



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bussysteme und Sensorik - Projekt

**Johanna Böttcher, Marcel Eßmann,
Peter Fischer, Jannik Wendt**

Modulares Eingabesystem

*Fakultät Technik und Informatik
Studiendepartment Elektro- und
Informationstechnik*

Johanna Böttcher, Marcel Eßmann,
Peter Fischer, Jannik Wendt

Modulares Eingabesystem

Bussysteme und Sensorik - Projekt

im Studiengang Elektro- und Informationstechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuer Prüfer: Prof. Dr. Robert Fitz

Eingereicht am: 17. Dezember 2024

Inhaltsverzeichnis

1 Einleitung	1
2 Modulares Eingabesystem	2
2.1 Projektbeschreibung	2
2.1.1 Verbindung	4
2.1.2 Topologien	5
2.2 Herangehensweise	8
2.2.1 Erfolge	9
2.2.2 Probleme	12
2.2.3 Budget	14
2.2.4 Weiteres Vorgehen	15
2.3 Datenbus	17
2.3.1 Aufbau des Datenbusses	17
2.3.2 Technische Eigenschaften	18
2.4 Programme	23
2.4.1 Modulübergreifende Funktionen	23
2.4.2 Controller Modul - Programmcode	29
2.4.3 Code der Eingabemodule	35
2.5 Input-Mapper Programm	39
2.5.1 Hauptfunktionen	40
2.6 Schaltpläne	43
2.6.1 Controller Modul	43
2.6.2 Tastatur-Modul	45
2.6.3 Alternativen zum Transceiver	46
2.7 Platine	48
2.7.1 Verbesserungen	48
2.7.2 Layout	52

Inhaltsverzeichnis

2.8 Fazit	54
2.8.1 Persönliches Fazit	54
3 Anhang	56
3.0.1 Main Programm	56
3.0.2 Main Header	78
3.0.3 Main Header	82

Abbildungsverzeichnis

2.1	Vereinfachte Darstellung des Eingabesystems	3
2.2	Verbindung der Module durch Pogo-Stecker und Magnete	4
2.3	Vermischte Topologie	5
2.4	Bustopologie	6
2.5	Sterntopologie	7
2.6	Aktueller Entwicklungs-Prototyp	9
2.7	GitGraph stand 19.01.2025. Die Linie links stellen die Unterschiedlichen Branches dar. Ein Punkt auf der Linie bedeutet das dort ein Commit erstellt wurde. Das Datum ist das Commitdatum und der Name die Person welche den Commit erstellt hat.	11
2.8	Zeitmanagement stand 17.12.2024	13
2.9	Exceltable der Ausgaben, stand 19.01.2025. Darstellung der gekauften Bauteile und von wem diese gekauft wurden. Mit Grün hinterlegte Felder sind auf dem aktuellen Prototypen verbaute Bauteile. In F1 steht der Gesamtbetrag der bisher investiert wurde	14
2.10	Stream Deck Preis von elgato. Dies dient lediglich als Vergleich in welcher Preis-Kategorie unser Produkt vermarktet werden könnte. [https://www.elgato.com/de/de/p/stream-deck-mk2-black , 19.01.2025]	15
2.11	Aufbau des Datenbusses	18
2.12	Kommunikation zwischen zwei ESP32 über unseren Datenbus.	19
2.13	Beispiel differenzielle Datenübertragung	19
2.14	CRC-16-Bit Generatorpolynom (0x1021)	20
2.15	Darstellung von Bitstuffing an einem Beispiel. In diesem Beispiel handelt es sich um 5-Bit Bitstuffing. [https://prayasnotty.wordpress.com/wp-content/uploads/2010/11/bit12.png , 17.01.2025]	22
2.16	Programmablauf Controller Modul	30
2.17	Programmablauf eines Eingabemoduls	36
2.18	Benutzeroberfläche des Inputmapper-Programms	39

Abbildungsverzeichnis

2.19	JSON-Datei zum Speichern und Laden von Konfigurationen	41
2.20	Infofenster mit beispielhaften Belegungen	42
2.21	ESP32 Verschaltung	43
2.22	UART-Bridge	44
2.23	Transmit-(SN65LVS1D) und Receiver(SN65LVDT2D) Bausteine	44
2.24	Tastatur Schaltung	45
2.25	ATmega Verschaltung	46
2.26	Optokoppler Schaltung	47
2.27	Mikrostrip-Topology	51
2.28	Stripline-Topology	51
2.29	Abstand Leiterbahnen	51
2.30	Layout Controller-Modul	52
2.31	Layout Tastatur-Modul	53

Listings

2.1	CRC-Encoder Funktion. Noch nicht implementiert, dies ist lediglich ein Ansatz der noch im Programm einzubinden ist.	20
2.2	CRC-Decoder Funktion. Noch nicht implementiert, dies ist lediglich ein Ansatz der noch im Programm einzubinden ist.	20
2.3	Bitstuffing	23
2.4	Senden einer Nachricht	24
2.5	Empfangen einer Nachricht	25
2.6	decodeBitstuffedMessage()	27
2.7	parseDecodedMessage()	28
2.8	Struct device	30
2.9	Zuweisen einer neuen Adresse	31
2.10	Timeout Funktion	32
2.11	Polling Funktion	33
2.12	Datenübertragung an den angeschlossenen Computer	34
2.13	Anfordern einer Adresse	37
2.14	Beispiel Funktion Tastaturmodul	37
3.1	main.cpp	56
3.2	header.h	78
3.3	Input-Mapper Program; Durch die Formatierung in LaTeX lässt sich dieser schwierig lesen, am besten die mitgelieferte Python Datei öffnen und anschauen.	82

1 Einleitung

Moderne Eingabegeräte sind ein essenzieller Bestandteil des digitalen Zeitalters und finden in verschiedensten Bereichen wie der Datenverarbeitung, Musikproduktion und erst recht in der industriellen Steuerung Anwendung. Die Anforderungen an diese Geräte können dabei sehr vielseitig sein. Um diesen Anforderungen gerecht zu werden, befasst sich das Projekt mit dem Ziel, ein modulares Eingabesystem zu entwickeln.

Die Modularität ermöglicht es, verschiedene Module wie z.B. ein Tastatur-Modul mit einem 4x4-Tastenraster oder ein Audiomodul mit Potenziometern und Pegelstellern in einem erweiterbaren System zu kombinieren. Dies bietet die Möglichkeit, das System entsprechend der individuellen Vorstellung zu erweitern.

Im Mittelpunkt des Projekts steht die Entwicklung eines speziell entworfenen Bussystems, das eine Kommunikation zwischen den einzelnen Modulen ermöglicht. Zu diesem Datenbus gehört die Umsetzung der Bitübertragung, die Adressierung und Skalierbarkeit der Module, sowie Timing und Synchronisation. Ziel ist es, einen robusten und effizienten Datenbus zu schaffen, der eine hohe Busgeschwindigkeit sowie fehlerfreie Übertragungen gewährleistet.

Ergänzend dazu umfasst das Projekt die Auswahl und Validierung geeigneter elektronischer Komponenten, die Optimierung des PCB-Designs sowie die Entwicklung eines Gehäuses, das die Modularität des Systems unterstützt. Darüber hinaus wird ein Übersetzerprogramm entwickelt, das die Daten zwischen den Modulen und des benutzten Computers verarbeitet. Der Benutzer wird, in dem Programm die Möglichkeit bekommen, die Tasten nach Belieben zu belegen.

Das Ziel ist eine vielseitige und individuell anpassbare Lösung für Anwender, insbesondere im Bereich von Programmen wie den Adobe-Bearbeitungstools, die zahlreiche wichtige Tastenkombinationen erfordern.

2 Modulares Eingabesystem

2.1 Projektbeschreibung

Das Eingabesystem verfolgt die Vision eines flexiblen und erweiterbaren Systems, das sich an die individuellen Bedürfnisse der Benutzer anpasst. Im Zentrum steht ein Hauptmodul, das als Mikrocontroller-Einheit dient und über eine USB-Schnittstelle mit einem Computer verbunden wird. Dieses Hauptmodul koordiniert die Kommunikation und Funktionalität der verbundenen Module.

Die Module, wie ein Tastatur-Modul mit einer 4x4-Tastenmatrix oder ein Audio-Modul mit Potentiometern und Fadern, sind frei kombinierbar und können je nach Anwendungsfall hinzugefügt oder entfernt werden. Dies ermöglicht eine Konfiguration für unterschiedliche Aufgaben wie Textverarbeitung, Bildbearbeitung, Audiobearbeitung oder Gaming.

In Abbildung 2.1 ist ein Überblick des Eingabesystems zu sehen.

2 Modulares Eingabesystem

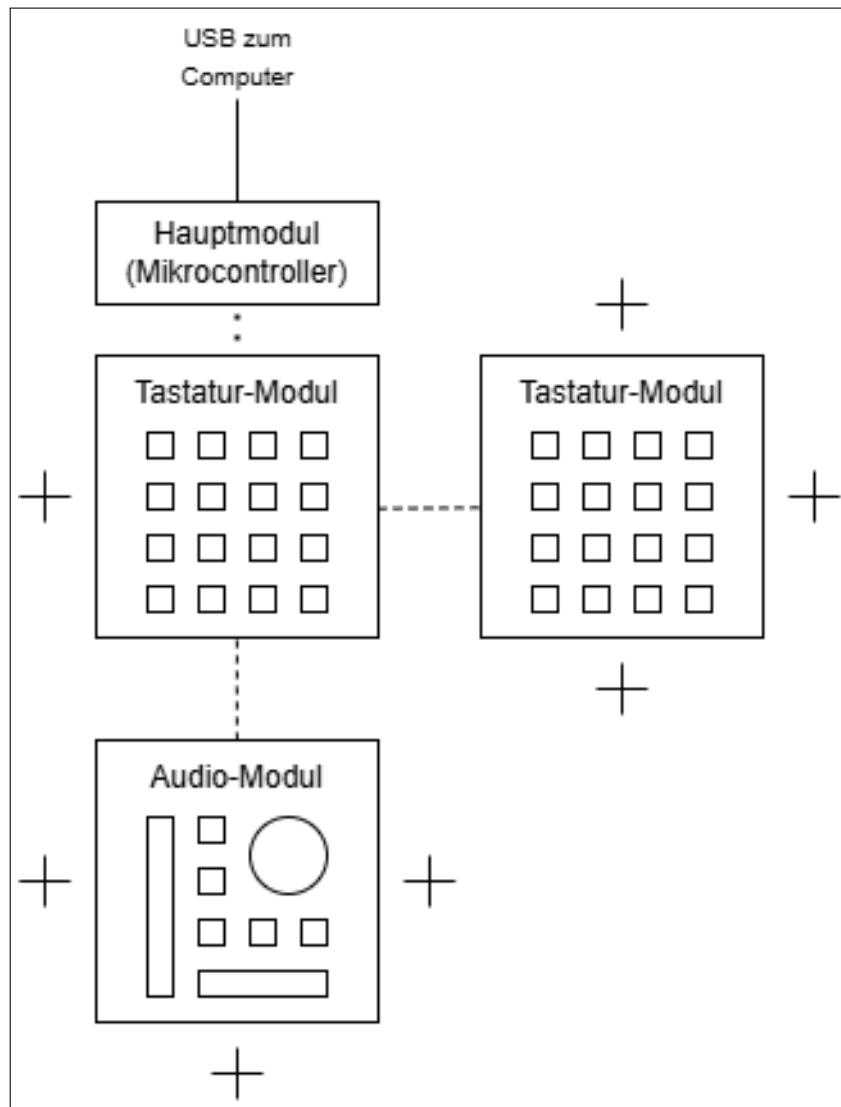


Abbildung 2.1: Vereinfachte Darstellung des Eingabesystems

Die gestrichelten Linien stellen eine bereits hergestellte Verbindung zwischen den Modulen dar, wobei die „Plus“ Symbole einen weiteren möglichen Modulanschluss darstellen. Die Module sollen fest aneinander sitzen.

2.1.1 Verbindung

Zwischen den Modulen werden Daten und Spannungsversorgung über eine 4 Pin-Verbindung realisiert. Diese sind VCC (Spannungsversorgung), Data + (Datenbus), Data - (invertierter Datenbus) und GND (Masse). Da der Benutzer die Möglichkeit haben soll, an jede Seite eines Moduls ein weiteres anzustecken, werden pro Seite zwei 2-Pin Pogo Stecker verwendet, ein 2-Pin Male und ein 2-Pin Female Stecker. Dadurch ist das System in jede Richtung erweiterbar. Um eine feste Verbindung zu gewährleisten, werden Neodym-Magnete ins Gehäuse verbaut. Dies stellt sicher, dass die Pins fest miteinander verbunden sind und Kontakt haben. Zusätzlich soll mit einem Verlängerungskabel auch die Möglichkeit bestehen eine kurze Distanz (< 50cm) zwischen zwei Modulen zu haben.

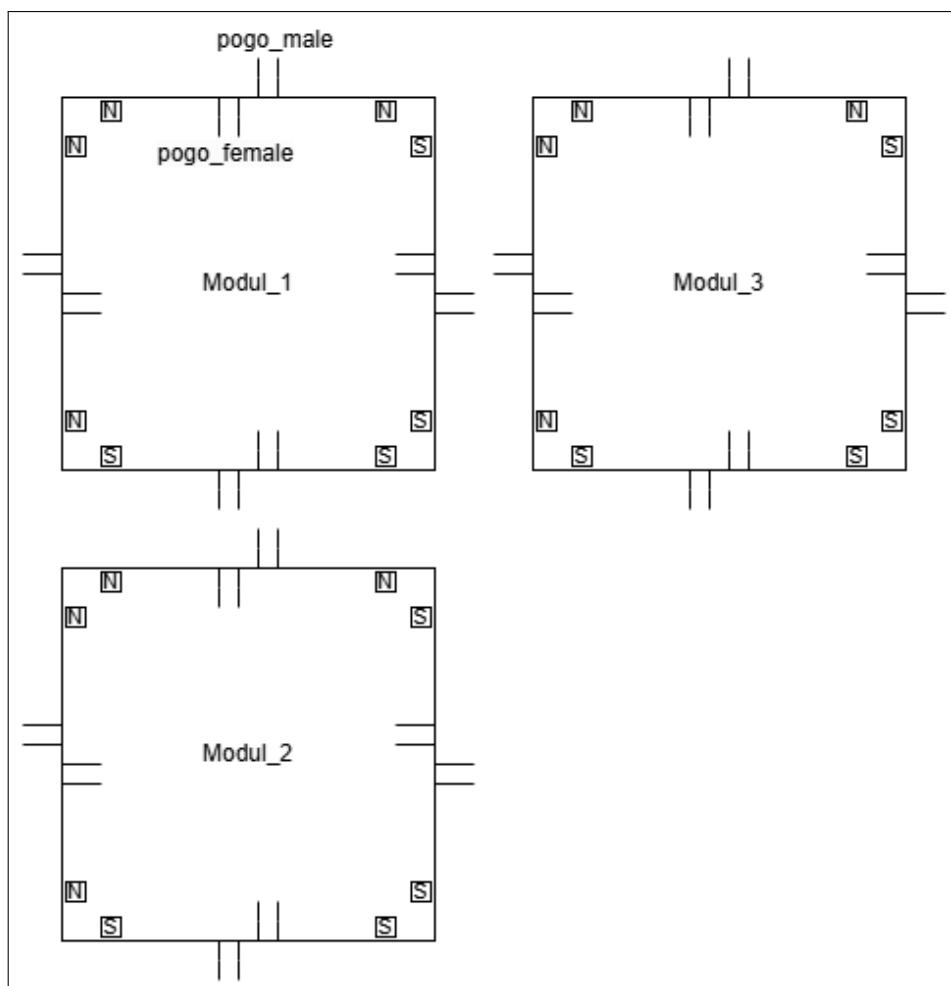


Abbildung 2.2: Verbindung der Module durch Pogo-Stecker und Magnete

Abbildung 2.2 zeigt, dass die Steckverbindungen eine nahezu beliebige Anordnung der Module zulassen. Durch die Anordnung der Magneten lassen sich nicht alle Module in jeder beliebigen Anordnung verbinden, wie z.B. Oberseite und Oberseite, die aufgrund der physikalischen Eigenschaften der Magnete nicht verbunden werden können. Die „N“ und „S“ Symbole stellen die im Gehäuse verbauten Magneten (Nord- und Südpol) zum Zusammenhalt der Module dar.

2.1.2 Topologien

Der Hardware-Aufbau des Eingabesystems entspricht einer Vermaschten-Topologie. Das verdeutlicht die Abbildung 2.1, denn mit jedem neuen Modul kann auch eine Verbindung zu einem schon vernetzen Modul entstehen. Die Vorteile dieser Topologie werden im Gesamtsystem nicht genutzt, denn die Vermaschung kommt lediglich durch die notwendige Verbindung der Module untereinander zustande.

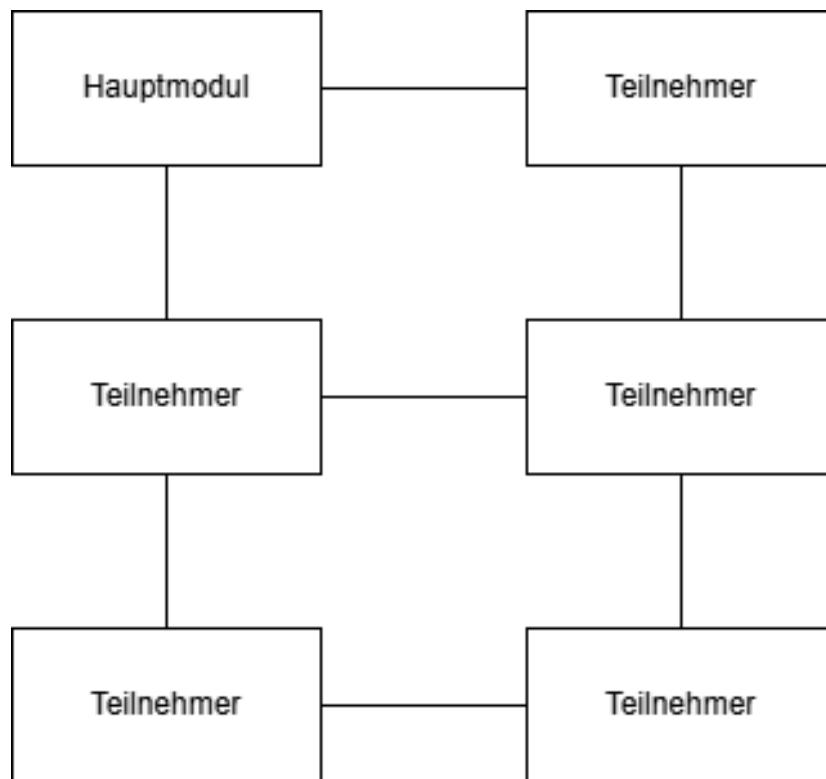


Abbildung 2.3: Vermaschte Topologie

2 Modulares Eingabesystem

Der Bus hingegen nutzt die Bus-Topologie, denn alle Teilnehmer sind über den Bus miteinander verbunden. Dies hat die Vorteile:

- **Einfache Installation:** Weniger Kabel und einfacher Aufbau, da alle Geräte an einer gemeinsamen Leitung (Bus) angeschlossen sind.
- **Einfache Erweiterung:** Neue Geräte können leicht angeschlossen werden.
- **Effizient für kleine Netzwerke:** Gut geeignet für kleine Netzwerke mit wenigen Geräten.

Mit einem Nachteil der Störanfälligkeit. Da der Bus als zentrale Verbindung genutzt wird, können beispielsweise Schäden am Bus die Störanfälligkeit erhöhen und die Verbindung beeinträchtigen. Es gilt, diese Schwachstelle bestmöglich zu schützen.

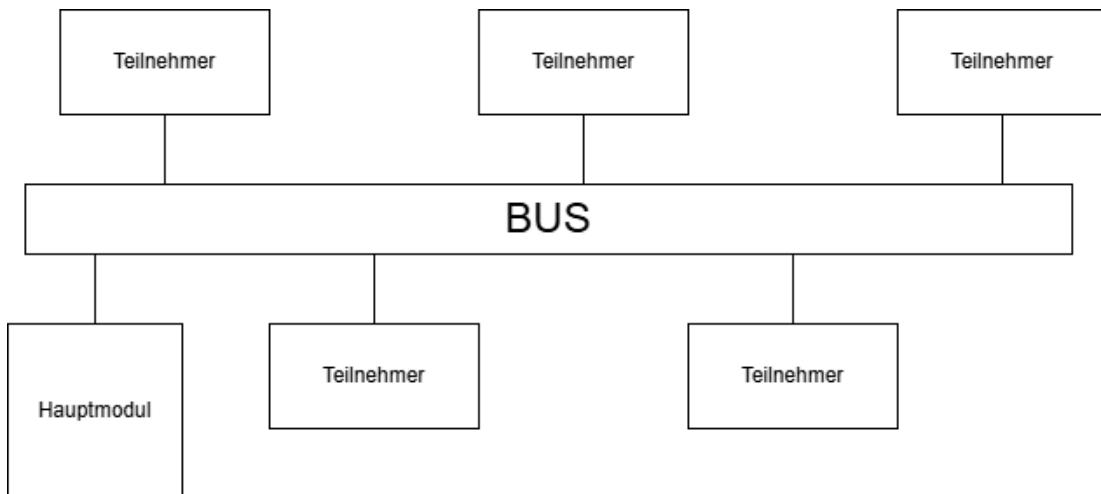


Abbildung 2.4: Bustopologie

Auf der Softwareebene ist das Konzept nach der Stern-Topologie aufgebaut, denn das Hauptmodul ist eine zentrale Steuereinheit, an die alle anderen Module angeschlossen sind. Die verschiedene Module sind jeweils direkt an den Bus angebunden. Die Verbindungen laufen sternförmig von dem Hauptmodul zu den einzelnen Modulen. Die Vorteile einer Stern-Topologie in diesem Projekt sind:

- **Skalierbarkeit:** Weitere Module können einfach an das Hauptmodul angeschlossen werden.
- **Einfache Fehlersuche:** Da jedes Modul eine direkte Verbindung zum Hauptmodul hat, ist der Ausfall eines Moduls leichter zu erkennen.
- **Kontrolle:** Alle Daten laufen durch das Hauptmodul.

Wobei nur ein Nachteil festgestellt werden kann:

- **Abhängigkeit:** Fällt das Hauptmodul aus, funktioniert das gesamte System nicht mehr.

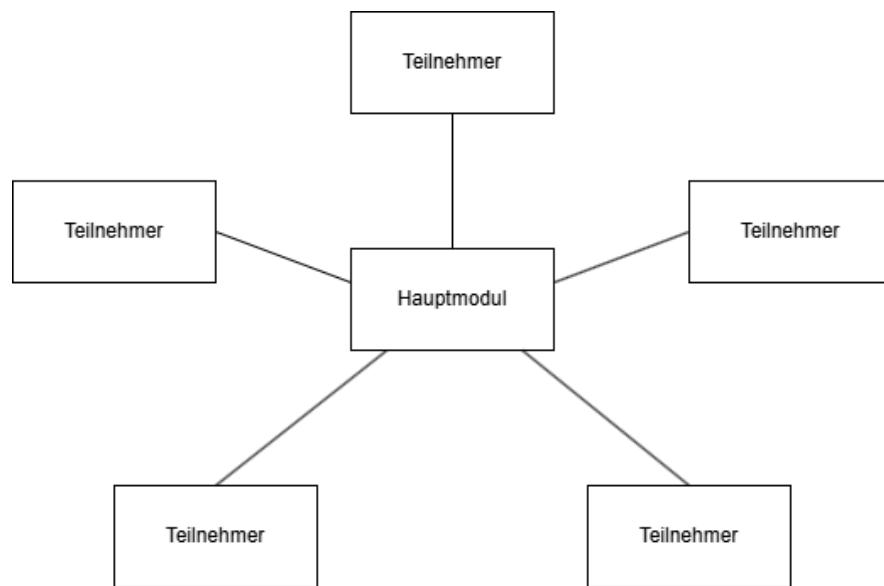


Abbildung 2.5: Stern topologie

2.2 Herangehensweise

Für die Umsetzung des Projekts wurde agiles Projektmanagement gewählt. Im Rahmen der agilen Entwicklung wurden wöchentliche Sprints festgelegt, in denen Gruppenmitglieder den Fortschritt der jeweiligen Aufgaben präsentieren. Dies ermöglicht es, bei auftretenden Problemen gemeinsam Lösungen zu erarbeiten. Durch diese Struktur lassen sich Projektziele dynamisch anpassen, wodurch eine schnelle Reaktion auf Probleme oder Engpässe gewährleistet ist.

Zu Beginn wurde ein Endziel definiert, das in Abschnitt 2.1 beschrieben ist und auf das konsequent hingearbeitet wird. Innerhalb des ersten Sprints konnte erfolgreich eine einfache Datenübertragung zwischen einem ESP32 und einem Arduino-Nano hergestellt werden.

In den darauf folgenden zwei Sprints wurde ein erster einfacher Prototyp des Busses entwickelt. Dieser umfasste einen Eindraht-Bus, wobei die Software zu diesem Zeitpunkt lediglich unidirektionale Kommunikation zwischen einem Sender und mehreren Empfängern unterstützte.

Im weiteren Verlauf des Projekts wurden Tasten sowie zusätzliche Teilnehmer auf dem Prototyp implementiert. Die Tasten dienten der Simulation verschiedener Busteilnehmer. Jedem Taster wurde ein Mikrocontroller (ATtiny oder ATmega) zugeordnet, um die Adressierung der unterschiedlichen Module zu entwickeln und zu validieren. Das gewünschte Verhalten bestand darin, dass ausschließlich der Mikrocontroller mit der entsprechenden Adresse aktiviert wird, wodurch eine LED zum Leuchten gebracht wird.

Anschließend wurde der Prototyp um eine weitere Busleitung erweitert und die differenzielle Datenübertragung angegangen. Hierbei sind einige Schwierigkeiten bezüglich der Hardware aufgetreten. Nach einem Neubau des Prototyps auf dem Breadboard und Reduzierung der Teilnehmer aufgrund von Platzmangel auf dem Breadboard ließ sich ein Keypad-Modul und der Hauptcontroller aufbauen (Abb. 2.6).

Mit diesem Stand des Prototyps lässt sich ein 2x2 Keypad beliebig belegen und zum Beispiel eine PowerPoint-Präsentation steuern.

2 Modulares Eingabesystem

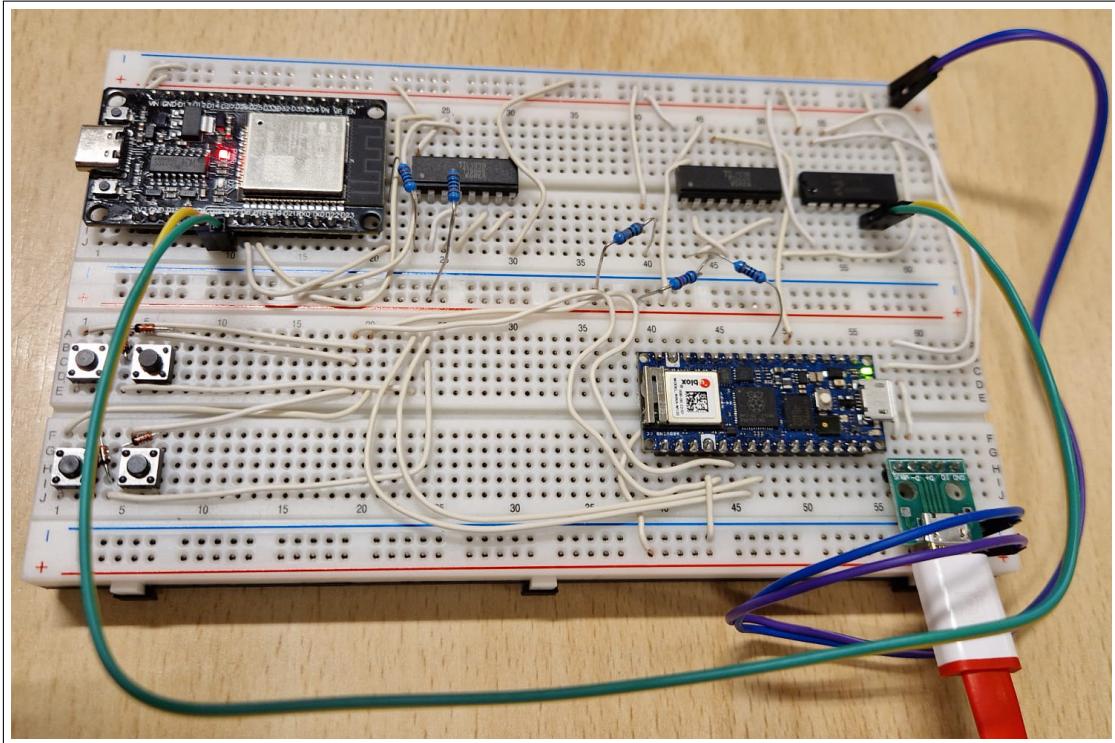


Abbildung 2.6: Aktueller Entwicklungs-Prototyp

Nach Bestellen der PCBs konnte ein erster richtiger Aufbau der Module realisiert werden und auch die Funktionalität der Hardware Bedingungen (Pogo-Pin Verbindung und Mangetische Halterung) können nun getestet werden. Die PCBs sind, aufgrund des Budget, teilbestückt bestellt worden, daher werden diese jetzt von Hand final bestückt.

2.2.1 Erfolge

Synchronisation

Zur Synchronisation des Empfängers mit dem Sender haben wir eine Funktion geschrieben, welche bei einem HIGH-Value auf dem Bus einen Zeitstempel setzt und sobald der Bus auf LOW fällt, die Übertragungsgeschwindigkeit ausrechnet. Damit wird sichergestellt, dass die übertragenen Bits richtig ausgelesen werden.

Datenübertragung

Durch die selbst geschriebene Software lassen sich in der Theorie beliebig große Datenmengen übertragen, solange die Anzahl an Datenbytes im Code festgelegt wurde. In der Praxis sind große Datenmengen vermutlich problematisch, da die Synchronisation nur zu Beginn des Frames stattfindet. Da wir in unserem Projekt nur Eingabewerte übertragen, welche 8 Byte nicht überschreiten werden, haben wir uns auf ein Maximum von 8 Byte festgelegt. Sollte sich an einem zukünftigen Entwicklungspunkt herausstellen, dass wir mehr Daten übertragen wollen, lässt sich das leicht anpassen, aber es sollte auch eventuell eine Neusynchronisation überlegt werden.

Adressvergabe und Polling loop

Das Hauptmodul startet eine dauerhafte vordefinierte Abfrage der Adressen. Über die erste abgefragte Adresse wird einem neuen Modul eine Adresse vergeben und diese mit der Art des Moduls in Hauptmodul abgespeichert. Die erhaltenen Werte der Module gibt das Hauptmodul zusammen mit der Modul-Adresse an den PC weiter.

Input-Mapper Program

Als Bedienoberfläche wurde ein Program geschrieben welches die Empfangen Daten auf vom Nutzer definierte Keybindings mappt.

Parallele Entwicklung

Das parallele Entwickeln war durch das Verwenden des Versionsverwaltungs-Programms „Git“ und einem zentralen Repository auf „GitHub“(<https://github.com/Matzel13/BU-Projekt>) von Beginn an erfolgreich und hat sich durch die Länge des Projekts gezogen. Durch das Erstellen von persönlichen „Branches“ konnte jeder Entwickler an seinem eigenen Fortschritt arbeiten und bei Bedarf mit dem Hauptzweig zusammenführen. In Abb. 2.7 ist ein Ausschnitt des GitGraphen unseres Repository zu sehen.

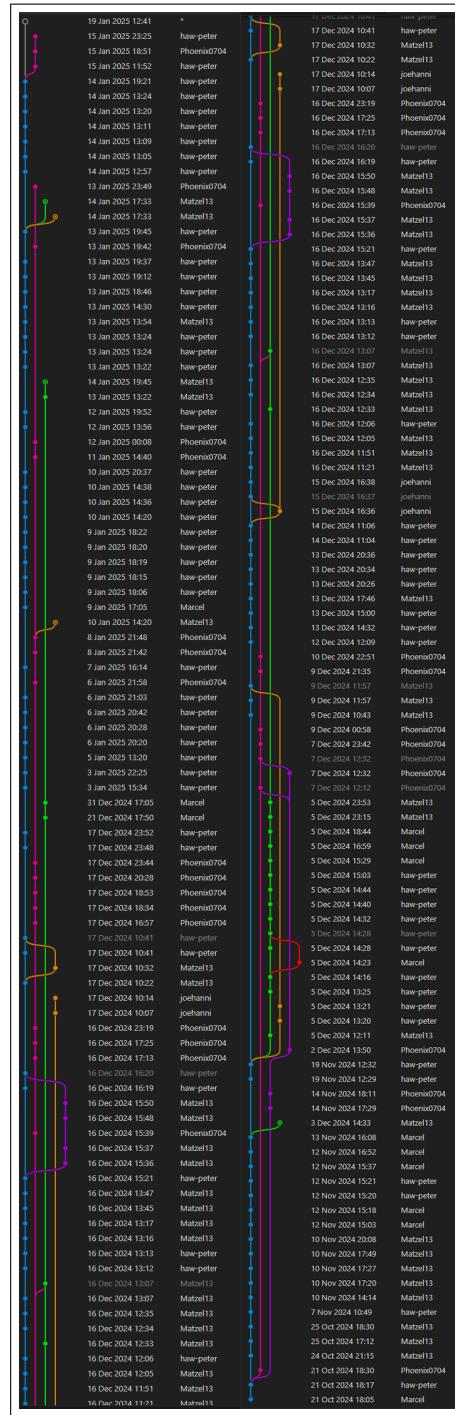


Abbildung 2.7: GitGraph stand 19.01.2025. Die Linie links stellen die Unterschiedlichen Branches dar. Ein Punkt auf der Linie bedeutet das dort ein Commit erstellt wurde. Das Datum ist das Commitdatum und der Name die Person welche den Commit erstellt hat.

2.2.2 Probleme

ATtiny und ATmega

Die eingesetzten Mikrocontroller (ATtiny und ATmega) lassen sich nicht direkt über USB programmieren. Aus diesem Grund wurde innerhalb zweier Sprints der versucht, einen eigenen ISP-Programmer (In-System Programmer) zu entwickeln. Trotz vieler verfügbarer Anleitungen und Forenbeiträge zu bereits selbstgebauten Flashern gelang es nicht innerhalb des vorgegebenen Zeitrahmens, einen funktionierenden Programmer zu erstellen. Schließlich wurde ein fertiger ISP-Programmer gekauft, wodurch die Programmierung der Mikrocontroller schnell und erfolgreich möglich wurde.

Differenzielle Datenübertragung

Der erste Ansatz zum Erstellen einer differenziellen Datenübertragung war es, dies über die Software zu lösen. Jedoch kam nach gemeinsamer Absprache der Entschluss, dass dies nicht sonderlich förderlich sei und hauptsächlich der Geschwindigkeit des Busses schadet. Bei der Erweiterung des Eindraht Prototypen auf einen Zweidraht Bus mit differenzieller Datenübertragung entstand das Problem, dass in einigen Fällen die Busleitung kurzgeschlossen wird. Dieser Kurzschluss lässt sich durch galvanische Trennung der Teilnehmer vom Bus, ähnlich wie beim CAN und USB, lösen. Um weiterhin mit den zur Verfügung gestellten Produkten zu arbeiten, wurde erstmal ein Optokoppler der HAW getestet. Scheinbar war dieser Optokoppler defekt, da sich die Schaltzeiten im ms Bereich bewegt haben. Dies wäre ein viel zu hohe Latenz für die Datenübertragung und daher wurden vier mögliche alternative Bausteine bestellt, zwei verschiedene Optokoppler, einen Operationsverstärker und einen digitalen Isolator. Die Optokoppler lieferten ein zufriedenstellendes Ergebnis mit einer Latenz im einstelligen μs Bereich. Zusätzlich zu den Optokopplern wurde ein Transmitter und Receiver Bauteil entdeckt, da jedoch die Lieferzeit für diese Bauteile sehr hoch ist, werden diese Bauteile vom PCB Hersteller direkt auf das PCB designt, welches das Testen der Bauteile leider nicht ermöglicht. Der aktuelle Prototyp wurde mit diskret aufgebauten Invertierern und Optokopplern realisiert.

Ungetestete Bauteile

Wie im vorherigen Absatz erwähnt, werden Bauteile benutzt, die leider im Vorfeld nicht getestet werden können. Daher wurden Alternativen im PCB-Design implementiert, falls diese Bauteile nicht die Erwartungen erfüllen. Bei der Entwicklung der PCB und der Lösung des Problems der differenziellen Datenübertragung werden die Bauteile der SN65LVDXXX Familie benutzt.

2 Modulares Eingabesystem

Aufgrund der erwähnten Lieferzeit konnten die Bauteile nicht getestet werden. Um etwaige Probleme mit den Bauteilen zu umgehen, werden mehrere Jumper ins Design geplant, um die Bauteile mit Optokopplern oder eine externe Testschaltung oder Testbauteil zu ersetzen.

Krankheit und Lieferzeiten

Aufgrund von Krankheiten konnte die Gruppe nicht an jedem Termin vollständig arbeiten, oder musste bereits fertige Pläne umstrukturieren. Zusätzlich kam es zu Verzögerungen durch Lieferzeiten von ein bis zwei Wochen der Bauteile.

Einen groben Übersichtsplan des Zeitmanagements ist in Abbildung 2.12 zu sehen. Deutlich erkennbar sind die KW44 und KW48, in denen kein Meeting stattfinden konnte. Zusätzlich sind alle Vorlesungszeiten mit im Plan aufgenommen, da die Vorlesungen teilweise genutzt wurden, um am Projekt zu arbeiten. Der Zeitplan dient lediglich einer groben Analyse der investierten Zeit und hat keinen Einfluss auf das Vorgehen.

	Vorlesung	Praktikum	Meeting	Freizeit		Vorlesung	Praktikum	Meeting	Freizeit		Vorlesung	Praktikum	Meeting	Freizeit
KW41	12	0	0	0	KW42	12	0	12	3	KW43	12	12	8	4
Johanna	3				Johanna	3		3		Johanna	3	3	2	
Peter	3				Peter	3		3		Peter	3	3	2	
Jannik	3				Jannik	3		3		Jannik	3	3	2	
Marcel	3				Marcel	3		3	3	Marcel	3	3	2	4
KW44	12	0	0	2	KW45	12	0	4	7	KW46	12	0	20	3
Johanna	3				Johanna	3		1		Johanna	3		5	
Peter	3				Peter	3		1	2	Peter	3		5	
Jannik	3				Jannik	3		1	3	Jannik	3		5	
Marcel	3				Marcel	3		1	2	Marcel	3		5	3
KW47	12	12	8	0	KW48	0	0	0	4	KW49	12	0	8	14
Johanna	3	3	2		Johanna				1	Johanna	3		2	5
Peter	3	3	2		Peter				1	Peter	3		2	1
Jannik	3	3	2		Jannik				1	Jannik	3		2	5
Marcel	3	3	2		Marcel				1	Marcel	3		2	3
KW50	12	0	11	22	KW51	0	0	8	41	KW52	0	0	0	31
Johanna	3		3	6	Johanna				2	Johanna				20
Peter	3		3	6	Peter				2	Peter				
Jannik	3		3	5	Jannik				2	Jannik				11
Marcel	3		2	5	Marcel				2	Marcel				
KW01	0	0	0	47	KW02	12	0	12	32	KW03	0	12	0	21
Johanna				8	Johanna	3		3	4	Johanna	3		4	
Peter				25	Peter	3		3	15	Peter	3		4	
Jannik				8	Jannik	3		3	4	Jannik	3		4	
Marcel				6	Marcel	3		3	9	Marcel	3		9	
KW04	0	0	0	21						Gesamt	499			
Johanna				4						Johanna	134			
Peter				4						Peter	120			
Jannik				5						Jannik	122			
Marcel				8						Marcel	123			

Abbildung 2.8: Zeitmanagement stand 17.12.2024

2 Modulares Eingabesystem

2.2.3 Budget

Zur Entwicklung der Testaufbauten und des Prototypen wurden dieverse Bauteile bestellt. Da wir im Anschluss an das Hochschulmodul jeder eine Version unseres Projektes behalten wollten haben wir uns entschieden das Projekt selber zu finanzieren. Um die Kosten der Module gering zu halten haben wir bei jeder Entscheidung darauf geachtet die günstigste Version zu finden. In Abbildung 2.9 sind alle Bauteile die bestellt worden sind aufgelistet und diese wurde genutzt einen Überblick über unsere Ausgaben zu behalten. Durch die Liste in Abbildung 2.9 lässt sich auch gut erkennen wie teuer die Modul-Prototypen sind, das Hauptmodul befindet sich derzeitig bei 39,68€ und das Keypad-Modul bei 44,61€. Diese Preise lassen sich in Zukunft vermutlich noch reduzieren, da die PCBs mit verschiedenen Testcases bestückt sind.

	A	B	C	D	E	F	G	H	I
	gekauft von	Gegenstand	Anzahl	Stückpreis	Gesamtpreis	347,07 €		Legende:	
1									
2	Marcel	CHERRY MX Brown - Tastenmodul - Fixierzapfen	50	0,50 €	24,75 €			Im Prototypen verbaut	
3	Marcel	1-fach Optokoppler, 3,75kV, 80V, 50mA, 50-600%, MFP-4	20	0,20 €	4,00 €				
4	Marcel	Digitaler Isolator, 2-Kanal, 2,7 ... 5 V, 1 Mbps, 150 ns, SO-8	2	2,75 €	5,50 €				
5	Marcel	Operationsverstärker, 4-fach, 1 MHz, 1 V/µs, SO-14	4	0,50 €	2,00 €				
6	Marcel	RS422/423-Differenzempfänger, 4 Empfänger, 5 V, DIL-16	3	1,45 €	4,35 €				
7	Marcel	Optokoppler, 5kV, 15MBd, DIP-8	4	0,50 €	2,00 €				
8	Marcel	USB 2.0 zu I2C/UART Konverter, 3,0 ... 5,0 V, GPIO, DIP-14	1	3,20 €	3,20 €				
9	Marcel	Operationsverstärker, 1-fach, DIP-8	10	0,37 €	3,70 €				
10	Marcel	Schalt-Diode, 100 V, 150 mA, DO-35	100	0,02 €	1,90 €			Kosten Controller:	44,61 €
11	Marcel	POGO Pins 50 female/male 2pol	100	0,25 €	24,79 €			Kosten Keypad:	39,68 €
12	Johanna	Platine Controller	5	16,00 €	80,00 €				
13	Johanna	Platine Tastatur	5	14,00 €	70,00 €				
14	Johanna	SN65LVD12D (Receiver IC mit Terminations Widerstand)	5	3,49 €	17,45 €				
15	Johanna	Optokoppler ILQ021	20	1,81 €	36,20 €				
16	Johanna	BS170 N-Channel Mosfet	20	0,21 €	4,20 €				
17	Johanna	ESP 32	1	9,50 €	9,50 €				
18	Johanna	MCP2222 (USB 2.0 to I2C Converter)	5	3,04 €	15,20 €				
19	Johanna	Atmelga	5	2,47 €	12,35 €				
20	Johanna	Micro USB Buchse	10	0,40 €	4,00 €				
21	Johanna	verschiedene IC Sockel	122	0,09 €	10,98 €				
22	Johanna	Jumpers	275	0,04 €	11,00 €				
23					0,00 €				

Abbildung 2.9: Exceltablle der Ausgaben, stand 19.01.2025. Darstellung der gekauften Bauteile und von wem diese gekauft wurden. Mit Grün hinterlegte Felder sind auf dem aktuellen Prototypen verbaute Bauteile. In F1 steht der Gesamtbetrag der bisher investiert wurde

Das Produkt lässt sich sehr gut mit dem Stream Deck von elgato vergleichen. Bei einer Möglichen vermarktung unseres Produkt kann sich an dem Presi des Stream Deck orientert werden. Dazu ist in Abbildung 2.10 der offizielle Preis des 3x5 Keypad Stream Deck von elgato dargestellt.

2 Modulares Eingabesystem

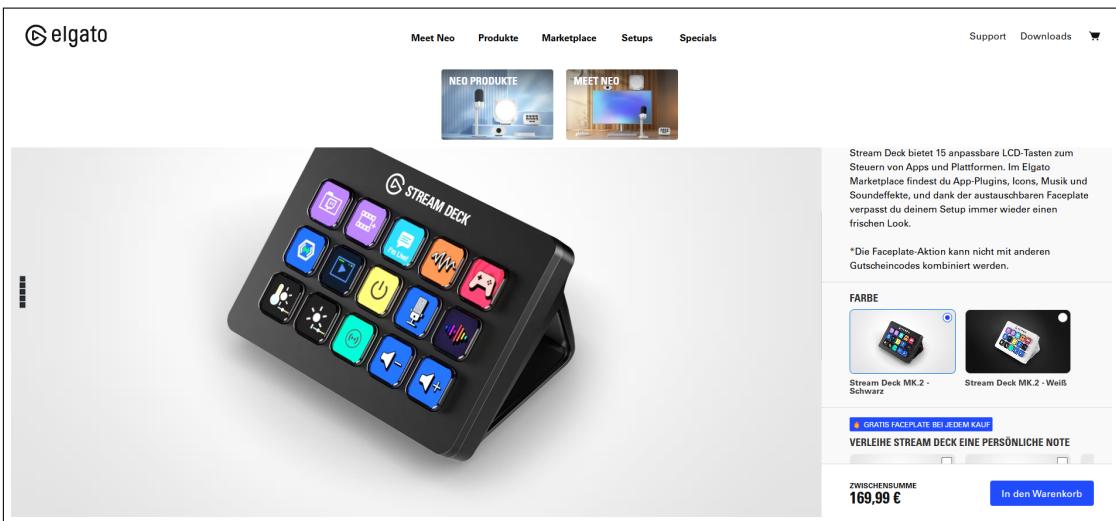


Abbildung 2.10: Stream Deck Preis von elgato. Dies dient lediglich als Vergleich in welcher Preis-Kategorie unser Produkt vermarktet werden könnte.
[<https://www.elgato.com/de/de/p/stream-deck-mk2-black>, 19.01.2025]

Mit dem Vergleich des elgato Produkts lässt sich ein Vermarktungspreis von ca 200€ für das Hauptmodul und Keypad Modul in Kombination festlegen. Dies lässt sich durch unsere Modularität und Erweiterungsmöglichkeiten begründen. Weitere Module sollten einen Einzelpreis von ca 60-80€ erhalten. Dies sind bisher allerdings lediglich Überlegungen, da wir uns noch in der Forschungsperiode befinden.

2.2.4 Weiteres Vorgehen

Wir wollen das Produkt auch nach Abschluss des Faches „Bussysteme und Sensorik“ weiterentwickeln, da wir großes Potential sehen dies auch in Zukunft vermarkten zu können.

Als nächstes wollen wir eine intensive Testphase des Prototypen durchführen. Dazu werden wir 5 Prototypen aufbauen und diese auch an externe Personen verteilen um eine größtmögliche Menge an Feedback Daten zu erhalten. Während der letzten Monate konnten wir bereits durch Gespräche mit Freunden mehrere interessierte Tester ausfindig machen.

Geplant ist, die Platinen durch Optimierung der Schaltung und Verwendung von möglichst vielen SMT Bauteilen kleiner und günstiger zu machen und um die Modularität zu optimieren.

2 Modulares Eingabesystem

Außerdem soll die Möglichkeit der Kabelverbindung zwischen den Modulen umgesetzt werden, um dem Nutzenden möglich hohe Flexibilität zu ermöglichen.

Die Funktionen der CRC Fehlererkennung werden fest in die Programme der Module eingearbeitet. Das Input-Mapper Programm bekommt noch weitere Features, zum Beispiel die Möglichkeit eine Zeitverzögerung in die Tastenkombination mit einzubauen.

Im Anschluss an die oben genannten Punkte wird mit der Entwicklung weiterer Module begonnen. Durch die verallgemeinerte Entwicklung des ersten Moduls sollte die Entwicklung weiterer Module relativ einfach sein, da sich lediglich mit der Funktion des Moduls auseinandergesetzt werden muss. Der Kommunikation zwischen dem Modul-Mikrocontrollers und dem Hauptmodul ist im Grunde die gleiche.

2.3 Datenbus

Eines der Hauptbestandteile des modularen Eingabesystems ist das Entwerfen und Umsetzen eines leistungsfähigen und flexiblen Datenbusses. Dieser Datenbus dient als Kommunikationsschnittstelle zwischen den verschiedenen Modulen, wie dem Keypad-Modul und dem Audiomodul und ermöglicht dadurch die Datenübertragung zum Hauptmodul. Skalierbarkeit, Zuverlässigkeit und Geschwindigkeit sind die Hauptkriterien des Datenbusses. Ziel ist es, eine einfache Erweiterbarkeit durch Hinzufügen oder Entfernen von Modulen zu ermöglichen, ohne dabei grundlegende Kommunikationsmechanismen zu beeinträchtigen.

2.3.1 Aufbau des Datenbusses

Der Bus ist nach folgendem Schema aufgebaut:

- **Start of Frame (SOF):** 2 Bit zum Synchronisieren der Frequenzen; durch einen HIGH-Value wird der Beginn des SOF bekanntgegeben und durch das darauf folgende LOW-Value können die Teilnehmer sich auf die Übertragungsgeschwindigkeit synchronisieren.
- **Sendeadresse:** 3 Bit zum Senden der Adresse; Die Adresse 0x00 ist für neue Teilnehmer reserviert, über diese Adresse findet die Adressvergabe statt. Die Adresse 0x01 ist fest an das Hauptmodul vergeben, alle Teilnehmer wissen das auf dieser Adresse das Hauptmodul liegt. Die Adressen 0x02 bis 0x07 stehen für Teilnehmer zur Verfügung. 3 Bit sind vollkommen ausreichend, da das System als Desktop Erweiterung geplant ist und mehr als 6 Module nicht vorgesehen ist. Sollten in Zukunft mehr Teilnehmer erforderlich sein, lässt sich das im Code leicht anpassen.
- **Content of Frame (COF):** 3 Bit für Datenlänge in Byte; das COF enthält die Länge der übertragenden Daten in Byte. Dies dient dazu, dass die Länge der zu übertragenden Daten dynamisch sein kann (bis zu 7 Byte). Zusätzlich können Teilnehmer, welche nicht angesprochen werden, festlegen, wie lange der Bus belegt ist, bevor diese versuchen sollten zu senden.
- **Daten:** 0-7 Byte
- **Fehlererkennung (CRC):** 15 Bit; Mithilfe eines 16-Bit Generatorpolynom lassen sich bis zu 15-Bit Fehler erkennen. Eine fehlerhafte Nachricht soll ignoriert werden.

- **End of Frame (EOF):** 4 Bit Signalende; Vier aufeinanderfolgende LOW-Values geben das Ende der Nachricht an. Ein Teilnehmer, der Senden möchte, wartet bis er 4 aufeinanderfolgende LOW-Value empfangen hat, bevor dieser Sendet.

In Abbildung 2.11 ist eine grafische Darstellung des Busaufbau zu sehen.

2 Bit	3 Bit	3 Bit	X Byte	15 Bit	4 Bit
SOF	Address	COF	Data	CRC	EOF

Abbildung 2.11: Aufbau des Datenbusses

2.3.2 Technische Eigenschaften

Zu dem jetzigen Zeitpunkt kann der selbst erstellte Bus Informationen mit einer Geschwindigkeit von $250 \frac{\text{Byte}}{\text{s}}$ übertragen. Dabei kann eine maximale Teilnehmeranzahl (Eingabemodule) von sieben erreicht werden. Dies liegt hauptsächlich an den Adressierungsbits des Busses und könnte beliebig erweitert werden. Die Datensignale haben einen Pegel von 0 V bis 3,3 V

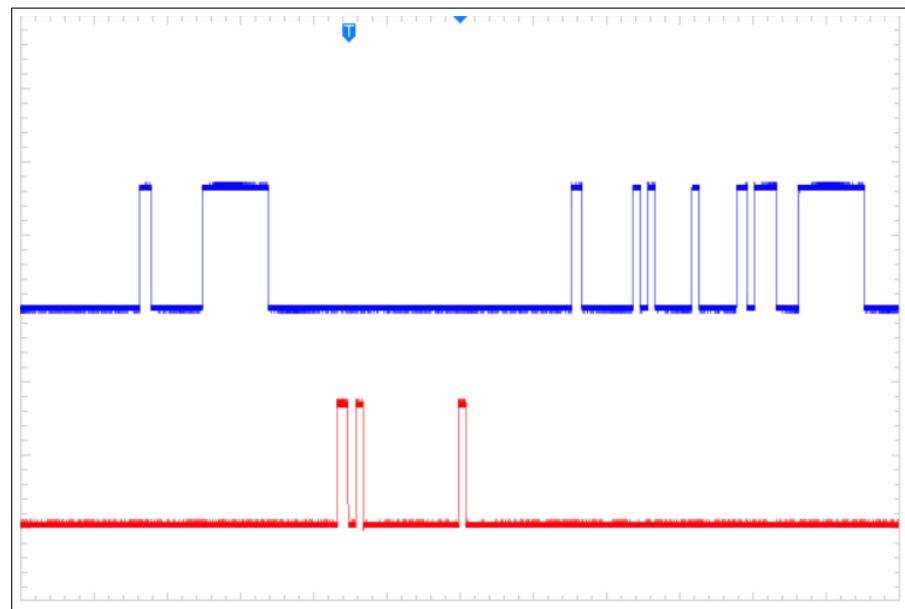


Abbildung 2.12: Kommunikation zwischen zwei ESP32 über unseren Datenbus.

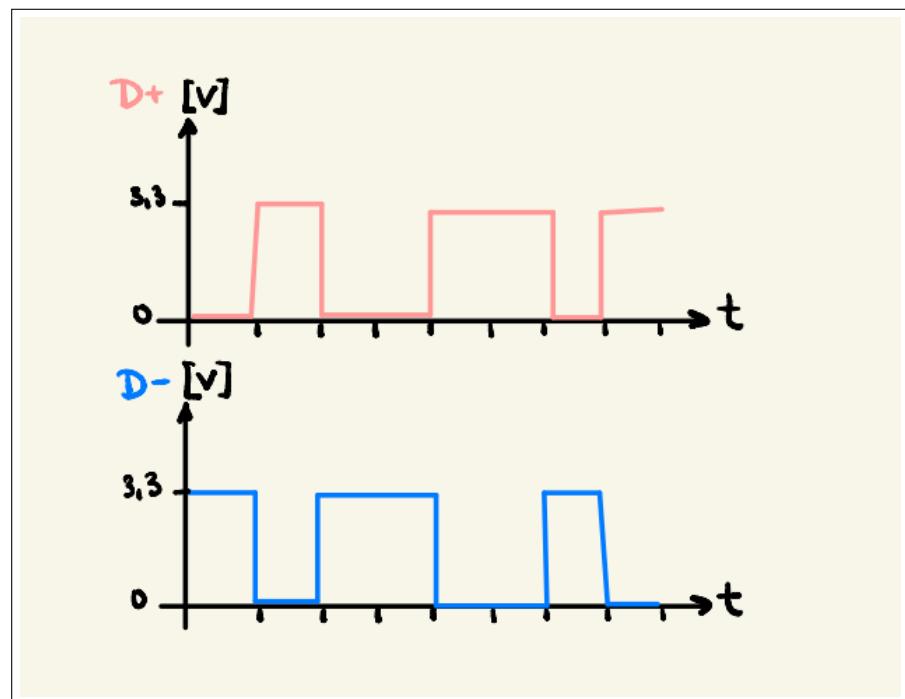


Abbildung 2.13: Beispiel differenzielle Datenübertragung

CRC Fehlererkennung

Zur Erkennung von Bitfehlern wird das 16-Bit Generatorpolynom in Abbildung 2.14 genutzt.



0001000000100001

Abbildung 2.14: CRC-16-Bit Generatorpolynom (0x1021)

```

1 void crcEncode(){
2     uint64_t remainder;
3
4     for(int i = 0; i < 8; i++){
5         // merge the message into a single variable
6         encodedMessage = (encodedMessage << 8)
7             | (uint8_t)global_message[i];
8     }
9     // append as many zeroes as the degree of the polynomial - 1
10    remainder = encodedMessage << 15;
11    // divide the message with the generator polynomial
12    for(int i = 15; i > 0; i--){
13        // check for leading 0 -> no XOR while a leading 0
14        if((remainder << i+1) != 0){
15            remainder = remainder ^ (generator_polynomial << i);
16        }
17    }
18    // append remainder to message
19    // should only append the remainder after the division
20    encodedMessage = (encodedMessage << 15) | remainder
21 }
```

Listing 2.1: CRC-Encoder Funktion. Noch nicht implementiert, dies ist lediglich ein Ansatz der noch im Programm einzubinden ist.

```

1 void crcDecode(){
2     uint64_t remainder;
3     uint64_t message;           // message received
4
5     for(int i = 15; i >= 0; i--){
6         // check for leading 0 -> no XOR while a leading 0
```

```
7 if((remainder << i+1) != 0){  
8     remainder = remainder ^ (generator_polynomial << i);  
9 }  
10 }  
11  
12 for(int i = 0; i < 17; i++){  
13     if((remainder & (0x01 << i)) == 0x01){  
14         // ERROR in the message, ignore this message  
15         break;  
16     }  
17 }  
18 }
```

Listing 2.2: CRC-Decoder Funktion. Noch nicht implementiert, dies ist lediglich ein Ansatz der noch im Programm einzubinden ist.

Mit diesem Generatorpolynom lassen sich folgende Fehler erkennen:

- **Single-Bit Errors:** alle Fehler
- **Double-Bit Errors:** alle Fehler, solange die Distanz zwischen den Fehlern kleiner als 16 Bit ist
- **Burst Errors:** bis zu 16 Bit

Bitstuffing

Da jeder Teilnehmer auf den Bus hört und auf das EOF wartet bevor eine neue Nachricht gesendet wird, haben wir ein 3-Bit Bitstuffing eingebaut. Entsprechend wird nach dem dritten gleichen Bit ein invertiertes Bit eingefügt.

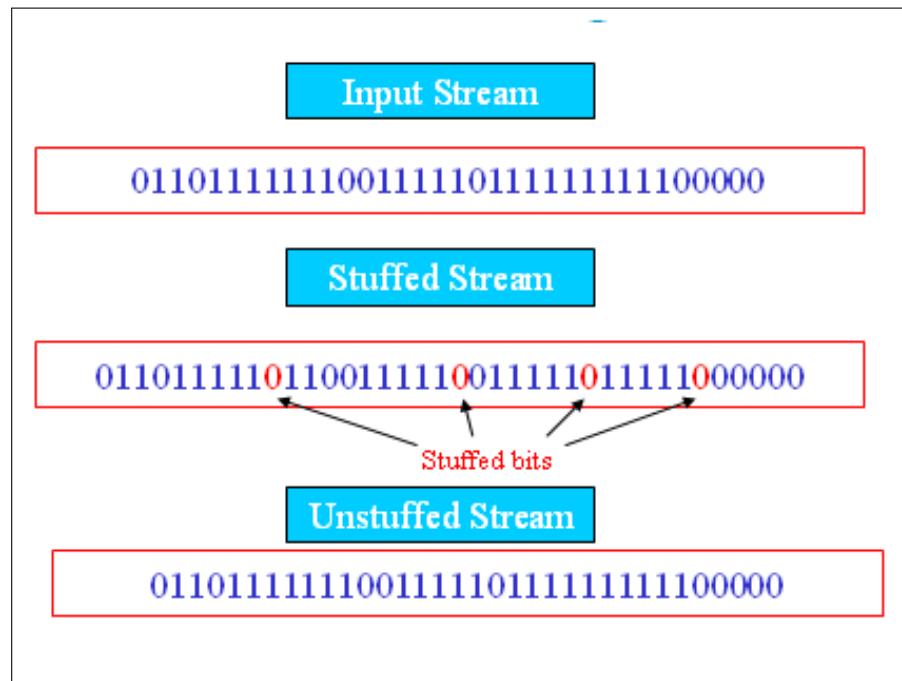


Abbildung 2.15: Darstellung von Bitstuffing an einem Beispiel. In diesem Beispiel handelt es sich um 5-Bit Bitstuffing. [<https://prayasnotty.wordpress.com/wp-content/uploads/2010/11/bit12.png>, 17.01.2025]

2.4 Programme

Für alle Module wurde ein gemeinsamer Programmcode geschrieben, bei dem mithilfe von einem #define ausgewählt wird, welche Funktionen kompiliert und auf den jeweiligen Mikrocontroller geflasht werden. Die Wichtigsten dieser Funktionen werden in den Abschnitten 2.4.1 bis 2.4.3 näher erläutert. Diese Herangehensweise wurde gewählt, da bei der Programmierung größerer Mengen von Mikrocontrollern die Wahrscheinlichkeit hoch ist, falsche Software zu flashen, wenn es mehrere sehr ähnliche Programme gibt. Zusätzlich ist die Entwicklung neuer Funktionen deutlich einfacher, da eine für alle Module notwendige Änderung nicht in mehrere Quelldateien angepasst werden muss.

2.4.1 Modulübergreifende Funktionen

Die Funktionen zum Lesen und Senden von Nachrichten müssen auf allen Modulen implementiert werden.

Senden einer Nachricht

Um eine Nachricht auf den Bus zu schreiben, muss zunächst festgestellt werden, dass dieser gerade nicht beschrieben wird. Nur wenn dies der Fall ist, kann die Nachricht gesendet werden. Die Nachricht setzt sich aus den in Kapitel 2.11 beschriebenen Teilen zusammen. Diese einzelnen Komponenten werden in je einer eigenen Funktion implementiert. Das Bitstuffing des Nachrichteninhalts wird dann in der Funktion 'writeBitstuffedMessage' durchgeführt.

```
1 void writeBitstuffedMessage(int x) {
2     if (DEBUG) Serial.println("writeBitstuffedMessage");
3     std::string stuffedMessage;
4     int count = 0;
5     char lastBit = '\0';
6     for (int i = 0; i < strlen(global_decoded_message); i++) {
7         char currentBit = global_decoded_message[i];
8         stuffedMessage += currentBit;
9         if (currentBit == lastBit) {
10             count++;
11             // Wenn x gleiche Bits gefunden wurden
12             if (count == x) {
13                 // Bitwechsel einfuegen
14                 char stuffedBit = (currentBit == '1') ? '0' : '1';
```

```

15         stuffedMessage += stuffedBit;
16         count = 1; // Zaehler zuruecksetzen
17         switch (currentBit)
18         {
19             case '0': lastBit = '1';
20                 /* code */
21                 break;
22             case '1': lastBit = '0';
23                 /* code */
24                 break;
25             default:
26                 break;
27         }
28     }
29 } else {
30     //Zaehler zuruecksetzen und auf 1 setzen
31     count = 1;
32     lastBit = currentBit;
33 }
34 }

35 // Update the global_BITSTUFFED_message with the stuffed message
36 delete[] global_BITSTUFFED_message;
37 global_BITSTUFFED_message = new char[stuffedMessage.length() + 1];
38 strcpy(global_BITSTUFFED_message, stuffedMessage.c_str());
39 if (DEBUG3) Serial.println("Decoded Message:");
40 if (DEBUG3) Serial.println(global_decoded_message);
41 if (DEBUG3) Serial.println("Bitstuffed Message:");
42 if (DEBUG3) Serial.println(global_BITSTUFFED_message);
43 }
```

Listing 2.3: Bitstuffing

Schlussendlich werden diese Nachrichtenbestandteile dann in folgender Funktion auf den Bus geschrieben.

```

1 void sendMessage(char adress, char dataSize, char* data){
2     if (DEBUG) Serial.println("sendMessage");
3     writeAddress(adress);
4     writeCOF(dataSize);
5     writeData(dataSize,data);
```

```

6   writeBitstuffedMessage(BITSTUFFING);
7   //Abfrage Bus frei
8   if (awaitBusFree()) {
9
10  if (DEBUG3) Serial.println("Sending!");
11  sendSOF();
12  sendDataOnBus(global_BITSTUFFED_message);
13  sendEOF();
14  global_decoded_message = NULL;
15 }
16 }
```

Listing 2.4: Senden einer Nachricht

Empfangen einer Nachricht

Solange ein Modul keine Nachricht senden möchte, hört es auf die Kommunikation auf dem Bus. Bei jedem empfangenen Start of Frame wird die Busfrequenz berechnet und gespeichert. Dann wird solange eingelesen, bis das EOF empfangen wurde.

```

1 bool readMessage() {
2   //if (DEBUG) Serial.println("readMessage");
3   // neue Nachricht auf dem Bus
4   if (syncronisation()) {
5     // Initialisieren Sie die globale bitgestopfte Nachricht
6     global_BITSTUFFED_message_recv = new char[1];
7     global_BITSTUFFED_message_recv[0] = '\0';
8     // Lesen Sie die Nachricht bitweise ein
9     int eofCounter = 0;
10    while (true) {
11      char bit = (digitalRead(COMM_IN) == HIGH) ? '1' : '0';
12      appendChar(global_BITSTUFFED_message_recv, bit);
13      DELAY(delayTime);
14      // ueberpruefen Sie auf EOF (Ende der Nachricht)
15      if (bit == '1') {
16        eofCounter = 1; // Start des EOF-Musters erkannt
17      } else if (eofCounter > 0) {
18        eofCounter++;
19        if (eofCounter == 6) {
20          // EOF-Muster erkannt: 1 gefolgt von 5 Nullen
21        }
22      }
23    }
24  }
25 }
```

```

21 // Entfernen Sie das EOF-Muster aus der Nachricht
22 global_BITSTUFFED_message_recv[strlen(
23     global_BITSTUFFED_message_recv) - 6] = '\0';
24 break;
25 }
26 } else {
27     // Zaehler zuruecksetzen, wenn das Muster unterbrochen wird
28     eofCounter = 0;
29 }
30 }
31 if (DEBUG3) Serial.println("Empfangene Nachricht:");
32 if (DEBUG3) Serial.println((global_BITSTUFFED_message_recv));
33 // Dekodieren Sie die bitgestopfte Nachricht
34 char* decodedMessage = NULL;
35 decodeBitstuffedMessage(global_BITSTUFFED_message_recv,
36     decodedMessage, BITSTUFFING);
37 global_decoded_message_recv = decodedMessage;
38 if (DEBUG3) Serial.println("Empfangene decodierte Nachricht:");
39 if (DEBUG3) Serial.println(global_decoded_message_recv);

40 parseDecodedMessage(global_decoded_message_recv);

41
42
43 if (DEBUG3) Serial.println("Adresse:");
44 if (DEBUG3) Serial.println(global_adress,HEX);
45 if (DEBUG3) Serial.println("COF:");
46 if (DEBUG3) Serial.println(global_COF,HEX);
47 if (DEBUG3) Serial.println("Daten:");
48 if (DEBUG3) for (int i = 0; i <= global_COF; i++) {
49     Serial.println(global_message[i],HEX);
50 }
51 delete[] decodedMessage; // Speicher freigeben
52 return true;
53 } else {
54     return false;
55 }
56 }
```

Listing 2.5: Empfangen einer Nachricht

Im Anschluss wird die empfangene Bitgestopfte Nachricht in der Funktion 'decodeBitstuffedMessage' in die Originalnachricht zurückgewandelt.

```

1 void decodeBitstuffedMessage(const char* stuffedMessage,
2                             char* &decodedMessage, int x) {
3     int count;
4     char lastBit = '\0';
5     size_t len = strlen(stuffedMessage);
6     decodedMessage = new char[len + 1]; // +1 fuer den Nullterminator
7     size_t decodedIndex = 0;
8     count = 1; // Zaehler auf 1 setzen
9
10    // Durchlaufen der bitgestopften Nachricht
11    for (size_t i = 0; i < len; ++i) {
12        char currentBit = stuffedMessage[i];
13        if (currentBit == lastBit) {
14            count++;
15            if (count < x) {
16                decodedMessage[decodedIndex++] = currentBit;
17            } else { // Wenn x gleiche Bits gefunden wurden
18                // Fuege das Bit hinzu
19                decodedMessage[decodedIndex++] = currentBit;
20                i++; // ueberspringe das naechste Bit
21                count = 1; // Zaehler zuruecksetzen
22                if (i >= len) break; // Vermeiden von ueberlaeufen
23                currentBit = stuffedMessage[i];
24            }
25        } else {
26            decodedMessage[decodedIndex++] = currentBit;
27            count = 1; // Zaehler zuruecksetzen und auf 1 setzen
28        }
29
30        lastBit = currentBit;
31
32    }
33
34    decodedMessage[decodedIndex] = '\0'; // Nullterminator hinzufuegen
35    if (DEBUG3) Serial.println("Empfangene decodierte Nachricht:");
36    if (DEBUG3) Serial.println(decodedMessage);

```

37 }

Listing 2.6: decodeBitstuffedMessage()

Zuletzt werden dann die einzelnen Teile der Nachrichten mit 'parseDecodedMessage' extrahiert.

```

1 void parseDecodedMessage(const char* decodedMessage) {
2     // Adresse extrahieren
3     global_adress = 0;
4     for (int i = 0; i < global_adressSize; i++) {
5         if (decodedMessage[i] == '1') {
6             global_adress |= (1 << i);
7         }
8     }
9
10    // COF extrahieren
11    global_COF = 0;
12    for (int i = 0; i < global_COFSIZE; i++) {
13        if (decodedMessage[global_adressSize + i] == '1') {
14            global_COF |= (1 << i);
15        }
16    }
17
18    // Daten extrahieren basierend auf der Groesse des COF
19    // COF gibt die Groesse der Nachricht in Bytes an (0 bedeutet 1 Byte)
20    int dataSize = global_COF + 1;
21    for (int i = 0; i < dataSize; i++) {
22        char message[8] = {0};
23        for (int j = 0; j < 8; j++) {
24            if (decodedMessage[global_adressSize +
25                global_COFSIZE + i * 8 + j] == '1') {
26                message[i] |= (1 << j);
27            }
28        }
29        global_message[dataSize-1-i] = message[i];
30        //global_message[i] = message[i];
31    }
32
33
34
35    // Ausgabe der extrahierten Bestandteile

```

```
36  if (DEBUG3) Serial.print("Empfangene Adresse: ");
37  if (DEBUG3) Serial.println(global_adress, HEX);
38  if (DEBUG3) Serial.print("Empfangener COF: ");
39  if (DEBUG3) Serial.println(global_COF, HEX);
40  if (DEBUG3) Serial.print("Empfangene Daten: ");
41  if (DEBUG3) for (int i = 0; i < dataSize; i++) {
42      Serial.print(global_message[i], HEX);
43      Serial.print(" ");
44  }
45  if (DEBUG3) Serial.println();
46 }
```

Listing 2.7: parseDecodedMessage()

Nur wenn die hier ermittelte Adresse mit der eigenen Adresse übereinstimmt werden die empfangenen Daten weiter verarbeitet, ansonsten wird die Nachricht ignoriert.

2.4.2 Controller Modul - Programmcode

Das Controller Modul übernimmt die folgenden Aufgaben:

1. Verwaltung der Adressvergabe an angeschlossene Module
2. Abfrage nach neuen Daten der einzelnen Module
3. Übermittlung dieser Daten über eine UART-Schnittstelle an den angeschlossenen Rechner

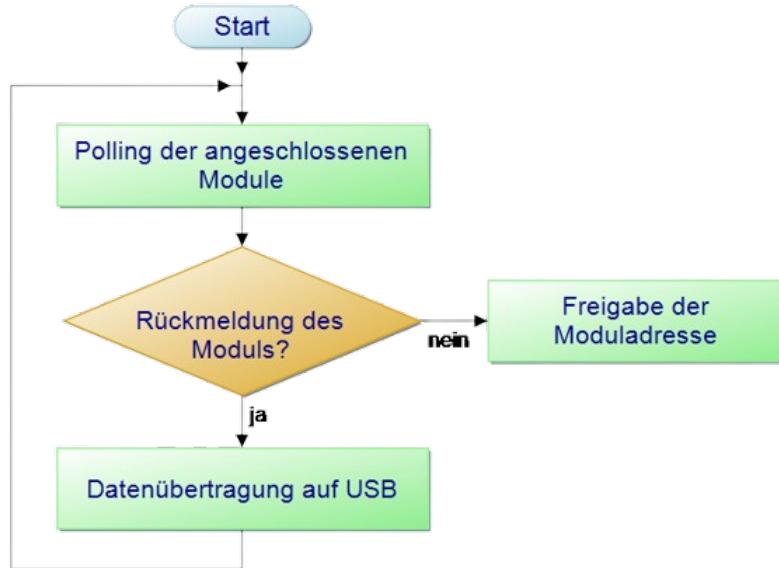


Abbildung 2.16: Programmablauf Controller Modul

Adressvergabe und Verwaltung

Das Controller Modul speichert alle vergebenen Adressen zusammen mit den zugehörigen Funktionen sowie weiteren Variablen der jeweiligen Module in einer Liste von Structs. Diese Liste wird für das Polling der Teilnehmer verwendet, wodurch sichergestellt wird, dass ausschließlich die tatsächlich angeschlossenen Module abgefragt werden.

```

1 struct device
2 {
3     char adress;
4     char funktion;
5     volatile char* latest_data;
6     unsigned int timeout;
7
8     device() : adress(0x00), funktion(0x00),
9                 latest_data(NULL), timeout(0){}
10
11 public:
  
```

```

12     device(char Adresse, char Funktion, char* Daten,
13         unsigned int Zaehler)
14     : adress(Adresse), funktion(Funktion), latest_data(Daten),
15       timeout(Zaehler){}
16 }
```

Listing 2.8: Struct device

Die jeweiligen Module haben keine festgelegte Adresse. Wenn diese also mit Spannung versorgt werden, warten sie auf die Adressvergabe des Controller Moduls. In der Polling Liste wird eine bestimmte Adresse (0x00) abgefragt, um festzustellen, ob ein neuer Teilnehmer an den Bus angeschlossen wurde. Wenn eine Rückmeldung auf diese Adresse festgestellt wird, dann enthält die nächste Nachricht die Adresse für den neuen Teilnehmer. In der Rückmeldung eines Moduls ist dessen Funktion hinterlegt. Diese Informationen werden in der Adressliste abgespeichert und das Gerät ab dem Zeitpunkt mit abgefragt.

```

1 char newAdress(char function){
2     if (DEBUG4) Serial.println("newAdress");
3 //es sind noch Adressen frei
4 if (unusedAdresses.empty() == false)
5 {
6     //nehme eine freie Adresse
7     char freeAdress = unusedAdresses.back();
8     if (DEBUG) Serial.println(freeAdress,HEX);
9     //entferne diese aus 'unused Adresses'
10    unusedAdresses.pop_back();
11    //fuege Sie zur Pollingliste hinzu
12    Adresses.push_back(freeAdress);
13    //neuen Teilnehmer der device Liste hinzufuegen
14    Serial.print("device function: ");
15    Serial.println(function,HEX);
16    device teilnehmer(freeAdress,function,nullptr,0);
17    Serial.print("device adress: ");
18    Serial.println(teilnehmer.adress,HEX);
19    Serial.print("device funktion: ");
20    Serial.println(teilnehmer.funktion,HEX);
21    devices.push_back(teilnehmer);
22    //Pollingschleife vergroessern
23    num_adresses++;
```

2 Modulares Eingabesystem

```
24 //Uebermittlung der Geraeteliste, wenn ein Teilnehmer dazukommt
25 FLAG_NEWDEVICE = true;
26 if (DEBUG4) Serial.println("Neue Adresse:");
27 if (DEBUG4) Serial.println(freeAdress,HEX);
28 return freeAdress;
29 }
30 // keine Adressen mehr frei!
31 else return 0x00;
32 }
```

Listing 2.9: Zuweisen einer neuen Adresse

Wenn ein Modul nicht mehr über den Bus kommuniziert, wird dessen Adresse nach mehreren fehlgeschlagenen Versuchen (Timeout) aus der Polling-Liste entfernt und wieder den verfügbaren Adressen zugewiesen. Dadurch wird sichergestellt, dass alle aktiven Geräte schnellstmöglich vom Controller Modul abgefragt werden und Adressen für neu verbundene Geräte freigegeben werden. Die Anzahl der nicht Rückmeldungen wurde auf drei festgelegt, kann aber in den 'defines' des Controller Moduls angepasst werden.

```
1 else if (no_response && adress != NOADDRESS){
2     // inkrementieren des Timeout-counters
3     iterator_devices->timeout++;
4     if (DEBUG2) Serial.println("keine Antwort von:");
5     if (DEBUG2) Serial.println(iterator_devices->adress,HEX);
6     if (iterator_devices->timeout >= TIMEOUT_THRESHOLD) {
7         //Adresse wieder freigeben
8         unusedAdresses.push_back(iterator_devices->adress);
9         //Adresse aus der Pollingliste entfernen
10        Adresses.remove(iterator_devices->adress);
11        //Adresse aus der device Liste entfernen
12        devices.erase(iterator_devices);
13        //Pollingschleife verkleinern
14        num_adresses--;
15        //Uebermittlung der Geraeteliste, wenn ein Teilnehmer wegfaellt
16        FLAG_NEWDEVICE = true;
17        //keine Iteration in der Pollingschleife
18        FLAG_DELETEDEVICE = true;
19    }
20 }
```

Listing 2.10: Timeout Funktion

Wenn keine Antwort eines Eingabemoduls in der vorgegebenen Wartezeit empfangen wird, wird der Timeout-Counter iteriert. Sobald dieser seinen Maximalwert erreicht, wird das Gerät nicht mehr abgefragt und dessen Adresse freigegeben.

Abfrage neuer Daten

Die Daten jedes Moduls werden einmal pro Durchlauf der Polling Liste abgefragt. Sobald neue Daten empfangen wurden, werden diese abgespeichert und dem angeschlossenen Computer über die UART-Schnittstelle übermittelt.

```
1 void polling(){
2     if (DEBUG) Serial.println("polling");
3     //naechstes element der adressliste
4     if (!FLAG_DELETEDDEVICE)
5     {
6         polling_counter++;
7     }
8     // ein Teilnehmer ist weggefallen -> nicht iterieren
9     else
10    {
11        FLAG_DELETEDDEVICE = false;
12    }
13    // ueberpruefen Sie, ob der polling_counter die
14    // Anzahl der Adressen erreicht hat
15    if (polling_counter >= num_adresses + 1) {
16        polling_counter = 0; // Zuruecksetzen des polling_counter
17    }
18    //Setzen des Iterators an die entsprechende Stelle
19    iterator_devices = devices.begin();
20    std::advance(iterator_devices,polling_counter);
21    bool response = false;
22    char request[] = {0xFF};
23    global_reset();
24
25    //request fuer abzufragende Adresse
```

```

26 sendMessage(iterator_devices->adress, 0, request);
27 //warten auf Rueckmeldung
28 response = await_response(controllerAdress);
29 //Ruecksetzen des Timeouts (zudem Zuweisung von neuen Adressen)
30 timeout(iterator_devices->adress, !response);
31 //Uebermitteln der Daten an den PC
32 USB(iterator_devices->adress, iterator_devices->
33     funktion, iterator_devices->latest_data);
34 }
```

Listing 2.11: Polling Funktion

Übermittlung der Daten an den angeschlossenen Computer

Sobald neue Daten von einem Modul empfangen worden sind, werden diese an den Computer übermittelt. Zu den übermittelten Daten wird neben der jeweiligen Funktion auch die zugehörige Adresse übermittelt, sodass die Daten nicht nur abhängig von der Funktion ausgewertet werden können, sondern auch mehrere gleiche Module voneinander unterschieden werden können. Damit alle Busteilnehmer in das Input-Mapper Programm eingebunden und konfiguriert werden können, wird alle 10 Polling Durchläufe eine Liste aller Module einschließlich ihrer Funktion übermittelt. Diese Liste wird auch gesendet, sobald sich die Anzahl der Busteilnehmer ändert.

Zudem wird zur Ermittlung des COM-Ports an dem das Controllermodul an den PC angeschlossen alle 10 Polling Durchläufe ist ein festgelegter String („INIT“) übermittelt.

```

1 // uebermitteln der Daten des jeweiligen Slaves
2 Serial2.println("START");
3 if (DEBUG5) Serial.println("START");
4 Serial2.println(adress,HEX);
5 if (DEBUG5) Serial.println(adress,HEX);
6 for (int i = 0; i <= global_COF; i++)
7 {
8     Serial2.println(data[i],HEX);
9     if (DEBUG5) Serial.println(data[i],HEX);
10 }
11 Serial2.println("END");
12 if (DEBUG5) Serial.println("END");
```

Listing 2.12: Datenübertragung an den angeschlossenen Computer

2.4.3 Code der Eingabemodule

Im Folgendem wird der Programm-Ablauf eines Eingabemoduls genauer beschrieben. Dieses Modul hat folgende Aufgaben:

1. Anforderung einer Adresse bei Neustart
2. Auswerten der Modulfunktion
3. Übermitteln der Daten an das Controller Modul bei Anfrage

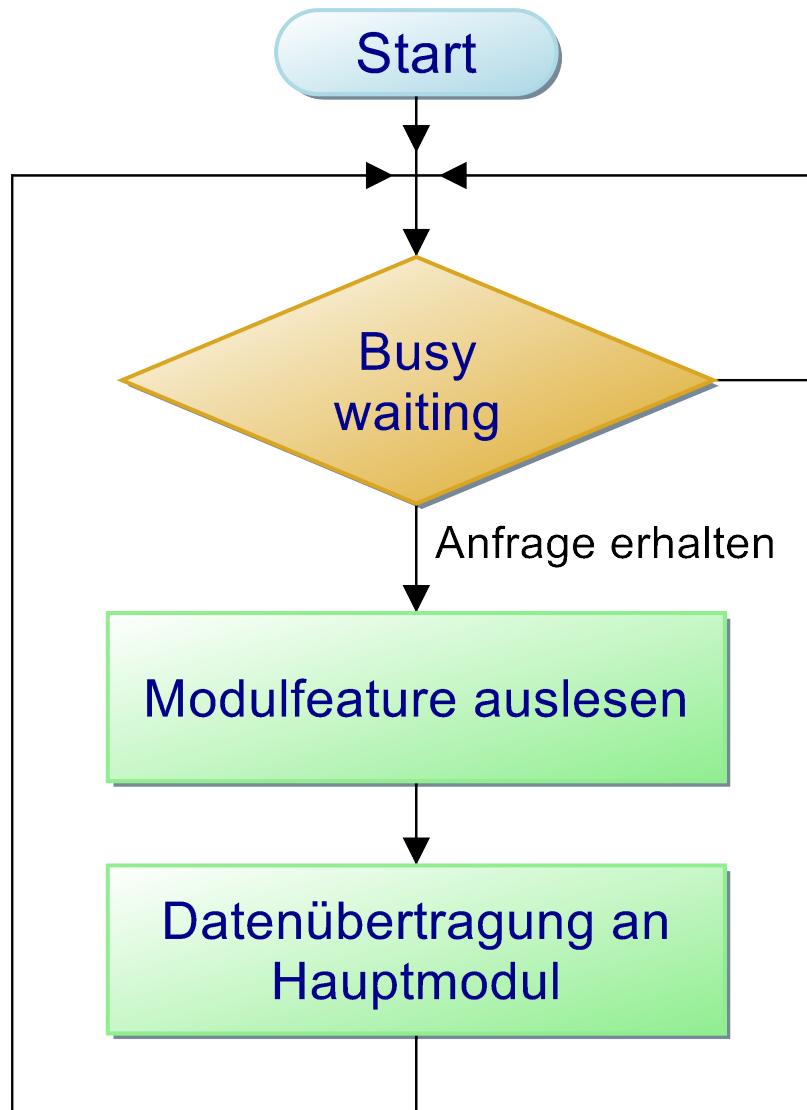


Abbildung 2.17: Programmablauf eines Eingabemoduls

Anforderung einer Adresse

Wenn einem Modul noch keine Adresse zugewiesen wurde und der Controller nach neuen Teilnehmern auf dem Bus sucht, antwortet das Modul mit einer Nachricht, in der seine Funktion in den übertragenen Daten enthalten ist. Anschließend übermittelt der Hauptcontroller eine neue Adresse an das Modul. Diese Adresse wird vom Modul gespeichert, sodass es danach

ausschließlich auf diese Adresse reagiert.

```
1 void getAddress(){
2     //Bitte um Adresse + 2. byte = Funktion
3     char message[] = {REQUESTADDRESS, FUNCTION};
4     sendMessage(controllerAdress, 0x01, message);
5     Serial.println("Adresse angefragt!");
6     //Antwort erhalten
7     if (await_response(NOADDRESS)){
8         //Speichern der neuen Adresse
9         myAdress = global_message[0];
10        if (DEBUG3) Serial.print("meine Adresse: ");
11        if (DEBUG3) Serial.print(myAdress, HEX);
12        if (DEBUG3) Serial.println();
13    }
14 }
```

Listing 2.13: Anfordern einer Adresse

Auswertung der Modulfunktion

Die Auswertung der Modulfunktion ist abhängig von der jeweiligen Funktion des Moduls und kann daher nicht verallgemeinert werden.

Im Fall des Tastatur-Moduls werden die Tasten nur dann ausgewertet, wenn der Controller das Modul abfragt. Anschließend werden die Daten übermittelt. Für diese Auswertung wurde eine allgemeine Funktion geschrieben, die jedoch für jedes Modul spezifisch angepasst werden muss.

```
1 void myFunction(){
2     Serial.println("myFunction!");
3     char myDatasize = 0;
4     char myData[8] = {0};
5     //-----
6
7     //-----
8
9     //-----
10    for (int row = 0; row < 2; row++) {
11        digitalWrite(ROWS[row], HIGH);
```

```
12  for (int col = 0; col < 2; col++) {  
13      if (digitalRead(COLS[col]) == HIGH) {  
14          //bei OUTPUT ROWS[row] ist COLS[col] HIGH  
15          myData[myDatasize] = tasten2x2[row][col];  
16          myDatasize++;  
17      }  
18  }  
19  digitalWrite(ROWS[row], LOW);  
20  DELAY(5);  
21}  
22  
23 //-----  
24 // Uebermitteln der Daten  
25 if (myDatasize == 0){  
26     sendMessage(controllerAdress,myDatasize,myData);  
27 }else{  
28     sendMessage(controllerAdress,myDatasize-1,myData);  
29 }  
30}
```

Listing 2.14: Beispiel Funktion Tastaturmodul

Übermittlung der Daten an das Controller-Modul

Sobald eine Anfrage des Controller Moduls an die Adresse des Eingabemoduls empfangen wird, übermittelt dieses die Information der Modulfunktion. Im Falle der Tastatur wären dies die zurzeit gedrückten Tasten. Dabei nimmt jede gedrückte Taste ein Byte der Daten ein.

2.5 Input-Mapper Programm

Das Input-Mapper Programm dient mit einer grafischen Oberfläche zu Steuerung, Konfiguration und Kontrolle der angeschlossenen Module. Es ist der Hauptbestandteil der Kommunikation zwischen dem Hauptmodul und dem verbundenen Computer. Das Programm wurde entwickelt, um den Tasten des Keypads und den Funktionen zukünftiger Module bestimmte Funktionen zuzuweisen, wie z.B. das Ausführen von Tastenkombinationen auf dem Computer. Die derzeitige Hauptfunktionalität besteht darin, das Keypad mit dem 4x4 Tastenraster zu individualisieren. Das Programm ist in Python geschrieben und verwendet die grafische Benutzeroberfläche der Bibliothek „customtkinter“.

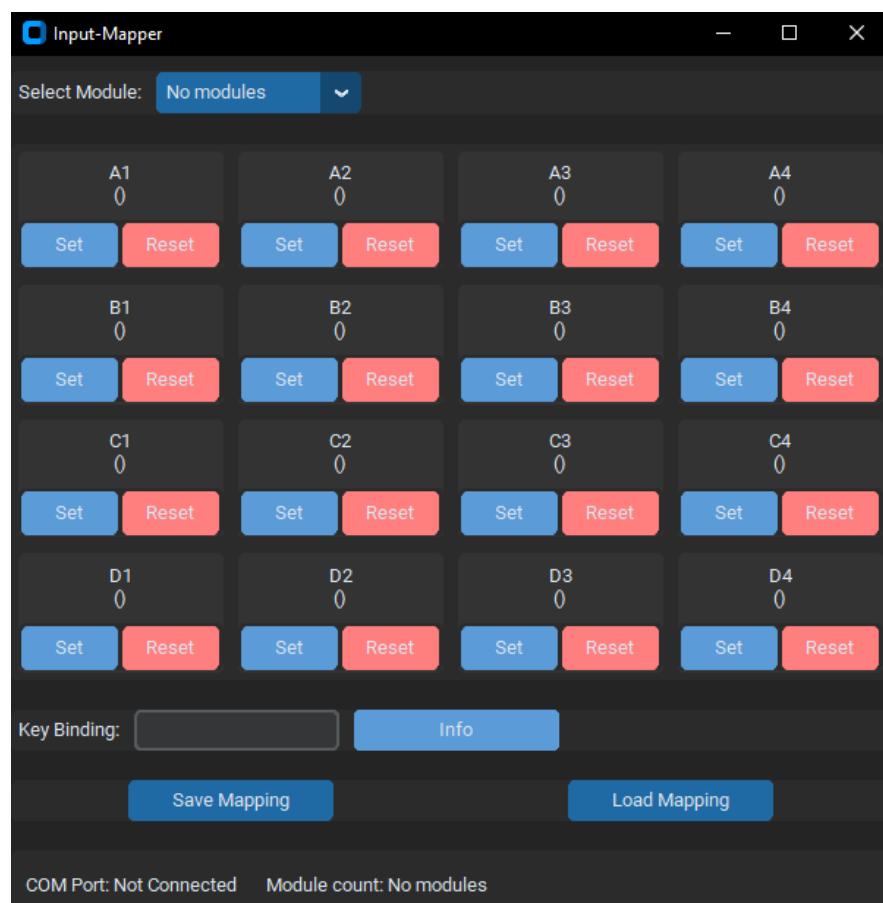


Abbildung 2.18: Benutzeroberfläche des Inputmapper-Programms

Abbildung 2.18 zeigt die grafische Benutzeroberfläche des Input-Mapper Programms. Es bietet eine Auswahl der verbundenen Module und die Möglichkeit, die Tasten dieses Moduls nach Belieben zu individualisieren. In das Eingabefenster „Key Binding:“ wird die gewünschte Tastenkombination eingetragen und danach bei der einer Taste auf „Set“ gedrückt. Ab diesem Zeitpunkt wird diese Taste beim Drücken, die hinterlegte Kombination ausführen oder Zeichenkette ausgeben. Das Infofenster bietet eine beispielhafte Übersicht einiger Tastenbelegungen. „Save Mapping“ und „Load Mapping“ werden benutzt, um Konfigurationen zu speichern oder zu laden. Zuletzt wird dem Benutzer eine Übersicht über den verwendeten COM-Port und die Anzahl aller verbundenen Module geboten.

2.5.1 Hauptfunktionen

1. **Hardware-Erkennung:** Das Programm durchsucht alle verfügbaren COM-Ports des angeschlossenen Computers und vergleicht deren Inputs mit der Initialisierungsnachricht. An dem Port, an dem eine Nachricht „INIT“ anliegt, wird als Kontroller-Port für das modulare Eingabesystem erkannt.
2. **Serielle-Kommunikation:** Über den COM-Port werden die Daten des Kontrollers (Modulliste oder gedrückte Taste des Moduls X) seriell übertragen und vom Programm ausgelesen. In der Funktion „read_from_com_port()“ werden kontinuierlich die verschiedenen möglichen Nachrichten geprüft und je nach Muster und Inhalt des Datenstroms verschiedene Funktionen im Programm aufgerufen.
3. **Input-Mapping:** Der Benutzer bekommt die Möglichkeit, den Tasten des 4x4 Keypads individuelle Tastenkombinationen zuzuweisen. Dabei werden zusätzlich Normalisierungen vorgenommen, wie z.B. die Eingabe von „ctrl“ in „Control“ für eine bessere Lesbarkeit und Eingabefehler Minimierung.
4. **Speichern und Laden:** Die Tastenzuordnungen können in einer JSON-Datei gespeichert und später wieder geladen werden. Diese Funktionen ermöglichen es dem Benutzer, bereits erstellte Konfigurationen auch zukünftig zu laden und zu verändern.
5. **Modulverwaltung:** Das Programm erkennt automatisch verbundene Module und entfernt diese auch, sobald die Verbindung getrennt wurde. Dementsprechend wird das Dropdown-Menü zur Modulauswahl dynamisch angepasst.

6. GUI:

- **Tastendarstellung:** Bei dem Keypad Modul wird jede Taste des 4x4 Rasters mit der aktuellen Belegung dargestellt.
- **Steuerelemente:** Es sind Knöpfe zum Setzen und Zurücksetzen von Tastenbelegungen, sowie zum Speichern und Laden von Konfigurationen vorhanden.
- **Kontrollelemente:** Das Programm bietet eine Übersicht der Anzahl an angeschlossenen Modulen und den COM-Port, über den die Nachrichten vom Kontroller-Modul übertragen werden.
- **Informationsfenster:** Der Benutzer kann ein Informationsfenster öffnen, in dem eine Vielzahl von Beispielbelegungen vorgegeben sind.

```
key_mappings.json > ...
1  {
2    "Keypad (Address 2)": {"A1": "Alt+Tab", "A2": "", "A3": "", "A4": "", "B1": "", "B2": "", "B3": ""
3    "Keypad (Address 5)": {"A1": "Windows+Shift+S", "A2": "", "A3": "", "A4": "", "B1": "", "B2": "", "
4 }
```

Abbildung 2.19: JSON-Datei zum Speichern und Laden von Konfigurationen

Abbildung 2.19 zeigt das JSON Format, in welchem die Tastenbelegungen gespeichert werden. Wobei der Name des Moduls die ID oder auch „Schlüssel“ zum eindeutigen Identifizieren ist. Zu dieser ID gehört dann eine Liste von Tasten und deren Zuweisung. Wobei bei der Taste „A1“ das „A“ für die Reihe der Taste und die „1“ für die Zeile der Taste steht.

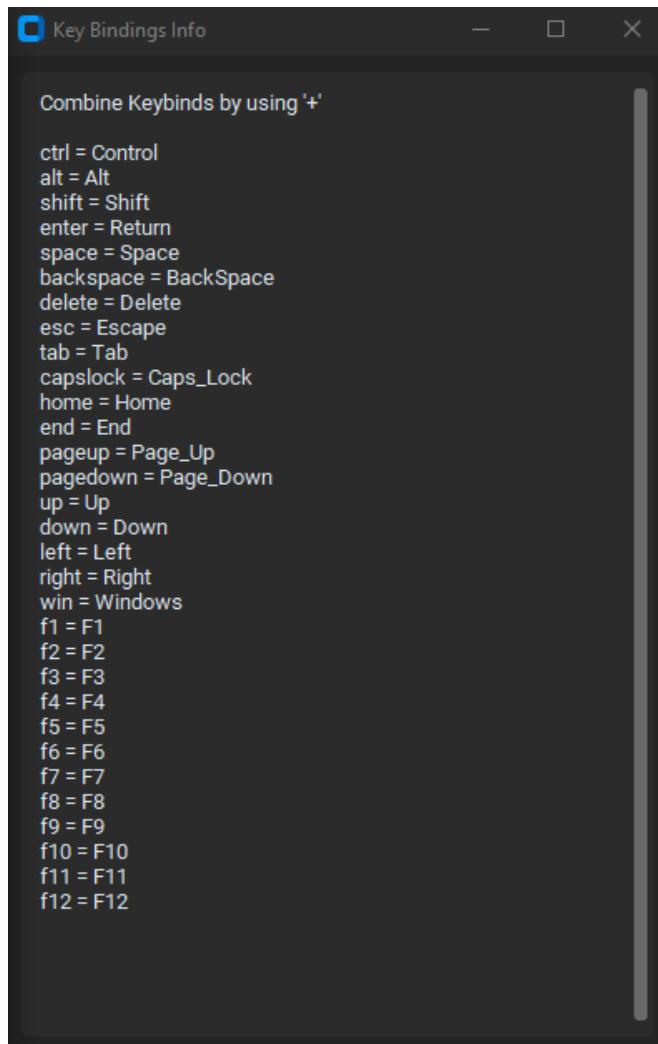


Abbildung 2.20: Infofenster mit beispielhaften Belegungen

Abbildung 2.20 zeigt das geöffnete Infofenster, in dem der Benutzer eine Vielzahl von beispielhaften Tastenkombinationen vorgegeben bekommt.

2.6 Schaltpläne

Zur Umsetzung des Datenbusses wird für jedes Modul ein Mikrocontroller zur Kommunikation benötigt. Über die Mikrocontroller werden die zu verarbeitenden Eingangssignale vom jeweiligen Modul über den Bus an das Controller Modul übertragen. Für die unterschiedlichen Funktionen der Module müssen jeweils eigene Platinen entwickelt werden.

2.6.1 Controller Modul

Der verwendete ESP32 ist wie in Abb. 2.21 verschaltet. Die UART-Bridge wird an den Pins D34 und D35 angeschlossen, da diese als UART Pins vom ESP32 vorgesehen sind. Der interne Datenbus wird an den Pins D13 und D15 angeschlossen, wobei D13 als Receive-Pin und D15 als Transmit-Pin fungiert. Der ESP32 wird über V_{in} mit 5V über die USB Spannung versorgt.

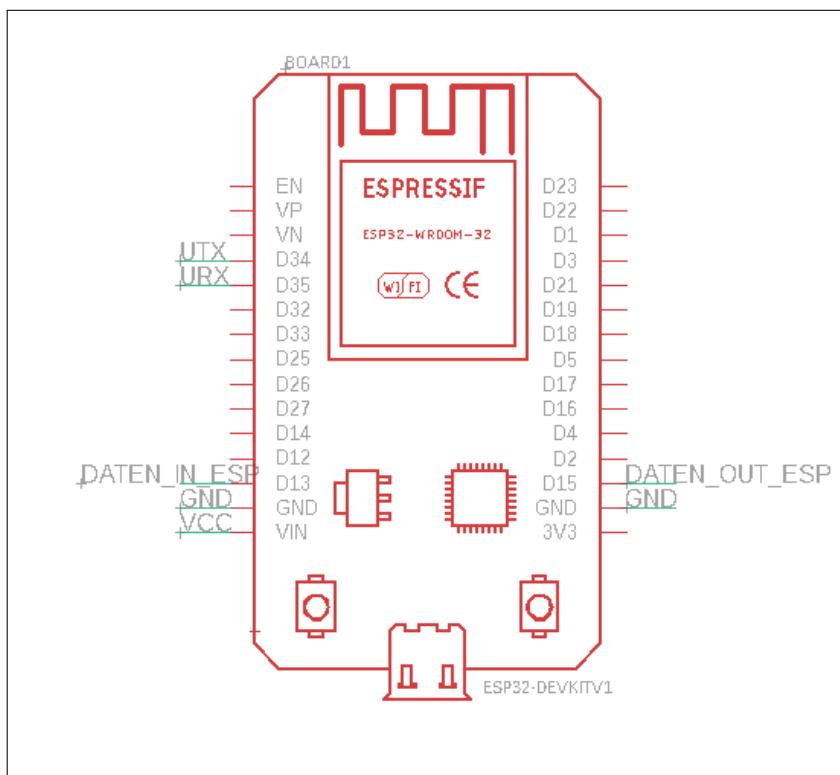


Abbildung 2.21: ESP32 Verschaltung

Für die Kommunikation mit einem PC, werden die Daten über eine UART-Bridge gegeben, bevor sie an die USB Schnittstelle übertragen werden. D+ und D- sind in diesem Fall

2 Modulares Eingabesystem

die USB-Datenleitungen, welche über eine USB-Buchse herausgeführt werden. Die UART-Datenleitungen vom ESP32 zur UART-Bridge sind in Abb. 2.22 verdreht, da es sich um die Labels aus Sicht des ESP32 handelt.

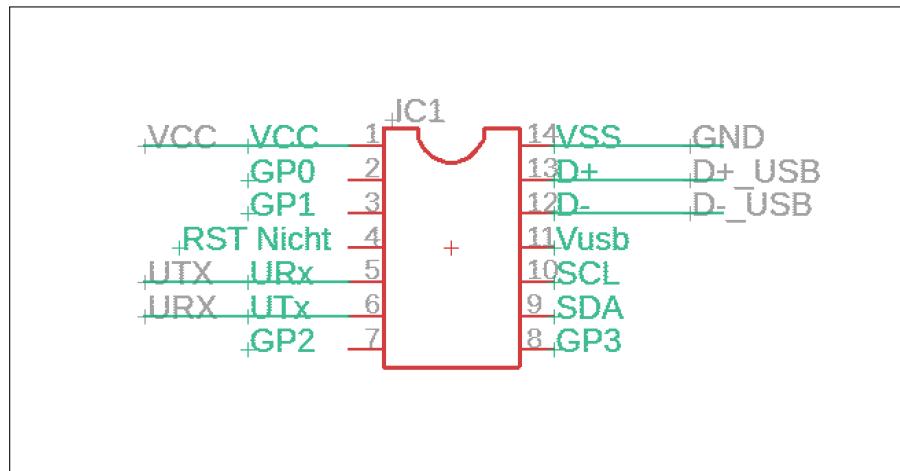


Abbildung 2.22: UART-Bridge

Zur Kommunikation auf unserem Datenbus werden die Daten differenziell übertragen. Welches über den Transmitter SN65LVS1D und den Receiver SN65LVDT2D passiert.

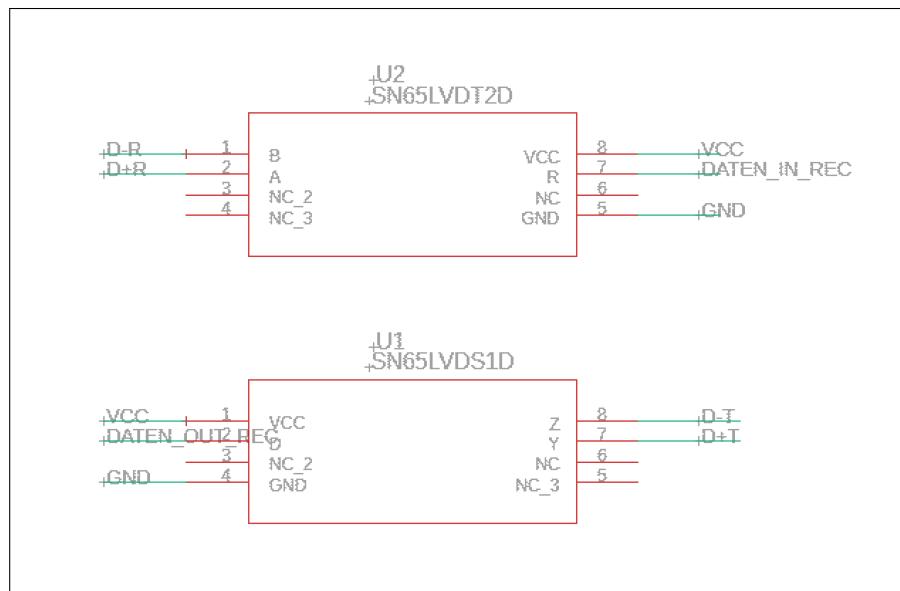


Abbildung 2.23: Transmit-(SN65LVS1D) und Receiver(SN65LVDT2D) Bausteine

2.6.2 Tastatur-Modul

Das Tastatur-Modul besteht aus 4x4 Tasten, die, wie in Abb. 2.24 zu sehen, in einer Matrix verschaltet sind. Über die vier Ausgänge (PD2, PD3, PD4, PD5) des ATmega werden nacheinander Ausgangssignale gegeben, während über die vier Eingänge (PD6, PD7, PB0, PB1) die Spalten abgefragt werden. Die Verschaltung des ATmega ist in Abb. 2.25 zu sehen. Um das korrekte Auslesen mehrerer gedrückten Tasten zu gewährleisten, werden in jedem Ausgangspfad Schaltdioden eingesetzt.

Die Datenübertragung findet über die Pins PD0 und PD1 statt.

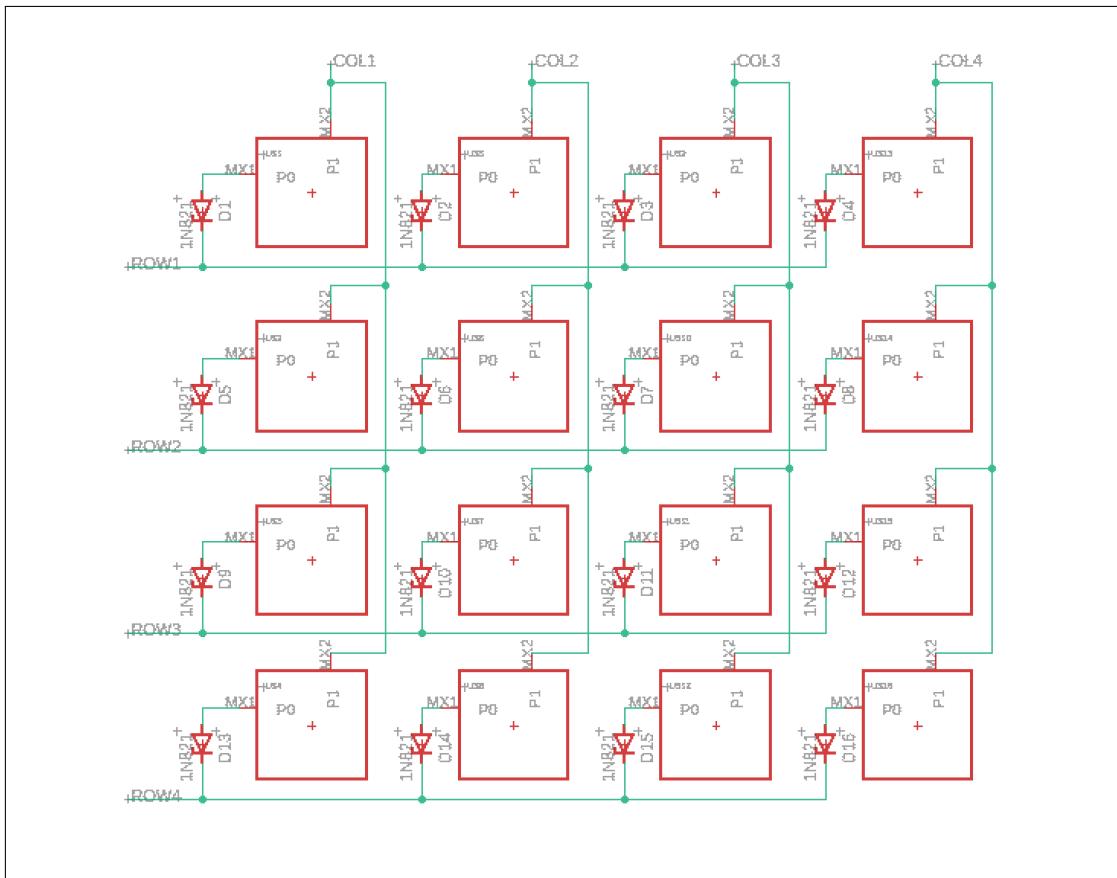


Abbildung 2.24: Tastatur Schaltung

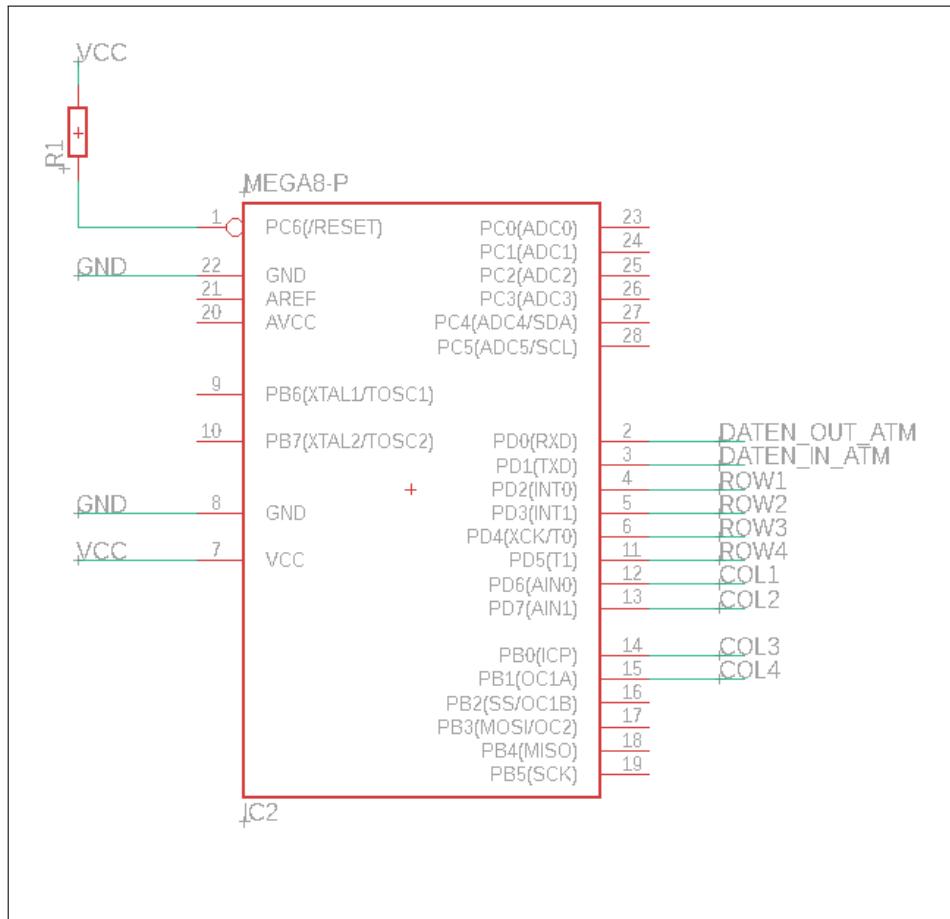


Abbildung 2.25: ATmega Verschaltung

2.6.3 Alternativen zum Transceiver

Aufgrund der Probleme bei der Realisierung wurde sich dazu entschieden auf dem PCB mehrere Möglichkeiten zur differenziellen Übertragung vorzusehen und über Jumper schalten zu können. Es wird die folgenden Möglichkeiten geben:

- nur die Transmit/Receive ICs zu verwenden,
- die Transmit/Receive ICs mit zusätzlichen Optokopplern zu verwenden,
- die Optokoppler mit diskret aufgebautem Invertierern und
- das Umschalten auf eine Lochrasterplatine, andere Testschaltungen/Testbausteine

2 Modulares Eingabesystem

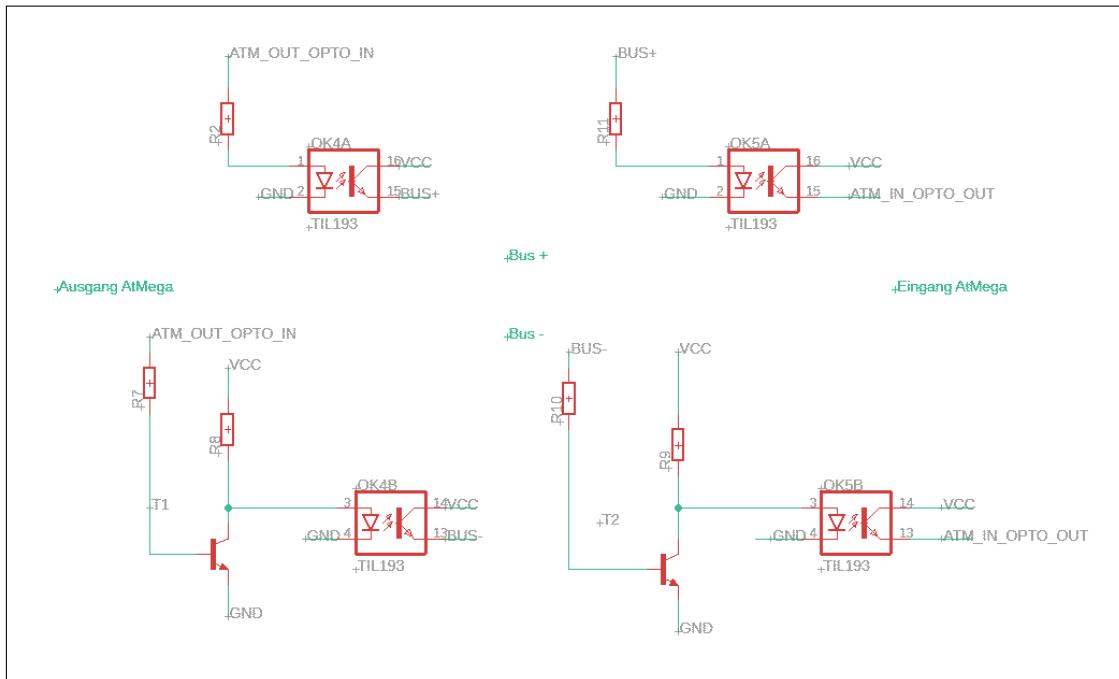


Abbildung 2.26: Optokoppler Schaltung

2.7 Platine

Beim Layout der Platine wurde auf die folgenden Punkte geachtet:

- Platzierung der Kontaktstecker, entsprechend der Modularen Verbindmöglichkeit
- Räumliche Nähe der zusammengehörenden Bauteile
- Optimierung der Masseführung durch Verwendung einer möglichst durchgehenden Massefläche
- Vermeidung von 90°-Ecken der Leiterbahnen, um Reflexionen zu minimieren
- Platzierung von Abblock- und Stützkondensator nahe den Versorgungspins der ICs.

Differenzielle Übertragung

Bei einem Layout für differenzielle Übertragung sollte zusätzlich auf die folgen Punkte geachtet werden:

- Geringer Abstand zwischen den differenziellen Leitungen
- Gleiche Länge der differenziellen Leitungen
- Abstand zu anderen Leiterbahnen entsprechend Abb. 2.29
- möglichst kurze Leitungen

Wie in Abb. 2.29 vom Hersteller Empfohlen müssen die Leiterbahnen, von oben betrachtet zueinander einen möglichst geringen Abstand haben und zu anderen Leiterbahnen mindestens das Doppelte der Breite einer Leiterbahn. Diese Einstellungen kann beispielsweise in Eagle anhand der Signalnamen als Regel vorgenommen werden und werden dann automatisch eingehalten.

2.7.1 Verbesserungen

Optimierung

Da die erste Platine hauptsächlich zum Testen der differenziellen Übertragungsbauteile dient, sind einige Anpassungen geplant.

Wenn die Inbetriebnahme wie erwartet gelingt und keine weiteren zu behebenden Fehler auftreten, soll vor allem Platz gespart werden, indem möglichst alle Bauteile durch SMD

ersetzt werden. Die Jumper und Optokoppler sollen entfernt werden und es soll der Fokus auf Impedanzkontrolle gelegt werden, damit die differenzielle Übertragung gut funktioniert. Durch diese Änderungen wird die Platine voraussichtlich kleiner und günstiger werden. Ein Ziel ist es das Tastatur-Modul auf die Maße des Controller-Moduls anzupassen, um die Modularität wie geplant möglichst ansprechend zu gestalten.

««« Updated upstream

Dafür soll von einer zwei Layer Platine auf vier Layer umgestiegen werden. Die Abstände der Signale zur Massefläche wie in Formel 2.1 zu sehen, einen Einfluss auf die Impedanz hat. Es gibt unterschiedliche Topologien zur Anordnung der Leiterbahnen auf den Layern der Platine.

=====

Kabelverbindung

Außerdem soll als weitere Möglichkeit eine Kabelverbindung zwischen den Modulen entwickelt werden, um noch höhere Flexibilität zu ermöglichen.

Anpassung der Layer

Es soll von einer zwei Layer Platine auf vier Layer umgestiegen werden. Die Abstände der Signale zur Massefläche hat einen Einfluss auf die Impedanz, wie in ?? zu sehen. Vom Hersteller wird ein vier Layer Stackup empfohlen, da sie die nötigen Spezifikationen für eine Impedanz von 100 Ohm ermöglichen. Es gibt unterschiedliche Topologien zur Anordnung der Leiterbahnen auf den Layern der Platine. »»» Stashed changes

Bei einer Stripline-Topologie wie in Abb. 2.28 liegen die Leiterbahnen zwischen zwei Masseflächen eingebettet, was eine bessere Abschirmung bedeutet, aber auch eine höhere Kapazität. Daher wird vom Hersteller bei hochfrequenten Siganlen eine Microstrip-Topologie, wie in Abb. 2.27 empfohlen.

««« Updated upstream

Außerdem wird die Breite und der Abstand der Leiterbahnen in der Software entsprechend der vom Hersteller angegebenen Werte für die Höhe und die Konstante des Dielektrikums, eingestellt. Bei dem von uns verwendeten Hersteller JLCPCB kann für die verschiedenen Dicken der Leiterplatte und entsprechend des verwendeten Dielektrikums die Dicke der einzelnen Layer eingesehen werden. In Formel 2.3 ist die Berechnung der Leiterbahnbreite mit

Werten, die im Datenblatt der SN65LVDXXX Bauteile angegeben werden, entsprechend des Terminationswiderstandes von 100 Ohm. =====

Anpassung der Leiterbahnbreite

Außerdem wird die Breite und der Abstand der Leiterbahnen in der Software entsprechend der vom Hersteller angegebenen Werte für die Höhe und die Konstante des Dielektrikums, eingestellt. Bei dem von uns verwendeten Hersteller JLCPCB kann für die verschiedenen Dicken der Leiterplatte und entsprechend des verwendeten Dielektrikums die Dicke der einzelnen Layer eingesehen werden. In ?? ist die Berechnung der Leiterbahnbreite mit Werten, die im Datenblatt der SN65LVDXXX Bauteile angegeben werden, entsprechend des Terminationswiderstandes von 100 Ohm. >>> Stashed changes

Eingesetzte Werte:

- Dielektrizitätskonstante $\varepsilon_r = 3.4$
- Abstand zwischen den differenziellen Leitungen $S = 0.15 \text{ mm}$
- Dicke des Dielektrikums $H = 0.18 \text{ mm}$
- Dicke der Leiterbahn $T = 0.035 \text{ mm}$
- Zielimpedanz $Z_0 = 100 \Omega$
- Leiterbahnbreite W

$$Z_{diff} = \frac{Z_0}{\sqrt{2}} \left(1 + \frac{0.5}{\sqrt{1 + 4 \cdot \frac{S}{W}}} \right) \quad (2.1)$$

Berechnung der Leiterbahnbreite W mit den eingesetzten Werten:

$$W = \frac{5.98 \cdot 0.18}{\exp \left(\frac{100 \cdot \sqrt{3.4+1.41}}{87} \right)} - \frac{0.035}{0.8} \quad (2.2)$$

$$W = 0.043 \text{ mm} \quad (2.3)$$

2 Modulares Eingabesystem

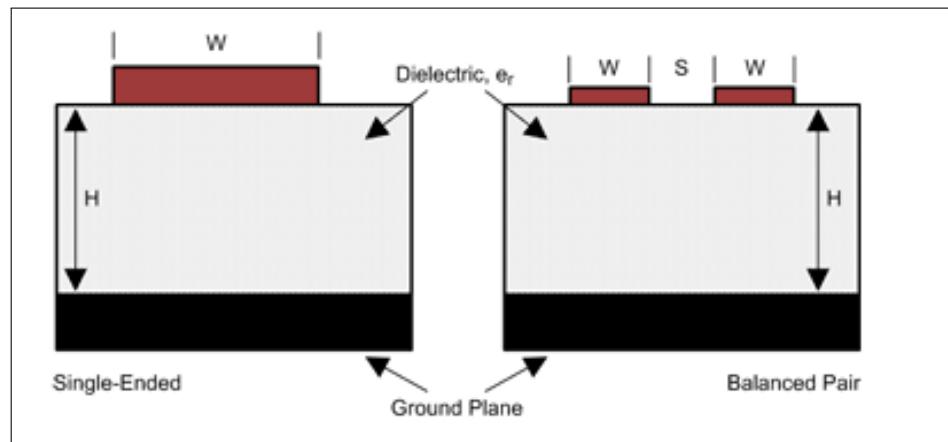


Abbildung 2.27: Mikrostrip-Topology

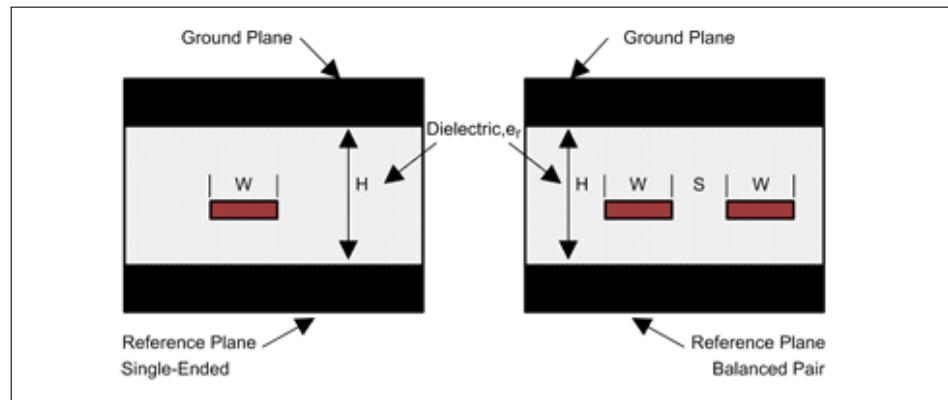


Abbildung 2.28: Stripline-Topology

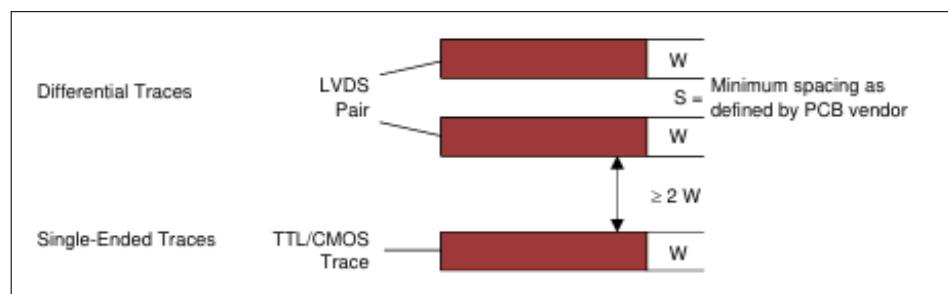


Abbildung 2.29: Abstand Leiterbahnen

2.7.2 Layout

In Abb. 2.30 wird das Layout des Controller Moduls dargestellt.

In Abb. 2.31 wird das Layout des Tastatur Moduls dargestellt.

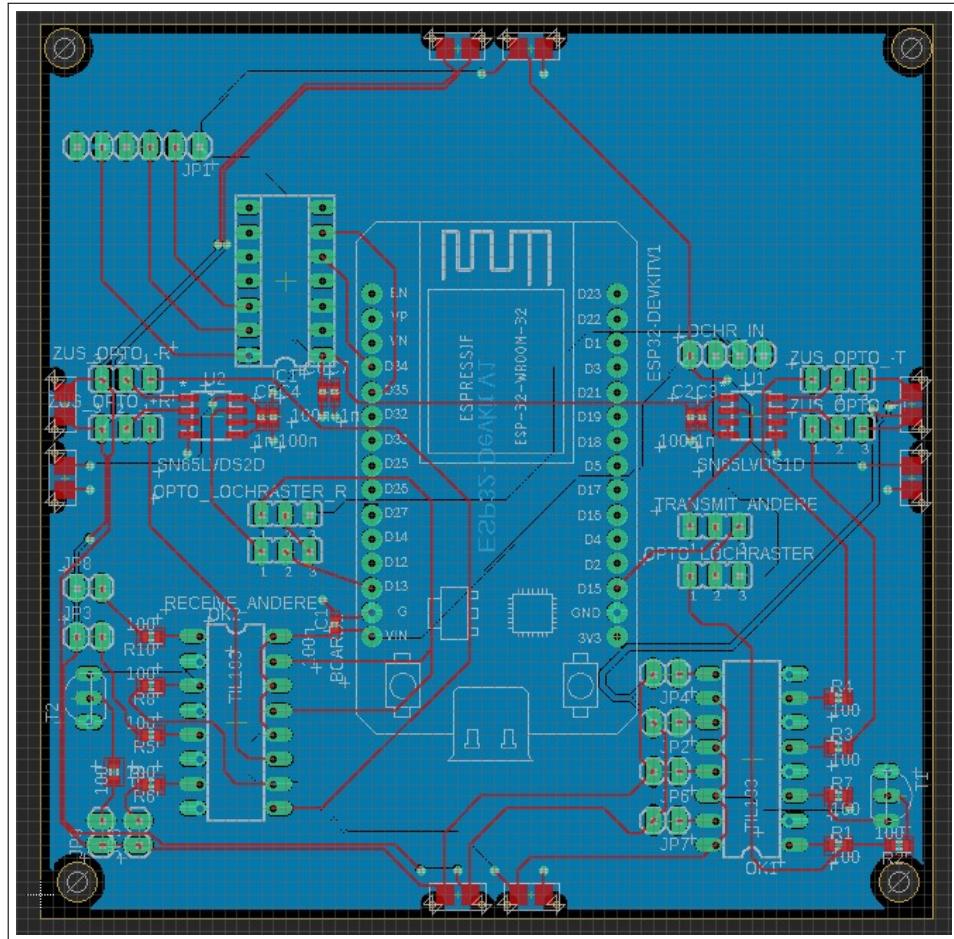


Abbildung 2.30: Layout Controller-Modul

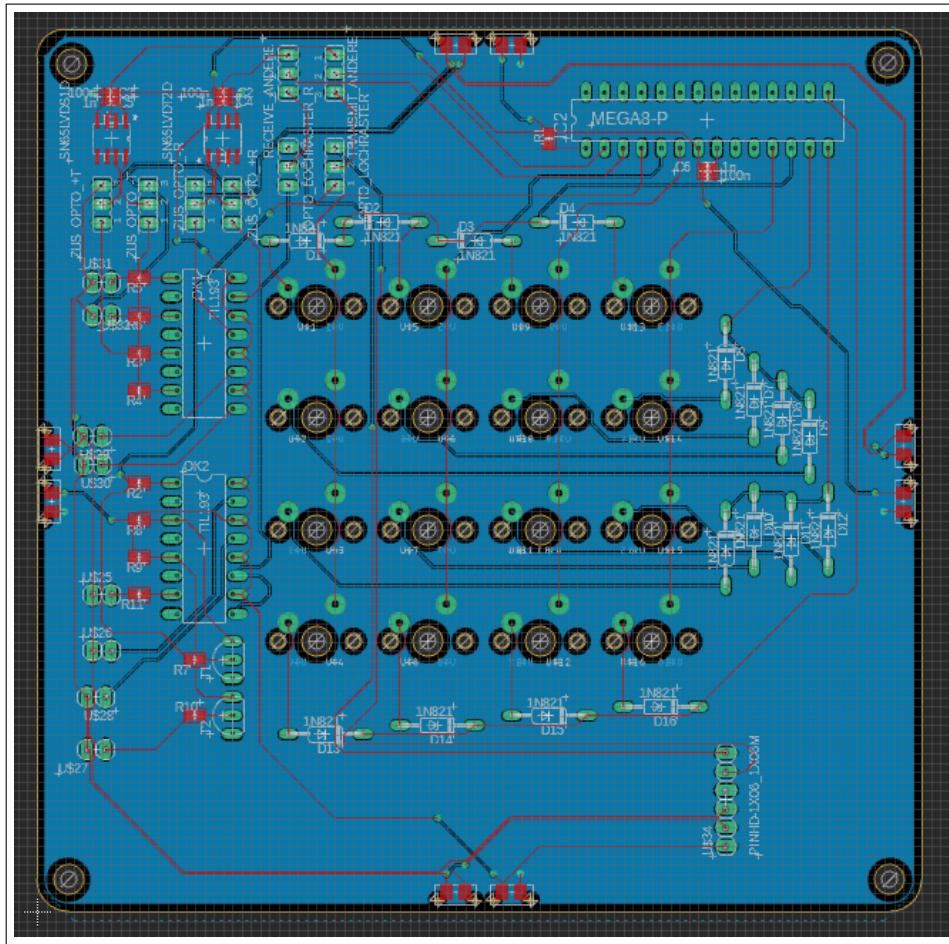


Abbildung 2.31: Layout Tastatur-Modul

2.8 Fazit

Die agile Herangehensweise stellt Herausforderungen dar, insbesondere wenn bisher wenig praktische Erfahrung mit dieser Art der Projektleitung vorhanden ist. Dennoch konnten klare Vorteile der agilen Entwicklung festgestellt werden. So wurde beispielsweise das anfängliche Ziel, mehrere Module zu entwickeln, rechtzeitig angepasst, um den zeitlichen Einschränkungen gerecht zu werden. Darüber hinaus war es möglich, Aufgaben gemeinsam zu bearbeiten oder neu zuzuweisen, wenn einzelne Gruppenmitglieder ausfielen.

Es zeigt sich jedoch, dass eine gründlichere Planungsphase dazu hätte beitragen können, einige Probleme frühzeitig zu erkennen und zu vermeiden. So haben beispielsweise die differenzielle Datenübertragung und das Flashen der ATtiny und ATmega viel Zeit in Anspruch genommen. Schwierigkeiten, die vermutlich durch eine bessere Abstimmung der Komponenten im Vorfeld hätten reduziert oder ganz vermieden werden können.

Des Weiteren wird in Anbetracht der noch verbleibenden Zeit das Audio-Modul aus dem Plan für dieses Projekt gestrichen und sich erstmal lediglich auf das Tastaturmodul fokussiert. Da dieses Projekt nach Abschluss des Faches „Bussysteme und Sensorik“ für den privaten Gebrauch der Gruppenmitglieder weiterentwickelt wird, wird dieses Modul wahrscheinlich in Zukunft eingeführt. Das grobe Layout eines weiteren Moduls wird sich nicht grundlegend von dem des Tastaturmoduls unterscheiden, daher liegt die Priorität erstmal in der Optimierung des vorhandenen Moduls.

2.8.1 Persönliches Fazit

Johanna Boettcher

Aus dem Projekt habe ich sehr viel mitnehmen können. Für mich persönlich war eine der größten Herausforderungen, die Umsetzung der differenziellen Übertragung im Layout, vor allem die Schwierigkeit trotz der vielen Jumper Verbindungen die Leiterbahnen nah beieinander und gleichlang zu halten. Doch wir konnten sehr viel aus genau diesen Schwierigkeiten für unseren zweiten Prototypen lernen. Ich bin sehr glücklich mit dem Ergebnis, da wir einen sehr guten Eindruck haben, was verbessert werden kann und wie die nächsten Schritte aussehen. Im Rückblick haben wir den Zeitaufwand für das Projekt etwas unterschätzt. Gerade durch unerwartete Herausforderungen wie Lieferschwierigkeiten und das Einarbeiten in neue Konzepte, war es schwierig, den Zeitrahmen einzuhalten. Dennoch war es eine wertvolle Erfahrung, die vor allem bei der Gestaltung und der Planung der nächsten Platine sehr wertvoll sein wird.

Marcel Eßmann

Das Projekt hat mir sehr viel Freude bereitet. Da wir alle von der Idee überzeugt waren, ließ sich recht schnell ein Plan entwickeln. Die Probleme, die wir hatten konnten wir meist mit gemeinsamer Denkleistung beheben oder alternativen finden. Wir haben uns vermutlich etwas viel vorgenommen, da wir jetzt am Ende eigentliche in komplett fertiges Produkt haben und nutzen wollten. Dies hat jedoch dazu geführt, dass wir als Gruppe auch nach Abschluss des Faches „Bussysteme und Sensorik“ noch weiter an diesem Produkt arbeiten werden und zumindest ein für uns zufriedenstellendes Produkt entwickeln. Durch das Projekt konnten wir einen guten Einblick in den Ablauf der Prototypenentwicklung bekommen und feststellen, dass nicht immer alles rund läuft.

Peter Fischer

Auch wenn die Entwicklung nicht immer glattlief und wir uns zu viel für diesen Zeitraum vorgenommen haben, hat mir das Projekt wirklich Spaß gemacht. Ich fand es gut, dass man eine Gruppe finden musste, die sich auf ein gemeinsames Thema einigt und jeder seine Vorstellungen und Ideen einbringen kann. Ich bin nicht enttäuscht, dass unsere Vorstellung nicht ganz erfüllt werden konnte, sondern eher erfreut, was wir erreicht haben. Am Anfang des Semesters hatte keiner von uns eine Idee, wie wir einen Datenbus selber schreiben, eine Platine für ein Keypad entwickeln oder unsere Hardware Befehle auf dem Computer ausführen lassen. Das alles mussten wir uns teilweise frühzeitig selbstständig erarbeiten und ein Protokoll entwerfen, über das wir mit dem Computer kommunizieren. Hätten wir gewusst, was alles auf uns zukommt, dann hätten wir unsere Vorstellung von vorne rein nicht so voll gestaltet. Nichtsdestotrotz, bin ich sehr froh über unseren funktionierenden Prototypen und möchte ihn weiterentwickeln, damit ich das „modulare Eingabesystem“ in voller Form selber nutzen kann.

Jannik Wendt

3 Anhang

3.0.1 Main Programm

```
1 #include "header.h"
2
3 #include <sstream>
4 #include <iomanip>
5
6 // true -> Upload fuer Master | false -> Upload fur Slave
7 #define SENDER true
8 //Spezifische Konfiguration fuer Slaves
9 #if SENDER == false
10 //Kommunikations Pin (MOSI) ATMEGA 15 | ATTINY 5
11 #define COMM_IN 12
12 //Kommunikations Pin (MISO) ATMEGA 16 | ATTINY 6
13 #define COMM_OUT 14
14 //0x01 := Tastatur, 0x02 := Audiopad, ...
15 #define FUNCTION 0x01
16 //Standardadresse eines neuen Busteilnehmers
17 #define STARTADDRESS 0x00
18 #define REQUESTADDRESS 0xFF
19 //PIN-Belegung spezifisch fuer Slave
20 #define ROW1 2
21 #define ROW2 0
22 #define ROW3 4
23 #define ROW4 16
24
25 #define COL1 17
26 #define COL2 5
27 #define COL3 18
28 #define COL4 19
29 //Spezifische Konfiguration fuer Master
30 #elif SENDER == true
```

```
31 //Kommunikations Pin
32 #define COMM_IN 12
33 //Kommunikations Pin
34 #define COMM_OUT 14
35 //Adresse des Hauptmoduls
36 #define STARTADDRESS 0x01
37 #define REQUESTADDRESS 0xFF
38 //bei n = INIT Polling Durchlaufen wird die INIT
39 //Nachricht an den PC gesendet
40 #define INIT 10
41 #define LIST 10
42 //Timeout-Schwellwert
43 #define TIMEOUT_THRESHOLD 3
44 #endif
45 //nach 'BITSTUFFING' gleichwertigen Bits wird ein
46 // anderswertiges Bit eingeschoben
47 #define BITSTUFFING 3
48 #define DELAY(x) delayMicroseconds(x)
49 //laenge eines Bits in us
50 #define SENDDELAY 500
51 #define NOADDRESS 0x00
52 #define WAITLOOP 100000
53 // Zum testen!
54 #define DEBUG false
55 #define DEBUG1 false
56 #define DEBUG2 false
57 #define DEBUG3 false
58 #define DEBUG4 false
59 #define DEBUG5 false
60 #define DEBUG6 false
61 #if SENDER == false
62 //Matrix-Tastatur-----
63 //           row col
64 int tasten2x2[2][2] = {
65     {0xA1, 0xB1},
66     {0xA2, 0xB2}
67 };
68 int tasten4x4[4][4] = {
69     {0xA1, 0xB1, 0xC1, 0xD1},
70     {0xA2, 0xB2, 0xC2, 0xD2},
```

```
71     {0xA3,0xB3,0xC3,0xD3},  
72     {0xA4,0xB4,0xC4,0xD4}  
73 };  
74  
75 int ROWS[2] {  
76     ROW1,  
77     ROW2  
78 };  
79 int COLS[2] {  
80     COL1,  
81     COL2  
82 };  
83  
84 int ROWS4[4] {  
85     ROW1,ROW2,ROW3,ROW4  
86 };  
87 int COLS4[4] {  
88     COL1,COL2,COL3,COL4  
89 };  
90 #endif  
91 //Variablen-----  
92 bool FLAG_NEWDVICE = false;  
93 bool FLAG_DELETEDVICE = false;  
94 volatile unsigned long timeStamp;      //Zeit Merker  
95 volatile unsigned long delayTime;       //berechnete Taktzeit  
96 volatile char global_adress = 0x00;    //eingelese Adresse  
97 volatile char global_COF = 0x00;        //Groesse der Nachricht  
98 volatile char global_message[8];        //eingelese Daten  
99 volatile char mask = 0x01;             //0000 0001 Binaermaske  
100 //Adressengroesse in Bits (maximal (2^adressSize)-2 Teilnehmer)  
101 volatile int global_adressSize = 3;  
102 volatile int global_COFSIZE = 3;         //Groesse des COF Pakets  
103 //Standardadresse eines neuen Busteilnehmers  
104 volatile char myAdress = STARTADRESS;  
105 //Feste Adresse des Hauptcontrollers  
106 volatile char controllerAdress = 0x01;  
107 //Zaehler fuer die Initialisierung  
108 volatile int init_cnt = 100;  
109 volatile int list_cnt = 100;  
110 char* global_decoded_message; //Bitstuffed message
```

```
111 char* global_BITSTUFFED_message; //Bitstuffed message
112 char* global_BITSTUFFED_message_recv; //Bitstuffed message empfangen
113 char* global_decoded_message_recv; //decodierte message empfangen
114 //Listen des Hauptcontrollers-----
115 //Speichern der Adressen in die Liste mit der zugehoerigen Funktion
116
117 //Spezifische Konfiguration fuer Master
118 #if SENDER == true
119     //Freie Adressen
120     std::list<char> unusedAdresses;
121     auto iterator_unusedAdresses = unusedAdresses.begin();
122     //Adressen aller Teilnehmer
123     std::list<char> Adresses;
124     auto iterator_Adresses = Adresses.begin();
125     //polling counter iteriert durch die liste der adressen
126     int polling_counter = 0;
127     //Anzahl der vergebenen Adressen
128     int num_adresses = 0;
129     std::list<device> devices;
130     auto iterator_devices = devices.begin();
131 #endif
132
133
134 //allg. Funktionen -----
135 #if SENDER == true
136     bool isArrayZero(volatile char* array, int size) {
137         for (int i = 0; i < size; i++) {
138             if (array[i] != 0) {
139                 return false; // Ein Element ist nicht 0
140             }
141         }
142         return true; // Alle Elemente sind 0
143     }
144 #endif
145
146 void appendChar(char* &str, char c) {
147     if (str == NULL) {
148         // Wenn str null ist, initialisieren Sie es mit dem neuen Zeichen
149         str = new char[2]; // 1 Zeichen + Nullterminator
150         str[0] = c;
```

```
151         str[1] = '\0';
152     } else {
153         size_t len = strlen(str);
154         // +1 fuer das neue Zeichen, +1 fuer den Nullterminator
155         char* newStr = new char[len + 2];
156         strcpy(newStr, str);
157         newStr[len] = c;
158         newStr[len + 1] = '\0';
159         delete[] str;
160         str = newStr;
161     }
162 }
163
164 //receive
165 bool syncronisation(){
166     //if (DEBUG) Serial.println("syncronisation");
167     unsigned long lok_delayTime;
168     timeStamp = micros();
169     while (digitalRead(COMM_IN) == HIGH);
170     lok_delayTime = micros() - timeStamp;
171     if (lok_delayTime > SENDDELAY*1.25 &&
172         lok_delayTime < SENDDELAY*1.75){
173         delayTime = lok_delayTime/1.5;
174         DELAY(delayTime*1.2);
175         return true;
176     }
177     else return false;
178 }
179
180
181 // Funktion zum Dekodieren von Bitstuffing
182 void decodeBitstuffedMessage(const char* stuffedMessage,
183                             char* &decodedMessage, int x) {
184     int count;
185     char lastBit = '\0';
186     size_t len = strlen(stuffedMessage);
187     decodedMessage = new char[len + 1]; // +1 fuer den Nullterminator
188     size_t decodedIndex = 0;
189     count = 1; // Zaehler auf 1 setzen
190 }
```

```
191 // Durchlaufen der bitgestopften Nachricht
192 for (size_t i = 0; i < len; ++i) {
193     char currentBit = stuffedMessage[i];
194     if (currentBit == lastBit) {
195         count++;
196         if (count < x) {
197             decodedMessage[decodedIndex++] = currentBit;
198         } else {// Wenn x gleiche Bits gefunden wurden
199             // Fuege das Bit hinzu
200             decodedMessage[decodedIndex++] = currentBit;
201             i++; // ueberspringe das naechste Bit
202             count = 1; // Zaehler zuruecksetzen
203             if (i >= len) break; // Vermeiden von ueberlaeufen
204             currentBit = stuffedMessage[i];
205         }
206     } else {
207         decodedMessage[decodedIndex++] = currentBit;
208         count = 1; // Zaehler zuruecksetzen und auf 1 setzen
209     }
210
211     lastBit = currentBit;
212
213 }
214
215 decodedMessage[decodedIndex] = '\0'; // Nullterminator hinzufuegen
216 if (DEBUG3) Serial.println("Empfangene decodierte Nachricht:");
217 if (DEBUG3) Serial.println(decodedMessage);
218 }
219
220
221
222 void parseDecodedMessage(const char* decodedMessage) {
223     // Adresse extrahieren
224     global_adress = 0;
225     for (int i = 0; i < global_adressSize; i++) {
226         if (decodedMessage[i] == '1') {
227             global_adress |= (1 << i);
228         }
229     }
230 }
```

```
231 // COF extrahieren
232 global_COF = 0;
233 for (int i = 0; i < global_COFSIZE; i++) {
234     if (decodedMessage[global_adressSize + i] == '1') {
235         global_COF |= (1 << i);
236     }
237 }
238
239 // Daten extrahieren basierend auf der Groesse des COF
240 // COF gibt die Groesse der Nachricht in Bytes an (0 bedeutet 1 Byte)
241 int dataSize = global_COF + 1;
242 for (int i = 0; i < dataSize; i++) {
243     char message[8] = {0};
244     for (int j = 0; j < 8; j++) {
245         if (decodedMessage[global_adressSize +
246                         global_COFSIZE + i * 8 + j] == '1') {
247             message[i] |= (1 << j);
248         }
249     }
250     global_message[dataSize-1-i] = message[i];
251     //global_message[i] = message[i];
252 }
253
254
255
256 // Ausgabe der extrahierten Bestandteile
257 if (DEBUG3) Serial.print("Empfangene Adresse: ");
258 if (DEBUG3) Serial.println(global_adress, HEX);
259 if (DEBUG3) Serial.print("Empfangener COF: ");
260 if (DEBUG3) Serial.println(global_COF, HEX);
261 if (DEBUG3) Serial.print("Empfangene Daten: ");
262 if (DEBUG3) for (int i = 0; i < dataSize; i++) {
263     Serial.print(global_message[i], HEX);
264     Serial.print(" ");
265 }
266 if (DEBUG3) Serial.println();
267 }
268
269
270
```

```
271 bool readMessage() {
272     //if (DEBUG) Serial.println("readMessage");
273     // neue Nachricht auf dem Bus
274     if (syncronisation()) {
275         // Initialisieren Sie die globale bitgestopfte Nachricht
276         global_BITSTUFFED_message_recv = new char[1];
277         global_BITSTUFFED_message_recv[0] = '\0';
278         // Lesen Sie die Nachricht bitweise ein
279         int eofCounter = 0;
280         while (true) {
281             char bit = (digitalRead(COMM_IN) == HIGH) ? '1' : '0';
282             appendChar(global_BITSTUFFED_message_recv, bit);
283             DELAY(delayTime);
284             // ueberpruefen Sie auf EOF (Ende der Nachricht)
285             if (bit == '1') {
286                 eofCounter = 1; // Start des EOF-Musters erkannt
287             } else if (eofCounter > 0) {
288                 eofCounter++;
289                 if (eofCounter == 6) {
290                     // EOF-Muster erkannt: 1 gefolgt von 5 Nullen
291                     // Entfernen Sie das EOF-Muster aus der Nachricht
292                     global_BITSTUFFED_message_recv[strlen(
293                         global_BITSTUFFED_message_recv) - 6] = '\0';
294                     break;
295                 }
296             } else {
297                 // Zaehler zuruecksetzen, wenn das Muster unterbrochen wird
298                 eofCounter = 0;
299             }
300         }
301         if (DEBUG3) Serial.println("Empfangene Nachricht:");
302         if (DEBUG3) Serial.println((global_BITSTUFFED_message_recv));
303         // Dekodieren Sie die bitgestopfte Nachricht
304         char* decodedMessage = NULL;
305         decodeBitstuffedMessage(global_BITSTUFFED_message_recv,
306             decodedMessage, BITSTUFFING);
307         global_decoded_message_recv = decodedMessage;
308         if (DEBUG3) Serial.println("Empfangene decodierte Nachricht:");
309         if (DEBUG3) Serial.println(global_decoded_message_recv);
310 }
```

```
311     parseDecodedMessage(global_decoded_message_recv);  
312  
313     if (DEBUG3) Serial.println("Adresse:");  
314     if (DEBUG3) Serial.println(global_adress,HEX);  
315     if (DEBUG3) Serial.println("COF:");  
316     if (DEBUG3) Serial.println(global_COF,HEX);  
317     if (DEBUG3) Serial.println("Daten:");  
318     if (DEBUG3) for (int i = 0; i <= global_COF; i++) {  
319         Serial.println(global_message[i],HEX);  
320     }  
321     delete[] decodedMessage; // Speicher freigeben  
322     return true;  
323 } else {  
324     return false;  
325 }  
326 }  
327  
328 //-----  
329 //send  
330  
331 bool awaitBusFree(){  
332     if (DEBUG2) Serial.println("awaitBusFree");  
333     for (int i = 0; i < 10000; i++){  
334         bool busFree = true;  
335         for (int j = 0; j < 4; j++) {  
336             if (digitalRead(COMM_IN) == HIGH) {  
337                 busFree = false;  
338                 break;  
339             }  
340         }  
341         DELAY(delayTime);  
342     }  
343     if (busFree) return true;  
344 }  
345     return false;  
346 }  
347 //Bleibt gleich  
348 void sendSOF(){  
349     if (DEBUG) Serial.println("sendSOF");  
350     digitalWrite(COMM_OUT, HIGH);
```

```
351     DELAY(SENDDELAY* 1.5) ;
352     digitalWrite(COMM_OUT, LOW) ;
353     DELAY(SENDDELAY) ;
354 }
355
356 void writeAdress(char adress) {
357     if (DEBUG) Serial.println("writeAdress");
358     for (int i = 0; i < (global_adressSize); i++) {
359         if ((adress & 0x01) == 0x01) {
360             appendChar(global_decoded_message, '1') ;
361             //digitalWrite(COMM_OUT, HIGH) ;
362             if (DEBUG) Serial.print("1") ;
363         } else {
364             //digitalWrite(COMM_OUT, LOW) ;
365             appendChar(global_decoded_message, '0') ;
366             if (DEBUG) Serial.print("0") ;
367         }
368         adress = adress >> 1;
369     }
370     if (DEBUG) Serial.println();
371 }
372
373
374 void writeCOF(char dataSize){
375     if (DEBUG) Serial.println("writeCOF");
376     for (int i = 0; i < (global_COFSIZE); i++) {
377         if ((dataSize & 0x01) == 0x01) {
378             appendChar(global_decoded_message, '1') ;
379             if (DEBUG) Serial.print("1") ;
380         } else {
381             appendChar(global_decoded_message, '0') ;
382             if (DEBUG) Serial.print("0") ;
383         }
384         dataSize = dataSize >> 1;
385     }
386     if (DEBUG) Serial.println();
387 }
388
389 void writeData(char dataSize, char* data) {
390     if (DEBUG) Serial.println("writeData");
```

```
391     for (int i = dataSize; i >= 0; i--)  
392     {  
393         for (int j = 0; j < 8; j++)  
394         {  
395             if ((data[i] & 0x01) == 0x01) {  
396                 appendChar(global_decoded_message, '1');  
397                 if (DEBUG) Serial.print("1");  
398             } else {  
399                 appendChar(global_decoded_message, '0');  
400                 if (DEBUG) Serial.print("0");  
401             }  
402             data[i] = data[i] >> 1;  
403         }  
404     }  
405 }  
406  
407 //Bleibt gleich  
408 void sendEOF(){  
409     if (DEBUG) Serial.println("sendEOF");  
410     digitalWrite(COMM_OUT, HIGH);  
411     if (DEBUG) Serial.print("1");  
412     DELAY(SENDDELAY);  
413     digitalWrite(COMM_OUT, LOW);  
414     for (int i = 0; i < 5; i++) { //5 mal?  
415         if (DEBUG) Serial.print("0");  
416         DELAY(SENDDELAY);  
417     }  
418     if (DEBUG) Serial.println();  
419 }  
420  
421 void sendDataOnBus(char* stuffedMessage) {  
422     if (DEBUG) Serial.println("sendDataOnBus");  
423     for (int i = 0; i < sizeof(stuffedMessage); i++) {  
424         if (stuffedMessage[i] == '1') {  
425             digitalWrite(COMM_OUT, HIGH);  
426         } else {  
427             digitalWrite(COMM_OUT, LOW);  
428         }  
429         DELAY(SENDDELAY);  
430     }  
431 }
```

```

431
432 }
433 void writeBitstuffedMessage(int x) {
434     if (DEBUG) Serial.println("writeBitstuffedMessage");
435     std::string stuffedMessage;
436     int count = 0;
437     char lastBit = '\0';
438     for (int i = 0; i < strlen(global_decoded_message); i++) {
439         char currentBit = global_decoded_message[i];
440         stuffedMessage += currentBit;
441         if (currentBit == lastBit) {
442             count++;
443             // Wenn x gleiche Bits gefunden wurden
444             if (count == x) {
445                 // Bitwechsel einfuegen
446                 char stuffedBit = (currentBit == '1') ? '0' : '1';
447                 stuffedMessage += stuffedBit;
448                 count = 1; // Zaehler zuruecksetzen
449                 switch (currentBit)
450                 {
451                     case '0': lastBit = '1';
452                         /* code */
453                         break;
454                     case '1': lastBit = '0';
455                         /* code */
456                         break;
457                     default:
458                         break;
459                 }
460             }
461         } else {
462             //Zaehler zuruecksetzen und auf 1 setzen
463             count = 1;
464             lastBit = currentBit;
465         }
466     }
467
468     // Update the global_BITSTUFFED_message with the stuffed message
469     delete[] global_BITSTUFFED_message;
470     global_BITSTUFFED_message = new char[stuffedMessage.length() + 1];

```

```
471     strcpy(global_BITSTUFFED_message, stuffedMessage.c_str());  
472     if (DEBUG3) Serial.println("Decoded Message:");  
473     if (DEBUG3) Serial.println(global_decoded_message);  
474     if (DEBUG3) Serial.println("Bitstuffed Message:");  
475     if (DEBUG3) Serial.println(global_BITSTUFFED_message);  
476 }  
477  
478 void sendMessage(char adress,char dataSize,char* data){  
479     if (DEBUG) Serial.println("sendMessage");  
480     writeAddress(adress);  
481     writeCOF(dataSize);  
482     writeData(dataSize,data);  
483     writeBitstuffedMessage(BITSTUFFING);  
484     //Abfrage Bus frei  
485     if (awaitBusFree()) {  
486  
487         if (DEBUG3) Serial.println("Sending!");  
488         sendSOF();  
489         sendDataOnBus(global_BITSTUFFED_message);  
490         sendEOF();  
491         global_decoded_message = NULL;  
492     }  
493 }  
494 //-----  
495  
496 void global_reset(){  
497     for (int i = 0; i < 8; i++)  
498     {  
499         global_message[i] = 0;  
500     }  
501     global_COF = 0;  
502     global_adress = 0;  
503 }  
504  
505  
506 bool await_response(char adress) {  
507     for (int i = 0; i < WAITLOOP; i++) {  
508         if (readMessage()) {  
509             if (DEBUG3) Serial.println("eingelesene Adresse:");  
510             if (DEBUG3) Serial.println(global_adress,HEX);
```

```
511     if (global_adress == adress) {
512         Serial.println("Antwort erhalten!");
513         return true;
514     }
515 }
516 }
517 return false;
518 }

519
520 //-----
521 //Spezifische Funktionen fuer Slaves
522 #if SENDER == false
523 void getAddress(){
524     //Bitte um Adresse + 2. byte = Funktion
525     char message[] = {REQUESTADDRESS,FUNCTION};
526     sendMessage(controllerAdress,0x01,message);
527     Serial.println("Adresse angefragt!");
528     //Antwort erhalten
529     if (await_response(NOADRESS)){
530         //Speichern der neuen Adresse
531         myAdress = global_message[0];
532         if (DEBUG3) Serial.print("meine Adresse: ");
533         if (DEBUG3) Serial.print(myAdress,HEX);
534         if (DEBUG3) Serial.println();
535     }
536 }
537 }

538
539 void myFunction(){
540     Serial.println("myFunction!");
541     char myDatasize = 0;
542     char myData[8] = {0};
543     //-----
544
545 //-----
546
547 //-----
548 for (int row = 0; row < 2; row++) {
549     digitalWrite(ROWS[row], HIGH);
550     for (int col = 0; col < 2; col++) {
```

```
551     if (digitalRead(COLS[col]) == HIGH) {
552         //bei OUTPUT ROWS[row] ist COLS[col] HIGH
553         myData[myDatasize] = tasten2x2[row][col];
554         myDatasize++;
555     }
556 }
557 digitalWrite(ROWS[row], LOW);
558 DELAY(5);
559 }
560
561 //-----
562 // Uebermitteln der Daten
563 if (myDatasize == 0){
564     sendMessage(controllerAdress,myDatasize,myData);
565 }else{
566     sendMessage(controllerAdress,myDatasize-1,myData);
567 }
568 }
569
570
571 //Spezifische Funktionen fuer Master
572 #elif SENDER == true
573
574 void printDeviceList(const std::list<device>& deviceList) {
575     // ueberpruefen, ob die Liste leer ist
576     if (deviceList.empty()) {
577         Serial.println("Die Liste ist leer");
578         return;
579     }
580     // Elemente der Liste ausgeben
581     auto it = deviceList.begin();
582     if (it != deviceList.end()) {
583         ++it; // Erster Teilnehmer wird uebersprungen
584     }
585     for ( ; it != deviceList.end(); ++it) {
586         //Serial.print("Adresse: ");
587         Serial2.println(it->adress, HEX);
588         //Serial.print(", Funktion: ");
589         Serial2.println(it->funktion, HEX);
590     }
591 }
```

```
591     }
592
593
594 void USB(char adress,char function,volatile char* data){
595     // uebermitteln der INIT Nachricht, zum feststellen des COM-Ports
596     if (adress == 0x00){
597         init_cnt++;
598         //INIT legt die fes, wie oft diese Nachricht uebermittelt wird
599         if (init_cnt >= INIT){
600             Serial2.println("INIT");
601             if (DEBUG5) Serial.println("INIT");
602             init_cnt = 0;
603         }
604     }
605     // uebermitteln der Geraeteliste mit den entsprechenden Funktionen
606     if (adress == 0x00){
607         list_cnt++;
608         //INIT legt die fes, wie oft diese Liste uebermittelt wird
609         //Die Flag wird immer dann gesetzt, wenn ein neues Geraet
610         //angeschlossen wird
611         if (list_cnt >= LIST | FLAG_NEWDEVICE){
612             FLAG_NEWDEVICE = false;
613             list_cnt = 0;
614             Serial2.println("LIST");
615             if (DEBUG5) Serial.println("LIST");
616             printDeviceList(devices);
617             Serial2.println("END");
618         }
619         return;
620     }
621     // Nullpointer abfangen
622     if (data == NULL){
623         if (DEBUG4) Serial.println("data == NULL");
624         return;
625     }
626     //keine Daten zu uebertragen
627     if (isArrayZero(data,8)){
628         if (DEBUG4) Serial.println("keine Daten zu uebertragen");
629         return;
630     }
```

```
631 // uebermitteln der Daten des jeweiligen Slaves
632 Serial2.println("START");
633 if (DEBUG5) Serial.println("START");
634 Serial2.println(adress,HEX);
635 if (DEBUG5) Serial.println(adress,HEX);
636 for (int i = 0; i <= global_COF; i++)
637 {
638     Serial2.println(data[i],HEX);
639     if (DEBUG5) Serial.println(data[i],HEX);
640 }
641 Serial2.println("END");
642 if (DEBUG5) Serial.println("END");
643 }

644

645

646 char newAddress(char function){
647     if (DEBUG4) Serial.println("newAdress");
648 //es sind noch Adressen frei
649 if (unusedAdresses.empty() == false)
650 {
651     //nehme eine freie Adresse
652     char freeAdress = unusedAdresses.back();
653     if (DEBUG) Serial.println(freeAdress,HEX);
654     //entferne diese aus 'unused Adresses'
655     unusedAdresses.pop_back();
656     //fuege Sie zur Pollingliste hinzu
657     Adresses.push_back(freeAdress);
658     //neuen Teilnehmer der device Liste hinzufuegen
659     Serial.print("device function: ");
660     Serial.println(function,HEX);
661     device teilnehmer(freeAdress,function,nullptr,0);
662     Serial.print("device adress: ");
663     Serial.println(teilnehmer.adress,HEX);
664     Serial.print("device funktion: ");
665     Serial.println(teilnehmer.funktion,HEX);
666     devices.push_back(teilnehmer);
667     //Pollingschleife vergroessern
668     num_adresses++;
669     //Uebermittlung der Geraeteliste, wenn ein Teilnehmer dazukommt
670     FLAG_NEWDEVICE = true;
```

```
671     if (DEBUG4) Serial.println("Neue Adresse: ");
672     if (DEBUG4) Serial.println(freeAdress,HEX);
673     return freeAdress;
674 }
675 // keine Adressen mehr frei!
676 else return 0x00;
677 }
678 void printList(const std::list<char>& lst) {
679     // Ueberpruefen, ob die Liste leer ist
680     if (lst.empty()) {
681         Serial.println("Die Liste ist leer");
682         return;
683     }
684
685     // Elemente der Liste ausgeben
686     for (const char &element : lst) {
687         Serial.print(element,HEX);
688         Serial.print(",");
689     }
690     std::cout << std::endl;
691 }
692
693 void timeout(char adress, bool no_response){
694     if (DEBUG) Serial.println("timeout");
695     //neuer Teilnehmer!
696     if (global_message[0] == REQUESTADDRESS && global_adress ==
697         controllerAdress && !no_response ) {
698         if (DEBUG2) Serial.println("Neuer Teilnehmer");
699         // neuen Teilnehmer hinzufuegen mit seiner Funktion
700         char freeAdress[] = {newAddress(global_message[1])};
701         // freie Adresse senden an noaddress
702         sendMessage(NOADDRESS,0,freeAdress);
703     }
704     //kein neuer Teilnehmer
705     else if (no_response && adress == NOADDRESS){
706         if (DEBUG2) Serial.println("kein Neuer Teilnehmer");
707         return;
708     }
709     //Keine Antwort des angefragten Teilnehmers
710     else if (no_response && adress != NOADDRESS){
```

```
711 // inkrementieren des Timeout-counters
712 iterator_devices->timeout++;
713 if (DEBUG2) Serial.println("keine Antwort von:");
714 if (DEBUG2) Serial.println(iterator_devices->adress,HEX);
715 if (iterator_devices->timeout >= TIMEOUT_THRESHOLD) {
716     //Adresse wieder freigeben
717     unusedAdresses.push_back(iterator_devices->adress);
718     //Adresse aus der Pollingliste entfernen
719     Adresses.remove(iterator_devices->adress);
720     //Adresse aus der device Liste entfernen
721     devices.erase(iterator_devices);
722     //Pollingschleife verkleinern
723     num_adresses--;
724     //Uebermittlung der Geraeteliste, wenn ein Teilnehmer wegfaellt
725     FLAG_NEWDEVICE = true;
726     //keine Iteration in der Pollingschleife
727     FLAG_DELETEDDEVICE = true;
728 }
729 }
730 //Antwort erhalten
731 else if (!no_response && adress != NOADDRESS){
732     //Timeout zuruecksetzen
733     iterator_devices->timeout = 0;
734     if (DEBUG2) Serial.println("Antwort von:");
735     if (DEBUG2) Serial.println(iterator_devices->adress,HEX);
736     //Daten speichern
737     iterator_devices->latest_data = global_message;
738 }
739 }
740
741 void polling(){
742     if (DEBUG) Serial.println("polling");
743     //naechstes element der adressliste
744     if (!FLAG_DELETEDDEVICE)
745     {
746         polling_counter++;
747     }
748     // ein Teilnehmer ist weggefallen -> nicht iterieren
749     else
750     {
```

```
751     FLAG_DELETEDDEVICE = false;
752 }
753 // ueberpruefen Sie, ob der polling_counter die
754 // Anzahl der Adressen erreicht hat
755 if (polling_counter >= num_adresses + 1) {
756     polling_counter = 0; // Zuruecksetzen des polling_counter
757 }
758 //Setzen des Iterators an die entsprechende Stelle
759 iterator_devices = devices.begin();
760 std::advance(iterator_devices,polling_counter);
761 bool response = false;
762 char request[] = {0xFF};
763 global_reset();

764 //request fuer abzufragende Adresse
765 sendMessage(iterator_devices->adress,0,request);
766 //warten auf Rueckmeldung
767 response = await_response(controllerAdress);
768 //Ruecksetzen des Timeouts (zudem Zuweisung von neuen Adressen)
769 timeout(iterator_devices->adress,!response);
770 //Uebermitteln der Daten an den PC
771 USB(iterator_devices->adress,iterator_devices->
772         funktion,iterator_devices->latest_data);
773 }

774 }
775
776
777 #endif
778
779 //SETUP-----
780
781 //Spezifisches Setup fuer Slaves
782 #if SENDER == false
783 void setup() {
784     Serial.begin(115200);
785 //__DON'T CHANGE!!!---
786     pinMode(COMM_IN, INPUT);
787     pinMode(COMM_OUT, OUTPUT);
788     if (DEBUG) Serial.println("COMM Pins are setup!");
789 //-----
790     pinMode(ROW1, OUTPUT);
```

```
791 pinMode(ROW2, OUTPUT);
792 pinMode(ROW3, OUTPUT);
793 pinMode(ROW4, OUTPUT);

794
795 pinMode(COL1, INPUT_PULLDOWN);
796 pinMode(COL2, INPUT_PULLDOWN);
797 pinMode(COL3, INPUT_PULLDOWN);
798 pinMode(COL4, INPUT_PULLDOWN);

799
800 }
801 //Spezifisches Setup fuer Master
802 #elif SENDER == true
803 void setup(){
804     Serial.begin(115200);
805     Serial2.begin(9600);
806     if (DEBUG) Serial.println("serial is setup!");

807
808
809     // Elemente hinzufuegen
810     //Adresse 0x01 fuer den Controller reserviert!
811     for (int i = 2; i < pow(2,global_adressSize); i++)
812     {
813         unusedAdresses.push_back(i);
814         if (DEBUG2) Serial.println(unusedAdresses.back(),HEX);
815     }
816     device geraet;

817
818     Adresses.push_front(NOADRESS);
819     device noaddress = device(NOADRESS,0,0,0);
820     devices.push_back(noadress);
821     iterator_devices = devices.end();
822     geraet = *iterator_devices;
823     Serial.print(geraet.adress,HEX);

824
825     //__DON'T CHANGE!!-----
826     pinMode(COMM_IN, INPUT);
827     pinMode(COMM_OUT, OUTPUT);
828     if (DEBUG) Serial.println("COMM Pins are setup!");
829     //-----
830 }
```

```
831 #endif
832
833 #if SENDER == true
834
835 void loop() {
836     if(SENDER){
837         polling();
838
839         //Debug Delay
840         if (DEBUG | DEBUG1 | DEBUG2 | DEBUG3 | DEBUG4) {
841             delay(1000);
842         }
843     }
844 }
845 #elif SENDER == false
846 void loop() {
847     if (readMessage())
848     {
849         // Noch keine Adresse
850         if (global_adress == myAdress && myAdress == NOADDRESS) {
851             getAddress();
852             global_reset();
853         }
854         //Aufruf meiner Adresse -> fuehre meine Funktion aus
855         else if (global_adress == myAdress && myAdress != NOADDRESS)
856         {
857             myFunction();
858         }
859     }
860 }
861 #endif
```

Listing 3.1: main.cpp

3.0.2 Main Header

```
1 #include <Arduino.h>
2 #include <iostream>
3 #include <list>
4 #include <cstring>
5 struct device
6 {
7     char adress;
8     char funktion;
9     volatile char* latest_data;
10    unsigned int timeout;
11
12    device() : adress(0x00), funktion(0x00),
13                latest_data(NULL), timeout(0){}
14
15 public:
16     device(char Adresse, char Funktion, char* Daten,
17            unsigned int Zaehler)
18     : adress(Adresse), funktion(Funktion), latest_data(Daten),
19       timeout(Zaehler){}
20 };
21
22
23 //recieve funktionen-----
24
25 /*
26 synchronisiert die Taktgeschwindigkeit
27 @return SOF wurde empfangen
28 */
29 bool syncronisation();
30
31 /*
32 liest die Adresse ein (global gespeichert)
33 @return eingelesene Adresse
34 */
35 char readAddress();
36
37 /*
38 liest COF ein (global gespeichert)
```

```
39 @return eingelesenes COF
40 /*
41 char readCOF();
42
43 /*
44 liest die Daten ein (global gespeichert)
45
46 @dataSize anzahl der einzulesenden datenbytes
47 @return
48 /*
49 void readData(volatile char* data, char dataSize);
50
51 /*
52 Kombiniert die vorangegangenen Funktionen um eine
53 gesamte Nachricht einzulesen
54 @return fertig eingelesen
55 /*
56 bool readMessage();
57
58
59 //transmit funktionen-----
60
61
62 /*
63
64 @return Bus ist frei
65 /*
66 bool awaitBusFree();
67 /*
68
69 @return
70 /*
71 void writeAdress(char adress) ;
72 /*
73
74 @return
75 /*
76 void writeCOF(char dataSize);
77 /*
78
```

```
79 @return
80 /*
81 void writeData(char data) ;
82 /*
83
84
85
86 /*
87
88 @return
89 /*
90 void setup() ;
91
92 /*
93
94 @return
95 /*
96 void transmitDeviceInfo(int stelle);
97 /*
98
99 @return
100 /*
101 int whichFunction(char adresse, volatile char* data);
102
103 /*
104
105 @return
106 /*
107 void functionKeypad(volatile char[ ]) ;
108
109 /*
110
111 @return
112 /*
113 void functionAudiopad(volatile char[ ]) ;
114
115 /*
116
117 @return
118 /*
```

```
119 void switchFunction(char adresse);  
120  
121 /*  
122 Vergibt neue Adressen  
123 @return neue Adresse (0x00, wenn keine Adressen mehr frei sind)  
124 */  
125 char newAddress(char function);  
126  
127 /*  
128 wartet auf die Antwort eines Busteilnehmers  
129 @address Adresse von der die Antwort kommen soll  
130 @return Antwort erhalten  
131 */  
132 bool await_response(char address);  
133  
134 /*  
135 Handling des timeouts von teilnehmern  
136 @return  
137 */  
138 void timeout(char address, bool no_response);  
139  
140 void global_reset();  
141  
142 void myFunction();
```

Listing 3.2: header.h

3.0.3 Main Header

```
1 import customtkinter as ctk # GUI elements
2 import serial.tools.list_ports # List available COM ports
3 import serial # Serial communication
4 import concurrent.futures # Manage concurrent tasks
5 import threading # Handle background threads
6 import keyboard # Simulate keypresses
7 import json # Save/load key mappings
8 import time # Manage delays
9 import os # File and system interactions
10
11
12 # Variable for debug output
13 debug = True
14
15 # 4x4 Key Layout
16 key_layout = [
17     f"{{chr(65 + row)}{col + 1}}" for col in range(4)] \
18     for row in range(4)
19 ]
20
21 # Initialize a 4x4 list to hold label references for the keys
22 labels = [[None for _ in range(4)] for _ in range(4)]
23
24
25 # Global variables for data processing
26 current_module = []
27 key_mappings = {}
28 modules = {}
29
30 # GUI Info window
31 info_window = None
32
33 # Text to display in the keybinds info window
34 key_mapping_info = """
35 Combine Keybinds by using '+'
36
37 ctrl = Control
38 alt = Alt
```

```
39 shift = Shift
40 enter = Return
41 space = Space
42 backspace = BackSpace
43 delete = Delete
44 esc = Escape
45 tab = Tab
46 capslock = Caps_Lock
47 home = Home
48 end = End
49 pageup = Page_Up
50 pagedown = Page_Down
51 up = Up
52 down = Down
53 left = Left
54 right = Right
55 win = Windows
56 f1 = F1
57 f2 = F2
58 f3 = F3
59 f4 = F4
60 f5 = F5
61 f6 = F6
62 f7 = F7
63 f8 = F8
64 f9 = F9
65 f10 = F10
66 f11 = F11
67 f12 = F12
68 """ .strip()
69
70 # Function for normalizing common user inputs
71 def normalize_key_binding(binding):
72     mapping = {
73         "ctrl": "Control",
74         "alt": "Alt",
75         "shift": "Shift",
76         "enter": "Return",
77         "space": "Space",
78         "backspace": "BackSpace",
```

```
79     "delete": "Delete",
80     "esc": "Escape",
81     "tab": "Tab",
82     "capslock": "Caps_Lock",
83     "home": "Home",
84     "end": "End",
85     "pageup": "Page_Up",
86     "pagedown": "Page_Down",
87     "up": "Up",
88     "down": "Down",
89     "left": "Left",
90     "right": "Right",
91     "win": "Windows",
92     "f1": "F1",
93     "f2": "F2",
94     "f3": "F3",
95     "f4": "F4",
96     "f5": "F5",
97     "f6": "F6",
98     "f7": "F7",
99     "f8": "F8",
100    "f9": "F9",
101    "f10": "F10",
102    "f11": "F11",
103    "f12": "F12",
104  }
105 parts = binding.split("+")
106 normalized_parts = [mapping.get(part.lower(), part) \
107                         for part in parts]
108 return "+".join(normalized_parts)
109
110
111 def find_keypad_port(com_ports, expected_message="INIT"):
112     # Nested function to check if a given port contains the
113     # expected message
114     def check_port(port_info):
115         # Debug: Log the port being checked
116         print(f"Check port: {port_info.device}")
117         try:
118             # Open the serial port with specified parameters
```

```
119     with serial.Serial(
120         port_info.device, baudrate=9600, timeout=2
121     ) as ser:
122         # Read and decode
123         data = ser.read(100).decode(errors="ignore")
124         # Check if "INIT" is received
125         if expected_message in data:
126             return port_info.device
127     except (serial.SerialException, UnicodeDecodeError):
128         pass
129     return None
130
131 # Use thread pool to check multiple ports at the same time
132 with concurrent.futures.ThreadPoolExecutor() as executor:
133     # Create a dictionary of future tasks for checking ports
134     futures = {
135         executor.submit(check_port, port): \
136             port for port in com_ports
137     }
138     # Process completed tasks as they finish
139     for future in concurrent.futures.as_completed(futures):
140         result = future.result()
141         if result:
142             # If a port returns a valid device then return
143             return result
144     return None
145
146
147 # Get all available Ports
148 available_ports = list(serial.tools.list_ports.comports())
149 # Find the Keypad among all ports
150 keypad_port = find_keypad_port(available_ports)
151
152
153 # This function maps data about the function a string (name)
154 def set_function_name(function):
155     function_map = {1: "Keypad", 2: "Audiomodul"}
156     return function_map.get(function)
157
158 # This function sets a given module to the currently
```

```
159 # selected module and updates the gui
160 def switch_module(module):
161     # Declare global variables for tracking
162     global current_module, key_mappings
163     current_module = module # Update current module
164
165     # Update the module dropdown
166     module_dropdown.configure(values=list(modules.keys()))
167     if (modules):
168         # Set the dropdown selection to the current module
169         module_var.set(current_module)
170     else:
171         module_var.set("No modules")
172
173     # Update the display
174     update_display(is_initializing=False)
175
176
177 # This is the main communication function between this script and
178 # the main controller (the hardware) via COM-Port
179 def read_from_com_port(serial_connection):
180     global module_count, modules, current_module, key_mappings
181     address = None
182     data = None
183     pressed_keys = [] # List to store all keys that are pressed
184
185     while True:
186         try:
187             # Check for data waiting in serial port
188             if serial_connection.in_waiting > 0:
189                 # Read and decode a line of data from the
190                 # serial connection
191                 message = (
192                     serial_connection.readline().decode(errors="ignore").strip()
193                 )
194                 if message:
195                     # START signal
196                     if message == "START":
197                         if debug: print("START")
198                         address = None
```

```

199         data = None
200         # Reset the pressed keys list
201         pressed_keys = []
202
203     # END signal
204     elif message == "END":
205         if debug: print("END")
206         # Process pressed keys if address and
207         # keys are set
208         if address is not None and pressed_keys:
209             module_name = f"{function_name} \
210             (Address {address})"
211             if module_name:
212                 for key in pressed_keys:
213                     # Check if the key has a
214                     # mapping for the current module
215                     if key in \
216                         key_mappings[module_name]:
217                         # Saves the mapped key
218                         # for the pressed Button
219                         press = \
220                             key_mappings[module_name][key]
221                         if press:
222                             if debug: print(
223                                 f"Key {key} on \
224                                 {module_name} activated.")
225                             # Simulate the press of
226                             # the mapped key or keys
227                             keyboard.press_and_release(press)
228             else:
229                 if debug: print(\n
230                     "No binding assigned.")
231             else:
232                 if debug: print(
233                     f"Invalid key {key} for \
234                     {module_name}.")
235             else:
236                 if debug: print(f"Invalid address: \
237                     {address}")
238             else:

```

```
239         if debug: print("Error: \
240                         Missing address or keys!")
241
242     # LIST signal to update module information
243     elif message == "LIST":
244         if debug: print("LIST")
245         new_modules = {}
246         while True:
247             # Read module information until END
248             # is received
249             module_message = (
250                 serial_connection.readline()
251                 .decode(errors="ignore")
252                 .strip()
253             )
254             if module_message == "END":
255                 if debug: print("END")
256                 break
257
258             # If the first message is a digit,
259             # then its the module address
260             elif module_message.isdigit() and \
261                 int(module_message) in range(2,8):
262                 address = int(module_message)
263                 if debug: print(f"{address} \
264                               (Address)")
265
266             # Read function associated with the
267             # address
268             function_message = (
269                 serial_connection.readline()
270                 .decode(errors="ignore")
271                 .strip()
272             )
273             # If the second message is a digit,
274             # then its the function number
275             if function_message.isdigit():
276                 function = int(function_message)
277                 if debug: print(f"{function} \
278                               (Function)")
279
280             # Get the function name from the
```

```
279         # function number
280         function_name = \
281             set_function_name(function)
282         # Safety measure: If there is a
283         # function name then set the
284         # module name
285         if function_name:
286             module_name = \
287                 f"{function_name} \
288                     (Address {address})"
289             new_modules[module_name] = {}
290             # Initialize key mappings
291             # if not already present
292             # (May be duplicate
293             # code -> switch_module
294             # function)
295             if module_name not in \
296                 key_mappings:
297                 key_mappings[module_name] \
298                     = {
299                         key: ""
300                         for row in key_layout
301                             for key in row
302                     }
303                     if debug: print( \
304                         f"Key mappings created for {module_name}.")
305             else:
306                 print(f"Unknown function \
307                     {function} for Address {address}.")
308
309             # Update current module and module list
310             # if needed
311             if not current_module and new_modules:
312                 current_module = sorted(
313                     new_modules.keys(), reverse=True
314                 )[0]
315                 module_count = len(new_modules)
316
317             # Update the current module only if
```

```
319         # there are changes in the modules
320     if new_modules != modules:
321         # Replace old modules with new modules
322         modules = new_modules
323         # Update module count
324         module_count = len(modules)
325
326         # Sort the module names and pick the
327         # first one as the current module
328     if (new_modules):
329         current_module = sorted(
330             new_modules.keys(), reverse=True
331         )[0]
332     else:
333         print("No Modules")
334
335         # Switch to the updated current module
336         switch_module(current_module)
337         if debug: print(f"Module count: \
338                         {module_count}")
339
340         # Handle messages (Data) that are not
341         # "END" or "INIT"
342     elif message not in ("END", "INIT"):
343         try:
344             if int(message) in range(2, 8):
345                 address = int(message)
346                 if debug: print(f"{address} \
347                               (Address)")
348             except ValueError:
349                 data = message
350                 if debug: print(f"{data} (Data)")
351                 if address is not None:
352                     pressed_keys.append(data)
353
354         # Small delay to prevent overwhelming the serial
355         # connection
356         time.sleep(0.05)
357
358     except serial.SerialException as e:
```

```
359     print(f"Error reading from COM port: {e}")
360     break
361 except Exception as e:
362     print(f"Unexpected error: {e}")
363     break
364
365
366
367 if keypad_port:
368     print(f"Start reading PORT: {keypad_port}")
369
370     # Open serial connection
371     ser = serial.Serial(keypad_port, 9600, timeout=1)
372
373     # Create a thread to handle reading from the COM port
374     read_thread = threading.Thread(
375         target=read_from_com_port,    # Function to run in the thread
376         args=(ser,),               # Pass serial connection as argument
377         daemon=True                # Set thread as a daemon to terminate
378                         # with the main program
379     )
380     read_thread.start()          # Start thread
381 else:
382     print("No controller connected!")
383
384
385 def update_display(is_initializing=True):
386     global current_module, key_mappings
387
388     # Update the display for each key in the 4x4 layout
389     for row in range(4):
390         for col in range(4):
391             # Get key at the current position
392             key = key_layout[row][col]
393             if is_initializing:
394                 # During display initialization, use global key mappings
395                 key_text = f"{key}\n({key_mappings.get(key, '')})"
396             else:
397                 # For display updates, use module specific key mappings
398                 key_text = (
```

```
399         f"\n{key}\n({key_mappings[current_module].get(\n400             key, '')})\"\n401     )\n402     # Update label text for the current key\n403     labels[row][col].configure(text=key_text)\n404\n405     # Update COM port label\n406     if keypad_port:\n407         com_port_label.configure(text=f"COM Port: {keypad_port}")\n408     else:\n409         com_port_label.configure(text="COM Port: Not Connected")\n410\n411     # Update module count label if not initializing\n412     if not is_initializing:\n413         if module_count:\n414             module_count_label.configure(text=\\n\n415                 f"Module count: {module_count}")\n416         else:\n417             module_count_label.configure(text=\\n\n418                 "Module count: No modules")\n419\n420\n421\n422 def save_mapping():\n423     if modules: # Check if modules are initialized\n424         try:\n425             # Open file "key_mappings.json" in write mode\n426             with open("key_mappings.json", "w") as f:\n427                 # Save key_mappings dictionary to the file in JSON format\n428                 json.dump(key_mappings, f, indent=4)\n429                 update_display(is_initializing=False)\n430                 print("Mappings successfully saved.")\n431         except Exception as e:\n432             print(f"An error occurred while saving mappings: {e}")\n433     else:\n434         print("Modules are not initialized. Nothing to save.")\n435\n436\n437\n438
```

```
439 def load_mapping():
440     global current_module, modules, key_mappings
441
442     # Check if the save file exists
443     if not os.path.exists("key_mappings.json"):
444         print("No save file found.")
445         return
446
447     # Open file "key_mappings.json" in read mode
448     with open("key_mappings.json", "r") as f:
449         loaded_data = json.load(f)
450
451     # Merge loaded data with the existing key mappings
452     for key in key_mappings.keys():
453         if key in loaded_data:
454             for sub_key in key_mappings[key].keys():
455                 if sub_key in loaded_data[key]:
456                     key_mappings[key][sub_key] = \
457                         loaded_data[key][sub_key]
458
459     # Update the module dropdown menu with the loaded keys
460     # module_dropdown.configure(values=list(key_mappings.keys()))
461     # -> Might be unnecessary
462
463     update_display(is_initializing=False)
464
465
466
467 def assign_key(row, col):
468     global current_module, key_mappings
469
470     # Identify key based on its position in the key layout
471     key = key_layout[row][col]
472
473     # Retrieve the new binding from the entry field
474     new_binding = entry_var.get()
475
476     # If the binding is empty, do nothing
477     if not new_binding:
478         return
```

```
479
480     # Normalize the key binding (important for basic keys like
481     # Control or Alt)
482     normalized_binding = normalize_key_binding(new_binding)
483
484     # Update the key mapping for the current module
485     key_mappings[current_module][key] = normalized_binding
486
487     update_display(is_initializing=False)
488
489
490
491 def reset_key(row, col):
492     global current_module, key_mappings
493
494     # Identify the key based on its position in the key layout
495     key = key_layout[row][col]
496
497     # Reset the mapping for the current module's key by
498     # setting it to an empty string
499     key_mappings[current_module][key] = ""
500
501     # Update the display to reflect the changes in key mappings
502     update_display(is_initializing=False)
503
504
505
506 # Set appearance and color for GUI
507 ctk.set_appearance_mode("dark")
508 ctk.set_default_color_theme("blue")
509
510 # Create the main window
511 root = ctk.CTk()
512 root.title("Input-Mapper")
513
514 # Get the screen dimensions for positioning the window
515 screen_width = root.winfo_screenwidth()
516 screen_height = root.winfo_screenheight()
517
518 # Define window size
```

```
519 initial_width = 600
520 initial_height = 600
521
522 # Calculate position of the screens center
523 position_top = int((screen_height - initial_height) / 2)
524 position_left = int((screen_width - initial_width) / 2)
525 root.resizable(True, True)
526
527 # Set the initial geometry and position of the window
528 root.geometry(f"{initial_width}x{initial_height}+\
529                 {position_left}+{position_top}")
530
531 # Configure grid layout of the root window
532 root.grid_rowconfigure(0, weight=1) # Row for module selection
533 root.grid_rowconfigure(1, weight=10) # Main frame for keys
534 root.grid_columnconfigure(0, weight=1) # Single column layout
535
536 # Create the module selection frame in the first row
537 module_frame = ctk.CTkFrame(root)
538 module_frame.grid(row=0, column=0, sticky="nsew", pady=10)
539
540 # Add a label to the module selection
541 module_label = ctk.CTkLabel(module_frame, text="Select Module:")
542 module_label.pack(side="left", padx=5)
543
544 # Create a dropdown menu for selecting modules
545 module_var = ctk.StringVar(value="No modules") # Default value
546 module_dropdown = ctk.CTkOptionMenu(
547     module_frame,
548     values=list(sorted(key_mappings.keys(), reverse=True)),
549     # Function to call when a module is selected
550     command=switch_module,
551     variable=module_var, # Variable of current selection
552 )
553 module_dropdown.pack(side="left", padx=5)
554
555 # Create the main frame for key mappings in the second row
556 frame = ctk.CTkFrame(root)
557 frame.grid(row=1, column=0, sticky="nsew", pady=10)
558
```

```
559 # Configure the 4x4 grid layout
560 frame.grid_rowconfigure(tuple(range(4)), weight=1)
561 frame.grid_columnconfigure(tuple(range(4)), weight=1)
562
563 # Loop through rows and columns to build the grid
564 for row in range(4):
565     for col in range(4):
566         # Get the key name from the key layout
567         key = key_layout[row][col]
568
569         # Create a frame for this cell and position it
570         label_frame = ctk.CTkFrame(frame)
571         label_frame.grid(row=row, column=col, padx=5, pady=5, \
572                           sticky="nsew")
573
574         # Create a label to display the key name and binding
575         # (initially empty)
576         labels[row][col] = ctk.CTkLabel(
577             label_frame,
578             text=f"{key}\n()",
579             corner_radius=5,
580         )
581         labels[row][col].pack(expand=True, fill="both", padx=5, \
582                               pady=5)
583
584         # Create frame for "Set" and "Reset" buttons
585         button_frame = ctk.CTkFrame(label_frame)
586         button_frame.pack(fill="x", pady=2)
587
588         # Configure the frame to have two columns with equal width
589         button_frame.grid_columnconfigure((0, 1), weight=1)
590
591         # Create a "Set" button
592         button = ctk.CTkButton(
593             button_frame,
594             text="Set",
595             # Function to call on press
596             command=lambda r=row, c=col: assign_key(r, c),
597             fg_color="#5A9BD8",
598             height=30,
```

```
599     width=50,  
600     )  
601     button.grid(row=0, column=0, padx=(2, 1), pady=(2, 0), \  
602                   sticky="ew")  
603  
604     # Create a "Reset" button  
605     reset_button = ctk.CTkButton(  
606         button_frame,  
607         text="Reset",  
608         # Function to call on press  
609         command=lambda r=row, c=col: reset_key(r, c),  
610         fg_color="#FF7F7F",  
611         height=30,  
612         width=50,  
613     )  
614     reset_button.grid(row=0, column=1, padx=(1, 2), \  
615                         pady=(2, 0), sticky="ew")  
616  
617  
618 # Create frame to hold key binding entry field and label  
619 entry_frame = ctk.CTkFrame(root)  
620 entry_frame.grid(row=2, column=0, sticky="nsew", pady=10)  
621  
622 # Create a label as explanation for the entry field  
623 entry_label = ctk.CTkLabel(entry_frame, text="Key Binding:")  
624 entry_label.pack(side="left", padx=5)  
625  
626 # Store user input  
627 entry_var = ctk.StringVar()  
628 entry_field = ctk.CTkEntry(entry_frame, textvariable=entry_var)  
629 entry_field.pack(side="left", padx=5)  
630  
631  
632 def open_info_window():  
633     global info_window  
634  
635     # Check if the window already exists and is open  
636     if info_window is not None and info_window.winfo_exists():  
637         return  
638
```

```
639 # Create a new window for key bindings info
640 info_window = ctk.CTkToplevel()
641 info_window.title("Key Bindings Info")
642 info_window.geometry("400x300")
643
644 # Calculate position to open info window
645 # on the right of the main window
646 window_width = 400
647 window_height = 600
648 x = root.winfo_x() + root.winfo_width()
649 y = root.winfo_y()
650 info_window.geometry(f"{window_width}x{window_height}+{x}+{y}")
651
652 # Add a scrollable frame
653 info_frame = ctk.CTkScrollableFrame(info_window)
654 info_frame.pack(fill="both", expand=True, padx=10, pady=10)
655
656 # Add a label to display key mapping information
657 info_label = ctk.CTkLabel(
658     info_frame,
659     text=key_mapping_info.strip(),
660     justify="left",
661     anchor="w",
662     fg_color=None,
663 )
664 info_label.pack(fill="both", expand=True, padx=5, pady=5)
665
666
667 # Create a button to open the info window
668 info_button = ctk.CTkButton(
669     entry_frame,
670     text="Info",
671     command=open_info_window,
672     fg_color="#5A9BD8",
673     corner_radius=5,
674 )
675 info_button.pack(side="left", padx=5)
676
677 # Create a frame for control labels
678 control_frame = ctk.CTkFrame(root)
```

```
679 control_frame.grid(row=4, column=0, sticky="nsew", pady=10)
680
681 # Control label for COM port status
682 com_port_label = ctk.CTkLabel(
683     control_frame, text="COM Port: Not Connected", anchor="w"
684 )
685 com_port_label.grid(row=0, column=0, sticky="w", padx=10, pady=10)
686
687 # Control label for module count
688 module_count_label = ctk.CTkLabel(
689     control_frame, text=f"Module count: No modules", anchor="w"
690 )
691 module_count_label.grid(row=0, column=1, sticky="w", \
692                         padx=10, pady=10)
693
694 control_frame = ctk.CTkFrame(root)
695 control_frame.grid(row=3, column=0, sticky="nsew", pady=10)
696
697 # Save button
698 save_button = ctk.CTkButton(
699     control_frame, text="Save Mapping", command=save_mapping
700 )
701 save_button.grid(row=0, column=0, padx=10)
702
703 # Load button
704 load_button = ctk.CTkButton(
705     control_frame, text="Load Mapping", command=load_mapping
706 )
707 load_button.grid(row=0, column=1, padx=10)
708
709 # Button alignment
710 control_frame.grid_columnconfigure(0, weight=1)
711 control_frame.grid_columnconfigure(1, weight=1)
712
713 update_display(is_initializing=True)
714 root.mainloop()
```

Listing 3.3: Input-Mapper Program; Durch die Formatierung in LaTeX lässt sich dieser schwierig lesen, am besten die mitgelieferte Python Datei öffnen und anschauen.