilfe des Tablet Oszilloskops TO2004 von Micsig erstellt worden.

6.1.1 Ermittlung des Leistungsverbrauchs der Platine

Eine zu erfüllende Anforderung ist es, dass die Platine einen Verbrauch von maximal 1 W aufweist. Um den Leistungsverbrauch zu ermitteln wird an die Platine des ersten Prototypen eine 12 V Versorgungsspannung angelegt und der Stromverbrauch mithilfe eines Multimeters gemessen. Es kann ein Stromverbrauch von 0.09 A festgestellt werden. Im folgenden wird der resultierende Leistungsverbrauch bestimmt.

$$\begin{aligned} P &= U \cdot I \\ &= 12\,V \cdot 0.09\,A \\ &= 1.08\,W \end{aligned} \tag{6.1}$$

Es wird ein Leistungsverbrauch von 1.08 W bestimmt. Somit liegt der bestimmte Wert 80 mW über der angegebenen Grenze. Um die geforderte Anforderung zu erfüllen, muss der Energieverbrauch des zweiten Prototyps optimiert werden.

6.1.2 Schaltungen zur galvanischer Trennung

In diesem Unterkapitel wird die entwickelte Schaltung zur galvanisch getrennten Messung von Spannungen verifiziert. Hierfür wird die Schaltung zur Messung der Magnetspannung M1 auf korrekte Funktion getestet. Da die drei auf der Platine vorhandenen Schaltungen zur galvanischen Trennung nahezu identisch sind, wird in diesem Kapitel nur eine Schaltung genauer betrachtet.
Zunächst wird an die Eingänge M1+ und M1- eine Spannung angelegt und mithilfe eines Multimeters die Ausgangsspannung am Optokoppler gemessen. Dabei wird der Eingang M1- auf GND gelegt. Der am Ausgang gemessene Wert wird anschließend mit dem erwarteten, berechneten Wert verglichen. Bei einer Eingangsspannung von 5 V an M1+ wird eine Ausgangsspannung ADC_M1 von 1.7 V gemessen. Im folgenden wird der zu erwartende Wert bestimmt. Hierbei wird die an M1+ angelegte Eingangsspannung durch 10 geteilt, was auf den eingebauten Spannungsteiler zurückzuführen ist. Anschließend wird der Verstärkungsfaktor des Trennverstärkers von 0.4 berücksichtigt. Abschließend werden die 1.5 V der Referenzspannung hinzuaddiert.

$$\begin{aligned} V_{\text{Erwartung}} &= \frac{V_{\text{M1+}}}{10} \cdot 0.4 + 1.5\,\text{V} \\ &= 1.7\,\text{V} \end{aligned} \tag{6.2}$$

Bei einer angelegten Eingangsspannung von 5 V an M1+ wird ein berechnete Spannung von 1.7 V an dem Optokopplerausgang für ADC_M1 erwartet. Der gemessene und der erwartete Wert stimmen somit überein. Dieser Vorgang wird für weitere Eingangsspannungen an M1+ wiederholt. Hierbei kann keine signifikante Abweichung des gemessenen und des zu erwartenden Werts zueinander festgestellt werden.

In einem weiteren Test werden das an die Schaltung angelegte Eingangssignal und das Ausgangssignal gegenüber gestellt. Hierfür wird mithilfe eines Oszilloskops der Spannungsverlauf beider Signale grafisch dargestellt. Das Oszilloskopbild ist in der Abbildung 6.1 dargestellt. In gelb wird das Eingangssignal des Trennverstärkers angezeigt. Dabei ist zu beachten, dass dieses Signal nach dem Spannungsteiler und somit nach der Reduzierung um den Faktor 10 angezeigt wird. Die Werte der tatsächlich angelegten Eingangsspannung sind somit um einen Faktor von 10 höher. Es wird ein Sinussignal mithilfe eines Funktionsgenerators erzeugt und auf die Schaltung gegeben. In türkis ist das Ausgangssignal des Optokopplers zu erkennen. Dieses Signal wird an den ADU des Mikrokontrollers weitergegeben. In der unteren Zeile des Bildes sind einige gemessene Werte dargestellt.

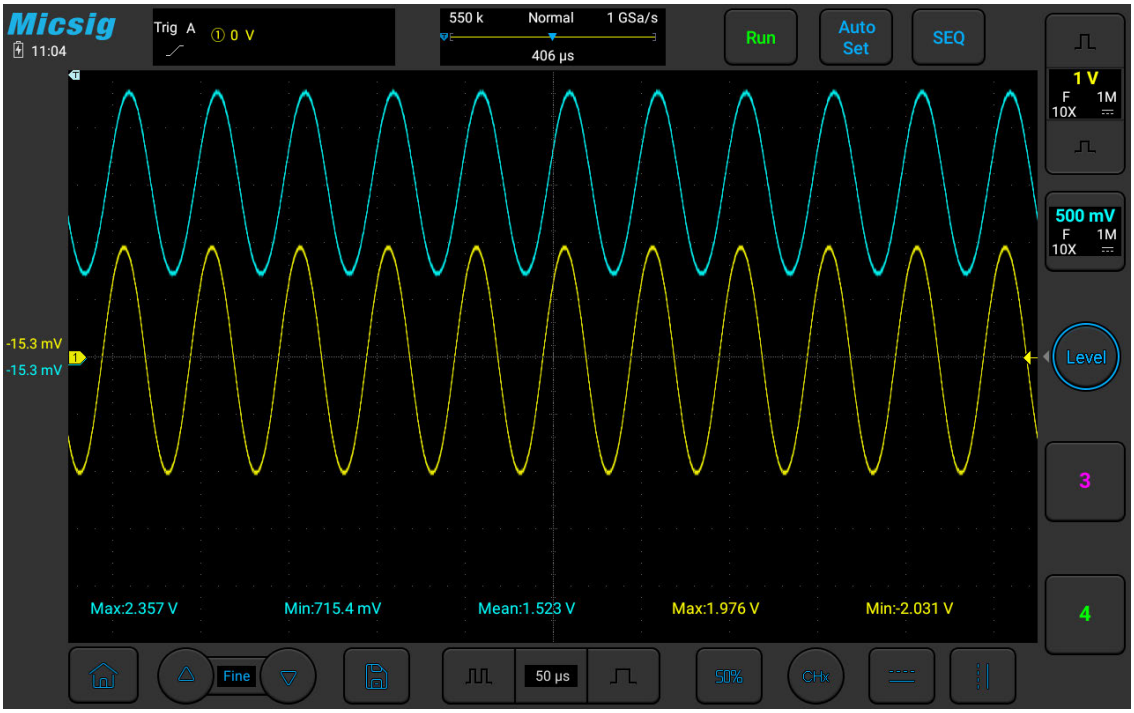


Abbildung 6.1: Oszilloskopbild zur Verifikation der Übertragung der Magnetspannung

Das gelbe Eingangssignal weist eine Sinusspannung mit den maximalen Werten von ±2 V auf. Diese Werte sind zum einen dem Spannungsverlauf des Signals zu entnehmen, zum anderen wird der minimale und der maximale Wert der Spannung als Messwert in der unteren Zeile des Bildes in gelb dargestellt. Dies bedeutet, dass mit der zusätzlichen Verstärkung von 10 eine Eingangsspannung von ±20 V an die Schaltung angelegt ist.
Bei der Betrachtung des türkisen Ausgangssignals kann ein Offset beziehungsweise eine

Verschiebung des Signals in den positiven Bereich festgestellt werden. Der ausgegebene Mittelwert (mean) für dieses Signal weist einen Wert von 1.523 V auf. Diese Verschiebung wird durch die 1.5 V Referenzspannung hervorgerufen. Mithilfe von der Referenzspannung werden die negativen Werte der Eingangsspannung auf einen positiven Wertebereich angehoben, sodass sowohl negativ als auch positive Eingangsspannungen von dem Mikrokontroller erfasst werden können.
Der in türkis erfasste maximale und minimale Spannungswert der Ausgangsspannung muss mit dem zu erwartenden Wert bei einer Eingangsspannung von −20 V beziehungsweise +20 V übereinstimmen. Zur Berechnung der zu erwartenden Werte wird die Formel 6.2 erneut verwendet. Es resultiert ein berechneter Wert von 2.3 V bei einer Eingangsspannung von +20 V und ein Wert von 0.7 V bei einer Eingangsspannung von −20 V. Diese Werte werden mit dem vom Oszilloskop angegebenen minimalen und maximalen Wert der Ausgangsspannung verglichen. Beide berechneten Werte stimmen mit der Angabe des Oszilloskops überein. Somit kann die korrekte Funktion für positive als auch negative Eingangsspannungen nachgewiesen werden.
Des Weiteren kann der Verstärkungsfaktor des Trennverstärkers innerhalb des Oszilloskopbilds nachgewiesen werden. Der Faktor ist in dem Datenblatt mit 0.4 angegeben. Um den Faktor zu bestimmen wird zunächst von dem bestimmten maximalen und minimalen Werten der Ausgangsspannung die 1.5 V Referenzspannung abgezogen. Anschließend wird die Ausgangsspannung durch die Eingangsspannung geteilt, um das Verhältnis zueinander zu bestimmen.

$$\beta_{\max} = \frac{(\mathrm{U}_{\mathrm{Ausgang,max}} - \mathrm{U}_{\mathrm{Ref}})}{\mathrm{U}_{\mathrm{Eingang,max}}} \quad (6.3)$$

$$= \frac{(2.357\,\mathrm{V} - 1.523\,\mathrm{V})}{1.976\,\mathrm{V}}$$

$$= 0.42$$

$$\beta_{\min} = \frac{(\mathrm{U}_{\mathrm{Ausgang,min}} - \mathrm{U}_{\mathrm{Ref}})}{\mathrm{U}_{\mathrm{Eingang,min}}} \quad (6.4)$$

$$= \frac{(715.4\,\mathrm{mV} - 1.523\,\mathrm{V})}{-2.031\,\mathrm{V}}$$

$$= 0.398$$

Der Verstärkungsfaktor entspricht hier für den maximalen Wert 0.42 und für den minimalen Wert 0.398. Dies entspricht einer Abweichung von 5 % beim maximalen Wert und von 0.5 % beim minimalen Wert von der Angabe des Herstellers im Datenblatt. Bei einem vom Hersteller angegebenen typischen Fehler des Verstärkungsfaktors von ±0.05% muss in Zukunft geklärt werden, ob diese gemessene Abweichung durch Messfehler oder durch den vorgenommenen Schaltungsaufbau hervorgerufen wird.

6.1.3 RS485 Transceiver-Schaltung

Mithilfe dieses Tests ist die Schaltung des RS485 Transceivers auf Korrektheit zu überprüfen. Hierfür wird zunächst ein Softwarecode geschrieben, welcher über den Mikrokontroller eine Nachricht über die UART Anbindung versendet. Innerhalb des Hardwareaufbaus wird ein oop erstellt, indem an dem Stecker J7 eine Brücke zwischen den Signalen BUS_OUT_1+ und BUS_IN_1+ sowie BUS_OUT_1- und BUS_IN_1- gesetzt wird. Durch das Setzen dieser Brücke wird das durch den Transceiver übertragene Ausgangssignal wieder als Eingangssignal an den Transceiver übermittelt. Anschließend wird dieses Signal erneut von dem Mikrokontroller empfangen. Bei einer korrekten Funktion wird erwartet, dass das durch den Mikrokontroller gesendete Signal dem wiederum empfangenen Signal entspricht. Es wird ein Oszilloskop an das UART Ausgangssignal des Mikrokontrollers (TX) und an das UART Eingangssignal des Mikrokontrollers (RX) angeschlossen. Es wird erwartet, dass beide Signale identisch sind. In der Abbildung 6.2 ist das Ergebnis der Messung dargestellt. In pink ist das UART-Eingangssignal (UART_TX) und in türkis das UART Ausgangssignal (UART_RX) zu erkennen. In orange werden die übertragenden UART Daten in hexadezimaler Schreibweise angezeigt.

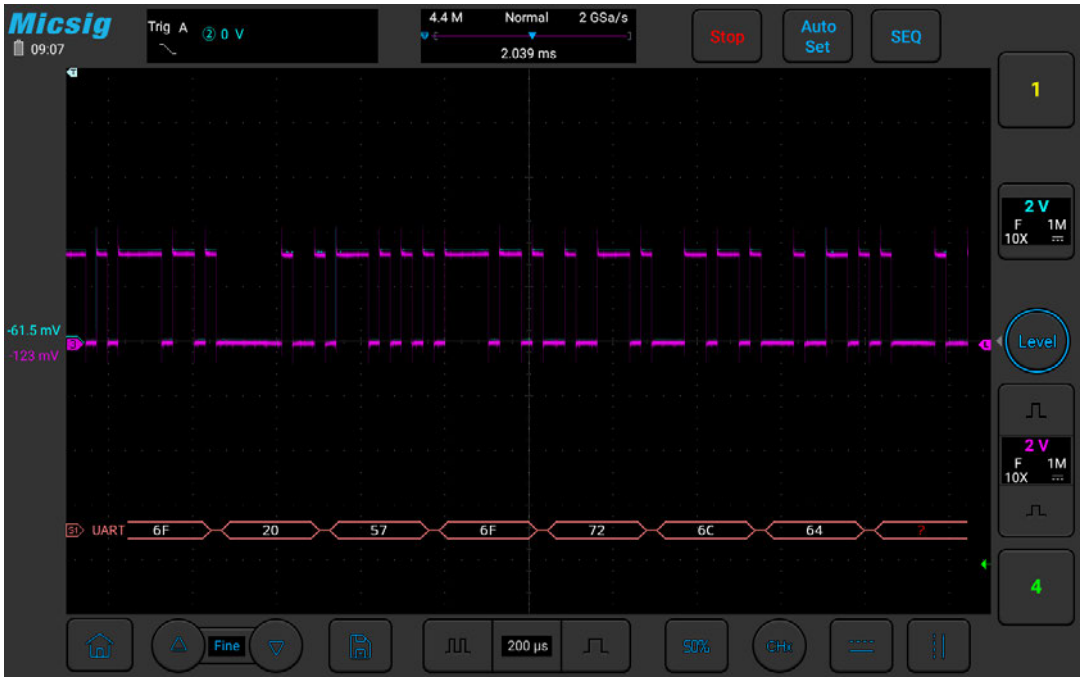


Abbildung 6.2: Oszilloskopbild zur Verifikation der Funktion der RS485 Transceiver Schaltung

Deutlich zu erkennen ist, dass das türkise UART Ausgangssignal von dem pinken UART Eingangssignal nahezu überdeckt wird. Dabei werden beide Signale mit den gleichen Einstellungen auf dem Oszilloskop dargestellt. Dies bedeutet, dass die beiden Signale einen identischen Signalverlauf aufweisen. Hieraus kann wiederum geschlussfolgert werden, dass die vom Mikrokontroller gesendeten Daten identisch wieder zurückgeführt werden.
In diesem Testprogramm wird zur Überprüfung der Verbindung die Nachricht Hello World über UART gesendet. In orange werden die über UART an dem Eingangssignal empfangenen Zeichen hexadezimal dargestellt. Wird der hexadezimale Code in ASCII umgewandelt, steht dort beginnend mit der 6F o World. Dies ist Teil der zum Testen versendeten Nachricht.
Mithilfe von diesem Test kann nachgewiesen werden, dass Daten über die Transceiver Schaltung korrekt versendet und empfangen werden können.

6.1.4 Auslesen der Temperaturdaten mithilfe von I^2C

Durch diesen Funktionstest wird sowohl die Funktion der Hardware als auch indirekt die Funktion der Software überprüft. Mithilfe eines geschriebenen Softwarecodes soll die Datenübermittlung der Temperaturdaten durch I^2C überprüft werden. Hierbei werden die Daten von den drei implementierten Temperatursensoren von der Software über I^2C abgefragt. Die Sensoren weisen dabei die hexadezimalen Adressen 0x48, 0x49 und 0x4A auf. Es wird erwartet, dass die Daten von allen Sensoren an den Mikrokontroller übermittelt werden. Dabei sind die Daten in zwei 8-Bit Datenpakete aufgeteilt.

In der Abbildung 6.3 ist das Oszilloskopbild der I^{2}C Übertragung zu erkennen. In türkis wird SCL, in pink SDA und in orange die I^{2}C Kommunikation angezeigt.

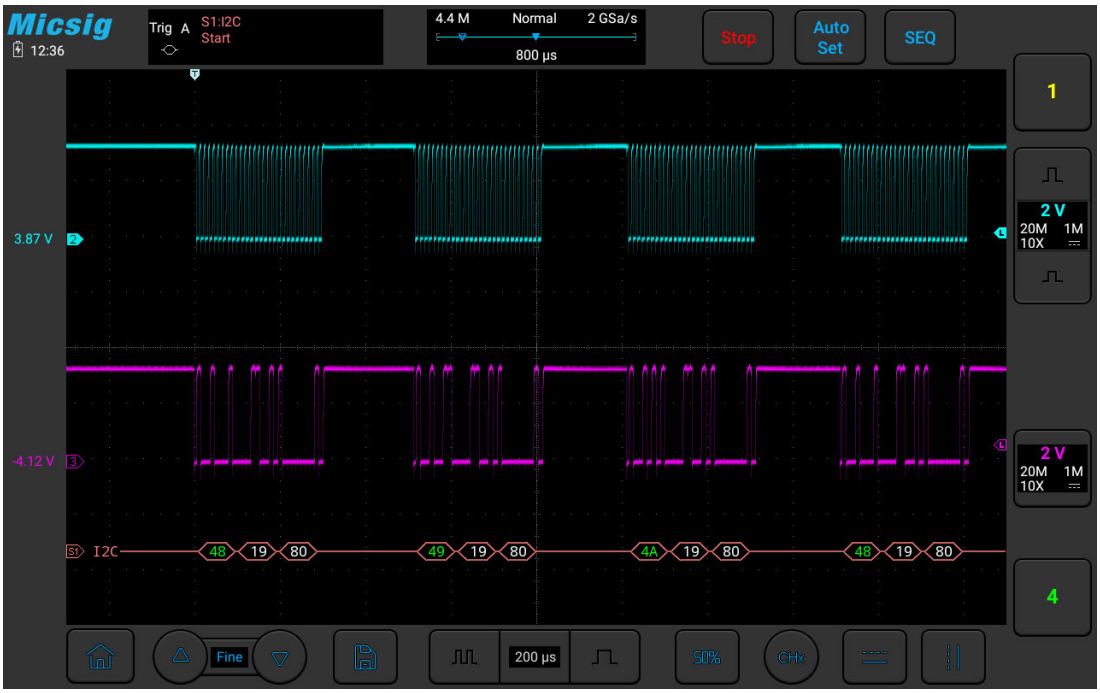


Abbildung 6.3: Oszilloskopbild zur korrekten I^{2}C-Übertragung

Zu erkennen ist, dass alle drei auf den Platinen implementierten Sensoren abgefragt werden. Die Adressen der angesprochenen Sensoren sind in der I^{2}C Übertragung grün ausgeschrieben. Anschließend werden die Temperaturdaten übermittelt. Bei diesem Test liefern alle drei Sensoren die gleiche Temperatur, welche durch die hexadezimalen Zahlen 0x19 und 0x80 übertragen wird. Dies entspricht nach der Umrechnung in einen dezimalen Wert einer Temperatur von 25.5 °C.
Mithilfe von diesem Test kann bewiesen werden, dass alle drei verbauten Temperatursensoren korrekt auf der Hardware implementiert wurden und durch die Software abgefragt werden können.

6.2 Software

Innerhalb von diesem Kapitel werden einige grundlegende Funktionen der Software überprüft. Hierzu zähen das korrekte Umwandeln der ADU-Werte auf die ursprünglichen

Spannungswerte, die Übertragung der Daten mittels UART und die korrekte Ausführung des CRC-Checks.

6.2.1 Umrechnung der ADU Werte

In diesem Unterkapitel soll die Umrechnung der ADU-Werte in die entsprechenden Spannungswerte verifiziert werden. Hierfür ist in der Abbildung 6.4 ein Auszug aus den Debug-Ausgaben des Entwicklungsprogramms zu erkennen.

Expression	Type	Value
ADC_M1	int16_t	1811
ADC_M1_P12	int	-38
ADC_M1_P10	int	-250
ADC_M1_P9	int16_t	-125
ADC_M2	int16_t	2292
ADC_M2_P12	int	1665
ADC_M2_P10	int	10400
ADC_M2_P9	int16_t	5225
ADC_VCC5V	uint16_t	3656
ADC_VCC5V_P10	int16_t	4980
ADC_12VPS	uint16_t	2516
ADC_12VPS_P12	int16_t	2399
ADC_12VPS_P10	int16_t	5241
ADC_12V	uint16_t	3588
ADC_12V_P10	int16_t	12199

Abbildung 6.4: Debugauszug zur Überprüfung der ADU-Werte

Dargestellt sind die beiden Magnetspannungen, die 12 V Spannung der Spannungsversorgungen, die 12 V Versorgungsspannung der Mikrokontrollerplatine und die 5 V VCC-Spannung. Die Spannung des PWM-Signals konnte bis zum Zeitpunkt der Abgabe nicht getestet werden, da die dafür benötigte Schaltung auf dem zweiten Prototypen vorgesehen ist.
An der Magnetspannung M1 liegt zum Zeitpunkt des Tests eine Eingangsspannung von 0 mV an. Für die Magnetspannung M1 wird der Wert 1811 vom ADU ausgelesen. Nach der Umrechnung ergibt sich ein Spannungswert in der P9 Skalierung von −125 V. Umrechnet bedeutet dieser Wert in der P1 Skalierung eine Spannung von −0.24 V. An der Magnetspannung M2 ist eine Eingangsspannung von 10 V angeschlossen. Es wird ein Digitwert von 2292 vom ADU ausgelesen. Dies bedeutet in der P9-Skalierung einen Wert von 5225 V. Umgerechnet ist dies eine Spannung von 10.2 V in der P1-Skalierung. Bei beiden Magnetspannung kann somit eine Abweichung von 0.2 V festgestellt werden.

An das 12 V PS-Signal wird eine Spannung von 5 V angelegt. Es ergibt sich ein P10-Wert von 5241 V. Dies entspricht einem Wert von 5.11 V. Die 12 V Versorgungsspannung der Platine besitzt einen Wert von 12199 in der P10 Skalierung. Dies sind 11.91 V in der P1 Skalierung. Zuletzt wird die 5 V VCC-Spannung betrachtet. Hier wird ein P10 Wert von 4980 V bestimmt. In P1 entspricht dies 4.86 V. Somit liegt dieser Wert geringfügig unter den erwarteten 5 V.
Mithilfe von diesem Test kann festgestellt werden, dass keiner der Spannungswerte genau dem erwarteten Wert entspricht. Dies ist unter anderem mit starken Störungen auf der ADU-Leitung zu begründen. Diese Störungen sind unter anderem auf die Signalführung über lange Wege auf der Platine und auf schlecht gesetzte oder fehlende Kondensatoren zurückzuführen. Durch das nachträgliche Einlöten von Kondensatoren an verschiedenen Punkten kann eine Verbesserung der Werte festgestellt werden. Diese Verbesserungen sind auf dem zweiten Prototypen eingearbeitet worden.

6.2.2 Erstellung des UART-Frames

Innerhalb von diesem Kapitel wird die Zusammensetzung des UART-Frames und die Funktion des CRC-Checks überprüft. Hierfür ist in der Abbildung 6.5 der Debug-Auszug eines zu sendenden UART-Frames zu erkennen.

Expression	Type	Value
dataBufferUARTTx1	uint8_t [11]	[11]
dataBufferUARTTx1[0]	uint8_t	1 '\001'
dataBufferUARTTx1[1]	uint8_t	255 'ÿ'
dataBufferUARTTx1[2]	uint8_t	193 'Á'
dataBufferUARTTx1[3]	uint8_t	30 '\036'
dataBufferUARTTx1[4]	uint8_t	107 'k'
dataBufferUARTTx1[5]	uint8_t	0 '\000'
dataBufferUARTTx1[6]	uint8_t	1 '\001'
dataBufferUARTTx1[7]	uint8_t	0 '\000'
dataBufferUARTTx1[8]	uint8_t	44 ','
dataBufferUARTTx1[9]	uint8_t	20 '\024'
dataBufferUARTTx1[10]	uint8_t	176 '°'

Abbildung 6.5: Debugauszug zur Erstellung des UART-Frames

An der Position 0 wird der Header des zu sendenden UART-Datenpakets mit einer dezimalen 1 dargestellt. Dies entspricht in hexadezimal die 0x01. Alle Datenübertragungen der Messdaten weisen diesen Header auf.

Anschließend werden in den Positionen 1 bis 4 die Magnetspannungen gesendet. Dabei wird das MSB der Spannung zuerst übertragen. Die Magnetspannung M1 weist somit die dezimalen Zahlen 255 und 193 auf. Dies entspricht in hexadezimal der Zahl 0xFFC1 und in dezimal der Zahl 65473. Binär weist diese Zahl die folgende Abfolge auf: 1111111111000001. Da das MSB dieser Zahl eine 1 aufweist und die Magnetspannung negative Werte annehmen kann, muss die Zahl $2^{16} = 65536$ von der 65473 subtrahiert werden. Dies ergibt einen dezimalen Wert von -63. Umgerechnet in die P1-Skalierung ergibt sich dementsprechend eine Spannung von $-0.12\,\mathrm{V}$. Der gleiche Vorgang wird für die Magnetspannung M2 durchgeführt. Hierbei sind die dezimalen Zahlen 30 und 107 zu übertragen. Dies entspricht der hexadezimalen Zahl 0x1E6B, was in dezimal der Zahl 7787 entspricht. Binär ergibt sich folgende Zahl: 0001111001101011. Das MSB (Vorzeichenbit) weist eine 0 auf, weshalb keine Zahl abgezogen werden muss. Es ergibt sich somit ein Wert in der P1-Skalierung von $15.21\,\mathrm{V}$. An der Magnetspannung M1 liegen zum Zeitpunkt des Tests $0\,\mathrm{V}$ und an der Magnetspannung M2 $15\,\mathrm{V}$ an. Somit sind die Magnetspannungen korrekt in das UART-Frame eingefügt worden.
An der Position 5 erfolgt die Übertragung einer Null. Hier können in Zukunft Fehlermeldungen übertragen werden, welche bisher nicht genauer definiert wurden.
Es folgt die Übermittlung des Identifiers an Position 6. Hier wird der Identifier 1 übermittelt. Dies bedeutet, dass die nachfolgenden zwei Positionen Temperaturdaten enthalten. Es werden die dezimalen 0 und 44 als Temperaturdaten übertragen. Die Temperaturdaten weisen bei der Umrechnung neun signifikante Bit auf. Bei der Übertragung durch 16-Bit sind somit die obersten sieben Bit nicht relevant und können bei der Berechnung des Temperaturwertes weggelassen werden. An Bit neun ist das MSB des Temperaturwerts vorhanden. Es sind somit die dezimalen Zahlen 0 und 44 zu übertragen. Dies ergibt den dezimalen Gesamtwert von 44. Um den genauen Temperaturwert zu erhalten, muss diese Zahl nach dem Datenblatt noch durch 0.5 geteilt werden. Es ergibt sich somit eine Temperatur von 22 Grad.
Die letzten zwei Positionen des UART-Frames erhalten die CRC-Checksumme. Um den korrekten Wert zu verifizieren, wird die Berechnung mithilfe eines Online-Tools vorgenommen [2]. Hier ist das Ergebnis des CRC-16/IBM-3740 relevant, da alle Parameter zu dem implementierten CRC-Check dieses Programms übereinstimmen. Das Ergebnis des Tools lautet in hexadezimal 0x14B0. In dezimal umgerechnet ergeben sich die zwei dezimalen Werte 20 und 176. Die Werte stimmen somit mit den angegebenen Werten in dem Datenbuffer überein.
Zusammenfassend erfolgt die korrekte Zusammenstellung der Daten innerhalb des Datenbuffers, welcher über UART zu versenden ist.

7 Fazit

In dieser Arbeit wurden die ersten Prototypen zur Überwachung der Hot-Swap-Matrix und der Magnetspannungen entwickelt. Diese Hot-Swap-Matrix dient zum verzugsfreien Umschalten einer Spannungsversorgung der Magnete auf eine Reserve-Spannungsversorgung. Hiermit soll der Ausfall eines Magneten und damit einhergehend der Ausfall des gesamten Beschleunigerbetriebs der zukünftigen Petra IV Anlage verhindert werden. Die Hot-Swap-Matrix ist auf der Hot-Swap-Platine implementiert. Das Ziel der entwickelten Prototypen liegt darin, kritische Signale, welche zu einem Ausfall des Beschleunigerbetriebs führen können, zu überwachen und die gesammelten Daten an ein übergeordnetes System weiterzuleiten. Anhand der übermittelten Daten kann das übergeordnete System Maßnahmen zur Sicherung des Betriebs einleiten.
Um die Überwachung gewährleisten zu können, wurde in dieser Arbeit eine Mikrokontrollerplatine mit dem STM32 Nucleo-64-Board NUCLEO-F072RB und die zugehörige Software entwickelt. Hierbei werden verschiedene Signale durch den Mikrokontroller überwacht und deren Daten mithilfe einer UART-Verbindung über die Hot-Swap-Platine an das übergeordnete System weitergeleitet. Zu den überwachten Signalen zählen unter anderem die Überwachung der Versorgungsspannung der Magneten, die Beobachtung der Dump-Schaltung, die Überprüfung der Schalterstellung in der Hot-Swap-Matrix und die Überwachung der Umgebungstemperatur. Hierbei wurde bei der Entwicklung der Schaltungen auf der Mikrokontrollerplatine besonders auf die galvanische Trennung zwischen den Potentialen der zwei durch die Hot-Swap-Matrix überwachbaren Magnetspannungkreise und dem Potential der Mikrokontrollerplatine geachtet. Ebenso mussten Schaltungen zur Spannungsreduktion von digitalen und analogen Signalen integriert werden. Innerhalb der Softwareentwicklung ist auf eine regelmäßige und konstante Datenabfrage und Datenübermittlung geachtet worden. Dabei wurden die besonders kritischen Signale identifiziert, die bei der Datenübermittlung eine deutlich höher priorisierte Behandlung erfordern. Um eine sichere und zuverlässige Übertragung über UART zu gewährleisten, wurde ein CRC-Check integriert.
In einigen abschließenden Tests konnten grundlegende Funktionen der Prototypen verifi-

ziert werden. Hierbei wurde vor allem die korrekte Funktion der implementierten Hardwareschaltungen nachgewiesen. Im Bereich der Software konnten grundlegende Funktionen verifiziert werden. Jedoch fehlte ein geeigneter Testaufbau, mit dessen Hilfe das gesamte Timing des Codes mit verschiedenen externen Signalen getestet werden kann. Diese Tests sollten in Zukunft unbedingt durchgeführt werden, um einen reibungslosen Gesamtbetrieb verifizieren zu können. Hierbei sind die verschiedenen implementierten Überwachungsmechanismen hinsichtlich ihrer Sinnhaftigkeit in dem realen Beschleunigerbetrieb zu beurteilen und zu bewerten. Darüber hinaus besteht insbesondere bei der autarken Systemfunktion weiterer Entwicklungsbedarf.
Mit dieser Arbeit wurde ein grundlegender Prototyp zur Überwachung der Spannungsversorgung der Magnete und der Hot-Swap-Platine entwickelt, der nun weiter optimiert werden kann, um in der zukünftigen Petra IV-Anlage effektiv eingesetzt zu werden.

Literaturverzeichnis

[1] : *CRC 16 CCITT in C#*. – URL http://sanity-free.org/133/crc_16_ccitt_in_csharp.html. – Zugriffsdatum: 31-01-2025

[2] : *CRC-Rechner*. – URL https://crccalc.com/?crc=0x01%200x00%200x00%200x00%200x00%200x00%200x05%200x00%200x00&method=CRC-16%20CCITT&datatype=hex&outtype=hex. – Zugriffsdatum: 14-01-2025

[3] : *ESERA 1-Wire Bussystem - Grundlagen, Tipps und Hintergrundinformationen*. – URL https://esera.de/Service-Support/1-Wire-Grundlagen/1-Wire-Grundlagen/. – Zugriffsdatum: 13-12-2024

[4] : *Isolationsverstärker*. – URL https://www.electricity-magnetism.org/de/isolationsverstaerker/. – Zugriffsdatum: 29-12-2024

[5] : *Leuchtdioden (LED) - Einführung*. – URL https://www.leifiphysik.de/elektronik/halbleiterdiode/grundwissen/leuchtdioden-led-einfuehrung. – Zugriffsdatum: 19-12-2024

[6] : *Optokoppler / Opto-Koppler*. – URL https://www.elektronik-kompendium.de/sites/bau/0411091.htm. – Zugriffsdatum: 19-12-2024

[7] : *SPI vs. I2C Kommunikationsprotokolle: Die wichtigsten Unterschiede*. – URL https://www.fs-pcba.com/de/spi-vs-i2c/. – Zugriffsdatum: 10-12-2024

[8] Tiefpass mit Tücken. In: *30 elektronik industrie 10 - 2006 electronica-VORBERICHTE* (2006). – URL https://www.all-electronics.de/wp-content/uploads/migrated/article-pdf/75751/ei06-10-030.pdf. – Zugriffsdatum: 11-01-2025

[9] PETRA IV - 3D-Röntgenmikroskop der Superlative. (2019), Dezember. – URL https://www.sciencecity.hamburg/wp-content/uploads/PETRA-IV-Broschuere_web.pdf. – Zugriffsdatum: 09-01-2025

[10] SPI vs. I2C: So wählen Sie das beste Protokoll für Ihre Speicherchips. In: *Altium* (2022), November. – URL https://resources.altium.com/de/p/spi-versus-i2c-how-choose-best-protocol-your-memory-chips. – Zugriffsdatum: 13-12-2024

[11] Analog Devices: *LTC3115-1 40V, 2A Synchronous Buck-Boost DC/DC Converter*. – URL https://www.mouser.de/datasheet/2/609/LTC3115_1-2956069.pdf

[12] Baker, Bonnie C.: *Wirkungsgrad-Überlegungen*, July 2002. – URL https://www.all-electronics.de/wp-content/uploads/migrated/article-pdf/66321/e4cb8433e80.pdf. – Zugriffsdatum: 27-12-2024

[13] Beningo, Jacob: *Einsatz von digitalen Isolatoren zum Schutz von Testgeräten*. Juli 2021. – URL https://www.digikey.de/de/blog/protect-your-test-equipment-using-digital-isolators?srsltid=AfmBOoqc11Ajag_OB7LH3Bq4ORRO_jr_9iMLOJPDqjGqP6emrmrJAvIe. – Zugriffsdatum: 21-12-2024

[14] Bernd Pompe: *Elektronik*. – URL https://physik.uni-greifswald.de/storages/uni-greifswald/fakultaet/mnf/physik/Studium/Elektronik/elektronik-vorlesung.pdf

[15] Carle, Georg ; Schmitt, Corinna ; Klein, Alexander ; Baumgarten, Uwe ; Söllner, Christoph: *Proceeding zum Seminar Sensorknoten*. 2010. – URL https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2010-09-1.pdf. – Zugriffsdatum: 10-12-2024

[16] DigiKey: *How To Calculate and Use RC Time Constants*. – URL https://www.digikey.de/de/maker/tutorials/2024/how-to-calculate-and-use-rc-time-constants. – Zugriffsdatum: 05-12-2024

[17] DigiKey: *PWM INPUT in STM32*. – URL https://controllerstech.com/pwm-input-in-stm32/. – Zugriffsdatum: 11-01-2025

[18] Erickson, Robert W.: DC-DC Power Converters. In: *Wiley Encyclopedia of Electrical and Electronics Engineering*. – URL https://web.archive.org/web/20170809015854/http://ecee.colorado.edu/~ecen4517//materials/Encyc.pdf. – Zugriffsdatum: 30-12-2024

[19] Falco, Eleazar: ANO007 | Grundlagen zu Phototransistor-Optokopplern. In: *Würth Elektronik* (2023), August. – URL https://www.we-online.com/components/media/o760905v410%20ANO007a_DE.pdf. – Zugriffsdatum: 19-12-2024

[20] Göbel, Holger: *Einführung in die Halbleiter- Schaltungstechnik*. Springer-Verlag GmbH Deutschland, 2019. – ISBN 978-3-662-56562-9

[21] Grotelüschen, Frank: DESY. (2012), August. – URL https://pr.desy.de/sites/sites_desygroups/sites_extern/site_pr/content/e104098/e104099/DESY_Broschuere_web_ger.pdf. – Zugriffsdatum: 09-01-2025

[22] Hillen, Heidrun: *Das Herz von PETRA IV*. – URL https://petra4.desy.de/petra_iv/beschleuniger/das_herz_von_petra_iv/index_ger.html. – Zugriffsdatum: 03-01-2025

[23] Horowitz, Paul ; Hill, Winfeld: *The Art of Electronics*. Cambridge University Press, 2015. – ISBN 978-0-521-80926-9

[24] Instruments, Texas: *TLV773 300mA, Small-Size, High-PSRR, Low-Dropout Regulator*. – URL https://www.ti.com/lit/ds/symlink/tlv773.pdf?ts=1733236688805&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTLV773. – Zugriffsdatum: 26-12-2024

[25] Johanneck, Don: Grundlagen zur Logikpegelwandlung. In: *DigiKey* (2021), August. – URL https://www.digikey.de/de/blog/logic-level-shifting-basics. – Zugriffsdatum: 05-12-2024

[26] K.H.Gonschorek ; H.Singer: *elektromagnetisce Verträglichkeit*. B.G. Teubner Stuttgart, 1992. – ISBN 978-3-332-82992-4

[27] Kompendium, Elektronik: *Spannungsstabilisierung mit Z-Diode*. – URL https://www.elektronik-kompendium.de/sites/slt/1012151.htm. – Zugriffsdatum: 24-12-2024

[28] Lerch, Reinhard: *Elektrische Messtechnik - Analoge, digitale und computergestützte Verfahren*. Springer-Verlag Berlin Heidelberg 2016, 2016. – ISBN 978-3-662-46940-8

[29] LMU München: *Erklärung Hamming Codes*. – URL https://www.mobile.ifi.lmu.de/wp-content/uploads/lehrveranstaltungen/rechnerarchitektur-sose17/Erkl%C3%A4rungHammingCodes.pdf. – Zugriffsdatum: 08-12-2024

[30] Maxino, Theresa C. ; Koopman, Philip J.: The Effectiveness of Checksums for Embedded Control Networks. In: *IEEE* (2009). – URL https://ieeexplore.ieee.org/document/4358707. – Zugriffsdatum: 07-12-2024

[31] Mouser Electronics: *Level Shifter Übersetzungsspannungs-Pegel.* – URL https://www.mouser.de/c/semiconductors/logic-ics/translation-voltage-levels/?type=Level%20Shifter&sort=pricing%7C1. – Zugriffsdatum: 05-12-2024

[32] Patel, Atul: *Integrated vs. Discrete Open Drain Level Translation.* Texas Instruments (Veranst.), Januar 2024. – URL https://www.ti.com/lit/ab/sdla007/sdla007.pdf. – Zugriffsdatum: 06-12-2024

[33] Paul, Steffen ; Paul, Reinhold: *Grundlagen der Elektrotechnik und Elektronik 1 - Gleichstromnetzwerke und ihre Anwendungen.* Springer-Verlag GmbH Deutschland, 2022. – ISBN 978-3-662-66187-1

[34] Peterson, Zachariah: *Abwärtsregler vs. Spannungsregler für DC, AC und HF: Welcher ist der Beste.* – URL https://resources.altium.com/de/p/buck-converter-regulator-vs-ldo-dc-ac-and-rf-which-best. – Zugriffsdatum: 26-12-2024

[35] Peterson, Zachariah: *LDO-Wirkungsgrad: Testen Sie die Grenzen Ihres Spannungsreglers.* – URL https://resources.altium.com/de/p/testing-limits-your-ldos-efficiency. – Zugriffsdatum: 27-12-2024

[36] Physik, Leifi: *Elektromagnetische Induktion - Selbstinduktion und Induktivität.* – URL https://www.leifiphysik.de/elektrizitaetslehre/elektromagnetische-induktion/grundwissen/selbstinduktion-und-induktivitaet. – Zugriffsdatum: 24-12-2024

[37] Pini, Art: Messung kleiner Signale auf hohen Spannungen und Vermeidung von Masseschleifen in Sensoren. In: *DigiKey* (2018), August. – URL https://www.digikey.de/de/articles/measure-small-signals-riding-on-high-voltages-avoid-sensor-ground-loops?srsltid=AfmBOoqj22trbqVhC8qvYWWWLGMFKBA4DyKM2zEhiN0RK3CeHLUpWFH7. – Zugriffsdatum: 18-12-2024

[38] Pini, Art: Die Grundlagen von LDOs und ihre Anwendung zur Verlängerung der Batterielebensdauer in tragbaren und Wearable-Geräten. (2022), November. – URL

https://www.digikey.de/de/articles/the-basics-of-ldos-and-how-to-apply-them-to-extend-battery-life-in-portables-and-wearables?srsltid=AfmBOorMGRwrpLgF__rQAc3n0F4aBVG0Q2Sij2mlRq4_yGrXD2qIK4uq. – Zugriffsdatum: 26-12-2024

[39] Renesas: *ISL85403 2.5A Regulator with Integrated High-side MOSFET for Synchronous Buck or Boost Buck Converter*. 2015. – URL https://www.mouser.de/pdfdocs/isl85403.pdf. – Zugriffsdatum: 28-12-2024

[40] Sack, Harald ; Meinel, Christoph: *Digitale Kommunikation - Vernetzen, Multimedia, Sicherheit*. Springer-Verlag Berlin Heidelberg, 2009. – ISBN 978-3-540-92922-2

[41] Scherz, Paul ; Monk, Simon: *Practical Electronics for Inventors*. McGraw-Hill Educatio, 2016. – ISBN 78-1-25-958754-2

[42] Schweber, Bill: *Understanding the Advantages and Disadvantages of Linear Regulators*. – URL https://www.digikey.de/en/articles/understanding-the-advantages-and-disadvantages-of-linear-regulators. – Zugriffsdatum: 27-12-2024

[43] Siegl, Johann: *Schaltungstechnik - analog und gemischt analog/digital*. Springer-Verlag Berlin Heidelberg, 2010. – ISBN 78-3-642-13303-9

[44] Texas Instruments: *TPS629210 3-V to 17-V, 1-A Low IQ Buck Converter in SOT-583 Package*. 2021. – URL https://www.ti.com/lit/ds/symlink/tps629210.pdf?ts=1735904987225&ref_url=https%253A%252F%252Fwww.bing.com%252F. – Zugriffsdatum: 28-12-2024

[45] TOSHIBA: *4-1. Efficiency of LDO regulators*. – URL https://toshiba.semicon-storage.com/us/semiconductor/knowledge/e-learning/basics-of-low-dropout-ldo-regulators/chap4/chap4-1.html. – Zugriffsdatum: 27-12-2024

[46] Trowbridge, Luke: *Isolation: Optokoppler versus Opto-Emulatoren*. Februar 2024. – URL https://www.elektronikpraxis.de/isolation-optokoppler-versus-opto-emulatoren-a-56c57f7c4c575513242bf33468c5254b/. – Zugriffsdatum: 21-12-2024

[47] Viehmann, Matthias: *Operationsverstärker - Grundlagen und Schaltungen und Anwendungen*. Carl Hanser Verlag München, 2020. – ISBN 978-3-446-45951-9

[48] Zach, Frank: *Leistungselektronik*. Springer Fachmedien Wiesbaden, 2010. – ISBN 978-3-658-04898-3

A Anhang

Der vollständige Anhang dieser Arbeit befindet sich auf CD und kann bei dem Erstgutachter eingesehen werden.

A.1 Verwendete Hilfsmittel

In der Tabelle A.1 sind die im Rahmen der Bearbeitung des Themas der Masterarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tabelle A.1: Verwendete Hilfsmittel und Werkzeuge

Tool	Verwendung
LaTeX	Textsatz- und Layout-Werkzeug verwendet zur Erstellung dieses Dokuments
Eagle	Erstellung der Schaltpläne und des Platinenlayouts
STMCubeIDE	Programmierung des Mikrokontrollers
Tablet Oszilloskop TO2004 von Micsig	Erstellung von Oszilloskopbildern

A.2 Anschlussplan an den Mikrokontroller

Tabelle A.2: Anschlussplan des CN7 Steckers

Pin	Pin-name	Signal/ Funktion	Pin	Pin-name	Signal/ Funktion
1	PC10		2	PC11	
3	PC12		4	PD2	
5	VDD		6	E5V	
7	BOOT0		8	GND	
9	NC		10	NC	
11	NC		12	IOREF	
13	PA13		14	Reset	
15	PA14		16	3V3	
17	PA15		18	5V	
19	GND		20	GND	
21	PB7	I2C_SDA I2C Daten	22	GND	
23	PC13	Button	24	VIN	
25	PC14	RCC	26	NC	
27	PC15	RCC	28	PA0	ADC_VCC5V 5V VCC-Spannung
29	PF0	RCC	30	PA1	ADC_12VPS 12V Power-Supplie-Spannung
31	PF1	RCC	32	PA4	ADC_12V 12V-Versorgungsspannung
33	VBAT		34	PB0	ADC_M1 Magnetspannung M1
35	PC2		36	PC1	ADC_M2 Magnetspannung M2
37	PC3		38	PC0	ADC_PWM PWM-Signal

Tabelle A.3: Anschlussplan des CN10 Steckers

Pin	Pin-name	Signal/ Funktion	Pin	Pin-name	Signal/ Funktion
1	PC9		2	PC8	
3	PB8	Detection_2_Low	4	PC6	Detection_2_High
5	PB9	Detection_1_Low	6	PC5	Detection_1_High
7	AVDD		8	U5V	
9	GND		10	NC	
11	PA5	LED	12	PA12	Dump_1
13	PA6	Matrix_2_2	14	PA11	Dump_2
15	PA7	Matrix_2_1	16	PB12	Matrix_1_2
17	PB6	I2C_SCL Taktleitung I2C	18	PB11	Matrix_1_1
19	PC7	ADJ_Detection_Bus PWM-Signal	20	GND	
21	PA9	UART 1_TX	22	PB2	Interrupt des Temperatursensors
23	PA8	TXS_Enable Freigabe der Pegel-wandler	24	PB1	ADDR1 Address-Signal
25	PB10		26	PB15	ADDR2 Address-Signal
27	PB4		28	PB14	ADDR3 Address-Signal
29	PB5		30	PB13	ADDR4 Address-Signal
31	PB3		32	AGND	
33	PA10	UART 1_RX	34	PC4	
35	PA2		36	NC	
37	PA3		38	NC	

A.3 Schaltungsauslegung zur galvanischen Trennung der 12 V Spannung

Der passende Schaltplanauszug ist in der Abbildung A.1 zu erkennen.

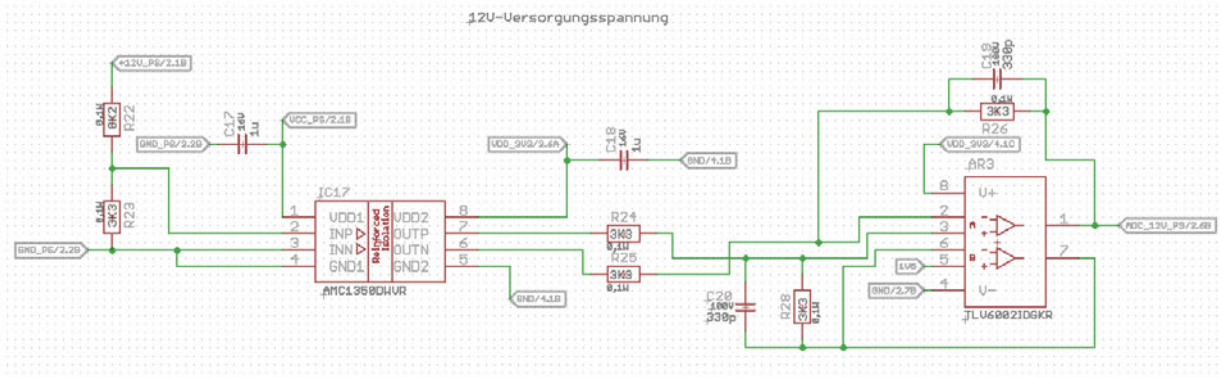


Abbildung A.1: Auszug aus dem Schaltplan: Galvanische Trennung 12 V-Versorgungsspannung

Die Auslegung erfolgt für die 12 V-Eingangsspannung. Anders als bei der Magnetspannung, weist dieses Eingangssignal keine negativen Signale auf. Um einen geringfügigen Puffer einzubauen, damit auch Spannungen über den vorgegebenen 12 V bestimmt werden können, wird die Schaltung so ausgelegt, dass bei einer Eingangsspannung von 12 V 3 V am Mikrokontroller anliegen. Dies bedeutet, dass das differentielle Signal am Trennverstärkerausgang ± 1.5 V aufweisen sollte. Um die Anzahl an verbauten Komponenten zu minimieren, wird der gleiche LDO für die Erzeugung der Referenzsspannung, wie bei den Magnetspannungen verwendet. Dieser LDO erzeugt eine Spannung von 1.5 V.
Anschließend wird nun der Spannungsteiler am Trennverstärkereingang ausgelegt. Es erfolgt zunächst die Bestimmung der Eingangsspannung am AMC. Der AMC weist auch hier einen Verstärkungsfaktor von 0.4 $\frac{\mathrm{V}}{\mathrm{V}}$ auf.

$$\begin{aligned} \mathrm{U_{IN}} &= \frac{\mathrm{U_{OUT}}}{\mathrm{Gain}} \\ &= \frac{1.5\,\mathrm{V}}{0.4\,\frac{\mathrm{V}}{\mathrm{V}}} \\ &= 3.75\,\mathrm{V} \end{aligned} \tag{A.1}$$

Es erfolgt nun die Ermittlung des Gesamtwiderstandes des Spannungsteilers. Es wird ein maximaler Stromfluss von 1 mA angenommen.

$$\begin{aligned} R_{Ges} &= \frac{U}{I} \\ &= \frac{12\,V}{1\,mA} \\ &= 12\,k\Omega \end{aligned} \tag{A.2}$$

Es folgt die Bestimmung von R22, oberer Widerstand des Spannungsteilers, und von R23, unterer Widerstand des Spannungsteilers. Dabei gilt, dass $U_{IN} = U_{R23}$.

$$\begin{aligned} R23 &= \frac{U_{R23}}{U_{Ges}} \cdot R_{Ges} \\ &= \frac{3.5\,V}{12\,V} \cdot 12\,k\Omega \\ &= 3.75\,k\Omega \end{aligned} \tag{A.3}$$

$$\begin{aligned} R22 &= R_{Ges} - R23 \\ &= 12\,k\Omega - 3.5\,k\Omega \\ &= 8.25\,k\Omega \end{aligned} \tag{A.4}$$

A.4 Schaltpläne

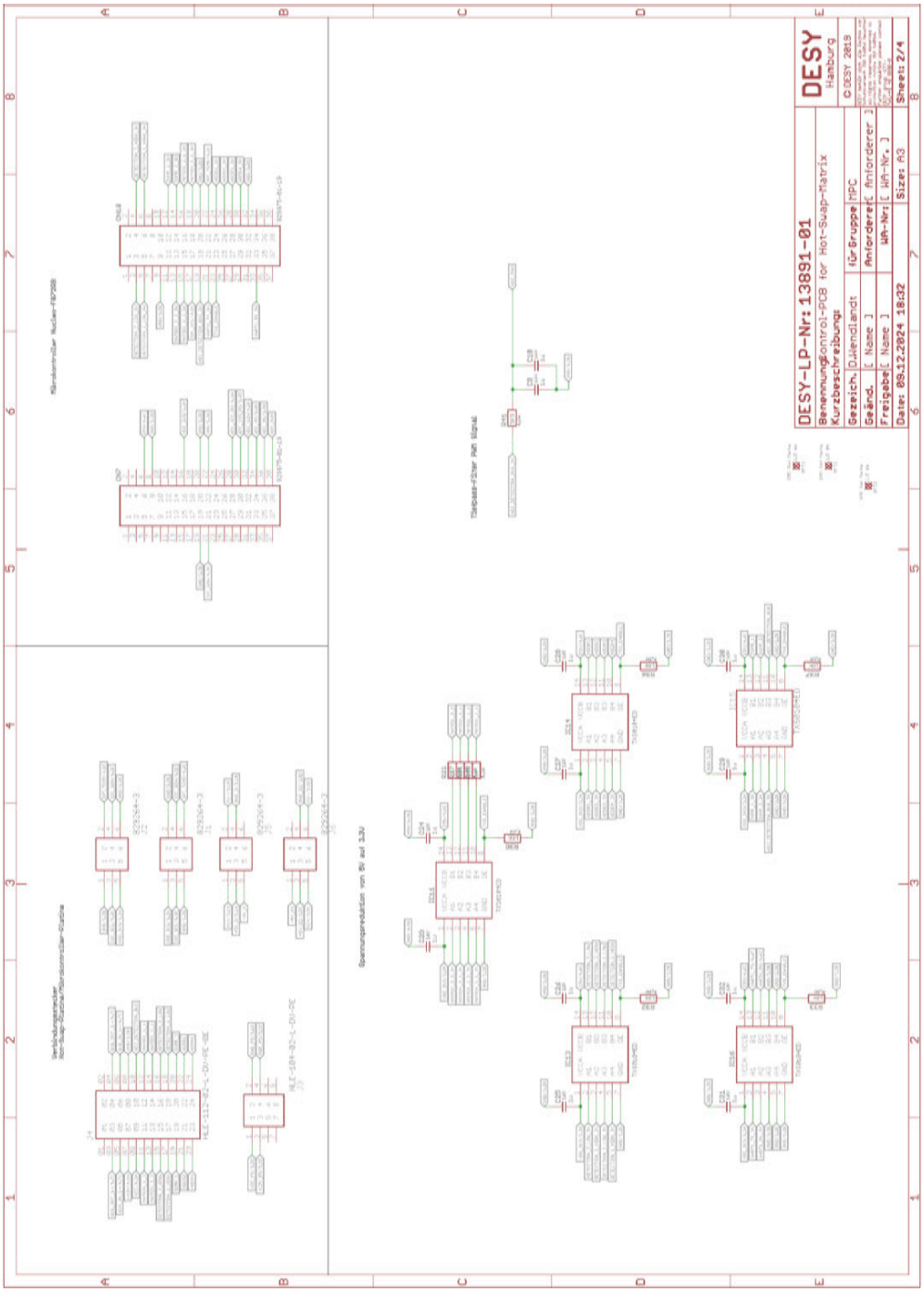


Abbildung A.2: Schaltplan Seite 1

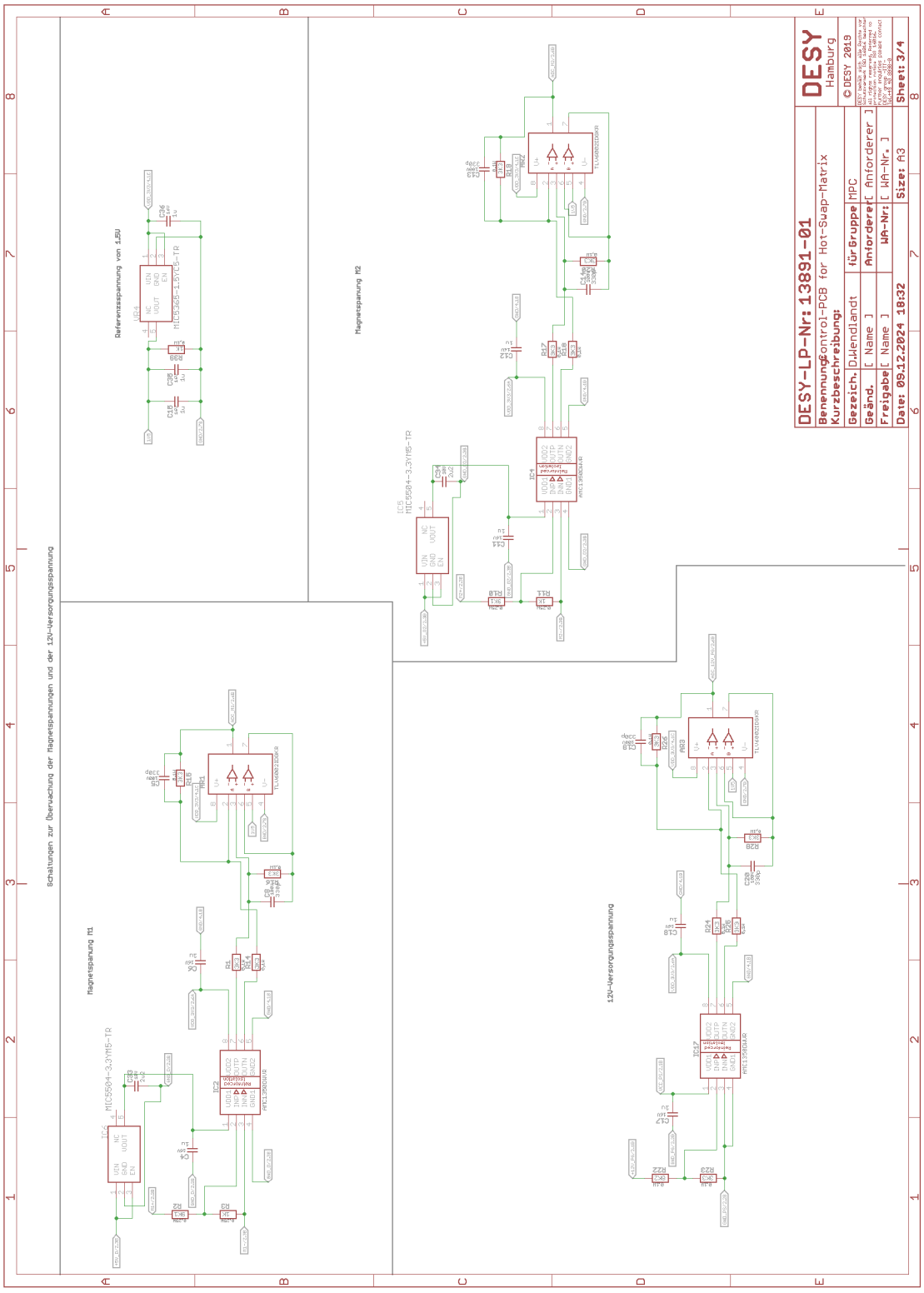


Abbildung A.3: Schaltplan Seite 2

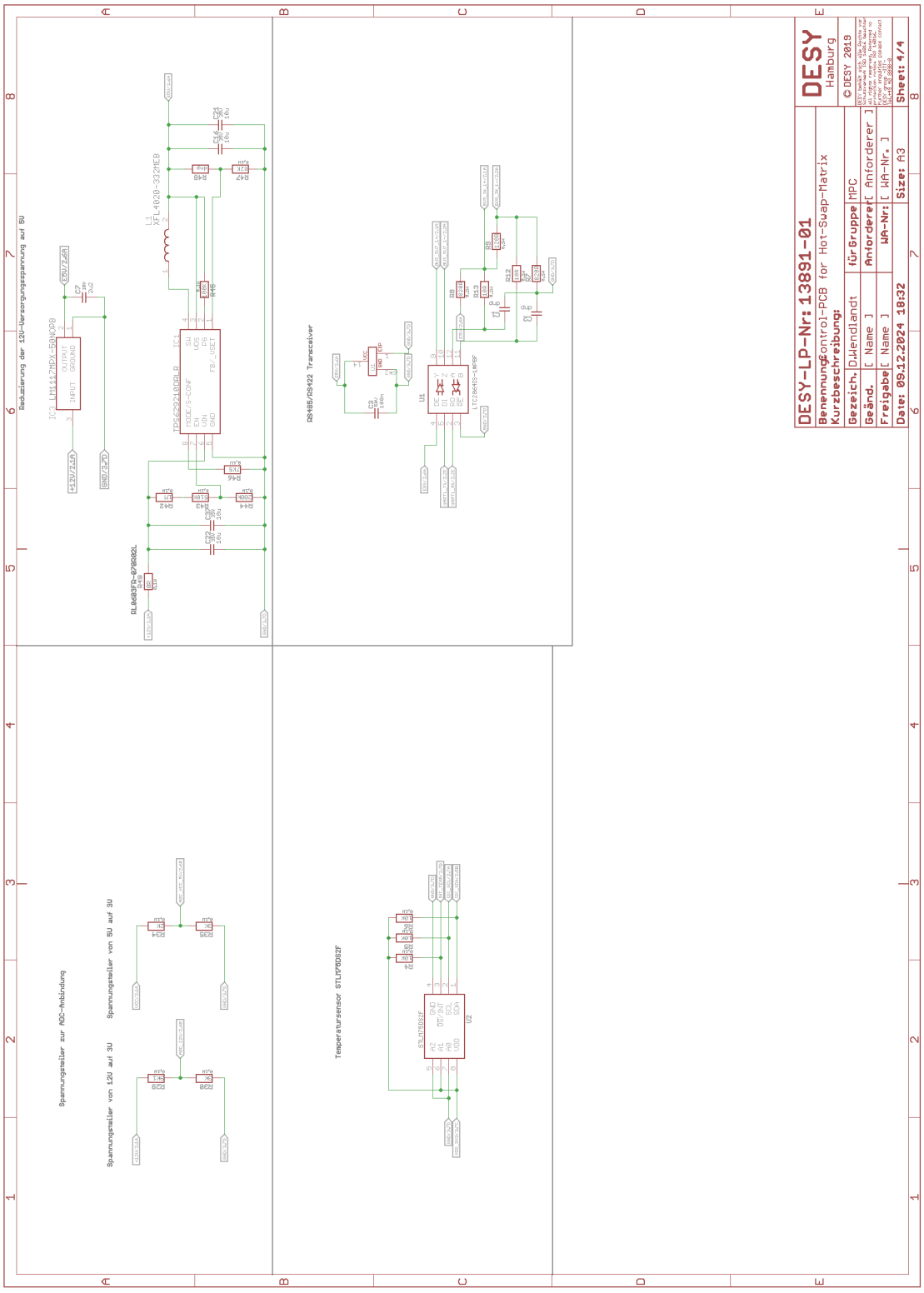


Abbildung A.4: Schaltplan Seite 3

A.5 Softwarecode

```
/*
 * Defines.h
 *
 *  Created on: Sep 4, 2024
 *      Author: Denise
 */

// Einstellungen UART
#define BAUDRATE_UART 230400
#define WORDLENGTH_UART UART_WORDLENGTH_9B //Datenbits + Stopbits
#define STOPBIT_UART UART_STOPBITS_1
#define PARITY_UART UART_PARITY_EVEN

// Einstellungen ADC
#define SAMPLE_TIME_ADC ADC_SAMPLETIME_41CYCLES_5//ADC_SAMPLETIME_7CYCLES_5

// Einstellungen Timer3  UART
#define PRESCALER_TIM3 48 - 1
#define PERIOD_TIM3 1000 - 1

// Einstellungen Timer 6 I2C
#define PRESCALER_TIM6 48000-1
#define PERIOD_TIM6 500-1

// Einstellungen Timer 7 ADC
#define PRESCALER_TIM7 48-1
#define PERIOD_TIM7 210-1 //70-1

// Einstellungen Timer 14 Ueberwachungstimer
// 10 UART Uebertragungen werden nicht durchgefuehrt, bevor das Programm
// neu gestartet wird
#define PRESCALER_TIM14 48 - 1;
#define PERIOD_TIM14  10000 - 1

// Einstellungen Timer 16 GPIO
#define PRESCALER_TIM16 48000 - 1
#define PERIOD_TIM16  200 - 1

//Sensoradressen
# define SENSOR1 72
# define SENSOR2 73
# define SENSOR3 74
```

```
43
44 // Einstellungen CRC Check
45 #define CRC_POLYNOM 0x1021
```

Quellcode A.1: Defines

```
1  /**********************************************************************
2   * @file           : main.c
3   * @brief          : Main program body
4   **********************************************************************
5   * @attention
6   * Copyright (c) 2024 STMicroelectronics.
7   * All rights reserved.
8   *
9   * This software is licensed under terms that can be found in the LICENSE
       file
10  * in the root directory of this software component.
11  * If no LICENSE file comes with this software, it is provided AS-IS.
12  **********************************************************************/
13
14  // Includes
15 #include "Header_Init.h"
16 #include "main.h"
17 #include "Defines.h"
18
19 // Variabeln
20 ADC_HandleTypeDef hadc;
21 DMA_HandleTypeDef hdma_adc;
22 I2C_HandleTypeDef hi2c1;
23 TIM_HandleTypeDef htim3;
24 TIM_HandleTypeDef htim6;
25 TIM_HandleTypeDef htim7;
26 TIM_HandleTypeDef htim14;
27 TIM_HandleTypeDef htim16;
28 UART_HandleTypeDef huart1;
29 DMA_HandleTypeDef hdma_usart1_rx;
30 DMA_HandleTypeDef hdma_usart1_tx;
31
32 // Prototypen Funktionen
33 void SystemClock_Config(void);
34 static void MX_GPIO_Init(void);
35 static void MX_DMA_Init(void);
36 static void MX_ADC_Init(void);
37 static void MX_I2C1_Init(void);
38 static void MX_USART1_UART_Init(void);
```

```
39 static void MX_TIM3_Init(void);
40 static void MX_TIM14_Init(void);
41 static void MX_TIM7_Init(void);
42 static void MX_TIM6_Init(void);
43 static void MX_TIM16_Init(void);
44
45 //Variablen I2C
46 HAL_StatusTypeDef statI2C;
47 int16_t temp_sensor1 = 0, temp_sensor2 = 0, temp_sensor3 = 0;
48 uint8_t count = 1;
49 uint8_t temperature_data[3][2];
50
51 //Variablen UART senden
52 uint8_t dataBufferUARTTx1[11] = { 0 };
53 uint8_t dataBufferUARTTx2[11] = { 0 };
54 uint8_t UARTTxBufferSelection = 0x01;
55 uint8_t uarttransmiterror = 0;
56 int i = 0;
57 uint8_t sendSerialnumber = 0;
58 uint8_t MagnetUART[5] = { 0 };
59 uint8_t newUARTTxData = 0;
60
61 //Variablen UART empfangen
62 uint8_t dataBufferUARTRx[11] = { 0 };
63 uint8_t dataBufferUARTnewData[11] = { 0 };
64 uint8_t newUARTRxData = 0;
65 uint8_t CountUARTRxData = 0;
66
67 //Variablen ADUs
68 uint16_t ADC_Values[6] = { 0 };
69 uint16_t ADC_12V = 0, ADC_12VPS = 0, ADC_VCC5V = 0;
70 int16_t ADC_M1 = 0, ADC_M2 = 0, ADC_PWM = 0, ADC_PWM_P10 = 0;
71 int ADC_M1_MW_P10 = 0, ADC_M2_MW_P10 = 0;
72 int ADC_M1_P12 = 0, ADC_M1_P10 = 0, ADC_M2_P12 = 0, ADC_M2_P10 = 0;
73 int16_t ADC_M1_P9 = 0, ADC_M2_P9 = 0;
74 int16_t ADC_12V_P10 = 0, ADC_12VPS_P12 = 0, ADC_12VPS_P10 = 0,
75     ADC_VCC5V_P10 = 0;
76 int16_t SchwellwertM1_max_P10 = 0, SchwellwertM1_min_P10 = 0,
77     SchwellwertM2_max_P10 = 0, SchwellwertM2_min_P10 = 0;
78 int newADCData = 0;
79 uint8_t MagnetProblem = 0;
80
81 //Variablen GPIO Auslesen
82 uint8_t Detect_2_L = 0, Detect_2_H = 0, Detect_1_L = 0, Detect_1_H = 0;
```

```
83 uint8_t Dump1 = 0, Dump2 = 0;
84 uint8_t Matrix_22 = 0, Matrix_21 = 0, Matrix_12 = 0, Matrix_11 = 0;
85 uint8_t INT_Temp = 0;
86 uint8_t ADDR1 = 0, ADDR2 = 0, ADDR3 = 0, ADDR4 = 0;
87
88 // Variablen zum Speichern der Seriennummmer
89 uint32_t serialNumber[3] = { 0 };
90 uint8_t serialNumberUART[13] = { 0 };
91
92 //Variablen zur CRC Ermittlung
93 uint16_t CRC16_Table[256];
94 uint8_t dataCRC[11];
95 uint8_t identifier[8] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
96 uint16_t crc_t;
97 uint16_t crc = 0;
98
99 int main(void) {
100
101   HAL_Init();
102
103   SystemClock_Config();
104
105   //Initialisierungen
106   MX_GPIO_Init();
107   MX_DMA_Init();
108   MX_ADC_Init();
109   MX_I2C1_Init();
110   MX_USART1_UART_Init();
111   MX_TIM14_Init();
112   MX_TIM7_Init();
113   MX_TIM14_Init();
114   MX_TIM6_Init();
115   MX_TIM3_Init();
116
117   // Enable-Signal fuer die Pegelkonverter
118   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
119
120   //Starten der Timer
121   HAL_TIM_Base_Start_IT(&htim14);
122   HAL_TIM_Base_Start_IT(&htim3); //UART
123   HAL_TIM_Base_Start_IT(&htim6); //I2C
124   HAL_TIM_Base_Start_IT(&htim7); //ADC
125   HAL_TIM_Base_Start_IT(&htim16); //GPIO
126
```

```
127   // Generierung der CRC-Tabelle
128   generateCRC16Table();
129   //Auslesen der Seriennummer
130   readSerialNumber();
131
132   //Starten des Datenempfangs ueber UART
133   if (HAL_UARTEx_ReceiveToIdle_DMA(&huart1, (uint8_t*) dataBufferUARTRx,
134       sizeof(dataBufferUARTRx)) != HAL_OK) {
135     //Fehler
136   }
137
138   while (1) {
139     // Abfrage, ob neue ADU Daten vorliegen
140     if (newADCData != 0) {
141       //Umwandlung der Daten
142       calcMagnetvoltage();
143       newADCData = 0;
144     }
145     // Abfrage, ob neuer UART Frame gefuellt werden soll
146     if (newUARTTxData != 0) {
147       newUARTTxData = 0;
148       CreateUartFrame();
149     }
150     // Verarbeitung der ueber UART empfangenen Daten
151     if (newUARTRxData != 0) {
152       newUARTRxData = 0;
153
154       // Durchfuehrung des CRC-Checks
155       uint16_t crc = calculateCRC16((uint8_t*) dataBufferUARTnewData,
156           CountUARTRxData);
157
158       CountUARTRxData = 0;
159
160       // Abfrage, ob CRC Check erfolgreich war
161       if (crc != 0) {
162         uarttransmiterror = 1;
163       }
164       else {
165         uarttransmiterror = 0;
166         // Senden der Seriennummer
167         if (dataBufferUARTnewData[0] == 0x02) {
168           sendSerialnumber = 1;
169         }
170       }
```

```
171     }
172     // Abfrage, ob enable-Pin aktiv ist
173      if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_8)!= GPIO_PIN_SET)
174      {
175      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
176      }
177   }
178
179 }
180
181 /************************************************************
182  * Funktionen zum Konvertieren der Daten
183  ************************************************************/
184 void CreateUartFrame(void) {
185   //Umrechung der Magnetspannungen in die P10-Skalierung
186   ADC_M1_P9 = ADC_M1_P10 >> 1;
187   ADC_M2_P9 = ADC_M2_P10 >> 1;
188   //Fuellen des Buffers 1
189   if (UARTTxBufferSelection == 0x01) {
190     UARTTxBufferSelection = 0x02;
191     // Header fuer normale UART Datenuebertragung
192     dataBufferUARTTx1[0] = 0x01;
193     // Einfuegen der Magnetspannungen
194     // MSB zuerst
195     dataBufferUARTTx1[1] = (uint8_t) ((ADC_M1_P9 >> 8) & 0xFF);
196     dataBufferUARTTx1[2] = (uint8_t) (ADC_M1_P9 & 0xFF);
197     dataBufferUARTTx1[3] = (uint8_t) ((ADC_M2_P9 >> 8) & 0xFF);
198     dataBufferUARTTx1[4] = (uint8_t) (ADC_M2_P9 & 0xFF);
199     // Fehlermeldungen
200     dataBufferUARTTx1[5] = 0x00;
201     //Identifier und Daten
202     switch (i) {
203     // Temperatur Sensor 1
204     case 0:
205       dataBufferUARTTx1[6] = identifier[i];
206       dataBufferUARTTx1[7] = (uint8_t) ((temp_sensor1 >> 8) & 0xFF);
207       dataBufferUARTTx1[8] = (uint8_t) (temp_sensor1 & 0xFF);
208       i++;
209       break;
210     // Temperatur Sensor 2
211     case 1:
212       dataBufferUARTTx1[6] = identifier[i];
213       dataBufferUARTTx1[7] = (uint8_t) ((temp_sensor2 >> 8) & 0xFF);
214       dataBufferUARTTx1[8] = (uint8_t) (temp_sensor2 & 0xFF);
```

```
215       i++;
216       break;
217     //Temperatur Sensor 3
218     case 2:
219       dataBufferUARTTx1[6] = identifier[i];
220       dataBufferUARTTx1[7] = (uint8_t) ((temp_sensor3 >> 8) & 0xFF);
221       dataBufferUARTTx1[8] = (uint8_t) (temp_sensor3 & 0xFF);
222       i++;
223       break;
224     // ADC 12V PS
225     case 3:
226
227       dataBufferUARTTx1[6] = identifier[i];
228       dataBufferUARTTx1[7] = (uint8_t) ((ADC_12VPS_P10 >> 8) & 0xFF);
229       dataBufferUARTTx1[8] = (uint8_t) (ADC_12VPS_P10 & 0xFF);
230       i++;
231       break;
232     // ADC 12V
233     case 4:
234
235       dataBufferUARTTx1[6] = identifier[i];
236       dataBufferUARTTx1[7] = (uint8_t) ((ADC_12V_P10 >> 8) & 0xFF);
237       dataBufferUARTTx1[8] = (uint8_t) (ADC_12V_P10 & 0xFF);
238       i++;
239       break;
240     // ADC 5V VCC
241     case 5:
242
243       dataBufferUARTTx1[6] = identifier[i];
244       dataBufferUARTTx1[7] = (uint8_t) ((ADC_12V_P10 >> 8) & 0xFF);
245       dataBufferUARTTx1[8] = (uint8_t) (ADC_12V_P10 & 0xFF);
246       i++;
247       break;
248     // Detection + Matrix + Adresse + Dump
249     case 6:
250       dataBufferUARTTx1[6] = identifier[i];
251       dataBufferUARTTx1[7] |= (Detect_1_H << 7) | (Detect_1_L << 6)
252           | (Detect_2_H << 5) | (Detect_2_L << 4) | (Matrix_11 << 3)
253           | (Matrix_12 << 2) | (Matrix_21 << 1) | Matrix_22;
254       dataBufferUARTTx1[8] |= (ADDR1 << 5) | (ADDR2 << 4) | (ADDR3 << 3)
255           | (ADDR4 << 2) | (Dump1 << 1) | Dump2;
256       i = 0;
257       break;
258     }
```

```
259     //Gernierung der CRC-Checksumme
260     crc = calculateCRC16(dataBufferUARTTx1, 9);
261     //Einfuegen der CRC-Checksumme
262     dataBufferUARTTx1[9] = (uint8_t) ((crc >> 8) & 0xFF);
263     dataBufferUARTTx1[10] = (uint8_t) (crc & 0xFF);
264
265   }
266   // Datenbuffer 2 fuellen
267   else if (UARTTxBufferSelection == 0x02) {
268     UARTTxBufferSelection = 0x01;
269     // Header fuer normale UART Datenuebertragung
270     dataBufferUARTTx2[0] = (uint8_t) 0x01;
271     // Magnetspannungen MSB zuerst
272     dataBufferUARTTx2[1] = (uint8_t) ((ADC_M1_P9 >> 8) & 0xFF);
273     dataBufferUARTTx2[2] = (uint8_t) (ADC_M1_P9 & 0xFF);
274     dataBufferUARTTx2[3] = (uint8_t) ((ADC_M2_P9 >> 8) & 0xFF);
275     dataBufferUARTTx2[4] = (uint8_t) (ADC_M2_P9 & 0xFF);
276     // Fehlermeldungen
277     dataBufferUARTTx2[5] = (uint8_t) 0x00;
278     //Identifier und Daten
279     switch (i) {
280     // Temperatursensor 1
281     case 0:
282       i++;
283       dataBufferUARTTx2[6] = identifier[i];
284       dataBufferUARTTx2[7] = (uint8_t) ((temp_sensor1 >> 8) & 0xFF);
285       dataBufferUARTTx2[8] = (uint8_t) (temp_sensor1 & 0xFF);
286       break;
287     // Temperatursensor 2
288     case 1:
289       i++;
290       dataBufferUARTTx2[6] = identifier[i];
291       dataBufferUARTTx2[7] = (int8_t) ((temp_sensor2 >> 8) & 0xFF);
292       dataBufferUARTTx2[8] = (int8_t) (temp_sensor2 & 0xFF);
293       break;
294     //Temperatursensor 3
295     case 2:
296       i++;
297       dataBufferUARTTx2[6] = identifier[i];
298       dataBufferUARTTx2[7] = (uint8_t) ((temp_sensor3 >> 8) & 0xFF);
299       dataBufferUARTTx2[8] = (uint8_t) (temp_sensor3 & 0xFF);
300       break;
301     // ADC 12V PS
302     case 3:
```

```
303       i++;
304       dataBufferUARTTx2[6] = identifier[i];
305       dataBufferUARTTx2[7] = (uint8_t) ((ADC_12VPS_P10 >> 8) & 0xFF);
306       dataBufferUARTTx2[8] = (uint8_t) (ADC_12VPS_P10 & 0xFF);
307       break;
308     // ADC 12V
309     case 4:
310       i++;
311       dataBufferUARTTx2[6] = identifier[i];
312       dataBufferUARTTx2[7] = (uint8_t) ((ADC_12V_P10 >> 8) & 0xFF);
313       dataBufferUARTTx2[8] = (uint8_t) (ADC_12V_P10 & 0xFF);
314       break;
315     // ADC 5V VCC
316     case 5:
317       i++;
318       dataBufferUARTTx2[6] = identifier[i];
319       dataBufferUARTTx2[7] = (uint8_t) ((ADC_12V_P10 >> 8) & 0xFF);
320       dataBufferUARTTx2[8] = (uint8_t) (ADC_12V_P10 & 0xFF);
321       break;
322     // Detection + Matrix + Adresse + Dump
323     case 6:
324       i = 0;
325       dataBufferUARTTx2[6] = identifier[i];
326       dataBufferUARTTx2[7] |= (Detect_1_H << 7) | (Detect_1_L << 6)
327           | (Detect_2_H << 5) | (Detect_2_L << 4) | (Matrix_11 << 3)
328           | (Matrix_12 << 2) | (Matrix_21 << 1) | Matrix_22;
329       dataBufferUARTTx2[8] |= (ADDR1 << 5) | (ADDR2 << 4) | (ADDR3 << 3)
330           | (ADDR4 << 2) | (Dump1 << 1) | Dump2;
331
332       break;
333     }
334
335     //CRC Check
336     uint16_t crc = calculateCRC16(dataBufferUARTTx2, 9);
337
338     dataBufferUARTTx2[9] = (uint8_t) ((crc >> 8) & 0xFF);
339     dataBufferUARTTx2[10] = (uint8_t) (crc & 0xFF);
340   }
341 }
342
343 // Funktion zum CRC-Check (CRC-16-CCITT-Standard)
344 uint16_t calculateCRC16(uint8_t *data, uint16_t length) {
345
346   uint16_t crc = 0xFFFF;  // Initialwert
```

```
347
348   for (uint16_t i = 0; i < length; i++) {
349     uint8_t byte = data[i];
350     crc = (crc << 8) ^ CRC16_Table[(crc >> 8) ^ byte];
351   }
352
353   return crc;
354 }
355
356 //Funktion zum Gernerieren der CRC-Tabelle
357 void generateCRC16Table(void) {
358   for (int i = 0; i < 256; i++) {
359     uint16_t crc = i << 8;
360     for (int j = 0; j < 8; j++) {
361       if (crc & 0x8000) {
362         crc = (crc << 1) ^ CRC_POLYNOM;
363       } else {
364         crc <<= 1;
365       }
366     }
367     CRC16_Table[i] = crc;
368   }
369 }
370
371 /************************************************************
372  * Funktion zum Auslesen der individuellen Seriennummer
373  ************************************************************/
374 void readSerialNumber(void) {
375   //Auslesen der Seriennummer
376   serialNumber[0] = *(uint32_t*) 0x1FFFF7AC;  // Low 32 bits
377   serialNumber[1] = *(uint32_t*) 0x1FFFF7B0;  // Mid 32 bits
378   serialNumber[2] = *(uint32_t*) 0x1FFFF7B4;  // High 32 bits
379
380   //Erstellung der UART-Frames
381   serialNumberUART[0] = (uint8_t) 0x02;
382   serialNumberUART[1] = (uint8_t) (serialNumber[0] & 0xFF);
383   serialNumberUART[2] = (uint8_t) ((serialNumber[0] >> 8) & 0xFF);
384   serialNumberUART[3] = (uint8_t) ((serialNumber[0] >> 16) & 0xFF);
385   serialNumberUART[4] = (uint8_t) ((serialNumber[0] >> 24) & 0xFF);
386
387   serialNumberUART[5] = (uint8_t) (serialNumber[1] & 0xFF);
388   serialNumberUART[6] = (uint8_t) ((serialNumber[1] >> 8) & 0xFF);
389   serialNumberUART[7] = (uint8_t) ((serialNumber[1] >> 16) & 0xFF);
390   serialNumberUART[8] = (uint8_t) ((serialNumber[1] >> 24) & 0xFF);
```

```
391
392   serialNumberUART[9] = (uint8_t) (serialNumber[2] & 0xFF);
393   serialNumberUART[10] = (uint8_t) ((serialNumber[2] >> 8) & 0xFF);
394   serialNumberUART[11] = (uint8_t) ((serialNumber[2] >> 16) & 0xFF);
395   serialNumberUART[12] = (uint8_t) ((serialNumber[2] >> 24) & 0xFF);
396 }
397
398 /*************************************************************
399  * Funktionen zum Auslesen der GPIO-Interrupts
400  *************************************************************/
401 void Read_GPIO(void) {
402
403   //Detection-Signale
404   Detect_2_L = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_9);
405   Detect_2_H = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_8);
406   Detect_1_L = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_8);
407   Detect_1_H = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_6);
408
409   //Dump-Signale
410   Dump1 = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_9);
411   Dump2 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_5);
412
413   //Matrix-Signale
414   Matrix_22 = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_12);
415   Matrix_21 = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_11);
416   Matrix_12 = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_12);
417   Matrix_11 = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_11);
418
419   //Interrupt-Signal Temperatursensor
420   INT_Temp = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_7);
421
422   //Address-Signale
423   ADDR1 = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_1);
424   ADDR2 = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_15);
425   ADDR3 = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_14);
426   ADDR4 = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_13);
427
428 }
429
430 // GPIO-Callback
431 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
432   if (GPIO_Pin == GPIO_PIN_8) // Interrupt fuer Pin 8
433   {
434
```

```
435   }
436
437   if (GPIO_Pin == GPIO_PIN_9) // Interrupt fuer Pin 9
438   {
439
440   }
441
442 }
443
444 /*************************************************************
445  * Funktionen zum Auslesen der ADCs
446  *************************************************************/
447
448 void calcMagnetvoltage() {
449   int Faktor34_P12 = 13926; //3.4 * 2^12
450   int RefSpannung15_P12 = 6144; //1.5 * 2^12
451   int SkalierungMagnet_P10 = 25600; // (10/0.4)*2^10
452   int SkalierungPWM_P12 = 1987; // (5/3.3) * 1/10 *3.4 * 2^12
453   int SkalierungVCC_P12 = 22282;  //16/10 * 3.4 * 2^12
454   int Skalierung12V_P12 = 55705; // 4* 3.4 * 2^12
455   int Skalierung12VPS_P10 = 8960; //(3.5/0.4)*2^10
456
457   ADC_M1_P12 = ((ADC_M1 * Faktor34_P12) >> 12) - RefSpannung15_P12;
458   ADC_M1_P10 = ((ADC_M1_P12 >> 2) * SkalierungMagnet_P10) >> 10;
459   ADC_M2_P12 = ((ADC_M2 * Faktor34_P12) >> 12) - RefSpannung15_P12;
460   ADC_M2_P10 = ((ADC_M2_P12 >> 2) * SkalierungMagnet_P10) >> 10;
461
462   ADC_PWM_P10 = (ADC_PWM * SkalierungPWM_P12) >> 14;
463
464   // Bestimmung Grenzen des Schwellwertes
465   SchwellwertM1_max_P10 = ADC_M1_MW_P10 + ADC_PWM_P10;
466   SchwellwertM1_min_P10 = ADC_M1_MW_P10 - ADC_PWM_P10;
467   SchwellwertM2_max_P10 = ADC_M2_MW_P10 + ADC_PWM_P10;
468   SchwellwertM2_min_P10 = ADC_M2_MW_P10 - ADC_PWM_P10;
469
470   //Vergleich der Schwellwerte mit aktuellem Messwert
471   if (ADC_M1_P10 < SchwellwertM1_min_P10
472       || ADC_M1_P10 > SchwellwertM1_max_P10) {
473     MagnetProblem = 1;
474     //Erstellung des UART-Frames
475     MagnetUART[0] = (uint8_t) 0x03;
476     MagnetUART[1] = (uint8_t) ((ADC_M1_P9 >> 8) & 0xFF);
477     MagnetUART[2] = (uint8_t) (ADC_M1_P9 & 0xFF);
478     MagnetUART[3] = (uint8_t) ((ADC_M2_P9 >> 8) & 0xFF);
```

```
479     MagnetUART[4] = (uint8_t) (ADC_M2_P9 & 0xFF);
480   }
481
482   else if (ADC_M2_P10 < SchwellwertM2_min_P10
483       || ADC_M2_P10 > SchwellwertM2_max_P10) {
484     MagnetProblem = 1;
485     //Erstellung des UART-Frames
486     MagnetUART[0] = (uint8_t) 0x03;
487     MagnetUART[1] = (uint8_t) ((ADC_M1_P9 >> 8) & 0xFF);
488     MagnetUART[2] = (uint8_t) (ADC_M1_P9 & 0xFF);
489     MagnetUART[3] = (uint8_t) ((ADC_M2_P9 >> 8) & 0xFF);
490     MagnetUART[4] = (uint8_t) (ADC_M2_P9 & 0xFF);
491
492   } else {
493     MagnetProblem = 0;
494   }
495
496   //Gleitenden MW bestimmen (Alpha 0.5)
497   ADC_M1_MW_P10 = (ADC_M1_P10 >> 1) + (ADC_M1_MW_P10 >> 1);
498   ADC_M2_MW_P10 = (ADC_M2_P10 >> 1) + (ADC_M2_MW_P10 >> 1);
499
500   // Skalierung der Versorgungsspannungen
501   ADC_VCC5V_P10 = (ADC_VCC5V * SkalierungVCC_P12) >> 14;
502   ADC_12VPS_P12 = ((ADC_12VPS * Faktor34_P12) >> 12) - RefSpannung15_P12;
503   ADC_12VPS_P10 = ((ADC_12VPS_P12 >> 2) * Skalierung12VPS_P10) >> 10;
504   ADC_12V_P10 = (ADC_12V * Skalierung12V_P12) >> 14;
505 }
506
507 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {
508
509   HAL_ADC_Stop_DMA(hadc);
510
511   if (hadc->Instance == ADC1) {
512     // Umspeichern der Werte
513     ADC_12V = ADC_Values[0];
514     ADC_12VPS = ADC_Values[1];
515     ADC_VCC5V = ADC_Values[5];
516     ADC_M1 = ADC_Values[2];
517     ADC_M2 = ADC_Values[3];
518     ADC_PWM = ADC_Values[4];
519
520     newADCData = 1;
521   }
522 }
```

```
523
524 /*************************************************************
525  * Funktionen zum Auswerten von I2C
526  * Uebermittlung der Temperaturdaten
527  *************************************************************/
528
529 void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c1) {
530
531   if (statI2C == HAL_OK) {
532     // Auswertung der Temperaturdaten
533     switch (count) {
534     case 1:
535       temp_sensor1 = (temperature_data[0][0] << 8 | temperature_data[0][1])
      >> 7;
536       count++;
537       break;
538
539     case 2:
540       temp_sensor2 =
541           (temperature_data[1][0] << 8 | temperature_data[1][1]) >> 7;
542       count++;
543       break;
544
545     case 3:
546       temp_sensor3 =
547           (temperature_data[2][0] << 8 | temperature_data[2][1]) >> 7;
548       count = 1;
549       break;
550     }
551   }
552   else {
553     if (HAL_I2C_GetError(hi2c1) != HAL_I2C_ERROR_AF) {
554       MX_I2C1_Init();
555     }
556   }
557 }
558
559 /*************************************************************
560  * Funktionen zur Datenuebertragung mit UART
561  *************************************************************/
562
563 void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size) {
564   if (huart->Instance == USART1) {
565
```

```
566     newUARTRxData = 1;
567
568     // Daten umspeichern, solange Daten vorhanden sind
569     for (int i = 0; i <= (sizeof(dataBufferUARTRx)); i++) {
570       if (dataBufferUARTRx[i] != 0) {
571         dataBufferUARTnewData[i] = dataBufferUARTRx[i];
572         CountUARTRxData = CountUARTRxData + 1;
573       }
574     }
575
576     // UART Empfangsarray leeren
577     for (int i = 0; i <= (sizeof(dataBufferUARTRx)); i++) {
578       dataBufferUARTRx[i] = 0;
579     }
580
581     // Datenempfang starten
582     if (HAL_UARTEx_ReceiveToIdle_DMA(&huart1, (uint8_t*) dataBufferUARTRx,
583         sizeof(dataBufferUARTRx)) != HAL_OK) {
584       //Fehler
585     }
586   }
587 }
588
589 void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart1) {
590   if (huart1->Instance == USART1) {
591     newUARTTxData = 1;
592   }
593 }
594
595 void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart) {
596
597   if (huart->ErrorCode & HAL_UART_ERROR_PE) {
598   } else if (huart->ErrorCode & HAL_UART_ERROR_NE) {
599     // Not-Ausfallfehler
600   } else if (huart->ErrorCode & HAL_UART_ERROR_FE) {
601     // Uebertragungsfehler
602   } else if (huart->ErrorCode & HAL_UART_ERROR_ORE) {
603     // Ueberlauf-Fehler
604   }
605 }
606
607 /*************************************************************
608  * Funktionen zur Timersteuerung
609  *************************************************************/
```

```
610 // Timer Interrupt Callback
611 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
612
613   // Timer 3 -> Intervall UART -> 1000 Mikrosekunden
614   if (htim->Instance == TIM3) {
615     if (UARTTxBufferSelection == 0x02 && sendSerialnumber != 1
616         && MagnetProblem != 1) {
617       HAL_UART_Transmit_DMA(&huart1, (uint8_t*) dataBufferUARTTx1, 11);
618     }
619     else if (UARTTxBufferSelection == 0x01 && sendSerialnumber != 1
620         && MagnetProblem != 1) {
621       HAL_UART_Transmit_DMA(&huart1, (uint8_t*) dataBufferUARTTx2, 11);
622     }
623     else if (sendSerialnumber == 1 && MagnetProblem != 1) {
624       HAL_UART_Transmit_DMA(&huart1, (uint8_t*) serialNumberUART, 13);
625     }
626     else if (MagnetProblem == 1) {
627       HAL_UART_Transmit_DMA(&huart1, (uint8_t*) MagnetUART, 5);
628     }
629   }
630
631   // Timer 6 Intervall I2C Messung (1s)
632   if (htim->Instance == TIM6) {
633     switch (count) {
634     case 1:
635       statI2C = HAL_I2C_Master_Receive_IT(&hi2c1, (SENSOR1 << 1),
636           temperature_data[0], 2);
637       break;
638
639     case 2:
640       statI2C = HAL_I2C_Master_Receive_IT(&hi2c1, (SENSOR2 << 1),
641           temperature_data[1], 2);
642       break;
643
644     case 3:
645       statI2C = HAL_I2C_Master_Receive_IT(&hi2c1, (SENSOR3 << 1),
646           temperature_data[2], 2);
647       break;
648     }
649
650   }
651
652   // Timer 7 Intervall fuer ADC Messung 55 Mikrosekunden
653   if (htim->Instance == TIM7) {
```

```
654     HAL_ADC_Start_DMA(&hadc, (uint32_t*) ADC_Values, 6);
655   }
656
657   // TIM14 Ueberwachungstimer
658   if (htim->Instance == TIM14) {
659
660   }
661
662   // Timer 16 Interval GPIO Pins abfragen
663   if (htim->Instance == TIM16) {
664     Read_GPIO();
665   }
666
667 }
668
669 /**
670  * @brief System Clock Configuration
671  * @retval None
672  */
673 void SystemClock_Config(void) {
674   RCC_OscInitTypeDef RCC_OscInitStruct = { 0 };
675   RCC_ClkInitTypeDef RCC_ClkInitStruct = { 0 };
676   RCC_PeriphCLKInitTypeDef PeriphClkInit = { 0 };
677
678   /** Initializes the RCC Oscillators according to the specified parameters
679    * in the RCC_OscInitTypeDef structure.
680    */
681   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI
682       | RCC_OSCILLATORTYPE_HSI14 | RCC_OSCILLATORTYPE_HSI48;
683   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
684   RCC_OscInitStruct.HSI48State = RCC_HSI48_ON;
685   RCC_OscInitStruct.HSI14State = RCC_HSI14_ON;
686   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
687   RCC_OscInitStruct.HSI14CalibrationValue = 16;
688   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
689   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
690     Error_Handler();
691   }
692
693   /** Initializes the CPU, AHB and APB buses clocks
694    */
695   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK
696       | RCC_CLOCKTYPE_PCLK1;
697   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI48;
```

```
698   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
699   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
700
701   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK) {
702     Error_Handler();
703   }
704   PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART1
705       | RCC_PERIPHCLK_I2C1;
706   PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK1;
707   PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_HSI;
708   if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK) {
709     Error_Handler();
710   }
711 }
712
713 /**
714  * @brief ADC Initialization Function
715  * @param None
716  * @retval None
717  */
718 static void MX_ADC_Init(void) {
719
720   ADC_ChannelConfTypeDef sConfig = { 0 };
721
722   /** Configure the global features of the ADC (Clock, Resolution, Data
      Alignment and number of conversion)
723    */
724   hadc.Instance = ADC1;
725   hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
726   hadc.Init.Resolution = ADC_RESOLUTION_12B;
727   hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
728   hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
729   hadc.Init.EOCSelection = ADC_EOC_SEQ_CONV;
730   hadc.Init.LowPowerAutoWait = DISABLE;
731   hadc.Init.LowPowerAutoPowerOff = DISABLE;
732   hadc.Init.ContinuousConvMode = DISABLE;
733   hadc.Init.DiscontinuousConvMode = DISABLE;
734   hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
735   hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
736   hadc.Init.DMAContinuousRequests = DISABLE;
737   hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
738   if (HAL_ADC_Init(&hadc) != HAL_OK) {
739     Error_Handler();
740   }
```

```
741
742   /** Configure for the selected ADC regular channel to be converted.
743    */
744   sConfig.Channel = ADC_CHANNEL_0;
745   sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
746   sConfig.SamplingTime = SAMPLE_TIME_ADC;
747   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK) {
748     Error_Handler();
749   }
750
751   /** Configure for the selected ADC regular channel to be converted.
752    */
753   sConfig.Channel = ADC_CHANNEL_1;
754   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK) {
755     Error_Handler();
756   }
757
758   /** Configure for the selected ADC regular channel to be converted*/
759   sConfig.Channel = ADC_CHANNEL_4;
760   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK) {
761     Error_Handler();
762   }
763
764   /** Configure for the selected ADC regular channel to be converted.*/
765   sConfig.Channel = ADC_CHANNEL_8;
766   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK) {
767     Error_Handler();
768   }
769
770   /** Configure for the selected ADC regular channel to be converted.
771    */
772   sConfig.Channel = ADC_CHANNEL_10;
773   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK) {
774     Error_Handler();
775   }
776
777   /** Configure for the selected ADC regular channel to be converted.
778    */
779   sConfig.Channel = ADC_CHANNEL_11;
780   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK) {
781     Error_Handler();
782   }
783 }
784
```

```
785 /**
786  * @brief I2C1 Initialization Function
787  * @param None
788  * @retval None
789  */
790 static void MX_I2C1_Init(void) {
791
792   hi2c1.Instance = I2C1;
793   hi2c1.Init.Timing = 0x2000090E;
794   hi2c1.Init.OwnAddress1 = 0;
795   hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
796   hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
797   hi2c1.Init.OwnAddress2 = 0;
798   hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
799   hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
800   hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
801   if (HAL_I2C_Init(&hi2c1) != HAL_OK) {
802     Error_Handler();
803   }
804
805   /** Configure Analogue filter
806    */
807   if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE)
808       != HAL_OK) {
809     Error_Handler();
810   }
811
812   /** Configure Digital filter
813    */
814   if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK) {
815     Error_Handler();
816   }
817 }
818
819 /**
820  * @brief TIM3 Initialization Function
821  * @param None
822  * @retval None
823  */
824 static void MX_TIM3_Init(void) {
825
826   TIM_ClockConfigTypeDef sClockSourceConfig = { 0 };
827   TIM_MasterConfigTypeDef sMasterConfig = { 0 };
828
```

```
829   htim3.Instance = TIM3;
830   htim3.Init.Prescaler = PRESCALER_TIM3;
831   htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
832   htim3.Init.Period = PERIOD_TIM3;
833   htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
834   htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
835   if (HAL_TIM_Base_Init(&htim3) != HAL_OK) {
836     Error_Handler();
837   }
838   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
839   if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK) {
840     Error_Handler();
841   }
842   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
843   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
844   if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig)
845       != HAL_OK) {
846     Error_Handler();
847   }
848 }
849
850 /**
851  * @brief TIM6 Initialization Function
852  * @param None
853  * @retval None
854  */
855 static void MX_TIM6_Init(void) {
856
857   TIM_MasterConfigTypeDef sMasterConfig = { 0 };
858
859   htim6.Instance = TIM6;
860   htim6.Init.Prescaler = PRESCALER_TIM6;
861   htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
862   htim6.Init.Period = PERIOD_TIM6;
863   htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
864   if (HAL_TIM_Base_Init(&htim6) != HAL_OK) {
865     Error_Handler();
866   }
867   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
868   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
869   if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig)
870       != HAL_OK) {
871     Error_Handler();
872   }
```

```
873 }
874
875 /**
876  * @brief TIM7 Initialization Function
877  * @param None
878  * @retval None
879  */
880 static void MX_TIM7_Init(void) {
881
882   TIM_MasterConfigTypeDef sMasterConfig = { 0 };
883
884   htim7.Instance = TIM7;
885   htim7.Init.Prescaler = PRESCALER_TIM7;
886   htim7.Init.CounterMode = TIM_COUNTERMODE_UP;
887   htim7.Init.Period = PERIOD_TIM7;
888   htim7.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
889   if (HAL_TIM_Base_Init(&htim7) != HAL_OK) {
890     Error_Handler();
891   }
892   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
893   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
894   if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig)
895       != HAL_OK) {
896     Error_Handler();
897   }
898 }
899
900 /**
901  * @brief TIM14 Initialization Function
902  * @param None
903  * @retval None
904  */
905 static void MX_TIM14_Init(void) {
906
907   htim14.Instance = TIM14;
908   htim14.Init.Prescaler = PRESCALER_TIM14;
909   htim14.Init.CounterMode = TIM_COUNTERMODE_UP;
910   htim14.Init.Period = PERIOD_TIM14;
911   htim14.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
912   htim14.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
913   if (HAL_TIM_Base_Init(&htim14) != HAL_OK) {
914     Error_Handler();
915   }
916 }
```

```
917
918 /**
919  * @brief TIM16 Initialization Function
920  * @param None
921  * @retval None
922  */
923 static void MX_TIM16_Init(void) {
924
925   htim16.Instance = TIM16;
926   htim16.Init.Prescaler = PRESCALER_TIM16;
927   htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
928   htim16.Init.Period = PERIOD_TIM16;
929   htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
930   htim16.Init.RepetitionCounter = 0;
931   htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
932   if (HAL_TIM_Base_Init(&htim16) != HAL_OK) {
933     Error_Handler();
934   }
935 }
936
937 /**
938  * @brief USART1 Initialization Function
939  * @param None
940  * @retval None
941  */
942 static void MX_USART1_UART_Init(void) {
943
944   huart1.Instance = USART1;
945   huart1.Init.BaudRate = BAUDRATE_UART;
946   huart1.Init.WordLength = WORDLENGTH_UART;
947   huart1.Init.StopBits = STOPBIT_UART;
948   huart1.Init.Parity = PARITY_UART;
949   huart1.Init.Mode = UART_MODE_TX_RX;
950   huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
951   huart1.Init.OverSampling = UART_OVERSAMPLING_16;
952   huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
953   huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
954   if (HAL_UART_Init(&huart1) != HAL_OK) {
955     Error_Handler();
956   }
957 }
958
959 /**
960  * Enable DMA controller clock
```

```
961  */
962 static void MX_DMA_Init(void) {
963
964   /* DMA controller clock enable */
965   __HAL_RCC_DMA1_CLK_ENABLE();
966
967   /* DMA interrupt init */
968   /* DMA1_Channel1_IRQn interrupt configuration */
969   HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 1, 0);
970   HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
971   /* DMA1_Channel2_3_IRQn interrupt configuration */
972   HAL_NVIC_SetPriority(DMA1_Channel2_3_IRQn, 0, 0);
973   HAL_NVIC_EnableIRQ(DMA1_Channel2_3_IRQn);
974
975 }
976
977 /**
978  * @brief GPIO Initialization Function
979  * @param None
980  * @retval None
981  */
982 static void MX_GPIO_Init(void) {
983   GPIO_InitTypeDef GPIO_InitStruct = { 0 };
984
985   /* GPIO Ports Clock Enable */
986   __HAL_RCC_GPIOC_CLK_ENABLE();
987   __HAL_RCC_GPIOF_CLK_ENABLE();
988   __HAL_RCC_GPIOA_CLK_ENABLE();
989   __HAL_RCC_GPIOB_CLK_ENABLE();
990
991   /*Configure GPIO pin Output Level */
992   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5 | GPIO_PIN_8, GPIO_PIN_SET);
993
994   /*Configure GPIO pin : PC13 */
995   GPIO_InitStruct.Pin = GPIO_PIN_13;
996   GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
997   GPIO_InitStruct.Pull = GPIO_NOPULL;
998   HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
999
1000  /*Configure GPIO pins : PA5 PA8 */
1001  GPIO_InitStruct.Pin = GPIO_PIN_5 | GPIO_PIN_8;
1002  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1003  GPIO_InitStruct.Pull = GPIO_NOPULL;
1004  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
1005   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
1006
1007   /*Configure GPIO pins : PA7 PA11 PA12 */
1008   GPIO_InitStruct.Pin = GPIO_PIN_7 | GPIO_PIN_11 | GPIO_PIN_12;
1009   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1010   GPIO_InitStruct.Pull = GPIO_NOPULL;
1011   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
1012
1013   /*Configure GPIO pins : PC5 PC6 */
1014   GPIO_InitStruct.Pin = GPIO_PIN_5 | GPIO_PIN_6;
1015   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1016   GPIO_InitStruct.Pull = GPIO_NOPULL;
1017   HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
1018
1019   /*Configure GPIO pins : PB1 PB11 PB12 PB13
1020    PB14 PB15 PB8 PB9 */
1021   GPIO_InitStruct.Pin = GPIO_PIN_1 | GPIO_PIN_11 | GPIO_PIN_12 |
       GPIO_PIN_13
1022       | GPIO_PIN_14 | GPIO_PIN_15 | GPIO_PIN_8 | GPIO_PIN_9;
1023   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1024   GPIO_InitStruct.Pull = GPIO_NOPULL;
1025   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
1026
1027   /* EXTI interrupt init*/
1028   HAL_NVIC_SetPriority(EXTI4_15_IRQn, 0, 0);
1029   HAL_NVIC_EnableIRQ(EXTI4_15_IRQn);
1030
1031 }
1032
1033 /**
1034  * @brief  This function is executed in case of error occurrence.
1035  * @retval None
1036  */
1037 void Error_Handler(void) {
1038   __disable_irq();
1039   while (1) {
1040   }
1041
1042 }
```

Quellcode A.2: Gesamter Softwarecode

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original