Entwicklung eines Sensorkonzeptes für eine R O B O T E R H A N D

Diplomarbeit

eingereicht bei
apl. Prof. Dr. Günter Kemnitz

Institut für Informatik
Arbeitsbereich Hardwareentwurf und Robotik
Technische Universität Clausthal

von
Matthias Arndt
Leibnizstrasse 20 App 41
38678 Clausthal-Zellerfeld

Studienrichtung: Wirtschaftsinformatik
15. Fachsemester
Matrikelnummer: 302726

Datum: 23. Oktober 2007

Inhaltsverzeichnis

Kapitel 1

Einleitung

Ein robotisches System ist nur dann sinnvoll einzusetzen, wenn es autonom agieren kann. Dazu muß es seine Umwelt erkennen und erfassen können. Der Greifarm des Roboters in der Abteilung Hardwareentwurf und Robotik kann zwar interagieren, aber er hat keinerlei Möglichkeit, seine Umwelt, etwa Werkzeuge oder Hindernisse im Bereich der Roboterhand, zu erkennen.
Gegenstand der vorliegenden Arbeit ist der Entwurf und die Implementierung eines Sensorkonzeptes für die konkrete Roboterhand. Das vorgestellte System verwendet einen Mikrocontroller, um die einzelnen Sensoren abzufragen und ihre Werte über den CAN-Bus in das restliche Robotersystem einzugliedern. Es wurde Wert auf eine einfache und robuste Implementierung gelegt. Dabei wurden ausschließlich Komponenten vorgesehen, die ohne größere Schwierigkeiten im Elektronikfachhandel zu beschaffen sind.
In Kapitel 2 der Arbeit wollen wir zunächst das betroffene Robotersystem vorstellen und auf seine Kommunikation mit der Aussenwelt eingehen. Kapitel 3 bringt einen Überblick über die eingesetzten Sensorkonzepte, während Kapitel 4 die entworfene Hardware und das Mikrocontrollersystem aus der Sicht der Implementierung betrachtet. Kapitel 5 schließlich beschreibt die Software zur Messwerterfassung und die Kommunikationsprotokolle, mit denen die Messwerte abgefragt werden können. Zuletzt gibt Kapitel 6 einen Ausblick über die Entwicklungsmöglichkeiten und Anwendung des entwickelten Sensorsystems.

Kapitel 2

Der Industrieroboter

2.1 Der Industrieroboter und seine Subsysteme

Der in der Abteilung Hardwareentwurf und Robotik[1] eingesetzte Industrieroboter ist für den Einsatz unter Laborbedingungen vorgesehen. Er besteht aus drei großen Subsystemen:

- einem Fahrwerk für den mobilen Einsatz
- einem Arm zu Manipulation der Umgebung
- ein CAN-Bus, um die Kommunikation der einzelnen Teile zu ermöglichen

Das Fahrwerk der Firma MIAG Fahrzeugbau besteht aus einem Rahmen aus Aluminiumprofilen. Es enthält die Steuerelektronik, eine autonome Energieversorgung über LKW-Akkumulatoren, sowie Platz für einen Industrie-PC. Zur Fortbewegung sind vier Räder mit diagonalen Rollen angebracht. Jedes Rad wird dabei von einem eigenen Antriebsmotor betrieben.
Der Arm des Roboters besteht aus Modulen der Firma Amtec. Jedes Modul kann autonom arbeiten und enthält Motoren, Getriebe und die erforderliche Elektronik. Zur Kommunikation untereinander und mit anderen Subsystemen des Roboters ist jedes der Armmodule mit dem CAN-Bus verbunden. Das letzte Armmodul enthält ein Paar Finger, die geöffnet und geschlossen werden können. Im Folgenden wollen wir dieses Modul mit dem Begriff Roboterhand oder Hand bezeichnen.

[1] http://techwww.in.tu-clausthal.de/

Ein PC oder jedes andere Endgerät mit Anschluß an den CAN-Bus kann angeschlossen und in das Robotersystem eingebunden werden. So können dann Roboterfunktionen gesteuert oder überwacht werden.
In [1] und den dort referenzierten Quellen findet man eine grundlegende Erläuterung aller Subsysteme sowie Bedienungsanleitungen für die Inbetriebnahme des Robotersystems.
Das hier vorgestellte Sensorsystem für die Roboterhand bildet dann ein weiteres Subsystem des gesamten Industrieroboters.

2.2 Kommunikation zwischen Subsystemen

2.2.1 Überblick über den CAN-Bus

Die Subsysteme des Industrieroboters kommunizieren über den sogenannten CAN-Bus, der eine Ausprägungsform des sogenannten Feldbus[2] darstellt. Ein Feldbus ist ein serielles, digitales Übertragungsmedium für Informationen im Feldbereich und besteht in der Regel aus einem paar Leitungen, an die die Busteilnehmer angeschlossen werden. Im OSI Referenzmodell[3] implementieren Feldbusse nur drei Schichten, die Bitübertragungsschicht, die Datensicherungschicht und die Anwendungsschicht. Feldbusse haben im allgemeinen folgende Merkmale:

- Echtzeitfähigkeit
- große Netzausdehnungen sind möglich (bis zu 2km, je nach verwendetem Bussystem)
- Übertragungsraten im Bereich von 100kBit/s bis zu 1MBit/s
- serielle Übertragung
- mehr als zwei Busteilnehmer, Addressierbarkeit von Informationen
- hohe effektive Datenrate bei kleinen Paketlängen
- niedrige Kosten
- geringe Störempfindlichkeit

[2] Eine prägnante Einführung in Feldbusse und den Begriff gibt [5], S.26ff.
[3] Eine ausführliche Erläuterung des OSI Referenzmodells und grundlegender Begriffe der Computervernetzung ist in [16] zu finden.

- hohe Interoperabilität durch die Möglichkeit, Protokollumsetzer zu verwenden

Die Abkürzung CAN steht für Controller Area Network und vernetzt Komponenten eines technischen Systems untereinander. Der CAN-Standard wurde von der Firma Bosch für den Einsatz in Kraftfahrzeugen entwickelt, um Steuerkomponenten im Fahrzeug für Einspritzung, Getriebesteuerung und ABS miteinander kommunizieren lassen zu können und Daten auszutauschen. Ohne den Einsatz eines gemeinsamen Buses wären unnötig viele Datenleitungen zu verlegen, wodurch der Wartungsaufwand und die Fehlersuche unnötig erschwert würden.[4] Da die Art der Teilnehmer am Bus nicht weiter spezifiziert ist, ist der Einsatz des CAN-Bus nicht auf Kraftfahrzeugkomponenten beschränkt, sondern kann auch die Kommunikation zwischen verschiedensten technischen Systemen regeln.
Der CAN-Bus ist nach ISO 11898 genormt, dabei insbesondere auch die elektrischen Parameter der physikalischen Übertragung. Spezifiziert sind nur die beiden unteren Schichten im OSI Referenzmodell. Die Anwendungsschicht kann dann durch weitere Systeme, unabhängig vom eigentlichen CAN-Standard, definiert werden, z.B. CANopen oder DeviceNet.[5]
Bei CAN werden gleichberechtigte Endgeräte, z.B. Steuereinheiten, Sensoren und Aktoren[6] über eine Zweidrahtleitung miteinander verbunden. Die Trennung von Nachrichten und physikalischem Bus erfolgt durch den sogenannten CAN-Controller. Von der Seite des Mikrocontrollers werden lediglich empfangene und gesendete Nachrichten betrachtet, während der CAN-Controller die gesamte physikalische Abbildung auf den Bus vornimmt. CAN-Controller werden dabei als eigenständige Bausteine oder als Komponente eines Mikrocontrollers realisiert.[7]

---

[4] [5] erläutert die Verallgemeinerung auf Feldbusse bei der Vernetzung. Der CAN-Bus ist immer eine konkrete Ausprägung eines Feldbusses.

[5] vgl. [7], S.211ff.

[6] Der Industrieroboter ist ein gutes Beispiel. Aktoren wie das Fahrwerk und der Arm werden über den CAN-Bus angesteuert, Sensoren, wie das hier vorgestellte System, können abgefragt werden, während ein PC den Roboter überwacht und mit allen Komponenten über den CAN-Bus kommuniziert.

[7] Eine detaillierte Betrachtung von möglichen CAN-Protokollcontrollern findet sich in [7], S.151ff., wobei diese nur eine Auswahl darstellt. Die Implementierung eines CAN-Controllers in einem programmierbarem Logikbaustein, etwa einem FPGA, ist natürlich ebenso möglich.

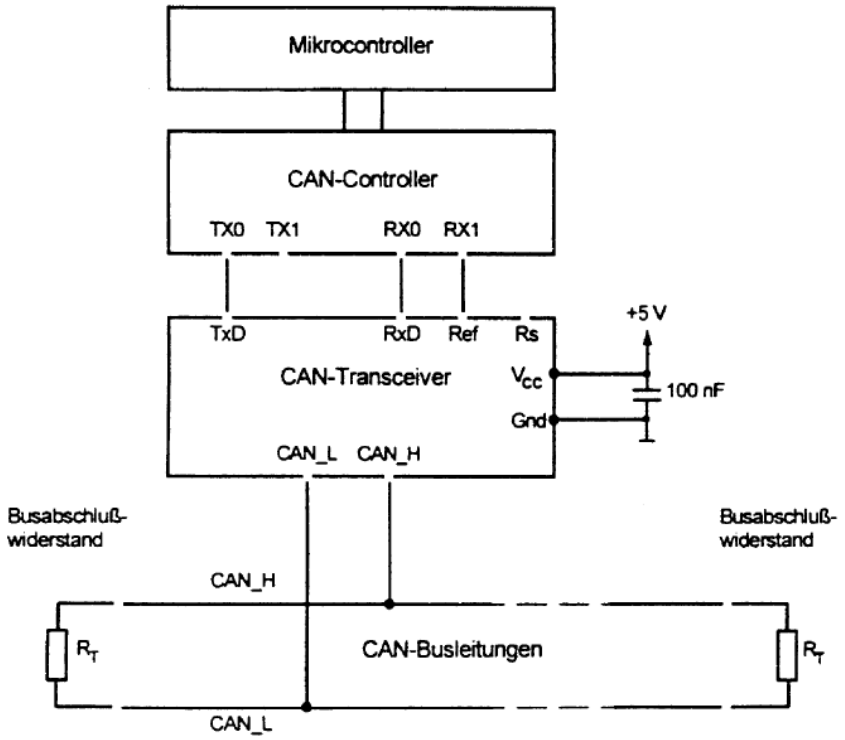


Abbildung 2.1: Ankoppelung von Teilnehmern an den CAN-Bus nach ISO 11898

Abbildung 2.1[8] verdeutlicht die prinzipielle Verschaltung von Teilnehmern am CAN-Bus, der CAN-Controller kann dabei als externer Schaltkreis vorliegen oder im Mikrocontroller integriert sein.
Zusammenfassend besitzt der CAN-Bus folgende Eigenschaften:

- lineare Bustopologie mit einer unbegrenzten Anzahl von möglichen Busteilnehmern
- Multi-Master-Netzwerk mit gleichberechtigter Busteilnahme
- Nachrichtengebundene Adressierung und Prioritätsstufen
- verlustlose Busarbitrierung
- kurze Nachrichten (acht Bytes pro Nachricht)
- hohe Sicherheit der Übertragung durch Fehlererkennung in Hardware
- netzweite Datenkonsistenz

[8] vgl. [6], S. 9

Vorteile des CAN-Bus liegen in der Erreichbarkeit aller Busteilnehmer ohne Hierarchie und der Gleichberechtigung aller Teilnehmer. Diese hilft dabei Komponenten zu entlasten, da auf diese Weise nicht mehr eine zentrale Stelle sämtliche Datenströme aufnehmen und verarbeiten muß, sondern einzelne Subsysteme in der Lage sind, gezielt eine spezielle Aufgabe zu bearbeiten. Im Falle des Roboters kann so ein Subsystem gezielt die Fahrwerksteuerung übernehmen, während ein weiteres eine Notabschaltung im Kollisionsfall regelt.

2.2.2 Funktionsweise des CAN-Bus

Ausgangspunkt der Kommunikation über den CAN-Bus sind immer Nachrichten, die über den Bus an die verschiedenen Teilnehmer verschickt werden sollen. Beim CAN-Bus werden keine Empfänger für eine bestimmte Nachricht benannt. Stattdessen verfügt jede Nachricht über einen eindeutigen Identifier, der sowohl den Inhalt kennzeichnet, als auch die Priorität der jeweiligen Nachricht festlegt. Im Netzwerk entspricht dies einer objektorientierten Adressierung, die an die Nachricht und nicht an einen bestimmten Empfänger gebunden ist. Auf diese Art und Weise können mehrere Busteilnehmer auf die gleiche Nachricht reagieren, falls dies nötig sein sollte.
Ob die Nachricht für den jeweiligen Empfänger relevant ist, wird anhand des Identifiers entschieden. Sind die Daten von Bedeutung, so werden sie übernommen, im anderen Fall ignoriert. Mehrfachempfang wird so ermöglicht, und Meßgrößen können leicht verteilt werden.
Die Buszuteilung wird anhand der Nachrichtenpriorität geregelt, wenn zum gleichen Zeitpunkt mehrere Busteilnehmer Nachrichten über den CAN-Bus verschicken wollen.
Soll eine Nachricht verschickt werden, so ist ihr Inhalt und ihr Identifier festzulegen und dem CAN-Controller die Absendung aufzutragen. Der Controller bildet dann das elektrische Äquivalent der Nachricht und kümmert sich um den Versand über den Bus. Versand und Empfang funktionieren nach dem Broadcastprinzip. In der Zeit, in der ein Controller den Buszugriff hat, sind sämtliche anderen Controller automatisch Empfänger der Nachricht.

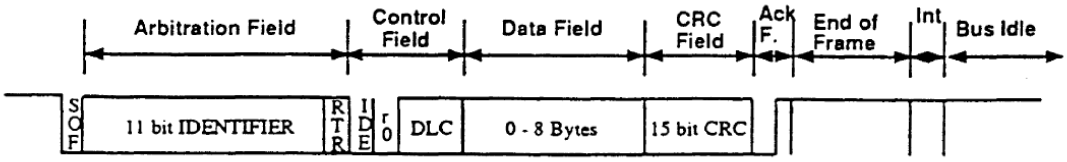


Abbildung 2.2: Nachricht auf dem CAN-Bus im Standardformat

Um Daten in Echtzeit[9] verarbeiten können, müssen die Daten schnell übertragen werden. Eine schnelle Buszuteilung muß also erreicht werden und dabei die Dringlichkeit von Nachrichten berücksichtigt werden.
Der Buszugriff erfolgt zerstörungsfrei. Bei freiem Bus können mehrere Teilnehmer gleichzeitig mit der Übertragung von Nachrichten beginnen. Dabei ist die Signalkodierung des Identifiers ausschlaggebend. CAN unterscheidet nach sogenannten rezessiven und dominanten Pegeln auf dem Bus. Werden ein dominantes und ein rezessives Bit gleichzeitig gesendet, so wird das rezessive Bit überschrieben. Dieser Moment muß vom Protokollcontroller erkannt werden. Er wird mit einer Umschaltung in den Empfang beantwortet, d.h. der Teilnehmer stellt seinen Sendevorgang zurück, bis der Bus wieder frei ist und empfängt in der Zwischenzeit die Nachricht mit dem dominanten Bit. Vereinbarungsgemäß sind Lowpegel auf dem Bus dominant, während Highpegel rezessiv sind. Technisch wird dies durch eine Wired-AND-Schaltung[10] realisiert. Für die Identifier bedeutet dies eine höhere Priorität, wenn die Binärdarstellung des Identifiers viele Nullen enthält, d.h. Identifier mit kleiner Nummer haben eine höhere Priorität auf dem Bus.
Abbildung 2.2[11] verdeutlicht das prinzipielle physikalische Format einer Nachricht im Standardformat[12] auf dem Bus. Die Arbitrierung erfolgt über die Bits im sogenannten Arbitrierungsfeld, welches den Identifier beinhaltet. Das Steuerfeld teilt mit, wieviele Datenbytes im eigentlichen Datenteil der Nachricht folgen. Um

[9] Bei Echtzeitverarbeitung stehen Verarbeitungsergebnisse innerhalb definierter Zeitspannen zur Verfügung. Die Länge der Zeitspanne ist dabei nicht relevant. vgl. [15], S.14
[10] vgl. [5], S. 122ff.
[11] vgl. [6], S.6 unten
[12] Im Standardformat haben Identifier eine Länge von 11 Bit, im erweiterten Format von 29 Bit. vgl. [7] S. 92 ff.

die Integrität der Daten zu gewährleisten wird zur Datensicherung eine 15bittige Prüfsumme nach dem CRC-Verfahren[13] im CRC Feld verschickt.
Der Bus gilt als frei, wenn im sogenannten Intermission field, das auf eine Nachricht folgt, kein dominantes Bit auftritt, d.h. am Bus liegt in diesem Zeitraum ein Highpegel an. Ein Wechsel auf Lowpegel signalisiert an dieser Stelle den Beginn einer neuen Nachricht.
Die Bitlängen der einzelnen Felder einer Nachricht sind wohldefiniert. Eine ausführliche Beschreibung des Nachrichtenformates kann [7], S. 64ff. entnommen werden.

[13] Das für den CAN-Bus verwendete CRC-Verfahren wird inklusive seines Generatorpolynoms in [7], S. 81-82 erläutert.

Kapitel 3

Sensorsysteme

3.1 Übersicht

Der Einsatz verschiedenster Sensorkonzepte ist möglich. Vordergründig soll eine Objekterkennung im Bereich der Roboterhand und der Greifer ermöglicht werden. Damit soll der Roboter feststellen können, ob er etwa ein Werkzeug gegriffen hat, an welcher Stelle und mit welchem Anpressdruck dies geschieht, sowie eine Erkennung von Kollisionen der Hand mit der Umgebung oder einer anderen Maschine.
Folgende Sensoren erscheinen geeignet, um diesem Zweck zu dienen:

(a) Druckempfindliche Kontakte, die eine Berührung und gegebenenfalls eine Kraftrückmeldung liefern

(b) Lichtschranken, die optisch feststellen, ob ein Objekt eine bestimmte Position erreicht

(c) Abstandssensoren, die zusätzlich zu einer Positionsmeldung den Abstand zu einem Objekt angeben.

Weitere Konzepte, die hier keiner praktischen Verifikation unterzogen wurden, umfassen eine Objekterkennung mit Schallwellen oder über Bilderkennung und Verfahren der Bildverarbeitung. Gerade für letztere Prinzipien bestehen große Möglichkeiten, die aber einen höhren Einsatz von Rechenkapazitäten und Parallelisierung erfordern.

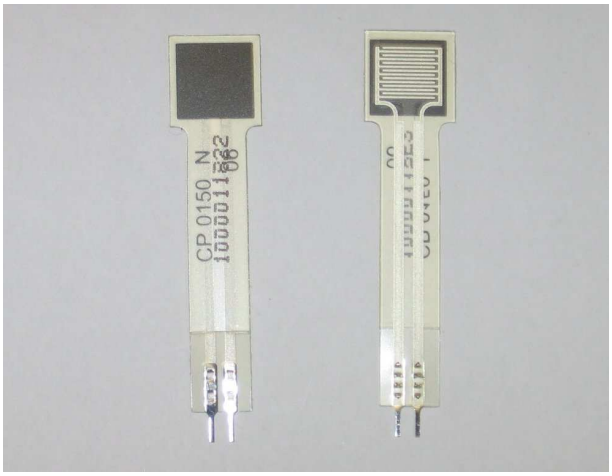


Abbildung 3.1: Ein Force Sensing Resistor vom Typ FSR-154N

Alle Messwerte werden als Spannungen aufgezeichnet. Ein A/D-Wandler im Mikrocontroller nimmt die Spannungen auf und übersetzt sie in auswertbare Zahlenwerte. Eine Verwendung digitaler Signale, z.B. ob die Lichtquelle einer Lichtschranke durch ein Objekt bedeckt ist oder nicht, ist prinzipiell möglich, erfordert aber zusätzliche Hardware für den Vergleich von Werten durch zusätzliche Komparatorschaltungen. Eine Auslagerung des Vergleichs in Software ist einfacher und flexibeler, da so keine Schaltung neuangepasst und umgebaut werden muss, nur um eine Schwelle um 5% in die eine oder andere Richtung zu verschieben.

3.2 Drucksensoren

Zur Erfassung von Berührungen und Kraftvektoren wurden insgesamt zwölf Drucksensoren vom Typ FSR-154N[1] verbaut, davon vier in den Fingern und acht als Bumper zur Kollisionserkennung.
Dabei handelt es sich um einen Force Sensing Resistor, also einen kraftabhängigen Widerstand, und nicht um einen Dehnungsmesstreifen. Ein einzelner Sensor besteht aus drei miteinander verbundenen Schichten. Eine leitende Trägerschicht, eine Klebeschicht, die als Abstandshalter wirkt, sowie eine Folie mit drei Elek-

[1] Näheres zu einem FSR kann [9] entnommen werden.

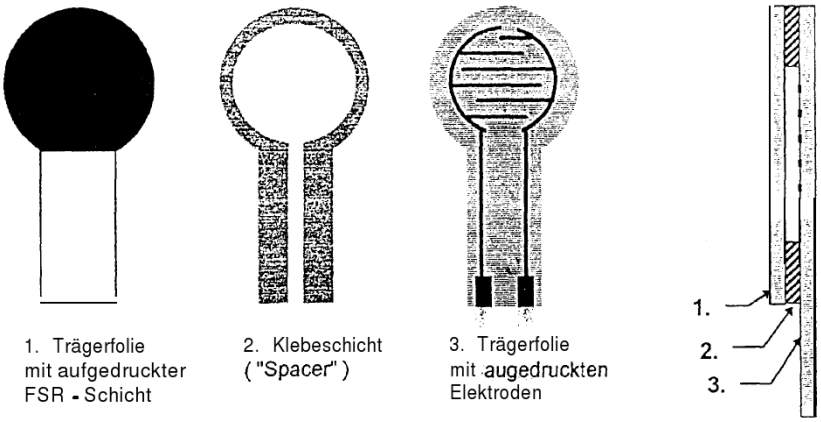


Abbildung 3.2: Schichten im FSR

troden. Die Elektroden greifen ineinander, sind aber nicht miteinander verbunden. Abbildung 3.2 aus [9] zeigt die Schichtungen für einen einzelnen FSR. Der Widerstandwert des FSR ändert sich in Abhängigkeit der Kraft, die auf die aktive Schicht wirkt. Die Trägerschicht verbindet die Elektroden bei Kontakt und es wird eine leitende Verbindung zwischen diesen aufgebaut. Je größer die wirkende Kraft ist, desto geringer wird der Widerstand zwischen den beiden Elektroden, da mehr Brücken geschlossen werden.

Die einzelnen Sensoren erscheinen elektrisch als veränderbare Widerstände. Ihr Verhalten ist nichtlinear, schon eine Kraftwirkung von 1N auf den Sensor läßt den Widerstandwert von über 1 $M\Omega$ auf einen Wert von 10 $k\Omega$ abfallen. Abbildung 3.3 zeigt eine typische Kennlinie für einen FSR Sensor, die annähernd einer invertierten Exponentialfunktion entspricht. Die tatsächliche Linie verändert sich mit dem jeweiligen Sensor, da sie von den Parametern der Folien, der Breite der Leiterbahnen und der Dicke der Klebeschicht abhängt. Hohe Kräfte führen zu seiner Sättigung des Widerstandswertes, während kleinere Kräfte eine Dynamik anzeigen.

Die Messwerterfassung erfolgt nach dem Spannungsteilerprinzip. Die grundlegende Beschaltung für jeden einzelnen Sensor wird in Abbildung 3.4 veranschaulicht. Der FSR wird mit einem Aktivierungssignal über den Transistor eingeschal-

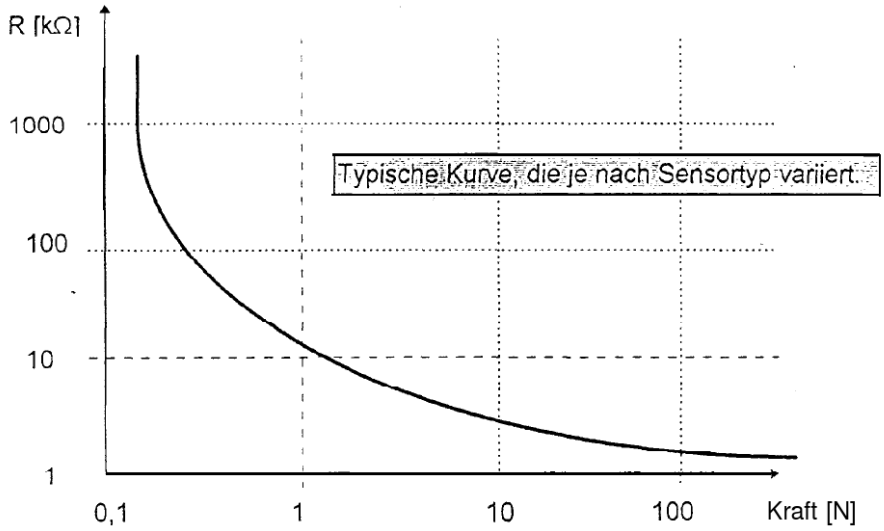


Abbildung 3.3: Kennlinie eines typischen FSR

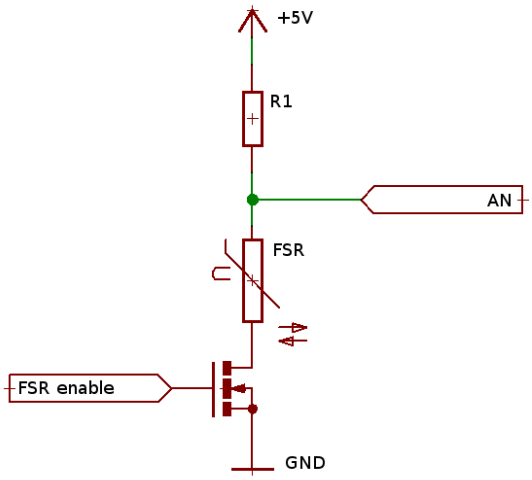


Abbildung 3.4: Prinzipschaltung für FSR

tet; danach wird die Spannung über dem FSR gemessen, während R_1 als Vorwiderstand dient. Im Folgenden wollen wir kurz die Größenordnung des Vorwiderstandes R_1 bestimmen. Für eine angenommene Kraftwirkung von 1N gehen wir von einem Wert von $R_{FSR} = 10k\Omega$[2]und einem Spannungsabfall von $U_{FSR} = 3V$ aus. Mit der Spannungsteilerregel $\frac{U_{FSR}}{U_{ges}} = \frac{R_{FSR}}{R_{FSR}+R_1}$erhalten wir auf einen Wert von $R_1 \approx 6666,667\Omega$. Wegen der physikalischen Parameter des FSR und der ungenauen Annahme des möglichen Spannungsabfalls U_{FSR}wird der tatsächliche Wert für R_1etwas erhöht. Nach praktischer Rundung wurde $R_1 = 10k\Omega$ festgelegt. Die gemessene Spannung wird so etwas absinken, aber experimentell bestätigt sich das Funktionsprinzip der Schaltung.
Je nach Subsystem entspricht das Ausgangssignal AN dem Signal B oder den Signalen AN1 und AN2.[3]Gerade die einfache Beschaltung der FSRs legt ihren Einsatz in einem robusten Sensorsystem nahe, da sie einfach aufzubauen und auszuwerten ist. Zur Auswertung ist lediglich eine Spannungsmessung mit Hilfe eines A/D-Wandlers erforderlich.

3.3 Lichtschranken

Lichtschranken stellen Objekte auf optischem Wege fest, d.h. das Vorhandensein oder Fehlen einer Lichtquelle gibt Auskunft darüber, ob ein Objekt erkannt wird oder nicht. Grundsätzlich bestehen sie aus zwei Komponenten. Einem aktiven Lichtsender, häufig im Infrarotbereich, steht eine Empfängerkomponente gegenüber, die das Licht des Senders empfängt. Wird kein Licht empfangen, so bedeutet dies eine Bedeckung der Lichtquelle. Je nach geometrischer Ausrichtung von Sender und Empfänger kann man zwischen Reflexlichtschranken und Gabellichtschranken unterscheiden. Bei einer Gabellichtschranke liegen Sender und Empfänger direkt gegenüber und ein Ausbleiben des Lichtes bedeutet, daß sich ein Objekt innerhalb der Lichtschranke befindet. Der Name erklärt sich durch den Gedanken, man würde ein Objekt zwischen die Zinken einer Gabel schieben, an

[2] vgl. Abbildung 3.3
[3] vgl. Seite 72 im Anhang

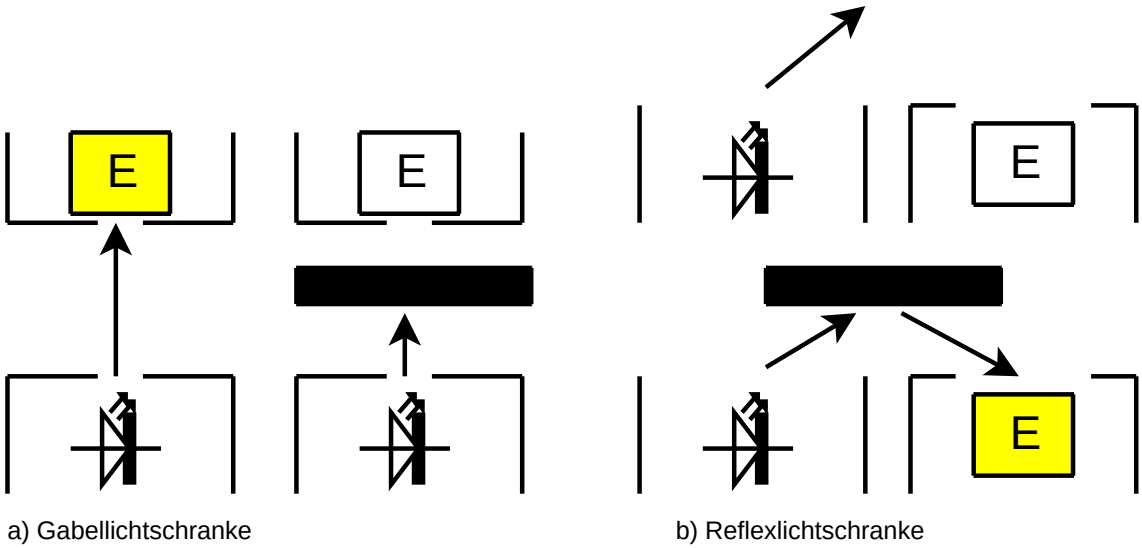


Abbildung 3.5: Lichtschrankentypen

denen Sender und Empfänger montiert sind.
Bei einer Reflexlichtschranke sieht der Empfänger die Lichtquelle nicht direkt, sondern empfängt die Reflexion des Lichtes vom Objekt. Hier bedeutet der Empfang von Licht also die Anwesenheit eines Objektes. Über die Auswertung der Laufzeit zwischen Lichtimpuls und Empfang kann eine Reflexlichtschranke sogar Entfernungen messen.
Abbildung 3.5 veranschaulicht die beiden Lichtschrankentypen, die LED symbolisiert dabei die Lichtquelle. Links ist eine Gabellichtschranke dargestellt. Befindet sich kein Objekt innerhalb der der Lichtschranke, so sieht der Empfänger das Licht, dargestellt durch den hinterlegten Empfänger. Wird die Lichtquelle nun durch ein Objekt bedeckt, so sieht der Empfänger das Licht nicht mehr. Daneben wird das gleiche Prinzip für eine Reflexlichtschranke gezeigt, bei der sich Sender und Empfänger nicht direkt sehen. Reflektiert nun ein Objekt das Licht im richtigen Winkel, so sieht der Empfänger wieder das Licht, gezeigt durch den hinterlegten Empfänger. Ist kein Objekt in Position, so strahlt der Sender sein Licht aus, ohne daß der Empfangsteil dieses sehen kann.
Konkret sollen im Bereich der Roboterhand drei Gabellichtschranken innerhalb

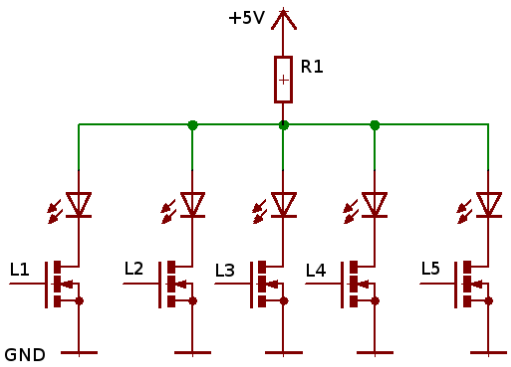


Abbildung 3.6: Schaltung für die Senderseite der Lichtschranken

der Finger feststellen, ob sich ein Objekt zwischen den Fingern befindet, sowie die ungefähre Position des Objektes, je nachdem ob nur die obere, die mittlere oder untere oder eine Kombination der Lichtschranken das Objekt erfasst. Zwei Reflexlichtschranken sind vorgesehen, um eine Objekterkennung oberhalb der Finger zu ermöglichen. Sie werden durch einen Abstandssensor[4] ergänzt.
Schaltungstechnisch besteht zwischen den beiden Typen keinen Unterschied. Der Lichtsender wird aktiviert, um den Empfänger zu beleuchten. Zeitgleich wird das Signal des Empfängers ausgewertet. Als Sender kommen Infrarotleuchtdioden vom Typ CQY37N[5] zum Einsatz. Die Verfügbarkeit der LED sowie ihre geringe Baugröße sprachen hauptsächlich für ihre Verwendung.
Abbildung 3.6 zeigt die Beschaltung der LEDs für die Lichtschranken, dabei ist $R_1 = 47\Omega$. Die Lichtschrankensender können individuell über die Signale L1 bis L5 ein- und ausgeschaltet werden. Die Transistoren wären unnötig gewesen, wenn man direkt über den Mikrocontroller die Masse schaltet. Da sämtliche anderen Sensoren über Transistoren geschaltet werden, wurden auch an dieser Stelle Transistoren vorgesehen, um die Kontinuität im Entwurf beizubehalten.
Als Empfängergegenstück kommen Fototransistoren vom Typ BPW17N[6] zum Einsatz, deren Empfangsspektrum zum verwendeten LED-Typ passt.

[4] vgl. Kapitel 3.4
[5] vgl. [18]
[6] vgl. [17]

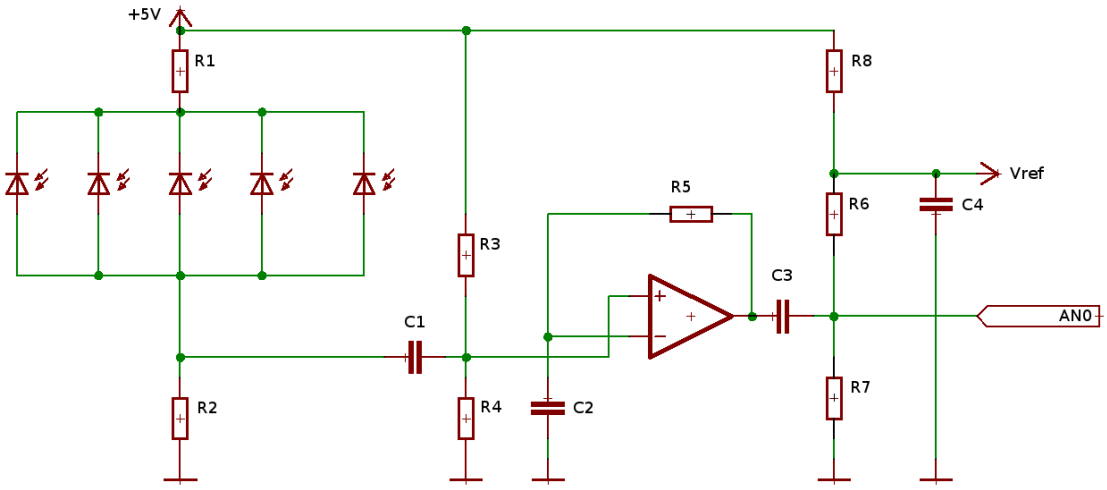


Abbildung 3.7: Verstärkerschaltung für Lichtschrankenempfänger

Da das Ausgangssignal des Transistors nicht ausreicht, um sinnvolle Werte[7] zu ergeben, muß es verstärkt werden. Der Schaltplan in Abbildung 3.7 zeigt die Verstärkerschaltung. Als Empfänger sind im Schaltplan Fotodioden statt Fototransistoren angegeben. Bis auf die Dimensionierung der Vorwiderstände ist dies für die Funktionalität unerheblich. Die Implementierung und die weitere Beschreibung geht von Fototransistoren vom Typ BPW17N aus.

Das Ausgangssignal liegt auf AN0 an. Sämtliche Empfängertransistoren sind parallel geschaltet, d.h. eine gleichzeitige Beleuchtung kann das Messergebnis verfälschen. Zum einen muß also dafür gesorgt werden, daß zu einem Zeitpunkt t jeweils nur eine LED aktiv ist, zum anderen muß dafür Sorge getragen werden, daß kein Störlicht erfasst wird. Softwareseitig wird dafür gesorgt, daß nur eine LED zur Zeit aktiv ist. Um Störlicht auszuschließen, wird das Lichtsignal auf eine charakteristische Weise moduliert, so daß es gut erkennbar bleibt. Dazu pulsen wir das Licht mit einer Frequenz von 10KHz und nehmen die Differenz zweier Messwerte pro Lichtschranke auf. Ein Hochpassfilter in der Verstärkerschaltung sorgt dafür, daß nur Signale im entsprechenden Frequenzbereich durchgelassen wer-

[7] Der Wert liegt im mV Bereich. Da der A/D-Wandler einheitlich für einen Bereich von 0 bis 5V beschaltet ist, ist ein entsprechend verstärktes Signal wünschenswert.

den. Praktisch liegen die folgenden Werte vor: C_1 , $C_3 = 3,3nF$, $C_2 = 1,5nF$, $R_{1,2} = 470\Omega$ und $R_{3,\ldots,8} = 100k\Omega$. Dabei wurde die Größenordnung von C_1 wie folgt bestimmt: $C_1 = \frac{1}{2*\pi*f*R}$, wobei $f = 10000Hz$ und $R = R_3 \parallel R_4 = 50k\Omega$
Der Vorteil der parallelgeschalteten Fototransistoren besteht darin, daß nur eine Verstärkerschaltung gebaut werden muss. Das spart Bauteile und Lötarbeit, was weniger Aufwand bei einer eventuellen Fehlersuche bedeutet. Ein etwas höherer Aufwand in Software erscheint nur leicht nachteilig, da der Mikrocontroller aufgrund seiner sequentiellen Arbeitsweise nicht in der Lage ist, fünf analoge Signale echt parallel einzulesen, und somit die Sensoren von vornherein hintereinander abfragt.
Die Anzahl der eingesetzten Lichtschranken lässt sich problemlos erhöhen, es sind lediglich weitere Fototransistoren parallel zu schalten und entsprechende LEDs mit Abschaltung vorzusehen.

3.4 Abstandssensor

Um Entfernungsangaben zu Objekten zu erhalten, die unmittelbar über den Fingern des Roboters im Arbeitsbereich liegen, wurde ein Sensor mit Abstandsmessung verbaut und dafür ein Sensor der Firma Sharp[8] eingesetzt. Der Sensor GP2D120 ist ein abstandsmessender Sensor mit integrierter Signalverarbeitung und einem Ausgang für analoge Spannungen, der automatisch die ermittelten Abstandsmessungen als Spannungspegel ausgibt.[9] Das Blockdiagramm in Abbildung 3.9 gibt einen Überblick über die im Sensor integrierten Komponenten.[10] Die Beschaltung für die Anwendung ist entsprechend einfach, neben der Spannungsversorgung muss nur das Ausgangssignal V_o des Sensors aufgenommen und durch einen D/A-Wandler gemessen werden. Zwischen V_o und dem D/A-Wandler liegt noch ein Schutzwiderstand von 100Ω. Das Signal V_o des GP2D120 entspricht dem Signal AN0 im Sensorsystem, sofern der Sensor über das Signal L0 aktiviert

[8] http://www.sharpsma.com/
[9] [14] gibt einen Überblick über die Eigenschaften und Anwendungsparameter des GP2D120.
[10] vgl. [14], S. 1

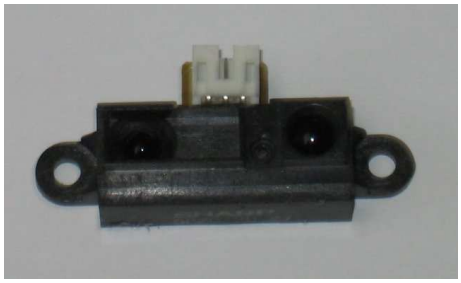


Abbildung 3.8: Sharp GP2D120

wurde.[11]

Zur Abstandsmessung verwendet der Sensor einen optischen Positionssensor, einen sogenannten PSD[12]. Ein flächiger Halbleiter wird durch die Reflexion des Lichtes der LED beleuchtet. Durch die unterschiedliche Belichtung ändert sich der Widerstand abhängig von der Position und Größe des Lichtpunktes. Die Spannung über dem PSD wird dann von der Einheit zur Signalverarbeitung ausgewertet und analog als Spannung ausgegeben.

Das Zeitverhalten des Sensors muß bei der Abfrage beachtet werden, sonst werden die Messwerte instabil. Der Sensor erfordert eine definierte Zeitspanne um seine Arbeit aufzunehmen, sowie eine feste Zeit, in der er eine neue Abstandsmessung veranlaßt. Aus dem Timingdiagramm in Abbildung 3.10 lassen sich eine Einschaltzeit von rund 45ms und eine ebensolange Zeitspanne bis zur nächsten Messung ablesen.[13] Ein Sensor kann daher rund 20mal in der Sekunde[14] abgefragt werden.

Abstandsmessungen für eine Objektentfernung von bis zu 30cm vom Sensor sind möglich. Abbildung 3.11 zeigt die Signalcharakteristik des Sensors laut Herstellerangabe.[15] Einer spezifischen Objektentfernung ist dabei ein Pegelwert für das Ausgangssignal zugeordnet. Aus dem Diagramm kann man folgende Schlüsse ziehen:

---

[11] Für die technische Zuordnung der Signale an Portpins des Mikrocontrollers sei im Anhang auf Seite 72 verwiesen.

[12] vgl. http://de.wikipedia.org/wiki/Position_Sensitive_Device

[13] vgl. [14], S. 2

[14] Bei durchschnittlich 50ms pro Messung passen 20 Messungen in 1000ms.

[15] vgl. [14], S. 4

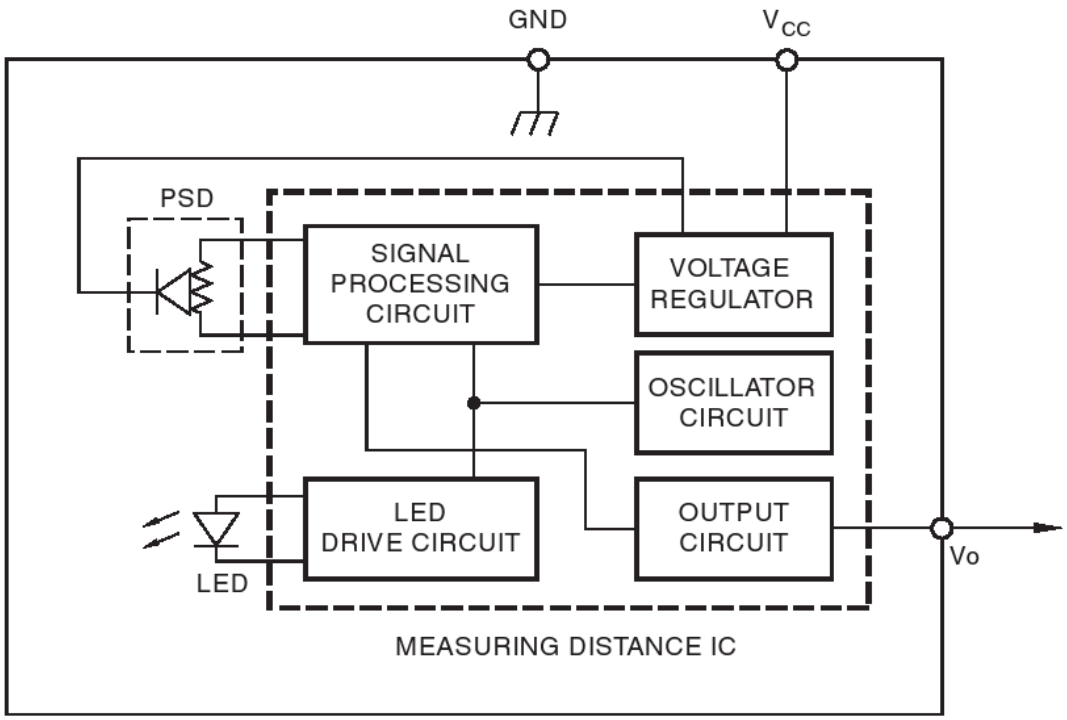


Abbildung 3.9: Blockdiagramm Sharp GP2D120

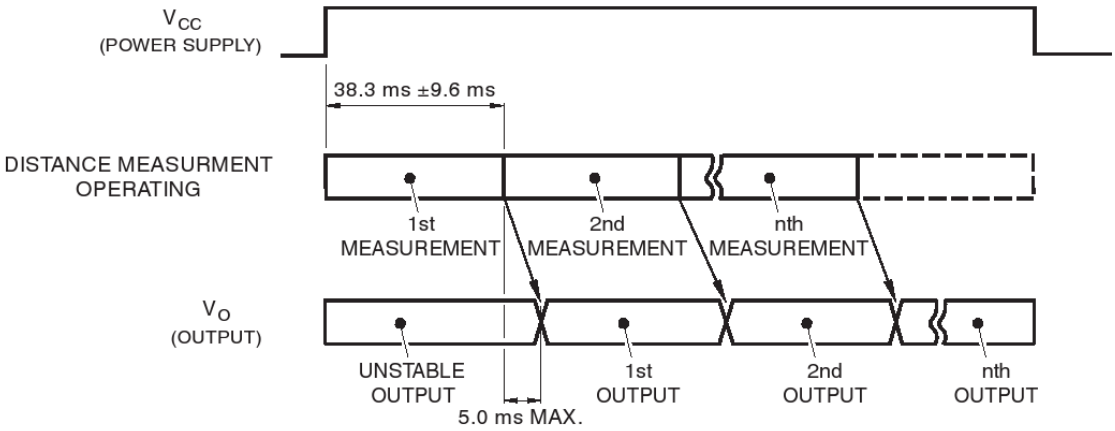


Abbildung 3.10: Timingdiagramm Sharp GP2D120

(a) Maximale Ausgangsspannungen von bis zu 3 Volt treten auf.

(b) Die Reflexionseigenschaften des vermessenen Objektes sind unerheblich. Experimentelle Versuche ergaben sowohl für ein Blatt Papier als auch eine massive dunkle Pappschachtel ähnliche bis identische Entfernungswerte.

(c) Das Messverhalten des Sensors ist nicht linear, der Spannungsanstieg des Ausgangssignales verläuft steiler bei einer größeren Annäherung und fällt ebenso steil bei einer zu großen Nähe ab.

Der Sharpsensor ist empfindlich gegenüber Störlicht, weswegen der gleichzeitige Betrieb von mehreren Exemplaren zu Störungen in den Messwerten führt.
Ein experimenteller Testaufbau[16] führte zur Verifikation des im Datenblatt genannten Timings, als auch zur Bestätigung von Störsignalen. Ein Ausfiltern der Störsignale erscheint nur unter erhöhtem Aufwand möglich. Weder eine getrennte Spannungsversorgung, noch eine Trennung paralleler Datenleitungen konnte die Störungen beseitigen. Eine Fehlerkorrektur in Software über digitale Filteralgorithmen erscheint prinzipiell möglich, aber das immanente Timing des Sensors bleibt weiterhin als Hürde bestehen. Allein durch die vom Sensor induzierte Zeitrasterung von 45ms zwischen aufeinanderfolgenden Messungen erscheint eine Fehlerrechnung der Messwerte unnötig.
Als Fazit bleibt stehen, daß der Sensor GP2D120 von Sharp zwar prinzipiell den Anforderungen genügt, sein Zeitverhalten aber den ortsnahen Einsatz mehrerer Exemplare ausschließt, sofern auf die Güte der Messung Wert gelegt wird. Von einer großvolumigen Verwendung des Sensors innerhalb einer Baugruppe mit ähnlicher Erfassungsrichtung wird daher abgeraten.

[16] vgl. Abbildung 3.12

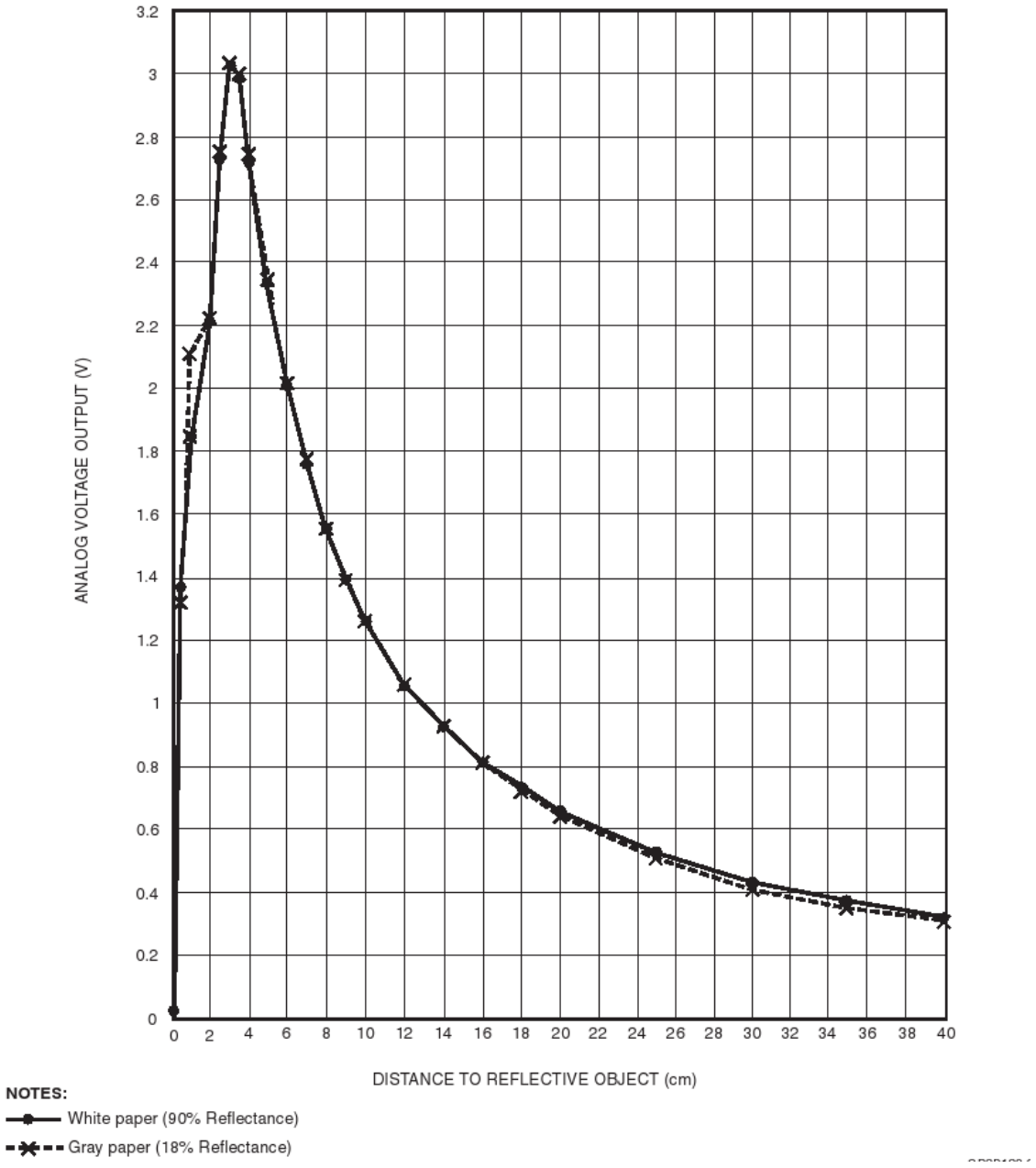


Abbildung 3.11: Signalcharakteristik Sharp GP2D120

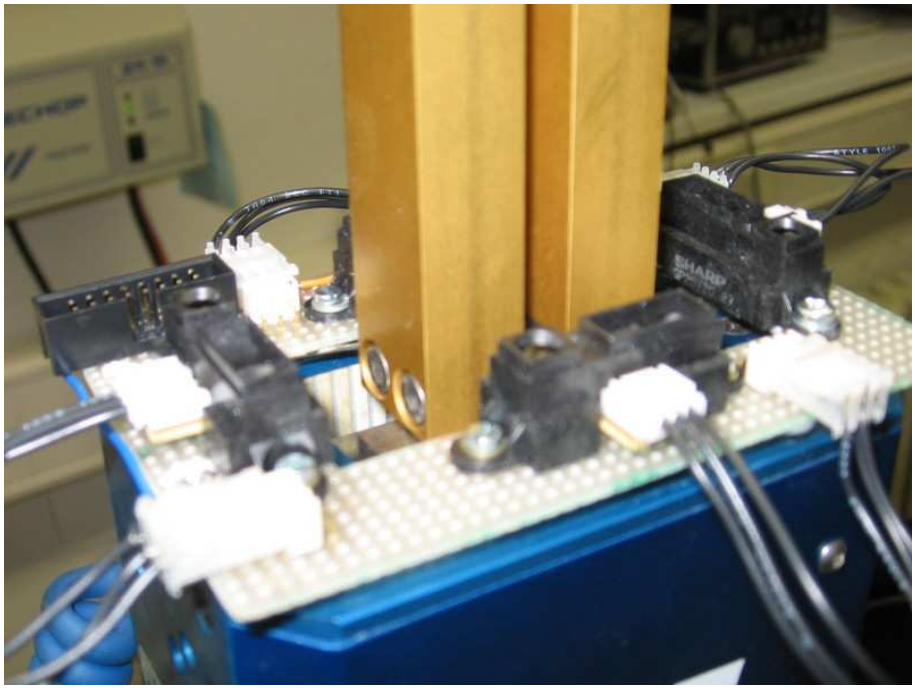


Abbildung 3.12: Versuchsaufbau für 4 parallel betriebene Sharp GP2D120 Sensoren

Kapitel 4

Die Hardware der Sensorabfrage

4.1 Übersicht

Die Hardware für die Sensoren im Bereich der Roboterhand besteht aus den folgenden Komponenten:

- Den eigentlichen „Fingern“, bzw. Greifern des Roboters
- einer Sensorplatine, die die Anschlüsse an die Finger und den Mikrocontroller bereitstellt, sowie selber Reflexlichtschranken enthält, die sogenannte Interfaceplatine
- eine Mikrocontrollerplatine, die die Kommunikation zwischen Roboter und Sensoren abwickelt

Die Mikrocontrollerplatine stellt die Verbindung zwischen den Sensoren und dem CAN-Bus, und damit den restlichen Subsystemen des Industrieroboters, her. Zur Regelung der Kommunikation wird ein T89C51CC02 Mikrocontroller[1] von der Firma Atmel[2] eingesetzt. Der Mikrocontroller hat folgende Hardwareeigenschaften:

- 8 Bit 8051[3] CPU mit 16KB flashbasiertem Programmspeicher, 256 Bytes

[1] [4] gibt einen detaillierten Überblick über die Hard- und Softwareeigenschaften des T89C51CC02

[2] http://www.atmel.com/

[3] [12] gibt einen Überblick über Programmiermodell und Befehlssatz eines 8051-kompatibelen Mikrocontrollers wie dem T89C51CC02

internem Datenspeicher und 2KB externem Datenspeicher

- 2K EEPROM Datenspeicher
- bis zu 20 digitale Ein- und Ausgabesignale[4]
- A/D Wandler mit 8 Kanälen und einer Genauigkeit von 12 Bit
- CAN-Protokollcontroller mit Schnittstelle für den CAN-Bus mit bis zu 4 Kanälen für Senden und Empfangen von Nachrichten
- serielle Schnittstelle nach RS232 Norm[5]
- Bootloadermechanismus wahlweise über CAN oder RS232

Die 8051 Architektur wurde gewählt, da sie einen Industriestandard darstellt. Prinzipiell entfällt so die Abhängigkeit, von einem Hersteller Bausteine beziehen zu müssen. Sie ermöglicht mehr Bezugsquellen für konkrete Bausteine. Für das Sensorsystem entfällt dieser Vorteil allerdings, da das System um einen bestimmten Mikrocontroller herum entwickelt wurde. Prinzipiell ist aber die Verwendung anderer Bausteine möglich.
Ferner ist eine Vielfalt kommerzieller und freier Softwareprodukte auf dem Markt erhältlich, welche die Entwicklung von Firmware für einen 8051 basierten Mikrocontroller erleichtern und von der Entwicklungsplattform unabhängig machen.
Für die Verwendung des T89C51CC02 spricht die Vielfalt an Optionen, die er bietet. Zentraler Punkt ist das vorhandene Interface zum CAN-Bus. Abgesehen von der nötigen Anpassung der elektrischen Pegel, stellt er vollen Zugriff auf die Nachrichten bereit, die über den CAN-Bus verschickt werden. Die Applikation kann so über reine Softwareroutinen mit der Außenwelt und anderen Robotersubsystemen kommunizieren.
Der A/D-Wandler ist essentiell für die Erfassung der Messwerte. Die Möglichkeit, mehrere Kanäle messen zu können, erlaubt ein Schaltungsdesign mit strikter Trennung von Subsystemen, d.h. die Signale der Lichtschranken werden auf einem anderen Kanal gemessen als die Kontaktsensoren an den Fingern. Die digitalen Ausgabemöglichkeiten erlauben die gezielte Aktivierung und Deaktivierung von

[4] Da die Anschlüsse des Mikrocontrollers mehrfach belegt sind, verringert die Verwendung der A/D-Wandler und Kommunikationsschnittstellen die Anzahl der verfügbaren digitalen Signale

[5] Der Mikrocontroller arbeitet direkt mit TTL Pegeln. Es ist ein Pegelwandlerbaustein vom Typ MAX232 nötig, um dem vollen RS232 Standard zu genügen.

Sensorsubsystemen, so daß Schaltungsteile abgeschaltet werden können, wenn sie nicht benötigt werden. Auf diesem Wege können Störungen durch verschiedene Sensorsysteme vermieden werden. Gerade die Lichtschranken profitieren davon.
Der externe Datenspeicher kann als Protokollspeicher für die Sensormessung verwendet werden, während das EEPROM dazu dienen kann, Einstellungen des Messverhaltens des Systems dauerhaft festzuhalten und sie bei einem Neustart sofort wieder verfügbar zu machen.
Die Möglichkeit, die Firmware über die serielle Schnittstelle und einen entsprechenden Bootloader einzuspielen, hat den großen Vorteil, daß einerseits der Mikrocontroller nicht wegen Reprogrammierens aus der Schaltung entfernt werden muß. Zum anderen kommt ein wohldefiniertes Protokoll[6] zum Einsatz, welches das verwendete Entwicklungssystem von einer bestimmten Umgebung oder Softwarekonfiguration unabhängig macht, solange über die RS232-Schnittstelle kommuniziert werden kann. Eine prinzipielle Verwendung des CAN-Bus zum Flashen der Firmware ist möglich, wurde aber nicht in Erwägung gezogen, da die CAN-Bus-Interfaces für die Entwicklungssysteme keinem offenen Standard folgen[7], während RS232 seit Jahrzehnten offen und wohldefiniert ist.

4.2 Die Finger

Das Fingerpaar ist eine Handanfertigung. An den Fingerspitzen sind FSR angebracht, die eine Objektberührung nach oben feststellen sollen. In der Innenseite eines Fingers sind zwei weitere FSR unter einer sogenannten Wippe[8] montiert. Damit kann festgestellt werden, ob ein Objekt gegriffen wurde, und ob dies im oberen oder mittleren Teil des Fingers geschah, je nachdem, ob einer oder beide der Sensoren unter der Wippe ansprechen.
Drei Lichtschranken, die horizontal in den Fingern eingelassen sind, erfassen das vertikale Eintauchen eines Objektes zwischen die Finger. Dabei enthält ein Finger

[6] vgl. [2]
[7] Hier im Sinne, daß die zum Flashen nötige Software nicht mit jedem CAN-Bus-Interface auf der PC-Seite zusammenarbeiten kann.
[8] Die Wippe ist in Abbildung 4.1 im rechten Finger zu erkennen.

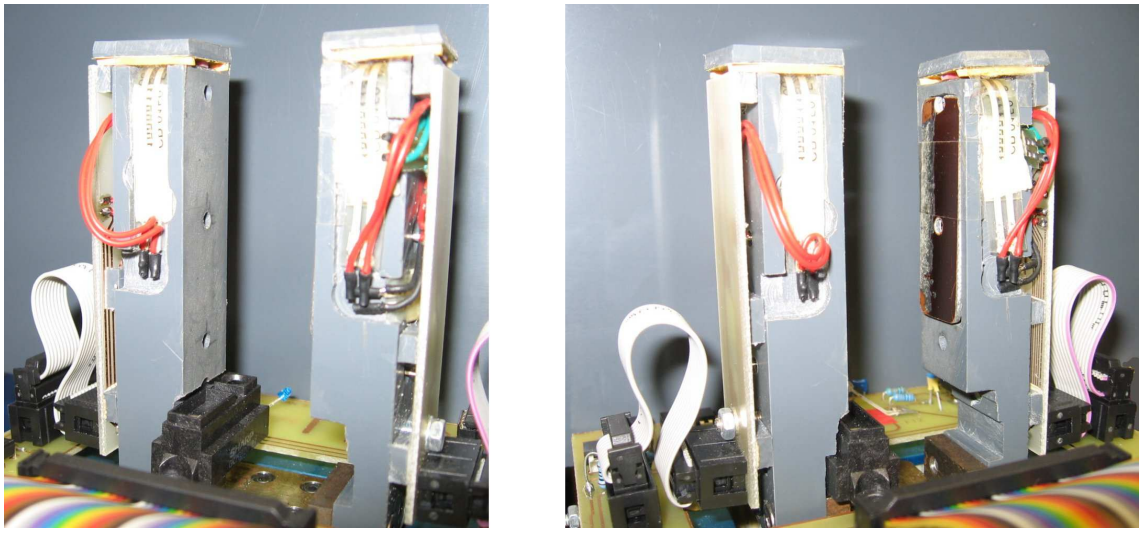


Abbildung 4.1: Die Roboterfinger mit den montierten Sensoren

die Leuchtdioden, die die Fototransistoren auf der anderen Fingerseite beleuchten. Ein einzelner Abstandssensor GP2D120 ist in einem Finger montiert und misst sowohl den Abstand zu einem Objekt zwischen den Fingern als auch über den Fingern. Sein Erfassungsbereich von ca. 30cm erlaubt die Erkennung einer festen Platte oder eines Tisches in Fingernähe. Der besonders sensitive Bereich nahe des Sensors bis knapp über die Fingerspitzen wird durch das Messverhalten des GP2D120[9] maßgeblich unterstützt. Höhere Spannungswerte bedeuten automatisch eine größere Objektnähe zum Sensor. Sein Ausgangssignal AN0 wird direkt zurückgereicht, während der Sensor über das Signal L0 und einen Steuertransistor ein- und ausgeschaltet wird.

Die Finger enthalten keinerlei Elektronik bis auf den integrierten Sharp GP2D120[10] sowie LEDs und Fototransistoren. Lediglich Leitungen und Signale werden zusammengefasst und über ein jeweils zugeordnetes Kabel an die Sensorplatine weitergereicht. Insbesondere wird im Finger keine Sensorabschaltung durch Transistoren vorgenommen. Es handelt sich nur um Anschlüsse, die intern verwendet werden. Ein Routing von Steuersignalen wäre prinzipiell möglich, aber der be-

[9] vgl. Abbildung 3.11

[10] Im Folgenden bedeutet Abstandssensor synonym GP2D120 und umgekehrt.

Pinnummer	Signal
1	Analogausgang GP2D120 (AN0)
2	Versorgungsspannung GP2D120
3	Masse
4	Versorgungsspannung Leuchtdioden Lichtschranken
5	nicht verbunden
6	Masse Leuchtdiode unten
7	Masse Leuchtdiode oben
8	Masse Leuchtdiode Mitte
9	FSR Fingerspitze Eingang
10	FSR Fingerspitze Ausgang

Tabelle 4.1: Pinbelegung Finger mit GP2D120

Pinnummer	Signal
1	FSR Fingerspitze Eingang
2	FSR Fingerspitze Ausgang
3	FSR Wippe unten Eingang
4	FSR Wippe unten Ausgang
5	FSR Wippe oben Eingang
6	FSR Wippe oben Ausgang
7	gemeinsamer Kollektor Fototransistoren/+5V
8	Emitter Fototransistor unten/GND
9	Emitter Fototransistor oben/GND
10	Emitter Fototransistor Mitte/GND

Tabelle 4.2: Pinbelegung Finger mit Wippe

schränkte Platz am und im Finger legt eine möglichst einfache Beschaltung nahe. Die nötigen Anschlußleitungen sind entsprechend paarweise verlegt: Dabei wurden gemeinsame Signale wie die Spannungsversorgung zusammengefasst und einzeln abschaltbare Leitungen belassen. Pro Finger konnten die benötigten Anschlüsse auf insgesamt 10 Stück reduziert werden.
Die Pinbelegung der 10poligen Pfostenstecker kann den Tabellen 4.1 und 4.2 entnommen werden.
Die Kontaktsensoren auf den Fingern und in der Wippe sind über eine Matrix beschaltet. Jeweils eines der beiden Widerstandspaare wird über die Steuersignale

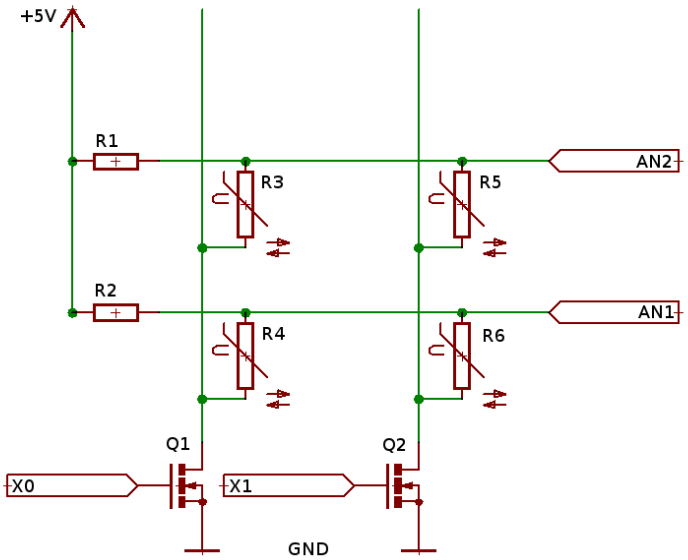


Abbildung 4.2: Matrixbeschaltung für die Drucksensoren in den Fingern

X0 oder X1[11] angewählt. Der Wert des einen Sensors liegt dann am Punkt AN1 und der Wert des anderen Sensors liegt an AN2 an. Beide können dann entsprechend vom Mikrocontroller ausgelesen werden. Abbildung 4.2 zeigt den Schaltplan. R_3, R_4, R_5 und R_6 stellen dabei die veränderbaren Widerstände der einzelnen Sensoren dar. Die Vorwiderstände R_1 und R_2 bilden die Eingangswiderstände der beiden Spannungsteiler, beide haben einen Wert von 10 $k\Omega$. Wenn X0 den Transistor Q1 einschaltet und Transistor Q2 ausgeschaltet ist, dann bilden R_1 und R_3 einen Spannungsteiler, sowie R_2 und R_4. Der Spannungsabfall über R_3 entspricht AN2, sowie AN1 dem Spannungsabfall über R_4 entspricht. Analoges gilt für die Alternative X0 deaktiviert und X1 ist aktiviert für R_5 und R_6.
In einem der Finger sind die Geber für drei Lichtschranken entlang der Hochachse des Fingers montiert. Sie werden über einen gemeinsamen Vorwiderstand und eine gemeinsame Spannung versorgt. Über Signale L1,L2 und L3 können die LEDs einzeln über Transistoren, die als Lowsideschalter[12] dienen, ein und ausgeschaltet

[11] Hier ist ein ausschließendes, bzw. exklusives Oder gemeint, d.h. entweder X0 oder X1 ist aktiv, aber niemals beide. Die Einhaltung dieser Nebenbedingung muß durch die Steuersoftware garantiert werden.

[12] Die entsprechenden Transistoren sind nicht in den Fingern, sondern auf der oberen Sensorpla-

werden.
Im Finger gegenüber der Geber sind die entsprechenden Fototransistoren als Empfänger eingelassen. Sie sind über einen gemeinsamen Kollektor angeschlossen, und ihre Emitter werden auf der Sensorplatine zusammengefasst. Wiederum enthalten die Finger nicht mehr Elektronik und routen nur die rohen Signale.

4.3 Die obere Sensorplatine

Die oben auf der Roboterhand montierte Platine stellt das Bindeglied zwischen den Fingern und der Mikrocontrollerplatine her. Synonym wird sie auch als Interfaceplatine bezeichnet. Sie erfüllt folgende Aufgaben:

- Routing der Signale zwischen Sensoren und Mikrocontroller
- Bereitstellung von Ein- und Ausschaltern für individuelle Sensoren
- Verstärkerschaltung für die Lichtschranken.
- Bereitstellung zweier Reflexlichtschranken zur Unterstützung des Abstandssensors

Für die Routingaufgaben enthält die Platine Busverbinder. Ein großer 34poliger Verbindungsstecker stellt die Verbindung mit dem Mikrocontrollerboard her. Die beiden Finger und die in ihnen montierten Sensoren sind über Flachbandkabel mit der Platine verbunden. Pfostenstecker und -wannen wurden eingesetzt, um neben einem einfachem Austauschen der Kabel eine einfache Demontage der Komponenten zu ermöglichen.
Die Platine enthält keinerlei Bedienelemente oder Anzeigen. Abbildung 4.3 zeigt die Platine an der Roboterhand montiert. Erkennbar sind von links hinten nach rechts hinten:

(a) Verstärkerbeschaltung für die Lichtschranken
(b) hintere Reflexlichtschranke
(c) 5 NMOS-FET-Transistoren für Ein- und Ausschalten der Reflexlichtschranken
(d) Anschluß der Sensoren vom linken Finger

tine verbaut. vgl. Kapitel 4.3

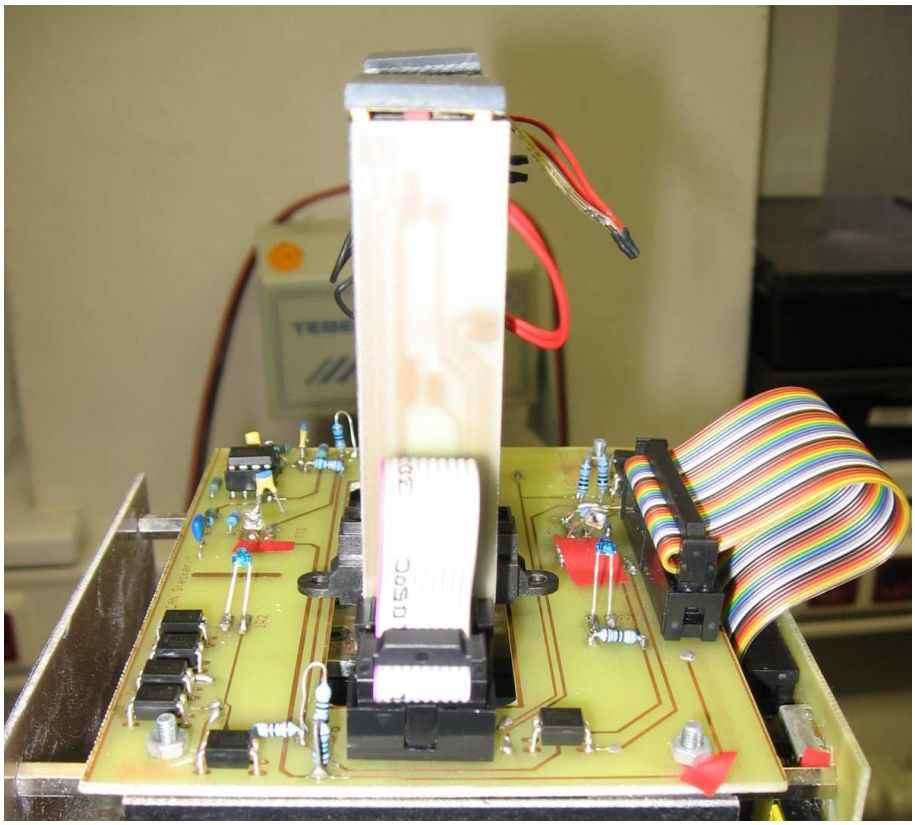


Abbildung 4.3: obere Sensorplatine in Montagestellung

(e) NMOS-FET-Transistor für die Aktivierung und Deaktivierung des Sharp GP2D120
(f) vordere Reflexlichtschranke
(g) Busverbinder zur Mikrocontrollerplatine
(h) N-MOS-FET-Transistoren für die Matrixbeschaltung der Drucksensoren in den Fingern
(i) Anschluß für rechten Finger (verdeckt durch Fingerpaar)

Die einzelnen Transistoren werden vom Mikrocontroller über digitale Steuersignale aktiviert. Die Signale liegen am Bus mit an und führen zu den jeweils zugehörigen Portpins des Mikrocontrollers.

4.4 Kollisionserkennung am oberen PowerCube

Zur Kollisionserkennung sind an den acht Eckpunkten des obersten Elementes des Roboterarmes Drucksensoren angebracht. Abstandshalter aus Kunststoff sorgen für die korrekten Auflagepunkte. Ein Teil der Leiterplatten und einer Verkleidung dient dazu flächigen Kontakt aufzunehmen, der dann über die Abstandshalter die Sensoren betätigt. Eine Betätigung der Seitenflächen führt so zur Auslösung von mehreren Sensoren.
Abbildung 4.4 zeigt die mechanische Montage. Auf den Ecken des Cubes sitzen Abstandshalter aus Plastik, die zur Innenseite abgeschrägt sind. Zwischen Abstandshalter und und Cube ist der jeweilige Sensor montiert. Auf der Abbildung sind vier von den acht Sensoren mit ihren Anschlußfahnen sichtbar. Eine kleine Scheibe sorgt für einen sauberen Druckpunkt vom Abstandshalter auf den Sensor. Dies ist links unten im Bild gut erkennbar. Auf die Metallhülsen wird die Mikrocontrollerplatine festgeschraubt. Das Flachbandkabel an der rechten Seite führt schließlich die Anschlüsse aller acht Sensoren heraus.
Elektrisch sind die Sensoren als reguläre Widerstände beschaltet und werden über ein Kabel mit 16 Adern zum entsprechenden Busverbinder auf der Mikrocontrollerplatine geführt. Eine Reihe von NMOS-Transistoren arbeitet als Lowside-Schalter zu den einzelnen Sensoren. Ein Decoderbaustein vom Typ 74HCT237[13] nimmt eine Auswahldekodierung vor und wählt dann einen von den acht Sensoren über den zugehörigen Transistor an und der Mikrocontroller kann den Wert des Sensors messen. Auf diese Art und Weise ist nur ein Messpunkt nötig. Nachteilig ist, daß auf diese Art und Weise immer nur einer der Sensoren abgefragt werden kann. Von Vorteil ist das Einsparen der Leitungsinfrastruktur für acht parallele Analogsignale und die logische Auswahlsequenz, die dem sequentiellen Programmablauf in der Firmware entgegenkommt. Der Vorwiderstand für den Spannungsteiler ist nahe am Verbindungsstecker auf der Mikrocontrollerplatine zu finden. Abbildung 4.5 verdeutlicht das Schaltungsprinzip.
Um die Sensoren abfragen zu können, wird ein Aktivierungssignal BEN[14] auf

[13] [11] gibt einen Überblick über Beschaltung und Logik des Bausteins.
[14] Das Aktivierungssignal BEN entspricht dem Signal E2 des 74HCT237. Die verbleibenden Si-

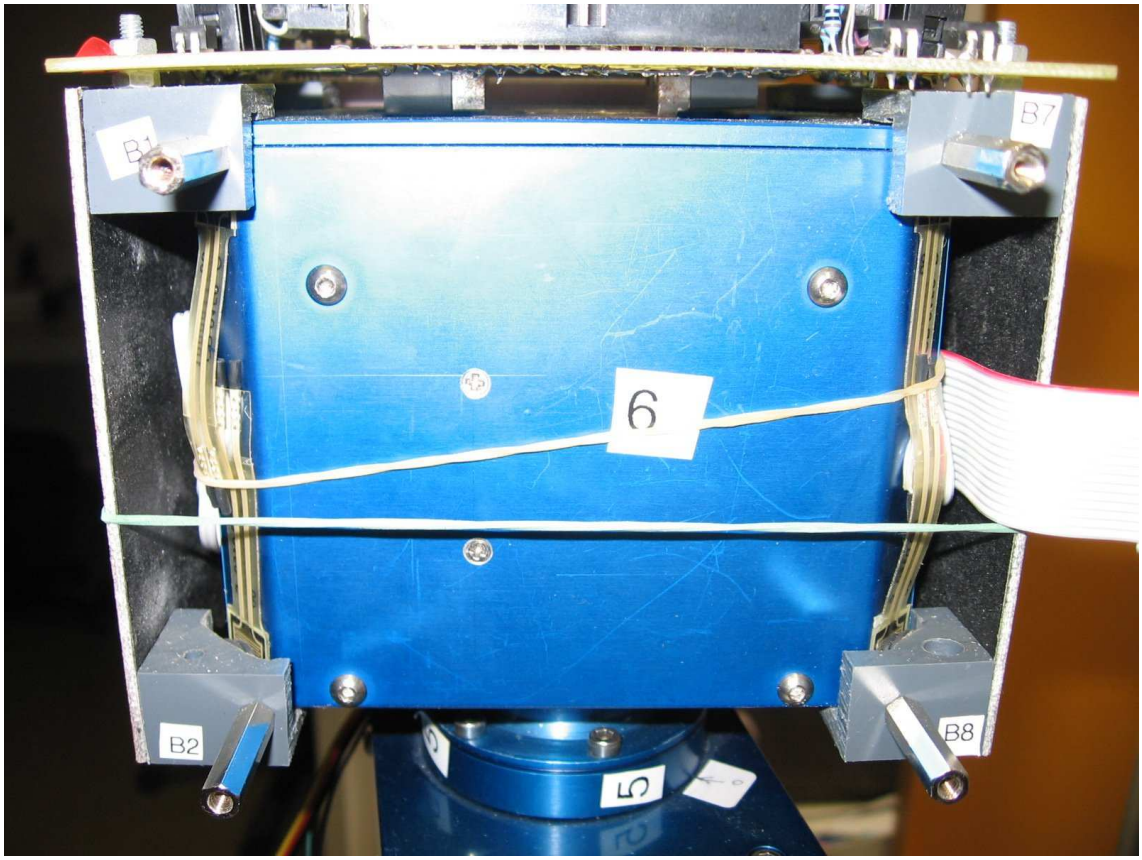


Abbildung 4.4: Montage der Bumper zur Kollisionserkennung

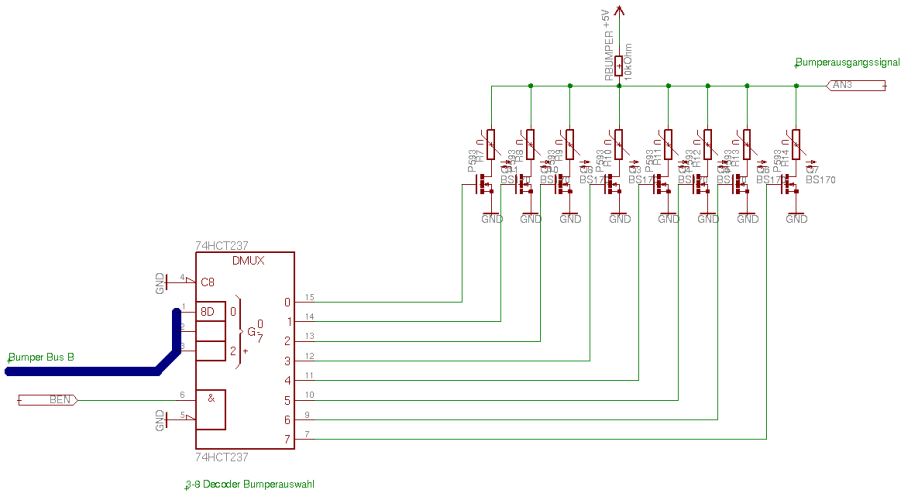


Abbildung 4.5: Beschaltung der Kollisionserkennung

High-Pegel geschaltet und die Nummer des abzufragenden Sensors binärcodiert am Decoderbaustein angelegt. Das Aktivierungssignal gibt den Decoderbaustein frei, so daß einer der acht Sensoren über seinen entsprechenden Transistor aktiviert wird. Die Auswahl des jeweiligen Sensors geschieht über den Bus B. B0 entspricht dabei A0 des 74HCT237, analog für B1 und B2. Das Ausgangssignal des Sensors (AN3) kann dann vom Mikrocontroller mit dem A/D-Wandler ausgelesen und ausgewertet werden.
Anstelle des Decoders wäre eine Matrixbeschaltung ähnlich der Fingersensoren möglich gewesen. Die Lösung mit dem Decoderbaustein bietet den Vorteil, daß sie übersichtlich ist. Sowohl im Schaltplan, als auch in der Montage ist ihre Funktionalität sofort ersichtlich. Allerdings ist eine Vereinheitlichung der Anwahl von Sensoren für zukünftige Projekte von Vorteil, da der Bauteilaufwand abnimmt und das Prinzip der Ansteuerung vereinheitlicht wird.

4.5 Die Mikrocontrollerplatine

4.5.1 Überblick

Abbildung 4.6 zeigt die bestückte Mikrocontrollerplatine. An Funktionskomponenten sind zu finden:

- der T89C51CC02 Mikrocontroller im PLCC28 Sockel
- Anpassung der CAN-Bus Pegel mit einem PCA82C251 und Abschlusswiderstand
- Überwachung und Erzeugung des Resetsignals für den Mikrocontroller unter Verwendung eines MAX700 Reset Supervisors[15]
- Spannungsversorgung
- Taktversorgung für den Mikrocontroller

gnale /LE und /E1 des 74HCT237 sind bis auf den Auswahlbus A0-A2 fest mit Masse verdrahtet. Fehlt das Aktivierungssignal, so schaltet der Decoder alle Ausgangsleitungen auf Low-Pegel und damit die Transistoren ab.

[15] [10] beschreibt das verwendete IC und verdeutlicht seine Verwendung in einer Schaltung. Auf der Platine ist der MAX700 entsprechend Abbildung 2 auf S. 5 von [10] beschaltet. Lediglich wird das positive Resetsignal anstelle des negativen verwendet, da der T89C51CC02 ein Resetsignal mit positiver Flanke erwartet.

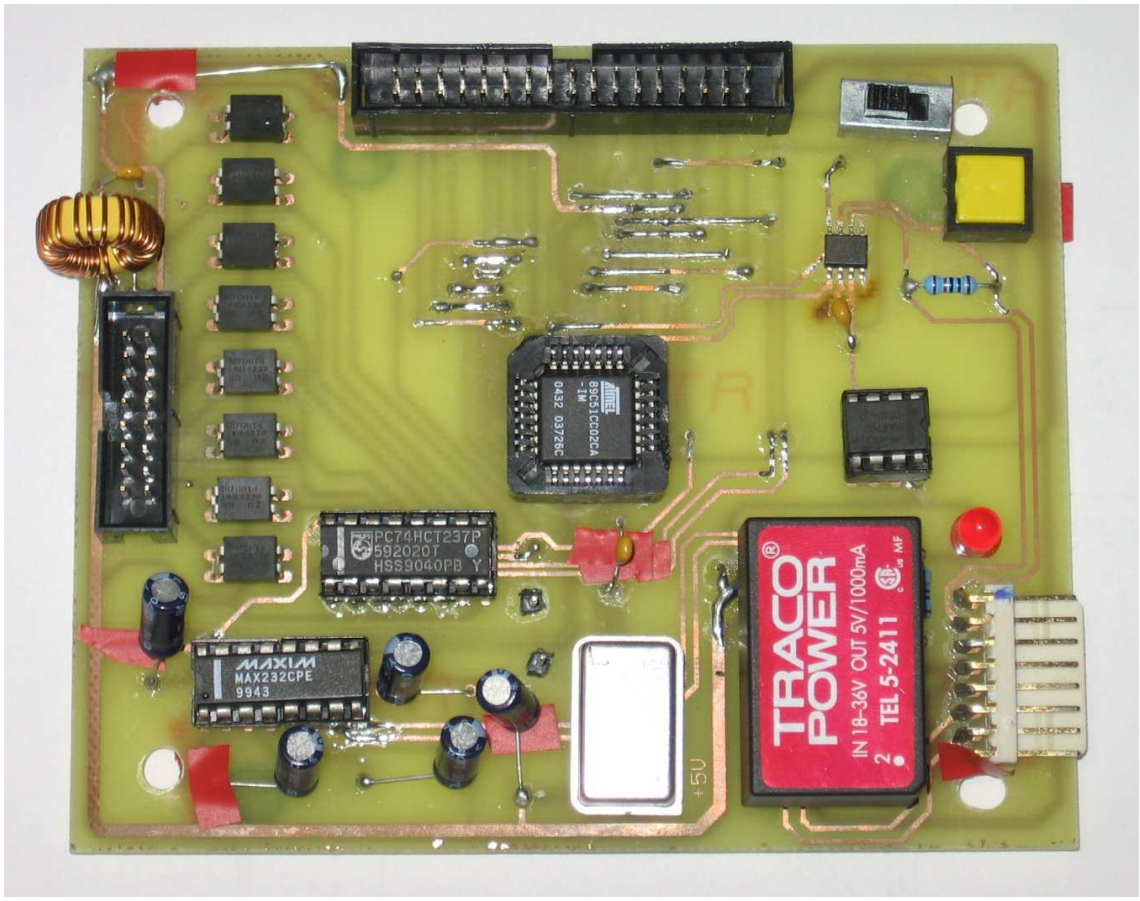


Abbildung 4.6: Die Mikrocontrollerplatine (Bestückungsseite)

- Pegelanpassung für RS232 Schnittstelle
- 3 aus 8 Decoder zur Anwahl der Bumpersensoren
- 8 NMOS-Transistoren, um die Bumper ab zu schalten
- Verbindungsstecker zu den Bumpersensoren
- Erzeugung einer entkoppelten 5V Versorgung für die analogen Systemteile
- Busverbinder zu den Fingersensoren und Lichtschranken
- Schiebeschalter zur Aktivierung des Bootloaders
- Resettaster, um das System von Hand neustarten zu können

Verschiedene Bussysteme sind auf der Mikrocontrollerplatine zu finden. Sie stellen die Verbindung der Systemkomponenten mit dem Mikrocontroller her. Insgesamt beinhalten sie die Anschlüsse an die obere Platine und die Finger, die

Ansteuerung der Bumpersensoren, sowie den Anschluß an den CAN-Bus und die serielle Schnittstelle.
Die acht Transistoren im DIL-Gehäuse, sowie der 3-8 Decoder sind Bestandteil der Abfrage der Bumper am oberen Würfel.

4.5.2 Das CAN-Bus-Interface

Um das Sensorsystem mit dem CAN-Bus zu verbinden, wird ein Interfacebaustein der Firma Philips Semiconductors verwendet. Der PCA82C251[16] stellt die Verbindung zwischen dem Protokollcontroller im Mikrocontroller und dem physikalischen Bus her.[17] Er setzt dabei die TTL-Signale des Mikrocontrollers in die entsprechenden Pegel für den CAN-Bus um und sorgt so für eine Trennung von Datenfluss und Signalerzeugung.[18] Über ein Slopeeingangssignal ist eine Abschaltung des Bausteins möglich.[19] Da von einer ständigen Kommunikationsbereitschaft des Sensorsystems ausgegangen wird, ist der Transcieverbaustein dauerhaft aktiviert und eine Kommunikation jederzeit möglich.
Auf der Platine sind nur noch der Terminatorwiderstand mit einem Wert von 120Ω zu finden[20], sowie die Leitungen zum physikalischen Bus. Bedingt durch den eingebauten Terminator muss die Mikrocontrollerplatine den letzten Teilnehmer am CAN-Bus darstellen. Wegen ihrer exponierten Lage am letzten Powercube, sowie der Tatsache, daß bisher[21] am Anschlußpunkt für die Mikrocontrollerplatine direkt ein Terminatorwiderstand angeschlossen war, wurde auf eine Abschaltbarkeit des Terminators verzichtet. Für den unwahrscheinlichen Fall, einen weiteren Teilnehmer hier am CAN-Bus hinzufügen zu wollen, sollte ein entsprechender Signalabzweig vor der Platine vorgesehen werden.
Die Ausgangssignale CAN-Hi und CAN-Lo stehen an der Anschlußleiste zur Verfügung und können über entsprechende Kabel und Stecker verbunden wer-

[16] Für Bausteineigenschaften und Beschaltungsdetails vgl. [13]
[17] vgl. [13], S.2
[18] vgl. Kapitel 2.2.1
[19] vgl [13], S.4
[20] Er liegt direkt unterhalb des Resettasters, gut erkennbar auf der rechten Seite von Abbildung 4.6.
[21] „Bisher“ entspricht dem Stand von Juni 2007, zu dem die Platine noch nicht fest montiert war.

Pinnummer	Anschluß am Mikrocontroller
1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33	Masse
2,4,6,8,10,12,14,16	P1.0,..., P1.7
18,20	P2.0, P2.1
22,24,26,28,30,32	P3.2,..., P3.7
34	+5V (analoge Versorgungsspannung)

Tabelle 4.3: Pinbelegung Busstecker

Abbildung 4.7: Schiebeschalter für die Bootloaderaktivierung (Position „Bootloader“)

den. Die prinzipielle Beschaltung entspricht im Ganzen Abbildung 2.1, wobei der CAN-Controller im T89C51CC02 integriert ist und der PCA82C251 den CAN-Transciever darstellt.

4.5.3 Busverbinder und Bedienungselemente

Der Busverbinder zu den Bumpern enthält lediglich paarweise Leitungen zu den einzelnen Widerständen der Bumper. Nacheinander ist jeweils ein Pin mit ungerader und ein Pin mit gerader Nummer für einen einzelnen Sensor zuständig.
Der zentrale Busverbinder enthält neben einer gemeinsamen Masse Verbindungen zu sämtlichen Ein- und Ausgabepins, sowie einen Pin mit einer Versorgungsspannung von 5V. Die genaue Pinbelegung kann Tabelle 4.3 entnommen werden.
Abbildung 4.7 zeigt den Schiebeschalter für die Bootloaderaktivierung. Wird der Schalter nach links geschoben, so ist nach dem nächsten Reset des Mikrocontrollers die Benutzerapplikation aktiv, Stellung „Applikation“. Nach rechts geschoben wird nach dem nächsten Reset der Bootloader aktiviert, Stellung „Bootloader“. Die Abbildung zeigt den Schalter in der Stellung „Bootloader“.[22]

[22] Da die Schalterstellung „Bootloader“ eine der analogen Eingangsleitungen auf Masse zieht, bedeutet diese Schalterstellung für den Applikationsbetrieb eine fehlerbehaftete Erfassung der

Pinnummer	Verwendungszweck
1	CAN-Bus Hi
2	CAN-Bus Lo
3	Masse
4	Versorgungsspannung +24V
5	Masse
6	RS232 RxD
7	RS232 TxD
8	Masse

Tabelle 4.4: Pinbelegung Roboterinterface

Ein Druck auf den Resettaster startet den Mikrocontroller neu. Im Normalbetrieb ist seine Betätigung nicht nötig und sollte vermieden werden.

4.5.4 Anschlüsse

Den Anschluß der Platine an das restliche Robotersystem stellt ein 8poliger Steckverbinder sicher. Tabelle 4.4 gibt die Pinbelegung des Steckers wieder. Pin 1 ist dabei der obere des Steckverbinders, wenn man die Platine mit dem breiten Busstecker nach oben mit Blick auf die Bestückungsseite betrachtet. Es wurden redundante Verbindungen zur Masse vorgesehen, um nötigenfalls drei unterschiedliche Kabel und Stecker für CAN-Bus, Spannungsversorgung und serielle Schnittstelle verwenden zu können. Pegelmäßig sind die drei Masseleitungen äquivalent.

Sensorwerte. Es ist ratsam, die Stellung „Bootloader“ nur dann anzuwählen, wenn tatsächlich eine neue Firmware geflasht werden soll. In diesem Fall sollte nach dem Verschieben des Schalters sofort der Resettaster betätigt werden.

Kapitel 5

Die Firmware der Sensorabfrage

5.1 Übersicht

Zentrale Aufgabe der Firmware ist es, eine regelmäßige Abfrage aller Sensoren zu gewährleisten und damit dem Benutzer oder dem übergeordneten Steuerprogramm zu ermöglichen, die Werte der Sensoren auszulesen. Eine direkte Auswertung von Sensorwerten in der Firmware erscheint möglich, doch die Verarbeitungskapazität des eingesetzten Mikrocontrollers setzt Grenzen. Ferner erlaubt die Auslagerung der Applikationslogik eine sehr viel weitergehende Verarbeitung und Kombination der Messwerte mit anderen Verfahren, etwa aus der Bilderkennung und von anderen Subsystemen.
Hauptaugenmerk der Firmware liegt darin, eine regelmäßige Sensorabfrage beizubehalten, gleichzeitig aber eine unterschiedliche Auswahl an Abfrage- und Meldeoptionen zu bieten. Der Benutzer kann über ein eigenes Kommunikationsprotokoll[1] verschiedene Arten der Sensorabfrage mittels geeigneter Kommandonachrichten ein- und ausschalten. Insgesamt stehen dem Benutzer die folgenden Optionen zur Verfügung:

- gezieltes Auslesen eines oder mehrerer Sensoren
- sogenanntes Monitoring eines oder mehrerer Sensoren, bei dem die Unter- oder Überschreitung des Messwertes im Vergleich zu einem einstellbaren

[1] vgl. Kapitel 5.4

Schwellwert gemeldet wird

- ein sogenannter Report, bei dem die Messwerte aller Sensoren ausgegeben werden

Ein Backend, welches dezentral auf einem anderen Rechner ausgeführt wird, kann so die Sensoren auslesen und auswerten. Als Reaktion darauf können dann an Arm und Fahrwerk des Roboters weitere Kommandos gesendet werden, die beispielsweise einen Greifvorgang abbrechen oder korrigieren. Die Möglichkeiten erscheinen vielfältig und flexibel und bieten ein breites Feld der Anwendung für die Sensoren.

Die Firmware für die Sensorabfrage wurde in C geschrieben und ist in verschiedene Module aufgeteilt. Das erhöht die Übersicht, ermöglicht die Wiederverwendung von Komponenten und vereinfacht die Wartung. Ferner erlaubt sie eine prinzipielle Portierung auf andere Mikrocontroller.[2] Die Firmware ist in eine Reihe von Subsystemen gegliedert:

- Messwerterfassung der Fingersensoren
- Erfassung der Lichtschranken
- Abfrage des Abstandssensors GP2D120
- Auslesen der Bumper zur Kollisionserkennung
- Kommunikation über den CAN-Bus
- Kommandoverwaltung und Kontrolle für Abfragebefehle
- Ansteuerung des A/D-Wandlers
- Timerkontrolle zur Generierung des Messrasters
- Hauptprogramm

Das Hauptprogramm initialisiert die einzelnen Subsysteme, meldet über den CAN-Bus seine Bereitschaft zum Empfang von Kontrollkommandos und beginnt dann mit der Messwertaufnahme. Parallel dazu wird der CAN-Bus auf Steuerkommandos überwacht. Nach jeder Messung werden entsprechend der Konfigurationsoptionen Messwerte versendet. Ein Report aller Sensoren überschreibt dabei Anforderungen zum Lesen einzelner Sensoren oder die Überwachung von Schwellwer-

[2] Für eine Portierung wären natürlich alle Hardwareeigenschaften und -ansteuerungen anzupassen. Der dafür nötige Arbeitsaufwand kann durchaus höher liegen, als bei einer kompletten Neuentwicklung für einen anderen Mikrocontrollertyp.

ten, da der Report diese Informationen ebenso beinhaltet. Hauptsächlich wurde das Ausschlußverfahren gewählt, um Rechenzeit einzusparen und um den CAN-Bus nur soweit wie erforderlich mit neuen Nachrichten zu belasten.

5.2 Zeitgefüge und Ablauf der Datenerfassung

Um eine sinnvolle Auswertung der Sensordaten zu erzielen, ist eine Aufnahme von Messwerten in einem festen Zeitraster vorteilhaft. Irreguläre Zeitabstände zwischen den Messungen können die Bildung von Mittelwerten verschlechtern. Die Einhaltung eines festen Abstandes der Messungen ist daher sinnvoll.
Wenn möglich soll dabei die Messwerterfassung in Echtzeit geschehen. Echtzeit bedeutet in diesem Fall eine Einhaltung definierter Zeiten, bis eine Antwort erfolgt.[3]
Für die insgesamt 18 Sensoren können wir den Zeitbedarf abschätzen.[4] Der Abstandssensor GP2D120 benötigt ca. 45ms bis er einen gültigen Messwertwert liefert, die 17 anderen Sensoren benötigen hauptsächlich die Zeit für eine A/D-Wandlung, die ca 1 Mikrosekunde[5] dauert. Beachtet man den Zeitaufwand, um Sensoren anzuwählen und abzuschalten und um Messergebnisse in Register zu transferieren, so kommt man auf eine grobe Schätzung von 50 bis 55ms für den Zeitbedarf aller Messungen.[6]Daher nehmen wir ein Zeitraster von 60ms an, in dem Messungen durchgeführt werden und etwas Zeit verbleibt, um Messergebnisse abhängig von der aktuellen Parametereinstellung zu versenden.
Die Theorie unterscheidet zwischen harter und weicher Echtzeit. Bei harter Echtzeit hat eine Verletzung der Zeitbedingung erhebliche Folgen für das System, wäh-

[3] vgl. [15], S.14

[4] Für eine genauere Zeitanalyse muss der erzeugte Maschinencode und die genaue Taktung des Systems berücksichtigt werden. Allein durch den festen Zeitaufwand von ca. 45ms für den GP2D120 ist aber schon eine untere Schranke für das Zeitfenster vorgegeben, deren Größenordnung nahelegt, daß eine Abweichung der Zeitschätzung im Mikrosekundenbereich irrelevant ist.

[5] Der A/D-Wandler des T89C51CC02 wird dem gleichen Takt wie die CPU betrieben. Eine Wandlung dauert 11 Taktschritte, vgl. [4], S. 125. Bei einer Frequenz von 16MHz entspricht dies einer Zeit von $t_{A/D-Wandlung} = \frac{11}{f_{Osc}} \approx 0,7 * 10^{-6}s$ bei $f_{Osc} = 16000000$Hz

[6] Dies beinhaltet noch nicht die nötige Zeit, um die Messergebnisse auszuwerten und zu versenden.

rend weiche Echtzeit bedeutet, daß die Qualität der Ausgabe abnimmt, z.B. ungenauer wird.[7] Harte Echtzeit ist wünschenswert für unser Sensorsystem, damit ein Steuerprozess sofort auf Kollisionen und Kontakt reagieren kann. Der Nachweis der Echtzeitfähigkeit ist allerdings schwierig, da der exakte Zeitbedarf für die Messung von verschiedenen Faktoren abhängt. Unteranderem beeinflussen der Compiler, irreguläre Unterbrechungen durch die Kommunikation über den CAN-Bus sowie der spezifische Systemtakt die Dauer einer Messung. Wir begnügen uns daher mit weicher Echtzeit für das Sensorsystem, bei der eine Nichteinhaltung des Zeitfensters über eine spezielle Fehlernachricht signalisiert wird[8], um somit die mangelnde Güte der vorhergehenden Messergebnisse anzuzeigen. Experimentelle Versuche zeigen dabei, daß die 60ms in der Regel ausreichen, solange nicht ein Dauerstrom von Einstellungen über den CAN-Bus versendet wird. Dies liegt daran, daß die Unterbrechung der Messungen durch den CAN-Interrupt[9] nichtabschätzbare Mengen an CPU-Zeit verursacht, begründet in der unterschiedlichen Art der zu empfangenden Nachrichten und dem wechselnden Aufwand bei ihrer Verarbeitung.

Der Messablauf wird mit einem Zeitsignal synchronisiert. Ein Timer[10] im Mikrocontroller meldet, wenn eine definierte Zeitspanne[11] vergangen ist. In diesem Fall wird das Auslesen der Sensoren durchgeführt. Damit wird das Sensorsystem zu einem zeitgesteuerten System, auch wenn eine Ereignissteuerung bis zu einem gewissen Grad vorliegt, da der Empfang von Nachrichten Ereignisse darstellt, auf die reagiert werden muss.

Abbildung 5.1 gibt einen Überblick über das Hauptprogramm der Firmware. Die Hauptschleife wartet auf definierte Prozessanforderungssignale. Dies sind spezi-

[7] Eine verständliche Analyse von harter und weicher Echtzeit ist in [15], S.15ff zu finden.

[8] Konkret kann über ein entsprechendes Kommando konfiguriert werden, ob tatsächlich eine Überschreitung des Zeitfensters gemeldet wird. Für Arbeitsabläufe, deren längere Dauer vorhersehbar ist, z.B. Bereitstellen einer großen und langen Konfiguration, kann so explizit die Einhaltung abgeschaltet werden.

[9] vgl. Kapitel 5.3

[10] Ein Timer ist ein Zähler innerhalb des Mikrocontrollers, der ständig und unabhängig im gleichen Takt läuft. Wenn er einen bestimmten Zählwert erreicht, wird ein Signal ausgegeben, auf das das Programm dann reagieren kann.

[11] Die konkrete Implementierung zählt einzelne Millisekunden, vgl. timer.c.

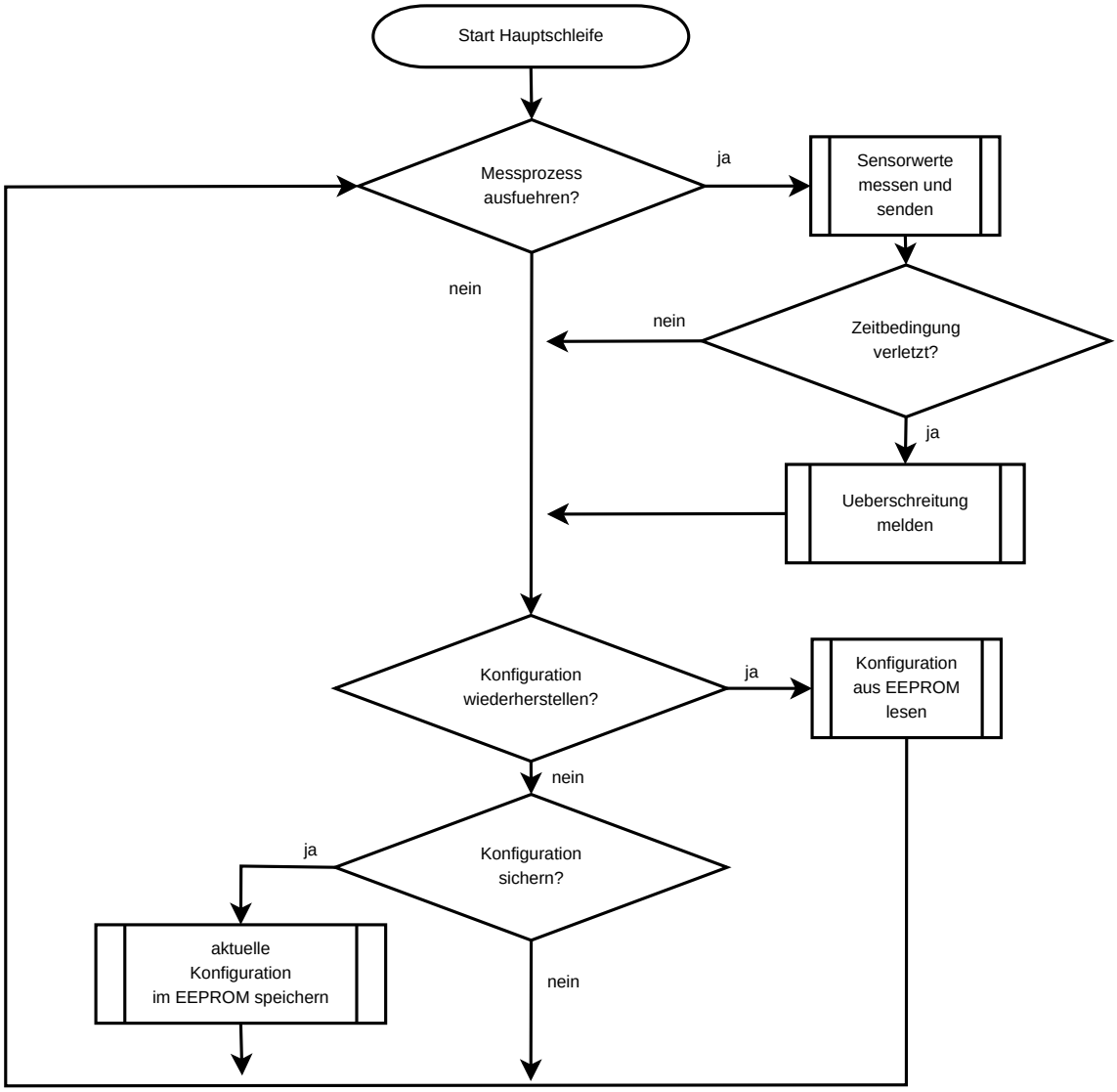


Abbildung 5.1: Flußdiagramm Hauptschleife

elle binäre Flags[12], die nur von den verschiedenen Interrupts gesetzt werden und nicht in anderen Programmteilen gesetzt werden. Das ist einerseits ein Timerinterrupt, der als Heartbeatfunktion agiert und alle 60ms ein Flag setzt, das die Messwerterfassung auszulösen hat, andererseits der CAN-Interrupt, der über weitere Flags signalisieren kann, daß eine neue Konfiguration gesichert oder wiederhergestellt werden soll. Auf diese Art kann auch einfach überprüft werden, ob die Zeitbedingung eingehalten wird. Ist am Ende eines Messchrittes das Flag für eine neue Messung schon wieder gesetzt, dann sind bis zu diesem Zeitpunkt mehr als die geplanten 60ms verstrichen. Die Bedingung harter Echtzeit wurde verletzt, und eine Verspätungsmeldung muß verschickt werden.
Innerhalb eines Messchrittes muß die Zeit sinnvoll verwendet werden. Da der GP2D120 Zeit benötigt, bis er eine Messung liefert, kann das Zeitintervall bis dahin für andere Messungen verwendet werden. Konkret wird der GP2D120 eingeschaltet. Dann werden die Bumpersensoren der Reihe nach abgefragt, während der GP2D120 auf sein Signal wartet und dann erst der Messwert vom Abstandssensor aufgenommen. Um die Bereitschaft des GP2D120 zu signalisieren, wird mit dem Einschalten des Abstandssensors ein weiterer Timer aktiviert, der über ein weiteres Flag die Bereitschaft des GP2D120 meldet.
Um bei den Lichtschranken Störlicht zu vermeiden, wird ein ähnliches Verfahren eingesetzt. Die Lichtschranken werden nicht der Reihe nach abgefragt, sondern abwechselnd mit den Fingersensoren abgefragt. Eine Lichtschranke wird ausgewertet, dann einer der Fingersensoren, und dann wieder eine Lichtschranke. Die Idee besteht darin, nur eine Lichtquelle für eine kurze Zeit zu aktivieren.
Der Ablauf der Messwertaufnahme ist in Abbildung 5.2 als Flußdiagramm dargestellt. Dabei ist festzuhalten, daß in der aktuellen Version[13] der Firmware eine Abfrage der Reflexlichtschranken noch nicht durchgeführt wird.
Sobald die Messwerte vorliegen, wird entschieden, ob diese mitzuteilen sind. Dabei wird entschieden, ob ein Report aller Werte aktiviert wurde. Ist dies der

[12] Faktisch wird über diese Flags eine einfache Art der Interprozeßkommunikation realisiert. In diesem Fall besteht die Botschaft darin, daß der Kontrollfluß innerhalb der Hauptschleife verändert werden soll.
[13] Stand Oktober 2007

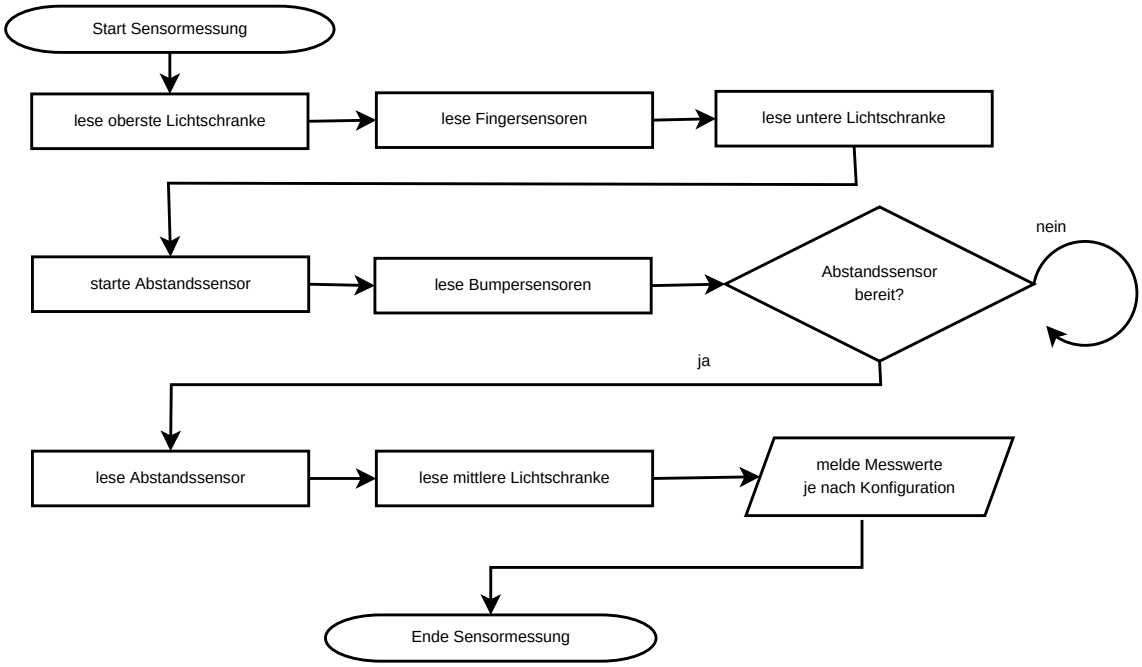


Abbildung 5.2: Flußdiagramm Messwertaufnahme

Fall, so werden sämtliche Sensorwerte ausgegeben. Falls das Reporting nicht aktiviert wurde, so werden Einzelwerte ausgegeben, sofern angefordert. Liegt dies für einen Sensor nicht vor, dann werden die Schwellwerte überprüft, sofern sie für den entsprechenden Sensor eingeschaltet wurden. Abbildung 5.3 verdeutlicht den Ablauf der Messwertausgabe mit einem Flußdiagramm.

5.3 Kommunikation über den CAN-Bus

Das Subsystem für den CAN-Bus stellt verschiedene Routinen für die Kommunikation bereit. Darunter fällt die Initialisierung der Kommunikation über den Bus, sowie Mehrzweckroutinen, die den Versand von Nachrichten über den CAN-Bus veranlassen. Eine Interrupt Service Routine regelt schließlich den Empfang und die Verarbeitung von Kommandos. Dabei wird vom physikalischen Medium abstrahiert und nur vom Empfang und Senden von Datenpaketen ausgegangen.
Interruptbasierter Empfang bietet den Vorteil, daß die Wartezeiten, zu denen der

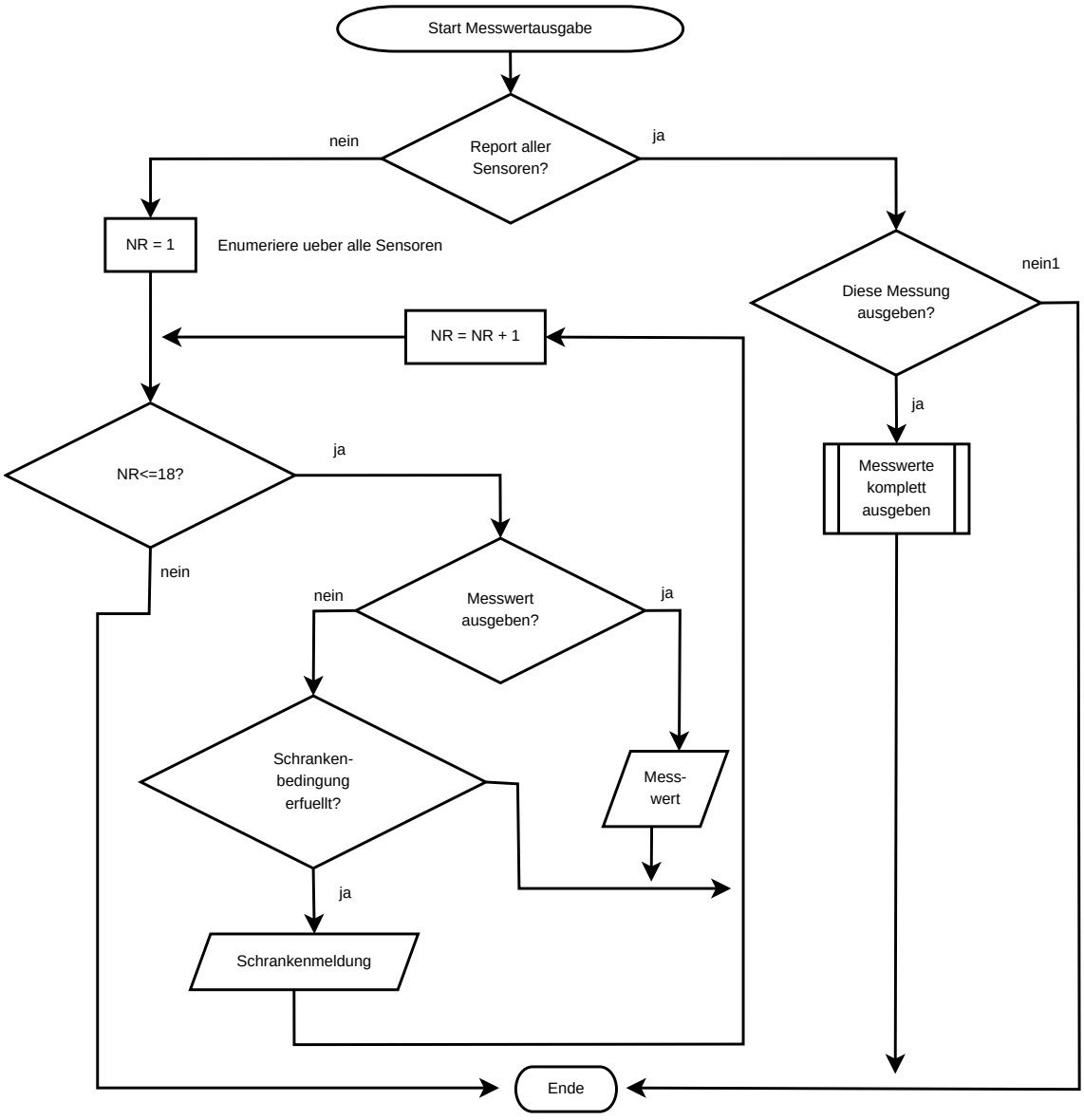


Abbildung 5.3: Flußdiagramm Messwertausgabe

Mikrocontroller auf die A/D-Wandlung der Messwerte wartet, zum Empfang von Nachrichten verwendet werden können. Ferner können so Einstellungen für das Monitoring von Sensoren übernommen und mit verhältnismäßig geringer Latenz[14] direkt übernommen werden, was bei einem Empfang im Pollingbetrieb nur mit einer Verzögerung möglich wäre.
Falls eine Nachricht empfangen wurde, die für die Sensorabfrage bestimmt ist, so wird anhand einer Fallunterscheidung die entsprechende Reaktionsroutine ausgewählt. Kompliziertere Vorgänge wie das Laden und Speichern einer Konfiguration im EEPROM[15] werden nicht sofort ausgeführt, sondern lediglich über Flags zur baldigen Ausführung vorgesehen, analog zur Auslösung des Messvorgangs durch den Timer. Einstellungen von Parametern und Resetkommandos werden sofort ausgeführt, da sie lediglich aus einigen kurzer Zuweisungen bestehen.
Unbekannte Nachrichten werden von der Firmware ignoriert[16]; bekannte Nachrichten werden in jedem Fall beantwortet. Die Antwort ist wiederum abhängig davon, ob die Nachricht alle zum darin enthaltenen Befehl nötigen Parameter beinhaltet oder nicht. Das Backend hat somit die volle Kontrolle, ob Kommandos, die zur Steuerung der Sensoren dienen, von der Firmware akzeptiert wurden.
Abbildung 5.4 erläutert den Ablauf des Nachrichtenempfangs mit einem Flußdiagramm. Das Prinzip des endlichen Automatens[17] kommt hier zur Anwendung, wobei die Erkennung einer Nachricht lediglich zwei Zustände kennt: „Nachricht wurde erkannt“ und „Nachricht wurde nicht erkannt“. Eine Nachricht gilt als erkannt, wenn das erste Zeichen der Nachricht einem gültigem Kommando entspricht und die zum Kommando zugehörige Anzahl von Datenbytes folgt. Aufgrund der mangelnden Komplexität wird auf eine formale Analyse des Automaten

[14] Die Erfassung der 18 Messwerte dauert ca. 52ms, wovon 45 bis 50ms auf die Abfrage des Sharp GP2D120 entfallen. Pro Sekunde sind so nur ca 16 bis 17 Messungen möglich. Es ist wünschenswert, wenn die Messwerte direkt nach ihrer Erfassung zur Verfügung stehen. Man bedenke, daß der Roboter zwischenzeitlich Bewegungen ausführt, die zum einen eine Veränderung der Werte bewirken, zum anderen eine ebensoschnelle Reaktion von Seite des Backends erforderlich machen.
[15] vgl. Kapitel 5.5
[16] Die Firmware nutzt die objektorientierte Betrachtungsweise auf dem CAN-Bus aus, vgl. Kapitel 2.2.1.
[17] Eine verständliche Einführung in die mathematische Automatentheorie findet sich in [8].

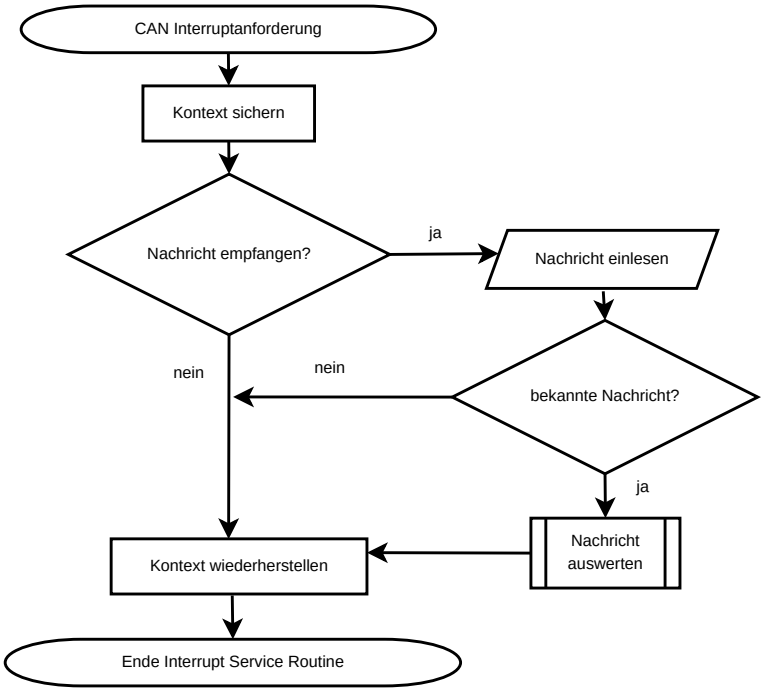


Abbildung 5.4: Flußdiagramm zum Nachrichtenempfang im CAN-Interrupt

verzichtet.
In [4] findet sich ab S. 71ff. ein ausführliches Kapitel über den technischen Ablauf des Empfangens und Sendens von Nachrichten über den CAN-Protokollcontroller auf der Registerebene des T89C51CC02. Die konkrete Umsetzung der Kommunikation über den CAN-Bus in der Sensorabfrage orientiert sich stark an [3]. Dort sind Beispielprogramme für den T89C51CC02 in C und 8051 Assembler zu finden, die die Ansteuerung des Protokollcontrollers im Sourcecode erläutern.

5.4 Das Kommunikationsprotokoll

5.4.1 Übersicht

Grundlage für die Kommunikation mit der Firmware bilden Datenpakete mit einer maximalen Länge von 8 Byte, die über den CAN-Bus verschickt werden.
Dabei wartet der Mikrocontroller grundsätzlich erst auf den Empfang eines für

ihn spezifischen Kommandos mit einem bekannten Identifier[18]. Bekannte Kommandos werden mit einer positiven oder negativen Antwort quittiert[19], während unbekannte Kommandos und Nachrichten ignoriert werden. Der positive Empfang und Ausführung bzw. Vormerkung der gewünschten Aktivität wird mit einem sogennanten Acknowledge beantwortet, während die Nichtausführung oder ein falsches Datenformat mit einem Negative Acknowledge (NAK) beantwortet wird. Prinzipiell besteht eine gültige Nachricht aus einem Befehlsbyte oder Antwortbyte, welches den Pakettyp signalisiert. Dieses Byte wird von einer Menge von Datenbytes gefolgt, deren genaue Anzahl vom jeweiligen Pakettyp abhängt und die auch Null sein kann.

Datenpakete mit Sensorwerten werden nur auf Benutzerwunsch erzeugt, entweder als gezielte Wertabfrage, als Antwort auf eine Monitoringschwelle oder als Dump sämtlicher Messwerte. Insgesamt stehen Kommandos für die folgenden Funktionalitäten zur Verfügung:

- Anforderung einzelner Messwerte
- Setzen und Löschen von Überwachungsschwellen für die Messwerte benannter Sensoren
- Ein- und Ausschalten des sogenannten Reportings aller Sensoren
- Speichern und Auslesen von Überwachungsschwellwerten als Konfiguration im Festwertspeicher
- Debugging und Statusabfragen von Subsystemen
- Resetfunktionalität, um sämtliche Meldeaktivitäten einzustellen

Tabelle 5.1 enthält eine Übersicht der möglichen Kommandos und ihre Bedeutung.

[18] Die Identifier sind im Quellcode nachlesbar, bzw. könnten beim Start des Sensorsystems aus dem Konfigurationseeprom ausgelesen werden. Letzte Funktionalität ist zur Zeit noch nicht implementiert, kann aber mit wenigen Codezeilen nachgerüstet werden.

[19] So kann jedes Programm, welches mit der Sensorabfrage kommuniziert, jederzeit verifizieren, ob Kommandos angenommen oder abgelehnt wurden.

Befehl	Beschreibung
COMMAND_RESET	Rücksetzen aller Einstellungen
COMMAND_READ	Sensor direkt auslesen
COMMAND_MONITOR	Schwellwertkontrolle für Sensor konfigurieren
COMMAND_STOPMONITOR	Schwellwertkontrolle abschalten
COMMAND_MONITORSTATUS	Status der Schwellwertüberwachung
COMMAND_RECALLMONITOR	Schwellwerte aus EEPROM auslesen
COMMAND_STOPALLMONITORS	alle Schwellwertkontrollen ausschalten
COMMAND_REPORT	Report aller Sensoren einschalten
COMMAND_STOPREPORT	Report aller Sensoren ausschalten
COMMAND_TIMECHECK_ENABLE	aktiviert Einhaltung der Zeitbedingungen
COMMAND_TIMECHECK_DISABLE	deaktiviert Zeitüberwachung
COMMAND_TIMECHECKSTATUS	Statusmeldung, ob Zeitbedingung einzuhalten ist
COMMAND_EEPROM_SAVEMONITOR	Schwellwerte im EEPROM sichern
COMMAND_EEPROM_CLEAR	Konfiguration im EEPROM löschen

Tabelle 5.1: Befehle für die Sensorabfrage

5.4.2 Die Kommandos im Überblick

Die Codierung der einzelnen Befehlsbytes kann der Datei command.h bzw S. 67 im Anhang entnommen werden. Im Folgenden werden wir die dort vereinbarten symbolischen Bezeichner verwenden. Die Komponenten bzw. Parameter eines Befehls sind byteweise anzugeben und, sofern nicht anders erläutert, binärcodiert. Lediglich der Schwellwert für den Überwachungsbefehl ist als 16Bit Wert anzugeben. Dabei ist die Motorolanotation zu verwenden, d.h. das höherwertige Byte wird zuerst angegeben.
Sofern nicht anders angegeben, wird jeder Befehl mit einem Acknowledge beantwortet.
Die Anzahl der zu sendenden Bytes variiert mit dem jeweiligen Befehl. Dabei sind die meisten Befehle kurz und nur ein Byte lang.

COMMAND_RESET

Der Einschaltzustand des Sensorsystems wird mit dem Resetkommando <COMMAND_RESET> wiederhergestellt. Keinerlei Schwellwerte werden überwacht

und das Reporting aller Sensoren ist ausgeschaltet.
Das Resetkommando impliziert die Befehle <COMMAND_STOPALLMONITORS> und <COMMAND_STOPREPORT> in der Ausführung!

COMMAND_READ <Sensor>

Mit dem Lesebefehl wird der aktuelle Messwert eines gegebenen Sensors nach Ende des nächsten Messzyklusses ausgegeben. Die Sensornummer ist binärcodiert anzugeben.
Um den Messwert des Sensors auf der rechten Fingerspitze[20] auszulesen, wird folgendes Kommando an die Sensorsteuerung gesendet, welches aus 2 Bytes besteht:
<COMMAND_READ> 0x10

COMMAND_MONITOR <Sensor> <Schwelle> <Richtung>

Mit diesem Kommando wird die Schwellwertüberwachung eines bestimmten Sensors eingeschaltet. Mit der Überwachungsrichtung wird eingestellt, ob eine Überschreitung oder Unterschreitung des Schwellwertes gemeldet werden soll.
Der Schwellwert ist als 16Bit-Wert anzugeben, dabei wird zunächst das höherwertige Byte angegeben.
Das Bestimmungsbyte für die Richtung wird binär interpretiert. Ein Wert von 0 bedeutet eine Meldung von einer Unterschreitung des Schwellwertes, ein Wert von 1 bedeutet die Meldung einer Überschreitung des Schwellwertes.
Ein gültiger Befehl zur Schwellwertüberwachung wird mit einem Acknowledge quittiert. Es wird allerdings keine Überprüfung der Schwellwerte auf Gültigkeit vorgenommen.
Insgesamt sind fünf Bytes inklusive des Befehls zu senden. Beispielsweise stellt die folgende Befehlssequenz den Sensor 0x10 auf eine Schwellunterschreitung von 0x100 ein, d.h. es werden Nachrichten erzeugt, sobald der Messwert des Sensors den Wert 0x100 unterschreitet:
<COMMAND_MONITOR> 0x10 0x1 0x00 0x00

[20] vgl. Tabelle mit der Zuordnung interner Nummer zu konkretem Sensor auf S. 66

COMMAND_STOPMONITOR <Sensor>

Mit diesem Kommando wird eine zuvor aktivierte Schwellwertüberwachung für einen bestimmten Sensor ausgeschaltet und mit einem Acknowledge beantwortet. Um beispielsweise eine zuvor aktivierte Schwellwertüberwachung für Sensor 2 abzuschalten, sendet man folgenden Befehl:
<COMMAND_STOPMONITOR> 0x02

COMMAND_MONITORSTATUS

Dieser kurze Befehl gibt den Status von Schwellwertüberwachungen aus. Er wird nicht mit einem Acknowledge, sondern mit der konkreten Statusmeldung beantwortet.

COMMAND_RECALLMONITOR

Mit diesem Befehl wird die im EEPROM abgelegte Konfiguration[21] von Schwellwertüberwachungen wiederhergestellt. Der Befehl überschreibt alle zuvor zur Laufzeit veränderten Schwellwertparameter. Das Abrufen der Konfiguration erfolgt nicht sofort, sondern erst nach Abschluß der laufenden Sensormessung. Quittiert wird wieder mit einem Acknowledge, allerdings bedeutet der Empfang des Acknowledge nicht, daß die Konfiguration schon aktiv ist, da das tatsächliche Auslesen der Konfiguration zum Absenden der Quittungsnachricht zeitversetzt erfolgen kann.
Falls im selben Messintervall ein Kommando zum Abspeichern der Konfiguration gesendet wurde, so wird dieses Kommando ignoriert. Lesen und Abspeichern der Konfiguration schließen sich aus, um unnötige Schreibzugriffe auf den Konfigurationsspeicher zu unterbinden.

COMMAND_STOPALLMONITORS

Dieser Befehl schaltet alle aktiven Schwellwertüberwachungen ab. Dabei wird eine im EEPROM vorhandene Konfiguration nicht gelöscht. Auch dieser Befehl

[21] vgl. Kapitel 5.5

wird einem Acknowledge quittiert.

COMMAND_REPORT <auszulassende Messintervalle>

Mit diesem Befehl wird ein Reporting aller Sensoren eingeschaltet. Dabei erfolgt die Ausgabe der Werte nach der angegebenen Anzahl von Messschritten. Es werden die Werte aller Sensoren ausgegeben; dabei werden einzelne Abfragen oder aktive Schwellwertabfragen ignoriert.
Um etwa jedes zweite Erfassungsintervall auszulassen und die Werte der Sensoren auszugeben, ist die folgende Sequenz aus 2 Bytes zu senden:
<COMMAND_REPORT> 0x01

COMMAND_STOPREPORT

Mit diesem Befehl wird ein zuvor aktiviertes Reporting aller Sensoren abgeschaltet. Danach sind wieder einzelne Lesekommandos und konfigurierte Schwellwerte aktiv.

COMMAND_TIMECHECK_ENABLE

Mit diesem Kommando wird die Einhaltung der Zeitbedingungen eingeschaltet. Wenn eine Messung und die Zusammenstellung aller nötigen Antworten länger dauert, als vorhergesehen, so wird eine Zeitüberschreitungsnachricht versendet. Im Einschaltzustand des Systems ist die Zeitüberwachung aktiviert.

COMMAND_TIMECHECK_DISABLE

Dieser Befehl deaktiviert die Benachrichtigung bei Zeitüberschreitung. Er ist sinnvoll, wenn viele Kommandos über den Bus gesendet werden sollen und die Qualität der Sensormessung nachrangig ist. Auch beim Speichern der Konfiguration kann er sinnvoll sein, da der EEPROM-Zugriff langsamer[22] erfolgt, als der Zugriff auf RAM.

[22] Es sind zusätzliche Instruktionen an den Mikrocontroller nötig, um den Zugriff zu realisieren. Dabei wird das EEPROM in den externen Datenspeicher gemappt. vgl. [4], S. 30 ff.

COMMAND_TIMECHECKSTATUS

Hiermit kann abgefragt werden, ob bei einer Zeitüberschreitung eine Benachrichtigung verschickt wird oder nicht. Statt eines Acknowledge wird mit einem speziellen Statuspaket geantwortet.

COMMAND_EEPROM_SAVEMONITOR

Mit diesem Kommando wird die aktuelle Einstellung der Schwellwertüberwachung für alle Sensoren als Konfiguration im EEPROM gespeichert. Das Ablegen der Konfiguration wird nicht zeitgleich mit dem Empfang durchgeführt, sondern erst nach Abschluss der laufenden Messung.

COMMAND_EEPROM_CLEAR

Der Inhalt des Festwertspeichers für die Konfiguration kann mit diesem Befehl gelöscht werden. Der Befehl ist noch nicht implementiert, kann aber einfach durch Löschung des Prüfsummenbytes im EEPROM realisiert werden.

5.4.3 Antwortpakete der Sensorabfrage

Je nach Betriebszustand und als Antwort auf entsprechende Kommandos werden Antworten generiert und über den CAN-Bus verschickt. Jedes Antwortpaket besteht aus einer definierten Anzahl von Bytes, bestimmte Charakteristika sind ASCII codiert, hauptsächlich um ihre Lesbarkeit beim Debuggen zu erhöhen.

Einschaltmeldung

Beim Einschalten des Moduls zur Sensorabfrage wird nach erfolgter Inititalisierung eine Einschaltmeldung aus 2 Bytes der Form ’S’ ’C’ <Versionsnummer> gesendet. Die Versionsnummer ist dabei ebenfalls ASCII codiert ab ’0’ aufwärts.

Acknowledge

Um den Empfang von Nachrichten zu bestätigen, werden sogenannte Acknowledge-Pakete verschickt. Sie sind 2 Bytes lang und beinhalten im ersten Byte den Befehlscode der bestätigten Nachricht und im zweiten Byte das ASCII-Kürzel für ACK, hexadezimal 0x06. Auf ein Kommando zum Abschalten aller Schwellwertüberwachungen wird also geantwortet:
<COMMAND_STOPALLMONITORS> 0x06

Negative Acknowledge

Wird eine Nachricht empfangen, deren Format falsch ist oder deren Befehl nicht ausgeführt werden kann, so wird sie mit einen Negative Acknowledge quittiert. Diese Nachricht funktioniert analog zum Acknowledge, dabei wird aber NAK, ASCII 0x15, gesendet.

Sensormesswerte

Sensormesswertpakete werden entweder durch den Report oder die Einzelabfrage versendet. Sie sind 4 Bytes lang, ein S signalisiert die Messwertnachricht, darauf folgt der jeweilige Sensor, dessen Ordnungsnummer binärcodiert wird und schließlich der eigentliche Meßwert als 16Bit-Zahl, wobei das höherwertige Byte zuerst gesendet wird. Ein Messwertpaket für die obere Lichtschranke sieht also wie folgt aus:
'S' 0x00 0x02 0x00

Schwellwertüberwachung

Wurde für einen Sensor eine Schwellwertüberwachung eingeschaltet, so werden Nachrichten ähnlich zu den Sensormesswerten generiert, allerdings wird das Paket mit einem M eingeleitet. Der Schwellwert selber und die Richtung wird nicht übertragen, die auswertende Software muß selber darüber Buch führen, welche Schwellen wie konfiguriert sind. Die Schwellwertmeldung wird nur erzeugt, wenn die konfigurierte Schwelle über- oder unterschritten wurde. Die Richtung der

Schwellüberschreitung wird dabei mitausgewertet. So wird zum Beispiel für eine Unterschreitungsschwelle von 0x200 für jeden Messwert, der kleiner als 0x200 ist, ein Schwellwertüberwachungspaket gesendet. Analoges gilt für eine Überschreitungschwelle, bei der Überwachungspakete gesendet werden, wenn der gemessene Wert größer als der konfigurierte Schwellwert ist.
Konkret könnte eine Schwellwertmeldung für den rechten Fingersensor wie folgt aussehen:
'M' 0x10 0x01 0xc3

Statusmeldung für Schwellwertüberwachung

Wird der Status der Schwellwertüberwachung abgefragt, so wird anstelle eines Acknowledge mit einem Statuspaket geantwortet. Es beginnt mit m und darauf folgt ein Statuswort mit 32 Bits, das höchstwertige Byte wird wieder zuerst genannt. Für jeden Sensor, für den eine Schwelle programmiert wurde, wird ein Bit im Statuswort gesetzt. Die genauen Schwellwertparameter, d.h. Richtung und Größenordnung der Schwelle, werden nicht angegeben.
Falls lediglich Sensor 0 überwacht wird, würde die Statusmeldung wie folgt aussehen:
'm' 0x00 0x00 0x00 0x01

Zeitüberschreitung

Wurde die Meldung einer Überschreitung des Zeitfensters aktiviert, so wird ein Zeitüberschreitungspaket generiert, wenn die Zeitbedingung tatsächlich verletzt wird. Das Paket besteht aus 2 Bytes, einem T und einem Negative Acknowledge in ASCII Codierung, d.h. das Paket sieht wie folgt aus 'T' 0x15.

Statusmeldung für Zeitüberschreitungsmeldung

Der Status der Zeitüberwachung wird mit einem Statuspaket anstelle von einem Acknowledge beantwortet. Das Paket besteht aus 2 Bytes, einem t und einem Statusbyte. Ist der Wert des Statusbytes 0, so wurde die Überwachung der Zeitre-

striktion abgeschaltet, ist er 1, so wurde die Überwachung aktiviert.
Wurde 't' 0x01 empfangen, so bedeutet dies eine aktivierte Zeitüberwachung.

5.5 Konfigurationsspeicherung

Die Konfiguration für die Schwellwertüberwachung benannter Sensoren kann dauerhaft in einem Festwertspeicher im Mikrocontroller gespeichert werden. Verwendet wird der sogenannte EEPROM Datenspeicher des T89C51CC02.[23] Für eine dauerhafte Speicherung von Parametern sprechen verschiedene Gründe:

- In einer definierten Testumgebung sind gleichförmige Arbeitsabläufe auszuführen. Es erscheint sinnvoll, die dafür notwendigen Parameter nur einmal einzustellen und dann dauerhaft abzulegen.
- Im Falle der Unterbrechung der Spannungsversorgung (z.B. wenn der Roboter ungeplant seinen eigenen Notausschalter betätigt) gehen sämtlichen Registerinhalte und Konfigurationsdaten im Mikrocontroller verloren. Anstelle nun erneut beim Einschalten einen langen Strom von Kommandos zur Wiederherstellung der alten Konfiguration zu schicken, kann der Mikrocontroller bei einem Neustart selbstätig die Werte für die Konfiguration aus einem dauerhaften Festwertspeicher auslesen.

Folgende Parameter werden als Konfiguration abgespeichert:

- Angaben, welche Sensoren überwacht werden und welche nicht
- Angaben, ob für den jeweiligen Sensor eine Unter- oder Überschreitung der Schwelle gemeldet werden soll
- die expliziten Schwellenwerte für jeden betroffenen Sensor

Um die Integrität der gespeicherten Konfiguration sicherzustellen, wird ein umkompliziertes Checksummenverfahren angewendet. Es soll eine einfache Überprüfung der Konfigurationsdaten ermöglichen. Hauptsächlich soll festgestellt werden, ob es einen Fehler beim Ablegen der Konfiguration gab. Der verwendete Algorithmus summiert die einzelnen Bytes der Konfigurationsdaten. Die Differenz zwischen 255 und dem niederwertigen Byte der Summe bildet dann die Check-

[23] vgl. [4], S. 30ff.

summe über die Konfigurationsdaten. Um einem Datensatz aus lauter Nullen eine sinnvolle Checksumme ungleich 0 zukommen zu lassen, werden die einzelnen Datenbytes nicht ab 0, sondern von einer Konstanten an aufaddiert. Abbildung 5.5 veranschaulicht das Verfahren mit einem Flußdiagramm.
Für den einfachen Algorithmus sprechen:

- schnelle Abarbeitung aufgrund seiner Kürze
- seine Übersichtlichkeit[24]
- die 8051 Architektur, deren Zugriffe immer mit Bytes erfolgen. Ein Algorithmus der mit größeren Datentypen arbeitet, würde unnötige Bibliotheksfunktionen einbinden und aufrufen, die die Abarbeitung verlangsamen würden.
- Eine Erkennung von sämtlichen Bitfehlern ist nicht nötig. Es soll nur entschieden werden, ob der Konfigurationssatz im Ganzen korrekt[25] ist.

Die abgelegte Konfiguration wird beim Einschalten oder Reset des Systems ausgelesen und über ihre Checksumme validiert. Es wird die Checksumme über den Inhalt des EEPROMs gebildet. Falls die Checksumme mit den Daten übereinstimmt[26], so werden die Konfigurationsdaten aus dem Speicher ausgelesen und für die betroffenen Sensoren eingestellt. Die Schwellwerte sind dann sofort ab der ersten Aufnahme von Messwerten wieder aktiv.
Beim Ablegen der Konfiguration im Festwertspeicher wird zunächst die Checksumme über die Konfigurationsdaten gebildet und dann erst die Daten und dann der berechnete Checksummenwert im EEPROM selbst abgelegt.
Über entsprechende Kommandos[27] kann die aktuelle Konfiguration des Sensormonitorings abgelegt, wieder eingelesen und auch dauerhaft gelöscht werden.
Das Speichern und Auslesen der Konfiguration im Festwertspeicher beeinflußt ggfs. das Zeitgefüge laufender Messungen. Daher ist es ratsam, diese Schritte nur dann auszuführen, wenn keine Gefahr durch Bewegung des Roboterarmes

[24] In C besteht der Algorithmus aus 2 Zuweisungen und einer kurzen Summationsschleife. vgl. Funktion Command_ReadDefaultConfiguration() im Modul command.c
[25] Korrekt hier im Sinne der Checksumme über die Konfigurationsdaten.
[26] Das bedeutet, daß die Daten rechnerisch die gleiche Checksumme ergeben, wie die zusammen mit den Daten im EEPROM gespeicherte Checksumme.
[27] vgl. Kapitel 5.4

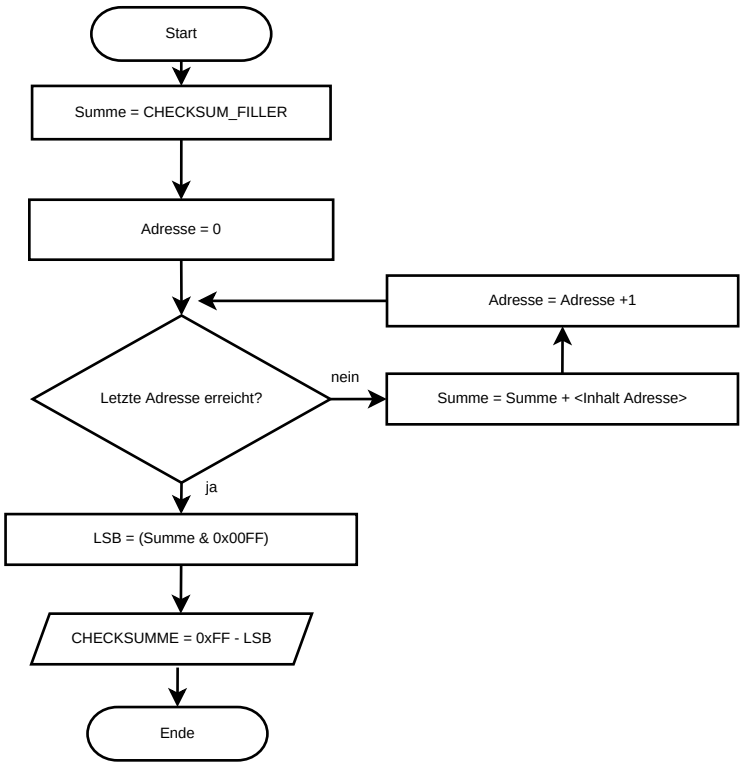


Abbildung 5.5: Flußdiagramm zur Checksummenbildung

besteht und wenn ein möglicher Verlust aktueller Sensorwerte keine Gefahr für den Roboter, den Bediener und die Umgebung bedeutet.

Kapitel 6

Ausblick

Die gezeigten Prinzipien eröffnen neue Möglichkeiten. Eine massive Erhöhung der Anzahl der verwendeten Sensoren kann die Auflösung der Objekterkennung entscheidend verbessern. Beim Vergleich mit der menschlichen Hand fällt auf, daß die Natur eine so feine Auflösung ihrer Rezeptoren erreicht, bei der die Positionsmeldung einer Berührung ohne erkennbare Rasterung wahrgenommen wird. Im Rückschluß auf den Roboter bedeutet also eine Erhöhung der Zahl der Sensoren eine feinere Auflösung bis hin zu einer stufenlosen Wahrnehmung der Umgebung. Eine Erhöhung der zeitlichen Auflösung, d.h. eine schnellere Abfrage der Sensoren, ermöglicht feinere Bewegungsabläufe und damit eine größere Ähnlichkeit mit natürlichen Vorbildern. Eine Erhöhung der Taktfrequenz des verwendeten Mikrocontrollers bietet hier eine einfache Variante, allerdings werden hierdurch keine Probleme beseitigt, die etwa durch die Rahmenparameter gegebener Sensoren entstehen, wie sie etwa der Abstandssensor von Sharp so deutlich zeigt.
Möglich wäre auch eine Erhöhung des Durchsatzes an Sensormessungen durch den Einsatz schnellerer A/D-Wandler. Verbesserungspotentiale sind daher allein durch eine Verbesserung der eingesetzten Technologie gegeben.
Die Datensicherheit kann durch eine Erweiterung des Kommunikations um Prüfsummen erhöht werden, sowie durch eine Bewertung des Messprotokolls. Ein Protokoll über die letzten Sensormessungen wird automatisch im Datenspeicher des Mikrocontrollers geführt. Dies ermöglicht prinzipiell die Anwendung von di-

gitalen Filteralgorithmen sowie eine ausführlichere Auswertung. Von einer Implementierung dieser wurde allerdings aus Zeitgründen abgesehen.
Die Abfrage der Sensoren durch die Firmware kann durch den Einsatz eines echtzeitfähigen Betriebssystems verbessert werden. Zeitbedingungen können so besser eingehalten werden, allerdings zum Preis eines erhöhten Hardwareaufwandes. Der Einsatz eines leistungsfähigeren Mikrocontrollersystems oder die Übertragung auf eine reine Hardwarelösung mit programmierbaren Logikschaltkreisen ist natürlich möglich und der gezeigte Lösungsweg stellt keine endgültige Lösungsmöglichkeit dar.

Kapitel 7

Anhang

Zuordnung Sensornummer zu tatsächlichem Sensor

Intern und im Rahmen der Abfrage der Sensoren werden Nummern verwendet. Die folgenden Tabelle gibt den jeweiligen Sensor zu seiner Ordnungsnummer an. Die Ordnungsnummern werden binärcodiert in den Steuerkommandos und in der Firmware selbst verwendet.

Ordnungsnummer	Sensor
0	obere Lichtschranke im Finger
1	mittlere Lichtschranke im Finger
2	untere Lichtschranke im Finger
3	hintere Reflexlichtschranke
4	vordere Reflexlichtschranke
5	Abstandssensor GP2D120
6. . . 13	Bumper 1. . . 8
14	Drucksensor Wippe oben
15	Drucksensor Wippe unten
16	Drucksensor rechte Fingerspitze
17	Drucksensor linke Fingerspitze

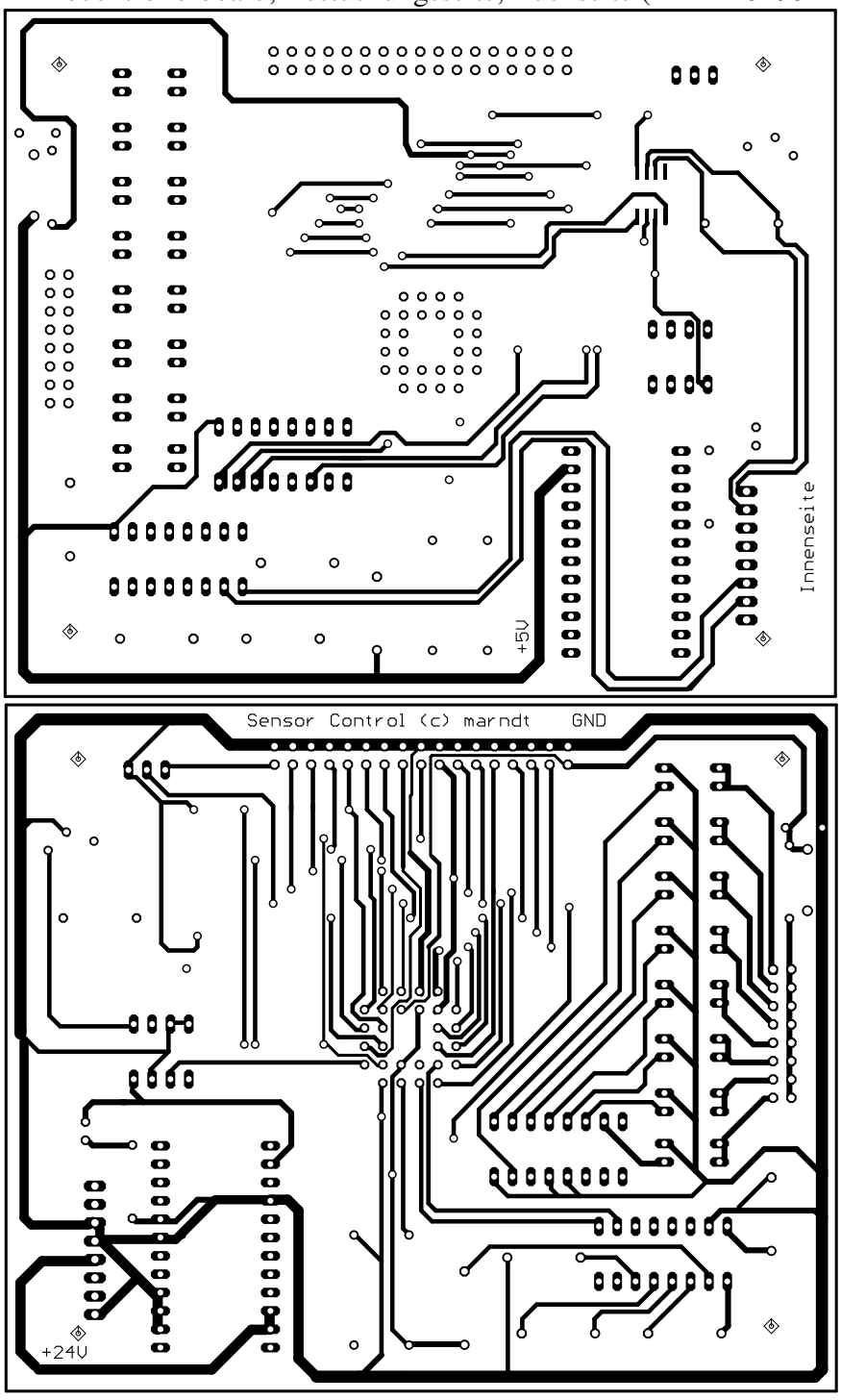
Codierungstabelle Steuerkommandos

Die numerische Kodierung der einzelnen Kommandos für die Steuerung kann der folgenden Tabelle entnommen werden. Definiert werden sie als Konstanten in der Quelldatei command.h.

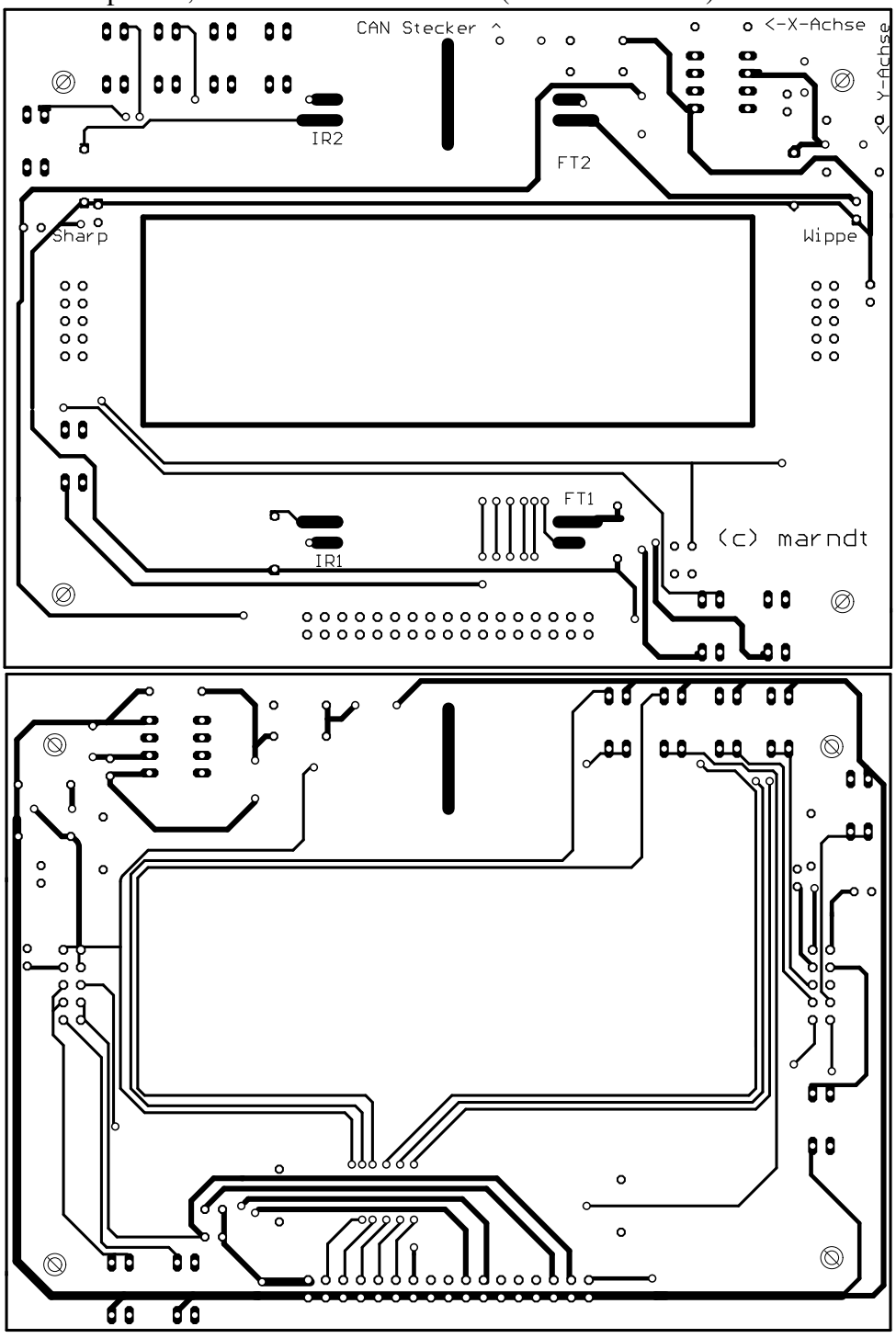
symbolischer Name	Code (hexadezimal)
COMMAND_READ	0x00
COMMAND_MONITOR	0x10
COMMAND_STOPMONITOR	0x11
COMMAND_MONITORSTATUS	0x12
COMMAND_RECALLMONITOR	0x13
COMMAND_STOPALLMONITORS	0x14
COMMAND_REPORT	0x20
COMMAND_STOPREPORT	0x21
COMMAND_TIMECHECK_ENABLE	0xA0
COMMAND_TIMECHECK_DISABLE	0xA1
COMMAND_TIMECHECKSTATUS	0xA2
COMMAND_EEPROM_SAVEMONITOR	0xE0
COMMAND_EEPROM_CLEAR	0xE1
COMMAND_RESET	0xFF

Leiterplattenentwürfe

Mikrocontrollerboard, Bestückungsseite, Rückseite (BxH 115x95mm):



Interfaceplatine, Oberseite und Unterseite (BxH 128x95mm):



Bei Bedarf exakte Belichtungsvorlagen mit Cadsoft Eagle neu erstellen.

Bauteilstücklisten

Folgende Bauteile sind auf der Mikrocontrollerplatine verbaut:

Bauteil	Anzahl	Bemerkung
Atmel T89C51CC02	1	Mikrocontroller im PLCC28-Gehäuse
TEL 5-2411	1	DC/DC Spannungskonverter 24V auf 5V
MAX700	1	Resetsupervisor
Quarzoszillator 16MHz	1	Taktversorgung Mikrocontroller
74HCT237	1	3-8 Decoder für Bumperanwahl
PCA82C51	1	CAN-Transciever 24V im SOIC8 Gehäuse
MAX232CPE	1	Pegelkonverter für RS232 Schnittstelle
IRFD 014	8	N-MOSFETs zur Bumperaktivierung
Pfostenwanne 34polig	1	Busverbinder zur Sensorinterfaceplatine
Pfostenwanne 16polig	1	Busverbinder Bumper
Taster	1	Resettaster
Schiebeschalter	1	Anwahl Bootloadermodus (2,5mm Pinabstand)
Steckleiste gewinkelt, 8polig	1	Anschluß Versorgungsspannung, CAN, RS232
LED 5mm rot	1	Anzeige Spannungsversorgung
Widerstand 470Ω	1	Vorwiderstand LED
Widerstand 10 kΩ	1	Vorwiderstand Spannungsteiler Bumper
Widerstand 120Ω	1	Terminatorwiderstand CAN-Bus
Kondensator 100nF	3	Trennkondensatoren
Elektrolytkondensator 1μF	5	Spannungspumpe RS232
Spule	1	Versorgungsspannung Analogteil

Auf der Interfaceplatine sind folgende Bauteile verbaut:

Bauteil	Anzahl	Bemerkung
TLC272CP	1	Operationsverstärker für Lichtschranken
IRFD 014	8	Lowsideschalter für Sensoren (N-MOSFET)
CQY37N	2	Infrarot-LEDs für Reflexlichtschranken
BPW17N	2	Fototransistoren Reflexlichtschranken
Pfostenwanne 10polig	2	Anschluß Fingersensoren
Pfostenwanne 34polig	1	Busverbinder zur Mikrocontrollerplatine
Widerstand 470Ω	7	Vorwiderstände LichtschrankenCQY37N
Widerstand 100Ω	1	Schutzwiderstand Sharp GP2D120
Widerstand 100 kΩ	6	Beschaltung Lichtschrankenverstärker
Widerstand 10 kΩ	2	Vorwiderstände FSR Finger
Kondensator 100nF	1	Verstärker Lichtschranken
Kondensator 3,3nF	2	Verstärker Lichtschranken
Kondensator 1,5nF	1	Verstärker Lichtschranken

Signale und Pinbelegung des Mikrocontrollers

Die Portsignale des T89C51CC02 werden wie folgt von der Schaltung und der Firmware verwendet, die Spalte Signal gibt dabei die abgekürzte Signalbezeichnung[1] an:

Portpin	Signal	Verwendungszweck
P1.0	AN0	analoger Eingang für Lichtschranken und den Sharpsensor
P1.1	AN1	Analogeingang für Zeile 1 aus Matrix Fingersensoren
P1.2	AN2	Analogeingang für Zeile 2 aus Matrix Fingersensoren
P1.3	AN3	Eingang für analoges Signal der Bumper
P1.4	BEN	Bumperaktivierung
P1.5	B2	oberstes Bit für Bumperauswahl
P1.6, P1.7	X0, X1	Spaltenauswahl für Fingersensormatrix
P2.0, P2.1	B0, B1	untere Bits für Bumperaktivierung
P3.2	L0	Aktivierungssignal Sharp GP2D120
P3.3,...,P3.7	L1,...,L5	Aktivierungssignale für Lichtschranken
P3.1		TxD für RS232
P3.0		RxD für RS232
P4.0		TxDC für CAN-Bus
P4.1		RxDC für CAN-Bus
VAREF, VAVCC		mit +5V beschaltet
VAGND		GND

Sofern nicht anders angegeben, wird von einer Codierung in positiver Logik ausgegangen, d.h. +5V bedeutet logisch 1.

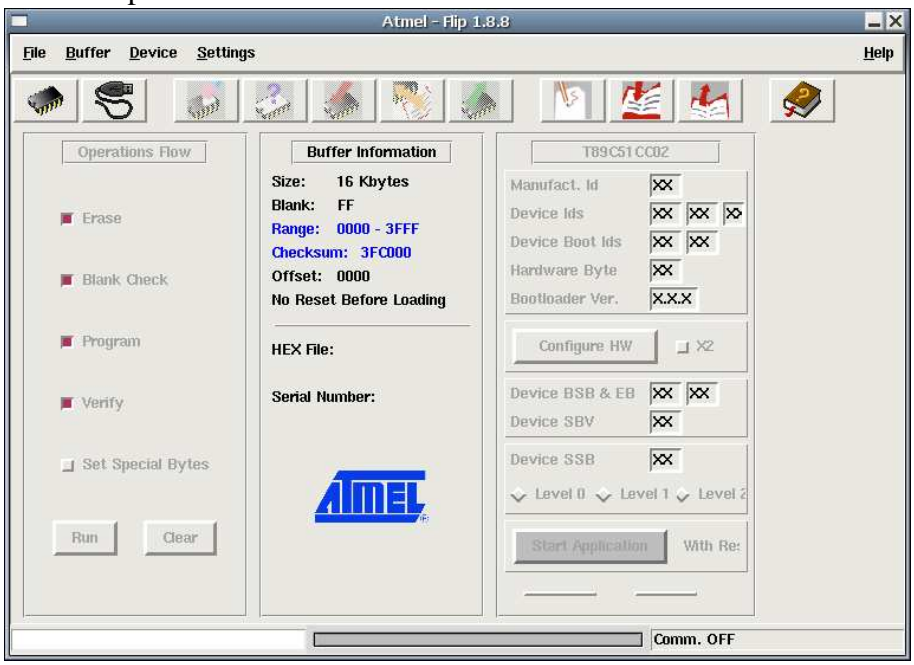
[1] Eine Referenz auf die Portnummern würden auf die Dauer zu unübersichtlich werden.

Das Flashen der Firmware

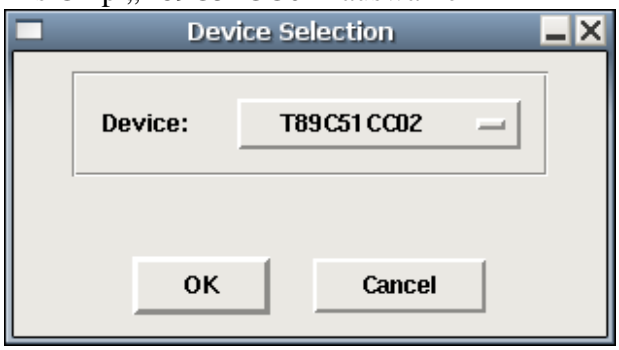
Um den Mikrocontroller mit einer neuen Firmware zu versehen, ist wie folgt vorzugehen. Natürlich muss das System mit Spannung versorgt werden, zur Kontrolle die LED auf der Mikrocontrollerbaugruppe beachten.

(a) neue Firmware übersetzen und ggfs. in das Intel HEX Format überführen

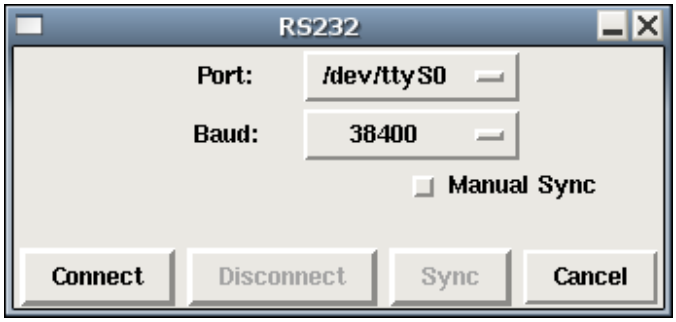
(b) Atmel Flip Software starten



(c) Als Chip „T89C51CC02“ auswählen



(d) Kommunikation über RS232 auswählen, verwendete Schnittstelle und Baudrate auswählen

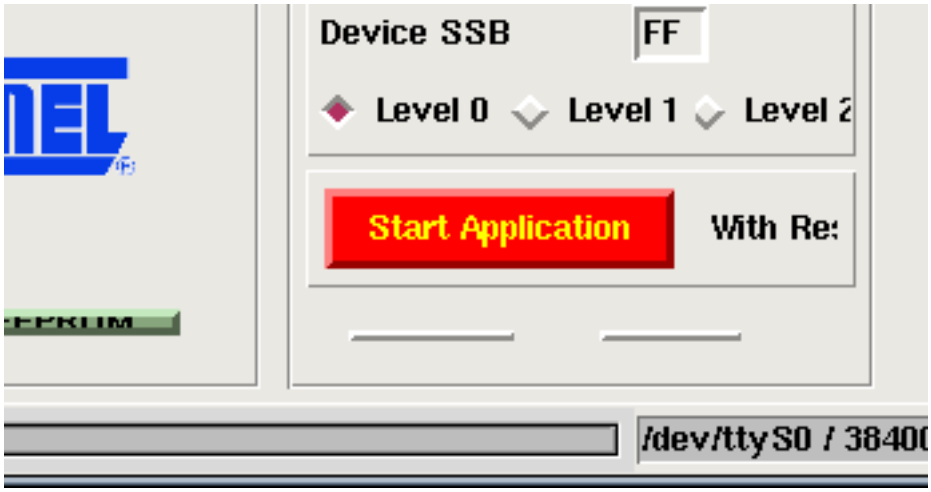


(e) Mikrocontrollerbaugruppe über ein passendes Kabel mit dem PC verbinden

(f) Am Mikrocontroller den Bootloader aktivieren, in dem der Schiebeschalter in die Position „Bootloader“ geschoben wird

(g) Reset des Mikrocontroller über den Resettaster

(h) Im Fenster für die RS232 Verbindung „Connect“ auswählen, bei erfolgreicher Kommunikation mit der Baugruppe wechselt der Knopf „Start Application“ im Hauptfenster seine Farbe



(i) Zu Testzwecken kann die Funktion „Read“ ausgewählt werden. Der aktuelle Inhalt des Programmspeichers wird so ausgelesen.

(j) Im Menü über die die Funktion „Load Hex“ die neue Firmware laden

(k) Auf den Knopf „Run“ klicken, dabei sind die Optionen wie im Bild einzuschalten

(l) Nach erfolgtem Programmiervorgang kann die neue Firmware in Betrieb genommen werden, in dem der Schiebeschalter in die Position „Applikation“ geschoben wird und der Resetknopf einmal betätigt wird.

Inhalt der CDROM

Die CDROM enthält folgende Komponenten:

- HTML Navigation zum Zugriff auf die CDROM
- eine Kopie des vorliegenden Textes der Diplomarbeit in der Datei *Diplomarbeit.pdf*
- im Literaturverzeichnis angegebene Dokumente und elektronische Quellen im Unterverzeichnis *Literatur/*
- Quellcode der Firmware im Unterverzeichnis *Source/*
- mit Doxygen erzeugte Dokumentation des Quellcodes im Unterverzeichnis *Source/doxygen/html/*
- Leiterplattenentwürfe und Schaltpläne zur Verwendung mit Cadsoft Eagle im Unterverzeichnis *Eagle/*
- eine Fotogallerie zur visuellen Dokumentation des Projektes

Unter MS Windows startet die CDROM automatisch eine Navigation über den Webbrowser. Benutzer alternativer Betriebssysteme öffnen die Datei *index.html* mit einem geeigneten Webbrowser.

Abkürzungsverzeichnis

A/D-Wandler Analog-Digital-Wandler

ASCII American Standard Code for Information Interchange

CAN Controller Area Network

CRC cyclic redundancy check

EEPROM Electrically Erasable Programmable Read Only Memory

FET Feldeffekttransistor

FPGA Field Programmable Gate Array

FSR Force Sensing Resistor (kraftabhängiger Widerstand)

ISR Interrupt Service Routine

LED light emitting diode (Leuchtdiode)

PSD Position Sensitive Device

RAM Random Access Memory

ROM Read Only Memory

TTL Transistor Transistor Logic

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] C. Asam. Diplomarbeit: Inbetriebnahme eines Power-Cube-Roboterarmes; Entwicklung eines C167-Monitors und des PC-Gegenstücks zur Steuerung der Armgrundfunktionen. Master's thesis, TU Clausthal, 2003.

[2] Atmel Corporation. *T89C51CC02 UART Bootloader*, 2003.

[3] Atmel Corporation. *T89C51CC02 CAN Program Examples*, 2004.

[4] Atmel Corporation. *T89C51CC02 - Enhanced 8-Bit MCU with CAN Controller and Flash*, 2005.

[5] H. Engels. *CAN-Bus: CAN-Bus-Technik einfach, anschaulich und praxisnah vorgestellt*. Franzis Verlag, 2002.

[6] esd GmbH. *CAN - Ein serielles Bussystem nicht nur für Kraftfahrzeuge*, 2002.

[7] K. Etschberger. *Controller-Area-Network: Grundlagen, Protokolle, Bausteine, Anwendungen*. Hanser Verlag, 2002.

[8] J. E. Hopcroft and J. D. Ullman. *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Oldenbourg Verlag, 4 edition, 2000.

[9] International Electronics Engineering. *FSR-Sensoren: Daten, Eigenschaften und Hinweise zur Handhabung*, 1998.

[10] Maxim Integrated Products. *MAX700/701/702 Power-Supply Monitor with Reset*, 2005.

[11] Philips Semiconductors. *75HC/HCT237 3-to-8 line decoder/demultiplexer with address latches*, 1990.

[12] Philips Semiconductors. *80C51 family programmer's guide and instruction set*, 1997.

[13] Philips Semiconductors. *PCA82C251 CAN transciever for 24V systems*, 2000.

[14] Sharp Corporation. *GP2D120 optoelectronic device*, 2006.

[15] C. Siemers. *Embedded Systems Engineering*. Skript, 2005.

[16] A. S. Tanenbaum. *Computernetzwerke*. Pearson-Studium, 2007.

[17] Vishay Semiconductors. *BPW17N*, 2005.

[18] Vishay Semiconductors. *CQY37N*, 2005.

Erklärung

Hiermit versichere ich diese Arbeit selbständig verfaßt und keine anderen als die genannten Quellen und Hilfsmittel benutzt zu haben.

Matthias Arndt