Dietmar Cordes
Gunther Lemm

Entwicklung einer mikroprozessorbasierten Sensor- / Aktorerweiterung für das LEGO-Mindstorm System

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Technische Informatik
am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Ing Gunter Klemke
Zweitgutachter : Prof. Dr. Kai von Luck

Abgegeben am 17. Oktober 2003

Inhaltsverzeichnis

Vorwort

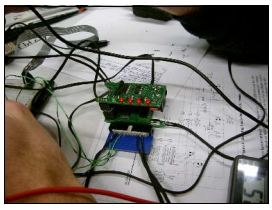
Wer kennt sie nicht, die bunten Kunststoffsteine aus dem Hause LEGO? Mittlerweile sind diverse Generationen von Kinder damit aufgewachsen und haben alle erdenklichen Formen von Häusern und Fahrzeugen aus diesen LEGO-Steinen erbaut. Der Grund für seine Popularität ist der weitgehend von Mode-Erscheinungen unberührte, universelle Charakter dieses Spielzeugs. Besonderes Merkmal der LEGO-Steine ist ihre ausgefeilte Mechanik, die der Kreativität kaum Grenzen setzt.

Die fortschreitende Computerisierung in allen Lebensbereichen machte auch nicht vor den Produkten der Firma LEGO halt. Um seinem Spielzeug ein „intelligentes“ Verhalten beibringen zu können, entwickelte der dänische Hersteller Ende der neunziger Jahre einen programmierbaren LEGO-Baustein mit dem Namen RCX. Die Produktreihe wurde auf „LEGO Mindstorms“ getauft.

Lehrer und Professoren erkannten bald das Potential dieser Entwicklung und begannen das Mindstorms-System für didaktische Zwecke zu nutzen. Leider ist der darin enthaltene RCX-Baustein nicht flexibel genug ausgelegt, um auch den didaktischen Bedürfnissen im Umfeld einer Hochschule gerecht zu werden. Das liegt vor allem darin begründet, daß er nur sehr wenige Schnittstellen zur Kommunikation mit seiner Umgebung besitzt. Für den Bau mobiler Roboter ist das Mindstorms-System damit nur bedingt einsetzbar. Das ist schade, denn das Konzept, Roboter komplett auf der Basis von LEGO-Steinen zu bauen, ist durchaus bestechend.

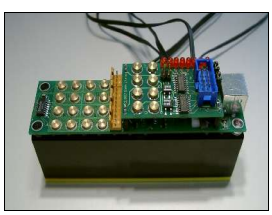
Es gibt unzählige Ansätze, das Problem der schlechten Schittstellenausstattung zu lösen. Allerdings beziehen sich diese Lösungen oftmals nur auf die Verwendung spezieller Sensorik und nicht auf eine universelle Erweiterung der Schnittstellen.

Die Studienarbeit „Konzeption von I/O-Erweiterungen für Lego Mindstorms“ [1] stellt eine Reihe theoretischer Vorüberlegungen für den Bau universeller RCX-Erweiterungen an. Ziel der vorliegenden Diplomarbeit ist die Entwicklung einer praxistauglichen I/O-Erweiterung für das Mindstorms-System auf der Basis der dort gewonnenen Erkenntnisse. Dabei findet die praktische Umsetzung in drei Schritten statt:



Der erste Schritt ist eine Machbarkeitsstudie, bei der die prinzipielle Funktion der Schaltungsteile und die Realisierbarkeit einer Kommunikation zwischen RCX und dem Erweiterungsbaustein geprüft wird.

Im zweiten Schritt erfolgt die Verbesserung der Machbarkeitsstudie hinsichtlich der Entwicklung eines praxistauglichen Prototyps. Dabei soll sichergestellt werden, daß das Gerät möglichst unabhängig von Betriebssystemvorgaben und anderen speziellen Rahmenbedingungen arbeitet.



Im dritten Schritt erfolgt schließlich eine Optimierung des Prototyps bezüglich der Ausstattung und des Designs auf die Anforderungen des RoboLabs der HAW-Hamburg.

1 Analyse & Konzeption

Vor einigen Jahren entwickelte die Firma LEGO einen programmierbaren LEGO-Stein. Dieser als RCX bezeichnete Baustein soll eine autonome Steuerung von sonst nur manuell bedienbaren LEGO-Modellen ermöglichen. Die Programmierung findet dabei über einen PC statt. [2]

Die Zielgruppe für dieses „Mindstorms“ genannte System sind Schulkinder. Ihnen wäre sicher nicht zuzumuten, eine Programmiersprache erlernen zu müssen, um mit dem RCX spielen zu können. Aus diesem Grund existiert eine spezielle Entwicklungsumgebung, die die Programmierung durch ein grafisches Anordnen logischer Funktionsblöcke erlaubt. Abbildung 1.1 zeigt die Oberfläche dieses Programmiersystems, mit dem sich zwar sehr schnell Ergebnisse erzielen lassen, das aber nicht zur Programmierung komplexerer Abläufe taugt.

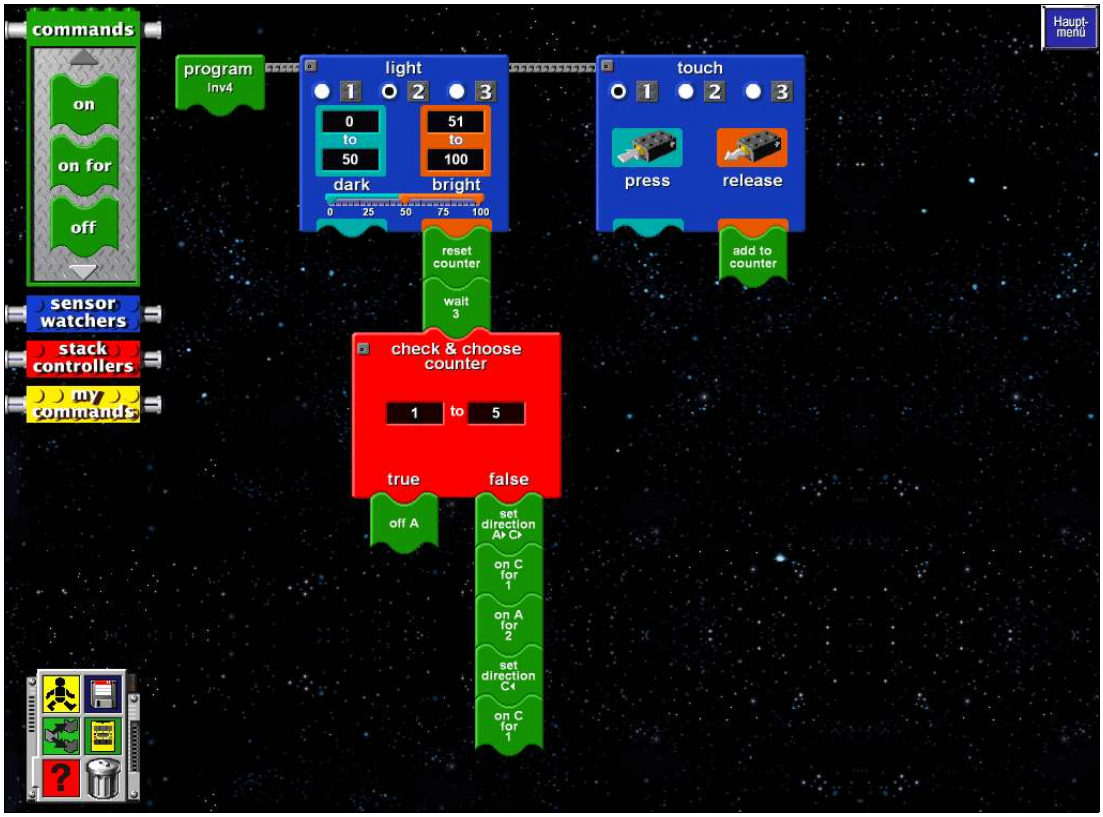


Abbildung 1.1: Screenshot der LEGO-Entwicklungsumgebung

Die tatsächliche Käufergemeinde der Mindstorms-Systeme entspricht in vielen Fällen aber nicht der genannten Zielgruppe. Besonders in Schulen und Universitäten erfreuen sich die RIS[1] genannten Komplettsysteme großer Beliebtheit. Das liegt vor allem daran, daß sich damit in kürzester Zeit kleine Roboter zusammenbauen lassen, die sich gut für didaktische Zwecke eignen. Ein typisches Lehrziel ist dabei die Programmierung autonom agierender mobiler Roboter.

Bisher in diesem Umfeld eingesetzte Systeme basieren üblicherweise auf gehäuse-

1 RIS steht für Robotics Invention System und bezeichnet einen speziell für kleinere Roboterbauprojekte zusammengestellten Baukasten. Er enthält neben einem RCX und diversen LEGO-Bausteinen alle zur Programmierung nötigen Komponenten.

losen Prozessorplatinen, die in oftmals abenteuerlicher Weise mit Sensoren und Aktoren zu einer fahrbaren Plattform kombiniert werden.

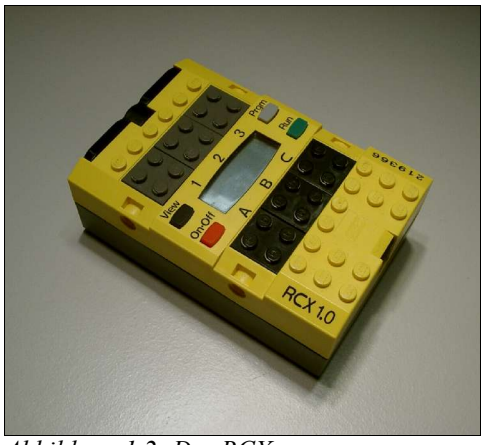


Abbildung 1.2: Der RCX

Der Ansatz der Firma LEGO, Mikroprozessortechnik in LEGO-Steine zu verpacken, ist dagegen deutlich eleganter. Beim Bau der Plattform kann somit praktisch die gesamte mechanische Infrastruktur der bisher üblichen LEGO-Baukästen genutzt werden. Es entfällt also der meist zeitraubende Eigenbau mechanischer Elemente.

Zusätzlich zum RCX enthält ein RIS eine Grundausstattung an Sensoren und Motoren, die aber nur für kleinere Bauvorhaben bzw. einfache Aufgabenstellungen taugt.

Auch im RoboLab[2] der HAW-Hamburg wurden vor einigen Semestern LEGO-Mindstorms angeschafft. Die ursprüngliche Überlegung war dabei, ein handliches System zu Repräsentationszwecken zu haben, mit dem sich außer Haus für das RoboLab werben läßt.

In den folgenden Semestern wurde erprobt, wie weit ein RCX-basierter Roboter für Aufgaben geeignet ist, die sonst mit einer 6.270 genannten Plattform bewältigt wurden.[3] Letzteres ist die Bezeichnung für ein Prozessorboard, das vom MIT für den Betrieb in mobilen Robotern enwickelt wurde. Beide Systeme beinhalten einen 8-Bit-Prozessor[4] ähnlicher Leistungsfähigkeit. Sie unterscheiden sich aber deutlich in der Struktur ihrer Schnittstellen.

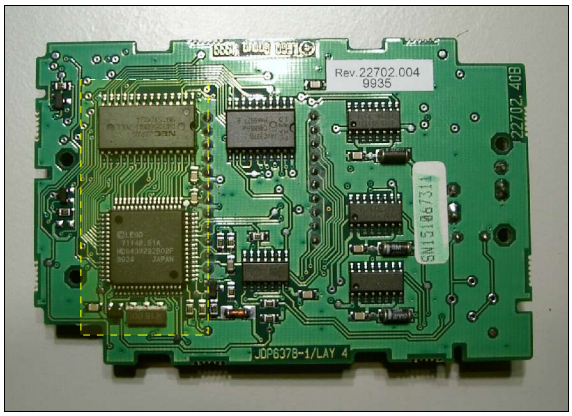


Abbildung 1.3: Der H8-Prozessor des RCX mit zusätzlichen 32kB SRAM als Programmspeicher

1.1 Systemvergleich

Die Schittstellen-Ausstattung des RCX gestaltet sich äußerst übersichtlich (siehe Abbildung 1.2). Er stellt drei Sensoreingänge (bezeichnet mit 1 bis 3) und drei Motorausgänge (bezeichnet mit A bis C) zur Verfügung. Allerdings sind diese Anschlüsse verpolungs- sowie kurzschlußsicher und bieten je nach Bedarf unterschiedliche Betriebsmodi. Eine Erweiterung der

2 Es ist nicht ganz klar, ob dies die offizielle Bezeichnung für das Roboterlabor der HAW-Hamburg ist. Da sich der Name „RoboLab“ aber bei Studenten und Mitarbeitern inzwischen eingebürgert hat, findet er auch in dieser Arbeit Verwendung. Informationen über die Aktivitäten des RoboLab finden sich in [3].

3 Das „6.270er Board“ entstand Anfang der 1990er Jahre am Massachusetts Institute of Technology (MIT). Eine Abwandlung des 6.270 Boards ist das „Handy Board“. In [4] und [5] sind die Geschichte und der Aufbau der Hard- und Software beschrieben.

4 Die 6.270-Boards basieren auf einem Motorola 68HC11, der RCX dagegen auf einem Hitachi H8-Prozessor. Allerdings wird der H8 nicht wie der 68HC11 mit 8 MHz, sondern mit 16 MHz betrieben. In beiden Fällen werden die Benutzerprogramme in ein zusätzliches 32 kB großes SRAM geladen. [5] [6]

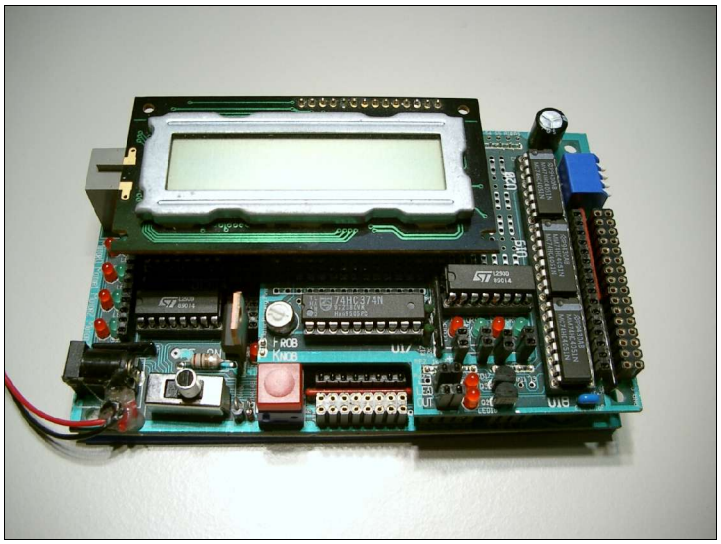


Abbildung 1.4: Das 6.270-Board

vorhandenen Schnittstellen ist nicht vorgesehen und deshalb prinzipiell nur mit relativ großem Aufwand möglich.

Außer einem Infrarot-Transceiver, der primär zur Programmierung des RCX dient, ist noch LC-Display vorhanden, das aber aussschließlich zur Anzeige von Zahlenwerten taugt.

Verglichen mit dem RCX bietet das 6.270-Board eine Fülle von Anschlußmöglichkeiten. Neben acht Analog- und ebenso vielen Digitaleingängen besitzt es vier Motorausgänge. Da das ganze System auf einer Bus-Architektur basiert, lassen sich die vorhandenen Schnittstellen problemlos durch Aufsteckmodule erweitern. Ein zweizeiliges LC-Display dient zur Anzeige beliebiger alphanumerischer Informationen. Die Programmübertragung geschieht über eine serielle Kabelverbindung, was zur Folge hat, daß ein Roboter zur Programmierung zwangsläufig in die Nähe des Rechners gebracht werden muß.

Im Gegensatz zu den Sensoren und Motoren, die für das LEGO-System mitgeliefert werden, muß hier jegliches Zubehör selbst gebaut werden. Zwar ist das in vielen Fällen nicht sonderlich schwer, da die Schnittstellen des Boards jedoch eine beliebige Verkabelung zulassen, ist dieses System potentiell fehleranfällig.

Der Eigenbau des Zubehörs muß aber nicht zwangsläufig als negativ bewertet werden. So lassen sich beispielsweise mit einfachen Mitteln leistungsfähige Sensoren für das 6.270-System bauen, die wesentliche präzisere Meßwerte liefern als die Sensoren des RCX.

Bezüglich der Aktoren ist die Lage eher umgekehrt. Die kompakte Bauform bei entsprechender Leistungsfähigkeit der LEGO-Motoren läßt sich im Eigenbau praktisch nicht erreichen. Üblicherweise basieren Antriebe für das 6.270-System auf modifizierten Modellbau-Servos. Diese haben den Nach-

	LEGO RCX	***MIT 6.270***
Ausstattung	--	+
Erweiterbarkeit	--	++
Flexibilität	-	++
Fehleranfälligkeit	++	-
Rechenleistung	+	o
Betriebssystem	++	+
Programmierung	++	+
Handhabung	++	-
Mechanische Integration	++	--
Platzbedarf	+	-
Sensoren	-	++
Aktoren	++	-

Abbildung 1.5: System-Vergleichstabelle

teil, daß sie weder besonders flexibel, noch leise sind. Allerding lassen sich auch LEGO-Motoren verwenden, was natürlich nur dann Sinn macht, wenn die restliche Mechanik auch aus LEGO-steinen aufgebaut wird.

Abbildung 1.6: Motoren des LEGO-Systems

Ähnliches gilt für den Platzbedarf und die Handhabung der beiden Systeme. Hier benötigt der RCX inklusive der nötigen Batterien gerade soviel Platz wie ein 6.270-Board ohne jegliche Stromversorgung. Obendrein bietet letzteres Board keinerlei Möglichkeiten zur mechanischen Fixierung auf der Roboterplattfom. Üblicherweise wird es nur mit Klebeband befestigt oder in einen passenden Halter gelegt.

Was die Programmierung betrifft, sollte man vermuten, daß hier das 6.270-Board prinzipiell überlegen ist, da es aus dem Umfeld einer Universität stammt und auf die Verwendung in Unterrichtsgruppen abgestimmt ist. Betrachtet man ausschließlich das von LEGO mitgelieferte Entwicklungssytem, so stimmt das auch. Allerdings gibt es eine große Fangemeinde, in der diverse Gruppen verschiedene Betriebssysteme für den RCX entwickelt haben. Besonders populär ist BrickOS, ein Multitasking-Kernel, der eine Programmentwicklung in C oder C++ gestattet.[5]

Das 6.270-Board wird über eine Entwicklungsumgebung mit dem Namen „Interactive C“ programmiert. Der zugehörige Kernel unterstützt ähnlich wie BrickOS auf dem RCX eine gleichzeitige Ausführung mehrerer Tasks.

Die Tabelle in Abbildung 1.5 gibt einen Überblick, der Stärken und Schwächen beider Systeme. Der RCX hat dabei in seiner unmodifizierten Form kaum eine Chance, dem 6.270-Board Konkurrenz zu machen. Zwar punktet er durch seine Ergonomie und Kompaktheit, was aber wenig hilft, wenn es bei einer Aufgabenstellung an Flexibilität oder Sensorik mangelt.

Das folgende Kapitel soll klären, was ein solches LEGO-System können müsste, um ein ernstzunehmender Konkurrent für die 6.270-Boards zu werden.

1.2 Bedarfsanalyse

Die Aufgabenstellungen, mit denen sich die Teilnehmer der Roboterbau-Kurse an der HAW-Hamburg beschäftigen, erfordern meist eine große Menge unterschiedlicher Sensorik. Die folgenden drei Beispiele sollen veranschaulichen, welche Ausstattung ein Roboter üblicherweise besitzen muß, um die Aufgaben meistern zu können. Das Kapitel 1.2.4 beschreibt dabei, welche Eigenschaften für den RCX in diesem Zusammenhang wünschenswert wären.

5 Neben BrickOS ist auch NQC eine sehr beliebtes Betriebssystem für den LEGO-RCX. Im Vergleich zu BrickOS ist diese C-ähnliche Entwicklungsumgebung wesentlich benutzerfreundlicher, eignet sich aber nur für relativ einfache Programme ohne Multitasking[8] [9] Ein weiteres System nennt sich LejOS und gestattet die Programmierung des RCX in Java. Allerdings ist die Hardware nur bedingt für solch ein Vorhaben geeignet, da die JVM bereits ohne ein Benutzerprogramm einen Großteil des Speichers belegt.[7]

1.2.1 Discokugelsammler

Eine typische Aufgabenstellung beinhaltet das Erkennen und Einsammeln von Objekten, die keine definierte Position auf dem Spielfeld haben. Das Aufsammeln und Abliefern kleiner Discokugeln ist eine Aufgabe, die vor einigen Semester in der unten beschriebenen Form sowohl mit RCX- als auch mit 6.270-Systemen gelöst wurde.

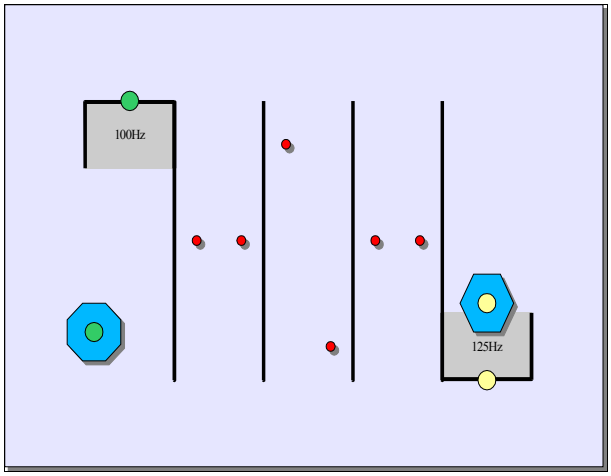


Abbildung 1.7: Spielfeldaufbau „Discokugelsammler“

Die Aufgabenstellung sieht folgendermaßen aus:

- Ziel ist es, innerhalb von drei Minuten möglichst viele der auf dem Spielfeld verteilten, ca. 5cm großen Discokugeln zu erkennen und in der eigenen „Garage“ abzuliefern.
- Garagen und Wände sind entsprechend der Abbildung 1.7 aufgestellt. Ihre Länge beträgt 2 Meter, der Abstand zwischen ihnen liegt bei 50 cm.
- Zwei Roboter starten zeitgleich aus ihrer Garage.
- Die Garagen sind mit IR-Beacons gekennzeichnet und besitzen einen reflektierenden Metallboden.
- 6 verchromte Discokugeln werden beliebig, mit einem Abstand von ca. 15 cm von einer Wand plaziert.

Die zu verfolgende Strategie ist stark von der Sensorausstattung und den mechanischen Fähigkeiten des Roboters abhängig. Die Analyse der Aufgabenstellung soll zeigen, welche Ausstattung wünschenswert wäre.


Abbildung 1.8: Eine der ca. 5cm großen Discokugeln

Der grundsätzliche Ansatz besteht darin, den Roboter an den Wänden entlangfahren zu lassen und die auf dem Weg liegenden Kugeln einzusammeln. Dazu muß er einerseits dazu in der Lage sein, mit einem definierten Abstand an den Wänden entlang zu fahren und andererseits möglichst viele Kugeln aufnehmen können, um nicht sofort wieder zur heimischen Garage fahren zu müssen. Erschwerend kommt hinzu, daß die Wände nicht miteinander verbunden sind und sich der Roboter somit von einer Wand zur nächsten „hangeln“ muß.

Die Erkennung der Kugeln ist relativ einfach, da sie durch ihre verchromten Oberflächen sehr gut reflektieren und sich schon allein durch diese optischen Eigenschaften vom gegnerischen Roboter unterscheiden lassen. Außerdem haben Kugeln allgemein den Vorteil, daß sie sich rollen lassen. Somit reicht eine trichterartiger Fangmechanismus aus, um sie zu einer Sammelvorrichtung zu bringen.

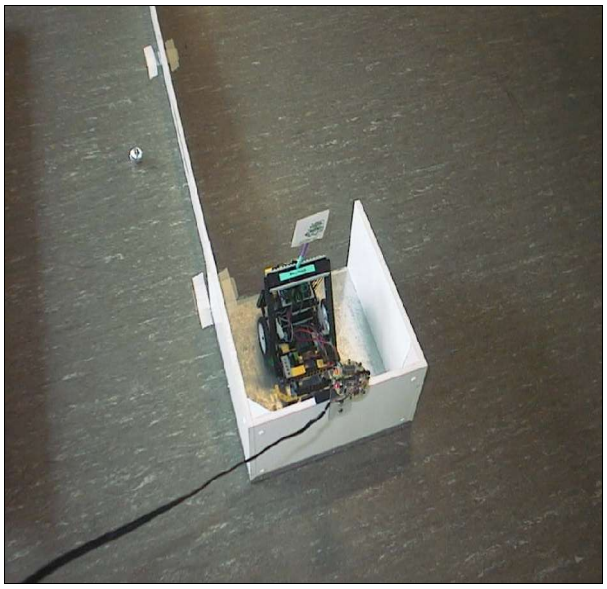


Abbildung 1.9: Discokugelsammler im Einsatz

Es macht allerdings nicht viel Sinn, jede gefundene Kugel einzeln in der Garage abzuliefern, da das zuviel Zeit in Anspruch nehmen würde. Der Roboter muß deshalb in der Lage sein, mehrere Discokugeln aufzunehmen und erst später in der heimischen Garage abzuliefern. Dazu ist eine Mechanik erforderlich, die bereits eingefangene Kugeln bei beliebigen Fahrbewegungen festhält. Außerdem muß nicht jedes Objekt eine Kugel sein, denn auch der gegnerische Roboter kann einem in die Quere kommen. Sofern nicht jede Blickrichtung durch einen Distanzsensor abgedeckt ist, empfiehlt sich die Verwendung von Bumpern[6].

Um die eigene Garage wiederzufinden, gibt es zwei naheliegende Strategien. Entweder es wird der gesamte Parcours abgefahren und dabei gezählt, wie oft man auf eine Garage trifft (denn jede zweite Garage ist das eigene Heim) oder aber der Roboter wertet die Signale der an den Garagen angebrachten IR-Beacons aus. Abbildung 1.10 zeigt eines dieser Infrarot-Leuchtfeuer.[7] Über die von ihnen abgestrahlte Modulationsfrequenz läßt sich die heimische Garage eindeutig identifizieren.

Die zweite Methode hat deutliche Vorteile, denn das Abfahren des gesamten Parcours beansprucht unnötig viel Zeit. Durch das direkte Auffinden der eigenen Garage lassen sich somit wesentlich schneller Punkte sammeln.

Ist man gezwungen, ohne Beacon-Erkennung auszukommen, kann eine Garage durch die gleichzeitige Auswertung dreier Distanzsensoren erfolgen. Alternativ bleibt die Möglichkeit, über Schleifkontakte oder einen zusätzlichen Lichtsensor die Beschaffenheit des Bodens zu untersuchen.

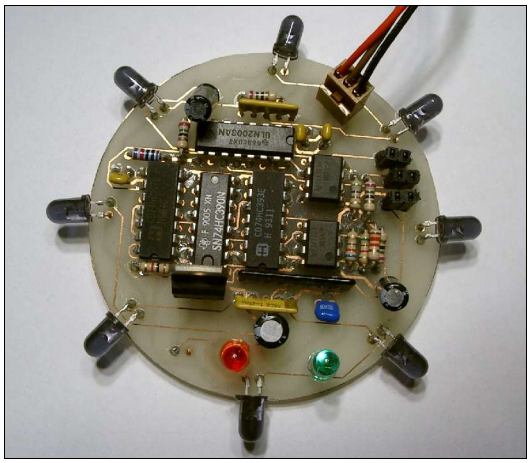


Abbildung 1.10: IR-Beacon

Für die Ausstattung des Roboters ergeben sich die folgenden Anforderungen:

- Drei Distanzsensoren für berührungsloses Wall-Following und zur Hinderniserkennung (links, rechts, vorn)

6 Hinter dem Begriff „Bumper“ verbirgt sich nichts weiter als ein Berührungssensor auf der Basis eines Schalters.

7 Der hier angesprochene IR-Beacon ist eine Entwicklung von Herrn Prof. Dr. Klemke. Die ursprüngliche Konzeption sah eine Verwendung im Zusammenhang mit Robotern vor, die das 6.270-Board als Basis benutzen. Nähere Informationen zur Funktionsweise des Beacons und der empfängerseitigen Auswertung finden sich in Kapitel 2.1.5.

- Bumper hinten und ggf. vorn
- Lichtsensor mit eigener Beleuchtung zur Erkennung, wann sich eine Discokugel vor der Sammelmechanik befindet
- Beacon-Detektor zur Garagen-Identifizierung
- Motorbetriebene Mechanik zum Festhalten gesammelter Kugeln (z.B. rotierendes Tor). Hier ist noch mindestens ein weiterer Sensor zur Statusrückmeldung des Tors nötig, sofern kein winkelgesteuerter Modellbau-Servos eingesetzt wird.

1.2.2 Dosensortierer

Diese Aufgabe ist eine Abwandlung des Sammelns der Discokugeln. Der wesentliche Unterschied besteht dabei darin, daß die Position der Wände variabel ist und daß nur eine bestimmte Art von unterscheidbaren Objekten gesammelt werden soll.

Die Aufgabenstellung sieht folgendermaßen aus:

- Ziel ist es, innerhalb von 3 Minuten möglichst viele Dosen der eigenen Farbe zu erkennen und in die heimische Garage zu bringen.
- Dosen sind entweder schwarz oder weiß und deutlich höher als die Wände des Parcours. Je drei Dosen einer Farbe werden beliebig plaziert.
- Der Parcours ist insgesamt 4,5m x 2,5m groß und mit 10 cm hohen Wänden umgeben.
- Die Position der inneren Wände ist beim Start unbekannt. Ihre Länge beträgt jeweils ein Vielfaches von 50 cm.
- Die Garagen der beiden gegnerischen Roboter sind durch IR-Beacons markiert und besitzen einen reflektierenden Metallboden.

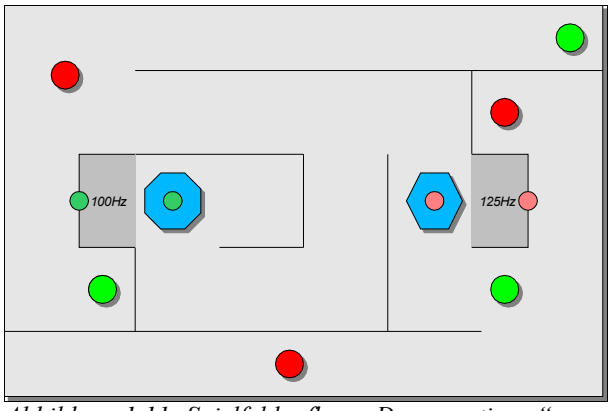


Abbildung 1.11: Spielfeldaufbau „Dosensortierer“

Das Lösen dieser Aufgabe ist ungleich schwerer als das Einsammeln von Kugeln, die auf einem fest abgesteckten Parcours liegen. Zwar war im ersten Beispiel nicht bekannt, wo genau die Kugeln plaziert sind, aber immerhin war die Lage der Wände fest kalkulierbar. Hier muß der Roboter in wesentlich stärkerem Maße auf seine Umwelt reagieren.

Neben den in Kapitel 1.2.1 beschriebenen Fähigkeiten sollte ein Roboter bei dieser Aufgabe dazu in der Lage sein, die zu transportierenden Objekte bereits aus größerer Entfernung zu erkennen und sich nach Möglichkeit auch zu merken, wo welche Hindernisse stehen.

Das Erkennen der Dosen ist relativ leicht, da sie höher als die Wände sind und sich damit auch aus größerer Entfernung per Distanzsensor erkennen lassen. Dagegen stellt die Kartierung der Wände auf dem Parcours eine ganz andere Herausforderung

dar. Über zwei Shaft-Encoder[8] kann hierzu beispielsweise der zurückgelegte Weg pro Antriebsrad gemessen werden. Die Berechnung eines optimalen Weges anhand der durch das Wall-Following gewonnenen Daten kann dabei den entscheidenden Geschwindigkeitsvorteil bringen.

Die bereits für den Discokugelsammler beschriebene Sensorausstattung erweitert sich um mindestens einen Distanzsensor und die zur Wegstreckenmessung erforderlichen Shaft-Encoder. Der Distanzsensor wird dabei so angebracht, daß er über die Wände „sehen“ kann, um Dosen aufzufinden.

1.2.3 Roboterfußball

Eine generell deutlich einfacher strukturierte Ausfgabe ist das Spielen von Roboterfußball. Der wesentliche Unterschied bezüglich der anderen beiden Aufgaben liegt darin, daß hier nur ein zu sammeldes Objekt (der Ball) vorhanden ist. Hier ist es demnach nicht unbedingt klug, stumpfsinnig nach dem Ball zu suchen und ihn im Erfolgsfall in die gegnerische „Garage“ zu bringen.

Die Aufgabenstellung sieht folgendermaßen aus:

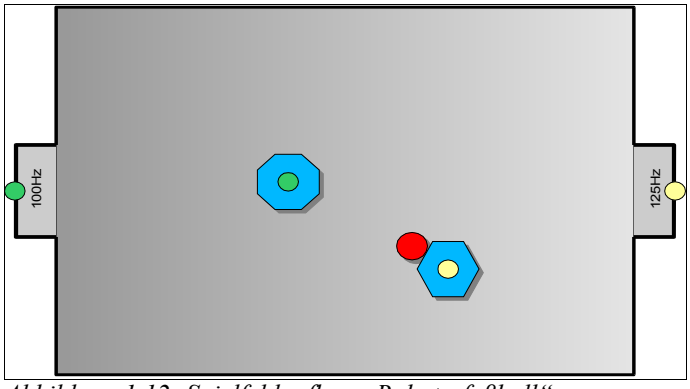


Abbildung 1.12: Spielfeldaufbau „Roboterfußball“

- Ziel ist es, den Ball möglichst schnell in das gegnerische Tor zu befördern.
- Das Spielfeld ist 3m x 1,5m groß. An den kurzen Enden befindet sich jeweils ein mit einem IR-Beacon markiertes, 70 cm breites Tor mit Metallboden.Die IR-Strahlung der Beacons ist nur auf einer Ebene mit der Höhe von 20 cm „sichtbar“.
- Der Ball hat einen Durchmesser von 7 cm und strahlt permanent ein unmoduliertes IR-Signal ab.
- Der Boden des Spielfelds ist mit einem Grauverlauf bedruckt, der zur Navigationszwecken benutzt werden kann.
- Die Ecken des Spielfelds sind abgeschrägt, damit sich der Ball dort nicht festsetzen kann.
- Die Roboter dürfen in ihrer Größe nicht den Durchmesser von 22 cm überschreiten.

Wie bei den beiden vorangegangenen Aufgabenstellungen sind Distanzsensoren essentieller Bestandteil der benötigten Sensorausstattung. Zusätzlich wird in diesem Fall ein spezieller Sensor benötigt, der die Erkennung der IR-Strahlung des Balls ermöglicht. Je nach Strategie muß der Roboter auch über einen motorgetriebenen Schußmechanismus verfügen.

8 Shaft-Encoder sind üblicherweise Drehwinkelgeber, die keinen Anschlag besitzen. Sie liefern eine bestimmte Anzahl von Zählimpulsen pro Umdrehung und können dabei gleichzeitig die Drehrichtung signalisieren.

1.2.4 Wünschenswerte RCX-Ausstattung

Anhand der drei Beispiele läßt sich leicht feststellen, über welche Aktoren und Sensoren ein Roboter verfügen muß, um die Aufgabenstellungen möglichst gut erfüllen zu können. Üblicherweise werden die folgenden Komponenten benötigt. In Klammern ist dabei angegeben, wieviele Bausteine einer Art vorhanden sein sollten:

- Bumper (1 bis 2)
- Distanzsensor (2 bis 4)
- Lichtsensor (1 bis 2)
- Shaft-Encoder (1 oder 3)
- Beacon-Detektor (1)
- Antriebs-Motoren (2)
- Motoren für Greif- und Schußmechanismen (1 bis 2)

Addiert man jeweils die kleinste benötigte Menge einer Komponente, so sind dies sechs Sensoren und drei Motoren. Am RCX lassen sich aber in seiner ursprünglichen Form nur drei Sensoren und drei Motoren betreiben. Hinzu kommt, daß weder Distanzsensoren noch Beacon-Detektoren zur Ausstattung des LEGO-Systems gehören.

Kategorie	***Funktion***	***Realisierung***	***Status / Umsetzung***	***Universelle Verwendbarkeit***
Hindernis-Erkennung	Bumper	Schalter	vorhanden	häufig
	Infrarot-Distanzmessung	Sharp GP2D12	aufwändig	sehr häufig
	Ultraschall-Distanzmessung	Sonar	sehr aufwändig	häufig
Objekt-Erkennung	Reflexionsmessung	Lichtsensor mit Beleuchtung	vorhanden	häufig
	Ballerkennung (IR-Strahlung)	IR-Fotodiode	recht einfach	sehr selten
	Farberkennung	Farbsensor	sehr aufwändig	selten
	Induktive Objekterkennung	Metalldetektor	aufwändig	selten
	Metallerkennung	Schleifkontakte	sehr einfach	selten
Navigation	Wegstreckenmessung	Shaft-Encoder	vorhanden	selten
	Beacon-Erkennung	Beacon-Detektor	sehr aufwändig	häufig
	Kompaß-Sensor	Hall-Sensor	aufwändig	selten
Objekt-Interaktion	Greifer / Klappe	Mikromotor	vorhanden	sehr häufig
		Modellbau-Servo	sehr aufwändig	sehr häufig
	Schußmechanismus	Modellbau-Servo	sehr aufwändig	häufig
	Winkelgeber für Schußmechanismus	Potentiometer	sehr einfach	häufig

Abbildung 1.13: RCX Sensor- / Aktor-Übersicht

Die Tabelle in Abbildung 1.13 gibt darüber Aufschluß, welche Sensoren und Aktoren in welchem Maße zum Einsatz kommen und mit wieviel Aufwand sie sich in Zusammenhang mit dem RCX benutzen lassen. Die dort verzeichneten Komponenten stellen allerdings nur eine Auswahl möglicher Erweitungen dar.

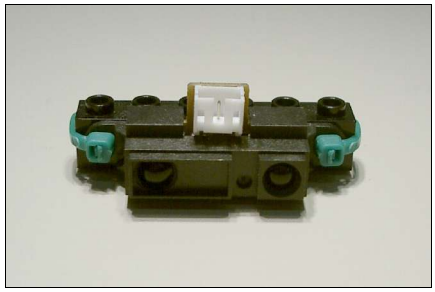


Abbildung 1.14: Sharp GP2D12 Distanzsensor

Die folgenden beiden Kapitel sollen klären, welche speziellen RCX-Erweiterungen bereits existieren und wie solch eine Erweiterung beschaffen sein muß, um die oben genannten Anforderungen möglichst universell zu erfüllen.

1.3 Bisherige RCX-Erweiterungen

Jeder Versucht, den RCX für relativ anspruchsvollere Roboterbau-Vorhaben zu nutzen, macht schnell die Grenzen dieses Systems deutlich. Zwar bietet der RCX bezüglich des Betriebssystems und der Prozessorleistung eine gute Basis, aber die spärliche Schnittstellenausstattung taugt nur für simple Aufgaben.

Viele enthusiastische LEGO-Bastler haben dieses Problem erkannt und Eigenbau-Lösungen entwickelt. Auf diversen Internetseiten finden sich deshalb unzählige Bauanleitungen für spezielle Sensoren und ähnliche RCX-Erweiterungen.

Eine sehr umfangreiche Sammlung von Schaltungen zu diesem Thema ist auf der Webseite von Michael Gasperi zu finden. [10] Er ist zudem Co-Autor des Buchs „Extreme Mindstorms“, in dem unter anderem die Grundlagen zum Bau von Sensor-Erweiterungen vermittelt werden. [11]

Die meisten der dort vorgestellten Bauanleitungen befassen sich allerdings damit, wie sich einzelne Spezialsensoren am RCX betreiben lassen. Das grundlegende Problem, das es zu lösen gilt, betrifft dagegen eher die Anzahl der gleichzeitig verwendbaren Sensoren.

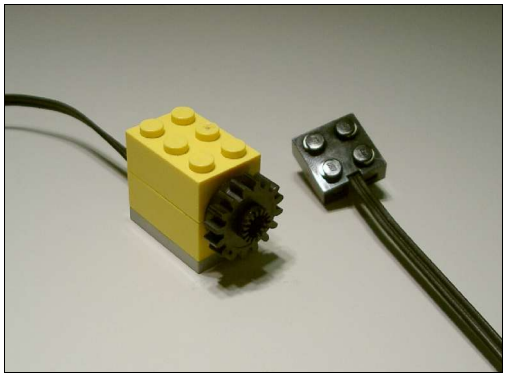


Abbildung 1.15: Eigenbau-Winkelgeber auf Basis eines Potentiometers

Leider sind Anleitungen für den Betrieb mehrerer Sensoren an einem einzelnen RCX-Eingang eher selten zu finden und beruhen üblicherweise auf dem in [12] beschriebenen Ansatz von Michael Gasperi.

Das sogenannte „Multiplexen“ des Sensor-Eingangs geschieht dabei über digital gesteuerte Analogschalter. Um bestimmen zu können, welcher der angeschlossenen Sensoren aktiv sein soll, wertet eine Steuerschaltung den Betriebsmodus des RCX-Eingangs aus. Jede Modus-Umschaltung bewirkt dabei das Schalten zum nächsten Eingang des Multiplexers.

Dieser Ansatz der Multiplexer-Steuerung ist zwar trickreich, weil dabei kein weiterer Anschluß des RCX belegt wird, aber leider hat er auch gravierende Nachteile:

Da die Modus-Umschaltung als Schaltimpuls dient und der Multiplexer seine Versorgungsspannung aus dem RCX-Eingang gewinnt, können nur passive Sensoren verwendet werden. Außerdem dauert das Schalten von einem zum nächsten Sensor systembedingt mindestens 10 Millisekunden. Je nach Anzahl der Multiplexer-Kanäle dauert es entsprechend lange, bis jeder Sensor einmal abgefragt wurde. Da Kanäle nicht einzeln adressierbar sind, müssen zwangsläufig auch unbenutzte Eingänge „durchgezählt" werden. Für zeitkritische Auswertungen dürfen demnach kaum mehr als drei Eingangskanäle am Multiplexer vorhanden sein.

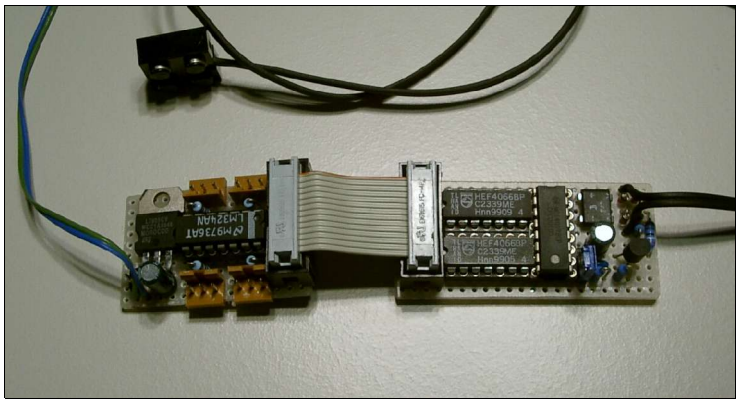


Abbildung 1.16: Sharp-Distanzsensor-Multiplexer

Rainer Balzerowski kombinierte die oben angesprochenen Multiplexer-Ansatz mit einer Vorverstärkerschaltung, die den Anschluß von Infrarot-Distanzsensoren der Firma Sharp ermöglicht. Damit war es möglich, drei Distanzsensoren an einem RCX-Eingang zu betreiben. Allerdings benötigt man zur Versorgung der Sharp-Sensoren zusätzliche Batterien und hat mit den beschriebenen Timing-Problemen zu kämpfen.

Die Abbildung 1.16 zeigt diesen Multiplexer in einer erweiterten Variante, die vier oder acht Eingangskanäle unterstützt. Die Verstärkerschaltung ist dabei auf einer getrennten Leiterplatte untergebracht, womit sich bei Bedarf auch andere Eigenbau-Sensoren anschließen lassen. Trotz des kompakten Aufbaus beansprucht die Schaltung eine nicht unwesentliche Menge an Platz.

Neben Webseiten, auf denen Bauanleitungen und Schaltpläne präsentiert werden, finden sich im Internet auch Firmen, die RCX-Erweiterungen verkaufen. Die Firma Mindsensors hat dabei zwei Arten von Eingangs-Multiplexern im Angebot, die sich auch im Zusammenspiel mit aktiven Sensoren verwenden lassen. [13] Die Steuerung erfolgt hier ähnlich wie beim Gasperi-Multiplexer über die Umschaltung des Betriebsmodus.

Was die Erweiterung der vorhandenen Motorausgänge angeht, ist das Angebot ähnlich mager. Es existiert offenbar keine Lösung, die die Ein- und Ausgänge des RCX in universeller, professionell nutzbarer Weise erweitert.

1.4 Konzeption

Wie bereits angesprochen, fehlt der LEGO-Gemeinde eine wirklich universelle RCX-Eweiterung, die flexibel genug ist, den üblichen Anforderungen gerecht zu werden. In Kapitel 1.2.4 wurde bereits darauf eingegangen, welche Sensor- und Aktortypen in den meisten Fällen unverzichtbar sind. Besonders deutlich ist der Bedarf für Distanzsensoren und einen Beacon-Detektor erkennbar. Letzterer ist in einer RCX-Variante bisher überhaupt nicht vorhanden, da diese IR-Leuchtfeuer eine Eigenentwicklung der HAW-Hamburg sind.[9]

9 Das Kapitel 2 der Studienarbeit „Konzeption von I/O-Erweiterungen für LEGO Mindstorms"

Die folgenden Abschnitte sollen klären, wie eine Erweiterung im Detail beschaffen sein muß, um diese Lücke zu füllen.

1.4.1 Anforderungsprofil & Komponentenauswahl

Bei der Entwicklung von RCX-Erweiterungen gilt es einige grundsätzliche Dinge zu beachten, die sich im späteren Einsatz negativ auswirken können. Dabei muß das gesamte System betrachtet werden und nicht nur die Anzahl der Erweiterungskanäle oder der Typ der verwendbaren Sensoren.

Wichtig ist, daß eine Erweiterung so wenig RCX-Anschlüsse wie möglich beansprucht. Es ist nicht viel damit gewonnen, wenn ein Multiplexer zwei Sensoren an einem Eingang betreiben kann, dafür aber einen weiteren RCX-Anschluß zur Steuerung belegt. Für eine Erweiterung, die gleichzeitig auch weitere Motorausgänge zur Verfügung stellt, ist ein dafür geopferter Motorport des RCX leicht zu verschmerzen.

Die hier angestrebte RCX-Erweiterung verfolgt diesen Ansatz. Sie soll unter Verwendung eines Motorports möglichst viele Sensoren schalten und Aktoren steuern können. Dies mit diskreter Logik zu erreichen, ist nahezu unmöglich, da allein schon die Übertragung der Steuerdaten zumindest einen programmierbaren Logikbaustein erfordert. Solche Bausteine verbrauchen wiederum unverhältnismäßig viel Strom, was sie für unsere Zwecke untauglich macht. Glücklicherweise gibt es mittlerweile sehr sparsame Mikrocontroller, die sich in ihrem Preis kaum von einem programmierbaren Logikbaustein unterscheiden. Gleichzeitig bietet solch ein Controller mittels entsprechender Software die Möglichkeit, sich sehr flexibel den Hardware-Gegebenheiten anzupassen. Natürlich darf dabei nicht außer Acht gelassen werden, daß es eigentlich widersinnig ist, eine Erweiterung für den bereits im RCX vorhandenen Mikrocontroller mit einem weiteren Prozessor zu versehen. Die in unserem Fall favorisierte MCU der MSP-430-Familie von Texas Instruments ist allerdings so klein und kompakt aufgebaut, daß sie sich optimal für die Protokollauswertungs und Steuerungsaufgaben eignet.[10]

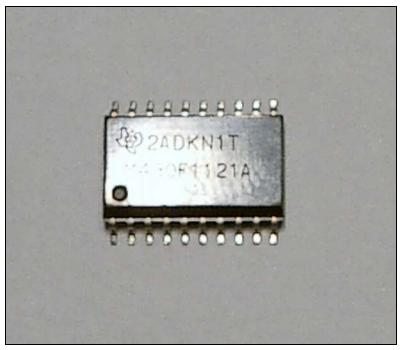


Abbildung 1.17: MSP 430 F 1121 A

Bezüglich Anzahl und Art der gemultiplexten Eingänge sollte sichergestellt sein, daß sich eine für die meisten Anwendungsfälle ausreichende Menge beliebiger LEGO-kompatibler Sensoren anschließen läßt. Erreicht werden kann das durch gleichzeitiges Umschalten beider Pole eines Sensors mittels zweier Analog-Multiplexer.[11]

Auch der Anschluß von Sharp-Distanzsensoren muß mit einfachen Mitteln möglich sein. Da die bereits existierende Multiplexer-Variante eine getrennte Platine für die Verstärkerschaltung vorsieht, läßt sich dieses Modulkonzept hier weiterverwenden.

---

beschreibt den Ansatz für einen RCX-Beacon-Sensor. [1]

10 Abbildung 1.17 Zeigt diesen Prozessor im 20-poligen SMD-Gehäuse. Er wird in Kapitel 2.1.1 näher beschrieben.

11 Je nach gewünschter Kanalzahl kann hierfür auf Bausteine der CD40xx-Reihe zurückgegriffen werden. Neben den bereits im oben angesprochenen Gasperi-Multiplexer eingesetzten Einzelschaltern CD 4066 gibt es weitere Bausteine, in die bereits die Steuerlogik für einen 4:1 (CD4052), einen 8:1 (CD4051) oder einen 16:1 (CD4067) -Multiplexer integriert ist. [10] [14] [15]

Statt einen Beacon-Detektor später als getrennten Sensor anzuschießen, ist es naheliegend, die zur Beacon-Erkennung nötige Frequenzauswertung vom Mikrocontroller erledigen zu lassen. Der Detektor besteht damit nur aus drei IR-Empfängern, die ihre Signale direkt zur MCU liefern.[12] Nachdem per Software die Frequenz des gesichteten Beacons errechnet wurde, muß diese Information noch an den RCX weitergegeben werden. Dazu eignet sich ein simpler, aus einem Widerstandsnetzwerk bestehender D/A-Wandler. Hierüber läßt sich gleichzeitig ein Acknowledge-Signal als Rückmeldung für den Steuerdaten-Empfang realisieren.[13]

- möglichst wenige blockierte Ports am RCX
- muß Ersatz für belegte RCX-Ports bieten
- Multiplexer für möglichst viele Sensoren
- kompatibel zu Sharp-Distanz-sensor
- integrierter IR-Becon-Detektor
- pulsweitenmodulierte Motor-ausgänge
- geringe Stromaufnahme
- geringe Abmaße
- bevorzugte Verwendung von Standardbauteilen
- kostengünstige Fertigung

Abbildung 1.18: Anforderungsprofil

Die Verwendung eines Mikrocontrollers als zentraler Bestandteil der Schaltung hat einen weiteren großen Vorteil: Zur Kompensation des für die Datenübertragung zweckentfremdeten Motorports sollte die RCX-Erweiterung für mindestens einen zusätzlichen Motorausgang sorgen. Um diesem die gleichen Fähigkeiten zu verleihen wie einem der RCX-Motorports, kann per Software ein pulsweitenmoduliertes Signal an den Motortreiber ausgegeben werden. Zusammen mit dem passenden Treiberbaustein[14] entspricht auch das elektrische Verhalten dem Original. Da für Schuß- oder Greifmechanismen oftmals mehr als ein Motor benötigt wird, empfiehlt es sich, gleich zwei dieser Motorausgänge vorzusehen.

Weitere wichtige Design-Aspekte sind Stromverbrauch und Größe der Schaltung. Gerade bei Aufgaben wie dem Roboterfußball kommt es auf eine sehr kompakte Bauform an. Die Verwendung von Bauteilen im SMD-Gehäuse wirkt sich zwar nachteilig auf die Reparaturfreundlichkeit der Schaltung aus, stellt jedoch den einzigen Weg dar, die nötige Packungsdichte zu erreichen.

Daß ein batteriebetriebenes Gerät eine möglichst geringe Stromaufnahme haben sollte, erklärt sich von selbst. Die vorgesehenen CMOS-Bausteine sind in dieser Hinsicht aber vollkommen unkritisch. Sofern nicht unbedingt nötig, empfiehlt es sich allein schon aus Kostengründen, auf Spezialbauteile zu verzichten. Die recht teuren Motortreiber sind dabei ein Sonderfall, bei dem die Mehrkosten durch wesentlich geringeren Platzbedarf und andere positive Eigenschaften gerechtfertigt werden.

Die folgenden drei Kapitel gehen detailliert auf die grundlegende Problematik der Datenübertragung vom und zum RCX ein, da dies die größte Hürde beim Bau von RCX-Erweiterungen darstellt.

12 Drei Sensoren sind nur dann nötig, wenn über einen „Strahlungstrichter“ bestimmte Winkel des Sichtfelds ausgeblendet werden, um Rückschlüsse auf den genauen Standort eines Leuchtturms zu ermöglichen. Eine genaue Beschreibung der Umsetzung des Beacon-Detektors ist in den Kapiteln 2.1.5 und 2.2.1.6 zu finden.

13 Die Kapitel 1.4.4 und 2.1.3 geben darüber Aufschluß, welche Feinheiten dabei beachtet werden müssen.

14 Der RCX beinhaltet Treiberbausteinedes Typs MLX 10402. Siehe Kapitel 2.1.6.

1.4.2 Datentransfer

Die Übertragung von Daten über einen der RCX-Motorausgänge ist keine triviale Angelegenheit. Zwar ist eine Interface-Schaltung bestehend aus einem als Verpolungsschutz dienendem Gleichrichter und einem Optokoppler schnell aufgebaut, aber damit ist man von einer funtionierenden Übertragung noch weit entfernt.

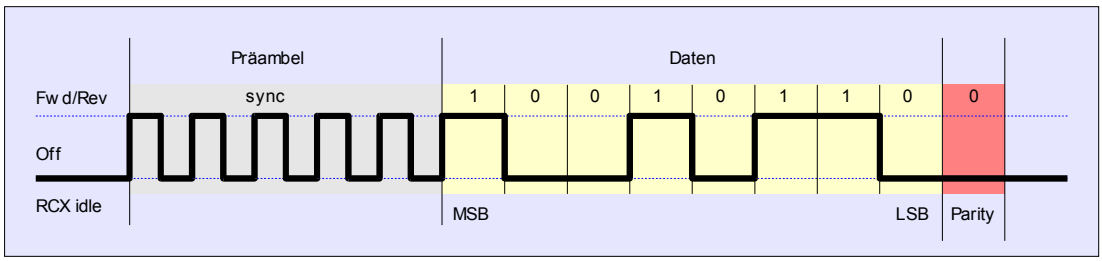


Abbildung 1.19: Theoretischer Protokollaufbau

Um Datenworte versenden und auch korrekt wieder empfangen zu können, wird ein serielles Protokoll benötigt, mit dem sich Takt und Daten über eine einzige Leitung transportieren lassen. Das in Abbildung 1.19 skizzierte Protokoll entstand aus der Vorgabe, acht Datenbits mit Paritätsbit zu übertragen. Da der Sender nicht über eine gesonderte Taktleitung verfügt und der Empfänger keine festen Taktzeiten benutzt, wird vor jedem Datenwort eine fünf Bit lange Präambel gesendet. Der Empfänger mißt die Zeiten zwischen den Flanken dieser Präambel und berechnet daraus die passenden Sampling-Zeitpunkte bezüglich der Datenbits.[15]

Sender und Empfänger berechnen das Parity-Bit anhand einer schrittweisen XOR-Verknüpfung der Datenbits.

1.4.3 Schnittstellen-Eigenschaften

Ein wesentlicher Aspekt beim Bau von RCX-Erweiterungen ist die Beschaltung und das Verhalten der RCX-Schnittstellen. Da der RCX als Spielzeug konzipiert wurde, legte man größten Wert auf ein tolerantes Verhalten im Falle einer Fehlbedienung. Dies schlägt sich nicht nur in kurzschlußsicheren Steckverbindern, sondern auch in der Innenbeschaltung der Anschlüsse nieder, was den Bau von Erweiterungen deutlich erschwert.

Besonders ungünstig ist hierbei das Verhalten der Sensor-Eingänge. Solch ein Eingang ist jeweils an einen A/D-Wandler mit 10-Bit Auflösung angeschlossen und kann in zwei Modi (passiv und aktiv) betrieben werden. Je nach Programmierung kann er auch Strom für aktive Sensoren liefern.

Im passiven Modus ist der A/D-Wandler-Eingang mit einem 10-Kiloohm Pull-Up-Widerstand gegen 5 Volt geschaltet. Der zweite Pol des Sensoreingangs ist fest mit Masse verdrahtet. Durch einen entsprechenden Widerstandswert zwischen dem Masse-Pol und dem A/D-Wandler-Pol des Sensoreingangs ergibt sich der zugehörige Meßwert. Schaltet man den Eingang aber in den aktiven Modus, so wird die volle Batteriespannung des RCX auf den A/D-Wandler-Pol des Eingangs gelegt. Nur für den Moment der A/D-Wandlung wird diese Spannung abgeschaltet, da sich sonst kein sinnvoller Sensorwert ermitteln ließe.

15 Eine fünf Bit lange Präambel erscheint zwar etwas übertrieben, begründet sich aber durch das instabile Timing des RCX in Zusammenhang mit BrickOS. Siehe Kapitel 2.2.2 und 2.2.1.5.

Dieses Verhalten hat weitrechende Konsequenzen für die angeschlossene Peripherie, denn es ist nicht vorhersagbar, welche Spannung zu welcher Zeit am Sensoreingang anliegt. Zudem ist auch die Polung nicht festgelegt, denn sie hängt von der Ausrichtung des angeschlossenen Steckers ab.

In unserem Fall hat dieses Verhalten besonders weitreichende Konsequenzen für das Schaltungsdesign. Die Analogschalter, die für den Multiplexer verwendet werden sollen, besitzen zwar einen großen Versorgungsspannungsbereich, haben aber die Einschränkung, daß die von ihnen geschalteten Spannungen innerhalb des Versorgunsspannungsbereichs liegen müssen.

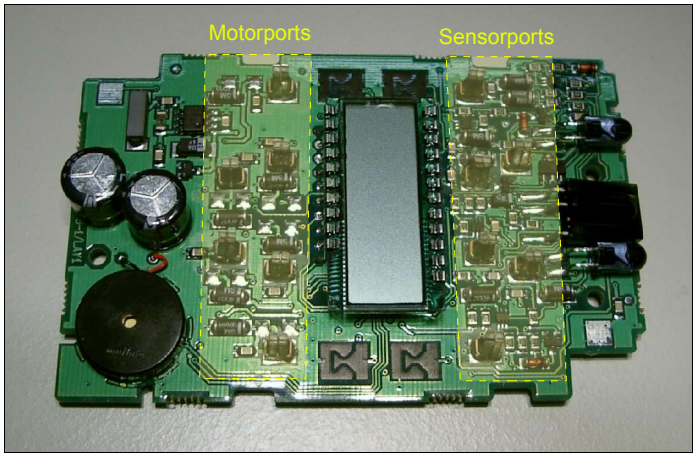


Abbildung 1.20: Innenansicht des RCX

Würden wir also den Multiplexer mit 5 Volt betreiben, was für den passiven Modus problemlos möglich wäre, so würden bei einer versehentlichen Umschaltung in den aktiven Modus bis zu 9 Volt am Ausgang des Multiplexers anliegen. Dies würde dann vermutlich zur Zerstörung der Analogschalter führen, da die im Verhältnis zur Versorgunsspannung negative Eingangsspannung einen Kurzschluß über die internen Schutzdioden zur Folge hätte.

Denkbar wäre auch, daß die Multiplexer zwar mit einer für beide Betriebsmodi passenden Versorgungsspannung betrieben werden, die aber durch eine leere Batterie nicht mehr vorhanden ist. Genauso könnte der Multiplexer noch abgeschaltet sein, während der RCX schon eingeschaltet ist.

Um sich gegen derartige Sonderfälle abzusichern, bedarf es relativ aufwendiger Schutzmaßnahmen, auf die in den Kapiteln 2.1.3, 2.1.4 und 2.1.7 detailliert eingegangen wird.

1.4.4 Datenübertragung in Richtung des RCX

Daten aus dem RCX zu versenden ist ein relativ kleines Problem verglichen mit den Hürden, auf die man beim Versuch eines RCX-seitigen Datenempfangs stößt. Der interne Aufbau der Sensoreingänge erschwert solch ein Vorhaben ungemein, da ausschließlich A/D-Wandler-Eingänge zur Verfügung stehen. Solche Eingänge sind bedingt durch ihren Aufbau nicht interruptfähig. Anderenfalls hätte man ein Datenwort flankengetriggert übertragen können. Würde man diesen Ansatz ohne Interrupt-Möglichkeit umsetzen, würde ein Großteil der Rechenzeit für das „Polling“ des A/D-Wandlers verschwendet werden. Ein gleichzeitiger Betrieb eines Betriebssystems wie z.B. BrickOS wäre unmöglich.

Es bleibt also nur die Möglichkeit, ein Datenwort zeitgleich über den 10-Bit-Wertebereich des A/D-Wandlers zu codieren. Auf diese Weise eine sichere Datenübertragung für mehr als drei bis vier Bit zu realisieren, ist aus verschiedenen Gründen äußerst schwierig.

Wirft man einen Blick auf Schaltungen von LEGO-kompatiblen Sensoren, so fällt auf, daß die Rückmeldung des Sensorwertes üblicherweise über zwei Dioden geschieht, deren Anoden an jeweils einen der RCX-Eingangspole angeschlossen sind. Damit ist für Verpolungssicherheit gesorgt und gleichzeitig kann keine positive Spannung an die Elektronik des Sensors gelangen. Leider hat diese Beschaltung den gravierenden Nachteil, daß die jeweils aktive Diode den Sensorwert erheblich beeinflusst. Allein der Spannungsabfall von 0,7 Volt an der üblicherweise benutzten Siliziumdiode sorgt dafür, daß die Spannung am Wandler-Eingang nicht mehr bis auf 0 Volt gesenkt werden kann. Somit ist etwa ein Siebtel des Wertebereichs nicht nutzbar.

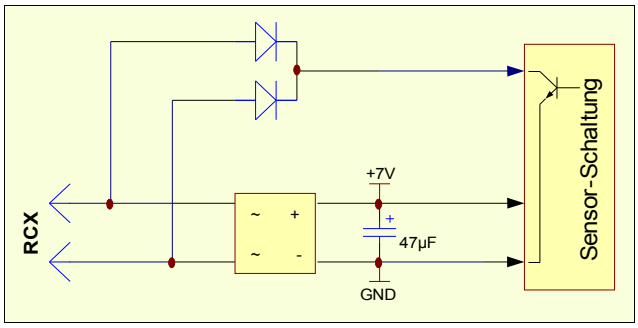


Abbildung 1.21: Typische Sensorbeschaltung mit zwei Dioden im Signalweg (eine davon im Gleichrichter)

Hinzu kommt, daß auch die Verbindung zur RCX-Masse verpolungssicher sein muß, was einen weiteren Diodenübergang zur Folge hat. Das Potential der Sensor-Masse liegt damit 0,7 Volt über dem der RCX-Masse. Auch diese Differenz wirkt sich negativ auf den Wertebereich aus. Selbst selbst wenn die Sensor-Masse ohne weitere Diodenverfälschung an den Wandler-Eingang gelangen würde, könnte der tatsächliche 0-Volt-Pegel nicht mehr erreicht werden.

Unpraktischerweise addieren sich die beiden Diodenübergänge zu einem Minimal-Pegel von 1,4 Volt am Wandler-Eingang, was gleichbedeutend mit dem Verlust von mehr als einem Viertel des Wertebereichs ist.

Eigentlich sollte man erwarten, daß der verbleibende Werteumfang von etwas mehr als 9 Bit für eine tatsächliche Übertragung von mindestens 6 Bit ausreicht. Prinzipiell stimmt das auch, nur müssten diese 6 Bit dann auf den gesamten verfügbaren Bereich abgebildet werden. Um allerdings im RCX wieder das ursprüngliche Datenwort zu erhalten, wäre ein extremer Rechenaufwand nötig. Schließlich entsprechen die Sensorwerte nicht mehr den ursprünglichen Bitgrenzen.

Beschränkt man sich auf die Übertragung weniger Bits, läßt sich dieses Problem weitestgehend dadurch umgehen, daß die Sensorwerte passend zu den Bitgrenzen der höchstwertigen Bits im Wertebereich gewählt werden. Die Unschärfe in den niederwertigen Bits kann beim Auslesen innerhalb des RCX einfach durch eine logische UND-Operation ausmaskiert werden. Schiebt man die verbleibenden Bits danach um die Länge der Maskierung nach rechts, erhält man die Originaldaten.

Die konkrete Umsetzung bezüglich dieses Projekts ist in Kapitel 2.1.3 beschrieben. Weitere Überlegungen zu dem Thema finden sich in Kapitel 6.2.

1.4.5 Namensgebung

Es läßt sich schlecht an einer Entwicklung arbeiten, ohne zumindest einen Arbeitstitel dafür zu haben. Da relativ schnell klar war, was für ein Gerät bei der Umsetzung entstehen sollte, gingen die ersten Namensüberlegungen in Richtung „LePRA“ - ein Kürzel für ***LE**GO-**P**ort-**R**eplication-**A**ttachment*.

Dieser offensichtlich nicht gerade verkaufsfördernde Name wurde dann relativ schnell von „LEPoMux“ abgelöst - dem ***LE****GO*-***Po***rt-***Mu***ltiple***x***er.

2 Machbarkeitsstudie: Lepomux v1.0

Dieses Kapitel beschäftigt sich mit der praktische Umsetzung der im ersten Kapitel zusammengetragenen theoretischen Vorüberlegungen. Entsprechend dem oben genannten Arbeitstitel, ist diese Machbarkeitsstudie auf den Namen „Lepomux v1.0“ getauft worden.

2.1 Hardware

Im folgenden Abschnitt wird detailliert auf die Schaltung des Lepomux eingegangen. Das Anforderungsprofil aus Abbildung 1.18 war dabei maßgebend für die Komponentenauswahl. Allerdings lassen sich verschiedene Anforderungen oftmals nicht miteinander vereinbaren. Am deutlichsten wird das in unserem Fall bei den verwendeten Motortreibern. Diese ansich unnötig teuren Bausteine werden in einem, aus Platzgründen zwingend benötigten, SMD-Gehäuse angeboten. Außerdem werden sie auch innerhalb des RCX für die Motorausgänge verwendet, was ein hohes Maß an Kompatibilität sicherstellt. Das Erfüllen der Anforderungen ist also meistens eine Gratwanderung zwischen Kosten und Leistung.

Die Abbildung 2.1 zeigt den schematischen Aufbau und die Komponenten des Lepomux, auf die in den folgenden Abschnitten näher eingegangen wird.

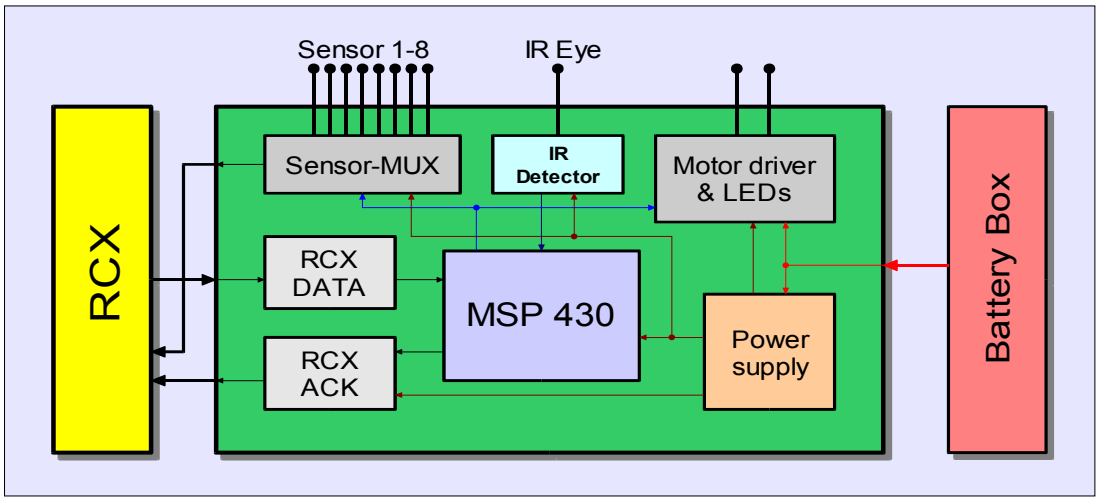


Abbildung 2.1: Lepomux v1.0 Komponentenschema

Schaltungsdetails, die in diesem Kapitel beschrieben werden, beziehen sich auf die beiden Schaltpläne im Anhang und . Sie entsprechen einer mechanischen Umsetzung auf zwei Leiterplatten.[16]

2.1.1 Der Prozessor

Zentraler Bestandteil des Lepomux ist ein Mikrocontroller der MSP-430-Familie von Texas Instruments. Der verwendete ***MSP430 F1121A***[17] ist in einem 20-poligen SMD-Gehäuse untergebracht und eignet sich Dank seiner Ausstattung[18] hervorragend

16 Siehe Kapitel 2.1.8.

17 Im Folgenden MSP genannt.

18 Der MSP430F1121A besitzt neben 4kB Flash-ROM und 256 Byte RAM noch drei Input/Capture und Output/Compare-Blöcke. Über ein JTAG-Interface bzw. eine serielle Schnittstelle kann er bei entsprechendem Schaltungsdesign auch in eingebautem Zustand programmiert werden. Bemer-

für unsere Zwecke. Er wird mit einem 8 MHz-Quarz getaktet[19] und mit einer Spannung von 3,3 Volt betrieben.

Diese Betriebsspannung bringt allerdings einige Probleme mit sich: Ausgelegt ist der Prozessor für Spannungen von 1,8 bis 3,6 Volt, wobei zu kleine Spannungen das Programmieren des Flash-ROMs verbieten. Für einen Betrieb mit vollen 8 MHz wird eine Versorgungsspannung von 3,6 Volt empfohlen.[20] Leider gibt es keine kostengünstigen Festspannungsregler mit einer Ausgangsspannung von 3,6 Volt. Die Verwendung eines 3,3-Volt-Reglers ist somit ein Kompromiß hinsichtlich der Bauteilkosten bzw. -größe.

Desweiteren führt diese relativ niedrige Versorgungsspannung zu „Verbindungsproblemen“ mit den übrigen Komponenten, da beispielsweise die IR-Sensoren und die Motortreiber mit Logikpegeln von 5 Volt arbeiten. Es sind demnach an den meisten Anschlüssen des MSP Pegelwandler notwendig, um die Anpassung an die höheren Betriebsspannungen der anderen Komponenten vorzunehmen. Eine kostengünstige und kleine Lösung hierfür sind die Levelshifter ***MC 14504*** von Motorola. [17]

2.1.2 RCX-Daten-Eingang

Die Steuerdaten für den Lepomux sendet der RCX über einen seiner Motorports. Um diese Daten per Mikrocontroller auswerten zu können, muß das Signal entsprechend aufbereitet werden. Dazu durchläuft es entsprechend der Abbildung 2.2 zunächst den Brückengleichrichter REC-1, der als Verpolungsschutz dient, und gelangt danach an den LED-Anschluß des Optokopplers U12. Er dient primär als Pegelwandler, der die RCX-Batteriespannung am Motorport auf 3,3 Volt umsetzt.
Im Ruhezustand sorgt der Pull-Up-Widerstand R6 für einen 3,3-Volt-Pegel am Dateneingang des MSP. Wird der Ausgangstransistor des Optokopplers durchgeschaltet,

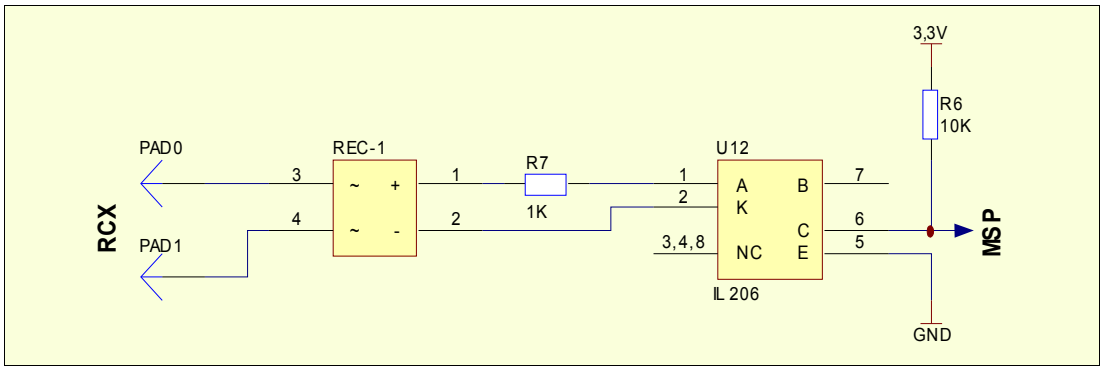


Abbildung 2.2: Schaltung des Dateneingang

kenswert ist die niedrige Stromaufnahme von weniger als einem Milliampère und der geringe Preis von ca. €1,60 bei 1000 Stück.[16]

19 Wie sich erst in einer späteren Version des Lepomux herausstellte, hatte ein Programmierfehler in der zugehörigen Software allerdings dafür gesorgt, daß der Prozessor praktisch immer mit dem internen RC-Oszillator (DCO) und somit mit einer Frequenz von ca. 1 MHz betrieben wurde. Da der freilaufende Timer (TAR) in jedem Fall mit einer Frequenz von 500 kHz getaktet werden sollte und diese Taktung in diesem Fall nur vom Vorteiler des externen Oszillators abhängt, fiel der Fehler nicht auf. Es hätte also in dieser Version auf den externen Quarz komplett verzichtet werden können, denn die Rechenleistung war auch so ausreichend.

20 In der Praxis zeigte sich, daß der MSP auch bei 3,3 Volt und voller Taktfrequenz zuverlässig arbeitet. Für eine marktreife Version des Lepomux sollte allerdings ermittelt werden, welche Taktfrequenz tatsächlich benötigt wird, um die benötigte Rechenleistung zur Verfügung zu stellen.

zieht er diesen Pin auf das Masse-Potential. Somit liegen die RCX-Daten invertiert am Eingangspin des MSP an.
Die galvanische Trennung ist wegen der H-Brücken-Struktur der RCX-internen Motortreiber vorteilhaft.
Um mit der Lepomux-Masse möglichst nahe am Potential der RCX-Masse zu liegen[21], wäre solch eine Verbindung über den Motorport ein denkbar ungünstiger Punkt. Grund dafür ist der zusätzliche Diodenübergang innerhalb des Motortreibers. Zusammen mit dem Diodenübergang innerhalb des Verpolungsschutz-Gleichrichters ergäbe sich eine Spannungsdifferenz von 1,4 Volt. Wesentlich geschickter läßt sich diese Verbindung über einen der RCX-Sensor-Eingänge herstellen.[22]

2.1.3 RCX-Achnowledge-Ausgang

Um dem RCX mitteilen zu können, ob ein Datenwort korrekt empfangen wurde, benötigt der Lepomux einen Rückkanal, der an einen der RCX-Sensoreingänge angeschlossen wird. Hierbei aber nur eine boolsche Information zu übertragen, wäre reine Resourcenverschwendung, schließlich sammelt der Lepomux auch Informationen über die Frequenzauswertung von drei IR-Sensoren. Um diese Informationen zeitgleich an den RCX übertragen zu können, bedürfte es einem Wertebereich von sechs Bit. Davon entfallen jeweils zwei Bit auf einen Sensor.

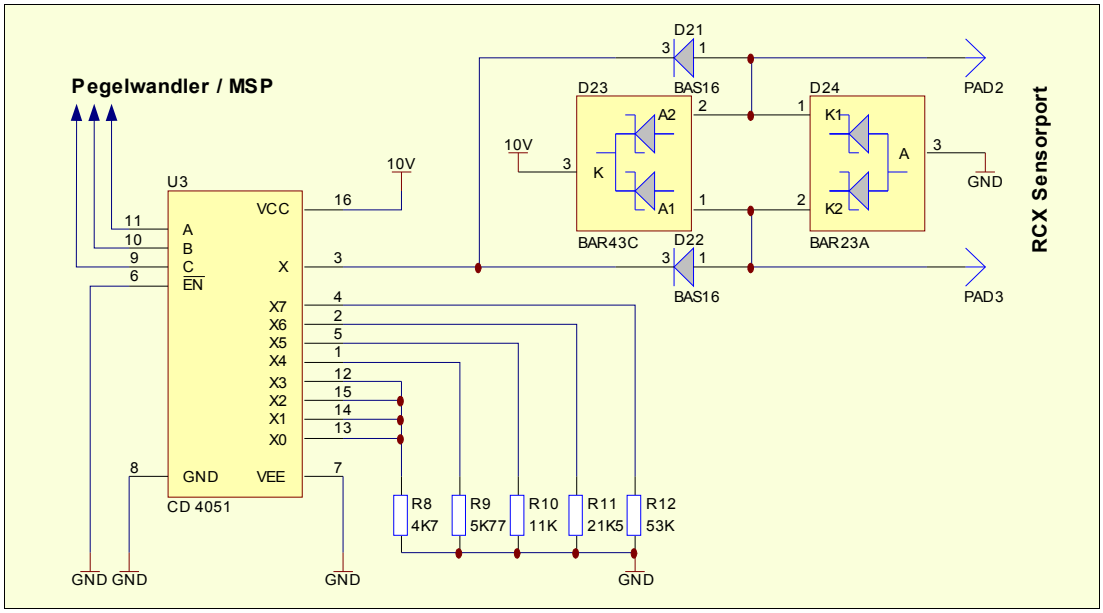


Abbildung 2.3: Schaltung des RCX-Acknowledge-Ausgangs

Zusammen mit dem Acknowledge-Bit[23] müssten also 7 Bit auf einen Bereich von 10 Bit abgebildet werden – ein mit einfachen Mitteln nahezu hoffnungsloses Unterfangen.[24]
Um die in Abbildung 2.3 gezeigte Schaltung einfach zu halten, wurde deshalb der

21 Die Art und Weise, wie die beiden Masse-Potentiale miteinander verbunden werden, hat Einfluß auf eine ganze Reihe weiterer Designüberlegungen und das Verhalten bzw. die Störanfälligkeit der Schaltung (siehe Kapitel 1.4.3).
22 Siehe Kapitel 2.1.3.
23 Im folgenden ACK-Bit genannt.
24 Das Abbilden vieler Bits auf einen RCX-Sensoreingang hat gewisse elektrisch bedingte Tücken, auf die in Kapitel 1.4.4 näher eingegangen wird.

Kompromiß gemacht, das ACK-Bit zusammen mit zwei weiteren Bits zu übertragen. Die beiden zusätzlichen Bits entsprechen dem Status von jeweils einem der drei IR-Sensoren. Der Zustand des ACK-Bits und damit die Qualität der Datenübertragung, ist über die Leuchtdiode D13 ablesbar.[25]

Wegen der Dioden D21 und D22[26] ist der Bereich unterhalb der Mitte des prinzipiell möglichen Wertebereichs unbrauchbar. Durch einen kleinen Trick stört das nicht weiter: Die IR-Daten sind nur dann gültig, wenn das ACK-Bit gesetzt ist. Die untere Hälfte des Wertebereichs würde demnach einem nicht gesetzten ACK-Bit entsprechen, sofern es als MSB codiert ist. In der oberen Hälfte werden zusätzlich die zwei niederwertigen IR-Bits übertragen. Somit werden je nach Bedarf ein bis drei Bit codiert. Aus elektrischer Sicht hat diese Codierung fünf verschiedene Widerstandswerte zur Folge, die jeweils zwischen den A/D-Wandler-Eingang und Masse geschaltet werden. Allerdings muß dabei beachtet werden, daß die Dioden D21, D22 und D24 den zugehörigen Sensorwert beeinflussen. Abbildung 2.4 veranschaulicht den Zusammenhang zwischen Codierung und Widerstandswert.[27]

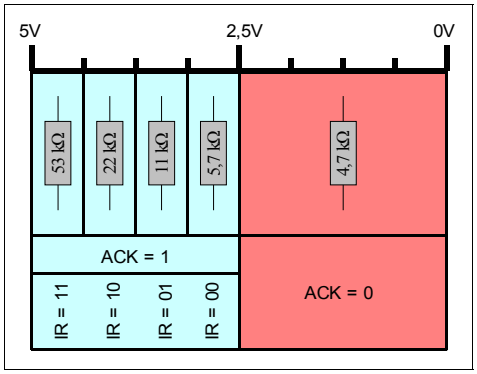


Abbildung 2.4: Zusammenhang zwischen der Spannung am A/D-Wandler, den Widerstandswerten und den gesetzten Bits

Ein 8-zu-1 Analogmultiplexer des Typs ***CD 4051*** dient dazu, die Widerstände auf den Sensoreingang zu schalten [14]. Die Multiplexer-Ansteuerung entspricht dabei genau den beiden IR-Bits und dem ACK-Bit. Da, wie oben beschrieben, die untere Hälfte des Wertebereichs ungenutzt ist, sind für diesen Fall die Multiplexer-Eingänge null bis drei über den Widerstand R8 zusammengefasst und entsprechen damit jeweils dem Sensorwert für ein nicht gesetztes ACK-Bit.

Da sich die Widerstandwerte auf das Massepotential des RCX beziehen, muß die Lepomux-Masse mit der des RCX verbunden werden. Dies sollte allerdings wegen des nötigen Verpolungsschutzes über Dioden geschehen. Die Schottky-Doppeldiode D24 hat genau diese Aufgabe. Dabei erzeugt sie im Vergleich zu einer Siliziumdiode nur eine Potentialdifferenz von ca. 0,3 Volt. Diese Differenz so gering wie möglich zu halten ist nicht nur wegen des Einflusses auf den Sensorwert wichtig. Ein Potentialunterschied von 0,7 Volt, der beispielsweise durch eine Siliziumdiode hervorgerufen werden würde, könnte in einem ungünstigen Fall zu einer negativen Eingangsspannung von -0,7 Volt aus Richtung des RCX führen.[28] Dieses Verhalten

25 Da Acknowledge-Ausgang und die zugehörige Leuchtdiode aus mechanischen Gründen auf unterschiedlichen Leiterplatten untergebracht sind, ist die LED nicht in dem in Abbildung 2.3 gezeigten Schaltplanausschnitt zu finden. Siehe dazu auch Anhang .

26 Diese beiden Dioden dienen dem Verpolungsschutz und sorgen dafür, daß die am Sensoreingang anliegende Spannung nicht an den Komponenten des Acknowledge-Ausgangs anliegt.

27 Die hier verwendeten Widerstandswerte wurden in einem Testaufbau durch Ablesen des tatsächlichen Sensorwertes ermittelt, um eine möglichst lineare Verteilung bezüglich des Wertebereichs zu erhalten.

28 Negative Eingangsspannungen sind gerade bei integrierten Schaltkreisen problematisch, da die Eingangspins dort üblicherweise mit Schutzdioden beschaltet sind. Zu hohe negative Spannungen führen demnach zu einem Stromfluß über die Schutzdiode in Richtung der Bauteil-Masse. Ein zu hoher Strom führt zur Zerstörung der Schutzdiode oder des gesamten Bauteils.

muß gerade in Bezug auf den Sensor-Multiplexer bedacht werden.[29]
Um auch gegen die in Kapitel 1.4.3 angesprochenen Widrigkeiten abgesichert zu sein, sorgt eine weitere Schottky-Doppeldiode (D23) dafür, daß die Versorgungsspannung des Multiplexer-Bausteins in keinem Fall mehr als 0,3 Volt unter dem maximalen Eingangspegel liegt. Im Normalfall wird dieser Baustein allerdings mit 8,6 Volt gespeist, was üblicherweise über der Batteriespannung des RCX liegen dürfte. Da der ***CD 4051*** praktisch immer mit mehr als 3 Volt gespeist wird, ist eine Pegelwandlung aus Richtung des MSP unumgänglich. Diese Aufgabe übernimmt der in Kapitel 2.1.1 angesprochene Pegelwandler ***MC 14504***.

2.1.4 Sensor-Multiplexer

Diese Komponente des Lepomux ermöglicht es, bis zu acht der herkömmlichen LEGO-Sensoren an einem Sensoreingang des RCX zu betreiben. Da dies sowohl für passive, als auch für aktive Sensoren funktionieren soll, muß der RCX-Anschluß quasi transparent zwischen verschiedenen Sensoren umschaltbar sein. Erreicht wird dies mit zwei parallel angesteuerten 8-zu-1 Multiplexern des Typs ***CD 4051*** (U4 und U5 in Abbildung 2.5). Ließe sich die Polung des Steckers voraussagen, würde es ausreichen, den A/D-Wandler-Eingang des Sensoranschlusses zu multiplexen.
Da die beiden Multiplexer unmittelbar als Schalter zwischen den Sensoren und dem

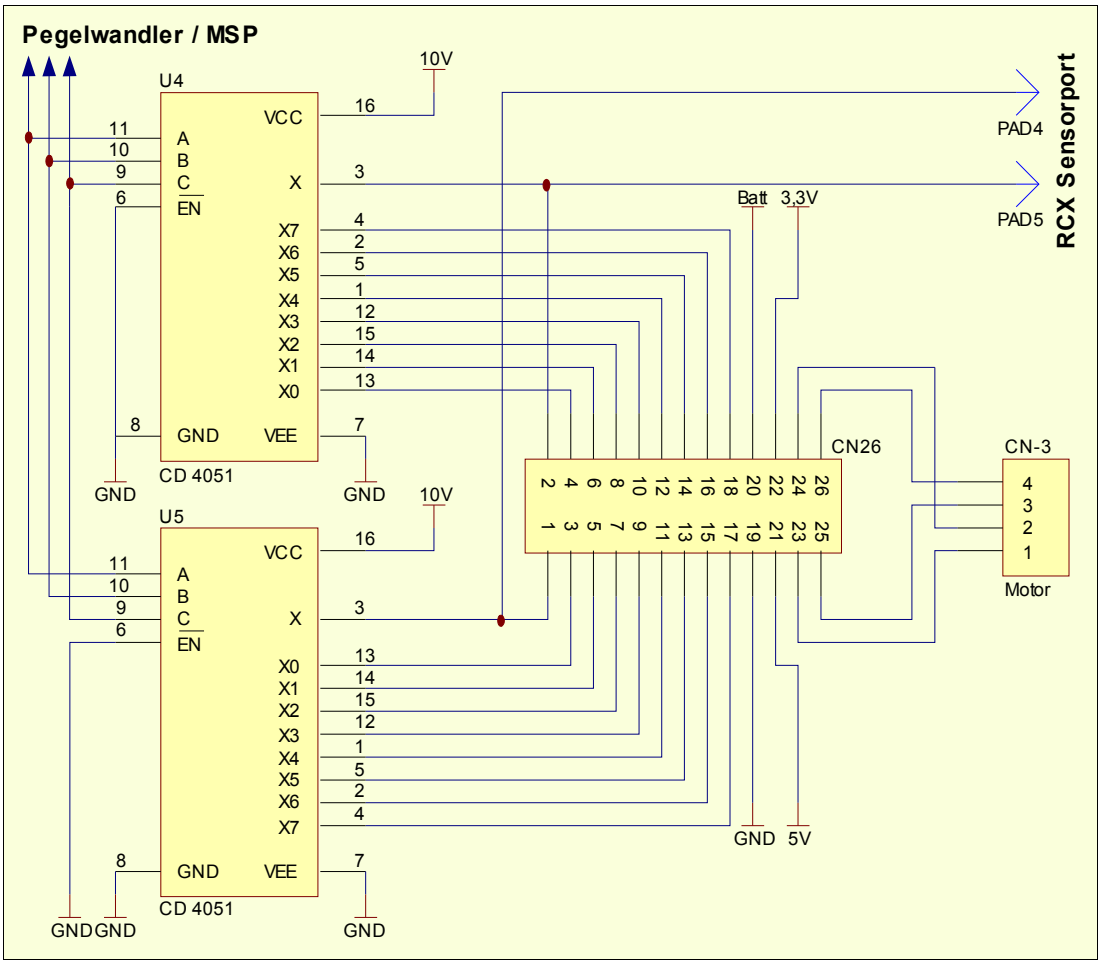


Abbildung 2.5: Schaltung des Sensormultiplexers

29Siehe Kapitel 2.1.4.

RCX-Eingang fungieren, tritt auch hier die in den Kapiteln 1.4.3 und 2.1.3 beschriebene Problematik negativer oder zu hoher Eingangsspannungen relativ zur Versorgungsspannung auf. Wie im Falle des Acknowledge-Ausgangs, sind auch hier die Schottky-Dioden D23 und D24 dafür verantwortlich, die Versorgungsspannung der beiden Multiplexerbausteine innerhalb der Spezifikation zu halten.
Der ON-Widerstand eines Multiplexerkanals liegt laut Datenblatt bei ca. 150 Ohm. Es kommt also für jeden der Angeschlossenen Sensoren ein Widerstandswert von ca. 300 Ohm hinzu, der entsprechend eine leichte Auswirkung auf den RCX-Meßwert hat. Bezüglich der Stromversorgung aktiver Sensoren sollte dieser Widerstand kein Hindernis darstellen, da der Stromfluß RCX-seitig bereits auf etwa 15 Milliampère begrenzt ist. Allerdings sollte bei der Programmierung darauf geachtet werden, daß aktive Sensoren schnell bzw. langsam genug abgfragt werden, da immer nur ein Sensor mit dem RCX verbunden ist. Zu große Abfrageintervalle führen demnach dazu, daß die Kapazität des Pufferkondensators innerhalb des Sensors zu klein ist, um seine Stromversorgung aufrecht zu erhalten.

2.1.5 IR-Beacon-Detektor

Die Beacon-Erkennung dient dazu, zwei Infrarot-Leutchtfeuer mit unterschiedlicher Kennung voneinander unterscheiden zu können (siehe Abbildung 1.7). Damit ist ein Roboter in der Lage, per Leuchtfeuer markierte Orte direkt anzusteuern.

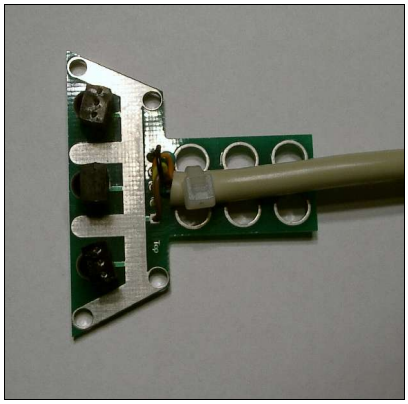


Abbildung 2.6: Platine des Beacon-Sensors „IR Eye“

Die Beacons senden dabei ein Rechtecksignal mit 100 bzw. 125 Hz aus, das auf einen 40 kHz-Trägersignal aufmoduliert ist. Die Filterung des Trägers und Aufbereitung zu einem TTL-Signal übernehmen Standard-IR-Empfänger.
Die Auswahl der erhältlichen IR-Empfänger ist recht groß. Üblicherweise werden diese Bausteine dazu verwendet, IR-Fernbedienungsignale zu empfangen und aufzubereiten. Allerdings eignen sich nicht alle dieser Empfänger für unseren Zweck, da sie nicht zur Datendekodierung, sondern zur Frequenzzählung eingesetzt werden sollen. Ein schlecht empfangenes Signal sollte demnach vom Empfänger ignoriert werden. Bausteine wie beispielsweise der ***TSOP 1840*** von Vishay liefern aber auch bei schlecht empfangenen Signalen einige Flanken, was zu einer falschen Frequenzzählung führen kann und damit auch zur Erkennung eines Beacons, der gar nicht in Blickrichtung steht.[30]
Der Lepomux ist so ausgelegt, daß er drei dieser Empfänger gleichzeitig auswerten

30 In der Praxis zeigte sich, daß die Empfänger SFH 506-40 von Siemens und GP1UD28YK von Sharp deutlich besser geeignet sind als die recht populären Sensoren TSOP 1840 von Vishay[18] [19] [20]
Um das Prolem der fehlerhaften Frequenzzählung durch ein schlecht empfangenes Beacon-Signal zu lösen, bedürfte es einer deutlich aufwendigeren Elektronik innerhalb der Beacons. Naheliegend wäre hier die Verwendung von IR-Transmitter-Bausteinen, die sonst in Fernbedienungen zum Einsatz kommen. Allerdings bietet eine Codierung wie beispielsweise der RC5-Code von Philips (siehe Kapitel 4.2.2) keinerlei Fehlerkorrektur. Ein Ausweg wäre demnach, auch den Beacon mit einem Mikrocontroller auszustatten und damit ein Prüfsummen-basiertes Protokoll auszusenden.
Dies hätte den Vorteil, daß so auch Nutzinformationen zum Roboter übertragen werden könnten. Natürlich ist damit auch der Dekodierungsaufwand ungleich größer.

kann. Damit läßt sich das IR-Auge in die Blickrichtungen "Links", "Mitte" und "Rechts" einteilen, was eine wesentlich bessere Navigation ermöglicht als ein einzelner IR-Sensor. Um die gesamte IR-Sensorik drehbar anbringen zu können, sind die Empfänger zusammen mit drei Abblock-Kondensatoren auf einer separaten Platine untergebracht. Sie wird über ein Kabel, das mit einem 5-poligen Mini-DIN-Stecker ausgestattet ist, mit dem Lepomux verbunden. Das fertige Modul ohne Blickwinkelaufteilung ist in Abbildung 2.6 zu sehen.
Die Ausgänge der IR-Empfänger sind über einen weiteren Pegelwandler des Typs ***MC 14504*** mit dem MSP verbunden, der den 5-Volt-Pegel der IR-Sensoren auf 3 Volt herabsetzt.
Die eigentliche Frequenzauswertung geschieht per Software. Dabei wird die Anzahl fallender Flanken pro Sensor innerhalb eines bestimmten Zeitraums ermittelt. Aus den Zählerwerten läßt sich dann bestimmen, welchen der folgenden vier Zustände ein Sensor gerade hat: <100Hz, 100Hz, 125Hz oder >125Hz.
Der jeweilige Zustand wird dann pro Empfänger als 2-Bit-Wert codiert und läßt sich vom RCX über den Acknowledge-Ausgang abfragen (siehe Abbildung 2.6).[31]

2.1.6 Motortreiber & Leuchtdioden

Um einen Ersatz für den als Datenausgang umfunktionierten Motorport des RCX zu bieten, besitzt der Lepomux zwei Motorausgänge. Hierbei wurden die gleichen Motortreiber[32] verwendet, wie sie auch im RCX zum Einsatz kommen. Per Pulsweitenmodulation läßt sich die Drehgeschwindigkeit der Motoren in 16 Stufen pro Drehrichtung regeln. Sie werden dabei direkt aus der Batteriespannung des Lepomux gespeist und können jeweils bis zu 250mA Strom aufnehmen. Die Gleichrichter REC-2 und REC-3 dienen als Freilaufdioden (siehe Abbildung 2.7).
Zur Steuerung der Motortreiber werden je zwei Portleitungen des MSP verwendet. Da die Treiber TTL-kompatible Eingänge besitzen und damit die Schaltschwelle bei ca. 1,4 Volt liegt, reicht der 3-Volt-Pegel aus, um auch ohne Pegelanpassung einen High-Pegel zu erzeugen.
Mit den beiden Steuerleitungen lassen sich die Motortreiber in einen der vier Zustände *off*, *forward*, *reverse* oder *brake* schalten. Der Zustand *brake* nimmt dabei eine Sonderstellung ein, denn im Vergleich zu *off* sind die beiden Motoranschlüsse hier nicht unbeschaltet, sondern werden über die Masse des Treibers miteinander kurzgeschlossen. Das hat zur Folge, daß ein angeschlossener Motor gebremst wird, denn die beim Drehen des Motors entstehende Energie wirkt durch den Kurzschluß direkt dieser Bewegung entgegen.
Die vier Motorsteuerleitungen sind parallel mit den Kathoden der vier Leuchtdioden D14 bis D17 verbunden, wobei die Vorwiderstände R3 bis R5 für eine Strombegrenzung auf ein Milliampère sorgen.[33] Für normale LEDs wäre dieser Strom

31 Wie diese Auswertung innerhalb der Software geschieht, ist in Kapitel 2.2.1.6 beschrieben.

32 Die verwendeten Motortreiber sind relativ exotische und demnach auch teure Bausteine der Firma Melexis. Leider ist es praktisch unmöglich, einen Treiber zu finden, der ähnliche Eigenschaften zu einem günstigen Preis bietet.
Der MLX 10402 besitzt eine H-Brücken-Struktur und beinhaltet Schutzmechanismen gegen Überhitzung und zu hohen Stromfluß. Die Steuerung erfolgt über zwei TTL-kompatible Eingänge. Wichtig für ein kompaktes Design ist die platzsparende Bauform des SO-16 SMD-Gehäuses.[21]

33 Der Grund für die inverse Leuchtdiodenbeschaltung war die Überlegung, daß die Ausgangspins des Prozessors nur geringe Lasten treiben können und somit nicht die zusätzliche Last der Leuchtdioden in positive Schaltrichtung hinzukommen soll. Siehe dazu Kapitel 2.3.5.

allerdings viel zu gering, um sie zum Leuchten zu bringen. Deshalb kommen hier Low-Current-LEDs zum Einsatz, die bereits bei einem Strom von zwei Milliampère ihre maximale Leuchtkraft entfalten. Die Begrenzung auf 1 mA hilft Strom zu sparen und bietet dabei eine ausreichende Helligkeit.

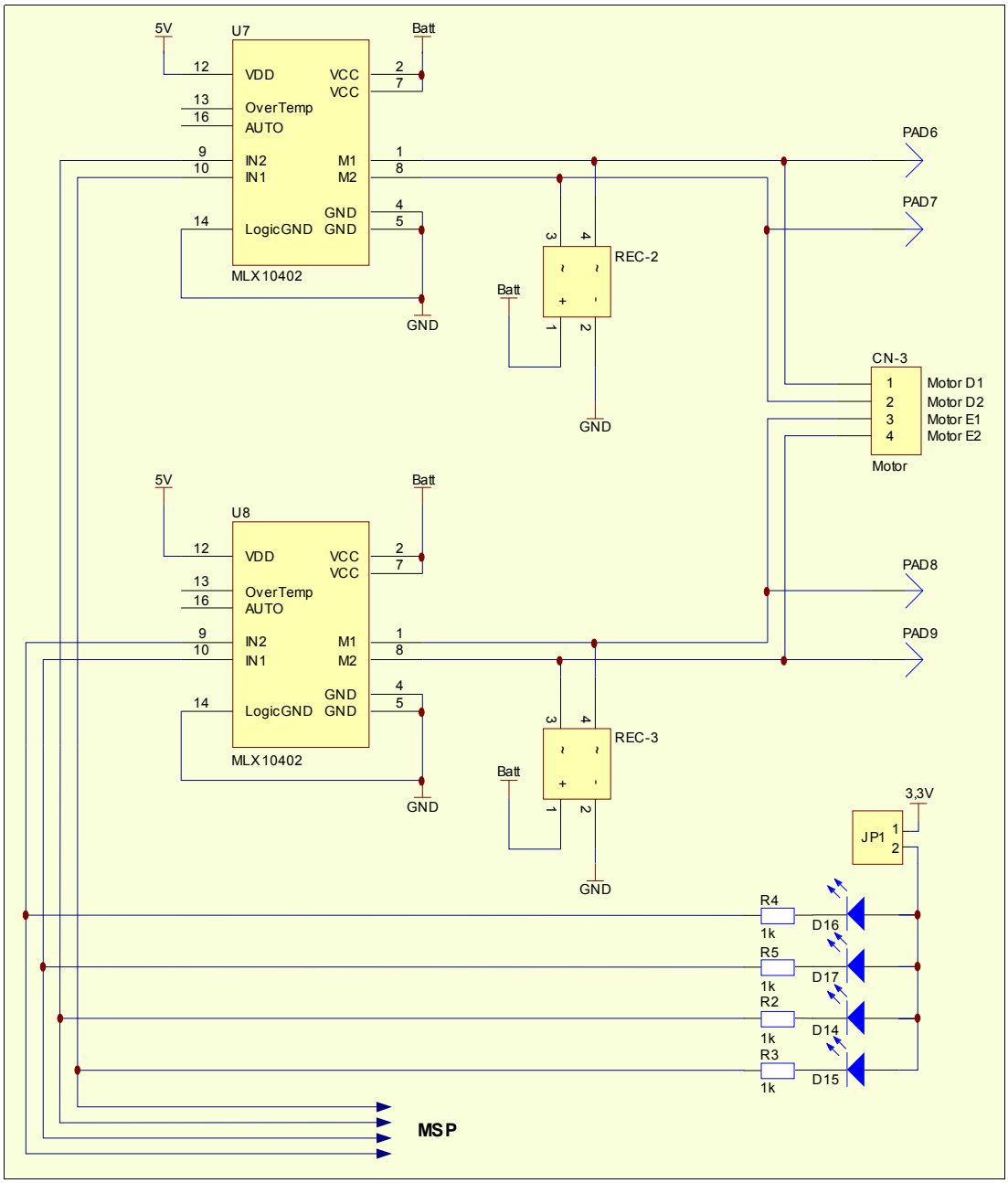


Abbildung 2.7: Die Beschaltung der Motortreiber

Die LEDs vereinen drei Funktionen:

- Sie können als Indikator bei der RCX-Programmierung benutzt werden, da sie leichter ablesbar sind als das LC-Display des RCX. Hierzu sollten allerdings keine Motoren angeschlossen sein.
- Im normalen Motorbetrieb signalisieren sie (invertiert), welcher Motorport gerade in welcher Richtung betrieben wird.
- Im Debug-Modus werden einige Einsprünge in Routinen der Lepomux-Software über die LEDs signalisiert.

Über den Jumper JP1 lassen sich diese vier Leuchtdioden bei Bedarf von der Versorgungsspannung trennen. Da sie aber nur maximal 4 mA aufnehmen, sollte das in den meisten Fällen unnötig sein.

2.1.7 Stromversorgung

Da viele der Bauteile unterschiedliche Versorgungspannungen benötigen, ist die Stromversorgung des Lepomux relativ aufwendig. Die relevanten Schaltungsteile sind in Abbildung 2.8 zusammengefasst:
Die auch zur Speisung der Motoren verwendete Batteriespannung wird über einen Spannungsregler des Typs ***78 L 05*** auf fünf Volt gesenkt. Mit dieser Spannung werden unter anderem die IR-Empfänger betrieben. Über den Spannungsregler U11 des Typs ***LE 33 CZ*** werden daraus 3,3 Volt für den MSP-430 erzeugt.
Die Betriebsspannung der Analogmultiplexer sollte maximal 0,5 Volt unter der zu schaltenden Spannung aus dem RCX-Eingang liegen. Diese Forderung ist nicht leicht zu erfüllen, denn selbst wenn man hier direkt auf die Batteriespannung zurückgreift, kann der Füllstand und damit die Spannung des Lepomux-Batteriepacks niedriger sein als die der RCX-Batterien. Ideal wäre eine Spannung, die in jedem Fall über der liegt, die am RCX-Eingang vorhanden ist. Ein kleiner Kunstgriff verhilft uns dazu: Mit Hilfe einer Ladungspumpe werden die bereits vorhandenen 5 Volt verdoppelt. Dazu bietet sich der Baustein ***LTC 1044***[34] an. Er schaltet mit einer relativ hohen Frequenz den negativen Anschluß des Pump-Kondensators C3 zwischen Masse und 5 Volt hin und her. Ist dieser Anschluß mit Masse verbunden, wird C3 über die Diode D18 auf 5 Volt geladen. Im anderen Fall liegt das Potential des negativen Anschlusses von C3 um 5 Volt höher, was dazu führt, daß sich der Kondensator über die Diode D19 entladen kann und damit den Pufferkondensator C4 auf eine Spannung von ca. 8,6 Volt auflädt. Allerdings werden wegen des Spannungsabfalls an den beiden Dioden D18 und D19 bei diesem Vorgang keine vollen 10 Volt erreicht.[35]

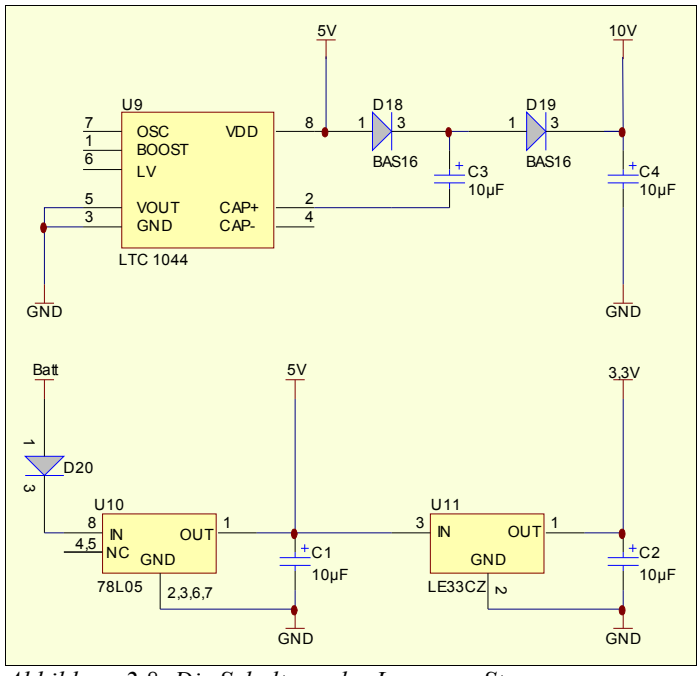


Abbildung 2.8: Die Schaltung der Lepomux-Stromversorgung

Nun kann es dennoch passieren, daß der Lepomux stromlos an einen aktiven RCX-Eingang angeschlossen wird. In diesem Fall sorgen die beiden Schottky-Dioden in

34 Der LTC 1044 ist eine universelle Ladungspumpe, die sich für diverse Zwecke wie zum Beispiel einer Spannungsverdopplung oder auch der Erzeugung einer gleichgroßen negativen Spannung aus einer positiven Eingangsspannung eignet. Der Baustein ist nahezu baugleich zu den Typen ICL 7660 und MAX 7660.[22]

35 Im Schaltplan sind Punkte, an denen 8,6 Volt anliegen fälschlicherweise mit 10 Volt ausgewiesen, da es sich aus logischer Sicht um eine Spannungsverdopplung handelt.

D23 dafür, daß die Multiplexer trotzdem mit einer angemessenen Spannung versorgt werden. In diesem Fall sperrt die Diode D19 und verhindert damit einen Stromfluß in Richtung der inaktiven Ladungspumpe.
Wie schon in Kapitel 2.1.3 angesprochen, sorgt die doppelte Schottky-Diode D24 für ein Massepotential, das dem des RCX entspricht.
Der Stromverbrauch der gesamten Schaltung liegt (ohne angeschlossene Motoren) bei unter 50 mA, wobei der größte Teil davon auf die IR-Empfänger und die Motortreiber entfällt. Der Prozessor und die Multiplexer benötigen nicht mehr als ein Zehntel davon.
Es sollte unbedingt darauf geachtet werden, niemals die Batteriespannung zu verpolen. Die Komponenten hinter dem 5-Volt-Spannungsregler sind zwar durch die Diode D20 geschützt, aber über die Freilaufdioden der Motorports würde ein Kurzschluß entstehen, der im schlimmsten Fall mit dem Tod dieser Dioden quittiert werden würde.

2.1.8 Mechanische Besonderheiten

Als Gehäuse für den Lepomux dient ein LEGO-Batteriekasten für 9-Volt-Blöcke. Nachdem man unnötige Bestandteile daraus entfernt hat, bietet dieser Kasten einen Raum von ca. 60 x 30 x 17 Millimetern - nicht viel, um die relativ komplexe Schaltung unterzubringen.

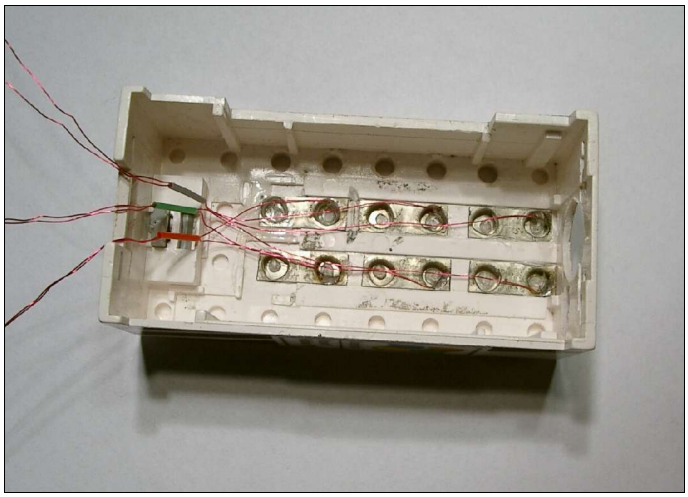


Abbildung 2.9: Umrüstung des Batteriekastens

Das Gehäuse besitzt auf der Oberseite zwei Stromschienen, über die normalerweise die Verbraucher angeschlossen werden. Mit einer Modifikation lassen sie sich zu drei getrennten LEGO-Anschlüssen umwandeln. Dazu wird jede Stromschiene gedrittelt und mit lackisoliertem Draht von der Innenseite des Gehäuses aus kontaktiert (siehe Abbildung 2.9). Diese Drähte werden dann mit Lötpads auf den Platinen verbunden. Extern bilden die Anschlüsse die Verbindung zum RCX (Daten-Eingang, Acknowledge-Ausgang und Sensor-Mux-Ausgang).
Die übrigen Anschlüsse wie zum Beispiel Sensor-Mux-Eingänge, Motor-Ausgänge und die verschiedenen Versorgungsspannungen sind über eine 26-polige Pfostenwanne seitlich aus dem Gehäuse geführt. Zudem gibt es noch eine 5-polige Mini-DIN-Buchse zum Anschluß des IR-Empfänger-Moduls und eine 3,8 mm-Buchse über die ein 7,2-Volt-Batteriepack angeschlossen wird.
Die Schaltung ist auf zwei Platinen verteilt, die über Federleisten huckepack montiert werden. Dabei sind sämtliche SMD-Bauteile, die dicker als 2 mm sind, auf der jeweils ins Gehäuseinnere zeigenden Seite der beiden Leiterplatten untergebracht. Sonst ließe sich die Gesamtdicke von 17 mm nicht einhalten.
Auf dem *Board A* befinden sich der Prozessor, die beiden Sensor-Multiplexer und die Schaltung für den Acknowledge-Ausgang. *Board B* beherbergt die Stromversor-

gung, die Motortreiber, die LEDs und das Dateneingangs-Interface.

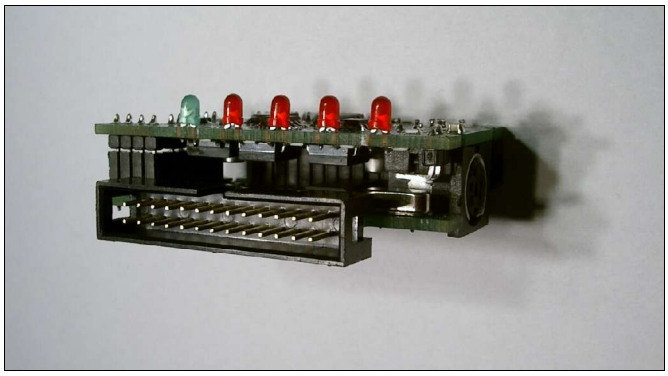


Abbildung 2.10: Platinen in Sandwich-Montage

2.1.9 Programmier-Steckverbinder

Da der MSP direkt auf das *Board A* gelötet wird, wäre es äußerst kurzsichtig gewesen, sich jeglichen Weg für ein Software-Update zu verbauen. Auf der dreipoligen Federleiste *CN-MSP* liegen deshalb einige für die Programmierung unverzichtbare Pins des MSP. Zusammen mit vier Kontakten des *CN-2* und einem Adapterkabel kann der verlötete Prozessor mit Hilfe des Evaluation-Boards programmiert werden.[36]

36 Die genaue Pinbelegung der Platinenverbinder ist dem Anhang und zu entnehmen.

2.2 Software

Damit sich mit dem Lepomux etwas Sinnvolles anfangen läßt, ist eine nicht unerhebliche Menge an Software nötig. Sie läßt sich in drei Kategorien einteilen:

- Die Lepomux-Firmware innerhalb der MCU.
- Der Kernel-Patch bezüglich BrickOS.
- Die eigentliche Benutzer-Software innerhalb des RCX.

2.2.1 Lepomux-Firmware

Um die vom RCX verschickten Daten auszuwerten und die Steuerung des Multiplexers sowie der anderen Komponenten zu realisieren, muß der integrierte Mikrocontroller für diese Aufgaben programmiert werden. Die Software muß natürlich passend für die verwendete MCU geschrieben werden und wegen der zeitkritischen Aufgaben möglichst maschinennah sein. Ein solches Projekt allerdings in Assembler umzusetzen, erschien unnötig komplex.

2.2.1.1 Entwicklungsumgebung

Zur Entwicklung der Lepomux-internen Software wurde die beim MSP-430-Evaluation-Board mitgelieferte Version eines C-Compilers der Firma IAR benutzt. Die darin enthaltene IDE ist eine etwas eingeschränkte Version, die nur eine maximale Codegröße von 2 kB erlaubt. [23]

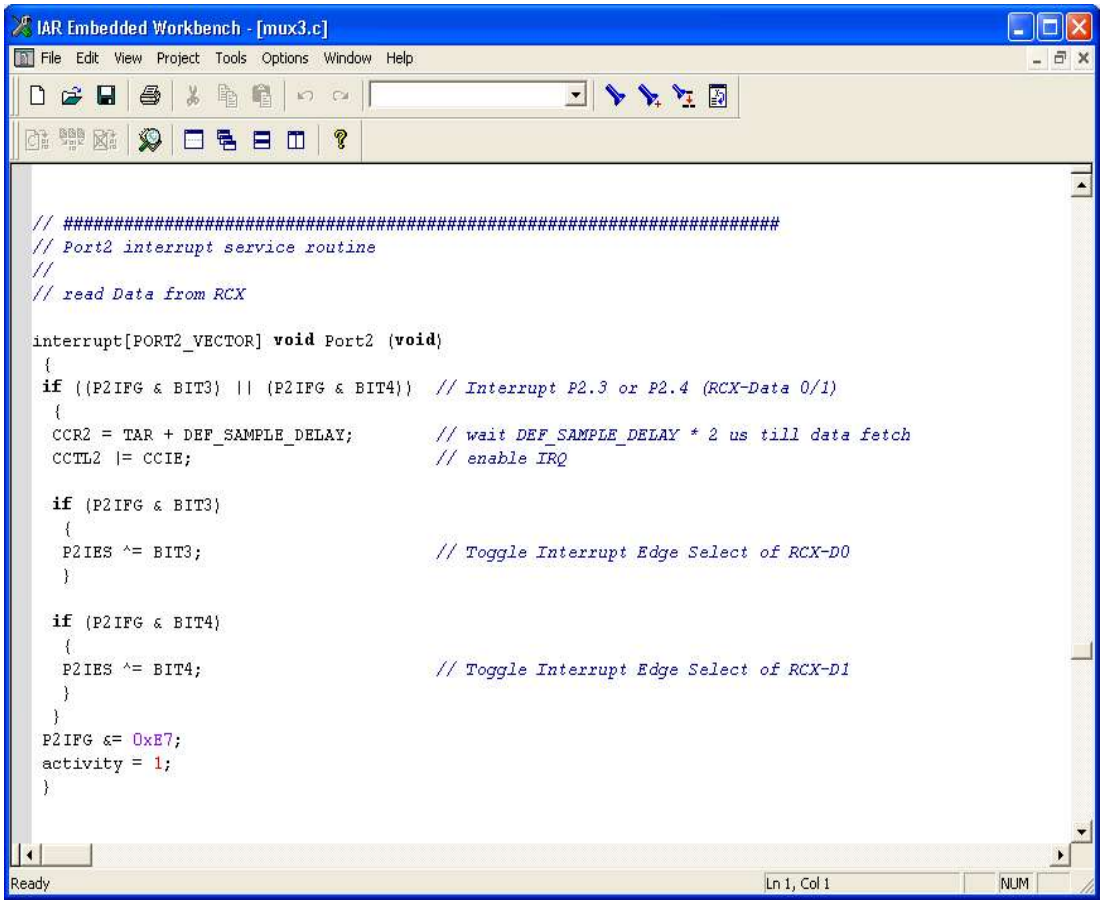


Abbildung 2.11: IAR Enwicklungsumgebung

Die Programmentwicklung gestaltet sich recht angenehm, da ein direktes Debuggen des Quellcodes im Controller unterstützt wird. Allerdings sollte beachtet werden, daß die Prozessor-Anschlüsse P1.4 bis P1.7 für die Kommunikation des Contollers mit dem Debugger (über JTAG) reserviert sind. Funktionen, die auf diese Ports zugreifen, können demnach nicht sinnvoll im Debugger untersucht werden.

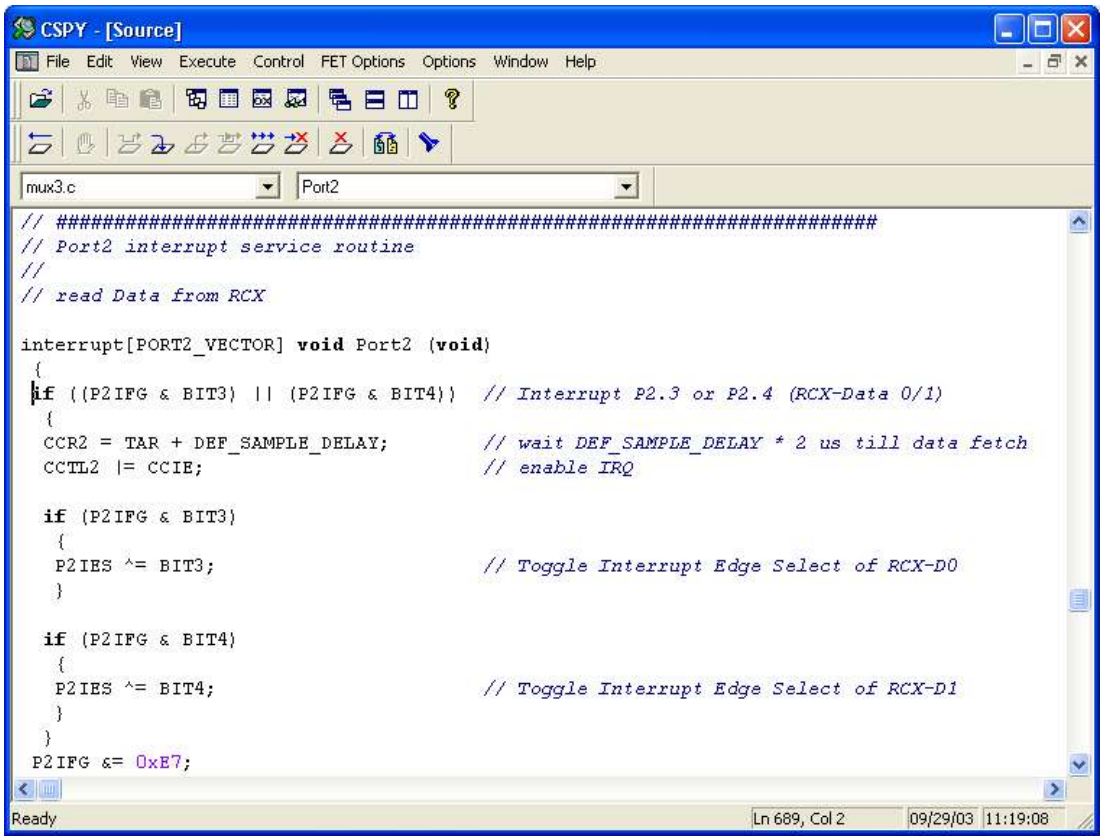


Abbildung 2.12: Debugger

Der Compiler ist speziell auf kleine MCU-Typen abgestimmt und erzeugt somit einen sehr kompakten Code, der sich auch in den zeitkritischen Routinen bewehrt hat. Allerdings sollte man eine gewisse Vorstellung davon haben, wie der Compiler die C-Statements verarbeitet, um ihm überhaupt die Chance zu geben, optimierten Code erzeugen zu können.
Mittlerweile ist auch eine Version des GCC[37] für den MSP-430 erschienen. Diese ist komplett frei und für verschiedene Betriebssysteme erhältlich. [24]

2.2.1.2 MCU-Aufbau

Der verwendete MSP 430 besitzt 14 frei programmierbare Ein-/Ausgänge, die in zwei Ports[38] zusammengefasst sind. Auf Wunsch kann pro Eingang ein Interrupt an einer steigenden oder fallenden Flanke erzeugt werden. Zur Auswertung steht jedem

37 Für den GCC werden andere Include-Dateien gebraucht und einige MSP-spezifische Identifier sind anders benannt. Somit kann man leider nicht ohne Änderungen im Sourcecode von einem Compiler zum anderen wechseln.

38 Port 1 hat die Bezeichner P1.0 bis P1.7 und Port 2 P2.0 bis P2.5. Die beiden Portpins P2.6 und P2.7 sind nicht nach außen geführt, jedoch sind sämtliche Register für diese beiden Pins vorhanden.

Port ein Interruptvektor zur Verfügung. Auf welchem Pin des entsprechenden Ports ein Interrupt generiert wurde, wird über ein Interruptflag (PxIFG)[39] signalisiert.
Außerdem ist ein 16 Bit Timer (TimerA) vorhanden, dem ebenfalls zwei Interruptvektoren zugeordnet sind (TimerA0 und TimerA1). Ein Vektor steht dabei ausschließlich dem Capture/Compare-Block 0 (CCR0) zur Verfügung. Den zweiten Vektor müssen sich der TimerA-Überlauf (TAR) und die beiden anderen vorhandenen Capture/Compare-Blöcke (CCR1 und CCR2) teilen. Die übrigen Interruptvektoren werden in unserer Software nicht verwendent.

2.2.1.3 Interruptverteilung

Es gilt die Interrupts geschickt zu Verteilen, damit die Software reibungslos funktioniert. Zur Verarbeitung des vom RCX benutzten Übertragungsprotokolls benötigt der Lepomux einen Timer, der im Capture-Modus für das Einmessen auf die Präambel und im Compare-Modus für das zeitlich präzise Auslesen der Datenbits betrieben wird.[40] Da diese Auswertung durchaus zeitkritisch ist, wird hierfür der TimerA0 mit der höchsten Interrupt-Priorität verwendet (siehe Abbildung 2.13).

Interrupt Source	*Interrupt Flag*	*Priorität*
TimerA0	CCR0: CCIFG0	9
TimerA1	CCR1: CCIFG1 CCR2: CCIFG2 TAR: TAIFG	8
I/O Port P2	P2IFG.0 bis P2IFG.7	3
I/O Port P1	P1IFG.0 bis P1IFG.7	2

Abbildung 2.13: Tabelle der Interrupt-Prioritäten

Zur Auswertung des IR-Beacon-Detektors müssen die Frequenzen der einzelnen IR-Sensoren gemessen werden.
Eine exakte Frequenzmessung wäre über einen Timer Möglich, der jeweils die Zeit zwischen zwei Flanken mißt. Allerdings würde das pro Sensor einen Capture-Interrupt erfordern. Da aber bereits ein Timer für den Datemempfang benutzt wird und insgesamt nur drei vorhanden sind, macht dieser Ansatz wenig Sinn.
Um die empfangenen Frequenzen zu bestimmen, werden deshalb die fallenden Flanken jedes einzelnen Sensors gezählt. Über die Anzahl der Flanken in einem definierten Zeitraum (TAR-Überlauf) läßt sich dann die Frequenz des empfangenen Signales bestimmen. Zwar läßt diese Methode nur eine grobe Frequenzeinstufung zu, die aber für unsere Zwecke hinreichend genau ist. Wegen der geringen Frequenzen von 100 oder 125 Hz ist die Auswertung über den Port2-Interrupts vollkommen unkritisch.
Die Motoransteuerung erfolgt mittels Pulsweitenmodulation (PWM). Der Takt für die PWM wird mit dem TimerA1 / CCR1 generiert.[41]

2.2.1.4 Prinzipieller Programmaufbau

Die Programmstruktur kennzeichnet sich im Wesentlichen dadurch, daß alle Funktionen, die Ein- oder Ausgaben der MCU verarbeiten, in Interruptroutinen untergebracht sind. Hierdurch ergibt sich ein fast nebenläufiges Verhalten der verschie-

39 Sämtliche Register sind für beide Ports vorhanden und unterscheiden sich in der Präfix des Bezeichners. Px bezieht sich hier immer auf den jeweiligen Port P1 oder P2
40 Siehe Kapitel 1.4.2.
41 Der genaue Aufbau der PWM-Signale wird in Kapitel 2.2.1.7 beschrieben.

denen Programmteile. Innerhalb der Interruptroutinen werden Register oder Flags gesetzt, die dann zeitunabhängig in der main-Routine ausgewertet werden können.

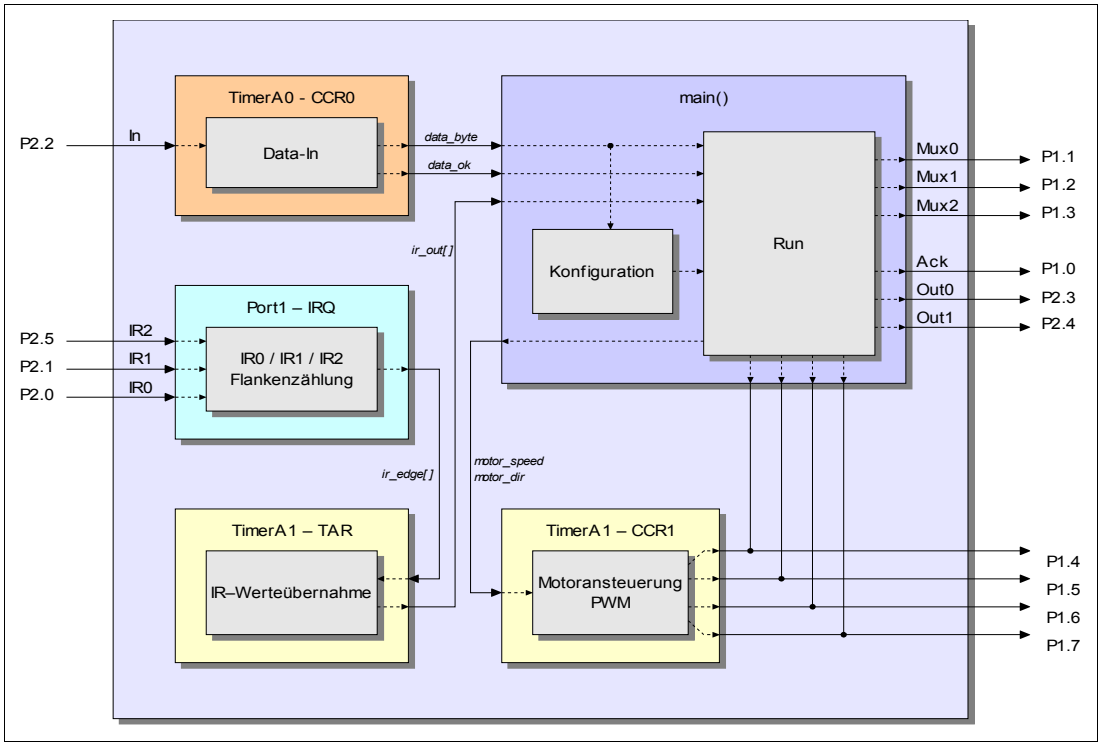


Abbildung 2.14: Blockschaltbild der Software v1.0

2.2.1.5 Datenübertragung

Für das Protokoll (siehe Abbildung 1.19) des Datentransfers vom RCX zum Lepomux wird ein Timer des MSP benötigt. Da der Daten-Empfang relativ zeitkritisch ist, sollte hierfür, wie oben bereits angesprochen, der Capture/Compare-Block mit der höchsten Interrupt-Priorität ausgewählt werden. Trotzdem muß darauf geachtet werden, daß andere Interruptroutinen nicht zu lang werden, da auch ein Interrupt höchster Priorität nicht zur Unterbrechnung einer bereits laufenden Interrupt-Bearbeitung niedrigerer Priorität führt. In diesem Fall wird der Timer A0 verwendet.
Um noch ein wenig mehr Sicherheit in die Übertragung zu bekommen, wurde die Präambel um ein Bit verlängert.
Die Dekodierung des Protokolls (siehe Abbildung 2.15) geschieht folgendermaßen: Mittels eines Capture-Interrupts, der auf beide beide Flanken triggert, wird über eine Länge von neun Halbbits (*t1* bis *t9*) die durchschnittliche Halbbit-Zeit *t* ermittelt. Um eventuell auftretende Fehlmessungen zu eliminieren, werden der minimale und der maximale Wert nicht in die Berechnung mit einbezogen. $T = (t1 + ... + t9 - tmin - tmax) / 7$ Nachdem t berechnet wurde, wird im Compare-Modus ein Interrupt in der Mitte der Halbbits 11 und 12 mit $T1 = (t / 4) * 3$[42] erzeugt und gemessen, ob tatsächlich eine gültige Präambel vorliegt. Sofern das der Fall ist, erfolgt das Sampling der weiteren Bits nach der berechneten Bitzeit per Interrupt im Compare-Modus mit $T2 = (t / 2) * 3$ und $T3 = 2 * t$. Anderenfalls wird die aktuelle Übertragung sofort im Lepomux verworfen.

42 Da die Berechnung von t erst nach der Messung von t9 erfolgt und Rechenzeit erfordert, wird T1 erst während t10 gesetzt. Der Faktor zur Berechnung von T1 aud t wurde durch eine Messung ermittelt.

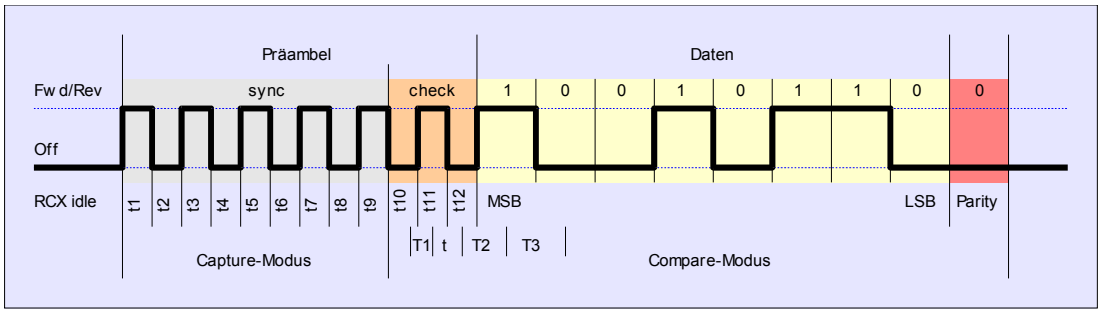


Abbildung 2.15: Protokoll der Datenübertragung

Mit jedem empfangenen Bit wird automatisch das für die bisher übertragenen Daten aktuell gültige Parity-Bit per XOR erzeugt. Nachdem neun Bit eingelesen wurden, wird das errechnete Parity-Bit mit dem übertragenen Parity-Bit verglichen und damit der korrekte Empfang des Datenworts überprüft.
Bei der Datenübertragung vom Lepomux zum RCX (siehe Abbildung 2.4) entspricht das MSB einem Acknowledge-Flag, das eine korrekte Datenübertragung vom RCX zum Lepomux signalisiert. Die unteren beiden Bits werden dem IR-Register[43] entnommen. Je nach adressiertem IR-Empfänger wird damit die zugehörige IR-Sensorinformation zum RCX gesendet.

2.2.1.6 IR-Detektor

Die Auswertung der IR-Empfänger entspricht einer Frequenzmessung mit drei Schaltschwellen zur Kategorisierung. Prinzipiell werden dabei fallende Flanken über einen festgelegten Zeitraum hinweg gezählt. Jede negative Flanke erzeugt einen Port-Interrupt, der den Flankenzähler für den jeweiligen IR-Empfänger hochzählt. Beim Überlauf des 16-Bit-Timers TAR[44] wird ein weiterer Interrupt erzeugt, in dem anhand der Flankenzählerstände eine Frequenzberechnung stattfindet. Dabei gibt es drei Grenzwerte, die entscheiden, ob es sich um 100 Hz, 125 Hz oder einen Wert darüber oder darunter handelt.

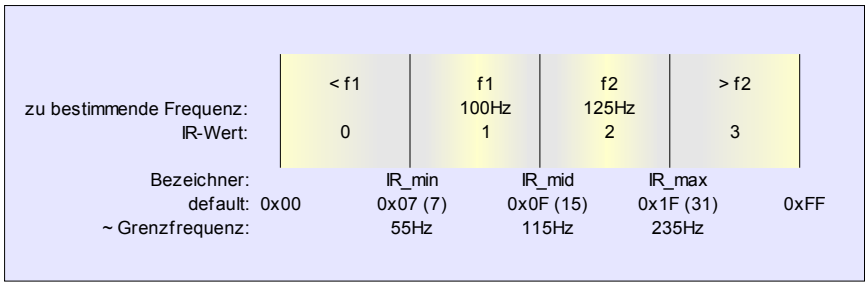


Abbildung 2.16: Grenzen zur Kategorisierung der Frequenzen

Die so ermittelte Frequenzkategorie wird als 2-Bit-Wert im IR-Register abgelegt. Für jeden Empfänger wird wegen der einfachen Handhabung ein eingenes Register verwendet, obwohl die Werte auch alle in einziges Byte passen würden. Die Default-Werte sind in Abbildung 2.16 zu sehen.

2.2.1.7 Pulsweitenmodulation

Um auf den beiden Motorausgängen eine Geschwindigkeitsregelung zu ermöglichen, wird statt eines konstanten Logikpegels ein PWM-Signal an die Motortreiber geschickt. Die Pulsweite kann dabei in 16 Stufen über die Motor-Speed-Register einge-

43 Siehe Kapitel 2.2.1.6.
44 Der Timer-Überlauf findet bei einer Taktung von 500kHz alle 131,072ms statt.

stellt werden. Alle 3 Millisekunden (siehe 2.2.1.9.c) wird ein Compare-Interrupt auf Timer A1 erzeugt. Ein PWM-Zyklus besteht aus 16 dieser Interrupts und hat damit eine Länge von ca. 50 ms.
Bei jedem Sprung in die Interrupt-Routine wird über den Zykluszählerstand (er zählt die 16 Teilschritte des PWM-Zyklus) das auszugebende Bit des Motor-Speed-Registers per Bitshifting gewählt und zum Motortreiber geschrieben. Damit das funktionieren kann, werden die vom RCX empfangenen Geschwindigkeitsdaten so aufbereitet, daß je nach Wert bis zu fünfzehn Bits im Motor-Speed-Register gesetzt sind.
Für den Fall, daß eine andere PWM-Charakteristik gewünscht ist, braucht so die eigentliche PWM-Routine nicht verändert zu werden. Die empfangenen Geschwindigkeitsdaten müssen dazu nur als passende Bitfolge in die Motor-Speed-Register übertragen werden. Standardmäßig entspricht ein Geschwindigkeitswert von *n* einem Motor-Speed-Registerinhalt von *n* hintereinander gesetzten Bits.

2.2.1.8 Adressierung der Komponenten

Ein vom RCX empfangenes Datenwort besteht aus einer vier Bit Adresse (MSB) und vier Bit Daten (LSB). Die Adressen werden zur Selektierung verschiedener Komponenten (Subdevices) benutzt. Im folgendenden werden die Module/Register wie in Abbildung 2.18 zu sehen einzeln besprochen.

2.2.1.8.a Betriebsmodus

Port1	*Run-Modus*	*Debug-Modus*
P1.4	Motor D forward	Main-Loop
P1.5	Motor D reverse	IR-Zähler (Port 2 Interrupt)
P1.6	Motor E forward	TAR-Overflow / PWM (TimerA1 Interrupt)
P1.7	Motor E reverse	RCX-Data-IN (TimerA0 Interrupt)

Abbildung 2.17: Motor / LED-Ausgabe

Die Adresse 0x00 wird zur Einstellung des Lepomux-Betriebsmodus benutzt.
Eine 1 in Bit 0 schaltet den Lepomux in den Debug-Modus und signalisiert die Interrupt-Einsprünge auf den Debug-LEDs. Dabei signalisieren die LEDs den Einsprung in die in Abbildung 2.17 aufgeführten Routinen.
Wenn das Bit 1 gesetzt ist, wird ein Soft-Reset ausgeführt. Die Bits 2 und 3 werden nicht weiter verwendet.

2.2.1.8.b Multiplexer

Über die Adresse 0x10 wird der Multiplexer für den RCX-Sensoreingang gesteuert. Der Datenwert entspricht dem augewählten Sensor. Da der MSP-430 mit nur relativ wenigen I/O-Pins ausgestattet ist, lassen sich nur acht Sensoren adressieren. Prinzipiell ist es aber möglich, gleichzeitig mehrere Multiplexermodule am MSP-430 zu betreiben, die verschiedene Eingänge des RCX versorgen.

2.2.1.8.c IR-Sensor-Auswahl

Um den Sensorwert eines der drei IR-Empfänger auf den Acknowledgement-Ausgang zu schalten, wird auf Adresse 0x20 ein Datenwert von null bis zwei ge-

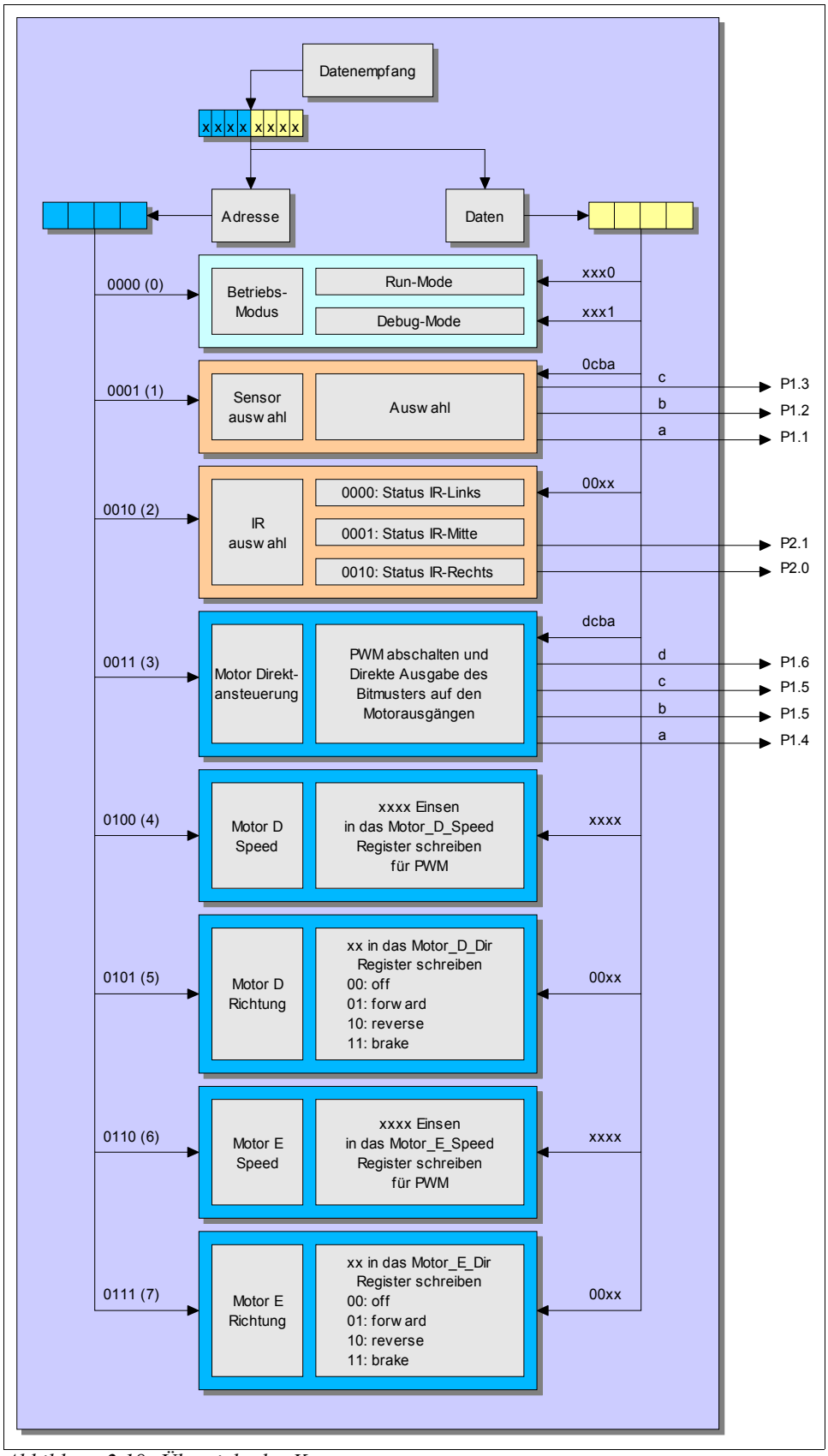


Abbildung 2.18: Übersicht der Komponentenansteuerung

schrieben. Die Daten liegen auch dann noch auf dem Acknowledge-Ausgang an, wenn ein anderes Subdevice selektiert wird. Dabei ist allerdings zu beachten, daß der IR-Wert nur zum Zeitpunkt der Adressierung aus dem IR-Register ausgelesen wird. Eine zwischenzeitliche Frequenzänderung ist somit nicht ohne Neuadressierung sichtbar.

2.2.1.8.d Direkte Ansteuerung der Ausgangskanäle

Ein an die Adresse 0x30 gesendetes Datenwort wird direkt auf die vier Steuerleitungen des Motortreiber ausgegeben. Damit ist der invertierte Datenwert sofort auf den parallel dazu angeschlossenen Leuchtdioden abzulesen.
Wenn Motoren oder Leuchtdioden über diese Adresse angesteuert wurden, ist die übrige Motorsteuerung deaktiviert, bis Motor D oder E[45] wieder ein normales Steuersignal erhalten.

2.2.1.8.e Motor-Steuerung / PWM

Die Motoren D und E werden über vier Adressen gesteuert. Für Motor-D läßt sich über Adresse 0x40 das Motor-Speed-Register mit Werten von 0 bis 15 setzen, für Motor E gilt hier die Adresse 0x60. Dabei wird je ein 16 Bit Register mit der entsprechenden Anzahl von Einsen gefüllt, die dann wie in Kapitel 2.2.1.7 beschrieben, ausgegeben werden.
Für beide Motoren kann die Drehrichtung getrennt über die Adressen 0x50 und 0x70 festgelegt werden. Dabei entsprechen die Einstellungen *off*, *forward*, *reverse* und *brake* den Werten 0 bis 3.

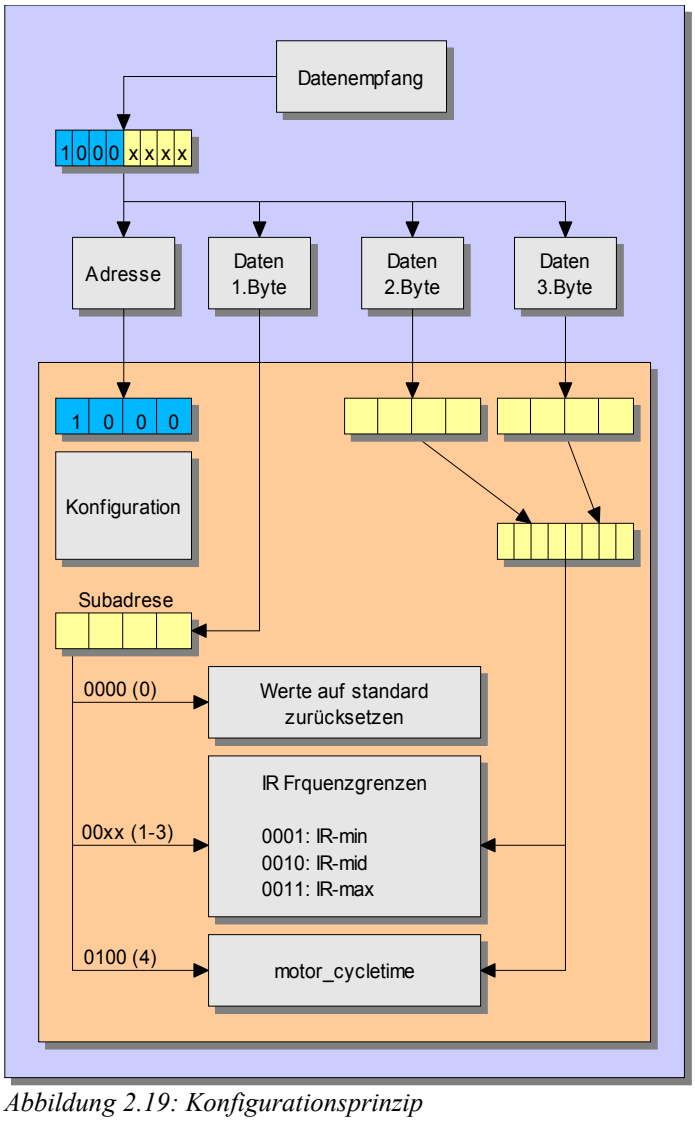


Abbildung 2.19: Konfigurationsprinzip

2.2.1.9 Konfigurationsmöglichkeiten

Einige timingspezifische Werte des Lepomux lassen sich im Betrieb mit Hilfe von Konfigurationsregistern ändern. Da die dort benötig-

45 Die Benennung D und E bezieht sich auf die drei Motorports A bis C, die vom RCX zur Verfügung gestellt werden.

ten Werte üblicherweise länger als vier Bit sind, werden Config-Befehle immer als eine Folge von drei Werten auf die selbe Adresse geschrieben. Dabei enthält das erste Nibble[46] die Subadresse des Config-Registers und die beiden folgenden Nibbles das Datenbyte des Registers (MSB, LSB). Abbildung 2.19 zeigt eine Übersicht der Register die konfiguriert werden können.

2.2.1.9.a Default-Einstellungen

Über Subadresse 0x00 lassen sich die Lepomux-internen Timerwerte wieder auf die Standardeinstellung zurücksetzen. Die übermittelten Daten des zweiten und dritten Nibbles sind dabei irrelevant.

2.2.1.9.b IR-Frequenzgrenzen

Um die Frequenzerkennung der IR-Empfänger möglichst flexibel zu halten, gibt es drei Config-Register, über die sich die Zählergrenzen festlegen lassen. Wie schon in 2.2.1.6 angesprochen, erfolgt die Frequenzbestimmung durch Flankenzählung innerhalb der Umlaufzeit des TAR-Counters. Da der TAR mit einem Takt von 500 kHz betrieben wird, ergibt sich alle 131 Millisekunden ein Überlauf.
Für 100 Hz sind die beiden Grenzen standardmäßig auf 7 bzw. 15 Flanken gesetzt. Dabei zählt die obere Grenze schon zum nächsten Frequenzbereich (siehe Abbildung 2.16).
Bei selbstdefinierten Bereichen sollte die Obergrenze sollte so gewählt werden, daß sie nur knapp über der maximal erreichbaren Flankenzahl liegt. Schließlich ist es wesentlich wahrscheinlicher, daß zu wenig Flanken „gesehen“ werden, als daß mehr auftreten, als vom Sender her möglich sind.
Die Grenzwerte können im Bereich von 0 bis 255 definiert werden, wobei es dem Programmierer überlassen bleibt, für sinnvolle Wertkombinationen zu sorgen. Die Subadressen 1 bis 3 entsprechen dabei den Grenzen *min*, *mid* und *max*, die die Frequenzbereiche aufteilen.

2.2.1.9.c PWM-Teilzykluszeit

Auch auf das Verhalten des PWM-Generators kann während des Betriebs Einfluß genommen werden. Hierbei läßt sich die Zykluszeit in gewissen Grenzen über die Subadresse 4 einstellen.
Da die PWM-Routine über einen Compare-Interrupt des Timers A1 aufgerufen wird, hängt auch hier das Timing vom TAR ab. Prinzipiell sind somit Zykluszeiten von 2 Mikrosekunden bis 131 Millisekunden möglich.
Der Wert für das Config-Register repräsentiert das obere Byte des Timer-Compare-Registers, wobei das untere Byte immer den Wert 255 enthält. Daraus ergeben sich Werte zwischen 255 und 65535 in 256er Schritten, die Zeiten von 510 Mikrosekunden bis 131 Millisekunden entsprechen (255 * 2 µs = 510 µs).
Die Standardeinstellung liegt bei 1536 (= 3,072 ms).
Bei sehr kleinen Werten kann es zu Timingproblemen bei der Datenübertragung oder der IR-Auswertung kommen.

46 Ein Nibble entspicht 4 Bit, also einem halben Byte.

2.2.2 Kernel-Patch

Der Lepomux ist speziell auf die Verwendung im Zusammenhang mit dem RCX-Betriebssystem BrickOS entwickelt worden.

BrickOS wird mit einem „Systemtakt“ von einem Kilohertz betrieben. Das bedeutet, daß einmal pro Millisekunde der Kernel innerhalb seines Systeminterrupts sämtliche Komponenten des RCX mit neuen Daten versorgt bzw. Meßwerte von den Sensoreingängen abfragt. Dazu zählen unter anderem auch der Display-Refresh, die Wiedergabe von Tönen und das Task-Scheduling.

Versucht man nun, einen Motorausgang zur Datenübertragung zweckzuentfremden, stellt sich schnell heraus, daß einem solch ein Multitasking-Kernel das Leben nicht gerade leicht macht.
Deutlich wird das bei der Ausgabe eines Signals, das bei jedem System-Interrupt (also jede Millisekunde) den Pegel wechselt. Dies funktioniert ansich gut, aber sowie mehrere Tasks benutzt werden oder das System andere zeitraubende Dinge zu erledigen hat, kommt es häufig vor, daß das Signal für bis zu zwei Millisekunden den Pegel beibehält. Für unser Protokoll hat das eine Menge an Übertragungsfehlern zur Folge.
Um zu verstehen, woher die Timingprobleme kommen, bedarf es einer genauen Untersuchung des BrickOS-Kernels.
Obwohl der H8-Prozessor des RCX eine ganze Reihe von Timern bietet, benutzt BrickOS davon nur den ORCA[47]. Dies ist ein 16-Bit-Timer, der so programmiert wird, daß er jede Millisekunde einen Interrupt auslöst.
Innerhalb dieser Interrupt-Routine kümmert sich BrickOS um die Systemresourcen. Neben dem Task-Scheduling erfolgt hier unter anderem das Auslesen der Sensoren und Schalten der Motorausgänge. Zur Sound-Ausgabe wird eine ROM-interne Routine benutzt, die auch innerhalb dieses Interrupts aufgerufen wird.
Da der Kernel verschiedene Sensortypen direkt unterstützt und die Sensorwerte gleich passend umrechnet, kann das Zeitverhalten des System-Interrupts stark variieren.
Zur Ansteuerung der Motoren berechnet BrickOS eine Pulsweitenmodulation anhand eines 8-Bit-Zählerwertes. Auf diesen Zähler wird jeweils der aktuelle Speed-Wert addiert. Bei einem Überlauf wird der zugehörige Motor eingeschaltet. Selbst bei einem Speed-Wert von 255 bedeutet das aber nicht, daß der Motor permanent eingeschaltet ist.[48]
Weder die Motoransteuerung, noch die Struktur des Systeminterrupts gestatten eine zeitkritische Datenausgabe über den Motorausgang. Um dennoch die Ausgabe des oben beschriebenen Protokolls zu ermöglichen, ist es nötig, den Kernel zu modifizieren.

47 Der H8 besitzt zwei Timer-Sektionen. Eine davon beherbergt vier 8-Bit-Timer (hier nicht näher erläutert) und die andere einen 16-Bit-Free-Running-Counter (FRC), an den Output-Compare- (OC) und Input-Capture-Register (IC) angelagert sind. Es existieren zwei OC-Register (OCRA und OCRB), die einen Interrupt auslösen können, sofern ihr Inhalt mit dem des FRC übereinstimmt.

48 Ist der aktuelle Wert des Counters gleich null, wird selbst bei einer Addition des maximalen Speed-Wertes von 255 kein Überlauf generiert. Der Motorausgang wird also trotz der maximalen Geschwindigkeitseinstellung für eine Millisekunde ausgeschaltet. Dieses Fehlverhalten ist in einer aktuelleren Version von BrickOS behoben worden (Kernel 0.2.6.10)

2.2.2.1 Anpassung des Interrupt-Systems

Zur Entkopplung der Motoransteuerung von den restlichen Aufgaben des Systeminterrupts, empfiehlt es sich, einen weiteren Interrupt mit höherer Priorität zu benutzen. Leider hat der OCRA-Interrupt bereits die höchste Priorität. Demnach würde eine Motoransteuerung über OCRB trotzdem vom Systeminterrupt blockiert werden können. Aus diesem Grund muß der Systeminterrupt auf OCRB verlegt werden.[49]
Nachdem OCRA nun ausschließlich zur Motoransteuerung verwendet wird, verbessert sich das Zeitverhalten deutlich. Nun würde es sich eigentlich anbieten, den ORCA-Interrupt öfter als den System-Interrupt ausführen zu lassen, um damit die Datenübertragung wesentlich zu beschleunigen. Leider ist das nur dann möglich, wenn man auf die Sound-Ausgabe verzeichtet, denn die ROM-Routinen, die dafür zuständig sind, wurden fest mit dem OCRA-Interrupt verdrahtet.

2.2.2.2 Kernelerweiterung

Um innerhalb des Motor-Interrupts tatsächlich Daten ausgeben zu können, muß die zugehörige Routine entsprechend umgeschrieben werden.
Die schon vorhandenen Funktionen zum Setzen von Drehrichtung und Geschwindigkeit werden für den Datentransport zweckentfremdet. Mit einem fünften Motor-State *data* läßt sich Motorausgang B in den Übertragungsmodus schalten. Der Speed-Wert repräsentiert das zu versendende Datenbyte. Sowie dieser Wert größer als null ist, beginnt der Kernel automatisch mit dem Versand der Daten. Der PWM-Counter dient in diesem Fall als Bitzähler. Nach einer kompletten Übertragung wird der Speed-Wert wieder auf null gesetzt.[50]
Da der Motor-Interrupt einmal pro Millisekunde erfolgt, ergibt sich daraus eine Bitzeit von zwei Millisekunden, denn innerhalb der Präambel wird ein Flankenwechsel pro Halbbit benötigt. Insgesamt dauert damit die Übertragung eines Datenworts 28 Millisekunden. Zusätzlich müssen hier aber noch einige Millisekunden für das Warten auf das Acknowledge-Signal eingerechnet werden, sodaß man im Endeffekt eine Übertragungsleistung von ca. 30 Byte/s erreicht.

49 Die nötigen Modifikationen werden in der Datei systime.c des BrickOS-Kernels vorgenommen. Dort wird ein weiterer Interrupthandler für OCRB implementiert, der die bisherige Systeminterrupt-Routine aufruft. Aus dieser Routine wird der Aufruf der Motoransteuerung auskommentiert und in einen getrennten Interrupthandler verlegt.

50 Die Änderungen betreffen den Assembler-Code innerhalb der Dateien dmotor.c und dmotor.h des Kernels. Um den Kernel nicht unnötig aufzublähen, wird die Datenausgabe ausschließlich für Motor B implementiert.

2.2.3 RCX-Software

Die dritte Kategorie von Software entspricht RCX-Programmen, die die oben beschriebenen Kernel-Funktionen zur Steuerung des RCX nutzen.

Das in Abbildung 2.20 abgedruckte Codefragment demonstriert die Benutzung dieser Funktionen. Die grundsätzliche Vorgehensweise ist dabei das Umschalten des Motorports B in den Datenmodus mit Hilfe des Aufrufs *motor_b_dir(data);*. Danach können über die Funktion *motor_b_speed();* Datenwerte an den Lepomux ausgegeben werden. Der BrickOS-Kernel kümmert sich dabei zwar um den kompletten Versand der Daten, das Prüfen des Acknowledge-Signals bleibt aber dem Programmierer überlassen.

```
#include <conio.h>
#include <unistd.h>

#include <dsensor.h>
#include <dsound.h>
#include <dmotor.h>

void senddata(char data)
    {
    // send until we receive ACK
    do  {
        motor_b_speed(data);
        msleep(50);
        }
    while ((SENSOR_1>>6) < 0x1d0);
    }

int main(int argc, char *argv[])
    {
    int i;

    // switch Motor B to data mode
    motor_b_dir(data);

    ds_init();
    // Ack-Sensor
    ds_active(&SENSOR_1);

    // reset Lepomux
    senddata(2);
    getchar();

    while(1)
        {
        // Motor_D_Dir = fwd
        senddata(0x01 | 0x50);
        for (i=0; i<15; i++)
            {
            // Motor_D_Speed = i
            senddata(i | 0x40);
            msleep(50);
            }
        }
    }
```

Abbildung 2.20: Codebeispiel für die Datenübertragung zum Lepomux

2.3 Evaluierung der Machbarkeitsstudie

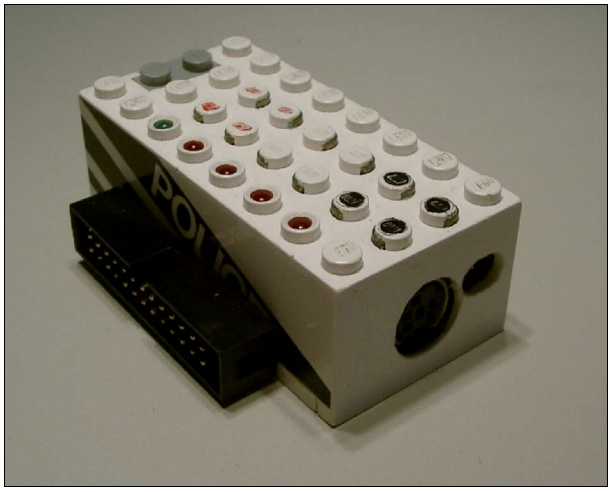


Abbildung 2.21: Lepomux v1.0

Mit der oben beschriebenen Entwicklungsstufe konnte sichergestellt werden, daß sich die theoretischen Vorüberlegungen aus Kapitel 1.4 auch in die Praxis umsetzen lassen. Besonderes Augenmerk lag dabei auf dem Platzbedarf der Schaltung. Wären hier nicht SMD-Bauteile verwendet worden, hätte der Lepomux soviel Platz eingenommen, daß er für sein typisches Einsatzgebiet auf mobilen Robotern uninteressant gewesen wäre. Im Vergleich zu den bekannten RCX-Erweiterungen ist die hier beschriebene Schaltung nämlich deutlich aufwändiger. Die Abbildung 2.21 zeigt das fertige Erweiterungsmodul im LEGO-Gehäuse.

Die ersten Tests zeigten, daß die Schaltung fehlerfrei funktioniert, aber der Sensor-Multiplexer unsinnige Meßwerte zurückliefert.[51] Auch die Implementierung des in Kapitel 1.4.2 beschriebenen Protokolls lieferte nicht ganz die erhofften Ergebnisse. Selbst mit den Kernel-Modifikationen aus Kapitel 2.2.2 war das Datensignal des RCX nicht komplett von Spikes zu befreien. Eine zusätzliche Prüfung des letzten Bits der Präambel (siehe Abbildung 2.15) konnte hier aber für Abhilfe sorgen. Insgesamt ist die Übertragungsleistung aber eher unbefriedigend, da relativ oft Datenpakete nach einem Übertragungsfehler erneut gesendet werden müssen.

Die folgenden Kapiteln befassen sich näher mit den Unvollkommenheiten dieser Machbarkeitsstudie. Einige der dort angesprochenen Punkte erfordern tiefgreifende Veränderungen sowohl der Hard- als auch der Software. Die Umsetzung dieser Änderungen soll den in Kapitel 3 beschriebenen Prototypen des Lepomux praxistauglich machen.

2.3.1 Multiplexer-Innenwiderstand

Beim Testen des Sensor-Multiplexers stellte sich recht schnell heraus, daß dieser keineswegs transparent arbeitet. Aktive Sensoren bekamen nicht genug Strom und auch das Verhalten passiver Sensoren war nicht wie gewohnt, da deren Wertebereich stark verfälscht wurde.

Nach einer Messung des On-Widerstandes der verwendeten Multiplexerbausteine ***CD 4051***, war dieses Verhalten leicht erklärbar. Der Widerstand lag um Größenordnungen über den im Datenblatt verzeichneten 150 Ohm.

Glücklicherweise wurde der schon recht betagte ***CD 4051*** inzwischen weiterentwickelt und in die 74-HC-Logikfamilie eingegliedert.[52] Unter der Bezeichnung ***74 HC***

51 Die Analyse dieses Fehlverhaltens ist in Kapitel 2.3.1 dokumentiert.

52 Auch die Firma Maxim bietet eine Weiterentwicklung dieses Multiplexers an. Allerdings ist der MAX 4051 nicht pinkompatibel zum CD 4051 und auch eher für das möglichst verlustfreie Schalten von NF-Quellen konzipiert.

4051 ist ein pinkompatibler Bruder des ***CD 4051*** zu finden, der aber deutlich bessere elektrische Eigenschaften hat. Im Gegensatz zur CD-Familie kann die 74-HC-Familie nur mit Versorgungsspannungen zwischen 3 und 10 Volt betrieben werden, was für unseren Anwendungsfall aber vollkommen ausreichend ist.

Ein Testaufbau mit zwei Analog-Multiplexern des Typs ***74 HC 4051*** ergab das gewünschte transparente Verhalten.

2.3.2 RCX-seitige Sensorunterstützung

Jeder Sensor, der sich am RCX betreiben läßt, besitzt einen für ihn typischen Wertebereich. Dabei ist weder ein lineares Verhalten der Meßwerte, noch eine Ausnutzung des A/D-Wandler-Wertebereichs sichergestellt. Da ein benutzerseitiges Normieren der Meßwerte für die oft benutzten Standardsensoren lästig wäre, wird diese Aufgabe von den meisten RCX-Betriebssystemen übernommen. Pro Sensoreingang teilt die Benutzer-Software dem Kernel mit, welcher Sensortyp verwendet wird.

Besonders nützlich ist dieses Verhalten bei der Verwendung von Shaft-Encodern. Dabei protokolliert das Betriebssystem Drehimpulse und berechnet daraus einen fortlaufenden Zählerwert, der je nach Drehrichtung zu- oder abnimmt.

Betreibt man aber einen Sensor-Multiplexer an einem der Eingänge, so kann das Betriebssystem nicht mehr wissen, welche Normierungsfunktion beim Abfragen des Meßwertes angewendet werden muß. Selbst wenn nur Sensoren des gleichen Typs angeschlossen sind, kann das zu Verwirrungen führen. Die Drehimpulse von Shaft-Encodern könnten beispielsweise nicht zugeordnet werden. Verschiedene dieser Sensoren würden also einen gemeinsamen Zählerstand beeinflussen.

Bei der Benutzung von Sensor-Multiplexern wie dem Lepomux muß deshalb der Programmierer selbst darauf achten, die zurückgelieferten Meßwerte richtig zuzuordnen und gegebenenfalls anzupassen. Bei einem Betriebssystem wie BrickOS ist es aber mit relativ geringem Aufwand möglich, einen kernelseitigen Treiber zu schreiben, der diese Aufgabe übernimmt. Voraussetzung dafür ist aber, daß auch der Datentransfer in Richtung des RCX über Kernel-Funktionen stattfindet. Erst dann weiß das Betriebssystem, welcher Sensor gerade aktiv ist. Natürlich muß vom Benutzer trotzdem noch der Sensortyp mitgeteilt werden.

2.3.3 Betriebssystem-Unabhängigkeit

Besonders hinsichtlich einer späteren Vermarktung wäre es wichtig, daß die Nutzbarkeit des Lepomux nicht von dem im RCX verwendeten Betriebssystem abhängig ist. Wenn für eine sichere Datenübertragung ein Großteil der Systemstruktur umgebaut werden muß, gleicht das doch eher einem Pyrrhussieg als einer marktreifen Lösung. Die bisherige Übertragungstechnik ist demnach für den Praxiseinsatz ungeeignet.

Um diese Betriebssystem-Abhängigkeit zu umgehen, bedarf es eines Übertragungsprotokolls, daß kein präzises Timing und damit auch keine Kernelmodifikation mehr erfordert.

2.3.4 Verbesserung des Übertragungsprotokolls

Was beim bestehenden Protokoll bisher komplett außer Acht gelassen wurde, ist die Tatsache, daß ein Motorausgang in zwei Richtungen betrieben werden kann. Anstatt diese Eigenschaft mit einem Verpolungsschutz-Gleichrichter zunichte zu machen, könnte man sie für die Datenübertragung nutzen.

Die Möglichen Zustände *off*, *forward* und *reverse* entsprechen den Spannungspegeln 0, +7 und -7 Volt, was wiederum als ein Bit mit drei statt zwei Zuständen zu deuten ist. Durch diesen zusätzlichen Zustand läßt sich der Takt der Datenübertragung innerhalb der eigentlichen Nutzdaten unterbringen. Ein Bit liegt damit nicht mehr in einem festen Zeitraster, sondern wird durch einen Flankenwechsel getriggert. Die Präambel kann demnach komplett entfallen.

Im Ruhezustand wird auf dem Motorport ein *reverse* ausgegeben. Hieran kann der Lepomux die Polung erkennen, mit der er an den RCX angeschlossen wurde. Wird hier ein *forward* statt eines *reverse* erkannt, müssen die RCX-Daten vor der Dekodierung invertiert werden.

Wie die Abbildung 2.22 zeigt, ist die Codierung der Datenbits eine relativ simple Angelegenheit: Bei einer Null wird der Motor auf *off* geschaltet, bei einer Eins dagegen auf forward. Folgen mehrere gleiche Bits aufeinander, so wird abwechselnd *off* und *reverse* beziehungsweise *forward* und reverse ausgegeben.

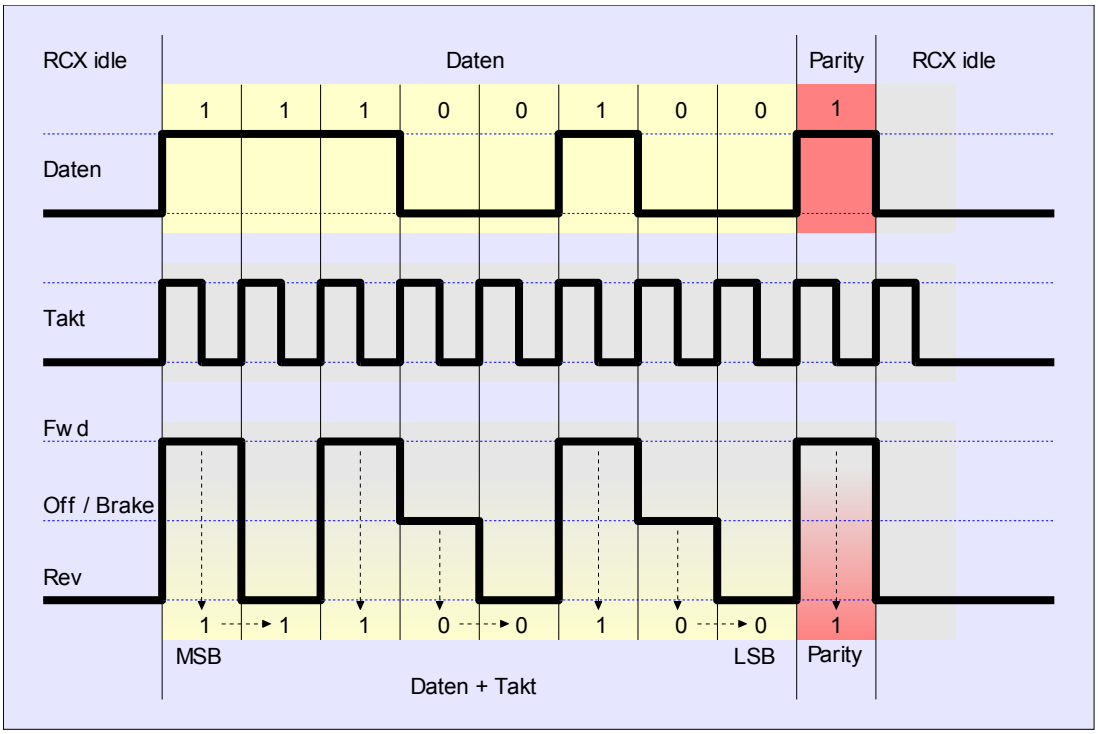


Abbildung 2.22: Verbessertes Übertragungsprotokoll

Ein Datenwort besteht auch weiterhin aus acht Datenbits und einem Parity-Bit. Ist der Motor nach der Übertragung des kompletten Datenwortes nicht wieder auf *reverse* eingestellt, so folgt ein Übergang dorthin, um wieder in die Ausgangsposition für die Übertragung eines weiteren Datenwortes zu gelangen. Die dadurch entstehende Flanke ist für die Übertragung uninteressant und wird vom Lepomux ignoriert.

Nach einer Pause von mindestens zwei Bitlängen[53] kann das nächste Datenwort gesendet werden. Diese Wartezeit ist nötig, um dem Empfänger einen Chance zu geben, den Anfang eines neuen Datenwortes korrekt zu identifizieren. Ohne eine Pause wäre nach einem Übertragungsfehler keine Synchronisation auf den Datenwort-Beginn mehr möglich. Die Mindestlänge der Übertragungspause ergibt sich aus der empfängerseitigen Einstellung für das Bit-Timeout. Tritt innerhalb dieses Zeitraums kein Flankenwechsel auf, so werden die bisher empfangenen Daten verworfen und die nächste Flanke als Beginn eines neuen Datenwortes interpretiert. Der Bit-Timeout ergibt sich aus maximalen Bitlänge des Senders.

Für ein unmodifiziertes BrickOS-System ergibt sich eine Bit-Zeit von ca. einer Millisekunde. Das Versenden eines Datenwortes dauert demnach 9 ms zuzüglich einer Wartezeit von mindestens 3 ms bis zur nächsten Übertragung. Damit ergibt sich eine Zeit von 12 Millisekunden pro Datenwort, was einer kontinuierlichen Datenrate von etwas mehr als 83 Byte pro Sekunde entspricht.

2.3.5 Konsequenzen für die Hardware

Damit das oben beschriebene Protokoll überhaupt auswertbar wird, muß die Hardware geringfügig modifiziert werden. Dies betrifft hauptsächlich den in Kapitel 2.1.2 beschriebenen Daten-Eingang. Hierbei entfällt der Gleichrichter REC-1.[54] Der negative Spannungspegel, der anliegt, wenn der Motor auf *reverse* geschaltet wird, läßt sich über einen weiteren, invers angeschlossenen Optokoppler detektieren.

An diesem Punkt stehen wir nun vor dem Problem, daß unser Mikrocontroller zu wenige Eingangspins bietet, denn das Signal des zweiten Optokopplers muß natürlich auch in den Prozessor geführt werden. Die Entscheidung, bei welcher der anderen Komponenten auf eine der Portleitungen verzichten werden kann, ist nicht leicht. Hierfür bieten sich eigentlich nur Pins an, die bisher als Ausgang genutzt wurden. Komponenten, die Eingänge des MSP belegen, nutzen auch seine Port-Interrupt-Fähigkeit.

Die Ausgänge, bei denen eine „geklaute“ Portleitung am wenigsten ins Gewicht fällt, sind die vier Motor-Steuerleitungen P1.4 bis P1.7. Um trotzdem zwei Motortreiber und entsprechend viele Leuchtdioden ansteuern zu können, müssen die verbleibenden drei Leitungen effektiver genutzt werden. Wie das im Detail bewerkstelligt wurde, läßt sich dem Kapitel 3.1.2 entnehmen.

Abbildung 2.23: Kontaktaufnahme: LEGO-Anschlüsse brauchen einen Adapter, um an die 26-polige Pfostenwanne zu passen

2.3.6 Sensor-Anschlüsse

Bisher sind die Anschlüsse des Sensor-Multiplexers zwar über eine 26-polige

53 Da die Länge eines Bits nicht mehr an ein festes Zeitraster gebunden ist, muß hier als Bitlänge aus der maximalen Dauer bis zu einem Flankenwechsel berechnet werden. Bezogen auf BrickOS können dies wegen des instabilen Timings bis zu zwei Millisekunden sein (siehe Kapitel 2.2.2).

54 Siehe Schaltplan im Anhang .

Pfostenwanne aus dem Gehäuse geführt, über die jedoch keine direkte Kontaktierung von LEGO-Sensoren möglich ist. Für den Praxiseinsatz ist demnach unbedingt ein Adapter nötig, der die Anschlüsse des Pfostensteckers auf LEGO-kompatible Stecker umsetzt.

Da LEGO-Stecker relativ teuer sind und in vielen Fällen gar nicht alle Ein- und Ausgänge des Lepomux benötigt werden, bietet sich als Lösung ein modulares Stecksystem an. Das Kapitel 3.1.4 befasst sich mit der konkreten Umsetzung dieser Überlegung.

3 Lepomux v2.0

Die zweite Version des Lepomux entspricht im Funktionsumfang weitgehend der Machbarkeitsstudie aus Kapitel 2. Zielsetzung für diese deutlich verbesserte Variante war das Erreichen eines praxistauglichen Status.

3.1 Änderungen der Hardware

Die hier angesprochenen Änderungen ergeben sich fast ausschließlich aus den Veränderungen der Hardware, die durch die Überarbeitung des Protokolls nötig wurden. Die beiden kompletten Schaltpläne der Version 2.0 sind im Anhang und abgedruckt. Wie beim Vorgänger, ist die Schaltung auf zwei Leiterplatten aufgeteilt. Einige der Komponenten mußten aber wegen der veränderten MSP-Beschaltung die Platine wechseln.

3.1.1 RCX-Dateneingang

Wie schon in Kapitel 2.3.5 angesprochen, ist zur Auswertung des negativen Spannungspegels ein invertiert angeschlossener zweiter Optokoppler nötig. Er ersetzt den nutzlos gewordenen Gleichrichter REC-1.[55] Die übrige Beschaltung entspricht der des Optokopplers U12 aus Abbildung 2.2: Jeweils ein Strombegrenzungswiderstand vor den LED-Eingängen der Optokoppler und je ein Pull-Up-Widerstand an den Ausgängen. Die entsprechend geänderte Schaltung ist Abbildung 3.1 zu entnehmen.

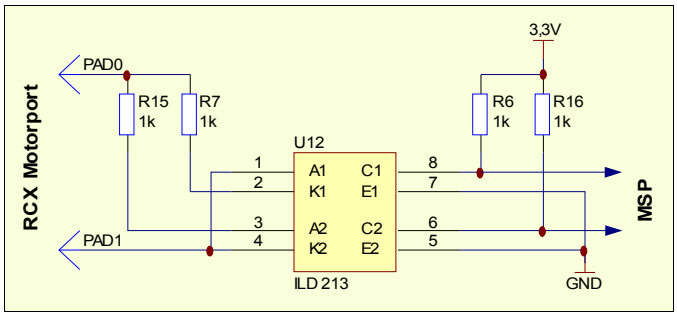


Abbildung 3.1: Die verbesserte Schaltung des Dateneingangs

3.1.2 Schieberegister für Motorausgänge & LEDs

Wegen des zusätzlichen Optokopplers am Dateneingang stehen nur noch drei Ausgangspins zur Steuerung der Motortreiber zur Verfügung. Die Treiber können also nicht mehr wie vorher direkt an den MSP angeschlossen werden. Es ist also zwangsläufig ein Bindeglied nötig, das über die drei Steuerleitungen des MSP mindestens vier Ausgänge zur Ansteuerung der Motoren bereitstellt.

Die Lösung ist denkbar einfach: Die Steuerdaten für die Motoren werden seriell über die verbleibenden drei MSP-Ausgänge in ein 8-Bit breites Schieberegister übertragen.

Vier der acht Ausgänge des Schieberegisters sind mit den Eingängen der Motortreiber verbunden, die anderen vier Ausgänge dienen zur Ansteuerung der Leuchtdioden. Somit sind Motoren und LEDs jetzt unabhängig voneinander schaltbar.

Als Register wird hier ein 74 HCT 595[56] verwendet, das mit einer Spannung von 5

55 Vergleiche Kapitel 2.1.2.

56 Der 74 HCT 595 ist ein 8-Bit-Schieberegister, das über getrennte Schiebe- und Ausgabe-Flipflops verfügt. Damit ist es möglich, neue Daten in der Register zu schieben, ohne dabei die Registeraus-

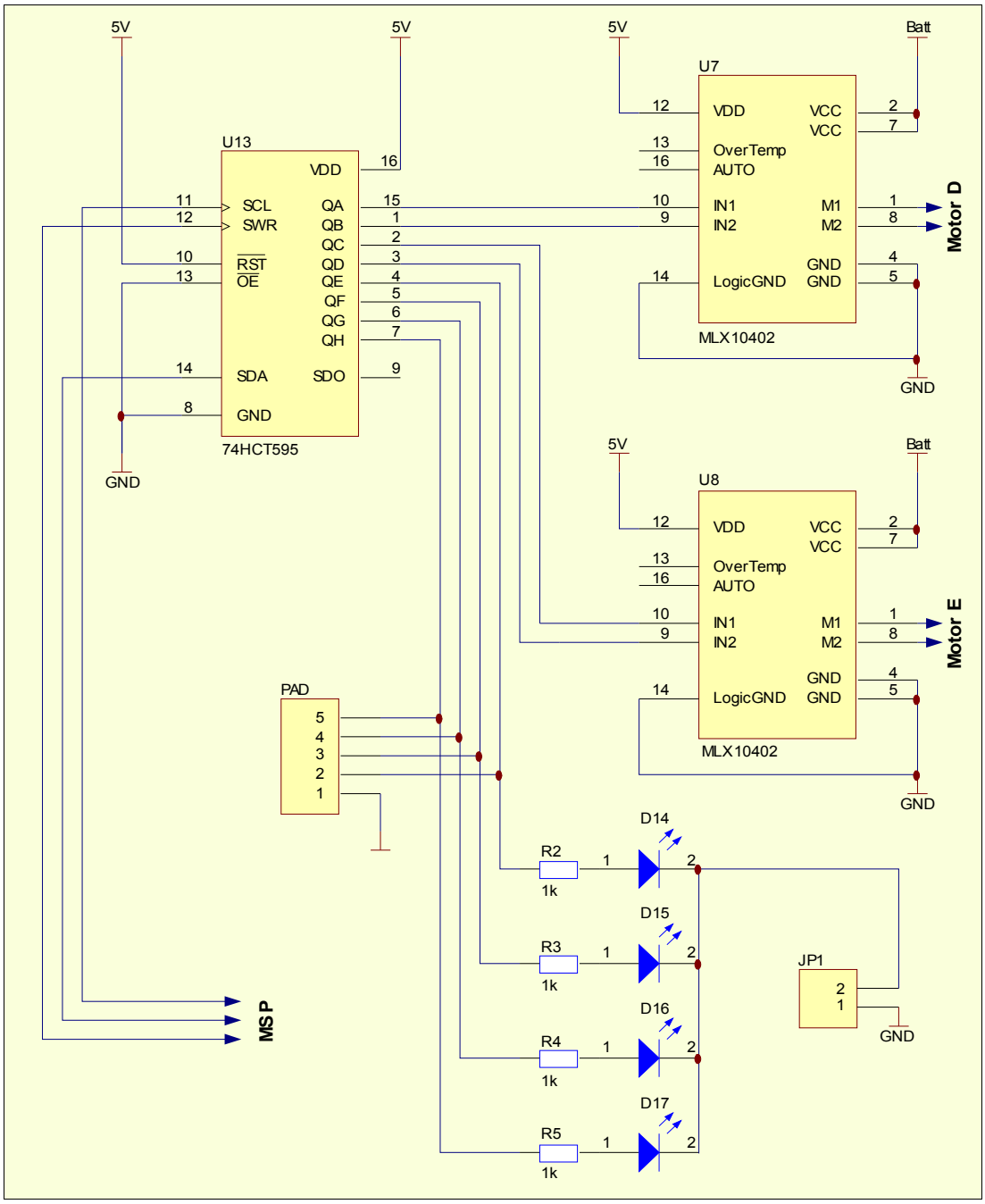


Abbildung 3.2: Beschaltung des Schieberegisters

Volt betrieben wird. Die HCT-Familie ist im Vergleich zur HC-Familie TTL-kompatibel, was den Vorteil hat, das Schieberegister ohne weiteren Pegelwandler mit dem MSP verbinden zu können.[57] Der Grund, weshalb das Register mit fünf statt mit drei Volt betrieben wird, ist aber ein anderer: Die Registerausgänge sind TTL-kompatibel und können demnach größere Lasten treiben als ein Baustein der HC-Familie. Damit eignen sich diese Ausgänge zur direkten Ansteuerung von

gänge zu beeinflussen. Erst eine explizite Taktung zur Übernahme der Daten sorgt dafür, daß diese in das Ausgangs-Register übernommen werden.[25]

57 Die niedrige TTL-Schaltschwelle konnte man sich schon bei der direkten Ansteuerung der Motortreiber zunutze machen, um einen zusätzlichen Pegelwandler einsparen zu können. Siehe Kapitel 2.1.6.

Modellbau-Servos, die üblicherweise mit einer Spannung von 4,8 bis 6 Volt betrieben werden. Der Stecker CN-LED ist für Experimente bezüglich dieser Servo-Steuerung vorgesehen, wird aber nicht aus dem Gehäuse geführt.

Nachteil der Verwendung des Schieberegisters ist allerdings, daß nicht mehr wie bisher ein einziger Registerzugriff genügt, um beide Motoren bzw. die LEDs auf den gewünschten Status zu setzen. Stattdessen wird jetzt eine Routine benötigt, die die Steuerdaten im Gänsemarsch zum Register überträgt. Dazu werden die verbliebenen drei Ausgangsleitungen des MSP benötigt. Die acht Datenbits werden nacheinander auf den Daten-Pin (SDA) gelegt. Die zweite Leitung dient dabei als Takt (SCL), um das jeweilige Datenbit ins Schieberegister zu übernehmen. Nachdem das gesamte Datenwort in das Register übernommen wurde, sorgt ein Impuls auf dem verbleibenden dritten Anschluß (SWR) dafür, daß diese Daten in die Ausgangs-Flipflops übernommen werden.[58]

Die Übertragung eines Bytes nimmt bei einem Systemtakt von 8 MHz etwa eine Zeit von 80µs in Anspruch. Dabei ist das Timing aber unkritisch, so daß auch eine kurze Unterbrechung der Übertragung durch einen Interrupt keine Probleme bereitet.

3.1.3 Alternativer Pegelwandler

Um Platz und Kosten zu sparen wurde der im alten Design auf *Board B* verwendete und nur zur Hälfte belegte Pegelwandler MC 14504 durch eine Ersatzschaltung aus je einem Widerstand und einer Diode ersetzt. Die Abbildung 3.3 zeigt die zugehörige Schaltung.

Aufgabe dieses Wandlers ist es, die 5-Volt-Pegel der IR-Sensoren auf MSP-verträgliche 3,3 Volt zu reduzieren. Da die Sensoren mit Push-Pull-Ausgängen[59] ausgestattet sind, müssen die 5 Volt, die während eines High-Pegels anliegen über eine Diode abgeblockt werden. Hinter der Diode sorgt ein Pull-Up-Widerstand für einen 3,3-Volt-Pegel, solange der Pin nicht über diese Diode gegen Masse gezogen wird.

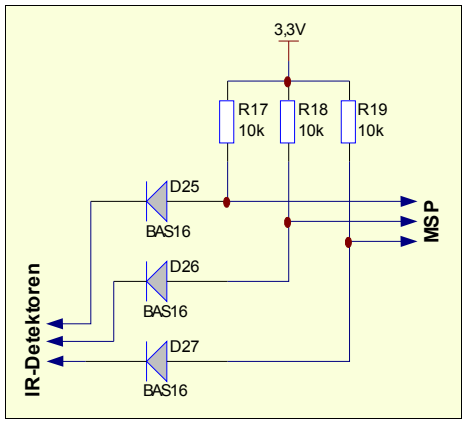


Abbildung 3.3: Diodenbasierter Pegelwandler

Bezogen auf die drei IR-Sensoren besteht der Pegelwandler nun aus den Bauteilen D25 bis D27 und R17 bis R19. Solch eine Art der Pegelanpassung lohnt sich allerdings nur bei wenigen Leitungen.

58 Die Bezeichnung dieser Pins mit SDA und SCL ist etwas irreführend, da diese Übertragung nichts mit dem I²C-Protokoll zu tun hat, wo diese Bezeichnungen gängig sind. Allerdings ließen sich diese Pins auch zur Implementierung von I²C verweden (siehe Kapitel 6.3).

59 Diese Ausgänge werden auch Totem-Pole-Ausgänge (TP) genannt und repräsentieren die gebräuchlichste Art von Logikbaustein-Ausgängen. Charakteristisch ist dabei die Beschaltung mit zwei komplementär angesteuerten Ausgangstransistoren, die den Ausgang entweder mit der Versorgungsspannung oder mit der Masse des Bausteins verbinden. Ein Open-Collector-Ausgang (OC) besitzt dagegen nur den Transistor, der in Richtung der Masse schalten kann. Mit einem OC-Ausgang läßt sich deshalb kein High-Pegel erzeugen, was die Verwendung eines Pull-Up-Widerstands nötig macht. Für die Funktion der hier angesprochenen Pegelwandler-Schaltung würde ein OC-Ausgang ausreichen.

3.1.4 Stecksystem

Der 26-polige Pfostensteckverbinder, über den die Sensor- und Motoranschlüsse aus dem Gehäuse herausgeführt sind, ist ohne ein zusätzliches Stecksystem, das diese Auschlüsse auf LEGO-kompatible Stecker umsetzt, relativ nutzlos.

Da das Gehäuse genau die Breite von vier LEGO-Anschlüssen besitzt und nur in seltenen Fällen alle acht Sensoreingänge benötigt werden, sind die Eingänge des Sensor-Multiplexers in zwei Blöcke aufgeteilt, die über jeweils ein Steckmodul kontaktiert werden können.

Ein solches Modul besteht jeweils aus eine Trägerplatine, die links und rechts mit einem Pfostenstecker bzw. einer Pfostenwanne ausgestattet ist. Auf diese Leiterplatte wird von oben das eigentliche Anschlußmodul aufgesteckt. Seitlich läßt es sich mit dem Lepomux verbinden und über die Pfostenwanne auf der gegenüberliegenden Seite kann ein weiteres Modul kontaktiert werden. Abbildung 3.4 zeigt ein derartiges Modul.

Je nach Bedarf hat man die Wahl zwischen zwei unterschiedlichen Anschlußmodulen. Eines mit vier LEGO-Anschlüssen oder ein anderes, das Stecker für die beiden Motorports zur Verfügung stellt und zusätzlich noch eine Anschlußmöglichkeit für bis zu vier IR-Distanzsensoren[60] der Firma Sharp bietet. Der Stromverbrauch von bis zu 50 mA pro Distanzsensor bei 5 Volt macht einen zusätzlichen Spannungsregler nötig, denn der Regler des Lepomux ist nur für einen Strom von 100 mA ausgelegt. Aus diesem Grund ist die Trägerplatine mit einem ***7805***-Regler bestückbar, der per Jumper die Speisung des 5-Volt-Pins für das Aufsteckmodul übernehmen kann.

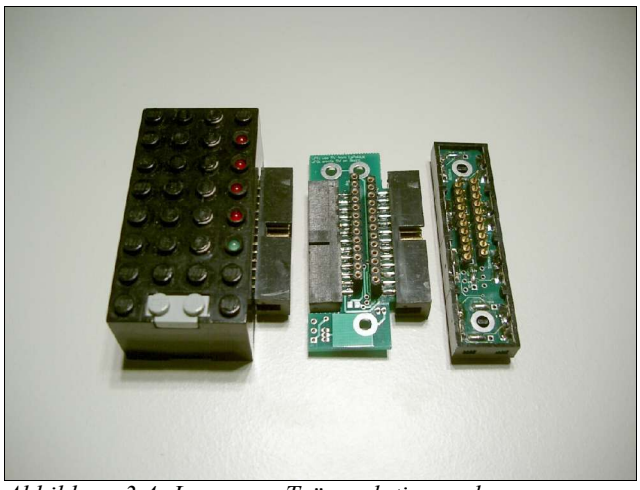


Abbildung 3.4: Lepomux, Trägerplatine und Aufsteckmodul für vier LEGO-Sensoren

Das Anschlußmodul mit den LEGO-Steckverbindern ist von der Leiterplatte her so gefräst, daß die Oberteile der sonst an LEGO-Kabeln zu findenden Stecker mitminimaler Modifikation aufsteckbar und von der Rückseite aus verlötbar sind. Somit ergibt sich eine nahtlose Leiste von vier LEGO-Kontakten.

Im Normalfall sollten nie mehr als zwei Trägerplatinen benötigt werden, da damit bereits alle Sensorports belegt sind. Allerdings gibt es kein Anschlußmodul, das ausschließlich Motorports oder LEGO-Kontakte in Verbindung mit Motorports bietet.

60 Der GP2D12 von Sharp ist ein Distanzsensor, der mit modulierter Infrarotstrahlung arbeitet und Entfernungen zu Objekten im Abstand von 10 bis 80 Zentimetern als analogen Spannungswert ausgibt.[26]
Um diesen Sensor in Zusammenhang mit dem RCX benutzen zu können, ist eine Aufbereitung des Ausgangssignals unumgänglich. Hierfür bietet sich ein als Puffer arbeitender Operationsverstärker oder eine einfache Transistorstufe an. Das Kapitel 4.1.2.2 setzt sich näher mit dieser Thematik auseinander.

3.2 Änderungen der Software

Die Änderungen der Hardware ziehen leider einen nicht unerheblichen Umbau der Software nach sich. So hat sich die komplette Belegung der MSP-Ein- und Ausgänge geändert und damit auch die Interrupt-Belegung und deren Prioritäten. Die bereits in Kapitel 2.3.4 beschriebene neue Variante der Datenübertragung kommt nicht mehr mit einer einzelnen Portleitung aus. Die dazu benötigte zweite Leitung war ursprünglich für die Ansteuerung der Motortreiber vorgesehen. Da dort nun eine Portleitung fehlt, erfolgt die Motoransteuerung nicht mehr direkt über die MCU, sondern mit einem Schieberegister, das über drei Portleitungen mit einem seriellen Protokoll angesprochen wird. Die genauen Änderungen finden sich in den nachfolgenden Kapiteln.

Das Blockschaltbild der Software in Abbildung 3.5 vermittelt einen Überblick, wie die einzelnen Komponenten miteinander verknüpft sind.

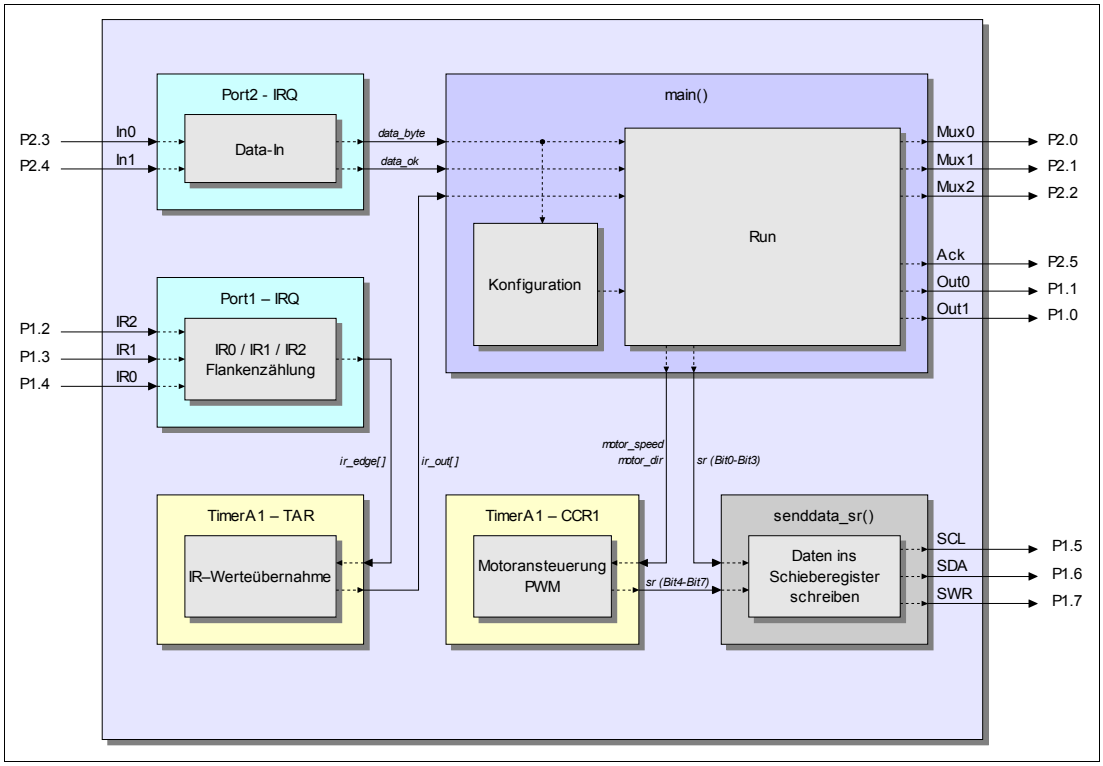


Abbildung 3.5: Blockschaltbild der Software Version 2

Für die Verteilung der Interruptprioritäten ergibt sich die folgenden Aufteilung (von oben nach unten mit abnehmender Priorität):

- TimerA0:
  - CCR0: (nicht benutzt)
- TimerA1:
  - TAR: Übernahme der IR-Zählerstände für die Frequenzauswertung
  - CCR1: Motoranteuerung / Pulsweitenmodulation
  - CCR2: Polungserkennung des Dateneingangs
- Port2: Datenübertragung von RCX zum Lepomux
- Port1: Impulszählung des IR-Detektors

Daß der Compare-Interrupt mit der höchsten Priorität unbelegt bleibt, hängt mit der internen Verdrahtung des MSP zusammen. Ein Timer-Modul ist jeweils nur im Zusammenhang mit bestimmten Portleitungen benutzbar.

3.2.1 Protokolländerung / Polungserkennung

Durch die Protokolländerung ist es nun nicht mehr nötig, den Datenempfang über einen Capure/Compare-Interrupt auszuwerten. Da für das Protokoll jetzt ausschließlich Flankenwechseln relevant sind, kann die Auswertung innerhalb eines Port-Interrupts stattfinden. Wegen der gegenüber Port 1 höheren Interrupt-Priorität, wird hierzu der Port 2 verwendet.

Bei diesem Protokoll wird nicht mehr wie in Version 1.0 (vgl. Kapitel 2.2.1.5) eine feste Bitzeit vorausgesetzt, sondern es wird zusammen mit den Daten ein Taktsignal übertragen. Somit ist dieses Übertragungsprotokoll gegenüber Timingproblemen seitens des RCX resistent. Bei der Übertragung eines Datenwortes, muß bei jedem Bitwechsel an mindestens einem der Dateneingänge ein Flankenwechsel stattfinden. Diese Flanken stellen damit quasi das Taktsignal dar. Die Abbildung 3.6 verdeutlicht diesen Zusammenhang.

Aufgrund Eingangsbeschaltung[61] liegen die Signale invertiert am MSP an. Das hat einen High-Pegel an den Eingängen Data-IN0 und Data-IN1 zur Folge, wenn der RCX ein *brake* oder *off* sendet. Für den Fall, daß *forward* oder *reverse* ausgegeben

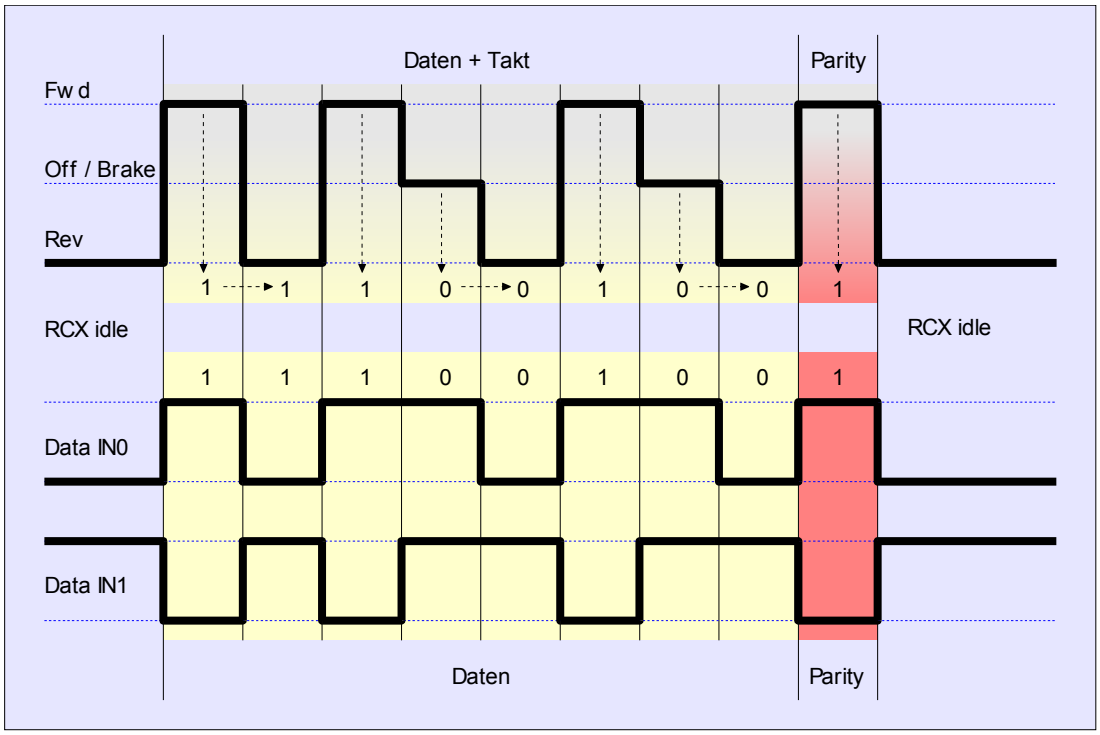


Abbildung 3.6: Datenübertragunsprotokoll Version 2

werden, führen diese beiden Leitungen unterschiedliche Pegel. Da sich beim Verpolen des Anschlußsteckers am RCX auch die Pegel an den beiden Eingängen umkehren, ist eine softwareseitige Berücksichtigung der Steckerpolung zwingend notwendig.

61 Siehe Kapitel 3.1.1.

Im Ruhezustand sendet der RCX immer ein *reverse,* welches zur oben genannten Polungserkennung verwendet wird. Anhand dieses Pegels wird intern die Variable *dir* gesetzt, um der Empfangsroutine zu signalisieren, ob die Leitungspegel invertiert werden müssen.

Eine sichere Erkennung dieses Ruhezustands ist allerdings mit einer Begrenzung der maximalen Bitzeit verbunden. Hierzu wird das im TimerA1-Modul enthaltene CompareRegister (CCR2) benutzt, welches so geladen wird, daß es alle 1,5 ms[62] einen Interrupt auslöst. Bei jedem Einsprung in die Routine wird getestet, ob sich der Bitzähler für den Datenempfang verändert hat. Ist der Wert beim nachfolgenden Einprung immer noch unverändert, werden die verwendeten Zähler des Dateneingangs zurückgesetzt und die bisher empfangenen Bits verworfen. Ohne diese Unterscheidung, wäre eine Resynchronisation auf den Datenwortbeginn nicht möglich und es würde zum Empfang falscher Datenworte kommen.

Bei der Auswertung der Leitungspegel wird anhand des Inhalts der oben angesprochenen Variable *dir,* der Bit-Zustand *forward* (1) oder *reverse* (2) aus den Pegeln der Eingänge IN0 und IN1 berechnet. Liegt auf beiden dieser Leitungen ein High-Pegel an, führt das unabhängig vom Wert der Variable *dir* immer zu einer Einstufung als *brake* bzw. *off* (0).

Direkt nachdem durch einen Flankenwechsel ein Interrupt ausglöst wurde, muß einige Mikrosekunden bis zum Abfragen der Portleitungen gewartet werden. Es kann sonst vorkommen, daß ein gleichzeitiger Pegelwechsel beider Leitungen nicht richtig erkannt wird.

Die oben für das aktuelle Datenbit berechneten Werte 0 und 1 werden als realer Bitwert interpretiert, eine 2 entspricht dagegen dem Wert des zuvor empfangenen Bits (Wiederholungsbit)[63]. Somit kann wie bereits in der ersten Protokollversion schon während des Datenempfangs das Parity-Bit berechnet werden. Am Ende eines Datenwortes wird es mit dem empfangenen Parity-Bit verglichen, um Übertragungsfehler auszuschließen.

Nach jedem empfangenen Bit muß an dem Eingang, der den Interrupt ausgelöst hat, der Typ der Flankenerkennung neu eingestellt werden[64].

RCX-seitig läßt sich dieses Protokoll mit wenigen Zeilen Code implementieren. In dem in Anhang abgedruckten Beispielcode ist die bereits aus Abbildung 2.20 bekannte *senddata*-Routine entsprechend angepaßt worden.

62 Diese Zeitvorgabe ist so klein wie möglich gewählt, um zwichen zwei vom RCX gesendeten Datenworten nicht unnötig Zeit zu verlieren. Dennoch muß sie groß genug sein, um nicht schon inmitten einer Datenübertagung einen Reset des Bitzähles auszulösen. Die 1,5 ms sind für eine Bitzeit von einer Millisekunde gewählt.

63 Die Ausgabe von *reverse* entspricht immer einer Wiederholung des vorher gesendeten Bits. Folgt ein weiteres Bit mit gleichem Wert, so wird wieder auf *off* bzw. *forward* umgeschaltet. Damit wird auch eine korrekte Erkennung mehrerer aufeinanderfolgender Bits gleicher Wertigkeit möglich.

64 Im Port-Modus kann nicht wie in den Capture/Compare-Modulen automatisch auf beide Flanken getriggert werden. Deshalb muß in unserem Fall nach jedem Interrupt auf einer Portleitung der zu triggernde Flankentyp neu eingestellt werden. Ein Wechsel des Flankentyps nach jeder erkannten Flanke entspricht demnach einem triggern auf beide Flanken, denn auf eine fallenden Flanke muß zwangsläufig eine steigende folgen und umgekehrt.

3.2.2 Datenübertragung zum Schieberegister

Wie bereits oben angesprochen, fehlt eine zusätzliche Portleitung, um die Motortreiber wie bisher direkt vom MSP aus anzusteuern. Die verbleibenden drei Anschlüsse sind mit den Steuereingängen des Schieberegisters U13 verbunden (siehe Abbildung 3.2).

Wie in Abbildung 3.7 zu erkennen ist, übernimmt das Register den am SDA-Eingang anliegenden Pegel während einer steigenden Flanke am SCL-Eingang. Auf diese Weise werden die Bits der Reihe nach in einen acht Bit breiten Zwischenspeicher übernommen. Erst zum Zeitpunkt einer steigenden Flanke am SWR-Anschluß erscheint das zuvor übertragene Datenwort an den Registerausgängen.

Die beschriebene Datenübertragung zum Schieberegister ist prinzipiell zeitunkritisch, da einerseits der verwendete ***74 HCT 595*** schnell genug ist um bei vollem Prozessortaktung beschrieben zu werden und andererseits die Daten erst mit einer Flanke an SWR übernommen werden. Dabei ist egal, wie schnell oder langsam die einzelnen Bits in das Register geschrieben werden.

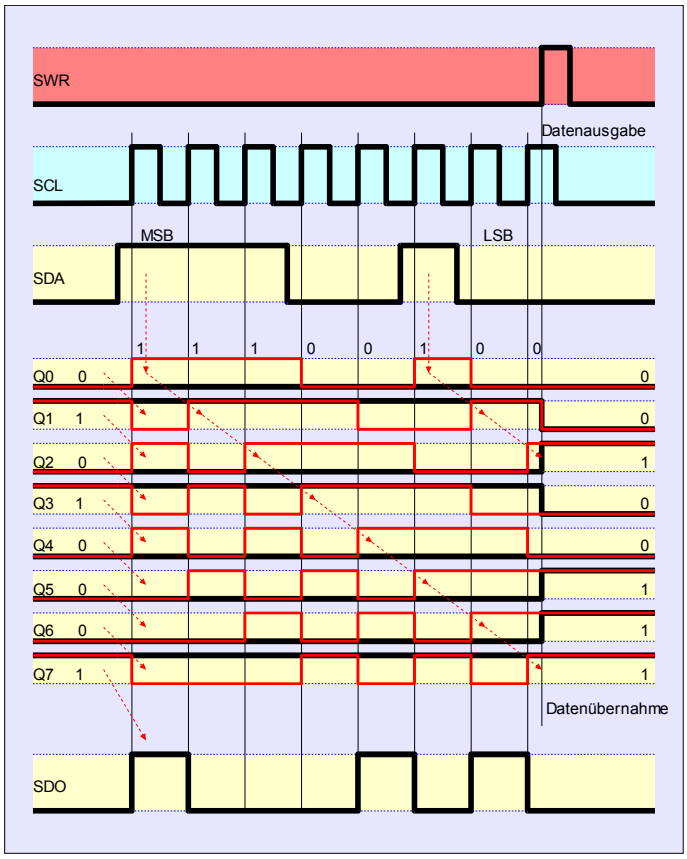


Abbildung 3.7: Datenübertragung zum Schieberegister

Eine Senderoutine faßt diesen Ablauf für die Übertragung eines Bytes zusammen. Dabei wurde auf eine optimierte Implementierung geachtet, um ein möglichst gutes Zeitverhalten zu erreichen. Das ist nötig, um auch PWM-Signale auf den Motorports ohne größere Latenzzeiten ausgeben zu können.

3.3 Evaluierung des Prototyps

Durch ein Redesign konnten die wesentlichen Nachteile der ersten Lepomux-Version aus der Welt geschafft werden (siehe Abbildung 3.8). Dies betrifft vor allem das aus elektrischer Sicht unbrauchbare Verhalten des Sensor-Multipexers, sowie die ursprüngliche Umsetzung der Datenübertragung. Das dort verwendete Protokoll war allein schon wegen der nötigen Kernel-Modifikation innerhalb des RCX-Betriessystems für den Praxiseinsatz untauglich. Aus diesem Grund wird das alte Protokoll nicht mehr unterstützt, obwohl das die Hardware zulassen würde.

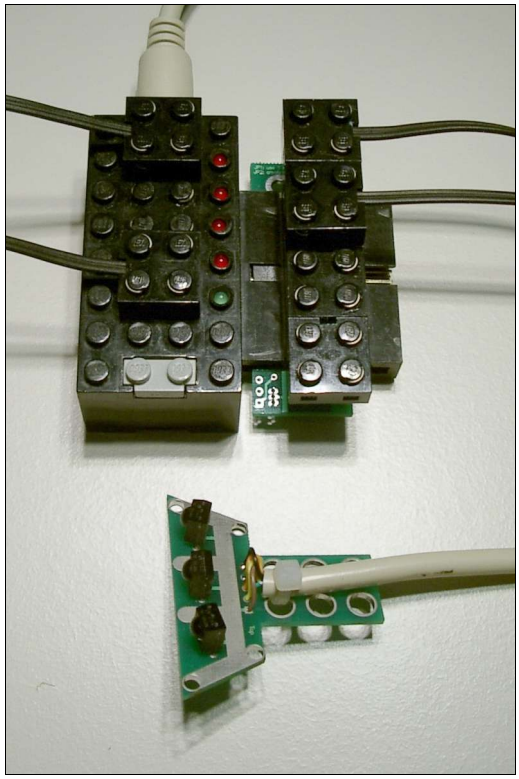


Abbildung 3.8: Lepomux v2.0

Allerdings ist die zweite Protokoll-Variante auch nicht perfekt. Bedingt durch die flankenbezogene Bitauswertung ist sie anfällig für Spikes. Zumindest die älteren Versionen des BrickOS-Kernels produzieren zeitweilig solche Pegel-Ausreißer und erfordern somit für eine stabile Datenübertragung einen minimalen Eingriff in den Code des Betriebssystems.

Die dritte Phase der Lepomux-Entwicklung soll den Prototyp so modifizieren, daß er sich optimal für den Einsatz im RoboLab der HAW-Hamburg eignet. Das betrifft im Speziellen die Sensor-/Aktor-Ausstattung und Zubehör, das den Prototyp zu einem Komplettsystem macht.

Verbunden mit dieser Optimierung war der Gedanke einer Fertigung von ca. 25 Lepomux für die Verwendung im RoboLab. Außerdem zeigte auch die LEGO-Fangemeinde Interesse.[65]

Da eine Fertigung des Lepomux in größeren Stückzahlen per Handarbeit einen enormen Aufwand bedeutet hätte, sollten nach Möglichkeit größere Platinen-Mengen mittels Automaten-Bestückung gefertigt werden. Leider ergab unsere Recherche, daß die Einrichtkosten für eine Bestückungsmaschine so hoch sind, daß sich eine automatisierte Fertigung erst ab ca. eintausend Stück lohnt. Außerdem bereiteten uns die folgenden beiden Punkte besonderes Kopfzerbrechen:

3.3.1 Gehäuse

Da der Lepomux in einem LEGO-Batteriekasten untergebracht ist, mußte schon bei der Kalkulation der Produktionskosten ermittelt werden, zu welchem Preis wir diese Kästen bei der Abnahme größerer Mengen bekommen könnten. Ein Anruf bei der Firma LEGO sorgte dann aber dafür, daß die gesamte Planung zunichte gemacht wurde: Der von uns verwendete Batteriekasten wurde schon seit geraumer Zeit nicht mehr von LEGO produziert und es waren nur noch Restbestände verfügbar. Keine gute Grundlage für ein größeres Produktionsvorhaben.

Auch die nötigen Umbauten am Gehäuse waren zwar bezogen auf die Fertigung von Einzelstücken kein großes Problem, ließen sich aber nicht maschinell durchführen. Gerade das Umrüsten der Stromanschlüsse an der Oberseite des Batteriekastens hätte einen enormen Aufwand bedeutet.

Wir waren demnach gezwungen, uns nach einem neuen Gehäuse für den Lepomux umzusehen, womit natürlich auch das gesamte Platinen-Design hinfällig wurde.

65 Wir hatten die Ergebnisse unserer Arbeit relativ frühzeitig auf unserer Website www.lepomux.org [27] zusammengefasst. Um zu sehen, wie groß das Interesse an solch einer Entwicklung ist, haben wir eine kleine Produktankündigung in das Forum des Lugnet [28] geschrieben.

3.3.2 Stecksystem

Das bisher verwendete Stecksystem (siehe Kapitel 3.1.4) ist ein ähnlicher Problemfall. Zwar ist hier eine maschinelle Montage bis auf das Aufstecken und Verlöten der LEGO-Stecker möglich, aber die vielen Steckkontakte machen die Sache unnötig Teuer. Außerdem wird die Flexibilität eines solchen Systems an dieser Stelle nicht wirklich gebraucht, da nur maximal acht Sensor-Eingänge und zwei Motorports verfügbar sind.

Auch das Zerlegen von LEGO-Kabeln zu Steckergewinnung ist weder eine angenehme, noch billige Angelegenheit. Üblicherweise kosten selbst kurze LEGO-Kabel unglaubliche acht Euro, und auch die Abnahmemenge macht dabei praktisch keinen Unterschied.

3.3.3 Konzept für ein produktionsfähiges Redesign

Die oben genannten Gründe erfordern ein deutlich umfangreicheres Redesign, als es die reine Optimierung auf die Gegebenheiten des RoboLab nach sich gezogen hätte. Dennoch lohnt sich dieser Schritt, da sich mit einer Konzeptänderung deutlich besser dem Anforderungsprofil aus Kapitel 1.4.1 entsprochen werden kann. Das betrifft vor allem eine kostengünstige Fertigung und die Kompaktheit des Gesamtsystems.

3.3.3.1 Gehäuse

Der einzige LEGO-Batteriekasten, der sich einigermaßen für unsere Zwecke eignet, ist mit sechs Mignon-Zelle ausgestattet und besitzt an der Oberseine einen Anschluß für ein LEGO-Kabel. Daneben befinden sich zwei Taster, über die sich die Batteriespannung normal oder umgekehrt gepolt auf den Ausgang schalten läßt. Üblicherweise nehmen Kinder diesen Kasten beim Spielen in die Hand und betätigen mit dem Daumen einen der beiden Taster, um ihr Gefährt durch die Gegend fahren zu lassen.

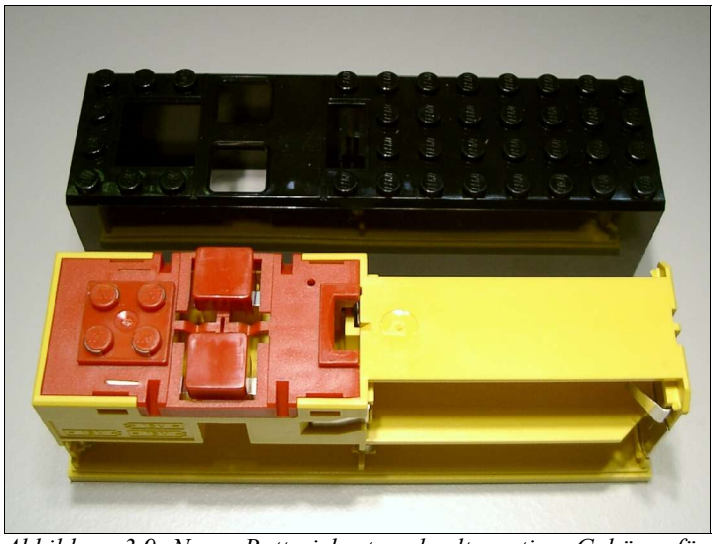


Abbildung 3.9: Neuer Batteriekasten als alternatives Gehäuse für Lepomux v3.0

Öffnet man das Gehäuse durch das Abziehen der oberen Halbschale, so kommt der eigentliche Batteriehalter zum Vorschein. Hierbei wird schnell klar, daß es bei dieser Konstruktion unmöglich ist, durch das Entfernen bestimmter Stege einen großen zusammenhängenden Raum zur Unterbringung der Schaltung zu schaffen. Der einzige nutzbare Teil des Batteriehalters ist durch die beiden Taster und den LEGO-Anschluß belegt. Da diese Bestandteile für unseren Zweck unnötig sind, können sie entfernt werden. Es ergibt sich ein kleiner rechteckiger Raum von ca. 44 x 26 x 10 Millimetern, der sich für die Unterbringung einer Leiterplatte nutzen läßt.

Bei diesen Abmaßen ist klar, daß nicht die gesamte Schaltung untergebracht werden kann. Der Batteriekasten enthält deshalb nur noch den Mikrocontroller, die Spannungsversorgung und alles, was für die Datenübertragung vom und zum RCX nötig ist. Alles, was darüber hinsausgeht, wird als Modul von oben auf den Batteriekasten gesteckt. Dies können Sensor-Multiplexer, Motortreiber oder beliebige andere Erweiterungen sein.

Ein großer Vorteil, der sich durch die schlechte Zerglegbarkeit des Batteriekastens ergibt ist, daß keine externen Batterien zur Versorgung des Lepomux mehr benötigt werden.

Das Entfernen der überflüssigen Bestandteile ist im Gegensatz zu dem Aufwand, der beim ursprünglich verwendeten Kasten getrieben werden musste, eine problemlose Prozedur aus wenigen Handgriffen. Zum Teil läßt sich dies auch maschinell mittels eines CNC-Fräsers erledigen.

3.3.3.2 Anschlüsse

Eine besonders kostspielige Angelegenheit sind die LEGO-Steckverbinder. Da es diese Stecker nicht einzeln gibt, sondern nur in Verbindung mit einem Kabel, fällt zusätzlich die Aufwand für die Demontage an. Die Kosten für ein Kabel betragen auch bei günstigem Einkauf über Wiederverkäufer im Internet mindestens noch stolze 4 Euro. Selbst unter günstigen Umständen kostet eine LEGO-Steckverbindung demnach zwei Euro zuzüglich des Arbeitsaufwandes. Für einen voll bestückten Lepomux der alten Generation wären das im günstigsten Fall allein 26 Euro, die auf solche Anschlüsse entfallen.

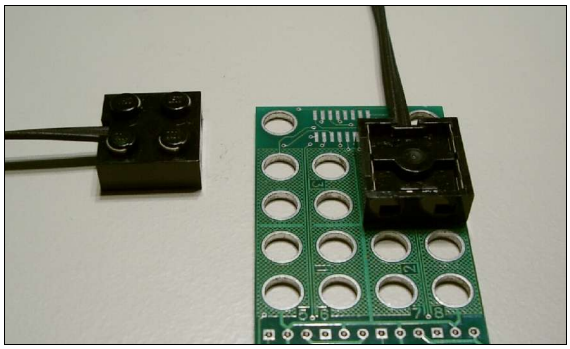


Abbildung 3.10: LEGO-Kontaktierung per Lötauge

Leider sind LEGO-kompatible Kontakte mit eigenen Mitteln nur sehr schwer herzustellen, denn sonst würde es sich anbieten, auf die teuren Originalteile zu verzichten. Ein Ansatz zu dieser Überlegung ist eher zufällig entstanden, als wir die Leiterplatten für die IR-Sensoren erhielten. Hierbei ist der Befestigungsschaft mit sechs Löchern versehen, so daß dieser direkt auf einen Legostein gesteckt werden kann. Um die Stabilität zu verbessern, hatten wir dort Lötaugen statt normaler Bohrungen verwendet. Nun zeigte sich, daß diese Lötaugen durchaus auch als Kontakte für einen LEGO-Stecker zu gebrauchen sind.[66] Somit war die Idee für eine sehr kostengünstige Kontaktleiste geboren, denn es mussten nur noch zwei diagonal gegenüberliegende der insgesamt vier Lötaugen eines Steckers mit einer Leiterbahn versehen werden.[67]

66 Fraglich ist allerdings, ob die Verzinnung der Lötaugen auch nach diversen Steckvorgängen noch für eine saubere Kontaktierung gesorgt hätte.

67 Die original LEGO-Stecker besitzen spezielle Kontaktflächen, die seitlich jeweils einen 45-Grad-Winkel der vier pro Stecker verwendeten Zapfen abdecken. Dabei sind je zwei Zapfen mit einem Kontakt verbunden, wobei die beiden zusammengehörigen Kontaktflächen dieses Pols die selbe Ausrichtung des 45-Grad-Segments haben. Die Kontaktflächen des Gegenpols sind dagegen um 180 Grad gedreht angebracht. Diese Konstruktion sorgt dafür, daß sich LEGO-Kontakte in beliebiger Weise miteinander verbinden lassen, ohne daß dabei ein Kurzschluß zwischen den beiden Polen eines Steckers entstehen kann.

LEGO-kompatible Kontakte können mit dieser Methode direkt in die Leiterplatte integriert werden. Die LEGO-Stecker werden zur Kontaktierung einfach über Kopf auf die Platine gesteckt.

3.3.3.3 Ausstattung

Wie bereits oben erwähnt, ergab sich die Ausstattung des neuen Lepomux hauptsächlich aus den Bedürfnissen, die sich für die Studenten im RoboLab der HAW Hamburg ergeben. Ziel ist es dabei, ein RCX-basiertes System ähnlich leistungsfähig zu machen wie eines, das als Basis ein 6.270-Board des MIT benutzt. Der wesentliche Schwachpunkt des RCX war bisher die schlechte Sensorausstattung. Mit viel Aufwand ließen sich zumindest IR-Distanzsensoren verwenden, aber an eine Erkennung der im Zusammenhang mit den 6.270-Boards oft benutzten IR-Beacons war nicht zu denken.[68]

Da für zur Verwendug der Distanzsensoren eine Vorverstärkerschaltung nötig ist, lag der Gedanke nahe, diese gleich in den Lepomux zu integrieren. Üblicherweise benötigt man an einem Roboter mindestens drei solcher Sensoren. Die neue Lepomux-Version sollte deshalb so beschaffen sein, daß vier der Sensorkanäle direkt mit den IR-Distanzsensoren der Firma Sharp betrieben werden können.[69] Die übrigen vier Kanäle der bisherigen Multiplexerschaltung bedienen weiterhin LEGO-kompatible Sensoren.

Die IR-Beacon-Erkennung geschieht wie bisher über eine Zusatzplatine, auf der die drei IR-Sensoren untergebracht sind. Angeschlossen wird dieses Modul aber über einen sechspoligen Mini-DIN-Stecker. Da diese Stecker auch bei PS/2-Anschlüssen von PCs Verwendung finden, sind z.B. Maus- oder Tastaturverlängerungskabel sehr günstig zu bekommen. Verwendet man an dieser Stelle ein PS/2-Verlängerungskabel, das an beiden Enden mit Steckern bestückt ist, so braucht man es nur zu zerschneiden und an den Schnittkanten an jeweils ein IR-Modul zu löten. Die gesamte Lötarbeit an den Steckern entfällt also.

Da für einen Roboter oft eine zusätzliche Schuß- oder Sammelmechanik nötig ist, die über zusätzliche Motoren angetrieben wird, muß mindestens der durch den Lepomux belegte Motorport kompensiert werden. Wichtig sind an dieser Stelle auch die Leuchtdioden, da sie sich wesentlich besser an einem fahrenden Roboter beobachten lassen als das LC-Display des RCX. Da das Schieberegister sowieso acht Steuerleitungen besitzt, können problemlos vier Motortreiber angesteuert werden. Die LEDs werden dabei parallel auf die Registerausgänge geschaltet.

Diese Ausstattung sollte für die meisten Aufgabenstellungen vollkommen ausreichend sein. Für den Fall, daß spezielle Sensor- oder Aktor-Kombinationen benötigt werden, kann jeder Zeit ein auf die Anforderungen zugeschnittenes Modul entwickelt werden, das dann auf den Lepomux-Batteriekasten aufgesteckt wird.[70]

68 Das Kapitel 1.1 beschäftigt sich mit den Unterschieden zwischen Mindstorm- und 6.270-Systemen.

69 Außer der Vorverstärkerschaltung muß hier aus die Spannungsversorgung entsprechend ausgelegt sein. Siehe Kapitel 3.1.4 und 4.1.2.2.

70 Tatsächlich ergab sich sehr schnell ein Bedarf für solch ein spezielles Modul. Einer der an der HAW ansässigen Assistenten berichtete von einem Bauvorhaben für einen sechsbeinigen Roboter. Dabei sollte jedes Bein mehrere Freiheitsgerade besitzen und demnach mit jeweils zwei Motoren ausgestattet sein. Die Gesamtzahl von 12 Motoren hätte dann allerdings vier RCX zur Steuerung erfordert.
Da die Ausgabe der Motordaten per Schieberegister nicht zwangsläufig auf 8 Bit begrenzt ist, läßt

4 Lepomux v3.0

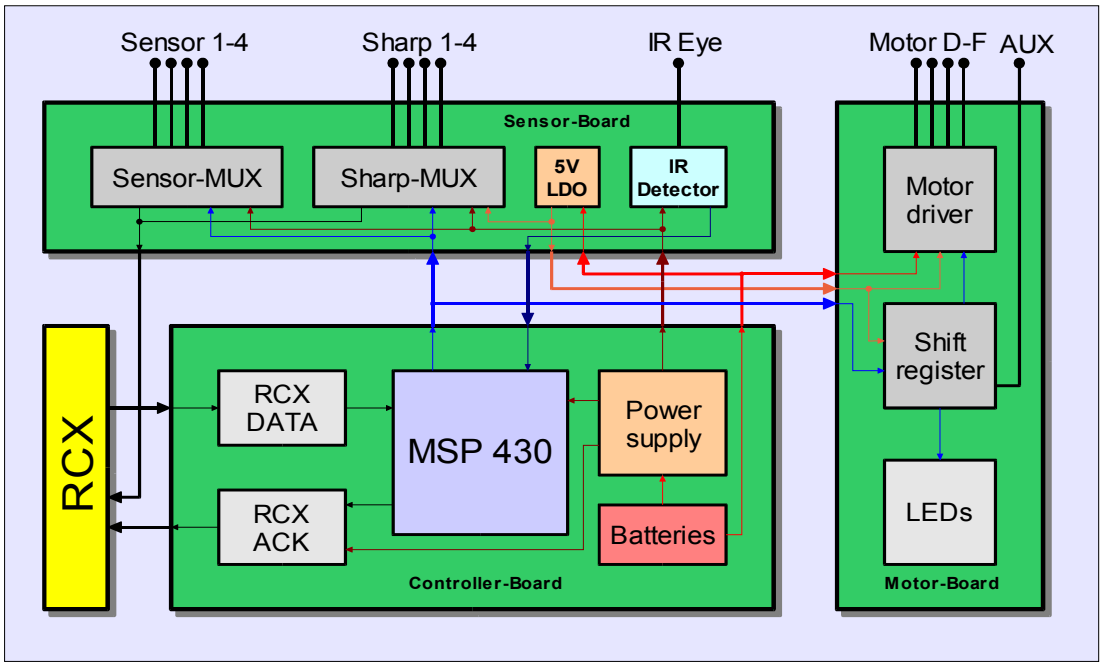


Abbildung 4.1: Komponentenschema des Lepomux v3.0

Nach der kompletten Umstrukturierung des ursprünglichen Designs hat der Lepomux zumindest äußerlich keinerlei Ähnlichkeit mehr zu seinen älteren Brüdern. Ein guter Grund, die neue Ausgabe mit einer höheren Versionsnummer zu versehen. Die Abbildung 4.1 verdeutlicht die modulare Struktur des dritten Lepomux.

4.1 Hardware

Beim Entwurf der Schaltungen wurden größtenteils die Ansätze der Version 2.0 übernommen. Wesentliche Änderungen gab es nur im Bereich der Stromversorgung, des Sensor-Multiplexers und diverser Steckverbinder. Durch die in Kapitel 3.3.3.1 angesprochene Gehäuse-Problematik ist die Schaltung jetzt allerdings auf mehrere Leiterplatten verteilt.

4.1.1 Controller-Board

Diese Platine beherbergt die Steuerzentrale des Lepomux. Da die maximal nutzbare Platinenfläche innerhalb des Batteriekastens nur knapp elf Quadratzentimeter beträgt, mußte ein Großteil der übrigen Schaltungsteile auf externe Module[71] ausgegliedert werden. Die Controller-Leiterplatte beinhaltet nur noch den Prozessor, die Stromversorgung und alle Komponenten, die zur Kommunikation mit dem RCX nötig sind.

Dieser Aufbau ist besonders Vorteilhaft, da Batteriekasten und Controller-Platine eine universelle Basis für nahezu beliebige RCX-Erweiterungen darstellen. Der Mikrocontroller kann durch seinen Flash-Speicher je nach Verwendungszweck mit spezieller Lepomux-Firmware programmiert werden, die dann auch eine Ansteuerung

sich ohne großen Aufwand ein Motortreiber-Array mit 16 oder auch mehr Kanälen aufbauen (siehe Kapitel 4.1.4).

71 Siehe Kapitel 4.1.1.3.

von Eigenbau-Erweiterungen ermöglicht.[72]

4.1.1.1 Stromversorgung

Der prinzipielle Aufbau der Stromversorgungs-Sektion gleicht dem der älteren Lepomux-Versionen.
Wegen des Platzmangels kommen hier aber deutlich kleinere Spannungsregler im fünfpoligen SOT23-Gehäuse zum Einsatz[73].

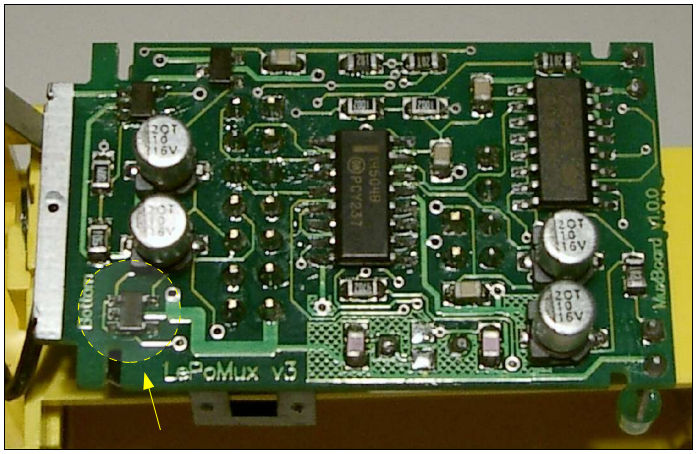


Abbildung 4.2: Controller-Board mit TPS 76150 im 5-poligen SOT-23-Gehäuse

Im Gegensatz zu den bisherigen Lepomux-Varianten mit externem Batterie-Pack ergibt sich hier eine ganz neue Problematik: Da die Batterien innerhalb des Lepomux-Batteriekastens permanent mit der Controller-Platine verbunden sind, wäre ein zusätzlicher Schalter nötig oder es müsste mindestens eine der Batterien entfernt werden, um den Lepomux auszuschalten.

Für einen Schalter bietet das Gehäuse keinen Platz und das Entfernen einer Batterie ist alles andere als praktisch – besonders, wenn der Lepomux in einen Roboter eingebaut ist. Glücklicherweise bietet der oben genannte 5-Volt-Spannungsregler mit seinem Enable-Eingang die Voraussetzung für eine wesentlich elegantere Lösung:

Üblicherweise ist der Lepomux immer dann in Betrieb, wenn auch der RCX benutzt wird. Da bei einem eingeschalteten RCX immer mindestens 5 Volt an jedem der Sensor-Eingänge und damit dann auch am Acknowledge-Ausgang des Lepomux anliegen, läßt sich diese Spannung zur Steuerung des Enable-Eingangs verwenden. Allerdings muß dabei beachtet werden, daß diese Spannung nie größer sein darf als die am Spannungsregler anliegende Batteriespannung. Um diese Voraussetzung auch bei leeren Batterien erfüllen zu können, wird eine eventuelle Überspannung über die Schottky-Diode D5 in Richtung der Batteriespannung abgeleitet.

Damit nicht jeder Schaltimpuls auf dem RCX-Eingang zum Ein- oder Ausschalten des Lepomux führt, wird über die Doppeldiode D3 der Kondensator C10 geladen. Zusammen mit den Widerständen R10 und R11 bildet er eine Ausschaltverzögerung. R10 dient dabei der Strombegrenzung auf weniger als 10 µA für den Fall einer Überspannung am Enable-Eingang.

72 Alle in diesem Kapitel angesprochenen Module können direkt mit der Standard-Fimware angesteuert werden. Ein Austausch bzw. eine Erweiterung dieser Software ist nur in besonderen Fällen nötig.

73 Für die 5-Volt-Versorgung wird hier ein TPS76150 die Firma Texas Instruments benutzt.[29] Dies ist ein Low-Drop-Regler, der einen maximalen Ausgangsstrom von 100 mA liefern kann. Der 3,3-Volt-Zweig wird über einen TPS76033 versorgt.[30] Dieser Regler ist im Vergleich zu den TPS761xx-Versionen nur für einen Strom von 50 mA spezifiziert. Eine Besonderheit dieser beiden Spannungsregler ist der Enable-Eingang.

4.1.1.2 Prozessor

Der Mikrocontroller würde mit seinem ca. 13 x 10 Millimeter großen SO-20-Gehäuse einen nicht unwesentlichen Teil der Leiterplatte beanspruchen. Glücklicherweise ist der MSP-430F1121A derzeit in drei unterschiedlichen Gehäusevarianten verfügbar.[31] Die PW-Bauform reduziert den Platzbedarf des Prozessors von 1,3 mm² auf 0,42 mm².[74] Allerdings führt dies dazu, daß eine manuelle Platinenbestückung damit fast unmöglich wird, denn die Pins im Abstand von 0,65 mm sind mit handelsüblichem Lötwerkzeug kaum zu bewältigen.

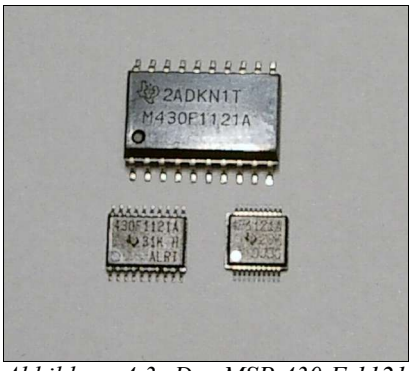


Abbildung 4.3: Der MSP 430 F 1121 A in drei verschiedenen Bauformen

4.1.1.3 Modul-Anschluß

Das ursprüngliche Stecksystem der Vorgängerversionen diente ausschließlich dem Anschluß von Sensoren und Aktoren. Bedingt durch die Trennung des Controllers von den restlichen Komponenten verändert sich auch das Stecksystem.

Über die 14-polige Pfostenwanne CN1 sind die wichtigsten I/O-Leitungen des MSP aus dem Gehäuse geführt. Die für den Multiplexer vorgesehenen Anschlüsse sind dabei bereits auf einen Pegel von 10 Volt angehoben. Zusätzlich sind alle Versorgungsspannungen des Controller-Boards auf diesem Anschluß abgreifbar.

Durch diese universelle Beschaltung können fast beliebig geartete Erweiterungsmodule entweder direkt oder per Flachbandkabel auf das Lepomux-Gehäuse gesteckt werden.

4.1.1.4 RCX- / Programmier-Anschluß

Abbildung 4.4: Modul- und RCX-/Programmieranschluß des Lepomux v3.0

Die Verbindung zum RCX wird, wie in Abbildung 4.4 zu sehen ist, über eine vierpolige Federleiste hergestellt, die auch von der Oberseite des Lepomux-Batteriekastens aus zugänglich ist. Je zwei Pole dieser Leiste bilden den RCX-Dateneingang und den RCX-Acknowledge-Ausgang[75]. Sie werden über LEGO-Kabel direkt mit dem RCX verbunden.

Direkt neben der oben genannten Federleiste befindet sich noch ein zweipoliger Stiftleiste, an dem

74 Die dritte Bauform nennt sich DGV und ist nur geringfügig kleiner als das PW-Gehäuse. Der resultierende Flächengewinn von 0,1 mm² hätte einen Pinabstand von 0,4 mm zur Folge gehabt, womit eine manuelle Verarbeitung selbst für Prototypen praktisch ausgeschlossen gewesen wäre.

75 Diese beiden RCX-Verbindungen sind in den Kapiteln 2.1.2 und 2.1.3 beschrieben.

die bei der Programmierung des Mikrocontrollers nötigen Signale RESET und TEST anliegen. Weitere Informationen zur Programmierung finden sich in Kapitel 4.1.5.

4.1.2 Sensor-Board

Diese Platine beinhaltet einen Großteil an Komponenten, für die auf dem Controller-Board der Platz zu knapp wurde. Das betrifft hauptsächlich den Sensor-Multiplexer, der bisher so ausgelegt war, daß er acht normale LEGO-Sensoren an einem RCX-Eingang betreiben konnte.

Die Aufgabenstellungen des RoboLab erfordern dabei üblicherweise die Verwendung diverser Sensoren:

- Lichtsensoren zur Erkennung reflektierender Objekte oder verschiedenfarbiger Untergründe
- Shaft-Encoder zum Aufnehmen zurückgelegter Wegstrecken oder als Winkelgeber von Schußmechanismen
- Schalter zur Berührungs- oder Drehwinkelerkennung
- Distanzsensoren zur berührungslosen Abstandsmessung oder Objekterkennung
- Beacon-Sensoren zur Erkennung von IR-Leuchtfeuern

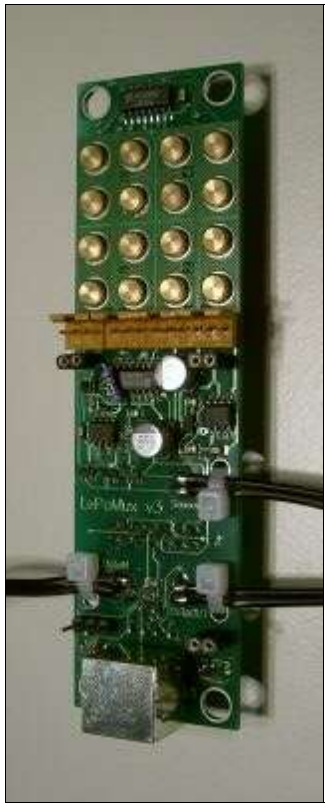


Abbildung 4.5: Sensor-Board

Hierbei nehmen die Distanzsensoren eine Sonderstellung ein, da sie erst durch eine Interfaceschaltung LEGO-tauglich werden. Üblicherweise benötigtein Roboter drei bis vier dieser Sensoren. Da der Multiplexer LEGO-kompatible Eingänge zur Verfügung stellt, muß demnach jeder einzelne Distanzsensor mit solch einer Interfaceschaltung ausgestattet werden.

Bedenkt man, daß der Multiplexer nie mehr als einen Sensor gleichzeitig ansteuern kann, würde für die Verwendung von Distanzsensoren eine einzige Interfaceschaltung vollkommen ausreichen. Es liegt also nahe, diese Schaltung in den Lepomux zu integrieren. Das würde aber wiederum bedeuten, daß die Sensoreingänge ausschließlich zur Verwendung von Distanzsensoren benutzt werden können.

Durch eine geringfügige Modifikation des Multiplexers ist es aber möglich, vier der acht Eingänge für Distanzsensoren zu benutzen und die andere Hälfte weiterhin LEGO-kompatibel zu belassen.

4.1.2.1 Sensor-Multiplexer

Die ursprüngliche Schaltung des Sensor-Multiplexers sah vor, zwei Bausteine des Typs ***74 HC 4051*** zu benutzen, um jeweils einen von acht Eingangskanälen auf einen Ausgang zu schalten[76].

76 Aus den Kapiteln 2.1.4 und 2.3.1 läßt sich die genaue Schaltungsgrundlage entnehmen.

Der hier verwendete ***74 HC 4052*** (U1) beinhaltet dagegen zwei synchron gesteuerte 4-zu-1-Multiplexer. Sie dienen zum transparenten Umschalten von maximal vier LEGO-Sensoren. [32]

Wie bisher wird die Multiplexer-Sektion über drei Portleitungen (MUX0-2) des MSP gesteuert. Das höchstwertigste Bit steuert dabei den Inhibit-Eingang des ***74 HC 4052*** und sorgt dafür, daß in der oberen Hälfte des Wertebereichs keiner der Sensoren auf den Ausgang durchgeschaltet wird. Stattdessen ist in diesem Fall der im Folgenden beschriebene Multiplexer für die Distanzsensoren aktiv.

4.1.2.2 Sharp-Distanzsensor-Mux

Die bereits in Kapitel 1.2.4 angesprochenen Distanzsensoren ***GP2D12*** der Firma Sharp sind in der Lage, unter Verwendung modulierter Infrarotstrahlung Abstände zu beliebigen Objekten mit Entfernungen von 10 bis 80 Zentimentern zu messen. Der Objektabstand wird dabei als analoger Meßwert mit einem Spannungspegel von 0 bis ca. 2,7 Volt ausgegeben.

An die Eingänge eines weiteren ***74 HC 4052*** (U2) lassen sich bis zu vier dieser Sensoren anschließen. Dazu dienen dreipolige, verpolungssichere Steckverbinder. Das Ausgangssignal der Distanzsensoren bezieht sich auf das Massepotential der Stromversorgung. Somit muß pro Sensor nur ein Pin umgeschaltet werden.

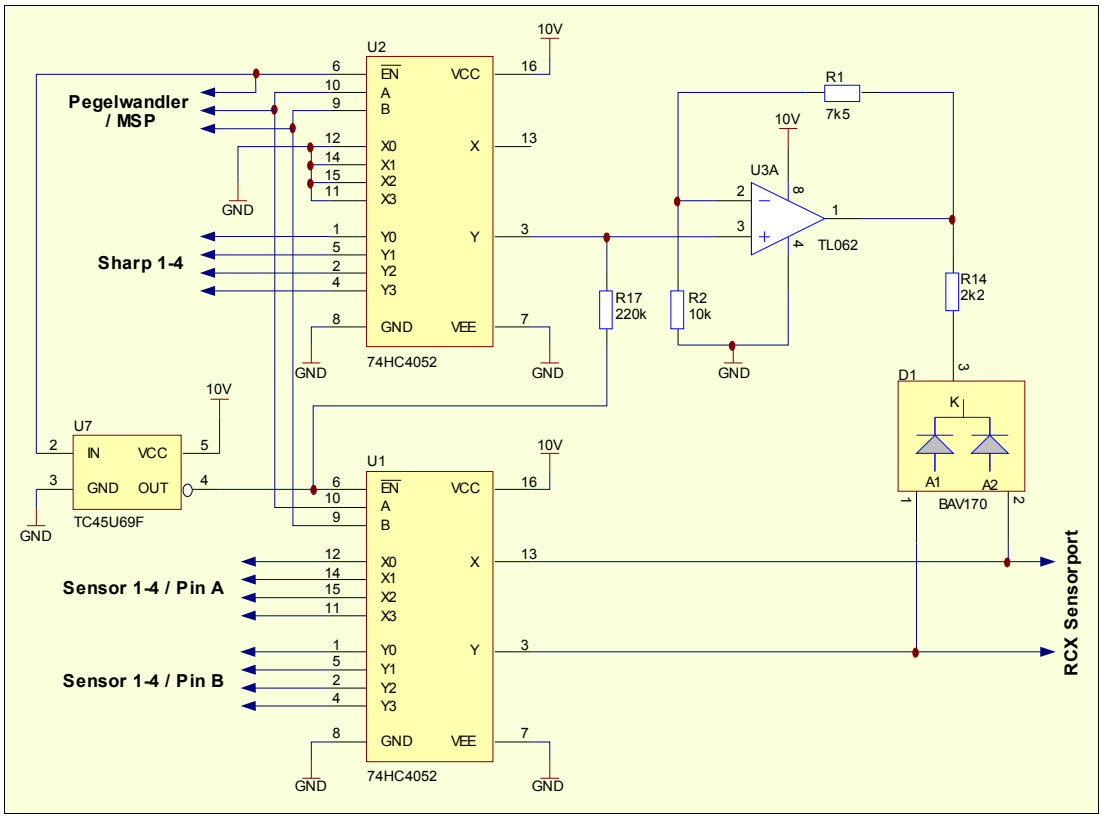


Abbildung 4.6: Schaltung des kombinierten LEGO- und Distanz-Sensor-Multiplexers

Der Baustein U2 darf im Gegensatz zu U1 nur in der oberen Hälfte des durch die drei Portleitungen steuerbaren 3-Bit-Bereichs aktiv sein. Der Inverter U7 dient dabei zur Aktivierung des Multiplexers U2 über seinen Inhibit-Eingang[77], sofern an der

77 Das Signal Inhibit entspricht einer low-aktiven Enable-Funktion. Demnach wird der Ausgang des

höchstwertigsten Steuerleitung (MUX2) ein High-Pegel anliegt.

Um die Ausgangssignale der Distanzsensoren für unsere Zwecke nutzen zu können, ist eine Interfaceschaltung zwingend nötig. Sie besteht aus einem Operationsverstärker U3, der dafür sorgt, daß das Signal bei Belastung nicht zusammenbricht. Der Verstärkungsfaktor wird dabei über das Verhältnis von Widerstand R2 zu Widerstand R1 festgelegt. Es ist so dimensioniert, daß der Ausgangsspannungsbereich des Sensors auf den Wertebereich des RCX-Eingangs skaliert wird. Der zur Strombegrenzung dienende Widerstand R14 und die Verpolungsschutz-Diode D1 machen allerdings einen Teil des Wertebereichs zunichte.[78]

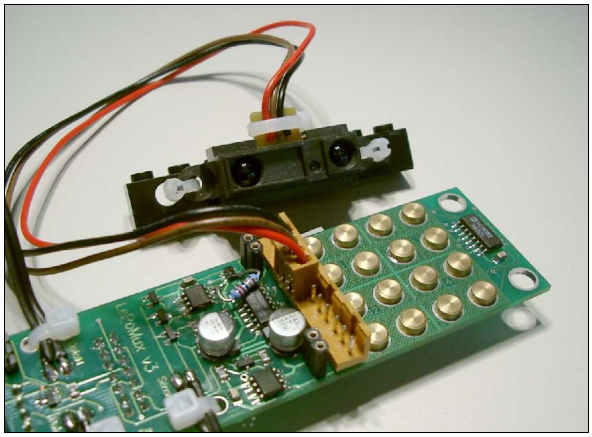


Abbildung 4.7: Dreipolige Steckverbinder für Sharp-Distanzsensoren

Verglichen mit anderen Sensoren sind die Distanzmesser wahre Stromfresser. Bedingt durch den integrierten IR-Sender benötigt jeder dieser Sensoren einen Strom von ca. 50 mA bei 5 Volt. Da der Spannungsregler des Controller-Boards nur für 100 mA ausgelegt ist, wird auf dem Sensor-Board ein getrennter Regler (U8) für die Sharp-Sensoren benötigt. Dieser Baustein des Typs ***TPS 7250*** liefert bis zu 250 mA. [33]
Damit die in Kapitel 4.1.1.1 angesprochene automatische Ein-/Ausschaltung funktioniert, muß auch dieser Spannungsregler abschaltbar sein. Dies ist beim ***TPS 7250*** zwar der Fall, aber im Gegensatz zu den Reglern des Controller-Boards, besitzt dieser Baustein einen low-aktiven Enable-Eingang.
Die nur im eingeschalteten Zustand anliegenden 5 Volt des Controller-Boards werden über den Transistor T1 invertiert. Bei ausgeschaltetem Lepomux sorgt R16 für einen definierten High-Pegel am Enable-Pin von U8.

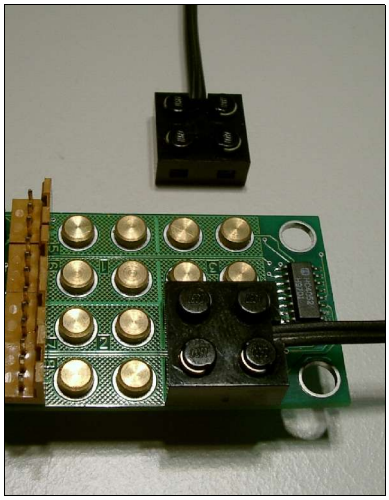


Abbildung 4.8: LEGO-Kontakte aus Messingrollen

4.1.2.3 IR-Eye-Connector

Diese 6-polige Mini-DIN-Buchse dient zum Anschluß der IR-Sensor-Platine (IR-Eye). Im Vergleich zu der bisher verwendeten 5-poligen Buchse, findet sich die hier benutzte Bauform in fast jedem PC. Somit lassen sich kostengünstige, vorkonfektionierte Kabel zur Verbindung der beiden Leiterplatten verwenden.

4.1.2.4 LEGO-Steckkontakte

Der in Kapitel 3.3.3.2 beschriebene Ansatz für kostengünstige LEGO-Buchsen war leider nicht wie geplant realisierbar. Nach der Fertigung der Platinen stellte sich heraus, daß die Toleranzen bezüglich der

Multiplexers beim Anlegen eines High-Pegels an den Inhibit-Eingang hochohmig.

78 Der Grund für die Verfälschung des Wertebereichs ist detailliert in Kapitel 1.4.4 beschrieben.

Lötaugendurchmesser zu groß sind, als daß sich Lötaugen als Kontaktflächen verwenden lassen.

Um die Platinen dennoch benutzen zu können, wurden kleine Messingrollen in die Lötaugen der LEGO-Kontakte gelötet. Dabei sind die Oberseiten dieser Messingstücke zuvor auf ein passendes LEGO-Maß gedreht worden. Abbildung 4.8 zeigt die fertig umgerüsteten Kontakte.

4.1.2.5 Motorboard-Anschluß

Das Sensor-Board bietet nicht genug Platz, um auch noch die Motortreiber inklusive der zugehörigen LEGO-Kontakte darauf unterzubringen. Aus diesem Grund wurden alle Schaltungsteile, die mit dem Schieberegister der Motorausgänge zu tun haben, auf das unten beschriebene Zusatzmodul ausgelagert. Das Sensor-Board läßt sich auch ohne dieses Modul betreiben.

Angeschlossen wird das sogenannte Motor-Board über vier Stift- bzw. Federleisten, die so auf der Platine verteilt sind, daß sie gleichzeitig als mechanische Eckpfeiler dienen. Auf diesen Anschlüssen liegen verschiedene Versorgungsspannungen und die Steuersignale für das Schieberegister.

4.1.3 Motor-Board

Dieses Modul ist ein Aufsatz für das Sensorboard. Es kann nicht direkt auf den Lepomux aufgesteckt werden.

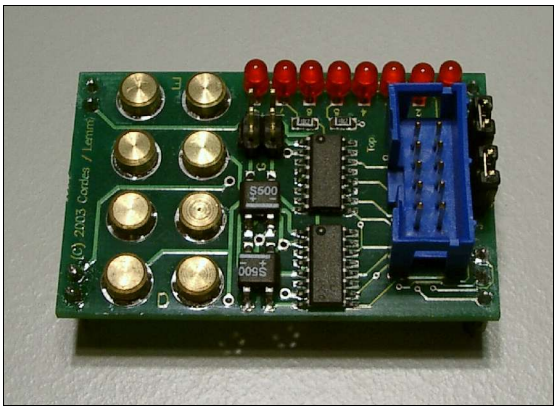


Abbildung 4.9: Motor-Board

Zentraler Bestandteil des Motor-Boards ist das Schieberegister, das in früheren Versionen auf der Lepomux-Platine untergebracht war. Angesteuert wird es über die drei Steuerleitungen SDA, SCL und SWR, die den Pins P1.5 bis P1.7 des MSP entsprechen. Diese Signale werden vom Sensor-Board auf das Motor-Board durchgeschleift. Über die Ausgänge des Registers werden die folgenden Bestandteile des Boards angesteuert:

4.1.3.1 LEDs

Da das Schieberegister eine Breite von 8 Bit besitzt, ist für jeden Registerausgang eine Leuchtdiode vorgesehen. Wie in den Vorgängerversionen kommen auch hier Low-Current-LEDs zum Einsatz, deren Stromaufnahme über Vorwiderstände jeweils auf 1 mA begrenzt ist.

Über die beiden Jumper JP1 und JP2 lassen sich die LEDs des oberen bzw. unteren Nibbles bei Bedarf abschalten.

4.1.3.2 Motortreiber

Parallel zu den Leuchtdioden sind die Schieberegisterausgänge mit den Steuereingängen von vier Motortreibern verbunden. Die Treiber sind wie auch bei

den älteren Lepomux-Varianten mit einem Gleichrichter als Freilaufdioden-Ersatz beschaltet. Zur weiteren Platzersparnis wurde hier aber eine kleinere Bauform gewählt.

In die Leiterplatte des Motorboards sind zwei LEGO-Kontakte[79] integriert, die die Ausgänge der ersten beiden Motortreiber bilden. Die anderen beiden Motorausgänge sind über CN22 abgreifbar. Für den Fall, daß nur maximal zwei Motoren über den Lepomux gesteuert werden sollen, können die zusätzlichen beiden Treiber und die zugehörigen Gleichrichter entfallen.

4.1.3.3 Aux-Anschluß

Über die 10-polige Pfostenwanne CN8 sind die acht Ausgangsleitungen des Schieberegisters zusammen mit Masse und 5 Volt abgreifbar. Dieser Anschluß dient insbesondere der Ansteuerung von Modellbauservos. Die 5-Volt-Versorgung geschieht dabei über den Spannungsregler des Sensorboards (U8), der einen maximalen Ausgangsstrom von 250 mA liefern kann. Da auch die Sharp-Sensoren hierüber versorgt werden, stehen bestenfalls 50 mA für den Aux-Anschluß zur Verfügung, sofern alle 4 Sensoren in Betrieb sind.

4.1.4 Multi-Motor-Board

Wie bereits in Kapitel 3.3.3.3 kurz angesprochen, entstand dieses Lepomux-Modul aus dem Bauvorhaben für einen 6-armigen Roboter. Es soll hier nur der Vollständigkeit halber kurz beschrieben werden und demonstriert, wie leicht sich der Lepomux v3.0 mit unterschiedlichen Aufsteckmodulen an spezielle Aufgabenstellungen anpassen läßt.

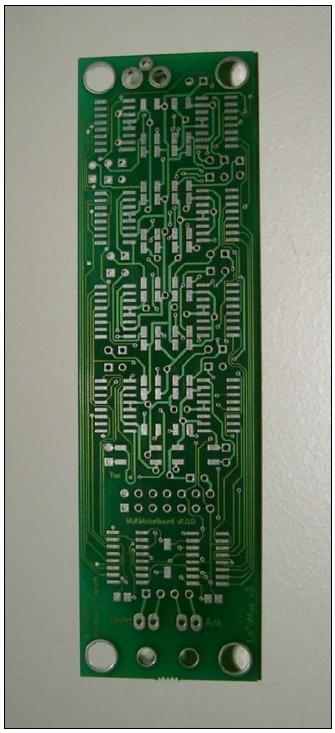


Abbildung 4.10: Das unbestückte Multi-Motor-Board

Das Modul entspricht einer eigenständigen, erweiterten Version des in Kapitel 4.1.3 beschriebenen Motor-Boards. Um insgesamt 16 Motorausgänge zur Verfügung stellen zu können, sind hier vier Schieberegister des Typs 74HCT595 kaskadiert. Sie verhalten sich aus Sicht der Programmierung wie ein 32-Bit-Schieberegister, denn ein Bit, das aus einem der Register herausgeschoben wird, gelangt durch die Kaskadierung an den Eingang des nächsten Registers.

Je zwei Ausgänge der Schieberegister sind an einen der bekannten Motortreiber MLX10402 angeschlossen und bilden jeweils einen Motorport. Da ein gleichzeitiger Betrieb von 16 Motoren einen nicht unerheblichen Stromfluß zur Folge hat, läßt sich per Jumper entscheiden, ob deren Speisung aus dem Lepomux-Batteriekasten oder aus einer externen Stromquelle geschehen soll.

Die Software gestattet es, jeden der 16 Motoren individuell über eine Pulsweitenmodulation in der Geschwindigkeit zu regeln und zusätzlich ihre Drehrichtung zu bestimmen[80].

79 Diese Kontakte entsprechen denen des Sensor-Boards. Siehe Kapitel 4.1.2.4.
80 Siehe Kapitel 4.2.3.

4.1.5 Programmieradapter

Dieser Adapter ist nichts weiter als ein spezieller Steckverbinder, der die Anschlüsse des Lepomux-Controllerboards mit der Pfostenleiste des Flash-Emulation-Tools (FET) verbindet. Das FET gehört zum MSP-430-Evaluation-Kit und dient normalerweise dazu, unverlötete Prozessoren über einen Nullkraft-Sockel zu programmieren oder zu debuggen. Die Programmierung bereits eingebauter MSP-Bausteine kann über Pfostenstecker erfolgen, auf denen die Anschlüsse des Sockels durchgeschleift sind.

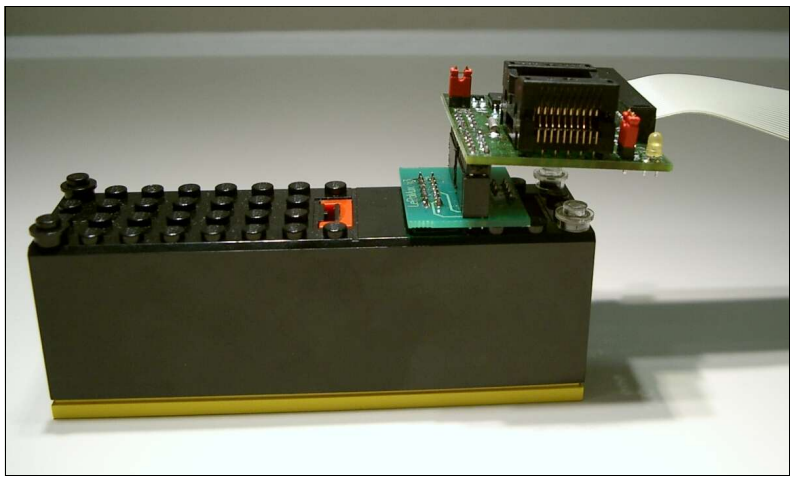


Abbildung 4.11: FET mit Programmieradapter auf Lepomux v3.0

Zur Programmierung des MSP müssen die sechs Signale P1.4 bis P1.7, RST und TEST mit dem FET verbunden sein. Wichtig ist hierbei, daß die Signalpegel nicht durch parallel angeschlossene Schaltungsteile beeinflusst werden. Im unserem Fall ist das unkritisch, da die betroffenen Portleitungen erst über ein Aufsteckmodul an eine bestimmte Funktion gebunden sind.

4.2 Software

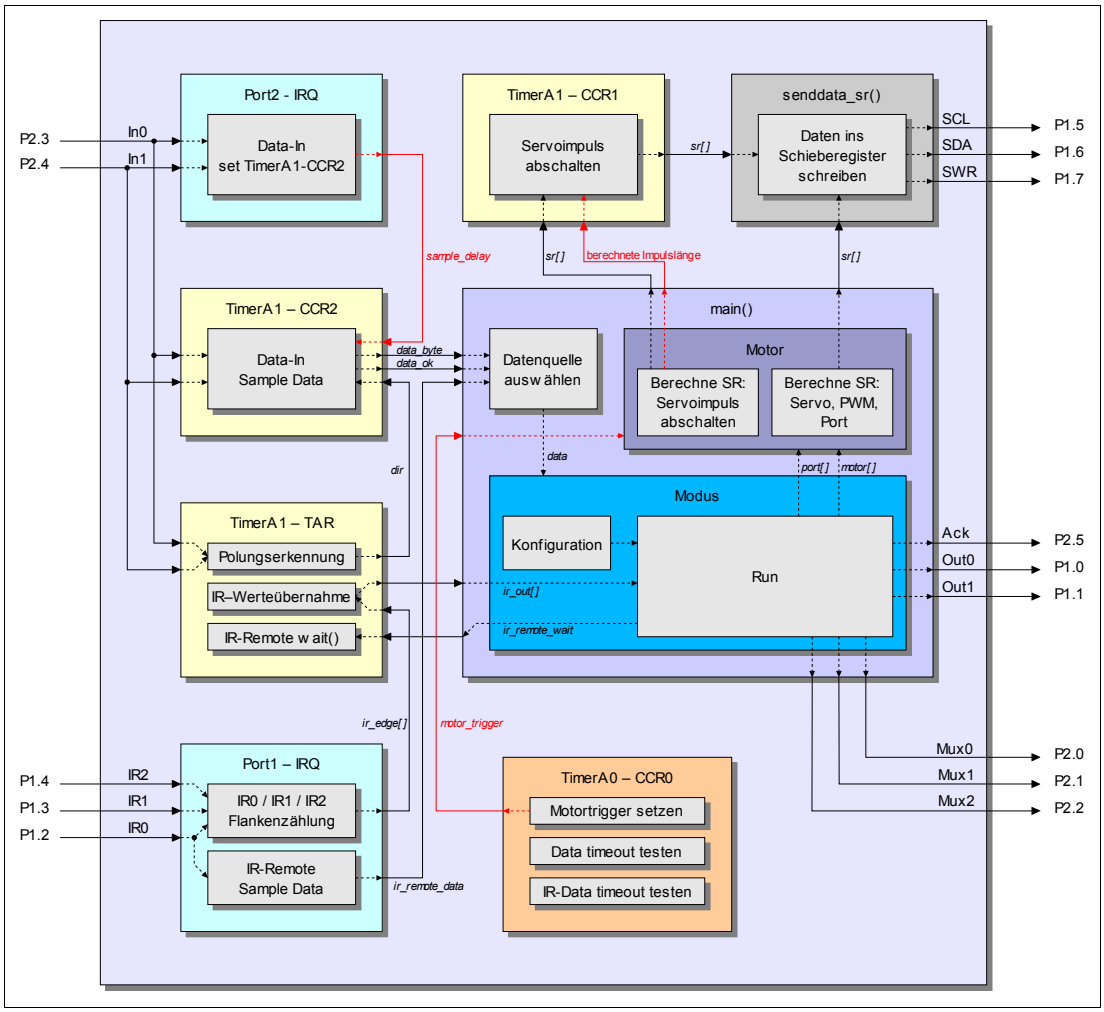


Abbildung 4.12: Blockdiagramm der Software v3.0

In der dritten Version der Lepomux-Software sind viele neue Funktionen hinzugekommen. Das betrifft vor allem die Möglichkeit, Modellbau-Servos anzusteuern und Motordaten auf bis zu vier kaskadierten Schieberegistern auszugeben. Außerdem ist der Lepomux nun in der Lage, Fernbedienungssignale mit Hilfe des normalerweise zur Beacon-Erkennung benutzen IR-Detektors zu dekodieren.

Die Abbildung 4.12 bietet eine Übersicht der dafür nötigen Programmstruktur. Auf die einzelnen Funktionsblöcke wird in den folgenden Kapiteln eingegangen.

4.2.1 Datenempfang

Die Auswertung der vom RCX gesendeten Datenworte findet nach dem bereits in der Version 2.0 benutzten Prinzip statt.[81] Allerdings sind trotzdem einige kleinere Änderungen vorgenommen worden, die die Flankenerkennung vom eigentlichen Datenempfang entkoppeln. In der Port2-Interrupt-Routine wird dabei nur noch die Wartezeit[82] bis zum Sampling der Daten in das Compare-Register CCR2

81 Siehe Kapitel 3.2.1.

82 Diese Zeitspanne wird über die Variable *sample_delay* festgelegt und beträgt normalerweise ca.

geschrieben. Nach Ablauf dieser Zeitspanne wird somit ein weiterer Interrupt ausgelöst, der die eigentliche Auswertung der Dateneingänge startet. Die dazu verwendete Routine entspricht weitgehend dem in Kapitel 3.1.1 beschriebenen Code.

4.2.2 IR

Die Infrarotempfänger-Auswertung ist um die Erkennung von Fernbedinungssignalen erweitert worden. Grund hierfür waren Artikel im Roboter-Forum des LUGNET, wo sich verschiedene Personen zu einer RCX-Erweiterung der Firma Mindsensors [13] äußerten. Offenbar sind viele LEGO-Bastler daran interessiert, ihre Modelle per IR-Fernbedienung zu steuern. Bezogen auf den Lepomux ergibt sich aus diesem Ansatz zum Beispiel die Möglichkeit, vordefinierte Aktionen ohne eine Programmierung des RCX ablaufen zu lassen.
Da die bereits für die Erkennung der IR-Beacon vorhandenen Sensoren auch zum Empfang der Ferbedienungscodes benutzbar sind, ließ sich diese Funktions-erweiterung ohne Eingriffe in die Hardware umsetzen. Die Aktivierung der Beacon-Erkennung bei gleichzeitiger Dekodierung von Fernbedienungssignalen ist allerdings nicht möglich. Zwar hätte dies aus Sicht des Prozessors kein Problem dargestellt, aber bedingt durch die ähnlichen Modulationsfrequenzen würden sich IR-Kommandos und Beacon-Signale überlagern.

4.2.2.1 IR-Beacon-Erkennung

Die Umsetzung der Beacon-Erkennung hat sich seit der Version 1.0 als zuverlässig und stabil erwiesen und wurde deshalb praktisch ohne Veränderungen übernommen.[83] Eine ausführliche Beschreibung dieses Themas ist in den Kapiteln 2.1.5, 2.2.1.6 und 2.2.1.8 nachzulesen.

4.2.2.2 IR-Remote Control

Für die Übermittlung von Fernbedienugnssignalen auf der Basis modulierter Infrarotstrahlung haben viele Hersteller von Home-Entertainment-Geräten eigene Lösungen entwickelt. Die verschiedenen Übertragungsprotokolle wie beispielsweise RC5 von Philips, SIRCS der Firma Sony oder der Denon-Code benutzen allesamt Modulationsfrequenzen im Bereich von 36 bis 48 kHz. [34]

Besonders populär ist dabei der RC5-Code, da er sich mit relativ einfachen Mitteln dekodieren läßt. Es ist deshalb auch bezogen auf den Lepomux das Protokoll der Wahl.

Das RC5-Protokoll ermöglicht das Adressieren von 32 Geräten mit je 64 Befehlen. Der Aufbau besteht aus zwei Startbits, einem Toggle-Bit, fünf Adressbits und sechs Befehlsbits. Die Abbildung 4.13 gibt einen Überblick, wie die Bitverteilung und das Timing zusammenhängen.

In einer neueren Version wurde das Protokoll erweitert und das zweite Startbit als invertiertes Befehlsbit verwendet. Somit lassen sich bis zu 128 Befehle versenden.

300 µs. Diese Zeit reicht aus, um auch für den Fall einer versetzt auftretenden Pegeländerung auf beiden Eingangsleitungen den korrekten Datenwert ermitteln zu können. Dieses Phänomen tritt beispielsweise bei einem Wechsel von *forward* zu *reverse* auf, da der Motortreiber im RCX dazu die Polung ändern muß.

83 Die genaue Funktionsweise der Beacon-Auswertung ist im Kapitel 2.2.1.6 beschrieben.

Dieser Modus wird allerdings von der Software des Lepomux nicht unterstützt.

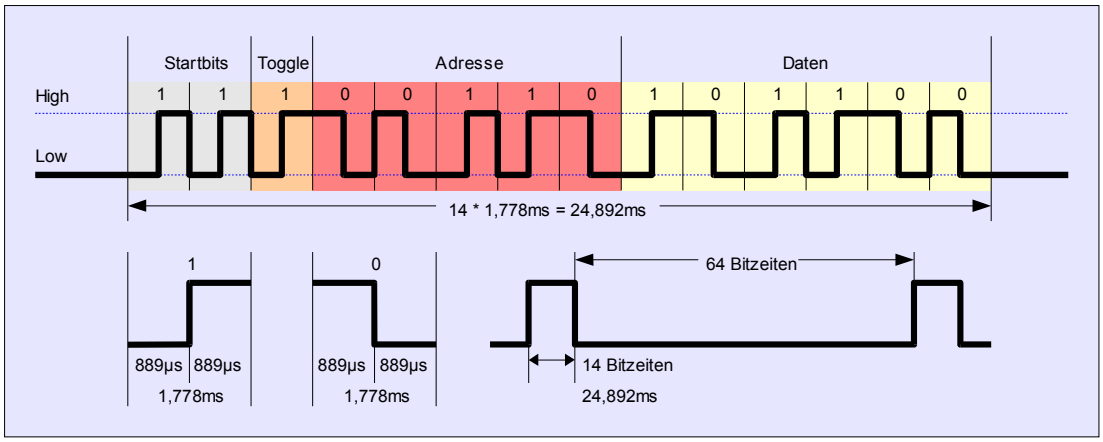


Abbildung 4.13: RC5 Protokoll

Das Toggle-Bit ermöglicht dem Empfänger zu erkennen, wann eine Kommando mit gleichem Adress- und Datencode durch einen erneuten Druck auf die Taste der Fernbedienung versandt wurde. Der Versuch, dies ohne ein entsprechendes Bit zu erreichen, indem geprüft wird, wann die Übertragung abreißt, scheitert spätestens bei einer durch ein Hindernis versperrten Sicht. Wird kurz darauf nochmal ein gleiches Datenwort empfangen, läßt sich nicht sagen, ob der Absender zwischenzeitlich die Taste losgelassen hat. Um dies zu ermöglichen, wechselt das Toggle-Bit bei einem erneuten Tastendruck seine Wertigkeit.

Der RC5-Code verwendet ein biphasencodiertes Format, was bedeutet, daß ein Bitwert durch einen Bitwechsel definiert wird. Steigende Flanken codieren dabei eine logische Eins, fallende eine Null. Die Gesamtlänge eines so kodierten Bits beträgt 1,778 ms, wodurch sich eine Gesamtzeit von 24,892 ms pro Übertragung ergibt. Die Pause zwischen zwei Übertragungen beträgt mindestens 64 Bitzeiten. Dadurch ergibt sich eine Übertragungszeit von ca 139 ms für aufeinanderfolgende Datenworte.

Die Dekodierung dieses Protokolls ist recht einfach, da es auf einer festen Bitzeit beruht. Die fallenden Flanken der beiden Startbits werden üblicherweise dazu benutzt, um mit Hilfe eines Capture-Interrupts die Bitzeit zu ermitteln. Danach reicht es aus, nach drei Vierteln dieser berechneten Bitzeit mit dem Sampling der Datenbits zu beginnen. Ab dort folgt jede weitere Bitzeit ein Sampling-Punkt bis das Ende des Datenwortes erreicht ist. Durch diese Methode liegt der Zeitpunkt der Datenübernahme immer in der vorderen Hälfte eines Bits. Die gelesenen Werte repräsentieren dabei gleichzeitig den eigentlichen Bitwert.

Wie anhand der Abbildung 4.12 leicht zu sehen ist, sind die drei Capture/Compare-Blöcke des MSP aber bereits für andere Aufgaben vergeben. Dennoch läßt sich die RC5-Erkennung realisieren – allerdings mit einer vollkommen anderen Vorgehensweise.

Betrachtet man die zeitliche Abfolge der Flanken innerhalb eines Datenwortes, so fällt ein Zusammenhang zwischen Flankenabstand und Bitänderung auf. Anhand der Abbildung 4.14 ist erkennbar, daß ausschließlich bei langen Flankenabständen ein Wechsel des Bitwerts stattfindet. Daraus ergibt sich, daß nur der Wert des Startbits bekannt sein muß, um das komplette Datenwort dekodieren zu können.

Bei der Auswertung ist zu beachten, daß der Ausgang des IR-Sensors ein invertiertes Signal liefert. Die Darstellung in Abbildung 4.14 berücksichtigt dieses Verhalten. Zur korrekten Dekodierung muß jede Flanke mit Hilfe eines Port-Interrupts registriert werden. Dazu ist eine Umschaltung des Flankentyps nach jedem Interrupt notwendig.[84]

Da die Timer-Module für andere Aufgaben benutzt werden, dient der Zählerstand des freilaufenden Timers TAR als Zeitbasis. Die Abstände der Flanken lassen sich damit durch eine einfache Subtraktion zweier aufeinanderfolgender Zählerstände ermitteln.

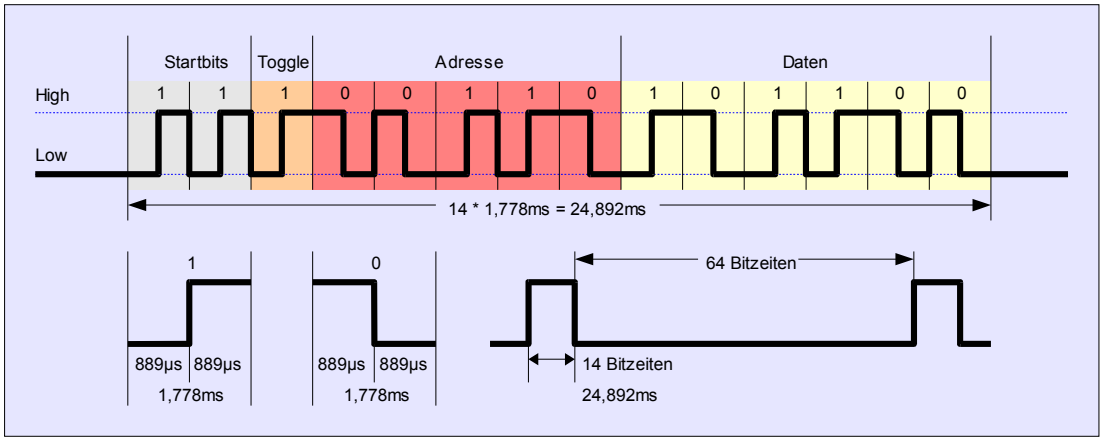


Abbildung 4.14: RC5 Protokoll

Die Halbbit-Zeit *t* auf diese Weise zwischen der ersten und zweiten, sowie der zweiten und dritten Flanke gemessen. Daraus errechnet sich die Zeit *T*, die den Mittelwert zwischen dem kurzen Flankenabstand *T1* und dem langen Flankenabstand *T2* bildet.

Da beide Startbits mit einer logischen Eins codiert sind, ergibt sich jedes weitere Bit aus einem Vergleich von *T* mit dem jeweils aktuell ermittelten Abstand zur vorangegangenen Flanke. Ist *T* dabei kleiner als dieser Abstand, so wechselt der Bitwert bezogen auf das vorangegangene Bit. Anderenfalls wird der letzte Bitwert für das aktuelle Bit übernommen. Um das Endes der Übertragung erkennen zu können, wird die Anzahl der empfangenen Bits berücksichtigt.

Nach dem vollständigen Empfang des Datenwortes wird die übermittelte Adresse mit der vorkonfigurierten Geräteadresse des Lepomux verglichen. Überlicherweise wird hier die für Experimente vorgesehene Adresse 0x07 benutzt. Je nach verfügbarer Fernbedienung ist aber eine Änderung problemlos möglich.[85]

Für den Fall, daß die Adressen übereinstimmen, wird der empfangene Datenwert als Index für eine Kommandotabelle benutzt. Über diese in Kapitel 4.2.5.3 beschriebene Tabelle ist es möglich, nahezu beliebige Kommandofolgen zur Steuerung des Lepomux zu generieren. Diese Befehle entsprechen dabei den sonst vom RCX empfangenen Datenworten.

84 Diese Notwendigkeit besteht auch im Zusammenhang mit der in Kapitel 2.2.1.6 beschriebenen Beacon-Erkennung.

85 Die Konfiguration wird in Kapitel 4.2.5.1 beschrieben.

4.2.3 Steuerung der Motorausgänge

Zur Steuerung der Motorausgänge wird je ein Byte mit Registerdaten im Array *motor []* verwendet. Je nach Anzahl der verwendeten Schieberegister stehen bis zu 16 Ports mit je 2 Leitungen zur Verfügung. Ein Motorport läßt sich in drei verschiedenen Modi betreiben. Standardmäßig erfolgt die Ausgabe konstanter Signalpegel im sogenannten Port-Modus. Das Port-Register-Array *port[]* dient dabei als Zwischenspeicher der auszugebenden Logikpegel. Zusätzlich ist aber auch das Generieren pulsweitenmodulierter Signale zur Geschwindigkeitsregelung von Motoren oder zum Ansteuern von Modellbau-Servos möglich.

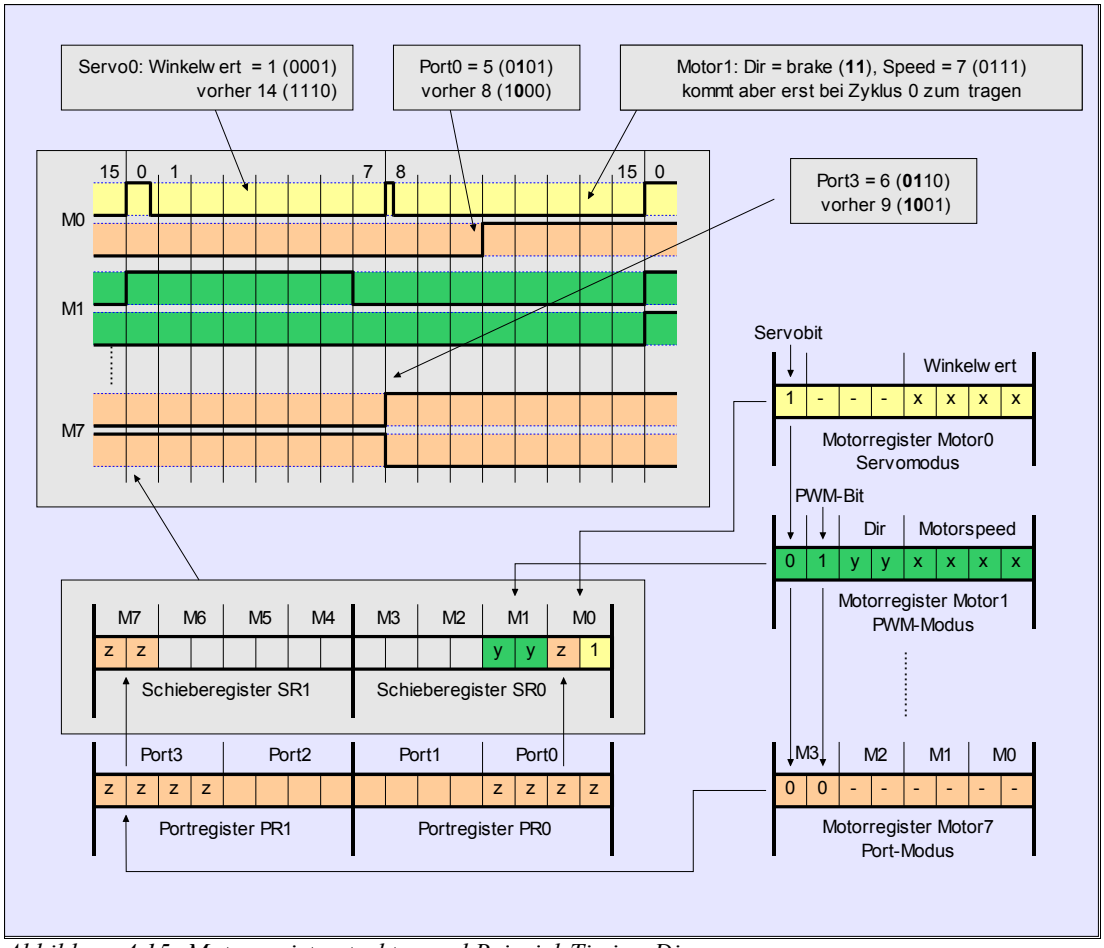


Abbildung 4.15: Motorregisterstruktur und Beispiel-Timing-Diagramm

Das gesamte Zeitverhalten der PWM-Ausgabe hängt vom Timer CCR0 ab. Er triggert die Verarbeitung der Motordaten innerhalb der *main*-Routine. In jedem Durchlauf werden dabei alle Motorausgänge abgearbeitet. Je nach Konfiguration der einzelnen Ports werden entsprechende Bits in das Ausgabe-Array *sr[]* geschrieben. Es wird alle 2,5 ms komplett neu berechnet und in die angeschlossenen Schieberegister übertragen. Nach sechzehn Einsprüngen in Ausgaberoutine ist ein PWM-Zyklus komplett abgearbeitet.

Abbildung 4.15 zeigt den Aufbau der im Array *motor[]* enthaltenen Motor-Register. Die beiden höchstwertigen Bits entscheiden darüber, welcher der drei möglichen Betriebsmodi aktiviert ist.

4.2.3.1 Servo-Modus

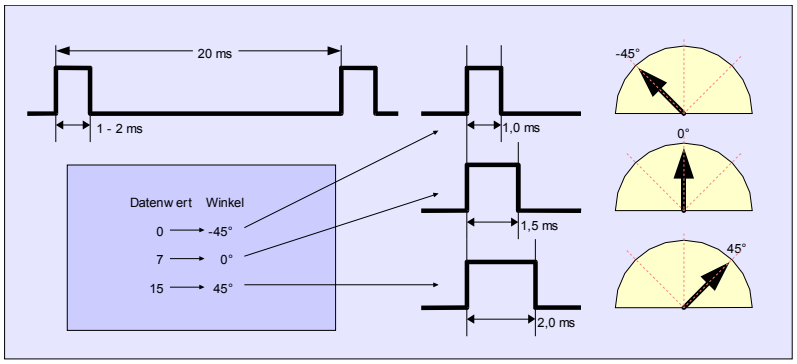


Abbildung 4.16: Servo-Timingdiagramm

Der Servo-Modus kann zeitgleich nur auf maximal acht Motorports benutzt werden, obwohl für die Steuerung eines Servos nur eine einzelne Steuerleitung benötigt wird.

Ein Servo erwartet alle 20 ms einen Steuerimpuls. Das Timing für die Pulsweite ist in Abbildung 4.16 zu sehen. Da normalerweise nicht alle Servos den gleichen Stellwinkel haben sollen, muß jedem Servo eine unterschiedliche Impulslänge übermittelt werden. Hierfür ist ein präzises Timing erforderlich, da sich bereits kleinste Änderungen der Impulslänge auf den Stellwinkel des Servos auswirken.

Um diese Vorgabe einhalten zu können, wird jeweils nur ein Servo pro Zyklus angesteuert. Bei der hier verwendeten Zykluszeit von 2,5 ms können somit maximal acht Servos angesprochen werden (20 ms / 2,5 ms = 8).

Ein Servo-Steuerimpuls hat eine definierte Länge von 1,5 ms für die Ausgangsstellung (0°). Um einen Stellbereich von -45° bis +45° abdecken zu können, ist ein Impuls von minimal einer Millisekunde bis maximal zwei Millisekunden nötig. Die 16 zu Verfügung stehenden Schritte des Datenwortes müssen dabei auf diesen 90°-Bereich aufgeteilt werden. Der Wert 8 entspricht der Mittelstellung des Servos.

Der für den Stellwinkel verwendete Wert wird dem zugehörigen Eintrag des *motor []*-Arrays entnommen. Die Motorgeschwindigkeit dient demnach an dieser Stelle als Winkelangabe.

Die Steuerung eines Servos erfordert jeweils nur einen der beiden Anschlüsse eines Motorports. Auf der unbenutzten Leitung wird das zugehörige Bit des Port-Registers ausgegeben.

Um das Servotiming einhalten zu können, wird der Timerwert für die Abschaltung des Servoimpulses direkt vor dem zeitintensiven Schreibzugriff auf die Schieberegister berechnet und in das Compare-Register CCR1 geschrieben.

Da die Abschaltung exakt zum Zeitpunkt des CCR1-Interrupts erfolgen soll, findet direkt nach dem Beschreiben der Schieberegister eine Neuberechnung der Bitwerte in *sr[]* statt. Da sich sich die Abschaltung des Steuerimpulses nur in einem Bit widerspiegelt, bleibt der Inhalt von *sr[]* aber weitestgehend unverändert.

Sofort nach dem Auslösen des CCR1-Interrupts wird der vorbereitete Inhalt von *sr[]* an die Schieberegister übertragen. Damit der Interrupt nicht nach einem Überlauf von TAR wiederholt ausgelöst wird, muß das zugehörige Interrupt-Enable-Flag nach erfolgter Servo-Abschaltung gelöscht werden.

Mit dieser Methode ist selbst bei einem 2 ms langen Servoimpuls noch genügend

Spielraum, um den Start des folgenden 2,5 ms Zyklus nicht zu gefährden. Jeder weitere Zyklus betrifft einen der anderen Servos.

4.2.3.2 PWM-Modus

Wenn ein Port im PWM-Modus geschaltet ist, werden bis zu 16 Geschwindigkeiten abgebildet. Da alle 2,5 ms die Daten neu ins Schieberegister geschrieben werden, entspricht das einer Gesamtzykluszeit von 40 ms (16 * 2,5 ms) für einen kompletten PWM-Durchlauf.

Der aktuelle Geschwindigkeitswert aus dem Motoregister wird mit dem Zykluszähler *cycle_counter* verglichen. Solange der Zählerwert nicht größer als der Geschwindigkeitswert ist, wird die ebenfalls im Motor-Register untergebrachte Drehrichtungsinformation in die entsprechenden Bits von *sr[]* geschrieben. Anderenfalls werden beide Bits des Motorports auf Null gesetzt.

Eine entsprechende Auswertung erfolgt pro Einsprung in die Ausgabe-Routine für jedes der Motor-Register.

4.2.3.3 Port-Modus

Wenn weder der Servo- noch PWM-Modus aktiviert ist, werden die beiden korrespondierenden Bits aus dem Portregister in das Ausgangsregister *sr[]* kopiert.

4.2.4 Ansteuerung der Komponenten

Wie schon in den beiden Vorgängerversionen werden die einzelnen Komponenten des Lepomux über die Aufteilung der RCX-Daten in einen vier Bit großen Adreßteil und einen eben so großen Datenteil angesprochen.

Zur bisherigen Software sind einige Komponentenadressen hinzugekommen, die mit der in Kapitel 4.2.2 beschriebenen Dekodierung von IR-Fernbedienungssignalen zusammenhängen. Das betrifft unter anderem Funktionen, die dazu dienen Einstellungen im 256-Byte großen User-Flash-Speicher des MSP abzulegen.[86]

Die Abbildung 4.17 bietet eine Übersicht der vorhandenen Komponenten und ihrer Programmierung. Um dabei die Übersichtlichkeit zu erhöhen, werden die Adressen nicht mehr als absolute Werte angegeben. Stattdessen sind dort symbolische Namen eingetragen, deren Wertigkeit sich dem Programmcode in Anhang entnehmen läßt.

4.2.4.1 Sensor-Multiplexer

Bei der Steuerung des Multiplexers geschieht die Auswahl des Sensors über die Adresse *ADDR_MUX*. Die unteren drei Bit des Datenwertes entsprechen dem gewählten Sensor. Dabei entfallen Sensornummern der Wertigkeit null bis drei auf die LEGO-kompatiblen Sensoren. Die Werte vier bis sieben aktivieren dagegen einen der vier Sharp-Distanzsensoren.

86 Die Umstrukturierung des Programmcodes zur Unterstützung der RC5-Dekodierung hat ein erhebliches Maß zusätzlicher Komplexität mitsich gebracht, weshalb bisher nur die Programmteile vollständig implementiert wurden, die zum normalen Betrieb des Lepomux nötig sind. Der Vollständigkeit halber werden in den Kapiteln 4.2.4.5, 4.2.4.6, 4.2.5.2 und 4.2.5.3 aber auch die bisher noch unvollständig umgesetzten Funktionsblöcke angesprochen.

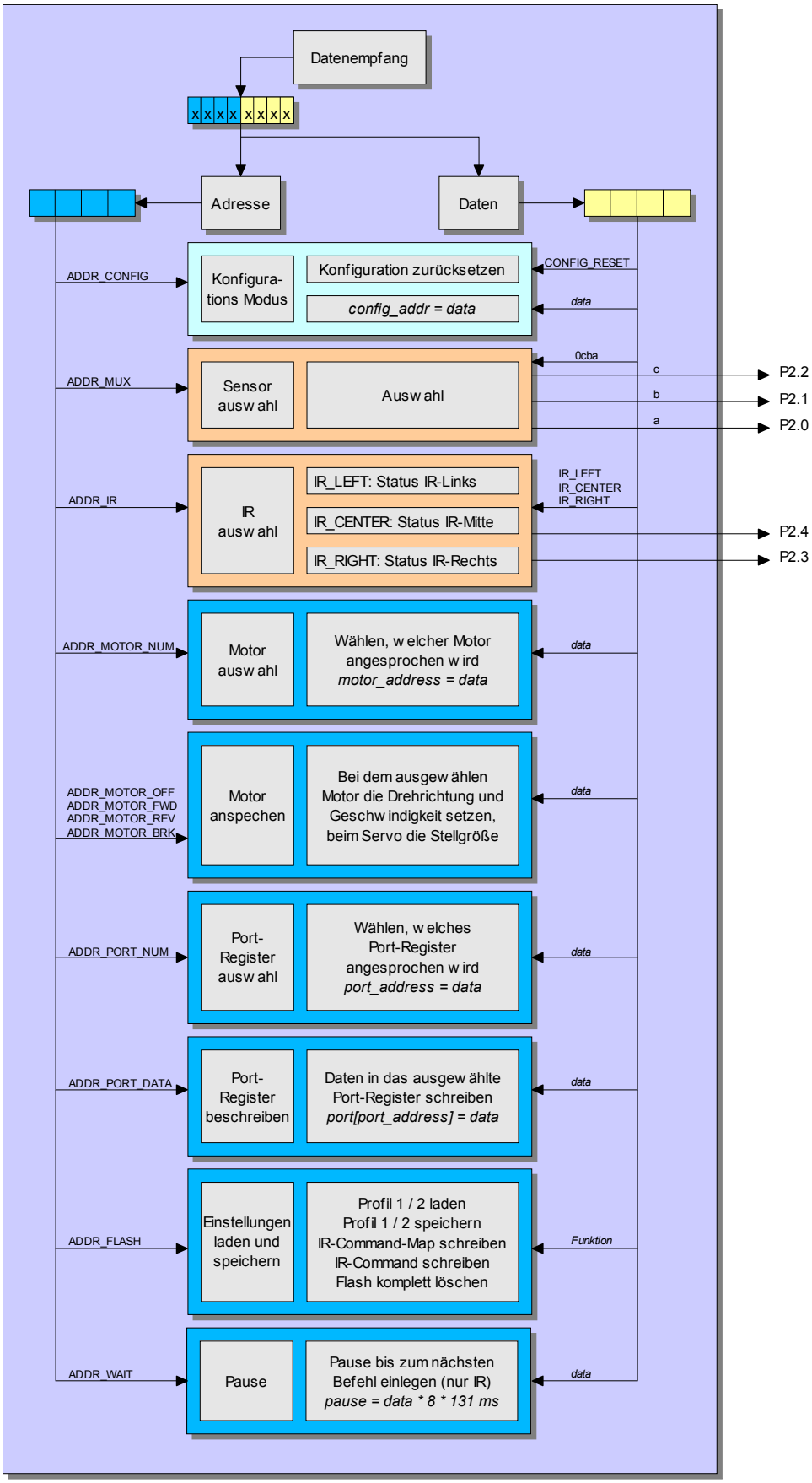


Abbildung 4.17: Komponentenadressierung

An der damit verbundenen Ansteuerung der Portleitungen hat sich im Vergleich zu den Vorgängerversionen nichts verändert. Die Unterscheidung zwischen den beiden Multiplexer-Sektionen geschieht per Hardware.[87]

4.2.4.2 IR-Beacon-Detektor

Auch die IR-Sensor-Auswahl funktioniert analog zur Vorgängerversion. Der in Zusammenhang mit der Adresse *ADDR_IR* übergebene Datenwert bestimmt dabei, welcher der drei Frequenzwerte auf den Acknowledge-Ausgang gelegt wird.

4.2.4.3 Motor-/ Servo-Steuerung

Bisher waren für jeden Motorport zwei Register reserviert, über die sich Geschwindigkeit und Drehrichtung eingestellen ließen. Bei einer inzwischen maximalen Anzahl von 16 Motorausgängen, wären die vier möglichen Adressbits schnell ausgeschöpft gewesen. Die neue Motoradressierung funktioniert dagegen deutlich resourcenschonender.

Um einen Motor oder einen Servo anzusteuern, muß der RCX wie gewohnt zwei Byte senden. Zuerst wird dabei der zu benutzende Motorport über die Adresse *ADDR_MOTOR_NUM* ausgewählt. Danach kann über die vier Adressen *ADDR_MOTOR_OFF*, *ADDR_MOTOR_FWD*, *ADDR_MOTOR_REV* und *ADDR_MOTOR_BRK* sowohl die Drehrichtung, als auch die Geschwindigkeit mit einem einzelnen Zugriff festgelegt werden.

Je nach Anzahl der angeschlossenen Schieberegister und Motortreiber stehen bis zu 16 programmierbare Motorports zur Verfügung, die aus je zwei zusammengefassten Portleitungen bestehen. Der Modus, in dem die Motorports betrieben werden sollen, läßt sich über die in Kapitel 4.2.5.1 angesprochenen Konfigurationsregister einstellen. Dabei kann individuell entschieden werden, ob unmodulierte Bitwerte oder pulsweitenmodulierte Signale zur Ansteuerung von Motoren oder Modellbau-Servos ausgegeben werden sollen.

4.2.4.4 Port-Zugriff

Mit der oben genannten Methode lassen sich zwar Motoren und Servos bequem benutzen, aber eine einzelne Ansteuerung von Leuchtdioden darüber zu realisieren ist eine unnötig aufwändige Prozedur. Aus diesem Grund gibt es den sogenannten Port-Modus, mit dem die Ausgänge der angeschlossenen Schieberegister direkt beeinflusst werden können.

Ähnlich wie bei der Adressierung der Motoren werden hier über eine Port-numerierung die zu ändernden Portleitungen ausgewählt. Dazu erfolgt zuerst ein Zugriff auf die Adresse *ADDR_PORT_NUM*, mit dem eines der maximal acht ansteuerbaren Nibbles (zwei pro Schieberegister bei maximal vier Registern) selektiert wird. Danach wird jedes Nibble, das an die *ADDR_PORT_DATA*-Adresse übertragen wird, direkt auf den zuvor gewählten 4-Bit-Port ausgegeben.

Allerdings ist dabei zu beachten, daß eine Ausgabe dieser Port-Daten nur dann erfolgt, wenn ein Port nicht durch die Servo- oder PWM-Funktion überlagert wird.[88]

87 Siehe Kapitel 4.1.2.1 und 4.1.2.2.
88 Siehe Kapitel 4.2.3.

4.2.4.5 Flash-Funktionen

Über die Adresse ADDR_FLASH sind Funktionen zum Laden und Speichern von Konfigurationsprofilen und zur Änderung der Komandotabelle für IR-Fernbedienungscodes erreichbar. Der auszuführende Flash-Befehl wird dabei über den Datenwert kodiert. Da sich diese Befehle ausschließlich auf die Konfiguration des Lepomux beziehen, werden sie in den Kapiteln 4.2.5.2 und 4.2.5.3 näher erläutert.

4.2.4.6 Wait-Kommando

ADDR_WAIT ist ein Pseudo-Kommando, das sich ausschließlich bei der Steuerung des Lepomux per IR-Fernbedienung auswirkt. Es sorgt dafür, daß die Ausführung von Kommandolisten für eine bestimmte Zeit angehalten wird. Somit wird das Zusammenstellen längerer Kommandosequenzen ermöglicht, deren Ausführung über einen einzigen Tastendruck an der Fernbedienung angestoßen wird. Auf die Definition solcher Sequenzen wird in Kapitel 4.2.5.3 eingegangen.

4.2.5 Konfiguration

Universelle Geräte zeichnen sich meist dadurch aus, daß sie sich an ihr Umfeld und den damit verbundenen Anwendungsfall anpassen können. Im Fall des Lepomux geschieht diese Anpassung über eine umfassende Konfigurierbarkeit seiner Subsysteme.

In den Konfigurationsmodus gelangt man über die Adresse *ADDR_CONFIG*. Die Daten, die dabei übermittelt werden, beinhalten die Subadresse der zu konfigurierenden Komponente. Das darauf folgende Datenwort wird komplett als zugehöriger Konfigurationswert betrachtet. Für die Übertragung von einem Konfigurations-Byte ist damit im Vergleich zu den Vorgängerversionen nur noch ein zusätzlicher RCX-Transfer nötig. Der ursprüngliche Ablauf sah hierfür den Transfer von drei Datenworten vor, bei denen jeweils das erste Nibble die Konfigurationsadresse enthielt.[89] Nach dem Empfang des zweiten Konfigurations-Bytes befindet sich der Lepomux wieder im normalen Run-Modus.

4.2.5.1 Register

Ähnlich wie die Adressierung der Komponenten findet auch die Auswahl der Konfigurationsregister über die oberen vier Bit des Datenwortes statt. Die Abbildung 4.18 vermittelt einen Überblick der verfügbaren Register. Hier sind auch die Formeln zur Berechnung der Konfigurationswerte zusammengefasst. Da diese Werte in den meisten Fällen vom Timimg mehrerer Systemkomponenten abhängen, verdeutlicht das Diagramm in Abbildung 4.19, welche Funktionsblöcke in welchem Zeitverhältnis aufgerufen werden. Deutlich erkennbar ist hier eine baumartige Taktverteilung an die Komponenten des Systems.

89 Der ursprüngliche Konfigurationsablauf ist der Abbildung 2.19 zu entnehmen.

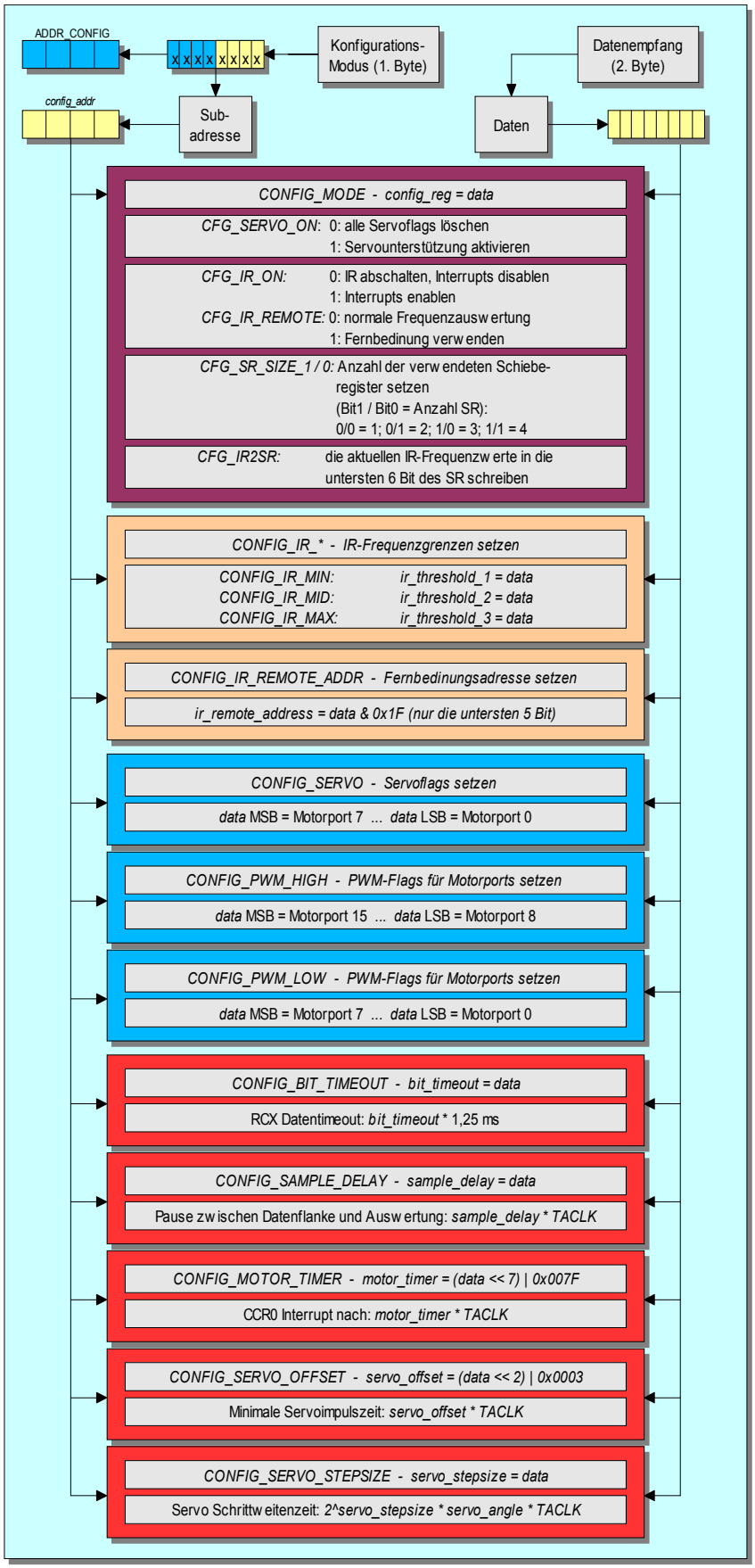


Abbildung 4.18: Konfigurationsregister des Lepomux v3.0

4.2.5.1.a Config-Register

Das Register *config_reg* faßt verschiedene Konfigurations-Flags in einem Byte zusammen. Diese betreffen hauptsächlich das komplette Ein- oder Ausschalten bestimmter Komponenten wie z.B. der IR-Auswertung oder der Servoansteuerung.

Die beiden *CFG_SR_SIZE*-Bits legen fest, wieviele Bytes zu eventuell kaskadierten Schieberegistern übertragen werden sollen. Der binäre Wert dieser Bits entspricht dabei der Anzahl der Schieberegister weniger eins.

Über das Bit *CFG_SERVO_ON* läßt sich die Erzeugung von Servo-Impulsen komplett Abschalten. Ein Löschen dieses Bits führt zu einem gleichzeitigen Zurücksetzen sämtlicher Servo-Modus-Bits innerhalb der Motor-Register.

Zur Aktivierung des Infrarotempfangs muß das Bit *CFG_IR_ON* gesetzt sein. Ist dies der Fall, so dient das *CFG_IR_REMOTE*-Bit zum Umschalten zwischen Beacon-Detektor und einer Auswertung der Fernbedienungscodes. Bei Abgeschaltetem IR-Empfang findet auch keine Interrupt-Erzeugung auf den IR-Sensor-Anschlüssen statt.

Das Bit *CFG_IR2SR* kann beim Einmessen des Beacon-Detektors hilfreich sein. Es sorgt dafür, daß unabhängig von allen anderen Motoreinstellungen, die Frequenzwerte der drei IR-Sensoren auf den unteren sechs Bit des ersten Schieberegisters ausgegeben werden. Üblicherweise sind Leuchtdioden an diese Ausgänge angeschlossen, über die das empfangene IR-Signal direkt kontrolliert werden kann.

4.2.5.1.b IR-Auswertung

Das Einstellen alternativer Frequenzgrenzen zur Beacon-Erkennung geschieht über die drei Register *CONFIG_IR_MIN*, *...MID* und *...MAX*. Die dazu nötige Berechnung der Konfigurationswerte ist dem Kapitel 2.2.1.9.b zu entnehmen.

Werden die IR-Empfänger im Fernbedienungsmodus benutzt, kann es sinnvoll sein, die RC5-Geräteadresse anzupassen. Dazu wird die fünf Bit lange Geräteadresse der verwendeten Fernbedienung in das Register *CONFIG_IR_REMOTE_ADDR* eingetragen.

4.2.5.1.c Datenempfang

Mit den Registern *CONFIG_BIT_TIMEOUT* und *CONFIG_SAMPLE_DELAY* läßt sich das Zeitverhalten des RCX-Dateneingangs beeinflussen. Im Normalfall sollte es unnötig sein, den Wert des Sample-Delay-Registers anzupassen, da hierüber nur die Verzögerung vom Auslösen des Port-Interrupts bis zur Übernahme der Daten verändert wird.

Wesentlich nützlicher ist hingegen das Bit-Timeout-Register. Es legt fest, welche Zeit maximal zwischen zwei Flanken eines Datenwortes liegen darf, bevor die bisher empfangenen Daten verworfen und die Zählerstände zurückgesetzt werden.

Je nach Struktur des verwendeten RCX-Betriebssystems kann es nötig sein, diese Zeit anzupassen. Ein zu niedriger Wert führt zu einem vorzeitigen Reset des Datenempfangs. Große Werte haben dagegen eine entsprechende Zwangspause zwischen zwei Datenworten zur Folge, da auch die Polungserkennung von diesem Timeout abhängt.

4.2.5.1.d Motoransteuerung

Zwei Register dienen der Aktivierung der Pulsweitenmodulation pro Motorausgang. Dabei können ja nach Anzahl der angeschlossenen Schieberegister maximal 16 Motorports unabhängig voneinander geregelt werden.

Ein gesetztes Bit im Register *CONFIG_PWM_LOW* aktiviert dabei die PWM-Ausgabe auf den ersten beiden Schieberegistern. Dabei entspricht das LSB dem ersten Motorport des ersten Schieberegisters. Analog dazu repräsentieren die Bits in *CONFIG_PWM_HIGH* die Motorports 15 bis 7 (von MSB nach LSB).

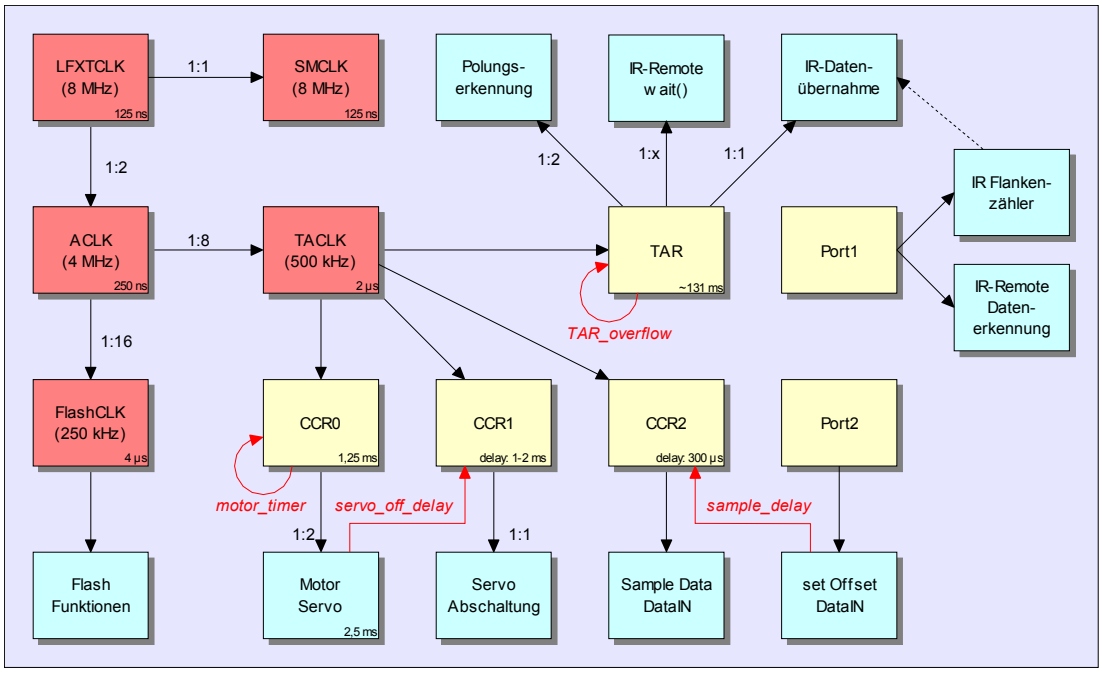


Abbildung 4.19: Timingdiagramm

Ist für einen Motorausgang das entsprechende Bit nicht gesetzt, so wird er im Port-Modus betrieben. Allerdings ist die Bedeutung der PWM-Bits irrelevant, sofern für einen Motorport der Servo-Modus aktiviert wurde (siehe unten).

Für den Fall, daß das Timing des kompletten PWM-Zyklus verändert werden soll, kann über das Register *CONFIG_MOTOR_TIMER* das Einsprungintervall in die Interrupt-Routine des Timers CCR0 festgelegt werden. Allerdings hat eine solche Modifikation weitreichende Konsequenzen für den zeitlichen Ablauf innerhalb anderer Module der Software. Das betrifft beispielsweise die zeitkritische Ausgabe der Servo-Impulse. Das Diagramm in Abbildung 4.19 verdeutlicht diese Zusammenhänge.

4.2.5.1.e Servoansteuerung

Nach dem selben Prinzip wie die Aktivierung der Pulsweitenmodulation funktioniert auch das Einschalten des Servo-Modus. Über das Register *CONFIG_SERVO* läßt sich für maximal acht Motorausgänge eine Ausgabe von Servo-Impulsen einschalten. Die Numerierung der Bits entspricht dabei den Motorports 7 bis 0. Damit ist die Ansteuerung von je vier Servos über die ersten beiden Schieberegister möglich.

Ein gesetztes Bit im CONFIG_SERVO-Register hat gegenüber dem Port- und dem

PWM-Modus Präzedenz. Da pro Motorport nur die niederwertige Leitung zur Ausgabe des Servosignals genutz wird, erscheinen auf den unbenutzen Leitungen die entsprechenden Werte des *port[]*-Registers.

Zusätzlich kann über zwei weitere Register die Timing-Charakteristik des Servo-Impulses angepaßt werden. Der Wert des Registers *CONFIG_SERVO_OFFSET* legt dabei fest, wie lang der Grundimpuls für die stärkste Linksdrehung des Servos sein soll. Die Einschaltzeit des eingestellten Winkels wird auf diesen Wert addiert. Über das Register *CONFIG_SERVO_STEPSIZE* ist eine Spreizung der Winkelschritte möglich. Die Formeln, nach denen sich das Timing der Servoimpulse berechnen läßt, sind der Abbildung 4.18 zu entnehmen.

4.2.5.2 Konfigurationsprofile

Die automatischen Abschaltung des Lepomux hat leider nicht nur Vorteile, denn dadurch gehen eventuell vorgenommene Konfigurationseinstellungen verloren. Da die damit bei jedem Start des RCX verbundene Lepomux-Konfguration mit der Zeit lästig wird, sieht die neue Software-Version das Speichern und Laden zweier benutzerdefinierter Einstellungsprofile vor. Das geschieht über die folgenden vier Befehle der Flash-Funktion aus Kapitel 4.2.4.5.

- *FLASH_LOAD_PROFILE_1*
- *FLASH_LOAD_PROFILE_2*
- *FLASH_SAVE_PROFILE_1*
- *FLASH_SAVE_PROFILE_2*

Beim Speichern eines Profils werden alle Einstellungen außer der in Kapitel 4.2.5.1.d und 4.2.5.1.e angesprochenen PWM- und Servo-Bits berücksichtig. Bevor ein Profil in den Flash-Speicher geschrieben wird, muß allerdings sichergestellt werden, daß der Speicherbereich keine Daten enthält.
Leider kann über den Befehl *FLASH_ERASE* nur der gesamte Speicherbereich beider Profile und der IR-Kommandoliste aus Kapitel 4.2.5.3 gelöscht werden. Das begründet sich durch die ausschließlich segmentweise löschbare Struktur des in Abbildung 4.20 skizzierten Flash-Speichers.

4.2.5.3 IR-Kommandotabelle

Die Fähigkeit, Fernbedienungscode verarbeiten zu können ist relativ nutzlos, solange keine Möglichkeit besteht, dem Lepomux mitzuteilen, wie er auf einen Tastendruck reagieren soll. Zu diesem Zweck wurde eine spezielle Tabellenstruktur entwickelt.

Der RC5-Code gestattet die Verwendung von maximal 64 verschiedenen Tasten auf einer Fernbedienung. Wertet man auch das Loslassen einer Tasten aus, ergeben sich 128 mögliche Kombinationen.

Üblicherweise werden längst nicht alle 64 Befehle von einer Fernbedienung benutzt. Da aber nicht vorausgesagt werden kann, welche Tasten wie belegt sind, empfieht sich die folgende, frei konfigurierbare Lösung:

Eine Zuordnungstabelle, die im 128 Byte großen Flash-Segment B untergebracht ist, wird so Programmiert, daß ihr Inhalt auf Definitionen von Kommando-Sequenzen verweist. Jeder Tabelleneintrag mit der Größe eines Bytes entspricht dabei einer Taste der Fernbedienung. Da sich sowohl gedrückte, als auch losgelassene Tasten zum Anstoßen von Kommando-Sequenzen nutzen lassen sollen, besteht die Tabelle

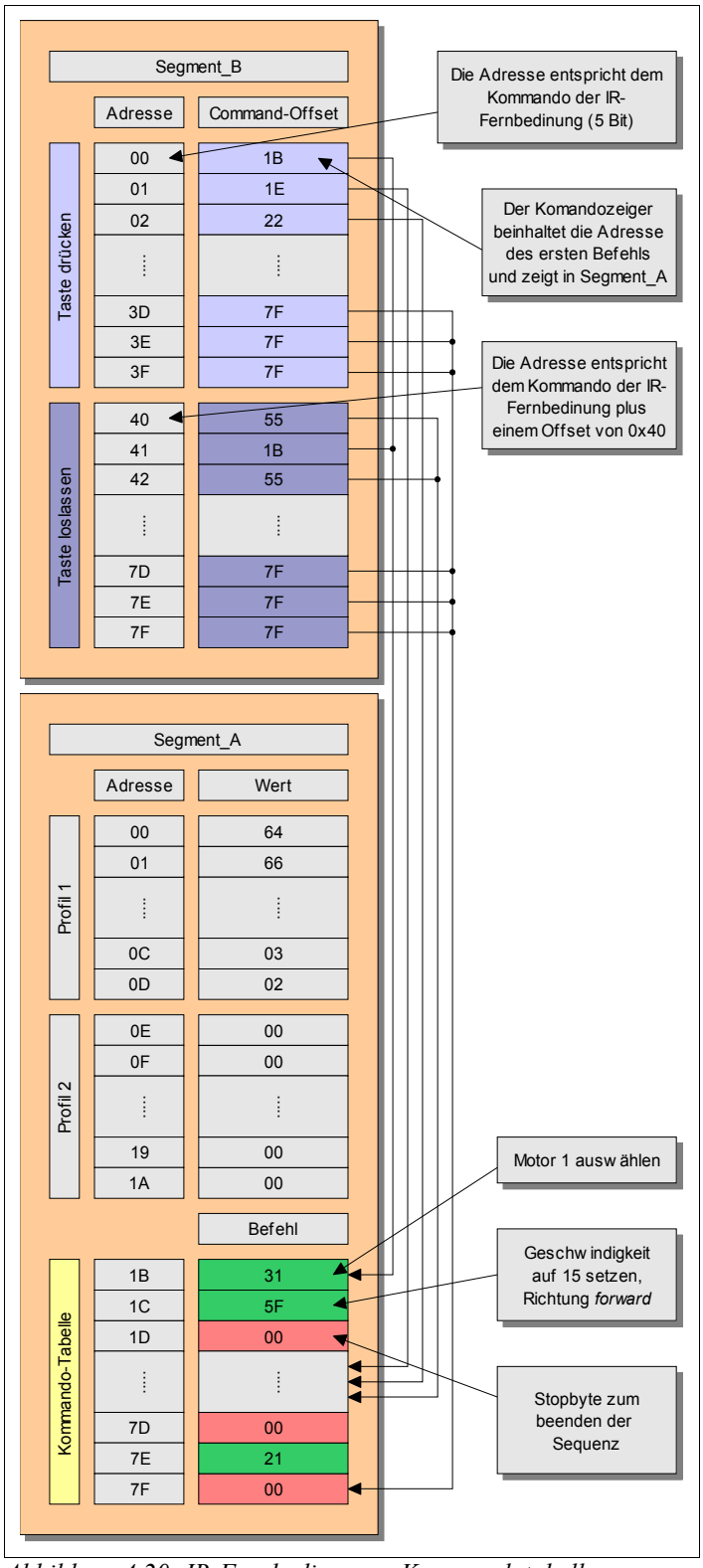


Abbildung 4.20: IR-Fernbedienungs-Kommandotabelle

aus zwei Bereichen mit einer Größe von je 64 Byte. Die dort hinterlegten Werte sind Offsets, die auf eine zweite Tabelle in Flash-Segment A verweisen. Dieser Kommandolisten-Bereich beinhaltet durch Null-Bytes terminierte Lepomux-Anweisungen.
Sie entsprechen den sonst vom RCX übermittelten Kommandos. Die Abbildung 4.20 veranschaulicht den Aufbau der beiden Tabellen.

Mit Hilfe der Befehle *FLASH_WRITE_IR_MAP* und *FLASH_WRITE_IR_CMD* lassen sich Zuweisungs- und Kommandotabelle in den Lepomux programmieren.[90] Um das zu erreichen, wird vom RCX aus die in Kapitel 4.2.4.5 beschriebenen Flash-Komponentenadresse in Kombination mit einem der beiden oben genannten Befehle gesendet. Danach folgen entsprechend 128 bzw. 64 Byte an Tabellendaten.

Vor dem Programmieren der Kommandotabelle muß das Flash-Segment A mit dem Befehl FLASH_ERASE gelöscht werden. Hierbei gehen allerdings auch die beiden Konfigurationsprofile verloren.[91]

90 Wie bereits in Kapitel 4.2.4 erwähnt, sind die Funktionsblöcke, die die Flash-Programmierung in der vorliegenden Software-Version nicht vollständig implementiert.

91 Wegen des knapp bemessenen RAM-Speichers innerhalb der MCU wurde auf ein Zwischenspeichern der Konfigurationsprofile verzichtet. Siehe auch Kapitel 4.2.5.2.

5 Evaluierung des Lepomux v3.0

Nachdem die ersten beiden Versionen bewiesen haben, daß eine praxistaugliche Umsetzung des in Kapitel 1.4 beschriebenen Konzepts durchaus möglich ist, zeigt die dritte Variante des Lepomux, wie eine Optimierung auf die Bedürfnisse des RoboLab aussehen kann. Besonders wichtig ist in diesem Zusammenhang die kompakte Bauform, bei der die zusätzliche Stromversorgung und die eigentliche RCX-Erweiterung eine Einheit bilden. Duch die modulare Konzeption ist selbst eine spätere Anpassung auf erweiterte Bedürfnisse problemlos möglich.

Im Vergleich zu den Vorgängerversionen konnte der in Abbildung 5.1 gezeigte Lepomux besonders hinsichtlich der Produktionskosten verbessert werden. Das liegt einerseits am neuartigen Konzept der LEGO-kompatiblen Steckverbinder und andererseits an der Möglichkeit einer weitgehend maschinellen Fertigung.[92]

Trotz der massiven mechanischen Umstrukturierung und dem daraus resultierenden Redesign der Leiterplatten haben sich nur zwei kleine Fehler eingeschlichen. Das betrifft die Dioden des für die IR-Detektoren vorgesehenen Pegelwandlers und einen fehlenden Widerstand in der Verstärkerschaltung des Distanzsensor-Multiplexers.

Bei den Dioden des Pegelwandlers ist dabei versehendlich der unbenutzte Pin der dreipoligen SOT-23-Gehäuse verwendet worden. Eine kleine Drahtbrücke zwischen den Pins 2 und 3 von D8 bis D10 behebt dieses Problem.

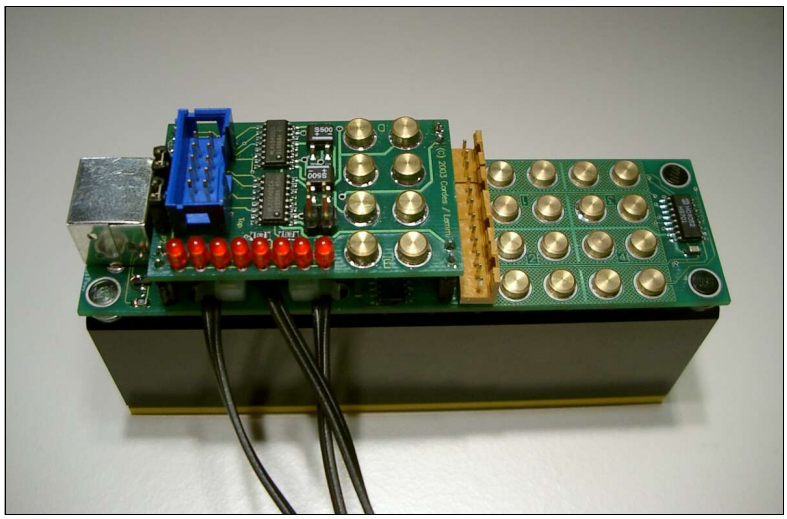


Abbildung 5.1: Lepomux v3.0 mit Sensor- und Motorboard

Bei der Entwicklung der Schaltung des Distanzsensor-Multiplexers wurde nicht bedacht, daß kein definierter Pegel am Vorverstärker anliegt, sofern keiner der Sharp-Eingänge aktiv ist. U2 ist in diesem Fall hochohmig, was dazu führt, daß der Ausgang des Operationsverstärkers U3A in Richtung des Massepotentials gezogen wird. Somit werden die zu diesem Zeitpunkt über den LEGO-Multiplexer U1 anliegenden Sensorwerte stark verfälscht.

Eine Lösung hierfür müsste immer dann einen High-Pegel auf dem Eingang des Operationsverstärkers erzeugen, wenn U2 inaktiv ist. Das läßt sich sehr einfach bewerkstelligen, indem das Ausgangssignal des Inverters U7 über einen Widerstand parallel auf den Verstärkereingang gelegt wird. Der im aktiven Zustand anliegende Low-Pegel beeinflusst den Meßwert nicht, solange der Widerstand groß genug ist.[93]

Die schönste RCX-Erweiterung nützt einem nicht viel, wenn sie zwar in theoretisch

92 Siehe dazu Kapitel 3.3.3 und 4.1.2.4.

93 Die angesprochenen Schaltungsfehler sind in den Schaltplänen bereits korrigiert worden. Der zusätzliche Widerstand hat die Bezeichnung R17 (Siehe Abbildung 4.6 und Anhang).

so funktionert wie geplant, sich aber im praktischen Einsatz unerwartete Hürden auftun. Aus diesem Grund soll anhand einer lebensnahen Aufgabenstellung gezeigt werden, ob der Lepomux den Anforderungen gewachsen ist.

5.1 Aufgabenstellung

Die zu bewältigende Aufgabe entspricht weitgehend dem Szenario aus Kapitel 1.2.3. Zwei Roboter sollen auf dem oben beschriebenen Spielfeld versuchen, möglichst viele Tore zu schießen. Dabei darf der in 5.2 abgebildete Ball auch bis ins Tor geschoben werden.

Abbildung 5.2: Roboter-"Fuß"-Ball zum Spielen von Roboterfußball

Im Vergleich zu den offiziellen Regeln der „Robocup Junior League"[94], sind hier einige Änderungen vorgenommen worden. Der Boden des Spielfelds ist nicht in einer Grauabstufung bedruckt. Eine einfache Positionserkennung durch Auswertung der Bodenreflexion ist somit nicht möglich. Außerdem sind beide Tore mit einem IR-Beacon versehen, was eine Unterscheidung und das direkte Anpeilen anhand der Frequenzkennung ermöglicht. Da die verwendeten Beacons nicht in ihrem Abstrahlwinkel begrenzt sind, ist die vom Ball ausgesandte unmodulierte IR-Strahlung schlecht detektierbar.

Alle übrigen Spielfeldeigenschaften entsprechen der Abbildung 1.12 und der zugehörigen Beschreibung.

5.2 Aufbau des Roboters

Der hier beschriebene Roboter wird von einem RCX gesteuert. Ein Lepomux der Version 3.0 dient als I/O-Erweiterung. Somit stehen insgesamt sechs Motorausgänge, ein Beacon-Detektor, vier Distanzsensor-Eingänge und fünf LEGO-kompatible Sensor-Anschlüsse zur Verfügung.


Abbildung 5.3: Umgerüsteter Modellbau-Servo

Als mechanische Basis dienen fast ausschließlich Standard-Bauteile des LEGO-Technik-Sortiments. Vier Motoren sind als Antrieb vorgesehen, wobei jeweils zwei davon parallel an den Motorausgängen des RCX betrieben werden. Sie treiben zwei seitlich angebrachte Räder an, über die durch entsprechende Anpassung der Drehgeschwindigkeit ein Lenken des Roboters ermöglicht wird.

An den Achsen der Antriebsräder sind zusätzlich zwei LEGO-Shaft-Encoder angebracht, die zur Messung der zurückgelegten Wegstrecke dienen. Damit läßt sich

94 Der RoboCup ist mittlerweile eine feste Institution. In Verschiedenen Kategorien treten hier Roboter unterschiedlicher Größe gegeneinander zum Fußballspiel an. [35]

bei geschickter Protokollierung ermitteln, wo sich der Roboter auf dem Spielfeld befindet. Dies kann beispielsweise bei der Berechnung der maximalen Schußentfernung nützlich sein.

Vier Sharp-Distanzsensoren sind so am Roboter angebracht, daß in jeder Richtung eine Abstandsmessung erfolgen kann. Mit ihnen lassen sich Hindernisse wie der gegnerische Roboter oder die Spielfeldeingrenzung erkennen. Eine besondere Bedeutung kommt dabei dem an der Vorderseite angebrachten Distanzsensor zu. Da sich der Ball nicht anhand einer aufmodulierten Frequenz erkennen läßt und ein reiner IR-Strahlungssensor auch auf die nicht abgeschirmten IR-Beacons reagieren würde, bleibt nur, den Ball mit Hilfe eines Distanzsensors zu suchen. Charakteristisch ist dabei ein kurzzeitiger Einbruch des Sensorwerts.

Der auf einem Modellbau-Servo angebrachte Beacon-Detektor dient der Lokalisierung des gegnerischen Tors. Mit Hilfe des Servos kann dabei die Peilung geprüft werden, ohne den Standort des Roboters zu verändern. Die Winkeleinstellung des Sensors kann daraufhin zur Korrektur der Fahrtrichtung benutzt werden. Selbst eine dynamische Nachregelung von Servo und Fahrtrichtung anhand der Sensorwerte des Beacon-Detektors ist mit entsprechendem Programmieraufwand machbar.

Ein weiterer Servo mit kurzer Stellzeit dient der Betätigung eines Schußmechanismus. Im Gegensatz zu einem geleichwertigen Aufbau mit einem LEGO-Motor, entfallen hier die Schalter zur Endpunkterkennung.

Die hier verwendeten Modellbau-Servos mußten vor dem Einbau umgerüstet werden, um sich mit der restlichen LEGO-Mechanik verbinden zu lassen. Abbildung 5.3 zeigt einen entsprechend modifizierten Servo. Da der AUX-Anschluß des Motor-Boards nicht genügend Strom zum Betrieb von Servos liefern kann, wird zusätzlich eine kleine Platine mit einem zusätzlichen Spannungsregler (7805) benötigt.

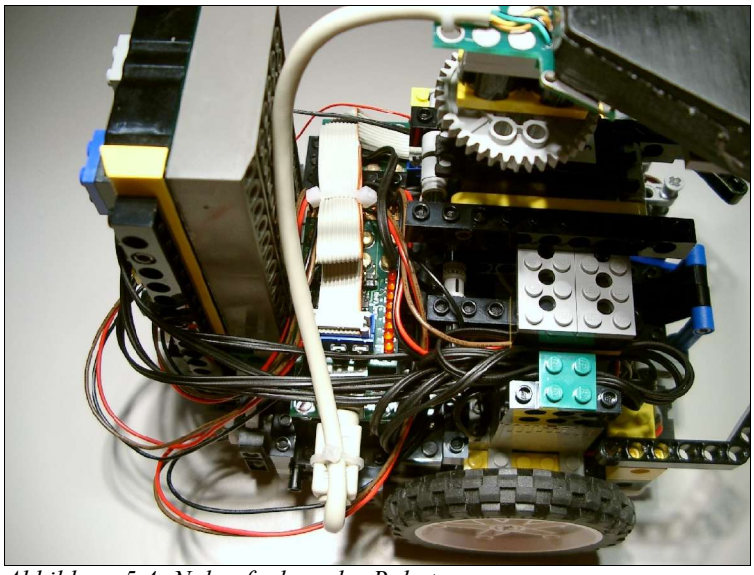


Abbildung 5.4: Nahaufnahme des Roboters

Der ursprünglich zur Auswertung der Bodenschattierung vorgesehen Lichtsensor ist in Zusammenhang mit dem hier verwendeten Spielfeld nutzlos.

5.3 Verhalten des Lepomux

Da eine Analyse der gesamten Lösung der Aufgabe den Rahmen dieser Arbeit sprengen würde, soll an dieser Stelle nur auf das in diesem Zusammenhang beobachtete Verhalten des Lepomux eingegangen werden.

Zur eindeutig wichtigsten Ausstattung gehören die Distanzsensoren. Ohne sie wäre eine Hinderniserkennung nur auf Basis von Bumpern umzusetzen und das Auffinden des Balls praktisch unmöglich. Im Vergleich zum bereits existierenden Sharp-Multiplexer von Rainer Balzerowski, hat der Lepomux den deutlichen Vorteil eines schnellen und gezielten Sensorwechsels. Für den Fall, daß beispielweise nur zwei der vier Sensoren zu einer bestimmten Zeit wichtige Daten liefern, müssen nicht wie bisher alle vier Sensoren der Reihe nach abgefragt werden. Es reicht aus, die beiden unwichtigen Sensoren in einem wesentlich langsameren Intervall abzufragen, um trotzdem eine generische Hinderniserkennung aufrecht zu erhalten.

Auch bei der Ballsuche, wo über eine quasi permanente Abfrage des Sensorwertes eine kurzzeitige Werteänderung registriert werden muß, wäre sonst keine zwischenzeitliche Auswertung der restlichen Sensoren mehr möglich. Hier würde sich übrigens empfehlen, auch den frontal angebrachten Distanzsensor auf einen Servo zu montieren, um die Ballsuche nicht mit Hilfe der Antriebsräder ansführen zu müssen.

Die Optimierung des Verstärkungsfaktors der Multiplexerschaltung auf die Sharp-Sensoren und die daraus resultierende Nutzung des gesamte RCX-Wertebereichs hat eine deutliche Qualitätsverbesserung der Meßwerte mitsich gebracht.

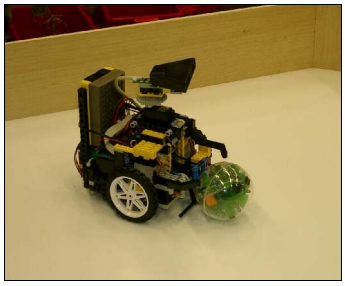


Abbildung 5.5: Tor anpeilen ...

Die Verwendung vieler unterschiedlicher Sensortypen kann allerdings ziemlich nervenaufreibend sein.[95] Gerade das zwischenzeitliche Auswerten von Shaft-Encodern ist eine Angelegenheit, bei der es darauf ankommt, ein gewisses Timing einzuhalten. Die notwendige Abtastrate ergibt sich dabei aus der minimalen Zeitspanne zwischen zwei Drehimpulsen.[96] Beim Bau eines Roboters sollte demnach darauf geachtet werden, daß je nach maximaler Drehgeschwindigkeit der Räder ein entsprechendes Getriebe für die Umsetzung auf die Shaft-Encoder benutzt wird.

Ein weiterer Knackpunkt, der sich durch das Umschalten zwischen verschiedenen Sensortypen ergibt, ist die Stromversorgung eventuell angeschlossener aktiver Sensoren. Sie müssen entweder so aufgebaut sein, daß sie eine Abschaltung der Versorgungsspannung von ca. 25 ms problemlos überbrücken können, oder es muß darauf geachtet werden, daß vor dem Auslesen des Sensorwertes eine gewisse Zeitspanne gewartet wird. Diese Wartezeit dient der Stabilisierung der Sensorelektronik und ist je nach verwendetem Sensor unterschiedlich.

Vollkommen unproblematisch gestaltet dagegen die Benutzung von Modellbau-Servos. Sie haben den Vorteil, daß eine programmierter Wert in einen Drehwinkel umgesetzt wird. Der Lepomux unterteilt den Drehbereich eines Servos in 16

95 Das liegt im Normalfall daran, daß das Betriebssystem des RCX nichts davon weiß, daß an einem der Eingänge verschiedenartige Sensoren angeschlossen sind. Daraus ergibt sich, daß eine eventuell nötige Umrechnung der Sensorwerte von der Benutzersoftware durchgeführt werden muß. Siehe dazu Kapitel 2.3.2.

96 Die Shaft-Encoder der Firma LEGO ändern pro Umdrehung 16 mal ihren Sensorwert. Dabei gibt es aber nur vier verschiedene Werte, die sich nach einem Viertel der Umdrehung wiederholen. Anhand zweier aufeinanderfolgender Sensorwerte läßt sich eindeutig bestimmen, mit welcher Drehrichtung die Achse bewegt wurde.

Positionen. Montiert man darauf einen Sensor, so läßt sich die Umgebung in einem 90 Grad großem Blickwinkel abtasten. Mit herkömmlicher LEGO-Mechanik ist eine präzise Winkelrückmeldung dabei ungleich aufwändiger. Die Abbildungen 5.5 und 5.6 zeigen den Roboter beim Anvisieren des IR-Beacons und beim Schießen auf das Tor. In beiden Fällen beruht die Mechanik auf einem Servo.

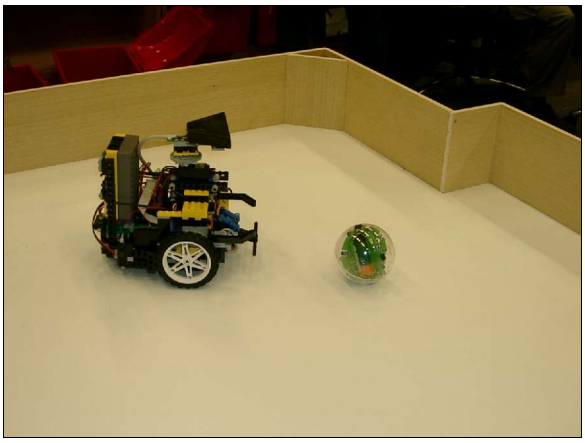


Abbildung 5.6: ... und Schießen

Als sehr praktisch hat sich auch die automatische Ein-/Ausschaltung des Lepomux erwiesen. Bei dem in Abbildung 1.16 gezeigten Distanzsensor-Multiplexer kam es dagegen aufgrund der Vergeßlichkeit der Benutzer immer wieder zu entleerten Akkus.

Abgesehen von den Widrigkeiten, die im Zusammenhang mit dem Betrieb mehrerer Sensoren an einem RCX-Eingang entstehen, verhält sich der Lepomux vollkommen unproblematisch. Bevor er allerdings für die Verwendung im RoboLab in größeren Stückzahlen produziert wird, empfiehlt sich eine Umrüstung des Motor-Boards zur direkten Unterstützung von Modellbau-Servos.[97]

97 Das betrifft nur den oben bereits angesprochenen 7805-Spannungsregler und einheitliche Steckverbinder.

6 Ausblick

In diesem Kapitel sind Gedanken bezüglich einer Weiterentwicklung des Lepomux zusammengetragen. Die derzeitige Umsetzung ist zwar für die meisten Aufgabenstellungen vollkommen ausreichend, aber trotzdem empfiehlt es sich, über einige Verbesserungen nachzudenken. Außerdem eignet sich die modulare Architektur des Lepomux eventuell noch für ganz andere Anwendungsfälle wie beispielsweise der Umsetzung eines I²C-Interface für den RCX.

6.1 Low-Power-Mux

Ein wesentliches Kriterium beim Bau des Lepomux war bisher eine universelle Einsatzfähigkeit und gute Erweiterbarkeit. Sofern ausschließlich Standardsensoren benutzt werden sollen, bietet sich eine deutlich einfacher gestaltete Variante des Lepomux an.

Es wäre durchaus denkbar, einen Sensor-Multiplexer so aufzubauen, daß keinerlei zusätzliche Stromversorgung benötigt wird. Der MSP und die Multiplexer-Bausteine beziehen dabei ihren Strom aus dem Acknowledge-Eingang des RCX. Ein Spannungsregler des Typs ***TPS 76033*** dient zur Erzeugung der für den Mikrocontroller benötigten 3,3 Volt.

Die Schaltung besteht demnach nur noch aus dem MSP, zwei ***74 HC 4051*** zum Multiplexen der Sensoren, einem weiteren ***74 HC 4051*** für den Acknowledge-Ausgang, einem Pegelwandler, der Optokoppler-Eingangsstufe und einigen passiven Bauteilen.

Denkbar wäre auch eine Stromversorgung aus dem Motorport des RCX, der mit dem Dateieingang des Lepomux verbunden ist. In diesem Fall ist entscheidend, wie das Übertragungsprotokoll aufgebaut ist, denn im Ruhezustand muß eine Spannung zur Versorgung des Lepomux am Motorport anliegen.

6.2 Byte-Transfer zum RCX

Betrachtet man den Lepomux als universellen I/O-Adapter für den RCX, so fällt auf, daß zwar die Datenübertragung von RCX zu Lepomux gut funktioniert, die umgekehrte Richtung dagegen aber eher unterentwickelt erscheint.

Grund hierfür sind die in Kapitel 1.4.4 beschriebenen Signalverfälschungen, die üblicherweise beim Bau von RCX-kompatiblen Sensoren auftreten. Allerdings beruhen die bisher angesprochenen Schaltungen auf dem in Abbildung 1.21 gezeigten Ansatz. Dabei werden die Meßwerte jeweils auf ein zuvor per Gleichrichter erzeugtes Massepotential bezogen.

Eine ganz andere Herangehensweise wäre es, die Meßwerterzeugung direkt am Sensoranschluß zu erzeugen, ohne dabei die Polung beachten zu müssen. Damit wäre der negative Einfluß der im Signal-Verpolungsschutz und dem Gleichrichter befindlichen Dioden eliminiert.

Um ein Datenwort auf den Wertebereich des A/D-Wandlers im RCX abzubilden, wird ein programmierbares Widerstandsnetzwerk benötigt. D/A-Wandler sind in diesem Fall nicht verwendbar, da sie ihr Ausgangssignal direkt auf einen Pol (Masse) ihrer Spannungsversorgung beziehen. Das würde die bisher benutzte Grundschaltung

erfordern, womit wiederum die bekannten Probleme verbunden sind.

Die in Abbildung 2.3 gezeigte Schaltung hat vom Prinzip her die gewünschten Eigenschaften, da über den Pin 3 und die acht Multiplexer-Anschlüsse von U3 ein quasi potentialfreier Widerstandswert erzeugt wird. Dabei ist allerdings zu beachten, daß die am Analog-Multiplexer anliegenden Spannungen nie mehr als 0,5 Volt über oder unter dem Versorgungsspannungsbereich liegen. Dies stellen im Fall des Lepomux die Schottky-Dioden D23 und D24 sicher.

Das Abbilden von mehr als drei Bit mit solch einem diskret aufgebauten Widerstands-Multiplexer ist aber kaum empfehlenswert, denn für jede zu übertragende Bitfolge wäre ein Widerstand nötig. Für den Wertebereich eines Bytes würde das eine Kaskadierung von insgesamt siebzehn ***CD 4067*** Multiplexern und ihre Beschaltung mit 255 Widerständen voraussetzen. [15]

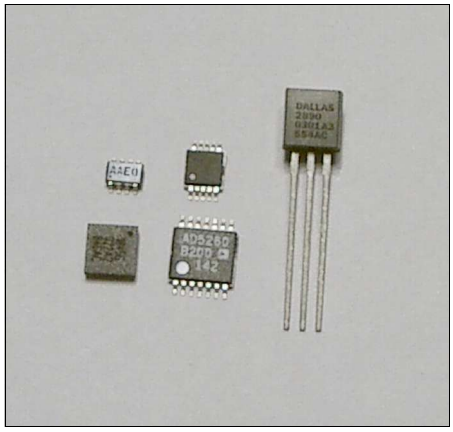


Abbildung 6.1: Digitalpotentiometer in verschiedenen Bauformen

Glücklicherweise gibt es eine große Auswahl an Digitalpotentiometern, die einen passenden Werteumfang bieten und zudem auch Schleiferspannungen von mehr als 5 Volt verkraften. Abbildung 6.1 gibt einen Überblick, welche Bauformen dieser Potentiometer derzeit verfügbar sind. Die zu erzeugenden Widerstandswerte werden üblicherweise über eine Zwei- oder Dreidraht-Schnittstelle programmiert, wobei ein festgelegter Gesamtwiderstand linear unterteilt ist. Der Baustein der Firma Dallas im Bild rechts verfügt über eine recht exotische 1-Draht-Schnittstelle.

Die für den Betrieb am RCX notwendige Voraussetzung einer Spannungsfestigkeit von mindestens 9 Volt an den Schleifer-Pins, erfüllen aber nur wenige Digitalpotentiometer. Wie für diesen Anwendungsfall geschaffen, ist der Baustein ***AD 5260*** von Analog Devices. Er bietet eine Auflösung von 8 Bit und wird über ein 3-Draht-Interface programmiert. [36]

Bei der Umsetzung sind zwei weitere Dinge zu beachten:[98]

Das Abbilden von acht Bit auf den RCX-Wandler ist nur dann sinnvoll, wenn sie den oberen acht Bit des 10-Bit-Wertebereichs entsprechen. Dazu muß der vom Digitalpotentiometer erzeugte Widerstand über einen Offset im Wertebereich des Wandlers verschiebbar sein. Dies läßt sich mit einem in Reihe geschalteten Trimmpotentiometer erreichen.

Der andere Knackpunkt ist der mögliche niederohmige Zustand des Digitalpotentiometers. Je nach Betriebsmodus des RCX-Eingangs kann es somit zu einem Kurzschluß-Stromfluß von unkritischen 500 µA oder ca. 10 Milliampere kommen. Letzterer Fall kann für den ***AD 5260*** kritisch werden, da er nur für einen maximalen kontinuierlichen Stromfluß von 5 mA konzipiert ist. Hier muß gegebenenfalls eine Schutzschaltung vorgesehen werden.

98 Die Grundlagen zu den beiden unten beschriebenen Punkten werden in Kapitel 1.4.4 angesprochen.

6.3 I²C

Der Inter Integrated Circuit Bus kurz I²C wurde Anfang der 80er Jahre von der Firma Philips entwickelt. Er diente ursprünglich dazu, die einzelnen Komponenten eines Fernsehers mit geringem Aufwand über einen Mikrocontroller steuern zu können.

Inzwischen ist dieser Zweidraht-Bus in fast jedem Gerät der Home-Entertainment-Welt zu finden. Moderne PCs benutzen eine abgewandelte Form des I²C-Bus, den SM-Bus, um ihre Systemkomponenten zu überwachen.[99]

Bedingt durch die weite Verbreitung dieser Infrastruktur, ist die Auswahl an I²C-kompatiblen Komponenten sehr groß. Neben Sensoren und A/D-Wandlern in allen erdenklichen Formen, gibt es auch EEPROM-Speicher und Dipsplay-Controller. Die Erweiterungsmöglichkeiten, die sich für den RCX durch einen I²C-Bus ergeben würden, wären demnach enorm groß.[100]

Leider scheitert eine direkte Umsetzung eines I²C-Bus über die Schnittstellen des RCX an den in Kapitel 1.4.3 beschriebenen Gegebenheiten. Zwar gab es bereits einen Implementierungsversuch, bei dem für die Kommunikation zwei Sensoreingänge verwendet wurden, aber bedingt durch die interne Struktur dieser Anschlüsse, konnte nur eine Übertragungsrate einzelner Bytes pro Sekunde erreicht werden.[101]

Um Daten effizient in den RCX transportieren zu können, sind zwei Voraussetzungen zu erfüllen: Es muß möglich sein, ein komplettes Datenwort auf den A/D-Wandler-Bereich des RCX-Sensoreingangs abzubilden[102] und das Timing für diese Übertragung muß mit dem Betriebssystem des RCX synchronisiert sein. Ohne einen derartig gesteuerten Datenabruf müsste das Betriebssystem den entsprechenden Sensoreingang permanent „pollen“, da sich prinzipbedingt keine Interrupts auf einem A/D-Wandler-Eingang generieren lassen.

Nutzt man den Lepomux als Basis für ein I²C-Interface, ist eine Synchronisation des Datentransfers mit Richtung in den RCX leicht zu realisieren und erzeugt dabei nur einen geringen Overhead.

Zur hardwareseitigen Realisierung sind nur wenige diskrete Bauelemente erforderlich. Verzichtet man auf die Benutzung des zur Motorsteuerung verwendeten Schieberegisters, lassen sich die Portleitungen P1.5 bis P1.7 des MSP als I²C-Interface zweckentfremden. Die eigentliche Arbeit liegt aber in der Implementierung des I²C-Protokolls in der Firmware des Lepomux. Derzeit wird bereits ein Großteil des vorhanden Programmspeichers von 4kB verwendet. Für den I²C-Protokollstack müsste demnach eine andere Funktionalität geopfert werden, sofern man nicht auf eine größere MSP-Variante mit mehr Flash-Speicher zurückgreifen möchte.

99 Überlicherweise mit dem Kürzel SMB (nicht zu verwechseln mit dem Begriff SMB = Server Message Block aus der Netzwerkwelt) bezeichnet, dient dieser Bus innerhalb eines PC hauptsächlich dazu, Temperatur- und Spannungs-Informationen abzufragen.

100 Die Umsetzung einer parallelen Bus-Architektur auf den I²C-Bus läßt sich leicht mit I²C-Controllern des Typ PCA 9564 bewerkstelligen. Dies ist der Nachfolger des bekannten PCF 8584. Die Bauteil-Dokumentation in [37] vermittelt einen guten Überblick bezüglich des Protokollaufbaus.

101 Der von Wim Huiskamp verfolgte Ansatz ist zwar von der Hardware her mit vertretbarem Aufwand realisierbar, aber von der Übertragungsleistung für die Verwendung innerhalb mobiler Roboter unbrauchbar. [38]

102 Lösungsansätze für die diese Voraussetzung wurden im Kapitel 6.2 beschrieben.

6.4 Bluetooth

Die bisher betrachteten Aufgabenstellungen befassen sich alle mit der Interaktion eines Roboters mit den auf dem Parcours befindlichen Objekten. Dabei ist keine Kommunikation mit der Außenwelt vorgesehen.

Es gibt aber auch andere Szenarien, in denen beispielsweise ein zentraler Server dazu benutzt wird, Informationen über die Position der auf dem Spielfeld befindlichen Roboter zu versenden. Das Spielfeld wird dabei über eine Kamera beobachtet. An den Robotern angebrachte Farbmarkierungen dienen der Positionsberechnung. Die so ermittelten Roboterstandorte werden permanent über einen IR-Sender abgestrahlt und können mit dem integrierten Infrarot-Transceiver der RCX-Bausteine empfangen werden.[103]

Genauso wäre eine Kommunikation zwischen verschiedenen Robotern denkbar, so daß zwei Teams durch Informationsaustausch eine gemeinsame Taktik verfolgen können. In der Praxis ist aber schon der reine Datenempfang von einem zentralen Sender problematisch. Der Grund dafür sind Abstrahlwinkel und Reichweite der IR-Transceiver. Eine positionsunabhängige Datenübertragung zwischen zwei Robotern ist somit praktisch unmöglich.

Die unter dem Namen „Bluetooth“ bekannte Funk-Infrastruktur hat nicht mit diesen Problemen zu kämpfen. Inzwischen gibt es davon Varianten, die die ursprüngliche Reichweite von zehn Metern auf bis zu einhundert Meter erweitern. Mit einer Bluetooth-Erweiterung wäre neben dem Informationsaustausch mit einem zentralen Server auch eine Kommunikation der Roboter untereinander ohne Sichtkontakt möglich. Außerdem hat der Bluetooth-Protokollstack den großen Vorteil, daß die „Identität“ der Kommunikationspartner geprüft wird. Somit sind eventuelle manipulierte Datenpakete unfairer Roboter nutzlos.

Abbildung 6.2: Bluetooth-Modul

Im RoboLab der HAW-Hamburg wird zur Zeit mit einer derartigen Serverlösung experimentiert. Dabei dient allerdings ein PDA als Bluetooth-Client, der das 6.270-Board nur noch als Motor- und Sensor-Adapter benutzt. Wünschenswert wäre an dieser Stelle eine kostengünstige Lösung, bei der ein RCX bzw. das 6.270-Board wie bisher als Basis dienen, aber kein PDA als Bluetooth-Empfänger benötig wird.

Die Abbildung 6.2[104] zeigt ein Bluetooth-Modul, das eine Fläche von weniger als einem Quadratzentimeter einnimmt. Es basiert auf der Single-Chip-Lösung ***BRF 6100*** der Firma Texas Instruments und zeichnet sich dadurch aus, daß abgesehen von einer Antenne praktisch alle benötigten Hard- und Software-Komponenten bereits integriert sind. Mit einer Stromaufnahme von weniger als 50 mA bei 1,8 Volt ist es ideal für die Verwendung in batteriebetriebenen Geräten.

Ein ähnliches Modul der Firma Taiyo Yuden ist das EYMF2CAMM. Auf einer Fläche von 34 x 16 mm sind die Antenne und der komplette Bluetooth-Chipsatz un-

103 Das von Rainer Balzerowski entwickelte System zur Ortung farblich markierter Objekte innerhalb eines von einer Kamera überwachten Spielfelds dient dabei als Basis für einen Roboter-Positions-Server. [39]

104 Quelle: Texas Instruments

tergebracht. Das Modul wird mit 3,3 Volt betrieben und über eine serielle Schnittstelle angesteuert. [40]

Ein derartiges Modul mit dem Mikrocontroller des Lepomux anzusteuern wäre relativ aufwändig, denn der verwendete ***MSP430 F1121A*** besitzt keine Hardware für eine serielle Schnittstelle wie z.B. RS-232. Andere Varianten wie beispielweise der ***MSP430 F1232*** sind dagegen mit einer derartigen Schnittstelle ausgestattet und besitzen außerdem noch einen 10-Bit A/D-Wandler. In Abbildung 6.3 ist dieser Prozessor neben seinem kleinen Bruder zu sehen.[105]

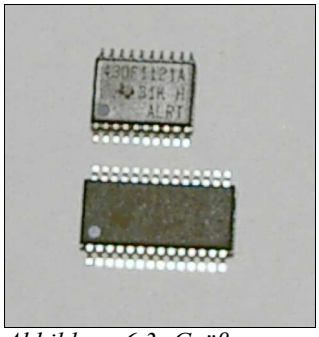


Abbildung 6.3: Größenvergleich von MSP430 F1232 zu F1121A

Die zusätzlich vorhandenen 4 kB Flash-Speicher sollten genug Raum bieten, um selbst relativ komplizierte Handshake-Mechanismen für den Bluetooth-Betrieb zu implementieren.

Insgesamt wäre die Kombination des Lepomux mit einem passenden Bluetooth-Modul deutlich günstiger als eine PDA-basierte Lösung. Allerdings muß genauso wie bei der oben angesprochenen I²C-Erweiterung sichergestellt sein, daß auch in Richtung des RCX eine effiziente Datenübertragung möglich ist.[106]

6.5 Optimierung des Übertragungsprotokolls

Mit der Veränderung des zwischen RCX und Lepomux benutzten Protokolls konnte bereits eine deutliche Steigerung der Übertragungsleistung erreicht werden. Mit einer weiteren Verbesserung läßt sich vor allem die zeitliche Komponente der maximalen Bit-Dauer eliminieren, denn momentan muß der Lepomux zum Erreichen einer optimalen Übertragungsrate auf das Timing des verwendeten Betriebssystems abgestimmt werden.

Das bisher benutzte Protokoll verwendet die drei Motorzustände *off*, *forward* und *reverse,* die Ausgabe von *brake* ist dabei gleichwertig mit der von *off*. Die zwei Bit, die über den Motor-Status codiert werden, sind also nur zu ¾ ausgeschöpft.

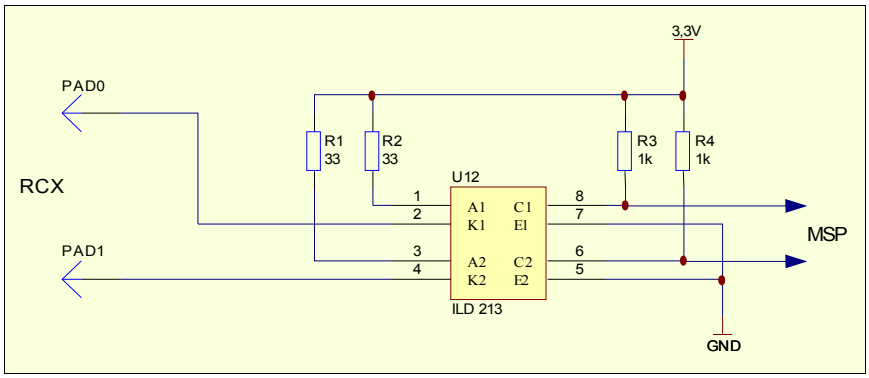


Abbildung 6.4: geänderte Beschaltung des Dateneingangs

Mit einer geringfügigen Modifikation der Dateneingangsbeschaltung des Lepomux lassen sich dagegen die vollen zwei Bit auswerten. Dies hat den großen Vorteil, daß

105 Die bessere Hardware-Ausstattung, die auch einen doppelt so großen Flash-Speicher einschließt, schlägt sich natürlich im Bauteilpreis nieder. Der MSP430 F1232 ist im Vergleich zu der bisher verwendeten MSP-Variante etwa ein Drittel teurer. [41]

106 Siehe Kapitel 6.2.

damit der Anfang eines Datenwortes eindeutig identifiziert werden kann. Es wird also kein Timeout seitens des Lepomux mehr benötigt, um nach einem Synchronisationsfehler wieder in den Startzustand zu gelangen.

In Abbildung 6.4 ist der geänderte Aufbau des Dateneingangs skizziert. Bisher wurde immer dann ein Signal erzeugt, wenn durch das Schalten von *forward* oder *reverse* ein positiver Spannungspegel auf dem Motorausgang lag. Die neue Schaltung prüft dagegen, ob ein Pol des Motorports über den H-Brücken-Treiber gegen Masse geschaltet wurde. Voraussetzung für die Funktionsfähigkeit ist ein gemeinsames Massepotential, das über die Schottky-Doppeldiode D2 des Acknowledge-Ausgangs hergestellt wird. Für die Berechnung der beiden Optokoppler-Vorwiderstände muß zusätzlich ein Diodenübergang innerhalb des Motortreibers berücksichtigt werden.[107]

Leider hat diese Änderung nicht nur Vorteile. Durch die zusätzliche Markierung des Datenwortbeginns entfällt der Ruhezustand. Zwar dürfen Bits nun beliebig lang sein, aber eine verläßliche Polungserkennung ist nicht mehr wie gewohnt möglich. Ein zusätzliches Startbit kann hier für Abhilfe sorgen. Da es immer den Wert eins hat, läßt sich eindeutig feststellen, mit welcher Polung der Stecker angeschlossen ist. Abbildung 6.5 zeigt den Aufbau eines derartig veränderten Protokolls.

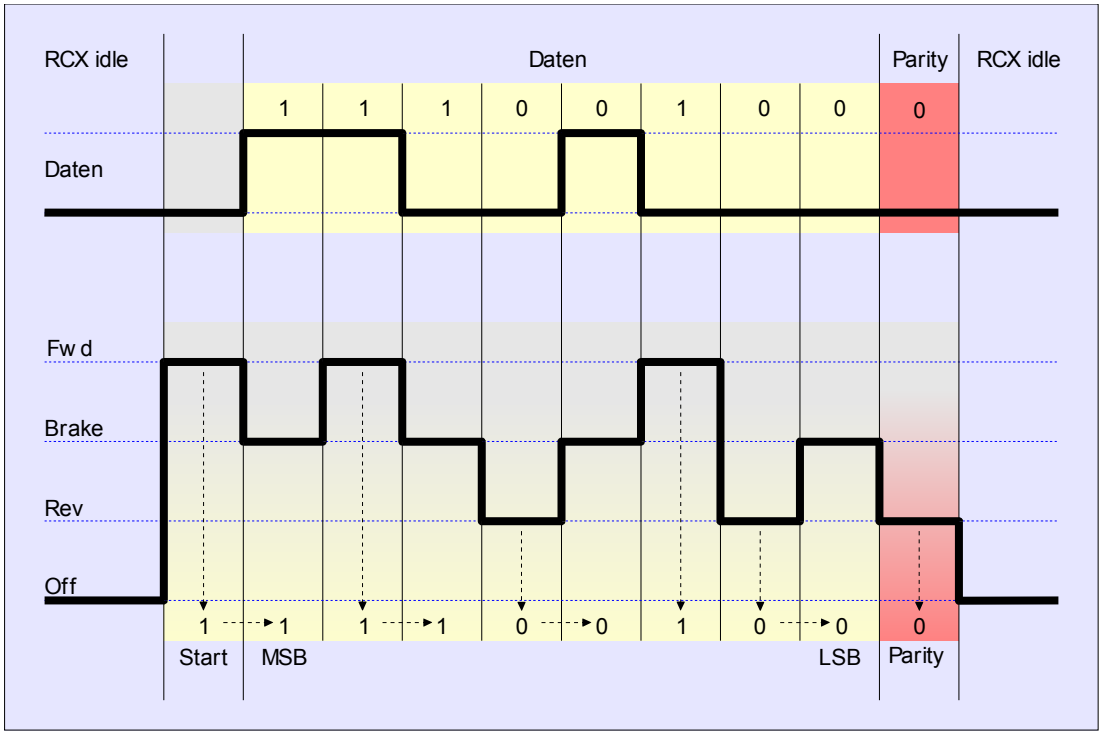


Abbildung 6.5: Erweitertes Datenübertragungsprotokoll mit zusätzlichem Startbit

Ein weniger eleganter Weg wäre eine explizite Polungsmarkierung seitens des RCX. Das könnte so aussehen, daß ein einzelnes *forward* gefolgt von einem *off* gesendet wird. Damit ist klar, daß es sich nicht um ein gültiges Datenwort handeln kann. Die Portleitung, auf der das *forward* detektiert wurde, gibt darüber Aufschluß, mit welcher Polung der RCX angeschlossen ist.

107 Die Zusammenhänge zwischen den verschidenen Spannungspegeln von RCX und Lepomux sind in den Kapiteln 1.4.3, 1.4.4 und 2.1.3 beschrieben.
Die Bauteilbezeichnungen beziehen sich auf den Schaltplan des Controller-Boards. Siehe Kapitel 4.1.1 und Anhang .

Diese Vorgehensweise hat den Vorteil, daß die Datenübertragung nicht durch ein zusätzliches Startbit verlangsamt wird. Sie funktioniert aber nur durch aktives Eingreifen des RCX. Wird der Stecker während des Betriebs gedreht, sind die Folgen nicht absehbar, denn auch mit umgekehrter Polung können Datenworte zufällig als korrekt interpretiert werden. Demnach ist die erste Lösung wegen ihrer Robustheit vorzuziehen.

Schlußwort

Mit der Entwicklung des Lepomux im Rahmen dieser Diplomarbeit konnte gezeigt werden, wie sich aufbauend auf einer nicht unwesentlichen Menge theoretischer Vorüberlegungen, eine zielstrebige Umsetzung bis hin zur Serienreife realisieren läßt.

Die in Kapitel 1.4.1 beschriebene Zielsetzung konnte dabei in praktisch allen Punkten erreicht werden. Damit ist der Lepomux eine universelle I/O-Erweiterung für den RCX, mit der die wesentlichen Bedürfnisse innerhalb des RoboLabs der HAW-Hamburg abdecken werden können.

Ein Gespann aus RCX und Lepomux als Basis für mobile Roboter ist somit von seiner Leistungsfähigkeit eine ernstzunehmende Konkurrenz zu den bestehenden 6.270-Systemen. Es gestattet schon in der Grundausstattung die gleichzeitige Nutzung von bis zu zehn Sensoren und sechs Aktoren. Hinzu kommt, daß spezielle, aber oft benötigte Sensorik, wie der Beacon-Detektor und die Sharp-Distanzsensoren ohne zusätzlichen Elektronik unterstützt werden.

Betrachtet man diese Aspekte unter der Voraussetzung, sehr kompakte Roboter bauen zu müssen, ist das RCX/Lepomux-System nahezu konkurrenzlos.[108]

Der modularen Aufbau des Lepomux gestattet zusätzlich eine zukünftige Aufrüstung. Die als Modul aufsteckbaren Erweiterungen können dabei Aufgaben wie beispielsweise die Ansteuerung diverser Motoren oder eine Nutzung bisher nicht untersützter Kommunikationsprotokolle übernehmen.

Einziger Wehrmutstopfen sind die relativ hohen Kosten für ein Gesamtsystem, die sich aber hauptsächlich aus der Verwendung der bekanntlich nicht gerade preiswerten LEGO-Komponenten ergeben.

108Das Regelwerk bestimmter RoboCup-Wettbewerbe schreibt einen maximalen Durchmesser des teilnehmenden Roboters bei ausgefahrener Mechanik mit nicht mehr als 22 cm vor. [35]

Hinweis zu Markennamen

Alle in dieser Arbeit verwendeten Firmen- und/oder Produktnamen sind Warenzeichen und/oder eingetragene Warenzeichen ihrer jeweiligen Hersteller in ihren Märkten und/oder Ländern.

Abbildungsverzeichnis

Literaturverzeichnis

1: Konzeption von I/O-Erweiterungen für Lego Mindstorms
http://www.informatik.haw-hamburg.de/~robots/lepomux/lepomux.pdf

2: Lego Mindstroms
http://mindstorms.lego.com/

3: RoboLab HAW Hamburg
http://www.informatik.haw-hamburg.de/~kvl/

4: Die Geschichte des 6.270
http://web.mit.edu/6.270/

5: Handy Board (6.270)
http://handyboard.com/techdocs/hbmanual.pdf

6: RCX internals
http://graphics.stanford.edu/~kekoa/rcx/

7: LejOS Hompage
http://lejos.sourceforge.net/

8: BrickOS Homepage
http://brickos.sourceforge.net

9: NQC - Not Quite C
http://www.baumfamily.org/nqc/

10: Michael Gasperi Webseite
http://www.plazaearth.com/usr/gasperi/lego.htm

11: Extreme Mindstorms
Dave Baum, Michael Gasperi, Ralph Hempel & Luis Villa
2000, ISBN: 1893115844

12: RCX Input Multiplexer
http://www.plazaearth.com/usr/gasperi/MUX.HTM

13: Mindsensors Homepage
http://www.mindsensors.com/

14: 8:1 Analog-Multiplexer (CD4051)
http://www.fairchildsemi.com/ds/CD/CD4051BC.pdf

15: 16:1 Analog-Multiplexer (CD4067)
http://www.vincenzov.net/datasheet/2058.pdf

16: MSP-430 Processor Family Description
http://focus.ti.com/docs/prod/productfolder.jhtml?genericPartNumber=MSP430F1121A&pfsection=desc

17: Hex-Levelshilfer (MC14504)
http://www.onsemi.com/pub/Collateral/MC14504B-D.PDF

18: IR-Sensor (TSOP18xx)
http://www.vishay.com/docs/82047/82047.pdf

19: IR-Sensor (SFH 506)
http://bray.velenje.cx/avr/PDFs/sfh506.pdf

20: Sharp IR-Sensor mit TTL-Ausgang (GP1UD28YK)
http://www.sharp.co.jp/products/device/ctlg/jsite23/table/pdf/ird/ir_ds_ir_durc/gp1ud26xk_j.pdf

21: Motor Treiber (MLX10402)
http://www.melexis.com/prodfiles/mlx10402.pdf

22: Ladungspumpe (LTC1044)
http://www.linear.com/pdf/lt1044.pdf

23: IAR Workbench
http://www.iar.com/Products/EW/

24: GCC Compiler für MSP-430 Prozessoren
http://www.mikrocontroller.net/mspgcc.htm

25: 8-Bit Schieberegister (74HCT595)
http://www.semiconductors.philips.com/pip/74HCT595.html

26: Sharp Distanzsensor (GP2D12)
http://sharp-world.com/products/device/ctlg/esite22_10/table/pdf/osd/optical_sd/gp2d120_e.pdf

27: Offizielle Lepomux Infoseite
http://www.lepomux.org

28: LUGNET - Lego User Group Net
http://www.lugnet.com

29: Spannungsregler 5 Volt / 100mA LDO (TPS76150)
http://www-s.ti.com/sc/ds/tps76150.pdf

30: Spannungsregler 3,3 Volt / 50mA LDO (TPS76033)
http://www-s.ti.com/sc/ds/tps76033.pdf

31: MSP430F1121A, 16-bit Ultra-Low-Power Microcontroller
http://www-s.ti.com/sc/ds/msp430f1121a.pdf

32: 4:1 Analog-Multiplexer (74HCT4052)
http://www.semiconductors.philips.com/pip/74HCT4052DB.html

33: Spannungsregler 5 Volt / 250 mA LDO (TPS7250)
http://www-s.ti.com/sc/ds/tps7250.pdf

34: IR-Fernbedienungsprotokolle
http://mikrocontroller.cco-ev.de/de/IR-Protokolle.htm

35: Offizielle RoboCup Webseite
http://www.robocup.org/

36: Digitalpotentiometer
http://www.analog.com/UploadedFiles/Data_Sheets/618372110AD5260_2_0.pdf

37: I²C-Bus Controller mit parallelem Interface
http://www.semiconductors.philips.com/pip/PCA9564.html

38: I²C Bus für den RCX
Wim Huiskamp, David Daamen
Elektor 4/2002, Seite 28

39: Realisierung eines Webcam basierten Kamera-Systems für mobile Roboter
http://www.informatik.haw-hamburg.de/~lego/Projekte/Balzerowski/Balzerowski_deutsch.html

40: BT-Modul der Firma Taiyo Yuden.
http://www.yuden.co.jp/ecatalog/PDF/JLEYSC1.PDF

41: MSP430F1232, 16-bit Ultra-Low-Power Microcontroller
http://www-s.ti.com/sc/ds/msp430f1232.pdf

Alle in diesem Verzeichnis angegebenen Links auf Webseiten sind am 16.10.2003 auf ihre Funktionfähigkeit überprüft worden.