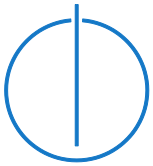
FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Masterarbeit in Informatik

HW-SW-Codesign einer Kompaktsteuerung mit besonderer Berücksichtigung der Echtzeitfähigkeit

Oliver Barta



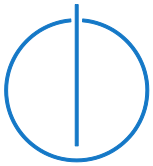
FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Masterarbeit in Informatik

HW-SW-Codesign einer Kompaktsteuerung mit besonderer Berücksichtigung der Echtzeitfähigkeit

HW-SW-Codesign of a compact PLC with special consideration of real-time performance

Bearbeiter: Oliver Barta
Aufgabensteller: Prof. Dr.-Ing. habil. Alois Knoll
Betreuer: Dr. techn. Alois Zoitl
Abgabedatum: 15.03.2016



Ich versichere, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Garching b. München, den 15.03.2016 _____

Ort, Datum Unterschrift

Zusammenfassung

In dieser Arbeit werden die Hardware und Software einer Kompaktsteuerung für den Forschungs- und Bildungsbereich auf Basis des Einplatinencomputers Raspberry Pi entwickelt und ein Prototyp gebaut. Dabei wird besonderer Wert auf eine hardwareseitige Unterstützung einer ereignisbasierten Steuerungssoftware, auf die Einhaltung einschlägiger Standards, auf die Integration angemessener Schutzfunktionen und natürlich auf das zeitliche Verhalten des Geräts gelegt. Letzteres, speziell die erreichbare Reaktionszeit, wird auf Grundlage des Prototypen untersucht und durch Anpassung der Software soweit verbessert bis es den aufgestellten Anforderungen genügt.

Die Arbeit beginnt mit einem Überblick über den Stand der Technik im Bereich von Speicherprogrammierbaren Steuerungen. Danach werden die Zielgruppe für das geplante Gerät und die von dieser Gruppe eingesetzten Automatisierungsanlagen betrachtet um daraus Anforderungen an die geplante Kompaktsteuerung abzuleiten. Diese werden dann wiederum genutzt um ein Konzept zu entwerfen und basierend darauf dann Hardware und Software zu entwickeln. Dabei liegt der Fokus anfangs mehr auf der Hardware und später mehr auf der Software, wobei die Anforderungen des jeweils anderen Teils nie außer Acht gelassen werden. Anschließend wird der Prototyp gebaut, welcher dann für die Untersuchung und Verbesserung des zeitlichen Verhaltens eingesetzt wird.

Stichworte:

HW-SW-Codesign; Kompaktsteuerung; PLC; SPS; Echtzeitfähigkeit

Abbildungsverzeichnis

Verzeichnis der Codeausschnitte

Tabellenverzeichnis

Abkürzungsverzeichnis

A/D	Analog/Digital
API	Application Programming Interface
ARP	Address Resolution Protocol
AS	Ablaufsprache
ASCII	American Standard Code for Information Interchange
AWL	Anweisungsliste
BIOS	Basic Input/Output System
BLOB	Binary Large Object
BSD	Berkeley Software Distribution
CD	Compact Disc
CDM	Charged Device Model
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CS	Chip Select
CSV	Comma-Separated Values
D/A	Digital/Analog
EEPROM	Electrically Erasable Programmable Read-Only Memory
eMMC	embedded Multimedia Card
ERP	Enterprise-Resource-Planning
ESD	Electrostatic Discharge
ETS	Equivalent Time Sampling
FBS	Funktionsbaustein-Sprache
FH	Fachhochschule
FIQ	Fast Interrupt Requests
FUP	Funktionsplan
GND	Ground
GPIO	General Purpose Input/Output
GPU	Graphics Processing Unit
HBM	Human Body Model
HDMI	High Definition Multimedia Interface
HTTP	Hypertext Transfer Protocol
HW	Hardware
ID	Identifier
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IO	Input/Output
IP	Internet Protocol
ISO	International Organization for Standardization
IT	Information Technology
KOP	Kontaktplan
LAN	Local Area Network
LED	Light-Emitting Diode
MAC	Media Access Control
MES	Manufacturing Execution System
MM	Machine Model
MPS	Modulares Produktions-System
MSO	Mixed-Signal Oszilloskop

OSI	Open Systems Interconnection
PC	Personal Computer
PCI	Peripheral Component Interconnect
PLC	Programmable Logic Controller
QFN	Quad Flat No Leads
RFC	Request for Comments
RT	Real-Time
SCADA	Supervisory Control and Data Acquisition
SCL	Serial Clock
SD	Secure Digital
SDA	Serial Data
SMD	Surface-Mounted Device
SPI	Serial Peripheral Interface
SPS	Speicherprogrammierbare Steuerung
SSH	Secure Shell
ST	Strukturierter Text
SW	Software
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
UEFI	Unified Extensible Firmware Interface
USB	Universal Serial Bus
UUID	Universally Unique Identifier
VGA	Video Graphics Array
WLAN	Wireless Local Area Network

Inhaltsverzeichnis

1. Einleitung

Automatisierung ist heute in fast allen Bereichen des Lebens allgegenwärtig. Im Alltag tritt sie in Erscheinung, wenn sich beispielsweise eine Tür selbstständig öffnet, weil sie erkannt hat, dass sich jemand nähert oder wenn die Waschmaschine selbstständig die Wäsche wäscht. Besondere Bedeutung hat Automatisierung aber in der Industrie, wo sie nicht mehr wegzudenken ist. Praktisch überall, wo etwas in größerer Stückzahl produziert wird, seien es Lebensmittel, Elektrogeräte, Werkzeuge, Möbel oder Kraftfahrzeuge werden Automatisierungsanlagen in großem Stil eingesetzt. Die Gründe dafür sind vielfältig. Häufig ist eine automatisierte Produktion mit Maschinen anstatt von Menschen einfach kostengünstiger, da das Produkt schneller, präziser und wiederholbarer als mit der Hand produziert werden kann. Sicherheitsaspekte können aber auch Grund für Automatisierung sein, wenn eine Aufgabe etwa eine hohe Verletzungsgefahr mit sich bringt oder die Gesundheit der Arbeiter in Gefahr bringen könnte, weil die Aufgabe beispielsweise den Umgang mit giftigen oder ätzenden Substanzen erfordert. Viele Aufgaben erfordern heutzutage aber auch einfach eine so hohe Präzision, wie sie von Menschen nicht erreicht werden kann.

Auf der untersten Ebene jeder Automatisierungsanlage finden sich Sensoren und Aktuatoren, welche Informationen sammeln und Werkstücke bearbeiten oder sonstige Aktionen ausführen. Um die gesammelten Informationen auszuwerten und die Aktionen der Aktuatoren dementsprechend anzupassen, werden heute überwiegend Speicherprogrammierbare Steuerungen (SPS) eingesetzt. Deren Vorteil im Gegensatz zu fest verdrahteten spezialisierten Steuerungen besteht darin, dass ihr Programm bei einer Änderung der Produktionsanlage modifiziert werden kann und die Ansteuerung der Aktuatoren den Bedürfnissen der Produktion angepasst werden kann.

Die eingesetzten Speicherprogrammierbaren Steuerungen sind heute vorwiegend proprietär. Ihre Hardwarekonstruktion ist nicht offengelegt und ihre Software jenseits des eigentlichen Steuerungsprogramms kann nur eingeschränkt angepasst werden. Selbst wenn eine Anpassung der Software möglich ist, unterliegt diese oft strengen Lizenzbedingungen, die eine Weitergabe der modifizierten Software verbieten.

Insbesondere im Forschungs- und Bildungsbereich ist es aber interessant, die Software einer SPS vollständig seinen Bedürfnissen anpassen zu können und seine Forschungsergebnisse auch problemlos veröffentlichen zu dürfen. Evtl. kann sogar eine Anpassung oder Erweiterung der Hardware gewünscht sein um neue Steuerungskonzepte testen zu können.

Im Gegensatz zu bisher weit verbreiteten Steuerungsprogrammen, die periodisch mit einer festen Zykluszeit ihr Programm abarbeiten, könnte so etwa eine ereignisbasierte Steuerungssoftware realisiert werden um deren Eigenschaften insbesondere im Hinblick auf die Echtzeitfähigkeit mit einer zyklisch arbeitenden Steuerung zu vergleichen. Um so etwas aber effizient umsetzen zu können, muss die Hardware der SPS dafür Unterstützung anbieten und beispielsweise mit Interrupts die Software über neue Ereignisse informieren.

In dieser Arbeit soll eine SPS mit Fokus auf die Bedürfnisse im Forschungs- und Bildungsbereich entwickelt werden, die sich sowohl zur ereignisbasierten als auch zur zyklischen Steuerung eignet, deren Software vollständig angepasst werden kann und deren Hardware erweiterbar ist. Die Eigenschaften der entwickelten Steuerung sollen dann im Hinblick auf die Echtzeitfähigkeit untersucht werden.

1.1. Vorgeschichte

Die Idee für die Entwicklung einer solchen SPS entstand an der Fakultät für Maschinenwesen an der Technischen Universität München. Dort wurde im Rahmen eines interdisziplinären 2-Mann-Projekts

ein Raspberry Pi basiertes Steuerungsgerät zum Betrieb der dort zu Studienzwecken eingesetzten Automatisierungsanlagen entwickelt. Es wurde ein erster Prototyp gebaut, der in Tests die Fähigkeit bewies, die dortigen Anlagen steuern zu können.

An dieser Stelle wurde das Potential dieses Projekts erkannt, da bereits Anfragen von Interessenten eingingen, die das Gerät für ihre Forschungsarbeit nutzen wollten. Leider konnte die erste Version nicht in Serie gehen, da ursprünglich nur eine geringe Stückzahl vorgesehen war und bei der Entwicklung auf eine einfache, kostengünstige manuelle Produktion Wert gelegt wurde. Auch wenn es möglich war ein einzelnes Gerät kostengünstig zu fertigen, war die Konstruktion für eine automatisierte Fertigung in größerer Stückzahl vollkommen ungeeignet.

Im Anschluss an das interdisziplinäre Projekt wurde das Gerät mit dem Ziel einer Serienfertigung weiterentwickelt und ein weiterer Prototyp gefertigt. Diese Version besaß entscheidende Verbesserungen, sie hätte größtenteils maschinell in Serie gefertigt werden können, besaß ein deutlich besseres robustes Gehäuse und bot dem Benutzer mehr Schnittstellen als der Vorgänger. Allerdings war diese Version, wie auch ihre Vorgänger speziell zur Steuerung der am Lehrstuhl eingesetzten Automatisierungsanlagen ausgelegt. Das Gerät war in der Lage die dortigen Anlagen zu steuern, erfüllte aber nicht vollständig die einschlägigen Standards für SPSen, weshalb Kompatibilitätsprobleme beim Einsatz mit anderen Anlagen zu erwarten waren. Daher wurde die weitere Entwicklung des Geräts eingestellt.

Da aber Interesse seitens der Forschung und Bildung vorhanden war, wurde beschlossen, die Idee nicht ganz aufzugeben, auch wenn das alte Design komplett verworfen werden musste. Es wurde beschlossen diese Arbeit als Gelegenheit zu nutzen eine solche SPS von Grund auf neu zu entwickeln.

1.2. Aufgabenstellung

Die Aufgabe besteht in der Entwicklung der Hardware und der Software einer Kompaktsteuerung, mit Fokus auf die Bedürfnisse der Forschung und Bildung, wobei die Steuerung alternativ auch als PC-basierte Lösung oder Slot-SPS (Begriffsklärung siehe Abschnitt 2.2) realisiert werden kann, sofern dies sinnvoll erscheint und die Bauform einer Kompaktsteuerung entspricht. Die zu entwickelnde Steuerung soll sowohl ereignis- als auch zyklusbasierte Steuerungsprogramme unterstützen. Die Software soll dabei vollständig anpassbar und die Hardware erweiterbar sein. Das Ergebnis sollte dabei möglichst konform zu einschlägigen Standards ausfallen um die Kompatibilität mit möglichst vielen Automatisierungsanlagen zu gewährleisten. Im Rahmen dieser Arbeit soll auch ein erster Prototyp der entwickelten Steuerung gefertigt und sein Verhaltens hinsichtlich der Echtzeitfähigkeit untersucht werden.

1.3. Lösungsansatz

Um dem Nutzer ein bekanntes System insbesondere seitens der Software zu liefern, soll als zentraler Logikbaustein der SPS einer der bekannten und weit verbreiteten Einplatinencomputer mit einem Linux basierten Betriebssystem zum Einsatz kommen. Dazu muss eine Schnittstellenplatine entwickelt werden, die die Schnittstellen zur Verfügung stellt, die zur Steuerung einer Automatisierungsanlage nötig sind. Die Platine muss dabei den Logikbaustein von der störungsreichen Umgebung der Automatisierungsanlage isolieren und vor Verkabelungsfehlern und Überlast schützt. Sie sollte relevante Ereignisse an den zentralen Logikbaustein weiterleiten, ohne dass dieser sie abfragen muss. Es sollte also eine Möglichkeit geschaffen werden aufgrund von Eingabeänderungen oder sonstigen Zustandsänderungen Interrupts im zentralen Logikbaustein auszulösen. Die gesamte Elektronik sollte in ein kompaktes Gehäuse integriert werden.

Um eine einfache Integration des entwickelten Geräts in Steuerungssoftware zu ermöglichen, soll eine Steuerungsbibliothek für C/C++ geschaffen werden, die von der Hardware abstrahiert ohne dem Nut-

zer übermäßig viel Kontrolle über das Laufzeitverhalten zu nehmen. Darüber hinaus soll eine Integration in 4DIAC bzw. das zugehörige Laufzeitsystem FORTE erfolgen um auch eine grafische Oberfläche zur Erstellung von Steuerungsprogrammen zur Verfügung zu haben.

Nach Abschluss der Hardwareentwicklung soll ein Prototyp gefertigt werden. Mit Hilfe der entwickelten Steuerungssoftware soll dann das Laufzeitverhalten insbesondere im Hinblick auf die Echtzeitfähigkeit untersucht werden, wozu interne Messungen durch den Logikbaustein, wie auch externe Messungen mit Messgeräten erfolgen sollen.

2. Stand der Technik

Dieses Kapitel dient dazu dem Leser einen Überblick über den aktuellen Stand der Technik zu liefern ohne sich dabei auf ein spezielles System zu konzentrieren. Begonnen wird mit dem allgemeinen Aufbau von Produktionsanlagen und einer Einordnung einer SPS ins Gesamtbild. Dabei werden auch die Aufgaben einer SPS, die sich aus ihrer Position innerhalb einer Produktionsanlage im Allgemeinen ergeben, geschildert. Darauf folgt eine Beschreibung des allgemeinen Aufbaus von SPSen und verschiedene Typen von SPSen werden vorgestellt. Bevor dann auf die Hardware einer SPS genauer eingegangen wird, werden zunächst die Komponenten, mit denen eine SPS direkt interagiert, beschrieben, da diese die Hardware insbesondere die Schnittstellen einer SPS maßgeblich bestimmen. Auf die Beschreibung der Hardware folgt eine Betrachtung der Softwarekomponenten. Am Schluss des Kapitels werden die Anforderungen an das zeitliche Verhalten einer SPS diskutiert.

2.1. SPS im Gesamtkontext

Eine Produktionsanlage kann allgemein in Ebenen, wie in Abbildung 1 dargestellt, untergliedert werden. Diese Unterteilung wird aufgrund der nach unten hin zunehmenden Anzahl an Einzelsystemen innerhalb einer Ebene üblicherweise als Pyramide, als die sog. Automatisierungspyramide, dargestellt.

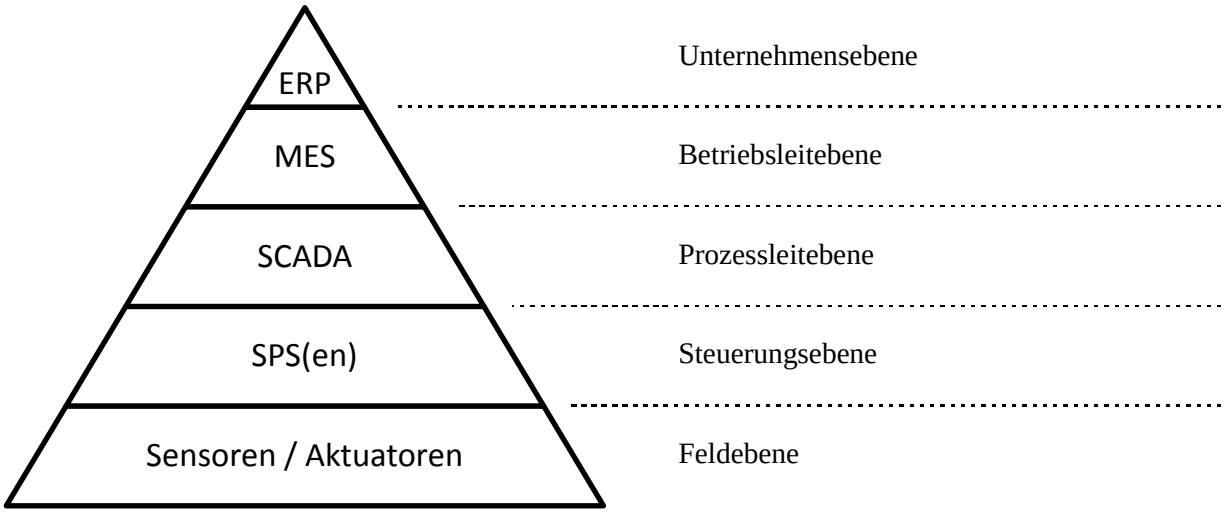


Abbildung 1: Automatisierungspyramide
Quelle: Angeleht an eine Darstellung in [1]

Auf der untersten Ebene, der Feldebene, finden sich Sensoren und Aktuatoren. Sensoren sammeln Daten über beispielsweise Größe, Gewicht und Position von Objekten, wie Aktuatoren und Werkstücken, und erfassen so den physikalischen Zustand der Anlage. Aktuatoren, wie Ventile, Motoren und Leuchtanzeigen verändern den Zustand der Anlage und führen so zum Transport und zur Bearbeitung der Werkstücke.

Sensordaten, z.B. die Position eines Werkstücks, welche von einer Lichtschranke oder einem Näherungssensoren erfasst wurde, werden auf der Steuerungsebene von Speicherprogrammierbaren Steuerungen ausgewertet. Basierend darauf werden die Aktuatoren so angesteuert, dass der gewünschte Ablauf der Produktion herbeigeführt wird. Es wird beispielsweise ein Motor eingeschaltet um ein Werkstück weiterzubewegen, nachdem ein Arbeitsschritt abgeschlossen wurde. Die Aufgabe der SPS besteht also hauptsächlich in der Auswertung von Sensordaten und dem Steuern und Regeln des Fertigungsprozesses.

Auf der darüber liegenden Prozessleitebene werden SCADA (Supervisory Control and Data Acquisition) Systeme und Prozessleitsysteme eingesetzt, die einem Benutzer Schnittstellen zur Über-

wachen und Visualisierung bereitstellen und einen Eingriff in den Fertigungsprozess ermöglichen. Diese Systeme dienen auch zur Archivierung von Messwerten um den Prozess auch nachträglich nachvollziehen zu können. [2, Seite /Supervisory_Control_and_Data_Acquisition] [2, Seite /Prozessleitsystem]

Die sich wiederum darüber befindliche Betriebsebene beschäftigt sich mit einer detaillierten Produktionsplanung, dem Materialmanagement und dem Qualitätsmanagement. Eingesetzt werden hier sog. MES (Manufacturing Execution System). Die Planung und das Management auf dieser Ebene beschränken sich immer auf eine lokale Produktionslinie.

Die globale Sicht auf das ganze Unternehmen über verschiedene Produktionslinien hinweg, findet sich erst auf der obersten Ebene, der Unternehmensebene, wo es um die Planung und Steuerung von Ressourcen, wie Kapital, Personal und auch Material, das sog. ERP (Enterprise-Resource-Planning) geht. Hier wird eine grobe unternehmensweite Produktionsplanung durchgeführt. [2, Seite /Leittechnik] [2, Seite /Automatisierungspyramide] [2, Seite /Enterprise-Resource-Planning]

Eine SPS dient wie gesagt, zur Steuerung und Regelung von Produktionsabläufen. Sie zeichnet sich im Vergleich zu einer festverdrahteten Steuerung dadurch aus, dass sie, wie der Name schon sagt, frei programmierbar ist und damit für verschiedene Aufgaben eingesetzt werden kann. Diese Flexibilität und günstige Preise, die durch Standardisierung und die Produktion in hohen Stückzahlen erreicht werden konnten, sind wesentliche Gründe für einen massenhaften Einsatz von SPSen. [2, Seite /Speicherprogrammierbare_Steuerung]

Die erste SPS wurde schon im Jahr 1969 in Amerika vorgestellt. Sie wurde von Dick Morley entwickelt und trug den Namen Modicon 084. Im Jahr 1974 kamen dann auch in Deutschland die ersten SPSen auf den Markt. Mittlerweile gibt es eine große Vielzahl von SPSen von zahlreichen Herstellern in verschiedensten Ausprägungen auf dem Markt. [2, Seite /Speicherprogrammierbare_Steuerung]

2.2. Aufbau und Typen von SPSen

Eine SPS besteht im Allgemeinen aus einer Zentralbaugruppe, einer Stromversorgungseinheit, sowie Schnittstellen um mit anderen Komponenten, beispielsweise Sensoren und Aktuatoren, zu interagieren (vgl. Abschnitt 2.3 f.). Die Zentralbaugruppe wiederum besteht aus einer CPU, Arbeitsspeicher und EEPROM oder einem anderen nicht flüchtigen Speicher. Sie dient dazu die Steuerungssoftware (vgl. Abschnitt 2.5 f.) auszuführen, welche Sensordaten auswertet und dementsprechend Aktuatoren schaltet. Die Stromversorgungseinheit kann auch als separates externes Gerät ausgeführt sein. [3]

Traditionell wird zwischen Hard-SPSen und Soft-SPSen unterschieden.

1. Hard-SPSen zeichnen sich dadurch aus, dass sie kein oder nur ein sehr minimalistisches meist proprietäres Echtzeit-Betriebssystem haben und im Wesentlichen nur das Steuerungsprogramm zur Steuerung einer Anlage ausführen. Darüber hinaus bieten sie keine Zusatzfunktionen. Ihre Hardware besitzt nur die oben beschriebenen Komponenten und üblicherweise keine Schnittstellen, die man aus dem PC-Bereich kennt, insbesondere kein VGA, HDMI, Ethernet (ausgenommen Industrial Ethernet) oder USB. Ihre CPU ist im Idealfall sogar speziell für die Ausführung von Steuerungssoftware, welche in den IEC 61131-3 Sprachen (vgl. Abschnitt 2.5) erstellt wurde, entwickelt und optimiert. Diese Art von SPS ist darauf spezialisiert harte Echtzeitanforderungen auch unter ungünstigsten Umständen stets zu erfüllen und besitzt geringe Fluktuationen im zeitlichen Verhalten. Hard-SPSen gelten als besonders zuverlässig und können üblicherweise sehr lange ohne Ausfälle betrieben werden.

2. Soft-SPSen besitzen hingegen ein umfangreiches Betriebssystem, üblicherweise Windows oder Linux und führen das Steuerungsprogramm zur Steuerung einer Anlage als Prozess in diesem System aus. Sie setzen Standard-CPUs aus dem PC-Bereich ein und bieten typischerweise auch die aus diesem Bereich bekannten Schnittstellen, insbesondere VGA, HDMI, USB und Ethernet, an. Es werden Zusatzfunktionen zur Bedienung und insbesondere auch Visualisierung angeboten. Dazu werden oft auch Funktionalitäten, wie HTTP-Server integriert um von Computern mittels Webbrowser die Steuerung überwachen und konfigurieren zu können. Obwohl diese Steuerungen traditionell in den gleichen Sprachen wie Hard-SPSen programmiert werden, besitzen sie keine spezialisierte Hardware zu deren Ausführung und können üblicherweise auch in anderen aus dem PC-Bereich bekannten Sprachen programmiert werden. Da die eingesetzten Betriebssysteme nicht dafür entwickelt wurden harte Echtzeitanforderungen zu erfüllen, sind, oft proprietäre, Echtzeiterweiterungen nötig um das zeitliche Verhalten zu verbessern. Allgemein haben Soft-SPSen den Ruf nicht in der Lage zu sein harte Echtzeitanforderungen zu erfüllen und nur für nicht zeitkritische Steuerungsaufgaben geeignet zu sein. Da ein Absturz des Betriebssystems, evtl. von einer anderen Anwendung verursacht, zwangsläufig auch die Steuerungssoftware betrifft, wird auch die Zuverlässigkeit von Soft-SPSen, was den unterbrechungsfreien Betrieb über lange Zeit hinweg betrifft, in Frage gestellt.

   PC-basierte Lösungen (siehe weiter unten) werden in der Literatur oft auch als Soft-SPS bezeichnet. Zu beachten ist, dass sich der Begriff Soft-SPS in der Literatur oft auch nur auf „reine Software“ [3, Seite 31] bezieht. Dabei kann sowohl die Programmierumgebung zur Erstellung von Steuerungssoftware, also auch eine Laufzeitumgebung, in der die Steuerungssoftware abläuft, oder beides gemeint sein. [3, Seite 31] [2, Seite /Codesys] Gelegentlich wird mit dem Begriff Soft-SPS auch die Simulation einer SPS in Software zum Testen von Steuerungsprogrammen oder zu Schulungszwecken bezeichnet. [4]

   Soft-SPSen werden zunehmend besser im Erfüllen von Echtzeitanforderungen, die Betriebssysteme sind zunehmend stabil und speziell konstruierte robuste Industrie PC sind verfügbar, wodurch Soft-SPSen auch in rauen Umgebungen über lange Zeiträume zuverlässig betrieben werden können.

Je nach Bauform einer SPS wird zwischen modularen SPSen, Kompakt-SPSen, PC-basierten Lösungen und Slot-SPSen unterschieden. Die Zuordnung eines Geräts ist nicht immer eindeutig, da hier auch verschiedenste Mischformen existieren, die Eigenschaften von mehreren Typen aufweisen. Im Folgenden werden die typischen Eigenschaften der vier Typen vorgestellt.

2.2.1. Modulare Steuerungen

Modulare Steuerungen zeichnen sich durch einen modularen Aufbau aus. Jede Komponente, also die Recheneinheit, die Stromversorgungseinheit, Ein- uns Ausgabe-Baugruppen (digital und analog), sowie weitere Schnittstellen sind als Module realisiert und können je nach Bedarf kombiniert werden. Dies garantiert eine hohe Flexibilität, da für jede Anlage genau die benötigten Bauteile zusammengesteckt werden können. Wenn später die Anlage modifiziert wird und weitere Schnittstellen benötigt werden, können diese problemlos ergänzt werden, ohne die ganze Steuerung austauschen zu müssen. Zur Programmierung solcher SPSen ist üblicherweise ein externer PC mit einer passenden Entwicklungsumgebung nötig, von wo das Programm an die SPS übertragen wird.

Modulare SPSen, sowie, die als nächstes vorgestellten, Kompakt-SPSen wurden historisch als Hard-SPSen mit spezialisierter Hardware und minimaler Software realisiert. Moderne Geräte setzten aber zunehmend umfangreichere Betriebssysteme und Standard-CPUs ein und bieten Zusatzfunktionen sowie auch zusätzliche Schnittstellen an.

2.2.2. Kompaktsteuerungen

Kompaktsteuerungen zeichnen sich dadurch aus, dass alle Funktionseinheiten einschließlich der Schnittstellen zur Automatisierungsanlage in einem Gehäuse integriert sind. Die Anzahl der zur Verfügung stehenden Schnittstellen ist üblicherweise stark beschränkt und kann, wenn überhaupt, nur sehr begrenzt erweitert werden. Diese Steuerungen können für kleinere Aufgaben, bei denen nur eine geringe Anzahl an Schnittstellen benötigt wird, eine preiswertere, einfachere Alternative zu modularen Steuerungen darstellen. Auch für die Programmierung von Kompaktsteuerungen ist üblicherweise ein externes Programmiergerät erforderlich. [5]

2.2.3. PC-basierte Lösungen

PC-basierte Steuerungen bestehen üblicherweise aus einer Kombination aus einem normalen PC oder einem Industrie PC (im Wesentlichen eine robustere Ausführung mit höherer Toleranz gegenüber Störungen und Umwelteinflüssen) und einer PCI-Schnittstellenerweiterungskarte oder externer Schnittstelle zur Ansteuerung einer Automatisierungsanlage.

Üblicherweise sind PC-basierte Lösungen als Soft-SPSen einzustufen. Sie setzen ein Desktop Betriebssystem, häufig Windows, gelegentlich auch Linux, ggf. mit Echtzeiterweiterungen, ein. Dazu kommt eine SPS-Software, die eine SPS Laufzeitumgebung zur Ausführung von Steuerungsprogrammen bereitstellt. Vorteil dieser Kombination besteht in der Möglichkeit Steuerungs- und Visualisierungsaufgaben sowie die Erstellung von Steuerungssoftware auf einem Gerät ausführen zu können. Darüber hinaus können eine gewohnte Desktop-Umgebung, die üblicherweise leistungsstarke CPU und der große Speicher des PC verwendet werden. Der große Nachteil ist die völlige Abhängigkeit vom Betriebssystem des PCs, welches nicht für Steuerungsaufgaben entwickelt wurde, was die Einhaltung von Echtzeitanforderungen erschwert und spezielle Erweiterungen erfordern kann. Darüber hinaus führt ein Absturz des Betriebssystems, welcher möglicherweise von einem ganz anderen Programm verursacht wurde, zur Unterbrechung der Steuerung. [6, Seite 83] [7] [3]

Prinzipiell ist es aber auch möglich eine PC-basierte Steuerung mit einem Echtzeitbetriebssystem oder auch ohne Betriebssystem zu betreiben. Dann weißt sie eher die Eigenschaften einer Hard-SPS auf. Es gibt auch Implementierungen bei denen ein Desktop-Betriebssystem, meist Windows und ein minimalistisches Echtzeitbetriebssystem auf einer PC-basierten Steuerung parallel ausgeführt werden, wodurch die Vorteile einer Hard- und einer Soft-SPS kombiniert werden können (vgl. Abschnitt 2.6).

2.2.4. Slot-SPSen

Eine Slot-SPS enthält, genau wie eine Kompakt-SPS, alle Komponenten einer SPS, ist aber nicht als eigenständiges Gerät, sondern als Einsteckkarte (üblicherweise PCI-Karte) für einen PC ausgeführt. Sie verfügt über ein eigenes Betriebssystem und führt die Steuerungsaufgaben unabhängig von PC aus. Der Vorteil besteht darin, dass über die PCI-Schnittstelle einfach Daten zwischen SPS und PC ausgetauscht werden können. So kann auf dem PC eine Bedienoberfläche und eine Visualisierung für den Nutzer angeboten werden. Die Steuerungsaufgaben werden aber unabhängig vom PC ausgeführt, wodurch das Zeitverhalten und die Zuverlässigkeit durch das Betriebssystem des PCs nicht negativ beeinflusst werden können. Diese Einsteckkarten verfügen häufig auch über eine eigene Stromversorgung, so dass selbst beim Abschalten oder im Falle eines kompletten Ausfalls des PC die Steueraufgaben erfüllt werden können. [7] Da bei PC-basierten Lösungen ebenfalls PCI-Karten zur Realisierung von Schnittstellen eingesetzt werden können, werden diese in der Literatur gelegentlich fälschlicherweise mit Slot-SPS verwechselt oder gleichgesetzt. [8, Seite 316]

Slot-SPSen wurden historische als Hard-SPSen realisiert, da die positiven Eigenschaften einer Soft-SPS durch den angebundenen PC bereitgestellt wurden und die Slot-SPS hauptsächlich für die zeitkri-

tische Steuerung zuständig war. Heute haben Slot-SPSen aber an Bedeutung verloren und wurden durch andere Bauformen weitestgehend verdrängt. Das Prinzip zwei separate Recheneinheiten bzw. Betriebssysteme einzusetzen wird aber weiterhin angewendet. Bei PC-basierten Lösungen werden teils zwei Betriebssysteme parallel auf derselben Recheneinheit eingesetzt (vgl. Abschnitt 2.6), bei modularen und Kompakt-SPS werden teils mehrere Recheneinheiten in einem Gerät verbaut um zeitkritische von weniger zeitkritischen Aufgaben zu trennen.

2.3. Komponenten mit Schnittstellen zur SPS

Da der Aufbau und die Funktionen einer SPS maßgeblich durch ihre Umgebung und die mit ihr interagierenden Komponenten bestimmt werden, werden in diesem Kapitel zunächst Komponenten mit Schnittstellen zur SPS und deren Eigenschaften betrachtet um im darauf folgenden Kapiteln den speziellen Aufbau und die Bestandteile einer SPS nachvollziehbar darlegen zu können.

2.3.1. Binäre Schalter und Sensoren

Die einfachsten und am weitesten verbreiteten Komponenten mit einer Schnittstelle zur SPS sind einfache Schalter und binäre Sensoren, welche nur zwei mögliche Zustände als Eingabe an die SPS liefern: Ein oder Aus. Auch wenn dies in der Theorie zunächst einfach klingt, kann sich die Entscheidung, ob Ein oder Aus in der Praxis schwierig gestalten, da kein diskreter Wert sondern nur ein, je nach eingesetzter Technologie und Anschlussverfahren, mehr oder weniger kontinuierlicher Wert an die SPS geliefert wird. Diese muss dann entscheiden, ob der geliefert Wert schon als Ein oder doch noch als Aus zu werten ist. Um Kompatibilität zwischen SPSen und Sensoren zu gewährleisten unterscheidet die IEC 61131-2 Norm zwischen drei Arten von Schaltern bzw. Sensoren und definiert jeweils passende Eingänge und Vorgaben, wie die Entscheidung über deren Zustand getroffen werden soll. Die Arten von Sensoren werden im Folgenden beschrieben, die Eingänge und das Entscheidungsverfahren später im Abschnitt 2.4.1 über die digitalen Eingänge einer SPS.

Elektromechanische Schaltgeräte

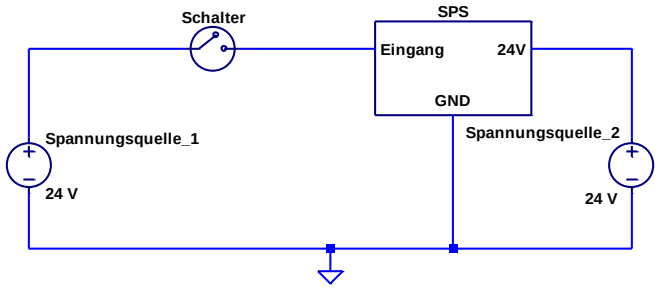


Abbildung 2: Schalter am Eingang einer SPS

Die erste Art stellen elektromechanische Schaltgeräte dar. Dazu zählen verschiedene mechanische Schalter, Drucktasten, Relais, Quecksilberschalter und ähnliche Vorrichtungen, die für ihre Funktion selber keinen Strom benötigen oder wie Relais dafür eine separate isolierte Stromversorgung haben. Im Falle eines Strom aufnehmenden Eingangs, werden diese einfach zwischen Spannungsquelle und den Eingang der SPS, wie in Abbildung 2 dargestellt, gehängt. Dabei kann für die Stromversorgung der SPS optional eine andere Spannungsquelle, wie in der Abbildung dargestellt, genutzt werden. Im Falle eines Strom liefernden Eingangs müsste der Schalter entsprechend zwischen Eingang und gemeinsamer Masse angeschlossen werden. Allen diesen Geräten ist gemein, dass im Aus-Zustand praktisch kein oder nur ein vernachlässigbar geringer Strom über den Schalter zum bzw. vom Eingang der SPS fließt.

Halbleiterschaltungen

Die anderen zwei Arten stellen Halbleiterschaltungen dar. Diesen gemein ist, dass diese für ihre Funktion Strom benötigen. Um zu verstehen, warum bei diesen eine weitere Unterteilung in zwei Arten nötig ist, muss man zunächst die gängigen Anschlussmethoden betrachten.

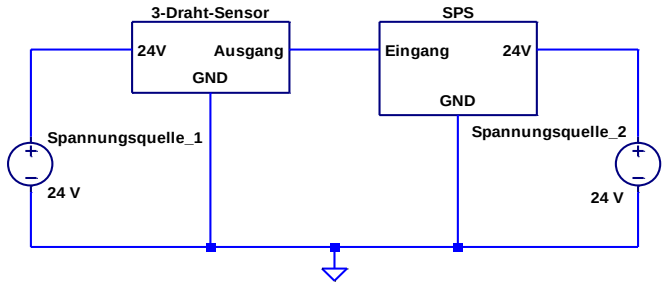


Abbildung 3: 3-Draht-Sensor am Eingang einer SPS

Abbildung 3 zeigt einen 3-Draht-Sensor dessen Ausgang mit dem Eingang der SPS verbunden ist. Die Stromversorgung des Sensors erfolgt über eine Leitung von der Spannungsquelle, welche übrigens gleichzeitig auch zur Versorgung der SPS genutzt werden könnte, und eine separate Leitung zur gemeinsamen Masse. Die Signalleitung zur SPS wird nur zur Signalübertragung genutzt. Für die SPS sieht ein solcher Sensor im Wesentlichen wie ein elektromechanisches Schaltgerät aus. Für den Anschluss werden allerdings 3 Leitungen benötigt, zwei zur Stromversorgung und eine zur Signalübertragung.

Da in Automatisierungsanlagen sehr viele Sensoren verbaut sein können, ist man bemüht Leitungen einzusparen. Dies wird durch den Einsatz von 2-Draht-Sensoren ermöglicht. Diese verzichten, wie in Abbildung 4 dargestellt, auf die zusätzliche Masseleitung und nutzen den Eingang der SPS als Weg zur Masse. Dies ist bei kleinen Sensoren möglich, da diese nur wenig Strom verbrauchen und es kein Problem darstellt, diese geringe Strommenge durch die SPS zu leiten. Allerdings müssen die Eingänge der SPS dafür ausgelegt sein und dürften den geringen Stromfluss, der im Aus-Zustand des Sensors zur Stromversorgung des Sensors aufrechterhalten werden muss, nicht als Ein-Zustand interpretieren.

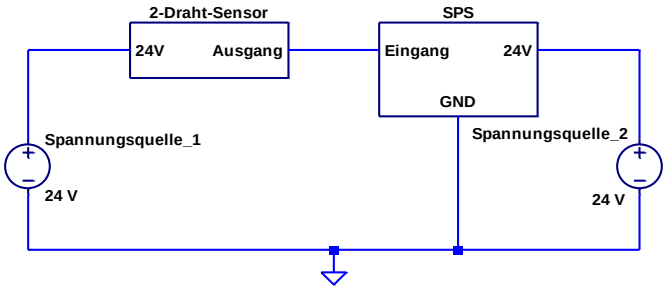


Abbildung 4: 2-Draht-Sensor am Eingang einer SPS

Natürlich will man aber nicht unnötig viel Strom durch den Eingang einer SPS leiten, weil dies zu einem erhöhten Stromverbrauch des Gesamtsystems und zu höherer Hitzeentwicklung in der SPS führen würde, sondern möglichst immer nur so viel, wie für den eingesetzten Sensor erforderlich. Dadurch entsteht die Einteilung in zwei weitere Arten, eine für Sensoren mit geringem und eine für Sensoren mit höherem Energiebedarf. Genaueres dazu findet sich im Abschnitt 2.4.1. Die hier beschriebenen Anschlussmethoden wären analog auch für Strom liefernde Eingänge realisierbar.

2.3.2. „Digitale“ Lasten

Eine Vielzahl von Aktuatoren, die mit einer SPS gesteuert werden, kennen, wie die zuvor beschriebenen Sensoren, nur zwei Zustände: Ein und Aus. Während größere Lasten, also die mit hohem Stromverbrauch, in der Regel einen eigenen Anschluss zur Stromversorgung haben und von der SPS über eine Steuerleitung nur ein Signal bekommen, das angibt, ob sie gerade ein oder aus sein sollen, werden kleinere Lasten direkt über die Steuerleitung von einem digitalen Ausgang der SPS mit Strom versorgt. Ja nachdem, ob ein Strom liefernder oder ein Strom aufnehmender Ausgang benutzt wird, wird die Last zwischen Ausgang und gemeinsamer Masse oder Stromversorgung und Ausgang gelegt. Ob beide Anschlussmöglichkeiten oder nur eine davon möglich sind, hängt immer von der konkreten Realisierung der Last ab. Um die Anforderungen an einen digitalen Ausgang herauszufinden, werden im Folgenden die Eigenschaften verschiedene Lasten betrachtet, die an eine SPS angeschlossen werden könnten.

Motoren

Motoren können in einer Produktionsanlage z.B. zum Betreiben von Fließbändern eingesetzt werden. Da SPSen typischerweise mit 24 V Gleichstrom arbeiten und die genauen Eigenschaften eines Motors nur bei direktem Betrieb an einem Ausgang ohne weitere Hilfsmittel wie Relais oder spezielle Motorsteuerungen von Interesse sind, beschränkt sich die Betrachtung im Folgenden auf Gleichstrommotoren, insbesondere auf klassische bürstenbehaftete Motoren, welche Schleifkontakte zur periodischen Änderung der Stromrichtung in ihren Magnetspulen nutzen, da diese lediglich eine Gleichspannung benötigen und direkt an einen Ausgang gehängt werden können, sofern dieser in der Lage ist genug Strom zu liefern.

Der erste interessante Punkt beim Betreiben eines Elektromotors ist der Einschaltvorgang. Beim Anlaufen eines Motors wird für das Beschleunigen zunächst mehr Energie benötigt als im normalen Betrieb mit gleichbleibender Drehzahl. Daher fließt zunächst ein sog. Anlaufstrom, der üblicherweise um ein Vielfaches höher ist als der Nennstrom der nach Erreichen der Nenndrehzahl fließt. Mit zunehmender Drehzahl sinkt der Anlaufstrom langsam ab bis schließlich die Nenndrehzahl erreicht wird. [2, Seite /Einschaltstrom]

Während ein bürstenbehafteter Motor läuft, können an den Schleifkontakten Funken, sog. Bürstenfeuer, entstehen, welche Störungen in die Anschlussleitung einspeisen können. [2, Seite /Bürstenfeuer] Der Abschaltvorgang ist ebenfalls von Interesse. Dabei verhält sich ein Motor wie eine induktive Last und verursacht damit die gleichen Probleme wie ein Ventil oder Relais. Diese Lasten und die entstehenden Probleme sind im nächsten Unterabschnitt beschrieben.

Ventile und Relais

Sehr häufig müssen Ventile oder Relais mit einer SPS gesteuert werden. Beide funktionieren nach demselben Prinzip und haben vergleichbare Eigenschaften. Sie besitzen jeweils eine Spule, in der ein Magnetfeld erzeugt wird, wodurch ein magnetischer oder ferromagnetische Zylinder bewegt wird, der einen Kontakt schließt oder öffnet oder ein Ventil bewegt. Neben Versorgungsspannung und benötigter Stromstärke ist hier vor allem die Induktivität der eingesetzten Spule für einen steuernden Ausgang interessant.

Die Induktivität sorgt nämlich dafür, dass beim Einschaltvorgang die Stromstärke nur langsam ansteigt und beim Ausschaltvorgang nur langsam abfällt. Während dieser Effekt beim Einschaltvorgang nicht stört bzw. eher von Vorteil ist, führt die Aufrechterhaltung des Stromflusses beim Ausschaltvorgang zu Problemen. Wird die Leitung beim Ausschalten durch einen Schalter einfach unterbrochen, führt der anhaltende Stromfluss zu einer kurzfristigen Überspannung. Diese macht sich bei einem me-

chanischen Schalter in Form von Funken, also einem kurzfristigen erneuten Schließen des Schalters durch die Luft, bemerkbar und führt zu einer erhöhten Abnutzung der Kontakte. Bei einem Halbleiterschaltelement kann diese Überspannung zu einer Beschädigung oder Zerstörung des Schalters führen. [2, Seite /Schutzbeschaltung]

Daher ist es notwendig den Stromfluss kontrolliert abzubauen. Eine einfache Lösung bietet eine Freilaufdiode, welche parallel zur Spule entgegengesetzt zur Versorgungsspannung eingesetzt wird und beim Abschaltvorgang weiteren Stromfluss durch die Spule erlaubt. Damit kann die in der Spule gespeicherte Energie kontrolliert innerhalb der Spule abgebaut, d.h. in Wärme umgewandelt, werden.

Bei diesem Vorgehen ist aber zu beachten, dass sich die Abschaltzeit dadurch verlängert und der mechanische Abschaltvorgang langsamer abläuft. Dies hat bei Ventilen lediglich den Nachteil, dass sie langsamer schalten, bei Relais führt dies unter Umständen aber zu einer schnelleren Abnutzung der Kontakte, da sich diese nur langsam auseinanderbewegen und so wiederum dort Funken entstehen können. [2, Seite /Schutzbeschaltung]

Wenn ein schnelles Schalten erforderlich ist, ist es daher erforderlich eine spezielle Schutzschaltung einzusetzen, die die gespeicherte Energie möglichst schnell außerhalb der Spule abbaut. Zu diesem Zweck gibt es eine ganze Reiher verschiedener Schutzschaltungen (vgl. [2, Seite /Schutzbeschaltung]). Ausgänge von SPSen haben aber häufig auch entsprechende Schutzmaßnahmen integriert, so dass bei kleinen induktiven Lasten auf eine externe Schutzschaltung gänzlich verzichtet werden kann (vgl. Unterabschnitt Überspannungsschutz im Abschnitt 2.4.2).

Anzeigen

Anzeigen werden heute überwiegend mit LEDs realisiert. Diese sind als Last am Ausgang einer SPS unproblematisch, da sie in der Regel mit einem hohen Vorwiederstand betrieben werden, weil sie typischerweise eine Vorwärtsspannung von nur wenigen Volt haben. Durch den hochohmigen Vorwiderstand weisen sie im Wesentlichen die Eigenschaften einer ohmschen Last auf und da LEDs, die als Statusanzeigen eingesetzt werden, typischerweise nur wenige mA benötigen, stellen sie einen Ausgang vor keine besondere Herausforderung.

Etwas problematischer können Glühbirnen und andere Geräte mit Heizelementen sein, da diese als Kaltleiter im kalten Zustand einen geringeren ohmschen Widerstand aufweisen und daher einen höheren Einschaltstrom haben, welcher erst mit zunehmender Temperatur auf den Nennwert absinkt, wodurch ein Ausgang der SPS kurzfristig stärker belastet wird. [2, Seite /Einschaltstrom]

2.3.3. Analoge Regler und Sensoren

Analoge Regler wie Potentiometer oder Sensoren, die Gewicht, Größe, Temperatur, Druck oder eine andere kontinuierliche Größe messen, liefern nicht nur die zwei Werte Ein und Aus, sondern Werte aus einem kontinuierlichen Wertebereich. Die gelieferten Werte sind in irgendeiner Weise, nicht notwendigerweise linear, proportional zum Messwert. Ist die SPS nur an der Über- oder Unterschreitung eines Grenzwertes oder nur an einer groben Quantisierung interessiert, können externe Schaltungen zur Digitalisierung eingesetzt werden und das Ergebnis kann an einen oder an einige wenige digitale Eingänge einer SPS geliefert werden. Ist hingegen eine feine Quantisierung gewünscht, ist ein analoger Eingang an der SPS nötig.

Variable Größe

Zur Übertragung der Messgröße an die SPS muss ein analoger Sensor irgendeine physikalische Größe ändern, die an den Eingang der SPS weitergeleitet und von der SPS gemessen werden kann. Prinzipiell könnten zur analogen Übertragung verschiedenste Größen eingesetzt werden, insbesondere Spannung,

Stromstärke, Frequenz und Widerstand. Denkbar wären auch optische oder akustische Übertragungen. In der Praxis eingesetzt werden heute überwiegend Spannung und Stromstärke, in speziellen Fällen auch Widerstandsänderungen. Letztere werden beispielsweise bei bestimmten Temperatursensoren eingesetzt. In besonderen Fällen werden auch pneumatische Systeme mit Luftdruckleitungen zur Übertragung analoger Signale verwendet (vgl. Abschnitt 2.4.7). [9] [10] [2, Seite /Einheitssignal]

Für geringe Distanzen kann eine Übertragung mittels Spannung eingesetzt werden. Diese ist am Eingang leicht auszuwerten und energiesparend, da kein Stromfluss nötig ist. Für längere Distanzen ist aber ein Spannungssignal ungeeignet, da es durch Leitungswiderstände verfälscht wird, was einen sehr hochohmigen Eingang erforderlich macht, der den Stromfluss gering hält und so den Spannungsabfall in der Leitung und damit die Verfälschung verringert. Dadurch wird das Signal aber anfällig gegen kapazitive und induktive Störungen.

Für größere Entfernungen wird eine Übertragung mittels Stromstärke bevorzugt, da diese weniger anfällig gegen externe Störungen ist. Die Stromstärke kann hinreichend groß gewählt werden um Störungen durch kapazitive Kopplung vernachlässigen zu können. Induktiven Störungen kann durch die Verwendung von verdrillten Kabeln entgegengewirkt werden. Übertragungen mittels Stromstärke werden nicht durch Leitungs- und Kontaktwiderstände, verfälscht, da jeglicher Spannungsabfall durch Widerstände vom Sensor kompensiert werden kann. Damit ist eine Übertragung mittels Stromstärke weitestgehend unabhängig von Leitungsquerschnitt und Länge der Signalleitung und damit sehr zuverlässig. [5, Seite 243] [2, Seite /Stromschnittstelle]

Wertebereiche

Bei einer Übertragung mittels Spannung sind die Wertebereiche -10 V bis 10 V und -5 V bis 5 V, sowie 0 V bis 10 V und 0 V bis 5 V üblich, aber auch andere Wertebereiche sind anzutreffen. Für spezielle Anwendungen, wie die Temperaturmessung mittels Thermoelementen, kann der Wertebereich auch nur wenige mV umfassen. [11] Bei der Stromstärke wird meist entweder der Bereich von 0 mA bis 20 mA oder von 4 mA bis 20 mA verwendet. [9]

Die Verwendung des Bereichs von 4 mA bis 20 mA für die Stromstärke hat deutliche Vorteile. Da immer ein Stromfluss von mindestens 4 mA aufrechterhalten wird, kann dieser gleichzeitig zur Stromversorgung des eingesetzten Sensors genutzt werden, wodurch eine zusätzliche Versorgungsleitung, für den Sensor entfallen kann (vgl. Ausführung über 2-Draht-Sensoren im Unterabschnitt Halbleiterschaltungen im Abschnitt 2.3.1). Darüber hinaus kann so ein Ausfall des Sensors oder ein Kabelbruch leicht aufgrund des fehlenden Stromflusses erkannt werden. [5, Seite 244]

2.3.4. Analoge Lasten

Als Lasten an einem analogen Ausgang einer SPS kommen in erster Linie analoge Eingänge von spezialisierten Steuerungen in Betracht (vgl. auch Abschnitt 2.3.5). Beispielsweise Motorsteuerungen, welche einen Elektromotor mit variabler Geschwindigkeit betreiben können und als Steuersignal ein analoges Signal erwarten. Dafür werden Spannungs- oder Stromsignale mit Bereichen, die den von analogen Sensoren verwendeten entsprechen, eingesetzt. Also beispielsweise von 0 V für Stillstand des Motors bis 10 V für maximale Geschwindigkeit bei einem Spannungseingang oder entsprechend 4 mA bis 20 mA bei einem Stromsignal. [12]

Abgesehen von analogen Eingängen kommen nur sehr schwache Verbraucher als analoge Lasten in Frage. Es könnte beispielsweise eine LED zur Statusanzeige an einen analogen Ausgang angeschlossen werden, wenn deren Helligkeit variiert werden soll.

2.3.5. Komplexere Geräte

Neben den bisher vorgestellten vergleichsweise einfachen Geräten, die mit einer SPS nur über die bisher betrachteten einfachen digitalen und analogen Signale kommunizieren, sind noch einige komplexere Geräte zu betrachten. Dazu zählen analoge Sensoren und Aktuatoren, die das HART Interface unterstützen, verschiedene spezialisierte Steuerungen, Sensoren und Aktuatoren mit Feldbussen, andere SPSen und Geräte auf der Prozessleitebene. Alle davon werden im Folgenden kurz betrachtet.

Analoge Sensoren und Aktuatoren mit HART Interface

Moderne analoge Sensoren und Aktuatoren stellen an ihrer analogen Schnittstelle oft auch eine Unterstützung für das HART Protokoll bereit. HART steht für "Highway Addressable Remote Transducer" und wurde von der "HART Communication Foundation" entwickelt. Das Protokoll nutzt eine analoge Leitung zwischen einem analogen Ausgang eines Sensors und einem analogen Eingang der SPS bzw. zwischen einem analogen Ausgang der SPS und einem analogen Eingang eines Aktuators zur Übertragung digitaler Signale. Dazu wird dem analogen Stromsignal, welches einen Wertebereich von 4 mA bis 20 mA nutzt, ein Frequenzsignal überlagert. Dieses wechselt zwischen einer Frequenz von 1200 Hz und 2400 Hz um eine logische 1 bzw. eine logische 0 darzustellen. Das analoge Signal ist dabei nach einer Tiefpassfilterung weiterhin nutzbar. Geräte, die das HART Protokoll nicht unterstützen, setzen üblicherweise Tiefpassfilter ein um Störungen zu unterdrücken und können das Frequenzsignal daher nicht erkennen. Mit Hilfe des HART Protokolls ist so eine bidirektionale digitale Kommunikation über eine analoge Leitung möglich. Vorausgesetzt, dass ein analoger Sensor bzw. Aktuator dieses Protokoll unterstützt, können damit beispielsweise digital Konfigurationsparameter an einen Sensor gesendet werden, der für seine normale Funktion sonst nur einen analogen Ausgang bereitstellt. So können zusätzliche Leitungen zur Konfiguration eingespart werden. [13, Seite 985] [5, Seite 247]

Spezialisierte Steuerungen

Eine SPS ist im Allgemeinen für eine große Vielzahl von Steuerungs- und Regelungsaufgaben ausgelegt und nicht auf eine konkrete Aufgabe spezialisiert. Auch wenn sie dadurch viele verschiedene Aufgaben erfüllen kann, gibt es Aufgaben, die von einer Steuerung, die speziell für diese eine konkrete Aufgabe entwickelt wurde, besser erfüllt werden können und andere Aufgaben die nur von spezialisierten Steuerungen übernommen werden können. Dafür kann es verschiedene Gründe geben, eine Aufgabe kann z.B. eine sehr hohe Leistung erfordern, die von einem normalen Ausgang einer SPS nicht geliefert werden kann. Beispielsweise der Betrieb eines größeren Elektromotors erfordert sehr hohe Stromstärken und Spannungen, die ein digitaler Ausgang einer SPS üblicherweise nicht liefern kann. Elektromotoren können Wechselspannungen benötigen, die von einer SPS in aller Regel nicht bereitgestellt werden können. Andere Aufgaben können wiederum so schelle Reaktionen auf Eingabeänderungen oder eine so schnelle spezialisierte Analyse der Eingabedaten erfordern, dass auch dazu eine SPS nicht in der Lage ist.

In solchen Fällen kommen spezialisierte Steuerungen zum Einsatz. Ein gutes Beispiel sind Motorsteuerungen, die die nötige Spannung und Stromstärke liefern können und gleichzeitig den Motor so ansteuern können, dass beispielsweise eine Geschwindigkeitsregelung möglich ist. Spezialisierte Steuerungen können üblicherweise über Schnittstellen mit einer SPS verbunden und von dieser gesteuert werden. So kann die SPS über die spezialisierte Steuerung z.B. einen Motor Steuern ohne sich um die genaue Ansteuerung und die hohe Leistung kümmern zu müssen. Oft reicht es aus, wenn die SPS die gewünschte Motorgeschwindigkeit vorgibt und die Motorsteuerung kümmert sich um den Rest.

Zur Kommunikation zwischen einer SPS und einer spezialisierten Steuerung können verschiedene Schnittstellen zum Einsatz kommen. In einfachen Fällen kann die Steuerung digitale Eingänge, wie

die an einer SPS, bereitstellen, die mit digitalen Ausgängen der SPS direkt verbunden werden können oder mit analogen Spannungs- oder Stromsignalen, wie im Abschnitt 2.3.4 erwähnt, arbeiten. Es werden aber auch Feldbussysteme, wie CAN, PROFIBUS oder PROFINET eingesetzt. Weitere Informationen zu Feldbussen finden sich im Abschnitt 2.4.5. Auch herstellerspezifische Anschlüsse und Protokolle werden eingesetzt, was natürlich eine Interoperabilität zwischen den Produkten verschiedener Hersteller beeinträchtigt. [14] [15] [12]

Sensoren und Aktuatoren mit Feldbussen

Auch mache Sensoren und teils auch einfache Aktuatoren können anstatt über digitale oder analoge Signalleitungen über Feldbussysteme angeschlossen werden, da sich dadurch im Allgemeinen der Verkabelungsaufwand deutlich reduzieren lässt. [2, Seite /Speicherprogrammierbare_Steuerung] [2, Seite /Feldbus]

Andere SPSen

Ebenfalls interessant kann es sein eine SPS direkt mit einer anderen SPS zu verbinden. Dies kann sinnvoll sein, wenn mehrere SPSen zusammenarbeiten müssen und daher Daten austauschen oder sich synchronisieren müssen. Es kann aber auch zum Testen eingesetzt werden. Eine SPS kann dabei eine Automatisierungsanlage simulieren und auf Ausgaben der zu testenden SPS wie eine Automatisierungsanlage reagieren und simulierte Sensordaten zurückliefern. Dies kann sowohl zum Testen der Hardware als auch zum Testen der Software, insbesondere des Steuerungsprogramms, eingesetzt werden. Dabei können digitale Eingänge der einen SPS mit den digitalen Ausgängen der zweiten und digitale Ausgänge der ersten mit den digitalen Eingängen der zweiten verbunden werden. Entsprechend kann dies auch mit analogen Ein- und Ausgängen geschähen. Auch eine Verbindung über Feldbusse wäre denkbar.

Prozessleitebene

Zur Kommunikation mit höheren Ebenen in der Automatisierungspyramide kommen beispielsweise Ethernet-Verbindungen basierend auf TCP oder Bussysteme, wie Modbus zum Einsatz. [2, Seite /Supervisory_Control_and_Data_Acquisition]

2.3.6. Umgebung

Besondere Anforderungen an eine SPS stellt die Umgebung in Produktionsanlagen dar. In solchen Anlagen sind viel stärkere elektrische Störungen, durch induktive und kapazitive Einkopplungen vorhanden als beispielsweise in einem normalen Haushalt, da dort viele elektrische Systeme mit teils sehr hoher Leistungsaufnahme auf engstem Raum arbeiten. Auch Faktoren wie Hitze, Kälte und Feuchtigkeit müssen viel stärker berücksichtigt werden als z.B. bei Desktopcomputern, die üblicherweise in halbwegs gut klimatisierten Wohnräumen oder Büros stehen. Mechanische Beanspruchungen des Gehäuses und der Verkabelung, als auch Vibrationen können auch viel eher und in stärkerem Ausmaß auftreten. Ein Schutz gegen elektrostatische Entladungen ist mindestens genau so nötig wie bei jedem anderen elektronischen Gerät. Welche Auswirkungen diese Anforderungen auf die Konstruktion von SPSen haben, wird im Abschnitt 2.4.7 erläutert.

Einen Spezialfall stellen explosive Umgebungen in manchen Produktionsanlagen dar, bei denen Gasgemische in der Umgebungsluft vorhanden sind, die sich durch Funken oder Hitze entzünden könnten. Funken können im normalen Betrieb beim Schalten von elektromechanischen Schaltern insbesondere mit induktiven Lasten oder auch bei ungenügendem Überspannungsschutz entstehen. Darüber hinaus könnten Funken bei einer Funktionsstörung einer SPS auftreten, wenn beispielsweise ein Kurzschluss entsteht. Durch besonders starke Belastung oder einen Kurzschluss könnte es auch zu einer Hitzeentwicklung kommen, die ebenfalls ein entsprechendes Gasgemisch entzünden könnte. Wenn in solchen

Umgebungen eine SPS betrieben werden soll, muss sichergestellt werden, dass es unter keinen Umständen zu einer Entzündung des Gasgemisches kommen kann. Welche Möglichkeiten es dafür gibt, wird im Abschnitt 2.4.7 beschrieben.

2.4. Schnittstellen einer SPS

In diesem Abschnitt wird basieren auf den im vorherigen Abschnitt beschriebenen Anforderungen an eine SPS die Hardware einer SPS vorgestellt. Der Fokus liegt dabei auf den Schnittstellen, die eine SPS typischerweise bietet um mit den zuvor beschriebenen Komponenten zu interagieren.

2.4.1. Digitale Eingänge

Zur Interaktion mit einfachen Schaltern und binären Sensoren, wie sie in Abschnitt 2.3.1 beschrieben sind, besitzen SPSen digitale Eingänge. Um Kompatibilität zwischen Schaltern und SPSen zu gewährleisten sind diese in der IEC 61131-2 Norm standardisiert. Diese definiert, um den Anforderungen verschiedener Anwendungsfälle gerecht zu werden, drei verschiedene Typen von Eingängen. Darüber hinaus fordert sie, dass jeder digitale Eingang einer SPS mit einer Anzeige ausgestattet sein muss, die den Ein-Zustand anzeigt. [16]

Die Norm setzt keinen linearen Zusammenhang zwischen Spannung und Stromstärke am digitalen Eingang voraus. Es sind beliebige insbesondere auch nicht lineare Eingangskennlinien erlaubt. Dies kompliziert zwar die Definition des Schaltverhaltens, lässt dem Entwickler eines digitalen Eingangs aber mehr Freiheit bei der konkreten Umsetzung und eröffnet so Potential zur Optimierung (vgl. Unterabschnitt Kennlinienoptimierung in diesem Abschnitt).

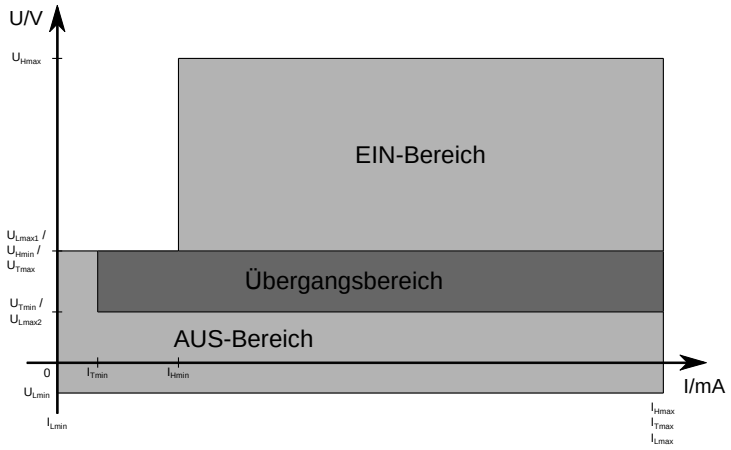


Abbildung 5: Arbeitsbereiche gemäß IEC 61131-2
Quelle: Angeleht an eine Abbildung in [16]

Das Schaltverhalten wird durch die Definition von U/I-Arbeitsbereichen festgelegt. Es gibt den EIN-Bereich, den AUS-Bereich und dazwischen einen Übergangsbereich, in dem die Implementierung nach Belieben den Schaltpunkt platzieren kann. Wenn es aufgrund einer Hysterese einen Schaltpunkt beim Einschalten und einen zweiten beim Ausschalten gibt, müssen logischerweise beide in diesem Bereich liegen. Die Bereiche sind stets, wie in Abbildung 5 dargestellt, angeordnet, nur die Grenzen verschieben sich je nach Typ des Eingangs. Die Kennlinie eines digitalen Eingangs muss so verlaufen, dass sie sich stets innerhalb eines dieser Bereiche befindet um der Norm zu genügen. In der Praxis werden stets möglichst steile und üblicherweise zumindest abschnittsweise lineare Kennlinien genutzt. Der Grund dafür wird im Unterabschnitt Kennlinienoptimierung in diesem Abschnitt erläutert. [16]

Typ 1

Typ 1 Eingänge sind für den Anschluss elektromechanischer Schaltgeräte, wie im Unterabschnitt Elektromechanische Schaltgeräte im Abschnitt 2.3.1 beschrieben, konzipiert. Sie zeichnen sich dadurch aus, dass sie sehr steile lineare Kennlinien erlauben, wodurch schon bei geringem Stromfluss die Spannung ansteigt und der EIN-Bereich erreicht wird. Abbildung 6 zeigt die Arbeitsbereiche eines Typ 1 Eingangs und eine Beispielkennlinie. Aufgrund der Möglichkeit derart steiler Kennlinien sind diese Eingänge ungeeignet zum Anschluss von Halbleiterschaltungen, die den Eingang zur Stromversorgung mitbenutzen wollen, wie 2-Draht-Sensoren, und auch im Aus-Zustand einen signifikanten Stromfluss aufrechterhalten müssen. [16]

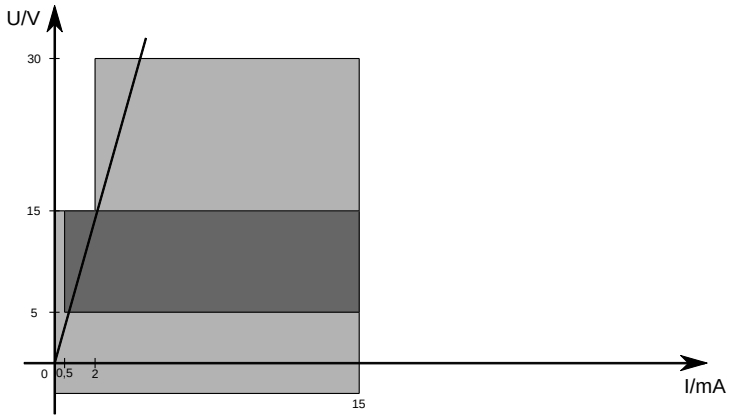


Abbildung 6: Arbeitsbereiche eines Typ 1 Eingangs
Quelle: Grenzwerte aus [16]

Typ 2

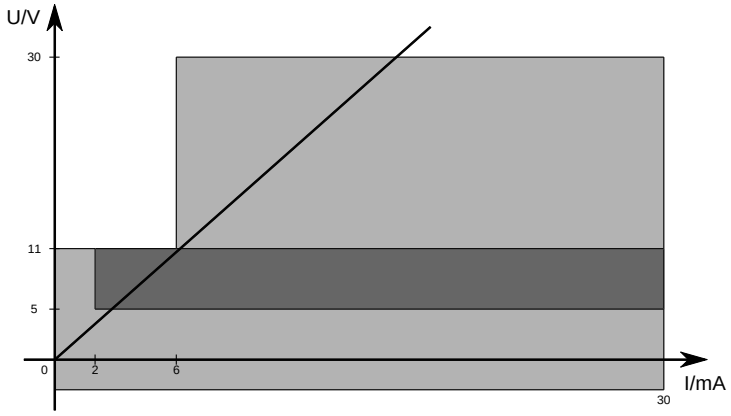


Abbildung 7: Arbeitsbereiche eines Typ 2 Eingangs
Quelle: Grenzwerte aus [16]

Typ 2 Eingänge fordern dahingegen eine deutlich flachere Kennlinie, wie in Abbildung 7 dargestellt. Damit eignen sie sich besonders gut zum Anschluss von 2-Draht-Sensoren, die einen hohen Stromfluss im Aus-Zustand benötigen, da durch die flache Kennlinie der Aus-Zustand erst später verlassen wird. [16]

Typ 3

Typ 3 Eingänge erfordern ebenfalls eine flachere Kennlinie als Typ 1 Eingänge, allerdings nicht so extrem wie Typ 2 Eingänge. Dadurch ist es möglich diese zum Anschluss von 2-Draht-Sensoren ein-

zusetzen, vorausgesetzt, dass der Strombedarf des eingesetzten Sensors nur gering ist, was bei vielen der Fall ist. [16]

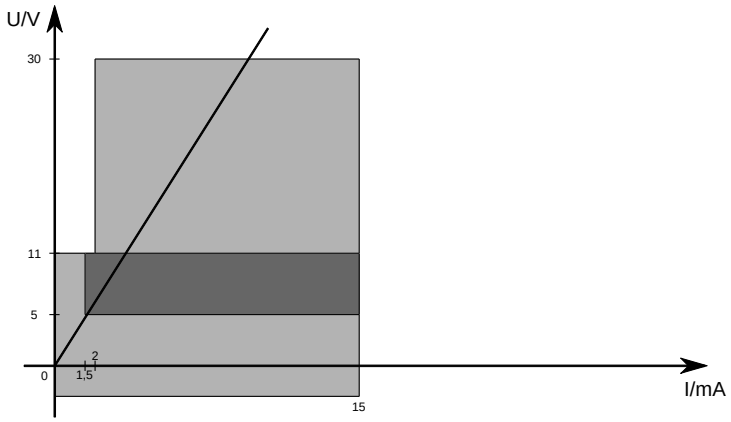


Abbildung 8: Arbeitsbereiche eines Typ 3 Eingangs
Quelle: Grenzwerte aus [16]

Kompatibilität

Im Allgemeinen ist es möglich einen Typ 2 Eingang mit Sensoren und Schaltern, die für einen Typ 1 oder Typ 3 Eingang konzipiert sind, einzusetzen, allerdings ist von einem höheren Energieverbrauch und damit einer höheren Hitzeentwicklung auszugehen. Ein Typ 3 Eingang kann im Allgemeinen mit den gleichen Nachteilen anstelle eines Typ 1 Eingangs eingesetzt werden. [16]

Kennlinienoptimierung

Da die IEC 61131-2 keine genauen Kennlinien für die einzelnen Eingangstypen vorgibt, sondern nur Arbeitsbereiche, kann eine Kennlinie mit möglichst günstigen Eigenschaften für die SPS und die Gesamtanwendung gewählt werden.

An die Kennlinie bestehen im Wesentlichen drei Forderungen. Sie sollte einfach sein, d.h. einfach in Hardware zu realisieren und kein unerwartetes Verhalten aufweisen, also zumindest monoton und stetig sein, sie muss innerhalb der Arbeitsbereiche verlaufen und sollte den Stromfluss so gering wie möglich halten. Den Stromfluss gering zu halten ist wünschenswert um den Energieverbrauch des Gesamtsystems gering zu halten und um die SPS vor übermäßiger Hitzeentwicklung zu schützen und so mehr Eingänge auf einem geringen Raum zu ermöglichen.

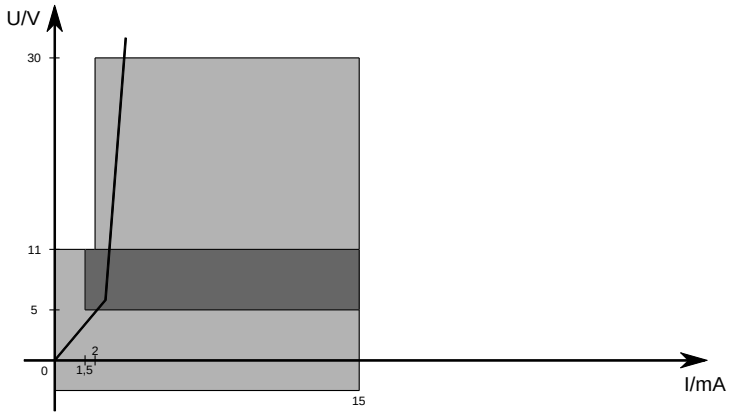


Abbildung 9: Verbesserte Kennlinie eines Typ 3 Eingangs
Quelle: Angeleht an eine Abbildung in [17]

Eine gute Näherung bietet eine lineare möglichst steile Kennlinie, wie in den vorhergehenden Abbildungen exemplarisch dargestellt. Diese ist leicht in Hardware zu realisieren, da hierfür ein einfacher Widerstand genügt, zeigt kein Verhalten, das man von einem Eingang nicht erwarten würde, verläuft, wenn entsprechend gewählt durch die Arbeitsbereiche und hält den Energieverbrauch, so gut es mit einer linieren Kennlinie eben geht, gering.

Der Energieverbrauch ist aber im Ein-Zustand insbesondere, wenn ein Sensor tatsächlich annähert 30 V liefert, unnötig hoch. Daher besitzen qualitativ hochwertigere SPSen eine abgeknickte Kennlinie, wie in Abbildung 9 dargestellt. Diese ist zwar deutlich komplizierter in Hardware zu realisieren, senkt aber erheblich den Energieverbrach, genügt der Norm und zeigt kein Verhalten, dass völlig den Erwartungen widersprechen würde. [17]

Fehlerdiagnose

Eine Herausforderung bei digitalen Schaltern und Sensoren stellt die Erkennung von Ausfällen von Sensoren dar, da beispielsweise eine Unterbrechung der Leitung, verursacht durch einen Kabelbruch, einfach nur jeglichen Stromfluss unterbricht, was ebenfalls einen gültigen Aus-Zustand darstellt. Dadurch kann im Allgemeinen nicht zwischen Aus-Zustand und Ausfall eines Sensors unterschieden werden.

Sind allerdings weitere Informationen über den eingesetzten Sensor bekannt, kann dies durchaus möglich sein. Wird beispielsweise ein 2-Draht-Sensor eingesetzt, wie dieser im Unterabschnitt Halbleiterschaltungen im Abschnitt 2.3.1 beschrieben ist, ist bekannt, dass auch im Aus-Zustand ein geringer Strom über den Eingang der SPS zur gemeinsamen Masse fließt. Ist dieser Stromfluss nicht vorhanden, kann von einem Ausfall des Sensors bzw. einem Kabelbruch ausgegangen werden. Einige SPSen bieten die Möglichkeit dies zu erkennen, was eine schnelle Fehlersuche ermöglicht. Diese Funktion kann auch mit elektromechanischen Schaltern, die im Aus-Zustand keinen Stromfluss verursachen, genutzt werden, wenn der entsprechende geringfügige Stromfluss, z.B. durch einen parallel zum Schalter angebrachten Widerstand, gewährleistet wird. [18]

2.4.2. Digitale Ausgänge

Zur Steuerung der in Abschnitt 2.3.2 beschriebenen „digitalen“ Lasten besitzen SPSen praktisch immer in irgendeiner Form digitale Ausgänge. Anders als bei CMOS Schaltungen, bei denen häufig sog. Push-Pull-Ausgänge zu finden sind, die in der Lage sind sowohl Strom zu liefern als auch Strom aufzunehmen, also den Ausgang auf die Versorgungsspannung hochzuziehen oder auf Masse herunterzuziehen, finden sich bei SPSen üblicherweise Ausgänge, die nur Strom liefern oder welche die nur Strom aufnehmen können. Vergleichbar ist dies mit open-drain Ausgängen bei CMOS Schaltungen, welche zwar Strom aufnehmen, aber nicht liefern können.

Im Unterschied zu Ausgängen in CMOS Schaltungen, welche nur innerhalb eines Geräts zugänglich sind, sind die Ausgänge einer SPS von außen zugänglich und daher mit verschieden Gefahren konfrontiert. Daher besitzen SPS Ausgänge üblicherweise eine ganze Reihe an Schutzfunktionen um sich und die SPS vor Schäden zu bewahren. Üblich sind ein Kurzschluss- bzw. Überlastschutz, ein Überspannungsschutz und ein Verpolungsschutz, wovon die ersten beiden im Folgenden noch genauer betrachtet werden.

Um Kompatibilität zwischen SPSen und verschiedenen Lasten zu gewährleisten sind auch digitale Ausgänge in der IEC 61131-2 Norm standardisiert. Die Anforderungen der Norm werden ebenfalls im Folgenden vorgestellt, wobei immer von der üblichen Betriebsspannung von 24 V, welche laut Norm um -15 % und +20 % abweichen darf, ausgegangen wird.

Kurzschluss- bzw. Überlastschutz

Kurzschluss- bzw. Überlastschutz hat zwei verschiedene Aufgaben. Einerseits soll der Ausgang vor dauerhafter Überlast geschützt werden, welche durch fehlerhaften Anschluss, insbesondere einen Kurzschluss des Ausgangs mit Masse oder Stromversorgung, je nach Ausgangstyp, oder den Anschluss eines zu starken Verbrauchers entstehen kann. Andererseits soll der Ausgang auch vor temporärer Überlast geschützt werden, welche z.B. beim Einschaltvorgang eines Elektromotors entsteht.

Kurzschluss- bzw. Überlastschutz wird entweder durch Sicherungen oder durch Schaltungen realisiert, die den Stromfluss begrenzen. Eine Realisierung mit Sicherungen ist einfach und beim Einsatz von selbstrückstellenden Sicherungen müssen diese nach dem Auslösen nicht gewechselt werden. Allerdings schützen Sicherungen hauptsächlich gegen eine dauerhafte Überlastung. Bei einer temporären Überlast beschränken sie den Stromfluss nicht und lösen evtl. sogar fälschlicherweise aus oder lassen zu, dass eine andere Sicherung etwa im Netzteil auslöst. Für temporäre Überlasten sind strombegrenzende Schaltungen besser geeignet, da diese den Stromfluss auf ein sicheres Maß begrenzen. Dieser Stromfluss kann dann für längere Zeit aufrechterhalten werden ohne die SPS zu überlasten. Problemtisch ist in diesem Fall nur die Erkennung und Behandlung einer dauerhaften Überlast, wofür separate Logik eingesetzt werden muss.

Standardisierung

Die IEC 61131-2 fordert erst mal allgemein für alle digitalen Ausgänge, wie auch schon für digitale Eingänge, dass jeder Ausgang mit einer Vorrichtung ausgestattet sein muss um den Ein-Zustand anzuzeigen. [16]

Darüber hinaus definiert die Norm die elektrischen Eigenschaften für digitale Ausgänge für Bemessungsströme von 0,1 A, 0,25 A, 0,5 A, 1 A und 2 A. Zunächst ist zu bemerken, dass gemäß Norm jeder Ausgang in der Lage sein muss etwas mehr Strom zu liefern, als die Bemessungsgrenze angibt. Beispielsweise muss ein Ausgang, der für 0,5 A ausgelegt ist, 0,6 A liefern können und einer, der für 1 A ausgelegt ist, 1,2 A. Die Norm gibt auch an, wie hoch der Spannungsabfall am Ausgang maximal sein darf. Wie hoch die Spannung am Ausgang also im Vergleich zur Versorgungsspannung mindestens sein muss. Es wird für alle Ausgänge unabhängig von der Bemessungsstromstärke ein Wert von 3 V angeben, nur für den Fall von 1 A und 2 A Ausgängen, die zusätzlich noch mit einem Verpolungsschutz versehen sind, erlaubt die Norm einen Spannungsabfall von bis zu 5 V. [16]

Da es sich bei den Ausgängen einer SPS, wie schon erwähnt, üblicherweise um nur Strom liefernde oder um nur Strom ziehende Ausgänge handelt, verbinden sie nur im Ein-Zustand den angeschlossenen Verbraucher mit Stromquelle bzw. Masse und verhalten sich im Aus-Zustand annähert wie ein offener Schalter. Dies führt z.B. dazu, dass sich an einem unbelasteten Strom liefernden Ausgang, welcher also beispielsweise überhaupt nicht mit einem Verbraucher verbunden oder nur einem sehr hochohmigen Verbraucher, wie einem Spannungsmessgerät, verbunden ist, aufgrund von technisch bedingten Leckströmen auch im Aus-Zustand des Ausgangs zwangsläufig eine Spannung aufbaut. Diese kann sogar annähernd die Höhe der Versorgungsspannung erreichen. Die IEC 61131-2 Norm gibt an, wie hoch der Leckstrom am Ausgang abhängig vom Bemessungsstrom maximal sein darf. Beispielsweise ein 0,5 A Ausgang darf somit maximal 0,5 mA Leckstrom aufweisen, wohingegen ein 1 A Ausgang bis zu 1 mA Leckstrom haben darf. Damit lässt sich problemlos berechnen wie stark eine am Ausgang angeschlossene Last sein muss um eine bestimmte Spannung allein durch Leckströme nicht zu überschreiten. [16]

Wie schon zuvor erwähnt, kann es gewünscht sein einen digitalen Ausgang einer SPS mit einem digitalen Eingang einer anderen SPS zu verbinden. Dies ist, sofern sich beide an die Vorgaben der IEC 61131-2 halten problemlos möglich. Allerdings ist zu bemerken, dass beim Anschluss von 1 A und

2 A Ausgängen an einem Typ 1 Eingang eine zusätzliche Last notwendig ist, da sonst der zulässige Leckstrom schon als Ein-Zustand interpretiert wird. [16]

Überspannungsschutz

Überspannungsschutz bezieht sich in erster Linie auf die Begrenzung der Spannung beim Ausschaltvorgang einer induktiven Last, aber auch ein Schutz gegen elektrostatische Entladungen kann als Überspannungsschutz bezeichnet werden.

Wie schon beschrieben muss die in einer induktiven Last gespeicherte Energie beim Ausschaltvorgang kontrolliert abgebaut werden um den Schalter bzw. Ausgang vor Überspannung zu schützen. Ein Abbauen der Energie in der induktiven Last kann, wie beschrieben, nicht gewünscht sein, weil dies die Schaltzeit erhöht. Wie beschrieben gibt es alternative Schutzschaltungen um die Energie außerhalb der induktiven Last abzubauen, aber da die gespeicherte Energie bei kleinen induktiven Lasten, wie kleinen Relais und Ventilen sehr klein sein kann, ist es oft möglich auf eine externe Schutzschaltung zu verzichten und die Energie in der SPS abzubauen. Dazu muss der Ausgang der SPS eine entsprechende Schutzschaltung bereitstellen. Manche SPSen bieten integrierte Schutzschaltungen, welche bei überschreiten einer bestimmten Spannung einen weiteren Stromfluss zulassen, sich so also gegen Überspannung schützen, aber dennoch durch einen hohen Spannungsabfall am Ausgang viel Energie von der induktiven Last aufnehmen und so den Ausschaltvorgang beschleunigen. Üblicherweise wird bei SPSen angegeben, wie viel gespeicherte Energie sie maximal aufnehmen können. Für größere induktive Lasten ist dann trotzdem eine externe Schutzschaltung erforderlich.

Fehlerdiagnose

Die Hardware einer SPS schützt sich häufig nicht nur gegen Überlast, Kurzschluss und Ähnliches sondern meldet den Fehler auch an die Software, damit diese über entsprechende Handlungsweisen entscheiden kann und dem Nutzer Bericht erstatten kann.

Einige SPSen bieten darüber hinaus noch weitere Diagnosefunktionen. Neben Überlast können so auch eine Überhitzung, das Fehlen einer Versorgungsspannung und sogar das Fehlen einer Last an einem Ausgang in Software erkannt werden. Dabei wird die Erkennung einer fehlenden Last im An- und im Aus-Zustand unterschieden. Im Ein-Zustand kann das Fehlen einer Last am Ausgang relativ einfach anhand des fehlenden Stromflusses erkannt werden. Im Aus-Zustand ist dazu die Erzeugung eines zusätzlichen geringfügigen Stromflusses erforderlich, der so gering ist, dass er über eine angeschlossene Last abfließt ohne diese in den Ein-Zustand zu versetzen bzw. ohne bei dieser eine erkennbare Reaktion auszulösen. Ist keine Last angeschlossen, kann dann das Ausbleiben des Stromflusses erkannt werden. [19]

2.4.3. Analoge Eingänge

Zum Anschluss von analogen Reglern und Sensoren, welche in Abschnitt 2.3.3 beschrieben wurden, bieten viele SPSen analoge Eingänge, an denen die gelieferte Spannung bzw. Stromstärke gemessen und digitalisiert wird um sie in Software verarbeiten zu können. Diese Eingänge sind ebenfalls in der IEC 61131-2 Norm standardisiert. Die Norm gibt verschiedene Spannungs- und Stromstärkebereiche vor und definiert für jeden Bereich die minimale bzw. maximale Eingangsimpedanz, die ein Eingang, der mit dem jeweiligen Bereich arbeitet, haben darf.

Eine Festlegung der minimalen zulässigen Eingangsimpedanz ist für spannungsliefernde Sensoren wichtig, um zu wissen, wie viel Strom der Sensor maximal liefern können muss um bei Belastung mit dem Eingang der SPS die benötigte Spannung aufrechterhalten zu können. Für stromliefernde Sensoren ist eine Festlegung der maximalen Eingangsimpedanz wichtig um zu wissen, wie viel Spannung

der Sensor maximal liefern können muss, um bei Belastung durch den Eingang die benötigte Stromstärke erzeugen zu können.

Definiert sind die Spannungsbereiche von -10 V bis 10 V, von 0 V bis 10 V, sowie von 1 V bis 5 V. Für die ersten beiden sind 10 kΩ als minimale Eingangsimpedanz vorgesehen für letzteren 5 kΩ. Als Strombereiche sind 4 mA bis 20 mA, sowie 0 mA bis 20 mA definiert, wobei letzteres nicht empfohlen wird. Für beide Bereiche wird eine maximale Eingangsimpedanz von 300 Ω festgelegt. [16]

In der Praxis besitzen analoge Eingänge zum Anschluss von spannungsliefernden Sensoren auch deutlich höhere Eingangsimpedanzen, weit über der Minimalforderung der Norm, im MΩ Bereich. Dadurch genügt eine geringere Stromstärke zum Ansteuern des Eingangs. Neben den drei in der Norm definierten Spannungsbereichen finden sich in der Praxis noch eine ganze Reihe weiterer Spannungsbereiche. Besondere Erwähnung verdienen an dieser Stelle Eingänge, welche nur einige hundert mV große Spannungsbereiche aufweisen. Diese werden speziell zum direkten Anschluss von Thermoelementen genutzt, welche der Temperaturmessung dienen und nur eine äußerst geringe Spannung liefern, welche aber sehr genau gemessen werden muss. Vereinzelt werden an SPSen auch Eingänge, die Widerstandsänderungen messen, angeboten. [9] [11] [10]

Innerhalb der definierten Spannungs- bzw. Strombereiche sind keine Werte reserviert um Fehler anzuzeigen. Wird ein Bereich genutzt, der 0 V bzw. 0 mA nicht enthält, kann, wie schon zuvor erwähnt, das Fehlen einer Spannung bzw. Stromstärke als Kabelbruch oder Sensorversagen interpretiert werden. Dies gibt aber einem Sensor nicht die Möglichkeit einen Fehler aktiv zu melden, wenn er diesen selber feststellt. Ein 2-Draht-Sensor, der die Signalleitung ebenfalls für seine Stromversorgung nutzt, kann das Signal nicht auf 0 mA setzen, da er sich dadurch seiner eigenen Stromversorgung berauben würde und so keine Möglichkeit mehr hätte selbstständig wieder seine normale Operation aufzunehmen, wenn der Fehler beseitigt wird. Um eine aktive Fehlerübermittlung zu ermöglichen implementieren einige SPSen und Sensoren die NAMUR NE43 Empfehlung, welche vorsieht, dass ein Strom liefernder Sensor im Fehlerfall, entweder einen Strom unter 3,6 mA liefern soll, sofern für seine Stromversorgung ausreichend oder einen Strom leicht oberhalb von 21 mA. [5, Seite 245]

Analoge Eingänge können auch das im 2.3.5 beschriebene HART Protokoll unterstützen um über die analoge Leitung zusätzlich digitale Informationen auszutauschen. Dies könnte sofern es der eingesetzte Sensor unterstützt zur Fehlerdiagnose oder zur Konfiguration eingesetzt werden. [5, Seite 247]

2.4.4. Analoge Ausgänge

Zum Ansteuern von analogen Lasten, was in erster Linie analoge Eingänge anderer Steuerungen sind, besitzen SPSen häufig analoge Ausgänge. Diese sind ebenfalls in der IEC 61131-2 standardisiert. Definiert sind für analoge Ausgänge dieselben Spannungs- und Strombereiche, wie für analoge Eingänge. Dazu gibt die Norm jeweils die minimale bzw. maximale Impedanz einer Last vor, die an den jeweiligen analogen Ausgang angeschlossen werden darf. Bei spannungsliefernden Ausgängen kann aufgrund der minimalen zulässigen Impedanz einer Last die Stromstärke ermittelt werden, die der Ausgang liefern können muss um bei maximaler Last die nötige Spannung liefern zu können. Bei stromliefernden Ausgängen kann aufgrund der maximalen Impedanz der Last die minimale Spannung ermittelt werden, die der Ausgang liefern können muss um selbst bei maximaler zulässiger Belastung die gewünschte Stromstärke erzeugen zu können.

Für die Spannungsbereiche -10 V bis 10 V und 0 V bis 10 V erlaubt die Norm eine maximale Last von 1000 Ω, d.h. jede angeschlossene Last muss einen Widerstand von mindestens 1000 Ω aufweisen. Für den Bereich 1 V bis 5 V liegt die Untergrenze für die Impedanz bei 500 Ω. Lasten an Strom liefernden Ausgängen dürfen nicht mehr als 600 Ω haben, wodurch sowohl die Anzahl der in Reihe geschalteten

Lasten an einem Ausgang als auch die maximale Leitungslänge für analoge Strom liefernde Ausgänge beschränkt wird.

Erwähnenswert ist auch, dass die IEC 61131-2 für alle analogen Ausgänge einen Überlastschutz fordert. So dürfen Spannungsausgänge selbst bei einem Kurzschluss keinen Schaden davontragen und Stromausgänge müssen auch mit einem Leerlauf zurechtkommen. Ein Leerlauf, also das Erzeugen eines Ausgangssignales ohne Verbraucher stellt für einen Stromausgang die größte denkbare Überlast dar, da dieser versuchen wird die Spannung so lange zu erhöhen, bis die gewünschte Stromstärke erreicht wird, was aber ohne Verbraucher nicht möglich ist. [16]

2.4.5. Feldbusse

Wie im Abschnitt 2.3.5 beschrieben werden zur Kommunikation mit spezialisierten Steuerungen, und auch speziellen Sensoren und Aktuatoren auch Bussysteme, in diesem Bereich speziell Feldbusse genannt, eingesetzt. Die Besonderheit dieser Systeme besteht darin, dass geringe möglichst konstante Verzögerung bei der Kommunikation sichergestellt sein müssen, da beispielsweise die Reaktion auf eine Änderung einer Sensoreingabe innerhalb einer festen Zeitspanne erfolgen muss, da es sonst zu ungewünschtem Verhalten der Anlage kommt. Dies wird genauer im Abschnitt 2.7.2 beschrieben, der sich mit der Echtzeitfähigkeit von Schnittstellen beschäftigt. Für solche zeitkritischen Aufgaben reicht aber meist eine geringe Bandbreite aus, da nur einige Sensordaten ausgetauscht werden müssen. [13, Seite 243]

Ein Bussystem hat im Vergleich zur Nutzung von einfachen digitalen oder analoge Ein- und Ausgängen den Vorteil, dass deutlich weniger Verbindungsleitungen notwendig sind. Es muss beispielsweise nicht von jedem Sensor eine separate Leitung zur SPS geführt werden. Es genügt, wenn alle Sensoren an eine gemeinsame Busleitung angeschlossen werden, die von Sensor zu Sensor läuft und diese dann an die SPS angeschlossen wird. So kann jeder Sensor über die gemeinsame Leitung mit der SPS kommunizieren, wobei allerdings immer nur ein Teilnehmer eines gemeinsamen Busses Daten senden kann. Es sind heute sehr viele verschiedene Feldbussysteme im Einsatz, im Folgenden werden exemplarisch der CAN-Bus, wegen seiner besonderen Behandlung von Kollisionen und PROFINET, als ein Vertreter von Industrial Ethernet, einer Reihe von industriellen Kommunikationsprotokollen, welche auf Ethernet basieren, vorgestellt. Erwähnt werden sollten darüber hinaus das AS-Interface, was speziell zur Anbindung von Aktuatoren und Sensoren entwickelt wurde und der PROFIBUS, der in seinen verschieden Varianten in der Industrie weit verbreitet ist. Auf das Überwachungs- und Konfigurationsnetzwerk wird dann im nächsten Abschnitt eingegangen. [20]

CAN-Bus

CAN (Controller Area Network) ist ein Netzwerkprotokoll auf der Bitübertragungs- und Sicherungsschicht gemäß OSI-Modell, das ursprünglich zum Einsatz in Fahrzeugen entwickelt wurde und ist in der ISO 11898-1 Norm standardisiert. Auf höheren Schichten des OSI Models gibt es eine ganze Reihe Protokolle die auf CAN aufsetzen, Beispiele sind CANopen und DeviceNet, beides Protokolle auf der Anwendungsschicht.

Zur Realisierung eines CAN-Bussystems können auf der physikalischen Schicht verschiedene Verbindungsmedien eingesetzt werden, wobei üblicherweise ein zweiadriges verdrilltes Kupferkabel zum Einsatz kommt. In diesem wird eine symmetrische Signalübertragung eingesetzt, was die Anfälligkeit gegenüber externen Störungen auch bei längeren Signalleitungen gering hält. Mit dem sog. Classical CAN können so Übertragungsraten von bis zu 1 MBit/s bei einer Leitungslänge von bis zu 40 m realisiert werden. Bei geringeren Datenraten sind auch längere Leitungen möglich. Mit dem 2015 standardisierten CAN FD, können Datenraten bis 8 MBit/s erreicht werden.

Ein CAN-Bus wird bevorzugt als Linienstruktur aufgebaut, d.h. mit einer einzelnen Leitung entlang derer Teilnehmer angeschlossen werden. Alle Teilnehmer des Bussystems können alle Nachrichten empfangen und jeder Teilnehmer darf zu jeder Zeit anfangen eine Nachricht zu senden, sofern gerade keine Nachricht übertragen wird. Jeder Nachricht wird eine Priorität zugeordnet. Beginnen mehrere Teilnehmer gleichzeitig mit dem Sender einer Nachricht, darf der, der die Nachricht mit der höchsten Priorität sendet, fortfahren, alle anderen müssen den Sendevorgang abbrechen und warten.

Um dieses Buszugriffverfahren zu realisieren wird der Bitzustand 0 als dominant und der Bitzustand 1 als rezessiv definiert und die Schnittstelle so realisiert, dass ein dominantes Bit ein rezessives überschreibt. D.h. wenn mindestens ein Teilnehmer eine 0 auf den Bus sendet, sehen alle Teilnehmer am Bus die 0 unabhängig davon, ob andere Teilnehmer gleichzeitig eine 1 senden oder nur mitlesen.

Jede Übertragung beginnt mit einer sog. Arbitrierungsphase, dabei wird die Priorität der Nachricht übertragen. Beim Senden von Bits liest der sendende Teilnehmer immer gleichzeitig den am Bus anliegenden Bitzustand. Beginnen mehrere Teilnehmer gleichzeitig zu senden, sendet jeder so lang, wie der am Bus anliegende Bitzustand mit dem von ihm gesendeten Bitzustand übereinstimmt. Sobald dies nicht mehr der Fall ist, d.h. wenn er eine 1 sendet, aber eine 0 sieht, weiß er, dass gleichzeitig ein anderer Teilnehmer sendet, da dieser seinen Bitzustand überschrieben hat. Erkennt ein Teilnehmer dies, muss er die Übertragung abbrechen, so dass der andere Teilnehmer weitersenden kann. Dadurch wird wie Nachricht mit dem dominanten Bit durch die Kollision nicht beschädigt und kann weiter übertragen und von allen empfangen werden.

Dies funktioniert bei beliebig vielen Nachrichten, die gleichzeitig gesendet werden. Unterschiedliche Sender brechen ggf. an unterschiedlichen Stellen ab, jeweils da wo sich ihre Nachricht von der auf dem Bus sichtbaren Nachricht unterscheidet. Offensichtlich gewinnt bei diesem Verfahren immer die Nachricht mit der kleinsten Zahl am Anfang der Nachricht, daher zählt ein kleinerer Wert im sog. Arbitration Field als höhere Priorität.

Die Länge der Nutzdaten pro Nachricht ist auf 8 Byte bzw. 64 Byte bei CAN FD beschränkt und die Übertragung wird durch eine CRC-Prüfsumme abgesichert um Übertragungsfehler zu erkennen. Das System ist objektorientiert, die Priorität einer Nachricht stellt gleichzeitig einen sog. Objekt-Identifier dar und gibt an, welche Daten innerhalb der Nutzdaten enthalten sind, so dass diese Information nicht zusätzlich in den Nutzdaten kodiert sein muss. Aufgrund dieses Objekt-Identifiers entscheiden Teilnehmer auch, ob die Nachricht für sie relevant ist oder nur für andere Teilnehmer bestimmt ist. Teilnehmer werden nicht und können nicht direkt adressiert. Nachrichten mit einem spezifischen Objekt-Identifier können von beliebig vielen Teilnehmern verarbeitet werden. Ein spezifischer Objekt-Identifier darf aber immer nur von einem Teilnehmer des Bussystems beim Versenden verwendet werden, da sonst eine Priorisierung nicht möglich ist. [21] [2, Seite /Controller_Area_Network] [20]

PROFINET

Unter dem Oberbegriff Industrial Ethernet versteht man Bestrebungen das aus dem IT-Bereich bekannte Ethernet für industrielle Kommunikation einzusetzen. Dies ist nicht ohne Weiteres möglich, da die Anforderungen an die Kommunikation in Fertigungsanlagen höher sind als in Büroumgebungen.

Es wird eine besonders hohe Zuverlässigkeit und Störungssicherheit der Übertragung gefordert und alle Komponenten müssen besonders robust konstruiert sein um mechanischen Beanspruchungen standzuhalten. Von besonderer Bedeutung sind aber Echtzeitfähigkeiten. In normalen Ethernet Netzwerken kann es z.B. aufgrund von Kollisionen oder Netzwerküberlastung zu nicht vorhersagbaren Verzögerungen in der Übertragung kommen, die bei der Steuerung von Automatisierungsanlagen nicht toleriert werden können.

PROFINET ist einer von vielen Standards, die entwickelt wurden um industrielle Kommunikation über Ethernet zu ermöglichen. PROFINET stellt zunächst einige grundlegende Forderungen an die Hardware. Es müssen entweder Kupferleitungen oder Glasfaserleitungen mit einer Übertragungsgeschwindigkeit von 100 MBit/s im Voll Duplex Modus eingesetzt werden. Es dürfen keine Hubs eingesetzt werden. Jedes Gerät muss über einen Switch ans Netzwerk angebunden werden, wobei der Switch im Gerät integriert sein kann. Diese Anschlussvariante stellt schon mal sicher, dass keine Kollisionen auftreten können, wodurch ein großer Teil der Nicht-Vorhersagbarkeit von Ethernet Netzwerken wegfällt.

Weitere Festlegungen des Standards betreffen die Software bzw. Firmware der eingesetzten Geräte und die Kommunikation an sich. PROFINET setzt für bestimmte nicht zeitkritische Übertagungen das bekannte UDP/IP ein. Für Echtzeitkommunikation wird hingegen ein spezielles sog. RT Protokoll eingesetzt. Dieses setzt direkt auf dem MAC-Header auf und verzichtet auf einen IP-Header, wodurch unter anderem Protokoll-Overhead gespart wird. Außerdem sind die für PROFINET eingesetzten Switche so konfiguriert, dass sie RT Pakete bevorzugt behandeln, was Verzögerungen durch die Übertragung von anderen Paketen verhindert.

Durch den Verzicht auf einen IP Header sind RT Pakete nicht routingfähig. Sie können also nur innerhalb eines Subnetzes verschickt werden. Dies stellt aber kein Problem dar, da die Echtzeitkommunikation nur lokal innerhalb einer Anlage nötig ist, wo alle betroffenen Geräte in einem Subnetz liegen können.

Für besonders zeitkritische Aufgaben wie Bewegungssteuerung reicht eine Priorisierung von Paketen nicht aus. PROFINET definiert daher für besonders zeitkritische Aufgaben Zeitslots. Jedes Gerät darf dann nur in einem bestimmten Zeitslot senden, was hoch spezialisierte Hardware und eine sehr präzise Synchronisierung aller Geräte erfordert. Ein Kommunikationszyklus wird zudem in einen deterministischen und einen offenen Teil gegliedert. Besonders zeitkritische Nachrichten werden im deterministischen Teil gesendet, während weniger zeitkritische, wie UPD/IP Nachrichten im offenen Teil übertragen werden.

Auch wenn PROFINET auf Ethernet aufbaut, ist der Einsatz von gewöhnlicher Hardware aus dem IT-Bereich leider nicht möglich. Alle Geräte, die PROFINET unterstützen, benötigen eine gesonderte Zertifizierung. Je nach Konformitätsklasse, welche im wesentlichen angibt, welche Zykluszeiten und Abweichungen erreicht werden und so den Einsatzbereich festlegt, genügt entweder eine Zertifizierung durch den Hersteller oder ist eine Zertifizierung durch besondere Prüflabors nötig.

Der wesentliche Vorteil von PROFINET scheint darin zu bestehen, dass neben der zeitkritischen Kommunikation mit RT Paketen auch die Kommunikation mittels UDP/IP und TCP/IP unterstützt wird. Dadurch ist eine einfache Anbindung an das restliche Unternehmensnetzwerk leicht möglich und Automatisierungsgeräte können von Computern im Firmennetz und sogar übers Internet zugegriffen werden. Automatisierungsgeräte können sogar das HTTP Protokoll unterstützen, wodurch der Zugriff mit einem Web-Browser ohne Zusatzsoftware möglich wird. [22]

2.4.6. Überwachungs- und Konfigurationsnetzwerk

Im Unterschied zu den zuvor beschriebenen Feldbussen, bei denen hohe Anforderungen an die Echtzeitfähigkeit bestehen, sind im Überwachungs- und Konfigurationsnetzwerk deutlich höhere auch variable Verzögerungen bei der Kommunikation akzeptabel, da hier keine zeitkritischen Steuerungsaufgaben erfüllt werden müssen. Dafür sind in diesem Netzwerk höhere Bandbreiten gewünscht um auch größere Datenmengen, beispielsweise aufgezeichnete Sensordaten, schnell übertragen zu können. In diesem Bereich werden Ethernet und WLAN basierte Lösungen unter Einsatz von TCP/IP eingesetzt, welches, wie schon zuvor erwähnt, aufgrund seiner Funktionsweise insbesondere im Fall von

Nachrichtenkollisionen auf Datenleitungen oder Überlastsituationen zu nicht vorhersagbaren Verzögerungen führen kann. Diese Verzögerungen sind zwar nicht vorhersagbar aber in der Regel so gering, dass sie in diesem Bereich problemlos akzeptiert werden können. [13, Seite 243]

2.4.7. Schutzmaßnahmen

SPSen arbeiten in störungsreichen Umgebungen, wie schon in Abschnitt 2.3.6 beschrieben, um sich gegen elektromagnetische Störungen zu schützen werden Ein- und Ausgänge von SPSen häufig galvanisch isoliert. In erster Linie erfolgt eine galvanische Trennung der Ein- und Ausgänge von der Recheneinheit, um diese von Störungen, die über die Signalleitungen zur SPS gelangen, zu schützen. Bei einigen SPSen werden auch einzelne Ein- und Ausgänge untereinander isoliert. Ein- und Ausgänge werden dabei entweder einzeln oder in Gruppen isoliert. Dadurch können Störungen an einem Eingang keinen anderen Eingang innerhalb einer anderen Gruppe beeinflussen. Dies hat auch den zusätzlichen Vorteil, dass sich verschiedene Baugruppen auf unterschiedlichen Potentialen befinden können. [18] [17] [2, Seite /Galvanische_Trennung]

Besondere Anforderungen müssen SPSen erfüllen, die in Umgebungen mit explosiven Gemischen arbeiten sollen. Diese müssen sicherstellen, dass sie das Gasgemisch in der Umgebung unter keinen Umständen entzünden können. Um dies sicherzustellen gibt es verschiedene Ansätze. Der einfachste, aber für Wartungsarbeiten ungünstigste Ansatz ist es, die SPS und alle Leitungen in druckdichte Gehäuse einzuschließen, so dass die Umgebungsluft nicht zur SPS und an die Leitungen gelangen kann. Dies macht aber jegliche Wartung im laufenden Betrieb unmöglich. Ein anderer inzwischen aber kaum noch eingesetzter Ansatz besteht in der Benutzung von pneumatischen Signalen zu Übertragung von Sensordaten und Befehlen an Aktuatoren in und aus der Gefahrenzone. Ein technisch anspruchsvollerer aber deutlich benutzerfreundlicherer Ansatz ist es sog. eigensichere Systeme einzusetzen. Diese sind so konstruiert, dass jegliche Funkenbildung und Hitzeentwicklung, die das umgebende Gasgemisch entzünden könnte, verhindert wird und das sowohl im normalen Betrieb als auch im Fehlerfall. Technisch kann dies durch die Begrenzung von Spannung und Stromstärke auf ein sicheres Maß realisiert werden. Dies ist zwar vergleichsweise einfach, beschränkt aber stark die maximale Leistungsaufnahme des Systems. Für Steuerungsgeräte mit höherer Leistungsaufnahme werden aktive Überwachungssysteme eingesetzt, die Spannung und Stromstärke ständig überwachen und beim Erkennen einer Störung, die Stromversorgung rechtzeitig unterbrechen um die Bildung von Funken oder eine Aufheizung zu verhindern. [2, Seite /Dynamic_Arc_Recognition_and_Termination] [2, Seite /Eigensicherheit]

2.5. Programmiersprachen für SPSen

Historisch bedingt werden SPSen im Allgemeinen nicht in den üblichen Hochsprachen, wie C oder C++ programmiert. Die ersten Anwender von SPSen waren Automatisierungsexperten, die oft keinerlei Kenntnisse im Umgang mit Computern, geschweige denn Programmierkenntnisse hatten. Daher wurden spezielle Programmiersprachen entwickelt, die von Elektrotechnikern ohne Programmierkenntnisse verstanden wurden, insbesondere die Darstellung als Kontaktplan. Diese Darstellung und einige weitere spezialisierte Sprachen werden heute immer noch bevorzugt verwendet, obwohl heute Elektrotechniker im Studium durchaus Programmierkenntnisse in Hochsprachen insbesondere C erwerben. Einige Programmiersysteme bieten aber durchaus die Möglichkeit Steuerungssoftware auch in C, C++ und noch einigen weiteren Sprachen zu implementieren. [23] [2, Seite /Speicherprogrammierbare_Steuerung]

Um die Programmierung herstellerübergreifend zu vereinheitlichen wurden die genutzten spezialisierten Sprachen in der IEC 61131-3 standardisiert. Diese Norm hat sich als einziger weltweiter Standard durchsetzen können. Sie definiert fünf Programmiersprachen zur Programmierung von SPSen. Jede davon wird im Folgenden kurz vorgestellt. Zur Erstellung eines Steuerungsprogramms können auch

mehrere Sprachen gleichzeitig eingesetzt werden. Es sind Methoden definiert wie Teile, die in einer Sprache realisiert sind, aus einer anderen Sprache heraus aufgerufen werden können. [24]

2.5.1. Anweisungsliste (AWL)

Die Anweisungsliste, kurz AWL, ist eine textuelle, maschinennahe Sprache. Codeausschnitt 1 zeigt einen kurzen Beispielcode in AWL. Auf den ersten Blick erinnert diese Sprache an Assemblersprachen, wie man sie von PCs kennt.

```
LABEL1: LD    Var1          (* AE = Var1                [INT] *)
        ADD   Var2          (* AE = Var1 + Var2         [INT] *)
        LE    10            (* AE = Var1 + Var2 <= 10   [BOOL] *)
        JMPC  LABEL2        (* AE unchanged             [BOOL] *)
        LD    Var3          (* AE = Var3                [INT] *)
        ADD   5             (* AE = Var3 + 5            [INT] *)
        ST    Var3          (* AE unchanged             [INT] *)
        LD    String1       (* AE = "Summe gross"       [STRING] *)
        JMP   LABEL3        (* AE unchanged             [STRING] *)
LABEL2: LD    String2       (* AE = "Summe klein"       [STRING] *)
LABEL3: ST    StringOut     (* AE unchanged             [STRING] *)
```

Codeausschnitt 1: AWL-Beispielcode

Anweisungen in AWL beginnen, wie im Beispiel zu erkennen ist, mit einer optionalen Sprungmarke, gefolgt von einem Operator, der angibt welche Operation die CPU ausführen soll und endet mit einer vom Operator abhängigen Anzahl an Operanden. Eine Sprungmarke kann auch alleine in einer Zeile stehen. Leerzeilen sind ebenfalls erlaubt. Kommentare werden mit „(*“ eingeleitet und mit „*)“ abgeschlossen und können an beliebiger Stelle stehen, wo ein Leerzeichen zulässig ist. [24, Seite 104 ff.]

2.5.2. Kontaktplan (KOP)

Der Kontaktplan, kurz KOP, ist eine grafische Programmiersprache angelehnt an eine Realisierung der Steuerungslogik mit Relais, wodurch sie auch für Elektrotechniker ohne Programmierkenntnisse besonders einfach verständlich ist. Sie ist besonders zur Verarbeitung boolescher Werte geeignet.

Ein Steuerungsprogramm in KOP wird als eine Reihe von Netzwerken dargestellt, welche üblicherweise, sofern vom Programmierer nicht anders angegeben, von oben nach unten abgearbeitet werden. Jedes Netzwerk wird links und rechts durch Stromschienen begrenzt. Die linke kann man sich als Pluspol einer Stromversorgung vorstellen, die rechte repräsentiert die Masse. Dazwischen werden Kontakte und Spulen angeordnet. Über jedem Element ist ein Variablenname vermerkt. Ein Kontakt ist als Relais zu verstehen, das abhängig vom Variablenwert schließt oder öffnet. Ein normaler Kontakt (Var1, Var3 und Var5 in Abbildung 10) lässt Stromfluss von links nach rechts nur dann zu, wenn der Variablenwert true ist. Ein invertierender Kontakt (Var2, Var4) entsprechend nur wenn der Variablenwert false ist. Eine Spule (Res) dient dem Setzen von Variablenwerten. Sie ist typischerweise immer ganz rechts unmittelbar vor der Masseschiene angeordnet und setzt den Wert der Variable auf true, wenn die links angeordneten Kontakte einen Stromfluss zulassen. Durch Hintereinanderschaltung von Kontakten bekommt am so eine UND-Verknüpfung der Variablen, durch eine Parallelschaltung eine ODER-Verknüpfung. Das in Abbildung 10 dargestellte Netz entspricht somit der Zuweisung: Res := Var1 ∧ ¬Var2 ∧ (Var3 ∨ (¬Var4 ∧ Var5)).

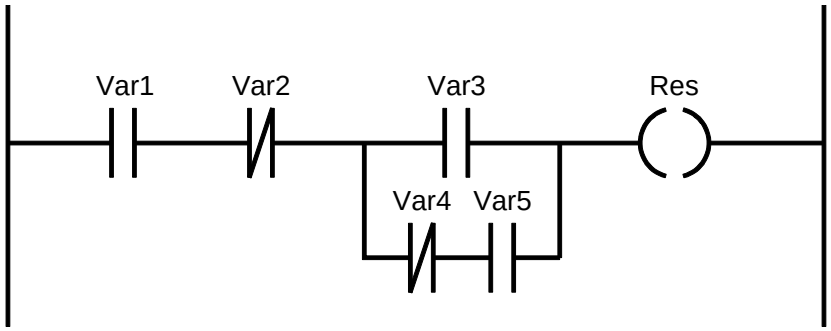


Abbildung 10: Beispiel eines KOP Netzwerks

Es gibt noch weite Elemente, beispielsweise Kontakte, die auf Änderungen der zugehörigen Variable reagieren und z.B. nur schließen, wenn die Variable im letzen Durchgang false war und im aktuellen Auswertungsdurchgang true ist. [24, Seite 152 ff.]

2.5.3. Funktionsbaustein-Sprache (FBS)

Die Funktionsbaustein-Sprache, kurz FBS, auch Funktionsplan, kurz FUP, genannt, ist eine grafische Programmiersprache, die ihren Ursprung im Bereich der Signalverarbeitung hat. In FBS werden Programme grafisch als Netzwerke dargestellt. Ein Programm besteht in der Regel aus mehreren Netzwerken, welche wiederum aus Funktionen, Funktionsbausteinen und Verbindungen bestehen. Abbildung 11 zeigt ein Beispiel für ein solches Netzwerk. Das Beispielnetzwerk realisiert die gleiche Funktion, die im vorhergehenden Abschnitt als KOP Netzwerk implementiert ist.

Funktionen und Funktionsbausteine werden jeweils als Rechtecke dargestellt, wobei links die Eingänge und rechts die Ausgänge dargestellt werden. Ein Ausgang eines Bausteins kann mit einem oder mehreren Eingängen eines oder mehrerer anderer Bausteine verbunden werden um Ausgabewerte des Bausteins an die anderen weiterzuleiten. Über Verbindungen zwischen Bausteinen können Werte beliebigen Datentyps übertragen werden, einschließlich Zeichenketten, wobei der Ausgabetyp mit dem Eingabetyp übereinstimmen muss. Nicht angeschlossene Eingänge können mit Konstanten oder Variablen belegt werden. Ausgängen können Variablen zugewiesen werden, in welche der Ausgabewert abgespeichert werden soll.

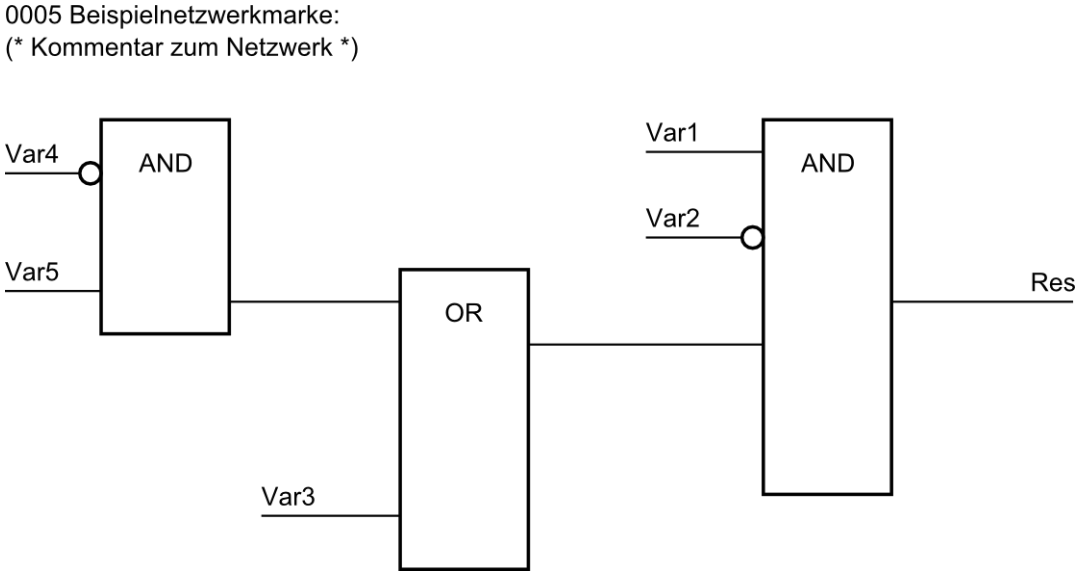


Abbildung 11: Beispiel eines FBS-Netzwerks

Die Reihenfolge der Ausführung einzelner Netzwerke innerhalb des Gesamtprogramms ist herstellerabhängig, allerdings darf die Auswertung eines Netzwerks erst beginnen, wenn alle Eingabewerte bekannt sind. [23] [24, Seite 137 ff.]

2.5.4. Ablaufsprache (AS)

Die Ablaufsprache, kurz AS, ist eine grafische Programmiersprache. Es gibt für AS auch eine textuelle Darstellung, allerdings ist deren Funktionsumfang nur eingeschränkt. AS wird nicht alleine, sondern zusammen mit den anderen Programmiersprachen der IEC 61131-3 verwendet. Sie dient der Definition des Kontrollflusses zwischen Teilen, die in anderen Sprachen geschrieben sind, wobei einzelne Teile zunächst wieder in AS formuliert sein können. Die Ablaufsprache ist besonders geeignet für Prozesse, die in Phasen ablaufen, da der Kontrollfluss, ähnlich wie in einem Zustandsautomaten, mit Zuständen und Übergängen mit Übergangsbedingungen beschrieben wird. Dabei sind auch Verzweigungen möglich, wodurch auch eine Beschreibung von parallelen oder alternativen Ausführungspfaden möglich ist.

Ist ein modellierter Zustand aktiv, wird die Aufgabe, die für diesen Zustand mittels einer anderen Sprache definiert wurde, zyklisch ausgeführt, bis die Bedingung, die für den nachfolgenden Übergang definiert wurde, erfüllt ist. Dann wird der Zustand deaktiviert und der nächste wird aktiviert. Nach Deaktivierung eines Zustandes wird die definierte Aufgabe noch ein letztes Mal ausgeführt. Innerhalb einer Aufgabe kann jeweils abgefragt werden, ob der betreffende Zustand momentan aktiv ist. Damit kann erkannt werden, wenn es sich um die letzte Ausführung handelt und entsprechende Abschlussaufgaben können ausgeführt werden. Dabei ist zu beachten, dass innerhalb einer Aufgabe zeitverzögerte Aktionen gestartet werden können, deren Ausführung dann evtl. zu einer Zeit erfolgen kann, wo der Zustand schon längst inaktiv ist.

Die Norm sieht verschiedene Möglichkeiten vor, die Aufgaben und Übergangsbedingungen in der grafischen Darstellung zu notieren. Je nach Sprache, die für die Aufgabendefinition verwendet wird, kann die Angabe entweder direkt im Diagramm erfolgen oder es werden nur Verweise auf die Implementierung im Diagramm notiert, es können auch beide Varianten zur Verfügung stehen.

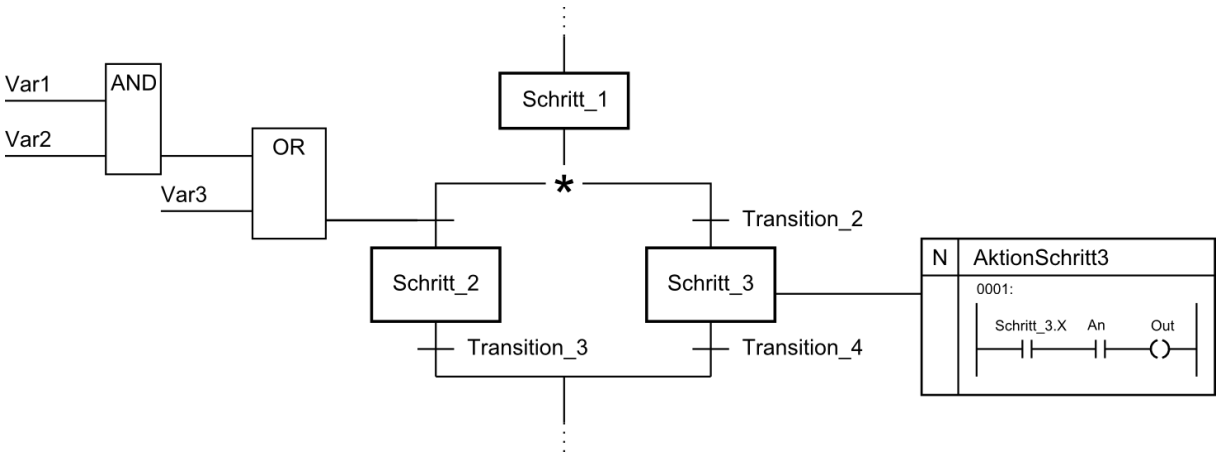


Abbildung 12: Ausschnitt eines AS Programms

Abbildung 12 zeigt einen Ausschnitt aus einem AS Programm. Wenn Schritt_1 aktiv ist, wird zunächst die Transition zu Schritt_2, welche in der Funktionsbaustein-Sprache definiert ist, ausgewertet. Ist die Bedingung erfüllt, wird Schritt_1 deaktiviert und Schritt_2 aktiviert. Sonst wird die Transitionsbedingung zu Schritt_3 ausgewertet. Ist diese erfüllt, wird Schritt_1 deaktiviert und Schritt_3 aktiviert. Die Aktion, die ausgeführt werden soll, solange dieser Schritt aktiv ist, ist hier als Kontaktplan definiert. Die Variable Schritt_3.X gibt dabei an, ob der Schritt aktiv ist. Wenn er aktiv ist, wird

die Variable Out auf den Wert der Variable An gesetzt. Ist Transition_4 irgendwann erfüllt, wird Schritt_3 deaktiviert und ein letztes Mal ausgeführt. Dabei ist der Wert von Schritt_3.X auf false gesetzt, was ein Setzen der Variable Out auf false unabhängig von der Variable An bewirkt. [24, Seite 174 ff.]

2.5.5. Strukturierter Text (ST)

Strukturierter Text, kurz ST, ist wie AWL ebenfalls eine textuelle Sprache, allerdings eine höhere, abstraktere Programmiersprache als AWL, vergleichbar etwa mit der Sprache PASCAL aus dem PC-Bereich. Programme, die in ST geschrieben werden, sind nach der Übersetzung in Maschinencode üblicherweise länger und langsamer als in AWL geschriebene Programme, dafür verfügt ST über mächtigere Kontrollstrukturen, wie IF, WHILE, FOR und CASE, wodurch eine einfachere und übersichtlichere Programmierung möglich ist. Codeausschnitt 2 zeigt einen Programmausschnitt in ST zum Berechnen von Zweierpotenzen. [24, Seite 119 ff.]

```
result := 1;
WHILE exponent > 0 DO
    result := result * 2;
    exponent := exponent - 1;
END_WHILE;
```

Codeausschnitt 2: ST Code zum Berechnen von Zweierpotenzen

2.5.6. IEC 61499

Die IEC 61499 basiert auf der IEC 61131-3 und wurde speziell zur Beschreibung der Struktur und des Verhaltens verteilter Systeme entwickelt. Systeme werden dabei aus Funktionsblocken zusammengesetzt, die miteinander interagieren, von denen jeder aber ggf. auf einem anderen Gerät ausgeführt werden kann. Abbildung 13 zeigt den Aufbau eines solchen Funktionsblocks.

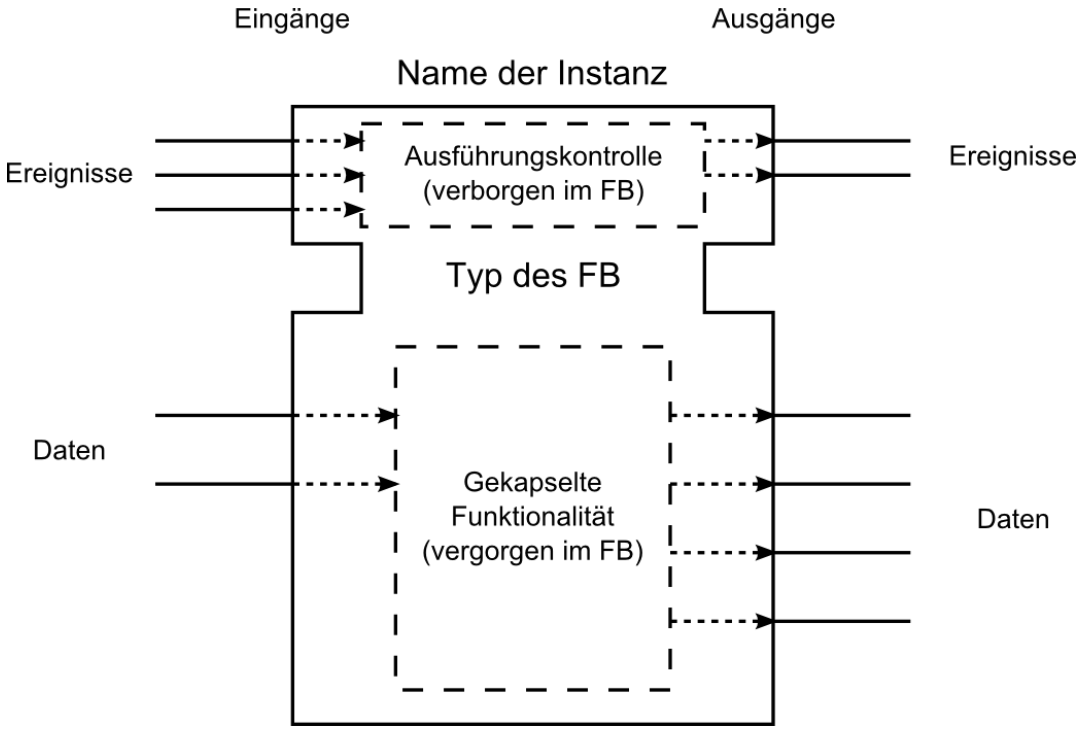


Abbildung 13: Aufbau eines IEC 61499 Funktionsblocks
Quelle: Vereinfachte Version einer Darstellung in [25, Seite 23]

Links am Funktionsblock werden die Eingänge für Daten und Ereignisse, rechts die Ausgänge für Daten und Ereignisse dargestellt. Ein Funktionsblock beinhaltet eine Ausführungskontrolle und gekapselte Funktionalität. Die Ausführungskontrolle, legt für jeden Ereigniseingang fest, welche Funktion aus der gekapselten Funktionalität beim Eintreffen eines Ereignisses auszuführen ist und gleichzeitig wann welche Ereignisse an den Ausgängen erzeugt werden sollen. Ein Funktionsblock kann sofern es für seine Funktion notwendig ist, internen Zustand enthalten.

Während die IEC 61131-3 vorwiegend von einer zyklischen Ausführung der Programme ausgeht, hat die IEC 61499 ein klar ereignisorientiertes Ausführungsmodell. Vor dem Erzeugen eines Ereignisses am Ausgang werden immer zunächst die Daten-Ausgänge aktualisiert, damit liegen an einem Baustein, der die Daten weiterverarbeiten soll und das Ereignis empfängt, die neuen Daten vor Erhalt des Ereignisses bereits an den Eingängen an. Wird ein Ereignis empfangen, werden die Daten an den Eingängen eingelesen und zwischengespeichert und die Ausführungskontrolle benachrichtigt. Diese veranlasst die Ausführung der entsprechenden Funktion des Funktionsbausteins, welche die Eingabedaten verarbeitet und die Daten-Ausgänge aktualisiert. Die Ausführungskontrolle erzeugt dann bei Bedarf ein Ereignis an entsprechenden Ereignisausgängen. Die Ausführungskontrolle ruft dabei die entsprechende Funktion der gekapselten Funktionalität nicht direkt auf, sondern überlässt dies einem Scheduler, wodurch eine Priorisierung von Tasks möglich ist. Eine klassische zyklische Ausführung des Programms innerhalb dieses Ausführungsmodells ist durch einfaches periodisches Erzeugen von Ereignissen modellierbar. [25]

2.6. Betriebssysteme und Laufzeitumgebungen

Von heutigen SPSen wird die Bereitstellung von zunehmend komplexeren Funktionen und Schnittstellen erwartet. Sie müssen genau wie PCs in der Lage sein mehrere priorisierte Tasks parallel oder quasi parallel abzuarbeiten [25] und müssen dem Benutzer einfache Schnittstellen für den Zugriff auf die Hardwarefunktionen, wie Timer oder einzelne Kommunikationsschnittstellen bieten. Eine Ausführung der Steuerungssoftware auf bare metal wäre sehr komplex und jedes Steuerungsprogramm müsste Funktionen für den Hardwarezugriff enthalten. Dies wäre zeitaufwendig und fehlerträchtig in der Entwicklung. Eine Kapselung von gemeinsamen Funktionen, wie Scheduling und Hardwareverwaltung ist erforderlich, was den Einsatz eines Betriebssystems, welches von der Hardware abstrahiert und Funktionen wie Scheduling, Speicherverwaltung, Interruptbehandlung und Kommunikation bereitstellt, nahelegt.

Da SPSen sehr spezifische Anforderungen an ihr zeitliches Verhalten (vgl. Abschnitt 2.7.1) erfüllen müssen, ergeben sich besondere Anforderungen an die eingesetzten Betriebssysteme hinsichtlich ihrer Echtzeitfähigkeit. Bei modularen, Kompakt- und Slot-SPSen werden üblicherweise proprietäre Echtzeitbetriebssysteme, welche auch die nötige Laufzeitumgebung enthält um Programme, die in den IEC 61131-3 Sprachen geschrieben sind, ablaufen zu lassen, eingesetzt. Bei PC-basierten Systemen werden oft Windows oder Linux, üblicherweise mit speziellen Echtzeiterweiterungen, die das zeitliche Verhalten modifizieren, zusammen mit SPS-Laufzeitumgebungen eingesetzt. Für besonders zeitkritische Steuerungsaufgaben werden auf PC-basierten Systemen häufig zwei parallel laufende Betriebssysteme eingesetzt, wobei eines speziell für zeitkritische Steuerungsaufgaben zuständig ist, während das andere beispielsweise nicht zeitkritische Visualisierungsaufgaben übernimmt. Die verschieden Möglichkeiten werden im Folgenden vorgestellt.

2.6.1. Echtzeitbetriebssysteme

Im PC-Bereich, bei Notebooks, Desktop-Computern und Servern ist man üblicherweise an einem möglichst hohen Durchsatz interessiert. Es sollen möglichst viele Aufgaben in gegebener Zeit abgearbeitet werden. Verzögerungen bei der Reaktion auf Ereignisse sind meist nicht von Interesse, insbesondere wenn diese in einer Größenordnung liegen, die vom Benutzer nicht wahrgenommen werden

kann. Beispielsweise soll ein Webserver möglichst viele Webseiten pro Sekunde ausliefern können, ob die Auslieferung einige Millisekunden früher oder später erfolgt, spielt keine Rolle. Fluktuationen bei den Verzögerungen spielen erst recht keine Rolle. Daher sind Nicht-Echtzeitbetriebssysteme wie Windows oder Linux auf hohen Durchsatz zu Lasten von Reaktionszeiten optimiert.

Echtzeitbetriebssysteme sind spezialisiert für zeitkritische Steuerungsaufgaben, bei denen es gerade auf die Reaktionszeiten und deren Fluktuationen ankommt. Sie sind auf möglichst kurze und vor allem konstante und vorhersagbare Reaktionszeiten optimiert, bieten ein deterministisches zeitliches Verhalten und sind speziell dafür entwickelt harte Echtzeitgarantieren (vgl. Abschnitt 2.7.1) geben zu können. Vor allem bei der Reaktion auf Interrupts und beim Taskwechel wird auf geringe Latenzzeiten Wert gelegt. Der Gesamtdurchsatz des Systems ist bei Echtzeitsystemen nur von nachrangiger Bedeutung. Die wesentlichen Unterschiede eines Echtzeitbetriebssystems zu anderen Systemen liegen beim Scheduling, der Interruptbehandlung (siehe Unterabschnitt über Echtzeiterweiterungen) und der Speicherverwaltung.

Ein deutlicher Unterschied beim Scheduling ist das unterschiedliche Verhalten in folgender Situation: Ein Task mit hoher Priorität ist blockiert, weil er auf ein bestimmtes Ereignis wartet. Dieses Ereignis tritt dann auf, während ein niedriger priorisierter Task ausgeführt wird. Ein Nicht-Echtzeitbetriebssystem kann durchaus den Task mit niedrigerer Priorität bis zum Ende einer vorher zugewiesenen Zeitscheibe laufen lassen und erst dann zum Task mit höherer Priorität wechseln. Zeitscheiben haben typischerweise eine Länge von etlichen Millisekunden. Die Reaktion auf ein Ereignis kann bei solchen Systemen problemlos so lange verzögert werden. Durch dieses Verhalten wird die Anzahl an Taskwechseln verringert, wodurch der Overhead reduziert wird und der Gesamtdurchsatz des Systems erhöht wird. Ein Echtzeitbetriebssystem wechselt in dieser Situation immer sofort zum Task mit der höheren Priorität. Dadurch sinkt zwar der Gesamtdurchsatz, aber es werden kürzere, konstantere Reaktionszeiten erreicht.

Bei Anwendungen bei denen es auf das Echtzeitverhalten ankommt, wird üblicherweise immer der Task mit der höchsten Priorität ausgeführt bis dieser fertig ist oder blockiert, weil er auf ein Ereignis wartet. Bei Nicht-Echtzeitbetriebssystemen muss dies nicht der Fall sein. Eine höhere Priorität kann einfach auch nur bedeuten, dass ein Task mehr Zeitscheiben bekommt, also länger am Stück oder öfter als andere Tasks laufen darf.

Die Speicherverwaltung ist ein weiteres kritisches Thema bei Echtzeitanwendungen. Bei nicht-Echtzeitbetriebssystemen kann es relativ lange und vor allem eine nicht vorher bekannte Zeitspanne dauern bis das System einer Anwendung Speicher bereitstellt, wenn diese ihn anfordert. Das liegt daran, dass Anwendungen typischerweise Speicher in verschiedenen Blockgrößen anfordern können. Durch permanentes Anfordern und Freigeben von Speicher durch verschiedene Anwendungen, wird dieser im Laufe des Betriebs zwangsläufig fragmentiert. Je nach Datenstruktur welche vom System verwendet wird um freie Speicherbereiche zu verwalten, kann die Suche nach einem geeigneten Speicherbereich mehr oder weniger Zeit in Anspruch nehmen. Bei Echtzeitanwendungen muss entweder sichergestellt sein, dass die Anwendung während der Zeit, in der zeitkritische Steuerungsaufgaben erledigt werden, keinen zusätzlichen Speicher anfordert oder ein Echtzeitbetriebssystem muss sicherstellen, dass eine Speicheranforderung in kurzer, konstanter Zeit erfüllt werden kann.

Ebenfalls interessant ist die Frage, was zu tun ist, wenn der physikalische Speicher voll ist und dennoch mehr Speicher benötigt wird. Nicht-Echtzeitsysteme lagern dabei nicht benötigte Teile des Speichers auf Datenträger aus und bei Bedarf wieder ein. Dieser Prozess ist als Swapping bekannt. Während dies bei Anwendungen, die keine besonderen Ansprüche an das zeitliche Verhalten haben, gut funktioniert, wäre dieses Vorgehen bei Echtzeitanwendungen fatal. Wenn eine Echtzeitanwendung auf einen Speicherbereich zugreifen würde, der gerade ausgelagert ist, würde sie angehalten werden, bis das System den Speicherbereich eingelagert hat. Da dazu ein Zugriff auf einen Datenträger notwendig

ist, dauert dies sehr lange. Diese Unterbrechung ist bei einer Echtzeitanwendung nicht akzeptabel. Daher kann Swapping bei Echtzeitanwendungen nicht eingesetzt werden. [26, Seite /Real-time_operating_system]

2.6.2. Echtzeiterweiterungen

Wie zuvor beschrieben sind Allzweck-Betriebssysteme wie Windows und Linux nicht für zeitkritische Steuerungsaufgaben entwickelt worden und sind für diese normalerweise nicht geeignet. Um diese aber dennoch für Steuerungsaufgaben einsetzen zu können, wurden verschiedenen Erweiterungen für einzelne Systeme entwickelt, die ihr zeitliches Verhalten modifizieren und ggf. Funktionen hinzufügen um ihnen Echtzeitfähigkeiten zu verleihen. Ein Beispiel für eine solche Erweiterung ist der CONFIG_PREEMPT_RT Patch für Linux, der hier kurz vorgestellt werden soll. Zu beachten ist, dass auch Doppelsysteme, wie sie im nächsten Unterabschnitt vorgestellt werden, oft als Erweiterungen für Windows oder Linux bezeichnet werden.

Der CONFIG_PREEMPT_RT Patch führt eine Vielzahl von Modifikationen durch mit dem Ziel die Reaktionszeiten des Systems zu verkürzen und vor allem ein deterministischeres zeitliches Verhalten zu erzielen. Im normalen Linux-Kernel befinden sich eine Reihe von kritischen Abschnitten, die nicht unterbrochen werden können. D.h. wenn ein Task mit höherer Priorität bereit ist, muss dieser dennoch warten, bis der kritische Abschnitt abgearbeitet wurde. Der Patch modifiziert diese durch Anpassung der eingesetzten Synchronisationsmechanismen um möglichst große Teile des Kernels unterbrechbar zu machen.

Eine weitere Modifikation findet bei der Interruptbehandlung statt, da diese, wie schon zuvor angedeutet, von entscheidender Bedeutung für das zeitliche Verhalten des Systems ist. Wird von der Hardware ein Interrupt ausgelöst, weil beispielsweise neue Eingabedaten zur Verfügung stehen, wird eine Interruptbehandlungsroutine aufgerufen, die den Interrupt gegenüber der Hardware in geeigneter Weise bestätigt, so dass er nicht nach Abschluss der Routine gleich wieder ausgelöst wird und die Eingabedaten verarbeitet. Je nachdem welche Operationen dazu nötig sind, kann dies eine lange, unbestimmte Zeit dauern. Eine Interruptbehandlungsroutine hat gegenüber allen Threads im System Vorrang und kann nicht von diesen unterbrochen werden, egal wie hoch deren Priorität ist. Was dazu führt, dass andere Anwendungen, wie etwa eine Steuerungssoftware für eine Anlage, unvorhersehbar auf unbestimmte Zeit unterbrochen werden, was verheerend für deren Echtzeitverhalten ist. Um dieses Problem zu lösen teilt der Patch Interruptbehandlungsroutinen in zwei Teile auf. Der erste Teil übernimmt nur die Bestätigung des Interrupts, der zweite Teil übernimmt die Verarbeitung der Eingabedaten. Wenn ein Interrupt auftritt wird zunächst nur der erste Teil aufgerufen. Dieser kann wie gewohnt nicht unterbrochen werden, ist dafür aber nur sehr kurz. Der zweite längere Teil wird dann nicht im Interruptkontext sondern in einem Thread ausgeführt. Dieser kann dann mit einer Priorität versehen und von Threads höherer Priorität unterbrochen werden.

Das Scheduling und die Speicherverwaltung, welche auch entscheidender Bedeutung für das Echtzeitverhalten sind, werden vom Patch nicht modifiziert. Das Scheduling des normalen Linux-Kernels bietet bereits passende Konfigurationsmöglichkeiten an. Um die Speicherverwaltung muss sich der Programmierer einer Echtzeitanwendung selber kümmern. Das heißt konkret, dass in zeitkritischen Abschnitten kein neuer Speicher angefordert werden darf und dass für alle genutzten Speicherbereiche das Swapping abgeschaltet werden muss. Außerdem muss der Programmierer sicherstellen, dass bei der Ausführung zeitkritischer Abschnitte keine Seitenfehler im Speicher auftreten. D.h. konkret, dass auf jede angeforderte Seite im Speicher, die im zeitkritischen Bereich benötigt wird, vorher zugegriffen werden muss um das System dazu zu zwingen, den Speicherbereich im physikalischen Speicher anzulegen. [27]

Der CONFIG_PREEMPT_RT Patch versucht das zeitliche Verhalten eines normalen Linux-Kernels weitestgehend dem Verhalten eines Echtzeitbetriebssystems anzugleichen und die maximale Dauer von Verzögerungen zu beschränken. Er ist aber nicht in der Lage in gleichem Maße harte Echtzeitgarantien (vgl. Abschnitt 2.7.1) zu geben, wie dies bei Echtzeitsystemen der Fall. Es besteht nicht der Anspruch das zeitliche Verhalten des Systems mathematisch beweisen zu können, wie dies bei sicherheitskritischen Echtzeitsystemen gefordert sein kann. [28]

2.6.3. Doppelsysteme

Ein anderer Ansatz, welcher besonders für PC-basierte Systeme geeignet ist, die harte Echtzeitanforderungen erfüllen müssen, besteht darin zwei Betriebssysteme parallel auf einem Rechner laufen zu lassen. Dabei wird ein Allzweck-Betriebssystem, üblicherweise Windows oder Linux, mit einem Echtzeitbetriebssystem kombiniert. Das Allzweck-Betriebssystem ist für nicht-zeitkritische Aufgaben, wie die Darstellung einer Benutzeroberfläche mit Visualisierungs- und Steuerungsmöglichkeiten oder die Bereitstellung eines Webservers für einen Zugriff übers Netzwerk zuständig. Das Echtzeitbetriebssystem ist unabhängig vom Allzweck-Betriebssystem und damit unbeeinflusst von dessen schlecht vorhersagbarem zeitlichen Verhalten und übernimmt alle zeitkritischen Steuerungsaufgaben.

Um dies technisch zu realisieren gibt es viele verschiedene Ansätze. Im Wesentlichen basieren aber alle auf demselben Prinzip. Es wird genau wie bei virtuellen Maschinen eine zusätzliche Schicht zwischen Hardware und Betriebssystem eingezogen. Das Allzweckbetriebssystem läuft dann quasi als Gastsystem in einer virtuellen Maschine. Die Zwischenschicht also das Hostsystem ist entweder gleichzeitig das Echtzeitbetriebssystem oder das Echtzeitbetriebssystem läuft als weiteres Gastsystem neben den Allzweck-Betriebssystem. In jedem Fall hat das Echtzeitbetriebssystem immer Vorrang vor dem Allzweckbetriebssystem, wodurch dessen gutes Echtzeitverhalten beibehalten werden kann. Diese Priorisierung zu erzielen ist kein Problem, da das Allzweckbetriebssystem als Gast nach Belieben pausiert werden kann.

Um den Overhead durch die Virtualisierung gering zu halten wird üblicherweise eine Paravirtualisierung eingesetzt, d.h. dass das Allzweck-Betriebssystem nicht unverändert, sondern mit entsprechenden Modifikationen, insbesondere speziellen Treibern aber auch Änderungen im Kernel, ausgeführt wird. Dadurch ist es nicht notwendig das genaue Verhalten der Hardware vollständig nachzubilden, das Gastsystem interagiert vielmehr direkt mit dem Hostsystem, was die Virtualisierung deutlich effizienter gestaltet. Oft wird auch die vorhandene Hardware partitioniert und auf beide Systeme aufgeteilt. Wenn eine Ressource nur von einem System benötigt wird, wenn beispielsweise die Grafikkarte nur vom Allzweckbetriebssystem benötigt wird, kann diesem direkter Zugriff auf diese gewährt werden. Sofern mehrere CPUs bzw. CPU-Kerne zur Verfügung stehen können auch diese exklusiv den einzelnen Systemen zugewiesen werden, der physikalische Speicher kann auch relativ einfach aufgeteilt werden. Die Virtualisierung kann sich im Wesentlichen auch nur auf die Interruptbehandlung beschränken. Es muss in jedem Fall sichergestellt sein, dass das Allzweckbetriebssystem nicht in der Lage ist Interrupts vorübergehend abzuschalten, damit zeitkritische Ereignisse vom Echtzeitbetriebssystem immer verarbeitet werden können. Schaltet das Allzweckbetriebssystem Interrupts vorübergehend aus, müsse die von der Virtualisierungsschicht weiterhin verarbeitet und ggf. an das Echtzeitbetriebssystem weitergeleitet werden, sie dürfen nur vorübergehend nicht an das Allzwecksystem weitergeleitet werden. [29] [26, Seite /RTLinux]

2.6.4. Laufzeitumgebungen

Die Entwicklung von Steuerungsprogrammen für SPSen in den IEC 61131-3 Sprachen erfolgt üblicherweise in einer Entwicklungsumgebung auf einem Windows- oder Linux-PC. Das Programm wird dann in geeigneter Form kompiliert, auf die SPS übertragen und dort von der Laufzeitumgebung ausgeführt. Eine herstellerunabhängige in der Basisversion kostenlose Entwicklungsumgebung, mit der

Geräte verschiedener Hersteller in IEC 61131-3 Sprachen programmiert werden können, ist CODESYS von der 3S-Smart Software Solutions GmbH. Die zugehörige Laufzeitumgebung ist CODESYS Control. Diese ist allerdings kostenpflichtig. Eine kostenlose, quelloffene, aus dem Forschungsbereich stammende Entwicklungsumgebung für Steuerungssoftware nach der IEC 61499 ist 4DIAC. Die zugehörige Laufzeitumgebung ist FORTE. Diese ist ebenfalls quelloffen und kann kostenlos eingesetzt werden.

Wichtige Funktionen der Laufzeitumgebung auf einer SPS beinhalten die Kommunikation mit der Entwicklungsumgebung auf dem PC und die Übertragung neuer Steuerungsprogramme, sowie das Starten und Stoppen von Steuerungsprogrammen. Sie leistet auch wichtige Dienste beim Debuggen von Steuerungsprogrammen, wobei eine Übertragung des aktuellen Zustandes der SPS ans Entwicklungssystem nötig ist. Je nach System kann die Laufzeitumgebung auch deutlich mehr Funktionalität bieten und beispielsweise eine Rekonfiguration des Systems zur Laufzeit unterstützen, wie dies bei FORTE der Fall ist. Die meisten Laufzeitumgebungen unterstützen diese Funktion aber nicht. [2, Seite /Codesys] [30, Seite /products/codesys-runtime.html] [31, Seite /en_rte.php]

2.7. Zeitliches Verhalten

Da bei SPSen das zeitliche Verhalten von entscheidender Bedeutung ist, widmet sich dieser Abschnitt der genaueren Betrachtung des Begriffs Echtzeitfähigkeit. Das Thema wird allgemein und speziell für Schnittstellen betrachtet und es werden die beiden gängigen Ansätze zum Erreichen von Echtzeitfähigkeit vorgestellt.

2.7.1. Echtzeitfähigkeit

Echtzeitfähigkeit beschreibt die Fähigkeit eines Systems bestimmte Anforderungen an sein zeitliches Verhalten erfüllen zu können. Üblicherweise wird gefordert auf ein Ereignis innerhalb einer festen Zeitspanne reagieren zu können oder eine Aufgabe innerhalb einer Zeitspanne erfüllen zu können. Es wird zwischen harten und weichen Echtzeitanforderungen unterschieden.

1. Von harten Echtzeitanforderungen bzw. einem harten Echtzeitsystem spricht man, wann ein nicht-Erfüllen der zeitlichen Anforderungen schwerwiegende Konsequenzen mit sich bringt und als Versagen des Systems gewertet werden muss. Ein Beispiel für ein solches System ist eine Sortieranlage, bei der zu sortierende Objekte zunächst an einer Kamera vorbeilaufen und kurz darauf an einem Gebläse oder Ähnlichem, das alle als fehlerhaft erkannte Objekt aussortiert. Angenommen zwischen Kamera und Gebläse liegen 0,5 Sekunden. Dann hat das System genau diese Zeit um das Bild zu verarbeiten und ggf. das Gebläse einzuschalten. Braucht das System zur Erkennung eines Fehlers am Objekt länger, ist das Objekt bereits weitergelaufen und die Information, dass ein fehlerhaftes Objekt vorliegt, ist nicht mehr hilfreich. Das Objekt kann nicht aussortiert werden, das System hat also versagt. Interessant ist bei diesem Beispiel auch die Beobachtung, dass eine verführte Reaktion des Systems ebenfalls ein Versagen darstellen würde, da dadurch kein oder womöglich sogar das vorhergehende Objekt aussortiert wird. Es gibt also ein festes Zeitfenster in dem die Reaktion erfolgen muss. Systeme werden also harte Echtzeitsysteme bezeichnet wenn sie in der Lage sind die Erfüllung von zeitlichen Vorgaben zu garantieren. Es spielt dabei keine Rolle, wie schnell ein solches System ist. Es können deutlich kürzere oder auch längere Reaktionszeiten gefordert sein, als in diesem Beispiel, sobald es (eine) feste Grenze(n) bezüglich des zeitlichen Verhaltens gibt, die unbedingt erfüllt werden müssen, spricht man von einem harten Echtzeitsystem.

2. Von weichen Echtzeitanforderungen bzw. einem weichen Echtzeitsystem spricht man, wenn es zwar zeitliche Beschränkungen gibt, innerhalb derer eine Aufgabe erfüllt werden muss, es aber keine schwerwiegenden Konsequenzen mit sich bringt, wenn diese Beschränkungen ge-

legentlich verletzt werden. Typische Beispiele für solche Systeme sind Videokonferenzsysteme. Wird bei diesen ein geringer Prozentsatz der Bilder nicht rechtzeitig verarbeitet und kann nicht dargestellt werden, führt dies in irgendeiner Form zu Bildstörungen wird aber nicht als Versagen des Systems interpretiert. Erst bei vielen Ausfällen, wird das System unbrauchbar. Wie viele Ausfälle tolerierbar sind, hängt immer vom konkreten Anwendungsfall eines weichen Echtzeitsystems ab. Das zeitliche Verhalten eines weichen Echtzeitsystems kann durchaus dem eines harten Echtzeitsystems entsprechen, der einzige Unterschied ist, dass im Gegensatz zum harten Echtzeitsystem ein weiches Echtzeitsystem keine Garantien bezüglich seines zeitlichen Verhaltens gibt. [13, Seite 382]

2.7.2. Klassifizierung der Echtzeitfähigkeit von Schnittstellen

Schnittstellen zusammen mit dem jeweiligen Übertragungsprotokoll können entsprechend ihrer Echtzeitfähigkeit klassifiziert werden. Unterschieden werden zunächst nicht echtzeitfähige Schnittstellen und echtzeitfähig Schnittstellen. Echtzeitfähige Schnittstellen werden weiter in drei Klassen unterteilt.

1. Zur Klasse 1 gehören sog. soft real-time fähige Schnittstellen. Diese erfüllen in der Regel Echtzeitanforderungen, können dies aber nicht garantieren. Diese sind nur einsetzbar, wenn sich bei der gelegentlichen Überschreitung von Zeitlimits keine schwerwiegenden Folgen ergeben. Beispielsweise bei den zuvor schon angesprochenen Videokonferenzsystemen können solche Schnittstellen eingesetzt werden. Im Allgemeinen werden die Zeitbedingungen erfüllt und eine gute Ton bzw. Bildqualität sind gegeben. Wird ein Zeitlimit gelegentlich nicht erfüllt, macht sich dies lediglich durch geringfügige, kurzzeitige Ton- und/oder Bildstörungen bemerkbar, was üblicherweise nicht als schwerwiegende Konsequenz zu sehen ist.

2. Zur Klasse 2 gehören sog. hard real-time fähigen Schnittstellen. Diese garantieren eine Erfüllung der Zeitlimits bei jeder Übertragung und sind für zeitkritische Steuerungsaufgaben einsetzbar, wo schwerwiegende Konsequenzen bei Nichterfüllung des Zeitlimits entstehen würden. Schnittstellen dieser Klasse werden typischerweise bei Zykluszeiten von 1 ms bis 10 ms eingesetzt.

3. Zur Klasse 3 zählen sog. isochronous real-time fähigen Schnittstellen. Diese garantieren ebenfalls die Einhaltung von Zeitlimits und zeichnen sich zudem durch besonders kleine Fluktuationen beim Zeitverhalten von typischerweise unter 1 µs aus. Diese werden für besonders zeitkritische Steuerungsaufgaben mit Zykluszeiten von typischerweise 250 µs bis zu 1 ms eingesetzt. [13, Seite 982]

Bewertung der vorgestellten Schnittstellen

Der im Abschnitt 2.4.5 im Unterabschnitt CAN-Bus beschriebene CAN-Bus kann nicht so einfach gemäß seiner Echtzeitfähigkeit klassifiziert werden. Aufgrund seines besonderen Buszugriffsverfahrens hängt das Zeitverhalten von der gesendeten Nachricht ab. Wird eine Nachricht mit der höchstmöglichen Priorität verschickt, ist bekannt, dass diese immer den Vorrang vor allen anderen Nachrichten bekommt. Warten muss diese nur, wenn gerade schon eine Nachricht übertragen wird. Aufgrund der maximalen Länge einer Nachricht im jeweiligen CAN-Bussystem kann so die maximale Wartezeit für die Nachricht höchster Priorität ermittelt werden. Zusammen mit der Nachrichtenlänge der zu sendenden Nachricht, kann so die maximale Zeitspanne ermittelt werden, die vergeht, bis die Nachricht übertragen wurde, immer vorausgesetzt, dass keine Übertragungsfehler auftreten. Für andere hoch priorisierte Nachrichten können mit genauer Kenntnis des betrachteten Systems evtl. noch Aussagen über die maximale Übertragungszeit getroffen werden. Für niedrig priorisierte Nachrichten dürfte eine Aussage bei den allermeisten Systemen nicht möglich sein.

Das im Abschnitt 2.4.5 im Unterabschnitt PROFINET vorgestellte PROFINET kann je nach verwendeter Konformitätsklasse soft real-time, hard real-time oder sogar isochronous real-time Anforderungen mit Zykluszeiten bis runter zu 31,25 µs und Abweichungen von nicht mehr als 1 µs erfüllen. [22]

2.7.3. Ansätze zum Erreichen von Echtzeitfähigkeit

Es gibt zwei verbreitete Ansätze um Echtzeitfähigkeit zu erreichen. Einerseits eine zyklische Abarbeitung des gesamten Steuerungsprogramms und andererseits ein ereignisorientiertes Konzept. Bei beiden Ansätzen ist natürlich stets eine Betrachtung des Gesamtsystems bestehend aus Hardware und Software erforderlich. Insbesondere das Betriebssystem hat einen maßgeblichen Einfluss auf die Echtzeitfähigkeit. Im Folgenden werden nur die beiden Konzepte vorgestellt ohne weitere Betrachtung der Details der Implementierung.

Zyklische Steuerung

Bei einer zyklischen Abarbeitung des Steuerungsprogramms wird zu Beginn eines jeden Zyklus der aktuelle Zustand aller Eingaben einer SPS eingelesen. Danach werden die Eingaben vom Steuerungsprogramm ausgewertet und neue Ausgabewerte werden bestimmt. Danach werden alle Ausgänge mit den neuen Ausgabewerten aktualisiert. Dieses Vorgehen ist sehr verbreitet und hat den Vorteil, dass es nur nötig ist sicherzustellen, dass der zeitlich längste Ausführungspfad durch einen Zyklus des Steuerungsprogramms kürzer ist, als die festgelegte Zyklusdauer. Damit ist garantiert, dass die Berechnungen stets innerhalb eines Zyklus abgeschlossen sind. Durch die feste Zykluszeit ist die maximale Reaktionszeit festgelegt, womit eine Obergrenze für die Reaktionszeit garantiert ist. Es ist natürlich auch möglich eine Ausgabe, die als Reaktion auf eine vorhergehende Eingabe erfolgt, erst nach einer beliebigen festen Zyklusanzahl auszugeben. Damit kann sehr einfach eine Reaktion innerhalb eines Zeitfensters in der Größe der Zykluszeit nach der Eingabeänderung erreicht werden. Ist es für einen Prozess von Vorteil, wenn die Ausgabewerte in präzisen Abständen aktualisiert werden, kann man anstatt die Ausgänge am Ende eines jeden Zyklus zu aktualisieren, diese erst am Anfang des nächsten Zyklus aktualisieren, wodurch ein sehr gleichmäßiges Verhalten unabhängig von der für die Bestimmung der Ausgaben benötigen Rechenzeit erreichet werden kann.

Ereignisbasierte Steuerung

Die Alternative ist eine ereignisbasierte Steuerung. Diese hat keine feste Zykluszeit, sondern reagiert auf Eingabeänderungen. Sobald sich eine Eingabe ändert, wird der neue Wert verarbeitet und die Ausgänge werden aktualisiert. Vorteil dieser Methode ist eine deutlich schnellere Reaktion auf Eingabeänderungen. Erfolgt eine Änderung bei einer zyklisch arbeitenden Steuerung kurz nach Beginn eines Zyklus, wird sie erst zu Beginn des nächsten Zyklus verarbeitet. Die durchschnittliche Verzögerung beträgt also die halbe Zykluslänge. Bei einer ereignisbasierten Steuerung kann die Verarbeitung sofort erfolgen. Eine Verzögerung tritt nur, wenn mehrere Eingabeänderungen in schneller Folge erfolgen und einige daher auf ihre Abarbeitung warten müssen, weil das Steuerungsprogramm noch mit vorhergehenden Änderungen beschäftigt ist. Zu bestimmen, wie hoch die maximale Verzögerung und damit die längste mögliche Reaktionszeit ist erweist sich bei dieser Art der Steuerung als schwieriger. [2, Seite /Echtzeitsystem] [32]

3. Konzeptentwicklung

Um ein Konzept für die Implementierung der geplanten SPS entwickeln zu können, ist es zunächst erforderlich die Zielgruppe und das Umfeld, in dem die Steuerung später eingesetzt werden soll, genau zu beschrieben. Daraus können dann Anforderungen an die zu konstruierenden Steuerung abgeleitet werden. Erst dann kann nach Abwägung der verschiedenen Möglichkeiten der Implementierung und unter Berücksichtigung der aufgestellten Anforderungen eine passende Realisierung für den konkreten Anwendungsfall ermittelt werden. Der Aufbau des vorliegenden Kapitels folgt genau diesem Vorgehen.

3.1. Zielgruppenbeschreibung

Eine gute Kenntnis der Zielgruppe für ein Produkt ist von entscheidender Bedeutung. Je genauer die Zielgruppe beschrieben werden kann umso genauer können im Anschluss die Bedürfnisse der Zielgruppe und damit die Anforderungen an das zu entwickelnde Gerät ermittelt werden. Es muss zunächst geklärt werden, wer das Gerät einsetzen soll und wofür er es einsetzen soll. Da hier eine SPS entstehen soll, was je nach Ausführung ein sehr komplexes technisches Gerät ist, sind auch die Vor- bzw. Kenntnisse der Zielgruppe von besonderem Interesse. Es ist auch wichtig sich über die Dauer und die Häufigkeit der Benutzung des Geräts durch einen einzelnen Nutzer bewusst zu sein. Da eine SPS dazu dient mit anderen technischen Geräten zu interagieren müssen auch deren Bedürfnisse und Eigenschaften betrachtet werden.

3.1.1. Nutzer

Die hier zu entwickelnde SPS soll, wie die Aufgabenstellung schon verrät, in erster Linie im Forschungs- und Bildungsbereich zum Einsatz kommen. Damit zählen zu den Nutzern hauptsächlich Studenten an Universitäten bzw. Schüler an Fachhochschulen, Mitarbeiter an selbigen, sowie Forscher an verschiedenen Forschungsinstituten. Explizit nicht Teil der Zielgruppe sind Anwender aus der Industrie, auch wenn SPSen sonst in erster Linie dort eingesetzt werden, aber der Fokus dieser Arbeit liegt klar auf Forschung und Bildung.

Studenten an Unis / Schüler an Fachhochschulen

Einen großen Teil der Zielgruppe stellen Studenten an Universitäten und Schüler an Fachhochschulen in der Fachrichtung Maschinenwesen dar. Diese programmieren zu Übungszwecken SPSen in den IEC 61131-3 Sprachen, sowie in C und steuern damit zur besseren Veranschaulichung meist miniaturisierte Automatisierungsanlagen. Die Programmierung kann teils auch als Prüfungsleistung erfolgen.

Bei diesen Studenten und Schülern sind gute Vorkenntnisse im Bereich von Automatisierungsanlagen, sowie Grundkenntnisse der eingesetzten Programmiersprachen zu erwarten. Grundlegende Kenntnisse im Computerbereich und im Bereich elektrischer Schaltungen dürften meist vorhanden sein, aber es handelt sich um keine Informatiker und auch um keine Elektrotechniker, daher können keine detaillierten Computerkenntnisse oder Kenntnisse elektrischer Schaltungen vorausgesetzt werden.

Die Programmierung der SPSen stellt nur einen kleinen Teil des Studiums dar und erfolgt oft im Rahmen eines Praktikums. Es ist also davon auszugehen, dass sich die meisten Mitglieder dieser Gruppe mit dem System nur ein oder zwei Wochen im Semester, möglicherweise am Stück, beschäftigen. Dieser Teil der Zielgruppe umfasst zwar viele Nutzer, jeder einzelne davon beschäftigt sich mit dem System und damit mit der zu entwickelnden SPS aber nur relativ kurz.

Mitarbeiter an Unis und FHs

Einen weiteren Teil der Zielgruppe stellen Mitarbeiter an Universitäten und Fachhochschulen dar. Diese müssen Übungs- und Prüfungsaufgaben für Studenten und Schüler an den Anlagen vorbereiten. Sie müssen auch die Anlagen und damit auch die zu entwickelnde SPS für die Übungen und Prüfungen vorbereiten und dafür sorgen, dass jeder Student mit den gleichen Anfangsbedingungen beginnen kann. Es ist davon auszugehen, dass ein großer Teil dieser Mitarbeiter auch selber Forschung betreibt. Im Zuge dieser könnte ein Mitarbeiter die SPS z.B. zum Testen von neuen Steueralgorithmen einsetzen. Dabei könnte eine Automatisierungsanlage, wie bei den Übungen der Studenten zur Veranschaulichung angeschlossen werden. Alternativ könnte der Wunsch bestehen ein zweites Gerät zur Simulation einer solchen Anlage anzuschließen. Bei dem zweiten Gerät könnte es sich ebenfalls um die hier zu entwickelnde SPS oder um eine andere SPS handeln. Die vorhandenen miniaturisierten Automatisierungsanlagen werden ebenfalls zu Demonstrationszwecken beispielsweise an Tagen der offenen Tür eingesetzt. Auch in diesem Fall könnte die zu entwickelnde SPS von Mitarbeitern der Universität oder Hochschule zur Steuerung der Anlagen eingesetzt werden.

Bei diesem Teil er Zielgruppe können fundierte Kenntnisse über Automatisierungslangen, IEC 61131-3 Programmiersprachen, sowie weitere Programmiersprachen, insbesondere C, C++ und Java vorausgesetzt werden. Die Mitarbeiter haben üblicherweise langjährige Erfahrungen im Umgang mit Computern. Allerdings handelt es sich auch hier meist nicht um Elektrotechniker, weshalb nur begrenzte Kenntnisse im Bereich der Elektrotechnik zu erwarten sind.

Im Unterschied zu Studenten und Schülern ist bei Mitarbeiter von einem häufigen Einsatz der zu entwickelnden SPS auszugehen und vor allem auch von einem längerfristigen Einsatz. Das Gerät dürfte über viele Jahre hinweg, wenn auch nicht jeden Tag, dann aber zumindest mehrere Male pro Semester eingesetzt werden.

Forscher

Forscher im Bereich der industriellen Automatisierung an Universitäten und Forschungsinstituten sind der heterogenste und am schwersten zu beschreibende Teil der Zielgruppe. Auch diese setzen oft miniaturisierte Automatisierungsanlagen zur Veranschaulichung von Steuerungsalgorithmen ein. Allerdings ist hier von einer deutlich größeren Vielfalt an Geräten, mit denen die zu entwickelnde SPS potenziell interagieren soll, auszugehen. Während der Fokus bei Studenten und auch bei den zuvor beschriebenen Mitarbeitern meist auf der Entwicklung von Steuerungssoftware in einer der IEC 61131-3 Sprachen oder einer Hochsprache liegt und die darunterliegende Laufzeitumgebung kaum eine Rolle spielt, können im Bereich der Forschung insbesondere auch Änderungen an einer Laufzeitumgebung oder am ggf. darunterliegenden Betriebssystem gewünscht sein um zu testen, ob sich diese für einen bestimmten Einsatzzweck verbessern ließen.

Bei den einzelnen Mitgliedern von diesem Teil der Zielgruppe ist von sehr unterschiedlichen Kenntnissen auszugehen. In jedem Fall sind umfassende und aktuellste Kenntnisse über Hardware und Software im Bereich Automatisierung vorhanden. Darüber hinaus können praktisch keine Annahmen getroffen werden, da die Kenntnisse sehr stark vom Interessens- und Forschungsgebiet abhängen.

Auch wenn Forscher zahlenmäßig der kleinste Teil der Zielgruppe sein dürften, ist bei ihnen von der intensivsten, womöglich täglichen Nutzung des zu entwickelnden Geräts auszugehen. Sie stellen auch den Teil der Zielgruppe dar, der die höchsten Ansprüche an das Gerät bezüglich Schnittstellen, Anpassbarkeit und zeitlichem Verhalten haben dürfte. Die Ansprüche der einzelnen Teile der Zielgruppe werden bei der Zusammenstellung der Anforderungen (vgl. Abschnitt 3.3.2) behandelt.

3.1.2. Eingesetzte Automatisierungsanlagen

Nachdem die Zielgruppe feststeht, müssen nun die Geräte, die von dieser Zielgruppe eingesetzt werden und mit denen die zu entwickelnde Steuerung interagieren soll, insbesondere die Automatisierungsgeräte, welche gesteuert werden sollen, betrachtet werden. Natürlich werden von verschiedenen Nutzern verschiedenste Geräte eingesetzt. Nach Erfahrung des Autors dieser Arbeit lassen sich die genutzten Geräte in die folgenden drei Kategorien einordnen.

1. Lernsysteme der Firma FESTO
2. Standardkomponenten aus der Industrie
3. Selbst entwickelte Komponenten

Bei Lernsystemen der Firma FESTO handelt es sind um miniaturisierte Automatisierungsanlagen, die speziell zu Lern- und Forschungszwecken gebaut werden. Diese werden in hoher Stückzahl sowohl an Universitäten als auch an Forschungsinstituten eingesetzt. Besonders in Bereich der Forschung können auch Standardkomponenten aus der Industrie, wie sie im Abschnitt 2.3 beschrieben sind, zum Einsatz kommen. Ebenfalls vor allem im Bereich der Forschung werden gelegentlich auch selbst entwickelte Komponenten eingesetzt, wenn dies für einen Test erforderlich ist.

3.2. Bevorzugte Automatisierungsanlagen der Zielgruppe

In diesem Abschnitt werden die eingesetzten Lernsysteme von FESTO genauer betrachtet, die industriellen Standardkomponenten wurden, wie gesagt, bereits zuvor beschrieben, selbst entwickelte Komponenten können so extrem verschieden ausfallen, dass weder eine allgemeine Betrachtung, noch die Betrachtung eines speziellen Geräts an dieser Stelle sinnvoll erscheint.

3.2.1. Aufbau

Die Firma FESTO bietet eine große Vielzahl verschiedener Lernsysteme an. Gerne eingesetzt werden, nach Erfahrungen des Autors dieser Arbeit, die MPS (Modulares Produktions-System) Stationen von FESTO. Es werden verschiedene Stationen angeboten, von denen jede eine spezielle Aufgabe, wie Lagern, Prüfen oder Sortieren von Werkstücke erfüllt. Die Werkstücke werden dabei stets durch verschiedene kleine Plastikzylinder repräsentiert. Die einzelnen Stationen können miteinander kombiniert werden, wodurch komplexere miniaturisierte Produktionsstraßen zusammengestellt werden können. Von Station zu Station werden die Werkstücke üblicherweise von Fließbändern transportiert. Eine Übersicht der verschiedenen gegenwärtig angebotenen Stationen findet sich in Tabelle 1 im nächsten Unterabschnitt.

Jede Station enthält einige Sensoren, insbesondere Lichtschranken, Farbsensoren, Taster und verschiedene Näherungssensoren, mit denen die Position und Art des Werkstücks bestimmt werden kann. Ebenfalls enthalten sind Aktuatoren, wie Motoren zum Bewegen des Fließbandes, Umsetzer zum Aufheben und Platzieren von Werkstücken und Bohrer zum simulierten Bearbeiten von Werkstücken. Es fällt auf, dass sehr viele der Aktuatoren nicht elektrisch, sondern mit Druckluft betrieben werden, welche von einem externen Kompressor bereitgestellt und über elektrische Ventile gesteuert wird. Dies dient mit hoher Wahrscheinlichkeit zur Erhöhung der Sicherheit und zur Erhöhung der Toleranz der Systeme gegenüber Bedienungsfehlern. Besonders starke elektrische Verbraucher werden, wenn elektrisch angetrieben, nicht direkt, sondern über Relais zusammen mit Anlaufstrombegrenzern betrieben. Jede Station hat an ihrer Vorderseite ein Bedienfeld, wo einige Tasten, LED-Anzeigen und auch einige digitale Ein- und Ausgänge in Form von 4 mm Sicherheitsbuchsen zur Verfügung stehen.

3.2.2. Art der Schnittstellen

Von besonderem Interesse für diese Arbeit sind die Schnittstellen, die diese Anlagen zur Steuerung bereitstellen, mit denen also die zu entwickelnde Steuerung interagieren muss um die Anlagen zu kontrollieren. Tabelle 1 zeigt eine Auflistung der verschiedenen angebotenen Stationen mit den jeweils zur Steuerung vorhandenen Schnittstellen. Die Angaben sind aus der Sicht einer SPS, so dass ein Eingang in der Tabelle jeweils einem Signal-Ausgang der Anlage bedeutet und umgekehrt. Werte in Klammern bedeuten, dass die Nutzung eines Eingang bzw. Ausgang optional ist.

Station	digitale Eing.	digitale Ausg.	analoge Eing.	analoge Ausg.
Verteilen/Band	6	4	-	-
Prüfen	8	5	(1)	-
Bearbeiten	8	8	-	-
Handhaben (pneumatisch)	8	5	-	-
Handhaben (elektrisch)	8	7	-	-
Verteilen	7	5	-	-
Pick&Place	7	6	-	-
Fluidic Muscle Presse	8	7	(1)	(1)
Roboter mit Modulen	12	5	-	-
Stanzen	8	8	-	-
Trennen	8	5	1	-
Lagern	8	8	-	-
Sortieren	8	4	-	-

Tabelle 1: Übersicht über verschiedene MPS Stationen mit Schnittstellenbedarf
Quelle: Eigene Zusammenstellung, einzelne Angaben basierend auf [33] und [34]

Es fällt auf, dass analoge Ein- und/oder Ausgaben fast immer optional sind. Dies liegt daran, dass ein Sensor beispielsweise bei der Bestimmung der Höhe eines Werkstücks zwar einen analogen Wert liefert, zur Steuerung der Anlage aber oft nur eine Information darüber benötigt wird, ob die Höhe oberhalb oder unterhalb einer Grenze liegt. Dies tritt auf, wenn die eingesetzten Werkstücke beispielsweise nur zwei verschiedene Höhen haben. In solchen Fällen wird ein Komparator eingesetzt, der den gelieferten analogen Wert mit einem Referenzwert, welcher der Grenze entspricht, vergleicht und dann nur noch ein digitales binäres Signal an die SPS liefert. Optional besteht aber die Möglichkeit das analoge Signal zu einem analogen Eingang einer SPS zu leiten und dort auszuwerten.

Neben digitalen und analogen Ein- und Ausgabesignalen bieten einige weitere Geräte von FESTO, welche nicht zu den MPS Stationen zählen und deswegen an dieser Stellen nicht genauer behandelt werden, auch andere Schnittstellen. Dazu zählen insbesondere das AS-Interface, der PROFIBUS-DP, sowie das PROFINET. Von einigen MPS Stationen werden auch Varianten mit diesen Schnittstellen angeboten. [33]

3.2.3. Anschlusstechnik

Sowohl digitale als auch analoge Ein- und Ausgangssignale werden typischerweise bevor sie die Anlage in Richtung einer Steuerung verlassen auf einem Terminalblock zusammengeführt. Alle Einzelleitungen werden dort mittels Schraubklemmen angeschlossen. Digitale Signale werden dort auf eine 24-polige Centronics Buchse, analoge Signale auf eine 15-polige Sub-D Buchse geführt, von wo aus sie dann mit einem mehradrigen Kabel in Richtung SPS geführt werden. Zu den digitalen Signalen von der Anlage selbst können noch weitere digitale Signale vom Bedienfeld dazukommen.

An die 24-polige Centronics Buchsen mit den digitalen Ein- und Ausgabesignalen werden mehradrige Kabel mit Centronics Steckern angeschlossen. Diese Stecker werden von FESTO konsequent als

„SysLink-Stecker nach IEEE 488“ [33] bezeichnet. Um Missverständnisse zu vermeiden, muss an dieser Stelle erwähnt werden, dass die von FESTO eingesetzten Anschlussvariante nicht die typischen IEEE 488 Stecker mit Schraubbefestigung und der Möglichkeit mehrere Stecker aufeinander zu stapel verwendet. Die Pin-Belegung der verwendeten Stecker entspricht auch nicht dem IEEE 488 Standard und es wird auch nicht das Übertragungsprotokol, welches im IEEE 488 Standard festgelegt ist, verwendet.

Der IEEE 488 Standard definiert ein paralleles Businterface mit 8 Bit Breite, bei dem acht Leitungen für bidirektionale Datenübertragung, drei für Handshakes und fünf für Busverwaltung genutzt werden. Gemäß Standard besitzt jedes Gerät eine Adresse und es können bis zu 15 Geräte einen physikalischen Bus gemeinsam nutzen, wozu sie einfach parallel an den Bus angeschlossen werden. Aus diesem Grund bieten die typischen IEEE 488 Stecker auch die Möglichkeit einen weiteren Stecker von hinten auf einen Stecker zu stecken um eine einfache Weiterführung des Busses von einem Gerät zum nächsten zu ermöglichen. [26, Seite /IEEE-488]

Die digitalen Schnittstellen von FESTO sind aber für eine Punkt-zu-Punkt Kommunikation ausgelegt und bilden kein Bussystem. Die von FESTO eingesetzte Pin-Belegung für die Stecker ist in Tabelle 2 zusammengefasst. Wie zu erkennen ist, gibt es nur Daten- und Versorgungsleitungen, keine Leitungen für Busverwaltung und Handshakes. Dies macht auch Sinn, da diese Schnittstelle ausschließlich zur Bereitstellung von digitalen Ein- und Ausgängen, wie sie in den Abschnitten 2.4.1 und 2.4.2 beschrieben sind, dient.

Pin	Funktion	Pin	Funktion
1	Ausgang 0	13	Eingang 0
2	Ausgang 1	14	Eingang 1
3	Ausgang 2	15	Eingang 2
4	Ausgang 3	16	Eingang 3
5	Ausgang 4	17	Eingang 4
6	Ausgang 5	18	Eingang 5
7	Ausgang 6	19	Eingang 6
8	Ausgang 7	20	Eingang 7
9	Versorgung 24 V	21	Versorgung 24 V
10	Versorgung 24 V	22	Versorgung 24 V
11	Masse	23	Masse
12	Masse	24	Masse

Tabelle 2: Pinbelegung von Centronics-Steckern bei FESTO-Geräten aus Sicht einer Steuerung
Quelle: Belegung gemäß [35]

Analoge Signale werden auf 15-poligen, zwei-reihigen D-Sub-Buchsen bereitgestellt. Die von FESTO verwendete Pin-Belegung dieser Buchsen ist in Tabelle 3 zusammengefasst.

Pin	Funktion	Pin	Funktion
1	Ausgang 0	9	-
2	Ausgang 1	10	-
3	Masse	11	10 V Referenz
4	-	12	-
5	-	13	-
6	Masse	14	Eingang 3
7	Eingang 1	15	Eingang 4
8	Eingang 2		

Tabelle 3: Pinbelegung von Sub-D Steckern bei FESTO-Geräten aus Sicht einer Steuerung
Quelle: Belegung gemäß [35]

3.2.4. Steuerungen

Gesteuert werden die miniaturisierten Automatisierungsanlagen entweder mit industriellen modulare oder Kompakt-SPSen von verschiedenen Herstellern, wie sie im Abschnitt 2 beschrieben sind, oder von einem PC aus. Da industrielle SPSen üblicherweise Schraub- oder Federkraftklemmen zum Anschluss von digitalen und analogen Signalleitungen besitzen, werden von FESTO Einbaurahmen für verschiedene Steuerungen angeboten, die Adapter enthalten. Die Leitungen werden entweder auf 4 mm Sicherheitssteckbuchsen oder auf die schon zuvor beschriebenen 24-poligen Centronics Buchsen für digitale Signale bzw. 15-poligen Sub-D Buchen für analoge Signale gelegt, so dass ein Kabel, das auf beiden Seiten Centronics Stecker bzw. Sub-D Stecker hat, zum Anschluss der Anlage genutzt werden kann.

Erfolgt die Steuerung von einem PC aus, wird üblicherweise das sog. Easy Port USB Gerät von FESTO zur Kommunikation zwischen Anlage und PC eingesetzt. Dabei handelt es sich um ein kleines Gerät, das 16 digitale 24 V Ausgänge, 16 digitale 24 V Eingänge, zwei analoge Ausgänge und vier analoge Eingänge bereitstellt, die vom PC aus geschaltet bzw. eingelesenen werden können. Der Anschluss am PC erfolgt wahlweise über USB oder eine serielle Schnittstelle. Die Datenübertragung erfolgt mit Hilfe eines ASCII basierten Protokolls. Die Ein- und Ausgänge zur Automatisierungsanlage sind bereits als zwei Centronics-Stecker und ein Sub-D Stecker ausgeführt um einen bequemen Anschluss zu ermöglichen. Die Schnittstellen zum PC besitzen eine galvanische Trennung, ansonsten arbeitet das gesamte Gerät, d.h. auch alle Ein- und Ausgänge mit einer gemeinsamen Masse. Um Störungen an den Eingängen zu unterdrücken, haben alle Eingänge einen Tiefpassfilter mit einer Filterlänge von 5 ms. Als Betriebssystem auf dem PC wird von FESTO gegenwärtig nur Windows unterstütz. [35]

3.2.5. Zeitliches Verhalten

In den MPS Stationen werden Werkstücke identifiziert, kontrolliert, (simuliert) bearbeitet, zusammengebaut und fortbewegt. Auf dem ersten Blick wäre zu erwarten, dass die Steuerung dieser Vorgänge sehr präzise zeitliche Abläufe erfordert. Bei genauerer Analyse der Stationen stellt sich allerdings heraus, dass diese so geschickt konstruiert sind, dass es kaum zeitliche Bedingungen gibt, die bei der Steuerung eingehalten werden müssen.

Beispielsweise werden alle Werkstücke bei jeder Identifikation oder Kontrolle zunächst mechanisch angehalten. Dann erfolgt eine Untersuchung des Werkstücks durch verschiedene Sensoren. Dabei bewegt sich das Werkstück nicht. Die Untersuchung kann beliebig lange dauern. Ist das Werkstück identifiziert bzw. kontrolliert, wird der Weg freigegeben und das Werkstück kann auf dem Fließband weiterfahren oder es wird von irgendeinem Aktuator bewegt.

Aktuatoren werden meist nur von einer Endlage zur anderen Endlage bewegt, in denen sie jeweils mechanisch ggf. mit Stoßdämpfern gestoppt werden. Es gibt zwar praktisch immer Endlagesensoren zur Kontrolle der Endlage, aber es ist in den allermeisten Fällen nicht notwendig die Bewegung eines Aktuators als Reaktion auf eine Sensoreingabe zu stoppen. Dies geschieht mechanisch oder die Endlagensensoren sind in Hardware bereits so verdrahtet, dass sie ein Stoppen der Bewegung selbstständig veranlassen. Das Signal des Endlagensensors wird an die SPS nur weitergeleitet, um anzuzeigen, dass die Position erreicht ist und dass jetzt die nächste Aktion eingeleitet werden kann.

Im nächsten Abschnitt wird zunächst die geschickte Vermeidung von zeitkritischen Steuerungsaufgaben am Beispiel des Pick&Place Vorgangs illustriert. Danach folgt die Betrachtung einiger Situationen, in denen zeitliche Bedingungen ausnahmsweise eine Rolle spielen.

Pick&Place Vorgang

Bei der Pick&Place Station werden zusätzlich zu den Zylindern, die Werkstücke repräsentieren, verschiedene Einsätze verwendet, die in bzw. auf die Zylinder aufgesteckt werden können. Die Pick&Place Station verfügt über einen Vakuumgreifer, der einen Einsatz vom Ende einer Rutsche aufnehmen und ihn auf einen passenden Zylinder, welcher sich auf einem Fließband befindet, aufsetzen kann. Dieser Vorgang erfordert offensichtlich sehr präzise Bewegungsabläufe. Der Einsatz muss präzise aufgenommen und genau platziert werden.

Der Greifer kann sich theoretisch mit bis zu 0,5 m/s bzw. 0,8 m/s [36] pro Bewegungsrichtung fortbewegen. In jede Bewegungsrichtung sind Sensoren angebracht, die das Erreichen der jeweiligen Zielposition melden. Es wäre anzunehmen, dass eine exakte zeitliche Steuerung und eine sehr schnelle Reaktion auf das Erreichen der Zielposition erforderlich sind um den Greifer mit der nötigen Präzision zu positionieren.

Bei genauerer Analyse der Station stellt sich aber heraus, dass aufgrund der geschickten Konstruktion der Anlage keine präzise zeitliche Steuerung erforderlich ist. Der Greifer kann sich nur in zwei Richtungen bewegen. Er kann sich vor und zurück, sowie hoch und runter bewegen. Das aufzunehmende Werkstück befindet sich am Ende einer abschüssigen Rutsche und damit immer ziemlich genau an derselben Stelle. Der Zylinder wird durch eine Aktuator auf dem Fleißband an einer genauen Stelle angehalten. Damit sind die Position, an der der Einsatz aufgenommen werden muss und die Position, an der es Abgesetzt werden muss, immer gleich. Zudem bewegen sich beide Teile für die Dauer der Aktion nicht, wodurch die Aktion beliebig lange dauern kann. Da es nur zwei immer gleiche Positionen gibt, die angefahren werden müssen, wird die Anlage so justiert, dass die beiden Positionen die beiden horizontalen Endlagen des Greifers darstellen. Der Greifer kann nicht über diese Endlagen hinausfahren, beim Erreichen der jeweiligen Endlage wird er mechanisch mit Stoßdämpfern gestoppt. Dadurch ist es nicht notwendig seine Position über Sensoren abzufragen, er wird einfach immer bis zum einen oder anderen Ende gefahren und dort sehr präzise mechanisch gestoppt. Die Bewegungssteuerung in vertikaler Richtung funktioniert genauso. Der Greifer fährt auch immer von Endlage zu Endlage, wo er jeweils mechanisch gestoppt wird. So wird auch ohne präzise zeitliche Steuerung eine hohe Positioniergenauigkeit erzielt. Effektiv unterliegt die Steuerung des Pick&Place Vorgangs keinen zeitlichen Bedingungen.

Drehtisch

In der Station Bearbeiten findet sich ein Drehtisch, der sechs außen liegenden Aufnahmeplätze für Werkstücke aufweist. Steht der Drehtisch befinden sich über mehreren Werkstücken Maschinen, die die Werkstücke Prüfen oder Bearbeiten können. Durch Drehen des Tisches um 60° wird jedes Werkstück jeweils zur nächsten Maschine bewegt. So können immer mehrere Werkstücke gleichzeitig an verschiedenen Maschinen bearbeitet werden, wie man es aus der typischen Fließbandproduktion kennt.

Der Drehtisch kann sich mit 6 Umdrehungen/min drehen, das Erreichen einer Endlage (alle 60°) wird durch einen Sensor angezeigt. [37] Es wäre anzunehmen, dass die Positioniergenauigkeit des Drehtisches von der Reaktionsgeschwindigkeit auf die Meldung des Sensors, dass eine Endlage erreicht wurde, abhängt. Aber auch hier wird durch eine geschickte Konstruktion, diese Abhängigkeit vermieden. Von einer Steuerung wird jeweils nur ein Impuls geliefert, der angibt, dass sich der Tisch anfangen soll zu drehen. Dieser dreht sich dann so lange bis die nächste Endlage vom Sensor gemeldet wird und bleibt dann automatisch stehen. Dieses Verhalten wird durch eine Schaltung aus mehreren Relais erreicht. [34]

Dennoch gibt es hier eine kleine zeitliche Bedingung. Der Impuls, der die Drehung einleitet, darf nicht länger sein als die Zeit, die benötigt wird um die nächste Endlage zu erreichen, da sich der Tisch sonst weiterdrehen würde. Er muss aber lang genug sein um ein Ansprechen des Relais sicherzustellen. Bei einer Drehgeschwindigkeit von 6 Umdrehungen/min und einer Endlage alle 60° dauert das Erreichen der nächsten Endlage jeweils grob 1,7 s, wenn man die Beschleunigung außer Acht lässt.

Lichtschranken

In einigen Stationen werden Lichtschranken eingesetzt um vorbeifahrenden Werkstücke zu erfassen. Im Gegensatz zur Identifizierung und Prüfungen von Werkstücken werden die Werkstücke dabei nicht angehalten. Ein Werkstück hat einen Außendurchmesser von 40 mm und eine Höhe von 22,5 mm oder 25 mm, je nach Werkstück. [37] Das Fleißband bewegt sich üblicherweise mit einer Geschwindigkeit von 9 m/min [38]. Das bedeutet, dass die Lichtschranke von einem flach aufliegenden vorbeifahrenden Werkstück für etwa 0,27 s verdeckt wird.

Station Handhaben

Besonders interessant bezüglich des zeitlichen Verhaltens ist die Station Handhaben. Diese hat einen Greifer ähnlicher der Pick&Place Station, allerdings hat dieser nicht nur 2, sondern 3 Endlagen entlang der horizontalen Bewegungsachse. Die Äußeren werden ebenfalls mechanisch begrenzt. Soll in der Zwischenstellung angehalten werden, welche durch einen Sensor angezeigt wird, muss dies von der Steuerung veranlasst werden. Die Positioniergenauigkeit ist hier direkt abhängig von der Reaktionszeit, die nötig ist um die Sensoreingabe zu verarbeiten und ein Anhalten des Greifers zu veranlassen. Die Hardware bietet dafür keine Hilfestellung. Die Geschwindigkeit des Greifers kann eingestellt werden. Eine genaue Ermittlung der Maximalgeschwindigkeit war nicht möglich, aber basierend auf den Daten ähnlicher Produkte ist von einer Maximalgeschwindigkeit zwischen 1 m/s und 3 m/s auszugehen. [39] [40] Die Bedienungsanleitung sieht eine Positionierung des Endlagensensors mit einer Genauigkeit von einigen Millimetern vor, [41] was nahelegt, dass beim Positionieren eine Abweichung im Bereich weniger Millimeter keine negativen Auswirkungen auf das Verhalten der Anlage hat.

Angenommen, der Greifer würde sich mit seiner Maximalgeschwindigkeit von 3 m/s bewegen, der Endlagensensor würde ohne jegliche Verzögerung bei Erreichen der gewünschten Positionen, ein Signal an die Steuerung senden und diese würde ohne jegliche Verzögerung ein Anhalten veranlassen. Die Schaltzeit der Magnetventile, die die Druckluft steuern, die den Greifer vorantreibt, beträgt etwa 20 ms. Es fällt auf, dass bereits während der Schaltzeit der Ventile die Position des Greifers eine Abweichung von ca. 6 cm erreichen würde, was definitiv außerhalb der Toleranz liegt, da die Werkstückgröße nur 4 cm beträgt. Erst dann würde der Bremsvorgang beginnen. Da der Antrieb aus massiven Metallbauteilen besteht, ist von einer nicht unerheblichen Trägheit auszugehen, welche einen langen Bremsweg zur Folge hätte und die Abweichung so nochmals stark erhöhen würde.

Folglich ist es nicht sinnvoll den Greifer mit seiner maximalen Geschwindigkeit zu bewegen. Wenn man beispielsweise annimmt, dass allein durch die Schaltzeit der Magnetventile nicht mehr als 2 mm Ungenauigkeit verursachen werden sollen, ergibt sich eine zulässige Höchstgeschwindigkeit von 0,1 m/s. Bei dieser Geschwindigkeit sollte auch die Trägheit der Konstruktion beherrschbar sein. Möchte man durch die Verarbeitung durch die Steuerung beispielsweise nicht mehr als 25 % der Ungenauigkeit, welche durch die Magnetventile verursacht wird, hinzufügen, also nicht mehr 0,5 mm, wäre eine Reaktion der Steuerung innerhalb von etwa 5 ms erforderlich. Zusammen mit dem Bremsweg sollte so eine Genauigkeit von einigen Millimetern erreicht werden.

Um bei höherer Geschwindigkeit eine sinnvolle Genauigkeit zu erreichen, wäre es denkbar, da bekannt ist, dass bei jedem Anhalten in der Zwischenposition eine Abweichung in Fahrtrichtung auftritt, nach dem Anhalten nochmals in umgekehrter Richtung anzufahren und bei Reaktion des Sensors ste-

hen zu bleiben. Durch die geringere Geschwindigkeit, bedingt durch den kurzen Beschleunigungsweg, könnte so eine geringere absolute Abweichung erreicht werden.

3.2.6. Elektrische Eigenschaften

Da die elektrischen Eigenschaften der eingesetzten Automatisierungsanlagen bei der späteren Anforderungsanalyse von entscheidender Bedeutung sein werden, soll im Folgenden nach der direkten Ermittlung der Eigenschaften, welche auf Schaltplänen und Datenblättern basieren wird, eine Plausibilitätsprüfung durchgeführt werden. Dazu werden die Eigenschaften einiger zur Steuerung empfohlenen Geräte betrachtet. Diese sind gut dokumentiert und es sollte möglich sein, aus deren Eigenschaften Rückschlüsse auf die elektrischen Eigenschaften der Anlagen zu ziehen. Durch diese Rückschüsse können zwar keinen genauen elektrischen Eigenschaften der Anlagen bestimmt werden, es kann aber überprüft werden, ob die zuvor ermittelten Eigenschaften plausible sind.

Direkte Ermittlung der Eigenschaften

Alle MPS Stationen von FESTO arbeiten mit einer Versorgungsspannung von 24 V. Die benutzen Netzteile liefern maximal 4 A bis 5 A. Aus den Schaltplänen der Anlagen ist ersichtlich, dass zur Erzeugung digitaler Ausgaben ausschließlich mechanische Schalter und 3-Draht Sensoren mit PNP Ausgängen eingesetzt werden. [34]

Den Schaltplänen ist zu entnehmen, dass nur sehr wenige verschiedene Bauteile direkt von digitalen Ausgängen der eingesetzten Steuerungen angesteuert werden. Dies sind vor allem Magnetventile, welche extern zugeführte Druckluft kontrollieren, mit welcher die restlichen Aktuatoren angetrieben werden. Tabelle 4 zeigt eine Auflistung der am häufigsten eingesetzten Aktuatoren, mit denen eine SPS mittels digitaler Signale direkt interagiert. Alle Aktuatoren die mittels digitalen Signalen angesteuert werden, sind stets so angeschlossen, dass sie mit Strom liefernden Ausgängen geschaltet werden können. [34]

Aktuator	Nennspannung	Leistung	Schaltzeit Ein / Aus
Magnetventil CPV10-M1H-5LS-M7	21 V	0,5 W	17 ms / 27 ms
Magnetventil CPV10-M1H-2x3OLS-M7	21 V	0,5 W	17 ms / 25 ms
Magnetventil CPV10-M1H-2x3GLS-M7	21 V	0,5 W	17 ms / 25 ms
Anlaufstrombegrenzer 150768	24 V	N/A (Relais)	N/A (max. 1/s)
Gleichstrom-Drehmagnet 665109	24 V	7 W	N/A
Relais	N/A	N/A	N/A
Hubmagnet	24 V	7 W	N/A

Tabelle 4: Eigenschaften verschiedener Aktuatoren
Quellen: Angaben gemäß [42], [43], [44], [45], [46], [34], sowie unzähliger Unterseiten von [37]

Wie aus der Tabelle abzulesen ist, haben die meisten Aktuatoren nur einen sehr geringen Strombedarf. Die mit Abstand am häufigsten in den Stationen vorzufindenden Aktuatoren sind Magnetventile. Alle eingesetzten Ventile nutzen dabei eine indirekte Ansteuerung. Sie nutzen also die Energie der Druckluft, die sie schalten, gleichzeitig zum Schalten des Ventils, wodurch ihre extrem geringere Leistungsaufnahme erklärt wird. Der in einigen Stationen eingesetzte Anlaufstrombegrenzer enthält neben der Strombegrenzungsschaltung gleichzeitig auch ein Relais, über welches die angeschlossene Last geschalten wird. Das Steuersignal schaltet lediglich das Relais. Dessen Energiebedarf ist nicht bekannt, aber da es sich um ein sehr kleines Relais handelt, ist kein hoher Strombedarf zu erwarten. Selbiges gilt für eine Vielzahl von dedizierten Relais, die sich in den Anlagen finden. Einen hohen Leistungsbedarf weißen nur der kaum eingesetzte Gleichstrom-Drehmagnet und der Hubmagnet auf. Diese haben einen Leistungsbedarf von 7 W und benötigt daher einen Stromfluss von fast 300 mA.

Plausibilitätsprüfung

Tabelle 5 vergleicht die Eigenschaften von zwei SPSen von Siemens und dem von FESTO angebotenen Schnittstellengerät Easy Port USB. Diese scheinen geeignet, da sie im Vergleich zu anderen empfohlenen Geräten nur einen geringen Funktionsumfang haben und so tendenziell eine engere Eingrenzung der tatsächlich benötigten Eigenschaften liefern sollten.

SPS	CPU 1214C	CPU 314C-2 DP	Easy Port USB
digitale Ausgänge			
Anzahl	10	16	16
minimaler Strom	N/A	5 mA	0 mA
Nennstrom	0,5 A	0,5 A	N/A
maximaler Strom	0,5 A	0,6 A	0,7 A
maximaler Leckstrom	0,1 mA	0,5 mA	N/A
Summenstrom	N/A	2 A	N/A
digitale Eingänge			
Anzahl	14	24	16
Schaltschwelle	Typ 1 (IEC 61131-2)	Typ 1 (IEC 61131-2)	12 V mit 3 V Hysterese
Eingangsverzögerung	0,2 / 0,4 / ... / 12,8 ms	0,1 / 0,3 / 3 / 15 ms	5 ms
analoge Ausgänge			
Anzahl	0	2	2
Spannung in V	-	0 - 10 / ±10	0 - 10 / ±10
Strom in mA	-	0 - 20 / ±20 / 4 - 20	-
analoge Eingänge			
Anzahl	2	4	4
Spannung in V	0 - 10	±10 / 0 - 10	0 - 10 / ±10
Strom im mA	-	0 - 20 / ±20 / 4 - 20	-

Tabelle 5: Vergleich der Eigenschaften verschiedener SPSen
Quellen: Daten gemäß [47], [48] und [35]

Es liegt nahe davon auszugehen, dass mit allen empfohlenen Steuerungsgeräten der volle Funktionsumfang der Anlagen genutzt werden kann. Alle Typen von Schnittstellen sind bei allen Geräten in ausreichender Anzahl vorhanden um jede der in Tabelle 1 aufgeführten Stationen steuern zu können. Basierend auf den Eigenschaften der digitalen Ausgänge der empfohlenen Steuerungen ist davon auszugehen, dass die Stationen pro Eingang nicht mehr als 0,5 A Stromstärke benötigen und einen Leckstrom von mindestens 0,5 mA tolerieren. Dies deckt sich mit dem zuvor festgestellten maximalen Strombedarf von 300 mA. Auch die Feststellung, dass alle Aktuatoren mit Strom liefernden Ausgängen angesteuert werden, bestätigt sich, da alle aufgeführten Steuerungen genau solche Ausgänge aufweisen.

Als neue Information aus diesem Vergleich ergibt sich, dass alle analogen Sensoren und Aktuatoren der Stationen offensichtlich Spannungswerte im Bereich von 0 V bis 10 V bzw. -10 V bis 10 V zur Signalübertragung von und zur SPS nutzen. Die abweichenden Eigenschaften bei den analogen Ein- und Ausgängen der ersten aufgeführten SPS können nicht herangezogen werden, da diese zusammen mit einer Erweiterung, die die analogen Funktionen ausbaut, eingesetzt wird.

Die Eigenschaften der digitalen Eingänge bestätigen die Feststellung, dass ausschließlich mechanische Schalter und 3-Draht-Sensoren mit Strom liefernden Ausgängen eingesetzt werden, da nur für diese Typ 1 Eingänge gemäß 61131-2 geeignet sind. Für 2-Draht-Sensoren wären Typ 2 oder Typ 3 Eingänge nötig.

3.3. Zusammenstellung der Anforderungen

In diesem Abschnitt werden basierend auf den vorhergehenden Erkenntnissen die Anforderungen an die zu entwickelnde SPS zusammengetragen. Die aufgestellten Anforderungen können sich dabei durchaus auch widersprechen, wenn sie sich aus verschieden Gründen ergeben. Die Lösung ggf. auftretender Konflikte und der Entwurf eines Systems, das die Anforderungen in hinreichender Weise erfüllt, wird im Abschnitt 3.4 diskutiert.

3.3.1. Automatisierungsanlagen

Als erstes sollen die Anforderungen betrachtet werden, welche die SPS erfüllen muss, um die eingesetzten Automatisierungsanlagen steuern zu können. Zunächst beschränkt sich die Betrachtung auf die eingesetzten Lernsysteme. Danach werden zusätzliche Anforderungen aufgeführt, die sich aus der Nutzung von industriellen Komponenten und Eigenentwicklungen ergeben.

Art der Schnittstellen

Es sollte möglich sein mit der zu entwickelnden SPS jede der vorgestellten MPS Stationen zu steuern. Gemäß Abschnitt 3.2.2 sind dazu mindestens 12 digitale Eingänge und mindestens 8 digitale Ausgänge erforderlich. Dazu kommen digitale Ein- und Ausgänge für das Bedienfeld, welches vorne an den Stationen angebracht ist. Darüber hinaus sind mindestens ein analoger Eingang und ein analoger Ausgang erforderlich. Um auch andere Lernsysteme steuern zu können sind das AS-Interface, sowie Feldbusschnittstellen insbesondere der PROFIBUS-DP und PROFINET vorzusehen.

Elektrische Eigenschaften

Wie in Abschnitt 3.2.6 festgestellt, nutzen alle betrachteten Stationen entweder mechanische Schalter oder 3-Draht-Sensoren mit PNP Ausgängen zur Erzeugung digitaler Ausgaben von etwa 0 V bzw. etwa 24 V. Daher sind Strom aufnehmende Eingänge, deren Schaltpunkt zwischen diesen Grenzen liegt, erforderlich.

Basierend auf den Erkenntnissen aus Abschnitt 3.2.6 sind Strom liefernde Ausgänge erforderlich. Es muss möglich sein etwa 24 V und mindestens 0,3 A pro Ausgang zu liefern. Die Ausgänge müssen gegen Überspannung, welche bei Abschaltung induktiver Lasten entsteht, geschützt sein und temporäre Überlastsituationen, welche durch Anlaufströme entstehen, aushalten, da induktive Verbraucher und potenziell Elektromotoren geschaltet werden müssen.

Um mit den eingesetzten analogen Sensoren und Aktuatoren interagieren zu können, sind analoge Ein- und Ausgänge nötig, die Spannungssignale in den beiden Wertebereichen ±10 V und 0 V bis 10 V auswerten bzw. erzeugen können.

Zeitverhalten

Typischerweise bestehen bei der Steuerung von Automatisierungsanlagen harte Echtzeitanforderungen, die erfüllt werden müssen, da sonst ernste Konsequenzen drohen. Im Forschungs- und Bildungsbereich wird aber von häufigen Bedienungsfehlern ausgegangen und die Anlagen sind daher so konstruiert, dass bei einem Versagen, das z.B. auch durch einen Programmierfehler verursacht werden könnte, kein Schaden entsteht. Werden die zeitlichen Anforderungen bei der Steuerung nicht eingehalten, führt dies womöglich zu unbeabsichtigtem Verhalten und ist ärgerlich für den Nutzer, aber es entstehen keine schwerwiegenden Konsequenzen. Daher bestehen hier, genau genommen, nur weiche Echtzeitanforderungen, wie diese im Abschnitt 2.7.1 beschrieben sind. Aus der kürzesten im Abschnitt 3.2.5 ermittelten Reaktionszeit, die benötigt werden kann um die Anlagen zufriedenstellend zu betreiben, ergibt sich die Anforderung auf jede beliebige Eingabeänderung, unabhängig von anderen evtl.

parallel auftretenden Eingabeänderungen, innerhalb von 5 ms reagieren zu können. Es besteht aber keine Notwendigkeit diese Reaktionszeit mathematisch zu beweisen oder anderweitig garantieren zu können, da es keine harte Echtzeitanforderung ist. Es muss nur sichergestellt sein, dass die nötige Reaktionszeit mit extrem hoher Wahrscheinlichkeit vom System erfüllt werden kann um dem Nutzer ein zufriedenstellendes Verhalten bieten zu können.

Störungsunterdrückung

An allen Eingängen sind geeignete Maßnahmen vorzusehen um ggf. auftretende Störungen unterdrücken zu können. Eine Auswirkung von Störungen, die ggf. auf den Signalleitungen und der Spannungsversorgung auftreten können, auf ggf. vorhandene sensible Bauteile und andere Signalleitungen ist zu verhindern. Selbstverständlich müssen alle von außen zugänglichen Leitungen gegen elektrostatischen Entladungen (ESD) geschützt werden, wie dies auch bei jedem anderen elektrischen Gerät erforderlich ist.

Industrielle Komponenten

Da neben den vorgestellten Lernsystemen vor allem im Forschungsumfeld auch industrielle Sensoren und Aktuatoren zum Einsatz kommen können, muss auch mit diesen Kompatibilität gewährleistet werden. Das bedeutet für die digitalen Eingänge, dass neben mechanischen Schaltern und 3-Draht-Sensoren insbesondere auch 2-Draht-Sensoren unterstützt werden müssen und dass das Schaltverhalten den Vorgaben der IEC 61131-2 genügen muss. Für Ausgänge bedeutet dies, dass die gemäß IEC 61131-2 Norm vorgeschriebenen Grenzwerte bezüglich Leckströmen und Spannungsabfällen an Ausgängen eingehalten werden müssen.

Da industrielle Komponenten für analoge Signale statt der Übertragung mittels Spannungswerten eher eine Übertragung mittels Stromstärke nutzen, müssen die analogen Ein- und Ausgänge auch die Wertebereiche 0 mA bis 20 mA und 4 mA bis 20 mA beherrschen.

Wie bei industriellen Steuerungen üblich, ist ein Anschluss von digitalen und analogen Signalen mittels Schraubklemmen wünschenswert.

Eigenentwicklungen

Da nicht alle beliebigen Eigenentwicklungen an Sensoren und Aktuatoren von vorherein unterstützt werden können, ist eine Erweiterungsmöglichkeit vorzusehen, damit die zu entwickelnde Steuerung vor allem im Forschungsbereich durch eigene Schnittstellen erweitert werden kann. Es soll möglich sein mit der Erweiterung Daten mit geringer Latenz und hoher Bandbreite auszutauschen.

3.3.2. Benutzer

Nachdem im letzten Abschnitt die Anforderungen der Automatisierungsanlagen betrachtet wurden, sollen nun die Anforderungen der Nutzer, welche mit den Anlagen arbeiten, genauer untersucht werden.

Schnittstellen

Um dem Benutzer einen einfachen, komfortablen und schnellen Datenaustausch zwischen der zu entwickelnden SPS und einen Desktop-PC oder Notebook, welche er höchstwahrscheinlich für die Entwicklung von Steuerungssoftware einsetzt, zu ermöglichen, sollte das Gerät einen Anschluss für Datenträger aufweisen und auch einen Anschluss, der es dem Gerät ermöglicht kontinuierlich mit einen Desktop PC oder Notebook zu kommunizieren. Eine kontinuierliche Kommunikation ist essentiell

insbesondere für Fernsteuerungs- und Debugging-Aufgaben, wenn Informationen über den Zustand der SPS zur Laufzeit kontinuierlich an eine Entwicklungsumgebung übertragen werden sollen.

Insbesondere im Bildungsbereich besteht bei Übungen und noch viel wichtiger bei Prüfungen die Forderung, dass alle Studenten den gleichen Ausgangszustand vorfinden. Um es den Mitarbeitern so einfach wie möglich zu machen auf allen Geräten den gleichen Zustand herzustellen, muss die gesamte durch den Nutzer änderbare Software auf einem entnehmbaren Datenspeicher untergebracht sein, der mit Hilfe eines PCs einfach ausgelesen, bearbeitet und geklont werden kann.

Um auch Visualisierungsaufgaben mit dem zu entwickelnden Gerät einfach lösen zu können, sollte das Gerät einen Videoausgang haben. Es bietet sich an die visuelle Ausgabe auch zur Realisierung einer Benutzeroberfläche einzusetzen. Daher muss auch eine Eingabemöglichkeit geschaffen werden. Um auch Entwicklungsaufgaben, insbesondere einfache Anpassungen der Steuerungssoftware oder der Konfiguration des Geräts auf diesem selber vornehmen zu können, sind Schnittstellen zum Anschluss geeigneter Eingabegeräte vorzusehen.

Fehlertoleranz

Es ist anzunehmen, dass Studenten, die nur eine kurze Zeit mit dem Gerät arbeiten werden, nicht die Zeit investieren werden um sich mit der Dokumentation des Geräts auseinanderzusetzen. Bedienungsfehler sind bei dieser Gruppe von Nutzern besonders wahrscheinlich und auch bei anderen Nutzern nicht auszuschließen. Somit sind Bedienungsfehler bei diesem Gerät viel wahrscheinlicher als bei industriell eingesetzten SPSen, welche von extra geschultem Personal, häufig mit langjähriger Erfahrung, bedient werden. Daher muss das zu entwickelnde Gerät besonders robust und tolerant gegen Bedienungsfehler gestaltet sein. Es darf nicht möglich sein, das Gerät durch eine falsche Verkabelung zu beschädigen. Das Gerät ist insbesondere gegen Kurzschluss, Verpolung und Überlast zu schützen. Es sind Möglichkeiten vorzusehen um derartige Fehlersituationen erkennen zu können und Fehlerdiagnosen durchführen zu können. Sollten Sicherungen vorgesehen werden, müssen diese einfach austauschbar sein. Es muss auch dafür gesorgt werden, dass es nicht möglich ist das Gerät durch eine falsche Programmierung zu beschädigen. Dies bezieht sich nicht nur auf die Steuerungsprogramme sondern auf die gesamte vom Benutzer änderbare Software. Es darf nicht möglich sein durch irgendeine Softwareänderung die Hardware dauerhaft zu beschädigen.

Zeitliches Verhalten

Im Idealfall sollte die Reaktionszeit des Systems unterhalb der Wahrnehmungsschwelle des Benutzers liegen, damit dieser die Reaktion, z.B. das Einschalten einer LED, auf eine Sensoreingabe, z.B. von einer Lichtschranke, als Gleichzeitig empfindet. Dazu muss die Reaktion auf jede beliebige Eingabeänderung innerhalb von nicht mehr als 10 ms erfolgen. [2, Seite /Echtzeitsystem] [2, Seite /Zeitwahrnehmung]

Es muss möglich sein, das zu entwickelnde Gerät sowohl mit zyklischen als auch mit ereignisbasierten Steuerungsprogrammen zu betreiben, um Forscher bei der Entwicklung von Steuerungsprogrammen nicht zu stark durch die Hardware einzuschränken.

Benutzerfreundlichkeit

Allgemein muss das zu entwickelnde Gerät einfach zu bedienen sein, es muss vor allem möglich sein es schon nach einer sehr kurzen Einarbeitungszeit zu bedienen, was vor allem für Nutzer wichtig ist, die insgesamt nur kurz mit dem System arbeiten.

Es muss ferner möglich sein das neue Gerät einfach in bestehende Systeme zu integrieren, damit dadurch kein zusätzlicher Arbeitsaufwand vor allem auf die Mitarbeiter von Universitäten und Hoch-

schulen zukommt, an welchen die Systeme in hoher Stückzahl vorhanden sind. Dazu gehört auch die Ausführung in einer kleinen Bauform um zusätzlichen Platzbedarf zu vermeiden.

Um die Suche nach Fehlern auch in Zukunft zu erleichtern, sind, wie auch bei den bestehenden Geräten und wie auch von der IEC 61131-2 gefordert für jeden Ein- und Ausgang Leuchtanzeigen vorzusehen, die den aktuellen Status anzeigen. Hinsichtlich der Versorgungsspannung sollte ein möglichst großer Spannungsbereich toleriert werden.

Es kann zu Test- und Simulationszwecken gewünscht sein, zwei SPSen zusammenzuschließen, so dass die Eingänge der einen mit den Ausgängen der anderen und umgekehrt geschaltet werden können. Eine solche Verbindung muss einfach herzustellen sein.

Anpassbarkeit

Besonders im Forschungsbereich wird hoher Wert auf Anpassbarkeit und Erweiterbarkeit gelegt. Um dies zu gewährleisten müssen Pläne bzw. Spezifikationen der entwickelten Hardware, sowie aller genutzten Komponenten und der Quellcode der gesamten zum Betrieb nötigen Software verfügbar sein und öffentlich zur Verfügung stehen bzw. gestellt werden. Es ist insbesondere bei der Hardware eine Möglichkeit zur Erweiterung zu schaffen.

3.3.3. Produktion

Da im Rahmen dieser Arbeit auch ein Prototyp entstehen soll, ergeben sich einige Anforderungen an die zu entwickelnde Hardware aus der Fertigung. Es muss möglich sein ein Prototyp des Geräts quasi von Hand ohne spezialisierte Geräte herzustellen. Es muss aber gleichzeitig eine maschinengestützte Serienfertigung des Geräts möglich sein.

Konkret bedeutet dies, dass Platinen beliebiger Größe und Form mit beliebig vielen Lagen entworfen werden können, da deren Produktion relativ kostengünstig auch in kleiner Stückzahl in Auftrag gegeben werden kann. Dabei sollten aber keine exotischen Eigenschaften oder Herstellungsverfahren vorausgesetzt werden um bei der Produktion keine Probleme zu verursachen. Es muss möglich sein alle benötigten Bauteil in der vorgesehenen Zeit zu beschaffen und eine manuelle Bestückung durchzuführen. Dazu stehen nur eine handelsübliche Lötstation und ein industrieller Heißluftföhn zur Verfügung.

3.3.4. Zusammenfassung

An dieser Stelle werden nochmals alle aufgestellten Anforderungen zur späteren Referenz als Liste zusammengefasst. Um die Übersichtlichkeit und Referenzierbarkeit der Liste zu verbessern wurde dem nächsten Kapitel schon minimal vorgegriffen. Die Anforderungen an die maximale Reaktionszeit von 10 ms und 5 ms wurden bereits zu einer Anforderung zusammengefasst und die Liste wurde grob nach Themenbereichen sortiert.

1. Digitale Eingänge
   - 1.1. Mindestens 12 Strom aufnehmende digitale Eingänge (24 V)
   - 1.2. Kompatibilität zu 3-Draht-Sensoren und mechanischen Schaltern
   - 1.3. Kompatibilität zu 2-Draht-Sensoren mit Schaltverhalten gemäß IEC 61131-2
   - 1.4. Störungsunterdrückung
2. Digitale Ausgänge
   - 2.1. Mindestens 8 Strom liefernde digitale Ausgänge (24 V)
   - 2.2. Pro Ausgang mindestens 0,3 A
   - 2.3. Überspannungsschutz
   - 2.4. Schutz gegen temporäre Überlast
   - 2.5. Schutz bei dauerhafter Überlast bzw. Kurzschluss

2.6. Einhaltung der Grenzwerte (IEC 61131-2) für Leckströme
2.7. Einhaltung der Grenzwerte (IEC 61131-2) für Spannungsabfälle
3. Digitale Ein- und Ausgänge
3.1. Leuchtanzeigen für den Zustand aller digitalen Ein- und Ausgänge
3.2. Zusätzliche digitale Ein- und Ausgänge für das Bedienfeld
4. Anlagenbezogenen Ein- und Ausgänge
4.1. Einfache Zusammenschaltung der Ausgänge einer SPS mit den Eingängen einer anderen
4.2. Anschluss von Ein- und Ausgängen mittels Schraubklemmen
5. Analoge Ein- und Ausgänge und komplexere Schnittstellen
5.1. Mindestens 1 analoger Eingang
5.2. Mindestens 1 analoger Ausgang
5.3. Wertebereiche ±10 V und 0 V bis 10 V für analoge Ein- und Ausgänge
5.4. Wertebereiche 0 mA bis 20 mA und 4 mA bis 20 mA für analoge Ein- und Ausgänge
5.5. AS-Interface, sowie Feldbusschnittstellen (PROFIBUS-DP und PROFINET)
6. Schutzfunktionen
6.1. Verhinderung der Ausbreitung von Störungen
6.2. Schutz gegen ESDs
6.3. Verpolungsschutz
6.4. Schutz gegen Programmierfehler (keine Beschädigung der Hardware)
7. Nicht anlagenbezogene Schnittstellen
7.1. Erweiterungsschnittstelle mit geringer Latenz und hoher Bandbreite
7.2. Schnittstelle zur kontinuierlichen Kommunikation mit PC
7.3. Gesamte vom Benutzer änderbare Software auf Wechseldatenträger
7.4. Videoausgang
7.5. Schnittstellen für Eingabegeräte (Bedienung, Konfiguration und Anpassung der Software)
7.6. Schnittstelle zum Anschluss von Datenträgern
8. Steuerung
8.1. Reaktionszeit auf beliebige Eingabeänderung maximal 5 ms (weiche Anforderung)
8.2. Möglichkeit zur zyklischen und ereignisorientierten Steuerung
9. Benutzerfreundlichkeit
9.1. Möglichkeiten zur Erkennung und Diagnose von Fehlersituationen
9.2. Einfache Austauschbarkeit der Sicherung (sofern vorhanden)
9.3. Kurze Einarbeitungszeit
9.4. Einfache Integration in bestehende Systeme
9.5. Kleine Bauform
9.6. Großer Toleranzbereich bei Versorgungsspannung
10. Offenheit
10.1. Öffentlich verfügbare Dokumentation verbauter Komponenten
10.2. Öffentlich verfügbarer Quellcode der auf der SPS eingesetzten Software
11. Fertigung
11.1. Prototypfertigung ohne spezialisierte Geräte in der vorgesehen Zeit
11.2. Möglichkeit zur Serienfertigung

3.4. Systementwurf

In diesem Abschnitt wird basierend auf den aufgestellten Anforderungen an die zu entwickelnde SPS ein Gesamtsystem entworfen. Es wird dabei noch nicht in allen Details auf die konkrete Implementierung eingegangen. Ziel dieses Abschnitts ist es die Anforderungen in Einklang zu bringen, ggf. auftretenden Konflikte zwischen Anforderungen durch Abwägung der Prioritäten und Kompromissfindung aufzulösen, zwischen verschiedenen möglichen Realisierungsvarianten abzuwägen und zu entscheiden

welche Realisierungsvariante davon unter Beachtung der Anforderungen im Folgenden verfolgt werden soll.

3.4.1. Art der Steuerung

Die erste grundlegende Frage beim Systementwurf ist die Frage, nach der Bauform, die das Gerät haben soll. Im Abschnitt 2.2 wurden die gängigen Formen vorgestellt. In diesem Abschnitt werden die Vor- und Nachteile der einzelnen Bauformen im Bezug auf den hier vorliegenden Anwendungsfall diskutiert. Obwohl die Aufgabenstellung den Bau einer modularen Steuerung nicht vorsieht, wird diese der Vollständigkeit halber in dieser Diskussion mit aufgeführt. Nach der Betrachtung der Vor- und Nachteile der einzelnen Bauformen wird eine Entscheidung bezüglich der Bauform der geplanten Steuerung getroffen.

Modulare Steuerung

Der wesentliche Vorteil einer modularen Steuerung wäre die extrem hohe Flexibilität. Je nach Anwendungsfall könnten die benötigten Module insbesondere Schnittstellen zusammengestellt werden. Dies wäre ein entscheidender Vorteil für den Forschungsbereich, da dort verschiedenste Geräte zum Einsatz kommen können. Für den Bildungsbereich ist diese Eigenschaft nicht so entscheidend, da hier relativ einfache Anlagen mit relativ geringen und vorher klar definierten Anforderungen an die Schnittstellen vorhanden sind.

Ein Nachteil der Realisierung als modulare Steuerung wäre die hohe Komplexität im Betrieb. Diese könnte zu einer längeren Einarbeitungszeit führen. Dies wäre im Forschungsbereich eher kein Problem aber im Bildungsbereich ist eine kurze Einarbeitungszeit ein wichtiger Faktor. Hauptnachteil einer modularen Steuerung für diese Arbeit wäre aber die hohe Komplexität bei der Entwicklung des Geräts.

Kompaktsteuerung

Vorteile einer Kompaktsteuerung wäre die geringere Komplexität sowohl im Betrieb als auch in der Entwicklung und ihre handliche Größe, die eine Integration in bestehende Systeme erlauben würde ohne Platzprobleme befürchten zu müssen. Eine geringe Komplexität würde eine kurze Einarbeitungszeit sicherstellen und es erleichtern das Gerät tolerant gegen Bedienungs- und Programmierfehler zu konstruieren. Beides sind wichtige Faktoren besonders im Bereich der Bildung. Offensichtlicher Nachteil wäre die geringe Flexibilität was zusätzliche Schnittstellen angeht.

PC-basierte Lösung

Eine PC-basierte Lösung, bestehend aus einem gewöhnlichen PC und einer Erweiterungskarte um die nötigen Schnittstellen bereitzustellen, hätte den Vorteil, dass von Vorherein viele der aufgestellten Anforderungen durch den PC erfüllt werden würden. Es wären Erweiterungsschnittstellen für Erweiterungskarten zum Anschluss von Eigenentwicklungen, insbesondere PCI Steckplätze vorhanden. Die gesamte Software könnte einfach auf Wechseldatenträgern platziert werden, es wären viele Schnittstellen zum Anschluss von Eingabegeräten und Schnittstellen zur kontinuierlichen Kommunikation, insbesondere LAN und serielle Schnittstellen, vorhanden. Durch Anschlussmöglichkeiten für Bildschirme wären auch Visualisierungsmöglichkeiten gegeben. Beim Einsatz eines Desktopbetriebssystems, könnte die Entwicklung praktischerweise auch gleich auf demselben Gerät erfolgen. Schnittstellen zum Anschluss von externen Datenträgern wären selbstverständlich auch vorhanden.

Diese Lösung hätte allerdings auch Nachteile. Auch wenn freie PCI Steckplätze zur Verfügung stünden ist zu bedenken, dass die PCI Schnittstelle eine vergleichsweise komplexe Schnittstelle darstellt. Es ist durchaus wahrscheinlich, dass in vielen Fällen der Wunsch besteht kurzfristig für einen Test sehr einfache zusätzliche Geräte, wie etwa einen Schalter, einen Zählerchip, eine LED, einen einfa-

chen Mikrokontroller oder ein Messgerät z.B. zur zeitlichen Synchronisation von Messungen mit internen Abläufen anzuschließen. Diese einfachen Geräte über eine PCI Schnittstelle anzuschließen würde einen extremen Zusatzaufwand bedeuten. Ein weiterer Nachteil einer PC-basierten Lösung wäre der hohe Platzbedarf.

Slot-SPS

Eine Realisierung als Slot-SPS hätte fast nur Nachteile. Es wäre wie bei einer PC-basierten Lösung ein hoher Platzbedarf vorhanden. Die am PC vorhandenen Schnittstellen könnten nicht direkt von der SPS genutzt werden. Eine Platzierung der gesamten Software auf einen Wechseldatenträger wäre schwierig, da die Software der SPS auf die PCI-Karte und den PC verteilt wäre. Die eigentliche Software der SPS wäre zwar auf die PCI-Karte beschränkt und könnte dort auf einem Wechseldatenträger liegen, aber es gäbe eine sehr enge Verbindung zur Software auf dem PC, so dass auch diese auf einem Wechseldatenträger liegen müsste und beide aufeinander abgestimmt sein müssten um bei Prüfungssituationen im Bildungsbereich für alle gleiche Anfangsbedingungen sicherstellen zu können. Die Flexibilität im Bezug auf Schnittstellen wäre nicht besser als bei einer Kompakt-SPS. In den PC könnten zwar Erweiterungskarten mit zusätzlichen Schnittstellen gesteckt werden, aber diese wären von der SPS nicht unabhängig vom PC nutzbar. Die Kommunikation müsste über den Hauptspeicher des PCs und das PCI System des PCs erfolgen, wäre damit nicht abhängig vom Betriebssystem und Stromversorgung des PCs. Die Realisierung als Slot-SPS wäre mit einer extrem hohen Komplexität im Betrieb und in der Entwicklung verbunden.

Der Hauptvorteil von Slot-SPSen besteht ja im Austausch von Daten zu Visualisierungszwecken über die PCI Schnittstelle des PCs. Diese Funktion ist aber auch mit deutlich geringerer Komplexität über andere Schnittstellen, wie LAN oder einer seriellen Schnittstelle realisierbar.

Folgerung

Bis auf eine Realisierung als Slot-SPS bringen alle Realisierungsmöglichkeiten interessante Vorteile mit sich. Eine Modulare Steuerung wäre vor allem aufgrund ihrer Flexibilität und eine PC-basierte SPS vor allem aufgrund der vielen Schnittstellen interessant. Insgesamt scheint aber eine Kompaktsteuerung aufgrund der überschaubaren Komplexität, die eine kurze Einarbeitungszeit sicherstellt, eine vergleichsweise kurze Entwicklungszeit verspricht und eine sehr robuste Bauweise ermöglicht und aufgrund ihrer kleinen Baugröße, die zu einer problemlosen Integration beitragen sollte, die beste Wahl für diese Arbeit. Bei der weiteren Entwicklung ist aber darauf zu achten, dass einfache Erweiterungsmöglichkeiten für Eigenentwicklungen vorgesehen werden um zumindest etwas Flexibilität bereitzustellen und alle aufgestellten Anforderungen bezüglich der gewünschten Schnittstellen abgedeckt werden um dem Benutzer einfaches, komfortables Arbeiten zu ermöglichen.

3.4.2. Art der anlagenbezogenen Schnittstellen

In diesem Abschnitt geht es darum festlegen, welche Schnittstellen jetzt konkret zu realisieren sind und welche Eigenschaften, diese aufweisen müssen. Zunächst ist festzustellen, dass die betrachteten Lernsysteme fast ausschließlich digitale Ein- und Ausgaben aufweisen. Analoge Schnittstellen sind nur bei wenigen Anlagen vorhanden und selbst dort meist nur optional. Nach Erfahrungen des Autors dieser Arbeit werden analoge Schnittstellen im Zusammenhang mit den vorgestellten Lernsystemen sowohl im Bildungs- noch im Forschungsbereich kaum eingesetzt. Diese Aussage basiert, wie gesagt, nur auf den persönlichen Erfahrungen des Autors und ist nicht als allgemeingültig anzusehen, aber angesichts dieser Erfahrungen und der angestrebten kurzen Entwicklungszeit liegt es nahe die zu entwickelnde SPS zunächst nur mit digitalen Schnittstellen auszustatten und analoge Schnittstellen bei Bedarf später als Erweiterung hinzuzufügen. Da ohnehin eine Erweiterungsschnittstelle vorzusehen ist, sollte es kein Problem darstellen, später an diese oder besser einen Teil davon eine Erweiterung für

analoge Ein- und Ausgänge anzuschließen. So hätte man auch den Vorteil ähnlich zu modularen Steuerungen, dass diese Schnittstelle nur bei Anlagen, wo sie auch benötigt wird, verbaut werden muss.

Da andere Schnittstellen, wie das AS-Interface und die verschiedenen Feldbusse zur Steuerung der primär betrachteten Systeme nicht benötigt werden, die Spezifikation der meisten nicht frei verfügbar ist und deren Integration in ein Gerät oft die Notwendigkeit eines Zertifizierungsprozesses mit sich bringt, wird deren Integration in dieser Arbeit nicht betrachtet.

Bisher ist bekannt, dass mindestens 12 digitale Eingänge und mindestens 8 digitale Ausgänge benötigt werden um den Bedarf an digitalen Ein- und Ausgängen aller betrachteten Automatisierungsanlagen decken zu können. Darüber hinaus ist bekannt, dass an jeder Station noch ein Bedienfeld vorhanden ist, welches weitere Ein- und Ausgänge benötigt. Eine aufgestellte Anforderung ist eine einfache Integration in bestehende Systeme. Die eingesetzten Lernsysteme werden, wie zuvor beschrieben, unabhängig von der eingesetzten Steuerung ggf. mit Adaptern stets über 24-polige Centronics Stecker angeschlossen. Um bei der Integration des zu entwickelnden Geräts an der Verkabelung keine Änderung durchführen zu müssen, sollte der Anschluss weiterhin über Centronics-Stecker erfolgen. Um dabei keine Adapter verwenden zu müssen, liegt es nahe digitale Ein- und Ausgänge am Gerät gleich in Form von passenden Centronics Buchsen bereitzustellen. Diese müssen natürlich Pin-kompatibel zu den von FESTO eingesetzten Buchsen sein um ein einfaches Umstecken zu ermöglichen. Die zu realisierende Pinbelegung ist somit bekannt. Sie muss der im Abschnitt 3.2.3 in Tabelle 2 aufgeführten Belegung entsprechen. Üblicherweise werden zwei Kabel beidseitig mit Centronics Steckern versehen zum Anschluss einer Anlage verwendet. Ein Kabel für die Anlage selbst und eines für das Bedienfeld. Da an jedem Stecker 8 digitale Eingänge und 8 digitale Ausgänge bereitgestellt werden, müssen also insgesamt jeweils 16 digitale Ein- und Ausgänge realisiert werden um die einfachste denkbare Integration, bestehend aus einem einfachen Umstecken, und die höchste mögliche Kompatibilität zu den eingesetzten Lernsystemen zu gewährleisten. Analoge Ein- und Ausgänge müssen in Zukunft ggf. auf eine Sub-D Buchse mit einer Pinbelegung, wie in Tabelle 3 gezeigt, ausgeführt werden um auch da ein einfaches Umstecken zu ermöglichen. Erforderlich wären dafür 4 analoge Eingänge und 2 analoge Ausgänge.

Die Ausführung der digitalen Anschlüsse als Centronics Buchsen wiederspricht erst mal der Anforderung die in der Industrie übliche Anschlussmethode mit Schraubklemmen zu verwenden. Dies ist aber kein Problem, da Adapter von Centronics Buchsen zu Schraubklemmen ohnehin in den Lernsystemen zum Anschluss der Sensoren und Aktuatoren Verwendung finden und damit im Allgemein bereits vorhanden sind und weiter ggf. als Zubehör zu den Lernsystemen von FESTO bezogen werden können.

3.4.3. Eigenschaften der anlagenbezogenen Schnittstellen

Damit steht an dieser Stelle fest, dass jeweils 16 digitale Ein- und Ausgänge zu realisieren sind und diese auf Centronics Buchsen bereitzustellen sind, die Pin-kompatibel zu den von FESTO in den Lernsystemen eingesetzten sind. Die benötigten elektrischen Eigenschaften dieser müssen aber im Folgenden noch betrachtet werden.

Digitale Eingänge

Zu realisieren sind 16 Strom aufnehmende digitale Eingänge. Erste Anforderung ist dabei die Kompatibilität der Eingänge zu 3-Draht-Sensoren und mechanischen Schaltern, welche mit einer Nennspannung von 24 V arbeiten. Dies stellt keine besondere Herausforderung dar. Bei dieser Art von Eingabesignalen ist der genaue Verlauf der Kennlinie weitestgehend irrelevant. Der Schaltpunkt sollte bestenfalls mittig zwischen beiden Grenzen also zwischen 0 V und 24 V, somit irgendwo im Bereich von

12 V liegen um größtmögliche Toleranzen zu gewährleisten. Alle drei von der IEC 61131-2 Norm spezifizierten Typen von Eingängen sind prinzipiell für diese Art von Eingabesignalen geeignet.

Wenn nur die Kompatibilität zu 3-Draht-Sensoren und elektromechanischen Schaltern von Interesse wäre, wären Eingänge vom Typ 1 am besten geeignet, da diese den geringsten Stromverbrauch haben. Eine weitere Anforderung ist aber die Kompatibilität zu industriellen Sensoren, insbesondere auch 2-Draht-Sensoren. Für diese sind Eingänge vom Typ 2 oder vom Typ 3 gemäß IEC 61131-2 vorzusehen. Wie zuvor schon festgestellt, können mechanische Schalter und 3-Draht-Sensoren auch an Typ 2 oder Typ 3 Eingängen betrieben werden, es erhöht sich nur mehr oder weniger der Stromverbrauch. Ein geringfügig höherer Stromverbrauch stellt aber im vorliegenden Fall kein großes Problem dar. Daher können alle Eingänge bedenkenlos als Typ 2 oder Typ 3 Eingänge realisiert werden.

Typ 2 Eingänge würden eine Kompatibilität mit möglichst vielen Sensoren gewährleisten. Da, wie schon im Abschnitt 2.4.1 erwähnt, für viele zeitgemäße 2-Draht-Sensoren aber Typ 3 Eingänge genügen und Typ 2 Eingänge eigentlich vor allem nur für ältere Sensoren benötigt werden, wäre es möglich, sich auf Typ 3 Eingänge zu beschränken, welche den geringeren Stromverbrauch von beiden haben. Da diese Entscheidung keine Auswirkungen auf den weiteren Systementwurf haben dürfte und vor allem die konkrete Implementierung betrifft, scheint es sinnvoll diese Entscheidung bis zur Implementierungsphase zu vertagen.

In jedem Fall sollten die Eingänge eine Hysterese aufweisen um eine Oszillation zu verhindern, wenn das Eingabesignal länger oder sogar dauerhaft im Bereich des Schaltpunkts liegt, was normalerweise zwar nicht vorkommen sollte, wenn alles fehlerfrei funktioniert. Allerdings kann es je nach Sensor vorkommen, dass die Signalflanken langsam sind und das Eingabesignal daher relativ lange dicht am Schaltpunkt verbleibt. Die andere Möglichkeit wäre natürlich das Auftreten eines Fehlers, der dazu führt, dass das Eingabesignal sich dauerhaft in der Nähe des Schaltpunkts bewegt. Auch diesem Fall darf es zu keinem unkontrollierten Schalten kommen.

Digitale Ausgänge

Bekannt ist bisher, dass 16 Strom liefernde digitale Ausgänge zu realisieren sind, von denen jeder dauerhaft mindestens 0,3 A liefern können muss. Um die Ausgänge nicht an der Belastungsgrenze zu betreiben, scheint es sinnvoll alle Ausgänge für etwa 0,5 A auszulegen.

Eine aufgestellte Anforderung ist die Einhaltung der Grenzwerte der IEC 61131-2 für Leckströme und Spannungsabfälle an den Ausgängen. Für den gerade festgelegten Bemessungsstrom von 0,5 A heißt das, dass der Leckstrom pro Ausgang maximal 0,5 mA betragen darf und die Ausgangsspannung bei aktiviertem Ausgang nicht mehr als 3 V unterhalb der Versorgungsspannung liegen darf, sofern keine Überlastsituation vorliegt.

Zusammenschaltung anlagenbezogener Schnittstellen

Eine Anforderung besteht darin eine einfache Zusammenschaltung von digitalen Eingängen und digitalen Ausgängen verschiedener Geräte zu ermöglichen. Das ist bei SPSen normalerweise kein Problem. Zu einem Problem kann es nur kommen, wenn der Leckstrom des beteiligten Ausgangs im Aus-Zustand so hoch ist, dass dieser vom Eingang schon als Ein-Signal erkannt wird. In diesem Fall wäre eine zusätzliche externe Last, vorzugsweise ein Widerstand notwendig. Dies möchte man natürlich vermeiden, wenn dies möglich ist. Um zu prüfen, ob es zu diesem Problem beim zu entwickelnden Gerät basierend auf den bisherigen Festlegungen kommen kann, sind drei Fälle zu prüfen.

1. Ein digitaler Ausgang des Geräts wird mit einem digitalen Eingang eines fremden Geräts verbunden. In diesem Fall sind die Eigenschaften des Eingangs nicht bekannt. Ausgehend davon, dass das andere Gerät bezüglich des Schaltverhaltes ebenfalls der IEC 61131-2 Norm folgt,

was eine SPS im Normalfall tun sollte, müssen die Schalteigenschaften des Eingangs denen eines Typ 1, Typ 2 oder Typ 3 Eingangs entsprechen. Mehr Eingangstypen werden von der Norm nicht definiert. Von allen drei Eingangstypen werden unabhängig von dem genauen Aussehen er implementierten Kennlinie alle Eingangssignale unterhalb von 0,5 A als Aus-Zustand interpretiert. Folglich darf ein digitaler Ausgang des zu entwickelnden Geräts nicht mehr als 0,5 mA Leckstrom aufweisen um mit allen normgerechten Eingängen ohne zusätzliche Last kompatible zu sein. Diese Forderung wurde ohnehin schon im vorhergehenden Abschnitt definiert. Wird diese eingehalten, ist keine externe Last notwendig. Zu beachten ist nur, dass die Forderung den Leckstrom auf maximal 0,5 mA zu beschränken nun auch unabhängig vom Bemessungsstrom einzuhalten ist. Sollte in einem späteren Entwurfsstadium der Bemessungsstrom pro Ausgang erhöht werden, muss die Obergrenze für den Leckstrom beibehalten werden und darf nicht gemäß der Norm angepasst werden.

2. Ein digitaler Ausgang eines fremden Geräts wird an einen digitalen Eingang des zu entwickelnden Geräts angeschlossen. In diesem Fall ist bekannt, dass der Eingang vom Typ 2 oder Typ 3 ist. Der Bemessungsstrom des fremden Ausgangs ist dafür nicht bekannt und damit auch nicht der zu erwartende Leckstrom. Also muss man vom höchsten von der Norm definierten Leckstrom von 1 mA ausgehen, der für die stärksten überhaupt definierten Ausgänge von 1 A und 2 A festgelegt ist. Selbst in diesem Fall kommt es aber zu keinem Problem da Typ 2 bzw. Typ 3 Eingänge Leckströme bis 2 mA bzw. 1,5 mA tolerieren.

3. Ein digitaler Ausgang des zu entwickelnden Geräts wird an einen digitalen Eingang desselben Geräts oder eines anderen Exemplars desselben. In diesen Fall sind sowohl die Eigenschaften des Eingang als auch des Ausgangs bekannt. Gemäß bisheriger Festlegung übersteigt der Leckstrom eines Ausgangs nicht 0,5 mA und der Eingang hat die Eigenschaften des Typs 2 oder des Typs 3, womit Leckströme bis 2 mA bzw. 1,5 mA toleriert werden. Damit kommt es auch hier zu keinem Problem.

Damit ist bei keiner der drei Anschlussmöglichkeiten eine zusätzliche externe Last notwendig, womit unter Annahme der bisher definierten Eigenschaften eine einfache Zusammenschaltung der digitalen Ein- und Ausgänge verschiedener Geräte gewährleistet ist.

3.4.4. Erweiterungsschnittstelle

Gefordert wird eine Erweiterungsschnittstelle, an die Eigenentwicklungen im Forschungsbereich angeschlossen werden können. In Frage kommen zwei Arten von Geräten.

1. Die erste Art stellen sehr einfache Geräte, wie Schalter, LEDs und Messgeräte dar. Ein Messgerät, z.B. ein Oszilloskop anzuschließen kann interessant sein, wenn es z.B. gewünscht ist eine Messung mit den Vorgängen in Software zu synchronisieren. Es könnte beispielsweise interessant sein, ein Signal für einen Triggereingang eines Oszilloskops von der Software erzeugen zu lassen. Solche einfachen Geräte, die nur ein binäres Signal benötigen oder liefern, müssen mit sehr geringem Aufwand angeschlossen werden können und leicht in Software zugreifbar sein. Es müssen dafür zusätzliche Ein-/Ausgabeleitungen bereitgestellt werden. Diese müssen aber keine hohe Spannung liefern, die üblichen 3,3 V oder 5 V mit denen Mikrochips arbeiten sind hier optimal. Es wird auch keine hohe Stromstärke benötigt, einige Milliampere sollten ausreichend sein. Da solche Leitungen, wie gerade beschrieben, auch für zeitkritische Messungen benutzt werden könnten, müssen diese Leitungen mit geringer, gut vorhersagbarer Verzögerung zugegriffen werden können. Im Falle von Eingängen sollte eine Möglichkeit bestehen, bei Eingabeänderungen Interrupts erzeugen zu lassen um auf ein ständiges Abfragen der Leitungen verzichten zu können und trotzdem schnell reagieren zu können.

2. Die zweite Art stellen Mikrochips oder komplexere Schaltungen dar, welche nicht nur ein einzelnes Bit sondern ein oder mehrere Byte an Daten bereitstellen oder erwarten. Dabei könnte es sich beispielsweise um Zählerchips oder A/D- und D/A-Wandler für die früher erwähnte analoge Schnittstellen handeln. Um bei solchen Chips eine einfache, effiziente Datenübertragung zu ermöglichen, sind im weitesten Sinne serielle Schnittstellen einzubauen. Insbesondere in Betracht kommen ein klassisches, einfaches UART Interface für Punkt-zu-Punkt Verbindungen, eine I^2C Schnittstelle um ein einfaches serielles Bussystem aufbauen zu können, sowie eine SPI Schnittstelle. Letztere ist zwar weniger gut standardisiert und Kompatibilitätsprobleme sind wahrscheinlicher, sie wird aber von sehr vielen Chips eingesetzt und bietet im Vergleich zu den anderen beiden typischerweise eine höhere Bandbreite.

In beiden Fällen ist davon auszugehen, dass Nutzer, die diese Erweiterungsschnittstellen einsetzen wollen, ausreichende Kenntnisse auf dem Bereich elektrischer Schaltungen haben und mit angemessener Sorgfalt vorgehen um auf besondere Schutzmaßnahmen, bei diesen Schnittstellen verzichten zu können.

3.4.5. Nutzerschnittstellen

Eine Anforderung besteht darin, eine Schnittstelle zur kontinuierlichen Kommunikation mit einem PC bereitzustellen, die für Debugging-Aufgaben, zur Fernsteuerung und zur komfortablen Übertragung von Steuerungsprogrammen genutzt werden kann. Entscheidend für die Wahl dieser Schnittstelle sind die Schnittstellen, die an der Gegenseite also einem PC oder Notebook vorhanden sind. Je nachdem, ob eine Netzwerk- oder Punkt-zu-Punkt-Verbindung bevorzugt wird, stehen dazu an handelsüblichen Geräten eine LAN-Schnittstelle und eine serielle Schnittstelle zur Verfügung, wobei serielle Schnittstellen meist schon mit USB-Adaptern nachgerüstet werden müssen. Die meisten Anwender dürften eine Netzwerkverbindung bevorzugen. Diese hat auch den Vorteil, dass problemlos auf mehrere Geräte auf einmal zugegriffen werden kann und mehrere Nutzer ein Gerät gemeinsam nutzen können. Ob eine gemeinsame gleichzeitige Nutzung einer SPS durch mehrere Nutzer sinnvoll ist, hängt stark von der dort eingesetzten Software ab, zu der bisher noch keine Festlegungen getroffen wurden. In jedem Fall sollte eine Ethernet-Schnittstelle am zu entwickelnden Gerät vorgesehen werden um eine Integration in ein Netzwerk zu ermöglichen. Diese sollte nach heutigem Stand der Technik mindestens 100 MBit/s oder mehr unterstützen. Optional könnte auch eine serielle Schnittstelle vorgesehen werden um dem Nutzer verschiedene Anschlussmöglichkeiten zu bieten. Diese wäre gemäß der im PC-Bereich genutzten Variante RS-232 auszuführen.

Eine weitere Anforderung ist ein Videoausgang zu Visualisierungs-, Konfigurations- und ggf. auch zu Entwicklungszwecken genutzt werden kann. Um einen normalen Bildschirm anschließen zu können, kommen die Schnittstellen VGA, DVI oder HDMI in Frage. Von diesen ist mindestens eine vorzusehen.

Auch Schnittstellen für Eingabegeräte sind eine wichtige Anforderung. Für Eingaben in einer Benutzeroberfläche, welche eine Steuerungssoftware zu Visualisierungs- und Steuerungszwecken bereitstellt, wäre ein Touchscreen denkbar. Für Entwicklungsaufgaben sind aber eine Tastatur und eine Maus deutlich besser geeignet. Diese sollten für alle erforderlichen Benutzereingaben hinreichend gut geeignet sein. Natürlich sollte es möglich sein, eine handelsübliche Tastatur anzuschließen. Heute werden fast nur noch Tastaturen und Mäuse mit USB-Schnittstelle angeboten. Um für den Anschluss von Maus und Tastatur, was die typische Konfiguration sein dürfte, kein Hub zu benötigen, sollten mindestens zwei USB-Schnittstellen vorgesehen werden. Diese können dann praktischerweise auch gleich zum Anschluss von externen Datenträgern genutzt werden, was eine weitere Anforderung abdeckt. Dafür müssen sie aber mindestens der USB Version 2.0 oder höher genügen, um eine angemessen schnelle Übertragung zu gewährleisten. Um auch beim Anschluss eines USB-Datenträgers weiter-

hin Maus und Tastatur einsetzen zu können und weiterhin kein Hub zu benötigen sollten idealerweise mindestens drei USB-Schnittstellen vorgesehen werden.

Eine weitere Anforderung besteht darin die gesamte vom Benutzer änderbare Software auf einem Wechseldatenträger zu platzieren, der einfach an einen PC angeschlossen, ausgelesen, bearbeitet und geklont werden kann. Dazu kommen eigentlich nur USB-Speichermedien, vorzugsweise ein USB-Stick oder Speicher-Karten, vorzugsweise eine SD-Karte, in Betracht. Im Falle einer SD-Karte wäre entweder eine Micro-SD-Karte oder eine in voller Größe vorzusehen, da Mini-SD-Karten praktisch ausgestorben sind.

3.4.6. Betriebssystem

Nachdem feststeht, dass eine Kompaktsteuerung mit digitalen Ein- und Ausgängen zu entwickeln ist und bekannt ist, welche Nutzerschnittstellen in Betracht kommen, scheint es an diesem Punkt sinnvoll sich für das Betriebssystem zu entscheiden, das zum Einsatz kommen soll. Wie die Hardwarestruktur gestaltet sein muss, wird nämlich vom Betriebssystem entscheidend beeinflusst, da jedes Betriebssystem gewisse Anforderungen an die Hardware mit sich bringt.

Bei der Wahl des Betriebssystems sind unter anderem die geplanten Schnittstellen zu berücksichtigen. Der Einbau einer LAN-Schnittstelle hat zur Folge, dass ein Netzwerkstack erforderlich sein wird, der die üblichen Protokolle, wie TCP/IP, UDP/IP und ARP unterstützt. Eine serielle Schnittstelle ist weitestgehend anspruchslos, aber Schnittstellen wie VGA, DVI, HDMI und USB setzen das Vorhandensein oder das Entwickeln von Treibern für verschiedenste Geräte voraus. Der Einsatz von Wechseldatenträgern, welche auch mit PCs ausgelesen werden können sollen, setzt die Unterstützung eines Dateisystems, insbesondere von FAT32, voraus.

Kein Betriebssystem

Eine Möglichkeit, die man nicht übersehen sollte, ist der Betrieb des Geräts ohne Betriebssystem. Eine Steuerungssoftware könnte auf bare metal ausgeführt werden. Dies hätte durchaus einige Vorteile, man hätte die beste denkbare Performance und könnte das zeitliche Verhalten, wie auch das Verhalten des Systems überhaupt, vollständig kontrollieren. Es wäre aus Softwaresicht möglich sogar harte Echtzeitanforderungen zu erfüllen. Dies könnte bei Bedarf auch vergleichsweise einfach nachgewiesen werden. Es wäre eine geringe Rechenleistung ausreichend, da außer der Steuerungssoftware selbst, keine weitere Software ausgeführt wird. Es gäbe auch keine von einem Betriebssystem auferlegten Anforderungen an die Hardware, womit man völlige Gestaltungsfreiheit beim Hardwareentwurf hätte.

Diese Möglichkeit bringt aber andererseits auch gravierende Nachteile mit sich. Für eine Ausführung ohne Betriebssystem ist die meiste verfügbare Software aus dem PC-Bereich nicht geeignet. Man wäre bei der Wahl der Software extrem eingeschränkt und müsste praktisch jede Funktionalität selber implementieren oder portieren. Dies wäre besonders gravierend da viele Schnittstellen unterstützt werden sollen und alle benötigten Treiber für diese selbst entwickelt werden müssten. Sie müssten in jedes Steuerungsprogramm, das die entsprechende Schnittstelle nutzen will integriert werden. Dies würde zu einer extrem hohen Komplexität bei der Entwicklung und der Nutzung des Geräts führen. Es wäre für die Nutzer ein völlig unbekanntes System und würde eine extrem lange Einarbeitungszeit erfordern.

Echtzeitbetriebssystem

Bei einer SPS liegt aufgrund der Echtzeitanforderungen der Einsatz eines Echtzeitbetriebssystem nahe. Dabei ist zunächst zu berücksichtigen, dass sehr viele Echtzeitbetriebssysteme proprietär sind und ihr Quellcode nicht öffentlich zur Verfügung steht, was der geforderten Offenheit widerspricht. Es könnte also nur zwischen denjenigen Systemen gewählt werden, deren Quellcode verfügbar ist.

Der offensichtliche Vorteil eines Echtzeitbetriebssystems wäre die Möglichkeit die Einhaltung von harten Echtzeitanforderungen garantieren zu können. Allerdings ist davon auszugehen, dass nur eine geringe Anzahl an Treibern und Benutzersoftware für einzelne Echtzeitbetriebssysteme zur Verfügung steht. Damit könnte keine so gute Kompatibilität z.B. mit Wechseldatenträgern, wie man sie von normalen PCs kennt und nur ein geringerer Funktionsumfang angeboten werden. Der wichtigste Nachteil beim Einsatz eines Echtzeitbetriebssystems bestünde aber darin, dass es sich für die meisten Benutzer, vor allem für Studenten und Schüler, um ein unbekanntes System handeln dürfte, was eine längere Einarbeitungszeit mit sich bringen würde. Im Forschungsbereich dürfte dies ein deutlich geringeres Problem sein, da dort viele Benutzer schon mit Echtzeitbetriebssystemen gearbeitet haben dürften.

Allzweckbetriebssystem mit Echtzeiterweiterung

Es wäre auch eine Möglichkeit ein bekanntes System, speziell ein Allzweckbetriebssystem wie Windows oder Linux mit einer Echtzeiterweiterung einzusetzen. Da Windows nicht quelloffen ist, käme von den beiden also nur Linux in Frage.

Vorteil hierbei wäre die gute Verfügbarkeit von Treibern, wodurch eine Kompatibilität mit sehr vielen Geräten, insbesondere vielen USB-Geräten, gewährleistet wäre. Abgesehen von Treibern wäre auch viel anderer Software für so ein System verfügbar. Es wäre eine Vielzahl von fertigen Programmen und Bibliotheken verfügbar. Da es eben ein bekanntes System wäre, könnte von einer sehr kurzen Einarbeitungszeit ausgegangen werden. Aufgrund der vielen verfügbaren Software wäre auch der Entwicklungsaufwand auf Softwareseite überschaubar.

Großer Nachteil dieser Variante wäre die Tatsache, dass es nicht möglich wäre harte Echtzeitgarantien für das System abzugeben. Mit einer Echtzeiterweiterung könnte zwar das zeitliche Verhalten dahingehend verbessert werden, aber harte Echtzeitbedingungen könnten nie garantiert werden.

Doppelsystem

Besonders interessant wäre der Einsatz eines Doppelsystems. Dadurch wäre es möglich die oben beschriebenen Vorteile eines Echtzeitbetriebssystems mit denen eines Allzweckbetriebssystems zu vereinen. Dadurch wären zwar die meisten Probleme hinsichtlich der Verfügbarkeit von Treibern, die bei einem reinen Echtzeitbetriebssystem zu befürchten sind, gelöst, und Garantien bezüglich des Echtzeitverhaltens könnten gegeben werden, allerdings muss klar sein, dass die gesamte Steuerungssoftware auf dem Echtzeitbetriebssystem läuft. Damit stehen die Studenten und Schüler, die Steuerungssoftware implementieren sollen, wieder vor dem Problem für ein unbekanntes System implementieren zu müssen.

Bewertung

Der Einsatz ohne Betriebssystem kommt offensichtlich aufgrund der extrem hohen Komplexität für alle Beteiligten nicht in Frage. Der Einsatz eines reinen Echtzeitbetriebssystems scheint im Vergleich zu einem Doppelsystem nicht Vorteilhaft. Folglich stehen noch das Doppelsystem und ein Allzweckbetriebssystem zur Wahl.

Beide Varianten sind sehr interessant. Aufgrund der zu erwartenden kurzen Einarbeitungszeit, der Möglichkeit bekannte Steuerungssoftware einzusetzen und der überschaubaren Komplexität bei der Entwicklung, fällt die Entscheidung auf das Allzweckbetriebssystem. Der wesentliche Nachteil dabei ist nur das schlechtere Echtzeitverhalten. Dies sollte aber kein Problem darstellen, da genau genommen nur weiche Echtzeitanforderungen bestehen und es nicht notwendig ist Echtzeitgarantien abgeben zu können. Es ist aber sehr wohl notwendig die geforderten 5 ms Reaktionszeit in einem extrem hohen Prozentsatz der Fälle erfüllen zu können. Es liegt nahe aufgrund der hohen Bekanntheit speziell Linux

als Allzweckbetriebssystem zu wählen. Dieses soll zusammen mit dem bereits vorgestellten CONFIG_PREEMPT_RT Patch zum Einsatz kommen.

3.4.7. Hardwarestruktur

Nachdem die gewünschte Bauform, die Schnittstellen und das Betriebssystem bekannt sind, liegen nun genügend Informationen vor um mit dem Entwurf einer Hardwarestruktur beginnen zu können. Aufgrund der Komplexität des Gesamtsystems bietet sich eher eine Realisierung mit einer halbwegs leistungsstarken CPU als mit einem Mikrokontroller an. Grundsätzlich braucht man also erst mal eine CPU und Arbeitsspeicher um Steuerungsprogramme ausführen zu können. Um das System auch hochfahren zu können wird eine Art BIOS oder UEFI nötig sein. Da potenziell komplexe Visualisierungsaufgaben zu realisieren sind, dürfte auch eine GPU erforderlich sein. Dazu kommen die gewünschten Schnittstellen, insbesondere LAN, USB, ggf. SD-Kartenslot, VGA / DVI / HDMI und I^2C / SPI / GPIO, sowie die zugehörigen Schnittstellencontroller. Schließlich sind natürlich auch die digitalen Ein- und Ausgänge zu realisieren.

Es wäre möglich das gesamte Gerät aus den einzelnen Komponenten zusammenzusetzen. Dies hätte den Vorteil, dass man völlige Gestaltungsfreiheit hätte und genau die Komponenten auswählen könnte, die am besten für das zu entwickelnde Gerät geeignet sind. Allerdings wäre dies auch eine extrem komplexe Aufgabe und es ist eigentlich unnötig. Es fällt nämlich auf, dass es Baugruppen gibt, die all diese Komponenten, bis auf die gerätspezifischen digitalen Ein- und Ausgänge bereitstellen, da diese auch für andere Geräte benötigt werden. Das Vorgehen sollte also darin bestehen, eine geeignete Baugruppe, im Folgenden als Logikbaustein bezeichnet, auszuwählen, welche möglichst viele der aufgestellten Anforderungen erfüllt, insbesondere möglichst viele der benötigten Schnittstellen bereitstellt. Diese muss dann nur noch um die fehlenden Komponenten ergänzt werden.

3.4.8. Wahl des Logikbausteins

Als Logikbaustein scheint einer der bekannten Einplatinencomputer geeignet, da diese üblicherweise eine Vielzahl der benötigten Eigenschaften, insbesondere auch Schnittstellen, aufweisen und für diese bereits viel quelloffene Software verfügbar ist, was den Entwicklungsaufwand auch auf Seiten der Software reduzieren würde. Darüber hinaus sind diese preiswert und bieten low-level Erweiterungsschnittstellen wie beispielsweise I^2C, SPI und GPIO Pins an, was eine einfache Anbindung von zusätzlich benötigten Komponenten ermöglicht. Es muss aber klar sein, dass diese Erweiterungsschnittstellen aufgrund einer geringen Bandbreite ungeeignet sind um zusätzlichen Speicher oder schnelle Schnittstellen wie USB oder HDMI zu ergänzen. Diese müssen also schon vorhanden sein oder es müssen geeignete Erweiterungsschnittstellen vorhanden sein. Beispielsweise könnte eine LAN Schnittstelle mit evtl. eingeschränkter Geschwindigkeit ergänzt werden, wenn eine USB Schnittstelle vorhanden ist. Eine bereits vorhandene LAN Schnittstelle wäre aber natürlich zu bevorzugen.

Die Anforderungen an einen Logikbaustein müssen also vorher klar definiert werden. Da alle benötigten Schnittstellen bereits diskutiert wurden, ist hier nur noch eine übersichtliche Liste der Anforderungen aufzustellen:

1. Linux als Betriebssystem verfügbar
2. I^2C / SPI / UART
3. GPIO (bevorzugt über 32 Pins)
4. RS-232
5. LAN (100 MBit/s oder mehr)
6. USB (mindestens 2, bevorzugt 3 oder mehr; mindestens USB 2.0 oder höher)
7. VGA / DVI / HDMI
8. USB-Port (USB 2.0 oder höher) für USB-Stick mit Betriebssystem / SD-Karten-Slot

9. bevorzugt Multi-Core CPU
10. wenigstens einige Hundert MB Arbeitsspeicher
11. Betriebssystem auf Wechseldatenträger
12. günstige Schnittstellenanordnung für Einbau in ein Gehäuse
13. öffentlich verfügbare Dokumentation

Bezüglich Rechenleistung und Arbeitsspeicher bestehen keine besonderen Anforderungen. Es muss nur ein normales Linux-Betriebssystem lauffähig sein. Es sollte nicht nötig sein eine speziell miniaturisierte Version des Systems zu benutzen. Es sollten also wenigstens einige Hundert MB an Arbeitsspeicher vorhanden sein. Eine Multi-Core CPU scheint nicht erforderlich, könnte aber interessant sein um Auswirkungen von mehreren Kernen auf das zeitliche Verhalten untersuchen zu können. Daher wäre eine Multi-Core CPU zu bevorzugen. Die Anzahl der benötigten GPIO Pins steht noch nicht fest, da aber 16 Ein- und 16 Ausgänge zu realisieren sind, dürfte einen Anzahl über 32 sinnvoll sein. Dies ist aber nicht zwingend erforderlich, da evtl. auch I^2C oder SPI zum Anschluss der Ein- und Ausgaben zum Einsatz kommen können. Da das Gesamtsystem schließlich in ein Gehäuse eingebaut werden soll, ist auf eine diesbezüglich günstige Schnittstellenverteilung zu achten. Da die Ansprüche insgesamt nicht allzu hoch zu sein scheinen, sollte der Logikbaustein auch nicht besonders teuer ausfallen.

Vorauswahl

Tabelle 6 zeigt das Ergebnis einer ersten Einschätzung der Eignung verschiedener bekannter, gut verfügbarer Einplatinencomputer als Logikbaustein für das zu entwickelnde Gerät.

Gerät		Anmerkung / Grund für Ausschluss
Arduino	x	– Mikrocontroller – nicht einmal ansatzweise die benötigten Schnittstellen
Banana Pi	?	– Kopie vom Raspberry Pi
BeagleBoard / BeagleBone	ok	– bestes Model: BeagleBone Black – 65 digitale GPIO Pins
Cubieboard	x	– neueste Version (Cubieboard 4) besitzt keine GPIO Pins mehr
Ethernut	x	– speziell für Ethernet-Anwendungen – bietet kaum andere Schnittstellen
Orange Pi	?	– Kopie vom Raspberry Pi
Raspberry Pi	ok	– bestes Model: Raspberry Pi 2 model B – nur 26 GPIO Pins – nicht quelloffene BLOBs enthalten

Tabelle 6: Erste Einschätzung verschiedener Einplatinencomputer
Quellen: [2], [26], [49], [50]

Gemäß dieser Einschätzung kommen nur das BeagleBone Black und der Raspberry Pi 2 in Frage. Die Kopien des Raspberry Pi, das Banana Pi und das Orange Pi, unterscheiden sich im Wesentlichen nur durch eine höhere Rechenleistung vom Original, was für das zu entwickelnde Gerät keinen entscheidenden Vorteil darstellt. Beide Geräte bringen im vorliegenden Fall keine nennenswerten Vorteile gegenüber dem Original. Es sind hingegen einige Nachteile vorhanden. Dazu zählen unter anderem eine ungünstigere Verteilung der Schnittstellen, eine schlechtere Verfügbarkeit und eine geringere Bekanntheit, also eine kleinere Entwicklergemeinde und damit weniger verfügbarer Software.

Vergleich zwischen Raspberry Pi 2 und BeagleBone Black

Es müssen im Folgenden also die Eigenschaften des BeagleBone Black und des Raspberry Pi 2 miteinander verglichen werden um das geeigneter Gerät auszuwählen.

Gerät	Raspberry Pi 2	BeagleBone Black
CPU	ARM Cortex-A7	ARM Cortex-A8
Frequenz	900 MHz	1000 MHz
Anzahl Kerne	4	1
Speicher	1 GB	512 MB
Anzahl USB 2.0 Ports	4	1
Video Ausgabe	HDMI, composite video	Micro-HDMI
Speicher für Betriebssystem	MicroSD Karte	4 GB eMMC / MicroSD Karte
Netzwerk	10 / 100 MBit Ethernet	10 / 100 MBit Ethernet
Anzahl GPIO Pins (digital)	26	65
frei nutzbare GPIO Pins	24 (UART)	32 (eMMC, HDMI)
Anzahl I^2C	1	2
Anzahl SPI (Anzahl CS)	1 (2)	2 (jeweils 2)
Anzahl UART	1	4
Anzahl analoger Eingänge	0	7
Preis	ca. 36 €	ca. 60 €

Tabelle 7: Vergleich zwischen Raspberry Pi 2 und Beagle Bone Black
Quellen: [2], [26], [51], [52], [49]

Tabelle 7 zeigt einen Vergleich der im vorliegenden Fall wesentlichen Eigenschaften beider Geräte. Die entscheidenden Vorteile sind in grün markiert. Der wesentliche Vorteil des BeagleBone Black ist die größere Anzahl an digitalen GPIO Pins, die für das zu entwickelnde Gerät sehr nützlich sein dürften. Allerdings wird dieser Vorteil dadurch verringert, dass nur etwa 32 der 65 Pins gleichzeitig mit HDMI und dem verbauten eMMC Speicher genutzt werden können. Auch wenn der eMMC Speicher evtl. deaktiviert werden könnte um weitere Pins nutzen zu können, werden die meisten durch das HDMI Interface belegt, welches auch zum Einsatz kommen soll und damit nicht deaktiviert werden kann. Damit dürfte die Anzahl der zur Verfügung stehenden Pins trotz der hohen Gesamtanzahl nicht ausreichend sein. Somit dürften unabhängig von der Wahl des Geräts zusätzliche Bausteine nötig sein um die Anzahl der Ein- und Ausgabepins zu erhöhen. Die Anbindung dieser könnte über I^2C, SPI oder UART erfolgen. Es wäre auch ein Einsatz von Latches zum zeitlichen Multiplexing mit einer Steuerleitung denkbar. Jedenfalls stellt dadurch die höhere Anzahl der GPIO Pins am BeagleBone Black keinen entscheidenden Vorteil dar, da sie immer noch zu gering ist. Die höhere Anzahl an I^2C, SPI und UART Schnittstellen dürfte beim zu entwickelnden Gerät nicht erforderlich sein.

Der Raspberry Pi bringt einige Vorteile mit sich. Er hat eine Multi-Core CPU, was zwar nicht als ein erforderliches, aber durchaus als vorteilhaftes Kriterium festgelegt wurde. Er hat genügend USB Schnittstellen um alle benötigten Geräte gleichzeitig anschließen zu können und einen normalen HDMI Anschluss. Dieser bietet zwar nicht mehr Funktionen, aber es ist davon auszugehen, dass Benutzer dafür eher Kabel besitzen, was ihnen den Kauf von Adaptern erspart. Als weiterer Vorteil kann auch der deutlich geringere Preis angesehen werden. Darüber hinaus weißt der Raspberry Pi 2 eine vorteilhaft erscheinende Anordnung der Anschlüsse auf. Es sind nur an zwei Seiten der Platine Anschlüsse vorhanden. Dies dürfte den Einbau in ein Gehäuse zusammen mit zusätzlichen Komponenten erleichtern. Ein unangenehmes Gefühl verursacht das Vorhandensein von nicht quelloffenen BLOBs. Diese werden aber nur zum Betreiben der GPU eingesetzt, welche für das zu entwickelnde Gerät nicht von großer Bedeutung sein dürfte. Daher verhindert dieser kleine Nachteil nicht die Entscheidung zugunsten des Raspberry Pi 2.

3.4.9. HW-SW Interaktion

Nachdem der allgemeine Aufbau, das Betriebssystem und der Logikbaustein, der zum Einsatz kommen soll, bekannt sind, scheint es an dieser Stelle sinnvoll einige grundlegende Überlegungen bezüg-

lich der Interaktion zwischen Hard- und Software anzustellen. Im Hinblick auf eine ereignisbasierte Steuerung und auch um nicht unnötig Rechenleistung zu verbrauchen, sollten grundsätzlich alle Ereignisse in Hardware mittels Interrupts an die Software gemeldet werden. Dies gilt insbesondere für Zustandsänderungen an den digitalen Eingängen, für das Auftreten von Fehlersituationen und allgemein für alle Zustandsänderungen in der Hardware, die für die Software von Interesse sein könnten, aber auch für Ereignisse, die Übertragungen mittels I^2C, SPI oder UART betreffen. Es sollte also beispielsweise auch das Vorhandensein von Daten im Empfangspuffer der Schnittstellen oder der Abschluss einer Übertragung signalisiert werden. Es stellt kein Problem dar, wenn in Software die Abfrage der aktuellen Zustände der Eingänge initiiert werden muss, wenn die Software mittels Interrupt über das Vorhandensein einer Änderung informiert wird. In jedem Fall muss die Notwendigkeit zum Polling zur Überwachung von Eingabeänderungen vermieden werden.

3.4.10. Steuerungsbibliothek

Da die Steuerung, wie eben beschrieben, sehr stark mit der Reaktion auf Interrupts verknüpft ist und überhaupt sehr hardwarenah ist, könnte es vorteilhaft sein, die gesamte Steuerungssoftware im Kernel-Modus laufen zu lassen. Es wäre möglich die gesamte Steuerung als Kernel-Modul, praktisch als Gerätetreiber, auszuführen. Dieses könnte dann bei Bedarf dynamisch geladen werden. Dies hätte durchaus Vorteile, es wäre schneller, genauer gesagt mit geringeren Latenzzeiten verbunden und es wäre eine direkte Interaktion mit der Hardware ohne zusätzliche Abstraktion durch Treiber möglich. Eine besonders schnelle Reaktion auf Ereignisse wäre möglich indem man den nötigen Code direkt in die Interrupt-Behandlungsroutine implementiert, welche aufgerufen wird um die Ereignisse zu verarbeiten. Dies könnte je nach Umfang zwar die Latenzzeiten für andere Operationen verlängern, aber das wäre in diesem Fall nicht weiter schlimm.

Dieses Vorgehen wäre aber nicht besonders benutzerfreundlich, da die Entwicklung von Software, die im Kernel-Bereich laufen soll im Allgemeinen deutlich aufwendiger ist, als die von Software für den Benutzer-Bereich. Angesichts einer geforderten maximalen Reaktionszeit von 5 ms scheint dies nicht gerechtfertigt. Es scheint viel sinnvoller die gesamte Steuerungssoftware als normalen Prozess im Benutzer-Bereich ablaufen zu lassen und mit der Hardware über mehr oder weniger abstrahierende Treiber zu interagieren. Dies reduziert die Komplexität bei der Erstellung von Steuerungssoftware deutlich und verkürzt damit wesentlich die Einarbeitungszeit. Ein wesentlicher Vorteil ist, dass bei Programmierfehlern nicht gleich das ganze System abstürzt und wesentlich einfacheres Debugging möglich ist.

Nutzer, die Forschung mit dem Gerät betreiben, für die lange Einarbeitungszeiten kein größeres Problem darstellen, die die volle Kontrolle über die Hardware haben wollen und den vollen Funktionsumfang des Geräts nutzen wollen, können direkt gegen die Schnittstelle der Treiber, die zur Interaktion mit der Hardware eingesetzt werden, implementieren oder im Extremfall sogar wirklich Steuerungssoftware im Kernel-Bereich implementieren. Es gibt aber auch viele Nutzer, für die eine kurze Einarbeitungszeit entscheidend ist. Es ist davon auszugehen, dass die meisten von ihnen nicht den vollen Funktionsumfang der Hardware ausreizen müssen und nicht die volle Kontrolle benötigen. Sie benötigen eher eine einfache Schnittstelle, die keine lange Einarbeitungszeit benötigt, aber dennoch die wesentlichen Funktionen bereitstellt um Steuerungsprogramme mit ihr zu erstellen. Es muss also eine Steuerungsbibliothek bereitgestellt werden, die eine solche Schnittstelle exportiert und die Interaktion mit den eingesetzten Treibern übernimmt ohne dass der Nutzer die Details der Interaktion und der Hardware kennen muss.

API

Die wesentlichen Funktionen, die bereitzustellen sind, sind das Setzen von Ausgängen und das Einlesen von Eingängen. Darüber hinaus muss eine Methode bereitstehen um auf Eingabeänderungen zu

reagieren ohne den aktuellen Zustand der Eingaben periodisch prüfen zu müssen. Für den Fall, dass ein Fehler bei einem Funktionsaufruf auftritt, muss dies zurückgemeldet werden und es muss eine Möglichkeit geben, ggf. weitere Information zu dem Fehler zu erhalten um ihn beheben zu können.

```
int read_pin(int port, int pin, int *value);
int read_port(int port, unsigned char *value);
int write_pin(int port, int pin, int value);
int write_port(int port, unsigned char value);
int wait_for_pin_change(int port, int pin, int *new_value, int time_out);
int wait_for_port_change(int port, unsigned char *new_value, int time_out);
char *format_error_message();
```

Codeausschnitt 3: Erster Entwurf einer API

Codeausschnitt 3 zeigt einen ersten Entwurf für die Schnittstelle der zu entwickelnden Steuerungsbibliothek für C / C++, was, abgesehen von den IEC 61131-3 Sprachen (siehe dazu Abschnitt 3.4.11), die beiden zur Entwicklung von Steuerungssoftware am ehesten eingesetzten Sprachen sein dürften. Die Namen der Funktionen sind selbsterklärend, read_pin und read_port dienen zum Einlesen des Wertes eines digitalen Eingangs bzw. aller digitalen Eingänge eines Ports. Der Begriff Port bezieht sich auf eine der geplanten Centronics-Buchsen, also auf 8 digitale Eingänge und 8 digitale Ausgänge. Die Funktionen write_pin und write_port setzen entsprechend den Wert eines digitalen Ausgangs bzw. aller digitalen Ausgänge eines Ports. Die beiden wait_* Funktionen blockieren so lange bis sich der Wert eines bestimmten Eingangs bzw. irgendeines Eingangs an einem der beiden Ports ändert und liefern dann den neuen Wert zurück. Diese Funktionen könnten auf verschiedenste Weisen eingesetzt werden. Ihre wesentliche Eigenschaft ist es, dass sie die Notwenigkeit Eingänge periodisch auf Veränderungen prüfen zu müssen eliminieren. Eine maximale Wartezeit bei diesen Funktionen dürfte notwendig sein um nicht unendlich zu blockieren, wenn sich ein Eingang nie ändert. Offensichtlichstes Beispiel für dieses Problem ist der Wunsch das Steuerungsprogramm zu beenden während auf eine Eingabeänderung gewartet wird. Gäbe es keine Beschränkung der Wartezeit könnte das Programm nicht sauber beendet werden ohne vorher eine Eingabeänderung herbeizuführen, was offensichtlich nicht praktikabel ist, da die Software üblicherweise keinen Einfluss auf die Eingänge hat.

Fehlerbehandlung

Schon an dieser Stelle interessant ist es ein grobes Konzept zur Fehlerbehandlung zu entwerfen. Jede Funktion sollte konsequent einen Rückgabewert liefern, der Auskunft darüber gibt, ob bei der Ausführung ein Fehler aufgetreten ist. Es wäre möglich gleich weitere Informationen zu ggf. aufgetretenen Fehlern zu liefern, allerdings müsste dann beim Aufruf jeder Funktion ein Puffer für diese Informationen übergeben werden. Alternativ könnte ein Puffer bei Bedarf angelegt werden, aber dies wäre sehr anfällig für Speicherlecks, da man sich auf die Nutzeranwendung verlassen müsste, dass diese den Puffer wieder freigibt.

Sinnvoller scheint es den zuletzt aufgetretenen Fehler intern zu speichern und eine Funktion, hier format_error_message genannt, bereitzustellen, welche das Anwendungsprogramm aufrufen kann, wenn es denn weitere Informationen zu einen Fehler wünscht. Es scheint sinnvoll diese ausführlichen Informationen gleich in nutzerlesbarer Form zu liefern, damit sie direkt zur Anzeige einer Fehlermeldung genutzt werden können.

3.4.11. Entwicklungs- und Laufzeitumgebung

Die eben im Abschnitt 3.4.10 beschriebene Steuerungsbibliothek soll eine Möglichkeit bieten die Hardware des Geräts einfach in Steuerungssoftware, welche in C/C++ erstellt wird, anzusprechen. Steuerungssoftware für SPSen wird aber üblicherweise in einer der IEC 61131-3 Sprachen erstellt. Bei neueren verteilten Systemen dürfte auch eine Entwicklung gemäß IEC 61499 wünschenswert sein. Es

sollte also eine Möglichkeit geschaffen werden, das Gerät auch gemäß dieser Standards zu programmieren.

Dazu sind üblicherweise eine integrierte Entwicklungsumgebung (IDE) und eine passende Laufzeitumgebung notwendig. Die Entwicklungsumgebung muss Texteditoren für textuelle Sprachen und grafische Editoren für grafische Sprachen bereitstellen. Diese müssen verschiedenste Komfortfunktionen, wie die Hervorhebung von Syntax und Befehlsvervollständigung unterstützen. Die IDE muss in der Lage sein Fehler im erstellten Programm zu erkennen und dem Benutzer Hilfestellung bei der Korrektur bieten. Sie muss ferner natürlich in der Lage sein, das Programm in geeigneter Weise zu kompilieren und an das Gerät bzw. die Laufzeitumgebung zu übertragen. Wird das Programm erst einmal ausgeführt, muss Debugging durch die Anzeige von Variablenwerten unterstützt werden. Dabei ist es auch wünschenswert Variablen, insbesondere die Werte von Eingängen, manuell auf einen festen Wert setzen zu können. Damit kann man Eingaben beim Testen simulieren oder einen defekten Sensor überbrücken.

Es steht außer Frage, dass es unsinnig wäre eine neue IDE speziell für das zu entwickelnde Gerät zu implementieren. Bestehende IDEs sind weitestgehend geräteunabhängig und können zur Erstellung von Steuerungsprogrammen für verschiedene Geräte eingesetzt werde. Es sollte auch nicht nötig sein eine neue Laufzeitumgebung zu implementieren. Bei dieser könnte aber eine Portierung auf das neue Gerät notwendig sein.

Gemäß Anforderungen muss die gesamte auf dem Gerät eingesetzte Software quelloffen sein, dies gilt insbesondere für die Laufzeitumgebung. Vorzugsweise sollte auch die IDE quelloffen sein, auch wenn diese nicht zwangsläufig auf dem Gerät selbst zum Einsatz kommt. Beide Teile sollten, insbesondere zu Forschungs- und Ausbildungszwecken, kostenlos genutzt werden dürfen und natürlich sowohl zyklische, als auch ereignisorientierte Steuerungssoftware unterstützen. Die IDE sollte eine grafische Oberfläche aufweisen und Debugging unterstützen.

All diese Kriterien werden von der Entwicklungsumgebung 4DIAC zusammen mit der zugehörigen Laufzeitumgebung FORTE erfüllt. Beide Teile sind quelloffen und können unentgeltlich genutzt werden. Diese haben sogar den großen Vorteil, dass sie, im Gegensatz zu kommerziell entwickelten Lösungen, direkt aus dem Forschungsbereich stammen, also direkt von Mitgliedern der Zielgruppe des zu entwickelnden Geräts entwickelt und bevorzugt eingesetzt werden. Außerdem sind sie damit immer auf dem neuesten Stand der Technik und unterstützen ggf. experimentelle Funktionen, welche nach erfolgreicher Erprobung erst später in kommerzielle Systeme integriert werden.

Integration

4DIAC ist eine Eclipse-basierte IDE, welche die Implementierung von Steuerungssoftware gemäß IEC 61499 erlaubt. Am Gerät sollen pro Port jeweils 8 Eingänge und 8 Ausgänge bereitstehen. Die einzelnen digitalen Ein- und Ausgänge sind logisch gesehen vollkommen unabhängig voneinander. Es besteht an dieser Stelle kein Anlass mehrere Ein- und/oder Ausgänge in irgendeiner Weise zu gruppieren. Damit hat man 16 einzelne unabhängige digitale Eingänge und 16 einzelne digitale Ausgänge. Digitale Ein- und Ausgänge mit einer Bitbreite von nur einem Bit werden in 4DIAC üblicherweise durch sog. IX bzw. QX Funktionsblöcke dargestellt. Diese sind implementiert und sollten auch für das zu entwickelnde Gerät eingesetzt werden können. Es besteht kein Anlass zu der Annahme, dass irgendwelche Modifikationen an 4DIAC notwendig sein sollten um das zu entwickelnde Gerät zu unterstützen. Gemäß Anforderungen sollen auch Möglichkeiten zur Erkennung und Diagnose von Fehlersituationen angeboten werden. Ob dies mit einfachen IX und QX Blöcken realisiert werden kann, muss geprüft werden, wenn die genaue Realisierung feststeht, aber vorerst spricht nichts dagegen, die Information, dass ein bestimmter Fehler vorliegt, wie eine digitale Eingabe zu betrachten.

FORTE ist die zugehörige Laufzeitumgebung zu 4DIAC. Sie unterstützt bereits Linux als darunterliegendes Betriebssystem, wurde bereits auf dem Raspberry Pi getestet und unterstützt eine Reihe verschiedener Geräte. Für jedes unterstützte Gerät enthält FORTE ein Modul. Zur Kompilierungszeit kann neben weiteren Optionen das entsprechende Modul ausgewählt werden um die Laufzeitumgebung speziell für das jeweilige Gerät zu erstellen.

Unter anderem gibt es auch ein Modul für das Raspberry Pi, welches es erlaubt, die GPIO Pins des Raspberry Pi als digitale Ein- und Ausgänge zu verwenden. Da insgesamt mehr Ein- und Ausgaben realisiert werden sollen als GPIO Pins am Raspberry Pi zur Verfügung stehen, wird eine direkte Ansteuerung der Ein- und Ausgänge über GPIO Pins nicht möglich sein. Damit dürfte auch der Einsatz des bereits vorhandenen Moduls nicht in Frage kommen, aber es sollte problemlos möglich sein für das zu entwickelnde Gerät ein eigenes davon unabhängiges Modul zu erstellen. Da FORTE in C++ implementiert ist, sollte das neue Modul für den Hardwarezugriff die zuvor angesprochene Steuerungsbibliothek verwenden können, was die Implementierung des Moduls einfach halten sollte.

3.4.12. Schutzmaßnahmen

Viele der geforderten Schutzmaßnahmen, wie Überspannungs-, Verpolungs-, ESD-, Überlast- und Kurzschlussschutz, sowie der Schutz gegen Programmierfehler können erst in der Implementierungsphase genauer betrachtet werden. An dieser Stelle werden nur einige grundlegende Überlegungen zur Unterdrückung von Störungen angestellt. Die genaue Realisierung kann natürlich auch erst in der Implementierungsphase festgelegt werden.

Im Forschungs- und Bildungsbereich, wo die zu entwickelnde Steuerung eingesetzt werden soll, sind deutlich geringere Störungen als im industriellen Umfeld zu erwarten, da hier weniger Geräte, aber vor allem deutlich leistungsschwächere Geräte und deutlich kürzere Leitungen eingesetzt werden. Speziell bei den Lernsystemen, wo die meisten Aktuatoren mit Druckluft anstatt elektrisch angetrieben werden, sind nur schwache elektrische Störungen zu erwarten. Dennoch sollten Möglichkeiten zur Unterdrückung von ggf. auftretenden Störungen an den digitalen Eingängen und zur Verhinderung der Ausbreitung von Störungen vorgesehen werden.

Störungsunterdrückung an digitalen Eingängen

Sinnvoll scheint es alle digitalen Eingänge mit Filtern zu versehen, die Störungen unterdrücken oder zumindest hinreichend reduzieren. Angesichts der geforderten maximalen Reaktionszeit auf Eingabeänderungen von 5 ms ist davon auszugehen, dass Änderungen an den Eingängen nur mit relativ geringen Frequenzen, unterhalb des kHz Bereichs auftreten und jeder Zustand für eine relativ lange Zeit, wenigstens im niedrigen ms Bereich gehalten wird. Kürzere Eingabeänderungen im µs Bereich können somit als Störungen angesehen werden.

Gemäß dieser Festlegung wären also Filter, welche Eingabeänderungen, welche nur kurz andauern herausfiltern, geeignet um Störungen zu unterdrücken ohne die Funktionalität zu beeinträchtigen. Für die meisten Anwendungen dürften Filterzeiten von einer oder einigen ms gut geeignet sein. Es ist aber zu berücksichtigen, dass sich dadurch alle Reaktionszeiten um die Filterzeit verlängern. Um keinen Kompromiss zwischen guter Störungsunterdrückung und kurzen Reaktionszeiten eingehen zu müssen, sollte eine Möglichkeit bestehen die Filter abzuschalten, wenn sie nicht benötigt werden. Idealerweise sollte eine Möglichkeit geschaffen werden die Filterdauer einzustellen, damit diese immer optimal den aktuellen Bedürfnissen angepasst werden kann.

Auch wenn die genaue Implementierung dieser Filter an dieser Stelle noch nicht von Interesse ist, sollte schon mal die grundlegende Frage geklärt werden, ob diese Filter in Software oder in Hardware implementiert werden sollen. Eine Implementierung in Software hat den Vorteil, dass sie leicht ange-

passt werden kann, insbesondere kann die Filterlänge beliebig geändert werden. Bringt aber auch den Nachteil einer potenziell extrem hohen CPU-Belastung mit sich. Diese muss entweder periodisch die Eingänge prüfen oder alternativ eine Interruptbehandlung bei jeder Änderung des Eingabewertes durchführen. Beide Möglichkeiten können zu einer hohen CPU-Belastung führen.

1. Eine periodische Prüfung müsste relativ häufig durchgeführt werden um die Reaktionszeit nicht übermäßig zu verlängern und gleichzeitig eine Filterung sicherzustellen. Bevor eine Reaktion erfolgen könnte, müsste ja mehrmals derselbe Eingabewert gelesen werden.

2. Eine auf Interrupts basierte Behandlung würde normalerweise zu keiner hohen CPU-Belastung führen, da tatsächliche Eingabeänderungen nur selten, allerhöchstens alle paar Millisekunden, zu erwarten sind. Durch Störungen verursachte Eingabeänderungen könnten aber deutlich häufiger auftreten und könnten so zu einer hohen, praktisch nicht vorhersagbaren CPU-Belastung führen.

Es wird klar, dass eine Software-basierte Unterdrückung von kurzen Eingabeänderungen durch den Raspberry Pi nicht praktikabel ist. Die Unterdrückung solcher Störungen muss entweder analog oder durch dedizierte Chips in Hardware erfolgen. Evtl. wäre auch eine Filterung durch die Firmware eines Mikrokontrollers denkbar, wenn dieser vielleicht ohnehin aus anderen Gründen verbaut werden sollte, aber dies wäre ebenfalls nicht optimal, da dieser dauerhaft stark belastet werden würde, was einen recht leistungsstarken Mikrokontroller voraussetzen würde, welcher für andere Aufgaben vielleicht gar nicht notwendig wäre.

Verhinderung der Ausbreitung von Störungen

Um der Ausbreitung von Störungen entgegenzuwirken, sollten die digitalen Ein- und Ausgänge von empfindlichen Bauteilen, insbesondere vom Raspberry Pi, galvanisch getrennt werden. Ideal wäre es auch einzelne Ein- und Ausgänge untereinander zu isolieren. In wieweit dies praktikabel ist, muss in der Implementierungsphase geklärt werden. Es ist schon jetzt klar, dass nicht jeder einzelne Ein- bzw. Ausgang isoliert sein kann, da pro Stecker 8 Ein- und 8 Ausgänge vorhanden sind, aber nur 4 Pins mit Masse und 4 Pins mit 24 V Stromversorgung, welche jeweils auch noch miteinander verbunden sind.

4. Implementierung

Dieses Kapitel befasst sich mit der konkreten Implementierung der einzelnen Teile des zu entwickelnden Geräts und deren Kombination zu einem Gesamtsystem. Grob betrachtet ist eine Kompaktsteuerung mit dem Raspberry Pi 2 als zentraler Recheneinheit, 16 digitalen Ausgängen und 16 digitalen Eingängen zu implementieren. Die Eingänge sind mit Störungsfiltern zu versehen und alle digitalen Ein- und Ausgänge sind auf 24-poligen Centronics Buchsen bereitzustellen. Für jeden Ein- und Ausgang ist eine Leuchtanzeige vorzusehen. Für Eigenentwicklungen und spätere Ergänzungen ist eine Erweiterungsschnittstelle mit GPIO Pins, sowie seriellen Schnittstellen, insbesondere I^2C und/oder SPI und/oder UART zu integrieren. Alle entwickelten Teile müssen in geeigneter Weise an den Raspberry Pi angebunden werden. Dabei ist darauf zu achten, dass alle Zustandsänderungen der Software mittels Interrupts mitgeteilt werden können um ein Polling seitens der Software, welche auf dem Raspberry Pi läuft, zu vermeiden. Schließlich muss der Raspberry Pi zusammen mit den zusätzlich entwickelten Komponenten in einem kompakten Gehäuse untergebracht werden.

Als Betriebssystem ist ein Linux zu installieren und mit dem CONFIG_PREEMPT_RT Patch zu versehen. Dann ist eine Steuerungsbibliothek für C/C++ zu implementieren, die den Hardwarezugriff übernimmt und dem Nutzer eine einfache Schnittstelle mit angemessenem Abstraktionsgrad bereitstellt. Schließlich ist eine Integration in FORTE vorzunehmen. Diese sollte auf die zuvor entwickelte Steuerungsbibliothek zurückgreifen.

4.1. Eingänge

Im Gesamtsystem sind die Ein- und Ausgabeschaltungen, die Signale von den Automatisierungsanlagen verarbeiten und für diese Steuersignale erzeugen, die zentralen und gleichzeitig komplexesten Teile, die entwickelt werden müssen, da diese vielen äußeren Faktoren unterliegen. Sie müssen jeweils eine ganze Reihe von Funktionen mit ganz bestimmten teils durch Normen festgelegten Eigenschaften erfüllen und müssen dabei mit nicht genau bekannten, potenziell ungünstigen Eigenschaften von angeschlossenen Geräten, Bedienungsfehlern, wie Kurzschlüssen und Anschlussfehlern und Fehlfunktionen angeschlossener Geräte zurechtkommen. Da die Ein- und Ausgabeschaltungen voneinander weitestgehend unabhängig sind, ist die Reihenfolge von deren Entwicklung nicht von Bedeutung. Es wird mit der Eingabeschaltung begonnen.

Aus Kapitel 3 ist bekannt, dass 16 Strom aufnehmende Eingänge mit Kennlinien und Schaltverhalten entsprechend Typ 2 oder Typ 3 gemäß IEC 61131-2 zu implementieren sind. Es sind dabei einige Schutzfunktionen zu integrieren. Es muss sichergestellt werden, dass auch bei langsamen oder ungültigen Eingabesignalen kein unkontrolliertes Schalten auftritt. Alle Eingänge müssen mit konfigurierbaren, insbesondere abschaltbaren Eingangsfiltern versehen werden, gegen elektrostatische Entladungen geschützt sein und Verpolungen tolerieren. Eingänge müssen von der restlichen Schaltung insbesondere dem Raspberry Pi galvanisch getrennt sein. Eine galvanische Trennung von Untergruppen von Eingängen ist auch wünschenswert. Änderungen an den Eingabesignalen sind dem Raspberry Pi mittels Interrupt mitzuteilen und alle Eingänge sind mit Leuchtanzeigen zu versehen.

4.1.1. Vorauswahl eines Eingangsbausteins

Da Strom aufnehmende Eingänge mit einer Kennlinie und einem Schaltverhalten gemäß IEC 61131-2 auch für andere Geräte, insbesondere natürlich andere SPSen notwendig sind, werden integrierte Schaltungen angeboten, die eine Reihe solcher Eingänge bereitstellen. Es bietet sich an einen solchen integrierten Baustein auch in dem zu entwickelnden Gerät einzusetzen. Um die Komponentenanzahl möglichst gering zu halten, sollten möglichst viele der geforderten Zusatzfunktionen, insbesondere Eingangsfilter, Interrupterzeugung und galvanische Isolierung bereits in diesem Baustein enthalten

sein. ESD-Schutz, Verpolungsschutz und eine Anschlussmöglichkeit für Leuchtanzeigen sind nicht so wichtig, da diese relativ leicht mit wenigen externen Komponenten hinzugefügt werden können.

Firma	Modell	Eingangs-filter	galvanische Isolierung	Interrupt-erzeugung	Kommunikation	
					seriell	direkt
Infineon	ISO1I811T	konfigurierbar	ja	nein	SPI	ja
Infineon	ISO1I813T	konfigurierbar	ja	nein	SPI	nein
Maxim	MAX31910	konfigurierbar	nein	nein	SPI	nein
Maxim	MAX31911	konfigurierbar	nein	nein	SPI	nein
Maxim	MAX31912	konfigurierbar	nein	nein	SPI	nein
Maxim	MAX31913	konfigurierbar	nein	nein	SPI	nein
Maxim	MAX31914	konfigurierbar	nein	-	nein	ja
Maxim	MAX31915	konfigurierbar	nein	-	nein	ja
ST	CLT01-38S4	N/A	nein	nein	SPI	nein
ST	CLT01-38SQ7	nein	nein	nein	SPI	nein
ST	CLT3-4B	nein	nein	-	nein	ja
ST	SCLT3-8BQ7	konfigurierbar	nein	nein	SPI	nein
ST	SCLT3-8BT8	konfigurierbar	nein	nein	SPI	nein
TI	SN65HVS880	konfigurierbar	nein	nein	SPI	nein
TI	SN65HVS881	konfigurierbar	nein	nein	SPI	nein
TI	SN65HVS882	konfigurierbar	nein	nein	SPI	nein
TI	SN65HVS885	konfigurierbar	nein	nein	SPI	nein

Tabelle 8: Vergleich verschiedener Eingabebausteine
Quellen: [17], [53], [54], [55] und [56]

Tabelle 8 zeigt einen Vergleich verschiedener Eingabebausteine hinsichtlich der hier besonders interessanten Funktionen. Einige bieten eine direkte Kommunikation an. Darunter ist zu verstehen, dass es zu jedem Eingang einen fest zugeordneten Ausgang gibt, der stets den aktuellen binären Wert des Eingangs reflektiert. Sofern ein Filter vorhanden ist, ist der Wert am Ausgang natürlich gefiltert. Bei dieser direkten Kommunikation ist eine Erzeugung eines Interrupts über eine separate Interruptleitung bei einer Änderung einer Eingabe nicht erforderlich, da jede Änderung sofort über den zugehörigen Ausgang angezeigt wird.

Die meisten integrierten Bausteine bieten aber SPI als Kommunikationsschnittstelle an. Die Eingabewerte werden über diese auf Anfrage serialisiert übertragen. Bei dieser Übertragungsmethode wäre eine separate Interruptleitung wünschenswert, über die eine Änderung irgendeiner Eingabe angezeigt wird, um gezielt die Änderung abfragen zu können. Ohne diese Funktion, müssen die Eingaben periodisch abgefragt werden um sie auf Änderungen zu prüfen. Leider bietet keiner der betrachteten Chips eine separate Interruptleitung.

Da keiner der Chips eine Interrupterzeugung unterstützt, muss diese in jedem Fall extern nachgerüstet werden. Damit spielen bei der Auswahl des Chips hauptsächlich das Vorhandensein eines Eingabefilters und das Vorhandensein einer galvanischen Isolierung eine Rolle, da diese sonst auch extern ergänzt werden müssten. Eingangsfilter weisen fast alle betrachteten Chips auf, eine galvanische Isolierung allerdings nur die beiden Chips von Infineon, was diese beiden zu den am besten geeigneten Bausteinen macht.

4.1.2. Vergleich zwischen ISO1I811T und ISO1I813T von Infineon

Wie eben festgestellt erscheinen die beiden Eingabebausteine ISO1I811T und ISO1I813T von Infineon am besten für das zu entwickelnde Gerät geeignet. Beide müssen aber noch detaillierter verglichen werden um herauszufinden, welcher von beiden zum Einsatz kommen soll.

Modell	ISO1I811T	ISO1I813T
Eingabeseite		
Anzahl Eingänge	8	8
als Typ 2 konfigurierbar	ja	ja
als Typ 3 konfigurierbar	ja	ja
optimierte Kennlinie	ja	ja
Hysterese	ja	ja
Eingangsfilter	10 µs - 10 ms (4 Schritte)	4 µs - 20 ms (9 Schritte)
Filterkonfiguration	in Hardware	in Software über SPI
Betriebsspannung	9,6 V - 35 V	9,6 V - 35 V
Abtastrate	100 kHz	500 kHz
Kabelbrucherkennung	nein	ja
Verpolungsschutz	ja	ja
ESD-Schutz (CDM / HBM)	bis 1,5 kV / 2,5 kV	bis 1,5 kV / 2,5 kV
Isolation		
Isolierspannung	500 V	500 V
Übertragungstechnik	induktiv	induktiv
Ausgabeseite		
Betriebsspannung	2,85 V - 5,5 V	2,85 V - 5,5 V
Diagnoseausgang	ja	ja
Fehlerdiagnose	nur Diagnoseausgang	Statusregister über SPI lesbar
serielle Schnittstelle	SPI	SPI
direkte Kommunikation	ja	nein

Tabelle 9: Vergleich zwischen ISO1I811T und ISO1I813T
Quellen: [17] und [53]

Tabelle 9 zeigt einen Vergleich zwischen den beiden Chips von Infineon. Der ISO1I813T hat im Allgemeinen einige Vorzüge. Er hat eine höhere Abtastrate, eine Kabelbrucherkennung, eine bessere Fehlerdiagnose und in Software konfigurierbare Eingangsfilter mit mehr Filterlängeneinstellungen.

Im vorliegenden Fall stellt die höhere Abtastrate aber keinen nennenswerten Vorteil dar, da eine Abtastrate von 100 kHz bereits eine Abtastung im Abstand von 10 µs sicherstellt, was im Verhältnis zur geforderten Reaktionszeit von 5 ms hinreichend erscheint.

Eine Kabelbrucherkennung kann vor allem im industriellen Umfeld sehr hilfreich sein, allerdings wurde bereits im Abschnitt 3.2.6 festgestellt, dass bei den bevorzugt eingesetzten Lernsystemen ausschließlich 3-Draht-Sensoren und elektromechanische Schaltgeräte zum Einsatz kommen. Diese verursachen im Aus-Zustand keinen signifikanten Stromfluss durch den Eingang der SPS. Daher kann bei diesen ein Kabelbruch ohne zusätzliche Bauteile ohnehin nicht erkannt werden. Es wäre zusätzlich ein hochohmiger parallel zum Schalter angebrachter Widerstand erforderlich, welcher auch im Aus-Zustand einen geringfügigen Stromfluss aufrechterhält und so eine Kabelbrucherkennung möglich macht. Da somit eine Kabelbrucherkennung bei den bevorzugt eingesetzten Automatisierungsanlagen normalerweise ohnehin nicht möglich ist, stellt eine Kabelbrucherkennung im vorliegenden Fall nur einen geringfügigen Vorteil dar.

Wenn die Kabelbrucherkennung nicht genutzt wird, beschränkt sich die Fehlerdiagnose auf folgende Fehler:

- Fehlen einer Versorgungsspannung für den Eingabeteil
- Zu geringen Versorgungsspannung für den Eingabeteil
- Interner Übertagungsfehler zwischen den isolierten Teilen

Das Auftreten eines internen Übertragungsfehlers dürfte ziemlich unwahrscheinlich sein und ob keine oder nur eine zu geringe Versorgungsspannung vorliegt, läuft in etwa aufs Gleiche hinaus. Damit genügt im Prinzip ein Bit um anzuzeigen, dass einer dieser Fehler vorliegt. Somit ist im Wesentlichen eine einzige Diagnoseleitung ausreicht. Damit stellt die verbesserte Fehlerdiagnose im vorliegenden Fall ebenfalls nur einen geringen Vorteil dar.

Die Möglichkeit die Eingabefilter genauer und vor allem in Software einstellen zu können stellt damit den wesentlichen Vorteil des ISO1I813T Chips für das zu entwickelnde Gerät dar. Ein großer Nachteil ist hingegen das Fehlen einer direkten Kommunikation. Da es keine separate Interruptleitung gibt und Eingaben nur über SPI abgefragt werden können, besteht die einzige Möglichkeit zur Erkennung einer Änderung darin die Eingaben periodisch zu prüfen. Eine wichtige Anforderung besteht aber darin, dass der Raspberry Pi die Eingaben eben nicht periodisch auf Änderungen prüfen soll, sondern über das Vorhandensein aller Änderungen über Interrupts informiert wird.

Die Verwendung des ISO1I813T Chips würde also zwangsläufig den Einsatz eines Mikrokontrollers oder eines vergleichbaren Chips erforderlich machen, der periodisch die Eingaben per SPI abfragt und auf Änderungen prüft. Wenn dieser eine Änderung feststellen sollte, müsste er den Raspberry Pi mittels Interrupt darüber informieren, dass neue Daten vorliegen und diese auf Anfrage übermitteln. Nach jetzigem Entwicklungsstand des Geräts scheint es unwahrscheinlich, dass ein Mikrokontroller für andere Teile erforderlich sein wird. Dieser würde also nur dazu dienen periodisch Eingaben zu prüfen und bei Bedarf einen Interrupt zu erzeugen. Dies scheint nicht ideal.

Der ISO1I811T Chip bietet hingegen eine direkte Kommunikation an, so dass der Zustand der acht Eingabeleitungen direkt auf acht Ausgabeleitungen übertragen werden kann. Dadurch eröffnet sich die Möglichkeit die Ausgabeleitungen direkt an GPIO Pins des Raspberry Pi anzuschließen oder, wenn nicht genügend freie GPIO Pins vorhanden sein sollten, einen einfachen IO-Expander einzusetzen, welcher die acht Ausgaben des ISO1I811T Chips parallel einliest und bei Änderungen einen Interrupt an den Raspberry Pi übermittelt, welcher die neuen Daten dann über SPI oder I^2C vom Expander abfragen kann. Dies ist in jedem Fall einfacher als einen Mikrokontroller einzusetzen und für diesen Firmware zu entwickeln. Ein Mikrokontroller wäre definitiv sinnvoll, wenn dieser noch mehr Funktionen übernehmen würde, aber wenn eine Interrupterzeugung bei einer Eingabeänderung die einzige benötigte Funktion ist, ist ein direkter Anschluss oder die Nutzung eines IO-Expanders vorzuziehen.

Sofern sich in der weiteren Entwicklung kein Bedarf für den Einsatz eines Mikrokontrollers ergibt, scheint der ISO1I811T Chip im vorliegenden Fall besser geeignet, auch wenn damit auf die zuvor genannten Vorzüge des ISO1I813T Chips verzichtet werden muss. [17] [53]

4.1.3. Infineon ISO1I811T Chips

Die ISO1I811T Chips von Infineon bieten pro Chip 8 digitale Eingänge. Viele der gewünschten Funktionen sind bereits in diesen Chips integriert. Alle Eingänge sind mit einer Hysterese versehen, wodurch auch Eingabesignale mit langsamen Übergängen und ungültige Eingabesignale, die sich dauerhaft in der Nähe des Schaltpunkts aufhalten, verarbeitet oder zumindest toleriert werden können. Die Chips tolerieren an den Eingängen Spannungen von -45 V bis 45 V, womit kein zusätzlicher Verpolungsschutz notwendig ist. Eine galvanische Trennung zwischen Ein- und Ausgabeseite ist ebenfalls schon integriert und die Kennlinie ist hinsichtlich des Energieverbrauchs optimiert, wie dies im Abschnitt 2.4.1 im Unterabschnitt Kennlinienoptimierung beschrieben wird. Konfigurierbare Eingangsfilter sind vorhanden, wie diese konfiguriert werden und wie genau sie funktionieren muss noch im Detail betrachtet werden. Eingänge können sowohl als Typ 2 als auch als Typ 3 konfiguriert werden. Die Entscheidung, welcher Typ eingesetzt werden soll, wurde in der Entwurfsphase vertagt und muss nun getroffen werden.

Ein ESD-Schutz ist vorhanden, die angegebenen Spannungswerte erscheinen aber recht gering. Es muss noch geprüft werden, ob der integrierte Schutz ausreichend ist oder ob noch zusätzlicher ESD-Schutz hinzugefügt werden muss. Eine Möglichkeit Interrupts bei Eingabeänderungen zu erzeugen, ist, wie gesagt, nicht vorhanden, diese Funktion muss also in jedem Fall extern realisiert werden. Es scheint aber sinnvoll, die Anbindung an den Raspberry Pi und die Erzeugung von Interrupts erst später zu betrachten, wenn auch die Ausgabebeschaltung bekannt ist, da diese ebenfalls an den Raspberry Pi angebunden werden muss. Am Chip ist zu jedem Eingang an der Eingabeseite ein zugehöriger Ausgang für den Anschluss einer LED, welche anzeigt, ob am Eingang aktuell ein Ein- oder ein Aus-Signal anliegt, vorhanden. Es muss untersucht werden, welche LEDs an dieser Stelle zur Anzeige des Zustandes genutzt werden können. [17]

Typ der Eingänge

Die Kennlinien und das Schaltverhalten aller Eingänge des ISO1I811T Chips können so konfiguriert werden, dass sie entweder dem Typ 2 oder dem Typ 3 gemäß IEC 61131-2 entsprechen. Pro Eingang sind dazu zwei externe Widerstände notwendig und zusätzlich gibt es einen entsprechenden Konfigurationseingang, der über einen Widerstand mit Masse verbunden wird (vgl. Abschnitt 4.1.6). Je nach gewünschtem Typ sind die Widerstandswerte entsprechend zu wählen. Eine Umschaltung zwischen beiden Typen mit einem Schalter oder Ähnlichem ist nicht praktikabel, da die Werte aller Widerstände angepasst werden müssten. Es müssen immer alle acht Eingänge vom gleichen Typ sein.

Zu beachten ist, dass aufgrund des höheren Stromflusses und der damit verbundenen höheren Hitzentwicklung beim Einsatz von Typ 2 Eingängen immer zwei Eingänge des Chips parallel geschaltet werden müssen, womit effektiv nur vier Eingänge pro Chip zur Verfügung stehen. Beim Einsatz von Typ 3 Eingängen stehen die vollen acht Eingänge pro Chip zur Verfügung. Schon in der Entwurfsphase wurde festgestellt, dass ohne größere Einschränkungen Typ 3 Eingänge verbaut werden können. Der Einsatz von Typ 2 Eingängen würde dazu führen, dass statt einem Eingabebaustein pro Port gleich zwei Bausteine nötig wären. Dies spricht klar gegen Typ 2 Eingänge. Damit kann nun endlich festgelegt werden, dass Typ 3 Eingänge zu realisieren sind. [17]

Eingangsfilter

Jeder der acht Eingänge des ISO1I811T Chips verfügt über einen digitalen Filter. Die Filterlänge kann für alle Eingänge gemeinsam in Hardware über zwei Konfigurationspins eingestellt werden. Leider verbietet das Datenblatt eine Änderung der Konfiguration zur Laufzeit, so dass eine Ansteuerung der Pins durch die Steuerungssoftware vom Raspberry Pi aus nicht realisiert werden kann. Die Einstellung der Filterlänge muss erfolgen bevor das Gerät eingeschaltet wird. Die Einstellung muss vom Benutzer einfach ohne spezielles Werkzeug durchgeführt werden können. Eine Realisierung mittels Jumper, auch als Kurzschlussbrücken bezeichnet, oder mit kleinen Schalteren scheint angemessen.

Über die zwei Konfigurationspins können Filterlängen von etwa 0 ms, 1,25 ms, 4 ms und 12,48 ms eingestellt werden. Um zu verstehen, was genau die Filterlänge aussagt, ist es notwendig die Funktionsweise der Filter zu betrachten. Pro Eingang gibt es einen eignen unabhängig arbeitenden Filter. Jeder Filter hat einen interner Zähler, welcher Werte von 0 bis N-1 annehmen kann. Konfiguriert wird über die Konfigurationspins der Wert von N. Er kann auf 1, 125, 400 oder 1248 gesetzt werden. Im Betrieb wird jeder Eingang mit etwa 100 kHz abgetastet, also etwa alle 10 µs. Jedes Mal wenn am Eingang gerade, gemäß eingestelltem vom Typ des Eingangs abhängigen Schaltpunkt unter Berücksichtigung der Hysterese, ein Ein-Signal anliegt, wird der interne Zähler um eins hochgezählt. Liegt ein Aus-Signal an, wird um eins runter gezählt. Ist die obere Grenze bereits erreicht, wenn ein Ein-Signal gelesen wird, erfolgt keine Änderung des Zählers. Selbiges gilt für die untere Grenze. Wird der Wert 0 erreicht, wird die Ausgabe auf den logischen Wert 0 gesetzt, dieser Ausgabewert wird beibehalten, bis irgendwann durch wiederholtes Erkennen eines Ein-Zustandes N-1 erreicht wird. Dann erst

wird der Wert des Ausgangs auf die logische 1 gesetzt. Dieser Wert wird wieder so lange beibehalten, bis irgendwann durch wiederholtes Lesen des Aus-Zustandes wieder 0 erreicht wird. Das Vorgehen wird in Abbildung 14 veranschaulicht. Oben wird der Verlauf eines Eingabesignals, bereits unter Berücksichtigung des Schaltpunkts und der Hysterese digitalisiert, aufgezeigt. Darunter der sich entsprechend verändernde Wert des internen Zählers. Der Wert von N ist in diesem Beispiel um die Übersichtlichkeit zu verbessern auf 4 gesetzt. Gang unten ist der Verlauf des Ausgangs angezeichnet. Die vertikalen gestrichelt dargestellten Linien markieren die Abtastzeitpunkte, die Zeit verläuft im Diagramm von links nach rechts.

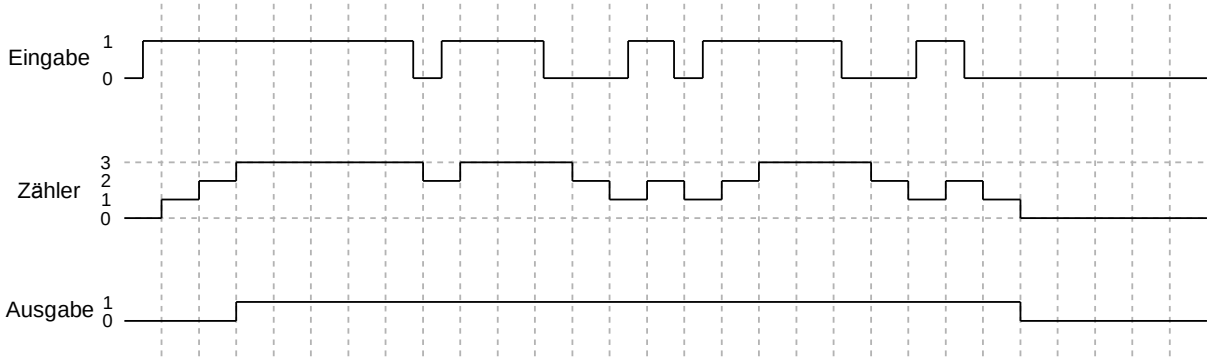


Abbildung 14: Funktionsprinzip der Eingabefilter in ISO1I811T Chips

Wie zu erkennen ist, liefert dies ein sehr gutes Filterverhalten, da nach einem Wechsel des Ausgabewertes mindestens N-1-mal der entgegengesetzte Wert eingelesen werden muss, bevor die Ausgabe wieder umgeschaltet wird. Kürzere isoliert auftretende Änderungen an der Eingabe werden vollständig unterdrückt. Die Reaktionszeit auf eine Eingabeänderung verlängert sich, wenn keine Störungen auftreten, genau um die Filterlänge. [17]

4.1.4. ESD-Schutz

Objekte und insbesondere auch Personen können elektrostatisch aufgeladen werden. Kommen diese dann mit anders geladenen oder insbesondere auch geerdeten also ungeladenen Objekten in Kontakt, kommt es zu einem Ladungsausgleich in Form einer elektrostatischen Entladung (ESD), wobei teils sehr hohe Spannungen und Stromstärken auftreten. Bei einem starken Ladungsunterschied kann die Verbindung sogar über die Luft unter Auftreten von Funken erfolgen. Fließt die Ladung über ein elektrisches Gerät, kann dieses durch die hohe Spannung und Stromstärke beschädigt oder zerstört werden. Viele moderne CMOS-basierte Bauteile können allein aufgrund der hohen auftretenden Spannung zerstört werden, weniger empfindlicher Bauteile werden eher durch die auftretende hohe Stromstärke zerstört.

Eine elektrostatische Entladung kann ein Bauteil oder ein Gerät während der Benutzung aber auch schon während der Produktion oder bei der Wartung treffen. Daher müssen einzelne Bauteile und Geräte entsprechend geschützt werden um eine Beschädigung bzw. Zerstörung zu vermeiden. Es wird bei den Schutzmaßnahmen zwischen der Komponentenebene und der Systemebene unterschieden. Auf der Komponentenebene geht es um den Schutz der einzelnen Bauteile während der Herstellung des Geräts, auf der Systemebene geht es um den Schutz des Gesamtsystems während des Betriebs.

Komponentenebene

Auf der Komponentenebene geht es, wie gesagt, um den Schutz eines einzelnen Bauteils während der Produktion. Die Produktion erfolgt in einer kontrollierten Umgebung, wo häufig sogar spezielle Maßnahmen zur Reduktion der Aufladung von Maschinen und Mitarbeitern getroffen werden. Es werden üblicherweise spezielle Bodenbeläge und Schuhe, sowie spezielles Werkzeug verwendet. Dadurch

treten insgesamt nur relativ geringe Aufladungen auf. Allerdings kann eine Entladung über jeden beliebigen Pin eines Bauteils erfolgen, da es kein schützendes Gehäuse gibt und daher alle Pins frei zugänglich sind.

Üblicherweise werden Schutzmaßnahmen, welche einer Zerstörung während der Produktion vorbeugen sollen in die Bauteile integriert. Standards definieren verschiedene Modelle um die Toleranz eines Bauteils gegenüber statischen Entladungen zu testen und Grenzwerte anzugeben. Üblicherweise verwenden Hersteller von Bauteilen eines oder mehrere der folgenden drei Modelle um die Toleranzschwellen für den in ihren Bauteilen integrierten ESD-Schutz anzugeben.

- Das Charged Device Model (CDM) geht davon aus, dass das Gehäuse des Bauteils selbst geladen ist und während der Produktion eine Entladung auf ein geerdetes leitfähiges Objekt erfolgt. Typischerweise wird hierbei von Spannungen zwischen 250 V und 750 V ausgegangen.
- Das Machine Model (MM) geht von einem geladenen metallischen Werkzeug aus, das über das zu testende Bauteil, welches geerdet ist, entladen wird. Hierbei geht man typischerweise von nur 100 V bis 200 V aus.
- Das Human Body Model (HBM) geht von einer elektrostatisch geladenen Person aus, die das betrachtete geerdete Bauteil berührt und so über dieses entladen wird. Spannungen zwischen 500 V und bis zu 2 kV sind üblich.

Die Standards definieren jeweils auch den zeitlichen Verlauf der Entladung. Zu beachten ist, dass hierbei bei allen Tests von der Produktion und nicht dem Betrieb ausgegangen wird. Da alle Bauteile während der gesamten Produktion, abgesehen vielleicht von einem abschließenden Funktionstest, abgeschaltet sind, werden die entsprechenden Tests ebenfalls im abgeschalteten Zustand durchgeführt. Die Reaktion eines Bauteils auf eine elektrostatische Entladung im abgeschalteten Zustand kann dabei von der Reaktion im eingeschalteten Zustand abweichen.

Systemebene

Auf der Systemebene geht es um den Schutz des Gesamtsystems während der Benutzung. Die Benutzung erfolgt üblicherweise in einer weitestgehend unbekannten und vor allem unkontrollierten Umgebung. In dieser muss mit deutlich stärkeren elektrostatischen Aufladungen gerechtet werden. Es werden im Alltag Spannungsunterschiede von teils bis zu 30 kV erreicht. Allerdings sind die meisten Geräte von einem schützenden Gehäuse umgeben. Die meisten Anschlüsse von Bauteilen sind damit nicht von außen zugänglich und können so in aller Regel keinen statischen Entladung ausgesetzt werden. Typischerweise sind nur einige wenige Anschlüsse von außen zugänglich. Nur diese müssen besonders geschützt werden. Dazu werden üblicherweise externe Schutzschaltungen verwendet. Es wäre prinzipiell möglich die benötigten Schutzschaltungen in die zu schützenden Bauteile zu integrieren, allerdings wird dies nur selten gemacht, da dafür physikalisch bedingt eine große Fläche auf dem Chip nötig wäre, welche meist eher dazu verwendet wird um mehr komplexe Logikschaltungen auf dem Chip realisieren zu können. Die vergleichsweise einfachen Schutzschaltungen werden typischerweise mit spezialisierten externen Bauteilen realisiert.

Schutz bei ISO1I811T Chips

Das Datenblatt des ISO1I811T Chips gibt an, dass dieser gegen elektrostatische Entladungen gemäß Charged Device Model bis 1,5 kV und gemäß Human Body Model bis 2,5 kV geschützt ist. Dies gibt den Schutz auf der Komponentenebene an und ist ausreichend um den Chip während der Produktion zu schützen. Für die normale Benutzung in unkontrolliertem Umfeld ist dies bei Weitem nicht ausreichend. Es muss also auf der Systemebene zusätzlicher Schutz gegen elektrostatische Entladungen vorgesehen werden.

Für diesen werden üblicherweise Überspannungsschutzdioden, welche die Spannung begrenzen indem sie die Ladung gegen Masse ableiten, Kondensatoren, welche die Ladung aufnehmen oder Varistoren, welche ebenfalls die Ladung ableiten, verwendet. Varistoren haben die Nachteile, dass die geschützten Bauteile im Falle einer elektrostatischen Entladung im Vergleich zu den anderen Möglichkeiten höheren Spannungen ausgesetzt werden und die Varistoren nur eine begrenzte Anzahl an Entladungen überleben. Kondensatoren haben den Nachteil, dass sie eine erhebliche zusätzliche Kapazität an der Leitung darstellen. Dies stellt im Wesentlichen nur bei sehr hohen Frequenzen ein Problem dar, weil die Signalqualität dadurch verschlechtert wird. Im vorliegenden Fall sollte der Einsatz von Kondensatoren prinzipiell möglich sein, wobei die Auswirkungen der zusätzlichen Kapazität an den Eingängen unter Berücksichtigung potenziell langer angeschlossener Leitungen mit hoher parasitärer Induktivität zunächst genauer untersucht werden müsste um eventuelle negative Auswirkungen, etwa das Erzeugen von ungewollten Schwingungen, auszuschließen. Überspannungsschutzdioden haben hingegen insgesamt sehr gute Eigenschaften, beschränken die Spannung üblicherweise auf ein deutlich geringeres Maß, als etwa Varistoren, reagieren typischerweise sehr schnell, bringen kaum zusätzliche Kapazität mit sich und haben nur vernachlässigbare Leckströme. Sie haben praktisch keine Auswirkungen auf den normalen Betrieb der Schaltung, weshalb sie zum Schutz der Eingänge des zu entwickelnden Geräts ausgewählt wurden. [17] [57]

4.1.5. Leuchtanzeigen an Eingängen

Wie schon zuvor erwähnt, besitzen die ISO1I811T Chips zu jedem Eingang auf der Eingangsseite einen zugehörigen Ausgang, an den eine LED angeschlossen werden kann um den aktuellen Zustand des Eingangs anzuzeigen. Notwendig sind dazu LEDs mit einer Vorwärtsspannung zwischen 1,9 V und 3 V. Dies ist nicht ungewöhnlich und schränkt die Auswahl an LEDs nicht nennenswert ein.

Der Strom, der gemäß IEC 61131-2 Norm von jedem Eingang im Aus-Zustand aufgenommen werden muss, wird von den ISO1I811T Chips nicht effektiv genutzt, die Energie wird einfach in Form von Wäre abgegeben. Im Ein-Zustand wird die Energie hingegen zum Betrieb der LEDs an den Statusausgängen genutzt. Dies hat den großen Vorteil, dass durch die LEDs kein zusätzlicher Strom verbraucht wird, hat allerdings auch den Nebeneffekt, dass die zur Verfügung stehende Stromstärke, welche von einer LED genutzt werden kann, beschränkt ist. Im Falle eines Typ 3 Eingangs stehen einer LED nur zwischen 1,5 mA und 3,1 mA zur Verfügung. Dies genügt durchaus für kleine LEDs mit geringem Stromverbrauch. Diese leuchten nur schwach, aber ausreichend, wenn man sie direkt beobachten kann und die Umgebungshelligkeit nicht besonders hoch ist.

Die zu entwickelnde Schaltung soll aber zum Schluss in ein Gehäuse integriert werden. Erste Überlegungen bezüglich dieser Integration legen nahe, dass Lichtwellenleiter zum Einsatz kommen werden, welche das Licht von den Statusanzeigen zur Außenseite des Gehäuses leiten. Darüber hinaus sollen die Anzeigen auch bei hellem Tageslicht gut erkennbar sein. Vorbereitende Tests haben ergeben, dass dafür auch bei kurzen Lichtwellenleitern deutlich hellere LEDs notwendig sein werden. Zu deren Betrieb werden mindestens 5 mA bis 10 mA notwendig sein. Damit sind insbesondere die 1,5 mA, welche geliefert werden, wenn das Eingabesignal nur leicht oberhalb des Schaltpunkts liegt, völlig unzureichend. Daher können im vorliegenden Fall die LEDs zur Anzeige leider nicht an die vorgesehenen Anschlüsse angebracht werden.

Das Datenblatt sieht für diesen Fall den Einbau von Widerständen anstelle der LEDs vor, wobei sich dadurch der Schaltpunkt der Eingänge verschieben kann. Da eine Anforderung aber darin besteht ein Schaltverhalten gemäß IEC 61131-2 sicherzustellen und sich das Datenblatt nicht über den Umfang der möglichen Verschiebung des Schaltpunkts äußert, müssen an den vorgesehenen Stellen leider LEDs eingebaut werden um ein normgerechtes Schaltverhalten zu gewährleisten, auch wenn die LEDs nicht zu Anzeigezwecken nutzbar sein werden. Zu Anzeigezwecken müssen an anderer Stelle weitere leistungsstärkere LEDs vorgesehen werden. [17]

4.1.6. Eingangsschaltung

Nachdem feststeht, welche Eingabebausteine zum Einsatz kommen sollen und deren Eigenschaften genauer betrachtet wurde, kann nun ein Schaltplan für den Eingabeteil des Geräts entwickelt werden.

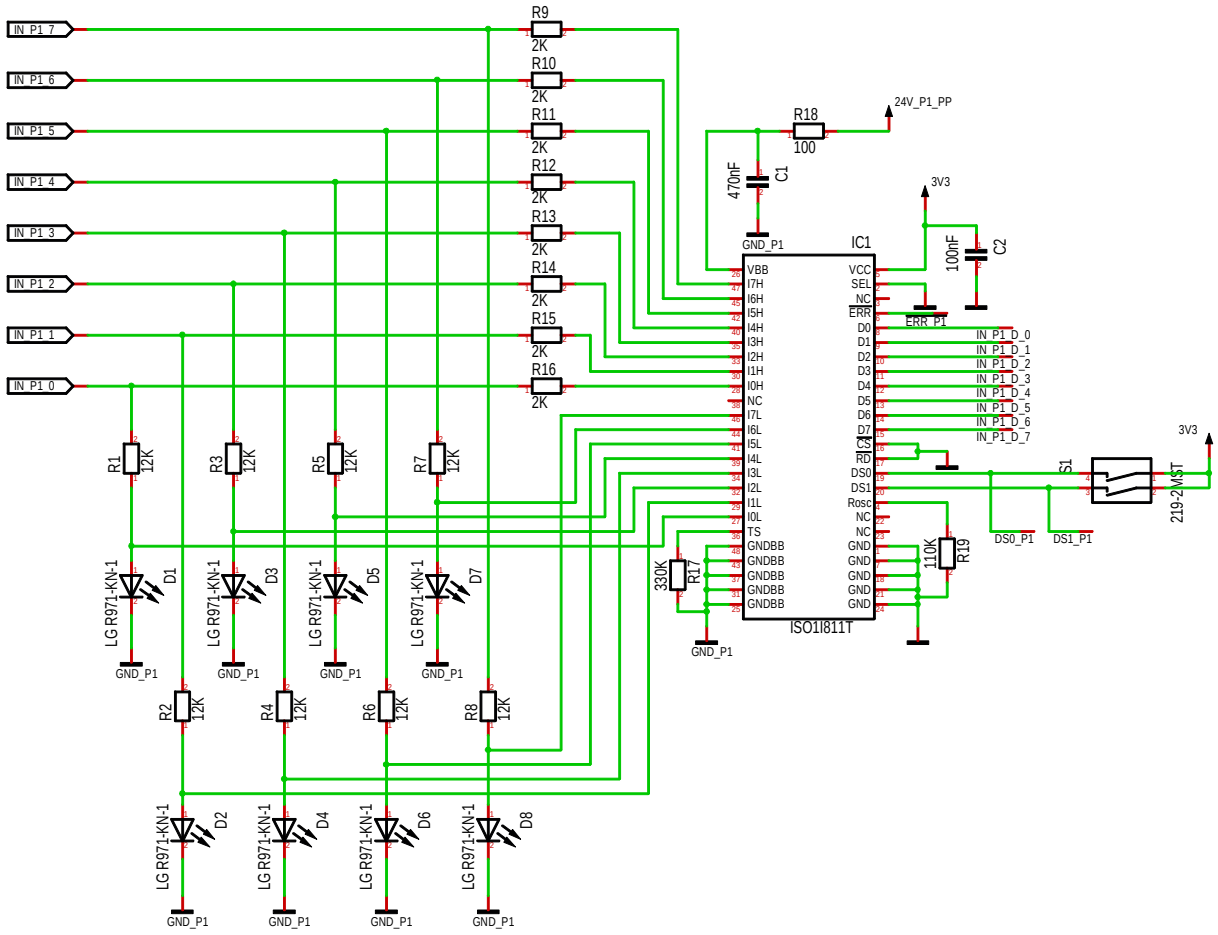


Abbildung 15: Schaltplan der Eingabeschaltung

Abbildung 15 zeigt den basierend auf den bisherigen Ergebnissen entwickelten Schaltplan für die Eingabeschaltung. Dieser umfasst acht digitale Eingänge, also genau so viele wie in einem Port enthalten sein sollen. Für den zweiten Port könnte die gleiche Schaltung ohne jegliche Modifikationen wiederverwendet werden.

Die Schaltung besteht aus einem ISO1I811T Chip als zentralem Baustein und ein wenig externer Beschaltung. Auf der linken Seite befindet sich die Eingangsseite des Chips. Die mit IN_P1_x bezeichneten Leitungen stellen die Eingänge dar, an denen Signale von Sensoren empfangen werden. Diese sind mit geeigneten Überspannungsschutzdioden, welche hier nicht darstellt sind, zu schützen. Die galvanische Trennung verläuft vertikal durch den ISO1I811T Chip, rechts befindet sich die Ausgabeseite des Chips. Die mit IN_P1_D_x bezeichneten Leitungen stellen die Ausgaben der Eingabeschaltung, welche den aktuellen Zustand der Eingänge liefern, dar. Diese und weitere Ausgaben werden im Folgenden noch genauer beschrieben.

Die Widerstände R1 bis R17 wurden gemäß dem Datenblatt des Chips passend für Typ 3 Eingänge gewählt. Die Beschaltung der Eingangsseite ist im Prinzip vollständig gemäß dem Datenblatt erfolgt. Die Kondensatoren C1 und C2, sowie der Widerstand R18 dienen der Entkopplung der Stromversorgung des Chips von der restlichen Schaltung und wurden gemäß dem Evaluation Board [58], welches von Infineon zu den ISO1I811T Chips angeboten wird, gewählt. R19 wurde so gewählt, dass eine

Abtastfrequenz von etwa 100 kHz erzielt wird. Diese Frequenz wird im Datenblatt mehrfach als typisch für diesen Chip angegeben. 125 kHz wäre das Maximum für diesen Chip.

Als Versorgungsspannung auf der Ausgabeseite wurden 3,3 V gewählt, da dies den Logikpegeln der GPIO Pins des Raspberry Pi entspricht und so ein direkter Anschluss der Ausgänge an diese möglich wäre. Am Erweiterungsstecker des Raspberry Pi, welcher voraussichtlich zur Stromversorgung der sich in Entwicklung befindlichen Schaltung genutzt werden wird, stehen sowohl 3,3 V als auch 5 V zur Verfügung. Die verfügbare Stromstärke am 3,3 V Ausgang des Raspberry Pi ist allerdings stark eingeschränkt. Es scheint keine offizielle Angabe dazu zu existieren, aber im Allgemeinen wird von einer maximalen Belastbarkeit von 50 mA [59] ausgegangen. Daher könnte es je nach weiterer Gestaltung der Gesamtschaltung erforderlich werden 5 V als Versorgungsspannung zu nutzten. Damit wäre allerdings kein direkter Anschluss der Ausgänge an GPIO Pins des Raspberry Pi mehr möglich.

LEDs D1 bis D8

Als LEDs D1 bis D8 wurden möglichst billige, kleine LEDs im 0805 Gehäuse gewählt. Deren Eigenschaften, wie Leuchtkraft oder Abstrahlwinkel sind nicht wichtig, da diese nicht der Anzeige dienen und nur notwendig sind um einen präzisen Schaltpunkt zu gewährleisten. Die Wahl von LEDs im 0805 Gehäuse hat den Vorteil, dass eine Vielzahl verschiedener Bauteile, insbesondere auch Widerstände, in dieser Bauform erhältlich ist. Dadurch besteht jederzeit die Möglichkeit die LEDs durch Widerstände zu ersetzen ohne das Platinenlayout anpassen zu müssen. Wenn später also eine geringfügige Verschiebung des Schaltpunkts als akzeptabel angesehen werden sollte, könnten durch den Einsatz von Widerständen anstelle von LEDs Kosten gespart werden, da Widerstände deutlich billiger erhältlich sind.

Eingangsfilterkonfiguration

Die Pins DS0 und DS1 sind Konfigurationseingänge für die Eingabefilter. Beide Pins sind mit integrierten Pull-Down Widerstände versehen. Damit genügen zwei Schalter, mit denen man die Pins mit 3,3 V bzw. 5 V, je nach genutzter Versorgungsspannung, verbinden kann, um die Eingabefilter zu konfigurieren. Eine Anbindung an den Raspberry Pi oder einen anderen Logikbaustein um die Filter zur Laufzeit umkonfigurieren zu können ist, wie schon zuvor festgestellt, nicht realisierbar, da sich die Konfiguration gemäß Datenblatt zur Laufzeit nicht ändern darf.

DS0	DS1	N
offen	offen	1248
geschlossen	offen	400
offen	geschlossen	125
geschlossen	geschlossen	1

Tabelle 10: Konfigurationen der Eingangsfilter
Quelle: Angaben gemäß [17]

Tabelle 10 zeigt die resultierende Konfiguration, speziell den Wert von N, also den Wertebereich des internen Zählers, der Eingangsfilter abhängig von der Schalterstellung.

Ausgaben IN_P1_D_x, ERR_P1, DS0_P1 und DS1_P1

Die Ausgänge D0 bis D7 sind Push-Pull CMOS Ausgänge mit 3,3 V bzw. 5 V Logik. Das Legen des SEL Eingangs auf Masse schaltet den Chip in den parallelen Kommunikationsmodus. Das Legen von !CS und !RD auf Masse schaltet die direkte Kommunikation ein. Damit wird die Ausgabe des Eingabefilters eines jeden Eingangs direkt auf den zugehörigen Ausgang gelegt und kann vom Raspberry Pi oder einem anderen Logikbaustein ausgewertet werden.

!ERR ist ein invertierender open-drain Ausgang mit integriertem Pull-Up Widerstand. Über diesen werden das Fehlen einer Versorgungsspannung und das Auftreten von internen Übertagungsfehlern zwischen den galvanisch isolierten Bereichen gemeldet.

An den DS0_P1 und DS1_P1 Leitungen kann die Einstellung der Eingabefilter abgelesen werden. Die Einstellung wird durch Software zwar nicht geändert werden können, aber es könnte nützlich sein, wenn die Software diese Einstellung zumindest auslesen kann.

Zusammenfassung der Hauptfunktionen

Die entwickelte Schaltung bietet acht mit einer Hysterese versehene Eingänge mit einer optimierten Kennlinie und einem Schaltverhalten das einem Typ 3 Eingang gemäß IEC 61131-2 entspricht. Die Zustände aller Eingänge werden periodisch alle 10 µs eingelesen und an konfigurierbare unabhängige Eingangsfilter weitergeleitet. Die Ausgaben der acht Filter werden direkt über Push-Pull CMOS Ausgänge ausgegeben. Zusätzlich gibt es eine Leitung, über die Fehlerzustände angezeigt werden und zwei Leitungen, über die die Konfiguration der Eingabefilter eingelesen werden kann.

4.2. Ausgänge

In Kapitel 3 wurde festgelegt, dass 16 digitale Strom liefernde Ausgänge, jeweils acht pro Port, zu implementieren sind. Diese sollen einen Bemessungsstrom von 0,5 A pro Ausgang und eine Nennspannung von 24 V aufweisen. Der Leckstrom darf pro Ausgang 0,5 mA nicht übersteigen und der Spannungsabfall darf selbst bei voller Belastung nur maximal 3 V betragen. Alle Ausgänge sind gegen Überspannung zu schützen. Es müssen sowohl die Überspannung, welche beim Abschalten von induktiven Lasten auftritt, als auch die, welche bei elektrostatischen Entladungen auftritt, berücksichtigt werden. Sowohl temporäre als auch dauerhafte Überlastsituationen müssen toleriert werden und dürfen zu keiner Beschädigung des Geräts führen. Die Ausgänge sind von der restlichen Schaltung insbesondere dem Raspberry Pi galvanisch zu isolieren und mit Leuchtanzeigen zu versehen, welche den aktuellen Zustand eines jeden Ausgangs anzeigen.

4.2.1. Vorauswahl eines Ausgabebausteins

Genau wie für Eingänge werden auch für Strom liefernde Ausgänge integrierte Chips angeboten, die viele der benötigten Funktionen enthalten. Es scheint sinnvoll einen derartigen Ausgabebaustein als Grundlage für eine Ausgabeschaltung zu verwenden. Um den am besten geeigneten Baustein auszusuchen, sind zunächst die genauen Anforderungen zu klären.

Wie im Abschnitt 2.4.2 festgehalten ist, sollte der Schutz gegen temporäre Überlast, wie sie beispielsweise beim Einschalten eines Elektromotors entsteht, vorzugweise darin bestehen, dass die Stromstärke einfach auf ein sicheres Maß beschränkt wird. Dies entspricht der Funktionsweise eines Anlaufstrombegrenzers und ermöglicht das Anlaufen des Motors ohne die Stromversorgung zu überlasten.

Diese Art des Überlastschutzes muss in den gewählten Chips integriert sein, da dafür ein Leistungsschalter in Form eines Transistors, bevorzugt eines Feldeffekttransistors, notwendig ist, über den der Stromfluss geregelt werden kann. Ein derartiger Schalter muss im Ausgabebaustein ohnehin enthalten sein um die Ausgänge schalten zu können. Würde man den Überlastschutz extern implementieren, wären also pro Ausgang zwei Leistungsschalter vorhanden, was nicht sinnvoll wäre. Abgesehen davon, dass insgesamt mehr Bauteile nötig wären, würde sich die Verlustleistung erhöhen, da an jedem der Schalter Spannung abfallen würde.

Der gewählte Chip muss einen Bemessungsstrom von mindestens 0,5 A aufweisen. Die Wahl eines Bausteins mit einem höheren Bemessungsstrom ist möglich, allerdings sollte dieser nicht übermäßig hoch sein, da die integrierte Strombegrenzung natürlich immer an den Bemessungsstrom angepasst ist

und ein hoher Bemessungsstrom im Vergleich zur benötigten Leistung eine unangemessen hohe Grenze für die Strombegrenzung zur Folge hätte und im Fehlerfall zwangsläufig auch einen hohen Kurzschlussstrom erlauben würde.

Der Ausgabebaustein muss 24 V als Arbeitsspannung akzeptieren. Die gesetzten Grenzwerte für den Leckstrom und den Spannungsabfall sind recht hoch und sollten daher kein Problem darstellen. Daher werden diese bei der Auswahl eines geeigneten Bausteins zunächst nicht betrachtet. Nachdem ein passender Baustein gefunden wurde, muss lediglich überprüft werden, ob diese Grenzwerte von ihm tatsächlich eingehalten werden. Ein Überspannungsschutz, zum Schutz bei Abschaltvorgängen bei induktiven Lasten sollte einhalten sein, kann aber auch, wie auch ein ESD-Schutz, nachgerüstet werden. Eine integrierte galvanische Isolierung wäre von Vorteil, könnte aber auch extern realisiert werden.

Üblicherweise stehen einer, zwei, vier oder acht Ausgänge pro Chip zur Verfügung. Da pro Port acht Ausgänge benötigt werden, werden im Folgenden zunächst nur Ausgabebausteine mit acht Ausgängen betrachtet. Sollte kein passender gefunden werden, könnten auch noch Chips mit weniger Ausgängen pro Chip betrachtet werden. Es werden nur Chips mit einem Bemessungsstrom, der zwischen 0,5 A und 1,5 A liegt, betrachtet und auch nur Chips bei denen 24 V eine zulässige Arbeitsspannung darstellen.

Firma	Modell	Nenn-strom	galvani-sche Iso-lierung.	Überlastschutz		Übersp.-schutz (ind.)	Diag-nose-ausgang
				tem-porär	dauer-haft		
Infineon	BTS4880R	0,6 A	nein	ja	ja	ja	ja
Infineon	ISO1H801G	0,6 A	ja	ja	ja	ja	nein
Infineon	ISO1H811G	0,6 A	ja	ja	ja	ja	ja
Infineon	ISO1H812G	0,6 A	ja	ja	ja	ja	ja
Infineon	ISO1H815G	1,2 A	ja	ja	ja	ja	ja
Infineon	ISO1H816G	1,2 A	ja	ja	ja	ja	ja
Infineon	ITS42008-SB-D	≈ 0,5 A	nein	ja	ja	ja	ja
Infineon	ITS4880R	0,6 A	nein	ja	ja	ja	ja
ST	ISO8200B	0,7 A	ja	ja	ja	ja	ja
ST	VN808-32-E	1 A	nein	ja	ja	ja	ja
ST	VN808-E	0,7 A	nein	ja	ja	ja	ja
ST	VN808CM-32-E	1 A	nein	ja	ja	ja	ja
ST	VN808CM-E	0,7 A	nein	ja	ja	ja	ja
ST	VNI8200XP	0,7 A	nein	ja	ja	ja	ja

Tabelle 11: Vergleich verschiedener Ausgabebausteine
Quellen: Daten gemäß [60], [61], [62], [63], [64], [65], [66],[67], [68], [69], [70], [71], [72] und [73]

Tabelle 11 zeigt einen Vergleich verschiedener Ausgabebausteine. Einige wenige Bausteine wurden in diesen Vergleich nicht aufgenommen, da sie nur in Gehäusen angeboten werden, die mit den zur Verfügung stehenden Geräten nicht sinnvoll verlötet werden können. Dazu zählen insbesondere Chips in sog. Quad Flat No Leads (QFN) Gehäusen, bei denen die Anschlusspins nicht vom Gehäuse abstehen, sondern an den vier Außenkanten in die Unterseite des Gehäuses integriert sind. [2, Seite /Quad_Flat_No_Leads_Package] Üblicherweise sind diese so integriert, dass ein geringer Teil auch noch an den Seiten des Gehäuses sichtbar ist, wenn das Gehäuse aufliegt. Erfahrungen des Autors aus früheren Projekten zeigen, dass diese Art von Chips zwar durchaus mit einer Lötstation oder einem Heißluftföhn verlötet werden kann, allerdings ist dies sehr mühsam und fehleranfällig, da eine optische Prüfung der Lötstellen kaum möglich ist.

Alle betrachteten Chips besitzen einen Überlastschutz in Form einer Begrenzung der Stromstärke. Die Stromstärke wird immer so lange beschränkt, wie die Überlastsituation andauert. Bei einer dauerhaften Überlastsituation, wie einem Kurzschluss, führt dies allerdings schnell zu einer signifikanten Hitzeentwicklung, da an der Strombegrenzerschaltung ein erheblicher Spannungsabfall vorhanden sein muss um die Stromstärke beschränken zu können. Alle betrachteten Chips sind so konstruiert, dass sie bei einer Überhitzung den betroffenen Ausgang oder alle Ausgänge automatisch abschalten. Dadurch sind sie auch gegen eine dauerhafte Überlast geschützt.

Ein Überspannungsschutz zum Schutz gegen Überspannungen, welche bei Abschaltvorgängen von induktiven Lasten auftreten, ist ebenfalls in allen Chips enthalten. Besonders interessant werden damit für die geplante Schaltung diejenigen Chips, die sowohl eine galvanische Isolierung als auch einen Diagnoseausgang aufweisen. Dies sind die Chips ISO1H811G, ISO1H812G, ISO1H815G und ISO1H816G von Infineon, sowie der ISO8200B von STMicroelectronics.

Zu bemerken ist, dass die ISO1H811G (0,6 A) und die ISO1H815G (1,2 A) Chips, sowie die ISO1H812G (0,6 A) und die ISO1H816G (1,2 A) Chips von Infineon jeweils bis auf den Bemessungsstrom und den Grenzwert für die Strombegrenzung vollkommen identisch sind. Dies hat den Vorteil, dass diese gegeneinander ausgetauscht werden können ohne das Platinenlayout ändern zu müssen. So kann auch noch bei der Bestückung der Platine entschieden werden, ob ein Bemessungsstrom von 0,6 A oder 1,2 A gewünscht ist. Dies ist ein wesentlicher Vorteil, weswegen die Entscheidung zu Gunsten der Chips von Infineon fällt. Fürs erste wird ein Bemessungsstrom von 0,6 A gewählt. Dies ist gemäß der aufgestellten Anforderungen ausreichend und stellt eine geringe Stromstärke im Falle eines Kurzschlusses sicher. Sollte später eine höhere Leistung gewünscht sein, kann der entsprechende Chip mit einem Bemessungsstrom von 1,2 A gewählt werden.

4.2.2. Vergleich zwischen ISO1H811G und ISO1H812G

Der vorhergehende Vergleich hat ergeben, dass entweder der ISO1H811G oder der ISO1H812G Chip zum Einsatz kommen soll. Tabelle 12 zeigt einen detaillierteren Vergleich der Eigenschaften beider Chips, der dazu dienen soll zu klären, welcher der beiden Chips besser geeignet ist. Die Eigenschaften der jeweils kompatiblen Chips mit einem höheren Bemessungsstrom sind mit aufgeführt.

Modell		ISO1H811G	ISO1H815G		ISO1H812G	ISO1H816G
Eingabeseite						
Kommunikation		parallel	parallel		seriell (SPI)	seriell (SPI)
direkte Steuerung		ja	ja		-	-
Arbeitsspannung		3 V - 5,5 V	3 V - 5,5 V		3 V - 5,5 V	3 V - 5,5 V
Isolation						
Technologie		induktiv	induktiv		induktiv	induktiv
Isolierspannung		500 V	500 V		500 V	500 V
Ausgabeseite						
Bemessungsstrom		0,6 A	1,2 A		0,6 A	1,2 A
Widerstand		200 mΩ	200 mΩ		200 mΩ	200 mΩ
Spannungsabfall (Volllast)		120 mV	240 mV		120 mV	240 mV
Arbeitsspannung		11 V - 35 V	11 V - 35 V		11 V - 35 V	11 V - 35 V
Leckstrom (max.)		30 µA	30 µA		30 µA	30 µA
Strombegrenzung (typ.)		1,4 A	3 A		1,4 A	3 A
Abschalttemperatur		135 °C	135 °C		135 °C	135 °C
ESD-Schutz (HBM /CDM)		2 kV / 1 kV	2 kV / 1 kV		2 kV / 1 kV	2 kV / 1 kV

Tabelle 12: Vergleich zwischen ISO1H811G und ISO1H812G
Quellen: Daten gemäß [62], [64], [63] und [65]

Bei beiden Chips und auch bei den funktions- und Pin-kompatiblen Varianten mit einem höheren Bemessungsstrom sind der maximale Spannungsabfall und der maximale Leckstrom erwartungsgemäß weit unterhalb der gesetzten Grenzwerte. Der wesentliche Unterschied der beiden Chips besteht im Kommunikationsinterface auf der Eingabeseite. Der ISO1H811G Chip hat ein paralleles Kommunikationsinterface, welches auch über einen direkten Steuerungsmodus verfügt. Dabei ist jedem Ausgang ein fester Eingang zugeordnet und der Ausgang wird immer entsprechend dem am zugehörigen Eingang anliegenden Wert geschalten. Der ISO1H812G Chip verfügt hingegen nur über eine serielle Schnittstelle über die neue Werte für die Ausgaben übertragen werden können.

Für jeden Ausgang ist eine LED vorzusehen, die den aktuellen Status des Ausgangs anzeigt. Es ist aus zwei Gründen sinnvoll diese LEDs auf der Eingabeseite des Ausgabechips zu platzieren.

1. LEDs, welche sich für derartige Statusanzeigen eignen, haben einen Spannungsabfall von nur 2 V bis 3 V. Es ist energieeffizienter diese mit 3,3 V oder 5 V zu betreiben, als mit 24 V, da die restliche Spannung jeweils auf einen Vorschaltwiderstand abfallen muss. Dies ist auf der Eingabeseite einfacher zu realisieren, da auf der Ausgabeseite nur 24 V zur Verfügung stehen.

2. Bei der Softwareentwicklung könnte es praktisch sein, das Ausgabeverhalten der Steuerungssoftware beobachten zu können ohne eine 24 V Stromversorgung anschließen zu müssen. Dies ist nur möglich, wenn die LEDs auf der Eingabeseite des Ausgabechips platziert werden, da die Schaltung auf der Ausgabeseite des Chips ohne eine 24 V Stromversorgung außer Betrieb ist.

Wenn die LEDs auf der Eingabeseite platziert werden, bedeutet dies aber, dass der gewünschte Zustand der acht Ausgaben so oder so zur Ansteuerung der LEDs auf acht parallelen Leitungen vorhanden sein muss. Wenn der Ausgabebaustein eine direkte Steuerung unterstützen würde, könnten diese acht parallelen Leitungen auch gleichzeitig zur Steuerung der Ausgänge genutzt werden. Für die geplante Schaltung wird folglich der ISO1H811G Ausgabebaustein gewählt, weil dieser im Gegensatz zum ISO1H812G Chip eine solche direkte Steuerung ermöglicht. Sollte später ein höherer Bemessungsstrom gewünscht sein, würde folglich der ISO1H815G zum Einsatz kommen.

4.2.3. Ausgabeschaltung

Nachdem entschieden wurde, welche Ausgabebausteine zum Einsatz kommen sollen und ihre Eigenschaften betrachtet wurden, kann nun eine Ausgabeschaltung entworfen werden. Abbildung 16 zeigt den Schaltplan der entwickelten Ausgabeschaltung. Der ISO1H811G Chip stellt den zentralen Baustein der Schaltung dar, es werden kaum zusätzliche externe Komponenten benötigt, da nahezu die gesamte gewünschte Funktionalität in den Ausgabebausteinen integriert ist.

Die Schaltung enthält acht digitale Strom liefernde Ausgänge. Folglich ist diese Schaltung einmal für jeden Port zu realisieren. Links befindet sich die Eingabeseite des Ausgabechips, rechts die Ausgabeseite, die galvanische Isolierung verläuft vertikal durch den Chip.

Durch Verbinden der Eingänge !CS und !WR mit Masse wird der Chip in den direkten Steuerungsmodus versetzt, so dass mit den Eingängen D0 bis D7 die Ausgänge OUT0 bis OUT7 direkt gesteuert werden können. Über den invertierenden Eingang !DIS können alle Ausgänge auf einmal abgeschaltet werden, der Zustand der Eingänge D0 bis D7 wird dabei ignoriert. !DIAG ist ein invertierender open-drain Ausgang mit integriertem Pull-Up Widerstand. Über diesen zeigt der Ausgabechip Fehlerzustände an. !DIAG wird in den low Zustand geschaltet, wenn an mindestens einem Ausgang eine Überhitzung, also eine Überlastung, festgestellt wird oder die Stromversorgung auf der Ausgabeseite abgeschaltet bzw. die gelieferte Versorgungsspannung zu gering ist. [62]

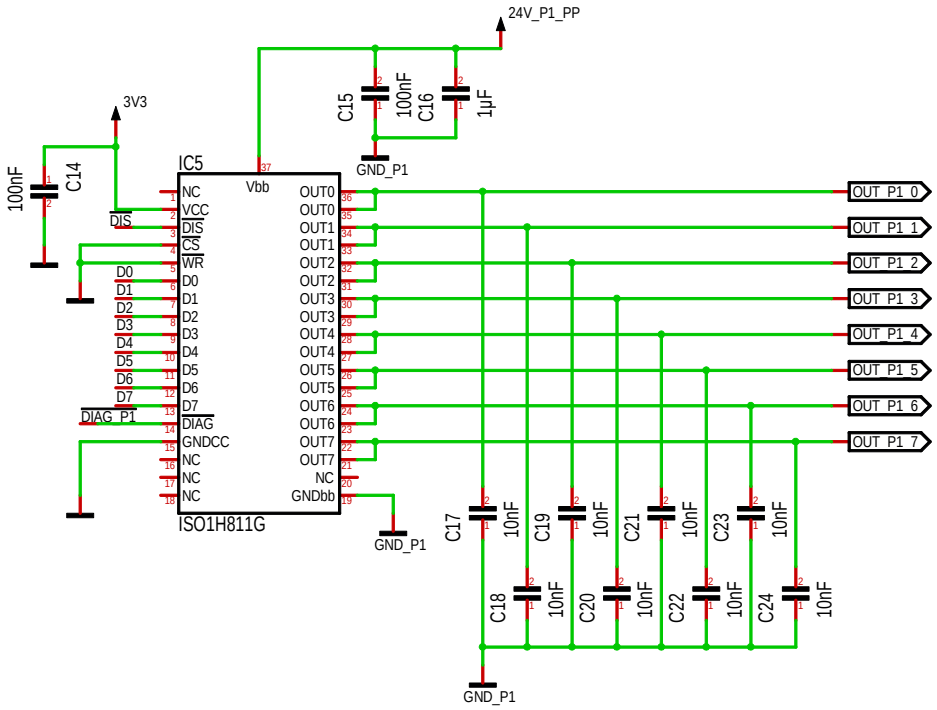


Abbildung 16: Schaltplan der Ausgabeschaltung

C14, C15 und C16 dienen der Entkopplung der Stromversorgung des Chips. Hier wurden die typischen Werte in Anlehnung an das von Infineon veröffentlichte Evaluation Board [74] zu diesen Chips gewählt. Die Kondensatoren C17 bis C24 wurden ebenfalls gemäß dem Evaluation Board gewählt. Diese sollen dazu dienen die Verträglichkeit gegenüber externen elektromagnetischen Störungen zu erhöhen [75, Seite 46]. Gleichzeitig sollten sie zusammen mit dem in den Ausgabechips integrierten ESD-Schutz einen effektiven Schutz gegen elektrostatische Entladungen bieten, indem sie die Ladung von ggf. auftretenden Entladungen aufnehmen.

Als Versorgungsspannung wurden auf der Eingabeseite 3,3 V gewählt, da so die Möglichkeit einer direkten Kommunikation mit dem Raspberry Pi besteht. Alternativ wäre auch hier eine Versorgung mit 5 V möglich. Ein Schutz gegen Überspannungen, welche bei Abschaltvorgängen von induktiven Lasten auftreten, wird von den Chips realisiert indem die Spannung zwischen der Stromversorgung Vbb und jeder einzelnen Ausgabe auf typischerweise 53 V beschränkt wird. Wird eine induktive Last abgeschaltet, wird der Stromkreis zunächst unterbrochen. Aufgrund der Induktivität der Last fließt aber weiterhin Strom durch die Last, so dass am Ausgang des Ausgabebausteins eine negative Spannung entsteht. Dadurch entsteht eine Überspannung über dem Ausgangsschalter, also zwischen der Spannungsversorgung und dem Ausgang. Erreicht diese 53 V (die Spannung am Ausgang also -29 V im Verhältnis zur Masse, bei einer typischen Versorgungsspannung von 24 V), erlaubt der Ausgabechip wieder einen Stromfluss über den Ausgabeschalter um diesen zu schützen. Dabei fällt aber weiterhin eine Spannung von etwa 53 V auf den Ausgabeschalter ab. Dadurch kommt es an dieser Stelle zu einer hohen Verlustleistung, wodurch die in der induktiven Last gespeicherte Energie schnell im Ausgabebaustein abgebaut wird. Dies verkürzt die Schaltzeit von Relais, elektrischen Ventilen und ähnlichem und verlängert ggf. deren Lebensdauer. Durch den Energieabbau kommt es im Ausgabebaustein zu einer entsprechenden Hitzentwicklung. Folglich kann nur eine begrenzte Menge Energie aufgenommen werden. Die gewählten Chips können pro Ausgang bis zu 10 J aufnehmen, vorausgesetzt, dass die Energieaufnahme nicht an mehreren Ausgängen parallel erfolgt und mehrere hintereinanderliegende Ausschaltvorgänge hinreichend lange auseinanderliegen um eine zwischenzeitliche Abkühlung zu ermöglichen. Insgesamt scheint diese Energiemenge (10 J ≈ 24 V · 0,5 A · 0,83 s) sehr großzügig, so dass nur in Ausnahmefällen eine zusätzliche externe Schutzbeschaltung erforderlich werden sollte. [62]

4.3. Anbindung an den Logikbaustein

Nachdem Ein- und Ausgabeschaltungen entwickelt worden sind, muss nun eine Anbindung an den Raspberry Pi entwickelt werden. Dazu muss zunächst geklärt werden, welche und wie viele Ein- und Ausgabesignale vorhanden sind und welche und wie viele Ein- und Ausgänge folglich benötigt werden. Danach müssen die Eigenschaften der Erweiterungsschnittstelle des Raspberry Pi genauer betrachtet werden. Erst dann kann eine geeignete Anbindung entwickelt werden.

4.3.1. Übersicht über Ein- und Ausgabesignale

Zunächst ist festzustellen, dass zur Anbindung der bisher entwickelten Schaltungsteile nur einfache digitale Signale mit 3,3 V CMOS Logik notwendig sind. Tabelle 13 zeigt eine Zusammenstellung aller Eingabesignale, welche in den beiden bisher entwickelten Schaltungen vorkommen. Signale mit ähnlichen Signalnamen und gleicher Funktion sind mit Hilfe von Platzhaltern zusammengefasst. Ein kleines x im Signalnamen steht stellvertretend für den Port und kann den Wert 1 oder 2 annehmen. Ein kleines y steht für eine fortlaufende Nummer von 0 bis einschließlich 7. Zusätzlich zu den hier aufgeführten Eingabesignalen sind noch einige für die geplante Erweiterungsschnittstelle einzurechnen.

Signal	Funktion	Anzahl pro Port	Gesamtanzahl
IN_Px_D_y	externe Eingabe	8	16
!ERR_Px	Diagnose	1	2
DS0_Px	Konfiguration	1	2
DS1_Px	Konfiguration	1	2
!DIAG_Px	Diagnose	1	2
Gesamtanzahl		12	24

Tabelle 13: Zusammenstellung der Eingabesignale

Die Konfiguration der Eingabefilter darf sich während der Laufzeit nicht ändern, folglich dürfen sich auch die Eingabesignale DS0_Px und DS1_Px nicht ändern. Damit ist es nicht erforderlich Änderungen an diesen Leitungen mittels Interrupts an die Software zu melden. Jede Änderung an einer der anderen 20 Leitungen muss aber gemäß den aufgestellten Anforderungen der Software per Interrupt mitgeteilt werden können, da dies eine Änderung einer externen Eingaben oder eine Änderung des internen Zustandes bedeutet. Kalkuliert man minimalistisch 2 zusätzliche Eingabeleitungen für eine Erweiterungsschnittstelle ein, bedeutet dies, dass 22 digitale Eingänge erforderlich sind, die so konfigurierbar sein müssen, dass eine Änderung des Eingabewertes einen Interrupt auslöst.

Signal	Funktion	Anzahl pro Port	Gesamtanzahl
Dy	externe Ausgabe	8	16
!DIS / !DISx	Steuerleitung	1	0 - 2
Gesamtanzahl		9	16 - 18

Tabelle 14: Zusammenstellung der Ausgabesignale

Die in den bisher entwickelten Schaltungen vorhandenen Ausgabesignale sind in Tabelle 14 zusammengefasst. Auch hier repräsentiert ein kleines x die Portnummer, ein kleines y steht hier für eine fortlaufende Nummer von 0 bis einschließlich 15. Für jeden Port gibt es ein !DIS Signal, welches zum gleichzeitigen Abschalten aller Ausgaben dient. Die !DIS Signale der beiden Ports könnten prinzipiell zusammengeschaltet werden. Dann könnten zwar die Ausgaben beider Ports nur gleichzeitig abgeschaltet werden, aber das sollte keine Einschränkung darstellen und würde einen Ausgang einsparen. Alternativ wäre es sogar möglich die !DIS Signale gar nicht zu benutzen, da ein Abschalten aller Ausgaben, auch durch das Setzen aller Dy Leistungen auf low erzielt werden kann. So könnten sogar zwei Ausgaben eingespart werden. Daher ist die Gesamtanzahl der !DIS Signale in der Tabelle mit 0 - 2 angegeben.

Zusätzlich zu den aufgeführten Ausgabesignalen sollten weitere für eine Erweiterungsschnittstelle und für evtl. einzubauende zusätzliche Statusanzeigen eingeplant werden. Es könnte praktisch sein, einige durch die Steuerungssoftware kontrollierbare LEDs zur Statusanzeige vorzusehen. Dies ist aber nicht zwingend erforderlich, da andere Möglichkeiten zur Statusanzeige zur Verfügung stehen. Es kann ein Bildschirm angeschossen werden, Statusinformationen können über die Netzwerkschnittstelle verschickt werden und LEDs zur Anzeige können bei Bedarf auch an die 24 V Ausgänge, welche durch die Steuerungssoftware kontrolliert werden, angeschlossen werden.

Selbst ohne Berücksichtigung der Erweiterungsschnittstelle, ohne zusätzliche steuerbare Statusanzeigen und ohne Nutzung der !DIS Signale werden 24 Eingänge und 16 Ausgänge, also insgesamt 40 Anschlüsse benötigt. Der Raspberry Pi hat allerdings nur 26 GPIO Pins. Damit kommt ein direkter Anschluss aller Signale an GPIO Pins des Raspberry Pi schon mal nicht in Frage.

4.3.2. Eigenschaften der Erweiterungsschnittstelle des Raspberry Pi

Der Raspberry Pi 2 Model B hat eine Erweiterungsschnittstelle in Form einer 40 poligen zweireihigen Stiftleiste mit Rastermaß 2,54 mm. Tabelle 15 zeigt die Pinbelegung dieser Erweiterungsschnittstelle.

	3,3 V	1	2	5 V	
I^2C (SDA)	GPIO2	3	4	5 V	
I^2C (SCL)	GPIO3	5	6	Masse	
	GPIO4	7	8	GPIO14	UART (TxD)
	Masse	9	10	GPIO15	UART (RxD)
	GPIO17	11	12	GPIO18	
	GPIO27	13	14	Masse	
	GPIO22	15	16	GPIO23	
	3,3 V	17	18	GPIO24	
SPI (MOSI)	GPIO10	19	20	Masse	
SPI (MISO)	GPIO9	21	22	GPIO25	
SPI (SCLK)	GPIO11	23	24	GPIO8	SPI (CE0)
	Masse	25	26	GPIO7	SPI (CE1)
I^2C (SDA)	ID_SD	27	28	ID_SC	I^2C (SCL)
	GPIO5	29	30	Masse	
	GPIO6	31	32	GPIO12	
	GPIO13	33	34	Masse	
	GPIO19	35	36	GPIO16	
	GPIO26	37	38	GPIO20	
	Masse	39	40	GPIO21	

Tabelle 15: Pinbelegung des Erweiterungssteckers des Raspberry Pi
Quelle: Belegung gemäß [51]

Es stehen insgesamt 26 frei nutzbare GPIO Pins, zwei 3,3 V Anschlüsse, zwei 5 V Anschlüsse, eine I^2C Schnittstelle mit vordefinierter Funktion und einige Messeanschlüsse zur Verfügung. Die 3,3 V und 5 V Anschlüsse können zur Versorgung von angeschlossenen Schaltungen genutzt werden. Alle 26 frei nutzbaren GPIO Pins, sowie die vorbelegte I^2C Schnittstelle arbeiten mit einer 3,3 V CMOS Logik. Jeder frei nutzbare GPIO Pin kann in Software als hochohmiger Eingang oder als Push-Pull Ausgang konfiguriert werden. Jeder Pin, der als Eingang konfiguriert ist, kann zusätzlich so konfiguriert werden, dass bei einer Änderung ein Interrupt ausgelöst wird.

Von den 26 frei nutzbaren GPIO Pins können 2 (GPIO14, GPIO15) als UART Schnittstelle konfiguriert werden bzw. sind standardmäßig so konfiguriert. Diese beiden Pins sollten folglich frei bleiben, falls eine UART Schnittstelle z.B. zum Kernel-Debugging benötigt wird. Zwei weitere Pins (GPIO2,

GPIO3) können als I^2C Interface konfiguriert werden. Fünf (GPIO7 - GPIO11) der 26 Pins können als SPI Interface mit 2 Chip Select Leitungen genutzt werden. Zusätzlich zu den 26 frei nutzbaren GPIO Pins steht, wie schon erwähnt, eine I^2C Schnittstelle (ID_SD, ID_SC) mit vordefinierter Funktion zur Verfügung. Diese dient dazu einen EEPROM Speicherchip anzuschließen, in dem Informationen über die am Raspberry Pi angeschlossene Schaltung hinterlegt werden können (vgl. Abschnitt 4.7). An jedem Pin kann in Software optional ein interner Pull-Up- oder ein Pull-Down-Widerstand eingeschaltet werden. Bei den Pins GPIO2 bis GPIO8 ist nach dem Start standardmäßig ein Pull-Up-Widerstand aktiviert, bei den Pins GPIO9 bis GPIO27 ist ein Pull-Down-Widerstand aktiviert. [51], [52]

4.3.3. Vorüberlegungen zur Erhöhung der GPIO Anzahl

Zur Anbindung der Ein- und Ausgabeschaltungen ist es offensichtlich notwendig die Anzahl der Ein- und/oder Ausgaben zu erhöhen. Eine einfache Möglichkeit dies zu tun besteht darin IO-Expander-Chip einzusetzen. Diese werden typischerweise über I^2C oder SPI angeschlossen und bieten typischerweise 8 oder 16 zusätzliche digitale GPIO-Pins, von denen jeder üblicherweise, genau wie beim Raspberry Pi, als Ein- oder Ausgabe konfiguriert werden kann, der aktuelle Wert der zusätzlichen GPIO Pins kann dann über die serielle Schnittstelle gelesen bzw. gesetzt werden. Typischerweise besitzen IO-Expander auch einen Interruptausgang, welcher anzeigt, ob es seit dem letzten Lesevorgang eine Änderung an einer der Eingaben gab, so dass die Eingaben nicht periodisch geprüft werden müssen um Änderungen zu erkennen.

IO-Expander könnten entweder zur Erhöhung der Anzahl der Eingänge oder zur Erhöhung der Anzahl der Ausgänge des Raspberry Pi oder für beides eingesetzt werden. Es gibt Gründe, die dafür sprechen IO-Expander im vorliegenden Fall bevorzugt zur Erhöhung der Anzahl von Eingängen einzusetzen. Ein wichtiger Grund ist, dass dies tendenziell zu einer geringeren Anzahl an Interrupts am Raspberry Pi führt, da die Informationen über Änderungen am IO-Expander gebündelt werden. Würden alle Eingabeleitungen direkt angeschlossen werden, könnten beliebig viele Interrupts in unbekannter Abfolge auftreten, was zur Folge hätte, das die entsprechenden Interrupts in Software vorübergehend abgeschaltet werden müssten oder aber mit schwer vorhersagbaren Auswirkungen auf das zeitliche Verhalten gerechnet werden müsste. Werden die Eingabeänderungen aber zunächst an einem IO-Expander gebündelt, können höchstens ein oder zwei Interrupts auftreten, je nachdem ob alle Leitungen an einem Chip angeschlossen sind oder auf zwei aufgeteilt sind. Es können auch keine weiteren Interrupts auftreten bevor ein Interrupt abgearbeitet wurde, da ein IO-Expander das Interruptsignal normalerweise so lange aufrechterhält bis die Eingabeänderung eingelesen wurde

Betrachtet man einen typischen IO-Expander, wie den PCA9538 [76] von Texas Instruments, stellt man fest, dass die Steuerungssoftware unter bestimmten Umständen mit einer unangenehmen Situation konfrontiert werden könnte. Ändert sich der Wert an einem beliebigen Eingang des IO-Expanders wird von diesem ein Interrupt auslösen. Das Interruptsignal wird dann normalerweise so lange aufrechterhalten bis die neue Eingabe über die serielle Schnittstelle eingelesen wurde. Die meisten IO-Expander sind allerdings so konstruiert, dass das Interruptsignal vorzeitig deaktiviert wird, wenn die Eingabe zu ihrem ursprünglichen Wert zurückkehrt bevor der neue Wert gelesen wurde. Erfolgt dann eine Leseoperation wird wieder der alte Wert ausgeliefert. Es besteht keine Möglichkeit mehr festzustellen wodurch der Interrupt ausgelöst wurde.

Dieses Verhalten dürfte in vielen anderen Schaltungen akzeptabel sein, wenn die Änderungen an den Eingaben bestimmten Gesetzmäßigkeiten folgen. Wenn beispielsweise aufgrund der Schaltung ausgesagt werden kann, dass eine Eingabe nie zu ihrem ursprünglichen Zustand zurückkehrt bevor sie verarbeitet wurde, da sie beispielsweise erst in Zuge der Verarbeitung zurückgesetzt wird. Im vorliegenden Fall ändern sich die Eingaben aber aufgrund von externen Ereignissen, die in keinster Weise beeinflusst oder vorhergesagt werden können. Es wäre sehr wohl möglich, dass die Steuerungssoftware mittels Interrupt über eine Änderung an einer Eingabeleitung informiert wird, deren Zustand wieder

auf den alten Wert zurückgesetzt wird, bevor die Steuerungssoftware den neuen Wert ausgelesen hat. Eingaben können sich, wenn der Eingabefilter abgeschaltet ist, also auf eine Filterlänge von 1 gesetzt ist, etwa alle 10 µs ändern. Die Steuerungssoftware kann innerhalb dieser Zeit die neue Eingabe definitiv nicht einlesen, da voraussichtlich schon die Leseoperation länger dauern dürfte. Die Steuerungssoftware könnte also mit der unangenehmen Situation konfrontiert werden, aufgrund eines vorhergehenden Interrupts zu wissen, dass eine kurzzeitige Eingabeänderung aufgetreten ist, aber nicht in der Lage sein, festzustellen, welche Eingabe sich geändert hatte.

Um diese unangenehme Situation zu verhindern, müsste sich der IO-Expander den Grund für das Auslösen des Interrupts merken und es müsste eine Möglichkeit bestehen diese Information abzufragen. So weit bekannt, gibt es nur drei IO-Expander Familien mit einer vergleichbaren Funktionalität. Das sind die IO-Expander mit Agile I/O von NXP, die IO-Expander mit Latching Transition Detection von Maxim Integrated und alle IO-Expander von Microchip.

4.3.4. IO-Expander mit Agile I/O von NXP

Jeder über I^2C oder SPI angebundene Chip verfügt üblicherweise über einige sog. Register. Jedes umfasst typischerweise 8 Bit. Die Kommunikation mit solchen Chips erfolgt indem die Werte der einzelnen Register gelesen und geschrieben werden. Jedes Register und jedes Bit darin haben eine feste Bedeutung. So kann beispielsweise ein digitaler Ausgang auf einen bestimmten Wert gesetzt, indem das zugehörige Bit im zugehörigen Register auf den gewünschten Wert gesetzt wird.

NXP bietet IO-Expander mit und ohne sog. Agile I/O an. IO-Expander ohne Agile I/O verfügen nur über einen sehr begrenzten Satz an Registern und damit nur über eine grundlegende Funktionalität. Das heißt konkret, dass IO-Expander üblicherweise ein Konfigurationsregister besitzen, mit dem jeder Pin als Ein- oder Ausgang konfiguriert werden kann, ein Register über das der aktuelle Wert aller Pins gelesen kann und eines, über das er geschrieben werden kann. Als kleine Komfortfunktion gibt es typischerweise noch ein Register mit dem die Polarität der Ein-/Ausgabepins invertiert werden kann.

IO-Expander von NXP mit Agile I/O verfügen über deutlich mehr Register und damit mehr Funktionalität. Ein Vertreter dieser IO-Expander ist beispielsweise der PCAL9538A Chip, welcher 8 digitale GPIO Pins besitzt, von denen jeder als Ein- oder Ausgang konfiguriert werden kann. Der Chip weißt neben den zuvor beschriebenen grundlegenden Registern eine ganze Reihe zusätzlicher Register auf. Es gibt Register mit denen die von den Ausgängen gelieferte maximale Stromstärke gesteuert werden kann. (Dabei geht es nicht um eine Strombegrenzung im eigentlichen Sinne, vielmehr wird jeder Ausgang von mehreren parallel angeordneten Ausgabetransistoren geschaltet. Es kann konfiguriert werden, wie viele davon tatsächlich genutzt werden sollen. Soll eine relativ starke Last, wie eine LED, geschalten werden, kann die Anzahl der Transistoren erhöht werden um den Spannungsabfall an den Transistoren gering zu halten. Soll hingegen eine schwache Last, wie ein digitaler Eingang, angesteuert werden, kann die Anzahl reduziert werden um die beim Schalten aufgrund von parasitären Kapazitäten auftretenden Stromspitze zu reduzieren und so das Entstehen von Störungen zu verringern. Dies kann notwendig sein, wenn viele Ausgänge gleichzeitig geschaltet werden müssen.) Weitere Register dienen zum optionalen Zuschalten von internen Pull-Up- oder Pull-Down-Widerständen an den GPIO-Pins, zum Maskieren von Interrupts und zum Umschalten einzelner Ausgänge zwischen einem Push-Pull und einem Open-Drain Betrieb. Das Maskieren von Interrupts bedeutet in diesem Fall, dass der Chip so konfiguriert werden kann, dass Änderungen an bestimmten Eingabeleitungen keinen Interrupt zur Folge haben. Zusätzliche gibt es noch ein Register welches angibt, welcher Pin einen Interrupt ausgelöst hat. Es wäre zu prüfen, ob der Wert dieses Registers erhalten bleibt, wenn der Interrupt vorzeitig aufgrund der Rückkehr eines Eingabesignals zu seinem ursprünglichen Wert deaktiviert wird.

Das eigentlich interessante Zusatzregister für die vorliegende Anwendung ist aber ein Register, mit dem tatsächlich konfiguriert werden kann, ob ein Interrupt vorzeitig deaktiviert werden soll, wenn das Eingabesignal, das ihn verursacht hat, wieder zum ursprünglichen Wert zurückkehrt. Entscheidet man sich gegen eine vorzeitige Deaktivierung, wird nicht nur das Interruptsignal so lange aufrechterhalten, bis ein Lesevorgang stattgefunden hat, es wird auch im Register über das die aktuellen Eingabewerte abgerufen werden können, die Änderung, die zu dem Interrupt geführt hat, gespeichert, bis diese tatsächlich eingelesen wurde.

Die IO-Expander Chips mit Agile I/O von NXP sind insgesamt sehr gut und wären in der Lage das bestehende Problem zu lösen. Sie bringen allerdings ein anderes kleines Problem mit sich. Im Datenblatt wird erklärt, dass das Deaktivieren des Interruptsignals jeweils beim Lesen an einer genau definierten Stelle stattfindet. Dabei wird erwähnt, dass Interrupts die zu diesem Zeitpunkt oder unmittelbar davor ausgelöst werden aufgrund des Deaktivierungsvorgangs verloren gehen können. Dieses Problem kann auch durch wiederholtes Lesen nicht behoben werden, da es bei jedem Lesevorgang erneut auftreten kann. Auch wenn das Auftreten dieses Problems sehr unwahrscheinlich sein mag, ist der Verlust von Interrupts und damit von Eingabeänderungen nicht akzeptabel. Damit kommt ein Einsatz eines IO-Expanders von NXP trotz derer insgesamt sehr guten Eigenschaften nicht in Frage. [77]

4.3.5. IO-Expander mit Latching Transition Detection von Maxim Integrated

Die IO-Expander von Maxim Integrated sind insgesamt ziemlich unkonventionell. Im Folgenden wird der MAX7321, ein IO-Expander von Maxim Integrated mit einer sog. Latching Transition Detection betrachtet.

Sein unkonventionelles Design beginnt bereits beim Festlegen der Adresse des Chips am I^2C Bus. Es stehen dafür zwei Konfigurationseingänge zur Verfügung. Normalerweise wären damit vier verschiedene Adresse wählbar, indem man die einzelnen Pins entweder auf low oder auf high setzt. Dieser Chip bietet allerdings zusätzlich die Möglichkeit die Adresspins mit einer der beiden Leitungen des I^2C Bus zu verbinden. Damit stehen pro Pin vier Anschlussmöglichkeiten und damit insgesamt 16 Adressen zur Verfügung.

Ungewöhnlich ist auch, dass die integrierten Pull-Up-Widerstände nicht über das I^2C Interface konfiguriert werden, sondern von der gewählten Adresse des Chips abhängen. Dies hat die beiden großen Vorteile, dass die Konfiguration gleich nach dem Einschalten wirksam ist und dass diese durch eine fehlerhafte Software nicht verändert werden kann. Dies hat aber auch den Nachteil, dass es nicht immer möglich ist, mehrere Chips mit der gleichen Konfiguration von Pull-Ups in einer Schaltung zu betreiben, da manche Pull-Up Kombinationen nur durch eine Adresse erzielt werden können und es am I^2C Bus keine Adresskonflikte geben darf.

Völlig untypisch für einen IO-Expander ist die Tatsache, dass keine Register zur Verfügung stehen, wie dies bei praktisch allen anderen über I^2C angebundenen Chips der Fall ist. Dies wird ermöglicht indem erstens keinerlei Konfigurationsmöglichkeiten bestehen und zweitens alle Pins die sowohl als Ein- und als Ausgabe genutzt werden können nur im Open-Drain Betrieb operieren können. Aufgrund dieser Einschränkungen ist ein Betrieb ohne Register möglich. Jede Leseoperation liefert einfach den aktuellen Wert aller Eingaben, jede Schreiboperation setzt den Wert aller Ausgänge. Können Pins als Ein- und Ausgabe genutzt werden, bewirkt das Schreiben einer 0 das Schalten der Ausgabe auf low, das Schreiben einer 1 bewirkt das Setzten des Ausgangs auf high. Im Falle eines Open-Drain Ausgangs bedeutet dies aber einfach, dass dieser hochohmig wird. Damit kann dieser als Eingabe genutzt werden. Somit können durch Schreiboperationen gleichzeitig Ausgänge gesetzt und Pins als Eingänge konfiguriert werden.

Was diesen Chip für die vorliegende Anwendung interessant macht, ist seine besondere Behandlung von Änderungen an den Eingabeleitungen. Jedes Mal wenn eine Leseoperation durchgeführt wird, wird zunächst ein Byte, welches den zu diesem Zeitpunkt aktuelle Wert aller Eingaben enthält, übertragen, danach kann optional aber noch ein zweites Byte gelesen werden, welches alle Änderungen seit dem vorhergehenden Lesevorgang enthält. Damit könnte die Leitung welche zum Erzeugen eines Interrupts geführt hat, identifiziert werden, auch wenn die Änderung nicht mehr vorhanden ist.

Unglücklicherweise ist Maxim Integrated bei der Implementierung des MAX7321 ein schwerwiegender Fehler unterlaufen, der dazu führt, dass effektiv nur ein solcher Chip pro I^2C Bus verwendet werden kann. [78, Seite 14] Dem Autor dieser Arbeit wurde per E-Mail von Maxim Integrated bestätigt, dass dieser Implementierungsfehler auch in allen anderen Chips dieser Produktreihe enthalten ist, auch wenn dies in den zugehörigen Datenblättern nicht vermerkt ist. Es wurde zugesichert einen entsprechenden Hinweis in diese hinzuzufügen. Aufgrund der großen Anzahl von Eingaben ist ein einzelner IO-Expander für die Anbindung der entwickelten Schaltungen an den Raspberry Pi aber nicht ausreichend. Daher kann ein IO-Expander von Maxim Integrated nicht zum Einsatz kommen. [78]

4.3.6. IO-Expander von Microchip

Von Microchip wird nur eine vergleichsweise kleine Auswahl an IO-Expandern angeboten. Diese unterscheiden sich durch die Anzahl der GPIO Pins und die Kommunikationsschnittstelle. Es werden Chips mit 8 oder mit 16 GPIO Pins angeboten, wobei sich aufgrund der hohen Anzahl an benötigten Eingängen offensichtlich IO-Expander mit 16 GPIO Pins anbieten. Als Kommunikationsschnittstellen stehen I^2C oder SPI zur Verfügung.

Vorteile der I^2C Schnittstelle sind eine bessere Standardisierung wodurch eine gute Kompatibilität sichergestellt ist und der geringe Bedarf an Leitungen. Zur Anbindung eines Chips über die I^2C Schnittstelle sind nur zwei Leitungen, Interruptleitungen nicht mitgerechnet, erforderlich. Für eine SPI Schnittstelle sind typischerweise vier Leitungen erforderlich. Nachteil der I^2C Schnittstelle sind die typischerweise geringen Übertragungsraten. Im Gegensatz zur SPI Schnittstelle, bei der Push-Pull-Ausgänge benutzt werden, welche in der Lage sind eine Signalleitung sowohl auf den Low-Pegel herunterzuziehen, wie auch auf den High-Pegel hochzudrücken, werden bei einer I^2C Schnittstelle Open-Drain Ausgänge benutzt. Diese sind nur in Lage Signalleitungen auf den Low-Pegel herunterzuziehen. Um eine Signalleitung wieder auf den High-Pegel zu drücken, werden Pull-Up-Widerstände eingesetzt. Die Pull-Up-Widerstände können nicht beliebig niederohmig sein, da die Open-Drain-Ausgänge in der Lage sein müssen, den Strom, der durch einen der Widerstände fließt, aufzunehmen, wenn eine Leitung auf den Low-Pegel gezogen werden soll. Damit sind die Widerstandswerte nach unten hin beschränkt, was den maximalen Stromfluss, der eine Leitung hochdrückt, nach oben beschränkt. Damit verlängert sich die Anstiegszeit der Signale auf den beiden I^2C Leitungen. Wodurch wiederum die maximale Übertragungsfrequenz beschränkt wird. Der Einsatz von Pull-Up Widerständen hat den Vorteil, dass keiner der angeschlossenen Chips im Falle einer Fehlfunktion beschädigt werden kann. Selbst wenn mehrere Chips beispielsweise aufgrund einer Adresskollision gleichzeitig beginnen Daten auf den Bus zu senden oder aufgrund einer Fehlfunktion die Leitung dauerhaft im Low-Zustand halten, führt dies zu keiner Beschädigung der Chips, da der Stromfluss stets durch die Pull-Up-Widerstände begrenzt wird.

Es gibt von Microchip zwei aktuelle Chips mit I^2C Interface und 16 GPIO Pins. Dies sind die Chips MCP23017 und MCP23018. Diese bieten Übertragungsraten von 1700 kBits/s bzw. 3400 kBits/s an. Es ist davon auszugehen, dass evtl. nicht die volle Übertragungsgeschwindigkeit nutzbar sein wird, da im Raspberry Pi 1,8 kΩ Pull-Up-Widerstände an den beiden I^2C Leitungen verbaut sind, während in den Datenblättern zu den beiden Chips 1 kΩ Pull-Up-Widerstände empfohlen werden. Angesichts des Verhältnisses zwischen den verbauten und den empfohlenen Wiederständen ist davon auszugehen, dass mindestens die halbe Übertragungsgeschwindigkeit nutzbar sein sollte. Dies würde etwa 1 MBit/s

entsprechen. Schätzt man grob ab, dass zur Übertragung eines Bytes aufgrund von Protokolloverhead etwa 10 Bit benötigt werden, dauert die Übertragung eines Bytes etwa 10 µs. Es muss berücksichtigt werden, dass nicht nur Nutzdaten, sondern auch Chip- und ggf. Registeradressen übertragen werden müssen. Dennoch ist die Übertragungszeit im Verhältnis zur geforderten Reaktionszeit von 5 ms hinreichend kurz. Sodass die geringere Übertragungsgeschwindigkeit der I^2C Schnittstelle nicht gegen deren Einsatz spricht. Aufgrund des robusten Designs und des geringen Bedarfs an Leitungen wurde entschieden im vorliegenden Fall die I^2C Schnittstelle einzusetzen. [79] [51] [80] [81]

Modell	MCP23017	MCP23018
Anzahl GPIO Pins	16	16
Ausgänge	Push-Pull	Open-Drain
Versorgungsspannung (VDD)	1,8 V - 5,5 V	1,8 V - 5,5 V
Kommunikationsschnittstelle	I^2C	I^2C
Übertragungsrate	1700 kBit/s	3400 kBit/s
Anzahl Adresspins /Adressen	3 / 8	1 / 8
zulässige Spannung an GPIO-Pins	-0,6 V - (VDD + 0,6 V)	-0,6 V - 5,5 V

Tabelle 16: Vergleich zwischen MCP23017 und MCP23018
Quellen: [80] und [81]

Die beiden IO-Expander MCP23017 und MCP23018 sind nicht Pin-kompatible, aber insgesamt sehr ähnlich. Insbesondere ist der Satz an verfügbaren Registern identisch, die Unterschiede liegen eigentlich nur in den elektrischen Eigenschaften und der Pinbelegung. Tabelle 16 zeigt einen kurzen Vergleich der beiden Chips um die wesentlichen Unterschiede aufzuzeigen.

Alle GPIO Pins können in Software als Ein- oder Ausgang konfiguriert werden, die Polarität, also der Wert, aller Pins kann invertiert werden, die Erzeugung von Interrupts kann für einzelne Pins ein- und abgeschaltet werden und interne Pull-Up-Widerstände können optional an jedem Pin zugeschaltet werden. Natürlich gibt es auch Register um den aktuellen Wert der Ein- und Ausgänge zu lesen bzw. zu schreiben. Neben der einfachen Möglichkeit Interrupts bei Eingabeänderungen auszulösen, bieten diese Chips auch die Möglichkeit einen Vergleichswert für alle Eingänge anzugeben. Weicht der tatsächliche Wert der Eingänge von diesem ab, wird ein Interrupt erzeugt und wieder deaktiviert, wenn die Werte an den Eingängen wieder dem Vergleichsmuster entsprechen.

Wirklich interessant ist aber das Verhalten der Chips, wenn sie so konfiguriert sind, dass sie Interrupts bei Eingabeänderungen auslösen. Bei Eingabeänderungen speichern sie zunächst den aktuellen Zustand aller Eingänge im Moment der Änderung, also einschließlich dieser Änderung, in einem separaten Register und lösen dann einen Interrupt aus. Der Host kann sich dann entscheiden, ob er den aktuellen Zustand aller Eingänge oder den Zustand der Eingänge zum Zeitpunkt der ersten Änderung oder beide Zustände einlesen will. So kann der Zustand zum Zeitpunkt der ersten Änderung nachvollzogen werden, auch wenn zum Zeitpunkt der Leseoperation der ursprüngliche Zustand wiederhergestellt wurde oder weitere Änderungen aufgetreten sind. Besonders praktisch ist auch die Möglichkeit, den Zeitpunkt zu dem ein Interrupt deaktiviert und damit das gesonderte Register für neue Werte freigegeben wird, zu konfigurieren. Es kann gewählt werden, ob ein Interrupt durch das Lesen des aktuellen Zustandes der Eingänge oder durch das Lesen des gespeicherten Zustandes deaktiviert werden soll. So kann jederzeit der aktuelle Zustand gelesen werden ohne mit der Interruptfunktionalität zu interferieren. [80] [81]

Wie aus Tabelle 16 ersichtlich ist, verfügt der MCP23017 über normale Push-Pull Ausgänge, während der MCP23018 über 5 V tolerante open-drain-Ausgänge verfügt. Da die IO-Expander bevorzugt zur Ergänzung weiterer Eingänge genutzt werden sollen, ist die Fähigkeit der Ausgänge eine Leitung auf den High-Pegel hochzudrücken nicht von Bedeutung. Die Fähigkeit 5 V Eingabepegel zu akzeptieren, auch wenn die Versorgungsspannung geringer ist, könnte hingegen nützlich sein. Da die IO-Expander

von einer auf dem Raspberry Pi laufenden Steuerungssoftware angesteuert werden, muss auch mit einer fehlerhaften Ansteuerung aufgrund von Bedienungsfehlern gerechnet werden. In diesem Fall bietet der Einsatz von Open-Drain Ausgängen den Vorteil, dass ein Nutzer einen Eingang nicht versehentlich zu einem Strom liefernden Ausgang umkonfigurieren kann. Es muss nur der Fehlerfall, dass ein Eingang zu einem Strom aufnehmenden Ausgang umkonfiguriert wird, berücksichtigt werden. Dies könnte die Komplexität der benötigten Schutzbeschaltung reduzieren. Beispielsweise sind keinerlei Schutzmaßnahmen erforderlich, wenn einer der Eingänge durch einen Open-Drain Ausgang eines anderen Chips, wie beispielsweise durch einen der Diagnoseausgänge der Ein- und Ausgabebausteine, angesteuert werden soll. Selbst wenn der Eingang versehentlich als Ausgang konfiguriert werden solle kann nichts passieren. Wird der so entstandene Ausgang auf einen High-Pegel gesetzt ist er immer noch hochohmig und unterscheidet sich so nicht von einem Eingang. Wird ein Low-Pegel gesetzt wird der Stromfluss durch den Pull-Up-Widerstand, welcher bei einer derartigen Verbindung nötig ist, auf ein sicheres Maß beschränkt.

Die höhere Übertragungsrate des MCP23018 könnte auch von Vorteil sein, allerdings ist wie schon zuvor beschrieben, davon auszugehen, dass diese ohnehin nicht voll ausgenutzt werden kann. Der MCP 23018 besitzt nur einen Adresspin, dennoch können 8 verschiedene Adressen definiert werden. Dies wird dadurch ermöglicht, dass es sich um einen analogen Eingang handelt. Die Andresse wird basierend auf der anliegenden Spannung festgelegt. Dies wird im Datenblatt als „Feature“ [81] angepriesen, genau betrachtet ist dies aber eher ein Nachteil. Es ermöglicht zwar den Chip in einem kleineren Gehäuse mit weniger Pins unterzubringen und so Platz auf der Platine zu sparen, allerdings werden, wenn mehr als zwei Chips an einem Bus eingesetzt werden sollen, zusätzliche Komponenten, speziell Widerstände, benötigt um die notwenige Spannung zu erzeugen. Sind hingegen einfach mehrere digitale Konfigurationspins, wie beim MCP23017 vorhanden, können acht verschiedene Adressen gewählt werden indem die entsprechenden Pins entweder mit Masse oder mit der Versorgungsspannung verbunden werden. Da aber in der geplanten Schaltung wohl nicht mehr als zwei solcher Chips zum Einsatz kommen werden, ist dies unbedeutend, da in dem Fall einmal Masse und einmal die Versorgungspannung als Eingaben für den analogen Adresseingang des MCP23018 gewählt werden können um zwei verschiedene Adressen zu erzeugen.

Aufgrund der vorhandenen Open-Drain Ausgänge, welche Bauteile bei der Schutzbeschaltung einsparen dürften und der potenziell höheren Übertragungsgeschwindigkeit, werden für die geplante Schaltung die MCP23018 Chips gewählt. Die Toleranz der Eingänge gegenüber Spannungen jenseits der Versorgungsspannung kann nur sehr bedingt als Vorteil angesehen werden, da bei Tests ein ziemlich merkwürdiges Verhalten, im Speziellen ungewöhnlich hohe Leckströme, an den Eingänge beobachtet wurde, wenn Spannungen jenseits der Versorgungsspannung angelegt werden. Abbildung 17 zeigt die experimentell ermittelte Kennlinie für einen Eingang der MCP23018 Chips bei einer Versorgungsspannung von 3,3 V. Ein solches Verhalten wird vom zugehörigen Datenblatt zwar nicht explizit ausgeschlossen, da über die Höhe des Leckstroms jenseits der Versorgungsspannung keine Angabe gemacht wird, es gibt aber auch keinen Hinweis auf eine derart ungewöhnliche Kennlinie. Eine Nutzung der Eingänge im Bereich jenseits der Versorgungsspannung scheint jedenfalls nicht ratsam.

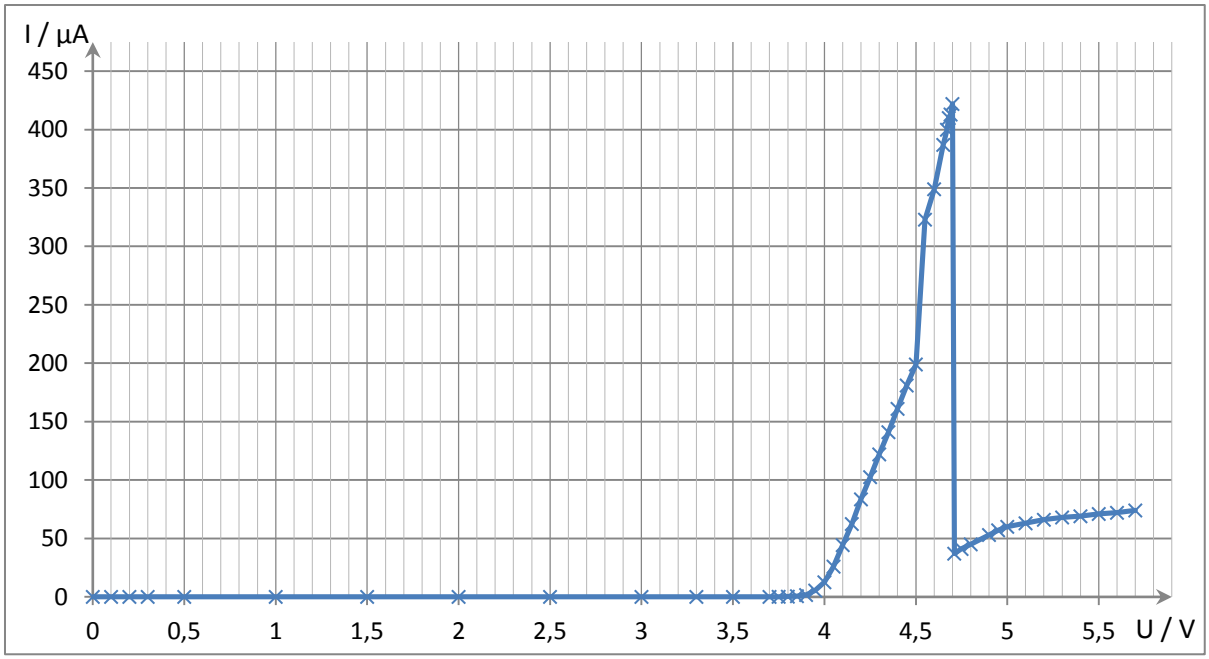


Abbildung 17: Kennlinie eines GPIO Pins am MCP23018

4.4. Gesamtschaltung

Basierend auf den bisherigen Erkenntnissen wurde ein Schaltplan für die geplante SPS entwickelt. Im Folgenden werden alle Teile der entwickelten Schaltung gezeigt und kommentiert, wobei mehrfach vorkommende Konstellationen von Bauteilen, die sich im Schaltplan nur durch die Wahl der Signalnamen unterscheiden nur einmal exemplarisch gezeigt werden, sofern dies zum Verständnis der Schaltung ausreichend erscheint. Die vollständige Schaltung in einer kompakten Darstellung findet sich im Anhang I. Eine Bauteilliste findet sich im Anhang II.

4.4.1. Eingangsschaltung

Die letztlich verwendete Eingangsschaltung unterscheidet sich kaum zu der im Abschnitt 4.1.6 entwickelten Version. Sie wurde planmäßig noch einmal für Port 2 dupliziert. Abbildung 18 zeigt die Eingabeschaltung mit den Bauteilnummern und Signalnamen für Port 2. Der einzige Unterschied zur früher gezeigten Schaltung sind die zusätzlichen Pull-Down-Widerstände R41 und R42. Entsprechende Widerstände wurden auch bei Port 1 hinzugefügt. Diese sind bei korrekter Konfiguration der IO-Expander, an die die Signalleitungen DS0_Px und DS1_Px angeschlossen sind (vgl. Abbildung 21 und Abbildung 22), im Prinzip nicht erforderlich, da an den Eingängen DS0 und DS1 bereits interne Pull-Down-Wiederstände enthalten sind. Allerdings ist es möglich an den GPIO Pins der IO-Expander Pull-Up-Widerstände zuzuschalten. Würde man diese irrtümlicherweise einschalten, würde sich, wenn die Konfigurationsschalter offen sind, in Zusammenwirkung mit den internen Pull-Down-Widerständen der DS0 und DS1 Eingänge ein Spannungsteiler ergeben, der zu einer ungültigen Spannung an den Konfigurationseingängen führen würde. Um sicher zu gehen, dass diese Situation nicht auftreten kann, können zusätzlich die externen Pull-Down-Widerstände R41 und R42 eingebaut werden, deren Wert so gewählt ist, dass sichergestellt ist, dass selbst bei aktivierten Pull-Up-Widerständen im IO-Expander bei offenen Schaltern immer noch eine Spannung anliegt, die sicher als Low-Pegel erkannt wird. Bei offenen Schaltern haben die zusätzlichen Pull-Down-Widerstände keine negativen Nebenwirkungen, bei geschlossenen Schaltern erhöhen sie jedoch geringfügig den Stromverbrauch. Da eine versehentliche Aktivierung der Pull-Up-Widerstände als unwahrscheinlich eingeschätzt wird, ist eine Bestückung der Widerstände zugunsten eines geringeren Stromverbrauchs zunächst nicht ge-

plant. Es sind nur die passenden Anschlusspads vorzusehen um die Widerstände bei Bedarf bestücken zu können. [81] [17]

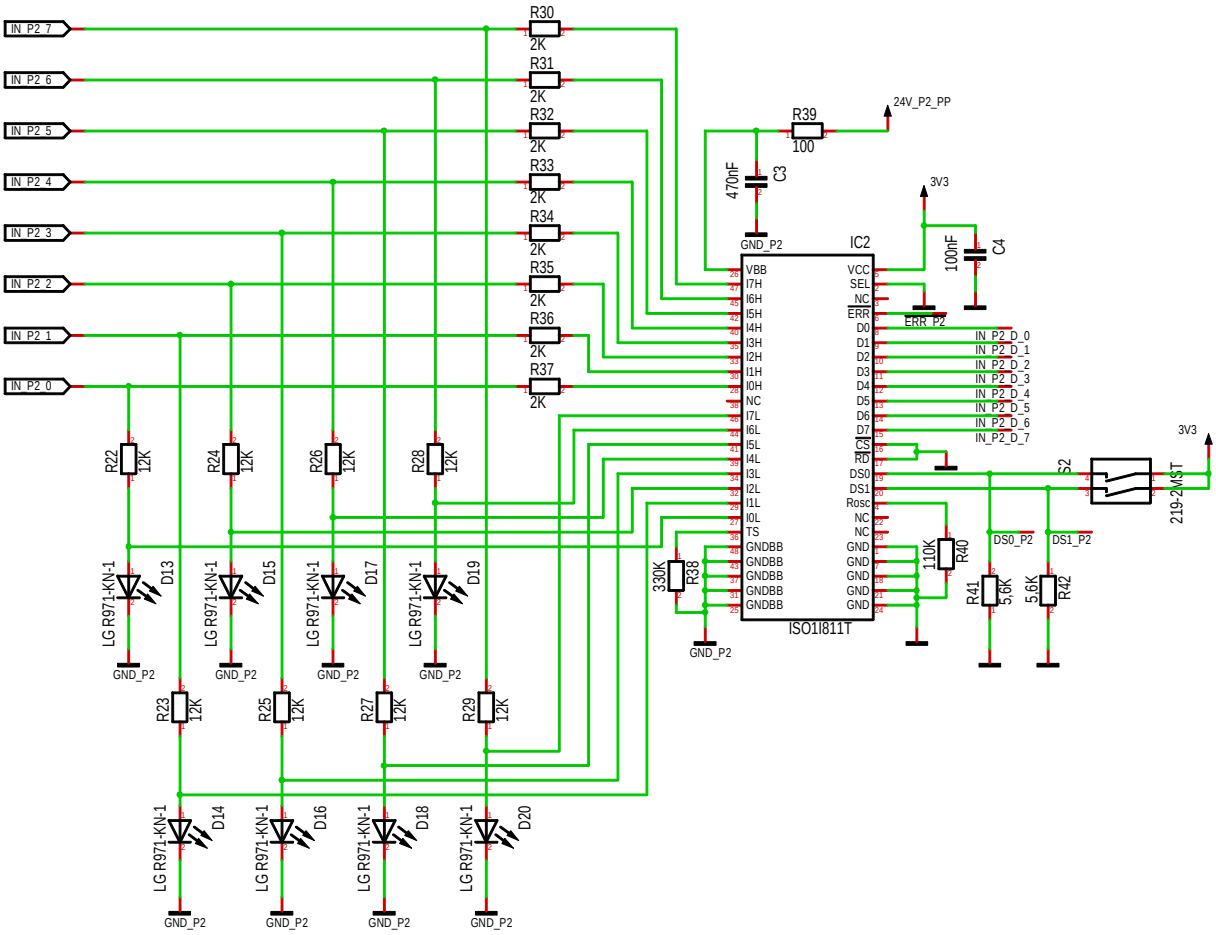


Abbildung 18: Eingabeschaltung für Port 2

An allen Eingängen wurden bidirektionale Überspannungsschutzdioden, wie in Abbildung 19 dargestellt, integriert um die Eingangsschaltung vor elektrostatischen Entladungen zu schützen. Diese wurden so gewählt um den an den Eingängen zulässigen Spannungsbereich möglichst wenig zu beschränken. Die Eingabeschaltung könnte prinzipiell Spannungen von ±45 V standhalten. Allerdings wird das Verhalten der Eingänge im Datenblatt des Eingabebausteins passend zur Festlegung der IEC 61131-2 Norm nur bis 30 V beschrieben. Dies ist auch vollkommen ausreichend, wenn man bedenkt, dass typischerweise eine Betriebsspannung von nur 24 V zum Einsatz kommt. Daher scheint eine Beschränkung des Arbeitsbereichs auf 0 V bis 30 V und eine Beschränkung des Toleranzbereichs, also des Bereichs in dem es zu keiner Beschädigung des Geräts kommt, in dem aber keine korrekte Funktion mehr garantiert wird, auf ±35 V sinnvoll.

Die 12 kΩ Widerstände wurden folglich so gewählt um dauerhaft einer Spannung von 35 V standzuhalten ohne zu überhitzen. Passend dazu wurden bidirektionale Überspannungsschutzdioden mit einer Arbeitsspannung von 32 V und einer Durchbruchspannung von 35,6 V gewählt. Das bedeutet, dass bei Eingabespannungen von bis zu ±32 V kein nennenswerter Strom über die Schutzdioden fließt und die Funktion so von diesen nicht beeinflusst wird. Bei Spannungen außerhalb dieses Bereichs bis zu ±35 V fließt nur ein geringer Strom über die Schutzdioden, so dass es zu keiner Beschädigung des Geräts kommt. Erst bei Spannungen jenseits von ±35,6 V erhöht sich der Stromfluss rapide. [17] [82]

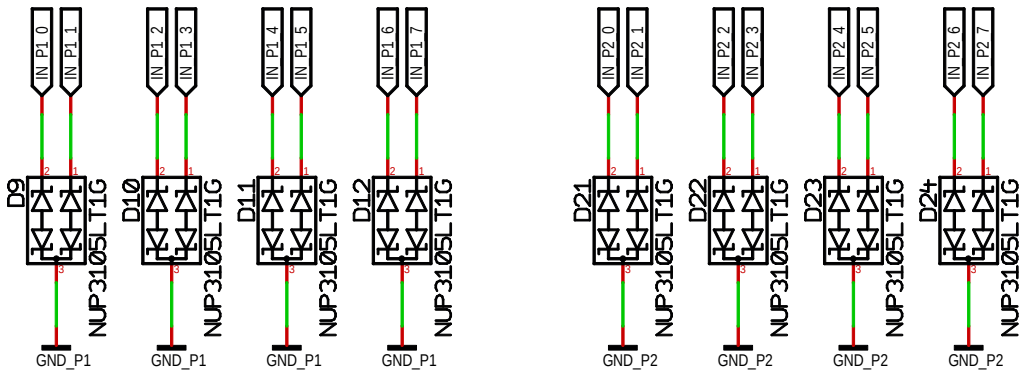


Abbildung 19: ESD-Schutzbeschaltung der Eingänge

4.4.2. Ausgabeschaltung

An der Ausgabeschaltung waren keine Änderungen gegenüber der im Abschnitt 4.2.3 vorgestellten Schaltung erforderlich. Abbildung 20 zeigt die Ausführung der Ausgabeschaltung für Port 2.

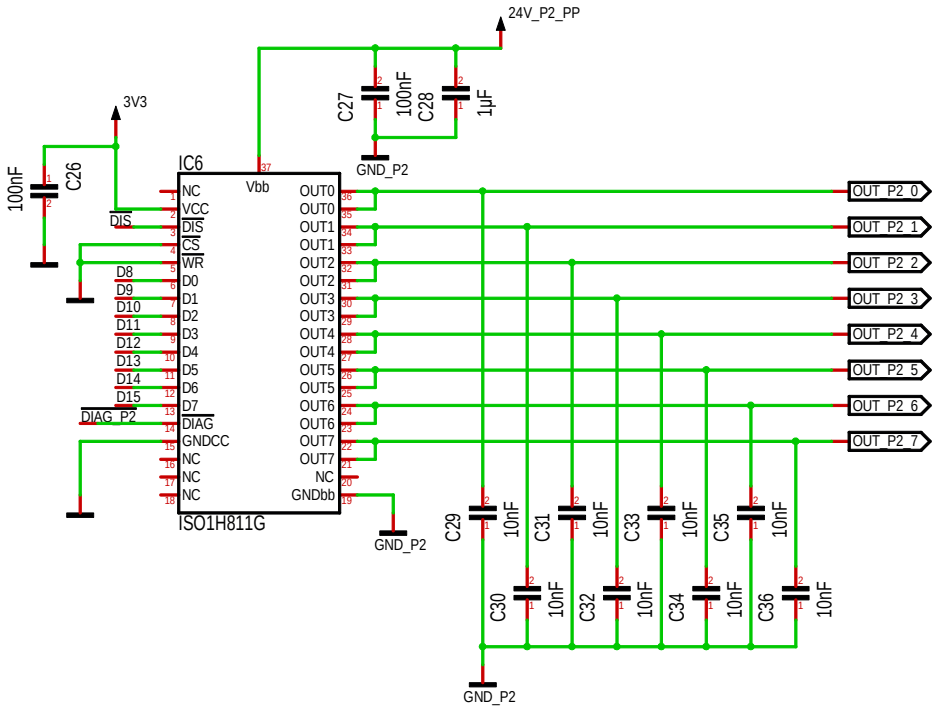


Abbildung 20: Ausgabeschaltung für Port 2

4.4.3. IO-Expander

Insgesamt wurden zwei IO-Expander vom Model MCP23018 in die Schaltung integriert. Die Abbildung 21 und die Abbildung 22 zeigen die Beschaltung dieser Chips. Beide sind direkt am freien I^2C Bus des Raspberry Pi angeschlossen. Um zwei unterschiedliche Adressen zu erhalten wurde der ADDR-Konfigurationseingang des einen Chips auf Masse und der des anderen auf 3,3 V gelegt. Damit erhält der eine Chips die kleineste und der andere die größer verfügbare Adresse. Die Kondensatoren C6, C7, C8, C10, C11 und C12 dienen der Entkopplung um eine stabile Stromversorgung der Chips sicherzustellen. Die !RESET Eingänge beider Chips wurden verbunden und direkt an einen freien GPIO Pin des Raspberry Pi angeschlossen. Damit können die Chips beim Auftreten von Problemen neu gestartet werden.

IC3 gehört zum Port 1 und wurde mit den IN_P1_D_x Leitungen, welche von den Eingabebausteinen kommen beschaltet. Zur Sicherheit wurden die Widerstände R49 und R50 in Reihe geschaltet um den Stromfluss auf ein sicheres Maß zu beschränken, falls die Eingänge des IO-Expanders versehentlich durch einen Programmierfehler zu Ausgängen umkonfiguriert und auf low geschaltet werden sollen. Würde der Eingabebaustein in diesem Fall nämlich versuchen eine der Leitungen auf den High-Pegel zu ziehen, würde es ohne die Widerstände zu unkontrolliertem Stromfluss zwischen den Chips kommen. Dies würde beide Ausgänge überlasten und könnte zu einer Beschädigung beider Chips führen. Ähnliches gilt auch für die angeschlossenen Leitungen DS1_P1 und DS0_P1 vom Konfigurationsschalter der Eingabefilter. Bei geschlossenen Konfigurationsschaltern sind diese Leitungen direkt mit der 3,3 V Stromversorgung verbunden. Würde man die Eingänge des IO-Expanders als Ausgang konfigurieren und auf low schalten, hätte man effektiv einen Kurzschluss. Die Widerstände R43 und R44 begrenzen in diesem Fall den Stromfluss und schützen damit den IO-Expander.

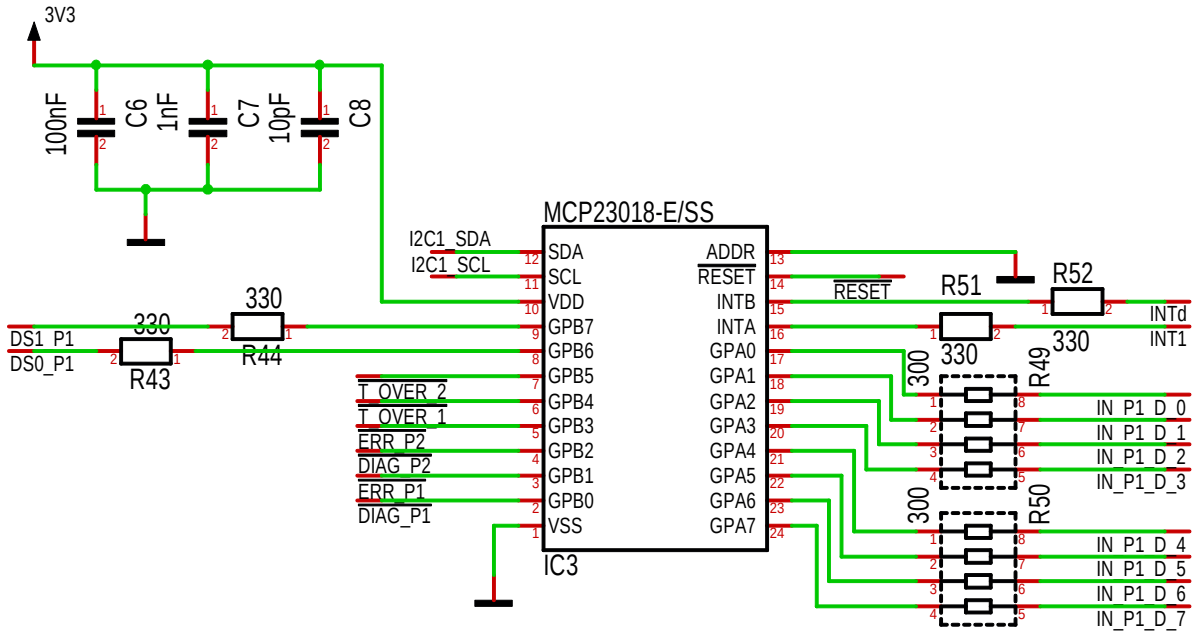


Abbildung 21: Beschaltung des IO-Expanders am Port 1

Zusätzlich zu den bisher beschriebenen Eingaben sind noch einige Diagnoseleitungen am IO-Expander vom Port 1 angeschlossen. Diese kommen von den Diagnoseausgängen der Ein- und der Ausgabebausteine und von zwei Temperatursensoren (vgl. Abbildung 31). In allen Fällen stammen die Signale von Open-Drain Ausgängen mit integrierten oder externen Pull-Up-Widerständen, wodurch keine zusätzliche Schutzbeschaltung notwendig ist. Selbst wenn die Eingänge zu Ausgängen umkonfiguriert werden, sind lediglich zwei Open-Drain Ausgänge miteinander verbunden, was offensichtlich kein Problem darstellt. Da es möglich sein muss Änderungen der Signale von den externen Eingängen und Änderungen an den Diagnoseleitungen der Steuerungssoftware per Interrupt mitzuteilen, sind die entsprechenden Interruptleitungen an GPIO Pins des Raspberry Pi angebunden. Diese sind ebenfalls mit Widerständen gegen Konfigurationsfehler, diesmal auf Seiten des Raspberry Pi, geschützt.

IC 4 ist ähnlich wie IC3 beschalten, allerdings sind an diesem keine Diagnoseleitungen angeschlossen. Durch den Anschluss aller Diagnoseleitungen von beiden Ports am IC3 konnte so einen Interruptleitung vom IC4 zum Raspberry Pi eingespart werden. IC4 benötigt nur eine Interruptleitung um Änderungen an den Eingaben vom Eingangsbaustein mitzuteilen. Änderungen an den Konfigurationsleitungen müssen, wie schon zuvor festgestellt, nicht per Interrupt gemeldet werden. Die sechs verbleibenden freien GPIO Pins des IC4 wurden zum Anschluss von frei programmierbaren LEDs zur Statusanzeige genutzt.

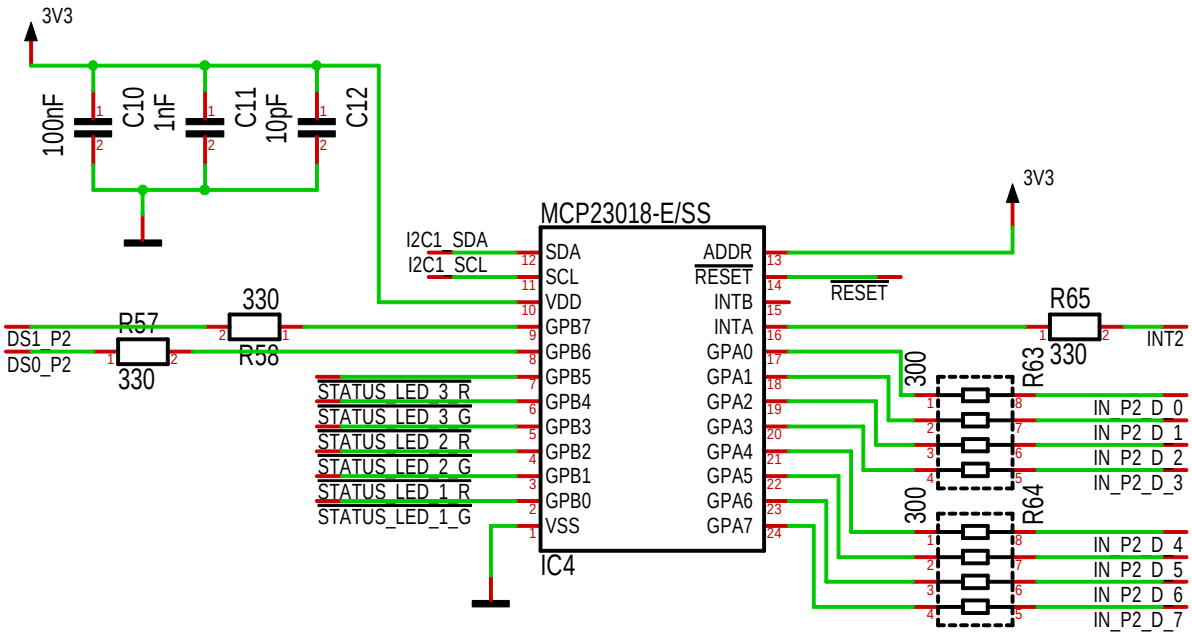


Abbildung 22: Beschaltung des IO-Expanders am Port 2

4.4.4. Anbindung an Raspberry Pi

Abbildung 23 zeigt die Beschaltung der Erweiterungsschnittstelle des Raspberry Pi. Die Beschaltung wurde so gewählt, dass die standardmäßige Konfiguration bezüglich der internen Pull-Up- und Pull-Down-Widerstände des Raspberry Pi zum Hochfahren des Geräts günstig ist. Die Beschaltung konnte praktischerweise sogar so gewählt werden, dass die Konfiguration während der gesamten Betriebszeit nicht verändert werden muss.

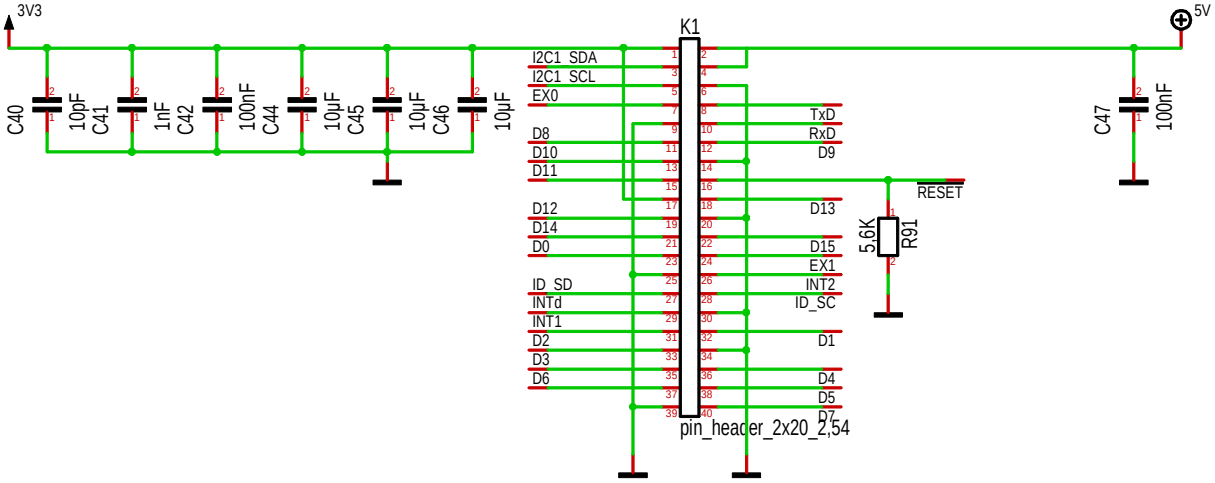


Abbildung 23: Beschaltung der Erweiterungsschnittstelle des Raspberry Pi

Die Signale Dx, welche direkt die Ausgänge steuern und daher initial unbedingt einen Low-Pegel aufweisen müssen, wurden an Pins mit Pull-Down-Widerständen angeschlossen, so dass beim Hochfahren des Geräts von Anfang an ein Low-Pegel anliegt. Später müssen diese Pins als Ausgänge konfiguriert werden um die Ausgaben steuern zu können. Werden diese zu irgendeinem Zeitpunkt aus welchem Grund auch immer wieder als Eingang konfiguriert, stellen die internen Pull-Down-Widerstände sicher, dass die Ausgänge deaktiviert, also auf den Low-Pegel geschalten werden, was als sicherer Zustand angesehen wird. Die !RESET Leitung wurde aus gleichem Grund ebenfalls an einem

Pin mit internen Pull-Down-Widerstand angeschlossen. Da eine versehentliche Aktivierung des internen Pull-Up-Widerstandes zu einem unerwarteten Verhalten führen könnte, dessen Grund potenziell sehr schwer zu finden sein dürfte, wurde an der !RESET Leitung zur Sicherheit ein zusätzlicher externer Pull-Down-Widerstand angebracht, der auch bei versehentlicher Aktivierung des internen Pull-Up-Widerstandes einen low-Pegel sicherstellt, solange der Pin als Eingang konfiguriert ist.

Die INTx Leitungen wurden an Pins mit Pull-Up-Widerständen angeschlossen, wobei dies im Normalfall nicht von Bedeutung ist, da die IO-Expander standardmäßig mit Push-Pull-Interrupt-Ausgängen arbeiten. Die Interrupt-Ausgänge der IO-Expander können zu einem Open-Drain-Betrieb umgeschaltet werden, in diesem Fall wären die Pull-Up-Widerstände für die korrekte Funktion sogar tatsächlich erforderlich, aber es ist nicht vorgesehen diese Konfigurationsmöglichkeit zu nutzen. [81]

Die Leitungen TxD, RxD, EX0, EX1 gehen direkt zur Erweiterungsschnittstelle (vgl. Abbildung 26). EX0 und EX1 sind standardmäßig mit internen Pull-Up-Wiederständen versehen, TxD und RxD sind standardmäßig als UART Schnittstelle konfiguriert. I2C1_x bilden die I^2C Schnittstelle, welche zu den IO-Expandern geht und auch an der Erweiterungsschnittstelle verfügbar ist. ID_SD und ID_SC bilden die I^2C Schnittstelle zum Anschluss eines EEPROM Chips (vgl. Abbildung 27).

4.4.5. Leuchtanzeigen an Ein- und Ausgängen

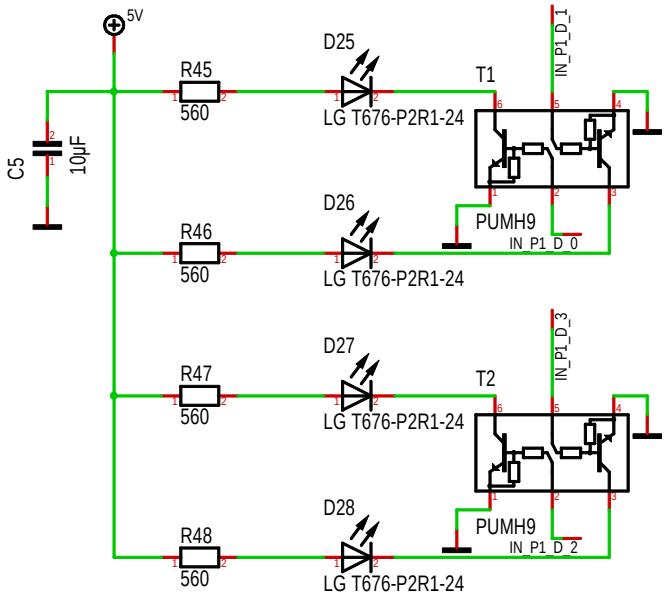


Abbildung 24: Leuchtanzeigen an den Eingängen 0 bis 3 von Port 1

Die LEDs zur Statusanzeige der Ein- und Ausgänge konnten leider nicht direkt an die entsprechenden Signalleitungen angeschlossen werden, da alle Signalleitungen mit 3,3 V Pegeln arbeiten. 3,3 V wären zwar zum Betrieb von LEDs ausreichend, allerdings ist die 3,3 V Stromversorgung von Raspberry Pi nicht für eine derart hohe Leistung, wie sie zum Betrieb von derart vielen LEDs erforderlich wäre, ausgelegt. Daher werden alle LEDs mit 5 V betrieben und über Transistoren von den Signalleitungen geschalten. Das hat nebenbei den Vorteil, dass die nutzbare Stromstärke für eine LED nicht durch den jeweiligen Ausgang beschränkt wird. Es wurden 560 Ω Vorschaltwiderstände für alle LEDs gewählt, wodurch ein Stromfluss von etwa 6 mA Zustand kommt. Dies ergibt bei den eingesetzten LEDs eine ausreichende Helligkeit. Sollte später eine größere Helligkeit der Anzeigen gewünscht sein, könnten aber dank der Transistoren problemlos auch 10 mA oder sogar 20 mA genutzt werden. Die Beschaltung der LEDs wird in Abbildung 24 bzw. Abbildung 25 für Ein- bzw. für Ausgänge exemplarische gezeigt.

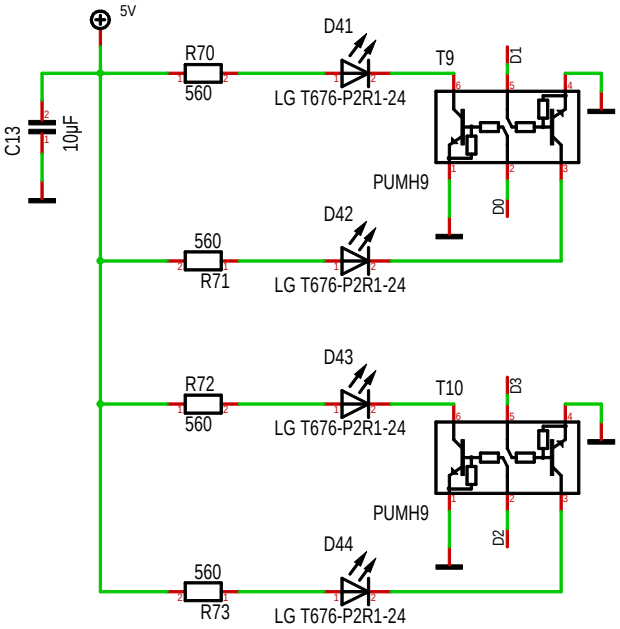


Abbildung 25: Leuchtanzeigen an den Ausgängen 0 bis 3 von Port 1

4.4.6. Erweiterungsschnittstelle

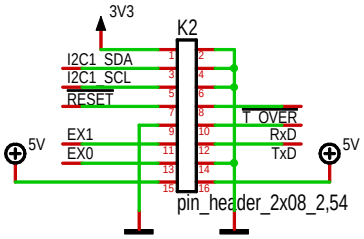


Abbildung 26: Beschaltung der Erweiterungsschnittstelle

Für eine spätere Erweiterung oder den Anschluss von Eigenentwicklungen wurde eine kleine Erweiterungsschnittstelle vorgesehen. Abbildung 26 zeigt die Beschaltung dieser Schnittstelle. Es sind die beiden Leitungen der I^2C Schnittstelle des Raspberry Pi, welche auch von den IO-Expandern genutzt wird enthalten. An diesen Bus könnten bei Bedarf weitere Chips angeschlossen werden. Das !RESET Signal, das zu den IO-Expandern geht, ist auch auf die Erweiterungsschnittstelle ausgeführt und könnte somit zum Neustart aller am I^2C Bus angeschlossenen Geräte genutzt werden. Da es sich um einen Bus handelt dürfte es kaum möglich sein, im Falle einer Kommunikationsstörung den verursachenden Chip festzustellen, daher ist es durchaus sinnvoll mit einem gemeinsamen Signal alle Chips, welche sich an diesem Bus beteilige, neu zu starten. Wie schon erwähnt werden unter den Bezeichnungen EX0, EX1, TxD und RxD einfach GPIO-Pins vom Raspberry Pi zur Erweiterungsschnittstelle durchgeleitet. Damit können diese Pins auch zur Auslösung von Interrupts am Raspberry Pi genutzt werden. Zur Versorgung einer weiteren Schaltung werden 3,3 V und 5 V bereitgestellt. Selbstverständlich stehen auch einige Masseanschlüsse zur Verfügung. Diese wurden so angeordnet, dass die Masseleitungen beim Anschluss eines Flachbandkabels möglichst zwischen den Leitungen liegen, an denen die höchsten Frequenzen zu erwarten sind, wodurch diese gegeneinander besser abgeschirmt werden.

4.4.7. ID-EEPROM

Abbildung 27 zeigt die Anbindung eines ID-EEPROM Chips. dieser Chip kann genutzt werden um Informationen über die Schaltung abzulegen. Die Verwendung dieses Chips und der abgelegten Informationen wird genauer im Abschnitt 4.7 beschrieben. Der Chip, die externen Komponenten, sowie

die Beschaltung selbst wurden gemäß der Empfehlung [83] der Entwickler des Raspberry Pi gewählt um eine gute Kompatibilität sicherzustellen. R88 sorgt dafür, dass der Inhalt des Chips schreibgeschützt ist und nicht versehentlich vom Benutzer geändert werden kann. Über den Testpunkt X8 besteht die Möglichkeit den Schreibschutz vorübergehend aufzuheben indem die Leitung auf Masse gelegt wird. R89 und R90 sind die Pull-Up-Widerstände für den I^2C Bus. Diese müssen extern nachgerüstet werden, da im Raspberry Pi an diesem Bus keine vorhanden sind. Die Adresspins wurden gemäß Empfehlung auf Masse gelegt um die kleinste verfügbare Adresse auszuwählen. [82]

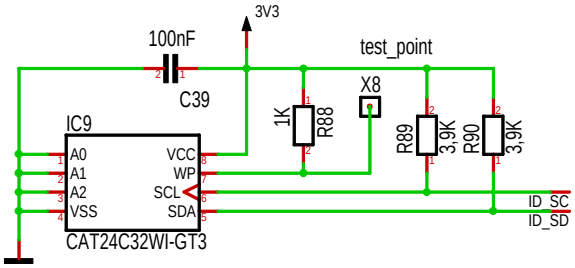


Abbildung 27: Anbindung eines ID-EEPROM Chips

4.4.8. Statusanzeigen

Um dem Benutzer anzuzeigen, dass eine 5 V Versorgungsspannung vorhanden ist, wurde eine weitere grüne LED direkt an die 5 V Versorgung, wie in Abbildung 28 gezeigt, angeschlossen.

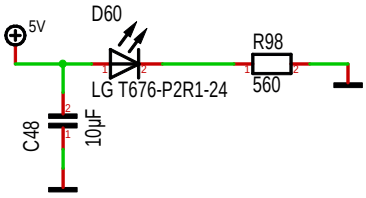


Abbildung 28: Leuchtanzeige für die 5 V Stromversorgung

Als frei programmierbare Leuchtanzeigen wurden drei zweifarbige LEDs gewählt. Jede Farbe ist, wie in Abbildung 29 dargestellt, mit einer eigenen Signalleitung verbunden, so dass jede Farbe einzeln über den IO-Expander geschaltet werden kann. Jede LED kann entweder ganz aus sein, rot leuchten, grün leuchten oder gelb leuchten. Dies kann später durch die Steuerungssoftware festgelegt werden.

Auch diese LEDs werden mit 5 V betrieben und über Transistoren geschaltet. Betrachtet man Abbildung 30 stellt man fest, dass es möglich sein sollte statt 3,3 V 5 V anzuschließen und die LEDs direkt an die Leitungen STATUS_LED_1_G und STATUS_LED_1_R anzuschließen. Dies ist allerdings aufgrund der im Abschnitt 4.3.6 beschriebenen Kennlinie der Eingänge nicht möglich. Da an einem Transistor zwischen Emitter und Basis typischerweise etwa 0,6 V abfallen, würden am GPIO Pin des IO-Expanders etwa 4,4 V anliegen. Berücksichtigt man den in der Abbildung dargestellten parallelen Widerstand zwischen Basis und Emitter ergibt sich sogar eine noch etwas höhere Spannung. In diesem Bereich zeigt die Kennlinie (vgl. Abbildung 17) einen hohen Leckstrom, welcher von einem Transistor verstärkt werden würde. Daher ist es notwendig nur 3,3 V anzuschließen und die LED über einen zusätzlichen Transistor, wie in Abbildung 29 gezeigt, anzuschließen.

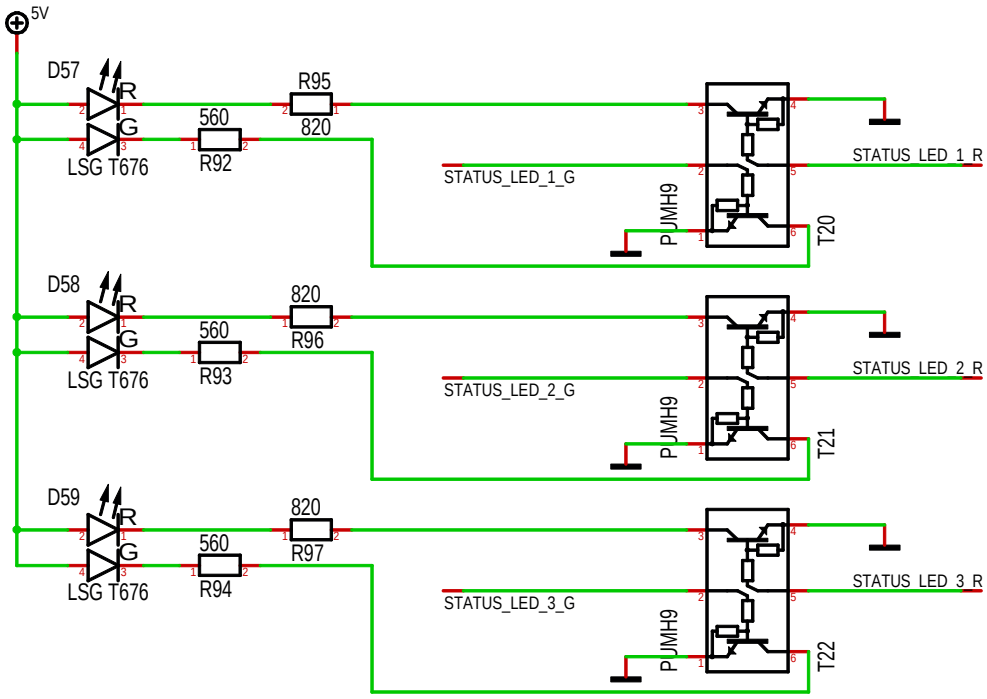


Abbildung 29: Anbindung frei programmierbarer Statusanzeigen

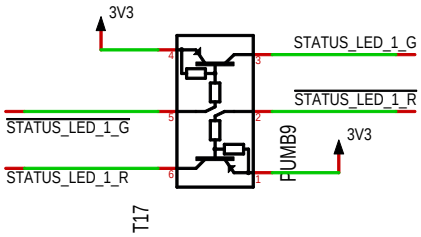


Abbildung 30: Signalinvertierung bei frei programmierbaren Statusanzeigen
Quelle: interner Aufbau des vorgespannten Transistors gemäß Datenblatt [84]

4.4.9. Überhitzungsschutz

Die eingesetzten Ausgabebausteine beschränken im Falle einer Überlastsituation den Ausgabestrom. Dauert eine Überlastsituation länger oder sogar dauerhaft an, kommt es aufgrund der Verlustleistung an den Ausgabeschaltern zu einer starken Hitzeentwicklung. Um sich selber zu schützen schalten die Ausgabebausteine beim Erreichen von 135 °C die Ausgabe ab. Erst wenn sie um etwa 10 °C abgekühlt sind, schalten sie die Ausgabe wieder ein. Dies beschränkt die Hitzentwicklung erheblich, so dass auch länger anhaltende Überlastsituationen kein Problem darstellen. Würde eine Überlastsituation allerdings dauerhaft anhalten, käme es zu einer langsamen Aufheizung des gesamten Geräts. Um auch in diesem Extremfall die Gehäusetemperatur auf ein für den Benutzer ungefährliches Nivea zu beschränken wurden zwei Temperatursensoren eingebaut. Diese Messen die Temperatur an verschiedenen Stellen des Geräts und melden, wenn diese eine bestimmte Grenze übersteigt. Der Grenzwert ist in den Temperatursensoren vom Hersteller fest eingestellt und kann nicht verändert werden. Die gewählten Chips werden mit verschiedenen Grenzwerten angeboten. Ausgesucht wurden Chips mit einer Grenze von 65 °C. Dies liegt unter der von der IEC 61131-2 Norm festgelegten Grenze für metallische Oberflächen, die vom Benutzer zeitweise berührt werden. Abbildung 31 zeigt die Beschaltung dieser Sensoren. [16] [62] [85]

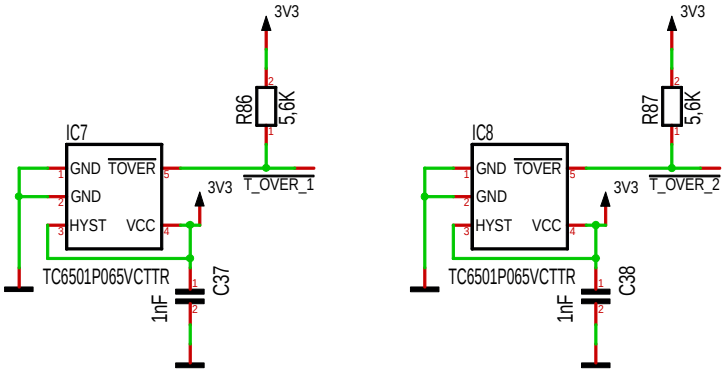


Abbildung 31: Beschaltung von Überhitzungsschutzschaltern

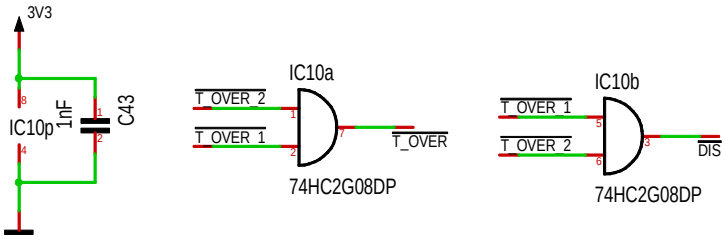


Abbildung 32: Schaltung zur Auswertug von Überhitzungssignalen

Die Ausgabebausteine liefern im Falle einer Überlastung ein Diagnosesignal um diese Fehlersituation anzuzeigen. Dieses wird über die IO-Expander an die Steuerungssoftware weitergeleitet, diese kann dann entscheiden, was zu tun ist. In jedem Fall sollte diese die Ausgänge abschalten noch lange bevor sich das Gerät so aufheizt, dass die Temperatursensoren ansprechen. Sollte sich das Gerät aber tatsächlich so stark aufgeheizt haben, dass diese ansprechen, muss eine Abschaltung des Geräts in jedem Fall gewährleistet sein. Daher kann die Auswertung des Überhitzungssignals nicht in Software erfolgen. Es muss in Hardware sichergestellt sein, dass bei Ansprechen der Temperatursensoren die Ausgaben umgehend abgeschaltet werden und dass dies durch die Steuerungssoftware nicht verhindert werden kann.

Daher werden die Überhitzungssignale in Hardware an einem UND-Baustein, wie in Abbildung 32 dargestellt, zusammengeführt. Sobald eines der Signale eine Überhitzung anzeigt, liefert der UND-Baustein ein Signal, das direkt zu den Ausgabebausteinen geführt wird und zu einer sofortigen Abschaltung der Ausgänge führt.

Zusätzlich werden die einzelnen Überhitzungssignale, wie zuvor beschrieben, auch über einen IO-Expander an die Steuerungssoftware weitergeleitet. Dies dient aber lediglich der Information. Eine Reaktion seitens der Software ist zur Abschaltung der Ausgänge nicht erforderlich. Die Software kann die Abschaltung auch nicht verhindern.

Die Temperatursensoren besitzen Open-Drain Ausgänge. Diese sind direkt mit den Eingängen des IO-Expanders verbunden. Da die Eingänge des IO-Expanders auch als Open-Drain Ausgänge konfiguriert werden können, besteht die erwähnenswerte Möglichkeit ein Überhitzungssignal in Software zu simulieren und so eine Notabschaltung herbeizuführen.

4.4.10. Anschlussstecker für Port 1 und Port 2

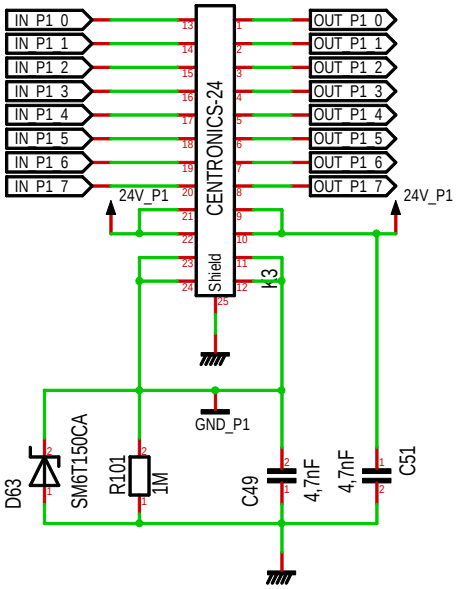


Abbildung 33: Beschaltung des Anschlussstecker von Port 1

Als Anschlussstecker für die beiden Ports wurden 24-polige Centronics-Buchsen mit einer Pinbelegung gemäß Tabelle 2 vorgesehen. Da pro Port auf diesen nur eine gemeinsame Masse zur Verfügung steht, müssen die Ein- und Ausgänge eines Ports zwangläufig mit einer gemeinsamen Masse arbeiten. Die beiden einzelnen Ports sind aber gegeneinander isoliert und von der restlichen Schaltung jeweils galvanisch getrennt. Abbildung 33 zeigt exemplarisch die Beschaltung des Anschlussstecker von Port 1. R101 dient dazu eine hochohmige Verbindung zwischen den galvanisch isolierten Bereichen aufrechtzuerhalten. Würde zwischen diesen Bereichen keinerlei Verbindung bestehen, könnten aufgrund von statischen Aufladungen oder Leckströmen in der Versorgungsspannung unbeabsichtigt sehr hohe Potenzialunterschiede auftreten. Eine hochohmige Verbindung verhindert dies, erlaubt aber gleichzeitig, dass beide Teile auf verschiedenen Potenzialen arbeiten, wenn dies explizit gewünscht ist. D63 ist eine Überspannungsschutzdiode und beschränkt den maximalen Potenzialunterschied, welcher z.B. während statischen Entladungen auftreten kann, soweit, dass sichergestellt ist, dass dieser nicht unkontrolliert an einer anderen Stelle der Schaltung ausgeglichen wird.

4.4.11. Verpolungsschutz

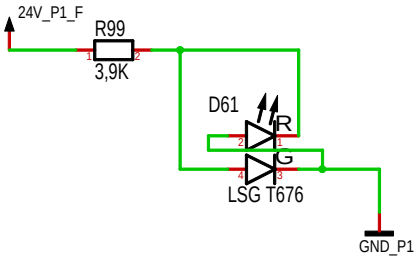


Abbildung 34: Leuchtanzeige zum Anzeigen der Polarität der Versorgungsspannung

Eine Anforderung bestand darin einen Verpolungsschutz zu implementieren. Für die Eingänge wurde dieser bereits realisiert, für die Stromversorgung wird er mit den Schaltungen, welche in Abbildung 34 und Abbildung 35 exemplarisch für Port 1 gezeigt werden, verwirklicht. Bevor Strom durch die Schaltung fließen kann, wird durch die Diode D67 sichergestellt, dass dieser auch in die richtige Richtung fließt. Wird die Stromversorgung verkehrt herum angeschlossen, blockiert D67 jeglichen Stromfluss. Vor dieser Diode sind nur wenige Bauteile, die keinen Verpolungsschutz benötigen. Dazu zählt insbe-

sondere die zweifarbige Leuchtdiode D61. Bei dieser sind die Einzeldioden entgegengesetzt parallel geschalten und zwar in der Art, dass bei korrekter Polung der Versorgungsspannung die grüne LED leuchtet und bei falscher Polung die rote LED leuchtet. So kann der Benutzer sofort sehen, ob eine Versorgungsspannung anliegt und ob diese richtig angeschlossen ist.

Wenn alle acht Ausgänge eines Ports mit dem Bemessungsstrom von 0,5 A belastet werden, müssen durch die Diode D67 etwa 4 A Strom fließen. Bei einer normalen Diode würde dies aufgrund des relativ hohen Spannungsabfalls zu einer erheblichen Verlustleistung und damit zu einer hohen Hitzeentwicklung führen. Daher wurde hier eine Schottky Diode gewählt, welche bei 4 A Stromfluss nur etwa 0,4 V Vorwärtsspannung aufweist. Dies reduziert die Hitzentwicklung im Vergleich zu einer normalen Diode. Dennoch muss beim Entwurf des Platinenlayouts eine angemessene Kühlfläche vorgesehen werden. Passend zum zuvor festgelegten maximalen Spannungsbereich von ±35 V, welcher an den Eingängen toleriert werden kann, bieten die gewählten Schottky Dioden einen Verpolungsschutz für Spannungen bis -45 V. [86]

Die restliche Beschaltung angefangen bei der Sicherung, über die Überspannungsschutzdioden bis hin zu den Kondensatoren, einschließlich der Kondensatoren C49 und C51 aus Abbildung 33, wurde in Anlehnung an das Evaluation Board [74] der Ausgabebausteine gewählt.

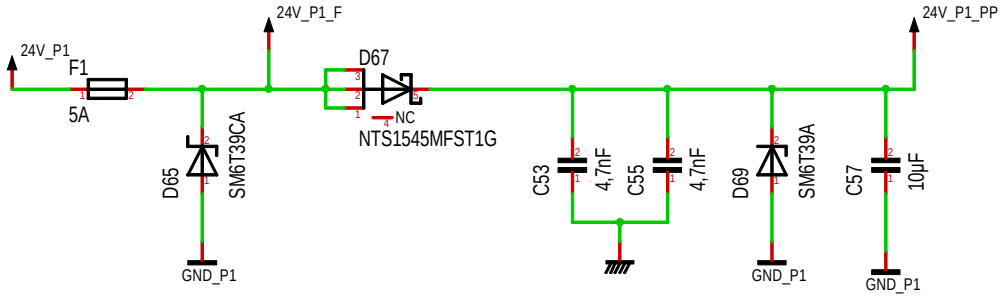


Abbildung 35: Schutzschaltung am Stromversorgungsanschluss von Port 1

4.5. Platinenlayout

Die Entwicklung eines Schaltplans ist nur der erste Schritt in Richtung eines Prototypen. Als nächstes muss ein Platinenlayout entwickelt werden. Ein kurzer Überblick darüber, was in einem Platinenlayout enthalten sein muss und welche Auswahlmöglichkeiten und Beschränkungen beim Entwurf einer Platine bestehen, wird im folgenden Abschnitt geliefert. Selbst aus diesem sehr kurzen und nur sehr oberflächlichen Überblick sollte ersichtlich werden, dass die Entwicklung eines Platinenlayouts insgesamt ein sehr komplexes Unterfangen ist. Daher sollen in den nachfolgenden Abschnitten dann nur die wichtigsten Überlegungen und Schritte, die bei dem Entwurf eines passenden Platinenlayouts für die SPS nötig waren, aufgezeigt werden.

4.5.1. Allgemeines zum Layout einer Platine

Ein Platinenlayout gibt die Form, Größe und Position der Lötpads, auf die später die Bauteile gelötet werden, der Kupferbahnen und Kupferflächen, sowie der Bohrlöcher und Ausfräsungen vor. Es werden auch die Größe und die äußere Form der Platine selbst definiert.

Darüber hinaus sollte ein Platinenlayout auch Informationen über Lötstopplack, Bestückungsdruck, sowie Lötpaste und Kleber enthalten. Die Verwendung von Lötstopplack ist nicht zwingend erforderlich, allerdings ist die Bedeckung aller Leiterbahnen und Kupferflächen, die später nicht zugänglich sein müssen, anzuraten, da dieser Lack die Leiterbahnen vor Schmutz und damit Kurzschlüssen schützt, die Isolation zwischen den Bahnen verbessert und das Löten deutlich vereinfacht, da an die-

sem kein Lötzinn haftet. Im Prinzip ist die gesamte Platine mit Ausnahme von Lötpads, eventueller Testpunkte und von Befestigungslöchern mit Lötstopplack zu bedecken.

Ein Bestückungsdruck gibt üblicherweise die Umrisse jedes Bauteils mit der zugehörigen Bauteilnummer, wie sie im Schaltplan verzeichnet ist, und den Namen der Platine wieder. Ein Bestückungsdruck ist natürlich nicht zwingend erforderlich, ist aber insbesondere bei einer manuellen Bestückung eine große Hilfe.

Bei einer manuellen Bestückung eines Prototypen sind Angaben über Lötpaste und Kleber nicht unbedingt erforderlich, aber für eine spätere maschinelle Serienfertigung muss das Platinenlayout auch Angaben darüber enthalten, an welchen Stellen Lötpaste aufzutragen ist, bevor die Bauteile platziert werden. Sinnvollerweise sollten diese Stellen genau den Lötpads entsprechen, auf die die Bauteile gelötet werden sollen. Durch eine Reduktion des Bereichs, in dem Lötpaste aufzutragen ist, im Verhältnis zum Lötpad kann die Menge an Lötpaste, die aufgetragen werden soll, für jedes einzelne Lötpad gesteuert werden. Bei besonders großen Lötpads kann es sogar erforderlich sein die Lötpaste nicht ganzflächig auf das Lötpad aufzutragen, sondern auf mehrere kleine einzelne Bereiche verteilt um eine ungleichmäßige Umverteilung beim späteren Erhitzen zu verhindern. [87]

Die Verwendung von Kleber kann erforderlich werden, wenn eine Platine beidseitig bestückt werden soll und auf beiden Seiten große und schwere Bauteile platziert werden müssen. Üblicherweise werden beim beidseitigen Bestücken zunächst Bauteile auf eine Seite gelötet. Danach werden Bauteile auf der anderen Seite platziert und die gesamte Platine wird erneut erhitzt um die neu hinzugefügte Lötpaste unter den neu hinzugefügten Bauteilen zu schmelzen und die Bauteile so festzulöten. Da dabei natürlich auch das Lötzinn, mit dem die Bauteile auf der Unterseite festgelötet sind, schmilzt, müssen große, schwere Bauteile auf der Unterseite festgeklebt werden, damit diese nicht abfallen. Bei kleinen, leichten Bauteilen oder Bauteilen mit vielen oder großflächigen Anschlüssen genügt oft die Oberflächenspannung des Lötzinns um die Bauteile festzuhalten. Beim manuellen Bestücken kann auf Kleber verzichtet werden, da nie die gesamte Platine auf einmal erhitzt wird.

Eine Platine kann verschieden viele übereinanderliegende durch Isolationsmaterial getrennte Kupferlagen enthalten. Jede Kupferlage kann beliebige Leiterbahnen und Kupferflächen enthalten. Leiterbahnen auf verschiedenen Ebenen können mittels Bohrlöchern, welche innen mit Kupfer verkleidet werden, miteinander verbunden werden. Diese Bohrlöcher können entweder gleichzeitig zum Einlöten von Komponenten zur Durchsteckmontage genutzt werden oder ausschließlich zum Verbinden von Leiterbahnen. Wird ein Bohrloch ausschließlich zum Verbinden von Leiterbahnen auf verschiedenen Ebenen genutzt, spricht man von einer Durchkontaktierung. Im einfachsten Fall gehen Durchkontaktierungen von der obersten bis zu untersten Lage und verbinden alle Leiterbahnen und Kupferflächen, die sie durchqueren miteinander. Die Firma Beta Layout, bei der die tatsächliche Herstellung der Platine in Auftrag gegeben werden soll, stellt Prototypen von Platinen mit zu 6 Lagen her. Bei größeren Stückzahlen bietet sie die Fertigung von Platinen mit bis zu 24 Lagen an. [88]

Die Leiterbahnbreiten und die Abstände zwischen einzelnen Leiterbahnen werden sowohl durch die elektrischen Eigenschaften als auch durch die Herstellungsverfahren limitiert. Die Breite einer Leiterbahn muss ausreichend groß gewählt werden um einen genügend geringen Widerstand zu bieten. Dies ist besonders bei Leitungen, welche eine hohe Stromstärke transportieren müssen, wichtig, damit diese nicht überhitzen. Breite Leiterbahnen benötigen aber auch mehr Platz und können sich eher gegenseitig aufgrund von kapazitiven und induktiven Kopplungen beeinflussen. Der Abstand zwischen Leiterbahnen muss groß genug gewählt werden um eine Isolation zwischen den Leitungen sicherzustellen, je höher die Spannungsdifferenz zweier Leitungen umso größer muss logischerweise der Abstand zwischen ihnen sein um ein Überschlagen der Spannung zu verhindern. Große Abstände zwischen Leiterbahnen bedeuten aber auch einen größeren Platzbedarf. Wie schon erwähnt verbessert sich die Isolation, wenn Leiterbahnen durch Isolationslack bzw. Lötstopplack bedeckt werden im Vergleich zu frei

liegenden Leiterbahnen zwischen denen nur Luft ist. Damit können geringer Isolationsabstände gewählt und Platz gespart werden, sofern die gegenseitige Störung der Signale aufgrund von kapazitiven und induktiven Kopplungen nicht der limitierende Faktor ist.

Vom Herstellungsverfahren werden Mindestbreiten und Mindestabstände für Leiterbahnen vorgegeben. Diese hängen vom Hersteller und dem eingesetzten Verfahren ab. Der gewählte Hersteller, Beta Layout, produziert standardmäßig, also ohne Aufpreis, Leiterbahnen mit einer Mindestbreite und einem Mindestabstand von jeweils 0,150 mm. Dabei ist zu beachten, dass es sich hierbei um eine Beschränkung handelt, die sich aus dem Produktionsverfahren ergibt. Das bedeutet, dass z.B. der Mindestabstand nicht nur zwischen zwei unterschiedlichen Leiterbahnen, sondern auch zwischen verschiedenen Teiler einer Leiterbahn eingehalten werden muss, auch wenn zwischen diesen kein Isolationsabstand notwendig wäre. Dies ist besonders an der Stelle, wo eine Leiterbahn an einem Lötpad angeschlossen ist, interessant. Macht die Leiterbahn nach Verlassen eines größeren Lötpads einen Knick, muss zwischen der Leiterban und dem Pad überall der produktionsbedingte Mindestabstand eingehalten werden um Probleme bei der Produktion zu vermeiden.

Die Mindestbreite von Leiterbahnen und der Mindestabstand zwischen zwei Leiterbahnen sind die wichtigsten Vorgaben seitens der Produktion, aber es sind produktionsbedingt auch noch viele weitere Kriterien einzuhalten. Beispiele sind eine Mindestgröße für Bohrlöcher, Mindestbreiten für Kupferrestringe, die um Komponentenbohrungen und Bohrungen für Durchkontaktierungen übrig bleiben müssen und Mindestabstände von Leiterbahnen und Kupferflächen zu Bohrungen und Ausfräsungen, welche nicht zur Verbindung von Kupferlagen genutzt werden sollen, also beispielsweise Montagelöcher, an denen keine elektrische Verbindung zum Gehäuse gewünscht ist.

4.5.2. Grundlegende Platineneigenschaften

Wie schon am Schaltplan zu erkennen ist, gibt es drei galvanisch isolierte Bereiche. Es gibt jeweils einen Bereich für jeden Port mit den jeweiligen Ein- und Ausgaben und einen für die Logikschaltung zur Anbindung an den Raspberry Pi, die Leuchtanzeigen und die Erweiterungsschnittstelle. Letzter Bereich wird im Folgenden als Logikbereich bezeichnet.

Es wurde entschieden eine Platine mit vier Kupferlagen einzusetzen, da dadurch im Logikbereich jeweils eine Lage für Masse und eine für die Versorgungsspannungen 3,3 V und 5 V reserviert werden kann. Dies vereinfacht das Verlegen der Leiterbahnen deutlich und erlaubt den Entwurf einer kleineren Platine, da die gemeinsame Masse und die beiden Versorgungsspannungen, was die drei mit Abstand am häufigsten benötigten Signale sind, überall leicht mit einer Durchkontaktierung erreicht werden können und für diese damit keine extra Leiterbahnen verlegt werden müssen. Die Masse wird in der entwickelten Schaltung an über 100 Anschlusspins benötigt, 5 V und 3,3 V werden jeweils an etwa 50 Anschlusspins benötigt. Außerdem hat der Einsatz von soliden großflächigen Masse- und Versorgungsflächen den Vorteil, dass stets eine Verbindung mit niedriger Impedanz zur Masse bzw. zur Versorgungsspannung hergestellt werden kann, was lokal auftretenden Störungen auf diesen Signalen reduziert.

Um die Fläche der Platine klein halten zu können wurde entschieden die Platine beidseitig zu bestücken. Es wurden bereits beim Schaltungsentwurf nahezu ausschließlich SMD Komponenten gewählt, da diese im Unterschied zu Komponenten zur Durchsteckmontage problemlos maschinell platziert werden können. Komponenten zur Durchsteckmontage könnten zwar leichter per Hand gelötet werden, was die Fertigung eines Prototypen erleichtern würde, allerdings können diese nur sehr eingeschränkt maschinell montiert werden, was die Möglichkeit einer Serienfertigung effektiv ausschließen würde.

4.5.3. Komponentenplatzierung

Der erste Schritt beim Entwurf der Platine bestand darin eine ungefähre Größe und Form für die Platine festzulegen und die wichtigsten Komponenten zu platzieren. Natürlich konnte an diesem Punkt der Entwicklung noch keine genaue Größe festgelegt werden und die meisten Komponenten konnten auch noch nicht exakt positioniert werden.

Abbildung 36 zeigt die endgültige Position aller Komponenten, welche auf der Oberseite der Platine positioniert wurden. Diese hat sich, wie gesamt, erst nach und nach entwickelt. Als erstes wurden die Anschlussstecker, welche in der Abbildung markiert sind, positioniert.

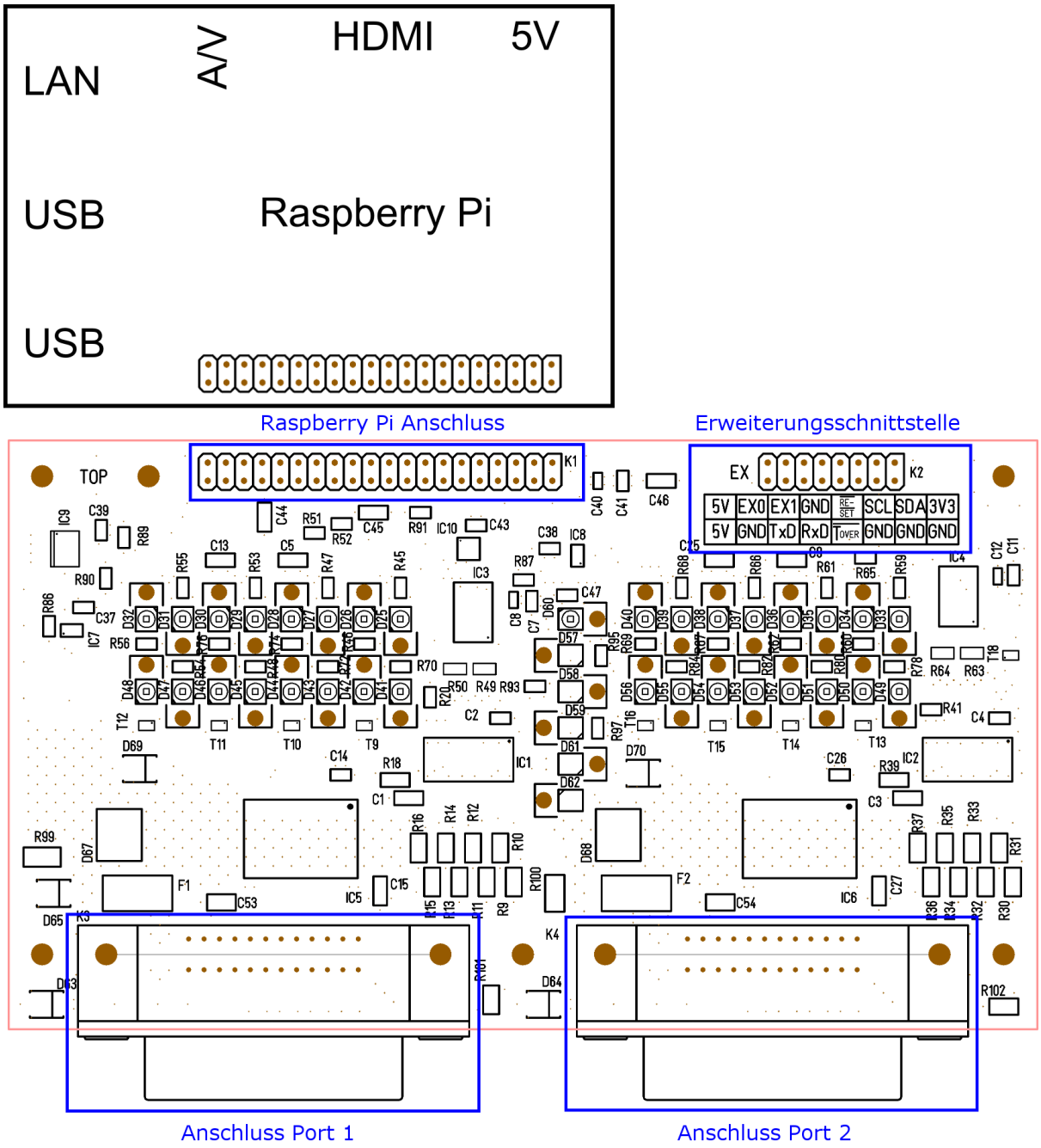


Abbildung 36: Platzierung der Anschlussstecker (Ansicht von oben)
Quelle: die Abmessungen des Raspberry Pi, welcher als Referenz dargestellt ist, stammen aus [89]

Die Anschlussstecker für Port 1 und 2 wurden auf einer Seite der Platine nebeneinander positioniert, da davon auszugehen ist, dass in der Regel alle Leitungen, welche zur Automatisierungsanlage gehen, mehr oder weniger in eine Richtung verlaufen werden. Die Breite der Platine wurde gerade so groß gewählt, dass die beiden Anschlussstecker der beiden Ports nebeneinander passen, an diesen bequem Kabel angeschlossen werden können und links, rechts und in der Mitte Platz für jeweils eine Befestigungsschraube, zum Fixieren der Platine in einem Gehäuse bleibt. Damit stand auch schon die endgültige Breite der Platine von 144 mm fest.

Der Anschlussstecker für den Raspberry Pi wurde gegenüber links oben so platziert, dass ein Raspberry Pi, welcher wie in Abbildung 36 unmittelbar neben der Platine liegt, mit einem kurzen Flachbandkabel direkt angeschlossen werden kann. Dabei wurde der Anschlussstecker so weit vom linken Rand entfernt gesetzt, dass der Raspberry Pi minimal über den Rand der Platine nach links herausragt. Dies ist sinnvoll, weil der Raspberry Pi auf der linken Seite Anschlussstecker hat, die außen an einem Gehäuse zugänglich sein müssen, die entworfene Platine hat hingegen auf der linken Seite keine Stecker. Der Rand der Platine kann also einigen Abstand zum Gehäuse haben. Wird dieser großzügig bemessen, bleibt mehr Spielraum für eventuelle Produktionsungenauigkeiten.

Geht man von einem rechteckigen Gehäuse aus, erkannt man, dass rechts oben neben dem Raspberry Pi Platz bleibt, da dieser nicht so breit ist wie die entworfene Platine. Dieser Platz könnte später für eine Erweiterung genutzt werden, daher wurde die Erweiterungsschnittstelle rechts oben an der Platine platziert.

Eine Platzierung des Raspberry Pi neben der entworfenen Platine scheint am sinnvollsten. Eine Platzierung des Raspberry Pi oberhalb oder unterhalb der Platine wurde in Erwägung gezogen, erscheint aber nicht sinnvoll. Platziert man den Raspberry Pi oberhalb der Platine, hat dies den Nachteil, dass ein großer Teil der Platinenfläche verdeckt wird und dort somit keine LEDs zu Anzeigezwecken platziert werden können. Eine Platzierung unterhalb wäre nur dann sinnvoll, wenn die Anschlussstecker für Port 1 und Port 2 von unten angebracht werden könnten, da sonst die Gesamthöhe zu groß wird. Eine Platzierung der Stecker von unten würde aber bedeuten, dass diese auf dem Kopf stehen würden, was ungeeignet scheint. Stecker, welche für eine Montage von unten konzipiert und entsprechend umgedreht sind, sind auf dem Markt offensichtlich nicht verfügbar.

Nach der Platzierung der Anschlussstecker wurden als nächstes alle Anzeige-LEDs platziert, da diese im Unterschied zu fast allen anderen Komponenten nicht so platziert werden können, wie dies für das Verlegen der Leiterbahnen am günstigsten ist, sondern in einem für den Nutzer nachvollziehbaren Muster angeordnet sein müssen. Diese wurden wie in Abbildung 37 dargestellt platziert. Sinnvollerweise befinden sich die LEDs, welche den Status der Ein- und Ausgänge von Port 1 anzeigen, oberhalb von diesem und die LEDs, welche zu Port 2 gehören, oberhalb von Port 2. Die restlichen LEDs, also die 5 V LED, die beiden Polaritäts-LEDs von den Ports und die drei zweifarbigen frei programmierbaren LEDs, wurden mittig platziert, weil nur dort genügend Platz übrig ist um auf dem Gehäuse neben jeder LED eine Beschriftung unterzubringen. Am linken und rechten Rand der Platine wäre nicht genügend Platz für eine solche Beschriftung, eine Platzierung weiter unten oder oben würde zu einer Vergrößerung der Platine führen. Prinzipiell könnte für jede LED ein separater Lichtwellenleiter eingesetzt werden um das Licht der LED an die Außenwand, in diesem Fall den Deckel, des Gehäuses zu führen. Dies hätte den Vorteil, dass die LEDs beliebig und auch einzeln angeordnet werden könnten. Beispielsweise könnte die 5 V LED einzeln an einer anderen Stelle platziert werden. Die Montage solcher Lichtwellenleiter wäre allerdings sehr aufwendig, da sehr viele LEDs vorhanden sind. Darüber hinaus sind Lichtwellenleiter im Vergleich zu an anderen verbauten Komponenten sehr teuer, so dass ein Lichtwellenleiter pro LED unverhältnismäßig wäre. Daher wurden die LEDs so angeordnet, dass Lichtwellenleiter eingesetzt werden können, bei denen jeder 8 bzw. 6 parallele Einzelleiter enthält. Damit sind insgesamt nur 5 Lichtwellenleiter nötig. Neben die LEDs wurden speziell für die ausgewählten Lichtwellenleiter passende Befestigungsbohrlöcher eingeplant, welche in der Abbildung

braun eingezeichnet sind. Der Durchmesser dieser Befestigungslöcher muss bei der Herstellung der Platine präzise eingehalten werden, da die Lichtwellenleiter in diese eingedrückt werden und dann ohne weitere Befestigung halten müssen.

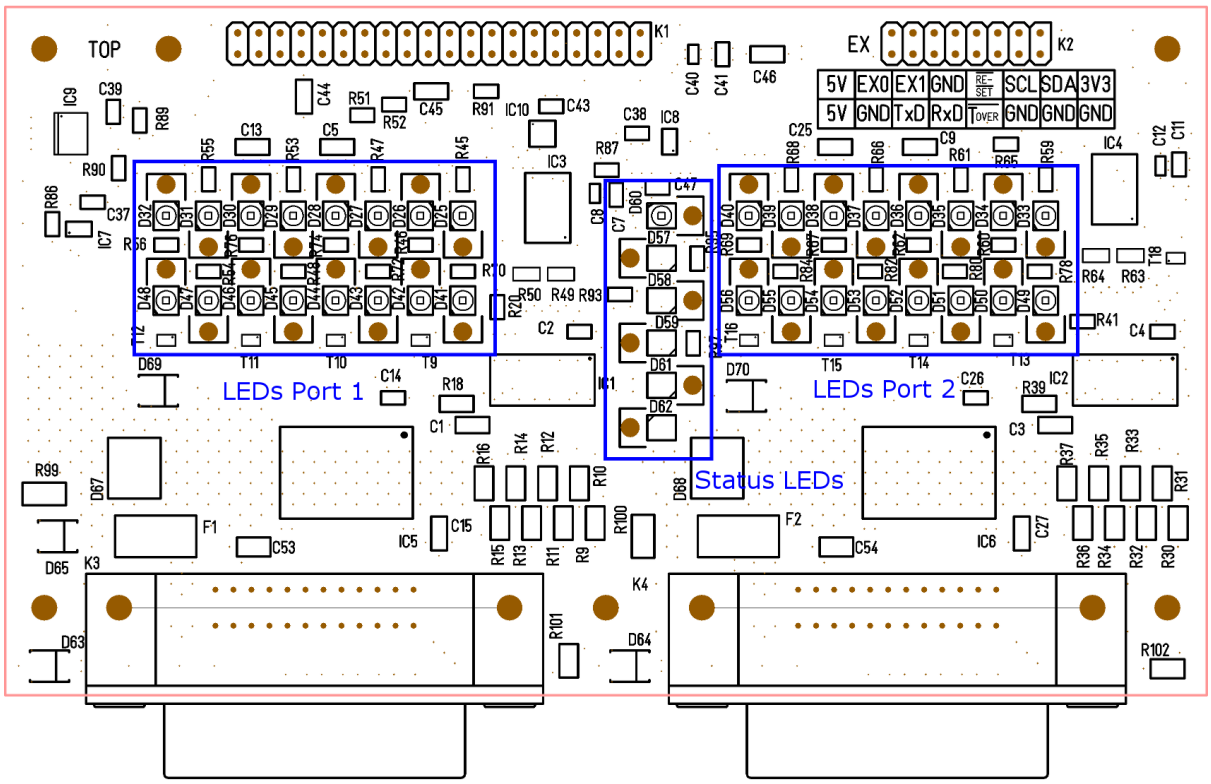


Abbildung 37: Platzierung der LEDs (Ansicht von oben)

Als nächstes wurden in die verbleibenden Zwischenräume die wichtigsten Bausteine platziert. Diese sind in Abbildung 38 markiert. Die Stromversorgungs- und Masseanschlüsse an den beiden Anschlusssteckern der beiden Ports befinden sich jeweils an der linken Seite des Steckers. Von den Stromversorgungsanschlüssen fließt unter Umständen ein nicht ganz unerheblicher Strom zu den Sicherungen (F) und über diese zu den Schottky-Dioden (D). Über diese Dioden fließt der Strom dann zu den Ausgabebausteinen und wieder zurück zu den Anschlusssteckern. Um die Verlustleistung gering zu halten und eine Überhitzung der Leitungen auf der Platine zu verhindern, müssen alle Leitungen über die dieser hohe Strom fließt möglichst kurz und breit sein. Daher wurden die Sicherungen direkt an den Stromanschlüssen platziert. Die Dioden daneben, da der Strom von den Sicherungen direkt zu diesen fließt. Die Ausgabebausteine wurden möglichst nah an den Pins platziert, welche zur Ausgabe dienen, um die Leitungen zu diesen möglichst kurz halten zu können. Um und zwischen den Dioden und den Ausgabebausteinen wurde etwas Platz für Kühlflächen gelassen.

Die Eingabebausteine wurden an der entgegengesetzten Seite der Anschlussstecker platziert um sicherzustellen, dass die Leitungen, welche die Eingabesignale transportieren, möglichst weit von den zuvor beschriebenen Leitungen mit hohen Stromstärken entfernt sind, da von den Leitungen mit hoher Stromstärke Störungen ausgehen könnten. Die IO-Expander wurden zwischen den Eingabebausteinen, den LEDs und dem Anschluss für den Raspberry Pi platziert, da sie mit allen diesen Komponenten mit vielen Leitungen verbunden sind.

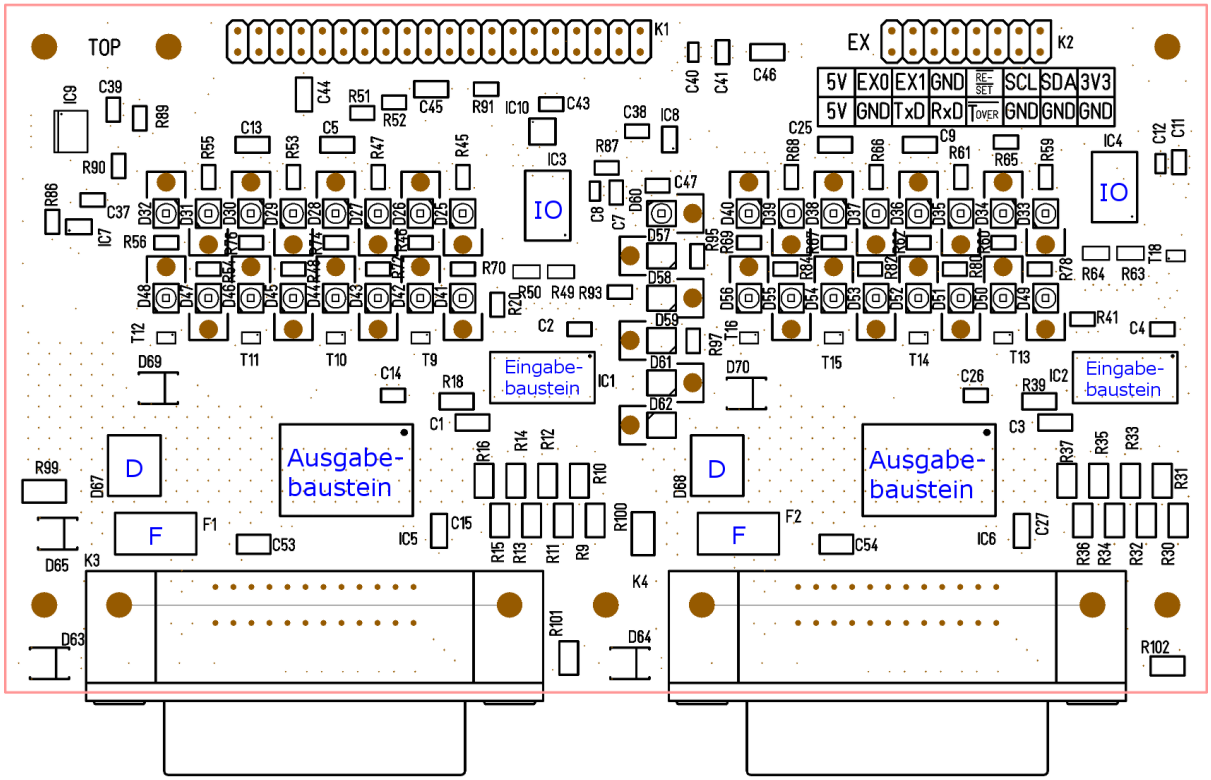


Abbildung 38: Platzierung der wichtigsten Bausteine (Ansicht von oben)

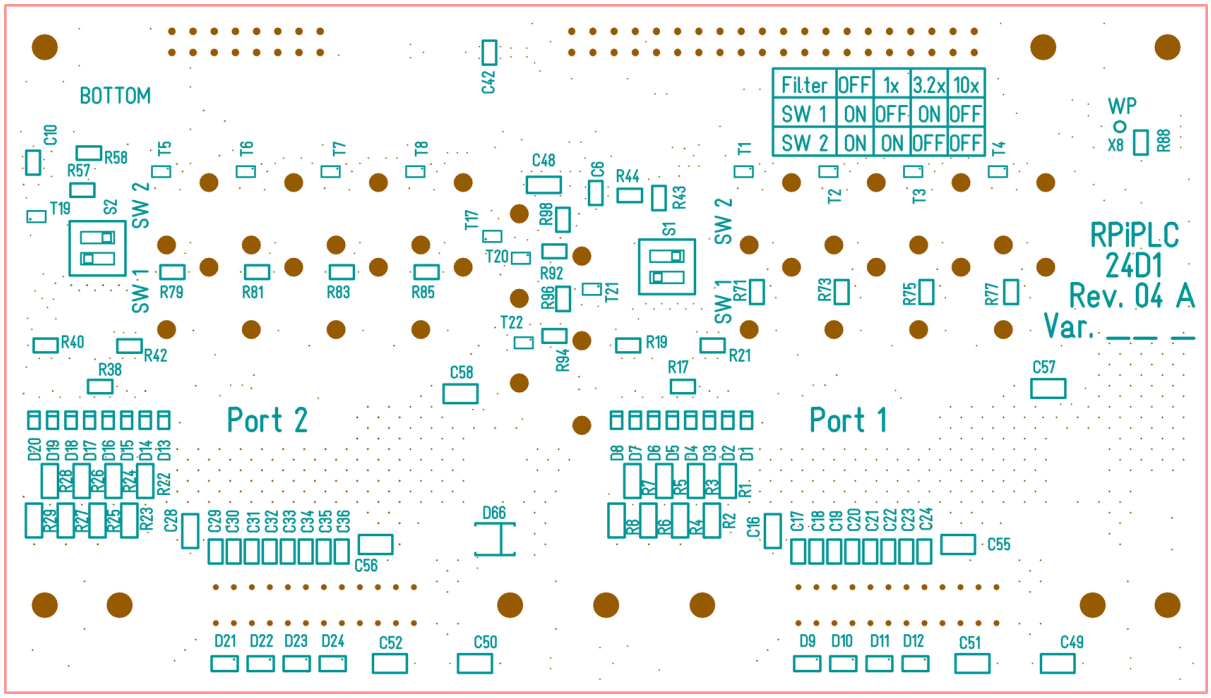


Abbildung 39: Komponenten auf der Unterseite (Ansicht von unten)

Abbildung 39 zeigt die Platzierung der Komponenten, welche sich auf der Unterseite der Platine befinden. Auf der Unterseite wurden nur wenige und vor allem kleine und leichte Bauteile platziert um den Einsatz von Kleber bei einer Serienfertigung gering zu halten.

Eine Ausnahme davon bilden die Schalter zur Konfiguration der Eingangsfilter, welche ebenfalls von unten platziert wurden, obwohl diese groß und vergleichsweise schwer sind. Es ist geplant, dass das Gehäuse von unten zu öffnen sein soll. Damit müssen die Schalter auch von unten platziert sein um relativ leicht zugänglich zu sein. Aus dem gleichen Grund wurde auch der Testpunkt, mit dem der Schreibschutz des ID-EEPROM deaktiviert werden kann, von unten platziert.

4.5.4. Masse- und Versorgungsflächen

Nachdem eine grobe Platzierung der wichtigsten Komponenten feststeht, muss bevor mit dem Verlegen der Leiterbahnen begonnen werden kann, festgelegt werden, wo Masse- und Versorgungsflächen platziert werden sollen. Wie schon einleitend erwähnt, wurde eine vier-lagige Platine gewählt um im Logikbereich jeweils eine Lage für Masse- und eine für Versorgungsflächen reservieren zu können.

Bezüglich der Platzierung der entsprechenden Kupferflächen gibt es eine wichtige Beschränkung. Die Platine muss immer symmetrisch sein. D.h. wenn die oberste Lage zum Verlegen von Leiterbahnen genutzt wird und die darunterliegende aus einer Kupferfläche besteht, muss auch die unterste Lage zum Verlegen von Bahnen genutzt werden und die darüber liegende Lage muss ebenfalls eine Kupferfläche sein. Es ist beispielsweis nicht möglich die oberen beiden Lagen als Kupferflächen auszuführen und die unteren beiden zum Verlegen von Signalleitungen zu nutzen. Auch wenn dies weder bei der Konstruktion noch bei der Herstellung der Platine ein Problem darstellen würde, könnte eine asymmetrische Platine zu Problemen bei der maschinellen Bestückung führen. Diese könnte sich beim Erhitzen nämlich ungleichmäßig ausdehnen und deshalb verbiegen.

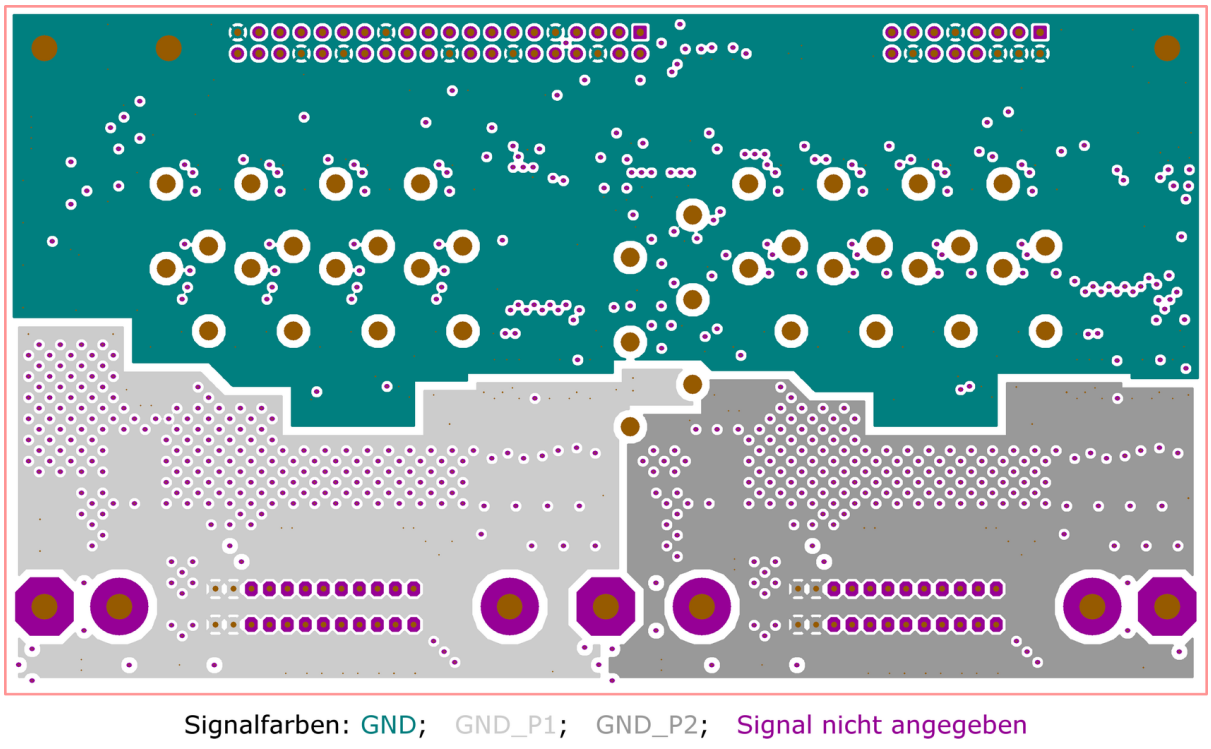


Abbildung 40: Innenlage mit Masseflächen (Ansicht von oben)

Damit bestehen nur zwei Möglichkeiten. Entweder können die beiden äußeren Lagen oder die beiden inneren Lagen zum Verlegen von Signalbahnen genutzt werden. Die anderen beiden können jeweils zum Realisieren einer Masse- und einer Versorgungsfläche genutzt werden. Da außen sehr viele Komponenten verteilt sind, wären außen angebrachte Kupferflächen häufig durch Bauteile unterbrochen. Außerdem wäre es erforderlich an jedem Anschluss eine Durchkontaktierung anzubringen um das Signal auf eine sich innen befindliche Leiterbahn zu bringen. Daher wurde entschieden die beiden

äußeren Lagen zum Verlegen von Signalbahnen zu nutzen und innen Versorgungs- und Masseflächen zu realisieren.

Abbildung 40 zeigt die Aufteilung der Innenlage, auf der Masseflächen realisiert wurden. Da es sich um eine rein digitale Schaltung handelt und alle Bauteile im Logikbereich mit einer gemeinsamen Masse (GND) arbeiten, wurde eine gemeinsame große Massefläche in diesem Bereich realisiert. Die beiden Ports sind galvanisch vom Logikbereich und untereinander isoliert. Alle Bauteile innerhalb eines Portbereichs arbeiten mit einer gemeinsamen Masse, daher wurde für jeden Port jeweils eine eigene große Massefläche vorgesehen.

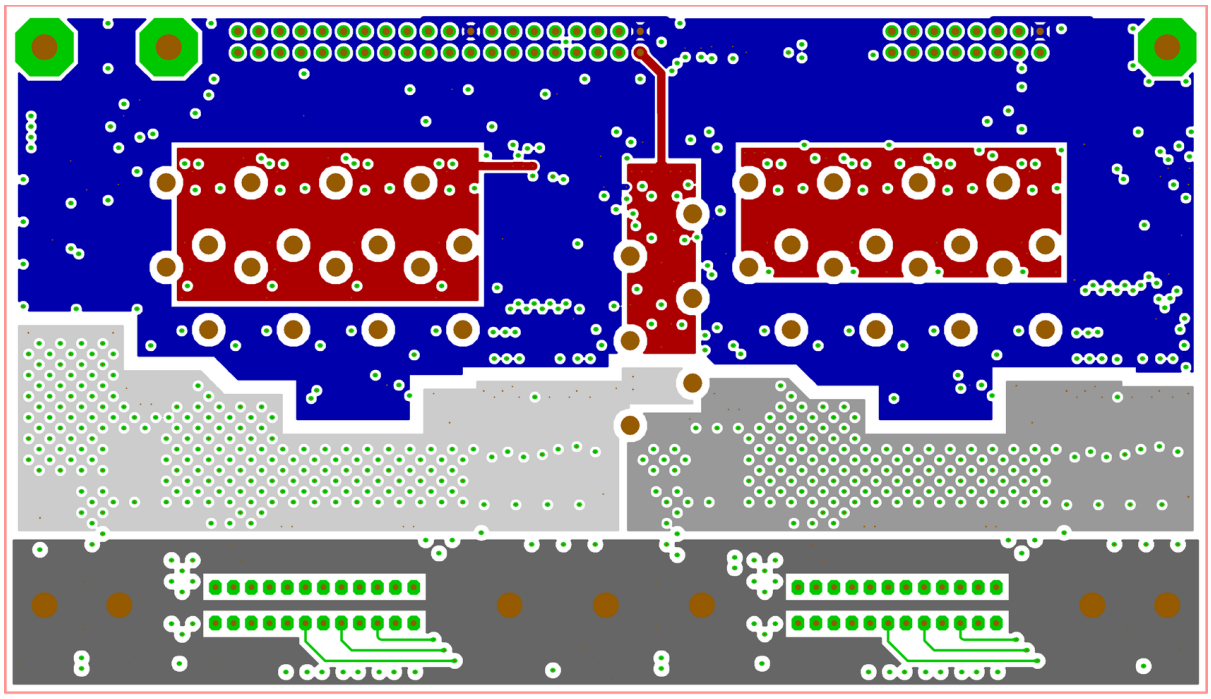


Abbildung 41: Innenlage mit Versorgungsflächen (Ansicht von oben)

Abbildung 41 zeigt die Aufteilung der Innenlage, auf welcher Versorgungsflächen realisiert wurden. Da im Logikbereich zwei verschiedene Spannungen, nämlich 3,3 V und 5 V zum Einsatz kommen und nur eine Lage zur Verfügung steht, konnte nicht für jede Spannung eine große solide Versorgungsfläche realisiert werden. Bei genauerer Betrachtung fällt allerdings auf, dass 5 V nur zur Versorgung der LEDs genutzt werden. Alle anderen Bauteile arbeiten mit 3,3 V. Daher wurde entschieden in den Bereichen, wo LEDs platziert sind, kleine 5 V Flächen und im übrigen Bereich eine zusammenhängende 3,3 V Fläche zu realisieren.

Da in den Portbereichen die Versorgungsflächen gleichzeitig als Kühlflächen dienen sollen, müssen diese außen platziert werden. Um die Platine symmetrisch zu halten, muss aber auch in diesem Bereich eine Kupferfläche auf der Innenlage, auf der sich die 3,3 V und 5 V Versorgungsflächen befinden, platziert werden. Wie in Abbildung 41 dargestellt, wurden hier einerseits weitere Masseflächen und eine Fläche, welche über Befestigungsschrauben mit dem Gehäuse verbunden wird und die Metallgehäuse der Stecker mit diesem verbindet, platziert. Damit muss die Verbindung der Stecker nicht auf einer der Außenlagen erfolgen.

4.5.5. Signalbahnen

Nachdem festgelegt wurde, wo welche Masse- und Versorgungsflächen untergebracht werden sollen und deren ungefähre Ausmaße festgelegt wurden, konnte mit der Verlegung von Leiterbahnen begon-

nen werden. Abbildung 42 zeigt die Leiterbahnen und Kupferflächen, welche auf der Oberseite der Platine platziert wurden. Abbildung 43 zeigt die Leiterbahnen und Kupferflächen auf der Unterseite.

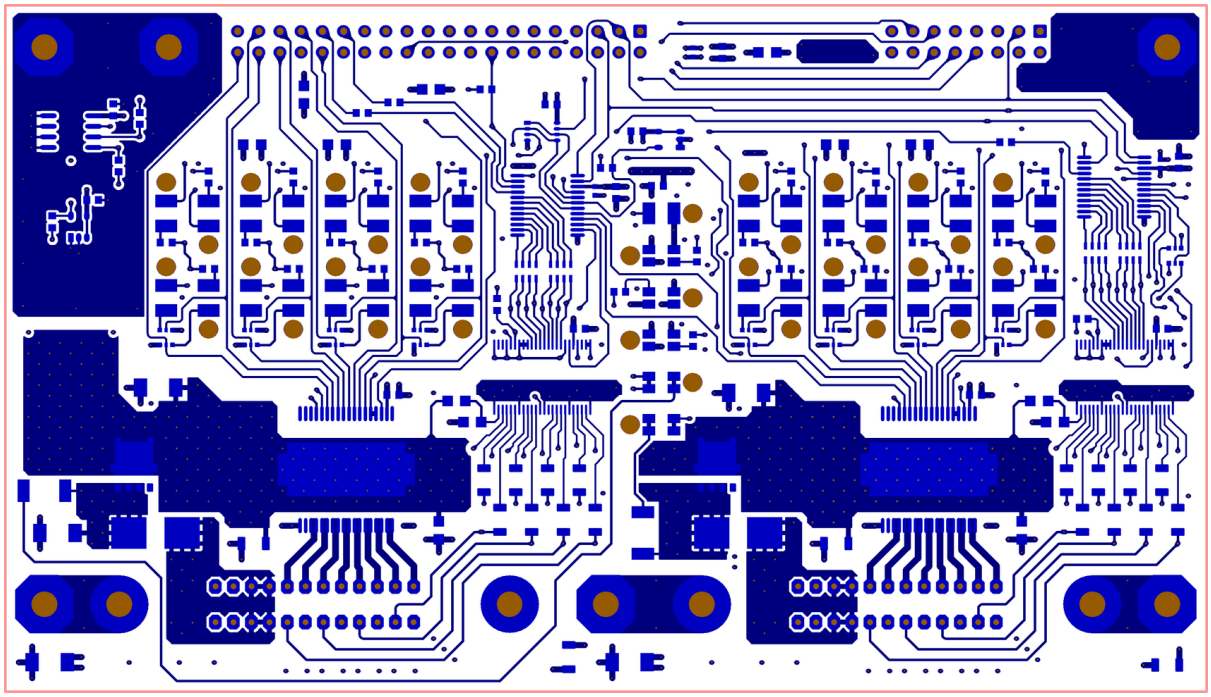


Abbildung 42: Leiterbahnen auf der Oberseite der Platine (Ansicht von oben)

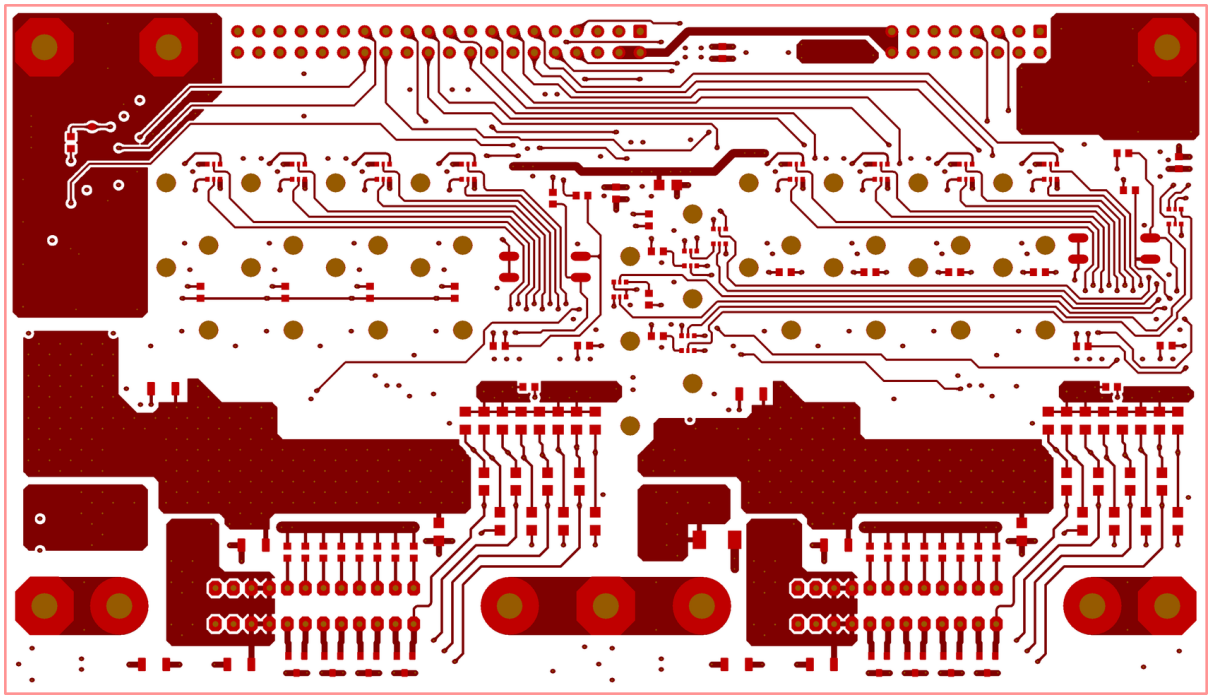


Abbildung 43: Leiterbahnen auf der Unterseite der Platine (Ansicht von oben)

Abbildung 43 ist so dargestellt, als würde man von oben durch die Platine hindurchschauen. Dies macht es einfacher die Verbindungen zwischen den oberen und den unteren Leiterbahnen nachzuvollziehen. In beiden Abbildungen sind jeweils die Bereiche, welche zwar mit Kupfer aber nicht mit Lötstopplack bedeckt sind, in einem helleren Farbton dargestellt. Weiße Bereiche sind in jedem Fall mit Lötstopplack bedeckt.

Wie zu erkennen ist, wurden horizontal verlaufende Signalbahnen tendenziell eher auf der Unterseite und vertikal verlaufende Signalbahnen bevorzugt auf der Oberseite verlegt. Dies ist in diesem Fall aufgrund der Platzierung der LEDs, zwischen denen vertikal Leiterbahnen verlegt werden können, horizontal aber kaum Platz für Leiterbahnen vorhanden ist, sinnvoll. Sowohl die Schottky Dioden als auch die Ausgabebausteine wurden in Kühlflächen eingebettet. Diese Flächen dienen gleichzeitig auch als Versorgungsflächen. Um die Kühlung weiter zu verstärken und die Platine besonders symmetrisch zu gestalten, wurden die Kühlflächen auf der Unterseite dupliziert und mit vielen Durchkontaktierungen mit den oberen Verbunden um eine gute Wärmeübertragung zu erreichen.

Links und rechts oben ergaben sich weitestgehend freie Bereiche. Diese wurden sowohl oben als auch unten mit weiteren Masseflächen ausgefüllt. Bei Anschlusspads in diesen Flächen und auch in allen anderen Flächen wurden Wärmefallen vorgesehen. Das bedeutet, dass die Anschlusspads nicht vollflächig in die Flächen integriert wurden, auch wenn sie das selbe Signal wie die Fläche tragen, sondern ein kleiner umlaufender Spalt zwischen Pad und Fläche frei gelassen wurde. Das Pad wurde dann mit einer oder mehreren Leiterbahnen mit der Fläche verbunden. Dies ist für eine manuelle Bestückung und evtl. Reparaturarbeiten notwendig. Würde man die Pads vollflächig in eine größere Flächen integrieren, wäre es praktisch unmöglich Bauteile an diese mit einem Lötkolben dran zu löten oder zu entfernen, da die Flächen so viel Wärme abführen würden, dass eine ausreichende Erhitzung des Pads mit einem Lötkolben nicht möglich wäre.

Eine Ausnahme bilden die Anschlusspads der Ausgabebausteine und der Schottky Dioden, an denen keine Wärmefallen vorgesehen wurden. Diese Bausteine können ohnehin nicht mit einem Lötkolben verlötet werden. Diese müssen, sofern kein Reflow-Ofen zur Verfügung steht, mit einem Heißluftföhn oder einem Heißluftlötgerät angebracht werden. Dabei wird Hitze auf eine größere Fläche übertragen, so dass auch die gesamte Kühlfläche aufgeheizt wird, womit das Fehlen von Wärmefallen kein Problem mehr darstellt. Durch den Verzicht auf Wärmefallen kann hingegen die Wärmeübertragung und damit die Kühlung im späteren Betrieb verbessert werden.

4.5.6. Namensgebung

Wie in der Einleitung dieser Arbeit beschrieben ist, wurde im Anschluss an das interdisziplinäre Projekt, in dem die Idee zur Entwicklung einer solchen SPS ursprünglich entstand, bereits eine weiterentwickelte Version des ursprünglichen Geräts konstruiert und gebaut. Dieses Gerät trug den Namen RPiPLC. Auch wenn es sich bei dem in dieser Arbeit entwickelten Gerät um eine vollständige Neuentwicklung handelt, wurde, wie anhand der Platinenbeschriftung in Abbildung 39 zu erkennen ist, der Name beibehalten.

Der erste Prototyp des RPiPLC trug die Revisionsnummer 1. Die Schaltung wurde später zur Revision 2 weiterentwickelt, von dieser wurde allerdings kein Prototyp mehr angefertigt, da die weitere Entwicklung, wie beschrieben, wegen Kompatibilitätsproblemen eingestellt wurde. Die in dieser Arbeit entwickelte Schaltung trug konsequenter Weise zunächst die Revisionsnummer 3. Von dieser wurde auch ein erster Prototyp gebaut, bei diesem kam es aber aufgrund der im Abschnitt 4.3.6 beschriebenen ungewöhnlichen Kennlinie der GPIO Pins der IO-Expander zu Problemen mit den Status-LEDs. Die Schaltung und das Platinenlayout wurden daraufhin minimal angepasst um die ungewöhnliche Kennlinie zu berücksichtigen. Die neue Version bekam die Revisionsnummer 4. Da beide Revisionen nahezu identisch sind, wird zur besseren Übersichtlichkeit Revision 3 in dieser Arbeit ansonsten nicht weiter erwähnt. Die Erkenntnisse über die ungewöhnliche Kennlinie wurden nahtlos in den Entwicklungsprozess in dieser Arbeit integriert.

4.5.7. Fertigung und Bestückung

Die eigentliche Fertigung (nicht die Bestückung) der Platine wurde bei der Firma Beta LAYOUT GmbH [88] in Auftrag gegeben. Abbildung 44 und Abbildung 45 zeigen die Ober- und Unterseite der Platine im Auslieferungszustand im Maßstab 1:1.

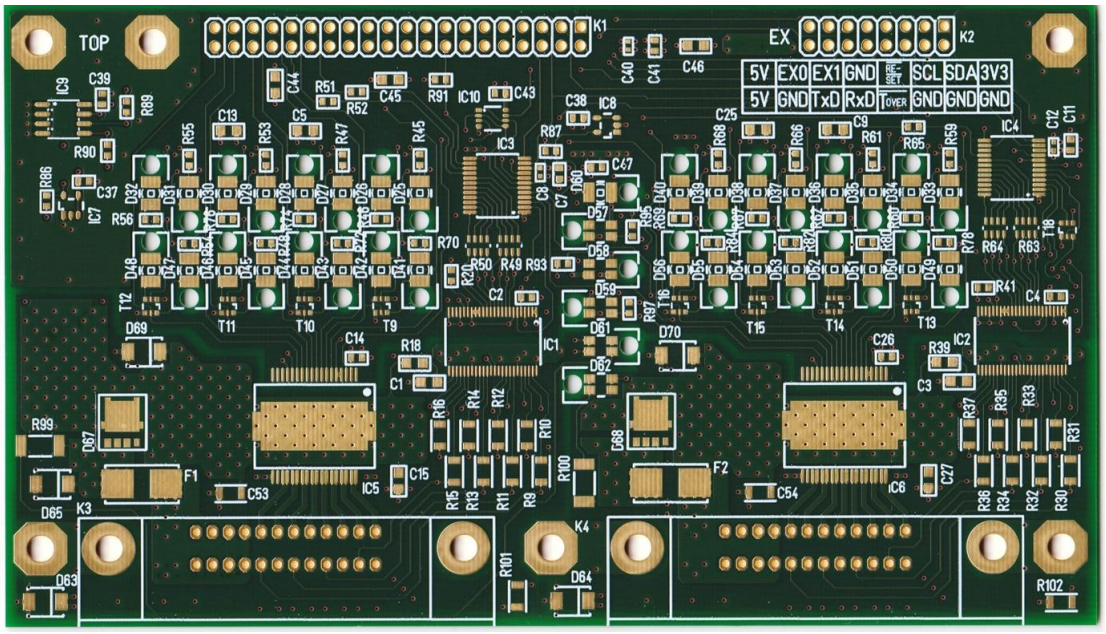


Abbildung 44: Oberseite der unbestückten Platine

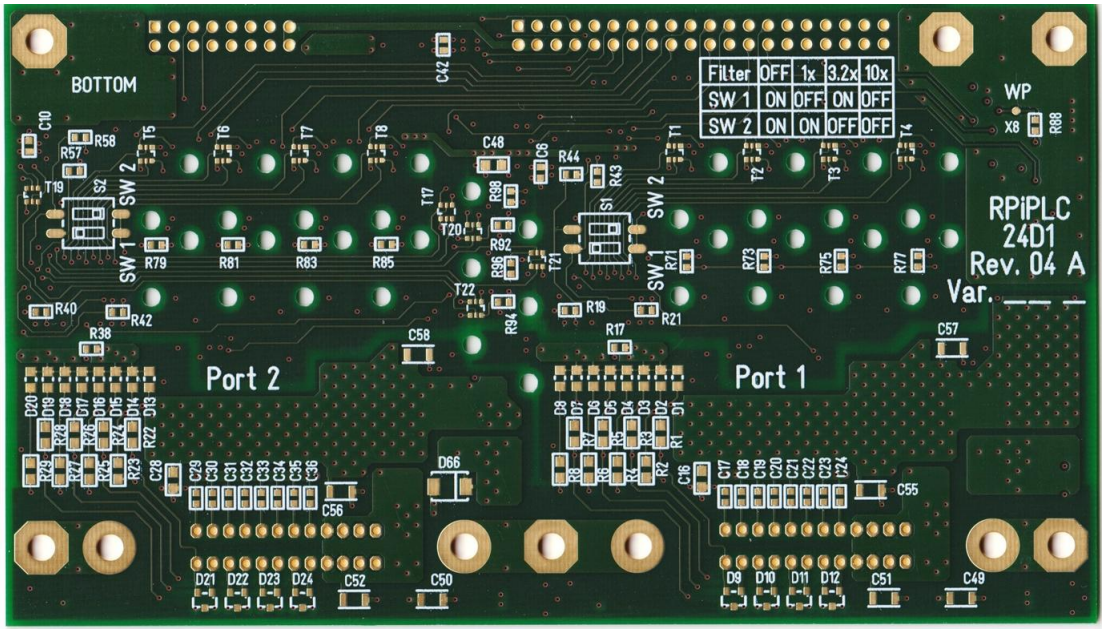


Abbildung 45: Unterseite der unbestückten Platine

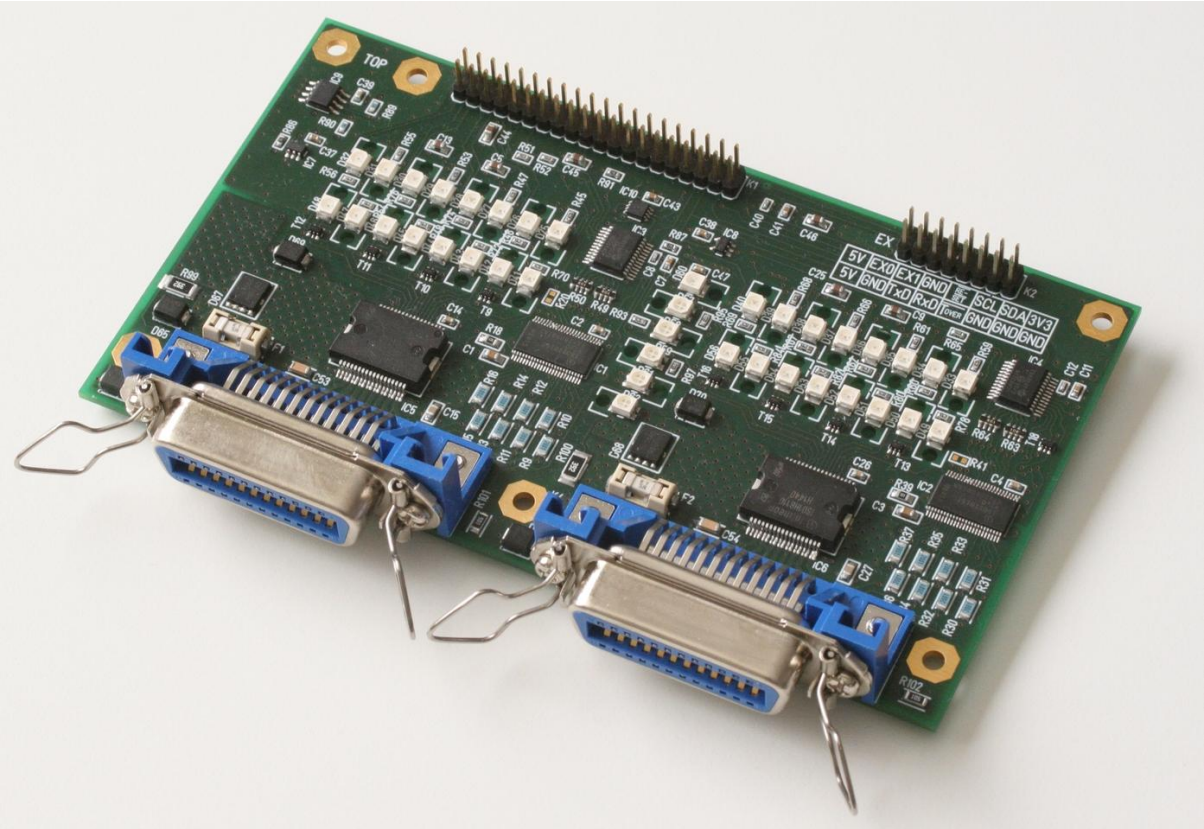


Abbildung 46: Oberseite der bestückten Platine

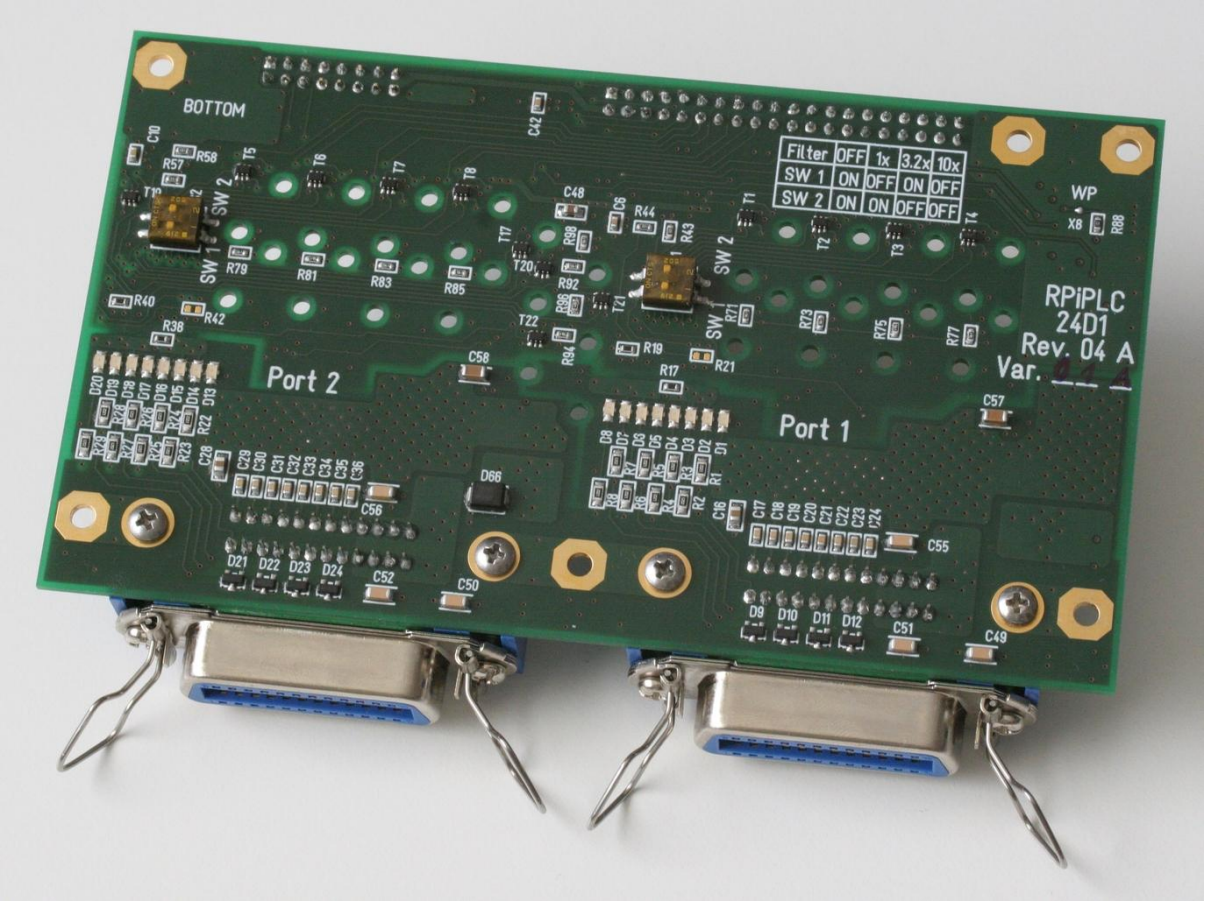


Abbildung 47: Unterseite der bestückten Platine

Die Bestückung der Platine konnte aus Kostengründen nicht in Auftrag gegeben werden, diese erfolgte vollständig manuell mit Lötstation und Heißluftföhn. Abbildung 46 zeigt ein Foto von der Oberseite der bestückten Platine, Abbildung 47 ein Foto der Unterseite. Fast alle Komponenten konnten mit einer Lötstation verlötet werden, ein Heißluftföhn war nur für die Ausgabebausteine und die Schottky Dioden erforderlich. Die Lichtleiter wurden zum Schluss in die Platine eingedrückt. Dank einer sehr präzisen Fertigung der Befestigungslöcher seitens des Platinenherstellers halten die Lichtleiter planmäßig ohne weitere Befestigungen. Abbildung 48 zeigt eine Detailaufnahme der montierten Lichtleiter.

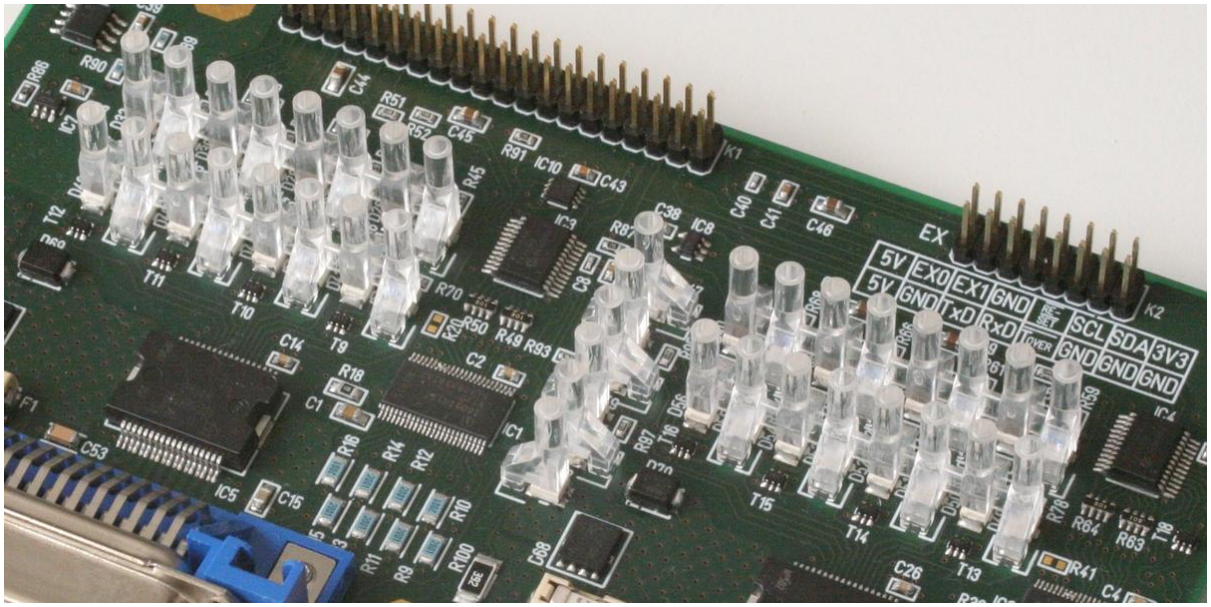


Abbildung 48: Detailaufnahme der Lichtleiter auf der Platine

Auf den Fotos sind stellenweise nicht entfernte Lötbrücken, also Kurzschlüsse zwischen einzelnen Pins, erkennbar. Wie durch einen einfachen Vergleich mit dem Platinenlayout zu erkennen ist, sind diese Lötbrücken nur an Stellen, an denen die Pins ohnehin auf der Platine miteinander verbunden sind. Daher haben die Kurzschlüsse an diesen Stellen keine Auswirkungen auf die Funktion. Diese wurden nicht entfernt um die betroffenen Bauteile nicht unnötigerweise zusätzlichem thermischem Stress auszusetzen.

4.6. Gehäuse

Zum Schutz der Gesamtschaltung wurde ein Gehäuse konstruiert, in dem der Raspberry Pi und die entwickelte Platine nebeneinander untergebracht werden können, wie dies in Abbildung 36 veranschaulicht wird. Die Verbindung der beiden Teile erfolgt mittels eines Flachbandkabels. Abbildung 49 zeigt eine computergenerierte Vorschau des konstruierten Gehäuses. Wie an den Anschlusssteckern und der zugehörigen Beschriftung zu erkennen ist, befindet sich der Raspberry Pi im hinteren Teil des Gehäuses auf der linken Seite, die entwickelte Platine erstreckt sich im vorderen Teil über die gesamte Breite abzüglich von etwa 2 mm links und rechts.

Das Gehäuse besteht im Wesentlichen nur aus zwei entsprechend gebogenen Aluminiumplatten. Zur Befestigung der Platine und des Raspberry Pi wurden im Inneren Abstandshalter in Form von Einpresselementen in der Deckplatte vorgesehen. Um die Oberfläche zu schützen ist eine Pulverbeschichtung vorgesehen, welche gleichzeitig auch die Enden der Einpresselemente verdeckt.

Die Höhe des Gehäuses wird durch die Höhe der Centronics-Buchsen definiert. Die Absenkung in der Deckplatte im Bereich der Leuchtanzeigen ermöglicht die Nutzung von Lichtleitern mit einer Stan-

dardlänge von 15 mm, sollte zur Gesamtstabilität beitragen und ist nicht so tief, dass die Sicht auf die Leuchtanzeigen dadurch nennenswert beeinträchtigt wäre.

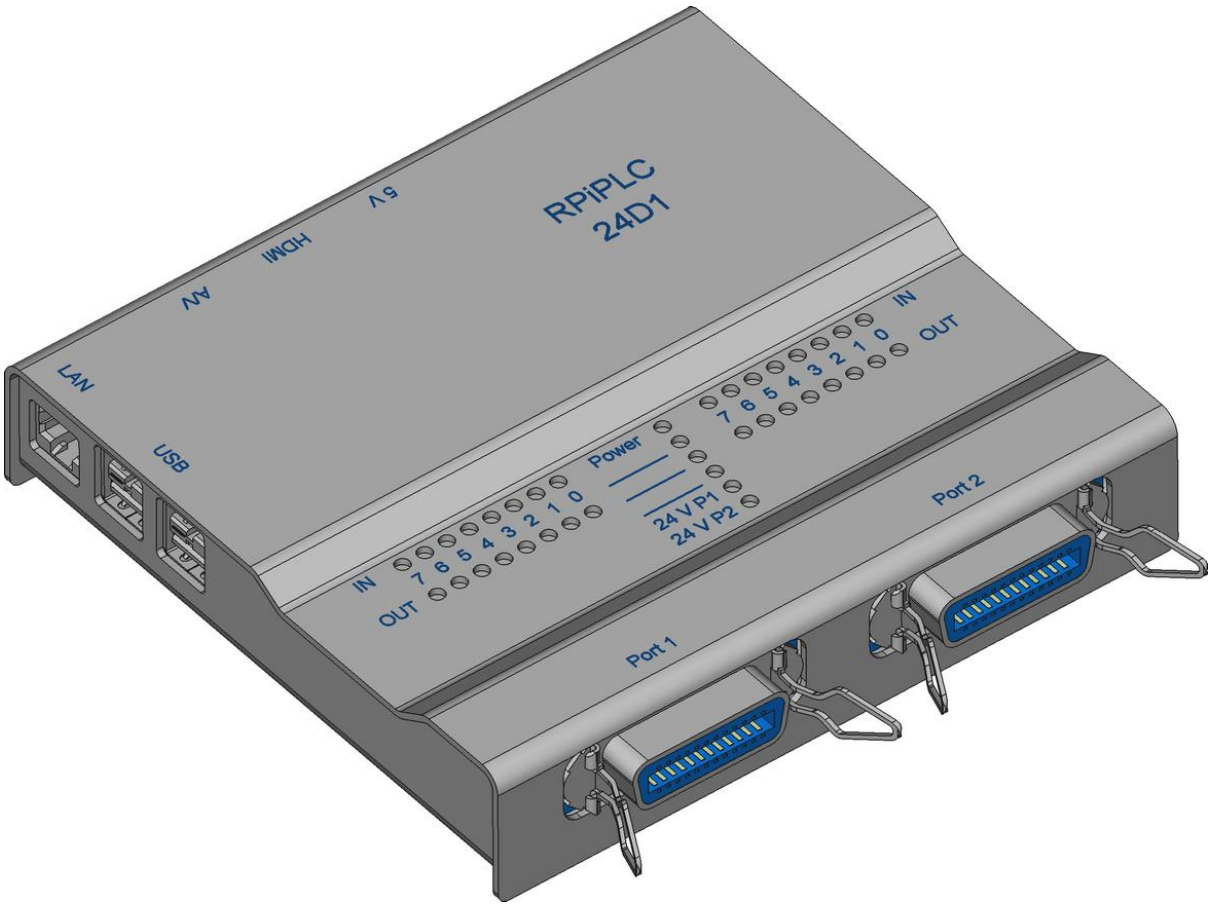


Abbildung 49: Vorschau vom Gehäuse
Quelle: Zur Darstellung der Anschlusstecker wurden 3D-Modelle von den jeweiligen Herstellern eingesetzt

Im Rahmen dieser Arbeit wurde ein vollständiger Konstruktionsplan für das Gehäuse angefertigt, allerdings wurde aus Zeit- und Kostengründen und aufgrund fehlender Geräte zum präzisen Biegen von Aluminiumblechen und zum Montieren von Einpresselementen kein Prototyp des Gehäuses angefertigt.

4.7. ID-EEPROM und Betriebssystemkonfiguration

Am Raspberry Pi ist eine I^2C Schnittstelle für den Anschluss eines sog. ID-EEPROM Chips reserviert. An dieser wurde, wie bereits im Abschnitt 4.4.7 beschrieben, der von den Herstellern empfohlene EEPROM Chip des Herstellers OnSemi angeschlossen. Dieser besitzt eine Speicherkapazität von 32 kBit bzw. 4 kByte und unterstützt Übertragungsgeschwindigkeiten von 100 kHz bis 1 MHz. [90]

Auf diesem Chip können Daten hinterlegt werden, welche die am Raspberry Pi angeschlossene Schaltung identifizieren. Diese Daten werden beim Bootvorgang vom Bootloader, welcher von den Herstellern des Raspberry Pi entwickelt wurde, automatisch ausgelesen und ausgewertet. Erwartet werden Herstellerinformationen wie Produktname, Hersteller und Seriennummer der Schaltung, die gewünschten Einstellungen, die für GPIO Pins des Raspberry Pi durchgeführt werden sollen, sowie ein Device Tree Overlay. Die einzelnen Informationen werden in den folgenden Abschnitten genauer diskutiert. Es wird ebenfalls beschrieben, welche Daten auf dem ID-EEPROM Chip des gebauten Prototypen hinterlegt wurden und wie dabei vorgegangen wurde. [91]

4.7.1. Herstellerinformationen

Als erstes werden die Herstellerinformationen erwartet. Diese bestehen aus einer Seriennummer, genau gesamt einer universal eindeutigen Identifikationsnummer (UUID) mit 128 Bit, einer Produkt ID, einer Produkt Version, dem Namen des Herstellers und einer Produktbezeichnung.

Die Seriennummer soll nach Vorgabe der Hersteller des Raspberry Pi der RFC 4122 genügen. Die RFC 4122 definiert verschiedene Möglichkeiten zur dezentralen Erzeugung von Nummern, die mit hoher Wahrscheinlichkeit global eindeutig sind. Als Eingabe zur Erzeugung einer solchen Nummer können je nach Einsatzzweck beispielsweise MAC-Adressen, Zeitstempel, Benutzernummern oder Prüfsummen von verschiedenen Objekten eingesetzt werden. Alternativ können auch Zufallszahlen genutzt werden.

Da im Falle einer Seriennummer keine der vorgeschlagenen Informationen sinnvoll scheint, wurde entschieden die Version zu wählen, welche auf zufälligen Zahlen basiert. UUIDs werden typischerweise in hexadezimaler Schreibweise wie im Codeausschnitt 4 in der zweiten Zeile gezeigt, dargestellt. Werden zur Erzeugung einer UUID zufällige Zahlen als Grundlage genutzt, müssen nur die erste Ziffer des dritten Blocks 4 und die erste Ziffer des vierten Blocks entweder 8, 9, A oder B sein, alle anderen Ziffern können zufällig bzw. frei gewählt werden. [26, Seite /Universally_unique_identifier] [91]

```
# Seriennummer
product_uuid 52506950-4c43-4c46-b60e-000000000000

# Produkt ID
product_id 0x0001

# Produkt Version
product_ver 0x0004

# Hersteller
vendor "Oliver Barta"

# Produktbezeichnung
product "RPiPLC 24D1 Rev. 04"
```

Codeausschnitt 4: Herstellerinformationen für den ID-EEPROM Chip

Es wurde beschlossen bei dem Prototypen und bei alle nachfolgenden Ausführungen des entwickelten Geräts die ersten vier Blöcke auf den Wert, welcher im Codeausschnitt 4 gezeigt wird, zu setzen. Diese Ziffern entsprechen der ASCII Kodierung von „RPiPLC“ gefolgt von wirklich zufällig gewählten Ziffern. Der mit Nullen gefüllte Bereich soll für jedes einzelne Gerät auf einen eigenen Wert gesetzt werden. Die Produkt ID wurde einfach auf 1 gesetzt, die Version entsprechend der Revisionsnummer der Schaltung, Hersteller und Produktbezeichnung wurden wie dargestellt gesetzt.

```
pi@raspberrypi /proc/device-tree/hat $ for file in *;
> do echo -e "$file    \t`cat $file`";
> done
name            hat
product         RPiPLC 24D1 Rev. 04
product_id      0x0001
product_ver     0x0004
uuid            52506950-4c43-4c46-b60e-8716b429d48a
vendor          Oliver Barta
```

Codeausschnitt 5: Herstellerinformationen im Dateisystem

Die Herstellerinformationen, welche aus dem ID-EEPROM ausgelesen werden, werden für den Zugriff durch Anwendungsprogramme im Dateisystem unter /proc/device-tree/hat bereitgestellt. Für jede Information wird eine eigene Datei angelegt. Codeausschnitt 5 zeigt den Inhalt der entsprechenden Dateien, wenn die Daten aus Codeausschnitt 4 im ID-EEPROM hinterlegt werden.

4.7.2. Konfiguration für GPIO-Pins

Nach den Herstellerinformationen wird als nächstes die Konfiguration der GPIO Pins erwartet. Hier kann theoretisch angegeben werden, wie viel Strom ein GPIO Pin liefern soll (vgl. Beschreibung der entsprechenden Einstellung für die GPIO Pins der IO-Expandern von NXP im Abschnitt 4.3.4), ob die Anstiegs- und Abfallgeschwindigkeit beim Schalten von Ausgängen beschränkt werden soll, ob als Eingang konfigurierte Pins eine Hysterese bei der Erkennung des Eingabewertes nutzen sollen und ob die angeschlossene Schaltung den Raspberry Pi mit Strom versorgt. Diese grundlegenden Einstellungen werden für alle GPIO Pins gemeinsam festgelegt. Darüber hinaus kann für jeden Pin einzeln die Funktion gewählt und ein interner Pull-Up oder Pull-Down Widerstand aktiviert werden.

Die in diesem Bereich des ID-EEPROM angegebenen Einstellungen werden gegenwärtig allerdings nur teilweise angewendet, da die Firmware des Raspberry Pi an dieser Stelle noch nicht vollständig implementiert ist. [91, Seite /blob/master/designguide.md] Für das vorliegende Gerät stellt dies allerdings kein Problem dar, da an dieser Stelle keine von der Standkonfiguration abweichenden Einstellungen erforderlich sind. Alle benötigten Einstellungen können später von der Anwendungssoftware, speziell der geplanten Steuerungsbibliothek, vorgenommen werden.

Auch wenn viele der Einstellungen noch gar nicht ausgewertet werden, wurden an dieser Stelle vorsorglich folgende Einstellungen vorgenommen. Es wurde ein mittlerer Wert für die Stromstärke gewählt, eine Beschränkung der Anstiegs- und Abfallzeiten und die Nutzung einer Hysterese wurden aktiviert. Es wurde angegeben, dass die angeschlossene Schaltung den Raspberry Pi nicht mit Strom versorgt, da das Gegenteil der Fall ist. Alle GPIO Pins wurden als Eingang konfiguriert und es wurde für jeden Pin der standardmäßig festgelegte Pull-Up- bzw. Pull-Down-Widerstand gewählt. Eine Anwendung dieser Einstellungen sollte das Verhalten der Schaltung tendenziell verbessern, es stellt aber kein Problem dar, wenn bei all diesen Konfigurationswerten der Standwert beibehalten wird. [91]

4.7.3. Device Tree Overlay

Den letzten Teil, der im ID-EEPROM erwartet wird, stellt ein sog. Device Tree Overlay dar. Unter Linux wird seit einiger Zeit bevorzugt ein sog. Device Tree zur Beschreibung der Hardware genutzt. Dieser wird kompiliert in binärer Form auf einem Gerät hinterlegt und beim Hochfahren des Geräts vom System ausgewertet. Der Device Tree informiert das System über die vorhandene Hardware und die nötige Konfiguration. Das System kann dann entsprechend dieser Informationen die gewünschte Konfiguration der Hardware vornehmen und passende Treiber laden.

Dieses Konzept wird mittlerweise auch von Raspbian, der für den Raspberry Pi empfohlenen und im Weiteren eingesetzten Linux Distribution, genutzt. Folglich steht für den Raspberry Pi ein Device Tree zur Verfügung. Wird, wie im vorliegenden Fall, zur üblichen Hardware des Raspberry Pi weitere Hardware hinzugefügt, können Informationen über diese ebenfalls im Device Tree hinterlegt werden um das Betriebssystem über deren Vorhandensein und deren Eigenschaften zu informieren. Diese zusätzlichen Informationen werden bevorzugt nicht im normalen Device Tree des Raspberry Pi, sondern in Form eines sog. Device Tree Overlays hinterlegt. Da dieses Informationen speziell über die zusätzlich angeschlossene Hardware enthält, wird dieses bevorzugt nicht auf der SD-Karte, wo das restliche System liegt, sondern direkt auf der angeschlossenen Hardware, genauer im ID-EEPROM Chip, in kompilierter Form hinterlegt. Von dort wird es dann vom Bootloader des Raspberry Pi automatisch ausgelesen. Das Device Tree Overlay und der normal Device Tree des Raspberry Pi werden

dann zu einem gemeinsamen Device Tree kombiniert. Mit einem solchen Device Tree Overlay können neue Informationen zum bestehenden Device Tree hinzugefügt oder einzelne bereits vorhandene Informationen mit neuen Werten überschrieben werden. [91]

Um dem Nutzer größtmögliche Kontrolle über die Hardware zu gewähren, wurde beim vorliegenden Gerät beschlossen möglichst generische Treiber zu nutzen und die Interaktion mit der Hardware weitestgehend aus dem Nutzer-Modus durchzuführen. Daher ist es nicht erforderlich das Betriebssystem über die Struktur der neuen Hardware zu informieren. Es ist lediglich erforderlich dem Betriebssystem mitzuteilen, dass die I^2C Schnittstelle genutzt werden soll und mit welcher Übertragungsgeschwindigkeit diese arbeiten soll.

Der Device Tree des Raspberry Pi liegt im Quellcode des Linux-Kernels, wie dieser im offiziellen git-Repository der Raspberry Pi Entwickler vorliegt [92], unter linux/arch/arm/boot/dts/ in der Datei bcm2709-rpi-2-b.dts. Diese bindet die Datei bcm2709.dtsi ein, welche wiederum die Datei bcm2708_common.dtsi einbindet. Die nötige Konfiguration zur Nutzung der I^2C Schnittstelle wird in bcm2709-rpi-2-b.dts und in bcm2708_common.dtsi in Abschnitten mit dem Namen „i2c1“ vorgenommen. Dort wird unter anderem die Übertragungsgeschwindigkeit auf 100 kHz festgelegt und das Interface wird als deaktiviert markiert. [92]

```
// Definitions for RPiPLC 24D1 Rev. 04
/dts-v1/;
/plugin/;

/ {

    compatible = "brcm,bcm2709";

    // enable i2c interface
    fragment@0 {
           target = <&i2c1>;
           __overlay__ {
                  status = "okay";
                  clock-frequency = <1000000>;
           };
    };

};
```

Codeausschnitt 6: Quellcode des genutzten Device Tree Overlays

Zur Nutzung des I^2C genügt es das Interface als aktiv zu markieren. Dazu wurde ein Device Tree Overlay erstellt, dessen Quellcode im Codeausschnitt 6 gezeigte wird. Dieses Overlay besteht aus nur einem Fragment, welches zunächst angibt, dass der Abschnitt mit der Bezeichnung „i2c1“ im normalen Device-Tree des Raspberry Pi modifiziert werden soll. Danach muss nur noch der Wert des Attributes „status“, welcher in bcm2708_common.dtsi auf „disabled“ gesetzt wird, auf „okay“ gesetzt werden. Dadurch wird das I^2C Interface beim Hochfahren des Systems automatisch aktiviert und der passende Treiber wird geladen. Da die eingesetzten IO-Expander deutlich höhere Übertragungsgeschwindigkeiten unterstützen, wird zusätzlich noch die Übertragungsgeschwindigkeit von 100 kHz auf 1 MHz erhöht.

4.7.4. Hinterlegen der Informationen im ID-EEPROM Chip

Der ID-EEPROM Chip kann vom Raspberry Pi aus beschrieben werden. Passende Hilfsprogramme dafür sind im git-Repository der Raspberry Pi Entwickler [91, Seite /tree/master/eepromutils] verfügbar. Normalerweise wird die I^2C Schnittstelle, an der das ID-EEPROM angeschlossen ist, von der GPU kontrolliert. Bevor über diese die gewünschten Daten zum EEPROM Chip übertragen werden

können, muss das System so konfiguriert werden, dass auf die I²C Schnittstelle von der CPU aus zugegriffen werden kann. Dazu muss in die Datei „/boot/config.txt“ die Zeile, welche in Codeausschnitt 7 gezeigt wird, hinzugefügt werden. Nach einem Neustart des Systems kann dann auf die I²C Schnittstelle von der CPU aus zugegriffen werden.

```
dtparam=i2c_vc=on
```

Codeausschnitt 7: Konfigurationsparameter zum Zugriff auf die I²C Schnittstelle der GPU

Die gewünschte Konfiguration, welche in das EEPROM geschrieben werden soll, kann in menschenlesbarer Form, wie in Codeausschnitt 4 exemplarisch gezeigt, verfasst werden. Um diese dann ins richtige maschinenlesbare Format zu konvertieren, kann das Hilfsprogramm eepmake aus dem git-Repository genutzt werden. Dieses wird, wie in Codeausschnitt 8 gezeigt, aufgerufen. eeprom_settings.txt ist dabei der Name der Datei, die die Konfiguration in menschenlesbarer Form enthält. Eine Beispielkonfiguration liegt dem Programm bei. eeprom.eep ist der Name der Ausgabedatei, in die das erzeugte binäre Speicherabbild für den EEPROM Chip gespeichert werden soll und rpiplc-24d1-r04-overlay.dtb gibt die Datei an, welche die kompilierte Version des Device Tree Overlays enthält, das eingebunden werden soll.

```
./eepmake eeprom_settings.txt eeprom.eep rpiplc-24d1-r04-overlay.dtb
```

Codeausschnitt 8: Aufruf des Hilfsprogramms eepmake

Das erzeugte Speicherabbild kann dann, wie in Codeausschnitt 9 gezeigt, auf den EEPROM Chips geschrieben werden. Das Shell-Script eepflash.sh ist ebenfalls im git-Repository enthalten.

```
sudo ./eepflash.sh -w -f=eeprom.eep -t=24c32
```

Codeausschnitt 9: Befehl zum Beschreiben des ID-EEPROM Chips

Während des Schreibvorgangs muss natürlich der Schreibschutz des ID-EEPROM Chips deaktiviert werden. Dazu muss der Testpunkt auf der Unterseite der entwickelten Platine während des Schreibvorgangs mit Masse verbunden werden. Nach Abschluss des Vorgangs muss die zu Beginn hinzugefügte Konfigurationszeile aus „/boot/config.txt“ wieder entfernt werden. Diese Änderung wird ebenfalls erst nach einem Neustart wirksam.

4.7.5. Betriebssystem und Konfiguration

Prinzipiell kann eine normale Version der Raspbian Distribution, wie diese für den Raspberry Pi von den Entwicklern angeboten wird, zum Betrieb der entwickelten SPS genutzt werden. Es sind an sich keinerlei Modifikationen notwendig, was die Einarbeitung erleichtert und so der Benutzerfreundlichkeit des entwickelten Geräts zugutekommt.

Der benötigte Treiber i2c-bcm2708 für die I²C Schnittstelle, über die die IO-Expander angebunden sind, wird vom System beim Hochfahren automatisch geladen, da diesem aufgrund des Device Tree Overlays, welches im ID-EEPROM platziert wurde, bekannt ist, dass diese Schnittstelle genutzt werden soll. Die Konfiguration der Schnittstelle ist ebenfalls bekannt, da diese, wie beschrieben, bereits im normalen Device Tree des Raspberry Pi enthalten ist

Wie bereits zuvor diskutiert, erscheint es zur Sicherstellung einer hohen Kontrolle über das zeitliche Verhalten der Hardware beim vorliegenden Gerät sinnvoll auf die Abstraktion, welche von Treibern geboten wird, teils zu verzichten, bevorzugt generische Treiber zu nutzen und die Interaktion mit der Hardware überwiegend in einer Anwendungssoftware bzw. einer Steuerungsbibliothek zu implementieren. Daher soll kein spezieller Treiber für die eingesetzten IO-Expander verwendet werden, sondern

nur ein Treiber, der eine Kommunikation über die I^2C Schnittstelle aus dem User-Modus erlaubt. Diese Funktionalität stellt das i2c-dev Modul bereit, welches die I^2C Schnittstelle in Form einer Device File verfügbar macht. Dieses Modul muss vor der Benutzung geladen werden. Dies kann entweder manuell oder durch Hinzufügen von „i2c-dev“ in die Konfigurationsdatei „/etc/modules“ erfolgen. Wird letztere Möglichkeit gewählt, wird das Modul automatisch bei jedem Hochfahren geladen. Die I^2C Schnittstelle steht dann als „/dev/i2c-1” zur Verfügung und kann beispielsweise mit den Programmen, welche im i2c-tools Package für Raspbian enthalten sind, oder mit eigener Anwendungssoftware genutzt werden.

Auf die GPIO Pins des Raspberry Pi kann komfortable über ein sysfs Interface zugegriffen werden. Einzelne Pins können durch Schreiben der Pin-Nummer in die Datei „/sys/class/gpio/export“ exportiert werden. Für jeden exportierten Pin wird in „/sys/class/gpio“ ein Unterverzeichnis der Form gpioN, wobei N für die Nummer des Pins steht, erzeugt. Der Inhalt eines solchen Ordners wird exemplarisch im Codeausschnitt 10 gezeigt. Von besonderem Interesse sind die Dateien „direction“, „edge“ und „value“. Durch Schreiben von „in“ oder „out“ / „low“ / „high“ in die Datei „direction“ kann der entsprechende GPIO Pin als Ein- oder als Ausgabe konfiguriert werden. Durch Schreiben von „none“, „rising“, „falling“ oder „both“ in die Datei „edge“ kann festgelegt werden, ob bei einer steigenden, einer fallenden oder bei beiden Flanken ein Interrupt erzeugt werden soll. Die Datei „value“ gibt den aktuellen Wert eines GPIO Pins wieder. Im Falle einer Ausgabe kann dieser über diese Datei auch gesetzt werden. Im Falle einer Eingabe kann mit den Funktionen poll und select auf das Eintreten eines Interrupts auf dieser Datei gewartet werden. [93]

```
pi@raspberrypi /sys/class/gpio/gpio21 $ ls -l
total 0
-rwxrwx--- 1 root gpio 4096 Jan 25 13:25 active_low
lrwxrwxrwx 1 root gpio    0 Jan 25 13:25 device -> ../../../3f200000.gpio
-rwxrwx--- 1 root gpio 4096 Jan 25 13:25 direction
-rwxrwx--- 1 root gpio 4096 Jan 25 13:25 edge
lrwxrwxrwx 1 root gpio    0 Jan 25 13:25 subsystem -> ../../../../../class/gpio
-rwxrwx--- 1 root gpio 4096 Jan 25 13:25 uevent
-rwxrwx--- 1 root gpio 4096 Jan 25 13:25 value
```

Codeausschnitt 10: Inhalt eines gpioN Unterordners

Um erste Referenzwerte zu bekommen sollen die ersten Messungen des zeitlichen Verhaltens der entwickelten SPS, welche im nächsten Kapitel behandelt werden, mit einer unmodifizierten Version der Raspbian Distribution, wie diese von den Entwicklern angeboten wird, durchgeführt werden. Erst danach soll die geplante Integration des CONFIG_PREEMPT_RT Patches durchgeführt werden. Anschließend sollen weitere Messungen erfolgen um die Verbesserung des zeitlichen Verhaltens, welche von dem Patch erwartet wird, zu bewerten.

4.8. Steuerungsbibliothek

Um eine einfache Integration in Steuerungssoftware zu ermöglichen wurde für die entwickelte SPS eine Steuerungsbibliothek entwickelt. Diese wurde in C geschrieben und kann direkt aus C und C++ Programmen genutzt werden. Sie ist prinzipiell für verschiedene Linux-Distributionen geeignet, der vollständige Funktionsumfang wurde bisher aber nur unter Raspbian getestet. Passend zur entwickelten Bibliothek wurde auch eine make-Datei erstellt, welche dafür sorgt, dass der Code kompiliert und zu einer statischen Bibliothek gepackt wird. Diese kann dann in andere Programme eingebunden werden.

Im Folgenden wird zunächst die API der Steuerungsbibliothek vorgestellt, wobei jede einzelne Funktion kurz erläutert wird. Danach wird auf die eingesetzten Methoden zur Fehlerbehandlung, einige Hilfsfunktionen, die zur Datenübertragung über die I^2C Schnittstelle implementiert wurden und eine

kleine Hilfsfunktion, welche vom Nutzer optional zur Verbesserung des Laufzeitverhaltens aufgerufen werden kann, eingegangen. Anschließend werden die internen Abläufe beim Aufruf der verschiedenen Funktionen vorgestellt. An entscheidenden Stellen werden zur Verdeutlichung einige Ausschnitte des Quellcodes präsentiert. Der vollständige Quellcode findet sich auf der beiliegenden CD. Die komplette Steuerungsbibliothek wurde unter eine BSD Lizenz gestellt um die Nutzung möglichst wenig einzuschränken.

4.8.1. API

Das öffentliche Interface der entwickelten Steuerungsbibliothek wird in der Header-Datei rpiplc.h, welche primär zum Einbinden in Software, welche die Bibliothek nutzt, gedacht ist, definiert. Dort sind alle Funktionen und ihre Parameter auch so weit dokumentiert, wie dies zur Nutzung dieser notwendig ist. Codeausschnitt 11 zeigt eine Zusammenstellung der Funktionen, wie diese anderen Programmen zur Verfügung stehen.

```
int rpiplc_open();
int rpiplc_close();
int rpiplc_digital_write_port(const int port, const unsigned char output);
int rpiplc_digital_write_pin(const int port, const int pin, const int output);
int rpiplc_digital_read_port(const int port, unsigned char * const input);
int rpiplc_get_digital_filter(const int port, unsigned int * const usec);
int rpiplc_get_status_flags(unsigned int * const status);
int rpiplc_set_status_LEDs(const unsigned int pattern);
const char *rpiplc_format_error_message();
int rpiplc_setup_rt();
int rpiplc_wait_for_change(
  unsigned char * const port1,
  unsigned char * const port2,
  unsigned int * const status,
  const int timeout);
```

Codeausschnitt 11: API der Steuerungsbibliothek

Im Folgenden werden die einzelnen Funktionen kurz beschrieben und es wird ggf. auf eine spätere detailliertere Erklärung der internen Vorgänge beim Aufruf dieser Funktion verwiesen. Wie an den Signaturen der Funktionen zu erkennen ist, liefern alle Funktionen mit einer einzigen Ausnahme immer einen integer Wert zurück. Dieser wird konsequent zur Fehlerbehandlung eingesetzt. An dem zurückgelieferten Wert kann die aufrufende Applikation feststellen, ob der Aufruf erfolgreich ausgeführt werden konnte. Eine genauere Beschreibung der Methoden welche zur Fehlerbehandlung eingesetzt werden findet sich im Abschnitt 4.8.2.

rpiplc_open / rpiplc_close

Die Funktion rpiplc_open initialisiert sowohl die Steuerungsbibliothek selbst als auch die Hardware der SPS. Nur rpiplc_open, rpiplc_setup_rt und rpiplc_format_error_message können genutzt werden ohne vorher rpiplc_open aufzurufen. Alle anderen Funktionen setzten einen vorangegangenen erfolgreichen Aufruf von rpiplc_open voraus, ist dieser nicht erfolgt, liefern alle anderen Funktionen einen entsprechenden Fehlercode zurück.

Die Funktion rpiplc_close deinitialisiert sowohl die Bibliothek als auch die Hardware. Nach einem Aufruf von close können abgesehen von einem erneuten Aufruf von rpiplc_open wieder nur die Funktionen rpiplc_setup_rt und rpiplc_format_error_message genutzt werden.

Eine genauere Beschreibung der Vorgänge, welche intern beim Aufruf der rpiplc_open und rpiplc_close Funktionen durchgeführt werden, findet sich im Abschnitt 4.8.5.

rpiplc_digital_write_port / rpiplc_digital_write_pin

Diese Funktionen dienen dazu die Werte der 16 digitalen Ausgänge der SPS zu setzen. Mit der Funktion rpiplc_digital_write_pin wird ein einzelner digitaler Ausgang auf den gewünschten Wert gesetzt, mit der Funktion rpiplc_digital_write_port werden alle acht digitalen Ausgänge eines Ports auf die jeweils gewünschten Werte gesetzt.

Da die digitalen Ausgaben direkt über GPIO Pins des Raspberry Pi gesteuert werden und diese einzeln gesetzt werden, entspricht ein Aufruf von rpiplc_digital_write_port im Wesentlichen acht aufeinanderfolgenden Aufrufen von rpiplc_digital_write_pin. Da die gesamte Steuerungsbibliothek thread-safe implementiert wurde, sind an einigen Stellen Synchronisationsmaßnahmen erforderlich. Diese verursachen natürlich einen gewissen Overhead. Die Funktion rpiplc_digital_write_port wird bereitgestellt, da es ein üblicher Anwendungsfall sein dürfte alle 8 Ausgaben eines Ports unmittelbar hintereinander zu aktualisieren und durch Nutzung dieser Funktion Overhead im Vergleich zu acht aufeinanderfolgenden Aufrufen der Funktion rpiplc_digital_write_pin gespart werden kann. Insbesondere muss nur einmal gelockt werden und es muss nur einmal überprüft werden, ob die Steuerungsbibliothek initialisiert ist. Eine genauere Beschreibung der internen Vorgänge beim Aufruf dieser Funktionen findet sich im Abschnitt 4.8.6.

rpiplc_digital_read_port

Die Funktion rpiplc_digital_read_port liefert die aktuellen Werte aller acht digitalen Eingänge eines Ports. Im Vergleich zum ersten Entwurf für eine API (vgl. Codeausschnitt 3) wurde keine zusätzliche Funktion zum Lesen eines einzelnen Eingangs implementiert. Die Eingänge werden über die I^2C Schnittstelle von einem IO-Expander gelesen. Dieser liefert immer die Werte für alle acht Eingänge eines Ports. Wäre eine Funktion implementiert, welche den Wert von nur einem Eingang liest, müsste diese intern dennoch alle acht Eingänge lesen und 7 von 8 Werten verwerfen. Dies wäre nicht schlimm, wenn tatsächlich nur ein Wert von Interesse sein sollte. Allerdings könnte ein Benutzer auf die Idee kommen mehrere Eingänge mit aufeinanderfolgenden Aufrufen einer solchen Funktion auszulesen, anstatt alle acht zu lesen und die nicht benötigten Werte zu verwerfen. Im Gegensatz zur rpiplc_digital_write_pin Funktion, bei deren wiederholtem Aufrufen nur ein geringer Overhead im Vergleich zur rpiplc_digital_write_port Funktion entsteht, würde beim wiederholten Aufruf einer Lesefunktion ein erheblicher Overhead entstehen, da über den I^2C Bus tatsächlich unnötige Werte gelesen werden würden. Dieser Tatsache könnte sich ein Benutzter der den Schaltplan nicht eingesehen hat, nicht bewusst sein. Um dieses Problem von vornherein auszuschließen, wird keine Funktion zum Lesen eines einzelnen Eingangs angeboten. Genaueres über die internen Abläufe bei einer Leseoperation findet sich im Abschnitt 4.8.7.

Natürlich ist es prinzipiell möglich eine Funktion zum Lesen eines einzelnen Eingangs effizient zu implementieren indem intern jeweils nur beim ersten Aufruf eine echte Leseoperation durchgeführt wird und die ermittelten Werte für alle Eingänge eines Ports zwischengespeichert werden. Die gepufferten Werte können dann bei nachfolgenden Aufrufen sehr effizient aus dem Puffer geliefert werden ohne eine tatsächliche Leseoperation durchführen zu müssen. Um nicht beliebig alte Werte auszuliefern, müsste natürlich eine Obergrenze für die Zeit für die Werte zwischengespeichert werden festgelegt werden. Danach müsste bei einem erneuten Aufruf erneut eine echte Leseoperation erfolgen.

Alternativ könnte ein eigener Thread implementiert werden, der entweder periodisch oder als Reaktion auf das Auftreten eines Interrupts die Eingaben einliest und in einem Puffer ablegt, aus dem diese dann bei Leseoperationen ausgeliefert werden können. Dies wäre effizient, würde aber eine hohe Abstraktion und für den Benutzer einen erheblichen Verlust an Kontrolle über das Laufzeitverhalten bedeuten, speziell da intern ein ganzer zusätzlicher Thread laufen würde. Eine solche Funktionalität zu bieten ist nicht Sinn dieser Steuerungsbibliothek. Diese Bibliothek dient nur dazu einen angemessen

einfachen Zugriff auf die Hardware zu bieten. Alle anderen Funktionen, speziell eine Pufferung von Werten im Arbeitsspeicher, können bei Bedarf auf einer höheren Abstraktionsschicht (vgl. Abschnitt 4.9) in der Anwendungssoftware implementiert werden. Diese Bibliothek muss nur die nötigen Funktionen bereitstellen um eine Implementierung auf höheren Abstraktionsschichten zu ermöglichen.

rpiplc_get_digital_filter / rpiplc_get_status_flags

Die Funktion rpiplc_get_digital_filter dient dazu die in Hardware vorgenommene Einstellung, konkret die Filterzeit, der Eingabefilter einzulesen. Die Funktion rpiplc_get_status_flags dient dazu den Status der Hardware abzufragen. Mit Hilfe dieser Funktion können eine Überlastung der Ausgänge, eine Fehlfunktion der Eingänge, eine Überhitzung, sowie das Fehler einer Versorgungsspannung abgefragt werden. Beide Funktionen arbeiten intern analog zur Funktion, die zum Einlesen der Eingänge genutzt wird, da diese Informationen ebenfalls über den I^2C Bus von den IO-Expandern bezogen werden. Daher finden sich weitere Informationen über ihre Funktionsweise ebenfalls im Abschnitt 4.8.7. Auch hier ist es wichtig, dass nicht einzelne Statusflags abgefragt werden, sondern immer der gesamte Status, da, wie bei den Eingängen, über den I^2C Bus immer der gesamte Status auf einmal gelesen werden muss, da alle zugehörigen Leitungen an einem Port des IO-Expanders angeschlossen sind. Eine Möglichkeit zum Lesen einzelner Statusflags ist bei Bedarf ebenfalls auf einer höheren Abstraktionsschicht (vgl. Abschnitt 4.9) zu implementieren.

rpiplc_set_status_LEDs

Diese Funktion dient dazu die drei frei programmierbaren zweifarbigen Status-LEDs zu steuern. Da die LEDs an einem IO-Expander angeschlossen sind, müssen zum Schalten Daten über den I^2C Bus übertragen werden. Da alle LEDs an einem Port des IO-Expanders angeschlossen sind, können alle LEDs mit nur einer Übertragung geschaltet werden. Umgekehrt bedeutet dies aber auch, dass stets der Zustand aller LEDs übertragen werden muss, auch wenn nur eine LED geschaltet werden soll. Um dieser Tatsache Rechnung zu tragen, wurde nur diese Funktion zum Schalten aller LEDs implementiert und keine Funktion zum Schalten einer einzelnen LED. Sollte dies gewünscht sein, muss dies ebenfalls auf einer höheren Abstraktionsschicht erfolgen. Im Unterschied zum Lesen von Eingängen und des Status sollte hier die Performance aber kein Problem sein, da nicht davon auszugehen ist, dass Status-LEDs oft geschaltet werden. Selbst wenn ein Zustand mit einer blinkenden LED symbolisiert werden sollte, sind allerhöchstens einige Schaltvorgänge pro Sekunde zu erwarten. Details zur Implementierung der rpiplc_set_status_LEDs Funktion finden sich im Abschnitt 4.8.8.

rpiplc_format_error_message

Die rpiplc_format_error_message Funktion ist für den Benutzer die zentrale Funktion im Rahmen der Fehlerbehandlung. Sie liefert stets eine menschenlesbare Fehlermeldung, welche Informationen zum letzten aufgetretenen Fehler enthält. Jede andere Funktion der Steuerungsbibliothek liefert einen Rückgebewert, welcher angibt, ob die Funktion erfolgreich ausgeführt werden konnte. Besagt dieser Wert, dass ein Fehler aufgetreten ist, kann die Anwendungssoftware, sofern sie den Fehler nicht selber behandeln kann oder will, unmittelbar im Anschluss die rpiplc_format_error_message Funktion aufrufen um eine passende Fehlermeldung zu erhalten, welche sie dem Benutzer anzeigen kann. Die Steuerungsbibliothek führt eine sehr ausgiebige Fehlerbehandlung durch. Eine ausführliche Erklärung findet sich im Abschnitt 4.8.2.

rpiplc_setup_rt

Die rpiplc_setup_rt Funktion hat nicht direkt etwas mit der entwickelten SPS zu tun. Diese nimmt einige einfache Einstellungen vor um das zeitliche Verhalten des aufrufenden Threads ein wenig zu verbessern, insbesondere um seine Priorität gegenüber anderen Threads im System zu erhöhen. Wel-

che Einstellungen genau vorgenommen werden, wird im Abschnitt 4.8.4 beschrieben. Diese Funktion kann von einer Anwendungssoftware genutzt werden, hängt aber in keinster Weise mit der restlichen Steuerungsbibliothek zusammen, so dass ihre Verwendung vollkommen optional ist.

Diese Funktion ist speziell für einfache winzige Testprogramme mit nur einem Thread gedacht. Es werden im Folgenden zum Testen und zur Untersuchung des zeitlichen Verhaltens der entwickelten SPS eine ganze Reihe von winzigen Testprogrammen nötig sein. Es wäre nicht sinnvoll diese Einstellungen in jedem Programm erneut zu implementieren. Daher ist eine Auslagerung dieser Funktionalität erforderlich. Die Platzierung dieser Funktion in der Steuerungsbibliothek ist im Moment einfach am praktischsten. Eine Nutzung dieser Funktion durch komplexere Steuerungssoftware ist nicht vorgesehen.

rpiplc_wait_for_change

Die bisher beschriebenen Funktionen zum Lesen der digitalen Eingänge und der Statusflags erlauben keine Reaktion auf Interrupts und erfordern somit ein Polling. Um auf Änderungen reagieren zu können ohne die Eingaben periodisch abfragen zu müssen, wurde die Funktion rpiplc_wait_for_change implementiert. Diese blockiert so lange bis sich eine Eingabe ändert oder die maximale angegebene Wartezeit abläuft und liefert dann den aktuellen Wert aller Eingänge.

Entgegen dem ersten Entwurf einer API wurde keine Funktion zum Warten auf einzelne Eingänge implementiert, da Eingaben Port-weise gelesen und Interrupts Port-weise erzeugt werden. Dies kann aber durchaus auf einer höheren Abstraktionsschicht erfolgen. Die Funktion zum Warten auf eine Änderung an einem Port wurde so erweitert, dass auf beide Ports und gleichzeitig auch auf eine Statusänderung gewartet werden kann. Würde die Funktion nur das Warten auf eine Quelle von Änderungen, also einen einzelnen Port oder den Status, erlauben, wären allein zum Warten auf eine beliebige Änderung drei Threads, also ein Thread pro Quelle, erforderlich.

Die implementierte Funktion macht es möglich mit nur einem Thread auf eine beliebige Änderung zu warten. Sollte es allerdings gewünscht sein mehrere Threads einzusetzen, beispielsweise für jeden Port einen eigenen Thread, ist auch dies möglich. Die Funktion wurde so konstruiert, dass jeder Thread auf jede beliebige Teilmenge an Quellen warten kann. Unterschiedliche Threads können auf überschneidungsfreie Teilmengen an Quellen warten. Es können also beispielsweise ein Thread auf eine Änderung am Port 1 und ein anderer auf eine Änderung am Port 2 oder am Status warten.

Die Steuerungsbibliothek wurde so konstruiert, dass mehrere Threads in einer komplexen Steuerungssoftware konkurrierend die Ausgänge und die Status-LEDs schalten und die Eingänge und den Status lesen können. Dies dürfte in den meisten Fällen nicht notwendig und auch nicht sinnvoll sein, ist aber prinzipiell mit der Steuerungsbibliothek möglich. Geht man nun also von einer Anwendung mit mehreren Threads aus, die konkurrierend die digitalen Eingänge lesen, ist festzuhalten, dass es nicht unbedingt möglich ist, festzustellen, welcher Thread welchen Zustand der Eingänge zuletzt gesehen hat. Es wäre zwar durchaus möglich festzustellen, welcher Wert an welchen Thread als letztes ausgeliefert wurde, allerdings sind die Datenpfade zwischen den Threads in einer Anwendungssoftware aus Sicht der Steuerungsbibliothek unbekannt.

Ruft ein Thread nun die rpiplc_wait_for_change auf um zu warten, bis sich eine Eingabe ändert, kann durch die Steuerungsbibliothek nicht festgestellt werden, welchen Wert der Eingänge der Thread zuletzt gesehen hat. Es kann nicht festgestellt werden, ob der aktuelle Zustand der Eingänge für diesen Thread bereits eine Änderung darstellt oder nicht. Daher ist es erforderlich, dass beim Aufruf der Funktion der aufrufende Thread den für ihn aktuellen Zustand der Eingaben übergibt. Weicht der Zustand der Eingaben sofort oder an einem späteren Zeitpunkt von dem übergebenen Muster ab, ist dies für den aufrufenden Thread eine Änderung, welche zurückgeliefert werden muss.

Die Implementierung der Funktion rpiplc_wait_for_change stellt mit Abstand den komplexesten Teil der Steuerungsbibliothek dar. Die interne Arbeitsweise dieser Funktion wird im Abschnitt 4.8.9 beschrieben.

4.8.2. Fehlerbehandlung

Jede Funktion der Steuerungsbibliothek gibt konsequent einen integer Wert zurück. In rpiplc.h sind dafür einige Konstanten definiert. Konnte eine Funktion erfolgreich ausgeführt werden, liefert sie RPIPLC_ERROR_SUCCESS, was 0 entspricht, zurück. Ansonsten wird ein Fehlercode geliefert. Wurde beispielsweise ein ungültiger Parameter an die Funktion übergeben, liefert diese RPIPLC_ERROR_INVALID_ARG zurück, ist intern ein Synchronisationsfehler aufgetreten, wird RPIPLC_ERROR_INTERNAL_SYNC geliefert.

Um der Anwendungssoftware bei Bedarf eine aussagekräftige Fehlermeldung liefern zu können, die diese an den Nutzer weiterleiten kann, wird zusätzlich jeweils der letzte Fehler bzw. die letzten Fehler intern abgespeichert. Eine genauere Erklärung dazu findet sich im Unterabschnitt „Interner Fehlerspeicher“. Die Anwendung kann nachdem ein Fehler aufgetreten ist, eine entsprechende Fehlermeldung mit der Funktion rpiplc_format_error_message anfordern.

Es wäre zwar möglich beim Auftreten eines Fehlers gleich weitere Informationen zurückzuliefern, allerdings müsst die Anwendungssoftware dazu stets einen Puffer übergeben, in den die Meldung abgelegt werden kann oder es wäre notwendig einen Puffer anzulegen, welchen die Anwendungssoftware freigeben müsste. Ersteres ist unangemessen aufwendig, da davon auszugehen ist, dass Fehler nur in den seltensten Fällen auftreten und der Puffer damit die meiste Zeit ungenutzt bliebe. Die zweite Variante würde zu Speicherlecks führen, wenn die Anwendungssoftware vergessen sollte den Puffer freizugeben.

Es wäre auch möglich auf eine interne Speicherung des Fehlers zu verzichten und die Fehlermeldung allein aufgrund des Fehlercodes zu generieren. Dies würde aber zu deutlich allgemeineren Fehlermeldungen führen. Werden intern Information gespeichert, können beispielsweise bei Dateizugriffsproblemen die Dateinamen der betroffenen Dateien in der Fehlermeldung mit aufgeführt werden.

Fehlerprioritäten

Es fällt auf, dass Fehler verschieden schwerwiegende Konsequenzen haben können. Wird an eine Funktion ein ungültiger Parameter übergeben, wird dies sofort erkannt und es wird keine weitere Operation durchgeführt. Eine Anwendungssoftware kann durchaus die Möglichkeit haben diesen Fehler selber zu korrigieren und die Funktion erneut aufzurufen. Stammte der Parameter beispielsweise aus einer Nutzereingabe, könnte der Nutzer aufgefordert werden, die Eingabe zu wiederholen. In jedem Fall sind keine weiteren Konsequenzen aus diesem Fehler zu befürchten.

Tritt hingegen intern ein Fehler beim Synchronisieren auf, kann die Anwendungssoftware selbst nichts weiter unternehmen und es können sich durchaus weitreichende Konsequenzen aus diesem Fehler ergeben. Die genauen Konsequenzen hängen davon ab, wo genau der Fehler aufgetreten ist. Konnte ein Mutex beispielsweise nicht gelockt werden, wird es mit hoher Wahrscheinlichkeit auch bei nachfolgenden Aufrufen nicht gelingen und es wird einfach immer wieder der gleiche Fehler geliefert. Konnte ein Mutex hingegen nicht freigegeben werden, könnte ein weiterer Aufruf einer Funktion zu einem Dead-Lock führen.

Um der Anwendungssoftware die Entscheidung über die weitere Vorgehensweise zu erleichtern, wurden in der Steuerungsbibliothek Fehlerprioritäten implementiert. Für jede Priorität wurde eine Konstante definiert, welche mit dem normalen Fehlercode verodert wird, so dass der zurückgelieferte Wert sowohl die Priorität als auch den normalen Fehlercode enthält. Für die niedrigste Priorität steht die

Konstante RPIPLC_ERROR_PRIORITY_NORMAL, welche 0 entspricht und somit den normalen Fehlercode nicht ändert. Fehler, die mit dieser Priorität versehen sind, kann eine Anwendungssoftware entweder selbst behandel oder an den Nutzer melden, in jedem Fall sind keine weiteren Konsequenzen zu befürchten.

Fehler mit einer höheren Priorität dürfen und können von einer Anwendungssoftware nicht behandelt werden, diese müssen an den Nutzer gemeldet werden und die Software muss sich beenden, da sonst schwerwiegende Konsequenzen, wie Speicherlecks oder sogar Dead-Locks, zu befürchten sind. Es wurden drei Fehlerprioritäten über der normalen Priorität definiert. Jede gibt an, welche Folgen die so markierten Fehler haben können und welche Aktionen die Anwendungssoftware noch ausführen kann oder sollte bevor sie sich beendet. Die einzelnen Prioritäten werden in der rpiplc.h-Datei genauer erklärt.

Interner Fehlerspeicher

Jeder Fehler wird intern mit einer _RPIPLC_ERROR Struktur repräsentiert, wie sie im Codeausschnitt 12 gezeigt ist. In dieser Struktur werden der Fehlercode, welcher von der Funktion auch zurückgeliefert wird, einige Textteile, welche später in die Fehlermeldung aufgenommen werden sollen, ein Verweis auf die Stelle, wo der Fehler aufgetreten ist, sowie die Fehlernummer, welcher von der C-Bibliothek geliefert wurde, sofern der Fehler in einer C-Funktion aufgetreten ist, aufgenommen.

Der Verweis auf die Stelle, wo der Fehler aufgetreten ist, besteht aus dem Namen der Quellcodedatei und der Zeilennummer. Dieser ist nicht für den Nutzer gedacht, sondern für einen Entwickler zur späteren Fehlerdiagnose, falls der Fehler aufgrund eines Programmierfehlers entstanden ist. Tritt ein Fehler beim Aufruf einer Bibliotheksfunktion der C-Bibliothek auf, weil beispielsweise der Zugriff auf eine Datei vom System verweigert wurde, wird der entsprechende Fehlercode ebenfalls aufgenommen um später die zugehörige Fehlermeldung der C-Bibliothek in die generierte Fehlermeldung integrieren zu könne.

```
struct _RPIPLC_ERROR{
    int code;
    const char *message[3];
    char dynamic_message[MAX_SIZE_OF_DYNAMIC_MESSAGE];
    const char *file;
    unsigned int line;
    int clib_errno;
};
```

Codeausschnitt 12: Struktur zum Speichern eines einzelnen Fehlers

Tritt beim Aufruf einer Funktion der Steuerungsbibliothek ein Fehler auf, wird die weitere Verarbeitung der Funktion grundsätzlich eingestellt und der Fehler wird an den Aufrufer gemeldet. Aufgrund der sofortigen Einstellung der Verarbeitung, tritt pro Funktionsaufruf in der Regel nur ein Fehler auf. Allerdings wird beim Auftreten eines Fehlers stets versucht, alle Operationen die bis zum Auftreten des Fehlers von der Funktion bereits durchgeführt worden sind, rückgängig zu machen um einen konsistenten Zustand wiederherzustellen. Dabei können theoretisch weitere Fehler auftreten. Die interessante Frage ist, welcher der aufgetretenen Fehler an die Anwendungssoftware und damit auch den Nutzer gemeldet werden soll.

In der Tat hängt es davon ab, wer die Fehlermeldung liest. Für einen Anwender ist in aller Regel der erste Fehler, der aufgetreten ist, am interessantesten. Wäre dieser nicht aufgetreten, wären höchstwahrscheinlich auch die anderen Fehler nicht aufgetreten. Gelingt es dem Anwender also den ersten Fehler zu beheben, sollte die Anwendung wieder funktionieren.

Für die Anwendungssoftware selbst ist nicht unbedingt der erste, sondern der schwerwiegendste Fehler wichtig, da aufgrund dieses Fehlers entschieden werden muss, ob eine weitere Ausführung der Software möglich ist oder ein Abbruch erfolgen muss.

Für den Entwickler der Steuerungsbibliothek sind eigentlich alle Fehler wichtig. Der wichtigste ist aber der letzte Fehler der aufgetreten ist, da dieser als erstes behoben werden muss, falls es sich um einen Programmierfehler handelt. Würde man einen vorhergehenden Fehler zuerst beheben, würde der letzte Fehler womöglich nicht mehr auftreten und man hätte keine Möglichkeit ihn zu untersuchen. Sofern nicht alle Fehler gespeichert werden können, sind neben dem letzten Fehler für einen Entwickler vor allem die letzten paar Fehler wichtig um nachvollziehen zu können, auf welchem Wege es zum letzten Fehler gekommen ist.

```
struct _RPIPLC_ERROR_LOG{
    struct _RPIPLC_ERROR root_cause;
    struct _RPIPLC_ERROR last_errors[MAX_ERRORS_IN_LOG];
    int index;
    int count;
    struct _RPIPLC_ERROR serious_error;
    struct _RPIPLC_ERROR *most_serious_error;
};
```

Codeausschnitt 13: Struktur zum Speichern mehrerer Fehler

Um den unterschiedlichen Anforderungen gerecht zu werden, wird beim Auftreten von mehreren Fehlern innerhalb eines Funktionsaufrufs nicht nur ein einzelner Fehler, sondern eine ganze Reihe von Fehlern gespeichert. Wird von der Anwendungssoftware später eine Fehlermeldung angefordert, werden alle gespeicherten Fehler in diese aufgenommen.

Die Speicherung erfolgt in der Struktur, welche im Codeausschnitt 13 gezeigt wird. Diese enthält den ersten Fehler, der aufgetreten ist, die MAX_ERRORS_IN_LOG letzen Fehler die aufgetreten sind, sowie einen Zeiger auf den schwerwiegendsten Fehler, der aufgetreten ist. Sollte der schwerwiegendste Fehler weder der erste noch einer der letzten sein, wird dieser separat im Feld serious_error gespeichert und der Zeiger auf dieses Feld gelegt. Diese Fehlerstruktur wird für jeden Thread separat gespeichert, so dass in einem Thread aufgetretene Fehler nicht verloren gehen, wenn in einem anderen Thread Fehler auftreten.

Zum Arbeiten mit dieser Fehlerstruktur wurden einige Hilfsfunktionen in der Datei error.c implementiert. Jede Funktion der Steuerungsbibliothek, welche von außen aufgerufen werden kann, ruft typischerweise als erstes die Hilfsfunktion _rpiplc_clear_error_log auf. Diese löscht alle gespeicherten Fehler in der Fehlerstruktur des aufrufenden Threads. Da Fehler im Allgemeinen nur sehr selten auftreten, wurden Regeln zur Lazy-Iinitialization der Fehlerstruktur festgelegt. Es wurde definiert, dass die Struktur als leer zu betrachten ist, wenn der Fehlercode des ersten Fehlers RPIPLC_ERROR_SUCCESS entspricht. Daher muss die _rpiplc_clear_error_log Funktion nur diesen einen Wert setzen um die Fehlerstruktur zu löschen. Das spart Zeit, da in den meisten Fällen kein Fehler auftreten wird und die Struktur daher nicht benötigt werden wird. Erst wenn ein Fehler tatsächlich auftritt wird eine Initialisierung der Struktur durchgeführt. Auch in diesem Fall wird aber zunächst nur der erste Fehler eingetragen und das Feld count wird auf 0 gesetzt um anzuzeigen, dass der Rest der Struktur immer noch nicht initialisiert ist. Dies ist sinnvoll, da in der Regel nur ein Fehler pro Funktionsaufruf auftreten sollte. Erst wenn tatsächlich mehrere Fehler auftreten, wird die Fehlerstruktur vollständig initialisiert.

Nach der Löschung des Fehlerspeichers führt die von der Anwendungssoftware aufgerufene Funktion die gewünschten Aktionen aus, wobei ggf. auftretenden Fehler mit Hilfe der entsprechenden Hilfsfunktionen in der Fehlerstruktur vermerkt werden. Zum Schluss ruft eine extern aufgerufene Funktion

typischerweise die Hilfsfunktion _rpiplc_get_most_serious_error auf, welche den Fehlercode des schwerwiegendsten Fehlers, welcher im Fehlerspeicher vermerkt ist, zurückliefert Dieser wird dann an die Anwendungssoftware weitergeleitet. Diese kann dann ggf. über rpiplc_format_error_message eine Fehlermeldung anfordern, welche basierend auf dem Inhalt des Fehlerspeichers zusammengestellt wird.

4.8.3. Datenübertragung über I^2C

Die eingesetzten IO-Expander werden durch Schreiben und Lesen von Registern gesteuert. Sie besitzen 22 zugängliche Register, von denen jedes genau ein Byte bestehend aus 8 Bit enthält. Die Chips haben einen internen Zeiger der stets auf eines dieser Register zeigt. Daten werden immer in das Register geschrieben bzw. aus dem Register gelesen, auf das dieser Zeiger verweist.

Bei jeder Schreiboperation wird dieser interne Zeiger explizit gesetzt indem nach der Adresse des Chips die Adresse des Registers übertragen wird, bevor die eigentlichen Nutzdaten übertragen werden. Um ein beliebiges Register zu lesen, kann zunächst eine Schreiboperation ohne Daten durchgeführt werden, mit der der Zeiger auf das gewünschte Register gesetzt wird. Werden nach der Registeradresse keine Nutzdaten übertragen, hat diese Schreiboperation keine weiteren Auswirkungen. Danach können Daten aus dem voreingestellte Register gelesen werden. Bei dem eigentlichen Lesevorgang wird nur noch die Chipadresse und nicht mehr die Registeradresse übertragen.

Will man immer dasselbe Register lesen, kann nach der ersten Leseoperation für alle nachfolgenden die Schreiboperation zum Setzen der Registeradresse entfallen, wodurch Zeit eingespart werden kann. Außerdem kann zwischen dem Setzen der Registeradresse und der eigentlichen Leseoperation beliebig viel Zeit vergehen, sofern zwischendurch keine weiteren Schreiboperationen notwendig sind. Diese Eigenschaft wird von der entwickelten Steuerungsbibliothek ausgenutzt um die benötigte Übertragungszeit zum Setzen der Adresse aus einem zeitlich kritischen Bereich in einen unkritischen Zeitraum zu verlagern (vgl. Abschnitt 4.8.9).

Die Chips können so konfiguriert werden, dass sich der Zeiger nach jedem gelesenen oder geschriebenen Byte nach bestimmten Kriterien automatisch zu einem nachfolgenden Register bewegt. Dies ermöglicht es mehrere hintereinanderliegende Register effizient in einem Vorgang zu lesen oder zu schreiben. Diese Funktion wird aber im vorliegenden Fall nicht benötigt, da immer nur einzelne Register gelesen werden. Daher werden die IO-Expander so konfiguriert, dass sich der Adresszeiger nicht automatisch ändert. Es kann im Folgenden also davon ausgegangen werden, dass der Wert des internen Adresszeigers dem Registerwert der letzten Schreiboperation entspricht. [81]

Wie beschrieben, kann die gewünschte Registeradresse lange vor der eigentlichen Leseoperation eingestellt werden und so an kritischen Stellen Zeit eingespart werden. Allerdings funktioniert dies nur, wenn dazwischen kein anderes Register gelesen wird und auch keine Schreiboperation durchgeführt wird. Um nicht an jeder Stelle der Steuerungsbibliothek, an der auf die IO-Expander zugegriffen wird, nachverfolgen zu müssen, welche Registeradresse gerade gesetzt ist um entscheiden zu können, ob ein erneutes Setzen notwendig ist, wurde diese Funktionalität in die Datei i2c.c ausgelagert. In dieser wurden einige Hilfsfunktionen zum Lesen und Schreiben über die I^2C Schnittstelle implementiert. Diese speichern intern für jeden Chips welche Registeradresse als letztes zugegriffen wurde. So kann transparent entschieden werden, ob ein erneutes Setzten der Registeradresse notwendig ist.

Gleichzeitig konnten einige Synchronisationsaufgaben in diese Hilfsfunktionen ausgelagert werden. Da sichergestellt werden muss, dass stets nur ein Thread auf den I^2C Bus zugreift, war hier der Einsatz eines Mutex ohnehin erforderlich. Dieser konnte dann noch für einige weitere Synchronisationsaufgaben verwendet werden, indem für jeden Chip gespeichert wird, ob er schon initialisiert ist oder nicht. Während der Initialisierung der Bibliothek und der Hardware (vgl. Abschnitt 4.8.5) werden nach der

Konfiguration der IO-Expander diese als initialisiert markiert. An diesem Punkt sind sie vollständig konfiguriert und können eingesetzt werden.

Sollen beispielsweise die Eingaben eines Ports gelesen werden, muss von der rpiplc_digital_read_port Funktion nicht geprüft werden, ob die Steuerungsbibliothek überhaupt initialisiert ist. Der Lesevorgang kann direkt eingeleitet werden. In der Hilfsfunktion zum Lesen über den I^2C Bus wird dann automatisch geprüft, ob der betroffene Chip als initialisiert markiert ist. Ist dies der Fall, kann der Lesevorgang durchgeführt werden. Ist der Chip nicht initialisiert, kann an dieser Stelle eine entsprechende Fehlermeldung erzeugt werden. Dadurch kann auf das Sperren des Mutex, welcher normalerweise benötigt wird um zu prüfen, ob die Bibliothek initialisiert ist, an dieser Stelle verzichtet werden und so Zeit gespart werden.

4.8.4. Verbesserung des zeitlichen Verhaltens

Die Funktion rpiplc_setup_rt, welche in rt1.c implementiert ist, ist innerhalb der Bibliothek, wie zuvor erklärt, nicht wichtig, da sie mit der entwickelten SPS nicht direkt zusammenhängt. Allerdings ist die Betrachtung des zeitlichen Verhaltens ein wichtiger Aspekt in dieser Arbeit und da die Funktion gerade der Modifizierung dieses Verhaltens dient und bei den geplanten Untersuchungen des Verhaltens zum Einsatz kommen soll, verdient auch diese Funktion an dieser Stelle eine genauere Betrachtung.

Wie schon im Abschnitt 2.6 beschrieben, sind unter anderem das Scheduling und die Speicherverwaltung von entscheidender Bedeutung für das zeitliche Verhalten einer Anwendung. Beides wird von der Funktion rpiplc_setup_rt für den aufrufenden Thread bzw. die gesamte Anwendung mit dem Ziel modifiziert die Zeit, welche zur Reaktion auf Ereignisse benötigt wird, zu verkürzen. Eine Verbesserung der Reaktionszeit erfolgt natürlich zu Lasten anderer Threads und des Gesamtdurchsatzes des Systems.

Scheduling

Unter Linux sind jedem Thread ein Schedulingverfahren und eine statische Priorität zugewiesen. Für jede mögliche statische Priorität verwaltet der Kernel eine Liste mit laufbereiten Threads mit dieser Priorität. Laufen darf immer der Thread, welcher sich am Anfang der Liste mit der höchsten Priorität befindet, wobei leere Listen natürlich ignoriert werden. Diese Regel gilt zu jeder Zeit und bedeutet insbesondere auch, dass ein Thread mit niedrigerer Priorität umgehend unterbrochen wird und in die zugehörige Warteliste zurückkehren muss, wenn ein Thread mit höherer Priorität laufbereit wird. Das Schedulingverfahren eines Threads bestimmt, wo ein Thread innerhalb der zugehörigen Liste eingefügt wird und wie er innerhalb dieser Liste verschoben wird.

Normalen Threads, bei denen es keine speziellen Ansprüche bezüglich des zeitlichen Verhaltens gibt, ist standardmäßig das Schedulingverfahren SCHED_OTHER zugewiesen. Threads mit diesem Schedulingverfahren dürfen nur die statische Priorität 0 haben. Damit sind die meisten Threads in der Warteliste mit der niedrigsten Priorität enthalten. Innerhalb dieser Liste wird für jeden Thread zusätzlich periodisch eine dynamische Priorität bestimmt, welche die Position des Threads innerhalb der Liste bestimmt. Die Sortierung der Threads innerhalb dieser Liste ist für die vorliegende Anwendung aber nicht weiter von Interesse.

Threads, bei denen besondere Ansprüche an das zeitliche Verhalten gestellt werden, können die Schedulingverfahren SCHED_IDLE, SCHED_BATCH, SCHED_FF, SCHED_RR oder SCHED_DEADLINE zugewiesen werden. SCHED_IDLE und SCHED_BATCH sind für den vorliegenden Anwendungsfall nicht interessant, da diese für niedrig priorisierte Aufgaben bestimmt sind und zu einer Benachteiligung des Threads gegenüber anderen Threads führen. SCHED_DEADLINE ist für Threads gedacht, die periodisch aufgeweckt werden, eine bestimmte zeitkritische Aufgabe erfüllen und wieder schlafen gehen. Für jeden Thread kann dabei angegeben werden, wie lange dieser

maximal für seine Aufgabe benötigen kann (Berechnungszeit), nach welcher Zeit die Aufgabe spätestens erledigt sein muss, nachdem der Thread aufgeweckt wurde (relative Deadline) und wann der Thread frühestens erneut aufgeweckt wird, nachdem er aufgeweckt wurde (Periodendauer). Wird der Thread aufgeweckt, bestimmt der Scheduler aufgrund der relativen Deadline den Zeitpunkt zu dem die Berechnung abgeschlossen sein muss (absolute Deadline). Basierend auf der Berechnungszeit wählt der Scheduler dann einen Zeitpunkt, an dem der Thread laufen darf, der sicherstellt, dass die Berechnung vor der absoluten Deadline abgeschlossen ist. Dieses Verfahren dürfte für komplexere Anwendungen mit mehreren Threads interessant sein, wenn mehrere verschiedene zeitliche Bedingungen für die einzelnen Threads zu erfüllen sind. Bei der Funktion rpiplc_setup_rt wird aber von einer einfachen Testanwendung mit nur einem Thread oder zumindest einer Anwendung, bei der nur ein Thread, nämlich der, welcher die Funktion aufruft, zeitkritische Aufgaben erfüllen muss, ausgegangen. Es gibt effektiv auch keine relative Deadline, die man an dieser Stelle angeben könnte und auch keine Periodendauer. Es geht einfach darum, dass der Thread, nachdem er aufgeweckt wurde, möglichst schnell Rechenzeit zugeteilt bekommt. Damit ist im vorliegenden Fall das SCHED_DEADLINE Verfahren ungeeignet.

Gut geeignet sind hingegen die beiden Verfahren SCHED_FIFO und SCHED_RR. Diese sind sich sehr ähnlich. Bei beiden muss dem Thread eine statischen Priorität größer 0 zugewiesen werden. Da, wie zuvor beschrieben, immer der Thread mit der höchsten statischen Priorität laufen darf, ist damit automatisch sichergestellt, dass jeder Thread, welchem eines dieser Verfahren zugeordnet ist, gegenüber allen normalen Threads bevorzugt wird. Durch Setzen verschiedener Prioritäten können solche Threads untereinander priorisiert werden. Es läuft immer der Thread mit der höchsten Priorität bis dieser blockiert oder ein Thread mit noch höherer Priorität laufbereit wird. Gibt es mehrere Threads mit derselben Priorität hängt das Verhalten vom gewählten Schedulingverfahren ab. Wird SCHED_FIFO genutzt darf der erste Thread in der am höchsten priorisierten Warteschlage immer so lange laufen bis er blockiert. Andere Threads mit gleicher Priorität müssen warten. Wird SCHED_RR genutzt darf jeder Thread nur eine feste Zeit laufen, danach wird zum nächsten Thread in der am höchsten priorisierten Warteschlage gewechselt und der alte Thread muss sich in dieser wieder hinten anstellen.

Da im vorliegenden Fall nur von einem Thread ausgegangen wird, welcher gegenüber allen anderen bevorzugt werden soll, spielt es keine Rolle, welches der beiden Verfahren gewählt wird. Es wurde SCHED_FIFO gewählt, da dies im Zweifelsfall den Overhead reduziert. Sollten die Funktion doch für mehrere Threads aufgerufen werden, muss zwischen diesen dann nicht periodisch gewechselt werden. Als Priorität setzt die Funktion rpiplc_setup_rt eine sehr hohe, aber nicht die höchste Priorität um die Stabilität des Systems nicht zu gefährden und dem Nutzer die Möglichkeit zu geben einen Thread mit einer noch höheren Priorität zu erstellen, welcher beispielsweise genutzt werden könnte um den Testthread zu beenden, wenn dieser nicht ordnungsgemäß arbeitet. [94]

Speicherverwaltung

Bezüglich der Speicherverwaltung besteht bei einer zeitkritischen Anwendung das Hauptproblem darin, dass das Betriebssystem unter Umständen aktuell nicht benötigte Teile des Speichers auf einen Datenträger auslagert und den Speicher für neue Daten wiederverwendet. Werden die Daten später wieder zugegriffen, werden diese vom System wieder automatisch eingelagert. Auf diese Weise kann den Anwendungen virtuell mehr Speicher zur Verfügung gestellt werden, als physikalisch vorhanden ist, wodurch mehr Anwendungen gleichzeitig ausgeführt werden können und jede Anwendung an sich auch mehr Speicher nutzen kann.

Greift eine Anwendung auf einen Speicherbereich zu, welcher sich aktuell nicht im Arbeitsspeicher befindet, kommt es zu einem sog. Seitenfehler. Dieser wird vom System behandelt indem der betroffene Bereich eingelagert wird und der Zugriff wiederholt wird. Dies erfolgt für die Anwendung völlig

transparent, allerdings dauert die Einlagerung verhältnismäßig sehr lange. Ein Seitenfehler hat also verheerende Folgen für das zeitliche Verhalten der betroffenen Anwendung. Daher muss bei zeitkritischen Anwendungen sichergestellt werden, dass keine Seitenfehler auftreten. Der erste Schritt um dies zu erreichen besteht darin zu verhindern, dass der Arbeitsspeicher der Anwendung ausgelagert wird. Die Funktion rpiplc_setup_rt deaktiviert daher für den aktuell genutzten Speicherbereich und alle evtl. später hinzukommenden Speicherbereiche das Auslagern.

Um sicherzugehen, dass in zeitlich kritischen Bereichen tatsächlich keine Seitenfehler auftreten, können über dies noch weitere Maßnahmen erforderlich sein. Beispielsweise sollte eine größere Menge Daten auf den Stack geschrieben werden um sicherzugehen, dass bei Funktionsaufrufen im zeitlich kritischen Bereich keine neuen Speicherseiten für den Stack angefordert werden müssen. Solche Maßnahmen werden von der rpiplc_setup_rt Funktion aber nicht durchgeführt und werden an dieser Stelle daher auch nicht weiter beschrieben.

4.8.5. Initialisierung und Hardwarekonfiguration

Bei der Initialisierung der Steuerungsbibliothek werden als allererstes die genutzten GPIO Pins des Raspberry Pi konfiguriert. Zu deren Ansteuerung wird durchgehend das sysfs Interface, welches bereits im Abschnitt 4.7.5 beschrieben wird, verwendet. Dieses ist zwar langsamer als ein direkter Zugriff über den Arbeitsspeicher, hat aber den Vorteil, dass über dieses problemlos aus dem Nutzer-Modus auf das Auftreten von Interrupts gewartet werden kann. Dieser Vorteil ist natürlich nur bei GPIO Pins, welche als Eingängen genutzt werden, von Bedeutung. Für die Ausgänge wäre ein direkter Zugriff über den Arbeitsspeicher besser, allerdings haben vorbereitende Messungen ergeben, dass das Setzen eines Ausgangs über das sysfs-Interface im Durchschnitt nur wenige Mikrosekunden dauert, so dass die Nutzung zweier verschiedener Schnittstellen für die Ein- und für die Ausgänge vorerst nicht gerechtfertigt scheint. Um die GPIO Pins über das sysfs-Interface ansteuern zu können, werden diese während der Initialisierung zunächst exportiert, sofern dies nicht ohnehin schon der Fall ist. Danach wird für jeden Pin die Polarität, die Funktion (Ein- oder Ausgabe) und bei Eingängen die Flanke, bei der ein Interrupt ausgelöst werden soll, konfiguriert.

Vor der Initialisierungsphase befinden sich die beiden IO-Expander in aller Regel im Reset-Zustand. Selbst wenn dies nicht der Fall ist, werden sie spätestens während der Konfiguration der GPIO Pins in diesen versetzt. Unmittelbar nach dieser Konfiguration werden die IO-Expander aus dem Reset-Zustand geholt. Danach wird die I^2C Schnittstelle initialisiert und die IO-Expander werden über diese konfiguriert. Dabei wird eingestellt, dass der interne Adresszeiger der IO-Expander, welcher festlegt auf welches Register bei Schreib- und Leseoperationen zugegriffen wird, nicht automatisch bewegt werden soll. Ab diesem Zeitpunkt kann die Optimierung bei Lesevorgänge, welche im Abschnitt 4.8.3 beschrieben wird, genutzt werden.

Außerdem wird festgelegt, dass ein Interrupt durch Lesen des INTCAP Registers, welches jeweils den Zustand der GPIO Pins zu dem Zeitpunkt, an dem der Interrupt aufgetreten ist, enthält, zurückgesetzt werden soll und nicht durch Lesen des aktuellen Zustandes der GPIO Pins. Diese Einstellung bringt den wesentlichen Vorteil mit sich, dass die Interrupt-basierte Verarbeitung von Änderungen durch das Lesen des aktuellen Zustandes nicht beeinflusst wird. Dies ermöglicht es zu jeder Zeit mit einem Thread den aktuellen Zustand der GPIO Pins einzulesen, während ein anderer Thread auf einen Interrupt wartet, ohne den anderen Thread zu beeinflussen. Dies macht es konkret möglich die Funktion rpiplc_digital_read_port für einen Port aufzurufen, während ein anderer Thread mit Hilfe der Funktion rpiplc_wait_for_change auf eine Änderung am selben Port wartet. Würde ein Interrupt durch das Lesen des aktuellen Zustandes zurückgesetzt werden, bestünde die Gefahr, dass bei ungünstigem Timing durch das Lesen des aktuellen Zustandes ein Interrupt zurückgesetzt wird, bevor er vom wartenden Thread verarbeitet werden kann. Dies könnte schlimmstenfalls dazu führen, dass ein Interrupt gar nicht registriert wird oder aber der Inhalt des INTCAP Registers überschrieben wird, bevor dieser

ausgelesen werden kann. Womit es nicht mehr möglich wäre nachzuvollziehen, wodurch ein Interrupt ausgelöst wurde.

Anschließend wird an ausgewählten GPIO-Pins der IO-Expander die Erzeugung von Interrupts bei Änderungen aktiviert und die IO-Expander werden in der Steuerungsbibliothek als initialisiert markiert (vgl. Abschnitt 4.8.3). Abschließend werden noch einige interne Variablen initialisiert und die Steuerungsbibliothek selbst wird intern als initialisiert markiert.

4.8.6. Setzen von Ausgängen

Zum Setzen von Ausgängen können die Funktionen rpiplc_digital_write_port und rpiplc_digital_write_pin genutzt werden. Bei beiden Funktionen wird zunächst ein Mutex gesperrt um prüfen zu können, ob die Steuerungsbibliothek initialisiert ist. Es gibt für jeden Ausgabeport, für die Interruptbehandlung und für den Zugriff auf die I^2C Schnittstelle jeweils einen eigenen Mutex, so dass unabhängige Operationen konkurrierend ohne gegenseitige Beeinträchtigung durchgeführt werden können. Der zugehörige Mutex für den zugegriffenen Ausgabeport wird für die gesamte Ausführungszeit der jeweiligen Funktion gehalten. Dadurch wird speziell im Falle von mehreren konkurrierenden Aufrufen von rpiplc_digital_write_port sichergestellt, dass zunächst alle acht Ausgaben entsprechend dem einen Aufruf und dann entsprechend dem anderen Aufruf gesetzt werden und keine inkonsistenten Zwischenzustände erzeugt werden.

Die einzelnen Ausgaben eines Ports werden durch Schrieben in die zugehörigen Dateien des sysfs-Interface der Reihe nach gesetzt. Sollte bei der ersten Schreiboperation ein Fehler auftreten, wurde noch nichts verändert, der Vorgang kann abgebrochen werden und der Fehler kann normal an die Anwendungssoftware gemeldet werden. Tritt ein Fehler erst bei einer späteren Schreiboperation auf, wurde bereits ein Teil der Ausgaben gesetzt, während der Rest nicht verändert werden kann, was einen inkonsistenten Zustand aus der Sicht des Nutzers darstellt. Daher wird in diesem Fall der Fehler mit einer höheren Priorität markiert (vgl. Abschnitt 4.8.2).

4.8.7. Lesen von Eingängen, der Filterkonfiguration und des Hardwarestatus

Die Funktionen zum Lesen des aktuellen Zustandes der Eingänge eines Ports, der Filterkonfiguration und des Hardwarestatus konnten sehr einfach gehalten werden. Jede dieser Funktionen muss im Wesentlichen nichts anderes tun als das entsprechende Register eines IO-Expanders zu lesen. Notwendige Synchronisation wird, wie im Abschnitt 4.8.3 beschrieben, von der Hilfsfunktion, welche zum Lesen über den I^2C Bus implementiert wurde, durchgeführt und wie im Abschnitt 4.8.5 geschildert, wird durch das Lesen der aktuellen Zustände der Eingänge auch die Interruptbehandlung (vgl. Abschnitt 4.8.9) nicht beeinflusst.

4.8.8. Setzen der Status-LEDs

Die drei zweifarbeigen Status-LEDs sind an den GPIO Pins eines der IO-Expander angeschlossen. Das Setzen erfolgt durch Schreiben des entsprechenden Registers. Auch hier wird die notwendige Synchronisation durch die Hilfsfunktion, welche zum Schreiben über den I^2C Bus implementiert wurde, durchgeführt, so dass die Funktion rpiplc_set_status_LEDs nichts weiter tun muss, als ins entsprechende Register des IO-Expanders zu schreiben.

4.8.9. Interruptbehandlung

Die Funktion rpiplc_wait_for_change ermöglicht es einem Thread auf eine Eingabeänderung oder eine Änderung des Status der Hardware zu warten. Wie schon zuvor erwähnt, kann mit der Funktion rpiplc_wait_for_change auf eine bestimmte Quelle, beispielsweise einen Port, zu jedem Zeitpunkt

maximal ein Thread warten. Die Hardware gibt keinen Anlass dazu in diese hardwarenahe Steuerungsbibliothek eine Funktionalität zu implementieren, die es ermöglichen würde, dass mehrere Threads auf dieselbe Quelle warten. Diese Funktionalität kann problemlos auf einer höheren Abstraktionsschicht implementiert werden. Um eine korrekte Funktion der gewählten Implementierung zu gewährleiten, wird von der rpiplc_wait_for_change Funktion zunächst sichergestellt, dass wirklich immer höchstens ein Thread auf eine Quelle wartet. Es wird intern für jede der drei Quellen von Änderungen (Port 1, Port 2 und Hardwarestatus) vermerkt, ob bereits ein Thread auf diese Quelle wartet. Versucht ein weiterer Thread auf dieselbe Quelle zu warten, wird ein entsprechender Fehlercode zurückgeliefert.

Nach dieser anfänglichen Prüfung beginnt der eigentlich interessante aber auch komplizierte Teil der Funktion. Im späteren Verlauf der Funktion kann einfach darauf gewartet werden, dass einer der IO-Expander eine Interruptleitung aktiviert, woraufhin die Eingabeänderung aus dem INTCAP Register des IO-Expanders ausgelesen werden und an die Anwendungssoftware zurückgeliefert werden kann. Zu Beginn der Funktion kann aber weder der Zustand der Interruptleitung noch der Inhalt des INTCAP Registers genutzt werden um festzustellen, ob eine Änderung bereits vorliegt.

Anfangszustand

Ist die Interruptleitung zu diesem Zeitpunkt inaktiv bedeuten dies zwar, dass seit dem letzten Lesen des INTCAP Registers keine Änderung eingetreten ist. Dies bedeutet aber nicht, dass der aktuelle Zustand der Eingaben dem entspricht der zuletzt aus dem INTCAP Register gelesen wurde. Es kann nämlich sein, dass nach der Änderung, welche den letzten Interrupt ausgelöst hat, weitere Änderungen eingetreten sind, bevor die erste Änderung, welche im INTCAP Register abgelegt wurde, ausgelesen wurde. Für die nachfolgenden Änderungen können offensichtlich keine Interrupts erzeugt werden, da die Interruptleitung bereits aktiv ist. Wird die erste Änderung aus dem INTCAP Register schließlich ausgelesen, wird in diesem Moment die Interruptleitung inaktiv gesetzt. Auch wenn der gelesene Zustand nicht dem aktuellen Zustand der Eingaben entspricht und offensichtlich weitere Änderungen erfolgt sind, reaktivieren die IO-Expander in dieser Situation nicht die Interruptleitung um dies anzuzeigen. Es können also Änderungen vorhanden sein, obwohl die Interruptleitung inaktiv ist.

Ist die Interruptleitung zu Beginn der Funktionsausführung bereits aktiv, bedeutet dies zwar, dass seit dem letzen Auslesen des INTCAP Registers eine Änderung erfolgt ist. Dies bedeutet aber nicht automatisch, dass der aktuelle Zustand der Eingaben vom zuletzt gelesenen Zustand abweicht. Erstens ist nicht bekannt, wann das INTCAP Register zuletzt ausgelesen wurde. Daher kann die Information beliebig alt sein und es können bereits beliebig viele Änderungen erfolgt sein. Zweitens kann ein ähnliches Problem, wie im Falle einer inaktiven Interruptleitung entstehen. Beispielsweise kann es passieren, dass eine Eingabeleitung ihren Zustand von low auf high wechselt, wodurch ein Interrupt erzeugt wird und sich der Zustand der Leitung wieder zu low ändert, bevor das INTCAP Register ausgelesen wurde. Aus dem INTCAP Register wird in diesem Fall dennoch der Zustand high ausgelesen, da dort immer der Zustand gespeichert wird, welcher vorlag, als der Interrupt erzeugt wurde. Durch Lesen des INTCAP Registers wird der Interrupt gelöscht und die Interruptleitung wird inaktiv. Ändert sich der Zustand der Eingabeleitung später erneut zu high, wird erneut ein Interrupt ausgelöst. Dabei entspricht der Zustand der Eingaben nun wieder dem, der zuletzt aus dem INTCAP Register ausgelesen wurde. Die Interruptleitung ist also aktiv, obwohl keine Änderung vorliegt.

Der Inhalt des INTCAP Registers ist zu diesem Zeitpunkt ebenfalls nutzlos, da er erstens beliebig alt sein kann und zweitens Änderungen aufgetreten sein konnten, welche nicht im INTCAP Register erfasst wurden. Wie eben beschrieben kann es unter bestimmten Umständen vorkommen, dass eine Änderung keinen Interrupt erzeugt. Diese wird dann natürlich auch nicht im INTCAP Register verzeichnet.

Die eben beschriebenen Probleme können nur auftreten, wenn sich die Eingaben zumindest zeitweise schneller ändern, als die Änderungen von der Steuerungssoftware verarbeitet werden können. Es ist offensichtlich, dass es unter diesen Umständen unvermeidlich ist, dass einige kurzfristige Änderungen verloren gehen. Es kann aber dennoch sichergestellt werden, dass nie auf eine Änderung gewartet wird, die bereits vorliegt. Um dies zu gewährleisten muss von der Funktion rpiplc_wait_for_change einige Vorarbeit geleistet werden, bevor schließlich auf einen Interrupt gewartet werden kann.

Vorbereitungsphase

Die Funktion rpiplc_wait_for_change liest zunächst den aktuellen Zustand aller Eingaben, nicht das INTCAP Register, ein. Weicht dieser bereits von dem Zustand, welcher der Anwendungssoftware als letzter Zustand bekannt ist, ab, wird der neue Zustand unmittelbar zurückgeliefert. Dadurch wird sichergestellt, dass nicht mit veralteten Werten gearbeitet wird. Wurde keine Änderung festgestellt, wird zunächst geprüft, welche der Interruptleitungen aktiv sind. An dieser Stelle kann aus einer aktiven Interruptleitung immer noch keine Information gewonnen werden, da nicht bekannt ist, ob der Interrupt vor oder nach dem Lesevorgang ausgelöst wurde. Wurde er bereits vorher ausgelöst, ist die Information im INTCAP Register veraltet. Wurde er nachher ausgelöst, ist im INTCAP Register tatsächlich eine neue Änderung vermerkt. Nur wenn die Interruptleitung inaktiv ist, ist bekannt, dass seit dem Lesen der aktuellen Zustände keine Änderung aufgetreten ist. Daher werden für alle aktiven Interruptleitungen an dieser Stelle von der Funktion zunächst die zugehörigen INTCAP Register ausgelesen um den Interrupt zu löschen und die zugehörige Interruptleitung inaktiv zu setzen. Der Wert, welcher aus dem INTCAP Register gelesen wurde, kann nicht verwertet werden und muss einfach verworfen werden.

Das Lesen des aktuellen Zustandes und das Prüfen, ob eine Interruptleitung aktiv ist, werden so lange wiederholt bis alle Interruptleitungen inaktiv sind oder eine Änderung gefunden wurde. Nur wenn an dieser Stelle alle Interruptleitungen inaktiv sind, ist sichergestellt, dass seit dem Lesevorgang keine Änderung erfolgt ist und dass jede nachfolgende Änderung durch einen Interrupt gemeldet werden wird. Selbst bei mehrmaliger Wiederholung kann aus einer aktiven Interruptleitung keine Information gewonnen werden, da nie bekannt ist, ob die Änderung vor oder nach dem jeweiligen Lesevorgang erfolgt ist. Ist sie vor dem Lesevorgang erfolgt, ist die Information im INTCAP Register veraltet und kann verworfen werden. Die Anzahl der Versuche ist in der Funktion beschränkt. Gelingt es selbst nach mehrmaliger Wiederholung nicht einen Zustand herbeizuführen in dem alle Interruptleitungen inaktive sind, ist davon auszugehen, dass die IO-Expander nicht ordnungsgemäß arbeiten. Würden tatsächlich ständig Änderungen erfolgen, würde mit hoher Wahrscheinlichkeit bei einem der Lesevorgänge des aktuellen Zustandes eine Änderung festgestellt werden.

Es fällt auf, dass bei extrem ungünstigem Timing sehr kurze Änderungen evtl. ignoriert werden. Dies ist, wie zuvor erwähnt, unvermeidlich, da die Änderungen schneller erfolgen, als sie von der Anwendungssoftware verarbeitet werden können. Unter diesen Umständen müssen zwangläufig irgendwo Änderungen verloren gehen. Die gewählte Implementierung stellt aber sicher, dass nur kurzfristige Änderungen verloren gehen können. Es kann nicht passieren, dass auf einen Zustand gewartet wird, der bereits vorliegt, was möglich wäre, wenn die beschriebene Vorarbeit von der Funktion nicht geleistet werden würde. Ohne diese Vorarbeit könnte ein Thread unbegrenzt lange auf einen Interrupt warten, der nie auftreten würde, weil die Änderung, welche er melden soll, bereits vorliegt.

Normalerweise ändern sich die Eingaben nicht schneller als sie verarbeitet werden können, d.h. dass die Anwendungssoftware die rpiplc_wait_for_change Funktion aufruft bevor eine Eingabeänderung erfolgt und dass diese die Vorbereitungsphase abschließen kann, bevor die erste Änderung eintritt. In diesem Fall, also im Normalfall, können beliebig kurze Änderungen erfasst werden, sofern sie von den IO-Expandern erfasst werden können. Wenn eine Änderung nur kurzfristig ist, heißt es ja per Definition, dass sich die Eingabe wieder schnell zu ihrem alten Zustand zurückändert. Dass diese zweite Än-

derung der Eingabe bei sehr kurzfristigen Änderungen erfolgt, bevor die erst Änderung verarbeitet werden konnte, also speziell bevor die Anwendungssoftware rpiplc_wait_for_change erneut aufrufen konnte, stellt kein Problem dar, solange der Zustand dann solange gehalten wird, bis er eingelesen werden konnte.

Eigentliche Interruptbehandlung mit verbessertem zeitlichen Verhalten

Nach dieser komplizierten Vorbereitungsphase kann nun endlich auf das Auftreten eines Interrupts gewartet werden. Tritt nun ein Interrupt auf, ist bekannt, dass im zugehörigen INTCAP Register eine Änderung gegenüber dem letzen bekannten Zustand vermerkt ist. Diese kann ausgelesen und direkt an die Anwendungssoftware weitergeleitet werden. Es wurde nämlich bereits sichergestellt, dass der dann gelesene Wert vom zuvor bekannten abweicht. Damit ist bekannt, dass das nächste Register, das gelesen werden wird, eines der INTCAP Register von einem der IO-Expander sein wird. Da Änderungen am Status eher selten zu erwarten sind, kann sogar mit hoher Wahrscheinlichkeit ausgesagt werden, dass das INTCAP Register, welches den Eingaben zugeordnet ist, von einem der IO-Expander gelesen werden wird. Das letzte Register, das gelesen wurde, ist zu diesem Zeitpunkt aber das Register, welches die aktuellen Werte aller Eingaben enthält, und nicht das INTCAP Register. Das bedeutet, dass beim künftigen Lesen des INTCAP Registers zunächst die Registeradresse übermittelt werden müsste, was die Dauer der Leseoperation deutlich verlängern würde. Da aber an dieser Stelle bekannt ist, dass höchstwahrscheinlich bis dahin kein anderes Register gelesen werden wird, setzt die Funktion noch bevor sie anfängt auf einen Interrupt zu warten, präventiv die benötigte Registeradresse. An dieser Stelle stellt die zusätzliche Zeit, die dafür benötigt wird, kein Problem dar, da mit hoher Wahrscheinlichkeit in der kurzen Zeit seit dem letzten Prüfen der Interruptleitung noch keine Änderung eingetreten ist und auf diese ohnehin erst gewartet werden muss. Ob sich der Thread gleich schlafen legt oder vorher noch die Adresse setzt, spielt hier keine Rolle.

Tritt dann irgendwann schließlich ein Interrupt auf, wird der Thread, welcher rpiplc_wait_for_change aufgerufen hat, aufgeweckt. Die Funktion liest das zugehörige INTCAP Register und gibt den neuen Wert an die aufrufende Funktion zurück. Dies erfolgt sehr schnell, da die Leseoperation an dieser Stelle meist sehr kurz ist, da die Registeradresse nicht mehr übertragen werden kann. Wurde in der Zwischenzeit ein anderes Register gelesen, weil ein anderer Thread beispielsweise die Eingaben eines Ports gelesen hat, muss die Registeradresse ausnahmsweise nochmals übertragen werden.

4.9. Integration in FORTE

Es wurde ein Modul zur Integration in die Laufzeitumgebung FORTE basierend auf der zuvor entwickelten Steuerungsbibliothek implementiert. Seine Funktionsfähigkeit wurde mit einfachen Beispielanwendungen getestet. Im Folgenden wird die Implementierung näher betrachtet. Der Quellcode findet sich auf der beiliegenden CD im Archiv FORTE_1.7.1_rpiplc.zip.

In 4DIAC stehen für digitale Ein- und Ausgaben sog. IX und QX Blöcke zur Verfügung, welche jeweils eine einzelne binäre Ein- oder Ausgabe repräsentieren. Das entwickelte Gerät besitzt neben digitalen Ein- und Ausgängen einige Statusbits, welche den Zustand der Hardware wiederspiegeln und 6 frei steuerbare Status-LEDs, was aber auch nur binäre Informationen sind, die gelesen bzw. geschrieben werden können. Daher wurde beschlossen, alle Funktionen des Geräts durch IX und QX Blöcke zu modellieren. Die 16 digitalen Eingänge des Geräts, sowie die diversen Statusbits können durch IX Blöcke repräsentiert werden, während die 16 digitalen Ausgänge, sowie die sechs Status-LEDs durch QX Blöcke repräsentiert werden können.

FORTE ist objektorientiert in C++ implementiert. Für jeden IX und jeden QX Block wird ein eigenes Objekt erzeugt, welches den entsprechenden Block repräsentiert. Die Klassen, die zur Darstellung von IX Blöcken instanziiert werden, müssen eine Funktion readPin bereitstellen, mit der der aktuelle Wert

der Eingabe abgefragt werden kann. Klassen, die für QX Blöcke genutzt werden, müssen eine writePin Funktion bereitstellen, mit der die Ausgabe gesetzt werden kann. Beide Klassen müssen außerdem die Funktionen initialise und deinitialise implementieren, welche von FORTE bei der Instanziierung bzw. beim Löschen des Objekts aufgerufen werden.

Die Vererbungsstruktur in FORTE ist dabei so aufgebaut, dass sich die Klasse, welche zur Erstellen von IX Objekten dient und die Klasse, die zum Erstellen von QX Objekten dient, eine gemeinsame Implementierung der initialise Funktion und eine gemeinsame Implementierung der deinitialise Funktion teilen. Der initialise Funktion wird ein Parameter übergeben, welcher angibt, ob es sich um ein IX oder um ein QX Objekt handelt. Im Wesentlichen mussten also die vier Funktionen readPin, writePin, initialise und deinitialise passend für das entwickelte Gerät implementiert werden.

4.9.1. Setzen einer Ausgabe

Die Implementierung der writePin Funktion war einfach, da ein Aufruf dieser Funktion nur in einen entsprechenden Aufruf der rpiplc_digital_write_pin Funktion bzw. der rpiplc_set_status_LEDs Funktion der Steuerungsbibliothek übersetzt werden muss. Beim Setzten einer digitalen Ausgabe kann die rpiplc_digital_write_pin Funktion mit passenden Parametern ohne jegliche Synchronisation aufgerufen werden, da die Steuerungsbibliothek selbst thread-safe implementiert wurde.

Beim Aufruf der Funktion rpiplc_set_status_LEDs muss der Zustand aller Status-LEDs übergeben werden. Da es aber für jede LED ein eigenes QX Objekt gibt, kennt das aufrufende Objekt nur den Zustand einer einzigen Status-LED. Daher war es hier notwendig den Zustand aller Status-LEDs in einer Klassenvariable zu hinterlegen. Diese wird bei jeder Änderung einer Status-LED ausgelesen und aktualisiert. Der Zugriff auf diese Variable wird durch ein Synchronisationsobjekt kontrolliert.

4.9.2. Lesen einer Eingabe

Die Realisierung der readPin Funktion war etwas kompilierter. Das wesentliche Problem bei dieser besteht darin, dass jeder Eingabepin durch ein eigenes Objekt repräsentiert wird, wodurch für die readPin Funktion immer nur der Status eines einzelnen Pins interessant ist, während in Hardware und folglich auch mit Hilfe der entwickelten Steuerungsbibliothek immer nur alle Eingänge eines Ports auf einmal gelesen werden können. Natürlich wäre es möglich bei jedem Aufruf der readPin Funktion die rpiplc_digital_read_port Funktion aufzurufen, welche jeweils acht Werte einliest, und die nicht benötigten sieben Werte zu ignorieren. Dies würde aber einen großen Overhead bedeuten.

Dieses Problem wurde durch das Hinzufügen eines Zwischenspeichers gelöst, welcher immer einen möglichst aktuellen Wert aller Eingänge enthält. Es wurde ein separater Thread implementiert, welcher mit Hilfe der Funktion rpiplc_wait_for_change alle Eingaben überwacht und den Zwischenspeicher bei einer Änderung umgehend aktualisiert. Die Funktion readPin muss bei jedem Aufruf nur noch den gewünschten Wert aus dem Zwischenspeicher entnehmen. Der Zwischenspeicher wird durch ein Synchronisationsobjekt vor konkurrierenden Zugriffen geschützt.

4.9.3. Initialisierung und Deinitialisierung

Die gemeinsame Initialisierungsmethode wird für jedes einzelne IX und QX Objekt bei dessen Erstellung aufgerufen. Die Anzahl der Aufrufe hängt davon ab, wie viele solcher Objekte eingesetzt werden. Selbiges gilt für die Deinitialisierungsmethode, welche bei der Zerstörung eines jeden solchen Objekts aufgerufen wird.

Es wurde eine Klassenvariable definiert, welche stets die Anzahl der Objekte angibt, für die die Initialisierungsmethode aber noch nicht die Deinitialisierungsmethode aufgerufen wurde, also die Anzahl der aktiven Objekte. Wird bei der Initialisierung festgestellt, dass keine anderen aktiven Objekte vor-

handen sind, wird die entwickelte Steuerungsbibliothek, welche zum Zugriff auf die Hardware genutzt wird, initialisiert und der Thread, welcher den im vorhergehenden Abschnitt beschriebenen Zwischenspeicher aktuell hält, wird gestartet. Dieser wird wieder beendet und die Steuerungsbibliothek wird deinitialisiert, wenn beim Aufruf der Deinitialisierungsmethode festgestellt wird, dass es sich um das einzige aktive Objekt handelt. Ansonsten wird nur das Objekt selbst initialisiert bzw. deinitialisiert.

4.9.4. Weiterleitung von Interrupts

Die Verarbeitung von Eingabeänderungen ist in weiten Teilen Interrupt-basiert. Der Thread, welcher den Zwischenspeicher mit den Werten der Eingaben stets auf dem aktuellen Stand hält, ist inaktiv solange keine Eingabeänderungen erfolgen. Nur bei Eingabeänderungen wird der Thread aufgeweckt um den Zwischenspeicher zu aktualisieren. (Aus praktischen Gründen wird der Thread auch bei besonders langen Perioden ohne Eingabeänderung durch einen Timeout kurz aufgeweckt um zu prüfen, ob versucht wird, das Programm zu beenden.) Das Abfragen der Werte aus dem Zwischenspeicher muss aber dennoch periodisch erfolgen unabhängig davon, ob eine Änderung vorhanden ist oder nicht.

Aktuelle Versionen von FORTE unterstützen die Möglichkeit die Information, dass eine Eingabeänderung erfolgt ist, weiterzuleiten, so dass diese gezielt verarbeitet werden kann ohne die Eingaben periodisch prüfen zu müssen. Es wäre also möglich den Interrupt quasi weiterzuleiten. Diese Funktionalität wurde aus zeitlichen Gründen nicht mehr implementiert, es spricht aber nichts dagegen diese in Zukunft zu ergänzen.

5. Untersuchung und Verbesserung des zeitlichen Verhaltens

Im bisherigen Verlauf dieser Arbeit wurde eine SPS entwickelt, ein Prototyp gebaut und eine passende Steuerungsbibliothek implementiert, die einfachen Zugriff auf die Hardware ermöglicht. Einfache Funktionstests haben gezeigt, dass die gebaute SPS grundsätzlich funktionsfähig ist und dass die elektrischen Eigenschaften, einschließlich der Schutzfunktionen, den Erwartungen entsprechen. In diesem Kapitel soll nun das zeitliche Verhalten des Geräts untersucht und nach Möglichkeit im Laufe des Kapitels verbessert werden.

5.1. Referenzmessungen

Als allererstes sollen einige Referenzmessungen durchgeführt werden. Diese sollen unter Einsatz des normalen Raspbian Betriebssystems, wie es von den Entwicklern des Raspberry Pi zum Download angeboten wird, ohne jegliche Modifikationen und der zuvor entwickelten Steuerungsbibliothek erfolgen. Diese Messungen sollen Referenzwerte liefern um eventuelle spätere Verbesserungen des Verhaltens erkennen und bewerten zu können.

Hinsichtlich des zeitlichen Verhaltens ist insbesondere die Reaktionszeit des Geräts von Interesse. Um diese messen zu können, ist es aber erforderlich, dass das Gerät auch aktiv eine Reaktion auf Eingabeänderungen produziert. Es muss also eine Steuerungssoftware ausgeführt werden, welche eine geeignete Reaktion erzeugt. Diese kann entweder in C/C++ unter Einsatz der entwickelten Steuerungsbibliothek implementiert oder komfortable in 4DIAC erstellt werden.

Im späteren Verlauf des Kapitels soll die Laufzeitumgebung FORTE zum Einsatz kommen, welche die Ausführung von Steuerungsprogrammen, welche mit 4DIAC erstellt wurden, erlaubt. Für die ersten Experimente soll aber eine einfache Steuerungssoftware in C implementiert werden um später das Laufzeitverhalten beider Varianten vergleichen zu können.

5.1.1. Fragestellung

Um einen ersten Eindruck vom zeitlichen Verhalten zu bekommen, soll im ersten Experiment die Reaktionszeit und deren Schwankung über mehrere Durchläufe hinweg gemessen werden. Konkret soll die Zeit zwischen einer Änderung an einer Eingabeleitung und der Reaktion der SPS bestimmt werden, wobei als Reaktion eine Änderung an einer Ausgabeleitung erwartet wird. Dazu soll die Zeit zwischen der steigenden Flanke auf einer Eingabeleitung und der steigenden Flanke auf einer Ausgabeleitung, welche die Reaktion auf die Eingabeänderung darstellt, ermittelt werden.

Es soll zunächst von möglichst idealen Bedingungen, also keiner übermäßigen CPU-Belastung durch parallel laufenden andere rechenintensive Programme, keiner Beeinflussung der Reaktionszeit durch gleichzeitige oder zeitlich nah gelegenen andere Änderungen von Eingaben und keiner zeitintensiven Verarbeitung der Eingabeänderung, ausgegangen werden.

5.1.2. Experimentdesign

Um die gewünschten Werte ermitteln zu können, sind verschiedene Aufgaben zu lösen. Das Gerät muss mit einem Betriebssystem bestückt werden, eine Steuerungssoftware, welche eine Reaktion auf eine Änderung an einer Eingabeleitung erzeugt, muss entwickelt und ausgeführt werden und ein Testsignal muss an eine Eingabeleitung angelegt werden, welches die nötigen Eingabeänderungen erzeugt. Schließlich muss dann nach der Zeitunterschied der beiden steigenden Flanken an Ein- und Ausgabeleitung gemessen werden.

Betriebssystem

Wie schon einleitend erwähnt, soll als Betriebssystem zunächst ein unmodifiziertes Raspbian zum Einsatz kommen. Gegenwärtig werden von den Entwicklern des Raspberry Pi zwei Versionen angeboten, eine normale und eine Minimalversion. Da für die geplanten Messungen keine Software benötigt wird, welche in der Minimalversion nicht enthalten wäre und möglichst ideale Bedingungen vorherrschen sollen, wird für dieses Experiment die aktuelle Minimalversion des Systems (Raspbian Jessie Lite; Version: November 2015; Release date: 2015-11-21; Kernel version: 4.1) gewählt. Bei dieser sollte die Grundlast des Systems tendenziell geringer oder zumindest nicht höher sein als bei der normalen Version. Das System wird als Datenträgerabbild zum Download angeboten. Dieses soll als Vorbereitung auf das Experiment auf eine Micro-SD geschrieben werden, welche ins Raspberry Pi eingesetzt wird. Vor Ausführung einer Steuerungssoftware ist dann noch das Modul i2c-dev zu laden um der entwickelten Steuerungsbibliothek eine Kommunikation über die I^2C Schnittstelle zu ermöglichen.

Steuerungssoftware zur Erzeugung einer Reaktion

Gemessen werden soll die Zeit zwischen einer steigenden Flanke auf einer Eingabeleitung und einer steigenden Flanke auf einer Ausgabeleitung. Es ist also nötig, dass die SPS auf eine steigende Flanke an einem Eingang mit einer steigenden Flanke an einem Ausgang reagiert. Um die Messung oft wiederholen zu können, muss zudem das Ausgabesignal noch im Verlauf des Experiments irgendwann auch wieder auf low gesetzt werden um erneut eine steigende Flanke erzeugen zu können. Da auf der Eingabeleitung logischerweise auch irgendwann eine fallende Flanke auftreten muss, bevor eine weitere steigende Flank erzeugt wird, kann diese zum Anlass genommen werden an der Ausgabe ebenfalls eine fallende Flanke zu erzeugen.

Dieses Verhalten soll durch ein kleines zu implementierendes Testprogramm erzeugt werden, welches die zuvor implementierte Steuerungsbibliothek nutzt um die Eingaben beider Ports einzulesen und jegliche Änderungen auf die zugehörigen Ausgaben kopiert. Es soll also dafür sorgen, dass die Zustände der Ausgabeleitungen immer den Zuständen der Eingabeleitungen entsprechen. Da die Eingaben nur auf die Ausgaben kopiert werden, ist keine zeitintensive Verarbeitung notwendig, was der Vorgabe für dieses Experiment entspricht.

In späteren Abschnitten sollen vergleichbare Messungen mit FORTE anstelle des hier genutzten Testprogramms vorgenommen werden. Um eine bessere Vergleichbarkeit der Messergebnisse zu schaffen, soll vom Testprogramm die Interrupt-basierte Verarbeitung von Eingaben genutzt werden, da die vorgenommene Integration in FORTE auf dieser basiert.

Testsignal

Ein Testsignal könnte in Software durch Schalten eines im Experiment nicht genutzten Ausgangs, welcher mit dem für das Experiment genutzten Eingang verbunden wird, erzeugt werden, allerdings bestünde die Möglichkeit, dass das Messergebnis durch das Schalten des Ausgangs, bzw. durch den Thread, der das Testsignals erzeugt, verfälscht wird. Um dies auszuschließen soll ein Testsignal extern erzeugt werden.

Da das geplante Testprogramm sowohl steigende als auch fallende Flanken verarbeitet, sollten beide weit genug auseinanderliegen um eine gegenseitige Beeinflussung zu verhindern. Insbesondere muss eine steigende Flanke weit genug von der vorhergehenden fallenden Flanke entfernt sein, um sicherzustellen, dass die Verarbeitung der fallenden Flanke abgeschlossen ist und keinen Einfluss mehr auf die Verarbeitung der steigenden Flanke hat, da dies sonst das Messergebnis verfälschen könnte.

Als Testsignal geeignet erscheint ein Rechtecksignal mit 50 Hz, welches periodisch zwischen 0 V und 24 V wechselt. Eine Frequenz von nur 50 Hz stellt sicher, dass jeder Wechsel vom vorhergehenden 10 ms entfernt ist. Dies dürfte ausreichend sein um eine gegenseitige Beeinflussung zu verhindern, da deutlich kürzere Reaktions- bzw. Verarbeitungszeiten erwartet werden. Sollten die Reaktionszeiten entgegen der Erwartung in die Größenordnung von 10 ms kommen, müsste die Frequenz entsprechend gesenkt werden.

Ein Rechtecksignal mit 50 Hz kann mit Hilfe eines vorhandenen Funktionsgenerators erzeugt werden, allerdings kann der zu Verfügung stehende Funktionsgenerator nur Spannungen im Bereich von ±2 V erzeugen. Daher ist eine einfache Schaltung zu entwickeln, welche aus einem Rechtecksignal in diesem Spannungsbereich ein Rechtecksignal erzeugt, das zwischen 0 V und 24 V wechselt.

Messung

Zur Zeitmessung sollen die Signale auf der genutzten Eingabe- und der zugehörigen Ausgabeleitung mit einem USB-Oszilloskop überwacht werden. Es steht ein PicoScope 3205D MSO von der Firma pico Technology zur Verfügung. Dieses verfügt über 2 analoge Kanäle mit 100 MHz Bandbreite, 16 digitale Kanäle (bis 100 MHz) und Speicherplatz für 256 MS. Es sind Abtastraten bis zu 1 GS/s (Echtzeit) möglich. Das Oszilloskop enthält auch den Funktionsgenerator, welcher zur Erzeugung des Testsignals genutzt werden soll.

Um den genauen Spannungsverlauf am Ein- und Ausgang beobachten zu können, soll jeweils ein analoger Kanal zur Überwachung des Eingangs und einer zur Überwachung des Ausgangs genutzt werden. Eine Visualisierung des Signalverlaufs und einfache statistische Auswertungen, können mit der zugehörigen Software des Oszilloskops durchgeführt werden. Um eine genauere statistische Auswertung durchführen zu können, ist ein Programm zu implementieren, das die Daten vom Oszilloskop einliest, auswertet und in geeigneter Weise ausgibt.

5.1.3. Experimentvorbereitung

Wie in der Designphase festgestellt, erfordert das in diesem Abschnitt behandelte Experiment einige teils umfangreiche Vorbereitungen. Diese wurden durchgeführt und werden in den folgenden Unterabschnitten beschrieben.

Betriebssystem

Als allererstes wurde der eingesetzte Raspberry Pi mit dem aktuellen minimalen Raspbian bestückt. Danach wurde mit dem enthaltenen Tool raspi-config das Dateisystem auf die gesamte SD-Karte (8 GB) ausgedehnt, der SSH-Server aktiviert und i2c-dev in die Datei /etc/modules hinzugefügt. Damit alle Einstellungen wirksam werden, wurde ein Neustart durchgeführt.

Schaltung zur Erzeugung des Testsignals

Als nächstes wurde eine Schaltung entwickelt, welche aus einem Rechtecksignal, welches zwischen 0 V und 2 V wechselt ein Rechtecksignal erzeugt, welches zwischen 0 V und 24 V wechselt. Dabei war darauf zu achten, dass sich die Spannung des Testsignals im Verlauf einer steigenden bzw. fallenden Flanke weitestgehend monoton ändert. Geringfügige Störungen auf dem Signal können durch Anwenden einer Hysterese bei der Auswertung kompensiert werden, starke Störungen beim Zustandswechsel, wie sie beispielsweise bei elektromechanischen Schaltern oder Relais auftreten, würden die Auswertung aber erschweren und könnten die Ergebnisse verfälschen. Die Anstiegs- und Abfallzeiten des Testsignals sind hingegen unwesentlich, solange der zeitliche Spannungsverlauf bei jeder Wiederholung annähernd gleich ist und die Zeitmessung konsistent bei einem festen Spannungs-

pegel erfolgt. Insbesondere die Abfallzeiten sind irrelevant, da an fallenden Flanken keine Messungen erfolgen sollen.

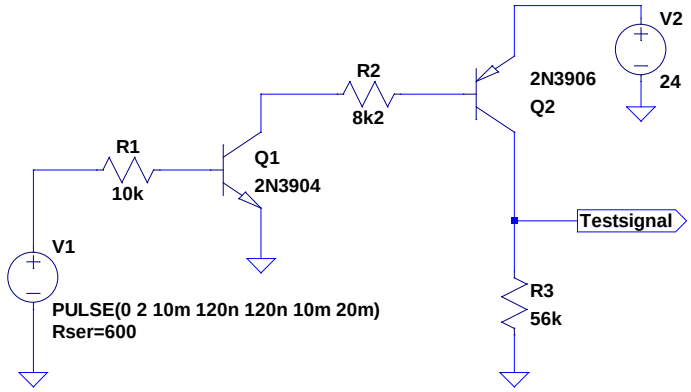


Abbildung 50: Schaltung zur Erzeugung eines Testsignals

Die benötigte Funktion konnte mit der Schaltung, welche in Abbildung 50 dargestellt ist, erreicht werden. V1 repräsentiert den Funktionsgenerator, welcher im Oszilloskop verbaut ist. Von diesem soll eine Rechteckspannung zwischen 0 V und 2 V mit einer Frequenz von 50 Hz erzeugt werden. Die Transistoren sorgen dafür, dass bei einem hohen Spannungspegel am Eingang der Ausgang auf annähernd 24 V hochgezogen wird. Um das Ausgabesignal auf etwa 0 V herunterzuziehen wurde ein Pull-Down-Widerstand verbaut. Dadurch sind die fallenden Flanken zwar relativ langsam, dies stellt aber, wie schon erwähnt, kein Problem dar. Die Schaltung wurde auf einem Experimentierboard aufgebaut.

Testprogramm

Zur Erzeugung der gewünschten Reaktion auf Eingabeänderungen wurde ein Testprogramm mit der Bezeichnung loopback in C implementiert. Der Quellcode dieses Programms findet sich auf der beiliegenden CD im Archiv librpiplc_1.0.0.zip unter /examples/loopback.

Das Testprogramm ruft zunächst die Funktion rpiplc_setup_rt auf um die Priorität des Threads, mit dem später Eingabeänderungen abgefragt werden, zu erhöhen und um die Auslagerung von Speicher zu unterbinden. Danach wird mit rpiplc_open die SPS initialisiert und die Konfiguration der Eingabefilter wird ausgegeben. Anschließend wird der Zustand aller Eingaben eingelesen und auf die Ausgaben kopiert.

Nach dieser einmaligen Initialisierungsphase wird mit rpiplc_wait_for_change auf eine Eingabeänderung gewartet. Sobald eine Eingabeänderung erkannt wird, wird diese auf den zugehörigen Ausgang geschrieben. Das Warten auf eine Eingabeänderung und das Kopieren auf den zugehörigen Ausgang wird dann so lange wiederholt bis die Eingabetaste gedrückt wird.

Ob die Eingabetaste gedrückt wurde, wird nach jeder Eingabeänderung nach dem zugehörigen Setzen des entsprechenden Ausgangs geprüft, was sicherstellen sollte, dass dadurch die Reaktionszeit nicht beeinflusst wird. Erfolgt, länger als 0,5 Sekunden keine Änderung wird ebenfalls geprüft, ob die Eingabetaste gedrückt wurde, bevor ein weiterer Wartevorgang eingeleitet wird. Was aber während des geplanten Experiments nicht auftreten kann, da ein 50 Hz Rechtecksignal am Eingang anliegen soll und sich die Eingabe damit stets ändert, bevor eine halbe Sekunde verstrichen ist. Wird die Eingabetaste gedrückt wird die SPS mit rpiplc_close in den Standby-Zustand versetzt.

Die Steuerungsbibliothek und die eben beschriebene Testanwendung wurden auf den Raspberry Pi übertragen, die Steuerungsbibliothek wurde kompiliert und installiert. Danach wurde das Testprogramm loopback kompiliert

Auswertungsprogramm

Es ist davon auszugehen, dass die Reaktionszeit der SPS von Durchlauf zu Durchlauf mehr oder weniger starken Schwankungen unterliegt. Ziel dieses Experiments ist es nicht nur die durchschnittliche Reaktionszeit, sondern auch die Schwankung der Reaktionszeit zu bestimmen. Dazu soll die Messung der Reaktionszeit möglichst häufig wiederholt werden. Zur Visualisierung der Ergebnisse scheint ein Histogramm am besten geeignet. Je nach Stärke der Schwankungen könnten die Messungen beispielsweise mit einer Auflösung von Mikrosekunden oder von Zehntel Mikrosekunden zusammengefasst werden. Dann kann mit Balken dargestellt werden, bei wie vielen Messungen die jeweilige Reaktionszeit gemessen wurde.

Um aussagekräftige Ergebnisse zu erhalten, scheint es sinnvoll die Messung wenigstens einige Tausend Mal zu wiederholen. Die Software, welche vom Hersteller des Oszilloskops geliefert wird, bietet zwar die Möglichkeit Zeitdifferenzen zu messen und auch einfache statistische Auswertungen durchzuführen, es besteht aber keine praktikable Möglichkeit die gewünschten Daten zur Erstellung eines Histogramms zu erheben. Dies könnte allerhöchstens manuell erfolgen, was äußerst zeitaufwendig wäre, wodurch die Anzahl der Messungen auf eine äußerst geringe Anzahl beschränkt werden müsste.

Die vorhandene Software bietet die Möglichkeit die erhobenen Messdaten im CSV-Format zu exportieren. Diese könnten dann von einem anderen Programm ausgewertet werden. Allerdings müsste dazu jeder einzelne Messwert exportiert werden. Ausgehend von einer Messdauer von 5 ms pro Durchlauf, einem Abtastintervall von 1 µs und zwei Kanälen, wäre der Export von 10.000 Messwerten pro Durchlauf erforderlich. Bei 1000 Durchläufen dauert ein solcher Export auf dem genutzten Notebook etwa 2 Minuten. Dies wäre noch durchaus akzeptabel, allerdings würde man schnell an die Grenzen stoßen, wenn man mehr Durchläufe machen möchte oder das Abtastintervall beispielsweise auf 0,1 µs verkürzen möchte. Zudem liegen die Daten nach dem Export im ASCII Format als Gleitkommazahlen vor. Diese müssten von einem Auswertungsprogramm also zunächst wieder umgewandelt werden.

Angesichts der Tatsache, dass im Laufe dieses Kapitels noch viele weitere Messungen geplant sind, bei denen unter Umständen deutlich mehr einzelne Messwerte notwendig sein werden, erschien der Export der Daten und die Auswertung mit einem zusätzlichen Programm nicht der geeignete Weg zu sein. Stattdessen wurde in C ein eigenes Programm mit dem Namen analyzer implementiert, welches die Messdaten direkt vom USB-Oszilloskop einliest, auswertet und in geeigneter Weise ausgibt um daraus ein Histogramm erstellen zu können.

Zur Steuerung des Oszilloskops und zum Abfragen von Messdaten aus einem eigenen Programm heraus wird von pico Technology eine API und zugehörige Dokumentation [95] angeboten. Diese wurden genutzt um das benötigte Programm zu entwickeln. Das Programm dient ausschließlich zur Erhebung der Daten, die für dieses Kapitel benötigt werden. Es ist nicht interaktiv, führt automatisch die nötigen Messungen durch, wertet sie aus, gibt die Ergebnisse aus und endet dann. Alle relevanten Einstellungen, beispielsweise die Abtastrate, der Messbereich, die Anzahl an Messungen und die Anzahl der Wiederholungen sind fest einprogrammiert und können nach dem Kompilieren nicht mehr verändert werden.

Um aber dennoch einfach verschiedene Einstellungen für verschiedene Messungen verwenden zu können, wurden alle Einstellungen als Konstanten hinterlegt und in eine extra Header-Datei ausgelagert. Auch die Art der Auswertung, welche erfolgen soll, ist in dieser Headerdatei hinterlegt. Für die einzelnen Experimente, welche in diesem Kapitel noch folgen sollen, können dann jeweils eigene Konfigurationsdateien angelegt werden, wo die benötigten Einstellungen gesetzt werden und einzelnen Auswertungen zu- oder abgeschaltet werden können. Welche Einstellungen gelten sollen und was getan werden soll, wird dann durch Einbinden der passenden Header-Datei bestimmt. Um fehlerhafte Werte in der Konfiguration schnell aufzufinden, werden alle gesetzten Werte in irgendeiner Weise zur

Laufzeit geprüft. Sollte ein ungültiger Wert gefunden werden, wird versucht den zulässigen Wertebereich zu ermitteln und auszugeben. Da die Konfigurationswerte aber nicht zur Laufzeit aus einer Konfigurationsdatei geladen werden und auch keine Benutzereingaben im eigentlichen Sinne darstellen, kann die Prüfung meist einfacher ausfallen, da nicht beliebige Fehleingaben zu erwarten sind.

Der Quellcode des entwickelten Programms findet sich auf der beiliegenden CD im Archiv analyzer_1.0.0.zip. Im Folgenden soll die Funktionsweise des Programms nur in groben Zügen beschreiben werden. Wird das Programm ausgeführt, initialisiert es das Oszilloskop, führt die festgelegten Einstellungen durch und gibt die gesamte aktive Konfiguration auf stdout aus. Dort landen später auch die Ergebnisse der Messungen. Dies erleichtert es im Nachhinein nachzuvollziehen, welche Messwerte mit welcher Konfiguration erhoben wurden ohne diese explizit notieren zu müssen.

Das Oszilloskop bietet folgende Möglichkeiten an um Messdaten zu erheben:

1. Streaming Modus: Ist das Abtastintervall lang, fallen pro Sekunde also nur wenige Daten an, können diese kontinuierlich an den PC übertragen werden. Dies sollte bis 125 MS/s funktionieren [95], was für dieses Experiment mehr als ausreichend wäre. Allerdings bietet dieser Modus im vorliegenden Fall keinen erkennbaren Vorteil um diese Einschränkung zu rechtfertigen.

2. Block Modus: Alternativ kann jeweils eine bestimmte Menge Messwerte im internen Speicher des Oszilloskops gesammelt und dann auf einmal an den PC Übertragen werden. Üblicherweise wird dabei immer auf einen Trigger gewartet. Tritt das Trigger-Ereignis auf werden eine bestimmte Zeit Daten gesammelt und dann an den PC übertrage. Danach wird auf das nächste Trigger-Ereignis gewartet. In diesem Modus gibt es keine Einschränkung bezüglich der maximalen Abtastrate. Es kann mit bis zu 1 GS/s abgetastet werden. Der Nachteil an diesem Modus besteht darin, dass zwischen dem Ende einer Aufzeichnung und dem Start der nächsten immer ein paar Millisekunden vergehen müssen, da die Daten zunächst übertagen werden müssen und der Trigger wieder scharf geschalten werden muss. Die benötigte Zeit hängt dabei von der Auslastung des PCs bzw. dessen Reaktionszeit ab.

3. ETS Modus: Entspricht in etwa dem Block Modus mit dem Unterschied, dass Messwerte über mehrerer Perioden des gemessenen Signals aggregiert werden. Dadurch kann die effektive zeitliche Auflösung erhöht werden. Dies funktioniert aber logischerweise nur bei periodischen Signalen, welche sich absolut präzise wiederholen. Damit kann dieser Modus im vorliegenden Fall nicht eingesetzt werden. Außerdem ist eine Erhöhung der zeitlichen Auflösung im aktuellen Fall nicht erforderlich. Da bereits im Block Modus ein Abtastintervall von nur 2 ns (zwei aktive Kanäle) erzielt werden kann.

4. Rapid Block Modus: Die letzte Möglichkeit stellt der Rapid Block Modus dar. Dieser entspricht dem Block Modus mit dem Unterschied, dass die Daten nicht nach jeder Aufzeichnung an den PC übertagen werden. Die Daten werden zunächst nur im internen Speicher abgelegt. Damit kann nach dem Ende einer Aufzeichnung praktisch sofort auf das nächste Trigger-Ereignis gewartet werden. Eine Aufzeichnung kann in diesem Modus nur wenige Mikrosekunden nach der vorhergehenden gestartet werden. Auf diese Weise können je nach Länge einer einzelnen Aufzeichnung Tausende Aufzeichnungen im internen Speicher abgelegt und dann auf einmal an den PC übertragen werden. Dieser Modus scheint im vorliegenden Fall am besten geeignet. Er bietet keine relevanten Einschränkungen und hat den geringsten Overhead.

Nachdem die aktuellen Einstellungen auf stdout ausgegeben wurden, wird folglich die Aufzeichnung von Daten im Rapid Block Modus gestartet. Wie viele Aufzeichnungen am Stück durchgeführt werden

ist in der Konfigurations-Header-Datei hinterlegt. Wurde die gewünschte Anzahl an Aufzeichnungen durchgeführt, werden die Daten vom Oszilloskop abgerufen und ausgewertet. Bevor mit der Auswertung begonnen wird, wird jeweils schon die nächste Aufzeichnung gestartet. Damit erfolgen die Auswertung und Aufzeichnung parallel. Da die Auswertung besonders bei der geringen Datenmenge, welche im ersten Experiment benötigt wird, schneller abgeschlossen sein dürfte, also die Aufzeichnung, ist die Gesamtgeschwindigkeit nur durch das Experiment selbst und nicht durch die Auswertung der Messwerte beschränkt, womit es möglich sein sollte, eine große Anzahl an Messungen durchzuführen.

Die empfangenen Messwerte werden nach einem festen Schema in einem Puffer abgelegt. Für die Auswertung ist eine eigene Funktion zuständig. Diese läuft über den Puffer, also über die gemessenen Werte und sucht nach steigenden Flanken. Erwartet wird, dass zunächst eine steigende Flanke auf Kanal A und dann eine steigende Flanke auf Kanal B auftritt und sonst keine Änderungen vorhanden sind. Ist dies der Fall wird der Abstand der beiden steigenden Flanken berechnet. Erfüllen die gemessenen Signale nicht die Erwartung, wird die Aufzeichnung als fehlerhaft eingestuft und der Grund für die Einstufung, beispielsweise das Fehlen einer steigenden Flanke auf Kanal B wird notiert. Die Auswertungsfunktion kann später problemlos durch eine andere ausgetaucht werden um die Aufzeichnungen nach anderen Kriterien auszuwerten.

Konfiguration des Auswertungsprogramms

Die genaue Konfiguration des Auswertungsprogramms, welche in der Konfigurations-Header-Datei für das erste Experiment hinterlegt wurde, kann im Abschnitt 5.1.6 anhand der Ausgabe des Programms nachvollzogen werden. An dieser Stelle folgen einige Erläuterungen, warum bestimmte Einstellungen gerade so gewählt wurden, wie dies aus der Ausgabe ersichtlich ist.

Da nur die analogen Ports zum Einsatz kommen sollen, wurden auch nur diese aktiviert. Der Spannungsbereich wurde auf ±2 V festgelegt, was tatsächlich ±20 V entspricht, da die Tastköpfe auf ein Teilungsverhältnis von 10:1 eingestellt sind. Der analoge Offset wurde auf -1,2 V bzw. -12 V eingestellt. Damit wird das Eingabesignal, was zwischen 0 V und 24 V wechselt, für den Analog-Digital-Wandler des Oszilloskops so verschoben, dass es so aussieht als würde es zwischen -12 V und 12 V wechseln. Dadurch ist der Messbereich von ±20 V ausreichend. Würde man das Signal nicht verschieben, müsste der nächstgrößere Messbereich gewählt werden, was die Messgenauigkeit reduzieren würde. Der Trigger wurde so konfiguriert, dass er bei einer steigenden Flanke beim Nulldurchgang des Signals auf Kanal A auslöst. Fürs erste wurde ein Abtastintervall von 1 µs und dazu passend eine Anzahl von 400 Messungen vor und 7000 Messungen noch dem Trigger pro Aufzeichnung gewählt. Damit werden 400 µs vor jeder steigenden Flanke auf Kanal A und 7 ms nach jeder steigenden Flanke aufgezeichnet. Dies sollte reichen um die zugehörige steigende Flanke auf Kanal B zu erfassen und um mögliche unerwartete Signalwechsel in der unmittelbaren Nähe der steigenden Flanken zu erkennen. Der Signalgenerator wurde konfiguriert um ein Rechtecksignal mit 50 Hz zu produzieren. Für die Auswertung wurde festgelegt, dass ein Signal als high zu erkennen ist, wenn es 80 % seines Maximalwertes erreicht hat. Die 24 V Signale werden also ab etwa 19,2 V als high angesehen. Berücksichtigt man die Verschiebung um 12 V, entspricht dies 7,2 V, was etwa 36 % des eingestellten Messbereichs von 20 V sind. Ferner wurde eingestellt, dass ein Signal als low zu erkennen ist, wenn es 20 % seines Maximalwertes unterschritten hat. Damit besteht eine Hysterese von 60 %, was jegliche Störungen unterdrücken sollte.

5.1.4. Experimentdurchführung

Nach Abschluss der zuvor beschriebenen Vorbereitungen wurde der vorbereitete Raspberry Pi mit der gebauten Platine verbunden und das Testprogramm wurde mit „sudo ./loopback“ ausgeführt. Am Raspberry Pi waren zu diesem Zeitpunkt ein Bildschirm und eine Tastatur angeschlossen. Während

des Testlaufs bestand zunächst keine Netzwerkverbindung, die Steuerung erfolgte aus der Kommandozeile ohne grafische Oberfläche.

Die zuvor beschriebene Schaltung zur Erzeugung des Testsignals wurde an den Funktionsgenerator des Oszilloskops angeschlossen und der Testsignalausgang auf Eingabepin 0 von Port 1 gelegt. Der genutzte Ausgang 0 von Port 1 wurde mit einem 10 kΩ Widerstand belastet. Der analoge Kanal A des Oszilloskops wurde am Experimentierboard am Ausgang des Testsignals angeschlossen, welcher über eine kurze Leitung direkt mit dem Eingabepin 0 von Port 1 verbunden war. Der analoge Kanal B des Oszilloskops wurde direkt am Ausgabepin 0 von Port 1 angeschlossen. Beide Tastköpfe waren auf ein Teilungsverhältnis von 10:1 eingestellt.

Zu diesem Zeitpunkt wurden erste Visualisierungen des Signalverlaufs mit der normalen Oszilloskop-Software erstellt. Der Funktionsgenerator des Oszilloskops wurde mit Hilfe der normalen Software angewiesen ein 50 Hz Rechtecksignal zu erzeugen. Die Ergebnisse werden im nächsten Abschnitt präsentiert.

Danach wurde der Raspberry Pi heruntergefahren, der Bildschirm und die Tastatur wurden getrennt, ein Netzwerkkabel wurde angeschlossen und das Gerät wurde erneut hochgefahren. Die Steuerung erfolge ab diesem Zeitpunkt aus praktischen Gründen ausschließlich über SSH. Sofern nicht anders angegeben erfolgt ab diesem Zeitpunkt die Steuerung bei allen nachfolgenden Testläufen in allen nachfolgenden Anschnitten über SSH. Ein Bildschirm und eine Tastatur werden sofern nicht anders angegeben nicht angeschlossen.

Nun wurde zur genaueren Untersuchung der Reaktionszeit das entwickelte Auswertungsprogramm ausgeführt. Die Ergebnisse werden im Abschnitt 5.1.6 besprochen.

5.1.5. Visualisierung des Signalverlaufs

Abbildung 51 zeigt eine Visualisierung des Signalverlaufs während des Experiments. In der oberen Hälfte ist das Signal vom Kanal A, also das am Eingang anliegende Testsignal und in der unteren Hälfte ist das Signal vom Kanal B, also das Signal am Ausgabepin der SPS, dargestellt. Das Testsignal wechselt planmäßig alle 10 ms den Zustand. Es ist zu erkennen, dass im high Zustand etwas mehr als die vorgesehenen 24 V gemessen werden, dies liegt daran, dass das verwendete Netzteil eine leicht höhere Spannung geliefert hat. Dies stellt aber kein Problem dar. Die Spannung am Ausgabepin im high Zustand ist erwartungsgemäß minimal geringer als die Spannung des Testsignals im high Zustand. Dies wird durch den Spannungsabfall an der verbauten Schottky Diode verursacht. Die steigenden Flanken des Testsignals sind erwartungsgemäß sehr steil, während die fallenden Flanken minimal flacher ausfallen, da das Signal nur durch einen Pull-Down-Widerstand heruntergezogen wird.

Das Ausgabesignal folgt der Eingabe erwartungsgemäß mit einer geringen Verzögerung. Genau diese Verzögerung stellt die Reaktionszeit dar, die in diesem Experiment ermittelt werden soll. Die fallenden Flanken des Ausgabesignals sind recht flach, da der Ausgang nur leicht belastet ist. Abbildung 52 zeigt eine vergrößerte Darstellung des Bereichs um die steigenden Flanken der beiden Signale. Zur Erzeugung der Abbildung wurde das Oszilloskop so konfiguriert, dass ein Trigger-Ereignis beim Nulldurchgang des Testsignals erzeugt wird, so dass die steigende Flanke des Testsignals immer an der gleichen Stelle aufgezeichnet wird. Darüber hinaus wurde eingestellt, dass dargestellte Signalverläufe nicht gelöscht, sondern durch neue Überlagert werden. Die Messung wurde etwa 60 s ausgeführt. Es kann nicht mit Sicherheit gesagt werden, ob jede einzelne steigende Flanke während dieser Zeit erfasst werden konnte, da dies von der Reaktionszeit des verwendeten PCs abhängt, die Anzahl der überlagerten Messungen liegt aber in der Größenordnung von $50 \cdot 60 = 3000$.

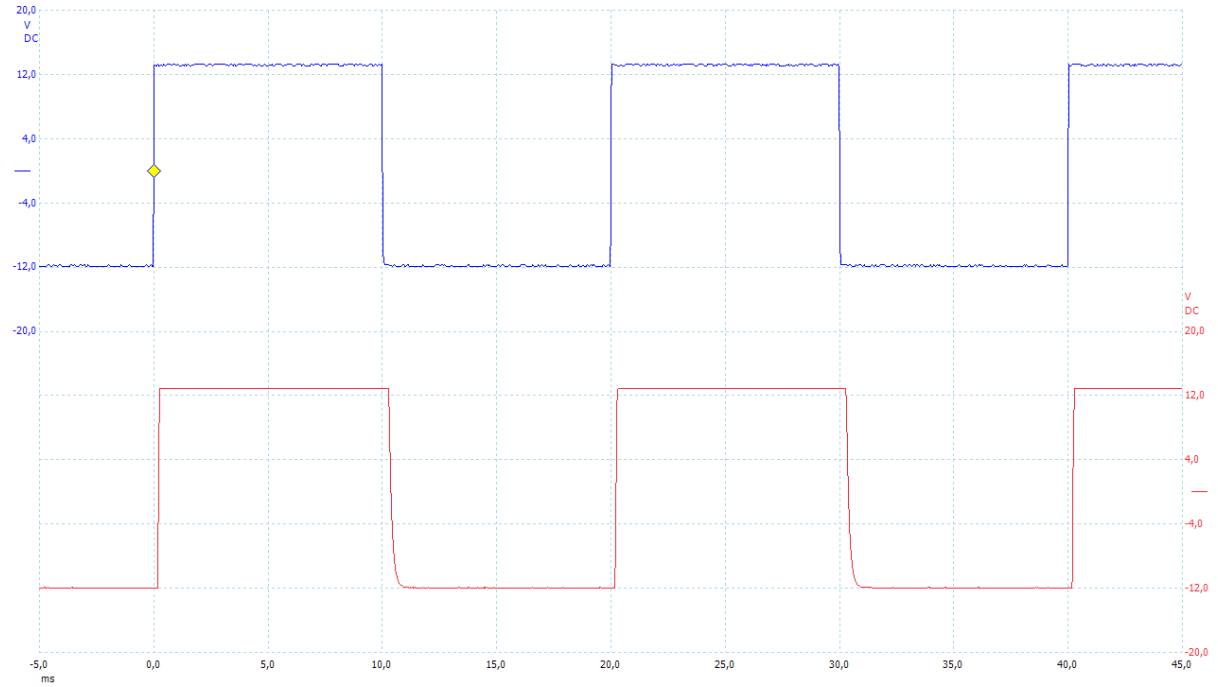


Abbildung 51: Visualisierung des Signalverlaufs bei der ersten Referenzmessung

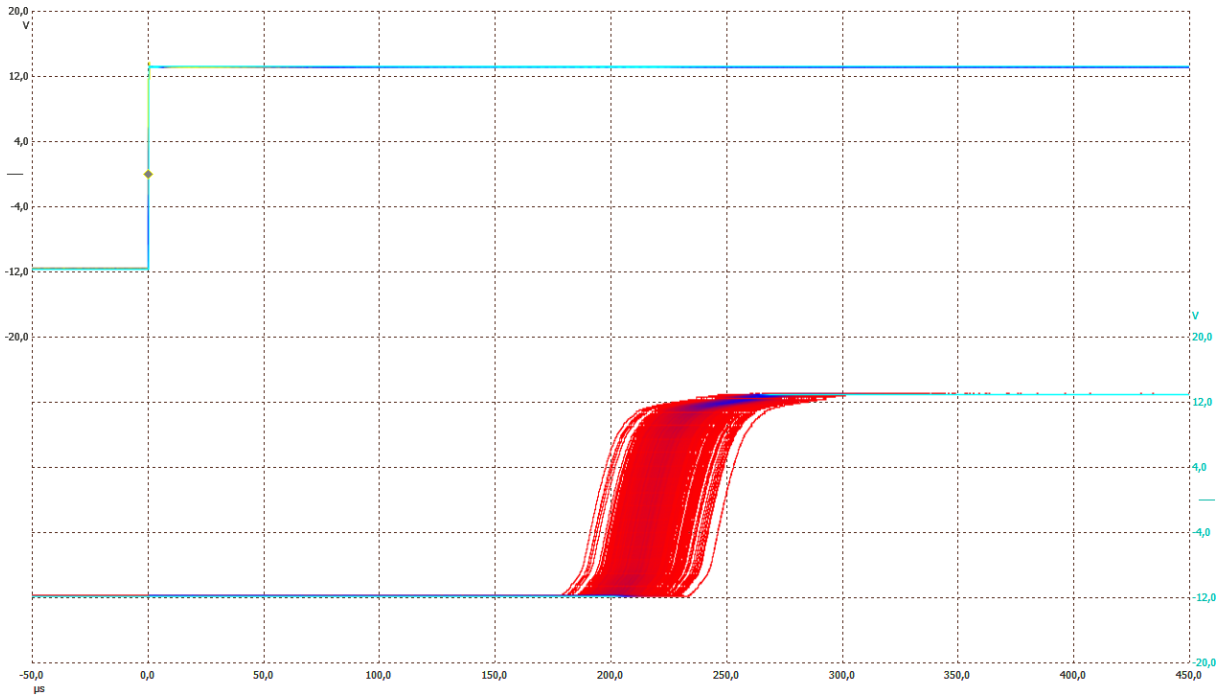


Abbildung 52: Überlagerung mehrerer Durchläufe der Referenzmessung

Es ist zu erkennen, dass die steigende Flanke auf der Ausgabeleitung etwa 220 µs nach der steigenden Flanke des Testsignals auftritt. Durch die Überlagerung vieler Durchläufe kann an dieser Stelle schon gut abgeschätzt werden, dass die Reaktionszeit einer Schwankung von etwa ±30 µs unterliegt. Wie zu erkennen ist, ist die steigende Flanke auf der Ausgabeleitung deutlich flacher als die des Testsignals. Dies ist ein beabsichtigtes Verhalten der Ausgabebausteine, das der Reduzierung von Störungen dient.

5.1.6. Messergebnisse

In diesem Abschnitt wird zunächst die Konfiguration des Auswertungsprogramms, welche für dieses Experiment genutzt wurde, gezeigt, da nur in Verbindung mit der genauen Konfiguration die Messergebnisse ausgewertet werden können. Anschließend wird die bei der Messung genutzte Quantisierung der Zeitintervalle im Detail untersucht um Rückschlüsse auf die Genauigkeit der Messergebnisse ziehen zu können. Abschließend werden dann die Messergebnisse ausgewertet und diskutiert.

Konfiguration des Auswertungsprogramms

Codeausschnitt 14 zeigt den Teil der Ausgabe des Auswertungsprogramms, welcher die aktive Konfiguration und die Anzahl an Fehlern bei der Auswertung wiedergibt. Wie zu erkennen ist, konnten alle aufgezeichneten Durchläufe erfolgreich verarbeitet werden, es sind keine Fehler aufgetreten. Es wurden jeweils 1000 Durchläufe am Stück im internen Speicher des Oszilloskops abgelegt, bevor diese an den PC übertrage und ausgewertet wurden. Dieser Vorgang wurde 100-mal wiederholt. So konnten 100.000 Messwerte für die Reaktionszeit des Geräts ermittelt werden. Dies erschien ausreichend um eine gute Einschätzung der Reaktionszeit und deren Schwankung vornehmen zu können und konnte in akzeptabler Zeit durchgeführt werden. Die Erhebung von 100.000 Messwerten dauerte nur etwas mehr als eine halbe Stunde und erfolgte voll automatisch.

```
Configuration:

Analog channel A:
  enabled:   Yes
  coupling:  DC
  range:     +-2 V
  offset:    -1.200000 V
  filter:    off

Analog channel B:
  enabled:   Yes
  coupling:  DC
  range:     +-2 V
  offset:    -1.200000 V
  filter:    off

Digital port 0:
  enabled:   No
  threshold: 1.599933 V

Digital port 1:
  enabled:   No
  threshold: 1.599933 V

Common settings:
  Sample intervall:     1000.000000 ns
  Number of segments:   1000
  Pre-trigger samples:  400
  Post-trigger samples: 7000
  Number of iterations: 100

Trigger settings:
  enabled:       Yes
  channel:       A
  threshold:     0.000000% ([-100%; 100%] of analog range selected)
  direction:     RISING
  delay:         0 samples

Signal generator setting:
```

```
  offset:        1000.000000 mV
  pk-to-pk:      2000.000000 mV
  wave form:     SQUARE
  frequency 1:   50.000000 Hz
  frequency 2:   50.000000 Hz
  sweep inc:     0.000000 Hz
  inc period:    1.000000 s
  sweep dir:     UP
  operation:     normal operation
  periods:       infinite
  trigger:       RISING
  trigger src:   immediate start
  ext threshold: 0 ([-32767; 32767])

Processing:

DT_AR_BR:
channel A positive going threshold: 35.999016% ([-100%; 100%] of analog range)
channel A negative going threshold: -35.999016% ([-100%; 100%] of analog range)
channel B positive going threshold: 35.999016% ([-100%; 100%] of analog range)
channel B negative going threshold: -35.999016% ([-100%; 100%] of analog range)

Results of dt_ar_br processing:
  total number of wave forms seen:                 100000
  successfully processed wave forms:               100000
  number of waveforms not processed due to errors: 0
    invalid condition at first sample:    0
    no rising edge found:                 0
    rising edge on B before A:            0
    no rising edge on channel B:          0
    falling edge on channel A detected:   0
    falling edge on channel B detected:   0
```

Codeausschnitt 14: Konfiguration des Auswertungsprogramms beim ersten Experiment

Genauigkeit der Messung

```
Time differences recorded:
Time difference in micro seconds; number of waveforms
195.00; 1
196.00; 0
197.00; 0
198.00; 1
199.00; 1
200.00; 1
201.00; 3
202.00; 2
203.00; 9
204.00; 13
205.00; 5
206.00; 21
207.00; 30
208.00; 37
209.00; 59
210.00; 108
211.00; 232
212.00; 401
213.00; 736
...
```

Codeausschnitt 15: Ausschnitt aus der Ausgabe des Auswertungsprogramms

In Codeausschnitt 15 wird exemplarisch der Anfang der Ergebnisausgabe des Auswertungsprogramms gezeigt. Wie zu erkennen ist, sind die Reaktionszeiten entsprechend des gewählten Abtastintervalls mit einer Auflösung von 1 µs angegeben. Die Auflösung der Ausgabe muss aber nicht zwangsläufig der Genauigkeit der Messung entsprechen. Um diese zu bestimmen, muss das Vorgehen bei der Quantisierung der Zeitintervalle betrachtet werden. Dieses wird exemplarisch in Abbildung 53 veranschaulicht.

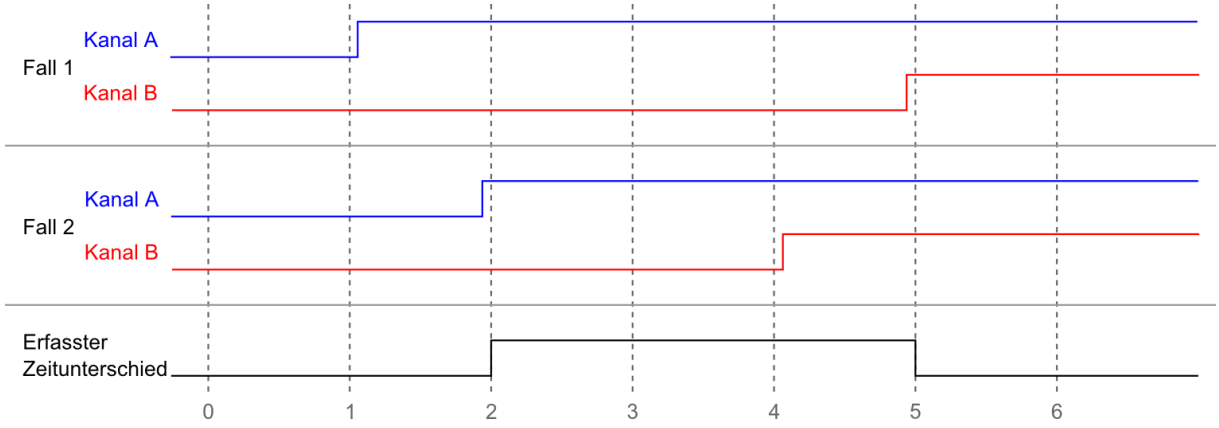


Abbildung 53: Veranschaulichung der Quantisierung der Reaktionszeit

In der Abbildung sind für zwei Durchläufe jeweils die Signal auf Kanal A und Kanal B vereinfacht dargestellt. Die vertikalen gestrichelten Linien in der Abbildung stellen die Abtastzeitpunkte dar. Im dargestellten Fall 1 beträgt die tatsächliche Reaktionszeit annähernd vier Abtastintervalle, im Fall 2 nur etwas mehr als zwei Abtastintervalle, dennoch wird der Zeitunterschied in beiden Fällen als drei Abtastintervalle quantisiert. Diese Ungenauigkeit ergibt sich aus dem genutzten Messverfahren selbst und ist nicht zu vermeiden. Die tatsächliche Reaktionszeit entspricht immer der ermittelten Reaktionszeit ±1 Abtastintervall. Die Ungenauigkeit kann also durch die Wahl des Abtastintervalls bestimmt werden. Da für dieses erste Experiment ein Abtastintervall von 1 µs gewählt wurde, kann die Abweichung bei jeder Messung bis zu ±1 µs betragen. Angesichts der Tatsache, dass Zeitintervalle im Bereich von 200 µs gemessen werden, die über etliche Mikrosekunden gestreut sind, wird diese Abweichung für dieses Experiment als tolerierbar eingestuft. Für spätere Experimente, bei denen voraussichtlich kürzere Zeitintervalle zu messen sein werden, ist eine Verkleinerung des Abtastintervalls in Betracht zu ziehen.

Auswertung der Messergebnisse

Eine Auswertung der gelieferten Messdaten ergab eine durchschnittliche Reaktionszeit von etwa 226 µs und eine Standardabweichung von 7,7 µs. Dies deckt sich in etwa mit den ersten Beobachtungen aus Abbildung 52.

Die ermittelten Reaktionszeiten werden in Abbildung 54 in Form eines Histogramms veranschaulicht. Für diese Abbildung und auch für Abbildung 55 wurden zur besseren Darstellbarkeit jeweils vier nebeneinander liegende Balken aggregiert. Im Unterschied zur ersten Abschätzung wird hier deutlich, dass, obwohl die allermeisten Messwerte (ca. 99 %) in einem Bereich von ±20 µs um den Durchschnittswert liegen, vereinzelt deutlich längere Reaktionszeiten von teils fast 500 µs auftreten.

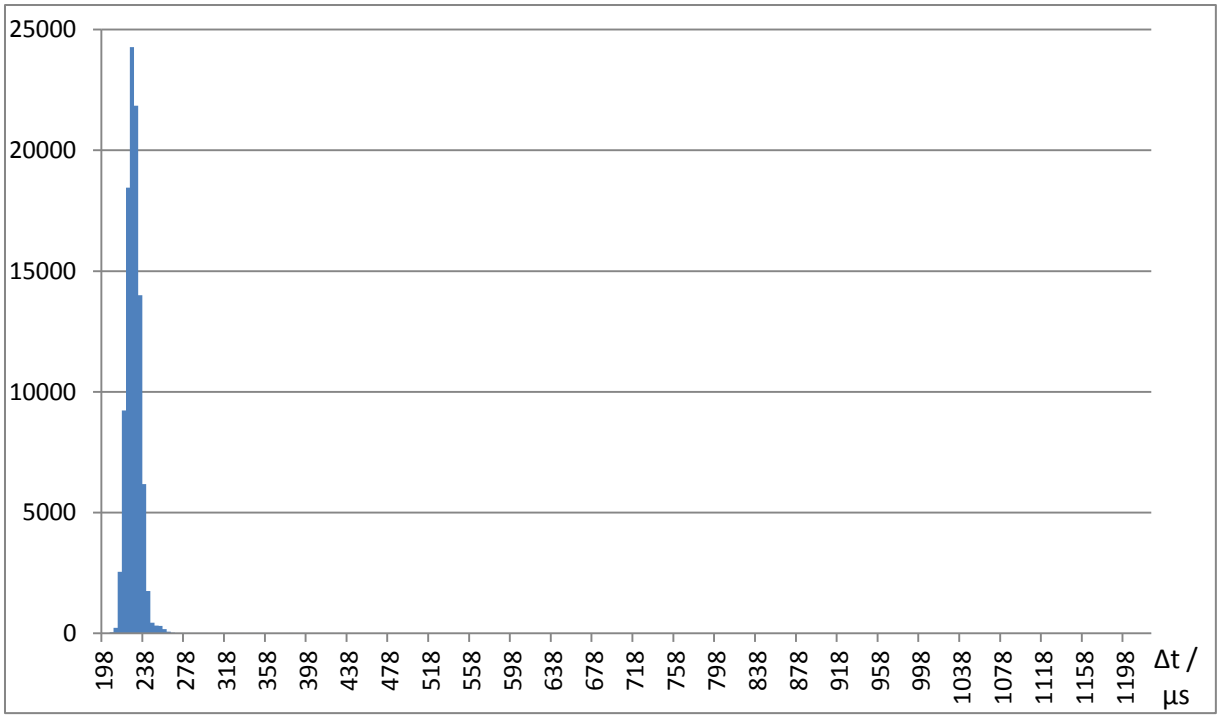


Abbildung 54: Histogramm der Reaktionszeiten bei der Referenzmessung

Abbildung 55 zeigt eine vergrößerte Darstellung des unteren Bereichs des Histogramms um diese selten auftretenden Reaktionszeiten besser darzustellen. Wie hier zu erkennen ist, wurde im Laufe des Experiments einmal sogar eine extrem lange Reaktionszeit von über 1 ms gemessen.

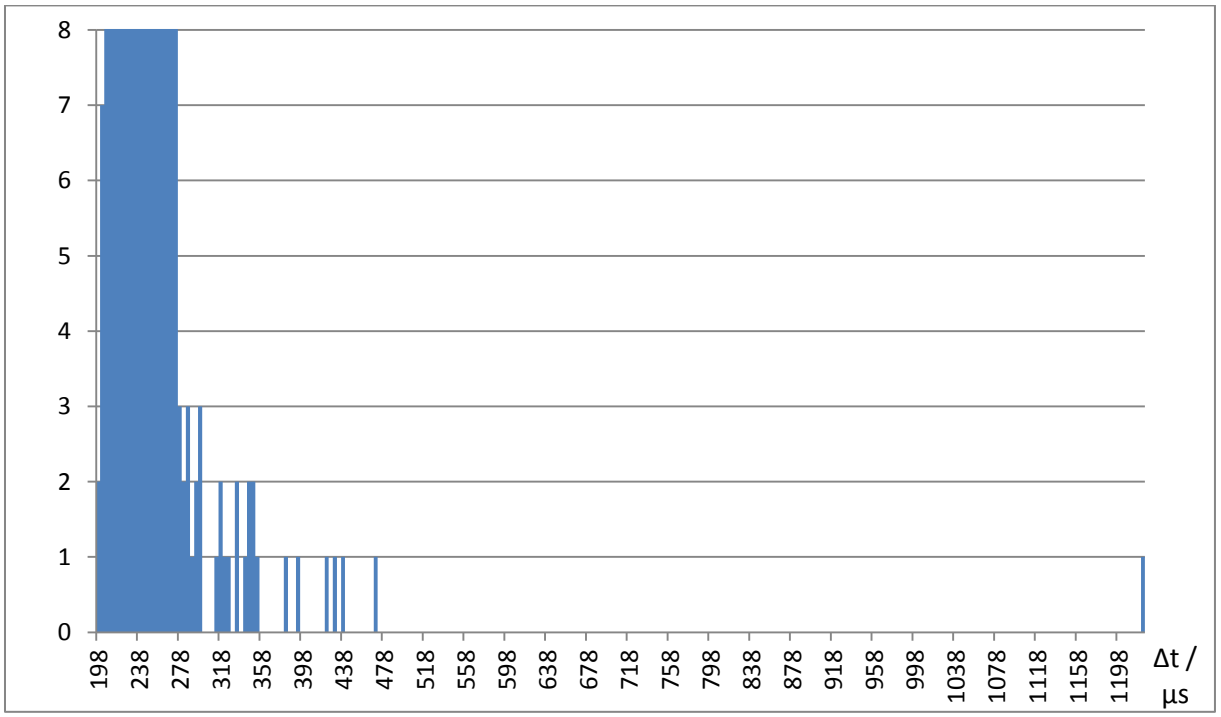


Abbildung 55: Detailansicht des unteren Bereichs des Histogramms zur Referenzmessung

Abbildung 56 zeigt eine vergrößerte Darstellung des linken Bereichs des Histogramms. In diesen Bereich befinden sich über 99,9 % der Messungen. Hier entspricht jeder Balken einer Zeile der Ausgabe des Auswertungsprogramms. Es wurde keine weitere Aggregation durchgeführt.

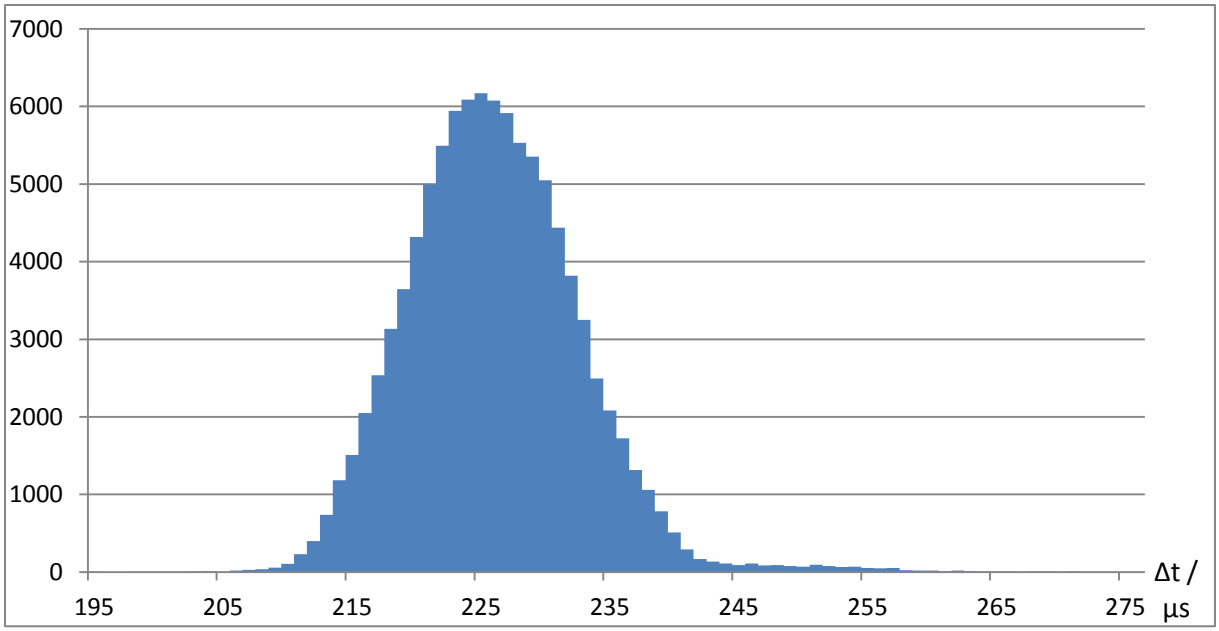


Abbildung 56: Detailansicht des linken Bereichs des Histogramms zur Referenzmessung

Das durchgeführte Experiment hat die gewünschten Erkenntnisse geliefert, es gibt jetzt Vergleichswerte für nachfolgende Experimente. Allerdings hat es auch neue Fragen aufgeworfen. Speziell die Frage, wodurch die gelegentlich auftretenden besonders langen Reaktionszeiten verursacht werden. Vom Experiment werden hierzu keine Hinweise geliefert. Dies ist gezielt durch weitere Experimente zu untersuchen.

5.2. Zusammensetzung der Reaktionszeit

Bisher wurde die Reaktionszeit von einer Eingabeänderung bis zur zugehörigen Ausgabeänderung ohne weitere Betrachtung der internen Abläufe untersucht. Nun soll im nächsten Schritt diese Reaktionszeit unter Berücksichtigung des internen Aufbaus genauer untersucht werden um festzustellen, wo Optimierungspotential besteht und wie weit sich die Reaktionszeit theoretisch verkürzen ließe. Teile der Reaktionszeit sind logischerweise hardwarebedingt, diese werden sich an dieser Stelle nicht mehr weiter optimieren lassen, wohingegen softwarebedingte Verzögerungen evtl. verkürzt werden können. Als Verbesserung der Reaktionszeit wäre auch eine Reduzierung der Steuerung anzusehen. Auch hier stellt sich die Frage, wie viel von der Steuerung hardwarebedingt ist und nicht weiter verbessert werden kann.

5.2.1. Fragestellung

Konkret soll in diesem Experiment untersucht werden, wie sich die Reaktionszeit zusammensetzt, wie lang die einzelnen Bestandteile sind und welcher Streuung die einzelnen Bestandteile unterliegen. Es soll insbesondere eine Aufschlüsselung der Gesamtreaktionszeit in hardware- und softwarebedingte Komponenten erfolgen. Eine genauere Betrachtung der Zusammensetzung softwarebedingter Verzögerungen soll in diesem Experiment nicht erfolgen.

5.2.2. Experimentdesign

Zur Untersuchung der einzelnen Bestandteile der Reaktionszeit soll das erste Experiment nochmal wiederholt werden. Dabei sollen diesmal aber nicht nur die Ein- und die Ausgabeleitung, sondern auch alle internen Leitungen überwacht werden, die an dem Vorgang beteiligt sind. Speziell die Verbindungsleitungen zwischen den einzelnen Bauteilen, die zur Übertragung der Information über die Ein-

gabeänderung zum Raspberry Pi und dann zum Übertragen der Information über die Ausgabeänderung vom Raspberry Pi zum Ausgang genutzt werden. Durch Überwachung dieser Leitungen soll festgestellt werden, wie lange die einzelnen Bauteile zur Verarbeitung benötigen, bevor sie die Information zum nächsten Baustein weiterleiten. Da alle beteiligten Leitungen höchstens einige Zentimeter lang sind und Reaktionszeiten im µs Bereich untersucht werden, kann die Ausbreitungsgeschwindigkeit der Signale auf den Leitungen von vornherein vernachlässigt werden.

Als erstes ist zu klären, welche Leitungen bzw. Signale überwacht werden sollen. Dazu muss der interne Signalverlauf nachvollzogen werden. Ein Eingabesignal wird zunächst von einem der Eingabebausteine ausgewertet. Erkennt dieser eine Änderung, wird diese zu einem der IO-Expander weitergeleitet. Der betroffene IO-Expander informiert über die zugehörige Interruptleitung den Raspberry Pi über das Vorhandensein einer Änderung. Dieser fragt dann über den I^2C Bus die Änderung ab, verarbeitet diese und setzt den Zustand eines GPIO Pins. Von diesem gelangt die Information, dass ein Ausgang geschaltet werden soll, zu einem der Ausgabebausteine. Dieser schaltet schließlich den 24 V Ausgang. Tabelle 17 zeigt die Signale, welche beteiligt sind, wenn eine Änderung am Eingabepin 0 von Port 1 erfolgt und der Ausgabepin 0 von Port 1 als Reaktion darauf geschaltet wird.

Signalbeschreibung	Signalname	Kanal
Signal am Eingabepin 0 von Port 1	IN_P1_0	A
zugehörige Ausgabe des Eingangsbausteins IC1	IN_P1_D_0	D0
Interruptleitung des zugehörigen IO-Expanders	INT1	D1
Datenleitung des I^2C Busses	I2C1_SDA	D2
Taktleitung des I^2C Busses	I2C1_SCL	D3
Steuerleitung der zugehörigen Ausgabe	D0	D4
Signal am Ausgabepin 0 von Port 1	OUT_P1_0	B

Tabelle 17: An einer Reaktion beteiligte Signale

Zur Überwachung der Ein- und Ausgabeleitungen, sollen wie beim ersten Experiment die analogen Kanäle des Oszilloskops zum Einsatz kommen, da dort die Spannungen zu hoch für die digitalen Eingänge sind und teils langsame Flanken auftreten. Die restlichen beteiligten Signale sollen mit digitalen Eingängen des genutzten Oszilloskops überwacht werden. Tabelle 17 zeigt ebenfalls die geplante Zuordnung von Kanälen zu Signalen.

Die Durchführung dieses Experiments soll in zwei Phasen erfolgen. In der ersten Phase soll nur eine Visualisierung der Signalverläufe mit der normalen Software des Oszilloskops erfolgen. Aufgrund der Erkenntnisse aus dieser Phase, soll dann das für das erste Experiment entwickelte Auswertungsprogramm erweitert werden um in der zweiten Phase eine genauere Auswertung mit diesem vornehmen zu können.

5.2.3. Experimentvorbereitung (Phase 1)

Die wesentliche Herausforderung bei der Vorbereitung der ersten Phase dieses Experiments besteht darin an allen benötigten Signalen geeignete Stellen zu finden, an denen das Signal abgegriffen werden kann. Praktischerweise stehen an nahezu allen betroffenen Signalen geeignete Stellen zur Verfügung, wo die analogen bzw. digitalen Tastköpfe des Oszilloskops angeschlossen werden können. Lediglich beim Signal IN_P1_D_0 besteht keine passende Anschlussmöglichkeit. Daher wurde für dieses Experiment an der Stelle, wo das Signal in den Schutzwiderstand geht, zusätzlich ein kleines Stück Draht angelötet, an welches ein digitaler Tastkopf angeschlossen werden kann. Tabelle 18 zeigt eine Übersicht der Anschlussstellen für alle Signale.

Signalname	Stelle für Abgriff
IN_P1_0	Ausgang des Testsignals am Experimentierboard
IN_P1_D_0	Schutzwiderstand
INT1	Verbindungsleitung zum Raspberry Pi
I2C1_SDA	Erweiterungsschnittstelle SDA Pin
I2C1_SCL	Erweiterungsschnittstelle SCL Pin
D0	Verbindungsleitung zum Raspberry Pi
OUT_P1_0	Ausgabestecker von Port 1

Tabelle 18: Anschlussstellen für Tastköpfe

5.2.4. Experimentdurchführung (Phase 1)

Die Durchführung der ersten Phase des Experiments erfolgte analog zur Durchführung des allerersten Experiments. Die mit Raspbian, der Steuerungsbibliothek und dem Testprogramm bestückte SD-Karte und die Schaltung zur Erzeugung eines Testsignals vom ersten Experiment konnten für dieses Experiment wiederverwendet werden. Die Schaltung zur Erzeugung des Testsignals wurde wieder am Oszilloskop und am Eingabepin 0 von Port 1 angeschlossen. Der Ausgang wurde wieder mit einem 10 kΩ Widerstand belastet. Die analogen und digitalen Tastköpfe wurden gemäß Tabelle 17 und Tabelle 18 angeschlossen. Das Testprogramm loopback wurde über SSH gestartet. Danach wurde der Funktionsgenerator des Oszilloskops angewiesen ein 50 Hz Rechtecksignal zu produzieren und der Signalverlauf wurde betrachtet. Die erzeugten Darstellungen des Signalverlaufs werden im nachfolgenden Abschnitt präsentiert. Zur Erzeugung einiger detaillierterer Darstellungen war es notwendig die analogen Kanäle des Oszilloskops an andere Signalleitungen anzuschießen. Dies wird in den entsprechenden Unterabschnitten, welche diese Darstellungen behandeln, beschrieben.

5.2.5. Visualisierung des Signalverlaufs

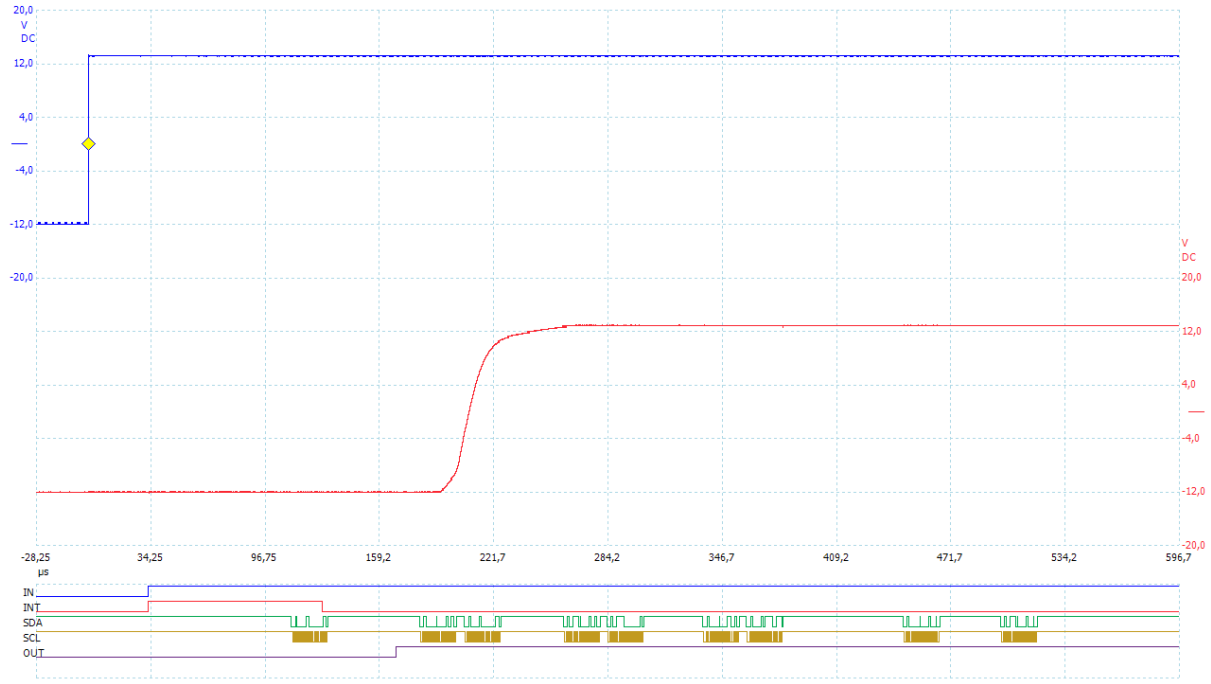


Abbildung 57: Signalverlauf auf allen beteiligten Leitungen

Abbildung 57 zeigt den ermittelten Signalverlauf auf allen beteiligten Leitungen. Ab diesem Zeitpunkt werden zur Vereinfachung statt der langen Signalnamen, welche im Schaltplan verwendet werden, die in Tabelle 19 aufgelisteten und in der Abbildung genutzten Kürzel verwendet. In der Abbildung ist oben gut die steigende Flanke des Testsignals zu erkennen. Die Reaktion auf der Ausgabeleitung er-

folgt, wie beim ersten Experiment, erwartungsgemäß in etwa 0,2 ms. Das IN Signal ist das erste zugängliche Signal mit 3,3 V Logik, dass die Änderung auf der Eingabeleitung wiederspiegelt. Die steigende Flanke auf diesem Signal kommt hier in knapp 35 µs nachdem die Änderung auf der Eingabe erfolgt ist. Die genaue Verzögerung und Streuung sind in der zweiten Phase dieses Experiments zu untersuchen.

Signalname	Kürzel
IN_P1_0	-
IN_P1_D_0	IN
INT1	INT
I2C1_SDA	SDA
I2C1_SCL	SCL
D0	OUT
OUT_P1_0	-

Tabelle 19: Kürzel für Signalnamen

Die Zeit, die benötigt wird um nach der Änderung des IN Signals das Interruptsignal zu erzeugen, ist im Vergleich zu allen anderen Verzögerungen offensichtlich extrem kurz und wurde daher noch in dieser Phase des Experiments genauer betrachtet. Eine genauere Darstellung findet sich weiter unten.

Nachdem das Interruptsignal ausgelöst wurde, wird die Änderung vom Raspberry Pi über den I^2C Bus abgefragt. Die Übertragung über diesen ist auf den SDA und SCL Leitungen zu erkennen. Die Zeit, die zwischen Interruptauslösung und der Datenübertragung vergeht, ist in der zweiten Phase des Experiments genauer zu untersuchen.

Abbildung 58 zeigt eine vergrößerte Darstellung der Datenübertragung. Wie zu erkennen ist, wird das Interruptsignal planmäßig gegen Ende der Übertragung deaktiviert, da das INTCAP Register ausgelesen wurde. Die Dauer der Datenübertragung und deren Schwankung werden weiter untern nochmal genauer betrachtet.

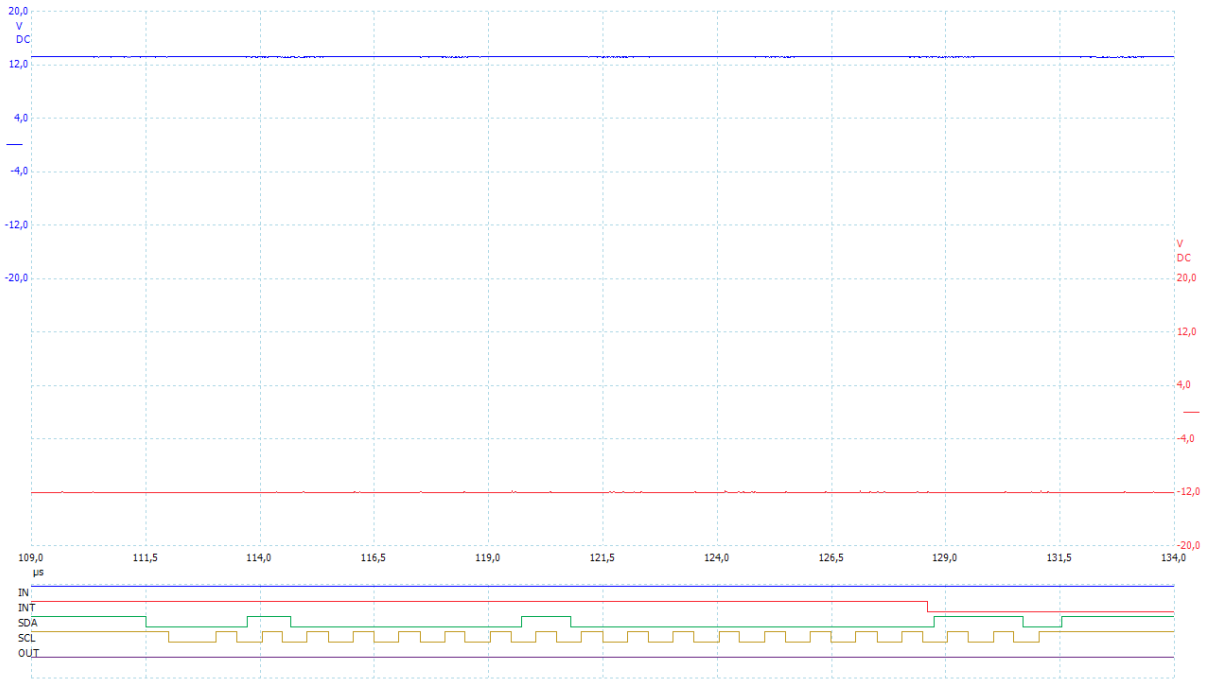


Abbildung 58: Vergrößerte Darstellung der Datenübertragung über den I^2C Bus

Nachdem die Änderung vom Raspberry Pi eingelesen wurde, wird diese verarbeitet und eine Änderung an der OUT Leitung erzeugt. Die dafür benötigte Zeit ist in der zweiten Phase des Experiments zu untersuchen.

Das Signal auf der OUT Leitung weist den Ausgabebaustein an, einen Ausgang zu schalten. Die Zeit die zwischen der Änderung auf der OUT Leitung und dem Schalten des Ausgangs vergeht, ist ebenfalls in der zweiten Phase des Experiments zu untersuchen.

Nach der ersten Übertragung sind in Abbildung 57 noch weitere Datenübertragungen über den I^2C Bus zu erkennen. Diese stellen Vorbereitungen für den nächsten Durchlauf dar und sind für dieses Experiment nicht von Bedeutung. Bei den ersten drei weiteren Übertragungen wird der aktuelle Zustand aller Eingänge der IO-Expander geprüft, wie dies im Abschnitt 4.8.9 beschrieben wird. Mit den letzten zwei Übertragungen wird der interne Adresszeiger der IO-Expander so gesetzt, dass er auf das INTCAP Register zeigt. Dadurch kann beim nächsten Durchlauf auf die Übertragung der Adresse des Registers beim ersten Lesevorgang verzichtetet werden, wodurch die Reaktionszeit verkürzt wird. Dies wird ebenfalls im Abschnitt 4.8.9 beschrieben.

Zeit zwischen den Flanken auf IN und INT

Um eine genauere Darstellung des Signalverlaufs, speziell der Verzögerung zwischen der steigenden Flanke auf der IN Leitung und der steigend Flanke auf der INT Leitungen zu erhalten, wurde der analoge Kanal A des Oszilloskops an die IN Leitung und der analoge Kanal B an die INT Leitung angeschlossen. Der Persistenzmodus der Oszilloskopsoftware, welcher zur ersten Einschätzung der Veränderung eines Signals über mehrere Durchläufe hinweg meist gut geeignet ist und bereits zur Erstellung der Abbildung 52 genutzt wurde, konnte hier leider nicht zum Einsatz kommen, da dieser nicht in der Lage war Signale in einem derart kleinen Zeitbereich sinnvoll darzustellen. Daher wurden einige Durchläufe aufgezeichnet und anschließend mit einem Grafikbearbeitungsprogramm manuell überlagert. Abbildung 59 zeigt die Überlagerung von 130 Durchläufen.

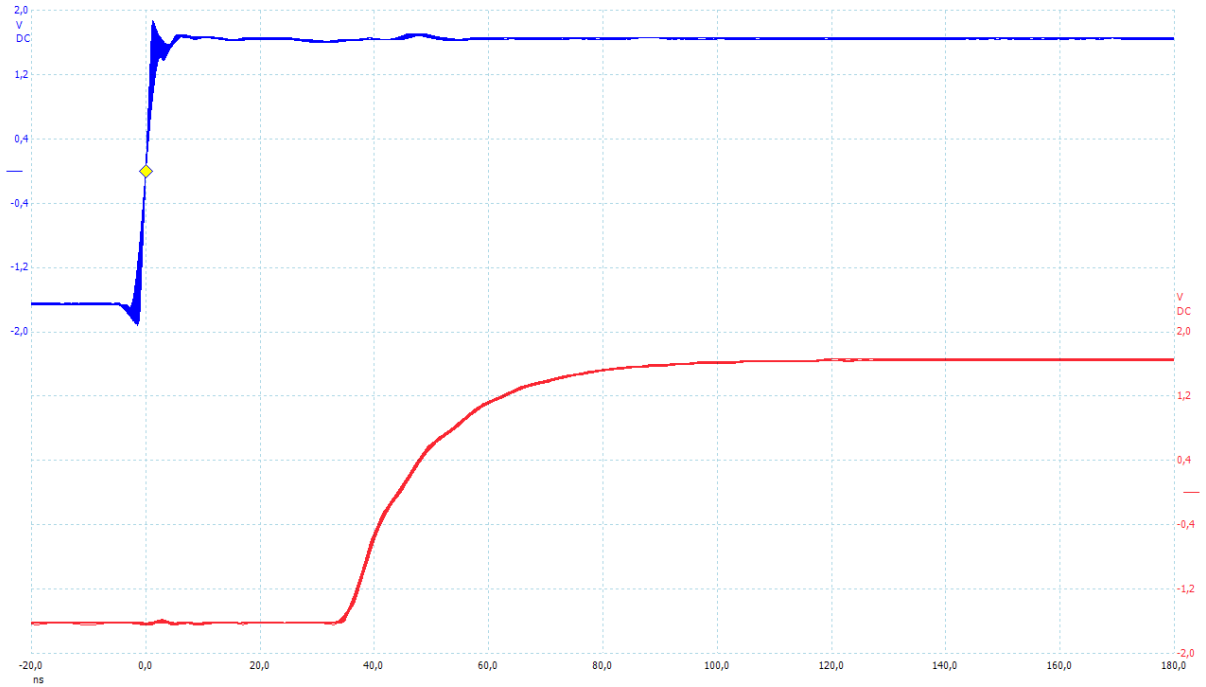


Abbildung 59: Detailansicht der Verzögerung zwischen dem IN und INT Signal

Wie zu erkennen ist, beträgt die Verzögerung etwa 52 ns, gemessen bei 70 % der Amplitude. Die Änderung der Verzögerung von Durchlauf zu Durchlauf liegt unterhalb des kleinsten verfügbaren Abtast-

intervalls von 2 ns und ist damit im Rahmen dieses Experiments vollkommen vernachlässigbar, da die anderen gemessenen Verzögerungen deutlich stärkeren Veränderungen unterliegen.

Auf beiden gemessenen Leitungen sind Schutzwiderstände verbaut, die das Gerät gegen Bedienungsfehler schützen sollen. Es fällt unweigerlich auf, dass die steigende Flanke auf Kanal B deutlich flacher ausfällt als die auf Kanal A. Dies liegt daran, dass bei Kanal B die Messung hinter dem Schutzwiderstand erfolgt, während auf Kanal A vor dem Schutzwiderstand gemessen wird. Das INT Signal wird durch den Eingang des Raspberry Pi, durch parasitäre Kapazitäten der Leitungen und nicht zuletzt auch durch den Tastkopf des Oszilloskops belastet. Da der Schutzwiderstand den Stromfluss beschränkt, ergibt sich eine flachere steigende Flanke. Ohne Tastkopf müsste die Flanke etwas steiler ausfallen.

Datenübertragung über I^2C

Zur Bestimmung der Dauer der Datenübertragung über den I^2C Bus wurde der analoge Kanal B an die SCL Leitung des I^2C Bus angeschlossen. Getriggert wurde bei der ersten fallenden Flanke. Um tatsächlich immer nur die erste Datenübertragung eines Durchlaufs aufzuzeichnen, wurde der Trigger zusätzlich zu konfiguriert, dass er nur auslöst, wenn das Signal auf der INT Leitung high ist. Diese wurde mit dem digitalen Kanal überwacht, der zu diesem Zweck auch schon im früheren Verlauf dieses Experiments genutzt wurde. Es ist zu beachten, dass damit nicht nur die Datenübertragung, welche steigenden Flanken auf der Eingabe folgt, sondern auch die Datenübertragung, welche fallenden Flanken folgt, aufgezeichnet wurde. Für die Messung der Übertragungszeit macht dies aber keinen Unterschied da in beiden Fällen die gleiche Anzahl an Bits übertragen wird und hier nur die Taktleitung überwacht wird.

An dieser Stelle konnte wieder der Persistenzmodus des Oszilloskops zum Einsatz kommen. Die Messung wurde etwa 60 s laufen gelassen. Um Störungen, welche durch die Induktivität der Messleitungen entstanden, zu unterdrücken wurde bei dieser Messung ein 20 MHz Tiefpassfilter zugeschalten. Das Ergebnis wird in Abbildung 60 dargestellt. Wie zu erkennen ist, dauert die Übertragung bei jedem Durchlauf ziemlich genau 19,0 µs. Eine Änderung der Dauer von Durchlauf zu Durchlauf ist in der Abbildung nicht zu erkennen. Diese wird daher als vernachlässigbar eingestuft.

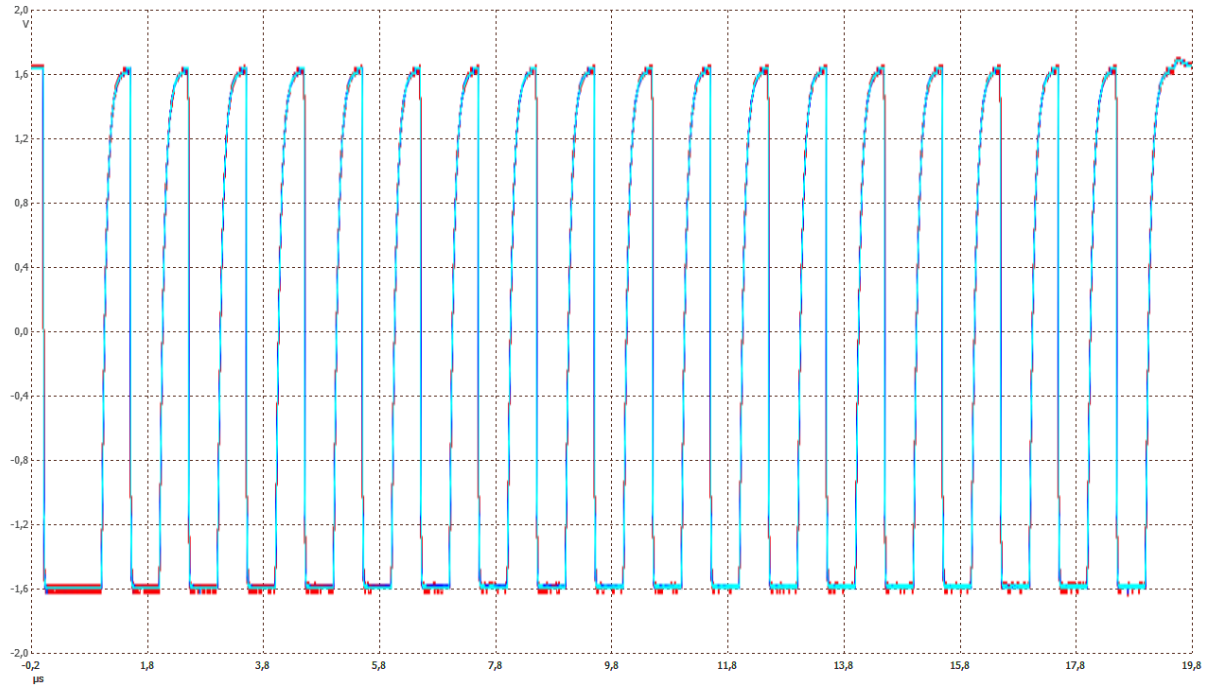


Abbildung 60: Überlagerung der Signale auf der SCL Leitung von mehreren Durchläufen

5.2.6. Experimentvorbereitung (Phase 2)

In der ersten Phase des Experiments konnten die Größenordnungen der einzelnen Verzögerungszeiten, die in diesem Experiment zu bestimmen sind, abgeschätzt werden. Die Angaben sind in Tabelle 20 zusammengefasst. Nun sollen, wie schon im ersten Experiment, Histogramme für die einzelnen Verzögerungszeiten erstellt werden um deren jeweilige Durchschnittswerte und die Stärke der Streuung um diese zu bestimmen. In Tabelle 20 ist zu erkennen, dass eine der Verzögerungszeiten im Nanosekundenbereich liegt, während alle anderen im Mikrosekundenbereich liegen. Um die Dauer der kürzesten Verzögerung sinnvoll zu bestimmen, müssten die Messungen mit einer extrem hohen zeitlichen Auflösung erfolgen. Dies ist aber gar nicht erforderlich, da die Verzögerung zwischen der steigenden Flanke auf dem IN Signal und der steigenden Flanke auf dem INT Signal bereits hinreichend untersucht wurde. Es ist bekannt, dass sie etwa 52 ns beträgt und sofern es dieses Experiment betrifft, effektiv keinen Schwankungen unterliegt.

Da die Zeit zwischen IN und INT um drei Zehnerpotenzen unterhalb aller andern zu messenden Zeiten liegt wurde beschlossen diese im weiteren Verlauf dieses Experiments vollständig zu vernachlässigen und gar nicht zu messen. Damit können die Messungen mit einer geringeren Genauigkeit durchgeführt werden, was die zu verarbeitende Datenmenge deutlich reduziert.

Zeit von ...	bis ...	Größenordnung
steigende Flanke auf A	steigende Flanke auf IN	35 µs
steigende Flanke auf IN	steigende Flanke auf INT	52 ns
steigende Flanke auf INT	Anfang SCL	80 µs
Anfang SCL	Ende SCL	19 µs
Ende SCL	steigende Flanke auf OUT	40 µs
steigende Flanke auf OUT	steigende Flanke auf B	50 µs

Tabelle 20: Größenordnungen der Verzögerungszeiten im zweiten Experiment

In der zweiten Phase dieses Experiments soll das Auswertungsprogramm, welches für das erste Experiment implementiert wurde, wieder zu Einsatz kommen. Dazu wurde es entsprechend erweitert um die gewünschten Daten ermitteln zu können. Das Sammeln von Messwerten von digitalen Eingängen wurde bereits in der Vorbereitungsphase des ersten Experiments implementiert, da zu erwarten war, dass dies später erforderlich sein wird. Also mussten als Vorbereitung auf die zweite Phase dieses Experiments die digitalen Kanäle in der Konfiguration nur noch aktiviert werden. Darüber hinaus wurde eine Auswertungsfunktion ergänzt, die die nötigen Messungen vornimmt und eine Ausgabefunktion, die die Ergebnisse in geeigneter Weise ausgibt um daraus Histogramme erstellen zu können.

Im ersten Experiment wurden Reaktionszeiten im Bereich von 200 µs gemessen. Die gewählte Abtastrate von 1 µs erwies sich dabei als geeignet. Aus Tabelle 20 wird ersichtlich, dass in diesem Experiment Zeitintervalle zu erwarten sind, die bis zu um den Faktor 10 kürzer sind. Entsprechend wurde das Abtastintervall um etwas mehr als den Faktor 10 auf 80 ns verkürzt.

5.2.7. Experimentdurchführung (Phase 2)

Die Durchführung der zweiten Phase dieses Experiments verlief genau, wie die erste Phase, mit dem Unterschied, dass anstelle der Visualisierung das erweiterte Auswertungsprogramm gestartet wurde.

5.2.8. Messergebnisse

In diesem Abschnitt werden die Messergebnisse präsentiert und diskutiert. Begonnen wird wieder mit der Konfiguration des Auswertungsprogramms.

Konfiguration des Auswertungsprogramms

```
Configuration:

Analog channel A:
  enabled:   Yes
  coupling:  DC
  range:     +-2 V
  offset:    -1.200000 V
  filter:    off

Analog channel B:
  enabled:   Yes
  coupling:  DC
  range:     +-2 V
  offset:    -1.200000 V
  filter:    off

Digital port 0:
  enabled:   Yes
  threshold: 1.649983 V

Digital port 1:
  enabled:   No
  threshold: 1.649983 V

Common settings:
  Sample intervall:     80.000000 ns
  Number of segments:   1000
  Pre-trigger samples:  1000
  Post-trigger samples: 62500
  Number of iterations: 100

Trigger settings:
  enabled:       Yes
  channel:       A
  threshold:     0.000000% ([-100%; 100%] of analog range selected)
  direction:     RISING
  delay:         0 samples

Signal generator setting:
  offset:        1000.000000 mV
  pk-to-pk:      2000.000000 mV
  wave form:     SQUARE
  frequency 1:   50.000000 Hz
  frequency 2:   50.000000 Hz
  sweep inc:     0.000000 Hz
  inc period:    1.000000 s
  sweep dir:     UP
  operation:     normal operation
  periods:       infinite
  trigger:       RISING
  trigger src:   immediate start
  ext threshold: 0 ([-32767; 32767])

Processing:

DT_HW:
channel A positive going threshold: 35.999016% ([-100%; 100%] of analog range)
channel A negative going threshold: -35.999016% ([-100%; 100%] of analog range)
channel B positive going threshold: 35.999016% ([-100%; 100%] of analog range)
channel B negative going threshold: -35.999016% ([-100%; 100%] of analog range)
```

```
Results of dt_hw processing:
  total number of wave forms seen:                100000
  successfully processed wave forms:              100000
  number of waveforms not processed due to errors: 0
    invalid condition at first sample:    0
    no change at all:                     0
    some change before rising edge on A:  0
    no INT signal:                        0
    some unexpected change before INT:    0
    no falling edge on SCL line:          0
    unexpected change before first fSCL:  0
    no rising edge on SCL line:           0
    unexpected change before first rSCL:  0
    not enough falling edges on SCL:      0
    unexpected change before some fSCL:   0
    not enough rising edges on SCL:       0
    unexpected change before some rSCL:   0
    no rising edge on OUT line:           0
    unexpected change before rising OUT:  0
    no rising edge on channel B:          0
    unexpected change before rising B:    0
    unexpected change after rising B:     0
```

Codeausschnitt 16: Konfiguration des Auswertungsprogramms beim zweiten Experiment

Codeausschnitt 16 zeigt den Teil der Ausgabe des Auswertungsprogramms, welcher die Konfiguration beinhaltet und anzeigt, ob Fehler aufgetreten sind. Wie zu erkennen ist, konnten hier alle 100.000 Durchläufe erfolgreich verarbeitet werden. Weiter oben ist zu erkennen, dass auch bei diesem Experiment jeweils 1.000 Durchläufe am Stück durchgeführt wurden, bevor die Messwerte an den PC übertragen wurden. Pro Durchlauf wurden 63.500 Messwerte aufgenommen. Davon wurden 1.000 Messwerte vor jedem Triggerereignis und 62.500 nach jedem Trigger aufgenommen. Somit wurden nach jedem Trigger, also jeweils nach jeder steigenden Flanke des Testsignals, 5 ms lang alle Signale überwacht. Ein längerer Zeitraum war offensichtlich nicht erforderlich, da alle Reaktionen innerhalb dieses Zeitraums erfolgten. Ein wesentlich längerer Zeitraum wäre auch nicht möglich gewesen, da dies den bei drei aktiven Kanälen (8 digitale Eingänge zählen jeweils wie ein analoger Kanal) pro Kanal verfügbaren Speicherplatz bereits fast vollständig ausnutzt. Es wäre sonst nötig gewesen die Anzahl der Durchläufe, welche am Stück ohne Übertragung an den PC ausgeführt werden, zu reduzieren.

Verzögerung zwischen den steigenden Flanken auf Kanal A und Kanal B

Die Verzögerung zwischen den steigenden Flanken auf Kanal A und Kanal B stellt die Gesamtreaktionszeit, welche bereits im ersten Experiment untersucht wurde, dar. Diese konnte in diesem Experiment ohne zusätzlichen Aufwand problemlos mitbestimmt werden. Der wesentliche Unterscheid zum ersten Experiment besteht darin, dass hier mit einer zehnmal höheren zeitlichen Auflösung gemessen wurde. Insgesamt decken sich die Ergebnisse mit denen aus dem ersten Experiment. Diesmal wurde eine leicht kürzere durchschnittliche Reaktionszeit von etwa 224,2 µs ermittelt. Dafür war die Standardabweichung mit etwa 8,1 µs leicht höher. Abbildung 61 zeigt das zugehörige Histogramm. Abbildung 62 zeigt wieder eine vergrößerte Darstellung des unteren Bereichs. In beiden Fällen wurden die ermittelten Messergebnisse im Verhältnis 10:1 aggregiert, also jeweils 10 nebeneinanderliegende Balken zusammengefasst, um eine bessere Darstellung zu ermöglichen. Abbildung 63 zeigt einen Bereich um den Durchschnittswert in dem über 99 % der Messwerte liegen. Dieser ist im Verhältnis 1:1, also ohne vorherige Aggregation, dargestellt. Es sind keine Auffälligkeiten erkennbar.

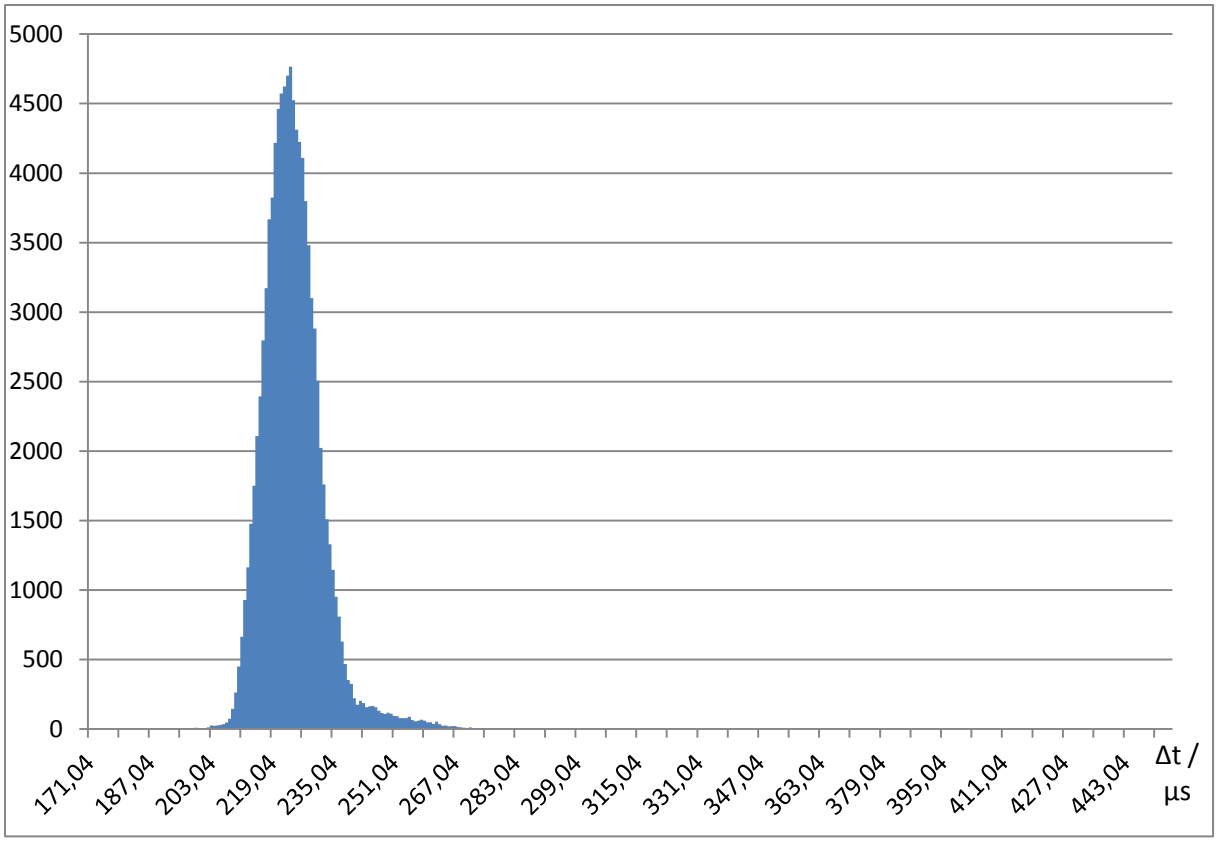


Abbildung 61: Histogramm der Reaktionszeiten beim zweiten Experiment

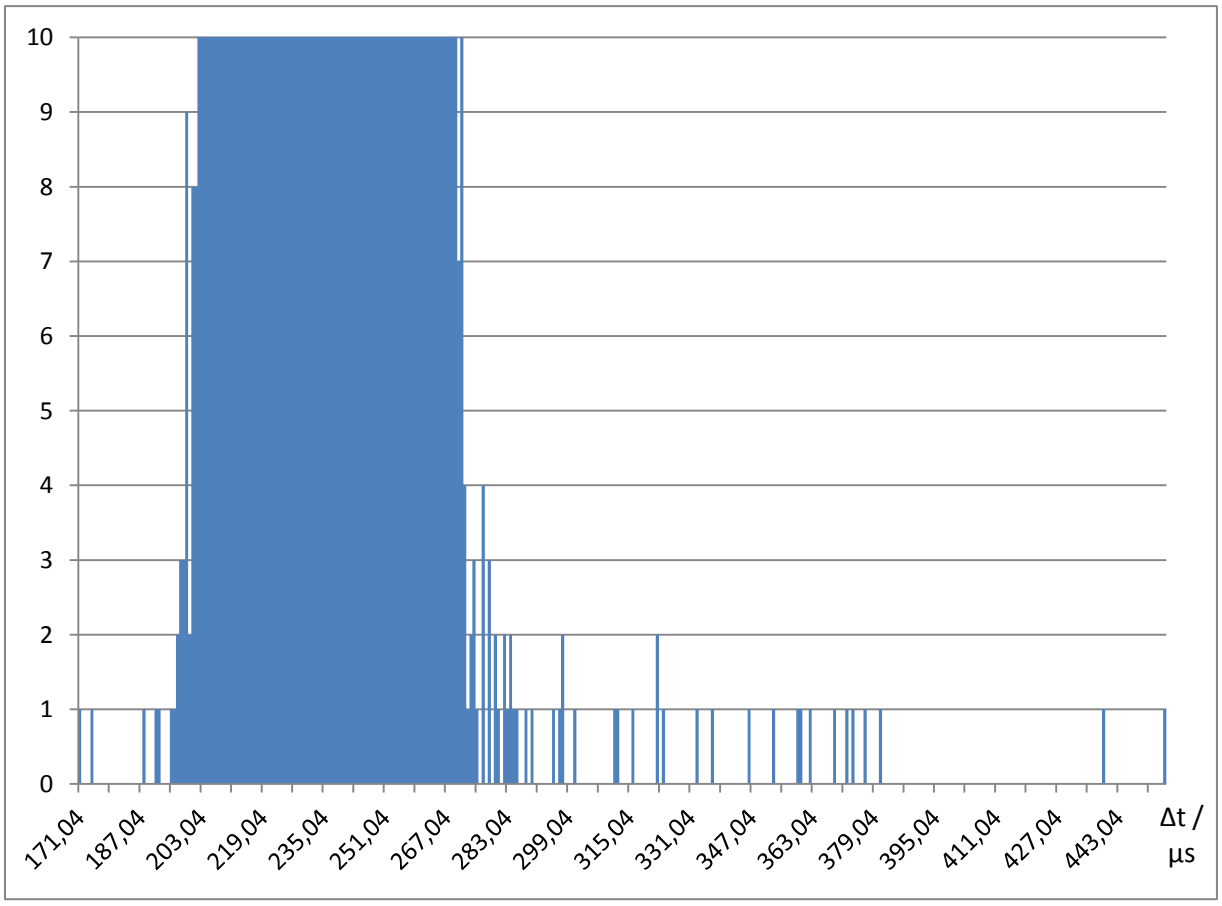


Abbildung 62: Detailansicht des unteren Bereichs des Histogramms aus Abbildung 61

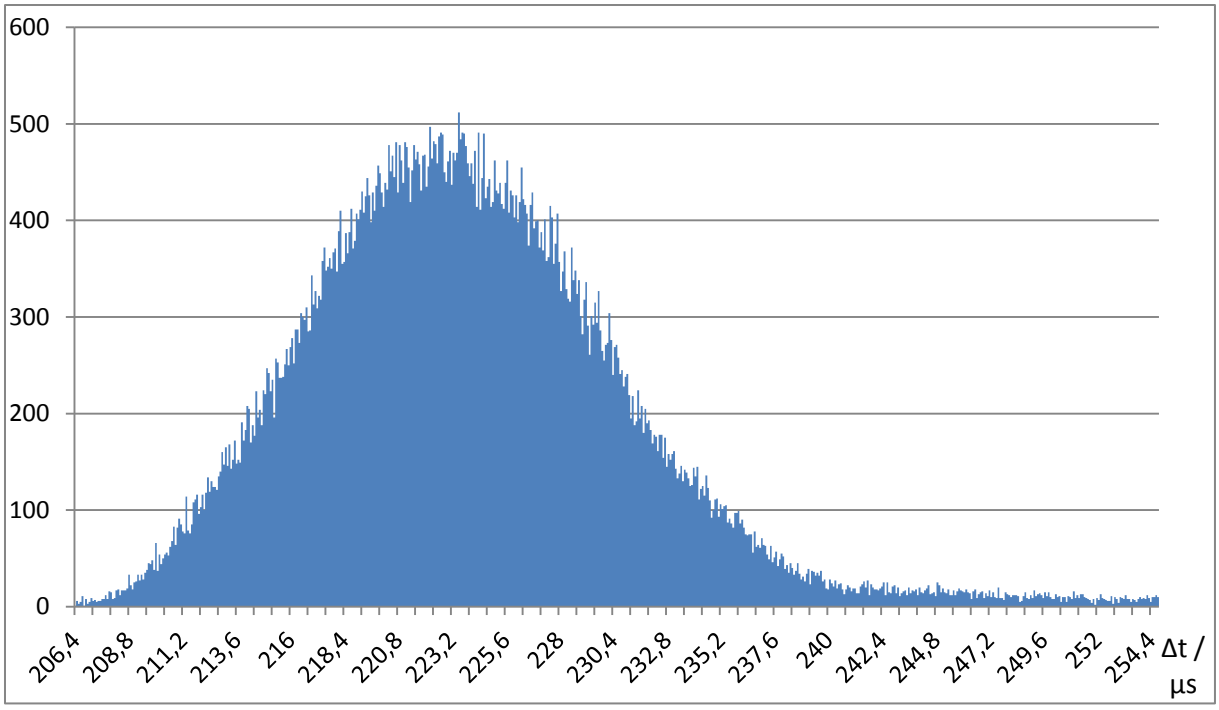


Abbildung 63: Detailansicht des Histogramms aus Abbildung 61

Verzögerung zwischen der steigenden Flanke auf Kanal A und dem INT Signal

Wie zuvor beschrieben wurde beschlossen die Zeit zwischen dem IN Signal und dem INT Signal, welche im Nanosekundenbereich liegt, zu ignorieren. Daher wurde gleich die Zeit zwischen der steigenden Flanke auf Kanal A und dem INT Signal gemessen.

Für diese Verzögerung wurde ein Durchschnittswert von etwa 36,8 µs ermittelt. Dem zugehörigen Histogramm in Abbildung 64 ist zu entnehmen, dass die Messergebnisse zwischen der kürzesten ermittelten Zeit von 31,6 µs und der längsten von 42 µs nahezu gleich verteilt sind.

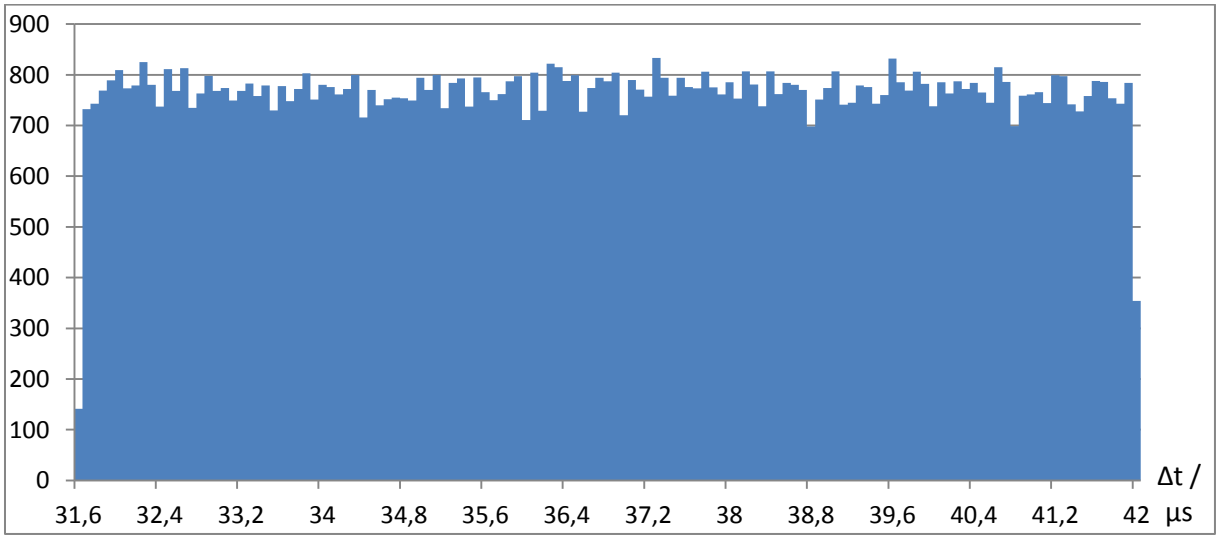


Abbildung 64: Verzögerung zwischen der steigenden Flanke auf Kanal A und dem INT Signal

Die hier beobachtete Streuung deckt sich mit der Angabe im Datenblatt des Eingabebausteins. Dort ist für die Streuung ein Wert von 10 µs angegeben. [17] Diese Streuung entsteht dadurch, dass die Eingabe nur alle 10 µs abgetastet wird. Erfolgt eine Änderung unmittelbar vor einer Abtastung, wird diese

sofort erkannt. Erfolgt sie unmittelbar nach einer Abtastung dauert es 10 µs bis zur nächsten Abtastung, bei der die Änderung erkannt werden kann. Da Änderungen zu jedem beliebigen Zeitpunkt zwischen zwei Abtastungen erfolgen können und auch erfolgen, ergibt sich eine gleichmäßige Verteilung über die Länge des Abtastintervalls. Darüber hinaus benötigt der Eingabebaustein offensichtlich etwa 30 µs zur Verarbeitung einer Änderung, bevor er diese weiterleitet.

Verzögerung zwischen dem INT Signal und der Datenübertragung über den I^2C Bus

Zur Ermittlung dieser Verzögerung wurde der Zeitunterschied zwischen der steigenden Flanke auf der INT Leitung und der ersten (fallenden) Flanke auf der Taktleitung des I^2C Busses bestimmt. Es wurde ein Mittelwert von etwa 75,6 µs und eine Standardabweichung von etwa 5,4 µs ermittelt. Die längste gemessene Verzögerung betrug etwa 312 µs. Diese Werte sind angesichts der Gesamtreaktionszeit nicht überraschend. Erstaunlich ist hingegen die Verteilung der Messwerte. Abbildung 65 zeigt das entsprechende Histogramm im Verhältnis 10:1. Es sind deutlich zwei separate Bereiche (Spitzen) zu erkennen, in denen sich jeweils viele der Messungen befinden.

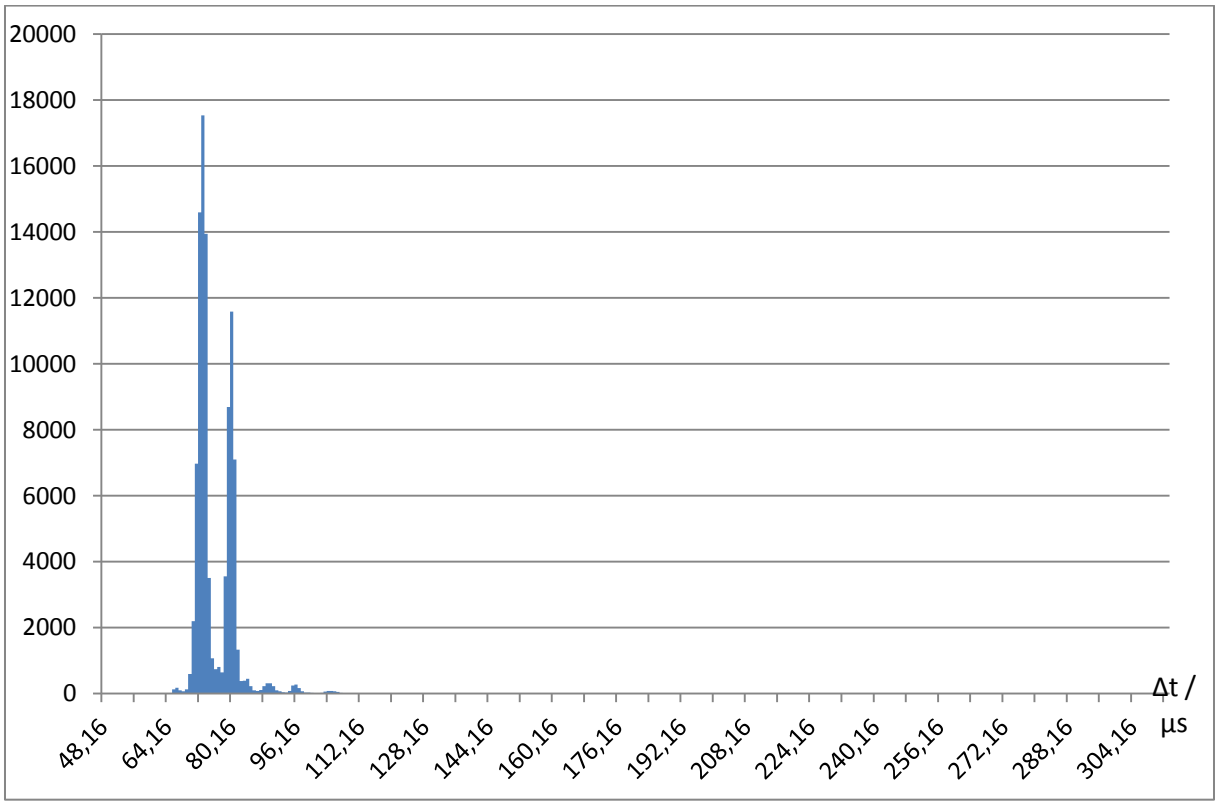


Abbildung 65: Histogramm der Verzögerungszeiten zwischen INT Signal und SCL Signal

Abbildung 66 zeigt eine vergrößerte Darstellung des Bereichs mit den beiden Spitzen im Verhältnis 1:1. Im dargestellten Bereich befinden sich über 96 % alle Messwerte. Die beiden Spitzen liegen grob 7 µs zueinander versetzt. Ebenfalls erstaunlich ist, dass sich diese Verteilung nicht deutlich im Histogramm, welches die Gesamtreaktionszeit des Systems zeigt (vgl. Abbildung 63), wiederspiegelt. An dieser Stelle liegen leider nicht genügend Informationen vor um diese Verteilung oder den ausbleibenden Einfluss auf die Gesamtreaktionszeit zu erklären. Dies müsste zunächst mit weiteren Experimenten untersucht werden.

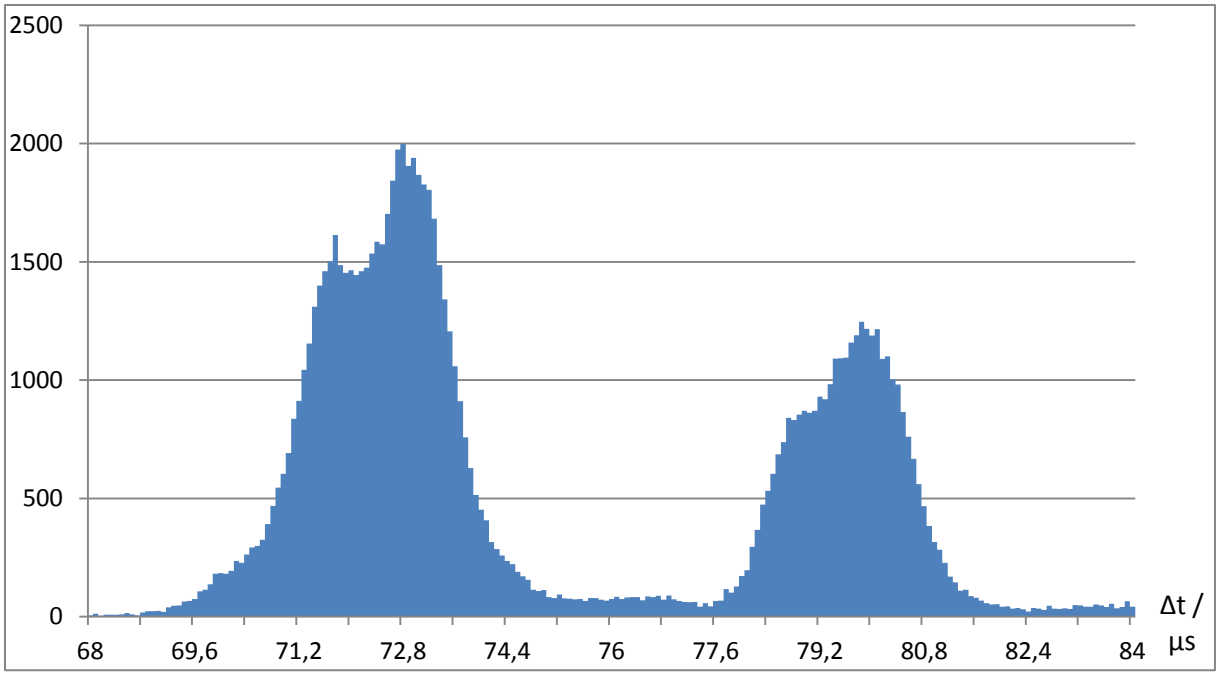


Abbildung 66: Detailansicht des Histogramms der Zeiten zwischen INT und SCL

Dauer der Datenübertragung über den I²C Bus

Bereits in der ersten Phase dieses Experiments wurde festgestellt, dass die Datenübertragung über den I²C Bus, gemessen von der ersten bis zur letzten Flanke auf der Taktleitung, ziemlich präzise 19 µs dauert. Dieses Ergebnis konnte in der zweiten Phase des Experiments bestätigt werden. Die Änderung der Übertragungszeit lag im Bereich der oder unterhalb der gewählten Messgenauigkeit von 80 ns, so dass überhaupt nur zwei verschiedene Zeitintervalle für die Übertragungszeit ermittelt wurden. Die meisten Messungen ergaben eine Zeit von 19,04 µs, einige ergaben 19,12 µs.

Verzögerung zwischen der Datenübertragung und dem OUT Signal

Die Messung der Verzögerung zwischen dem Ende der Datenübertragung auf dem I²C Bus, genauer gesagt der letzten Flanke auf der Taktleitung, und der steigenden Flanke auf dem OUT Signal, ergab einen Durchschnittswert von etwa 43,4 µs und eine Standardabweichung von etwa 3,5 µs.

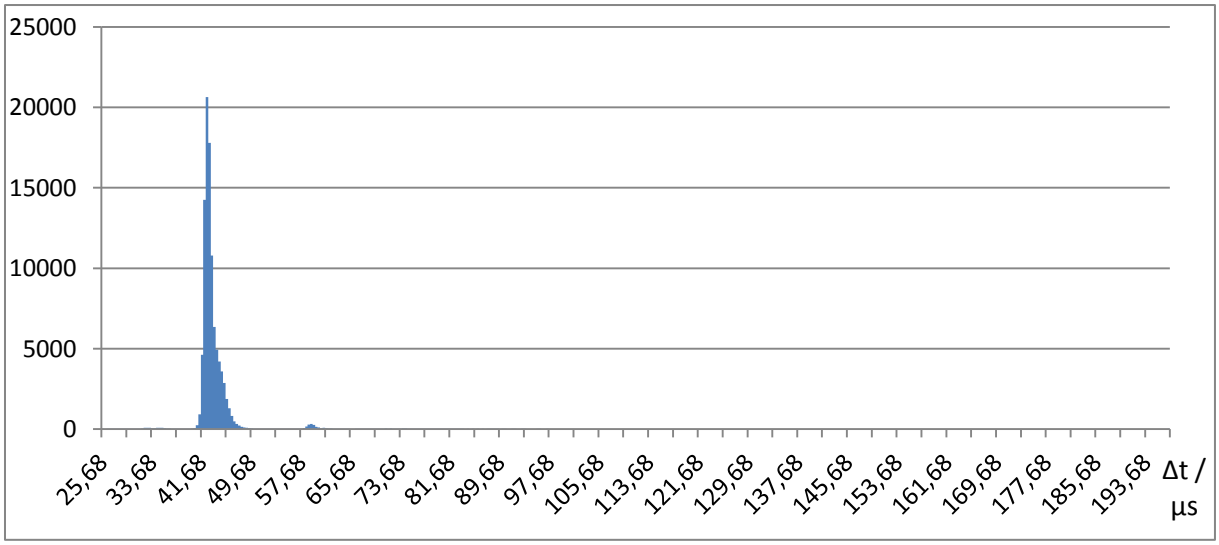


Abbildung 67: Histogramm der Verzögerung zwischen der Datenübertragung und OUT Signal

Abbildung 67 zeigt das zugehörige Histogramm im Verhältnis 5:1. Die längste ermittelte Verzögerung lag bei etwa 197 µs. Abbildung 68 zeigt eine vergrößerte Darstellung des Bereichs um den Durchschnittswert im Verhältnis 1:1. Im dargestellten Bereich befinden sich über 96 % der Messwerte. Die Ergebnisse dieser Messung, insbesondere die Verteilung der Messergebisse, wurden als nicht außergewöhnlich eingestuft.

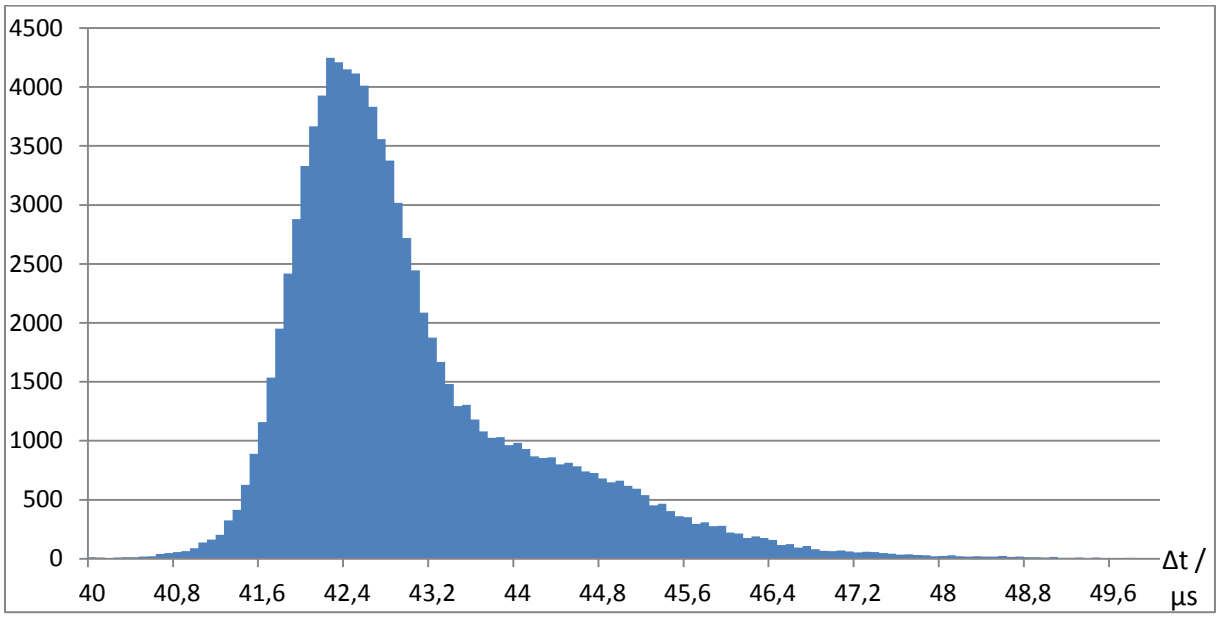


Abbildung 68: Detailansicht des Histogramms aus Abbildung 67

Verzögerung zwischen dem OUT Signal und der steigenden Flanke am Kanal B

Gemessen wurde die Zeit zwischen der steigenden Flanke auf dem OUT Signal und dem Punkt an dem die Ausgabeleitung, welche mit Kanal B überwacht wurde, 80 % des Nennwerts von 24 V erreicht hat. Ermittelt wurde ein Durchschnitt von 49,4 µs. Das zugehörige Histogramm in Abbildung 69 zeigt, dass die Messwerte über einen Bereich von knapp 13 µs nahezu gleichmäßig verteilt sind.

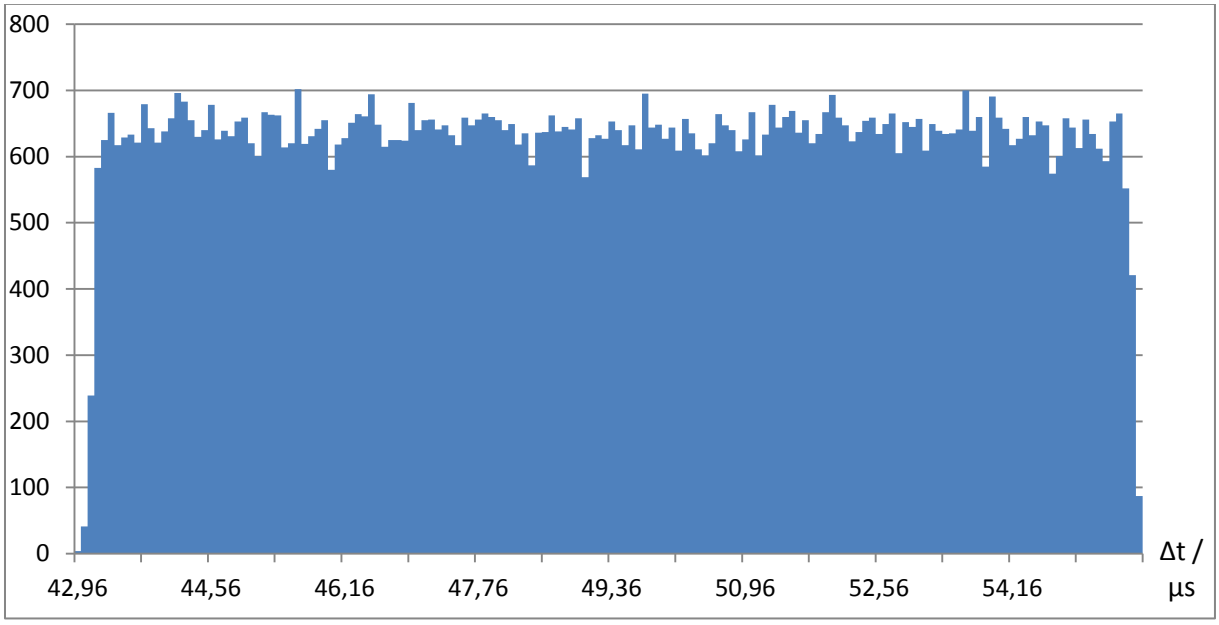


Abbildung 69: Histogramm der Verzögerung zwischen OUT Signal und der Flanke auf Kanal B

Die ermittelte durchschnittliche Verzögerung deckt sich grob mit der Angabe im zugehörigen Datenblatt des Ausgabebausteins. Dort wird eine Zeit von typischerweise 64 µs angegeben. Dabei wird allerdings von einer Messung bei 90 % und von einer stärkeren Last von nur 47 Ω ausgegangen. Da im Experiment eine Last von 10 kΩ verwendet wurde und schon bei 80 % des Maximalwerts gemessen wurde, ist die etwas kürzere ermittelte Reaktionszeit plausibel. Die ermittelte Streuung von knapp 13 µs deckt sich völlig mit der Angabe im Datenblatt. Dort ist angegeben, dass die Streuung zwischen 8 µs und 17,8 µs betragen kann. [62]

5.2.9. Zusammenfassung der Messergebnisse

Tabelle 21 fasst die Ergebnisse dieses Experiments zusammen. Für jede gemessene Verzögerung sind der ermittelte Durchschnittswert, die Quelle der Verzögerung, also, ob die Verzögerung hardware- oder softwarebedingt ist, der Bereich, über den die Werte gestreut sind, also der Abstand zwischen dem kleinsten und größten Messwert und der Anteil an der Gesamtreaktionszeit angegeben. Zusätzlich ist für Verzögerungen, deren ermittelte Verteilung nicht der Gleichverteilung ähnelt, die Stichprobenstandardabweichung aufgeführt. Obwohl natürlich auch für gleichverteilte Werte eine Standardabweichung abgegeben werden könnte, scheint diese Angabe nicht hilfreich.

Zeit von ...	bis ...	Ø	s	Δ	Quelle	Anteil
Flanke auf A	Flanke auf INT	36,8 µs	-	10,4 µs	Hardware	16,4 %
Flanke auf IN	Flanke auf INT	52 ns	-	< 2 ns	Hardware	0 %
Flanke auf INT	Anfang SCL	75,6 µs	5,4 µs	265,3 µs	Software	33,7 %
Anfang SCL	Ende SCL	19,0 µs	-	< 80 ns	Hardware	8,5 %
Ende SCL	Flanke auf OUT	43,4 µs	3,5 µs	171,9 µs	Software	19,4 %
Flanke auf OUT	Flanke auf B	49,4 µs	-	12,7 µs	Hardware	22,0 %
Flanke auf A	Flanke auf B	224,2 µs	8,1 µs	284,2 µs	HW/SW	100 %

Tabelle 21: Zusammensetzung der Reaktionszeit

Abbildung 70 visualisiert die Ergebnisse aus Tabelle 21. Grün dargestellte Teile stellen hardwarebedingte Verzögerungen dar, während softwarebedingte Verzögerungen blau dargestellt sind.

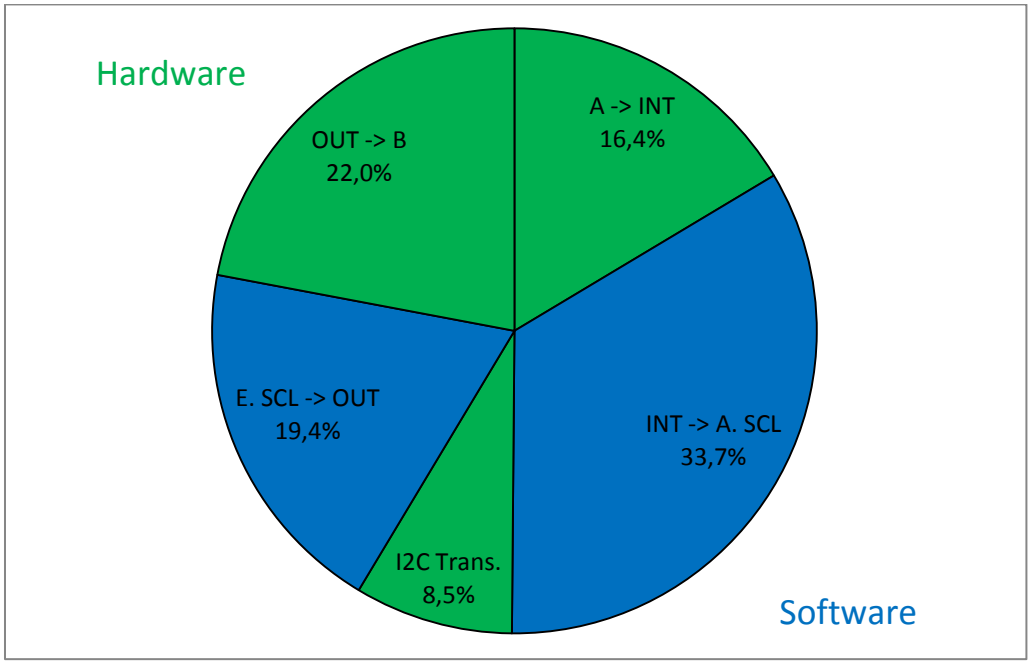


Abbildung 70: Visualisierung der Zusammensetzung der Reaktionszeit

Diskussion der durchschnittlichen Verzögerung

Die beiden großen grün dargestellten Verzögerungen, sind typisch für die eingesetzten Bausteine und können nicht beeinflusst werden. Nur durch den Einsatz anderer Bausteine könnten diese Verzögerungen verkürzt bzw. überhaupt verändert werden. Dies könnte allerdings negative Nebenwirkungen mit sich bringen. Würde man beispielsweise die Anstiegszeit der Ausgabeleitung deutlich verkürzen, könnte dies Störungen verursachen. Die Anstiegsgeschwindigkeit wird von den eingesetzten Ausgabebausteinen absichtlich beschränkt und damit die Anstiegszeit verlängert um die Entstehung von Störungen zu reduzieren.

Möchte man diese Teile folglich nicht verändern, bleibt nur noch die Übertragungszeit über den I^2C Bus als hardwarebedingte Verzögerung, die angepasst werden kann. Diese kann durch Veränderung der Übertragungsgeschwindigkeit leicht angepasst werden. Tests haben ergeben, dass eine fehlerfreie Datenübertragung mit bis zu 2 MHz auf den beiden Leitungen des Busses prinzipiell möglich ist. Allerdings ist bei dieser Übertragungsgeschwindigkeit die Anstiegszeit der Signale, welche durch die im Raspberry Pi verbauten Pull-Up-Widerstände gegeben ist, im Verhältnis zur Periodendauer so lang, dass das Auftreten von Übertragungsfehlern wahrscheinlich erscheint. Daher scheint es besser sich auf eine Übertragungsgeschwindigkeit von 1 MHz zu beschränken. Angesichts der Tatsache, dass die Datenübertragung nur 8,5 % der Gesamtreaktionszeit ausmacht, scheint eine Verkürzung dieser Zeit auch nicht erforderlich.

Theoretisch möglich wäre eine Verkürzung der softwarebedingten Verzögerungen. Betrachtet man allerdings das Verhältnis von software- und hardwarebedingten Verzögerungen, stellt man fest, dass beide etwa den gleichen Anteil an der Reaktionszeit haben. Würde es also beispielsweise gelingen die softwarebedingten Verzögerungen um 50 % zu reduzieren, würde sich die Gesamtreaktionszeit nur um 25 % verbessern. Bedenkt man zusätzlich, dass die durchschnittliche Gesamtreaktionszeit im Bereich von 0,2 ms liegt, was weit unter den geforderten 5 ms liegt, erscheint eine weitere Verkürzung der durchschnittlichen softwarebedingten Reaktionszeit nicht lohnenswert und auch nicht erforderlich.

Diskussion der Streuung der Reaktionszeit

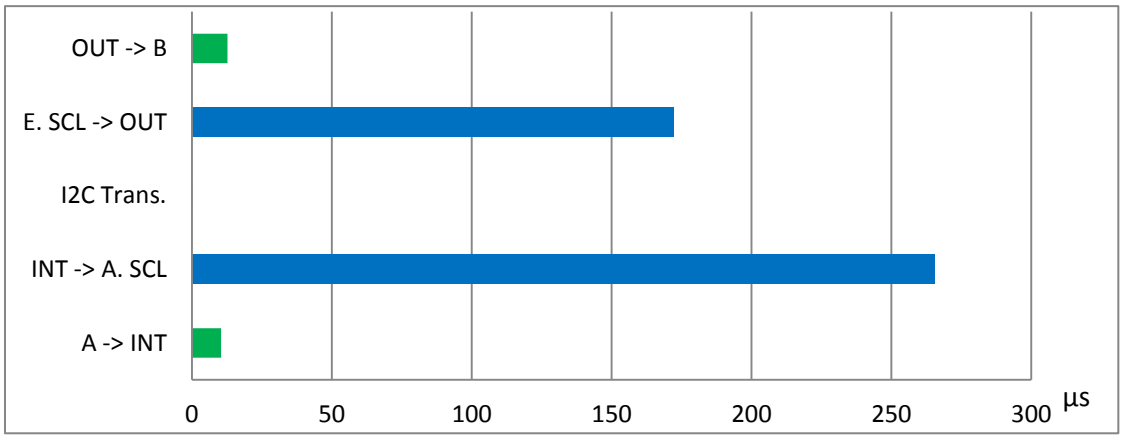


Abbildung 71: Streuungsbereiche der einzelnen Teile der Reaktionszeit

Der zweite Aspekt der neben dem Durchschnitt der Reaktionszeit betrachtet werden muss, ist die Veränderung der Reaktionszeit von Durchlauf zu Durchlauf. Wie Tabelle 21 zu entnehmen ist, unterliegen sowohl die software- wie auch die hardwarebedingten Teile der Reaktionszeit Schwankungen. Abbildung 71 stellt die Größen der Streuungsbereiche der einzelnen Teile graphisch dar. Legt man die Werte für die Schwankungen aus der Tabelle zugrunde und geht vom ungünstigsten Fall aus, stellt man fest, dass die hardwarebedingte Schwankung gerade einmal etwa 23 µs beträgt. Die softwarebedingten Schwankungen fallen dagegen deutlich stärker aus. Betrachtet man beispielsweise Abbildung 62 er-

kennt man, dass der Unterschied zweier Reaktionszeiten hunderte Mikrosekunden betragen kann. Damit ist die Gesamtschwankung klar durch softwarebedingte Schwankungen dominiert. Hardwarebedingte Schwankungen sind nahezu vernachlässigbar, was auch in Abbildung 71 deutlich wird. Sinnvollerweise sollten die softwarebedingten Schwankungen so weit reduziert werden, dass sie im gleichen Bereich, wie die hardwarebedingten Schwankungen liegen. Eine noch weitere Reduzierung wäre natürlich noch besser, aber nur noch von geringem Vorteil. Angenommen beide Schwankungen wären gleich groß. Würde man dann die softwarebedingte Schwankung um die Hälfte reduzieren, würde sich die Gesamtschwankung um nur 25 % reduzieren. Daher erscheint es sinnvoll zu versuchen die softwarebedingten Schwankungen in den Bereich der hardwarebedingten Schwankungen, die nicht weiter reduziert werden können, zu bringen.

5.3. Einfluss der CPU-Frequenz

Beim Experimentieren wurde zufällig entdeckt, dass sich die Reaktionszeit beim Ausführen bestimmter Befehle in der Kommandozeile vorübergehend deutlich verkürzt. Durch Ausprobieren konnte festgestellt werden, dass sich die Reaktionszeit immer dann verkürzt, wenn die CPU stark belastet wird.

Die Erkenntnis, dass sich die Reaktionszeit bei starker CPU-Belastung verkürzt, führte unweigerlich zu der Vermutung, dass das Betriebssystem die CPU-Frequenz erhöht, wenn die CPU-Belastung ansteigt, wodurch auch die Ausführungszeit der Softwareteile, welche an der Reaktion beteiligt sind, verkürzt wird. Was letztlich zu einer Verkürzung der Gesamtreaktionszeit führt. Die Vermutung, dass die CPU-Frequenz vom Betriebssystem verändert wird, konnte durch einem Blick ins /sys Dateisystem aufgrund der Angaben unter /sys/devices/system/cpu/cpu0/cpufreq bestätigt werden. Es wird standardmäßig ein „ondemand“ Governor ausgeführt, der die CPU-Frequenz bei geringer Belastung auf 600 MHz senkt und bei hoher Belastung auf 900 MHz erhöht. Es ist anzunehmen, dass durch dieses Vorgehen die Streuung der Reaktionszeit erhöht wird, da die Verarbeitung der entscheidenden Softwareteile in manchen Fällen mit 600 MHz und in anderen Fällen mit 900 MHz erfolgt, oder sich die Frequenz sogar während der Verarbeitung ändert.

5.3.1. Fragestellung

In diesem Experiment soll untersucht werden, ob durch Nutzung einer festen CPU-Frequenz eine Reduzierung der Streuung der Reaktionszeit erreicht werden kann.

5.3.2. Experimentdesign

Um eine evtl. Änderung des Verhaltens festzustellen, soll das letzte Experiment (vgl. Abschnitt 5.2) präzise wiederholt werden. Der einzige Unterschied soll darin bestehen, dass vor Beginn der Messungen die CPU-Frequenz auf einen festen Wert fixiert wird.

Dies soll erreicht werden indem anstelle des „ondemand“ Governors der „performance“ Governor aktiviert wird. Dieser sorgt dafür, dass die CPU immer mit der maximalen Frequenz betrieben wird. Dies wären standardmäßig 900 MHz. Da die CPU beim letzten Experiment praktisch nicht belastet wurde, ist davon auszugehen, dass sie die meiste Zeit mit nur 600 MHz betrieben wurde. Daher ist zu erwarten, dass das Festlegen der Frequenz auf 900 MHz zu einer deutlichen Verkürzung der durchschnittlichen Reaktionszeit führen würde. Dies wäre zwar im Allgemeinen wünschenswert, stellt aber im Rahmen dieses Experiments ein Problem dar, da speziell die Auswirkungen auf die Streuung untersucht werden sollen. Würde sich die durchschnittliche Reaktionszeit wesentlich verändern, wäre eine Bewertung der Änderung der Streuung schwierig. Daher soll die Maximalfrequenz zusätzlich auf 600 MHz festgelegt werden.

Es wird erwartet, dass sich die durchschnittliche Reaktionszeit bei Einsatz dieser Konfiguration nur unwesentlich ändert, während die Streuung abnimmt.

5.3.3. Experimentdurchführung

Die Durchführung des Experiments erfolgt analog zur Durchführung der zweiten Phase des letzten Experiments mit dem Unterschied, dass vor Beginn der Messungen die Befehle, welche im Codeausschnitt 17 gezeigt werden, ausgeführt wurden. Die Datei cpuinfo_min_freq enthielt dabei den Wert, welcher einer Frequenz von 600 MHz entspricht.

```
sudo -i
cd /sys/devices/system/cpu/cpu0/cpufreq
echo performance > scaling_governor
# 600 MHz
cat cpuinfo_min_freq > scaling_max_freq
```

Codeausschnitt 17: Festlegung der CPU-Frequenz

5.3.4. Messergebnisse

Dieses Experiment wurde mit derselben Konfiguration wie das letzte Durchgeführt. Die Ausgabe der Konfiguration entsprach jener, welche im Codeausschnitt 16 gezeigt wird. Eine Auswertung der Messergebenisse ergab eine minimale Verlängerung der durchschnittlichen Reaktionszeit auf etwa 225,2 µs, was einer Zunahme von nur etwa 0,4 % entspricht. Die Standardabweichung ist deutlich auf etwa 6,7 µs gefallen, was einer Verbesserung um mehr als 17 % entspricht.

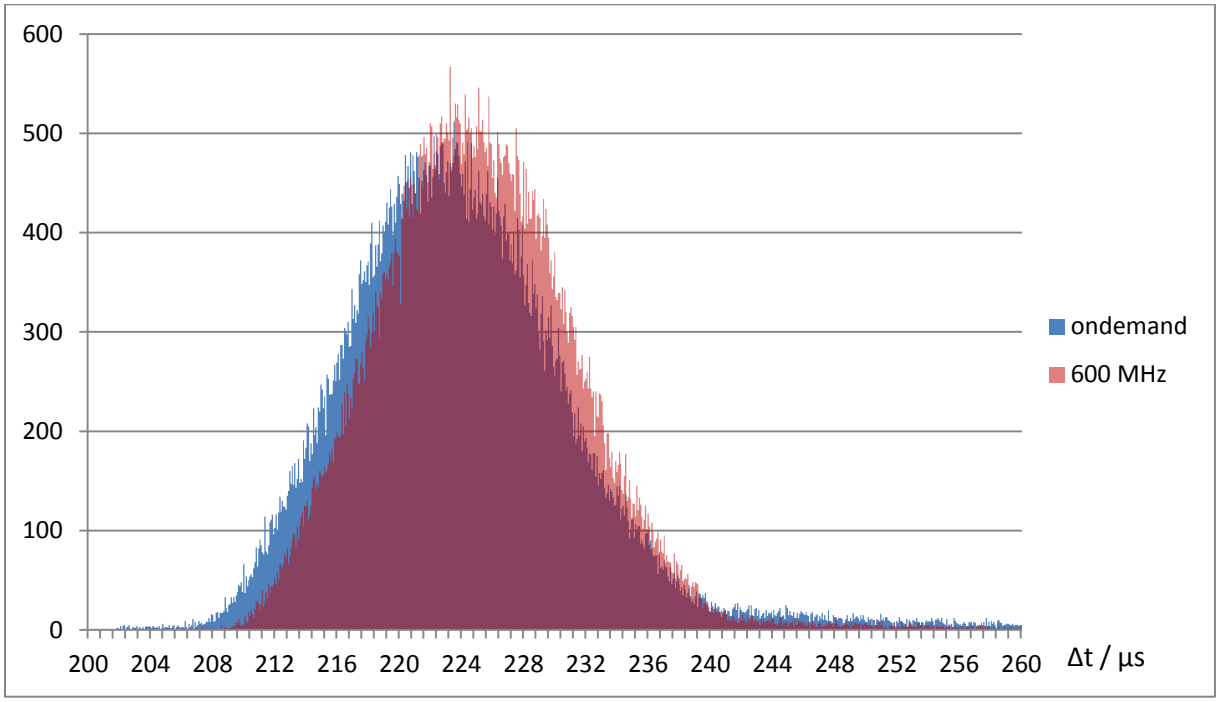


Abbildung 72: Vergleich der Histogramme der Reaktionszeit vom 2. und 3. Experiment

Um die Veränderung der Reaktionszeit besser beurteilen zu können, wurden ein Histogramm, das die gemessenen Gesamtreaktionszeiten zeigt, erstellt und über das entsprechende Histogramm des letzten Experiments, bei dem der „ondemand“ Governor genutzt wurde, gelegt. Abbildung 72 zeigt einen Ausschnitt des Resultats. Im gezeigten Ausschnitt sind jeweils weit über 99 % der Messwerte beider Experimente enthalten. Wie zu erwarten war, hat sich die Verteilung insgesamt nicht wesentlich geändert. Es gab nur eine minimale Rechtverschiebung in Verbindung mit einer Abnahme der Streuung.

Das Experiment hat die aufgestellte Vermutung bestätigt. Es ist möglich, durch Nutzung einer festen CPU-Frequenz die Streuung der Reaktionszeit zu reduzieren. Der Durchschnittswert ändert sich dabei nur unwesentlich.

5.4. CONFIG_PREEMPT_RT Patch

Nachdem im letzten Experiment ein Weg gefunden wurde, die Änderung der CPU-Frequenz zu unterbinden und gezeigt werden konnte, dass dadurch die Streuung verringert werden kann, soll nun unter Einsatz einer festen Frequenz versucht werden mit Hilfe des CONFIG_PREEMPT_RT Patches die Streuung der Reaktionszeit weiter zu reduzieren.

5.4.1. Fragestellung

Ziel dieses Experiments ist es zu untersuchen, ob und ggf. wie sich die Reaktionszeit verändert, wenn ein mit dem CONFIG_PREEMPT_RT Patch gepatchter Kernel eingesetzt wird. Es wird grundsätzlich erwartet, dass sich die durchschnittliche Reaktionszeit nur geringfügig verändert, während die Streuung deutlich abnimmt.

5.4.2. Experimentdesign

Um den Einfluss des CONFIG_PREEMPT_RT Patches zu untersuchen, soll das letzte Experiment erneut durchgeführt werden. Diesmal soll aber statt dem normalen Kernel, welcher in der bisher eingesetzten Version des Raspbian Systems enthalten ist, ein mit dem CONFIG_PREEMPT_RT Patch ausgestatteter Kernel verwendet werden.

Um mit den bisherigen Messungen vergleichbare Ergebnisse zu erhalten, ist es von entscheidender Bedeutung, dass abgesehen von den Änderungen, die vom RT Patch durchgeführt werden, keine weiteren Anpassungen vorgenommen werden. Die gesamte eingesetzte Software, einschließlich der vom Patch nicht betroffenen Teile des Kernels darf nicht verändert werden. Um dies zu erreichen, soll die bisher eingesetzte Version des Raspbian Betriebssystems weiterhin genutzt werden. Es soll lediglich der Kernel ausgetauscht werden, wobei dieser vom bisher eingesetzten Kernel nur an den Stellen abweichen darf, welche vom RT Patch modifiziert werden.

5.4.3. Experimentvorbereitung

Die Vorbereitung dieses Experiments bestand hauptsächlich darin einen neuen Kernel zu erstellen und zu installieren. Um sicherzustellen, dass der neue Kernel mit dem bisher eingesetzten Kernel weitestgehend identisch ist, musste zunächst der Quellcode beschafft werden, welcher zur Erstellung des bisher genutzten Kernels verwendet worden ist. Dieser musste dann mit der passenden Version des CONFIG_PREEMPT_RT Patches versehen und mit der Konfiguration, welche zur Erstellung des bisher eingesetzten Kernels verwendet worden ist, kompiliert werden.

Beschaffung des Quellcodes des Kernels

Eine von den Entwicklern des Raspberry Pi verwaltete Version des Linux-Kernels wird im in einem git-Repository unter https://github.com/raspberrypi/linux.git verwaltet. Es ist davon auszugehen, dass diese Version zur Erstellung des Kernels, welcher in der bisher verwendeten Version des Raspbian Systems enthalten ist, verwendet wurde. Mit dem Befehl uname konnte ermittelt werden, dass in der genutzten Version des Raspbian Systems ein Kernel mit der Version 4.1.13 verwendet wird und dass dieser am 13.11.2015 kompiliert wurde. Basierend auf diesen Angaben, wurde ermittelt, dass mit hoher Wahrscheinlichkeit zur Erstellung der Quellcode aus dem Repository verwendet wurde, wie er am 13.11.2015 mit dem Commit mit der ID bc1669c846b629cface0aaa367afb2b9c6226faf vorlag. Eine frühere Version konnte nicht verwendet werden, da erst mit diesem Commit einige wesentliche Rasp-

berry Pi spezifische Änderungen in die Branch rpi-4.1.y hinzugefügt wurden. Eine spätere Version war zum Zeitpunkt des ursprünglichen Kompilierens noch nicht vorhanden. Zusätzlich bekräftigt wurde diese Annahme dadurch, dass dieser Commit mit dem Tag rpi-bootloader-1.20151118-1 versehen wurde, was ihn gegenüber den meisten anderen Commits auszeichnet, da diese nicht mit Tags versehen sind. Folglich wurde genau diese Version des Quellcodes zur Erstellung des neuen Kernels ausgewählt und gepackt als linux-rpi-bootloader-1.20151118-1.zip abgespeichert.

Beschaffung der passenden Version des CONFIG_PREEMPT_RT Patches

Der CONFIG_PREEMPT_RT Patch wird unter https://www.kernel.org/pub/linux/kernel/projects/rt/ für verschiedene Versionen des Linux-Kernels angeboten. Passend zur genutzten Version des Kernels wurde die Datei https://www.kernel.org/pub/linux/kernel/projects/rt/4.1/older/patch-4.1.13-rt15.patch.gz heruntergeladen. Dies ist die neueste Version des Patches für die Kernel Version 4.1.13. Alle Änderungen, die vom RT Patch durchgeführt werden, sind in einer großen Patch-Datei enthalten.

Erstellung und Installation des Kernels

Zur Erstellung des neuen Kernels wurde zunächst der zuvor heruntergeladene Quellcode des Kernels in Form einer zip-Datei auf den Raspberry Pi übertragen und dort entpackt. In den neu erstellten Ordner mit dem entpackten Quellcode wurde die Datei patch-4.1.13-rt15.patch.gz übertragen und dort ebenfalls entpackt. Die nachfolgenden Schritte erfolgten dann in Anlehnung an das Vorgehen, welches von den Entwicklern des Raspberry Pi zum Patchen und zur Erstellung des Linux-Kernels empfohlen wird. [96] Die verwendeten Befehle sind im Codeausschnitt 18 dokumentiert.

```
# unzip kernel
unzip linux-rpi-bootloader-1.20151118-1.zip
cd linux-rpi-bootloader-1.20151118-1

# unzip patch (transfered over SFTP)
gunzip patch-4.1.13-rt15.patch.gz

# apply patch
cat patch-4.1.13-rt15.patch | patch -p1

# install bc
sudo apt-get install bc

# get current kernel config
sudo modprobe configs
zcat /proc/config.gz > .config

# update configuration
KERNEL=kernel7
make oldconfig
# Preemption Model
#  5. Fully Preemptible Kernel
# Testing module to detect hardware-induced latencies
#  M
# Interrupts-off Latency Histogram
#  N
# Scheduling Latency Histogram
#  N
# Missed Timer Offsets Histogram
#  N

# build
make -j6 zImage modules dtbs

# install
```

```
sudo make modules_install
sudo cp arch/arm/boot/dts/*.dtb /boot/
sudo cp arch/arm/boot/dts/overlays/*.dtb* /boot/overlays/
sudo cp arch/arm/boot/dts/overlays/README /boot/overlays/
sudo scripts/mkknlimg arch/arm/boot/zImage /boot/$KERNEL.img
```

Codeausschnitt 18: Befehle zur Erstellung des neuen Linux-Kernels

Wie zu erkennen ist, wurde als allererstes der Patch angewendet. Obwohl der verwendete Kernel nicht vollständig dem vanilla Kernel entspricht, für welchen der Patch eigentlich gedacht ist, verlief das Anwenden des Patches problemlos. Einige Zeilen waren im Vergleich zum vanilla Kernel zwar leicht verschoben, dies konnte aber vom patch Tool automatisch kompensiert werden.

Danach wurde vorsorglich das Tool bc installiert, da dies in der zuvor erwähnten Beschreibung der Raspberry Pi Entwickler als Abhängigkeit genannt wird. Die Konfiguration welche zur Erstellung des ursprünglichen Kernels genutzt wurde, wurde in diesem praktischerweise hinterlegt und konnte mit Hilfe des Moduls configs ausgelesen werden. Damit war es möglich den Kernel mit exakt derselben Konfiguration erneut zu kompilieren. Es war lediglich erforderlich die vom RT Patch hinzugefügten Optionen zu setzen. Dabei wurde natürlich der „Fully Preemptible Kernel“ ausgewählt. Diagnosefunktionen wurden deaktiviert um keinen unnötigen Overhead hinzuzufügen. Abschließend wurde der neue Kernel installiert. Es wurde ein Neustart durchgeführt und mit dem Befehl uname verifiziert, dass der neue Kernel aktiv ist.

5.4.4. Experimentdurchführung

Die eigentliche Durchführung des Experiments verlief analog zum letzten Experiment. Auch diesmal wurden vor Beginn der Messungen die Befehle, welche im Codeausschnitt 17 gezeigt werden, ausgeführt.

5.4.5. Messergebnisse

Die Ausgabe der Konfiguration entsprach auch bei diesem Experiment der Ausgabe, welche im Codeausschnitt 16 gezeigt wird. Eine Auswertung der Ergebnisse zeigte eine Verlängerung der durchschnittlichen Reaktionszeit auf etwa 247 µs, was einer Verschlechterung von knapp 10 % entspricht und eine Erhöhung der Standardabweichung auf über 14,5 µs. Eine geringfügige Änderung, insbesondere auch eine Verlängerung, der durchschnittlichen Reaktionszeit wurde erwartet, aber eine Erhöhung der Standardabweichung um mehr als 116 % kam völlig unerwartet. Es wurde eigentlich eine deutliche Reduzierung der Standardabweichung erwartet.

Die genauen Auswirkungen auf die Reaktionszeit werden aber erst bei einer Betrachtung des zugehörigen Histogramms deutlich. Dieses ist in Abbildung 73 im Verhältnis 8:1 dargestellt. Die längste gemessene Reaktionszeit betrug etwa 454 µs.

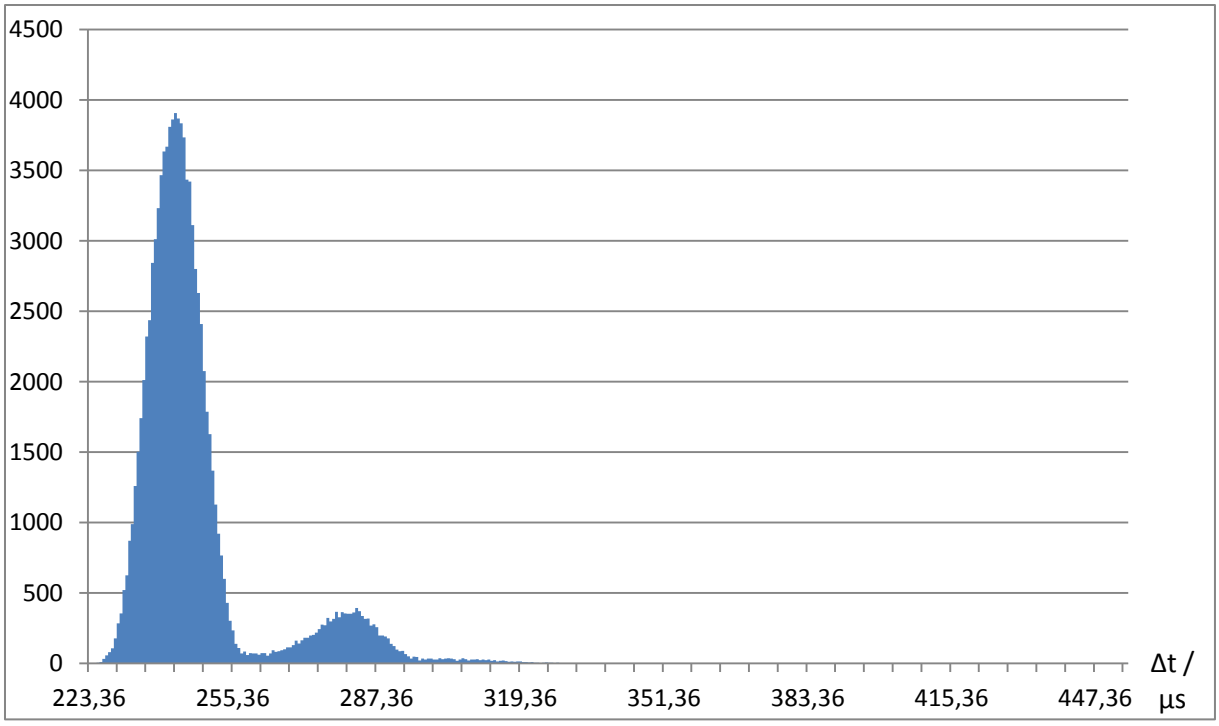


Abbildung 73: Histogramm der Reaktionszeiten mit dem RT Patch

Es sind ganz deutlich zwei separate Bereiche zu erkennen, auf die sich die Werte aufteilen. Die beiden Spitzen liegen etwa 40 µs auseinander, was die extreme Erhöhung der Standardabweichung erklärt. Ein ähnliches Verhalten wurde bereits in Abbildung 66 beobachtet. Dort war der Unterschied aber deutlich geringer und es hatte keine erkennbaren Auswirkungen auf die Gesamtreaktionszeit. Auch an dieser Stelle kann dieses Verhalten vorerst nicht erklärt werden. Das Verhalten wird daher im Folgenden weiter untersucht.

5.5. RT Patch mit Anpassung der Thread-Prioritäten

Im letzten Experiment wurden beim Einsatz des CONFIG_PREEMPT_RT Patches eine unerwartete Verteilung und eine damit verbundene hohe Standardabweichung bei den Reaktionszeiten beobachtet. Es konnte aufgrund der Messergebnisse vorerst keine Erklärung für dieses ungünstige Verhalten geliefert werden.

Ein wesentlicher Unterschied zwischen dem zuvor verwendeten normalen Kernel und dem neuen mit dem RT Patch versehenen Kernel, besteht darin, dass beim letzteren Interrupthandler in zwei Teile aufgeteilt sind. Die sog. obere Hälfte bestätigt meist nur den Erhalt des Interrupts und veranlasst die Ausführung der sog. unteren Hälfte des Interrupthandlers. Die untere Hälfte wird nicht im Interruptkontext sondern in einem eigenen Thread ausgeführt. Für jeden Interrupt gibt es im System einen entsprechenden Thread. Daher ist die Gesamtzahl der Threads beim Einsatz des RT Patches deutlich höher als bei einem normalen Kernel. Es sind daher deutlich mehr Wechsel zwischen Threads erforderlich. Dies dürfte für die höhere durchschnittliche Reaktionszeit verantwortlich sein, ist aber noch keine Erklärung für die ungewöhnliche Verteilung, welche im Histogramm zu beobachten ist.

Die Threads, welche die unteren Hälften der Interrupthandler verarbeiten, nutzen standardmäßig das SCHED_FIFO Schedulingverfahren und haben auf dem vorliegenden System eine Priorität von 50. Das heißt, dass die unteren Hälften aller Interrupthandler die gleiche Priorität haben, insbesondere auch jene, die an der im Experiment betrachteten Reaktion gar nicht beteiligt sind. Es wäre denkbar, dass diese Konstellation in irgendeiner Weise zu dem ungünstigen Verhalten führt.

5.5.1. Fragestellung

Dieses Experiment soll klären, ob durch eine Anpassung der Prioritäten der beteiligten Threads eine Verbesserung des Verhaltens erzielt werden kann. Es soll speziell untersucht werden, ob durch eine Erhöhung der Prioritäten der Threads, welche unmittelbar an der im Experiment erzeugten und untersuchten Reaktion des Systems beteiligt sind, eine Verbesserung erzielt werden kann.

5.5.2. Experimentdesign

Um zu klären, ob durch Anpassung der Prioritäten eine Verbesserung erzielt werden kann, soll das letzte Experiment (vgl. Abschnitt 5.4) wiederholt werden. Dabei sollen vor Beginn der Messungen den Threads, welche an der Reaktion beteiligt sind, höhere Prioritäten zugewiesen werden. Dadurch sollten diese gegenüber anderen Interrupthandlern bevorzugt behandelt werden, was zu einer Verkürzung der durchschnittlichen Reaktionszeit und zu einer Abnahme der Streuung führen sollte, da der Einfluss anderer Ereignisse im System abnehmen sollte.

5.5.3. Experimentvorbereitung

Bevor mit dem Experiment begonnen werden konnte, musste zunächst geklärt werden, welche Interrupts und welche Threads an der Reaktion beteiligt sind.

Bestimmung der genutzten Interrupts

Zuallererst wurden die Nummern der beteiligten Interrupts bestimmt. Dazu wurde der Inhalt der Datei /proc/interrupts herangezogen. Codeausschnitt 19 und Codeausschnitt 20 zeigen den Inhalt dieser Datei vor und nach dem Start des Testprogramms. Die relevanten Unterschiede sind rot markiert. Offensichtlich sind die Interrupts mit den Nummern 49, 79, 485, 486 und 487 an der Reaktion des Systems beteiligt. Wird das Testprogramm nicht ausgeführt, treten diese Interrupts überhaupt nicht auf.

```
pi@raspberrypi:~ $ cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
 16:          0          0          0          0   ARMCTRL  16 Edge      bcm2708_fb dma
 20:       1646          0          0          0   ARMCTRL  20 Edge      DMA IRQ
 32:      83551          0          0          0   ARMCTRL  32 Edge      dwc_otg, [...]
 49:          0          0          0          0   ARMCTRL  49 Edge      3f200000.gpio:bank0
 50:          0          0          0          0   ARMCTRL  50 Edge      3f200000.gpio:bank1
 65:         38          0          0          0   ARMCTRL  65 Edge      3f00b880.mailbox
 66:          2          0          0          0   ARMCTRL  66 Edge      VCHIQ doorbell
 75:          1          0          0          0   ARMCTRL  75 Edge
 77:        205          0          0          0   ARMCTRL  77 Edge      DMA IRQ
 79:          0          0          0          0   ARMCTRL  79 Edge      3f804000.i2c
 83:          5          0          0          0   ARMCTRL  83 Edge      uart-pl011
 84:       4272          0          0          0   ARMCTRL  84 Edge      mmc0
 96:          0          0          0          0   ARMCTRL  96 Edge      arch_timer
 97:      30872      31158      30921      30938   ARMCTRL  97 Edge      arch_timer
FIQ:           usb_fiq
IPI0:        0          0          0          0   CPU wakeup interrupts
IPI1:        0          0          0          0   Timer broadcast interrupts
IPI2:     7217      85036      77034      84578   Rescheduling interrupts
IPI3:        6          5          8          8   Function call interrupts
IPI4:      187         32         28         30   Single function call interrupts
IPI5:        0          0          0          0   CPU stop interrupts
IPI6:        0          0          0          0   IRQ work interrupts
IPI7:        0          0          0          0   completion interrupts
Err:         0
```

Codeausschnitt 19: Inhalt von /proc/interrupts vor dem Start des Testprogramms

Interrupt 49 wird scheinbar bei jeder Änderung an einem Eingabepin ausgelöst, die als interrupterzeugend konfiguriert wurde. Im vorliegenden Fall, also bei jeder steigenden Flanke auf einem der GPIO-Pins 5, 6 oder 7. Das Testprogramm wurde über längere Zeit ausgeführt. Danach konnte aufgrund der geschätzten Anzahl an steigenden Flanken, die während dieser Zeit aufgetreten sein müssen, ausgesagt werden, dass bei fallenden Flanken der Interrupt 49 nicht ausgelöst wird. Dies stimmt mit der vorgenommenen Konfiguration überein.

Interrupt 79 wird scheinbar während der Übertragung über den I^2C Bus genutzt. Die neu hinzugekommenen Interrupts 485 bis 487 gehören offensichtlich zu den GPIO Pins 5, 6 und 7, welche als INTd, INT1 und INT2 genutzt werden. Diese werden allem Anschein nach immer dann ausgelöst, wenn auf dem entsprechenden Pin eine steigende Flanke erkannt wird. Die Anzahl dieser drei Interrupts über alle CPU-Kerne entspricht in Summe genau der Zahl, die angibt, wie oft der Interrupt 49 ausgelöst wurde. Warum es notwendig ist pro steigende Flanke zwei Interrupts auszulösen, einmal den allgemeinen Interrupt 49 und einmal einen der speziellen Interrupts 485, 486 oder 487 ist nicht klar, aber offensichtlich wird dies getan.

```
pi@raspberrypi:~ $ cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
 16:          0          0          0          0   ARMCTRL  16 Edge      bcm2708_fb dma
 20:       1908          0          0          0   ARMCTRL  20 Edge      DMA IRQ
 32:     304596          0          0          0   ARMCTRL  32 Edge      dwc_otg, [...]
 49:      30742          0          0          0   ARMCTRL  49 Edge      3f200000.gpio:bank0
 50:          0          0          0          0   ARMCTRL  50 Edge      3f200000.gpio:bank1
 65:         39          0          0          0   ARMCTRL  65 Edge      3f00b880.mailbox
 66:          2          0          0          0   ARMCTRL  66 Edge      VCHIQ doorbell
 75:          1          0          0          0   ARMCTRL  75 Edge
 77:        412          0          0          0   ARMCTRL  77 Edge      DMA IRQ
 79:     429484          0          0          0   ARMCTRL  79 Edge      3f804000.i2c
 83:          5          0          0          0   ARMCTRL  83 Edge      uart-pl011
 84:       6332          0          0          0   ARMCTRL  84 Edge      mmc0
 96:          0          0          0          0   ARMCTRL  96 Edge      arch_timer
 97:     115139     116337     115251     115195   ARMCTRL  97 Edge      arch_timer
485:          4          0          2          0   pinctrl-bcm2835   5 Edge      gpiolib
486:       8416      11974       9425        919   pinctrl-bcm2835   6 Edge      gpiolib
487:          0          0          1          1   pinctrl-bcm2835   7 Edge      gpiolib
FIQ:              usb_fiq
IPI0:        0          0          0          0   CPU wakeup interrupts
IPI1:        0          0          0          0   Timer broadcast interrupts
IPI2:    51711     393028     449296     335165   Rescheduling interrupts
IPI3:        6          5          8          8   Function call interrupts
IPI4:     1532        406        802        245   Single function call interrupts
IPI5:        0          0          0          0   CPU stop interrupts
IPI6:        0          0          0          0   IRQ work interrupts
IPI7:        0          0          0          0   completion interrupts
Err:         0
```

Codeausschnitt 20: Inhalt von /proc/interrupts nach dem Start des Testprogramms

Es fällt auf, dass die meisten Interrupts auf dem CPU-Kern 0 verarbeitet werden. Normalerweise sollte es möglich sein, mit Hilfe der Dateien /proc/irq/[Nummer]/smp_affinity zu konfigurieren, an welche Kerne ein Interrupt weitergeleitet wird. Der Raspberry Pi scheint diese Funktion aber nicht zu unterstützen.

Bestimmung der zugehörigen Threads

Nachdem bekannt war, dass die Interrupts 49, 79, 485, 486 und 487 an der Reaktion beteiligt sind, mussten nun die zugehörigen Threads bestimmt werden. Dazu wurde die Ausgabe von ps genutzt. Diese wird in gekürzter Fassung in Codeausschnitt 21 wiedergegeben. Wie zu erkennen ist, handelt es

sich insgesamt um sechs Threads, wobei im vorliegenden Experiment von den Interrupts 485 bis 487 praktisch nur der Interrupt 486 zum Einsatz kommt, da nur eine Änderung an einem Port erzeugt wird. Daher sind effektiv insgesamt nur vier Threads an der Reaktion beteiligt.

```
pi@raspberrypi:~ $ ps -em --format pid,tid,class,rtprio,psr,args
  PID   TID CLS RTPRIO PSR COMMAND
[...]
   55     - -         -   - [irq/49-3f200000]
    -    55 FF       50   0 -
[...]
  175     - -         -   - [irq/79-3f804000]
    -   175 FF       50   0 -
[...]
  934     - -         -   - ./loopback
    -   934 FF       89   0 -
    -   935 TS        -   0 -
[...]
  936     - -         -   - [irq/486-gpiolib]
    -   936 FF       50   3 -
  937     - -         -   - [irq/487-gpiolib]
    -   937 FF       50   3 -
  938     - -         -   - [irq/485-gpiolib]
    -   938 FF       50   3 -
[...]
```

Codeausschnitt 21: Ausgabe von ps zur Bestimmung der beteiligten Threads (gekürzt)

5.5.4. Experimentdurchführung

Vor Beginn der Messungen wurde die Prioritäten der Threads, welche für die Verarbeitung der beteiligten Interrupts zuständig sind, mit den Befehlen, welche im Codeausschnitt 22 gezeigt werden, neu festgelegt. Ansonsten lief die Experimentdurchführung analog zur Durchführung des letzen Experiments ab. Die CPU-Frequenz wurde auch diesmal auf 600 MHz festgesetzt.

```
sudo chrt -vf -p 95 55
sudo chrt -vf -p 95 175
sudo chrt -vf -p 95 938
sudo chrt -vf -p 95 936
sudo chrt -vf -p 95 937
```

Codeausschnitt 22: Festlegung neuer Prioritäten

5.5.5. Messergebnisse

Die Konfiguration entsprach auch diesmal jener, welche im Codeausschnitt 16 gezeigt wird. Die erwarteten Abnahmen der durchschnittlichen Reaktionszeit und der Streuung konnten in diesem Experiment nicht erzielt werden. Eine Auswertung der Messergebnisse ergab ganz im Gegenteil eine weitere Verschlechterung der durchschnittlichen Reaktionszeit auf etwa 258 µs, was einer Zunahme von etwa 4 % entspricht und eine weitere Zunahme der Standardabweichung auf etwa 17,8 µs, was einer weiteren Verschlechterung von fast 23 % entspricht. Die längste gemessene Reaktionszeit betrug etwa 540 µs.

Das eigentliche Ausmaß der Veränderung wird aber erst bei einer Betrachtung des zugehörigen Histogramms der Reaktionszeiten, welches in Abbildung 74 in Verhältnis 10:1 dargestellt ist, erkennbar.

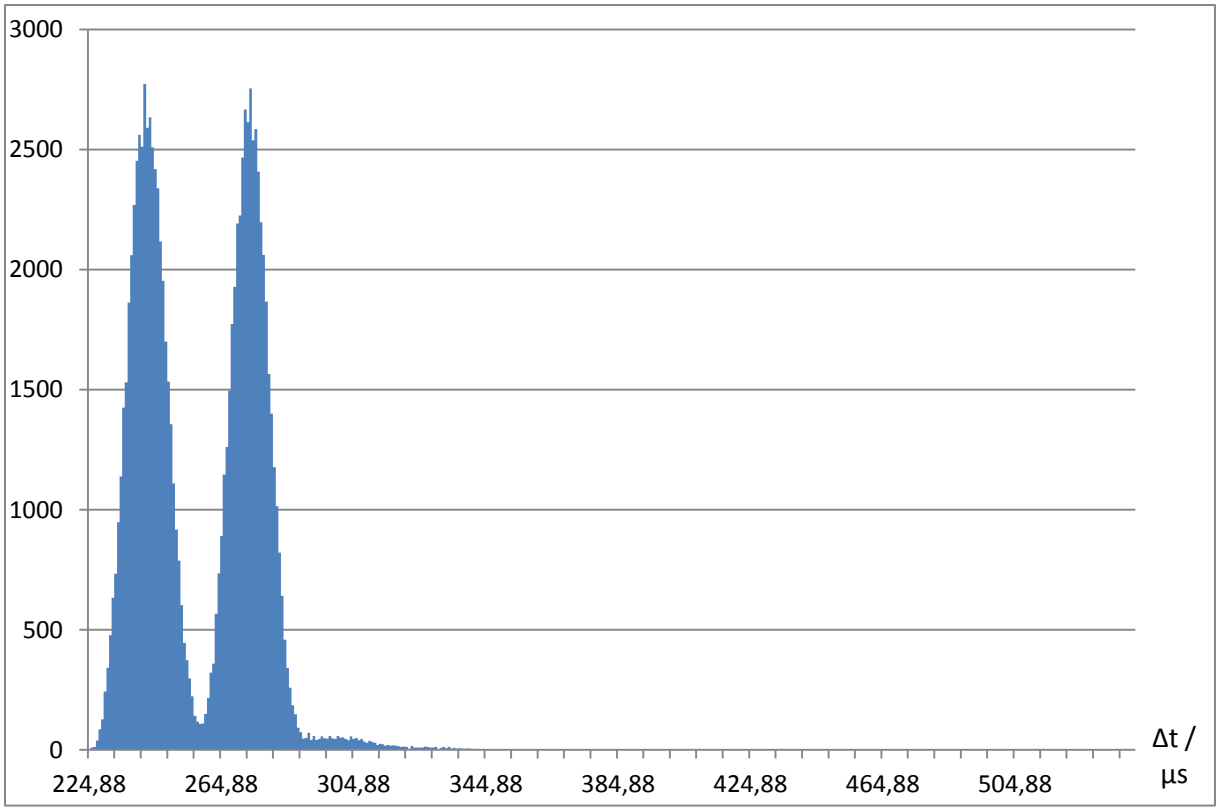


Abbildung 74: Histogramm der Reaktionszeiten nach Erhöhung der Thread-Prioritäten

Wie zu erkennen ist, hat sich der zuvor beobachtete unerwünschte Effekt, dass sich die Reaktionszeiten auf zwei Bereiche aufteilen, durch Erhöhung der Prioritäten der beteiligten Threads noch deutlich verstärkt. Die beiden Spitzen liegen diesmal etwa 30 µs auseinander.

Durch Erhöhung der Prioritäten der beteiligten Threads wurde versucht, den Einfluss anderer Threads, welche an der gewünschten Reaktion nicht direkt beteiligt sind, zu reduzieren. Dies hat aber zu keiner Verbesserung des Verhaltens geführt. Daraus ist zu schließen, dass das ungünstige Verhalten vermutlich nicht durch andere unbeteiligte Threads verursacht wird, sondern von den beteiligten Threads selbst. Wie dies zustande kommt und was dagegen getan werden kann, ist im Folgenden zu untersuchen.

5.6. RT Patch mit nur einem CPU-Kern

Die Ergebnisse des letzten Experiments haben nahegelegt, dass das ungünstige Verhalten von den vier bzw. sechs an der Reaktion des Systems unmittelbar beteiligten Threads selbst verursacht wird. Jeder einzelne dieser Threads führt aber bei jedem Durchlauf exakt dieselben Operationen aus. Es gibt keinen Grund zu der Annahme, dass sich das Verhalten von irgendeinem dieser Threads von einem Durchlauf zum nächsten derart stark ändern sollte um damit das beobachtete Verhalten erklären zu können.

Es fällt allerdings auf, dass keinem der Threads ein fester CPU-Kern zugewiesen ist. Der Raspberry Pi hat vier Kerne, es ist davon auszugehen, dass im Laufe des Experiments die einzelnen Threads von Kern zu Kern wechseln. Es ist nicht klar, wie oft dies geschieht, aber es ist davon auszugehen, dass dies von Zeit zu Zeit vorkommt. Es wäre denkbar, dass die Reaktionszeit davon abhängt, wie die einzelnen Threads vom System auf die CPU-Kerne verteilt werden. Die Ergebnisse des letzten Experiments legen nahe, dass es eine günstige und eine ungünstige Aufteilung geben könnte, welche jeweils etwa gleich oft gewählt werden.

5.6.1. Fragestellung

Sollte die aufgestellte Vermutung zutreffen, dürfte das unerwünschte Verhalten bei einem System mit nur einem CPU-Kern nicht auftreten. Dieses Experiment soll klären, ob eine Verbesserung des Verhaltens erzielt werden kann, wenn nur ein CPU-Kern eingesetzt wird. Sollte das unerwünschte Verhalten weiterhin auftreten, kann die Vermutung als widerlegt angesehen werden. Sollte das Verhalten nicht mehr auftreten, besteht eine hohe Wahrscheinlichkeit, dass die Vermutung richtig ist.

5.6.2. Experimentdesign

Es bestehen diverse Möglichkeiten das System auf einen CPU-Kern zu beschränken. In diesem Experiment soll die Kernel-Option „isolcpus“ eingesetzt werden. Im Unterschied zu anderen Möglichkeiten werden hierbei die ungenutzten Kerne nicht völlig abgeschaltet, sondern nur vom normalen Scheduling ausgeschlossen. Damit laufen auf den ungenutzten CPU-Kernen nur einige wenige Kernel-Threads, die unbedingt notwendig sind, wenn ein Kern nicht völlig abgeschaltet werden soll. Der große Vorteil dieser Option besteht darin, dass es bei Bedarf möglich ist Threads explizit den ungenutzten Kernen zuzuweisen. Dies könnte, falls in dieses Experiment eine Verbesserung des Verhaltens erzielt wird, für nachfolgende Experimente interessant sein.

Nach der Konfiguration der CPU-Kerne, soll das letzte Experiment wiederholt werden. Die CPU-Frequenz soll natürlich wieder auf 600 MHz festgelegt werden. Um vergleichbare Ergebnisse zu erzielen, soll, obwohl im letzten Experiment dadurch eine Verschlechterung des Verhaltens erzielt wurde, auch diesmal die Priorität der beteiligten Threads, welche Interrupthandler verarbeiten, erhöht werden.

5.6.3. Experimentdurchführung

Die Option „isolcpus=1-3“ wurde in die Datei „/boot/cmdline.txt“ hinzugefügt. Dadurch wird sie beim Hochfahren dem Kernel als Parameter übergeben und alle Kerne, außer dem Kern 0, werden vom normalen Scheduling ausgenommen. Damit diese Einstellung wirksam wird, wurde ein Neustart durchgeführt. Mit ps wurde überprüft, dass tatsächlich nur einige wenige Kernel-Threads auf den Kernen 1 bis 3 ausgeführt werden.

Danach wurde die CPU-Frequenz auf 600 MHz festgelegt, das Testprogramm wurde gestartet, die Prioritäten, der beteiligten Threads wurden, wie im letzten Experiment gezeigt, erhöht und die Messung wurde eingeleitet.

5.6.4. Messergebnisse

Auch für dieses Experiment wurden die gleichen Einstellungen für das Auswertungsprogramm verwendet, daher entsprach der obere Teil der Ausgabe der im Codeausschnitt 16 gezeigten Ausgabe. Allerdings konnte diesmal einer von den erfassten 100.000 Durchläufen nicht verarbeitet werden. Daher sah der untere Teil der Ausgabe abweichend, wie in Codeausschnitt 23 gezeigt, aus. Die Fehlerbeschreibung „no falling edge on SCL line“ sagt aus, dass bei einem der Durchläufe die Reaktion des Systems auf das Interruptsignal auf der INT Leitung so spät erfolgte, dass diese nicht mehr aufgezeichnet werden konnte, also außerhalb des Aufzeichnungsintervalls von 5 ms lag. Alle anderen Durchläufe konnten aber problemlos ausgewertet werden.

```
Results of dt_hw processing:
  total number of wave forms seen:                100000
  successfully processed wave forms:              99999
  number of waveforms not processed due to errors: 1
    invalid condition at first sample:    0
    no change at all:                     0
    some change before rising edge on A:  0
```

```
no INT signal:                          0
some unexpected change before INT:      0
no falling edge on SCL line:            1
unexpected change before first fSCL:    0
no rising edge on SCL line:             0
unexpected change before first rSCL:    0
not enough falling edges on SCL:        0
unexpected change before some fSCL:     0
not enough rising edges on SCL:         0
unexpected change before some rSCL:     0
no rising edge on OUT line:             0
unexpected change before rising OUT:    0
no rising edge on channel B:            0
unexpected change before rising B:      0
unexpected change after rising B:       0
```

Codeausschnitt 23: Abweichende Ausgabe beim Experiment mit nur einem CPU-Kern

Eine Auswertung ergab die bisher höchste durchschnittliche Reaktionszeit von etwa 291 µs und eine immer noch sehr hohe Standardabweichung von etwa 15,5 µs. Auch wenn diese Werte schlecht sind, kann im zugehörigen Histogramm insgesamt eine Verbesserung des Verhaltens beobachtet werden.

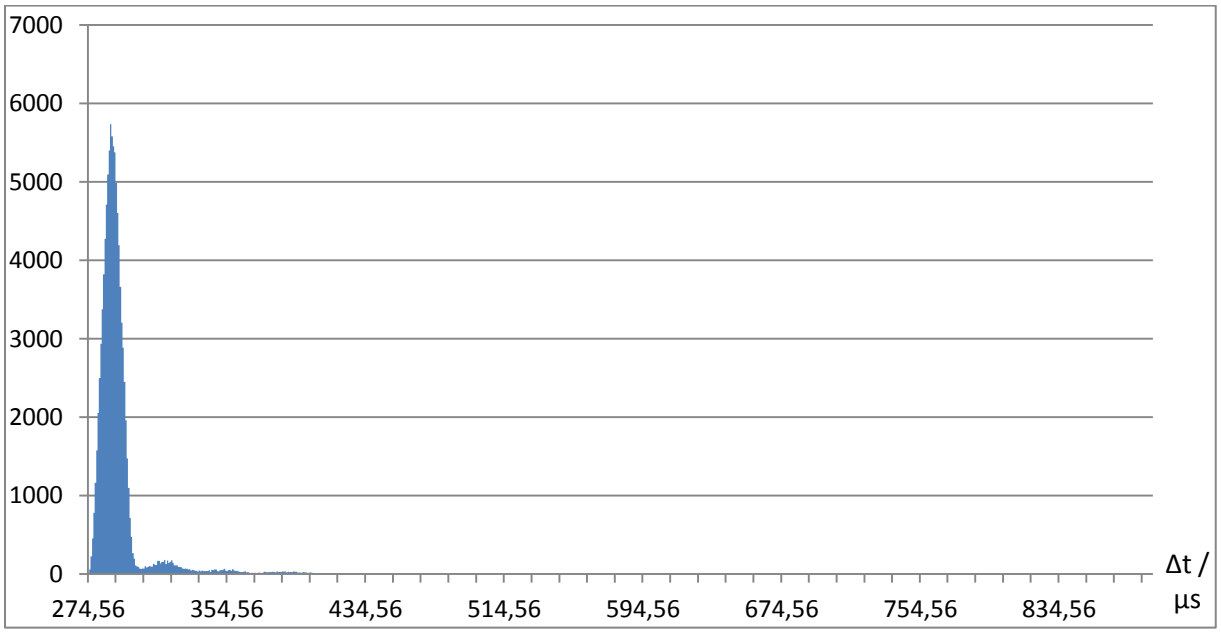


Abbildung 75: Histogramm der Reaktionszeiten bei Nutzung nur eines CPU-Kerns

Abbildung 75 zeigt das zugehörige Histogramm im Verhältnis 10:1. Die längste Reaktionszeit, die erfolgreich gemessen werden konnte, betrug über 887 µs. Es ist zu erkennen, dass es immer noch eine Ansammlung von Werten außerhalb der Haupansammlung gibt, allerdings ist diese schwächer ausgeprägt, als zuvor. Es scheint wahrscheinlich, dass die aufgestellte Vermutung, dass die Reaktionszeit von der Verteilung der Threads auf die CPU-Kerne abhängt oder durch die Umverteilung selbst beeinflusst wird, richtig ist. Die Ergebnisse lassen aber keinen eindeutigen Schluss zu. Das Problem muss im Folgenden weiter untersucht werden.

5.7. RT Patch mit verschiedenen Verteilungen der Threads

Die Ergebnisse des letzten Experiments deuten darauf hin, dass die Reaktionszeit von der Verteilung der beteiligten Threads auf die verfügbaren CPU-Kerne abhängt. Um dies weiter zu untersuchen sollen in diesem Experiment verschiedene Verteilungen der Threads miteinander verglichen werden.

5.7.1. Fragestellung

Dieses Experiment soll klären, ob die Reaktionszeit tatsächlich von der Verteilung der beteiligten Threads auf die verfügbaren CPU-Kerne abhängt und ob dies der Grund für das in Abbildung 73 beobachtete Verhalten ist.

5.7.2. Experimentdesign

Die im letzten Experiment genutzte Isolation der Kerne 1 bis 3 soll weiterhin genutzt werde. Damit stehen drei praktisch ungenutzte Kerne zur freien Verfügung. Auf diese freien CPU-Kerne sollen die beteiligten Threads verteilt werden. Es sollen Messungen mit unterschiedlichen Verteilungen durchgeführt werden und die Ergebnisse verglichen werden. Insbesondere sind drei Verteilungen zu testen. Einmal sollen alle beteiligten Threads auf einen einzigen freien Kern gelegt werden. Dies sollte die ungünstigste Konstellation darstellen und daher eine längere durchschnittliche Reaktionszeit zur Folge haben. Als nächstes sollen alle beteiligten Threads möglichst gleichmäßig über alle drei freien Kerne verteilt werden. Dies sollte die günstigste Konstellation darstellen, was die Reaktionszeit verkürzen sollte. Abschließend soll dem Scheduler die Möglichkeit gegeben werden, die beteiligten Threads frei auf die drei ungenutzten Kerne zu verteilen. Es wird erwartet, dass dabei eine Mischung aus den beiden vorhergehenden Messungen erzielt wird. Die Verteilung sollte mehr oder weniger der in Abbildung 73 oder der in Abbildung 74 beobachteten Verteilung entsprechen. Dies wäre dann ein klares Zeichen dafür, dass die Reaktionszeit von der Verteilung der Threads abhängt und dass der Scheduler für das ungünstige Verhalten verantwortlich ist.

5.7.3. Experimentdurchführung

Die Durchführung der einzelnen Messungen erfolgte ähnlich wie beim letzten Experiment. Durch die Kernel-Option "isolcpus=1-3" wurden fast alle Threads auf den CPU-Kern 0 beschränkt. Die CPU-Frequenz wurde auf 600 MHz festgelegt. Mit Hilfe des Inhalts von „/proc/interrupts“ und der Ausgabe von ps wurden die beteiligten Threads bestimmt. Den Threads, welche für die Verarbeitung von Interrupts verantwortlich sind, wurde eine Priorität von 95 zugewiesen.

Mit dem Kommando taskset wurden dann zunächst alle beteiligten Threads dem CPU-Kern 1 zugewiesen und eine Messreihe wurde durchgeführt. Danach wurden alle beteiligten Threads über die drei freien Kerne verteilt. Dabei wurde den vier Threads, welche für die Verarbeitung von GPIO bezogenen Interrupts zuständig sind, ein gemeinsamer Kern zugewiesen. Dem Interrupthandler für den I^2C Bus, sowie dem Thread des Testprogramms wurde jeweils ein eigener Kern zugewiesen (vgl. Codeausschnitt 24). Mit dieser Konfiguration wurde eine weitere Messreihe durchgeführt. Anschließend wurde dem Scheduler die Möglichkeit gegeben die beteiligten Threads selbstständig auf die drei ungenutzten Kerne zu verteilten und eine weitere Messreihe wurde erstellt.

```
# GPIO
sudo taskset -p 2 55
sudo taskset -p 2 942
sudo taskset -p 2 950
sudo taskset -p 2 951
# I2C
sudo taskset -p 4 165
# loopback
sudo taskset -p 8 940
```

Codeausschnitt 24: Zuweisung von Threads zu CPU-Kernen

5.7.4. Messergebnisse

Anhand der Messergebnisse dieses Experiments konnte ganz klar nachgewiesen werden, dass die Reaktionszeit, wie vermutet, tatsächlich von der Verteilung der beteiligten Threads auf die verfügbaren CPU-Kerne abhängt. Abbildung 76 veranschaulicht das Ergebnis. Dargestellt ist ein Ausschnitt einer Überlagerung der Histogramme der drei durchgeführten Messreihen im Verhältnis 3:1. Die Ausgabe der Konfiguration entsprach in allen Fällen der Darstellung in Codeausschnitt 16.

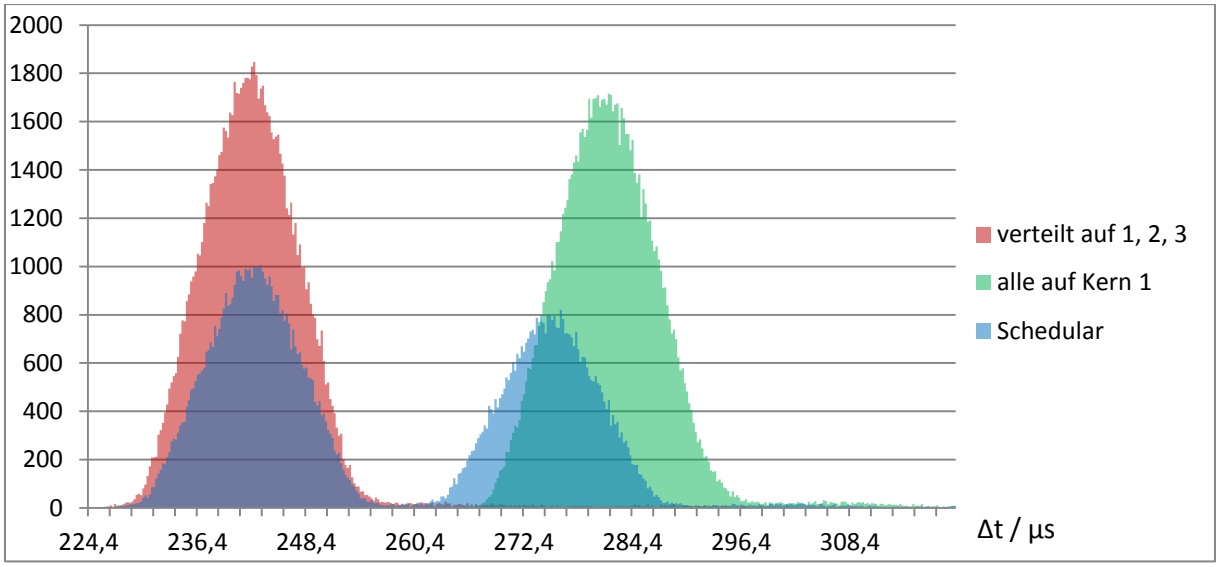


Abbildung 76: Vergleich der Reaktionszeiten in Abhängigkeit der Verteilung der Threads

Vergleicht man den Durchlauf, bei dem alle beteiligten Threads einem CPU-Kern zugewiesen wurden, mit dem Durchlauf, in dem die beteiligten Threads auf die drei freien Kerne verteilt wurden, ist deutlich zu erkennen, dass die Reaktionszeit stark zunimmt, wenn alle Threads auf nur einem Kern ausgeführt werden. In beiden Fällen ergibt sich aber eine Verteilung, welche der Normalverteilung ähnelt. Die Verteilungen unterscheiden sich praktisch nur durch ihren Mittelwert.

Wird dem Scheduler die Entscheidung über die Verteilung der Threads auf die drei freien CPU-Kerne überlassen, ändert dieser offensichtlich von Zeit zu Zeit die Verteilung der Threads, so dass die Reaktion in manchen Fällen schneller und in manchen Fällen langsamer erfolgt. Die Verteilung der Reaktionszeiten im Histogramm ist in diesem Fall dann eine Kombination der Verteilungen, welche sich für die einzelnen genutzten Zuteilungen ergeben. Dies ist in Abbildung 76 gut zu erkennen. Neben den untersuchten Zuteilungen, sind noch viele weitere denkbar. Die zweite Spitze der Verteilung, welche unter Einsatz des Schedulers ermittelt wurde, deckt sich nicht ganz mit der Verteilung, welche ermittelt wurde, indem alle beteiligten Threads einem einzigen Kern zugewiesen wurden. Dies legt nahe, dass der Scheduler diese Verteilung, welche die schlechteste denkbare Verteilung sein dürfte, nicht einsetzt. Es wird aber offensichtlich eine ähnlich schlechte Verteilung der Threads auf die CPU-Kerne genutzt. Die zweite Spitze könnte auch die Folge des Einsatzes mehrerer ähnlich schlechter Zuteilungen von Threads zu CPU-Kernen sein. Jedenfalls hat dieses Experiment klar belegt, dass der Scheduler für die ungünstigen Verteilungen der Reaktionszeiten, welche in Abbildung 73 und in Abbildung 74 beobachtet wurden, verantwortlich ist.

Die Ergebnisse der Messreihe, bei der die beteiligten Threads manuell auf die drei freien CPU-Kerne verteilt wurden, wurden weiter ausgewertet. Es wurden eine durchschnittliche Reaktionszeit von etwa 242,7 µs und eine Standardabweichung von etwa 7,0 µs ermittelt. Die Auswertung der Histogramme der einzelnen Teile der Reaktionszeit (vgl. Abschnitt 5.2), welche ebenfalls ermittelt wurden, ergab keine Auffälligkeiten.

5.8. Vergleich zwischen normalem Kernel und Kernel mit RT Patch

Im letzten Experiment konnten mit dem CONFIG_PREEMPT_RT Patch insbesondere durch explizite Zuweisung von Threads zu CPU-Kernen gute Resultate erzielt werden. Es konnte eine durchschnittliche Reaktionszeit von 242,7 µs mit einer Standardabweichung von nur 7,0 µs erzielt werden. Diese Werte sind allerdings immer noch minimal schlechter, als die besten Werte, welche ohne den CONFIG_PREEMPT_RT Patch erzielt werden konnten (vgl. Abschnitt 5.3). Die durchschnittliche Reaktionszeit ist um etwa 8 % schlechter, während die Standardabweichung um etwa 4 % zugenommen hat. Der Unterschied ist nicht groß, aber eigentlich wurde vom CONFIG_PREEMPT_RT Patch eine deutliche Verringerung der Standardabweichung erwartet.

Alle bisherigen Experimente wurden unter weitestgehend idealen Bedingungen durchgeführt. Die CPU des Raspberry Pi wurde während der Messungen nicht durch rechenintensive andere Programme belastet, über das Netzwerk wurden praktisch keine Pakete übertragen und es waren keine externen Komponenten wie Maus, Tastatur oder Bildschirm angeschlossen. Unter diesen Bedingungen führt der CONFIG_PREEMPT_RT Patch offensichtlich zu keiner Verbesserung des Verhaltens. Es ist aber durchaus wahrscheinlich, dass das System unter Belastung mit dem CONFIG_PREEMPT_RT Patch ein besseres Verhalten zeigt als ohne diesen.

5.8.1. Fragestellung

In diesem Experiment soll das Verhalten des Systems unter starker Belastung mit und ohne den CONFIG_PREEMPT_RT Patch verglichen werden. Es wird angenommen, dass sich ohne den Patch unter Belastung sowohl die durchschnittliche Reaktionszeit als auch die Standardabweichung der Reaktionszeit deutlich erhöhen. Auch mit dem Patch ist eine Verschlechterung beider Werte zu erwarten, allerdings sollte diese nicht so stark ausfallen. Diese Hypothese ist in diesem Experiment zu überprüfen.

5.8.2. Experimentdesign

Um einen Vergleich anstellen zu können, sollen zwei Messreihen durchgeführt werden. Bei der ersten Messreihe soll der Kernel mit dem CONFIG_PREEMPT_RT Patch eingesetzt werden, welcher im letzten Experiment genutzt wurde, während bei der zweiten ein normaler Kernel, wie er im Experiment im Abschnitt 5.3 zum Einsatz kam, verwendet werden soll.

Ansonsten sollen bei beiden Messreihen soweit möglich gleiche Bedingungen herrschen. In beiden Fällen sollen CPU-Kerne mit der Kernel-Option „isolcpus“ isoliert werden. Im ersten Fall sollen, wie beim letzten Experiment die Kerne 1 bis 3 isoliert werden. Auf diese sollen dann die sechs beteiligten Threads wie zuvor manuell verteilt werden. Im zweiten Fall werden die Interrupthandler nicht im Threadkontext verarbeitet. Daher können diese nicht explizit CPU-Kernen zugewiesen werden. Dennoch soll um möglichst ähnliche Bedingungen zu schaffen, ein CPU-Kern isoliert werden. Auf diesem soll dann der Thread des Testprogramms ausgeführt werden. Dies scheint dem ersten Fall am nächsten zu kommen. In beiden Fällen ist die CPU-Frequenz auf 600 MHz festzulegen.

In beiden Fällen muss eine vergleichbare Belastung erzeugt werden. Diese muss für die Dauer des Experiments anhalten und muss reproduzierbar sein. Es sollen in beiden Fällen vier Arten von Belastung erzeugt werden. Die CPU soll durch einen rechenintensiven Vorgang belastet werden, Lese- und Schreiboperation sollen auf dem Datenträger, also der SD-Karte, erfolgen, das Netzwerk soll ausgelastet werden und externe Komponenten, insbesondere ein Bildschirm, sollen angeschlossen werden.

CPU- und Datenträgerbelastung

Um eine starke Belastung der CPU und häufige Zugriffe auf den Datenträger zu erzeugen, soll der Linux-Kernel während des Experiments mit mehreren Threads kompiliert werden. Da ein Kompilieren des Linux-Kernels auf dem Raspberry Pi mehrere Stunden in Anspruch nimmt, ist sichergestellt, dass die Belastung über die gesamte Dauer des Experiments, welches nur etwas länger als eine halbe Stunde dauern sollte, anhält. Die Belastung kann auch relativ gut reproduziert werden.

Netzwerkbelastung

Eine einfache Möglichkeit das Netzwerk zu belasten, bestünde darin, eine beliebige Datei aus dem Internet herunterzuladen. Diese Methode hätte aber wesentliche Nachteilte. Besonders gravierend wären die Tatsachen, dass die Übertragungsrate durch die Verbindungsgeschwindigkeit zum Internet beschränkt wäre und dass die Belastung nur schlecht reproduzierbar wäre, da die Übertragungsrate abhängig von der Belastung des Netzwerks und des sendenden Servers wäre und variieren könnte.

Daher soll ein weiterer Computer im lokalen Netzwerk genutzt werden um eine Belastung des Netzwerkanschlusses zu erzielen. Sowohl auf dem anderen lokalen Computer wie auch auf dem Raspberry Pi soll das Tool nc zum Einsatz kommen. Dieses soll genutzt werden um eine einfache TCP Verbindung zwischen beiden Geräten aufzubauen. Der lokale Computer soll dann den Inhalt der Datei /dev/zero mit maximaler Geschwindigkeit übertragen. Der Raspberry Pi soll die Pakete empfangen und in /dev/null schreiben. Dies erzeugt eine hohe, gleichmäßige, gut reproduzierbare Netzwerkbelastung, ohne eine zusätzliche Belastung des Datenträgers zu verursachen. Die Übertragung kann zudem beliebig lange aufrechterhalten werden.

Externe Komponenten

Als externe Komponenten sollen ein Bildschirm und eine Tastatur angeschlossen werden, wobei die Tastatur während des Experiments nicht betätigt werden soll, da diese Art der Belastung schwer zu reproduzieren wäre. Auf dem Bildschirm soll die Kommandozeile angezeigt werden. Auf dieser sollen aber weder Ein- noch Ausgaben erfolgen.

5.8.3. Experimentdurchführung

Wie geplant wurden zwei Messreihen durchgeführt. Zunächst eine mit dem CONFIG_PREEMPT_RT Patch und danach eine ohne diesen. Beide Messreihen, aber insbesondere die erste, wurden jeweils noch einige Male wiederholt, wie der weiteren Beschreibung zu entnehmen ist.

Mit CONFIG_PREEMPT_RT Patch

Zunächst wurden planmäßig ein Bildschirm über HDMI und eine USB-Tastatur angeschlossen. Der bereits in den letzten Experimenten genutzte mit dem CONFIG_PREEMPT_RT Patch versehene Kernel kam auch in diesem Experiment zu Einsatz. Die CPU-Kerne 1 bis 3 wurden isoliert und die CPU-Frequenz wurde auf 600 MHz festgelegt. Das Testprogramm wurde gestartet und die sechs beteiligten Threads wurden ermittelt. Wie zuvor wurde die Priorität der beteiligten Threads, welche Interrupthandler verarbeiten, erhöht. Die Threads wurden dann wie zuvor auf die drei freien CPU-Kerne verteilt.

Auf der Konsole, welche auf dem angeschlossenen Bildschirm angezeigt wurde, wurde das Kommando „setterm --blank 0“ ausgeführt, um sicherzustellen, dass der Bildschirm nicht während des Experiments auf ein komplett schwarzes Bild umschaltet, was standardmäßig nach einigen Minuten erfolgt. Mit „nc -l 1234 > /dev/null“ wurde der Aufbau einer Verbindung vorbereitet. Auf einen anderen Computer im lokalen Netzwerk wurde „nc 192.168.178.25 1234 < /dev/zero“ ausgeführt um die gewünsch-

te Netzwerklast zu erzeugen. Schließlich wurde mit „make -j6 zImage modules dtbs“ das Kompilieren eines zuvor bereitgelegten Kernels eingeleitet. Danach wurde der Messvorgang gestartet.

Die geplanten 100.000 Messungen konnten beim ersten Versuch allerdings nicht planmäßig erhoben werden, da das System nach etwa der Hälfte der Zeit, die notwendig gewesen wäre um den Vorgang abzuschließen, ohne erkennbaren Grund einfror. Trotz mehrmaliger Wiederholung der gesamten Experimentdurchführung konnte die Messreihe nie vollständig durchgeführt werden, da das System immer wieder einfror.

Das Kernel-Log gab keine Auskunft über den Grund des Einfrierens. Internetrecherchen ergaben aber, dass das Problem vom genutzten USB Treiber verursacht werden könnte. [97] Im Kernel-Log wurden bezüglich dieses Treibers einige Warnungen gefunden, was den Verdacht weiter erhärtete. Der USB Treiber nutzt eine spezielle Art von Interrupts, FIQ genannt. Diese scheinen vom CONFIG_PREEMPT_RT Patch noch nicht unterstützt zu werden, wodurch es zu den Problemen kommt. Die Recherche ergab ebenfalls, dass es möglich sein müsste einen alternativen USB Treiber oder zumindest eine andere Methode innerhalb des Treibers zu nutzen. Dies soll durch Hinzufügen der Optionen „dwc_otg.fiq_enable=0 dwc_otg.fiq_fsm_enable=0 dwc_otg.nak_holdoff=0“ zur Kommandozeile des Kernels erreicht werden können. Folglich wurden diese in „/boot/cmdline.txt“ hinzugefügt. Nach einem Neustart konnte bestätigt werden, dass FIQs nicht mehr benutzt werden. Danach wurde das Experiment mehrere Male wiederholt. Es traten keine Probleme mehr auf. Die gewünschten Messwerte konnten problemlos ermittelt werden.

Das Problem wurde also offensichtlich vom genutzten USB Treiber verursacht. Obwohl der inkompatible USB Treiber auch in allen vorhergehenden Experimenten eingesetzt wurde, wurden diese davon nicht beeinträchtigt. Dies ist leicht zu erklären. In allen vorhergehenden Experimenten wurde das USB Interface kaum genutzt. Erst durch die starke Belastung der Netzwerkschnittstelle, welche intern über das USB Interface angebunden ist, kam es zu einer starken Nutzung der USB Schnittstelle. Dadurch kamen auch die FIQs vermehrt zum Einsatz, wodurch die Wahrscheinlichkeit des Auftretens eines Fehlers offensichtlich signifikant zunahm.

Ein Vergleich der Werte der letztendlich erfolgreich durchgeführten Messreihen mit den Werten der unvollständigen Messreihen, welche vor dem Wechsel des USB Treibers durchgeführt wurden, ergab eine leichte Verringerung der Streuung beim Einsatz des alternativen USB Treibers, welcher vom RT Patch unterstützt wird. Da bei den vorherigen Experimenten die USB Schnittstelle kaum verwendet wurde, ist nicht davon auszugehen, dass sich die Ergebnisse dieser Experimente beim Einsatz des alternativen Treibers nennenswert verändern würden. Daher wurden diese mit dem alternativen USB Treiber nicht wiederholt.

Ohne CONFIG_PREEMPT_RT Patch

Nach einigen Wiederholungen des ersten Teils des Experiments wurde der zweite Teil durchgeführt. Dabei wurde der Kernel, welcher im Experiment im Abschnitt 5.3 genutzt wurde, verwendet. Der CPU-Kern 3 wurde isoliert, die CPU-Frequenz wurde auf 600 MHz festgelegt, das Testprogramm wurde gestartet und der Thread des Testprogramms, welcher die Reaktion erzeugt, wurde dem freien CPU-Kern zugewiesen. Danach wurde analog zum ersten Teil des Experiments eine Lastsituation erzeugt. Da der Kernel bereits im ersten Teil mit sechs Threads kompilier wurde, obwohl nur ein Kern zur Verfügung stand, war hier keine Änderung notwendig um die nun verfügbaren drei Kerne auszulasten.

5.8.4. Messergebnisse

Die Ergebnisse dieses Experiments haben die Annahme, dass das zeitliche Verhalten des Systems unter Volllast mit dem CONFIG_PREEMPT_RT Patch deutlich besser ist als ohne diesen Patch, klar bestätigt. Der Unterschied im Verhalten mit und ohne den Patch war so deutlich, dass er schon an der Fehlerstatistik des Auswertungsprogramms erkennbar ist. Wie bereits erwähnt, wurden die einzelnen Teile dieses Experiments mehrfach wiederholt. Die Ergebnisse der einzelnen Wiederholungen, waren so ähnlich, dass hier repräsentativ jeweils nur die Ergebnisse einer Messreihe mit RT Patch und einer Messreihe ohne RT Patch präsentiert werden.

Konfiguration und Fehlerstatistik

Während beim Einsatz des CONFIG_PREEMPT_RT Patches alle 100.000 Durchläufe der Messreihe erfolgreich ausgewertet werden konnten, so dass die Ausgabe der Konfiguration und der Fehlerstatistik der Ausgaben in Codeausschnitt 16 entsprach, konnten beim Versuch ohne den CONFIG_PREEMPT_RT Patch nur knapp 95 % der Durchläufe erfolgreich ausgewertet werden. Codeausschnitt 25 zeigt die erstellte Fehlerstatistik beim Durchlauf ohne den RT Patch. Wie zu erkennen ist, ist der Fehler „no falling edge on SCL line“ mit Abstand am häufigsten aufgetreten. Dies bedeutet, dass das System auf den Interrupt, welcher bei der Eingabeänderung erzeugt wurde, so spät reagiert hat, dass dies nicht mehr erfasst werden konnte. Die Reaktion erfolgte also mehr als 5 ms nach der steigenden Flanke auf der Eingabeleitung. Am zweithäufigsten sind zusammengerechnet die Fehler „no rising edge on OUT line“ und „no rising edge on channel B“ aufgetreten. Diese sagen aus, dass das System zwar auf den Interrupt noch innerhalb der Aufzeichnungsdauer reagiert hat, die Gesamtreaktion aber so lange gedauert hat, dass das Endergebnis, also die Änderung an der Ausgabeleitung, mehr als 5 ms nach der Änderung an der Eingabeleitung erfolgte. Die Fehler „no rising edge on SCL line“, „not enough falling edges on SCL“ und „not enough rising edges on SCL” dürften aufgetreten sein, wenn das System auf den Interrupt kurz vor Ende der Aufzeichnung also annähernd 5 ms nach der Eingabeänderung reagiert hat. So konnte zwar noch der Anfang der Datenübertragung über den I^2C Bus erkannt werden, die Übertragung dauerte aber über das Ende der Aufzeichnung hinaus an.

```
Results of dt_hw processing:
  total number of wave forms seen:                100000
  successfully processed wave forms:              94925
  number of waveforms not processed due to errors: 5075
    invalid condition at first sample:    191
    no change at all:                     0
    some change before rising edge on A:  16
    no INT signal:                        0
    some unexpected change before INT:    2
    no falling edge on SCL line:          4346
    unexpected change before first fSCL:  0
    no rising edge on SCL line:           5
    unexpected change before first rSCL:  0
    not enough falling edges on SCL:      24
    unexpected change before some fSCL:   0
    not enough rising edges on SCL:       25
    unexpected change before some rSCL:   0
    no rising edge on OUT line:           297
    unexpected change before rising OUT:  0
    no rising edge on channel B:          169
    unexpected change before rising B:    0
    unexpected change after rising B:     0
```

Codeausschnitt 25: Fehlerstatistik beim Belastungstest ohne RT Patch

Die übrigen Fehler weisen darauf hin, dass einzelne Reaktionen des Systems so lang gedauert haben, dass beim Auftreten der nächsten steigenden Flanke auf der Eingabe, die vorherige Reaktion des Systems noch nicht abgeschlossen war. Dadurch entsprachen die Bedingungen am Anfang der Aufzeichnung nicht den Erwartungen, was als Fehler eingestuft wurde.

Verteilung der Reaktionszeiten

Beim Testlauf mit dem RT Patch wurde eine durchschnittliche Reaktionszeit von etwa 314 µs ermittelt, was einer Verlängerung der Reaktionszeit um etwa 29 % im Vergleich zum unbelasteten System entspricht. Als Standardabweichung wurden knapp 19,0 µs berechnet, was einer Verschlechterung von etwa 171 % entspricht. Trotz dieser prozentual hohen Verschlechterung hielt sich die Reaktionszeit aber insgesamt in Grenzen. Die längste gemessene Reaktionszeit betrug etwa 479 µs. Abbildung 77 zeigt das ermittelte Histogramm der Reaktionszeiten im Verhältnis 6:1.

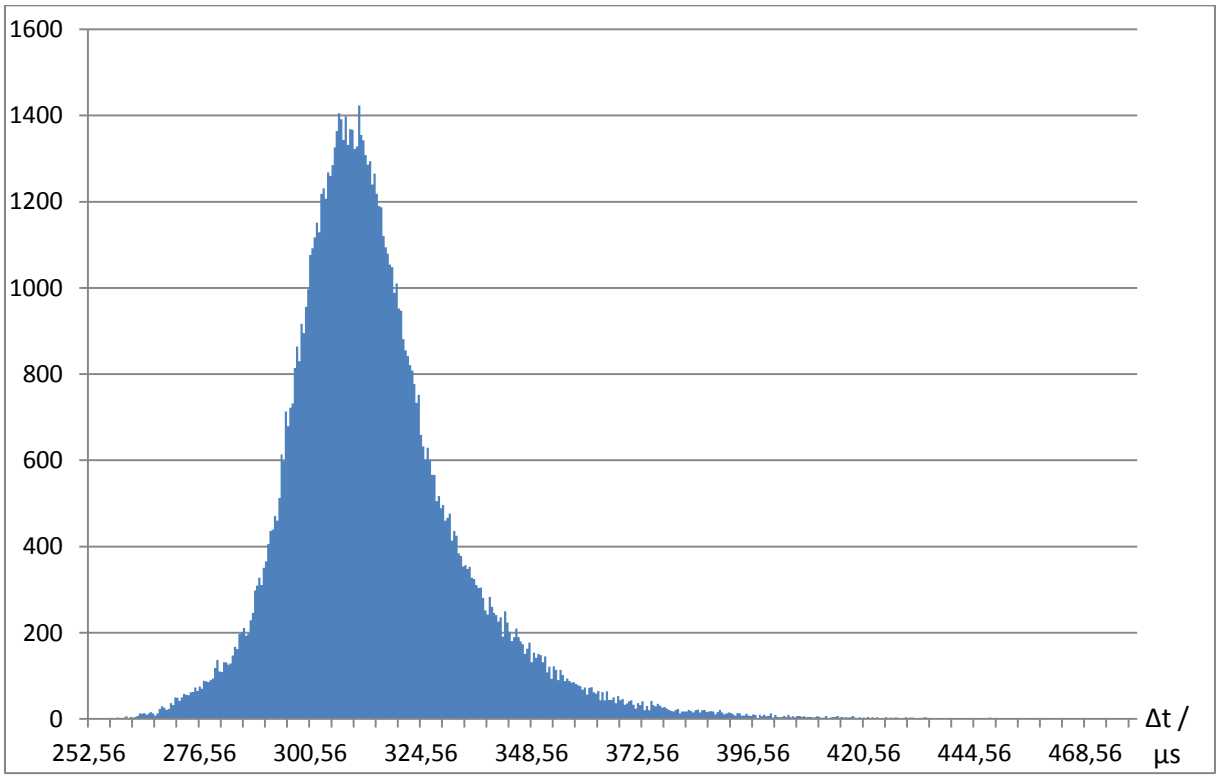


Abbildung 77: Histogramm der Reaktionszeiten beim Belastungstest mit RT Patch

Beim Versuch ohne RT Patch lagen etwa 5 % der Reaktionszeiten außerhalb des Messbereichs von 5 ms. Diese wurden in der weiteren Auswertung nicht berücksichtigt. Auf Basis der Messwerte, welche innerhalb des Messbereichs lagen, wurde für das System ohne CONFIG_PREEMPT_RT Patch eine durchschnittliche Reaktionszeit von etwa 1,1 ms mit einer Standardabweichung von etwa 1,2 ms ermittelt. Die längste erfolgreich gemessene Reaktionszeit lag bei knapp 5 ms. Aus der Fehlerstatistik ist zu entnehmen, dass die Reaktion in vielen Fällen noch länger gedauert hat.

Abbildung 78 zeigt das Histogramm der Reaktionszeiten beim Belastungstest ohne RT Patch im Verhältnis 100:1. Abbildung 79 zeigt eine Detailansicht des unteren Bereichs dieses Histogramms im selben Verhältnis. Obwohl eine klare Spitze vorhanden ist, ist deutlich zu erkennen, dass die Messwerte über den gesamten Messbereich verteilt sind und die Anzahl der Messungen mit einer bestimmten Reaktionszeit gegen 5 ms nur langsam abnimmt.

Dieses Experiment hat klar gemacht, dass der CONFIG_PREEMPT_RT Patch zwar bei geringer Belastung des Systems keinen signifikanten Vorteil mit sich bringt, aber bei einem belasteten System

unentbehrlich ist. Da sich die Belastung des Systems jederzeit plötzlich ändern kann, ist der Einsatz des RT Patches beim vorliegenden System dringend anzuraten.

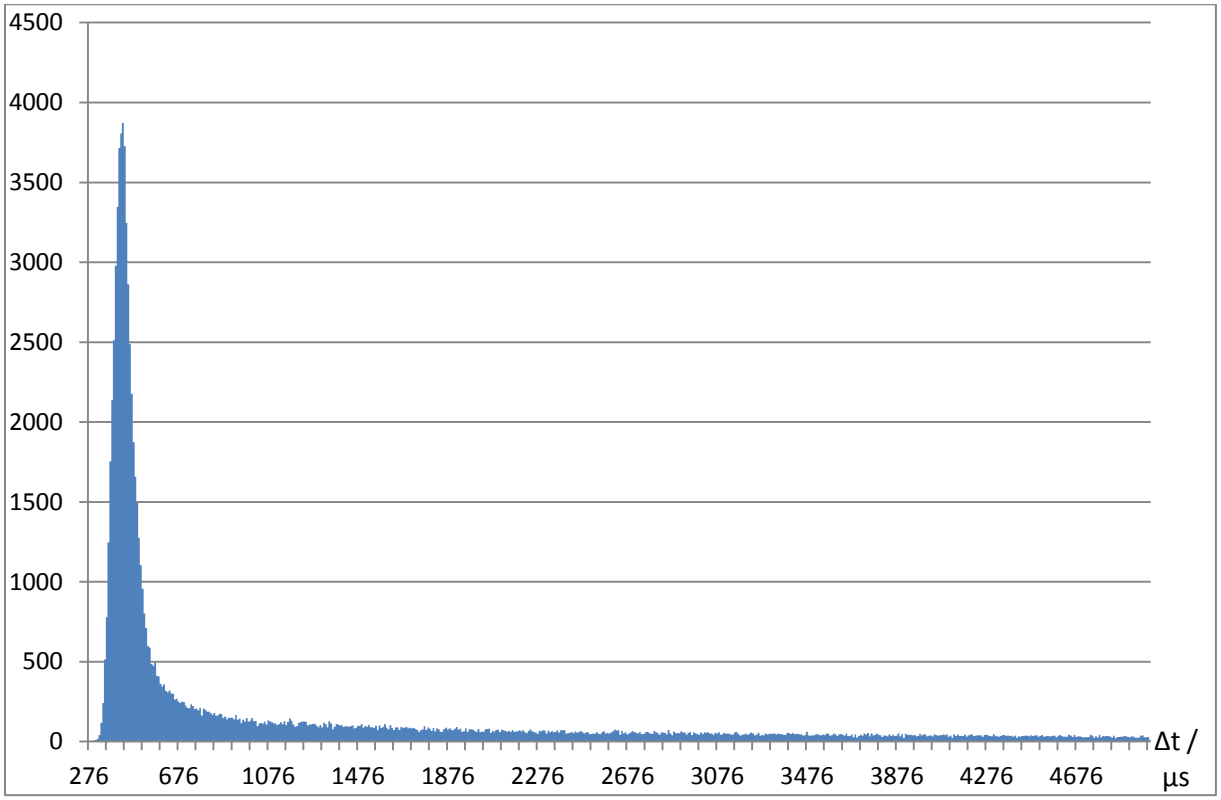


Abbildung 78: Histogramm der Reaktionszeiten beim Belastungstest ohne RT Patch

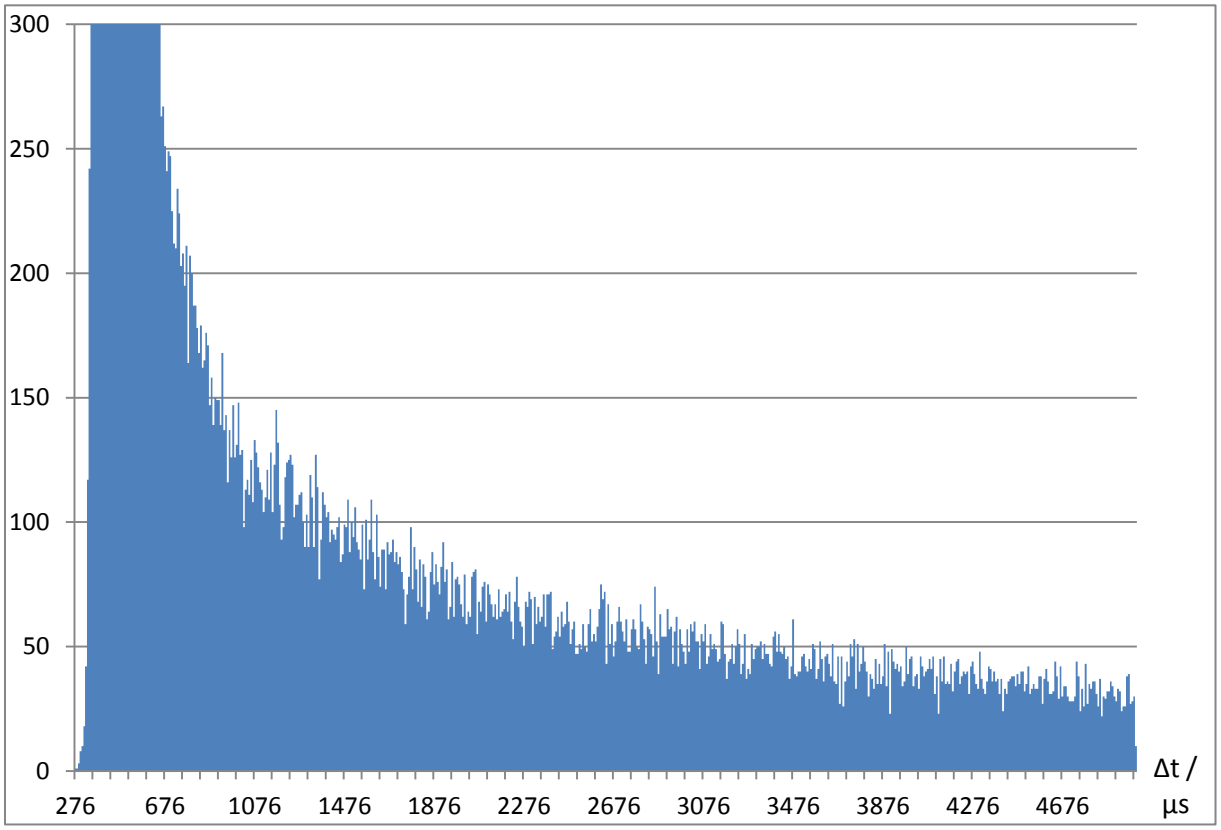


Abbildung 79: Detailansicht des Histogramms der Reaktionszeiten ohne RT Patch

5.9. Zusammenfassung

Im Verlauf dieses Kapitels wurden nach und nach weitere Konfigurationen vorgenommen um das zeitliche Verhalten des Systems zu verbessern. Im letzten Experiment konnte schließlich gezeigt werden, dass mit der genutzten Konfiguration selbst bei starker Belastung des Raspberry Pi ein sehr gutes zeitliches Verhalten erreicht werden kann.

Aufgrund der durchgeführten Experimente können natürlich keine Garantien bezüglich des zeitlichen Verhaltes abgegeben werden. Es können nur statistische Aussagen getroffen werden. Auch wenn während des Experiments keine der 100.000 gemessenen Reaktionszeiten die Obergrenze von 5 ms überschritten hat, besteht durchaus die Möglichkeit, dass dies von Zeit zu Zeit passiert, allerdings nur sehr selten. Die längste im letzten Experiment gemessene Reaktionszeit lag um mehr den Faktor 10 unterhalb der geforderten Obergrenze von 5 ms. Dabei muss berücksichtigt werden, dass in den durchgeführten Experimenten ausschließlich einzelne isolierte Eingabeänderungen untersucht wurden, während die Einhaltung einer Reaktionszeit von maximal 5 ms für jede beliebige Änderung gefordert wurde. Da die Obergrenze aber so deutlich unterschritten wurde, ist davon auszugehen, dass es auch bei mehreren kurz aufeinanderfolgenden Änderungen diesbezüglich keine Probleme geben sollte.

Angesichts des geplanten Einsatzbereichs bestand von vorherein nicht die Forderung die Einhaltung von Reaktionszeiten garantieren zu können, es wurde nur gefordert, dass diese mit einer sehr hohen Wahrscheinlichkeit eingehalten werden. Dies konnte durch die unternommenen Experimente erfolgreich gezeigt werden. Im Folgen sind die einzelnen Schritte zusammengefasst, welche durchzuführen sind um die Konfiguration herzustellen, mit der die besten Ergebnisse erzielt werden konnten.

Durchzuführende Schritte:	vgl. Abschnitt
• Linux-Kernel mit CONFIG_PREEMPT_RT Patch installieren o vgl. Codeausschnitt 18	5.4.3
• CPU-Kerne 1 bis 3 isolieren o `isolcpus=1-3` in /boot/cmdline.txt	5.6.2 f.
• FIQ USB Treiber deaktivieren o `dwc_otg.fiq_enable=0 dwc_otg.fiq_fsm_enable=0 dwc_otg.nak_holdoff=0` in /boot/cmdline.txt	5.8.3
• Neustart durchführen o `sudo reboot`	
• CPU-Frequenz festlegen o vgl. Codeausschnitt 17	5.3.3
• Testprogramm starten o `sudo ./loopback`	
• genutzte Interrupts bestimmen o `cat /proc/interrupts`	5.5.3
• beteiligte Threads bestimmen o `ps -em --format pid,tid,class,rtprio,psr,args`	5.5.3
• Priorität der Interrupthandler erhöhen o vgl. Codeausschnitt 22	5.5.4
• beteiligte Threads auf freie CPU-Kerne verteilen o vgl. Codeausschnitt 24	5.7.3

Wie am Anfang dieses Kapitels erwähnt, sollten in diesem Kapitel auch Messungen unter Einsatz der Laufzeitumgebung FORTE erfolgen. Diese konnten aber im Rahmen dieser Arbeit aus zeitlichen Gründen nicht mehr durchgeführt werden.

6. Zusammenfassung und Ausblick

Nach einer Betrachtung der speziellen Anforderungen an eine SPS, welche im Forschungs- und Bildungsbereich vorherrschen, wurde im Laufe dieser Arbeit erfolgreich eine Kompaktsteuerung auf Basis des Raspberry Pi entworfen. Es wurden ein Schaltplan und ein Platinenlayout für eine nötige Schnittstellenplatine entwickelt und ein Prototyp des Geräts wurde gebaut. Passend dazu wurde eine maschinennahe Steuerungsbibliothek implementiert, welche sowohl eine ereignisbasierte als auch eine zyklische Steuerung unterstützt. Darüber hinaus wurde auf Basis dieser Steuerungsbibliothek ein Modul zur Integration in die Laufzeitumgebung FORTE realisiert.

Untersuchungen des zeitlichen Verhaltens des Prototypen haben gezeigt, dass ein, für den geplanten Einsatzzweck, mehr als hinreichend gutes zeitliches Verhalten erzielt werden kann. Die entwickelte Kompaktsteuerung eignet sich zur Steuerung einer Vielzahl von FESTO Lernsystemen, aber auch zur Steuerung anderer Automatisierungsanlagen im Bildungs- und Forschungsbereich. Die gewählte Bauform, die Anschlüsse und das genutzte Betriebssystem versprechen eine leichte Integration in bestehende Systeme und eine kurze Einarbeitungszeit. Durch die Integration diverser Schutzfunktionen stellen auch die meisten Bedienungsfehler kein Problem dar.

Die nächste anstehende Aufgabe besteht darin, das Modul, welches zur Integration in FORTE entwickelt wurde, zu erweitern um eine Weiterleitung von Interrupts zu ermöglichen. Damit könnte auch beim Einsatz von FORTE eine vollständig ereignisbasierte Steuerung erfolgen. Diese Funktion wurde, wie beschrieben, aus zeitlichen Gründen bisher nicht implementiert. Danach könnte auch, wie geplant, das zeitliche Verhalten des Geräts beim Einsatz von FORTE untersucht und mit dem bisher ermittelten Verhalten verglichen werden.

Unter anderem aus zeitlichen Gründen kam auch das Gehäuse des entwickelten Geräts in dieser Arbeit etwas kurz. Dieses wurde zwar entworfen, aber nicht gebaut. Dies war für durchgeführte Funktionstests und für die Untersuchung des zeitlichen Verhaltens nicht erforderlich, dürfte aber für den praktischen Einsatz des Geräts erforderlich werden.

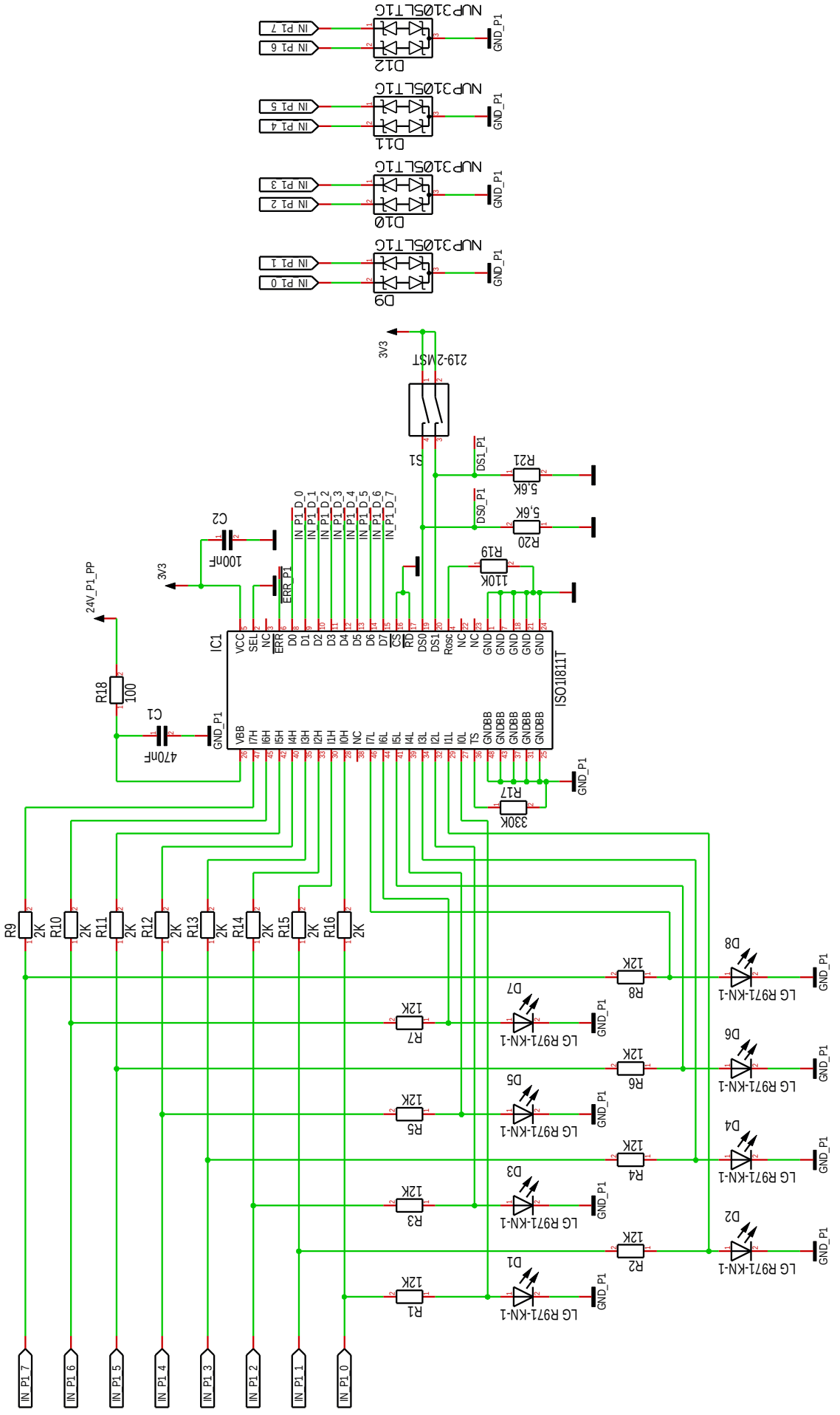
Das entwickelte Gerät beschränkt sich auf einfache digitale Ein- und Ausgänge zur Steuerung von Automatisierungsanlagen. Dies reicht für die meisten der betrachteten Systeme aus. In Zukunft könnten aber noch weitere Schnittstellen ergänzt werden um noch mehr verschiedene Automatisierungsanlagen steuern zu können.

Anhänge

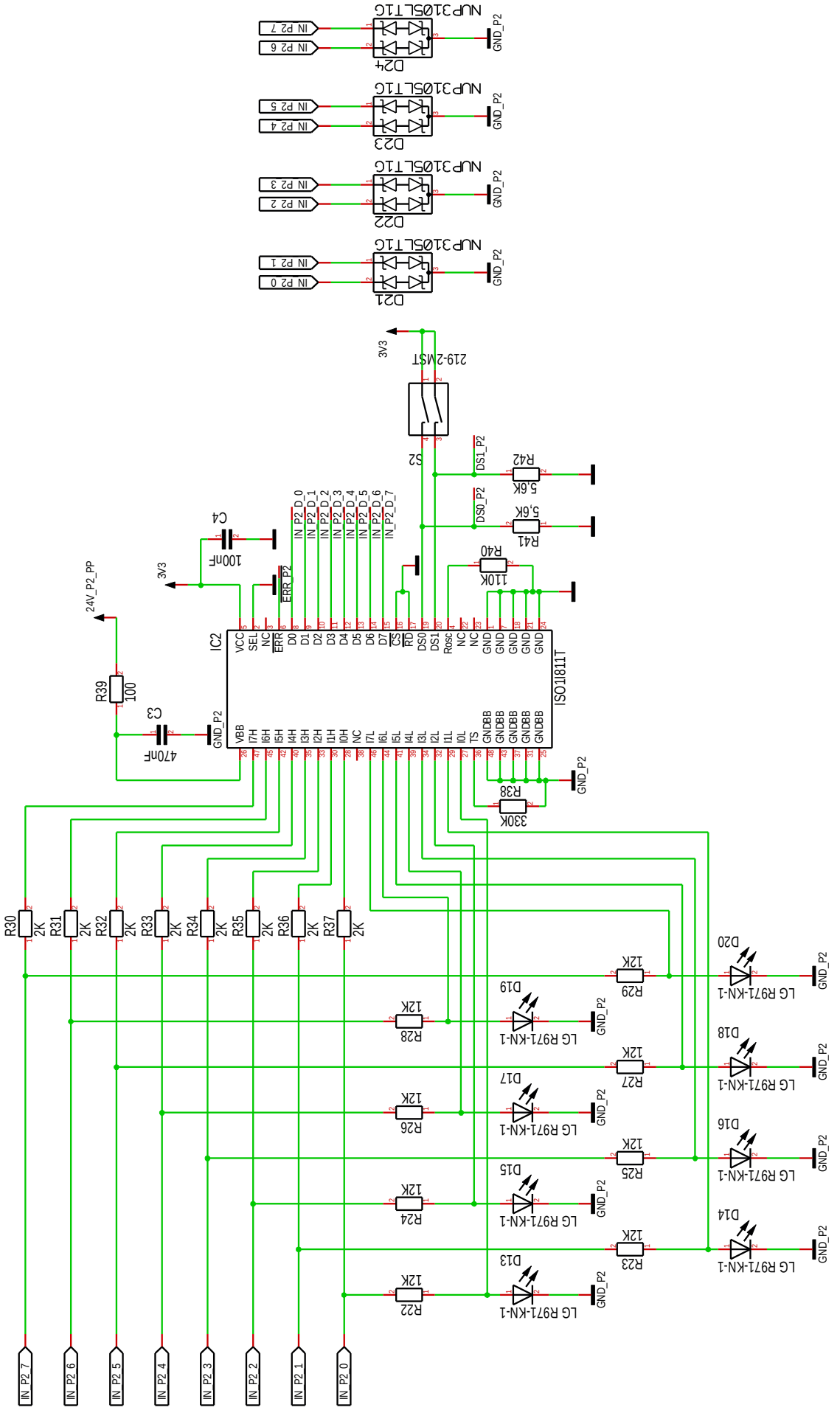
I. Schaltplan

Auf den nachfolgenden Seiten findet sich der komplette Schaltplan der entwickelten SPS.

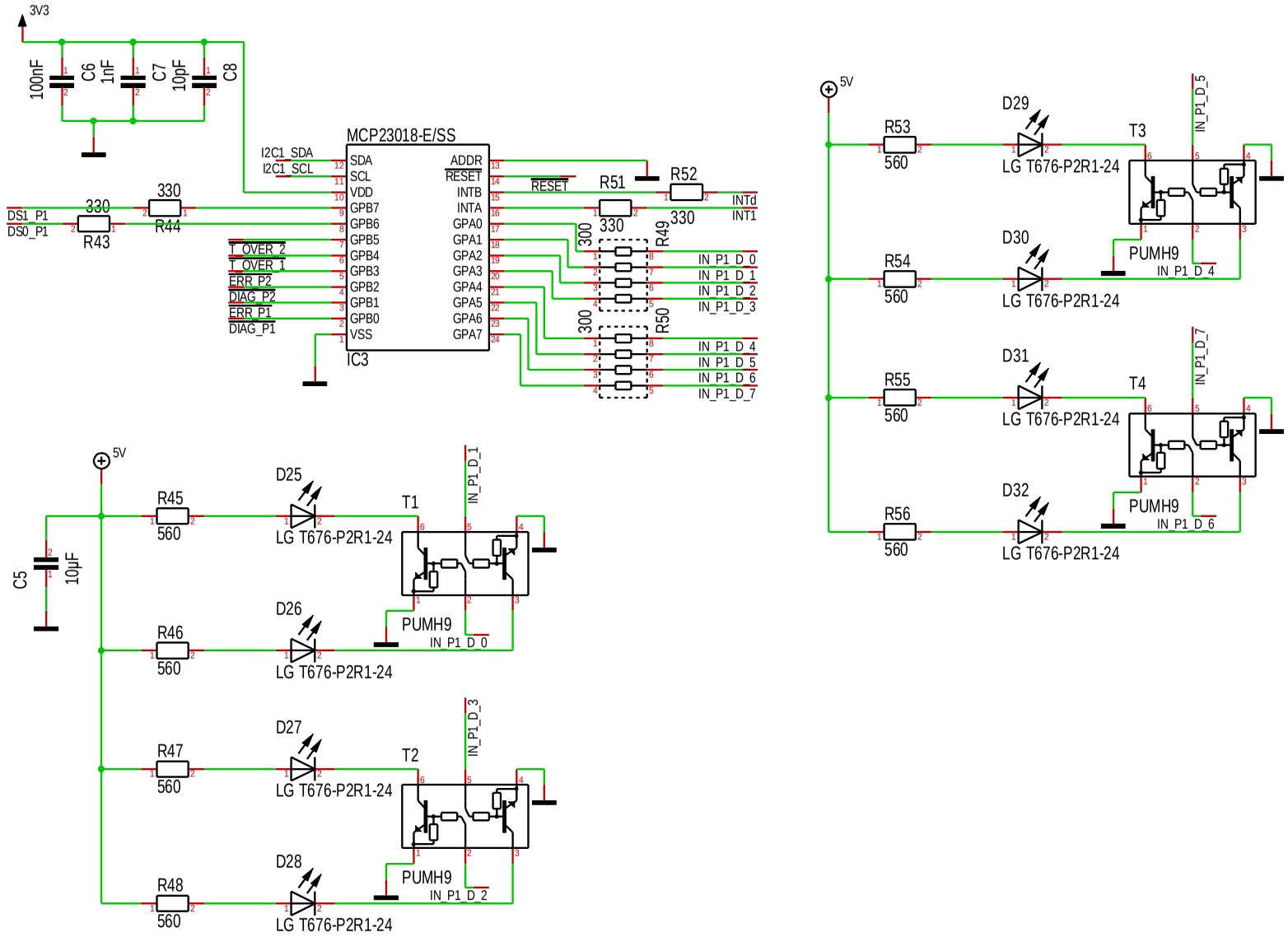
Eingänge von Port 1 (1/2)



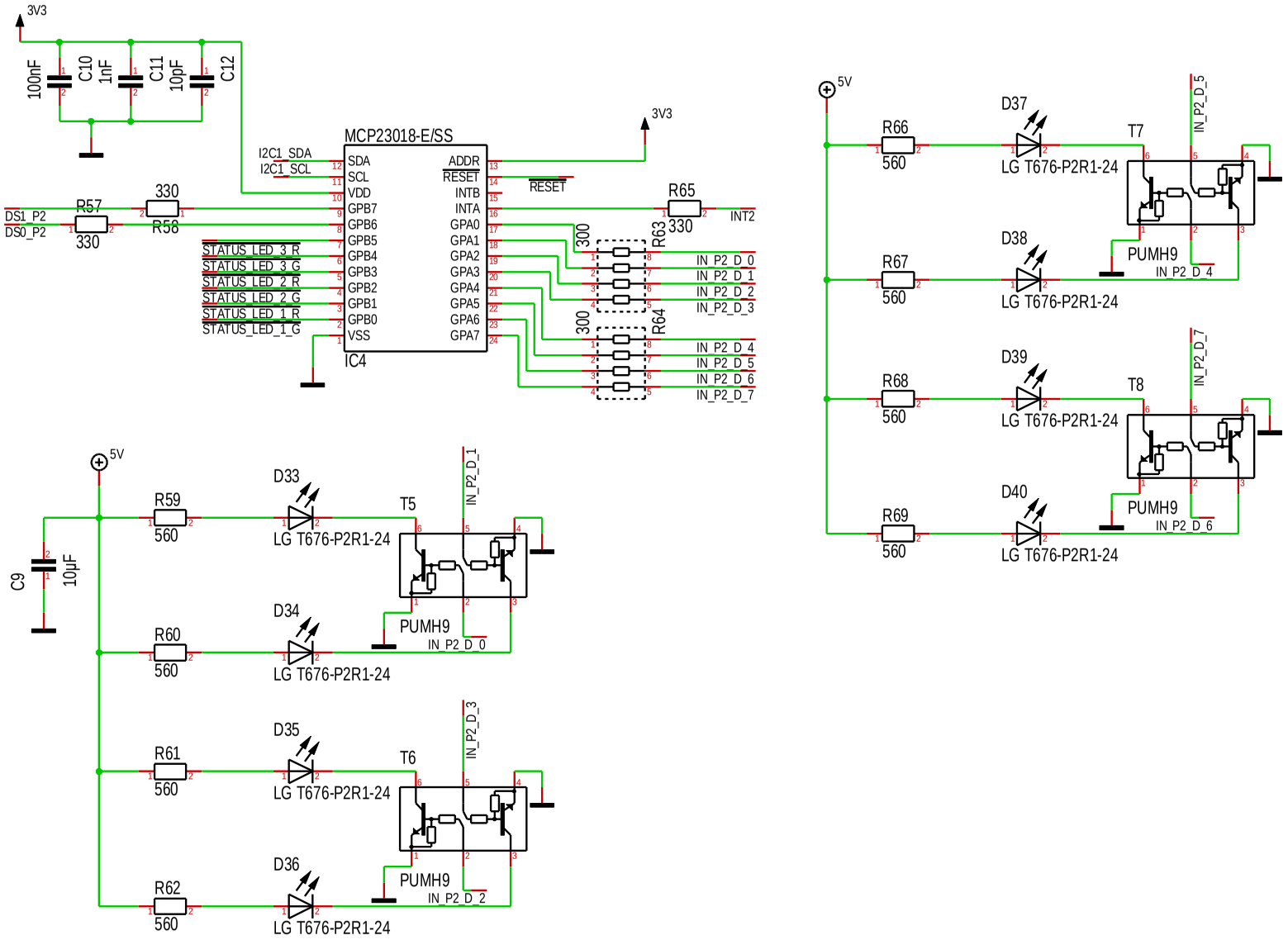
Eingänge von Port 2 (1/2)



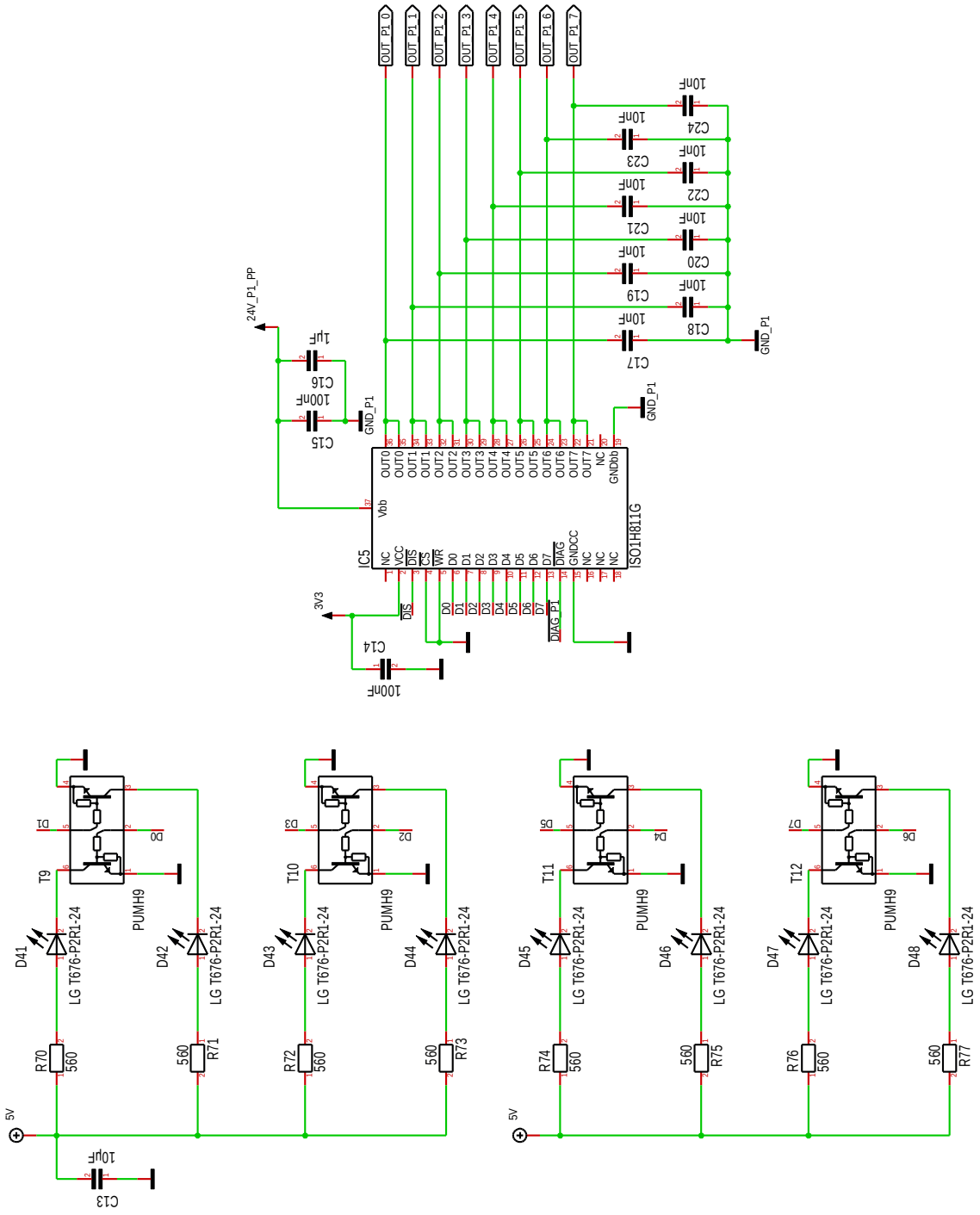
Eingänge von Port 1 (2/2)



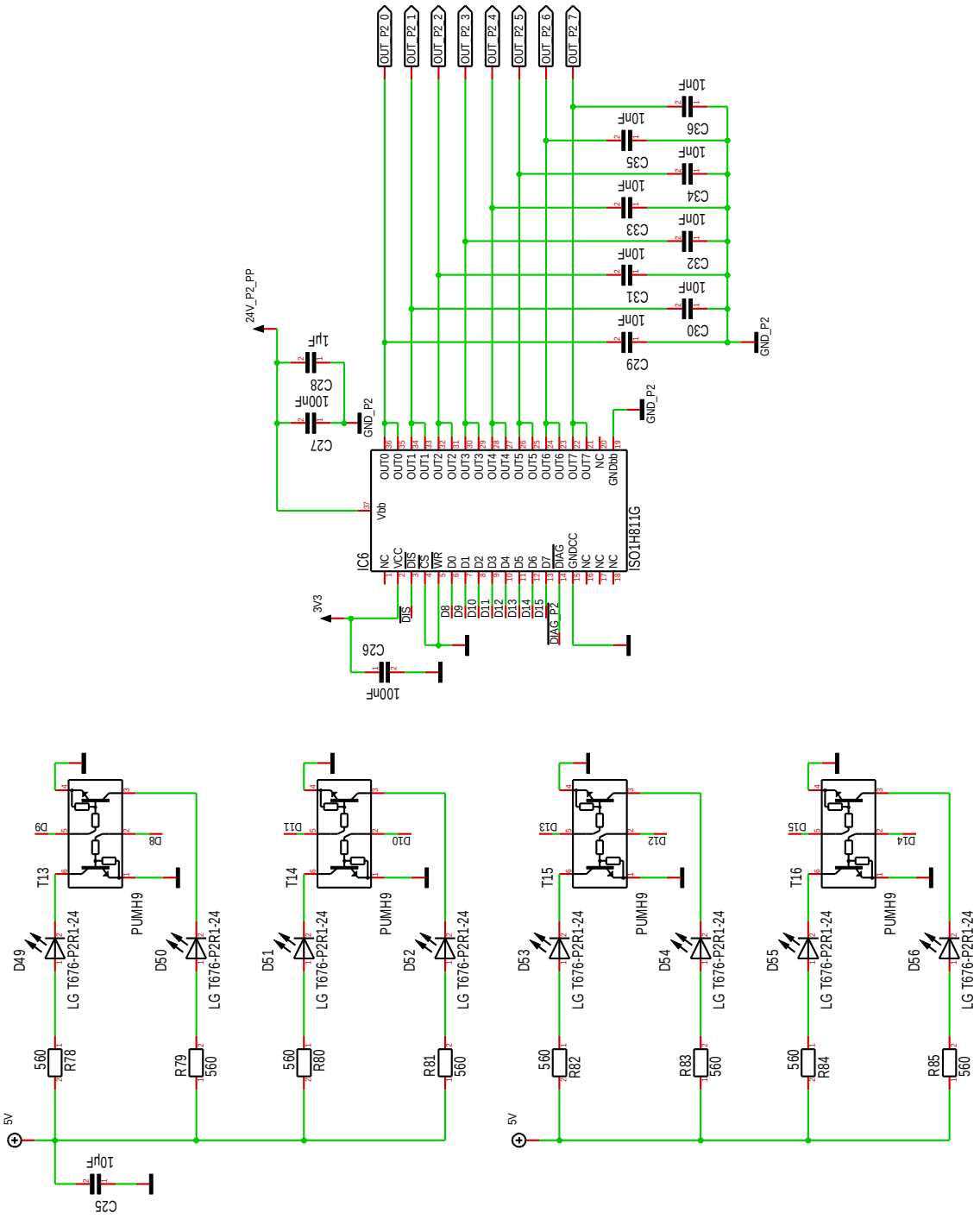
Eingänge von Port 2 (2/2)



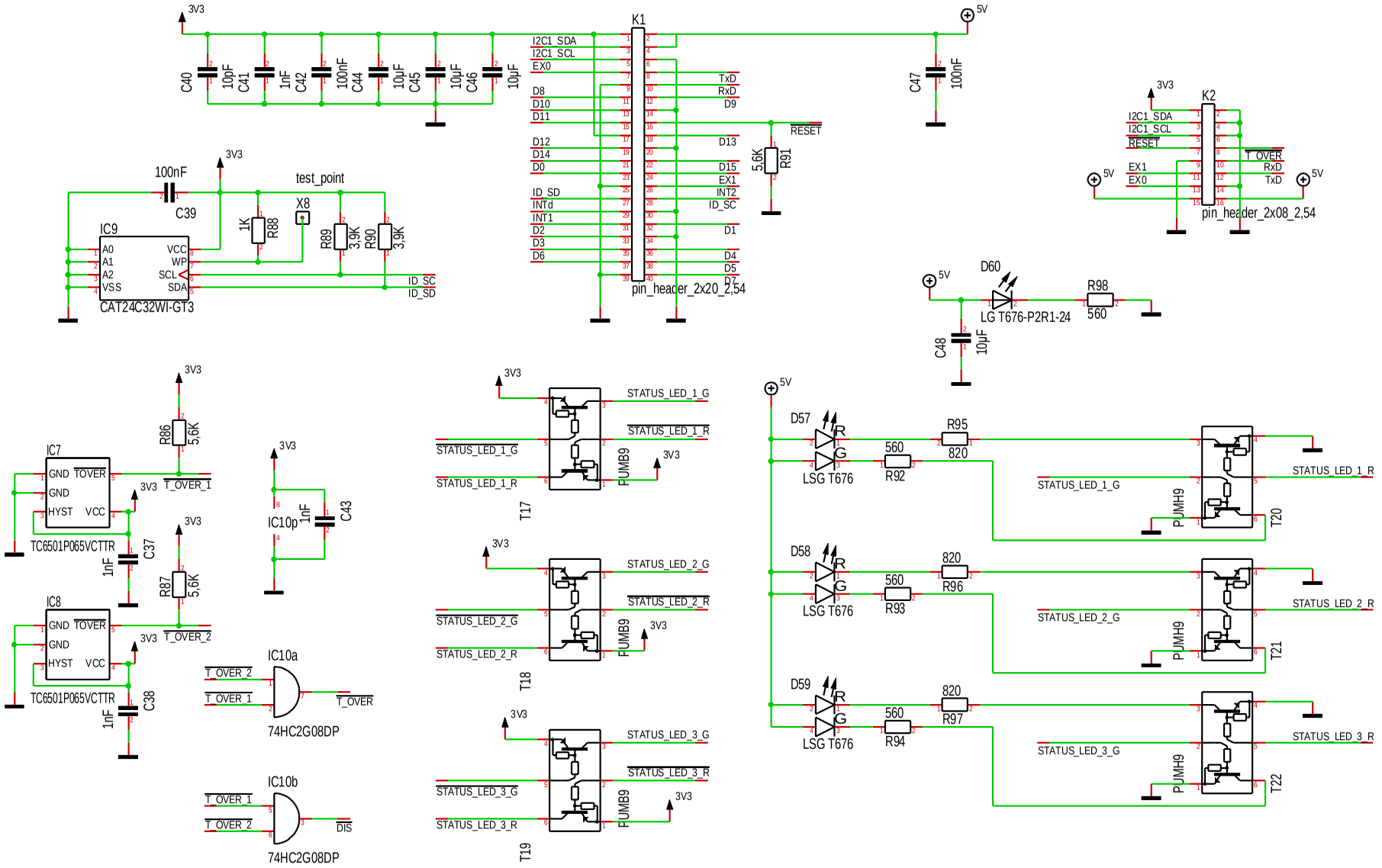
Ausgänge von Port 1



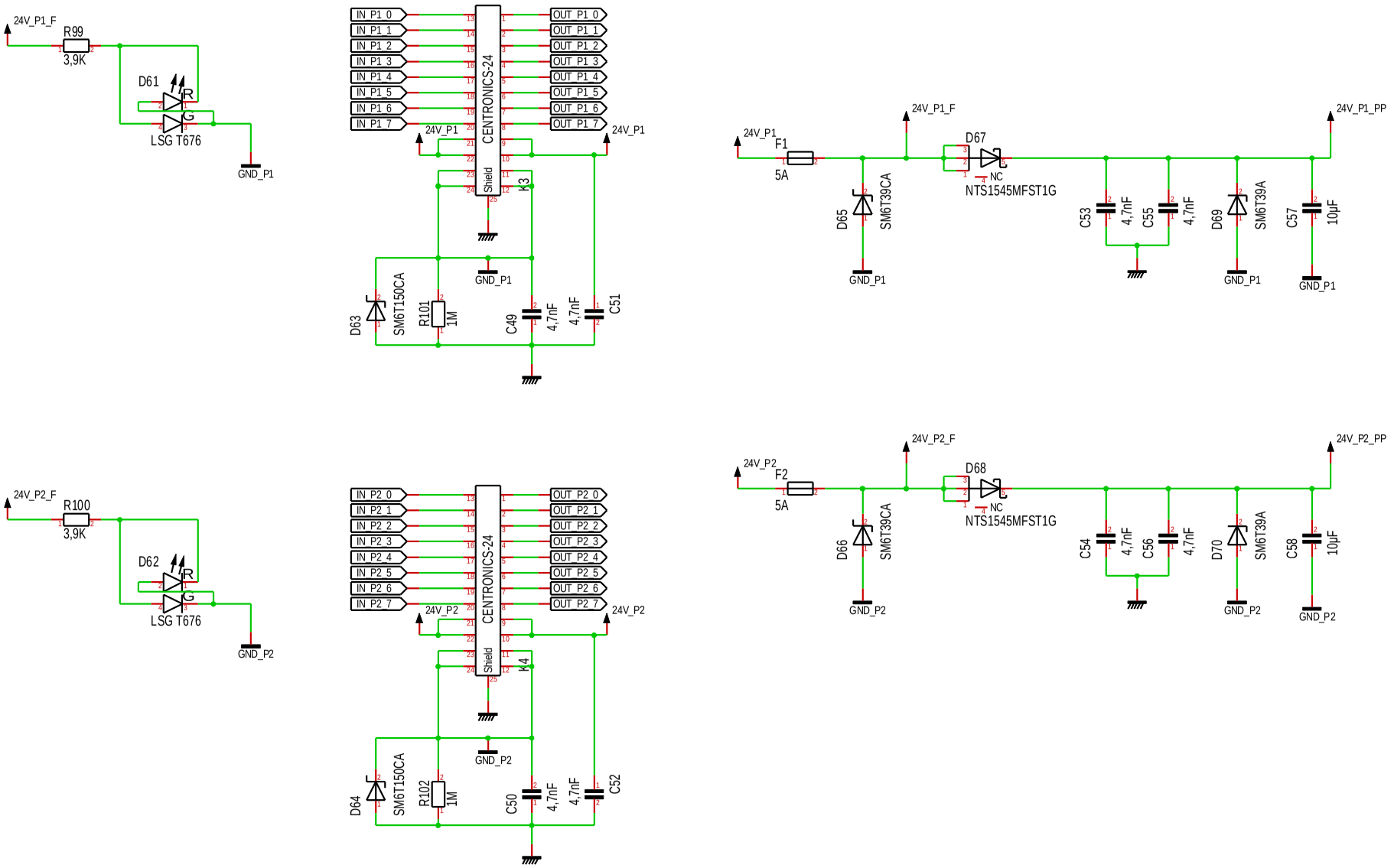
Ausgänge von Port 2



Anbindung an Raspberry Pi, ID-EEPROM, Status-LEDs, Überhitzungsschutz, Erweiterungsschnittstelle



Anschlussstecker für Port 1 und 2, Schutzbeschaltung für 24 V Stromversorgung



II. Bauteilliste

Pos	Anzahl	Namen	Wert
1	2	C1,C3	470nF
2	9	C2,C4,C6,C10,C14,C26,C39,C42,C47	100nF
3	8	C5,C9,C13,C25,C44,C45,C46,C48	10µF
4	6	C7,C11,C37,C38,C41,C43	1nF
5	3	C8,C12,C40	10pF
6	2	C15,C27	100nF
7	2	C16,C28	1µF
8	16	C17-C24,C29-C36	10nF
9	8	C49,C50,C51,C52,C53,C54,C55,C56	4,7nF
10	2	C57,C58	10µF
11	16	D1-D8,D13-D20	LG R971-KN-1
12	8	D9,D10,D11,D12,D21,D22,D23,D24	NUP3105LT1G
13	33	D25-D56,D60	LG T676-P2R1-24
14	5	D57,D58,D59,D61,D62	LSG T676
15	2	D63,D64	SM6T150CA
16	2	D65,D66	SM6T39CA
17	2	D67,D68	NTS1545MFST1G
18	2	D69,D70	SM6T39A
19	2	F1,F2	5A
20	4	GEH1,GEH3,GEH4,GEH5	1272.1018
21	1	GEH2	1272.1016
22	2	IC1,IC2	ISO1I811T
23	2	IC3,IC4	MCP23018-E/SS
24	2	IC5,IC6	ISO1H811G
25	2	IC7,IC8	TC6501P065VCTTR
26	1	IC9	CAT24C32WI-GT3
27	1	IC10	74HC2G08DP
28	1	K1	pin_header_2x20_2,54
29	1	K2	pin_header_2x08_2,54
30	2	K3,K4	CENTRONICS-24
31	16	R1-R8,R22-R29	12K
32	16	R9-R16,R30-R37	2K
33	2	R17,R38	330K
34	2	R18,R39	100
35	2	R19,R40	110K
36	7	R20,R21,R41,R42,R86,R87,R91	5,6K
37	7	R43,R44,R51,R52,R57,R58,R65	330
38	36	R45-R48,R53-R56,R59-R62,R66-R85,R92-R94,R98	560
39	4	R49,R50,R63,R64	300
40	1	R88	1K
41	2	R89,R90	3,9K
42	3	R95,R96,R97	820
43	2	R99,R100	3,9K
44	2	R101,R102	1M
45	2	S1,S2	219-2MST
46	19	T1-T16,T20-T22	PUMH9
47	3	T17,T18,T19	PUMB9

Bauteilliste (Fortsetzung):

Pos	Hersteller	Gehäuse	Händler	Artikelnummer
1	KEMET	0805	Mouser	80-C0805C474K5R
2	KEMET	0603	Mouser	80-C0603C104K3R
3	TDK	0805	Mouser	810-C2012X5R1A106K-3
4	KEMET	0603	Mouser	80-C0603C102K5R
5	KEMET	0402	Mouser	80-C0402C100J3G
6	KEMET	0805	Mouser	80-C0805C104K5R
7	KEMET	0805	Mouser	80-C0805C105K5R
8	AVX Corporation	0603	Mouser	581-ESD31C103K4T2A18
9	TDK	1206	Mouser	810-CGA5H4X7R2J472K
10	muRata	1206	Mouser	81-GRM31CR61H106KA2L
11	OSRAM	0805-D	Mouser	720-LGR971-KN-1
12	ON Semiconductor	SOT23/3	Mouser	863-NUP3105LT1G
13	OSRAM	PLCC2	Mouser	720-LGT676P2R124Z
14	OSRAM	PLCC4	Mouser	720-LSGT676P7R10N7P9
15	STMicroelectronics	SMB/DO-214AA	Mouser	511-SM6T150CA
16	STMicroelectronics	SMB/DO-214AA	Mouser	511-SM6T39CA
17	ON Semiconductor	SO-8 FL	Mouser	863-NTS1545MFST1G
18	STMicroelectronics	SMB/DO-214AA	Mouser	511-SM6T39A
19	Littlefuse	OMNI-BLOK	Mouser	576-0154005.DRT
20	Mentor GmbH	1272.1018	TME	MENTOR 1272.1018
21	Mentor GmbH	1272.1016	TME	MENTOR 1272.1018
22	Infineon	TSSOP48(6,1MM)	Digikey	ISO1I811TCT-ND
23	Microchip	SSOP24_SOT340-1	Mouser	579-MCP23018-E/SS
24	Infineon	PG-DSO-36-EP	Mouser	726-ISO1H811G
25	Microchip	SOT23/5	Mouser	579-TC6501P065VCTTR
26	ON Semiconductor	SOIC 8	Mouser	698-CAT24C32WI-GT3
27	NXP	TSSOP8_SOT505-2	Mouser	771-HC2G08DP125
28	MPE Garry	2x20_G_2,54	Reichelt	MPE 087-2-040
29	MPE Garry	2x08_G_2,54	Reichelt	MPE 087-2-016
30	ASSMANN	CENTRONIC-24	Reichelt	SE 5724FR
31	Panasonic	0805	Mouser	667-ERJ-6ENF1202V
32	KOA Speer	1206	Mouser	660-RK73H2BTTD2001F
33	Panasonic	0603	Mouser	667-ERJ-3EKF3303V
34	KOA Speer	0805	Mouser	660-RK73B2ATTD101J
35	Panasonic	0603	Mouser	667-ERJ-3EKF1103V
36	KOA Speer	0603	Mouser	660-RK73B1JTTD562J
37	Panasonic	0603	Mouser	667-ERJ-3GEYJ331V
38	Panasonic	0603	Mouser	667-ERJ-3GEYJ561V
39	Bourns	Bourns_CAY16-F4,-J4	Mouser	652-CAY16-301J4LF
40	KOA Speer	0603	Mouser	660-RK73B1JTTD102J
41	KOA Speer	0603	Mouser	660-RK73H1JTTD3901F
42	Panasonic	0603	Mouser	667-ERJ-3GEYJ821V
43	Panasonic	2010	Mouser	667-ERJ-12ZYJ392U
44	Panasonic	1206	Mouser	667-ERJ-P08J105V
45	CTS	SMD-7400	Mouser	774-2192MST
46	NXP	SOT363	Mouser	771-PUMH9-T/R
47	NXP	SOT363	Mouser	771-PUMB9115

Inhalt der beiliegenden CD

analyzer_1.0.0.zip	Quellcode des entwickelten Auswertungsprogramms
Barta.pdf	Diese Arbeit als PDF-Datei
FORTE_1.7.1_rpiplc.zip	Quellcode des entwickelten Moduls zur Integration in FORTE 1.7.1
librpiplc_1.0.0.zip	Quellcode der entwickelten Steuerungsbibliothek, einschließlich dem eingesetzten Testprogramm loopback, einiger Demoprogramme und einer Dokumentation

Literaturverzeichnis

[1] N. Prof. Dr. Gronau, „Enzyklopädie der Wirtschaftsinformatik – Online-Lexikon / Industrie 4.0," 23 Juni 2015. [Online]. URL: http://www.enzyklopaedie-der-wirtschaftsinformatik.de. [Zugriff am 23 Oktober 2015].

[2] „Wikipedia (DE)," [Online]. URL: https://de.wikipedia.org/wiki. [Zugriff am 13 März 2016].

[3] M. Seitz, Speicherprogrammierbare Steuerungen für die Fabrik- und Prozessautomation, München: Carl Hanser Verlag, 2015.

[4] W. A. Halang, Funktionale Sicherheit: Echtzeit 2013, Heidelberg: Springer-Verlag, 2013.

[5] K. F. Früh, Handbuch der Prozessautomatisierung: Prozessleittechnik für verfahrenstechnische Anlagen, Oldenbourg Industrieverlag, 2009.

[6] P. Hehenberger, Computerunterstützte Fertigung: Eine kompakte Einführung, Heidelberg: Springer-Verlag, 2011.

[7] „SPS Systeme," [Online]. URL: http://www.sps-lehrgang.de/sps-systeme/. [Zugriff am 23 November 2015].

[8] S. Zacher und M. Reuter, Regelungstechnik für Ingenieure: Analyse, Simulation und Entwurf von Regelkreisen, Wiesbaden: Springer Fachmedien, 2014.

[9] Siemens AG, „SIMATIC S7-1200 Data sheet (6ES7231-4HD32-0XB0 SIMATIC S7-1200, ANALOG INPUT, SM 1231, 4 AI, +/-10V, +/-5V, +/-2.5V, OR 0-20MA/4-20 MA, 12 BIT + SIGN BIT (13 BIT ADC))," 2015.

[10] Siemens AG, „SIMATIC S7-1200 Data sheet (6ES7231-5PD32-0XB0 SIMATIC S7-1200, ANALOG INPUT, SM 1231 RTD, 4 X AI RTD MODULE)," 2015.

[11] Siemens AG, „SIMATIC S7-1200 Data sheet (6ES7231-5QA30-0XB0 SIMATIC S7-1200, ANALOG SB 1231 T, 1 X AI THERMOCOUPLE, TYPE J OR K)".

[12] Siemens AG, Umrichter SINAMICS V20 Betriebsanleitung, 2015.

[13] S. Y. Nof, Springer Handbook of Automation, Berlin: Springer, 2009.

[14] Siemens AG, SINAMICS G110 Betriebsanleitung Ausgabe 04/2005, 2005.

[15] Siemens AG, SINAMICS G120C Listenhandbuch, 2015.

[16] Internationale Elektrotechnische Kommission, „IEC 61131-2".

[17] Infineon Technologies AG, „ISOFACE ISO1I811T Data Sheet," Infineon Technologies AG, München, 2012.

[18] Infineon Technologies AG, „AN-EVAL 2x8-ISO1I813T," Infineon Technologies AG, München, 2011.

[19] Infineon Technologies AG, „Smart Power High-Side-Switch for Industrial Applications (ISP752R)," Infineon Technologies AG, München, 2008.

[20] G. Schnell, Bussysteme in der Automatisierungs- und Prozesstechnik - Grundlagen, Systeme und Trends der industriellen Kommunikation, Wiesbaden: Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, 2003.

[21] CAN in Automation, „CAN in Automation (CiA)," 2015. [Online]. URL: http://www.can-cia.org/. [Zugriff am 19 November 2015].

[22] R. Zurawski, Industrial Communication Technology Handbook, Second Edition, CRC Press / Taylor & Francis Group, 2014.

[23] Beckhoff Automation GmbH & Co. KG, „Beckhoff Information System," [Online]. URL: https://infosys.beckhoff.com. [Zugriff am 13 März 2016].

[24] K. H. John und M. Tiegelkamp, SPS-Programmierung mit IEC 61131-3: Konzepte und

Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen, Heidelberg: Springer-Verlag, 2009.

[25] A. Zoitl und R. Lewis, Modelling Control Systems Using IEC 61499 - 2nd Edition, London: The Institution of Engineering and Technology, 2014.

[26] „Wikipedia (EN),“ [Online]. URL: https://en.wikipedia.org/wiki. [Zugriff am 13 März 2016].

[27] „Real-Time Linux Wiki,“ [Online]. URL: https://rt.wiki.kernel.org. [Zugriff am 13 März 2016].

[28] L. Clark, „Intro to Real-Time Linux for Embedded Developers,“ 21 März 2013. [Online]. URL: https://www.linux.com/news/featured-blogs/200-libby-clark/710319-intro-to-real-time-linux-for-embedded-developers. [Zugriff am 07 Dezember 2015].

[29] TenAsys Corporation, „INtime for Windows,“ 2015. [Online]. URL: http://www.tenasys.com/tenasys-products/intime-rtos-family/intime-for-windows. [Zugriff am 07 Dezember 2015].

[30] 3S-Smart Software Solutions GmbH, „CODESYS,“ [Online]. URL: https://www.codesys.com. [Zugriff am 13 März 2016].

[31] „4DIAC,“ [Online]. URL: http://www.eclipse.org/4diac. [Zugriff am 13 März 2016].

[32] A. Zoitl, Real-Time Execution for IEC 61499, Instrumentation, Systems, and Automation Society, 2009.

[33] Festo Didactic GmbH & Co. KG, „Festo Didactic Katalog 2015,“ 2015.

[34] Festo Didactic SE, „Schaltpläne, elektropneumatisch,“ 2007. [Online]. URL: http://www.festo-didactic.com/de-de/service/mps/schaltplaene/stationen.htm?fbid=ZGUuZGUuNTQ0LjEzLjMyLjk1OC41NzUy. [Zugriff am 12 Dezember 2015].

[35] Festo Didactic GmbH & Co. KG, „Easy Port USB Manual,“ Denkendorf, 2008.

[36] Festo AG & Co., „Bedienungsanleitung Schlitteneinheit Typ SLT-..., SLF-...,“ Esslingen, 1999.

[37] Festo Didactic SE, „Festo Didactic,“ 2015. [Online]. URL: http://www.festo-didactic.com. [Zugriff am 13 März 2016].

[38] Festo Didactic GmbH & Co KG, „AFB Handbuch Station Transportsystem,“ Denkendorf, 2009.

[39] FESTO, „Linearantriebe DGPL,“ 2015.

[40] FESTO, „Linearantriebe DGC,“ 2015.

[41] Festo Didactic GmbH & Co. KG, „Handbuch Station Handhaben,“ 2006.

[42] Festo AG & Co., „Datenblatt Magnetventil CPV10-M1H-5LS-M7,“ Esslingen, 2002.

[43] Festo AG & Co., „Datenblatt Magnetventil CPV10-M1H-2x3OLS-M7,“ Esslingen, 2002.

[44] Festo AG & Co., „Datenblatt Magnetventil CPV10-M1H-2x3GLS-M7,“ Esslingen, 2002.

[45] Festo Didactic, „Datenblatt Anlaufstrombegrenzer 150768“.

[46] Festo Didactic GmbH & Co. KG, „Datenblatt Gleichstrom-Drehmagnet mit Rückstellfeder 665109,“ 2004.

[47] Siemens AG, „Datenblatt SIMATIC S7-1200 CPU 1214C (6ES7313-6CG04-0AB0),“ 2015.

[48] Siemens AG, „Datenblatt SIMATIC S7-300 CPU 314C-2 DP (6ES7314-6CH04-0AB0),“ 2015.

[49] BeagleBoard.org Foundation, „BeagleBoard.org - community supported open hardware computers for making,“ [Online]. URL: http://beagleboard.org. [Zugriff am 13 März 2016].

[50] Cubietech Limited, „CubieBoard | A series of open source hardware,“ [Online]. URL: http://cubieboard.org. [Zugriff am 13 März 2016].

[51] Raspberry Pi Foundation, „Raspberry Pi B+ (Reduced Schematics),“ 2014.

[52] Broadcom Corporation, „BCM2835 ARM Peripherals,“ 2012.

[53] Infineon Technologies AG, „ISOFACE ISO1I813T Data Sheet," 2015.

[54] Maxim Integrated, „Datenblätter zu MAX31910, MAX31911, MAX31912, MAX31913, MAX31914 und MAX31915," 2015.

[55] STMicroelectronics, „"Data Briefs" / Datasheets zu CLT01-38S4, CLT01-38SQ7, CLT3-4B, SCLT3-8BQ7 und SCLT3-8BT8".

[56] Texas Instruments, „Datenblätter zu SN65HVS881, SN65HVS885, SN65HVS882 und SN65HVS880".

[57] V. Vashchenko und M. Scholz, System Level ESD Protection, Springer International Publishing, 2014.

[58] Infineon Technologies AG, Application Note AN-EVAL 2x8-ISO1I811T, 2011.

[59] „RPi Low-level peripherals," [Online]. URL: http://elinux.org/RPi_Low-level_peripherals. [Zugriff am 08 Januar 2016].

[60] Infineon Technologies AG, „BTS4880R Data sheet Rev. 1.2," München, 2014.

[61] Infineon Technologies AG, „ISO1H801G Datasheet Version 2.3," Neubiberg, 2009.

[62] Infineon Technologies AG, „ISO1H811G Datasheet Revision 2.5," München, 2014.

[63] Infineon Technologies AG, „ISO1H812G Datasheet Revision 2.6," München, 2014.

[64] Infineon Technologies AG, „ISO1H815G Datasheet Revision 2.4," München, 2014.

[65] Infineon Technologies AG, „ISO1H816G Datasheet Revision 2.4," München, 2014.

[66] Infineon Technologies AG, „ITS42008-SB-D Data Sheet Rev 1.01," München, 2014.

[67] Infineon Technologies AG, „ITS4880R Data Sheet Rev. 1.2," München, 2014.

[68] STMicroelectronics, „ISO8200B Datasheet," 2014.

[69] STMicroelectronics, „VN808-32-E Datasheet," 2013.

[70] STMicroelectronics, „VN808-E Datasheet," 2013.

[71] STMicroelectronics, „VN808CM-32-E Datasheet," 2013.

[72] STMicroelectronics, „VN808CM-E Datasheet," 2013.

[73] STMicroelectronics, „VNI8200XP Datasheet," 2015.

[74] Infineon Technologies AG, „AN-EVAL 2x8-ISO1H811G-1 Application Note," Neubiberg, 2010.

[75] Infineon Technologies AG, „ISO2H823V2.5 Datasheet Revision 2.0," 2015.

[76] Texas Instruments Incorporated, „Datenblatt zu PCA9538," 2014.

[77] NXP Semiconductors, „PCAL9538A Product data sheet Rev. 3.1," 2015.

[78] Maxim Integrated, „Datenblatt zu MAX7321 (Rev. 4)," 2014.

[79] Microchip Technology Inc., „Serial Peripherals Products," 2015. [Online]. URL: http://www.microchip.com/serialperipherals (http://www.microchip.com/ParamChartSearch/chart.aspx?branchID=11034&mid=11&lang=en&pageId=79). [Zugriff am 13 März 2016].

[80] Microchip Technology Inc., „Datenblatt zu MCP23017/MCP23S17," 2007.

[81] Microchip Technology Inc., „Datenblatt zu MCP23018/MCP23S18," 2008.

[82] ON Semiconductor, „NUP3105L Datasheet," 2014.

[83] J. Adams, 22 Juli 2014. [Online]. URL: https://github.com/raspberrypi/hats/blob/master/eeprom-circuit.png. [Zugriff am 16 Januar 2016].

[84] NXP, „PEMB9; PUMB9 Product data sheet," 2011.

[85] Microchip Technology Inc., „Datanblatt zu TC6501/2/3/4," 2004.

[86] ON Semiconductor, „Datenblatt zu NTS1545MFS, NRVTS1545MFS,“ 2014.

[87] Infineon Technologies AG, „Recommendations for Printed Circuit Board Assembly of Infineon DSO and SSOP Packages,“ München, 2009.

[88] Beta LAYOUT GmbH, „Leiterplatten Prototypen,“ [Online]. URL: https://www.pcb-pool.com/. [Zugriff am 13 März 2016].

[89] J. Adams, „Raspberry Pi Model B+ (mechanische Spezifikation),“ 2014.

[90] ON Semiconductor, „Datenblatt zu CAT24C32,“ 2014.

[91] J. Adams, A. Scheller, P. Elwell, R. Rayns, R. Hartmann und u. andere, „GitHub/raspberrypi/hats,“ [Online]. URL: https://github.com/raspberrypi/hats. [Zugriff am 24 Januar 2016].

[92] T. Iwai, M. Brown, H. H. Sweeten, G. Kroah-Hartman und u. andere, „GitHub/raspberrypi/linux Branch rpi-4.1.y,“ [Online]. URL: https://github.com/raspberrypi/linux. [Zugriff am 24 Januar 2016].

[93] „GPIO Sysfs Interface for Userspace,“ [Online]. URL: https://www.kernel.org/doc/Documentation/gpio/sysfs.txt. [Zugriff am 13 März 2016].

[94] M. Kerrisk, P. Zijlstra und J. Lelli, „sched(7) - Linux manual page,“ 2014.

[95] Pico Technology Limited, „PicoScope 3000 Series (A API) - PC Oscilloscopes and MSOs - Programmer's Guide,“ 2014.

[96] B. Nuttall, P. Elwell und andere, „raspberrypi / documentation,“ [Online]. URL: https://github.com/raspberrypi/documentation/tree/master/linux/kernel. [Zugriff am 13 März 2016].

[97] „Raspberry Pi - View topic - CONFIG_PREEMPT_RT on Raspberry Pi,“ [Online]. URL: https://www.raspberrypi.org/forums/viewtopic.php?f=29&t=39951&start=75. [Zugriff am 05 März 2016].

Hinweis: Zur Entwicklung der Schaltung kamen die offiziellen Datenblätter aller genutzten Bauteile zum Einsatz. Sofern die Informationen der Datenblätter nicht unmittelbar im Text genutzt werden, werden auch die Datenblätter nicht explizit im Literaturverzeichnis aufgeführt.