Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

DIPLOMARBEIT

Bewegungssteuerung mobiler Kleinroboter

ausgeführt am

Institut für Computertechnik der
Technischen Universität Wien

unter der Leitung von

O.Univ.Prof. Dipl.-Ing. Dr.techn. Dietmar Dietrich
und Projektass. Dipl.-Ing. Dr.techn. Gregor Novak

als verantwortlich mitwirkenden Universitätsassistenten

durch

Christian Perschl
Hauptstraße 34, 2123 Hautzendorf
Matrikelnummer 9327470

Wien, Februar 2008

Kurzfassung

Ein Traum der Menschheit war es schon immer, intelligente künstliche Gefährten zu schaffen, vor allem zum Zwecke der Durchführung von Arbeiten oder Arbeitsschritten, die der Mensch nicht durchführen kann oder will: Roboter („robota“ = arbeiten). Die meisten aktuellen Entwicklungen von Robotern können einer von 2 Klassen zugeordnet werden: stationäre Industrieroboter und mobile Roboter. Zweiteren widmet sich diese Arbeit.

Im Rahmen dieser Arbeit werden der Entwurf und die Realisierung einer mobilen Roboterplattform beschrieben, wobei der Schwerpunkt auf die Bewegungseinheit und der Regelungssoftware gelegt wird. Die wichtigsten Anforderungen an das Design sind Modularität, Geschwindigkeit und Präzision der Bewegung.

Grundlage für das Design ist die Erhebung des Stands der Technik: Von der Art der Fortbewegung, der Sensorik, Kommunikation, existierende mögliche Hardware- und Softwareplattformen. Aufgrund der vorliegenden Anforderungen an die Roboterplattform wird die Systementscheidung für einen Microcontroller basierten Ansatz getroffen.

Die im Rahmen der Arbeit vorgenommene Realisierung des mobilen Miniatur-Roboters „Tinyphoon“ wird erläutert. Die Kernaufgaben der Regelungssoftware werden beschrieben: Messen der Raddrehzahlen, Ansteuerung der Motoren, Kommunikation über Funk sind nur einige der Aufgaben der Software. Die detailierte Dokumentation der resultierenden Softwaremodule und der Einsatz der Mikrocontrollereigenen Peripherie erleichtert die Wiederverwendung und Verbesserung.

Ansätze für mögliche Erweiterungen und Verbesserungen sind ebenfalls beschrieben.

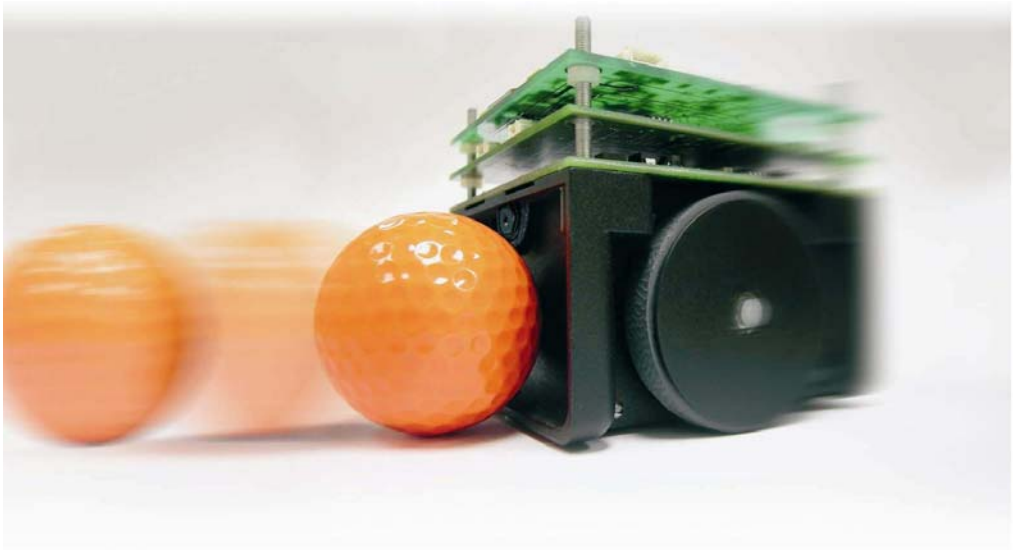

Abbildung 1.1 Die Miniatur-Roboterplattform Tinyphoon

Danksagung

Besonderen Dank möchte ich meiner Familie aussprechen, welche mein Studium ermöglichte und mich jederzeit voll unterstützten.

Ebenfalls herzlich bedanken möchte ich mich bei meinem Freund und Mentor Projektass. Dipl.-Ing. Dr.techn. Gregor Novak, der mich speziell im Zusammenhang mit der Diplomarbeit maßgeblich unterstützt hat.

Nicht zuletzt bedanke ich mich auch bei Herrn O.Univ.Prof. Dipl.-Ing. Dr.techn. Dietmar Dietrich für die Ermöglichung der Realisierung dieser Diplomarbeit.

Inhaltsverzeichnis

1 Einleitung 1
2 Problemstellung 4
2.1 Modulare Architektur 5
2.2 Geschwindigkeit 5
2.3 Beschleunigung 5
2.4 Bewegungspräzision 5
2.5 Laufzeit 5
2.6 Roboterfußballspezifische Eigenschaften 6
2.7 Diagnosemöglichkeiten 6
2.8 Mögliche Einsatzgebiete von Tinyphoon 6
3 Stand der Technik 8
3.1 Bewegung 9
3.2 Sensorik 12
3.3 Kommunikation 15
3.4 Steuerung 18
3.5 Hardware 18
4 Systemdesign und Realisierung 40
4.1 Plattformentscheidung 42
4.2 Einsatz des xc167 in Robotersystemen 43
4.3 Design von Tinyphoon 67
4.4 Mechanik 69
4.5 Elektronik 70
4.6 Software 77
4.7 Realisierung der Teilaufgaben 84
4.8 Modulbeschreibung 87
4.9 Konfiguration des xc167 für Tinyphoon 110
5 Verbesserungspotentiale 126
6 Conclusio 128
Literaturverzeichnis 129
Abbildungsverzeichnis 130
Tabellenverzeichnis 132

1 Einleitung

Schon lange ist es eine Vision der Menschheit, künstliche Ebenbilder zu schaffen. Die Vorstellung vom künstlichen Mensch reicht vom einfachen Roboter bis zum perfekt nachgebildeten Menschen, dem Avatar oder Android, welcher von seinem Vorbild und Schöpfer nicht zu unterscheiden ist, also lernfähig, intelligent, kreativ ist und unter Umständen sogar „fühlt".

Ein menschliches Ebenbild – Wozu?

Mehrere Motive für den Traum vom künstlichen menschlichen Ebenbild sind dabei besonders ausgeprägt:
Die wohl bedeutendste ist jene des Arbeiters (der Begriff Roboter kommt ja von „robota" – tschechisch Arbeiter): Die maschinellen Gefährten sollen dem Menschen Arbeiten abnehmen oder ihm bei deren Ausführung unterstützen. Dies betrifft erst einmal monotone und wiederkehrende, gleichartige Tätigkeiten, Arbeitsschritte, die für den Menschen bzw. für Organismen riskant oder gesundheitsgefährdend sind und natürlich Arbeiten, die in Umgebungen durchgeführt werden sollen, in denen der Mensch grundsätzlich nicht arbeiten kann. Diese Visionen können so weit gehen, dass von einer paradiesähnlichen Lebensumgebung für den Menschen gesprochen werden kann: sämtliche zu verrichtende Arbeiten – neben „körperlichen" auch organisatorische Aufgaben und jede Art von Dienstleistungen – werden von Robotern durchgeführt. So gut sich dieser Traum anhört, die soziale Komponente der Arbeit für den Menschen wird verschwiegen. Schließlich wollen wir Menschen ja arbeiten, nützlich sein und viel Kontakt mit anderen Menschen haben.
Die Zukunftsvision, dass sämtliche schweren, gesundheitsschädigenden oder auch nur anstrengenden Arbeiten von maschinellen Gehilfen durchgeführt wird und der Mensch sich auf kreative, „intelligente" und soziale Arbeit konzentrieren kann, ist zumindest in den sogenannten Industrieländern längst Realität. Nach der industriellen Revolution und der rasanten technischen Entwicklung im 20. Jahrhundert arbeitet der überwiegende Teil der Bevölkerung als Dienstleister und nicht mehr als Schwerstarbeiter in Fabriken oder bei der Feldarbeit als Landwirt. Erste Schritte sind also bereits getan.

Die zweite Motivation ist jene des „Entertainments": mobile Roboter als Spielzeug, als Unterhalter, als Lernobjekte oder auch als Ansprechpersonen. Auch hier gibt es bereits etliche Entwicklungen, vom Baukasten-Roboter zum Selbstprogrammieren bis zum lernfähigen Roboterhund. Ein weiteres Beispiel für diese Kategorie ist auch der Wettkampf von Robotern. Einerseits natürlich zur Unterhaltung, andererseits aber auch zu Forschungszwecken spielen Roboter gegeneinander Fußball, müssen bestimmte Aufgaben erledigen oder müssen im härtesten Fall im Kampf einander so lange demolieren, bis ein Roboter bewegungsunfähig ist.

Vom künstlichen, autonomen Gefährten wird aber noch für viele weitere Anwendungsbereiche geträumt: zum Beispiel Androiden als Barkeeper, als Bote, Erforschung anderer Planeten sowie unterschiedlichste Einsatzmöglichkeiten im militärischen Bereich wie Spionageroboter, Kampfroboter, Drohnen und vieles mehr.

Von der Vision zum Konzept

Während arbeitende Maschinen als Industrieroboter längst Realistät geworden sind, bleibt der menschenähnliche Roboter oder Android a la Isaac Asimov, Star Wars und Star Trek wohl noch länger eine Vision. Roboter können heute nur mit einem hohen Spezialisierungsgrad bestimmte Arbeitsschritte oder Aufgaben erfüllen.

Stationäre Industrieroboter sind ein typisches Beispiel für hochspezialisierte und bis zu einem gewissen Grad autonome Arbeitsmaschinen, und für viele Industriezweige sind sie heute unverzichtbar. Ihr Vorteil ist der vordefinierte, eingeschränkte Handlungsbereich (well structured controlled environment), ihre Aufgaben sind zumeist fixe Sequenzen von Arbeitsschritten. Bis zu einem gewissen Grad können sie auch auf unerwartete Situationen reagieren, doch bleibt ihr Handlungsspielraum gering.

Dagegen haben es autonome mobile Roboter wesentlich schwerer. Dies beginnt bereits mit dem vorgesehenen Einsatzgebiet. Wo soll sich der Roboter bewegen? An Land, im Wasser, in der Luft, wie sieht die Beschaffenheit aus? Ist die Bewegungsoberfläche eben und glatt, gibt es Steigungen, sind Hindernisse wie z. B. Stiegen zu überwinden, oder können überhaupt keine Annahmen getroffen werden? Wie schnell soll sich der Roboter bewegen können?. Wie die Evolution mit den unterschiedlichen Ausstattungen der Lebewesen für unterschiedliche Lebensräume zeigt, ist daher auch bei mobilen Robotern ein gewisser Spezialisierungsgrad erforderlich. Und das Vorbild Mensch mit seiner großartigen Anpassungsfähigkeit ist ein recht hoch gestecktes Ziel.

Doch mit der Bewegung alleine ist es nicht getan. Der Roboter muss die Möglichkeit haben, seine Umgebung kennenzulernen, also so wie wir Menschen zu sehen, zu tasten oder zu hören. Wie soll er sich sonst darin zurechtfinden? Daher sind eine Reihe von Sinnesorganen erforderlich: Sensoren, wie zum Beispiel Kameras, Raddrehzahlgeber, Beschleunigungssensoren oder Ultraschallsender. Ein Gutteil der Sensoren haben dabei Vorbilder im menschlichen Organismus oder anderen Lebewesen.

Diese Informationen über die Umwelt müssen aber auch zusammengesetzt und ausgewertet werden. Der Roboter muss sich ein Bild, ein Modell von seinem Umfeld machen. Die Modellierung der Umgebung wird vor allem von autonomen mobilen Robotern benötigt, um festzustellen: Wohin kann ich mich bewegen, was ist wo, und vor allem: Wohin will oder soll ich mich bewegen? Die Bewegungsplanung ist eine zentrale Aufgabe eines autonomen Roboters.

Orientiert man sich am Vorbild Mensch, so muss der Roboter dann noch in der Lage sein, sich mit anderen Robotern, Maschinen oder gar dem Menschen zu unterhalten: die Kommunikation

kommt vor allem in Multi-Agenten-Systemen eine zentrale Bedeutung zu. Soll direkte Kommunikation mit dem Menschen stattfinden? muss die menschliche Sprache nachgebildet und vor allem verstanden werden? Dies zählt aufgrund einerseits der Redundanz und andererseits der Mehrdeutigkeit der menschlichen Kommunikation – abgesehen von der Unzahl an existierenden Sprachen - sicher zu den schwierigsten Aufgaben eines mobilen menschenähnlichen Roboters.

Mit all diesen Fragen – Art der Bewegung, Sensorik, Modellierung, Bewegungsplanung und Kommunikation – beschäftigt sich der erste Teil der vorliegenden Arbeit. Überblicksmäßig werden alle genannten Systemaspekte behandelt. Man kann bereits erkennen, dass Robotertechnik einen weiten Bogen über die verschiedensten Wissenschaftsgebiete spannt: Maschinenbau, Mechantronik, Regelungstechnik, Bildverarbeitung, Mustererkennung, 3D-Modellierung, Neuronale Netze und Kybernetik bis hin zu KI (künstliche Intelligenz).

2 Problemstellung

Die vorgesehenen Aufgaben eines Roboters und sein Einsatzgebiet sind vom Beginn der Entwurfsphase an zu berücksichtigen. Nur mit ausreichend hoher Spezialisierung kann die Maschine einen bestimmten Aufgabenkreis erledigen und sich im vorgesehenen Anwendungsgebiet fortbewegen. Auch in der Robotertechnik existiert die eierlegende Wollmilchsau nicht.

Damit sind wir bereits bei der Motivation für die vorliegende Arbeit. Die Aufgabenstellung ist, einen mobilen Miniaturroboter zu entwerfen und zu realisieren. Das Konzept für den Roboter Tinyphoon sieht vor, dass dieser funktional beliebig erweiterbar und auch hardwaremäßig ausbaufähig ist.

Die Realisierung der mobilen Plattform Tinyphoons wird im Kapitel 4 der Arbeit ausführlich erläutert. Ein Hauptaugenmerk liegt dabei auf der am Roboter selbst befindlichen „Firmware“. Auch auf dessen Verbesserungs- und Erweiterungsmöglichkeiten wird eingegangen.

Der mobile Roboter Tinyphoon soll vor allem im Roboterfußball der Klasse MiroSOT eingesetzt werden. Dadurch sind bestimmte „mechanische“ Kriterien für den Roboter wie Dimensionen und die Fortbewegungsart vorgegeben. Bei der Auswahl und Dimensionierung der Elektronik-Hardware gibt es dahingehend keine Einschränkungen.

Die selbstdefinierten Anforderungen gehen allerdings noch weiter als lediglich die Konstruktion und Realisierung eines Fußballroboters: Vielmehr soll eine universelle Roboterplattform für unterschiedlichste wissenschaftliche Zwecke, aber auch für Anwendungsmöglichkeiten in Bauwirtschaft oder Logistik entwickelt werden.

Folgende Anforderungen werden für die mobile Roboterplattform festgelegt:

- modulare Architektur und gute Skalierbarkeit
- Geschwindigkeit größer 2 m/s
- Beschleunigung größer 5 m/s²
- Bewegungspräzision von weniger als 5 mm/m (0,5 %)
- hohe Akkulaufzeit: Bei permanenter Bewegung mindestens 10 Minuten Laufzeit
- Roboterfußballspezifische Eigenschaften
- Diagnosemöglichkeiten

2.1 Modulare Architektur

Tinyphoon soll generell offen gegenüber Hardware-Erweiterungen sein. Es soll jederzeit möglich sein, neue Sensortechnologien in das bestehende System zu integrieren, aber auch die Rechenleistung von Tinyphoon durch weitere Prozessoren zu verbessern.

Der Einsatz als wissenschaftliche Roboter-Plattform macht es zudem erforderlich, dass die Roboter-Software ebenfalls entsprechend modular aufgebaut ist. Beispielsweise sollen unterschiedliche Regelungsalgorithmen möglichst einfach in der Firmware praxisorientiert getestet werden können.

Die offene Architektur hat auch den Vorteil, dass Schwachstellen des Prototypen in den kommenden Generationen beseitigt werden können, ohne dass ein komplettes Re-Design von Hard- und Software erforderlich ist.

2.2 Geschwindigkeit

Der Roboter soll in der Lage sein, bei geradliniger Bewegung eine Endgeschwindigkeit von etwa 2,5 m/s zu erreichen. Damit der Roboter in Kurvenlagen ebenfalls gute Geschwindigkeitswerte erreichen kann, soll der Schwerpunkt möglichst tief liegen.

2.3 Beschleunigung

Speziell für den Einsatz als Fußballroboter ist die maximale Beschleunigung relevanter als die Endgeschwindigkeit. Der Roboter soll in weniger als 300 Millisekunden seine Endgeschwindigkeit erreichen können, die maximale Beschleunigung soll bei etwa 5 m/s^2 liegen. Die maximale Beschleunigung ist neben dem Antriebskraft des Motors in Relation zum Gesamtgewicht vor allem von der Haftreibung zwischen Rädern und dem Untergrund abhängig.

2.4 Bewegungspräzision

Die Bewegungen des Roboters sollen mit weniger als 5 mm / m Abweichung zur geforderten Bewegung durchgeführt werden.

2.5 Laufzeit

Auf den Gesamtstromverbrauch ist zu achten. Sämtliche Hardwareteile sollen in Hinsicht auf Stromverbrauch optimiert werden, so dass der Roboter ohne Akkuwechsel sich mindestens 10 Minuten ständig bewegen kann. Der Wert bezieht sich auf eine geradlinige Bewegung des Roboters ohne Beschleunigung. Sowohl beim Beschleunigen als auch beim Bremsen ist der Stromverbrauch speziell der Antriebsmotoren besonders hoch.

2.6 Roboterfußballspezifische Eigenschaften

Die folgenden Eigenschaften des Pflichtenhefts sind speziell für die Anwendung „Roboterfußball“ erforderlich:

- der Roboter soll in einen Würfel mit 75mm Seitenlänge passen
- symmetrischer Aufbau: Es gibt keine Vorder- oder Rückseite
- der Ball soll bei Bewegung des Roboters geführt werden können.

2.7 Diagnosemöglichkeiten

Um die korrekte Arbeitsweise des Roboters und seiner Komponenten gewährleisten zu können, sollen mehrere Diagnosefunktionen vorgesehen werden:

Diagnoseschnittstelle: Über eine eigene Schnittstelle sollen Diagnose- und Fehleranalyse-Informationen ausgegeben werden können. Damit soll die Roboter-Software getestet werden können.
Ladestandskontrolle: Der Roboter soll einen niedrigen Batterieladestand durch eine Kontrollleuchte anzeigen.

2.8 Mögliche Einsatzgebiete von Tinyphoon

Als erstes Einsatzgebiet, Spielwiese und Testumgebung dient Roboter-Fußball. In diesem Multi-Agenten-System spielen mehrere Roboter gegeneinander eine Art Fußball. Das Ziel ist – wie beim echten Fußballspiel – den Ball im gegnerischen Tor unterzubringen. Hier wird das Bewegungsverhalten und die Positioniergenauigkeit der Roboter untersucht.

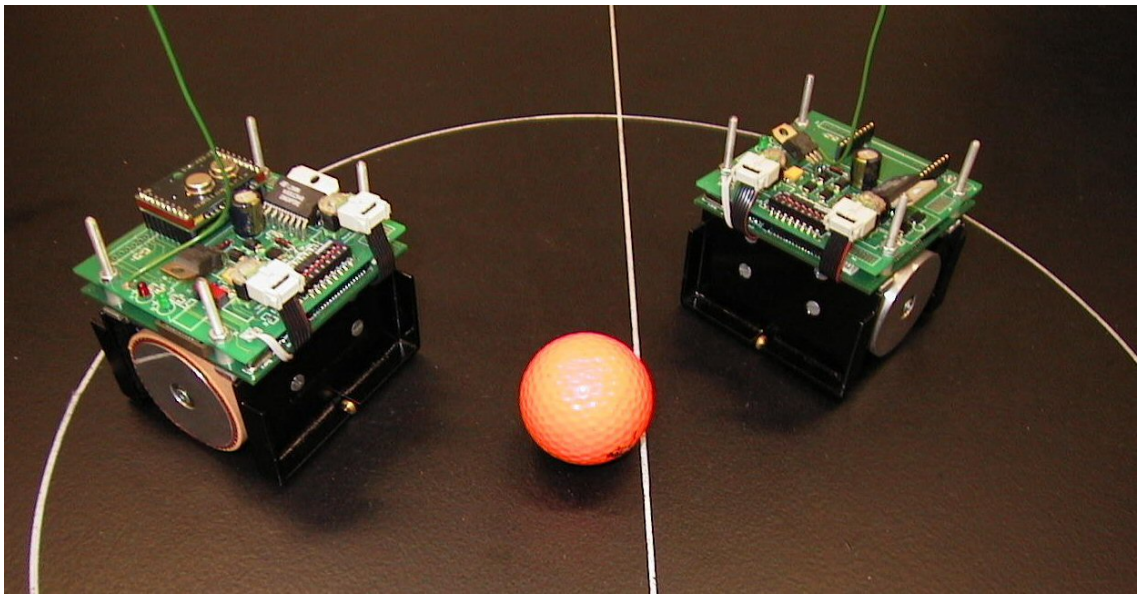

Abbildung 2.1: Mobile Miniroboter spielen Fussball

Prinzipiell ist der Roboter auch für andere Einsatzgebiete tauglich, beispielsweise im Bauwesen. Auch als Lernobjekt und für Untersuchungen der Robotertechnik ist Tinyphoon hervorragend geeignet.

3 Stand der Technik

Die Anforderungen an einen mobilen Roboter sind sehr eng mit der ihm zugedachten Aufgabe verbunden, also seiner Spezialisierung. In welchem Umfeld bewegt sich der Roboter? Wie schnell soll er sich bewegen können? Wie lange soll er ohne Energiezufuhr agieren können? Was soll er von seiner Umwelt wissen? Welche Tätigkeiten soll er verrichten? Soll er fest vorgegebene Arbeitsprogramme durchführen, oder soll er aus seinen Erfahrungen lernen können und danach sein Handeln richten oder gar planen?

All diese Fragen sind bereits beim Konzept für einen Roboter zu berücksichtigen und Lösungswege nach den gegebenen Möglichkeiten und Stand der Technik einzuplanen.

Die genannten Fragen und dementsprechenden Anforderungen lassen sich auf unterschiedlichste Art und Weise kategorisieren. Ein Ansatz wäre die Gliederung nach Wissenschaftsgebieten, ein anderer nach Teilaufgaben des Roboters, vorstellbar ist auch eine Einteilung nach Anforderungen. Betrachtet man den mobilen Roboter nach den „basic tasks“, also den rudimentären Teilaufgaben, so ergibt sich folgende Gliederung – analog zum menschlichen Vorbild und dem entsprechend zugeordnetem Wissenschaftsgebiet:

- Bewegung
- Sensorik
- Kommunikation
- Steuerung

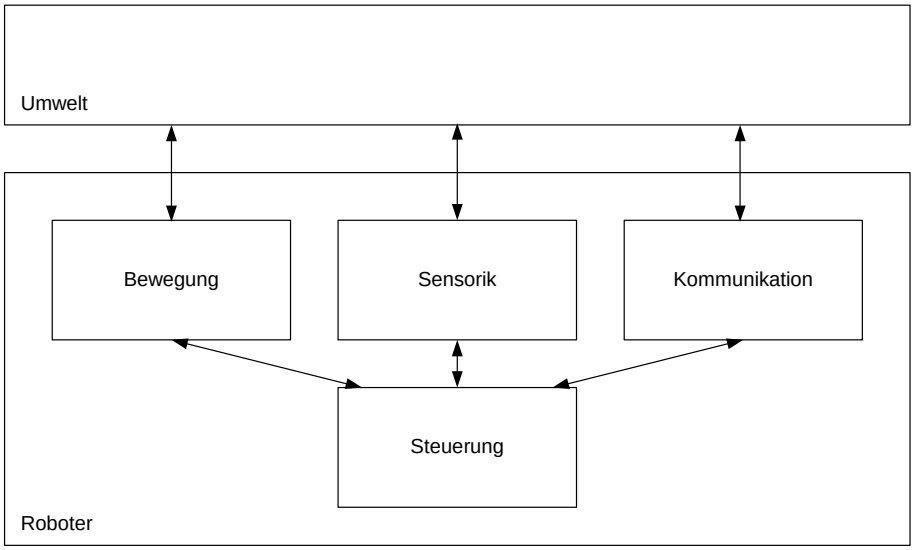

Abbildung 3.1:Teilaufgaben eines Roboters

Die folgende Tabelle ordnet die Teilaufgaben des Roboters den Wissenschaftsgebieten zu und stellt die Analogie zum Menschen her:

Teilaufgabe	*Beschreibung*	*Mensch*	*Wissenschaftsgebiet*
Bewegung	Fortbewegung und Aktoren	Extremitäten	Mechanik
Sensorik	Aufnahme von Information über die Umwelt sowie deren Verteilung	Sinnesorgane (Augen, Ohren, Nase, Haut)	Elektronik
Kommunikation	Informationsaustausch mit anderen Robotern, Computern, Menschen	Sinnesorgane (Mund, Ohren)	Nachrichtentechnik
Steuerung	Steuerung der Bewegung und Aktoren, Bewegungsplanung, Intelligenz, Lernfähigkeit	Vegetatives Nervensystem Gehirn	Informatik / Regelungstechnik

Tabelle 3.1: Vergleich der Roboteraufgaben mit menschlichen Funktionen

Robotik stellt daher einen sehr umfassenden, fachübergreifenden Forschungsraum dar. Jede Teilaufgabe für sich ist ein eigenes Forschungsgebiet. Aus diesem Grund ist der erste Teil der Arbeit mehr als Überblick zum Stand der Technik - verbunden mit einigen Anwendungsbeispielen - zu sehen, denn als Anleitung zum Roboterbau.

3.1 Bewegung

Die Art der Fortbewegung ist ein zentraler Aufgabenbereich eines mobilen Roboters. Durch sie werden im Großen und Ganzen folgende Attribute bestimmt:

- Geschwindigkeit
- Flexibilität
- Effizienz

Je nach Anforderungen an diese Attribute und Einsatzgebiet gibt es daher eine Reihe von Ansätzen für die Fortbewegung von mobilen Robotern. Folgende Bewegungssysteme werden vorrangig eingesetzt:

- Bewegung mit Rädern
- Bewegung mit Ketten
- Bewegung mit Gliedmaßen
- Bewegung in der Luft
- Bewegung im Wasser

In sehr vielen technischen Anwendungsgebieten wird bei der Natur Anleihe genommen und dementsprechend „abgeschaut“. Gerade bei der mechanisierten Fortbewegung hat der Mensch im Laufe der Zeit einige interessante Ansätze verfolgt, ohne dass es ein vergleichbares Vorbild

in der Natur gibt oder zumindest den Erfindern nicht bekannt war: Beispiele sind das Rad für die Fortbewegung an Land, das Schaufelrad und die Schiffsschraube für die Fortbewegung an der Wasseroberfläche, das Flugzeug oder die Rotorblätter des Helikopters für die Bewegung in der Luft. Daher ist nicht immer die Natur das Vorbild für technische Errungenschaften.

Dies gilt zum Teil auch für mobile Roboter, obwohl hier – wie bereits diskutiert – im weitesten Sinne der Mensch als Vorbild dient. Bei den heute existierenden Robotertypen können lediglich jene Exemplare, die sich mit Gliedmaßen fortbewegen, zu der Kategorie „designed by nature" zählen, und auch diese nur eingeschränkt: Es ist tatsächlich noch nicht gelungen, den Bewegungsapparat von Insekten oder Vierbeinern so nachzubilden, dass dieser puncto Geschwindigkeit und Effizienz in Relation zum Gewicht mit dem Vorbild mithalten kann. Die Nachbildung des menschlichen Bewegungsapparates ist noch viel komplexer, da hier nur 2 Gliedmaßen zur Verfügung stehen und zusätzlich der Gleichgewichtssinn nachgebildet werden muss. Allerdings hat die Wissenschaft heute bereits ganz andere Möglichkeiten, den Bewegungsvorgang von Lebewesen zu erforschen als noch vor 10-15 Jahren. Als Beispiel sei „Motion-Tracking", also das möglichst exakte Aufnehmen der Bewegungen von Menschen und Tieren genannt. Zumindest in 3D-Animationsfilmen zeigt diese Technik bereits ihr Potential. Warum soll auch der Mensch eine Technik in 20 Jahren beherrschen, für deren Verfeinerung die Evolution hunderte Millionen Jahre benötigt hat?

Ansonsten werden für mobile Roboter hauptsächlich konventionelle mechanische Fortbewegungstechniken eingesetzt, wie Räder, Ketten, Schiffs- oder Flugzeugtechnologie. Diese Technologien sind zwar bezüglich Einsatzmöglichkeiten weitaus weniger flexibel, dafür jedoch hoch spezialisiert. Außerdem kann auf eine recht ausgereifte Technik zurückgegriffen werden, und vor allem in puncto Geschwindigkeit sind alle diese Bewegungsarten unübertroffen. Kein Tier kann sich an Land so rasch bewegen wie ein berädertes Fahrzeug auf einer Straße, und kein Vogel so schnell fliegen wie ein Flugzeug.

Fasst man den Begriff „mobiler Roboter" etwas enger und betrachtet man lediglich terrestrische Roboter, so können zwei der genannten Bewegungssysteme als relevant angesehen werden: Roboter mit Rädern oder Ketten und Roboter mit Gliedmaßen.

Bewegung mit Rädern

Mit der Erfindung des Rades hat der Mensch einen weiteren Meilenstein zur Entwicklung unserer Zivilisation geleistet. Die schnellsten terrestrischen Fahrzeuge – also jene mit permanentem Bodenkontakt – sind berädert. Die Geschwindigkeit ist daher das Hauptargument für den Einsatz von Rädern bei mobilen Land-Robotern.
Allerdings haben Räder auch einen gravierenden Nachteil: Es setzt eine entsprechende Infrastruktur, und zwar eine möglichst geglättete, ebene Fahrunterlage voraus, sei das nun Straße oder Schiene. Sobald eine gewisse Unebenheit erreicht ist, kann sich der Roboter auf den Terrain nicht mehr bewegen. Abhilfe würde hier – wie bei Geländefahrzeugen üblich – lediglich eine Seilwinde schaffen, aber das auch nicht immer und überall. Die Bewegung wird dann äußerst mühsam und zeitintensiv, der Geschwindigkeitsvorteil wäre dahin.

Keinesfalls sind mit Rädern ausgestattete Roboter immer mit Automobilen zu vergleichen. Es gibt in puncto Anzahl, Anordnung und Antrieb der Räder mehrere grundverschiedene Systeme:

- differentieller Antrieb
- synchroner Antrieb
- 1 gelenktes Rad
- Ackerman Steuerung

Die Auswahl des entsprechenden Systems hängt hier vor allem von mechanischen und ökonomischen Gesichtspunkten ab, wenngleich es auch technische und praktische Unterschiede gibt (z. B. Wenderadius, Stabilität).

Bevor auf die einzelnen Systeme und deren Vor- und Nachteile eingegangen wird, werden kurz verwendete Terminologien und Grundlagen der Bewegung mit Rädern erläutert:

Bewegung mit Ketten

Im Grunde sind Kettenfahrzeuge Radfahrzeugen von der Antriebsart recht ähnlich. Die Kette soll hier vorrangig dazu dienen, die Geländegängigkeit durch bessere Bodenhaftung und höhere Stabilität des Fahrzeuges zu verbessern. Nachteile der Bewegung mit Ketten sind die geringere mögliche Geschwindigkeit wie auch der Wirkungsgrad der Umsetzung von Energie in Bewegung.

Bewegung mit Gliedmaßen

Die Bewegung mit Gliedmaßen stellt im Gegensatz zu jener mit Rädern den von der Natur entliehenen Ansatz dar. Diese Fortbewegungsart kann zwar bezüglich Geschwindigkeit nicht mit der beräderten Konkurrenz mithalten, ist aber bei gleichen Größenverhältnissen dafür weitaus flexibler und geländegängiger. Beispielsweise für Planeten-Expeditionen, bei denen die Oberflächenverhältnisse nicht von vorneherein feststehen, ist ein Roboter mit Gliedmaßen weit im Vorteil. Lediglich die erst junge Technik ist ein Grund, warum zu diesen Zwecken zumeist die Rad-Kollegen eingesetzt wurden.

Die Anzahl der Gliedmaßen kann dabei von 1 (!) bis 12 oder noch mehr gehen. Je mehr Gliedmaßen, desto höhere Stabilität kann erreicht werden.

Bewegung in der Luft

Der Vollständigkeit halber sei auch die Fortbewegung in der Luft genannt. Im Grunde handelt es sich hierbei um die bei mechanisierten Fluggeräten wie Flugzeugen und Helikoptern genutzten Technologien.

Flugroboter ähneln Modell-Fliegern oder –Hubschraubern und werden vor allem zur Erforschung von Vulkanen, aber vermehrt auch in der militärischen Aufklärung eingesetzt (Drohnen).

Bewegung im Wasser

Auch Roboter zur Bewegung im Wasser sind eher spärlich zu finden. Auch hier wird auf existierende und hochentwickelte Technologien gebaut, wie sie in der Schiffstechnik Einsatz finden. Hier werden ein oder mehrere Schiffsschrauben eingesetzt, mit denen der Roboter sowohl angetrieben als auch gesteuert wird. Zusätzlich können noch Ruder zum Einsatz kommen.

Unterwasserroboter werden vor allem zur Erforschung des Meeresbodens und der Tiefsee eingesetzt.

Die folgende Tabelle vergleicht die unterschiedlichen Bewegungsarten:

Kriterium	*Räder*	*Ketten*	*Gliedmaßen*	*Luft*	*Wasser*
Mögliche Geschwindigkeit	Hoch	Mittel	Niedrig	Hoch	Niedrig
Umgebung	mehr oder weniger ebene Oberflächen, wohldefinierte Umgebung	auch unwegiges Gelände	auch unwegiges Gelände	Luft	Wasser
Komplexität der Bewegungssteuerung	niedrig-mittel	niedrig	hoch	mittel	mittel
Wirkungsgrad	hoch	mittel	mittel	niedrig	hoch

Tabelle 3.2: Vergleich der Bewegungsarten

Zu den Grundlagen der Bewegung von Robotern siehe auch [Nehmzow2000].

3.2 Sensorik

Neben einem Bewegungsapparat benötigt ein mobiler Roboter Sinnesorgane – die Möglichkeit, Informationen aus seiner Umwelt aufzunehmen. Bewegung ohne die eigene Position zu wissen oder die Umwelt zu kennen, macht zumeist keinen Sinn. Die Sensorik bietet eine Unzahl von Möglichkeiten, Daten aus dem Umfeld aufzunehmen. Die Herausforderung liegt hier eher darin, aus der Fülle von Daten die relevanten herauszufiltern und sie vor allem richtig zu kombinieren und zu interpretieren.

Sensoren für mobile Roboter können nach unterschiedlichen Gesichtspunkten gegliedert werden:

- wofür wird die Information verwendet?
- Art der Information?
- wie werden die Daten aufgenommen?

Die Wahl der Sensoren hängt bereits davon ab, wofür die dadurch gewonnenen Informationen eingesetzt werden sollen. Die Daten, die aufgenommen werden, können unterschiedlichen Zwecken dienen:

- **Modellierung**: Die Sensordaten dienen dazu, die Umwelt zu modellieren
- **Bewegungsrückmeldung**: Der Roboter möchte Informationen, ob die Bewegungen, die er durchführen wollte, auch tatsächlich durchgeführt wurden. Der Sensor ist hier Teil einer Regelschleife.
- **Positionserfassung**: Der Sensor gibt relevante Daten über die eigene absolute oder relative Position zurück.
- **Kollision**: Mit Hilfe des Sensors werden Kollisionen mit der Umwelt – entweder durch direkten Kontakt oder auf Entfernung - erkannt.
- **Status**: Der Sensor liefert Informationen über den internen Status des Roboters, z. B. Batterieladung.

In vielen Fällen ist die gelieferte Information mehreren Zwecken dienlich, die Erfassung einer Distanz kann z. B. der Modellierung, der Positionserfassung oder der Kollisionserkennung dienen.

Die Umwelt

Die Auswahl, Anzahl und Plazierung der eingesetzten Sensoren bei einem mobilen Roboter hängt wiederum von seiner Aufgabe und vor allem von seiner Umwelt ab.

Man kann die Umwelt nach Struktur und Aufbau unterscheiden:

- strukturiert
- halb-Strukturiert
- unstrukturiert

Die Umwelt eines Roboters kann strukturiert sein, d.h. alle Hindernisse haben eine bestimmte Form oder liegen auf einem bestimmten Raster. Als Beispiel sei hier eine Lagerhalle genannt, die in bestimmte Bereiche gegliedert ist und der Roboter sich nur auf bestimmten Routen bewegen darf. Der Roboter kennt in etwa das Umfeld oder weiß zumindest, womit er rechnen kann.
Das Umfeld kann aber auch unstrukturiert sein, d.h. es gibt keinerlei a priori Wissen über den Aufbau, die Anordnung oder die Größe von Hindernissen. Die freie Natur ist ein Beispiel für eine unstrukturierte Umgebung.

Als halb-strukturierte Umwelt bezeichnet man sinngemäß Einsatzgebiete, die zwar eine gewisse Regelmäßigkeit aufweisen, aber nicht unbedingt eine feste Struktur aufweisen. Gebäude im Allgemeinen können als halb-strukturiert bezeichnet werden.

Ein weiteres Kriterium ist die Veränderlichkeit der Umgebung: Bleibt das Umfeld immer gleich, oder gibt es Hindernisse, die sich bewegen können? Man spricht von

- statischer bzw.
- dynamischer Umgebung.

Wie man sieht, ist bereits bei der grundlegenden Konzipierung darauf Rücksicht zu nehmen, wie das Einsatzgebiet des Roboters aussieht und was ihn dort erwartet und vor allem, was er an a priori Wissen einbringen kann. In einer strukturierten, von Menschen geschaffenen und für den Einsatz von mobilen Robotern gedachten Umgebung kann für bestimmte Sensoren erforderliche Infrastruktur vorgesehen und eingebaut werden (z. B. Farbmarkierungen, Induktionsgeber..). Selbst bei dynamischer Umgebung kann diese berücksichtigt werden, sofern es sich bei den beweglichen Hindernissen um Menschen oder andere Roboter handelt (z. B. bestimmtes Aussehen, Funksender, um den Roboter zu warnen).

Man kann sich allerdings vorstellen, dass die belebte Natur als unstrukturiertes und dynamisches Umfeld mit unterschiedlichsten Hindernissen die größte Herausforderung für einen mobilen Roboter darstellt.

Die wichtigsten Sensortypen für mobile Robotersysteme sind:

- Kontaktsensoren
- Beschleunigungssensoren
- Magnetische Sensoren (Kompass, ...)
- Raddrehzahlgeber
- Optische Sensoren (Kameras, Laserscanner, ...)
- Satellitenbasierende Sensorik (GPS)
- Infrarotsensoren
- Ultraschallsensoren

Die folgende Tabelle gibt einen Überblick über die möglichen Anwendungsgebiete verschiedener Sensortypen:

Sensortyp	*Anwendung*
Kontaktsensoren	Kollisionsdetektion
Beschleunigungssensoren	Bewegungsrückmeldung Kollisionsdetektion
Magnetische Sensoren	Richtungserkennung Positionserfassung (gemeinsam mit Raddrehzahlgeber) Bewegungsrückmeldung
Raddrehzahlgeber	Bewegungsrückmeldung Positionserfassung (gemeinsam mit Gyro)
Optische Sensoren	Modellierung

Satellitenbasierende Sensorik	Positionserfassung
Infrarotsensoren	Kollisionsdetektion
Ultraschallsensoren	Modellierung Kollisionsdetektion

Tabelle 3.3: Einsatzgebiete von Sensortypen für verschiedene Roboteraufgaben

3.3 Kommunikation

Die vom mobilen Roboter durchgeführten Bewegungen und Aktionen basieren auf einer oder mehrerer Grundaufgaben. Diese können entweder fester Bestandteil der Roboter-Firmware sein, vor allem, wenn es sich um immer wiederkehrende, gleichartige Aufgaben handelt. Diese hochspezialisierten Roboter durchlaufen zumeist fest vorgegebene Sequenzen von Tätigkeiten.

Oder aber der Roboter empfängt seine Aufgaben von einem externen Befehlsgeber. Dieser Befehlsgeber kann entweder ein Mensch oder eine Maschine bzw. Computeranlage sein. In diesem Falle ist eine Kommunikationsschnittstelle erforderlich.

Mobile Roboter benötigen aber auch eine Kommunikationseinheit, wenn sie mit anderen Robotern Daten bzw. Informationen austauschen.

Folgende Aspekte sind für die Auswahl des Kommunikationsverfahrens zu berücksichtigen:

- Kommunikationspartner: spricht der Roboter mit Menschen oder mit Maschinen bzw. anderen Robotern?
- Kommunikationsmedium: z. B. Schall, Funk, Licht
- Kommunikationsrichtung: unidirektional, bidirektional
- Anzahl der Kommunikationsteilnehmer
- Zugriff zum Kommunikationsmedium
- Störsicherheit

Kommunikationspartner

Die Art der Kommunikation ist vorrangig davon abhängig, von wo der Befehl kommt und welches Übertragungsmedium eingesetzt wird. Man kann grundsätzlich unterscheiden:

- Mensch-Maschine Kommunikation (HMI; Human Machine Interface)
- Maschine-Maschine Kommunikation

Ist der Kommunikationspartner ein Mensch, dann muss der Roboter menschliche Sprache „verstehen“ und unter Umständen auch sprechen können. Bei über 10.000 auf der ganzen Welt existierenden Sprachen und Dialekten ist auch die Frage nach der Sprache eine berechtigte.

Handelt es sich dagegen um Maschine – Maschine Kommunikation, so sind gänzlich andere Ansätze zu verfolgen. Je nach Anforderungen und Art der zu übertragenden Informationen ist ein Protokoll (die „Sprache“) zu entwerfen bzw. auszuwählen.

Kommunikationsmedium

Das Kommunikationsmedium ergibt sich einerseits aus dem erforderlichen Kommunikationspartner und andererseits aus den Umweltbedingungen. Grundsätzlich ist für mobile Roboter „drahtlose“ Kommunikation vorzusehen, da eine ständige Kabelverbindung wohl nicht praktikabel wäre. Folgende Kommunikationsmedien können grundsätzlich zur drahtlosen Informationsübertragung eingesetzt werden:

- Schall
- Funk
- Infrarot

Eine Informations-Übertragung mittels Schall wird wohl hauptsächlich für Mensch-Maschine Kommunikation eingesetzt werden.

Für Maschine – Maschine Kommunikation bietet sich die Übertragung per Funk an. Diese Art der Datenübertragung ist nach jahrzehntelangen Entwicklungen und Verbesserungen heute bereits recht ausgereift, es existieren eine Reihe von einsatzbereiten Verfahren und kommerziellen Produkten. Die gewählte Technologie und vor allem das verwendete Frequenzband entscheiden dabei über Reichweite und Datendurchsatz: Grundsätzlich gilt: je höher das Frequenzband, desto höher die nutzbare Bandbreite und damit verbunden der Datendurchsatz. Allerdings sinkt mit steigender Frequenz die Durchdringbarkeit von Gebäuden oder anderen Hindernissen, wodurch insgesamt die Reichweite geringer wird. Durch das Ausbreitungsverhalten von elektromagnetischen Wellen ist aber ein unmittelbarer Sichtkontakt nicht zwingend erforderlich.

Infrarot – Datenübertragung bietet sich vor allem für kurze Distanzen an, wenn es sich um eine extrem stark gestörte Funk-Umgebung handelt. Sichtkontakt zwischen Sender und Empfänger ist unbedingt erforderlich.

Die Wahl des Kommunikationsmediums ist daher vorrangig vom Kommunikationspartner, vom gewünschten Datendurchsatz und von den Umweltbedingungen abhängig.

Kommunikationsrichtung

Weiters muss unterschieden werden, ob der Roboter „zurückreden“ darf, sprich, ob er lediglich empfangen oder auch senden können soll. Man spricht von unidirektionaler bzw. bidirektionaler Kommunikation.

Unidirektionale Kommunikation hat nicht nur den Nachteil, dass Informationen nur in eine Richtung ausgetauscht werden können. Der Sender erhält auch keinerlei Rückmeldung darüber,

ob der Empfänger die Nachricht korrekt verstanden hat oder nicht (Acknowledge). Bidirektionale Kommunikation ist daher auch sicherer.

Anzahl der Kommunikationsteilnehmer

Ein vierter Aspekt ist die gleichzeitige Anzahl an Kommunikationsteilnehmern: sprechen immer maximal 2 Teilnehmer miteinander, oder soll auch eine „an alle“-Kommunikation (Broadcasting) vorgesehen sein?

Zugriff zum Kommunikationsmedium

Vor allem für die Situation, dass 2 oder mehr Gesprächspartner von sich aus eine Kommunikationsverbindung aufbauen dürfen, muss ein geeignetes Verfahren überlegt werden, wer wann sprechen darf. Dieses Verfahren wird auch Medium Access Control (MAC) genannt.

Störsicherheit

Auch wir Menschen verstehen nicht immer alles ganz genau, was uns gesagt wird. Die menschliche Sprache enthält aber so viel Redundanz – also grundsätzlich überflüssige Information – dass wir in vielen Fällen, in denen wir nur einen Teil verstanden haben, trotzdem wissen, was gemeint wurde. Nicht nur die Redundanz der Sprache, sondern auch die Causalität hilft uns, Unvollständiges zu verstehen.
Auch bei der Maschine-Maschine Kommunikation sollte immer eine gewisse Störsicherheit enthalten sein. Diese wird im Grunde ebenfalls durch Redundanz erreicht, im Allgemeinen durch Anfügen von Checksummen wie CRC (Cyclic Redundancy Check) Wörtern.
Man unterscheidet hier grundsätzlich zwischen

- Fehlererkennung und
- Fehlerkorrektur

Bei der Fehlererkennung wird lediglich erkannt, dass die empfangenen Informationen nicht korrekt sind. Der Roboter kann aber insofern reagieren, als er seinem Kommunikationspartner mitteilt, er soll diese Nachricht wiederholen. Im einfachsten Fall ignoriert er sie und wirft sie weg. Ein bekanntes Beispiel für Fehlererkennung ist das Parity Bit bei Speichern oder asynchronen Schnittstellen.

Dagegen nützt die Fehlerkorrektur die in der Nachricht enthaltene Redundanz dazu, den Fehler ausfindig zu machen und zu korrigieren. Dies ist aber – je nach Art und Umfang der redundanten Information – nur bis zu einem gewissen Störgrad möglich. Eine Fehlererkennung ist aber auch bei höherem Störgrad noch möglich. Bekannte Beispiel und Verfahren der Fehlerkorrektur sind CRC-Codes bei verschiedenen Protokollen oder der ECC (Error Correcting Code) bei Speichern.

3.4 Steuerung

Die Steuerung ist der weitaus komplexeste, umfangsreichste und zugleich interessanteste Teil eines mobilen Roboters. Sie sorgt – stark vereinfacht - quasi für das Aufnehmen der Sensordaten, die entsprechende Interpretation der gewonnenen Informationen und setzt als Ergebnis eine Handlung wie z. B. eine Bewegung in eine bestimmte Richtung.

Von diesem stark vereinfachten Modell ausgehend kann die Steuerung (der Begriff „Steuerung“ ist hier als allgemeiner Begriff zu verstehen und nicht zu verwechseln mit jenem in der Steuerungs- und Regelungstechnik) des mobilen Roboters als eine einzige große Regelung angesehen werden, allerdings eine sehr komplexe. Auch der Mensch an sich könnte ja im weitesten Sinne als „Regelung“ angesehen werden, die wiederum aus einer Unzahl an kleineren Regelungen besteht. Würde man allerdings den Menschen auf diese rein naturwissenschaftliche Sicht reduzieren, wäre bei Ethikexperten und Theologen ein starker Protest oder zumindest ein mildes Lächeln die Folge.

Grundsätzlich kann man die Steuerung eines mobilen Roboters in

- Hardware und
- Software

gliedern. Unter Hardware ist hier vor allem die oder eingesetzte Prozessor-, Speicher- und Input/Output - Technologie zu verstehen. Die Software nutzt die zugrundeliegende Hardware, um die vorgesehenen Steuerungsaufgaben zu übernehmen.

3.5 Hardware

Die Wahl der Steuerungshardware ist vor allem eine Frage der benötigten Leistungsfähigkeit. Es nicht immer leicht abzuschätzen, wie viel Rechenleistung und Arbeitsspeicher für die vorgesehene Steuerungsaufgabe benötigt wird. Daher: Je mehr Leistungsreserve desto besser? Hauptsächlich 2 Gründe sprechen dagegen, den besten und schnellsten Prozessor und Unmengen von Speicher einzusetzen:
Einerseits ist klar, dass die jeweils besten Prozessoren und Speicher einer Zeit auch die teuersten sind. Mag für Forschungszwecke und einzeln produzierte Roboter der Kostenfaktor ein vernachlässigbares Argument sein, so gilt dies keinesfalls für eine etwaige Serienfertigung oder komerzielle Nutzung.
Das zweite Argument ist ein technisches: der Stromverbrauch eines Systems ist bis zu einem gewissen Grad proportional zur Rechenleistung. Je mehr Transistoren auf einem Chip untergebracht sind und je höher die Taktfrequenz ist, desto mehr Leistung benötigt der Prozessor. Durch kleinere Fertigungsstrukturen (Stichwort „Nanotechnologie“) kann die Leistungsaufnahme jedoch wieder reduziert werden. Für ein mobiles System ist der

Energieverbrauch der einzelnen Komponenten von zentraler Bedeutung für die Gesamtlaufzeit des Systems, und dies gilt auch für mobile Roboter.

Auswahl und Dimensionierung

Zur Auswahl einer Prozessor-Speicher-Kombination gilt: Es sollte zuerst die Steuerungs-Software so weit konzipiert werden, dass die benötigte Rechenleistung in etwa abgeschätzt werden kann, und erst danach die Hardware ausgesucht werden, unter Berücksichtigung einer mehr oder weniger großzügigen Leistungsreserve.

Die Auswahl und Dimensionierung der Steuerhardware für einen mobilen Roboter reduziert sich im allgemeinen auf die Frage von Prozessor und Speicher. Dabei sind – wenn man einmal von den Kosten absieht - allerdings nicht nur Leistungsfähigkeit und Datendurchsatz, sondern auch weitere Kriterien relevant:

- **Leistungsfähigkeit**: Wie viele Befehle kann das System pro Sekunde verarbeiten?
- **Fließkomma-Befehle**: Kann der Prozessor Fließkommaberechnungen durchführen?
- **Multimedia-Befehle**: Beherrscht der Prozessor Spezialbefehle für bestimmte Daten?
- **Stromverbrauch**: Energieverbrauch unter Volllast, Welche Stromsparmodi gibt es?
- **Ein-, Ausgabe**: Wie kommuniziert der Prozessor mit Sensoren, Aktoren und anderen Einheiten?
- **Skalierbarkeit**: Falls weitere Leistung benötigt wird, wie kann eine Erhöhung erreicht werden?
- **Entwicklungswerkzeuge**: Welche Tools und Compiler existieren zu dem Prozessor?

Leistungsfähigkeit

Die Leistungsfähigkeit eines elektronischen Rechnersystems ist von verschiedenen Faktoren abhängig. Grundsätzlich sind folgende Eigenschaften von Prozessor und Arbeitsspeicher relevant:

- **Prozessortakt**: 1 MHz bis 2,5 GHz
- **Interne Rechenbreite** des Prozessors: 4 Bit bis 128 Bit
- **Prozessorarchitektur**: Pipelining, Befehlssatz (CISC, Risc)
- **Prozessorinterner Cache**: Größe, Geschwindigkeit und Zugriff, 1 KByte bis 1024 KByte
- **Arbeitsspeichergröße**: 1 Kbyte bis 4 Gbyte
- **Speichertechnologie**: Statisches RAM (Cache)
- **Speicherbusbreite**: 4 Bit bis 128 Bit
- **Speichertiming**: 1 MHz bis 500 MHz

Wieviel Leistung tatsächlich benötigt wird, lässt sich vielfach schwer abschätzen. Als Hauptkriterien können einerseits der benötigte Datendurchsatz bzw. die Datenmenge an den Schnittstellen und andererseits die geforderten Rechenaufgaben herangezogen werden.

Fließkomma-Befehle

Speziell für Rechenaufgaben oder Signalverarbeitung sind oft Berechnungen in Fließkomma-Arithmetik erforderlich. Gerade Regelungssysteme – und mobile Roboter können mehrere Regelungssysteme beinhalten - benötigen oft Fließkomma oder zumindest Festkomma-Arithmetik.

Sollte der ins Auge gefasste Prozessor keine derartigen Befehle bieten, so ist zu klären, ob Compiler existieren, die Fließkomma emulieren können. Dies ist allerdings mit großen Performanceeinbußen und hohem Speicherbedarf verbunden.

Multimedia-Befehle

Oft muss eine **große Datenmenge** wie z. B. Bilddaten, Audiodaten oder 3D-Informationen nach immer dem gleichen Schema bearbeitet werden. Die sogenannten „Multimedia“-Befehle wurden speziell für diese Zwecke geschaffen. Zumeist ist es eine eigene Recheneinheit innerhalb des Prozessors, welche diese Befehle abarbeitet. Die bekanntesten Multimedia-Befehlssätze sind jene der Desktop-Prozessoren der x86 Familie wie MMX ™, 3Dnow ™ oder ISS ™ der führenden Hersteller Intel ™ und AMD ™ .

Daneben existieren speziell bei digitalen Signalprozessoren Befehle zur Parallelverarbeitung von Daten, Befehle zum Addieren und Multiplizieren in einem Schritt oder das nfache Ausführen eines Befehles oder einer Befehlssequenz ohne Pipelinig-Zeitverluste.

Diese besonderen und oft vernachlässigten Befehle sollten jedenfalls bei der Auswahl eines Prozessors berücksichtigt werden, falls Video-, Bild- oder Audiodaten verarbeitet werden.

Stromverbrauch

Mobilität ist das Schlagwort für Elektronikprodukte der Gegenwart und der nahen Zukunft: Notebooks, Mobilfunktelefone, Personal Digital Assistants, MPEG-Player, Übersetzungsgeräte und vieles mehr sind derzeit die großen Wachstums- und Hoffnungsmärkte.
Nicht zuletzt aus diesem Grund sind immer mehr Halbleiterbausteine – und hier nicht nur Prozessoren – in Hinsicht auf Stromverbrauch optimiert.

Einerseits wird dies durch **Verkleinerung der Strukturen** erreicht („Nanotechnologie“ – Halbleiterstrukturen werden heute mit 120 nm Strukturbreite gefertigt). Je weniger Fläche ein Transistor auf dem Halbleiter einnimmt, desto weniger Strom benötigt er.

Ein weiterer Ansatz sind **Stromsparmodi**: dabei sollen nur jene Komponenten aktiv sein und damit Energie verbrauchen, die zur Zeit tatsächlich genutzt werden. Alle anderen Systemteile sind in einer Art Schlaf und verbrauchen nur wenig bis gar keinen Strom.
Immer mehr Prozessoren und Controller bieten daher Stromsparmodi, bei denen der Prozessor als einer der Hauptstromverbraucher oft mehrere Stufen dieses Schlafes einnehmen kann. Die wirkungsvollste Stromsparstufe wird dabei oft durch Ausschalten des Prozessortaktes erreicht, und nur ein externes Ereignis (z. B. Interrupt) kann ihn wieder aufwecken.

Ein-/Ausgabe

Die Kernaufgabe eines Steuerungsprozessors ist die Erfassung und Verarbeitung von Sensordaten und die Ansteuerung von Aktoren (z. B. Antriebsmotoren). Daher müssen für die Anbindung von Sensoren und Aktoren entsprechende Ein-/Ausgabe Schnittstellen existieren.

Sensoren können Daten in der unterschiedlichsten Form liefern. Dementsprechend gibt es eine Reihe von Schnittstellen zwischen Sensor und Steuerung:

Für Sensoren, die **analoge Daten** liefern, ist ein Analog/Digital-Wandler erforderlich. Je nach geforderter Präzision und Geschwindigkeit existieren A/D-Converter mit Auflösungen von 8 Bit bis 24 Bit und Wandelzeiten im Nanosekundenbereich bis zu 100µs.

Die Mehrzahl der Sensoren liefern allerdings bereits **digitale Daten**. Im einfachsten Fall handelt es sich um Schalter oder Taster die entweder offen oder geschlossen sein können. Intelligentere Sensoren liefern die Daten bereits formatiert, einem bestimmten Protokoll folgend. Es existieren unterschiedlichste Protokolle für den Datenaustausch, serielle und parallele Datenübertragung, synchrone oder asynchrone. Diese können proprietär oder standardisiert sein, wie zum Beispiel Feldbussysteme. CAN (Controller Area Network), I²C und Profibus sind derartige Datenaustauschsysteme.

Bei den anzusteuernden Aktoren handelt es sich zumeist um Elektromotoren, zumeist Gleichstrommotoren. Die Drehzahl eines Gleichstrommotors wird über die angelegte Gleichspannung eingestellt. Ein verbreitetes Verfahren ist die Pulsweitenmodulation. Näheres dazu im Kapitel 4.6 der vorliegenden Arbeit.

Es existieren also eine ganze Reihe von **Schnittstellen**, um von Sensoren Daten einzulesen oder Aktoren anzusteuern. Die Ein-/Ausgabe-Schnittstellen können entweder durch externe Bausteine realisiert werden, oder sie sind bereits im Prozessor integriert. Im zweiten Fall spricht man von Mikrocontrollern, die neben einem Prozessorkern und unterschiedlichsten Peripherieeinheiten oft noch Speicher enthalten und auch Ein-Chip-Computer genannt werden.

Skalierbarkeit

Eine bei der Konzipierung oft unterschätzte Eigenschaft einer Steuerung ist die Skalierbarkeit. Darunter versteht man im weitesten Sinne die **Erweiterungsfähigkeit** eines Systems. Auftretende Leistungsengpässe oder Speicherprobleme durch immer mehr Funktionalität sind in einem gut skalierbaren System leichter in den Griff zu bekommen.

Sollte durch die Komplexität der Software die Leistungsgrenze des Prozessors erreicht sein, gibt es mehrere Möglichkeiten, neue Ressourcen zu schaffen: Die einfachste Variante ist das Erhöhen des Prozessortaktes. Mangelt es an Arbeitsspeicher, so sollte es noch Spielraum für eine Speichererweiterung geben.

Sind diese beiden Varianten ausgeschöpft, so bieten sich 2 Wege an: Eine Möglichkeit wäre ein Redesign mit einer neuen Technologie. Dies bedeutet allerdings einige Schritte zurück, bei Inkompatibilität der Architektur sind – je nach Programmiersprache und –struktur – gute Teile des Codes nicht mehr verwendbar.

Daher ist der bessere Weg oft der Einsatz eines weiteren Prozessors. Die Skalierbarkeit einer CPU zeigt sich vorrangig durch die **Multiprozessorfähigkeit**, also vor allem der Verfahren, wie gemeinsame Daten ausgetauscht und verwaltet werden.

Wie bereits eingangs erwähnt, sollte die Leistungsfähigkeit eines Rechnersystems auf dem Software-Design basieren. Da das Abschätzen des Leistungsbedarfes oft ungenau und schwierig ist und oftmals noch nach der Entwurfsphase Änderungen und Erweiterungen zum System kommen, sollte die Skalierbarkeit des Prozessors berücksichtigt werden. Das richtige Maß an Leistungsreserve und die Multiprozessorfähigkeit eines Systems sind durchaus relevante Kriterien.

Entwicklungswerkzeuge

Nicht nur die technischen Eigenschaften eines Prozessors sind als Auswahlkriterium heranzuziehen. Was nützt der beste Allround Prozessor mit hoher Leistungsfähigkeit, mehreren Stromsparmodi und einer Unzahl von integrierter Peripherie, wenn keine adäquaten Entwicklungswerkzeuge dafür existieren? Etliche interessante, neuartige Prozessorarchitekturen haben sich nicht durchgesetzt, weil es unzureichende Tools und Compiler dafür gab, und nicht unbedingt zu schlechtes Marketing.

An forderster Front bei den Entwicklungswerkzeugen steht natürlich der **Assembler** und der **Hochsprachencompiler** (zumeist C oder C++). Diese sind Grundvoraussetzung dafür, dass ein Prozessor überhaupt vernünftig genutzt werden kann. Verbunden mit dem Compiler ist oft eine Entwicklungsumgebung, welche ein komfortables Editieren und Compilieren ermöglicht. Diese muss aber nicht zwangsläufig vom gleichen Hersteller sein, es existieren mehrere Universalumgebungen und Editoren, die diese Aufgabe übernehmen können.

Da ein In-System Test der Software – vor allem in der Projekt-Anfangsphase - nicht immer problemlos möglich ist, kann ein **Simulator** sehr nützlich sein. Oft wird zeitgleich mit der Hardware- und Softwareentwicklung begonnen, und es vergehen Monate, bis die erste funktionstüchtige Platine verfügbar ist. Damit kann die Software auf einem herkömmlichen Hostcomputer getestet und einzelne Module auf grundsätzliche Fehler untersucht werden.

Ein Simulator ersetzt allerdings keine **Echtzeit-Debug-Tools**, bei denen die Software direkt in der Zielhardware (Target) läuft. Es kann zwischen Software-Debugger und Hardware-Debugger unterschieden werden. Software-Debugger bedienen sich eines Software-Monitors, welcher mit der eigentlichen Software mit-compiliert wird und am Target läuft. Hardware-Debugger nutzen dagegen eine genormtes Hardware-Interface des Prozessors, um diesen gezielt anzuhalten oder

Speicherbereiche zu beschreiben bzw. auszulesen. Allerdings unterstützen bei weitem nicht alle Prozessoren diese Schnittstelle.

Nützlich sind auch verschiedene andere **Software-Werkzeuge**, mit welchen zum Beispiel direkt die Hardware angesteuert und getestet werden kann.

Nicht zu vergessen sind **Hardware-Kits**: Da eigene Hardware zu Beginn der Entwicklung noch nicht zur Verfügung steht, ist ein sogenanntes „Evaluation-Board“ (Entwicklungs-Platine) oder ein Starter Kit sehr nützlich. Möchte man nur 1 oder 2 Roboter bauen und stellt man an Form und Größe bei der Platine nicht allzu große Anforderungen, so können diese fertigen und recht günstigen Platinen auch gleich als Zielhardware eingesetzt werden. Die Hardwareentwicklung fällt dann ganz weg oder reduziert sich auf einige wenige externe Beschaltung.

Die Tool-Landschaft eines Prozessors ist mindestens genauso wichtig wie dessen Leistungsfähigkeit und andere technische Eigenschaften. Die Halbleiterhersteller sind sich dessen durchaus bewusst und sorgen dafür, dass die Software-Entwicklung so komfortabel wie möglich ist. Trotzdem sollte die Verfügbarkeit und die Fähigkeiten von Entwicklungstools in den Entscheidungsprozess eingebunden werden.

Elemente der Steuerungshardware

Die Steuerungshardware kann grob in 3 Elemente unterteilt werden:

- Prozessor
- Speicher
- Peripherie

Die folgende Abbildung gibt einen Überblick über die Elemente der Steuerungshardware und deren Interaktion mit der Bewegungseinheit selbst.

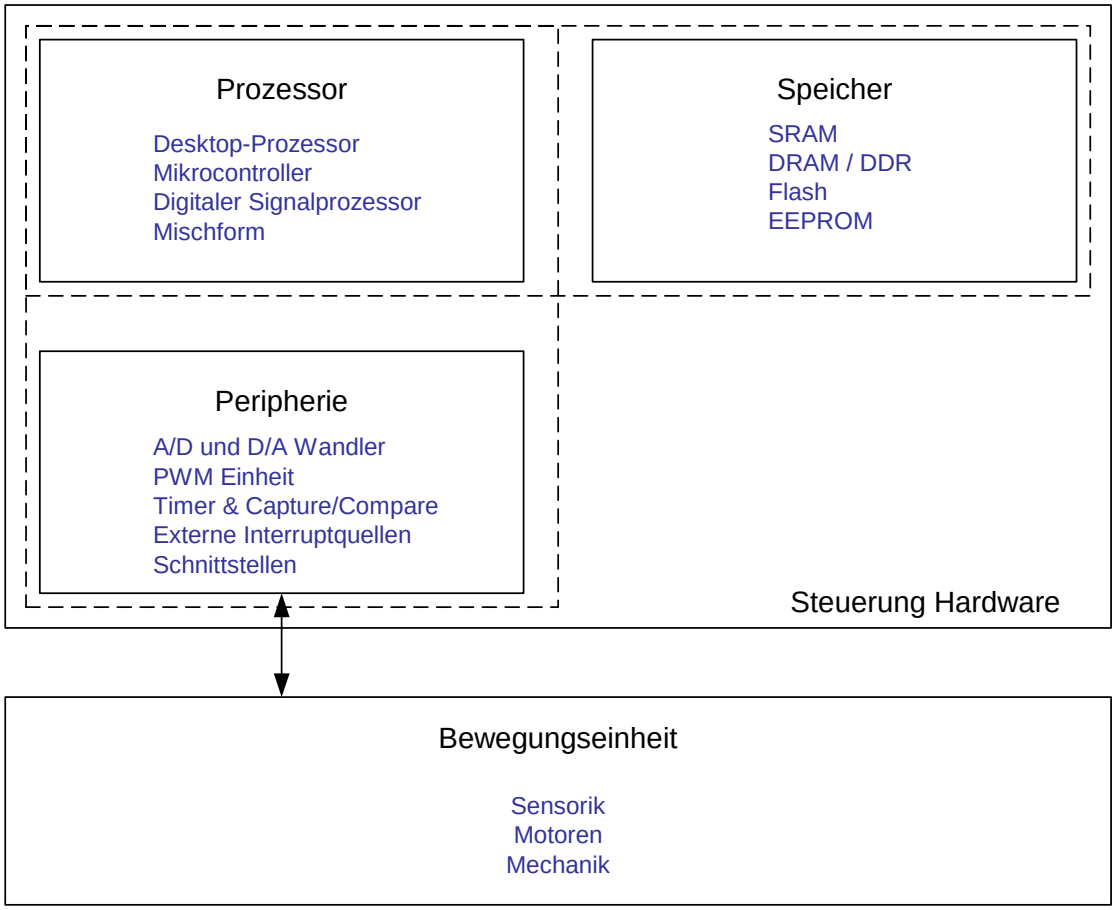

Abbildung 3.2: Elemente der Steuerungshardware

In den folgenden Kapiteln werden die einzelnen Elemente im Detail beschrieben und unterschiedliche Typen verglichen.

Prozessoren

Die eingesetzte Prozessortechnologie ist also vor allem von den zuvor besprochenen Auswahlkriterien abhängig. Es sollen an dieser Stelle kurz die zur Zeit gängigsten Prozessorkategorien und ihre Vor- und Nachteile speziell in Hinblick auf diese Kriterien und auf den Einsatz in mobilen Robotern vorgestellt werden:

Desktop – Prozessoren

Die weitläufig bekannteste Art von Prozessoren sind Desktop-Prozessoren, wie sie in herkömmlichen Personal Computern zu finden sind. Desktop Prozessoren haben sich in den letzten Jahren aufgrund der stark wachsenden Märkte für Büro und Heim rasant weiterentwickelt, derzeit erleben wir die 8.Generation der x86 Familie von Intel ™ bzw. AMD

™. Neben der Intel-Linie existieren noch weitere Prozessorfamilien in dieser Kategorie, beispielsweise Power PC ™ von IBM und den Transmeta Crusoe ™.

Diese Prozessoren sind mit Taktraten bis 2,5 GHz und 64 bzw. 32 Bit Bussen, bis zu 512 Kbyte internem Cache äußerst leistungsfähig, jedoch dementsprechend stromhungrig: Ein einzelner Prozessor kann bis zu 70 Watt verbrauchen. Ein großer Kühlkörper ist unabdingbar, zumeist ist auch ein Kühlungsventilator erforderlich.

Aus diesem Grund eignen sich Desktop Prozessoren nur bedingt für mobile Roboter. Wird tatsächlich so viel Rechenleistung benötigt, dann sollte man zumindest die Mobil-Derivate dieser Prozessoren in Erwägung ziehen, welche einige Stromsparmöglichkeiten wie z. B. Speed Step und mehrere Sleep-Modi enthalten.

Einige Prozessortypen sind auch Multiprozessorfähig (allerdings i.a. nicht die Mobil-Varianten), die Transmeta Crusoe™ Prozessoren sind ein Paradebeispiel für Multiprozessorsysteme. Die Skalierbarkeit dieser Prozessoren ist demnach recht gut.

Die Verfügbarkeit von Entwicklungswerkzeugen ist aufgrund des starken Verbreitungsgrades im PC-Segment vergleichsweise hoch. Keine andere Prozessorkategorie bietet derart viele Compiler und andere Werkzeuge oder fertige Software-Komponenten.

Ist die Leistungsfähigkeit eines Desktopprozessors für die Roboter-Steuerung erforderlich, so empfiehlt es sich, ein fertiges, handelsübliches Mainboard – am besten eine Industrie-PC Variante - zu verwenden. Dies verkürzt die Entwicklungszeit erheblich und ist zudem wesentlich günstiger als eine eigene Entwicklung.

Vielfach existieren von Desktopprozessoren auch „Embedded“-Varianten, also für den Industrie- und Automotivebereich. Beispielsweise sei hier die Motorola MP500-Familie genannt, welche Power-PC-Varianten mit verschiedener integrierter Peripherie anbietet. Diese Prozessoren sind hinsichtlich Stromverbrauch und daher auch Kühlung optimiert und laufen mit wesentlich geringeren Taktraten, sodaß keine aktive Kühlung mittels Lüfter erforderlich ist.

Digitale Signalprozessoren

Hochspezialisiert auf parallele Datenverarbeitung, Signalverarbeitung und Anwendung unterschiedlichster digitaler Filter sind DSPs (Digitale Signalprozessoren). Bei weitaus geringeren Taktraten und wesentlich weniger Stromverbrauch bieten sie zwangsläufig auch weniger Rechenleistung. Bei geschicktem Ausnützen des Befehlssatzes kann die Leistung für bestimmte Aufgaben an jene von Desktopprozessoren herankommen.

Eine Besonderheit moderner DSPs ist die Fähigkeit der Fließkomma- oder Festkomma-Arithmetik. Allerdings gilt auch hier: der Befehlssatz eines ins Auge gefassten Prozessors sollte genau geprüft werden, oftmals gibt es lediglich die rudimentärsten Befehle, Multiplikation und Division sind nicht als Befehle verfügbar und können nur durch Software-Algorithmen nachgebildet werden.

Etliche DSPs haben auch RAM- oder Flash-Speicher und verschiedene Schnittstellen integriert. Falls es sich um Dual-Port-RAM handelt, ist auch eine gewisse Multiprozessorfähigkeit gegeben. Ansonsten können mehrere Prozessoren lediglich über Schnittstellen kommunizieren.

Mikrocontroller

Die am meisten verbreiteten Prozessoren sind jedoch Mikrocontroller: Sie werden in der Automobiltechnik, Steuerungs- und Regelungstechnik, in Computerperipherie, in Haushaltsgeräten, etc. eingesetzt. Es handelt sich um sogenannte EinChip-Computer, da sie oft gleich Programm- und Datenspeicher sowie unterschiedlichste Peripherie mitbringen und im Betrieb nicht mehr allzu viel an externer Hardware benötigen.

Der Begriff des Mikrocontrollers umspannt allerdings ein recht weitreichendes Spektrum von Prozessoren: vom einfachen 4 Bit–Controller mit 1 Kbyte internem Arbeitsspeicher bis zum hochintegrierten 32 Bit Baustein mit nur aller erdenklicher Peripherie. Eine dezitierte Aussage zu Leistungsfähigkeit und Eignung für Aufgaben ist daher nicht möglich.

Die Brauchbarkeit eines Mikrocontrollers definiert sich am ehesten über die verfügbaren Entwicklungswerkzeuge, da die technische Leistungsfähigkeit innerhalb der einzelnen Klassen (4 Bit, 8 Bit, 16 Bit, 32 Bit) ohnehin recht knapp beieinanderliegt.

Mischformen: Embedded & Co.

In den letzten Jahren zeichnet sich allerdings immer mehr der Trend zu Kombinationen aus all den genannten Prozessorklassen ab: universelle, hochintegrierte Prozessorbausteine, welche RAM und Flash-Speicher, unterschiedlichste Schnittstellen, Peripherie wie Timer und A/D-Wandler und vieles mehr integriert haben. Es handelt sich um moderne RISC-Architekturen mit mehrstufigen Pipelines, mit zusätzlichen DSP-Befehlen. Da diese Universal-Bausteine vorrangig für den „Mobility“-Markt erzeugt werden, sind sie dementsprechend auf geringen Stromverbrauch optimiert - inklusive mehrerer Stromsparmodi.
Leider sind die Prozessoren in puncto Skalierbarkeit nicht mit Desktopprozessoren zu vergleichen, da sie eben für Single-Chip Applikationen konzipiert wurden. Mit den entsprechend schnellen Schnittstellen kann trotzdem eine gewisse Erweiterbarbeit erreicht werden.

Bleibt noch die Frage nach den Entwicklungstools: Da der Markt für diese eierlegenden Wollmilchsäue noch relativ jung ist, ist die Anzahl an verfügbaren Tools recht begrenzt. Wächst der „Mobility“-Markt weiterhin so stark wie bisher, so werden Qualität und Quantität der Entwicklungswerkzeuge sicher noch verbessert.

Kriterium	*Desktop Prozessoren*	*Mikrocontroller*	*Digitale Signal-Prozessoren*	*Mischformen*
Leistungsfähigkeit	Hoch	Niedrig-Mittel	Mittel-Hoch	Mittel-

				Hoch
Befehlssatz	Komplex Fließkommabefehle Multimediabefehle	Einfach	Hochspezialisiert, z.T. Fließkomma, Multimediabefehle	Einfach, z.T. mit
Ein/Ausgabe	wenig Möglichkeiten	Digital E/A Analog E/A Standardschnittstellen (asynchron und synchron seriell) Umfangreiche Peripherie möglich	wenig, digitale E/A Standardschnittstellen	hoch
Skalierbarkeit	niedrig	niedrig-mittel	niedrig	mittel
Stromverbrauch	hoch	niedrig-mittel	mittel	mittel
Software-Entwicklungsumgebungen	Große Auswahl, mächtige Entwicklungstools, EDE's und Compiler	zumeist Assembler oder C, Compiler nicht so mächtig wie von Desktopprozessoren	zumeist Assembler, C Compiler nicht so mächtig wie von Desktopprozessoren	Assembler, C, C++
Betriebssysteme	Verschiedene hochentwickelte OS möglich, Ressourcen	Real-Time Operating Systems möglich	i.A. keine Betriebssysteme verfügbar	RTOS verfügbar
Softwarekomplexität	Hoch, OOP möglich	Niedrig-Mittel, z.T. OOP möglich	Niedrig	Mittel, OOP möglich

Tabelle 3.4: Vergleich der unterschiedlichen Prozessortypen

Speicher

Wurde der Prozessor aufgrund der genannten Kriterien gewählt, bleibt noch die Frage nach dem externen Speicher, falls der vorhandene prozessorinterne Speicher nicht ausreichen sollte.

Grundsätzlich wird Arbeitsspeicher für 2 Zwecke benötigt:

- Datenspeicher und
- Programmspeicher

Je nach Prozessor-Architektur kann es sich um zwei logisch getrennte Speicherbereiche (von Neumann – Architektur) oder um einen unifizierten Adressraum (Harvard – Architektur) handeln. Beispielsweise ist für die bekannte 8-Bit Microcontrollerfamilie 80x51 Programmspeicher und Datenspeicher getrennt, wogegen bei den meisten moderneren

Prozessor-Designs Programm- und Datenspeicher vereinigt sind. Ein Vorteil der Harvard-Architektur ist, dass die Speicheraufteilung wesentlich flexibler ist.

Grundsätzlich werden an den Programmspeicher ohnehin andere Anforderungen gestellt: Er muss im allgemeinen nicht-flüchtig sein, d.h. auch nach Spannungsverlust seinen Inhalt behalten. Der Roboter soll ja beim Einschalten bereits sein Programm haben und nicht jedesmal mit seinem Programm neu beladen werden. Daher ist zumindest ein Teil des Programmspeichers Flash-Speicher: nichtflüchtig (non-volatile) und trotzdem jederzeit neu zu programmieren.

Als Datenspeicher kommt dagegen RAM in Frage: flüchtiger Speicher, welcher rasch und einfach sowohl lesbar als auch beschreibbar ist. Hier gibt es mehrere Typen: SRAM, DRAM und SDRAM.

Vielfach ist dieser durch das Speicherbus-Interface des Prozessors vorgegeben: „einfache“ Prozessoren wie 8 Bit- oder 16 Bit - Microcontroller unterstützen im allgemeinen lediglich Statisches RAM (SRAM) und Flash, wogegen die meisten 32 Bit Prozessoren auch DRAM ansteuern können. Desktop-Prozessoren stellen in vielerlei Hinsicht eine Ausnahme dar: sie können überhaupt nur in Verbindung mit einem zusätzlichen externen Chipsatz externen Speicher ansprechen – zur Zeit ist für Desktop-Prozessoren SDRAM oder DDR-SDRAM der vorgesehene Speicher.

Statisches RAM

Das SRAM oder Statische RAM ist die älteste, gewissermaßen einfachste und gleichzeitig teuerste RAM-Art. Mit SRAM-Technologie werden auch prozessorinterne und externe schnelle Cache-Speicher sowie Video-Speicher (VRAM) realisiert, so gesehen können sie auch als die schnellste Speicherart gesehen werden.
SRAMs werden über Adress-, Daten- und Steuerbus schreibend oder lesend angesprochen und behalten bei ständiger Stromzufuhr ihren Inhalt, bis sie erneut beschrieben werden.
Modernere SRAMs bieten auch Stromsparmodi.

Dynamisches RAM

Im Vergleich zum SRAM ist das DRAM vergleichsweise günstig. Diese Technologie ist allerdings nicht für schnelle Speicher geeignet, außerdem haben sie einen gravierenden Nachteil: Sie vergessen mit der Zeit ihren Inhalt, auch wenn sie permanent mit Spannung versorgt sind. Daher muss regelmäßig (im ms-Bereich) für einen sogenannten „Refresh“ gesorgt werden, also das Auffrischen der bestehenden Daten.
Ein einfacherer Prozessor wird daher alle Hände voll zu tun haben, wenn er andauernd dafür sorgen muss, die Daten des Speichers zu erneuern. Daher sind in vielen 32 Bit Controllern sogenannte DRAM-Controller bereits integriert, welche den Refresh automatisch und unabhängig vom Prozessor durchführen. Dass der Refresh das gesamte System nicht gerade schneller macht, kann man sich vorstellen.

Grundsätzlich gilt: Nur für Prozessoren, die für den Einsatz von DRAM vorgesehen sind, macht dieser auch Sinn. Der Einsatz von Dynamischem RAM ist dann vorteilhaft, wenn besonders viel Speicher benötigt wird und die Geschwindigkeit nicht im Vordergrund steht.

SDRAM

Eine wesentlich schnellere und trotzdem recht günstige Variante des Dynamischen RAMs ist das SDRAM. Dieser Speichertyp kommt vor allem in Desktop-Prozessorsystemen zum Einsatz. Auch hier gilt: nur wenn der Prozessor bzw. der erforderliche Chipsatz diesen Speicher unterstützt, macht ein Einsatz Sinn.

Flash

Im Gegensatz zu den RAM-Typen ist Flashspeicher nicht-flüchtig, d.h. er behält auch nach Stromausfall seinen Inhalt. Für den Programmspeicher ist Flash bzw. sein Verwandter EEPROM heute eigentlich die einzige Lösung. Ältere nichtflüchtige Speicher wie EPROMs sind aufgrund der schlechten Handhabung praktisch ausgestorben. Flashes haben in den letzten Jahren einen bedeutenden Aufschwung erlebt und sind in sehr vielen Systemen zu finden: als BIOS-Speicher in Desktop-PCs, in Geräten der Unterhaltungselektronik, in Mobilfunktelefonen, in Handhalds und in praktisch allen updatefähigen Geräten. Ausserdem setzen sich Flash-Speicher als universelle austauschbare Medien für MP3-Player, Digitalkameras, Notebooks immer mehr durch: Compact Flash™ Karten, Smartmedia Card™, Multimedia Card™ oder Memory Stick ™.

Der Einsatz als Programmspeicher ist vergleichsweise einfach: Der Flashbaustein ist in mehrere Segmente unterteilt, die mittels bestimmter Daten- und Adresssequenzen einzeln gelöscht und wieder neu beschrieben werden können. So muss nicht immer das gesammte Programm erneuert werden. Es macht Sinn, eine Art Boot-Block immer im Flash zu behalten, um bei fehlerhafter oder unterbrochener Programmierung trotzdem das System zumindest so weit wieder zum Laufen zu bringen, dass erneut programmiert werden kann. Zu beachten ist allerdings, dass ein Flashbaustein nicht direkt durch ein Programm, welches sich im gleichen Baustein befindet, gelöscht oder programmiert werden kann. Entweder befindet sich die Programmierroutine auf einem anderen Baustein, oder sie kopiert sich selbst in einen verfügbaren RAM-Bereich. Für diese Zwecke ist das Harvard-Speicherkonzept vorteilhaft.

Auch als Datenspeicher kann Flash verwendet werden: Lesezugriffe sind vergleichbar schnell wie bei SRAMs oder DRAMs, nur Schreibzugriffe sind durch das erforderliche vorangehende Löschen des ganzen Segments wesentlich langsamer. Als Datenspeicher kann Flash vor allem für Daten eingesetzt werden, die auch nach dem Ausschalten wieder verfügbar sein sollen und nicht allzu oft verändert werden.

Flashtechnologie ist eine der Schlüsseltechnologien der Zukunft, und durch die große Verbreitung und den stark wachsenden Markt sind sie heute kaum teurer als SRAMs oder DRAMs.

EEPROM

EEPROMs basieren auf der gleichen Technologie wie Flash-Speicher. Allerdings ist es möglich, einzelne Datenworte zu beschreiben, wogegen beim Flashspeicher immer ganze Blöcke ("Sektoren") auf einmal gelöscht und neu geschrieben werden.
Zumeist bieten EEPROMs kleine Speichermengen und sind über serielle Busse angebunden (asynchron, synchron oder I²C).

Die folgende Tabelle vergleicht die unterschiedlichen Speichertypen:

Kriterium	*Statisches RAM*	*Dynamisches RAM / DDR*	*Flash*	*EEPROM*
Speicherkapazität	bis 32MByte	bis einige GByte	bis zu 1 GByte	<64KByte
Einsatz	Daten- oder Programmspeicher	Daten- oder Programmspeicher	Daten- oder Programmspeicher	Datenspeicher
Anbindung	Adress/Datenbus	Adress/Datenbus	Adress/Datenbus	Serielles Interface
Flüchtig	ja	ja	nein	nein
Lesegeschwindigkeit	hoch	hoch	hoch	niedrig (bei serieller Anbindung)
Schreibgeschwindigkeit	hoch	hoch	mittel	niedrig (bei serieller Anbindung)
Nachteile	teuer, tw. hoher Stromverbrauch	Speicher-Refresh durch Prozessor erforderlich	Löschen/Schreiben nur block/sektorweise möglich	Erfordert entsprechendes serielles Interface

Tabelle 3.5: Vergleich der unterschiedlichen Speicherarten

Peripherie

Die Schnittstelle zwischen Steuerungsprozessor und Sensoren bzw. Aktoren bildet die sogenannten Peripherieeinheiten. Sie können entweder am Prozessor integriert sein oder als eigenständige Bausteine ausgeführt sein. Vor allem Mikrocontroller sind sehr gut mit Peripherieeinheiten ausgestattet, sodass externe Bausteine oft gar nicht erforderlich sind.

Folgende Peripherieeinheiten können zur Erfassung von Sensordaten oder zur Ansteuerung von Aktoren eingesetzt werden:

- Digitale Ein-, Ausgabe-Leitungen
- A/D-Wandler
- D/A-Wandler
- PWM-Einheit
- Timer
- externe Interruptquellen
- Capture-Compare-Einheiten
- asynchrone Schnittstellen
- synchrone Schnittstellen
- CAN-Bus
- sonstige Schnittstellen

Digitale Ein-/Ausgabeleitungen

Für digitale Anbindung von Sensoren, Aktoren oder andere Bausteine an den Prozessor werden digitale Ein- oder Ausgänge (I/O-Ports) benötigt. Oftmals sind diese Leitungen sehr flexibel nutzbar, sei es als einzelne Leitungen, oder als ganze Bitgruppen (4, 8, 16, 32 Bit) zur parallelen Ein-/Ausgabe. Des weiteren sind die Ports konfigurierbar, mindestens die Signalrichtung (Ein oder Ausgang) sollte einstellbar sein. Darüber hinaus sind Digitale Ein-/Ausgänge oftmals in Bitgruppen zu 4, 8 oder 16 Bit gruppiert, es ist daher bereits beim Schaltungsentwurf darauf zu achten, ob der entsprechende Baustein die bitweise Konfiguration bietet, oder ob jeweils eine ganze Anzahl an Bits gleichartig konfiguriert werden muss. Zu beachten ist auch, dass gerade bei Mikrocontrollern I/O-Ports doppelt belegt sind, vor allem im Zusammenhang mit anderen Peripherieeinheiten (z. B. Schnittstellen, A/D-Wandler-Eingänge, Counter-Eingang...). Sollen diese Peripherieeinheiten genutzt werden, so kann der entsprechende Anschluss nicht als digitale Ein-/Ausgabeleitung eingesetzt werden. Zur Bufferung der Ausgänge existieren eigene Register, welche den Ausgang so lange auf dem gesetzten Wert lassen, bis er wiederum geändert wird.

Verfügt der Prozessor selbst nicht über eigene Digitale Ein- oder Ausgänge – Desktopprozessoren verfügen im allgemeinen generell nicht über Peripherieeinheiten – so muss bei Bedarf mittels externer Bausteine dafür gesorgt werden. Zu diesem Zweck werden die Bauteile in den Adressraum des Prozessors gemappt und können nur indirekt über ihre Adresse angesprochen werden.
Werden viele Digitalleitungen benötigt und bietet der Prozessor diese Peripherie nicht, so empfiehlt sich der Einsatz von ASICs (Application Specific Integrated Circuits) oder FPGAs (Field Programmable Gate Arrays).

Digitale Ein-/Ausgabe kann zur Erzeugung von Steuerleitungen, zum Aktivieren oder Deaktivieren von Sensoren oder Aktoren, zur Generierung von proprietären, aber auch Standard-Protokollen, zum Ausgeben oder Einlesen von seriellen oder parallelen Daten und viele andere Anwendungen genutzt werden.

A/D-Wandler

Oftmals liefern Sensoren analoge Informationen – zumeist in Form eines Strom- oder Spannungsverlaufes. Analoge Informationen in der Robotik können beispielsweise Helligkeitswerte, Temperaturen, Stromverbrauch oder Betriebsspannung sein.

Analog/Digital-Wandler benötigen zumeist einen Wert in einem bestimmten Spannungsbereich. Steckt die Information des Sensors im Strom, so reicht vielfach ein Widerstand, aus der resultierenden abfallenden Spannung lässt sich der Stromwert rekonstruieren. Der Spannungsbereich ist entweder fest vorgegeben (beispielsweise 0V-1V, 0V-5V), oder es können Referenzspannungen angegeben werden. Oftmals ist der A/D-Wandler innerhalb der Referenzspannungen nochmals konfigurierbar. Enspricht der Spannungswert nicht dem Bereich des A/D-Wandlers, so wird bei zu kleinem Eingangs-Signal ein Vorverstärker benötigt, im

seltenen Fall eines zu großen Signales muss dieses mittels einfachen Spannungsteilers (2 Widerstände) reduziert werden.
Relevant für die Präzision der Aufzeichnung ist die Auflösung des A/D-Wandlers. Wenngleich 8 Bit Auflösung eine Untergrenze darstellt, sind 256 verschiedene Signalzustände für viele Aufgabenstellungen ausreichend. 10 oder 12 Bit sind bereits für Videoqualität ausreichend, 16 Bit sind gleichbedeutend mit CD-Audio-Qualität.

Die Abtastrate (Sampling Rate) des A/D-Wandlers bestimmt, wie lange dafür benötigt wird, den derzeit anliegenden Analogwert aufzuzeichnen und damit, wie oft der Wandler einen Wert liefern kann. Die Abtastrate eines A/D-Wandlers kann zumeist entweder softwaremäßig bzw. bei externen A/D-Wandler durch Hardwarebeschaltung oder durch den gelieferten Zeitrahmen eingestellt werden. Die Abtastraten gängiger ADCs liegt zwischen 10 ns und 100 us

In diesem Zusammenhang ist bei der Wandlung von analogen Signalen das Shannonsche Abtasttheorem zu beachten: Die Abtastrate muss mindestens doppelt so hoch sein wie die höchste vorkommende Frequenz des Eingangssignales. Ansonsten könnte es passieren, dass signifikante Signaländerungen zwischen 2 Messwerten nicht registriert werden. Kann diese Annahme nicht getroffen werden, so ist ein entsprechendes (Tiefpass-) Filter erforderlich, welches die Signalbandbreite begrenzt. Andernfalls muss die Abtastrate erhöht werden.

A/D-Wandler dienen vor allem dazu, analoge Sensordaten oder andere Informationen in eine geeignete digitale Form zu bringen. Relevant für Qualität und Performance eines Wandlers sind vor allem Auflösung und Abtastrate.

D/A-Wandler

Die entgegengesetzte Aufgabe erledigt der Digital/Analog-Wandler: aus bestehenden Digitalwerten soll ein Analogsignal erzeugt werden. Eingesetzt werden kann ein D/A-Wandler, um beispielsweise einen Gleichspannungsmotor mit unterschiedlichen Pegeln anzusteuern, oder um eine Lichtquelle oder Signalleuchte unterschiedlich stark anzusteuern. Weiters kann der D/A-Wandler auch zum Generieren von Audiosignalen eingesetzt werden.

Wie beim A/D-Wandler gelten hier ebenfalls Wandelzeit und Auflösung als primäre Qualitätskriterien. Dazu kommt noch die Signalqualität (SNR, Glättung).

PWM-Einheit

Ähnliche Aufgaben wie ein D/A-Wandler erledigen PWM-Einheiten: Puls-Weiten-Modulation (PWM) ist ein Verfahren zum Nachbilden von Gleichspannungssignalen. Dabei wird eine Digitalleitung periodisch (die PWM-Periode) auf logisch 1 gesetzt und nach Ablauf der Pulsweite, welche proportional zum auszugebenden Analogwert ist, auf Null gesetzt. Durch nachfolgende Integration des Signals mittels Tiefpass entsteht aus dem Digitalsignal ein Analogsignal. PWM-Einheiten stellen im wesentlichen Digital-Analog-Converter dar.

PWM-Signale werden vor allem zur Ansteuerung von Motoren, aber auch zur Generierung von Audiosignalen und allgemein zur Ansteuerung von gleichspannungsgesteuerten Komponenten

eingesetzt. Beispielsweise wurde zeitweilig in CD-Playern mit „schnellen 1-Bit Digital/Analog-Convertern“ geworben, wobei es sich in diesen Fällen um PWM-Signale handelte.

Induktive Lasten wie Lautsprecher oder Gleichspannungsmotoren stellen in Serie mit einer ohmschen Last ja einen Tiefpass dar, daher ist keine separate Tiefpassfilterung erforderlich. Zu beachten ist, dass mit einem Digitalausgang eines Prozessors oder Controllers keine Last angesteuert werden kann. Daher wird immer ein Leistungstransistor nachgeschaltet, welcher schnell genug für das PWM-Signal sein muss.

Einige Mikrocontroller haben als Peripherieeinheit PWM-Module integriert, oftmals mit mehreren voneinander unabhängigen Kanälen. Interessant ist bei PWM-Modulen vor allem die Zeitauflösung für die Pulsweite. Sie bestimmt im Endeffekt die Auflösung des mittels PWM-Einheit nachgebildeten Digital/Analog-Converters.

Timer

Kaum einem Controller fehlen periphäre Timer-Einheiten. Mit Hardware-Timern können periodische Signale, Interrupts oder Timeouts erzeugt oder generell periodische Vorgänge initiiert werden. Timer sind beispielsweise auch Basis für jedes Echzeitbetriebssystem mit zeitbasiertem Scheduling.

Auch zum Messen von externen Signalen eignen sich Timer hervorragend, beispielsweise um festzustellen, wann ein Eingang einen bestimmten Wert erreicht hat. In Verbindung mit Zeitmessungen sei hier auch auf die dafür prädestinierten Capture/Compare-Einheiten hingewiesen (siehe nächstes Kapitel). Weiters können im „Counter“ Mode externe Signale (beispielsweise Impulse) gezählt werden.

Oftmals sind mehrere einzelne und grundsätzlich voneinander unabhängige Hardwaretimer in Timer-Gruppen (Timer Units) zusammengefasst. Timer innerhalb einer Gruppe lassen sich dann auch verknüpfen bzw. kaskadieren. Die Serienschaltung (Kaskadierung) von Timern dient vorrangig dazu, besonders lange Timerperioden zu erzeugen, beispielsweise zur Generierung von Timeouts im Sekundenbereich.

Hardwaretimer bieten umfangreiche Einstellmöglichkeiten und Modi: Neben der Zählerperiode (also der Dauer von der Aktivierung bis zum Überlauf) ist die Einstellung der Zählrichtung, automatisches Nachladen, Ansteuerung einer Digitalleitung bei Überlauf (z. B. Toggle) und Aktivierung in Abhängigkeit von einer Digitalleitung möglich. Zudem kann bei Überlauf des Timers ein Interrupt ausgelöst werden.

Der Zähltakt sollte für jeden Timer separat einstellbar sein. Grundlage für den Zähltakt ist der CPU-Takt oder ein Teiler dessen. Dieser Grundtakt kann dann durch einen einstellbaren Vorteiler (Prescaler) reduziert werden. Die Reduktion des Grundtaktes ist für längere Timerperioden (z. B. Timeouts) erforderlich.

Neben dem Zählertakt ist auch die „Breite“ des Zählers eine wesentliche Größe, welche die maximal mögliche Zählerperiode beeinflusst. 16 Bit breite Timer haben bei gleichem Zähltakt eine 256fach größere Zählerperiode als 8 Bit-Timer. 32 Bit Timer entsprechend eine 65536fach größere Periode gegenüber 16 Bit Zählern. Allgemein üblich sind 16 Bit Zähler mit der Option der Kaskadierung – zur Nachbildung von 32 Bit Zählern durch Serienschaltung von 2 16 Bit Zählern.

Neben den Timermodi werden vielfach auch Countermodi geboten: dabei ist die Basis zum Zählen nicht der interne Takt, sondern ein Signalwechsel an einer externen Digitalleitung. Gute Timereinheiten lassen sich auch hierbei konfigurieren: Zählen bei Signalwechsel von 0 auf 1 (positiver Flanke), bei negativer Flanke oder bei beiden Arten Signalwechsel.

Capture/Compare-Einheiten

Zum Messen des Zeitverhaltens von Signalen an Digitalleitungen dienen Capture-Einheiten. Dabei wird der Zeitpunkt eines externen Ereignisses (i.a. Signalwechsel) festgehalten: Der Wert eines permanent laufenden Grundzählers wird beim Auftreten des Signalwechsels in ein Register geschrieben. Bei Bedarf wird aufgrund des Ereignisses auch ein Interrupt ausgelöst. Die Art des Ereignisses ist konfigurierbar: so kann bei steigender Flanke, bei fallender Flanke oder bei beiden Flanken der Zählerwert gespeichert werden.
Grundsätzlich könnte diese Funktion auch mittels Software auf Basis eines externem Interrupts und eines permanent laufenden Hardwaretimers realisiert werden. Durch die Realisierung in Hardware als Capture-Einheit ist jedoch die Genauigkeit der Messung wesentlich höher als bei Software, da hier die Abarbeitung des Interrupts eine gewisse Verzögerung mit sich bringt.

Im Gegensatz zum messenden Capture-Modus dient der Compare-Modus zur Erzeugung von Signalen. Dazu wird ebenfalls ein Zähler gestartet und ein Schwellwert angegeben. Erreicht der Zähler den Schwellwert, so wird am entsprechenden Pin ein Signal erzeugt und bei Bedarf ein Interrupt ausgelöst. Hier gibt es noch eine Reihe zusätzlicher Einstellungsmöglichkeiten: So kann auch der Schwellwert noch während der Periode nachgestellt werden (und so mehrere Impulse pro Periode erzeugen), oder der Pin beim Zählerüberlauf wieder rückgesetzt werden (dies wird z. B. für asymmetrisches PWM-Signal eingesetzt).

Externe Interruptquellen

Zur Anbindung weiterer externer Peripherie sollte der Prozessor mehrere externe Interruptquellen unabhängig voneinander bedienen können. Da externe Interruptquellen unterschiedlichste Interruptsignale generieren können (flanken- oder pegelaktiv, steigende oder fallende Flanke, low active oder high active), ist eine flexible Einstellbarkeit der einzelnen externen Interruptquellen erforderlich.

Relevant für externe Interruptverarbeitung ist auch die interne Verarbeitungszeit und damit die Reaktionszeit (Response Time) vom Hardwaresignal bis zum Verzweigen in den entsprechenden Softwareteil eine wesentliche Größe und sollte nicht zu groß sein (max. 7-8 Befehlszyklen).

Asynchrone Schnittstellen - UART

Zur Verbindung mit der Außenwelt des Roboters beispielsweise zur Diagnose, Firmwareupdate oder Servicetätigkeiten sind „verdrahtete“ Schnittstellen erforderlich. Auch im Automotive und Industriebereich sind diese Schnittstellen Grundlage jediglicher Kommunikation mit der Außenwelt. Daher existieren in vielen Controllern eigene Peripherie-Einheiten für asynchrone Schnittstellen (UART – universal asynchronous receiver transmitter).

Asynchrone Schnittstellen wie RS232, RS422, RS485 haben 2 Signalleitungen, jeweils eine für Senden und Empfangen, die Daten werden ohne begleiteten Takt und ohne zeitlichen Zusammenhang gesendet. Der Beginn einer Übertragung wird durch das sogenannte Startbit gekennzeichnet, welches immer den gleichen Zustand (Pegel) hat. Die Übertragung eines Datenwortes wird durch das Stopbit abgeschlossen, welches ebenfalls immer den gleichen Pegel hat: den inversen des Startbits.

Aufgrund der Vielfalt an existierenden Schnittstellen und vor allem ihrer spezifizierten variierenden Eigenschaften wie Baudrate, Bitbreite, Parity und Startbitanzahl sollte die UART-Einheit des Controllers dementsprechend viele Einstellungsmöglichkeiten bieten. Gleichzeitiges Senden und Empfangen („Full Duplex“) sollte möglich sein.

Vor allem die Einstellung von Standardbaudraten wie 9600, 19200, 38400, 57600 und 115200 Baud sollte möglich sein, um mit anderen Systemen wie beispielsweise einem PC kommunizieren zu können. Da der Baudratengenerator des UARTs und die einstellbaren Baudraten vom Prozessor-Grundtakt (CPU-Takt) abhängig ist, sollte bei der Auswahl des Prozessortaktes die gewünschten Baudraten aller vorgesehenen asynchronen Schnittstellen berücksichtigt werden.

Synchrone Schnittstellen

Speziell für schnelle serielle Kommunikation mit anderen Peripherieeinheiten wie Displays oder anderen Prozessoren sind synchrone Schnittstellen, vielfach auch SPI – Serial Peripheral Interface – genannt, gedacht. Bei diesen Schnittstellen wird im Gegensatz zu den asynchronen Schnittstellen auf einer eigenen Leitung ein Taktsignal zum Datensignal mitgeliefert, welcher angibt, wann die Daten auf der Datenleitung gültig sind.

Hier gilt wiederum: Es existieren eine Vielzahl an Einstellungsmöglichkeiten, von der Baudrate bis zur Anzahl der übertragenen Datenbits pro Wort. Auch das Taktsignal kann konfiguriert werden: sind die Daten bei fallender Flanke, bei steigender Flanke gültig, wie ist die Phase zwischen Daten- und Taktsignal.

Im Gegensatz zu asynchronen Schnittstellen gibt es bei synchronen Schnittstellen keine Standardbaudraten. Aufgrund der Synchronität der Daten zum mitgelieferten Taktsignal ist dies auch nicht notwendig. Daher müssen diese Schnittstellen bei der Dimensionierung des Prozessortaktes nicht berücksichtigt werden. Die einzige Einschränkung bei synchronen Schnittstellen ist die Maximal-Baudrate der beiden Kommunikationspartner.

CAN-Bus

Sowohl synchrone als auch asynchrone serielle Schnittstellen sind nur für Punkt-zu-Punkt-Verbindungen, also zum Verbinden von 2 Einheiten – gedacht. Um eine größere Anzahl von Gesprächspartnern zu erreichen, ist ein Bussystem erforderlich.

Bei mobilen Robotern ist damit lediglich die (drahtgebundene) Kommunikation innerhalb des Roboters gemeint, wenn es also wie beispielsweise in Automobilen mehrere dezentrale Recheneinheiten gibt, die Daten austauschen müssen. Daher sind für mobile Roboter vor allem Bussysteme aus dem Automotive-Bereich interessant.

Als Standardbus in der Automobilindustrie hat sich der CAN-Bus (Controller Area Network) durchgesetzt, welcher durchaus auch in anderen Anwendungsgebieten zum Einsatz kommt. Der CAN Bus ist ein robuster 2-Drahtbus mit Übertragungsraten bis zu 1 Mbit/s. Die Datenübertragung erfolgt asynchron, mit jeder Übertragung wird auch eine Checksumme zur Datensicherung mitübertragen. In einem Datenpaket können maximal 8 Bytes übetragen werden.

CAN-Messages werden nicht im Sinne von Station „A sendet Nachricht zu Station B“ adressiert, sondern enthalten eine Message ID, und jede Station am Bus kann bestimmen, ob sie die Information benötigt oder nicht. Die Priorisierung der Nachricht erfolgt auf Basis dieser Message ID: Ein CAN-Paket mit höherer Message ID setzt sich bei gleichzeitig gestarteter Übertragung gegenüber einer Nachricht mit kleinerer Message ID durch, eine bereits laufende Kommunikation darf aber nicht unterbrochen werden.

Ein Nachteil des CAN-Busses ist die Tatsache, dass es nur eine rudimentäre Verbindungssicherung gibt: Sobald ein Bus-Teilnehmer die Nachricht korrekt empfangen hat, wird der Empfang noch im gleichen Atemzug bestätigt. Ein Sender weiss aber nie, ob seine Nachricht auch wirklich bei allen Empfängern angekommen ist.

Die Integration der CAN-Bus Peripherie in einem Controller kann unterschiedlich ausfallen. Das Spektrum an existierenden CAN-Bus Peripherieeinheiten reicht von einem einfachen Buffer für eine einlangende Nachricht und eine ausgehende Nachricht über mehrere flexibel konfigurierbare Buffer bis hin zur Integration höherer Transportschichten.

Sonstige Schnittstellen

Neben synchronen, asynchronen seriellen Schnittstellen und dem CAN-Bus gibt es eine Reihe von weiteren, verbreiteten Schnittstellen, beispielsweise der J1850 (das Pedant zum europäischen CAN-Bus in den USA), I²C Bus oder der Profibus (Feldbussysteme der Mess- und Automatisierungstechik).

Neben der Anbindung von Peripherie an klassische serielle Schnittstellen oder Bussysteme ist natürlich jene an den Adress- und Datenbus erwähnenswert, bei der Peripheriebausteine wie externe Speicherbausteine an den Prozessor gekoppelt sind. Diese Art der Kommunikation ist

natürlich gegenüber seriellen Verbindungen die schnellere, da erstens parallel 8, 16 oder 32 Bits Daten übertragen werden und zweitens der Datenzugriff direkt über Speicheradressen erfolgen kann („memory mapped peripherals“). Aufgrund steigender Funktionalität und Performance speziell der synchronen seriellen Schnittstellen in Form von SPI (Serial Port Interface) sind parallele Schnittstellen jedoch aufgrund der größeren Leitungszahl und damit höheren Produktionskosten und höherer Fehleranfälligkeit immer seltener anzutreffen.

Sollte Peripherie eingesetzt werden, welche eine direkte Anbindung an Adress- und Datenbus erforderlich machen, so muss darauf geachtet werden, dass der Controller bzw. Prozessor die Anbindung umsetzen kann. Damit ist vor allem eine eigene Chip-Select-Leitung, variables Timing von Read- und Write-Signal sowie konfigurierbare Wait States gemeint. Diese Fähigkeiten sind auch für den Einsatz unterschiedlicher externer Speicher erforderlich.

Die folgende Tabelle gibt einen Überblick über die Anwendungsmöglichkeiten der unterschiedlichen Peripherietypen:

Peripherietyp	Anwendung
A/D-Wandler	Einlesen analoger Signale, beispielsweise von Sensoren, die analoge Ergebnisse liefern. Beispielsweise Spannungs- Strom- oder Temperaturmessung
D/A-Wandler	Ansteuerung von Gleichspannungsmotoren, Lichtquellen, Erzeugung von Audiosignalen
PWM-Einheit	Ansteuerung von Gleichspannungsmotoren, Lichtquellen, Erzeugung von Audiosignalen
Timer	Erzeugung periodischer Signale, Interrupts Überwachung von Timeouts Basis für zeitbasierte RTOS-Scheduler Ausmessen von digitalen Signalen
Externe Interruptquellen	Signalisierung externer Peripherie, beispielsweise Schnittstellencontroller, A/D Wandler oder andere Prozessoren, FPGAs
Capture/Compare-Einheiten	Capture: Exakte Zeitmessung von digitalen Signalen (z. B. PWM-Signale) Compare: Exakte Erzeugung von digitalen Signalen (z. B. PWM)
Asynchrone Schnittstellen	Kommunikation mit Host-PC oder Peripherieeinheiten wie beispielsweise einem GPS-Empfänger
Synchrone Schnittstellen	Kommunikation mit anderen Robotereinheiten, E²PROMs, FPGAs oder Displays
CAN-Bus	Universelles, standardisiertes Bussystem für Kommunikation der verschiedenen Robotereinheiten untereinander
Sonstige Schnittstellen	z. B. I²C Bus für serielle Kommunikation mit Displays, E²PROMs...

Tabelle 3.6: Anwendungsgebiete von unterschiedlichen Peripherietypen

4 Systemdesign und Realisierung

Die Realisierung mobiler Roboter bedingt den Einsatz und die Kombination unterschiedlichster Technologien und Wissenschaftsgebiete. Umso größer ist die Herausforderung, einen mobilen Roboter zu konstruieren und zu entwickeln.

Im Rahmen dieser Arbeit wurde der mobile Roboter Tinyphoon realisiert. Hauptaugenmerk lag dabei auf Entwicklung der Elektronik und der Software der modularen Bewegungseinheit des Roboters. Die mechanische Konstruktion, Dimensionen und sein Anforderungsprofil waren bereits vorgegeben.

In diesem Teil der Arbeit wird näher auf die zuvor gestellten Anforderungen (Pflichtenheft), auf die Realisierung – Mechanik, Hardware und Software – sowie auf mögliches Verbesserungspotential eingegangen.

Der in weiterer Folge beschriebene Roboter Tinyphoon wurde unter anderem auch in Hinblick auf einem Einsatz als Fußballroboter der Kategorie MiroSOT entwickelt. MiroSOT ist eine bestimmte Kategorie des Roboterfußballs, welche von der FIRA organisiert wird. Die *FIRA* (Federation of International Robot-Soccer Association) bietet eine Vielzahl von Spielklassen an [JongHwan98], von reinen Simulationen, die ohne jegliche Roboter-Hardware auskommen und Wettbewerbe im Speicher des Computers austragen, bis hin zu humanoiden Robotern, die völlig ohne Zentralrechner agieren.

Diese Arbeit beschäftigt sich mit Robotern der Spielklasse „MiroSOT“ - darauf beziehen sich die folgenden Erklärungen:

Die Hardware eines Spielers, ein würfelförmiger - jede Kantenlänge beträgt 75mm - Roboter, lässt sich in folgende drei Teile zerlegen:

- Bewegungseinheit,
- Steuereinheit und
- Stromversorgung.

Die Bewegungseinheit besteht aus zwei Rädern, zwei Gleichstromantrieben, deren Regelung und Leistungselektronik. Die Antriebsleistung eines Motors liegt bei ca. 4.5 Watt. Die elektronische Steuereinheit wiederum ist optional erweiterbar und besteht aus dem Kommunikationsmodul (zur Kommunikation mit einem Hostcomputer), Sensoren und dem Regelungsmodul.

Als Hostrechner dient ein handelsüblicher PC. Dieser hat die Aufgabe der Bildverarbeitung, der Kommunikation und der Planung der nächsten Aktionen seiner Mannschaft, sowie die Steuerung der Spieler. Zum Bildverarbeitungssystem gehört eine Farb-CCD-Kamera und eine „frame grabber“ Karte [Biribauer2002].

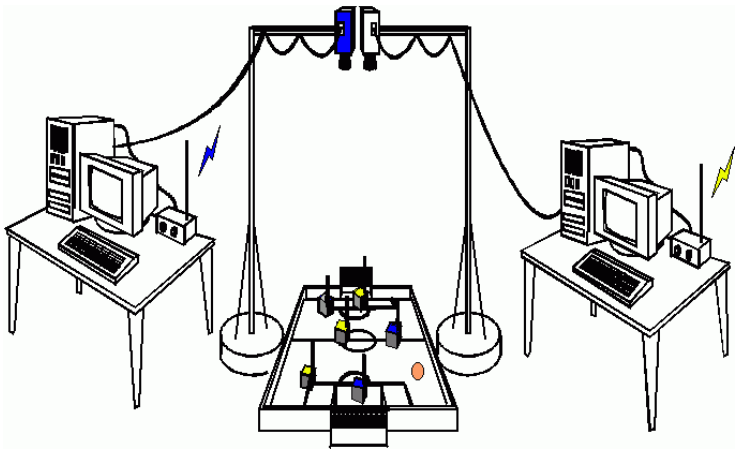

Abbildung 4.1: Roboterfussball als Spielwiese der Wissenschaften

Roboterfußball bietet die Möglichkeit, einen breiten Publikum das komplexe interdisziplinäre Forschungsgebiet Mechatronik mit den Teilgebieten: Maschinenbau, Elektrotechnik, Regelungstechnik, Informatik und Nachrichtentechnik spielerisch näher zu bringen.

Von der wissenschaftlichen Seite handelt es sich bei Roboterfußball um ein autonomes Multi-Agenten-System. Die einzelnen „Spieler“ müssen hierbei gemeinsam ein konkretes Ziel verfolgen, da das Spiel in einem höchst dynamischen Umfeld stattfindet. Neben der ungenauen Information über den Spielbereich sind es doch die Roboter selbst, die ständig und auch „vorsätzlich“ die Umgebung in einer unvorhersehbaren Art und Weise beeinflussen. Die einzelnen Roboter müssen deshalb, neben der Kontrolle über das eigene Verhalten, auch die Übersicht über andere Roboter und natürlich auch über den Ball behalten.

Hinter der spielerischen Anwendung von Multi-Agenten-Systemen in Form von Roboterfußball stehen bereits reale technische Anwendungen. So können die gewonnenen Erkenntnisse beispielsweise in der Produktion dazu dienen, die heute verwendeten fahrerlosen Transportsysteme zu ersetzen. Transportsysteme der Zukunft, bestehend aus autonomen Agenten, können sich nicht nur frei bewegen sondern sich auch die Aufgaben untereinander in optimaler Weise aufteilen.

4.1 Plattformentscheidung

Erfüllung der allgemeinen Anforderungen

Aus den allgemeinen Anforderungen an Tinyphoon ergeben sich folgende Robotereigenschaften:

Modulare Architektur: Ein performantes Kommunikationssystem (z. B. Bus) ist erforderlich, um einen Informationsaustausch der unterschiedlichen Robotermodule (auch zukünftiger) zu ermöglichen. Die Software muss entsprechend modular aufgebaut sein.
Geschwindigkeit & Beschleunigung: Da hohe Anforderungen an Geschwindigkeit und Beschleunigung gestellt werden, soll der Roboter durch 2 Räder angetrieben werden. Um eine optimale Beschleunigsrate zu erreichen, soll in der Robotersoftware eine Anti-Schlupfregelung konzipiert werden, die ein Durchrutschen der Räder frühzeitig erkennt und durch Gegenmaßnahmen verhindert.
Bewegungspräzision: Die Bewegungspräzision wird vor allem durch die Implementierung des Reglers bestimmt. Die Aufnahme der Raddrehzahlen soll durch hochpräzise Inkrementalgeber bewerkstelligt werden.
Laufzeit: Um eine hohe Laufzeit des Roboters zu gewährleisten, ist einerseits moderne Akkutechnologie (Lithium Ionen) und andererseits der Stromverbrauch der einzelnen Komponenten zu berücksichtigen. Die Motoren sind dabei der größte Stromverbraucher, aber auch bei Prozessor, Speicher und anderen Elementen ist auf geringen Stromverbrauch zu achten.
Roboterspezifische Eigenschaften: Diese Eigenschaften werden vor allem durch den mechanischen Aufbau umgesetzt. Relevant für die Plattformentscheidung ist die vorgegebene maximale Baugröße. Die Anzahl und benötigte Platinenfläche der integrierten Halbleiterbausteine sollte daher minimiert werden.
Diagnosemöglichkeiten: Sowohl für Entwicklung als auch für den Betrieb sind unterschiedliche Diagnosemöglichkeiten vorzusehen. Unbedingt erforderlich ist eine asynchrone serielle Debugschnittstelle. Weitere Diagnosemöglichkeiten ergeben sich aus Softwareumsetzung (Logging, Error-Tracing) sowie durch die verfügbaren Entwicklungstools.

Resultierende Anforderungen an Prozessorplattform

Keine Komponente bestimmt im gleichen Maße die Möglichkeiten und Fähigkeiten des mobilen Roboters wie die zugrundeliegende Prozessorplattform. Daher resultieren viele der definierten technischen Eigenschaften in Anforderungen an den auszuwählenden Prozessor und dessen Umgebung:

- performantes Bussystem für Kommunikation mit anderen Robotereinheiten
- Signalerzeugung für 2 Gleichspannungsmotoren – üblicherweise mittels PWM
- hochpräzise Messung der Raddrehzahlen – mittels Inkrementalgeber
- geringer Stromverbrauch der Elektronikkomponenten
- Minimale Anzahl an integrierten Halbleitern und verbrauchter Chipfläche
- Diagnoseschnittstelle – asynchrone serielle Schnittstelle

- umfangreiche Toollandschaft für die Prozessorfamilie, Debug-Tools

Gewählte Plattform: xc167

Der xc167 erfüllt in einem sehr guten Maße alle technischen Anforderungen. Die einzelnen Anforderungen werden folgendermaßen erfüllt:

Anforderung	*Realisierung*
Kommunikation zu anderen Einheiten	Die Kommunikation zu anderen Robotereinheiten erfolgt über CAN-Bus. 2 unabhängige Bussysteme sind möglich.
Signalerzeugung für Gleichspannungsmotoren	4 PWM Signale (2 pro Motor und Drehrichtung) werden mittels Peripherieeinheit umgesetzt. Der xc167 kann bis zu 20 PWM Signale erzeugen.
Messung der Raddrehzahlen	Der integrierte Inkrementalsignal-Decoder ermöglicht sowohl die Messung der Raddrehzahlen als auch die Bestimmung der Drehrichtung.
Geringer Stromverbrauch	Diese Anforderung erfüllt der xc167 im Vergleich zu anderen – ähnlich oder höher performanten Prozessoren – nur mäßig
Minimale Anzahl an integrierten Halbleitern	Durch integriertes Flash und allen erforderlichen Peripherieeinheiten OnChip ist grundsätzlich ein SingleChip Betrieb möglich. Lediglich aufgrund der komplexen Aufgaben und Bufferfunktionen ist ein externes RAM erforderlich.
Diagnoseschnittstelle	Der xc167 verfügt über 2 asynchrone serielle Schnittstelle. Eine der beiden Schnittstellen wird als reine Diagnoseschnittstelle dienen.
Umfangreiche Toollandschaft	Für die xc16x Familie existieren mehrere Compiler und etliche Debugtools unterschiedlicher Hersteller

Tabelle 4.1: Erfüllung der Anforderungen durch die gewählte Plattform

Wie der xc167 generell in Robotersystemen eingesetzt werden kann, beschreibt das nächste Kapitel.

4.2 Einsatz des xc167 in Robotersystemen

Der Mikrocontroller xc167 ist die Steuereinheit der mobilen Bewegungseinheit des Tinyphoons. Das folgende Kapitel liefert einen Überblick über die Architektur und vor allem die Peripherie des xc167. Des weiteren wird beschrieben, wie der xc167 und dessen Peripherie in Robotersystemen allgemein und bei Tinyphoon im speziellen eingesetzt werden kann.

Der 16-Bit Mikrocontroller xc167 von Infineon™ ist das Oberhaupt der Familie, nicht zuletzt aufgrund der vielfältigen Peripherie. Neben dem Architekturkonzept werden die RISC-CPU, das ausgereifte Interruptsystem, I/O-Ports und der interne Speicher beschrieben. Siehe dazu auch [Infineon2006].

xc167 – die zugrundeliegende Architektur

Die zentrale Einheit des xc167 ist der RISC-Prozessorkern (CPU). Alle Befehle werden in einer vierstufigen Pipeline durchgeführt, und fast alle Befehle benötigen nur jeweils 1 Prozessortakt zur Ausführung.

Der Baustein enthält sowohl 128 kByte internes Flash für Programmcode oder Daten als auch mehrere interne RAM-Bereiche. Die internen RAM-Bereiche sind insgesamt 8 kByte groß, wovon 6 kByte nur für Daten (Variablen, Stack) und 2 kByte sowohl für Code als auch für Daten zur Verfügung stehen.

Der Interrupt-Controller als eigene Einheit wird auch als „Betriebssystem in Silizium bezeichnet“: Insgesamt 77 Interruptquellen können beliebig auf 16 verschiedene Interruptebenen priorisiert werden. Interruptquellen sind vor allem die Peripherieeinheiten, welche bei Auftreten eines externen (z. B. Byte empfangen) oder internen Ereignisses (z. B. Timerüberlauf) einen Interrupt erzeugen.

Ein Watchdog Timer kümmert sich darum, dass der Controller richtig hochfährt. Er wird beim Start automatisch aktiviert und muss von der Software abgeschaltet oder permanent gewartet werden. Er kann auch dazu verwendet werden, um die Softwarefunktionalität zu überwachen.

Die Peripherieeinheiten können sich sehen lassen: 2 CAN-Module, A/D-Wandler, zwei Timereinheiten, zwei Capture/Compare Einheiten mit insgesamt 11 Timern, 2 asynchrone serielle Schnittstellen mit sehr flexibler Baudrateneinstellung, zwei synchrone serielle Schnittstellen (SPI), sehr flexible PWM-Einheit, I2C Bus sowie Realtime Clock.

Die über 100 I/O Ports können für allgemeine Ein-/Ausgabezwecke individuell konfiguriert, zum Zugriff auf externen Speicher oder von den Peripherieeinheiten genutzt werden.

Die folgende Abbildung zeigt das Blockschaltbild des xc167:

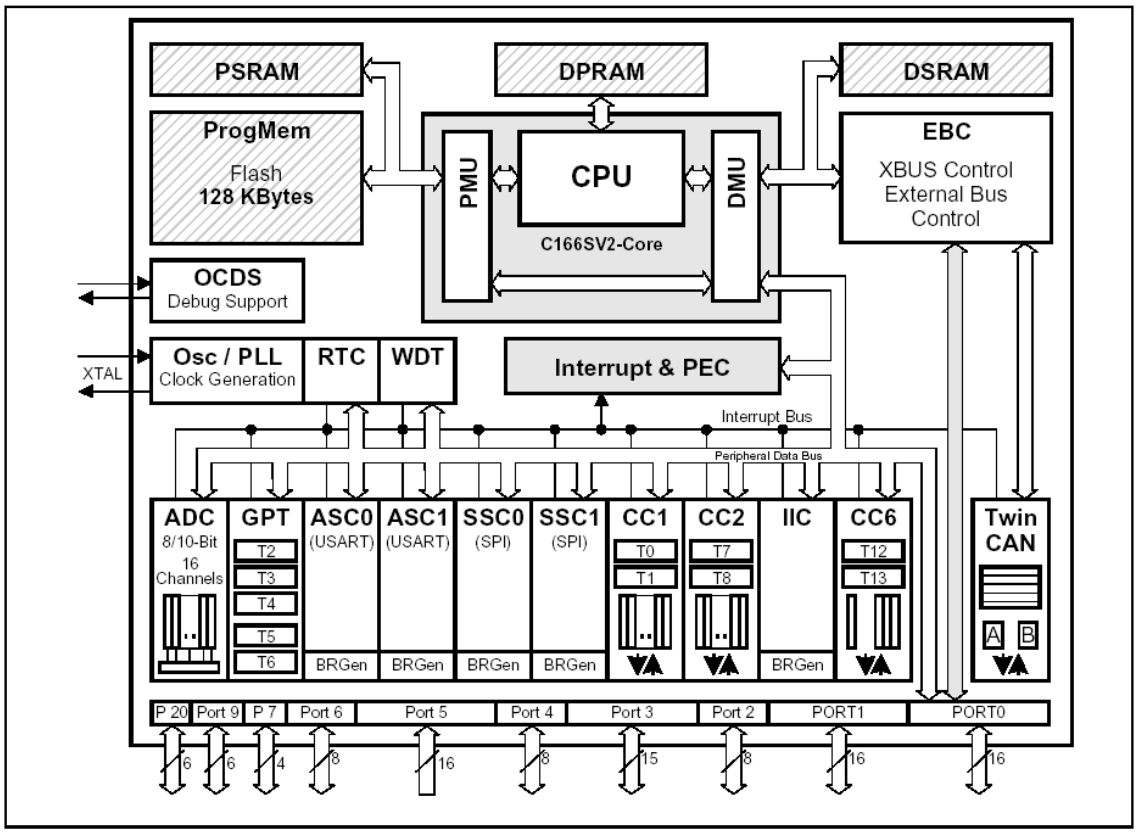

Abbildung 4.2: Blockschaltbild des xc167

CPU

Der Prozessorkern des xc167 kombiniert CISC- (Complex Instruction Set Computer) und RISC-Architektur (Reduced Instruction Set Computer). Eine Vielzahl von Befehlen mit großteils gleich langer Ausführungszeit wird in einer Pipeline abgearbeitet.

Funktionsblöcke der CPU

Die CPU kann in mehrere Funktionsblöcke gegliedert werden:

- PIPELINE: Alle Befehle werden in einer 2-stufigen Prefetch-Pipeline vorausgeladen und in einer 5-stufigen Ausführungs-Pipeline abgearbeitet. Jeder Befehl durchläuft damit insgesamt 7 Phasen.

- ALU: Die Arithmetic und Logic Unit führt logische und arithmetische Befehle aus.

- MAC-Unit: Die Multiplikations und Additionseinheit führt Multiplikations-, Divisions- und spezielle DSP-Befehle durch.

- SYSTEMREGISTER: Eine Reihe von Systemregistern dient zur Speicherung und Wiedergabe des Prozessorzustandes sowie zur Konfiguration von internem und externem Speicher

- REGISTERBANK: Die CPU des xc167 enthält keine separaten Arbeitsregister, sondern eine Registerbank, welche eigentlich im internen DPRAM liegt und deren Position veränderbar ist.

Pipelining

Der xc167 enthält 2 Pipelines: eine 2-stufige Fetch-Pipeline und die eigentliche 5-stufige Ausführungspipeline. Die Fetch-Pipeline lädt mehrere Befehle aus dem Speicher (z. B. Flash) im voraus und puffert diese in einem eigenen Befehls-Fifo. Der jeweils nächste auszuführende Befehl steht am Fifo-Ausgang der Ausführungs-Pipeline zur Verfügung.

Alle Befehle werden in der 5-stufigen Ausführungs-Pipeline abgearbeitet. Jeder Befehl durchläuft die Phasen

- DECODE: Befehl decodieren
- ADDRESS: Adresse für alle Operanden generieren
- MEMORY: Laden der Operanden
- EXECUTE: Befehl ausführen
- WRITEBACK: Ergebnis zurückschreiben

und braucht für jede Phase 1 Maschinenzyklus (50 ns bei 40 MHz). Die Gesamtdauer eines Befehles von Laden bis zum Ergebnis schreiben ist daher 4 Maschinenzyklen, der Befehlsdurchsatz liegt jedoch bei 1 Maschinenzyklus.

Auch die speziellen Befehle Multiplikation und kombinierte Multiplikation/Addition können dank MAC-Einheit (Multiply and Accumulate Unit) in einem Zyklus ausgeführt werden. Nur Divisionen benötigen mehrere Zyklen, wobei die Division im Hintergrund stattfinden kann und parallel dazu andere, vom Ergebnis unabhängige Befehle durchgeführt werden können.

Der Prefetch/Fetch-Mechanismus funktioniert für sequentielle Befehlsabarbeitung und unbedingte Sprünge reibungslos. Probleme kann es im Zusammenhang mit bedingten Sprüngen geben. Hier werden automatisch Leerbefehle eingefügt.

ALU und MAC-Einheit

Die Arithmetic and Logic Unit (ALU) dient zur Ausführung von logischen und arithmetischen Befehlen wie AND, OR, XOR, SHL, SHR, ROR oder ROL. Alle diese Befehle werden in 1 Zyklus ausgeführt.

Für Multiplikation, Division, kombinierte Multiplikation/Addition (Multiply-Add) sowie einige DSP-ähnliche Befehle zum Addieren, Subtrahieren, Bitweisen-Schieben mit 40 Bit Registerbreite sowie einfachem Wiederholmechanismus existiert die MAC-Einheit (Multiply Accumulate Unit). Diese läuft grundsätzlich unabhängig von der Abarbeitungs-Pipeline.

Systemregister

Die CPU beinhaltet auch einige Systemregister, welche u.a. den Zustand des Prozessors wiedergeben, zur Konfiguration des Programmcontextes (Registerbank, Stack, Tasklevel) bzw. der CPU selbst dienen:

CPUCON1&2: Diese Register dienen dazu, die XC167-CPU zu konfigurieren, wie beispielsweise die Breite der Interrupt Vektor Tabelle oder verschiedene Pipelining-Einstellungen. Im allgemeinen dürfen diese Einstellungen nicht durch das Anwenderprogramm verändert werden.
PSW: Das Program Status Word (PSW) enthält unter anderem den aktuellen Prozesslevel, das generelle Aktivierbit von Interrupts und die Zustandsflags der ALU (Arithmetic and Logic Unit).
IP: Das Register enthält die aktuelle Position (Offsetadresse) des Programmzeigers (Instruction Pointers) im Speicher-Adressraum.
CSP: Der Code Segment Pointer gehört zum IP und enthält die Segmentadresse des Programmzeigers.
DPPx: 4 Data Page Pointer enthalten die oberen Adressbits der jeweiligen Datenseite.
CP: Der Context Pointer gibt an, wo sich die Registerbank (siehe unten) gerade befindet.
SP: Der Stackpointer zeigt auf die aktuelle Position im Stack (Stapelspeicher). Die Größe und Position des Stacks ist über die weiteren Register SPSEG, STKOV und STKUN einstellbar.

Registerbank

Der xc167 hat keine eigenen Arbeits-Register in der CPU. Vielmehr gibt es eine Registerbank mit 16 Registern (R0-R15), welche im internen Dual Ported Ram (DPRAM) liegt und beliebig verschoben bzw. umgeschaltet werden kann. Damit ist die Anzahl der zu verwendenden Register fast beliebig groß. Das Positionieren bzw. Verschieben der Registerbank geschieht mittels Register CP (Context Pointer).
Die Funktionalität des Umschaltens der Registerbank wird auch Context Switch genannt und kann z. B. bei Interrupts genutzt werden, um jeder Interrupt Service Routine einen eigenen Registercontext zu verschaffen.

Blockschaltbild

Zusammenfassend die Einheiten und Register der CPU im Blockschaltbild dargestellt:

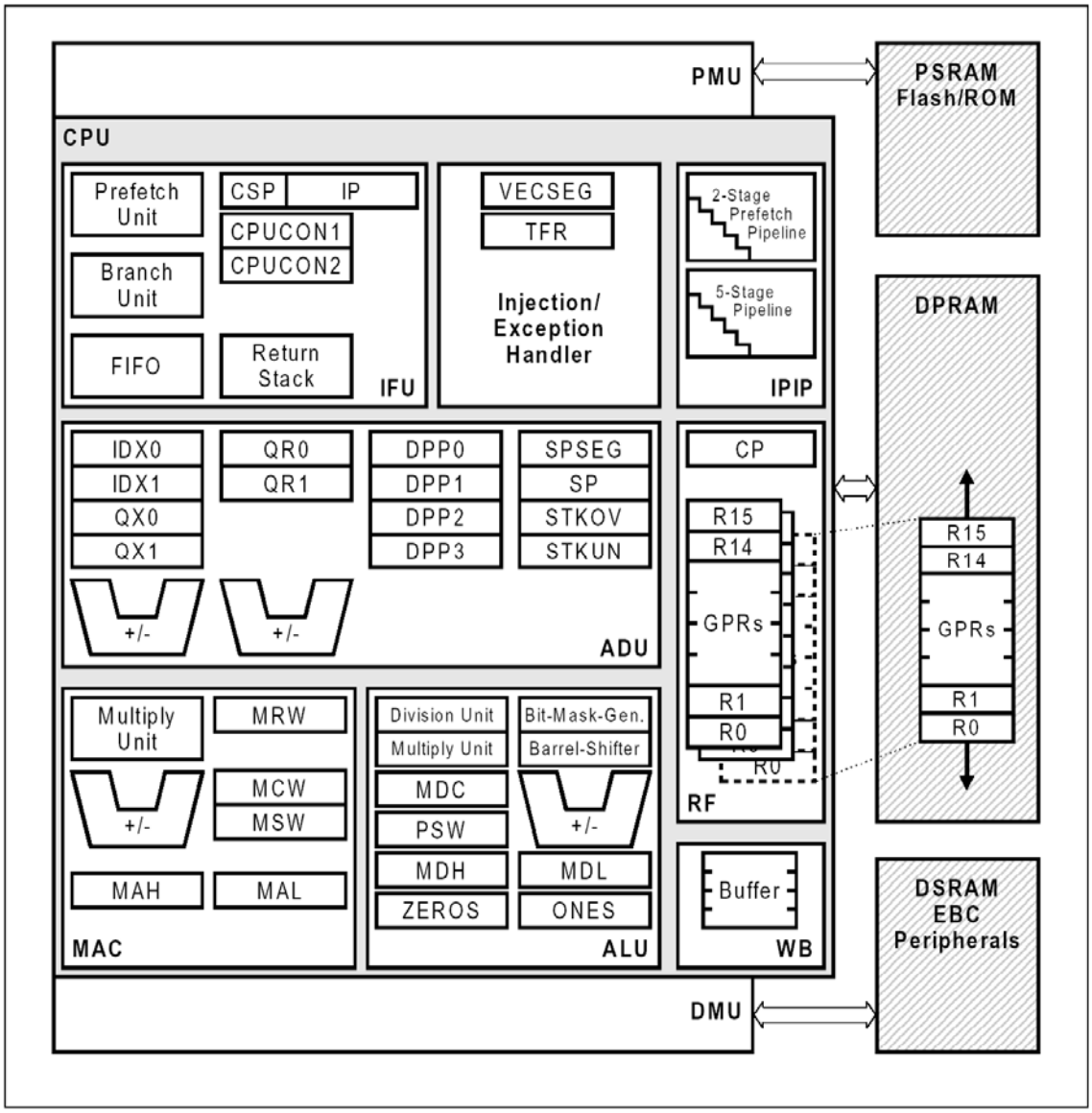

Abbildung 4.3: Systemarchitektur und Register der xc167-CPU

Speichermodell

Der xc167 kann einen Adressraum von 16 Mbyte (mit 24 Adressleitungen) ansprechen.

Segmentierung

Gebildet wird eine Code-Adresse aus Offsetadresse (Adressleitungen A0-A15) und Segmentadresse (die oberen Adressleitungen A16-A19).

Datenseiten

Daten werden etwas anders gegliedert als Code. Hier kommen sogenannte Datenseiten (Data Pages) zur Anwendung. Jede Datenseite ist 16 kByte groß.
4 Datenseiten stehen zur Verfügung, und deren Position im gesamten Adressraum wird in den 4 Data Page Pointern (DPP0-DPP3) gespeichert.

Besondere Speicherbereiche

Insgesamt kann der xc167 16 MB Speicher addressieren. Von diesem theortischem Adressbereich sind einige Bereiche bereits vorbestimmt und teilweise vorbelegt:

Segment 00h enthält von Adresse 00‘C000 bis 00‘FFFF internes RAM für Daten sowie sämtliche SFRs
Segment 20h enthält die beiden CAN-Module, welche memory-mapped sind.
Segment C0h und C1h enthält das interne, 128kB große Flash.
Segment E0h enthält das interne RAM für Programm und Daten, insgesamt
Segment FFh enthält den Bootcode (für Bootstrap-Mode) sowie die Flash-Statusregister.

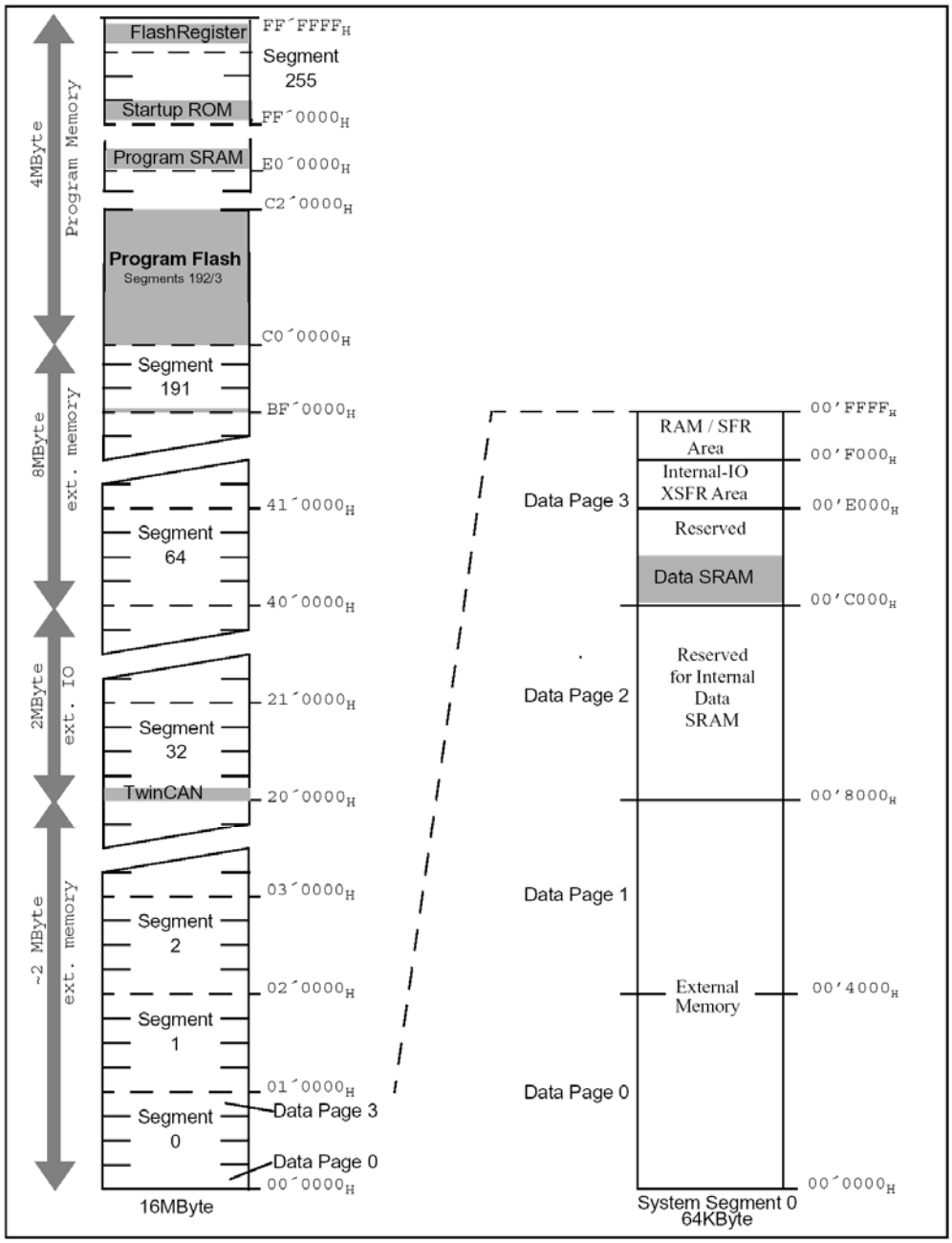

Abbildung 4.4: Speichermodell des xc167

Internes Flash

Des weiteren befindet sich in den Segmenten 0xE0 und 0xE1 das interne Flash, welches 128 kByte groß ist. Das Flash ist in insgesamt 6 Sektoren unterschiedlicher Größe geteilt. Die folgende Abbildung zeigt die Aufteilung und Adressen des internen Flashes:

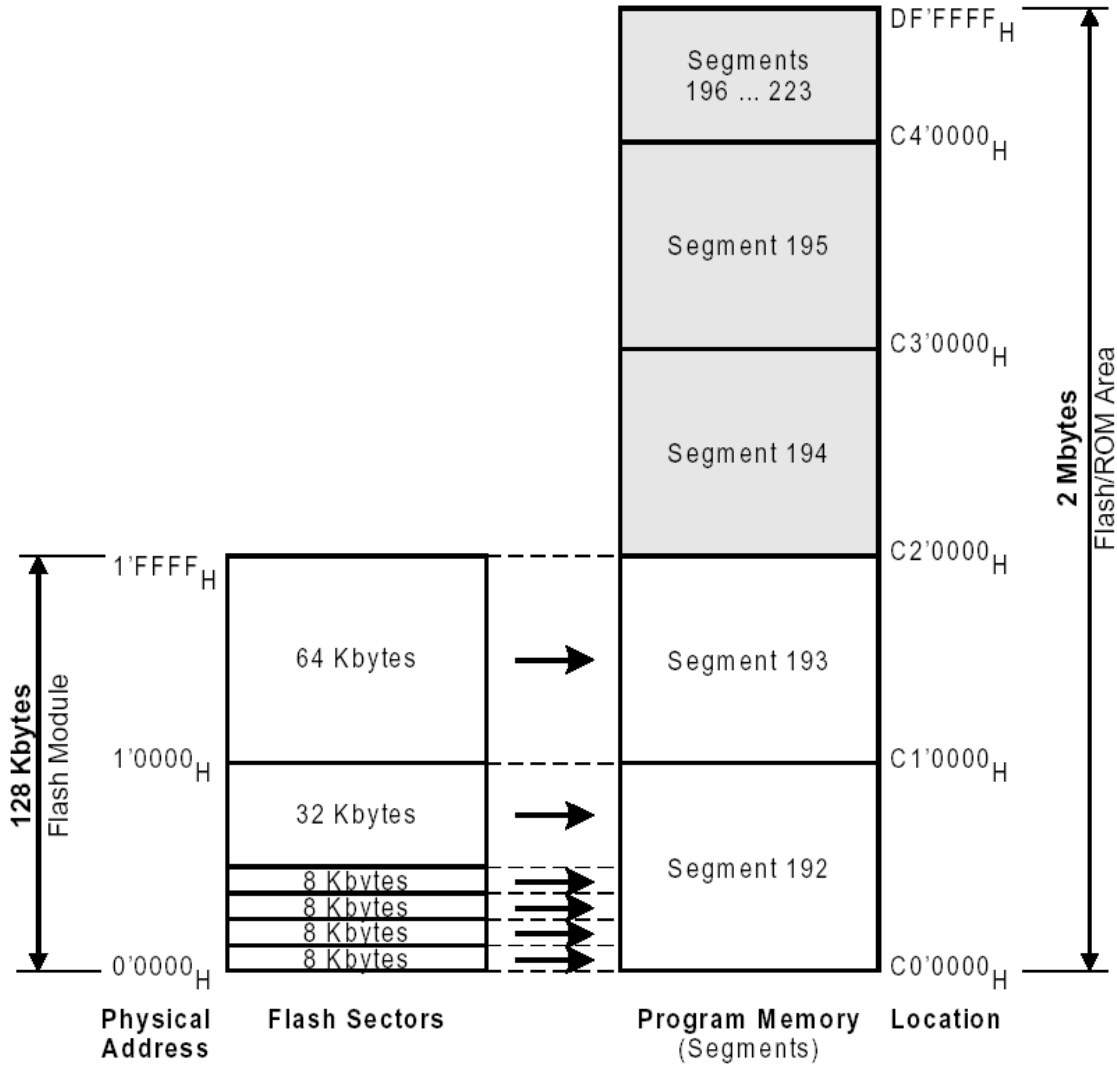

Abbildung 4.5: Das interne Flash des xc167

Internes RAM und SFRs

Der xc167 bietet mehrere Bereiche von internem RAM und memory-mapped SFR-Registern. Ab Adresse 0x00C000, also auf den oberen 16 kByte des Segment 0, sind zu finden:

DSRAM: 00‘C000-00‘CFFF 4kByte Daten-RAM

XSFR-Bereich	00‘E000-00‘F1FF	Adressbereich, in dem SFR-Register gemappt sind.
ESFR-Bereich	00‘F000-00‘F1FF	Adressbereich, in dem SFR-Register gemappt sind.
DPRAM	00‘F600-00‘FDFF	2kByte Dual Ported RAM, nur Daten, Registerbänke
SFR-Register:	00‘FE00-00‘FFFF	Adressbereich, in dem SFR-Register gemappt sind.

Die folgende Abbildung zeigt die vorbelegten Speicherbereiche des untersten Segmentes ab Adresse 00‘C000, insbesondere internes RAM und die verschiedenen SFR-Bereiche:

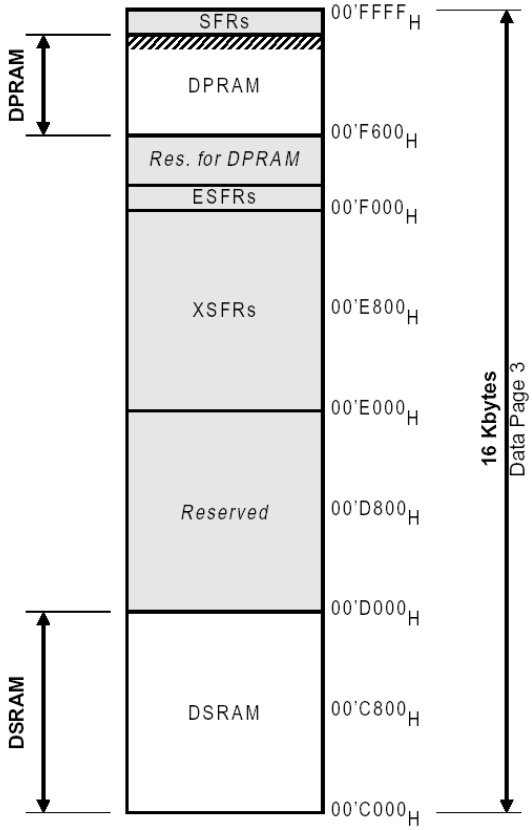

Abbildung 4.6: Belegung des xc167 Speicherbereiches von C000-FFFF

Zusätzlich bietet der xc167 noch 2kByte internes Ram ab Adresse E0‘0000, welches als Daten- oder Programmspeicher eingesetzt werden kann.

Externer Bus

Reicht das interne RAM bzw. FLASH nicht aus, so ist beim xc167 die Beschaltung von externen Speicherbausteinen vorgesehen. In diesem Fall werden je nach Busmodus bis zu 3 I/O-Ports zu Adress- und Datenleitungen.

Der externe Bus kann in vielerlei Hinsicht sehr flexibel eingestellt werden:

- Bus-Timing: Hier können Waitstates, verlängertes Read bzw. Write-Signal, verlängertes ALE-Signal u.v.m. ausgewählt werden
- Bus-Arbitration: Fixes Timing oder Warten auf ein READY-Signal
- Bus-Breite: 8 Bit oder 16 Bit Datenbreite
- Bus-Modus: gemultiplext (die Adressen werden gelatcht) oder nicht gemultiplext

Damit sind praktisch alle am Markt erhältlichen FLASH- und SRAM-Speicherbausteine und eine Vielzahl an anderen Controller- oder DSP-Bausteinen vom xc167 aus ansprechbar. Die genannten Einstellungen werden in den Registern EBCMOD0, EBCMOD1, TCONCSx und FCONCSx vorgenommen. Das x steht hier für die jeweilige Chipselect-Leitung

Adressfenster

Beim Einsatz mehrerer externer Speichertypen mit unterschiedlicher Konfiguration ist es auch möglich, Adressfenster festzulegen. Jeder Bausteintyp kann dann in ein eigenes Adressfenster gemappt und mit individuellen Buseinstellungsmöglichkeiten konfiguriert werden.

Der xc167 kann insgesamt 7 verschiedene Adressfenster und damit 7 verschiede Bausteintypen verwalten. Jeder Baustein wird dabei über ein eigenes CS-Signal angesprochen. Die Festlegung der Adressfenster und die zugehörige Buskonfiguration wird in den Registern ADDRSEL1-ADDRSEL7, TCONCS1 – TCONCS7 und FCONCS0 – FCONCS7 durchgeführt.

Für alle Bausteine, die nicht in einem so definierten Adressfenster liegen, gilt eine ebenfalls konfigurierbare Standardeinstellung in den Registern TCONCS0 und FCONCS0.

Letztendlich sind also 8 mögliche Bausteintypen mit unterschiedlichen Konfigurationen möglich.

Interrupts

Das Interrupt System des xc167 wird auch „Betriebssystem on Silizium“ genannt. Es ermöglicht, die 77 Interruptquellen in 16 Prioritätsebenen zu teilen.

Interruptquellen

Interruptquellen sind fast immer Peripherieeinheiten, welche auf Grund eines externen oder internen Ereignisses einen Interrupt auslösen können. Jede Interruptquelle kann ein- oder ausgeschaltet werden. Außerdem existiert ein „General Interrupt Enable“ Flag (das Flag IEN), welches generell die Erzeugung von Interrupts freigibt oder sperrt.

Es seien hier ein paar Beispiele für die Generierung von Interrupts durch die Peripherie angeführt:

- Asynchrone Schnittstelle: Byte empfangen
- A/D-Wandler: Wert fertig konvertiert
- Timer: Timerüberlauf

- CAPCOM: Externer Interrupt

Daneben gibt es noch viele andere Möglichkeiten, bei denen Interrupts erzeugt werden können.

Interruptebenen

Für jede der Interruptquellen ist es möglich, eine von 16 Interruptebenen zu wählen. Damit können alle Ereignisse in wichtigere und weniger wichtige „Tasks“ eingeteilt werden. Laufende Ereignisprozeduren (Interrupt-Handler) können von höherprioren unterbrochen werden (nicht jedoch von Ereignissen gleicher Interruptebene).

Innerhalb einer Interruptebene können auch noch Gruppenebenen definiert werden, um die Abarbeitung von 2 Interrupts gleicher Priorität nochmals zu gliedern. Interrupts gleicher Interruptebene sollten immer verschiedene Gruppenebenen haben.

PEC-Transfers

Neben dem herkömmlichen Interrupt-Handling durch die CPU gibt es die Möglichkeit, bei Auftreten eines Interrupts vorbei an der CPU einfache Transferoperationen durchzuführen. Diese Peripheral Event Controls (PECs) werden vom PEC-Controller durchgeführt, fast ohne Prozessorzeit zu verbrauchen (1 Befehlszyklus wird allerdings verbraucht). Dies entspricht in etwa dem Vorgang eines DMA (Direct Memory Access).

Der Datentransfer kann entweder wortweise oder byteweise erfolgen und wird von einer beliebigen Ursprungsadresse im Segment 0 in eine beliebige Zieladresse im Segment 0 durchgeführt. Sowohl Ursprung als auch Ziel können entweder RAM oder ein SFR-Register sein. Es ist sogar möglich, die Ursprungsadresse oder die Zieladresse (oder auch beide) nach dem Transfer automatisch zu erhöhen. Auch ist es möglich, nach einer bestimmten Anzahl an transferierten Datenwörtern einen Interrupt auszulösen, z. B. um das Ende des Transfers anzuzeigen.

Konfiguriert werden PEC-Transfers über die PECC-Kontrollregister und durch die Interrupt-Ebene und die Gruppenebene. Der Interrupt Level muß 14 oder 15 sein.

I/O-Ports

Der xc167 bietet bis zu 103 I/O-Leitungen, um digitale Zustände oder Signale zu lesen bzw. zu erzeugen. Die I/O Leitungen sind in 8 Bit bzw. 16 Bit Ports gegliedert. Dadurch ist neben dem bitweisen Zugriff auch ein byte- bzw. wortweiser Zugriff auf einen Port möglich. Insgesamt gibt es neun Ports.

Einstellungen

Fast jeder Portpin kann per Software individuell als digitaler Eingang oder Ausgang benutzt werden.

Außerdem gibt es für manche Ports auch die Möglichkeit, diese per Software als Open Drain zu definieren. Dabei wird ein externer Pull-Up Widerstand vorausgesetzt.

Des weiteren kann für ausgewählte Ports anstelle von TTL-Pegeln eine spezielle, CMOS-ähnliche Hysterese zum Erkennen der beiden logischen Zustände konfiguriert werden. Dies ist vor allem bei zu erwarteten verrauschten Signalen hilfreich.

Als weitere Besonderheit kann für jeden Port (in Vierergruppen) eingestellt werden, wie belastbar der Port bei Konfiguration als Ausgang ist. Hier gibt es 3 Stufen: Weak, Medium, Strong, die Standardeinstellung ist Strong. Hintergrund ist das Zurücknehmen des Stromverbrauchs, wenn keine hohe Leistung an den Ports benötigt wird.

Doppelfunktionen

Viele der I/O-Pins haben neben der Nutzung als digitale Ein/Ausgabeleitung noch eine alternative zweite Funktion. Diese sind zumeist Funktionen im Zusammenhang mit den verschiedenen Peripherieeinheiten (z. B. als Leitungen für die serielle Schnittstelle, als externe Interrupts, als Countereingang, als CAN-Leitungen oder als Analog-Eingang für den ADC).

Speziell Port 0, Port 1 und Port 4 werden – je nach Busmodus – für den externen Bus verwendet, falls externe Speicher-Bausteine eingesetzt werden. Diese Ports stehen dann nicht mehr frei zur Verfügung.

Ob ein Portpin als digitale Ausgabeleitung oder in einer Alternativfunktion genutzt wird, muss über das eigene Register ALTSEL0Px bzw. ALTSEL1Px eingestellt werden. Wenn ein Port für eine alternative Eingabefunktion genutzt wird, ist keine zusätzliche Konfiguration erforderlich.

Watchdog-Timer

Der xc167 bietet wie alle Familienmitglieder einen Watchdogtimer, welcher den ordnungsgemäßen Programmstart und Programmlauf überwacht.

Dieser Watchdogtimer wird nach Reset automatisch gestartet und löst nach einer gewissen Zeit einen Softwarereset aus. Daher muss der Watchdogtimer von der Anwendungssoftware sobald als möglich – zumeist während der Initialisierungsphase – ausgeschaltet oder gewartet werden. Nur bei ordnungsgemäßem Programmstart wird dann auch der Watchdogtimer ausgeschaltet.

Der xc167 sieht zum Ausschalten des Watchdogtimers einen eigenen Befehl vor: DISWDT. Ebenso gibt es zum Warten den Befehl SRVWDT.
Wird der Watchdogtimer zum Überwachen der Anwendersoftware eingesetzt, so sollte der Aufruf von SRVWDT möglichst nur im niedrigpriorsten, regelmäßig aufgerufenen Task erfolgen. Dieser Task ist entweder das Hauptprogramm oder ein eigener IDLE-Task (sofern ein Echtzeitbetriebssystem eingesetzt wird).

Peripherie

In diesem und den folgenden Kapiteln wird die integrierte Peripherie des xc167 detailiert beschrieben, besonders die Funktionen, die für die Robotik interessant sind. Der xc167 hat eine Vielzahl an On-Chip-Peripherieeinheiten:

- A/D-Wandler: Der xc167 beinhaltet einen 10 Bit A/D-Wandler mit minimaler Wandlungszeit von etwa 3 µs. Bis zu 16 Analogsignale können mit Hilfe eines Multiplexers erfasst werden.
- 2 Asynchrone Schnittstellen: Die asynchronen seriellen Schnittstelle sind vor allem zur Kommunikation mit Standard-Computersystemen z. B. für Programm-Update oder Datenaustausch erforderlich. Des weiteren bilden sie oft die Anbindung zu Kommunikationsmodulen wie Bluetooth, Dect oder GSM.
- 2 Synchrone Schnittstellen: Mit synchronen Schnittstellen (auch SPI) können mit nur 2 bzw. 3 Leitungen rasch Daten mit anderen Controllern, externen Bausteinen oder seriellen Speichern ausgetauscht werden.
- 3 Capture/Compare Einheiten: Diese Peripherie kann zum Aufzeichnen oder Erzeugen externer Signale und Impulse, im speziellen auch PWM-Signale verwendet werden. Dies betrifft auch die Erkennung externer Interrupts.
- 2 CAN Module: Dieses aus dem Automobilbereich kommende Bussystem dient zur Vernetzung beliebig vieler Mikrocontroller oder intelligenter Sensoren. Die Übertragung erfolgt seriell mit bis zu 2 MBit/s.
- 2 Zähler-Einheiten: Die beiden GPT Einheiten (General Purpose Timer) bieten insgesamt fünf 16 Bit Zähler mit einer Vielzahl von Einstellungsmöglichkeiten. Mit Zählern können nicht nur periodische Signale oder Programmaufrufe erzeugt oder abgetastet, sondern auch externe Impulse gezählt werden. Der xc167 bietet zusätzlich die Möglichkeit, einen Zähler als Inkrementaldecoder zu verwenden. Neben diesen eigens dafür vorgesehenen Zählern in den Zählereinheiten können auch die Zähler der Capture/Compare Einheiten für diese Zwecke eingesetzt werden, sofern sie dort nicht benötigt werden.
- I²C Bus: Dieser standardisierte, v.a. im Industrie- und Messtechnik eingesetzte serielle Bustyp steht auch im xc167 zur Verfügung und kann bis zu 400 kBit/s über nur 2 Leitungen übertragen.
- Real Time Clock: Die Echtzeituhr des xc167 läuft völlig unabhängig vom Prozessortakt mit eigenem Quarz, ist hinsichtlich Stromverbrauch optimiert und kann vielfältig konfiguriert werden. Es ist auch möglich, über eine Wecker/Alarmfunktion einen Interrupt auszulösen.

Interrupts

Jede Peripherieeinheit kann bei Bedarf Interrupts auslösen. Jeder mögliche Interrupt einer Peripherie (z. B. Empfang eines Bytes, Sendung komplett, Wandlung komplett, Timerüberlauf, Externer Interrupt) kann auf eine eigene Interruptebene (Priority Level) gesetzt werden. Dies geschieht immer im entsprechenden Interrupt Control Register (der Name jedes Interrupt Control Registers endet mit xxxIC).

Damit ein Interrupt auch wirklich ausgelöst wird, muss neben der Festlegung und Aktivierung eines individuellen Interrupts im Interrupt Control Register (xxxIC) auch global das Flag IEN (Interrupt Enable) im Program Status Word (PSW)gesetzt sein.

Als Beispiel soll der A/D-Wandler so konfiguriert werden, dass nach erfolgreicher Wandlung ein Interrupt der Ebene 12 ausgelöst wird.

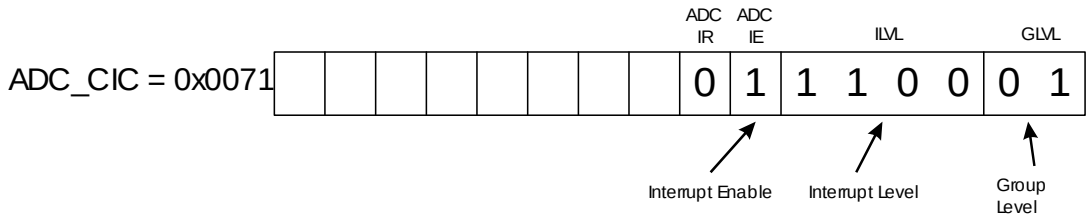

Abbildung 4.7: Beispielkonfiguration für das Register ADCIC

Alle Interrupt Control Register sind ähnlich aufgebaut: die beiden niederwertigsten Bits sind der Gruppenlevel (GLVL), danach folgen 4 Bits Interruptebene (ILVL), das Bit Interrupt Enable (xxxIE) zum Aktvieren dieses Interrupts und das Interrupt Request Bit (xxxIR), welches durch die Hardware gesetzt wird.

Ports

Da bestimmte I/O Ports vielfach auch im Zusammenhang mit der Peripherie eingesetzt werden können, ist zu beachten, dass der entsprechende Portpin je nach Funktionalität auch wirklich als Eingang bzw. Ausgang definiert ist. Für jeden Portpin kann das Direction Port Register (DPx.y) auf Eingang (0) oder Ausgang (1) gesetzt werden.

So muss beispielsweise beim Einsatz der asynchronen seriellen Schnittstelle der Port TXD (P3.10) als Ausgang (DP3.10=1) und der Port RXD (P3.11) als Eingang (DP3.11=0) konfiguriert werden.

Im Folgenden werden die wichtigsten Peripherieeinheiten des xc167, ihre Anwendungsmöglichkeiten, Betriebsmodi sowie ein Konfigurationsbeispiel beschrieben:

Analog/Digitalwandler

Der integrierte A/D-Wandler dient zur Digitalisierung von analogen Spannungswerten.

Der A/D Wandler hat eine Auflösung von 10 Bit und eine minimale Wandlerrate von 2,85 µs. Alternativ kann auch eine Auflösung von 8 Bit gewählt werden. Als Analogeingänge dienen die I/O-Leitungen von Port 5, an welcher Spannungen zwischen 0 und 5V eingesampled werden. Jede dieser 16 Portleitungen (=Kanal) kann dank des integrierten Multiplexers eingelesen und vom A/D-Wandler digitalisiert werden.

Anwendungen

Der A/D-Wandler dient allgemein zur Erfassung von analogen Spannungswerten. Anwendungsgebiete gibt es beliebig viele:

- Temperaturmessung (z. B. PTC oder NTC über Spannungsteiler)
- Werkstoffprüfung oder Statikmessung mittels Dehnmessstreifen (über Spannungsteiler)
- Abstandmessung mittels Induktionssensoren
- Betriebsspannungsüberwachung über Spannungsteiler

Betriebsmodi

Der im C167 integrierte A/D-Wandler bietet 4 Betriebsmodi:

FIXED CHANNEL SINGLE CONVERSION: In diesem Modus wird ein bestimmter einzelner Kanal einmal digitalisiert.
FIXED CHANNEL CONTINOUS CONVERSION: In diesem Modus wird ein bestimmter einzelner Kanal dauernd digitalisiert.
AUTO SCAN SINGLE CONVERSION: Dieser Modus dient dazu, mehrere angewählte Kanäle einmal zu digitalisieren.
AUTO SCAN CONTINOUS CONVERSION: Dieser Modus dient dazu, mehrere angewählte Kanäle dauernd zu digitalisieren.
WAIT FOR ADDAT READ: Bei diesem Modus wird eine neue Wandlung erst gestartet, wenn der alte Wert gelesen wurde.

Damit ist es möglich, den Wert von einem, zwei, drei ... bis zu sechzehn Kanälen entweder einmal oder dauernd hintereinander zu digitalisieren. Die AUTO SCAN Modi dienen vor allem dazu, um ohne dauernd per Software den Kanal zu wechseln, mehrere Kanäle hintereinander zu digitalisieren.

Speziell für die CONTINOUS CONVERSION Modi bietet es sich an, mittels PEC-Transfer die Daten vom A/D-Wandler-Register in einen vordefinierten Buffer zu schreiben und so Prozessorzeit einzusparen.Erst wenn der Buffer gefüllt ist, wird ein Interrupt ausgelöst.

Beispiel

Als Beispiel für den Einsatz des A/D-Wandlers wird die Überwachung der Betriebsspannung angeführt. Dazu soll Kanal 9 (Port 5.9) dauernd gewandelt werden.

Für den A/D-Wandler sind 3 SFR-Register relevant:

ADC_CON: Das Control-Register zum Konfigurieren des A/D-Wandlers (Auswahl des Kanales, Setzen des Modus...)
ADC_DAT: Das Datenregister des A/D-Wandlers. Nach jeder Wandlung enthält dieses Register den gewandelten 10 Bit-Datenwert.

ADC_IC: Das Interrupt Control Register für den A/D-Wandler. Hier kann der Interrupt aktiviert werden. Ein Interrupt wird ausgelöst, sobald eine Wandlung zu Ende ist und die Daten im Datenregister ADC_DAT stehen.

Das Register ADC_CON wird folgendermaßen initialisiert:

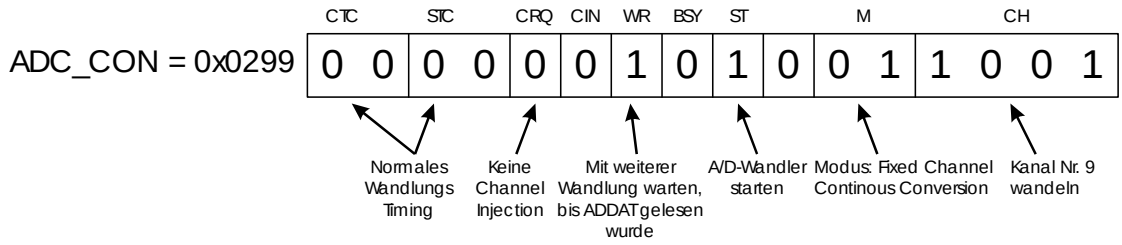

Abbildung 4.8: Beispielkonfiguration für den A/D-Wandler: Register ADC_CON

Asynchrone serielle Schnittstellen

Die asynchronen Schnittstellen ASC0 und ASC1 des xc167 bietet eine Reihe von Anwendungsmöglichkeiten:

Anwendungen

C167 dient vor allem zum:

- Datenaustausch mit herkömmlichen Computersystemen (z. B. Service Schnittstelle)
- Einspielen von Programmupdates
- Ausgabe von Debug- und Statusinformationen
- Ausgabe auf Drucker
- Übertragung von Daten über Funk (Bluetooth, GSM, DECT, ...)

Betriebsmodi

Bei der asynchronen seriellen Schnittstelle des C167 kann die Anzahl der Datenbits (7, 8 oder 9), das Parity (keines, gerade, ungerade) und die Anzahl der Stopbits (1 oder 2) eingestellt werden. Weiters gibt es die Möglichkeit eines IrDA-Modus und der automatischen Baudratenerkennung.
Die Baudrate kann je nach Prozessortaktfrequenz zwischen 50 Baud und 2,5 MBaud liegen und ist ebenfalls sehr großzügig einstellbar.

Beide asynchrone Schnittstellen besitzen ein 8 Byte großes FIFO sowohl für die Sende- als auch für die Empfangsrichtung. Die FIFOs können individuell aktiviert und deren Status abgefragt werden.

Eine weitere Besonderheit ist die Baudrateneinstellung: Mittels Teilerregister wird die CPU-Frequenz auf die gewünschte Baudrate gestellt. Ein immer wiederkehrendes Problem ist dabei die Tatsache, dass besonders bei hohen Baudraten diese aufgrund der vorgegebenen

Systemtaktfrequenz nicht genau erreicht werden kann. Um dies trotzdem zu erreichen, existiert beim xc167 ein zusätzlicher, nicht ganzzahliger Teiler: der fractional Divider. Mit diesem können Baudraten besonders genau eingestellt werden.

Wichtig ist, dass für RS232/V.24-Kompatibilität noch ein externer Treiberbaustein benötigt (z. B. MAX202). Wenn dagegen beispielsweise Kommunikations-Module mit asynchroner Schnittstelle auf TTL-Pegel angeschlossen werden, ist im allgemeinen kein Treiberbaustein erforderlich.

Beispiel

Die serielle Schnittstelle soll in diesem Beispiel für serielle Kommunikation mit 9600 Baud, 8 Datenbits, keinem Parity und 1 Stopbit (9600,n,8,1) initialisiert werden. Die CPU-Frequenz beträgt 20 MHz.

Folgende Register sind für die serielle Schnittstelle relevant:

ASC0_CON: Kontrollregister zum Setzen der Schnittstellenparameter und des Modus.
ASC0_S0BG: Baudratenregister zum Setzen der Baudrate
ASC0_FDV: Fractional Divider Register zum Erhöhen der Baudratengenauigkeit
ASC0_S0RBUF: Datenregister zum Empfangen eines Bytes
ASC0_TBUF: Datenregister zum Senden eines Bytes
ASC0_TIC: Transmit Interrupt Register: Hier wird konfiguriert, ob ein Interrupt beim Senden ausgelöst werden soll. Der Transmit Interrupt wird ausgelöst, sobald der gesamte Frame der Sendung übertragen wurden.
ASC0_RIC: Receive Interrupt Register: Hier wird konfiguriert, ob ein Interrupt beim Empfangen ausgelöst werden soll. Der Receive Interrupt wird ausgelöst, sobald ein kompletter und korrekter Datenframe empfangen wurde.

Da 9600 Baud bei 20 MHz recht genau erreicht werden können, muss der Fractional Divider nicht aktiviert werden.

Der Modus wird über das Register ASC0_CON eingestellt:

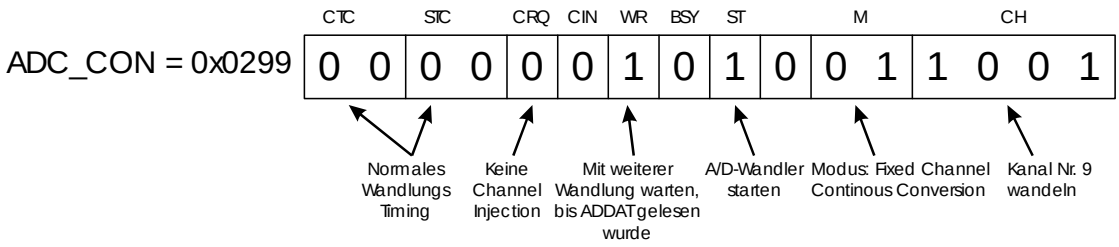

Abbildung 4.9: Beispielkonfiguration für die asynchrone serielle Schnittstelle: Register ASC0_CON

Die Baudrate wird auf 9600 eingestellt:

$$ASC0_BG = \frac{FCPU}{16*(2+BRS)*BAUDRATE} - 1 = \frac{20\,000\,000}{16*(2+0)*9600} - 1 = 0x0040$$

Abbildung 4.10: Einstellen der Baudrate für die asynchrone Schnittstelle über das Register ASC0_BG

Synchrone Serielle Schnittstellen

Die synchrone seriellen Schnittstellen (SSC0 und SSC1) benötigen zur Datenübertragung neben 1 bzw. 2 Datenleitungen noch eine Taktleitung. Mit dem Takt wird festgelegt, wann ein Datum gültig ist. Die synchronen seriellen Schnittstellen des xc167 sind SPI-kompatibel und unterstützen bis zu 20 MBaud bei 40 MHz.

Anwendungen

Die Synchrone Serielle Schnittstelle kann überall dort eingesetzt werden, wo ein weiterer Baustein ebenfalls seriell Daten bereitstellt bzw. liest. Das sind unter anderem:

- serielle EEPROMs
- serielle Displays
- Seriell-Parallelwandler
- Nachbildung eines asynchronen seriellen Signales
- Kommunikation mit anderen Prozessoren oder externen Peripherieeinheiten

Betriebsmodi

Die Datenbreite der übermittelten Daten kann zwischen 2 und 16 Bit liegen. Weiters kann eingestellt werden, ob die Sendung mit dem LSB (Least significant Bit, die letzte Stelle im Wert) oder dem MSB (Most Significant Bit) beginnen soll.

Die Übertragungsrate kann sehr großzügig festgelegt werden und liegt je nach Taktfrequenz zwischen 300 Baud und 20 MBaud (bei 40 MHz).

Auch das Taktsignal kann frei konfiguriert werden. Es ist einstellbar, ob die Daten mit steigender oder mit fallender Taktflanke übernommen werden sollen. Auch die Phase des Taktsignals ist einstellbar.
Der Modus (Master oder Slave) bestimmt, ob der Takt vom Controller erzeugt wird, oder ob er vom Kommunikationspartner generiert wird.

Beispiel

Mit der synchronen seriellen Schnittstelle des xc167 soll das Senden asynchroner Datenframes simuliert werden. Simuliert werden sollen asynchron gesendete Daten bei einer Baudrate von

9600, 8 Datenbits, 1 Startbit, 1 Stopbit, kein Parity, d.h. es werden pro Frame insgesamt 10 Bits übertragen. Die CPU-Frequenz beträgt 20 MHz. Folgende Register betreffen die SSC:

SSC0_CON: Kontrollregister für die SSC. Hier werden Datenbreite und Modus eingstellt.
SSC0_BR: Baudratenregister der SSC. Hier wird die Übertragungsrate eingestellt.
SSC0_TB: Sendebuffer. Enthält die Daten (in unserem Fall 10 Bits), welche übertragen werden sollen
SSC0_RB: Empfangsbuffer. Enthält empfangene Daten. Im Beispiel wird mit der Schnittstelle nur gesendet.
SSC0_TIC: Transmit Interrupt Register: Hier kann der Sendeinterrupt aktiviert werden, welcher nach der kompletten Übertragung eines Frames auftritt.
SSC0_RIC: Receive Interrupt Register: Tritt auf, wenn ein korrekter Datenframe empfangen wird.

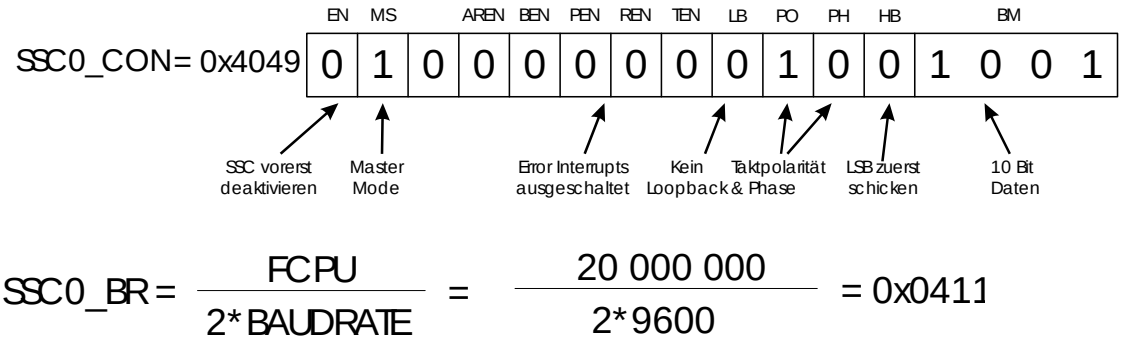

Abbildung 4.11: Beispielkonfiguration für die serielle synchrone Schnittstelle 0

Capture / Compare – Einheiten 1 und 2

Insgesamt stellt der xc167 3 Capture / Compare Einheiten zur Verfügung. Während Einheiten 1 und 2 „echte“ CAPCOMs darstellen, ist Einheit 6 eigentlich nur zur PWM-Signalgenerierung gedacht.

Die beiden Capture/Compare Einheiten CAPCOM1 und CAPCOM2 bieten eine Reihe von

Anwendungen

Die CAPCOM-Einheiten dienen vor allem für:

- Messung von Zeitpunkten, wann ein bestimmtes Signal oder ein Signalwechsel aufgetreten ist. (CAPTURE-Mode)
- Erzeugung von Impulsen und Signalen, z. B. PWM-Signal (COMPARE-Mode)
- Einsatz als Timer
- Einsatz als Externe Interrupts

Betriebsmodi

Bei den CAPCOM-Einheiten gibt es grundsätzlich 2 verschiedene Betriebsmodi:

CAPTURE: Dieser Modus wird dazu verwendet, den Zeitpunkt eines externen Ereignisses (z. B. Signalwechsel an einem bestimmten Port) festzuhalten. Dies geschieht dadurch, dass zuerst ein Zähler aktiviert wird. Sobald auf dem entsprechenden Kanal (I/O-Pin) ein bestimmter (konfigurierbarer) Signalwechsel eintritt, wird der Inhalt des Zähler-Wertes in ein Register geschrieben und - wenn erwüscht - ein Interrupt ausgelöst. Dies ist quasi der Lese-Modus und kann auch als externer Interrupt genutzt werden.

COMPARE: Dieser Modus wird verwendet, um Signale (z. B. PWM-Signale oder Impulse) zu erzeugen. Dazu wird ebenfalls ein Zähler gestartet und ein Schwellwert angegeben. Erreicht der Zähler den Schwellwert, so wird am entsprechenden Pin ein Signal erzeugt und bei Bedarf ein Interrupt ausgelöst. Hier gibt es noch eine Reihe zusätzlicher Einstellungsmöglichkeiten: So kann auch der Schwellwert noch während der Periode nachgestellt werden (und so mehrere Impulse pro Periode erzeugen), oder der Pin beim Zählerüberlauf wieder rückgesetzt werden (dies wird z. B. für asymmetrisches PWM-Signal eingesetzt).

Jede CAPCOM-Einheit ist mit je 2 Zählern ausgestattet. Diese beiden Zähler können auch unabhängig von den beiden Modi verwendet werden.

Beispiel

Die CAPCOM1 soll benutzt werden, um einen externen Interrupt (steigende Flanke) zu erfassen. Der Interrupt ist auf Portpin P2.13 (CC13IO) angeschlossen.

Folgende Register gehören zur CAPCOM1:

CC1_T0,CC1_T0REL, CC1_T1, CC1_T1REL, CC1_T01CON, CC1_T0IC, CC1_T1IC: Timer Register. Die Timer werden im Beispiel nicht verwendet.
CC1_M0-CC1_M7: Capture/Compare Modus Register. Jedes Register beinhaltet den Modus für vier Kanäle (4 Bits pro Kanal). Im Beispiel wird der Kanal 13 (I/O Pin CC13IO) genutzt und daher das Modusregister CC1_M3 benötigt.
CC1_CC0IC-CC1_CC15IC: Interrupt Register zur Aktivierung des Interrupts. Im Beispiel wird Interrupt Register CC1_CC12IC benutzt.

Das Modusregister CC1_M3 wird daher folgendermaßen gesetzt:

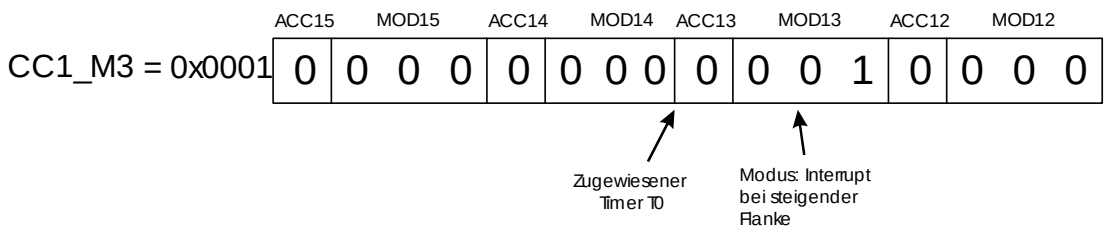

Abbildung 4.12: Beispielkonfiguration für die Capture/Compare Einheit als ext. Interrupt

Es muss ein Timer zugewiesen werden, auch wenn dieser keine Funktion erfüllt und nicht läuft. Zusätzlich muss auch noch das Interrupt Register CC1_CC13IC auf den gewünschten Level gesetzt und der Interrupt aktiviert werden.

CAN-Module

Eine weitere bedeutende OnChip-Peripherie des xc167 ist das TwinCAN-Modul. Der im Automotive-Bereich entwickelte und dort eingesetzte schnelle serielle 2-Draht Bus CAN (Controller Area Network) dient zum Datenaustausch und zur Vernetzung beliebig vieler Controller oder anderer Bausteine. Da eine detaillierte Beschreibung des CAN-Busses zu weit führen würde, soll hier nur kurz das Wichtigste beschrieben werden:

Anwendungen

Der CAN-Bus wird vor allem als Feldbus oder Controllerbus eingesetzt, um Daten zwischen verschiedenen, räumlich getrennten (einige Meter) Sensor- und Aktorstationen auszutauschen. Ein CAN-Netzwerk kann im Prinzip mit beliebig vielen Teilnehmern aufgebaut werden und bietet Übertragungsraten von bis zu 1 Mbit/s. Das Übertragungsmedium des Busses wird erst durch den Physical Layer-Baustein bestimmt – zumeist sind es 2-Draht Kupferleitungen.

Buskonzept

Im Gegensatz zu LAN oder WAN-Netzwerklösungen werden bei CAN nicht Pakete von einem Teilnehmer zu einem anderen verschickt (also adressiert), sondern die entsprechenden Datenpakete mit einer Nachrichtennummer (Message-ID) versehen. Jede Message ID steht für eine bestimmte (Meß-)Größe (z. B. Temperatur1, Drehzahl3, ...), und jeder Netzteilnehmer kann selbst entscheiden, welche Pakete mit welcher ID für ihn interessant sind und welche nicht. Dieser nachrichtenorientierte Datenaustausch hat den Vorteil, dass ein Sender nicht von vorneherein wissen muss, an wen er die Daten schicken soll. Auch eine Priorisierung ist vorgesehen: niedrigere Message-IDs sind höherprior als höhere.

Media Access Control

Da bei einem Bus immer nur ein Teilnehmer senden darf, ist auch bei CAN der Zugriff auf den Bus geregelt: Es wird ein modifiziertes CSMA/CD (Carrier Sensing Multiple Access with Collision Detection) verwendet, das bedeutet: Jeder Teilnehmer lauscht bevor er zu senden beginnt am Bus, ob nicht gerade ein anderer sendet. Ist dies der Fall, so muss dieser Teilnehmer warten, ansonsten kann er mit der Sendung beginnen.

Was ist allerdings, wenn 2 Teilnehmer gleichzeitig zu senden beginnen? Prinzipiell ist zu bemerken, dass sich ein LOW-Signal am Bus gegenüber einem HIGH-Signal durchsetzt. Während der Sendung kontrolliert der Teilnehmer, ob auch wirklich das, was er senden möchte, am Bus steht. Wenn nun ein Teilnehmer ein Paket mit einer niedrigeren Message ID verschickt als ein anderer, so setzt sich, da die Message ID gleich am Anfang des Paketes steht, jenes mit

der niedrigeren ID durch. Dieser merkt auch gar nicht, dass ein anderer ebenfalls senden wollte. Der andere Teilnehmer hat jedoch bemerkt, dass die am Bus stehende Message-ID nicht seine ist und bricht den Sendevorgang ab, um ihn später zu wiederholen.

CAN - Pakete

CAN-Pakete enthalten die Message ID, 1 bis 8 Datenbytes und mehrere Status Flags, unter anderem zur Sicherung der Übertragung.

Es ist möglich, dem C167 mitzuteilen, welche Message-IDs er empfangen soll und welche nicht. Dies geschieht mittels 15 CAN Message Objects, für welches jeweils eine zu empfangende Message ID festgelegt wird (mit Ausnahme von Message Object 15, welches im Prinzip alle anderen CAN-Pakete empfängt). In den Message Objects werden empfangene Datenpakete der zugewiesen Message ID laufend gespeichert und können jederzeit abgeholt werden.

Zählereinheit

Die Zählereinheit GPT besteht aus 2 Blöcken (Block 1 und 2), die enthaltenen Timer können für verschiedenste Anwendungen eingesetzt werden:

Anwendungen

Die Zähler dienen vor allem für:

- periodischen Taskaufruf
- Erzeugung periodischer Signale
- periodische Abtastung von Signalen
- Zählen externer Impulse oder Signalwechsel
- Frequenzmultiplikation

Betriebsmodi

Die Zählereinheiten setzen sich aus 3 (Block 1) bzw. 2 (Block 2) 16-Bit Einzelzählern zusammen, wobei jeweils einer der Hauptzähler ist (Core Timer). Jeder dieser 5 Zähler kann entweder für sich alleine oder in Kombination (z. B. kaskadiert) betrieben werden. Die Auflösung der Zähler reicht je nach Einstellung und Timer von 50 ns bis ca. 100 µs, die maximale Periode ohne Kaskadierung liegt damit bei 6,71 s (alle Werte bei 40 MHz Taktfrequenz).

Jeder einzelne Zähler hat mehrere prinzipielle Betriebsmodi:

TIMER: In diesem Modus zählt der Zähler den ausgewählten internen Takt, bis er 0 erreicht. Der Startwert, die Zählrichtung und die Taktfrequenz können dabei vorher gewählt werden.
GATED TIMER: Dieser Modus ist dem Timer-Modus ähnlich. Nur wird hier der Zählvorgang durch den Zustand am entsprechenden I/O-Pin angehalten oder nicht, die I/O Leitung dient dabei als Schalter. Der Zustand ist ebenfalls wählbar (z. B. Zähler läuft nur bei Low am Pin).

COUNTER: Dieser Modus zählt externe Ereignisse am entsprechenden I/O-Pin. Hier ist einstellbar, ob positiver, negativer oder beide Signalwechsel gezählt werden sollen.
INKREMENTAL MODE: Dieser Modus ist speziell beim Einsatz in der Robotertechnik relevant. Mit ihm können vom Inkrementalgeber erzeugte Inkrementalgebersignale dekodiert werden. Der Modus ist vergleichbar mit dem Counter-Mode, nur dass durch Vorliegen von 2 phasenverschobenen Impulssignalen die Drehrichtung und damit die Zählrichtung vorgegeben wird. Dieser Modus ist nicht für alle Timer verfügbar.

Beispiel

Der Timer T2 soll benutzt werden, um zwei Inkrementalsignale zu dekodieren. Die beiden phasenverschobenen Signale, welche von einem Inkrementalencoder generiert werden, sind an den I/O-Pins P2IN und P2EUD angeschlossen.

Folgende Register gehören zu T2:
GPT_T2CON: Control Register zum Setzen des Modus und Konfiguration des Timers.
GPT_T2: Timer-Datenregister. Dieses Register enthält den aktuellen Zählerwert.
GPT_T2IC: Interrupt Control Register. Beim Erreichen des Wertes 0 im Timerregister T2 kann ein Interrupt ausgelöst werden.

Im Register GPT_T2CON wird der Modus gesetzt:

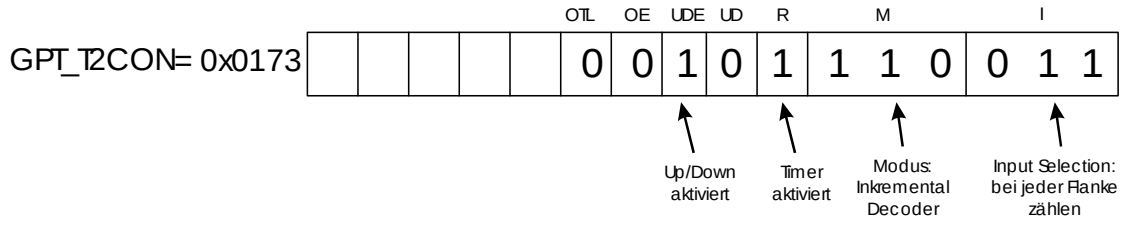

Abbildung 4.13: Beispielkonfiguration für Timer 2

Das Register GPT_T2 enthält den aktuellen Zählerstand. Je nach Drehrichtung des Inkrementalencoders wird der Wert erhöht oder erniedrigt.

Capture/Compare Einheit 6 für PWM-Generierung

Die Capture/Compare Einheit 6 wurde speziell zur Generierung von PWM-Signalen und weiteren Funktionen für Motoransteuerung eingeführt. Sie bietet 4 unabhängige Kanäle mit folgenden Einsatzmöglichkeiten:

Anwendungen

Die Erzeugung von PWM-Signalen kann eingesetzt werden für:

- Ansteuerung von Gleichstrommotoren
- Ansteuerung von Wechselstrommotoren
- Erzeugung von Sprachsignalen

- Allgemein einsetzbar für gleichspannungsgesteuerte Komponenten

Problematik mit externem Speicher

Im Zusammenhang mit Einsatz von externem Speicher im demultiplexed Mode ist die CAPCOM6 problematisch, da die PWM-Ausgabesignale ebenso wie der Adressbus als Alternativfunktionen des Port 1 vorliegen. Als Abhilfe ist hier lediglich die Nutzung einer anderen CAPCOM Einheit zur PWM-Generierung oder die Konfiguration des externen Buses als gemultiplexten Adress/Datenbus möglich. Diese Betriebsart ist allerdings deutlich langsamer.

4.3 Design von Tinyphoon

Tinyphoon (www.tinyphoon.com) ist der kleinste vollständig autonome Roboter mit eingebauter Kamera der Welt. Er ist ein Zweirad getriebener Roboter, mit den Maßen eines Würfels mit 75mm Kantenlänge. Für die Architektur des Roboters wurde ein neuartiges Konzept ausgearbeitet.

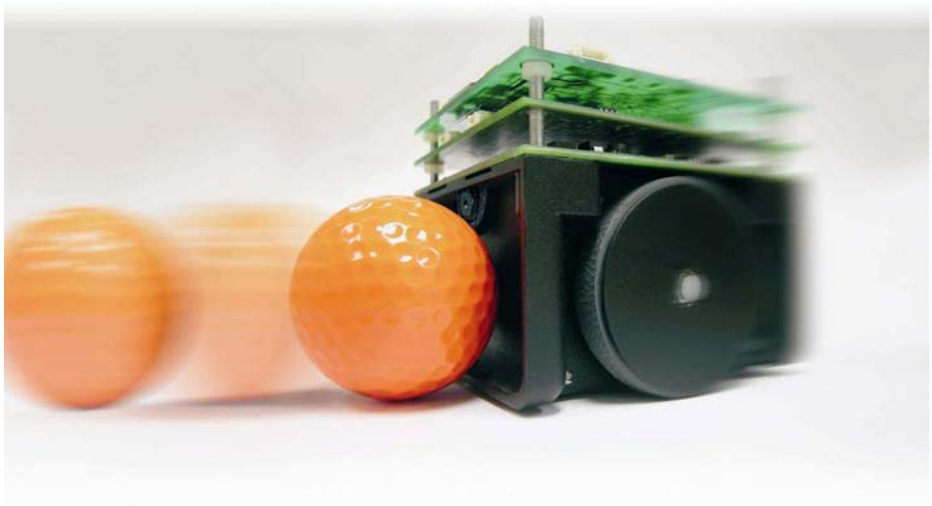

Abbildung 4.14: Tinyphoon in Aktion

Dieses neuartige Konzept für autonome mobile Roboter vereinfacht deren Aufbau durch ein modulares Design. Gleich der Architektur eines PCs unterstützt es die Integration bestehender Komponenten. Ähnlich dem Plug and Play-Prinzip können Roboter mit weiteren Komponenten ausgestattet werden, welche über einen Bus miteinander verbunden sind, um dessen Funktionalität zu erweitern und zu verbessern. Zur Kommunikation zwischen den Komponenten wird eine eigene Kommunikations- und Konfigurationssprache entwickelt. Dieses Konzept kann auf verschiedenste mechanische Plattformen angewandt werden.

Diese Struktur repräsentiert ein Schichtenmodell, wobei die unterste Schicht die Bewegungseinheit darstellt. Der Roboter ist in ein globales Kommando und Informationssystem eingebettet. Die Verbindung zum übergeordneten Kommando und Informationssystem erfolgt über die ‚High-Level-Control-Unit'.

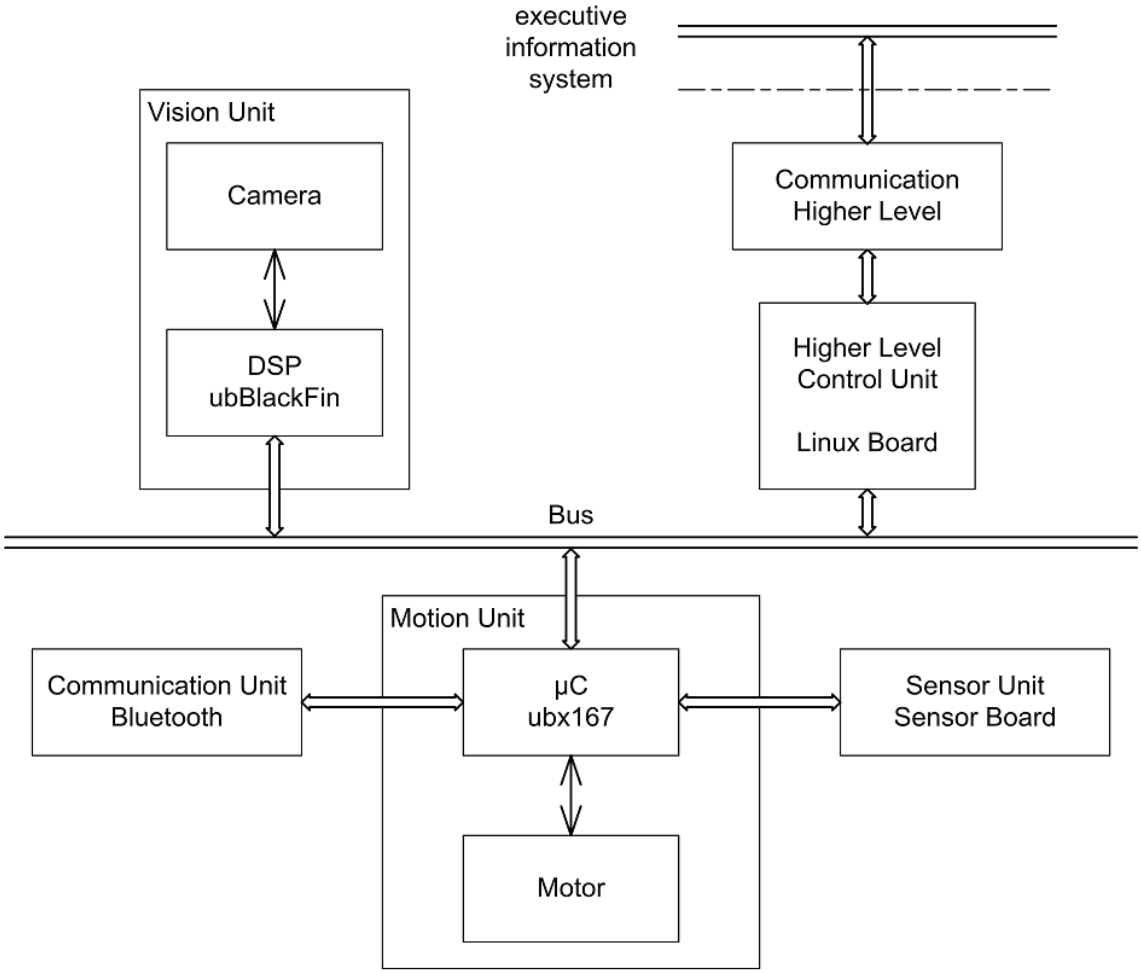

Abbildung 4.15: Komponentenaufbau des Tinyphoons

Tinyphoon besteht aktuell aus den folgenden Modulen:

1) Bewegungsmodul
   Das Bewegungsmodul ist für die Bewegung des Roboters verantwortlich und bildet eine abgeschlossene Einheit, welche über ein Bussystem angesteuert wird. Ihm zugeordnet sind Bewegungssensoren (Beschleunigungssensoren, Drehwinkelsensoren und Drehzahlmesser)

2) Kamerasystem
   Das Kamerasystem basierend auf Mini-CMOS Kameras und einem Blackfin DSP von Analog Devices dient zur Ortung von bestimmten Objekten.

3) Strategiemodul
   Dieses Modul dient zur Entscheidungsfindung.

4) Weitere intelligente Sensormodule

- Erdmagnetfeldsensor

Die Module des Tinyphoon werden über ein Bussystem miteinander vernetzt. Mit dieser zur Verfügung stehenden Hardware ist Tinyphoon in der Lage, selbständig auf einem fix definierten Feld Fußball zu spielen.

Jedoch für den Einsatz in einem MIROSOT Team ist die Bewegungseinheit alleine vollkommen ausreichend. In weiterer Folge dieser Arbeit wird explizit auf den Aufbau der Bewegungseinheit eingegangen.

4.4 Mechanik

Zum besseren Verständnis sei hier kurz der mechanische Aufbau der Bewegungseinheit erläutert. Er kann in folgende Komponenten geteilt werden:

- Chassis
- Räder und Antrieb
- Batteriefach
- Elektronikhalterung

Chassis

Tinyphoon ist gemäß den Anforderungen für Roboterfussball der Klasse MIROSOT insgesamt etwa 75mm x 75mm x 75mm groß. Das 40 mm hohe Chassis wurde aus glasfaservertärktem Kunststoff gefertigt. Die Radachsen sind an beiden Seitenwände gelagert. Weiters hat der Roboter an Front- und Heckwand seitliche Ballführungsschienen, sowie eine Bodenschiene welche zudem jeweils ein Stützrädchen hält.

Räder und Antrieb

Zwei Kunststoffräder mit ca. 1.5mm Spezialbelag und insgesamt 46mm Durchmesser werden unabhängig voneinander gelagert und angetrieben. Jedes Rad wird von jeweils einem eigenen DC-Motor bewegt, das Übersetzungsverhältnis der einstufigen Übersetzung beträgt 0,12. Das an der Motorachse befindliche Zahnrad mit 12 Zähnen und einem Modul von 0,3mm greift dabei direkt auf dem an der Innenseite des Rades befindlichen Innenzahnrad mit 100 Zähnen.

Batteriefach

An der Unterseite des Chassis befindet sich das Fach für das Batterien-Pack. Das Batterie-Pack besteht aus 9 fest verlöteten Batterien im AAA-Format, mittels einfach einsetzbarer und entfernbarer Befestigungsstange hält das Pack im Fach.

Elektronikhalterung

Sämtliche Elektronikmodule werden oberhalb des Chassis durch 4 Trägerstangen gehalten, entsprechende Bohrungen in den Platinen müssen vorgesehen sein. Mittels Abstandhaltern wird

ein direkter Kontakt der einzelnen Platinen vermieden, die Platinen sind durch Steckverbindungen direkt miteinander verbunden.

4.5 Elektronik

Die Dimensionierung und Auswahl der erforderlichen Elektronik basiert vorrangig auf den im Pflichtenheft vorgegebenen Eigenschaften und in weiterer Folge von der mechanischen Konstruktion.

Batterien

Die Auswahl und Dimensionierung der Batterien haben bei jedem mobilen Gerät eine besondere Bedeutung: Relevant sind Batterietechnologie, Nennspannung und Gesamtkapazität.

In Frage kommen ausschließlich Batterie-Technologien mit Wiederbeladung, Einweg-Batterien sind aus umwelttechnischen und wirtschaftlichen Gründen auszuschließen.

Drei Technologien sind derzeit in mobilen Geräten weit verbreitet: Nickel-Cadmium, Nickel-Metallhydrid und Lithium-Ionen Akkumulatoren. Für den Einsatz im mobilen Roboter eignen sich am besten NiMH-Akkumulatoren, da diese mit entsprechendem Ladegerät relativ kurze Ladezeiten besitzen und weiters hochstrombelastbar sind. Die Hochstrombelastung ist vor allem beim Beschleunigen und Bremsen beim Motor gegeben.

Als Energieversorgung für Tinyphoon wurden 9 in Serie geschaltete Nickel-Metallhydrid-Zellen, Type Panasonic PRO+ mit jeweils 1,2 Volt Nennspannung und 750mAh Kapazität gewählt. Damit ergibt sich eine Gesamtversorgungsspannung von etwa 10,8 Volt.

Motoren

Für die Auswahl der Motoren gelten folgende Kriterien:

- Technologie: Grundsätzlich kommen nur Gleichstrommotoren in Frage. Die an den Motor gelieferte Spannung hängt des weiteren vom Motortreiber ab, das bedeutet, an den Motor wird nicht die volle Versorgungsspannung abgegeben. Die Nennspannung für den DC-Motor sollte daher zwischen 5V und 10,8V liegen.
- Maximale Ausmaße: Der Motor darf auf Grund der mechanischen Gegebenheiten nicht mehr als 40 mm lang sein und einen maximalen Außendurchmesser von 23mm haben.
- Impulsgeber: Vorteilhaft wäre es, wenn der Motor einen Impulsgeber zur Messung der Drehzahl bereits integriert hat. Ansonsten ist ein externer und platzintensiver Impulsgeber erforderlich.

Als Radantrieb wurden DC-Motoren der Type Minimotor 224U012SR gewählt. Diese Motoren haben 6V Nennspannung und erfüllen mit 25mm Länge und 22mm Durchmesser auch die

mechanischen Anforderungen. Im Betrieb wird der Motor mit einer Überspannung von 10,1V betrieben.

Im Motor integriert ist ein induktiver Inkrementalenkoder mit 2 Kanälen und 512 Impulsen pro Motorumdrehung je Kanal. Mit dem 2-kanaligen Impulsgeber kann nicht nur die Drehzahl, sondern auch die Drehrichtung festgestellt werden.

Motortreiber

Die Motoren benötigen wesentlich mehr Strom, als ein Digitalausgang des Prozessors liefern kann. Deshalb muss mit dem Motorsteuersignal ein Transistor angesteuert werden, welcher den Motor mit Strom versorgt. Üblich ist heute der Einsatz von MOS-Transistoren mit TTL-Eingang, d.h. die Steuerleitung des Prozessors kann direkt an den „Eingang“ des Transistors (Basis bei Bipolartransistor bzw. Gate bei MOS-Transistor) angeschlossen werden. MOS-Technologie hat gegenüber Bipolartechnik den Vorteil, dass die Stromverluste und damit die

Zusätzlich muss die Leistungsbeschaltung des Motors mit stromstarken und schnellen Dioden versehen sein, um die kurzzeitigen „negativen“ Ströme beim Umschalten der induktiven Last abzufangen.

Anstelle von einzelnen Transistoren kann auch ein integrierter Leistungstreiber eingesetzt werden. Diese benötigen weniger Platinenfläche und haben oft zusätzliche Konfigurations- und Steuermöglichkeiten, natürlich auf TTL-Pegel.

Für die Ansteuerung der Motoren von Tinyphoon wurde der Einchip-Leistungstreiber L298 von STM (SGS-Thompson Microelectronics) gewählt. Dieser enthält Leistungstreiber für 2 Gleichstrommotoren (2 Kanäle) und kann pro Motor bis zu 2 Ampere dauerhaft schalten. Die maximal erlaubte Spannung beträgt 46V.

Prozessor und Speicher

Als zentraler Prozessor der Bewegungseinheit kommt bei Tinyphoon der 16 Bit Mikrocontroller xc167CI von Infineon zum Einsatz. Dieser bietet sämtliche erforderliche Peripherieeinheiten wie asynchrone serielle Schnittstelle, Timer und PWM-Einheit mit 4 Kanälen. Die Skalierbarkeit ist aufgrund einer synchronen Schnittstelle, des integrierten CAN-Controllers und der möglichen Anbindung von bis zu 5 unterschiedlichen externen Speicherbausteinen sehr gut. Auch die Leistungsfähigkeit liegt im erforderlichen Bereich von etwa 40 MIPS (Mega Instructions per Second) bei 40 MHz Systemtakt.

Da der xc167 zwar 128kByte internes Flash-ROM, jedoch nur ein sehr kleines internes RAM besitzt, ist ein externer RAM-Baustein erforderlich. Aus Platz und Kostengründen wird ein 16 Bit SRAM mit 256Kx16Bit eingesetzt. Der externe Datenbus des xc167 ist daher im 16-Bit Modus konfiguriert.

Der Einsatz des xc167 als Controller in mobilen Robotern ist im nächsten Kapitel ausführlich behandelt.

Funkmodul

Zur Kommunikation mit einem Hauptrechner kommt bei Tinyphoon eine Bluetooth-Funkübertragung zum Einsatz. Diese ermöglicht eine bidirektionale Kommunikation. Details hierzu finden sich in der Diplomarbeit von Oliver Fink [Fink03].

Baugruppen

Da bei der Realisierung von Tinyphoon auf Modularität und Flexibilität besonderer Wert gelegt wird, teilt sich selbst die Elektronik der Bewegungseinheit in 2 Baugruppen/Platinen. Die beiden Platinen sind durch einen 60poligen Stecker direkt verbunden, Versorgungsspannung, sowie sämtliche Steuer- und Datenleitungen werden über diese Verbindung übertragen.

Mikrocontrollerplatine

Die Mikrocontrollerplatine beherbergt den Mikrocontroller, Flash-Speicher, Spannungswandler, Pegelwandler für die asynchrone serielle Schnittstellen, CAN-Transceiver, DIP-Schalter und Hardwarekonfiguration sowie die bereits genannten Stecker. Die Mikrocontrollerplatine ist so konzipiert, dass sie als Starterkit Verwendung finden kann.

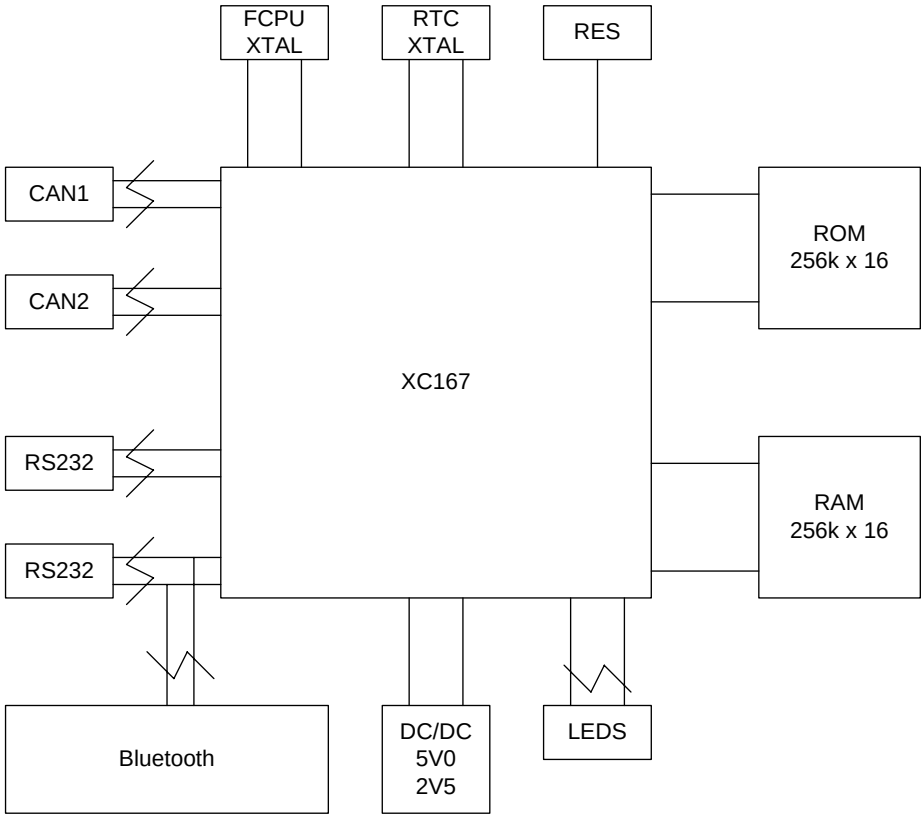

Abbildung 4.16: Mikrocontroller Blockschaltbild

Platinenlayout

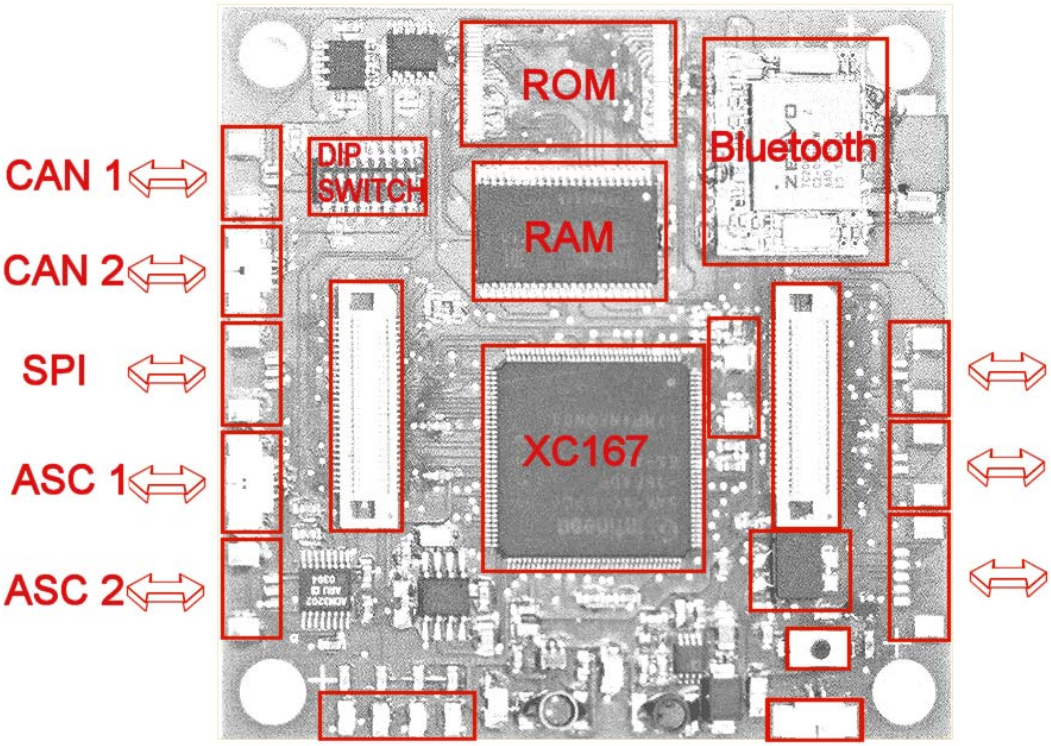

Abbildung 4.17: Die Tinyphoon Mikrocontrollerplatine ubx167

Pinbelegung xc167 – Sensorplatine

In weiterer Folge wird die Pinbelegung des xc167er angeführt, sowie die von der Sensorplatine verwendeten Pins (fett markiert).

Pin	Port	I/O	Description	Details	Plug
1	NC				
2	NC				
3	P6.20	I/O	RSTOUT	Ist im aktiven Zustand ‚low'. Ist mit Flash und Bluetooth verbunden. VCC for LED2 and LED3	P2 – 59
4	NMI	I	NMI	1K auf 5V0	P1 – 01
5	VSSP		GND		

6	VDDP		VC5VO		
7	P6.0	O, I/O	CS0, CC0IO??		P1 – 60
8	P6.1	O, I/O	CS1, CC1IO??	CS for SRAM	P1 – 02
9	P6.2	O, I/O	CS2, CC2IO	CS for Flash	P1 – 59
10	P6.3	O, I/O	CS3, CC3IO	**(GPIO C)**	P1 – 03
11	P6.4	O, I/O	CS4, CC4IO	**(GPIO B)**	P1 – 58
12	P6.5	I, I/O	HOLD, CC5IO	**(GPIO A)**	P1 – 04
13	P6.6	I/O	HLDA	LED 2	P1 – 57
14	P6.7	O, I/O	BREQ, CC6IO	LED 3	P1 – 05
15	P7.4		CAN2_RxD	CAN2 über DIPSwitch	P1 – 56
16	P7.5		CAN2_TxD	CAN2 über DIPSwitch	P1 – 06
17	P7.6		CAN1_RxD	CAN1 über DIPSwitch	P1 – 55
18	P7.7		CAN1_TxD	CAN1 über DIPSwitch	P1 – 07
19	VSSP		GND		
20	VDDP		VC5VO		
21	P9.0	I/O	SDA0	IIC Bus Data Line **(PWM Enable; IC2 Pin6; EN A)**	P1 – 54
22	P9.1	I/O	SCL0	IIC Bus Clock Line **(PWM Enable; IC2 Pin11; EN B)**	P1 – 08
23	P9.2	I/O	SDA1	**(Appl. LED1)**	P1 – 53
24	P9.3	I/O	SCA1	**(Appl. LED2)**	P1 – 09
25	P9.4	I/O	SDA2	**(Appl. LED3)**	P1 – 52
26	P9.5	I/O	SCL2	**(Appl. LED4)**	P1 – 10
27	VSSP		GND		
28	VDDP		VC5VO		
29	P5.0	I	AN0	**(Appl. DIPswitch, Pin1)**	P1 – 51
30	P5.1	I	AN1	**(Appl. DIPswitch, Pin2)**	P1 – 11
31	P5.2	I	AN2	**(Appl. DIPswitch, Pin3)**	P1 – 50
32	P5.3	I	AN43	**(Appl. DIPswitch, Pin4)**	P1 – 12
33	P5.4	I	AN4	**(Appl. DIPswitch, Pin5)**	P1 – 49
34	P5.5	I	AN5	**(Appl. DIPswitch, Pin6)**	P1 – 13
35	P5.10	I	AN10	**(frei)**	P1 – 48
36	P5.11	I	AN11	**(frei)**	P1 – 14
37	P5.8	I	AN8	**(frei)**	P1 – 47
38	P5.9	I	AN9	**(frei)**	P1 – 15
39	P5.6	I	AN6	**(Appl. DIPswitch, Pin7)**	P1 – 46
40	P5.7	I	AN7	**(Appl. DIPswitch, Pin8)**	P1 – 16
41	VAREF				P1 – 45
42	VAGND	I			P1 – 17
43	P5.12	I	AN12	**(frei)**	P1 – 44
44	P5.13	I	AN13	**(frei)**	P1 – 18
45	P5.14	I	AN14, T4EUD	**(Appl. Enkoder2; Pin2 auf P5)**	P1 – 43
46	P5.15	I	AN15, T2EUD	**(Appl. Enkoder1; Pin1 auf P7)**	P1 – 19
47	VSSI		GND		
48	VDDI	I/O	VC2V5		
49	P2.8	I/O	EX0IN, CC8IO	**(Appl. PWM; IC2 Pin5; IN1)**	P1 – 42
50	P2.9	I/O	EX1IN, CC9IO	**(Appl. PWM; IC2 Pin7; IN2)**	P1 – 20
51	P2.10	I/O	EX2IN, CC10IO	**(Appl. PWM; IC2 Pin10; IN3)**	P1 – 41
52	P2.11	I/O	EX3IN, CC11IO	**(Appl. PWM; IC2 Pin12; IN4)**	P1 – 21
53	P2.12	I/O	EX4IN, CC12IO	**(Appl. Beschleunigungssensor; IC4 Pin10; X_OUT1)**	P1 – 40
54	P2.13	I/O	EX5IN, CC13IO	**(Appl. Beschleunigungssensor; IC4 Pin9; Y_OUT1)**	P1 – 22
55	P2.14	I/O	EX6IN, CC14IO	**(Appl. Beschleunigungssensor; IC5 Pin10; X_OUT2)**	P1 – 39

56	P2.15	I/O	EX7IN, CC15IO	**(Appl. Beschleunigungssensor; IC5 Pin9; Y_OUT2)**	P1 – 23
57	TRST		JTAG	1K auf GND + JTAG Pin 1	P2 – 33
58	VDDP		VC5V0		
59	P3.0		TxD1	Bluetooth über DIP + MAX202 über DIP	P1 – 38
60	P3.1		RxD1	Bluetooth über DIP + MAX202 über DIP	P1 – 24
61	P3.2	I		**(frei)**	P1 – 37
62	P3.3	O		**(frei)**	P1 – 25
63	P3.4	I		**(frei)**	P1 – 36
64	P3.5	I	T4IN	**(Appl. Enkoder 2; Pin1 auf P5)**	P1 – 26
65	P3.6	I	T3IN	**(Appl. Taster)**	P1 – 35
66	P3.7	I	T2IN	**(Appl. Enoder 1; Pin2 auf P7)**	P1 – 27
67	P3.8		MRST0	SSC0 **(SPI AD-Wandler)**	P1 – 34
68	P3.9		MTSR0	SSC0 **(SPI AD-Wandler)**	P1 – 28
69	P3.10		TxD0	MAX202 über DIP	P1 – 33
70	P3.11		RxD0	MAX202 über DIP	P1 – 29
71	TCK		JTAG	JTAG Pin 2	P2 – 28
72	TDI		JTAG	JTAG Pin 3	P2 – 34
73	TDO		JTAG	JTAG Pin 4	P2 – 27
74	TMS		JTAG	JTAG Pin 5	P2 – 35
75	P3.12		BHE	Memory High Byte Enable	P2 – 26
76	P3.13		SCLK0	SSC0 **(SPI AD-Wandler)**	P1 – 32
77	P3.15		CLKOUT/F		P2 – 36
78	VDDI		VC2V5		
79	VSSI		GND		
80	P4.0	O	A16	Address bus	P2 – 25
81	P4.1	O	A17	Address bus	P2 – 37
82	P4.2	O	A18	Address bus	P2 – 24
83	P4.3	O	A19	Address bus	P2 – 38
84	P4.4	O	A20	Address bus	P2 – 23
85	P4.5	I/O	A21	**(frei)**	P2 – 39
86	P4.6	I/O	A22	**(frei)**	P2 – 22
87	P4.7	I/O	A23	**(frei)**	P2 – 40
88	VDDP		VC5V0		
89	VSSP		GND		
90	P20.0		_RD	Memory	P2 – 21
91	P20.1		_WR/WRL	Memory	P2 – 41
92	P20.2		READY	Memory	P2 – 20
93	P20.4		ALE	Memory	P2 – 42
94	P20.5		EA	Memory	P2 – 19
95	P0L.0		D0	Data	P2 – 43
96	P0L.1		D1	Data	P2 – 18
97	P0L.2		D2	Data	P2 – 44
98	P0L.3		D3	Data	P2 – 17
99	P0L.4		D4	Data	P2 – 45
100	P0L.5		D5	Data	P2 – 16
101	P0L.6		D6	Data	P2 – 46
102	P0L.7		D7	Data	P2 – 15
103	VDDP		VC5V0		
104	VSSP		GND		
105	P0H.0		D8	Data	P2 – 47
106	P0H.1		D9	Data	P2 – 14
107	NC				
108	NC				
109	NC				

110	NC				
111	P0H.2		D10	Data	P2 – 48
112	P0H.3		D11	Data	P2 – 13
113	P0H.4		D12	Data	P2 – 49
114	P0H.5		D13	Data	P2 – 12
115	P0H.6		D14	Data	P2 – 50
116	P0H.7		D15	Data	P2 – 11
117	P1L.0		A0	Address bus	P2 – 51
118	P1L.1		A1	Address bus	P2 – 10
119	P1L.2		A2	Address bus	P2 – 52
120	P1L.3		A3	Address bus	P2 – 09
121	P1L.4		A4	Address bus	P2 – 53
122	P1L.5		A5	Address bus	P2 – 08
123	P1L.6		A6	Address bus	P2 – 54
124	P1L.7		A7	Address bus	P2 – 07
125	VDDP		VC5V0		
126	VSSP		GND		
127	P1H.0		A8	Address bus	P2 – 55
128	P1H.1		A9	Address bus	P2 – 06
129	P1H.2		A10	Address bus	P2 – 56
130	P1H.3		A11	Address bus	P2 – 05
131	P1H.4		A12	Address bus	P2 – 57
132	P1H.5		A13	Address bus	P2 – 04
133	P1H.6		A14	Address bus	P2 – 58
134	P1H.7		A15	Address bus	P2 – 03
135	VDDI		VC2V5		
136	VSSI		GND		
137	XTAL2		Crystal	Quarz 8MHz	
138	XTAL1		Crystal	Quarz 8MHz	
139	VSSI				
140	XTAL3		Crystal	RTC Quarz 32kHz	
141	XTAL4		Crystal	RTC Quarz 32kHz	
142	RSTIN			Ein ‘low’ am Eingang für 100ns + 2 x CPU clock cycles resetet den XC167; Restbaustein	P2 – 02
143	BRKOUT		Debug Break	Debug Break Pin1	P2 – 60
144	BRKIN		Debug Break	Debug Break Pin2	P2 – 01

Tabelle 4.2: Mikrocontroller Pinbelegung

Sensorplatine

Auf der Sensorplatine sind Motortreiber, Funkmodul, Spannungswandler, Beschleunigungssensoren, Erdmagnetfeldsensor, Drehwinkelsensor sowie Steckverbinder zu den beiden Motoren und der Mikrocontrollerplatine zu finden.

4.6 Software

Die Software der Bewegungseinheit des Roboters bestimmt weitestgehend seine Fähigkeiten. Im folgenden sind die Kernaufgaben der Software der Tinyphoon Bewegungseinheit beschrieben.

Aufgaben der Software

Abbildung 4.18 zeigt die 6 Kernaufgaben der Bewegungseinheit und ihre Abhängigkeiten:

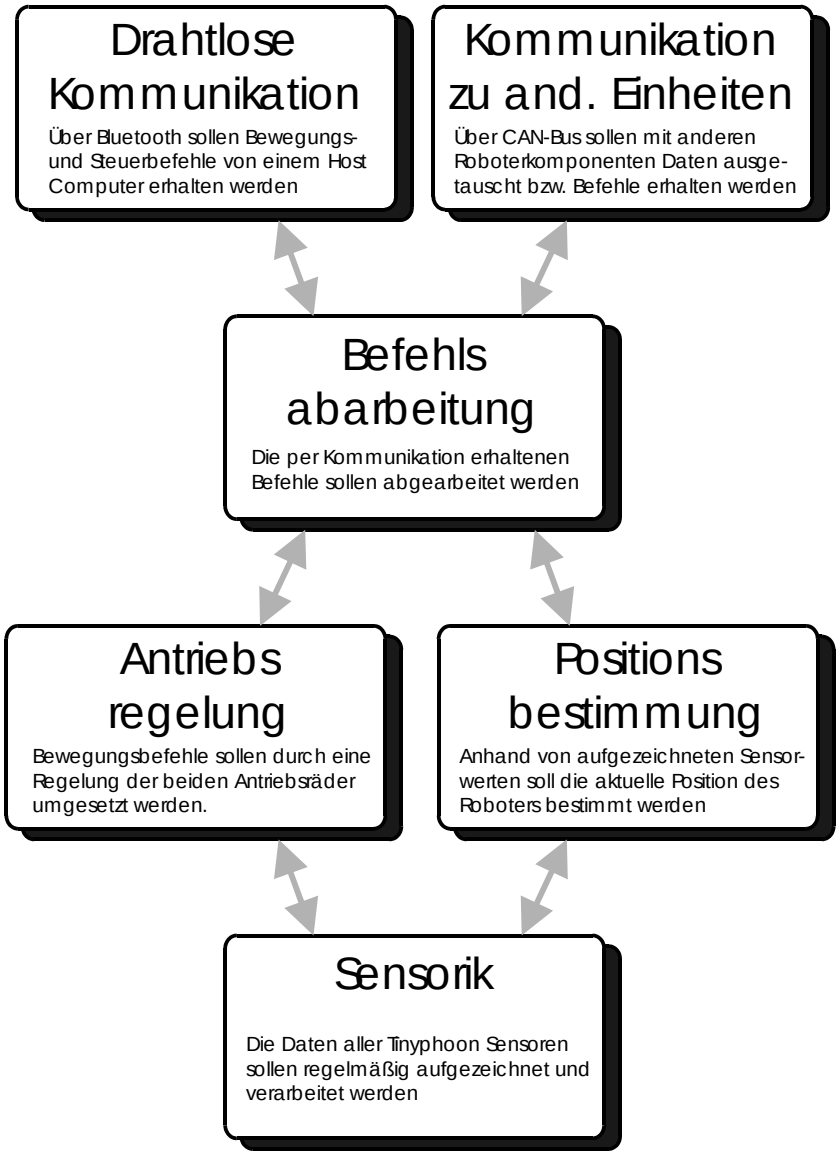

Abbildung 4.18: Aufgaben der Software der Tinyphoon Bewegungseinheit

Drahtlose Kommunikation

Tinyphoon soll in der Lage sein, Bewegungs-, aber auch beliebige andere Befehle von einem Host Computer zu empfangen und auszuführen. Basis für diese Funktion ist die drahtlose

Kommunikation über Funk (Bluetooth). Die Kommunikation soll so ausgelegt sein, dass mehrere grundsätzlich von einander unabhängige Roboter die gleichen Befehle erhalten können (Broadcast). Es soll aber auch möglich sein, nur einzelne Roboter gezielt mit Befehlen zu versorgen.

Kommunikation zu anderen Robotereinheiten

Neben der drahtlosen Kommunikation zu einem Hostcomputer soll Tinyphoon auch in der Lage sein, mit allen anderen Robotereinheiten, die über einen Prozessor verfügen, zu kommunizieren. Dabei soll es wiederum möglich sein, entweder mit einer bestimmten Robotereinheit zu kommunizieren oder Nachrichten an alle Einheiten zu schicken.
Basis für diese Funktionalität ist die CAN-Bus Anbindung des xc167.

Befehlsabarbeitung

Die entweder vom Hostcomputer oder von einer anderen Robotereinheit erhaltenen Befehle sollen interpretiert und ausgeführt werden. Der Roboter muss dabei erkennen, ob der Befehl tatsächlich an ihn adressiert ist oder nicht.

Antriebsregelung

Der Roboter soll in der Lage sein, eine vorgegebene Geschwindigkeit und Richtung zu erreichen und einzuhalten. Dazu ist eine Antriebsregelung erforderlich.
Für die Antriebsregelung werden einerseits die aktuellen Raddrehzahlen und andererseits die Messwerte der Beschleunigungssensoren herangezogen. Hierbei ist eine Antischlupfregelung inkludiert. Der Regler ermittelt die PWM Signale für die Motoren.

Positionsbestimmung

Die Positionsbestimmung des Roboters stützt sich auf die vom Host Computer übermittelten Daten. Um eine bessere Genauigkeit der vom Hostcomputer ermittelten Positionsdaten zu erhalten, wird nicht die gerade gemessene Position übertragen, sondern eine etwas ältere, die dadurch korrigiert wird, dass Messwerte zeitlich gesehen vor diesem Wert, sowie nach diesem Wert zur Schätzung herangezogen werden. Dadurch ist natürlich die übertragene Position noch älter als das zuletzt aufgenommene Bild. Die Aufgabe des Positionsbestimmungsalgorithmus besteht nun darin, die Position des Roboters zu bestimmen, zu welchem der Roboter erstmals reagieren kann. Dies geschieht über die durch die Sensorintegration ermittelten Bewegungsdaten.

Sensorik

Die am Sensorboard enthaltenen Sensoren (2x Beschleunigungssensoren, Erdmagnetfeldsensor, Drehwinkelsensor und 2x Enkoder) ermitteln die Relativbewegung des Roboters. Für die Digitalisierung der Messwerte des Erdmagnetfeldsensors und des Drehwinkelsensors ist am Sensorboard ein Analog Digitalwandler eingebaut. Dieser kommuniziert über die SPI mit dem xc167er. Diese Sensordaten sollen aufgezeichnet und beispielsweise für Antriebsregelung und Positionsbestimmung weiterverarbeitet werden.

Resultierende Teilaufgaben

Aus den Kernaufgaben für die Tinyphoon Bewegungssteuerung ergeben sich folgende Teilaufgaben:

- Empfang von Befehlen über Bluetooth
- Datenaustausch zu anderen Robotereinheiten über CAN
- Befehlsauswertung und –abarbeitung
- Positionsbestimmung
- Regelschleife
- Regelungsalgorithmus
- Erfassung und Aufzeichnung der Raddrehzahlen
- Erfassung und Aufzeichnen der Beschleunigungssensordaten
- Erfassung und Ansteuerung der Motoren

Daneben ergeben sich eine Reihe von Teilaufgaben, die zwar nicht unmittelbar im Zusammenhang mit den Kernaufgaben stehen, aber dennoch für Diagnose- und Wartungszwecke erforderlich sind:

- Ausgabe von Debugmeldungen an der Diagnoseschnittstelle (ASC0)
- Ansteuerung der LEDs zur Statusanzeige (LED)
- Abfrage des Tasters am Roboter (KEY)
- Speichern von Kalibrierungsdaten im Flash (FLASH)
- Erzeugung einer globalen Zeit (G_TIME)

Im wesentlichen spiegeln diese Teilaufgaben bereits die in Kapitel 4.8 beschriebenen einzelnen Softwaremodule wider.

Theoretische Grundlagen

Aufnahme der Raddrehzahl durch Inkrementalgeber

Die Messung der Raddrehzahl ist bei Kenntnis des exakten Raddurchmessers eine sehr genaue Methode, eine tatsächlich zurückgelegte Strecke zu vermessen.
Die Anzahl der Umdrehungen eines Motors kann im einfachsten Fall durch Generierung eines Impulses pro Umdrehung gemessen werden. Der Impuls kann beispielsweise durch optische oder induktive Signalgeber erzeugt werden. Die folgende Abbildung zeigt schematisch eine Impulssequenz für diese Messmethode, bei der 1 Impuls pro Umdrehung generiert wird.

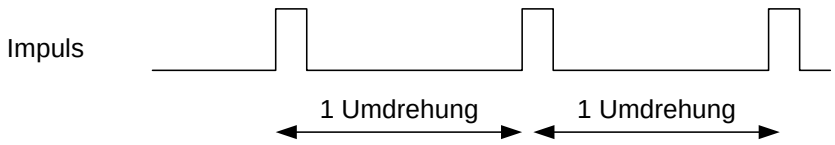

Abbildung 4.19: Einfacher Impuls zur Drehzahlmessung

Ein Nachteil dieser einfachen Methode ist die Genauigkeit: So wird bei langsamen Umdrehungszahlen eine etwaige Unregelmäßigkeit der Bewegung zwischen den Impulsen nicht registriert. Außerdem wird vor Stillstand die letzte Bewegung nicht erkannt. Ein weiterer Nachteil des einfachen Impulses ist die Erkennung der Bewegungsrichtung. Aus dem einzelnen Impuls kann nicht ermittelt werden, in welche Richtung sich der Roboter bewegt.

Durch Einführung einer höheren Anzahl an Impulsen pro Umdrehung kann die Auflösung der Messung erhöht werden. Bewegungsunregelmäßigkeiten werden rascher erkannt. In der folgenden Abbildung werden beispielsweise 7 Impulse pro Umdrehung geliefert.

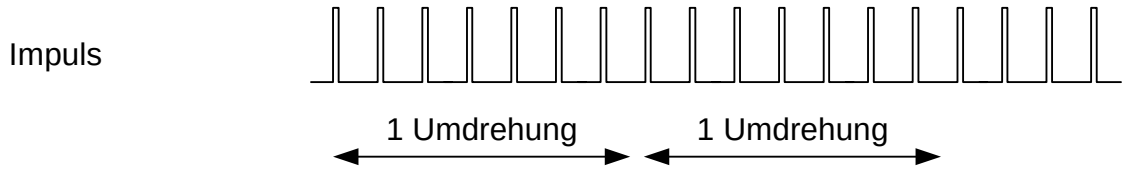

Abbildung 4.20: mehrere Impulse pro Umdrehung zur Drehzahlbestimmung

Damit wurde die Meßgenauigkeit erhöht, eine Geschwindigkeitsänderung wird schneller erkannt. Die Drehrichtung kann allerdings noch immer nicht festgestellt werden.

Um auch die Drehrichtung zu erkennen, wird ein zweiter Impuls eingeführt, welcher gegenüber dem ersten Impuls leicht versetzt ist. Die folgende Abbildung zeigt schematisch die Impulsfolge der beiden Impulse:

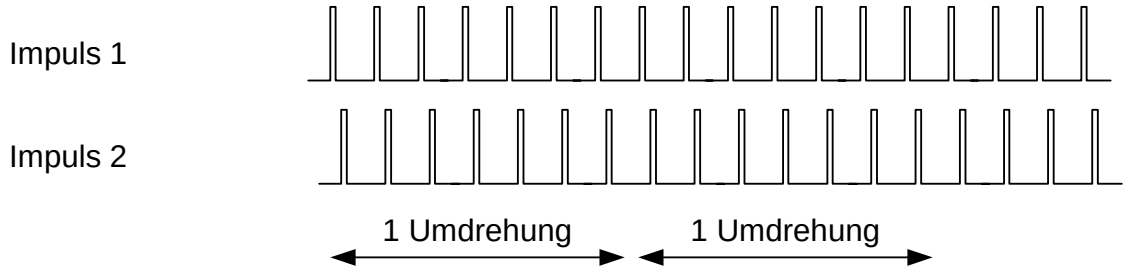

Abbildung 4.21: Inkrementalgebersignal

Gängige Inkrementalgebersensoren arbeiten nach diesem Prinzip. Sie liefern 2 versetzte digitale Impulse, mehrere hundert (z. B. 256 oder 512) pro Umdrehung. Der Prozessor oder eine Peripherieeinheit decodiert das Signal und wandelt es in eine vorzeichenbehaftete Drehzahl um.

Digitale PID – Regler

Der PID – Regler wird eingesetzt, um die notwenigen Motordrehzahlen zu bestimmen, welche den Roboter auf der ihm abschnittsweise vorgegeben Trajektorie hält. Diese Motordrehzahlen

werden über PWM Signale gesteuert. Ein PWM Signal, ein so genanntes ‚Pulse Width Modulated Signal' ist ein zerhacktes Gleichstromsignal. Das Integral über diesem liefert ein Gleichstromäquivalent, also eine entsprechende Gleichspannung, welche der Gleichstrommotor benötigt um seine entsprechende Drehzahl zu erreichen.

Für den Roboter wurden eine Höchstgeschwindigkeit von 2.54 m/s und eine maximale Drehwinkelgeschwindigkeit von 25.4 rad/s festgelegt. Neu Sollwerte für Geschwindigkeit und Winkelgeschwindigkeit werden alle 20ms gesetzt. Detaillierte Informationen finden sich in der Dissertation von Gregor Novak [Novak02].

Jeder Motor ist mit einem digitalen Zweikanal – Enkoder mit einer Auflösung von 512 Impulsen pro Umdrehung ausgestattet. Die Auswertung dieses Zweikanalsignals liefert eine reale Auflösung von 2048 Impulsen pro Umdrehung. Pro Motorumdrehung legt das Rad einen Weg von 18,1mm zurück. Dies bedeutet für eine Höchstgeschwindigkeit von 2.54 m/s eine Auflösung von 287 Impulsen. Dies ist ausreichend, da einer Genauigkeit von 0.01 m/s für die Sollgeschwindigkeit ein Minimum von 254 Impulsen notwenig ist.

Implementiert wurden zwei unabhängige PID – Regler, welche jedes Rad einzeln regeln. Für den Regler wurde ein digitaler rekursiver PID – Regler gewählt [Büttner, Seite 91].

$$u(k) = u(k-1) + g_0 \cdot e(k) + g_1 \cdot e(k-1) + g_2 \cdot e(k-2)$$

wobei:

$$g_0 = K_R \cdot \left(1 + \frac{T_D}{T}\right)$$

$$g_1 = -K_R \cdot \left(1 + \frac{2 \cdot T_D}{T} - \frac{T}{T_I}\right)$$

$$g_2 = K_R \cdot \frac{T_D}{T}$$

$u(k)$	Stellgröße des Schrittes k
$u(k-1)$	Stellgröße des vorherigen Schrittes
$e(k)$	Fehler des Schrittes k
K_R	Verstärkung
T_D	Ableitungszeit
T_I	Integrationszeit
T	Abtastzeit

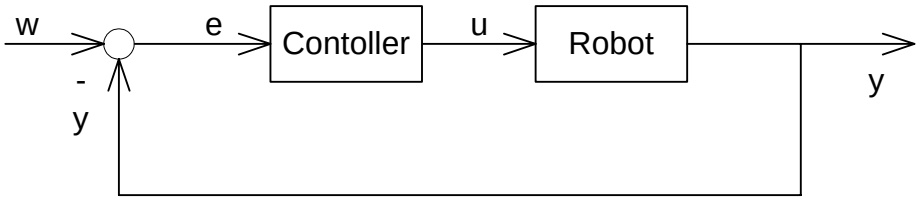

Abbildung 4.22: Grundschema des PID - Reglers

w	Sollgröße
y	Regelgröße

Der Regler berechnet nur die Änderung der Stellgröße. Es ist also notwenig die letzte Stellgröße zu speichern. Das Ergebnis für die Stellgröße muss im Bereich zwischen [-1;1] liegen. Es kann jedoch vorkommen, dass das Ergebnis nicht in diesem Intervall liegt. Dies führt zu einer Limitierung der Stellgröße. Da jedoch der Roboter mit zwei unabhängigen Reglern geregelt wird, muss die jeweils andere Stellgröße entsprechend adaptiert werden. Theoretisch müsste diese Adaptierung die Nichtlinearitäten des Roboters berücksichtigen. Jedoch hat sich gezeigt, dass eine lineare Adaptierung im Betrieb vollkommen ausreichend ist.

Bei Motorregelungen werden Sprünge der Sollgröße nicht direkt auf den Regler aufgetragen. Stattdessen wird eine Startrampe verwendet. Diese Startrampe führt den Regler nur eine Schrittweise Veränderung der Sollgröße zu. Damit wird ein Durchrutschen der Räder verhindert. Die Rampe selbst ist unterschiedlich für eine Steigerung und eine Verringerung der Sollgröße. Jedoch werden durch eine fixe Rampe das dynamische Verhalten, sowie unterschiedliche Bodenbeschaffenheiten nicht berücksichtigt.

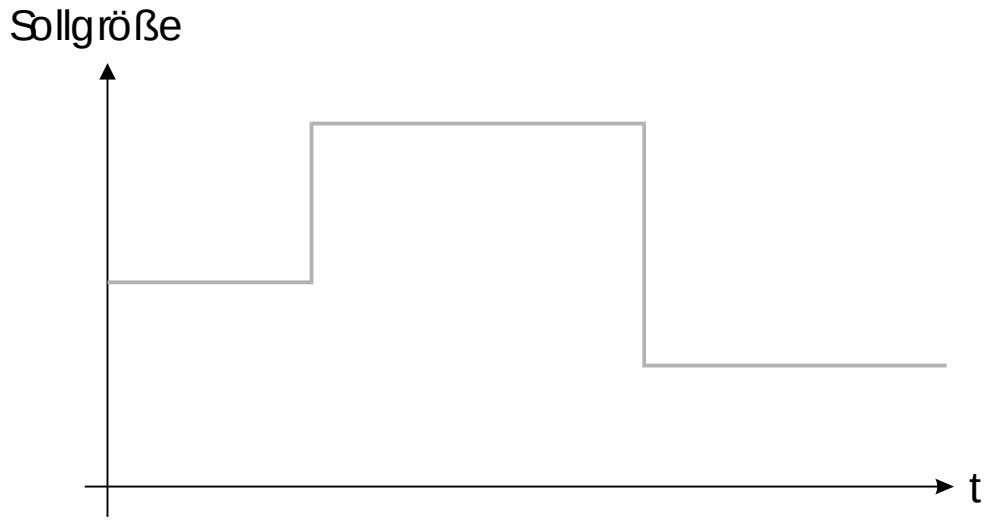

Abbildung 4.23: Sollgröße ohne Rampe

Durch Einführung einer Stellgrößenrampe werden die „harten“ Übergänge beim Sollgrößenwechsel vermieden:

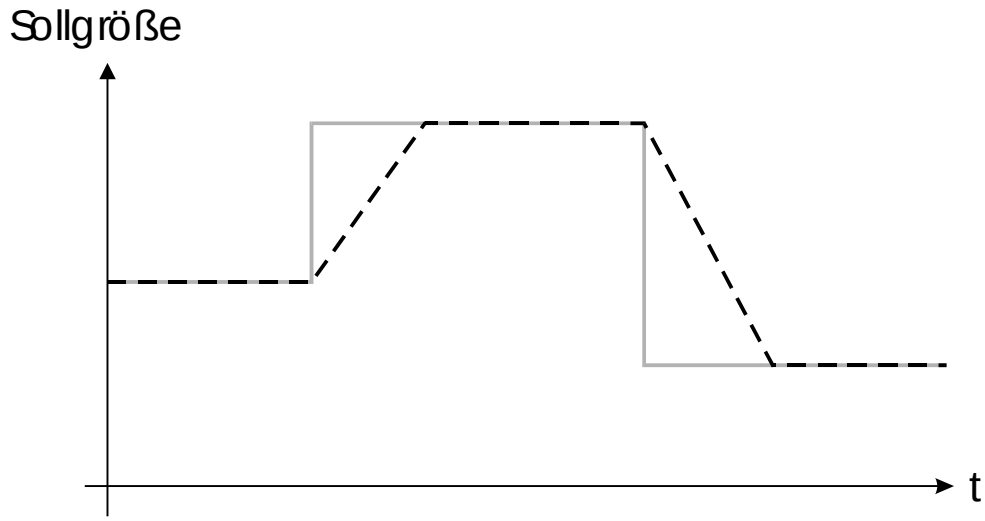

Abbildung 4.24: Sollgröße mit Rampe

Damit werden zu abrupte Wechsel der Sollgröße und damit im Fall des Roboters ein Durchrutschen der Räder vermieden. Dies wäre zwar grundsätzlich durch eine adäquate Einstellung des PID-Reglers ebenfalls möglich. Jedoch ergibt sich hier nur ein einziger Parameter, welcher je nach Bodenbeschaffenheit verstellt werden muss, damit der Roboter die Bodenhaftung optimal ausnützt.

Beschleunigungssensoren

Für eine Verbesserung der Regelung, im Speziellen um das Durchrutschen der Räder bei Beschleunigungen zu verhindern, beziehungsweise den Roboter an der Schlupfgrenze zu betreiben, sind Beschleunigungssensoren am Tinyphoon integriert. Mittels dieser Sensoren, kann der Roboter hart an der Schlupfgrenze betrieben werden. Die Möglichkeiten der Integration der Beschleunigungen reicht von komplexen adaptiven Reglern, bis hin zu einer einfachen Anpassung der gewählten Rampe (siehe Punkt PID - Regler). Näheres zum Thema Beschleunigungssensoren findet sich in der Diplomarbeit von Bernhard Putz [Putz02].

Modularität

Beim Design der Tinyphoon Software ist auf Modularität besonderes Augenmerk gerichtet worden.

Jedes Modul hat eine definierte Aufrufschnittstelle und verwendet eigene Ressourcen. Die Aufrufschnittstelle, also die zur Verfügung stehenden Funktionen sind im jeweiligen Header-File (*.h) definiert. Der Einsatz von globalen Variablen wurde weitestgehend vermieden.

Damit bleibt die Implementierung dem Gesamtkonzept der Modularität treu, alle Module können separat entwickelt, getestet und ausgetauscht werden.

Softwarestruktur und Taskkommunikation

Die Firmware für Tinyphoon nutzt vor allem die mächtige Interrupt-Maschinerie des xc167 aus, welche vielfach als „Betriebssystem on Silizium“ genannt wird: Etliche Aufgaben („Tasks“) werden durch Interrupts gesteuert und ausgelöst. Daher wurde auch auf den Einsatz eines Echtzeitbetriebsystems verzichtet, wie es sonst in derartigen Umgebungen durchaus üblich ist.

Entwicklungsumgebung

Für den xc167 existieren eine Reihe von Entwicklungstools. Bei der Softwareentwicklung für Tinyphoon kamen 2 Tools zum Einsatz:

C-Entwicklungsumgebung µVision 2 der Firma Keil
Freewaretool Minimon

Die C-Entwicklungsumgebung µVision 2 der Firma Keil diente als Compiler, Linker und Locater. Programmiert wurde ausschließlich in C, der Compiler ist ANSI-C kompatibel. Für den Zugriff auf den Bit-adressierbaren Bereich sowie auf die Special Function Register (SFR) ist der Compiler um entsprechende Zugriffsmöglichkeiten erweitert. Auch die sogenannten Intrinsic-Funktionen sowie die Definition der Interrupt Service Routinen stellen eine Erweiterung gegenüber Ansi-C dar.

Das Tool Minimon ist ein Freewaretool und diente zum Download des Programmes in das Flash. Weiters wurde Minimon als Terminalprogramm für die Debugausgaben sowie als Hardwaredebug-Tool verwendet. Auch die korrekten Einstellungen der verwendeten SFR wurden mit Minimon getestet.

Es kamen keine weiteren Debug- oder Testtools zum Einsatz. Fehlerdiagnose wurde vorrangig mit Hilfe von Debugausgaben an der seriellen Schnittstelle ASC0 betrieben.

4.7 Realisierung der Teilaufgaben

Empfang von Befehlen über Bluetooth

Für die Kommunikation mit einem Host Computer wurde eine Kommunikation über Bluetooth realisiert. Diese Bluetooth Übertragung ist rein auf HCI Ebene implementiert, wobei eine an den Hostcomputer angeschlossene Basisstation als Master fungiert und nach einer gewissen Anzahl an Robotern sucht.
Der xc167 kommuniziert mit dem Bluetooth Modul über ASC1. Die Bluetooth-Kommunikation wird dementsprechend in den Modulen BT und ASC1 realisiert.
Näheres hierzu findet sich in der Diplomarbeit von Oliver Fink [Fink03]

Datenaustausch zu anderen Robotereinheiten über CAN

Um die Anforderungen für die Kommunikation zu anderen Robotereinheiten zu erfüllen, wurde der CAN-Bus eingesetzt. Dieser erlaubt Datenversand über sogenannte Message Ids, d.h. Daten

werden nicht einem bestimmten Empfänger adressiert, sondern jeder am Bus angeschlossene Teilnehmer kann grundsätzlich jede Nachricht erhalten. Jeder Teilnehmer weiß, welche Nachrichten er benötigt und welche nicht. Über den CAN Bus können maximal 8 Byte große Nachrichten verschickt werden.
Die Buskommunikation über CAN wurde in 2 Modulen realisiert: Das Modul CAN abstrahiert die Hardware des xc167 CAN-Moduls und bietet bereits Funktionen zum Filtern bestimmter Messages.
Das Modul CAN_API erweitert die CAN-Funktionalitäten im wesentlichen dahingehend, dass Nachrichten größer 8 Byte verschickt bzw. empfangen werden können.

Befehlsauswertung und –abarbeitung

Für beide Kommunikationsarten – drahtlos oder über CAN-Bus – gibt es eine gemeinsame Befehlssprache. Diese Befehlssprache wird im Modul CPH („Control Protocol Handler“) interpretiert, korrekt erkannte Befehle werden übersetzt und an die entsprechenden Module weitergeleitet.

Positionsbestimmung

Die Positionsbestimmung des Roboters stützt sich auf die vom Host Computer übermittelten Daten. Um eine bessere Genauigkeit der vom Hostcomputer ermittelten Positionsdaten zu erhalten, wird nicht die gerade gemessene Position übertragen, sondern eine etwas ältere, die dadurch korrigiert wird, dass Messwerte zeitlich gesehen vor diesem Wert, sowie nach diesem Wert zur Schätzung herangezogen werden. Dadurch ist natürlich die übertragene Position noch älter als das zuletzt aufgenommene Bild. Die Aufgabe des Positionsbestimmungsalgorithmus besteht nun darin, die Position des Roboters zu bestimmen, zu welchem der Roboter erstmals reagieren kann. Dies geschieht über die durch die Sensorintegration ermittelten Bewegungsdaten.
Die Positionsbestimmung ist im Modul PM realisiert.

Regelschleife

Grundprinzip eines digitalen PID-Reglers ist eine genaue Zeitbasis. Bei Tinyphoon beträgt diese Zeitbasis 2ms. Die Regelschleife wird also alle 2ms aufgerufen, liest die aktuellen Raddrehzahlen ein, ruft mit diesen Werten und den Geschwindigkeits-Sollwerten den Regelungsalgorithmus auf.
Das Ergebnis aus dem Regelungsalgorithmus sind zwei Stellgrößen, jeweils eine für jedes Rad. Mit diesen werden die beiden PWM-Einheiten versorgt und damit die Motoren gesteuert.
Zusätzlich ist zu erwähnen, dass mit Hilfe der Beschleunigungssensorwerte die Sollgrößenrampe justiert wird.
Die Regelschleife ist im Modul CTR realisiert.

Regelungsalgorithmus

Als Regelungsalgorithmus kommt der bereits in diesem Kapitel beschriebene digitale PID Regler zum Einsatz. Die Parameter wurden empirisch festgestellt.

Ein Ziel der Arbeit war es, dass zukünftig unterschiedliche Regelungsalgorithmen eingesetzt werden können. Daher ist der PID-Regler ein eigenes Modul und wird über eine klar definierte Schnittstelle vom Modul CTR aufgerufen.

Erfassung und Aufzeichnung der Raddrehzahlen

Die Raddrehzahlen werden nach dem in diesem Kapitel beschriebenen Prinzip von Inkrementalgebern gemessen. Das Modul INCR, in welchem diese Funktion realisiert ist, bedient sich dabei der im xc167 vorgesehenen Funktion „Incremental Mode“ der Timer T2 und T4.

Erfassung und Ansteuerung der Motoren

Die beiden Gleichspannungsmotoren von Tinyphoon werden über ein PWM Signal gesteuert. Das Prinzip der PWM-Ansteuerung von Motoren wurde ebenfalls in diesem Kapitel bereits beschrieben.
Die PWM-Funktion ist im Modul PWM realisiert. Zum Einsatz kommt ein Timer im Compare-Mode

Ausgabe von Debugmeldungen an der Diagnoseschnittstelle

Zur besseren Diagnose von Fehlern oder zum Mitloggen von Daten ist eine eigene Diagnoseschnittstelle hilfreich. Bei Tinyphoon ist diese Schnittstelle die asynchrone serielle Schnittstelle ASC0 des xc167, mit einem gewöhnlichen Terminalprogramm können die Ausgaben an dieser Schnittstelle mitgeloggt werden.
Da keine weiteren Fehlerdiagnosetools (Emulator, JTAG, Simulator) zur Verfügung standen, ist eine Diagnoseschnittstelle für die Entwicklung unbedingt erforderlich.

Ansteuerung der LEDs zur Statusanzeige

Insgesamt stehen bei Tinyphoon 6 Leuchtdioden zur Verfügung. Für bestimmte Statusanzeigen (z. B. Datenverbindung aktiv, Kalibrierung abgeschlossen o.ä.) sollen die LEDs daher ein- oder ausgeschaltet werden können. Zusätzlich sollen LEDS auch kurz pulsen oder blinken können.
Für Tinyphoon wurde ein Timer als Zeitbasis eingesetzt, um neben dem Zustand ein/aus auch zeitgesteuerte Zustände wie langsam oder schnell blinken realisieren zu können. Die Funktionen zum Setzen und Ansteuern der LEDs sind im Modul LED realisiert.

Abfrage des Tasters am Roboter

Die Sensorplatine des Roboters beinhaltet auch einen Taster. Dieser Taster wird beispielsweise bei der Kalibrierung verwendet. Die Abfrage des Tasters wurde in ein eigenes Modul gepackt: KEY.

Speichern von Kalibrierungsdaten im Flash

Die Beschleunigungssensoren müssen auf 0g bzw. 1g kalibriert werden. Damit diese Kalibrierung nicht bei jedem Roboter-Neustart erfolgen muss, ist grundsätzlich eine

Möglichkeit vorzusehen, die Kalibrierungsdaten im Flash des xc167 und damit resistent gegen Spannungsverlust zu speichern.
Genutzt werden die letzten 128 Bytes des internen Flashes des xc167. Die entsprechenden Funktionen zum Schreiben der Daten ins Flash sind im Modul FLASH realisiert. Gelesen werden die Daten direkt über einen Datenzeiger, daher ist keine eigene Funktion dafür vorgesehen.

Erzeugung einer globalen Zeit

Die globale Zeit ist einerseits zur Synchronisierung mit den anderen Robotereinheiten und andererseits für Vorhersagen erforderlich. Zur Erzeugung einer globalen Zeit wird ein eigener Timer verwendet. Diese Funktion wurde im Modul G_TIME umgesetzt.

4.8 Modulbeschreibung

Basierend auf den Aufgaben der Roboter-Firmware ergeben sich eine Reihe von Softwareteilen bzw. Modulen. Diese sind noch weiter unterteilt in Modulgruppen.

Modulgruppen

Die einzelnen Softwaremodule können grob in folgende Modulgruppen unterteilt werden:

- Initialisierung: generelle Hardwareeinstellungen und Initialisierung aller Module
- Befehlsauswertung und Bewegungssteuerung: Auswertung von Kommunikationsbefehlen, Berechnung der nächsten Bewegungen und Sollvorgabe an den Regler
- Regelung: Regelung der Raddrehzahlen für die gewünschte Bewegung
- Kommunikation: Handling der Kommunikation über Bluetooth und CAN
- Sensordatenverarbeitung: Aufbereitung und Logging von Sensordaten
- Treiber: Low-Level Treiber für Peripherie
- Sonstige Funktionen: Globale Zeit und allgemeine Hilfsfunktionen

Modulübersicht

Abbildung 4.25 gibt eine Übersicht über die Modulgruppen von Tinyphoon:

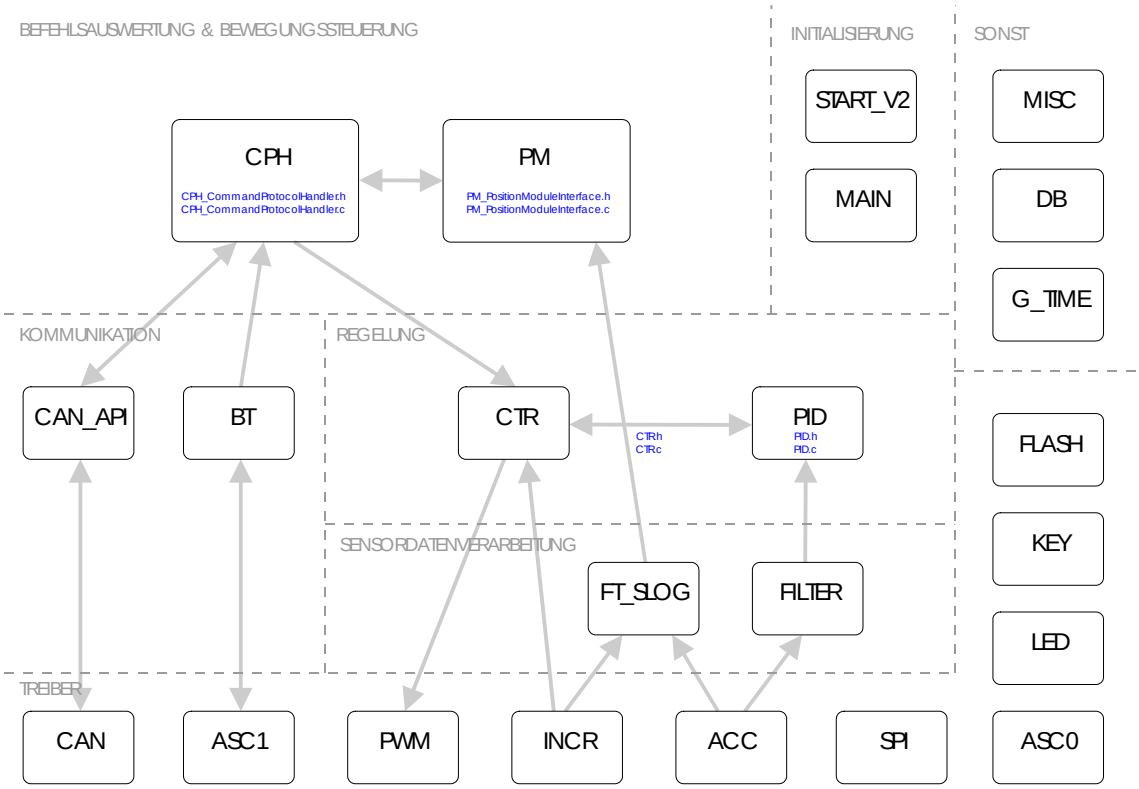

Abbildung 4.25: Softwaremodule und ihre Kommunikation

Modul	*Modulgruppe*	*Aufgabe*	*Dateien*
START_V2	Initialisierung	In der Startdatei wird die Konfiguration von Systemregistern, CPU-Takt und externer Speicherhardware vorgenommen. Dies ist der erste Programmcode, der vom xc167 ausgeführt wird.	start_v2.a66
MAIN	Initialisierung	Das Hauptprogramm wird von der Startdatei aufgerufen. Es initialisiert lediglich alle Module. Sobald die Initialisierung abgeschlossen ist, hat dieser Programmteil nichts mehr zu tun und führt eine leere Endlosschleife aus.	tinyphoon.c, tinyphoon.h, common.h
CPH	Befehlsauswertung & Bewegungssteuerung	Der ControlProtocollHandler erledigt Umsetzung der auf der Bluetooth bzw. CAN Schnittstelle verwendeten Protokolle	cph.c, cph.h
PM	Befehlsauswertung & Bewegungssteuerung	Der Roboter berechnet aufgrund eines Sensorlogs seine aktuelle Position sowie Geschwindigkeit und Winkelgeschwindigkeit	pm.c, pm.h
CTR	Regelung	Das Modul CTR steuert die zentrale	ctr.c,

		Regelschleife des Roboters: Umsetzung von Bewegungsbefehlen, Aufrufen des Regelalgorithmus und Ansteuerung der Motoren. Der eigentliche Regelungsalgorithmus befindet sich im Modul CTR_PID	ctr.h
PID	Regelung	Das Modul PID-Regler enthält den eigentlichen digitalen Regler, der aus vorgegebener Geschwindigkeit und Winkelgeschwindigkeit die Ansteuersignale für linkes und rechtes Rad erzeugt.	pid.c, pid.h
BT	Kommunikation	Das Modul BT ist für die Bluetooth-Kommunikation verantwortlich. Es ist für den initialen Verbindungsaufbau und für das Empfangen von Bluetooth-Paketen verantwortlich.	bt_interface.c, bt_interface.h , bt_hci.c, bt_hci.h
CAN_API	Kommunikation	Das Modul CAN_API abstrahiert die CAN-Schnittstelle und stellt Funktionen zum Empfang und Senden von Nachrichten größer 8 Byte zur Verfügung, indem es mehrere CAN-Nachrichten verkettet. (Normalerweise kann CAN nur maximal 8 Bytes übertragen).	can_api.c, can_api.h
FT_SLOG	Sensor datenverarbeitung	Das Modul FT_SLOG stellt im Wesentlichen einen Buffer für Sensordaten dar, auf den sequentiell geschrieben wird und beliebig die letzten n Werte gelesen werden können. Derzeit werden linke und rechte Geschwindigkeit sowie die 4 Beschleunigungswerte im Reglerintervall aufgezeichnet.	ft_slog.c, ft_slog.h
FILTER	Sensor datenverarbeitung	Das Modul FILTER bietet ein digitales IIR-Filter und wird zur Glättung der (heftig streuenden) Beschleunigungssensorwerte eingesetzt.	filter.c, filter.h
CAN	Treiber	Der CAN-Treiber abstrahiert das CAN-Interface des xc167 und bietet Funktionen zum Senden und Empfang von CAN-Messages. Beim Empfang wird eine frei definierbare Callbackfunktion aufgerufen.	can.c, can.h
ACC	Treiber	Das Modul ACC liest die 4 PWM-Signale der Beschleunigungssensoren ein und	acc.c, acc.h

		speichert die Werte. Diese können jederzeit abgerufen werden.	
ASC0	Treiber	Das Modul ASC0 abstrahiert die asynchrone serielle Schnittstelle 0 des xc167. Zu sendende und empfangende Daten werden in separaten Buffern gepuffert. Beim Empfang wird eine frei definierbare Callbackfunktion aufgerufen.	asc0.c,x asc0.h
ASC1	Treiber	Das Modul ASC1 abstrahiert die asynchrone serielle Schnittstelle 0 des xc167. Zu sendende und empfangende Daten werden in separaten Buffern gepuffert. Beim Empfang wird eine frei definierbare Callbackfunktion aufgerufen.	asc1.c, asc1.h
INCR	Treiber	Das Modul INCR liest auf Anforderung die Inkrementalgeberwerte der beiden Motoren ein. Damit wird im Modul CTR in weiterer Folge die Raddrehzahl ermittelt. Die gelieferten Werte sind vorzeichenbehaftet, geliefert wird lediglich die Differenz zum letzten Auslesevorgang.	incr.c, incr.h
KEY	Treiber	Im Modul KEY wird der Taster von Tinyphoon abstrahiert. Über eine einzelne Funktion kann der Zustand der Taste abgefragt werden.	key.c, key.h
LED	Treiber	Das Modul LED abstrahiert alle Leuchtdioden, die an Tinyphoon angeschlossen sind. Alle Leuchtdioden können auf Ein/Aus/Puls/Langsam blinkend und schnell blinkend eingestellt werden.	led.c, led.h
SPI	Treiber	Der SPI-Treiber bietet Funktionen zum Lesen und Schreiben auf der synchronen seriellen Schnittstelle. Die SPI wird bei Tinyphoon für den Datenaustausch mit dem Analog/Digitalwandler eingesetzt, daher ist die Konfiguration der SSC auf den ADC zugeschnitten	spi.c, spi.h
PWM	Treiber	Das Modul PWM dient zur Erzeugung der 4 PWM Signale für die beiden Motoren (jeweils 2 Signale für jede Richtung).	pwm.c, pwm.h
FLASH	Treiber	Das Modul Flash besteht aus mehreren Komponenten und bietet die Möglichkeit,	flash.c, flash.h,

		Daten im internen Flash des xc167 abzuspeichern. Damit bleiben beispielsweise Kalibrierungsdaten auch nach Spannungsverlust erhalten.	flashdat.c, flashdat.h, srom.c, srom.h
G_TIME	Sonstige Funktionen	Das Modul G_TIME erzeugt die roboterinterne globale Zeit, die als Bezug für zeitgesteuerte Befehle dient.	g_time.c, g_time.h
DB	Sonstige Funktionen	Das Modul DB bietet für alle anderen Module eine Abstrahierung von Debugausgaben. Die Debugausgaben werden über das Modul ASC0 getätigt.	db..c, db.h
MISC	Sonstige Funktionen	In diesem Modul sind allgemein einsetzbare Funktionen wie beispielsweise Wartebefehl, Ringbuffer, Quadratwurzel .... enthalten.	misc.c, misc.h

Tabelle 4.3: Übersicht über alle Module der TINYPHOON Firmware

Jedes dieser Module hat eine definierte Aufrufschnittstelle und verwendet eigene Ressourcen. Die Aufrufschnittstelle, also die zur Verfügung stehenden Funktionen sind im jeweiligen Header-File (*.h) definiert.

Initialisierung

Die Initialisierung der Tinyphoon Software erfolgt in mehreren Schritten:

Nach Reset des xc167 wird das im Projekt mitgebundene Modul START_V2 gestartet. Dieses Modul erledigt unter anderem das Abschalten des Watchdog-Timers, die Einstellung der PLL für die gewünschte CPU-Taktfrequenz, die Konfiguration des Externen Busses, Einstellung der Stackgröße und Position sowie Registerbank. Das Modul schließt die Initialisierungsphase mit dem speziellen xc167 Befehl EINIT (End of initialization) ab, wodurch irrtümliche nachträgliche Einstellungen für PLL, CPU, Watchdog und externen Bus gesperrt werden.
Des weiteren initialisiert das Modul START_V2 alle (globalen) Variablen und Datenbereiche mit 0. Letztendlich startet das Startup-Modul das Hauptprogramm main().

Das Hauptprogramm – Modul MAIN – erledigt die weitere Initialisierung der Module. Zuerst werden die von anderen Modulen weitgehend unabhängigen hardwarenahen Treibermodule initialisiert. Fast jedes Modul besitzt eine eigene Initialisierungsfunktion, welche genau einmal – nämlich im Modul MAIN in dieser Phase aufgerufen wird. Da die einzelnen Module weitgehend voneinander unabhängig sind, spielt die Aufrufreihenfolge eine untergeordnete Rolle.

Nach der Initialisierung der Treibermodule werden alle restlichen, höherschichtigen Module initialisiert, beispielsweise die Kommunikations- oder Regelungsmodule. Das Modul BT (Bluetooth) wird nur initialisiert, wenn der entsprechende DIP-Schalter aktiviert ist.

Bislang wurden zwar alle einzelnen Interrupts der Module konfiguriert und freigeschaltet, aber erst jetzt werden die Interrupts wirklich aktiviert. Dies geschieht durch Setzen des globalen Interrupt Enable Flags PSW_IEN auf 1.

Als nächstes wird die Roboternummer eingelesen. Diese wird durch die entsprechenden DIP-Schalter vor Inbetriebnahme des Roboters eingestellt. Eine nachfolgende Änderung der Roboternummer über die DIP-Schalter hat keine Wirkung.

Falls der entsprechende DIP-Schalter aktiviert ist, erfolgt die Kalibrierung der Beschleunigungssensoren. Bei diesem Prozess muss der Roboter in mehrere unterschiedliche Positionen gebracht werden. Kalibriert wird auf 0g (waagrechte Position) und 1g (senkrechte Position).

Mit diesem Schritt ist die Initialisierungsphase des Roboters abgeschlossen, der Roboter geht in den Betrieb über Das Hauptprogramm geht in eine leere Endlosschleife. Alle weiteren Aktionen werden durch Interrupts (zumeist in den Treiberroutinen) ausgelöst und finden in den Modulen statt.

Abbildung 4.26 zeigt den Ablauf der Programminitialisierung wie beschrieben:

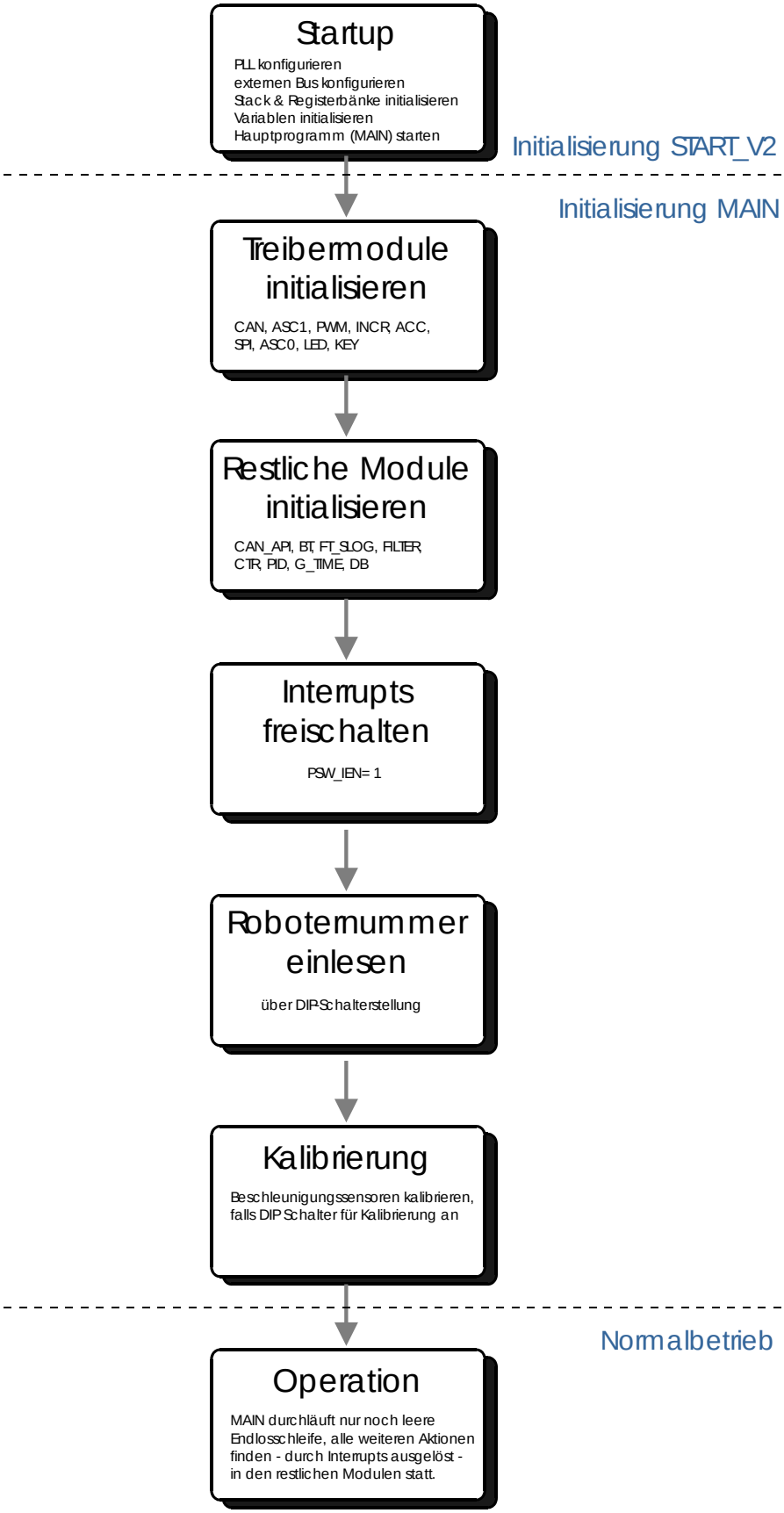

Abbildung 4.26: Ablauf der Programminitialisierung

Modul ASC0

Das Modul ASC0 abstrahiert die asynchrone serielle Schnittstelle 0 des xc167. Zu sendende und empfangende Daten werden in separaten Buffern gepuffert. Beim Empfang wird eine frei definierbare Callbackfunktion aufgerufen.

Funktionsweise

Der Standardmodus des Moduls ist der interruptgesteuerte Modus. Dabei werden sowohl ein Sende- als auch Empfangsbuffer (derzeit jeweils 2 KByte) eingesetzt. Beide Buffer sind FIFO-Ringbuffer, eine Funktion schreibt in den Buffer, eine andere liest daraus.

Sollen 1 oder mehrere Bytes gesendet werden, werden diese auf einmal in den Sendebuffer geschrieben und die Übertragung des ersten Bytes gestartet, sofern nicht bereits eine Übertragung läuft. Sobald das Byte gesendet ist, wird ein Sende-Interrupt ausgelöst. In dessen Interrupt Service Routine wird das nächste Byte aus dem Sendepuffer gelesen und gesendet.

Beim Empfang wird ein Interrupt ausgelöst, die entsprechende Interrupt Service Routine schreibt das Byte in den Empfangsbuffer und ruft – falls gesetzt – die angegebene Callbackfunktion auf, damit die darüberliegende Applikation vom Empfang informiert wird.

Zusätzlich zum gepufferten Senden und Empfangen von einzelnen Bytes bietet das Modul noch Funktionen zum Konvertieren und Senden von Zeichenfolgen und Hexadezimalzahlen.

Aufrufschnittstelle

Das Modul bietet im Wesentlichen folgende Funktionen:

Funktion	*Beschreibung*
void asc0_initialize(unsigned long baudrate)	Initialisiert die serielle Schnittstelle ASC0 mit der übergebenen Baudrate. Ansonsten ist die Schnittstelle konstant auf „no Parity,8 Datenbits, 1 Startbit, 1 Stopbit“ eingestellt. In dieser Routine werden alle relevanten Register gesetzt, die zugehörigen I/O-Ports konfiguriert und die Interrupts aktiviert.
short asc0_readchar(void)	Diese Funktion liest ein empfangenes Zeichen aus dem Empfangspuffer. Ist kein Zeichen im Puffer, so wird –1 zurückgeliefert.
void asc0_writechar(short ch)	Die Funktion asc0_writechar schreibt 1 Zeichen in den Sendepuffer und startet die Sendung, falls nicht schon eine Übertragung läuft. Wenn der Buffer voll ist, wird das Zeichen nicht in den Buffer geschrieben.
void asc0_setcallback(void (*callbackfunction) (short state));	Mit dieser Funktion kann eine Callback-Funktion definiert werden, die im wesentlichen

	aufgerufen wird, wenn ein Zeichen empfangen wird oder wenn der Empfangsbuffer überläuft.
void asc0_writestring(char *string)	Die Funktion asc0_writestring schreibt einen Characterstring in den Sendebuffer. Dabei wird die Funktion asc0_writechar zur zeichenweisen Ausgabe eingesetzt.
void asc0_writehexword(unsigned short w)	Diese Funktion gibt eine 16-Bit Zahl in Hexadezimaldarstellung an der seriellen Schnittstelle aus. Dabei wird die Funktion asc0_writechar zur zeichenweisen Ausgabe eingesetzt.
void asc0_writehexbyte(unsigned char ch)	Diese Funktion gibt eine 8-Bit Zahl in Hexadezimaldarstellung an der seriellen Schnittstelle aus. Dabei wird die Funktion asc0_writechar zur zeichenweisen Ausgabe eingesetzt.

Tabelle 4.4: Aufrufschnittstelle des Moduls ASC0

Verbindung mit anderen Modulen

Das Modul ASC0 wird vom Hauptprogramm (MAIN) mit 19200 Baud initialisiert. Vor allem die Funktion asc0_writestring wird von sehr vielen anderen Modulen für Status- und Debugausgaben genützt.

Dieses Modul verwendet die Ringpuffer-Datenstrukturen und -Funktionen im Modul MISC.

Konstanten und Externe Referenzen

Alternativ zum interruptgetriebenen, gepufferten Empfangs- und Sendemodus kann das Modul ASC0 auch ohne Interrupts, also mit Polling betrieben werden. Dazu muss das Define INTERRUPTMODE0 auf 0 gesetzt werden.

Das Define FCPU (bei Tinyphoon auf 40000000 für 40 MHz gesetzt) wird vom Modul ASC0 verwendet, um die gewünschte Baudrate einzustellen.

Verwendete Ressourcen

Das Modul ASC0 setzt logischerweise die Peripherieeinheit ASC0 ein. Die Schnittstelle wird interruptgetrieben eingesetzt (abhängig vom Flag INTERRUPTMODE0), das interne FIFO wird nicht eingesetzt. Für die exakte Einstellung der Baudrate wird der Fractional Divider eingesetzt. Die genaue Konfiguration der ASC0-Register ist im Kapitel 4.9 beschrieben.

Modul ASC1

Das Modul ASC1 abstrahiert die asynchrone serielle Schnittstelle 1 des xc167. Zu sendende und empfangende Daten werden in separaten Buffern gepuffert. Beim Empfang wird eine frei definierbare Callbackfunktion aufgerufen.

Funktionsweise

Der Standardmodus des Moduls ist der interruptgesteuerte Modus. Dabei werden sowohl ein Sende- als auch Empfangsbuffer (derzeit jeweils 2 KByte) eingesetzt. Beide Buffer sind FIFO-Ringbuffer, eine Funktion schreibt in den Buffer, eine andere liest daraus.

Sollen 1 oder mehrere Bytes gesendet werden, werden diese auf einmal in den Sendebuffer geschrieben und die Übertragung des ersten Bytes gestartet, sofern nicht bereits eine Übertragung läuft. Sobald das Byte gesendet ist, wird ein Sende-Interrupt ausgelöst. In dessen Interrupt Service Routine wird das nächste Byte aus dem Sendepuffer gelesen und gesendet.

Beim Empfang wird ein Interrupt ausgelöst, die entsprechende Interrupt Service Routine schreibt das Byte in den Empfangsbuffer und ruft – falls gesetzt – die angegebene Callbackfunktion auf, damit die darüberliegende Applikation vom Empfang informiert wird.

Zusätzlich zum gepufferten Senden und Empfangen von einzelnen Bytes bietet das Modul noch Funktionen zum Konvertieren und Senden von Zeichenfolgen und Hexadezimalzahlen.

Aufrufschnittstelle

Das Modul bietet im Wesentlichen folgende Funktionen:

Funktion	*Beschreibung*
void asc1_initialize(unsigned long baudrate)	Initialisiert die serielle Schnittstelle ASC1 mit der übergebenen Baudrate. Ansonsten ist die Schnittstelle konstant auf „no Parity,8 Datenbits, 1 Startbit, 1 Stopbit“ eingestellt. In dieser Routine werden alle relevanten Register gesetzt, die zugehörigen I/O-Ports konfiguriert und die Interrupts aktiviert.
short asc1_readchar(void)	Diese Funktion liest ein empfangenes Zeichen aus dem Empfangspuffer. Ist kein Zeichen im Puffer, so wird –1 zurückgeliefert.
void asc1_writechar(short ch)	Die Funktion asc1_writechar schreibt 1 Zeichen in den Sendepuffer und startet die Sendung, falls nicht schon eine Übertragung läuft. Wenn der Buffer voll ist, wird das Zeichen nicht in den Buffer geschrieben.
void asc1_setcallback(void (*callbackfunction) (short state));	Mit dieser Funktion kann eine Callback-Funktion definiert werden, die im wesentlichen

	aufgerufen wird, wenn ein Zeichen empfangen wird oder wenn der Empfangsbuffer überläuft.
void asc1_writestring(char *string)	Die Funktion asc1_writestring schreibt einen Characterstring in den Sendebuffer. Dabei wird die Funktion asc1_writechar zur zeichenweisen Ausgabe eingesetzt.
void asc1_writehexword(unsigned short w)	Diese Funktion gibt eine 16-Bit Zahl in Hexadezimaldarstellung an der seriellen Schnittstelle aus. Dabei wird die Funktion asc1_writechar zur zeichenweisen Ausgabe eingesetzt.
void asc1_writehexbyte(unsigned char ch)	Diese Funktion gibt eine 8-Bit Zahl in Hexadezimaldarstellung an der seriellen Schnittstelle aus. Dabei wird die Funktion asc1_writechar zur zeichenweisen Ausgabe eingesetzt.

Tabelle 4.5: Aufrufschnittstelle des Moduls ASC1

Verbindung mit anderen Modulen

Das Modul ASC1 wird vom Modul BT zur Kommunikation mit dem Bluetooth-Modul über das Host Computer Interface (HIC) eingesetzt.
Manche Module verwenden ASC1 zum Debuggen (hauptsächlich asc1_writestring). Dies ist natürlich nur dann möglich, wenn das Bluetooth-Modul ausgeschaltet ist.

Dieses Modul verwendet die Ringpuffer-Datenstrukturen und -Funktionen im Modul MISC.

Konstanten und Externe Referenzen

Alternativ zum interruptgetriebenen, gepufferten Empfangs- und Sendemodus kann das Modul ASC1 auch ohne Interrupts, also mit Polling betrieben werden. Dazu muss das Define INTERRUPTMODE1 auf 0 gesetzt werden.

Das Define FCPU (bei Tinyphoon auf 40000000 für 40 MHz gesetzt) wird vom Modul ASC0 verwendet, um die gewünschte Baudrate einzustellen.

Verwendete Ressourcen

Das Modul ASC1 setzt logischerweise die Peripherieeinheit ASC1 ein. Die Schnittstelle wird interruptgetrieben eingesetzt (abhängig vom Flag INTERRUPTMODE1), das interne FIFO wird nicht eingesetzt. Für die exakte Einstellung der Baudrate wird der Fractional Divider eingesetzt. Die genaue Konfiguration der ASC1-Register ist im Kapitel 4.9 beschrieben.

Modul PWM

Das Modul PWM dient zur Erzeugung der PWM Signale für die beiden Motoren. Insgesamt 4 PWM-Signale werden erzeugt – jeweils eines pro Motor und Drehrichtung. Die Motoren werden einerseits mit PWM-Signalen für jede Drehrichtung versorgt und können andererseits global aktiviert/deaktiviert werden.

Funktionsweise

Die Drehzahlregelung für Gleichspannungsmotoren basiert auf dem Wert der angelegten Gleichspannung. Anstelle von Gleichspannungssignalen können auch PWM-Signale eingesetzt werden. Da Motoren als induktive Lasten automatisch eine Tiefpass- und damit Integrationswirkung haben, enspricht ein PWM-Signal bei entsprechender PWM-Periode einem Gleichspannungssignal.

Für jede Drehrichtung eines Motores wird ein eigenes PWM-Signal eingesetzt. Daher müssen insgesamt 4 PWM-Leitungen „reserviert“ werden, zur gleichen Zeit sind aber immer nur 2 PWM-Signale erforderlich. Die Drehrichtung wird aus dem Vorzeichen des übergebenen Steuerwertes bestimmt.

Neben der Erzeugung der PWM-Signale bietet das Modul noch 2 Funktionen zum globalen Aktivieren/Deaktivieren eines Motors.

Aufrufschnittstelle

Das Modul bietet im Wesentlichen folgende Funktionen:

Funktion	*Beschreibung*
void pwm_initialize(void)	Die Funktion initialisiert die PWM-Einheit In dieser Routine werden alle relevanten Register gesetzt und die zugehörigen I/O-Ports konfiguriert.
void pwm_enable(unsigned short motor_nr)	Mit pwm_enable wird der angegebene Motor (0 oder 1) aktiviert.
void pwm_disable(unsigned short motor_nr)	Mit pwm_enable wird der angegebene Motor (0 oder 1) deaktiviert.
void pwm_setmotor(unsigned short motor_nr, short value)	Diese Funktion setzt den PWM-Wert und damit quasi das Ansteuersignal für den angegebenen Motor. Aus dem Vorzeichen des Steuerwertes wird die gewünschte Drehrichtung erkannt.
void pwm_stop(void)	Mit dieser Funktion werden alle PWM-Signale auf Minimum gesetzt, beide Motoren blockieren.

Tabelle 4.6: Aufrufschnittstelle des Moduls PWM

Verbindung mit anderen Modulen

Das Modul PWM wird im Modul MAIN initialisiert und vom Modul CONTROL periodisch aufgerufen.

Konstanten und Externe Referenzen

Das Define PWM_PERIODE bestimmt die PWM-Periode in µs. Bei der Einstellung der PWM-Einheit hat sich gezeigt, dass eine möglichst kurze PWM-Periode eingestellt werden sollte.

Das Flag STAGGERED_MODE bestimmt, ob der Staggered Mode eingesetzt wird oder nicht. Dieser Modus der CAPCOM-Einheit des xc167 existiert aus Kompatibilitätsgründen und beeinflusst die minimale PWM-Periode.

Der Wert MAXCONTROL bestimmt den Maximalwert des Steuerwertes beim Setzen des PWM-Signales. Derzeit liegt dieser Wert bei 1000, d.h. der übergebene Steuerparameter „value“ in der Funktion pwm_setmotor muss zwischen –1000 und +1000 liegen.

Verwendete Ressourcen

Eingesetzt wird die CAPCOM Einheit 1 des xc167 zur Erzeugung der 4 PWM-Signale. Grundsätzlich enthält der xc167 eine „spezialisierte“ PWM-Einheit – CAPCOM 6 – aber dessen zugeordnete Ein/Ausgabeleitungen liegen auf den gleichen Pins wie der Adressbus für externe Speicher.
Interrupts werden im Modul PWM nicht verwendet.
Die genaue Konfiguration der CAPCOM1-Register ist im Kapitel 4.9 beschrieben.

Modul INCR

Das Modul INCR dient zur Erfassung der Inkrementalgeberwerte der beiden Motoren. Durch periodischen Aufruf der Auslesefunktionen kann aus der Inkrementalgeberdifferenz die Drehzahl und Drehrichtung ermittelt werden.

Funktionsweise

Die beiden Motoren von Tinyphoon enthalten jeweils einen hochgenauen Inkrementalgeber. Dieser liefert pro Umdrehung 1024 Impulse auf einer Leitung. Eine zweite Leitung liefert ebenso die gleiche Anzahl an Impulsen, allerdings zeitversetzt. Dadurch ist die Drehrichtung eindeutig bestimmbar.

Der xc167 bietet in den Timereinheiten T2 und T4 jeweils die Möglichkeit, Inkrementalgebersignale zu erfassen. Das Zählerregister wird dabei mit jedem Impuls um 1 erhöht bzw. erniedrigt, je nach Drehrichtung. Durch periodisches Auslesen der zugehörigen Timerregister werden sowohl Anzahl der Impulse als auch die Zählrichtung erfasst und können

in einen Drehzahlwert und über den Raddurchmesser in die jeweilige Radgeschwindigkeit umgewandelt werden. Das Modul INCR bildet allerdings lediglich die Hardwareabstraktion, die Umrechnung in eine Drehzahl erfolgt im aufrufenden Modul CONTROL.

Damit der Zeitpunkt der Erfassung der Inkrementalgeberwerte bzw. Zählerdifferenzen für alle Motoren möglichst der gleiche ist, bietet das Modul INCR eine eigene Funktion INCR_Storevalues,

Aufrufschnittstelle

Das Modul bietet im Wesentlichen folgende Funktionen:

Funktion	*Beschreibung*
void INCR_Initialize(void)	Die Funktion initialisiert die beiden Timer T2 und T4 im Inkrementalgebermodus und konfiguriert die insgesamt 4 zugehörigen I/O Ports.
void INCR_StoreValues(void)	Bei Aufruf dieser Funktion werden die Inkrementalgeberwerte beider Zähler gesichert. Hier wird auch die Differenz zum alten Zählerstand gebildet und damit die eigentliche (relative) Information gespeichert.
short INCR_Getvalue(short motor_nr)	Diese Funktion dient zum Auslesen des letzten absoluten Zählerstandes für den angegebenen Motor. Die Funktion wird derzeit nicht genutzt, da nur die Zählerdifferenz wirklich eine Aussage liefert.
short INCR_GetDiff(short motor_nr)	Diese Funktion dient zum Auslesen der letzten Zählerdifferenz für den angegebenen Motor.

Tabelle 4.7: Aufrufschnittstelle des Moduls INCR

Verbindung mit anderen Modulen

Das Modul INCR wird im Modul MAIN initialisiert und vom Modul CONTROL periodisch aufgerufen. Zuerst wird in CONTROL die Funktion INCR_Storevalues aufgerufen, um die Zählerdifferenzen aller Motoren möglichst „gleichzeitig“ zu erstellen, und danach werden diese mit INCR_GetDiff für jeden Motor extra ausgelesen.

Konstanten und Externe Referenzen

Das Define MOTORS (standardmäßig auf 2) wird als Anzahl der Motoren des Roboters benutzt. Ansonsten werden keine externen Referenzen oder globale Variablen eingesetzt.

Verwendete Ressourcen

Eingesetzt werden die beiden Timereinheiten T2 und T4, welche einen speziellen Inkrementalgebermodus unterstützen. Interrupts werden nicht eingesetzt, das Auslesen der Zählerstände passiert durch periodischen Aufruf im Modul CTRL.
Die genaue Konfiguration der T2 und T4-Register ist im Kapitel 4.9 beschrieben.

Modul LED

Das Modul LED bietet Funktionen zum Ansteuern sämtlicher Leuchtdioden von Tinyphoon. Jede Led kann auf einen der folgenden Modi gestellt werden: Ein, Aus, kurzer Puls, langsam blinkend und schnell blinkend.

Funktionsweise

Sämtliche LEDs von Tinyphoon sind über digitale Ein/Ausgabeleitungen angeschlossen, teilweise sind sie mit logisch 1, teilweise mit logisch 0 aktivierbar.
Um Puls- und Blinkmodi zu unterstützten, ist grundsätzlich ein Zähler erforderlich. Da kein eigener Timer mehr für das Modul verfügbar ist, werden LEDs mit dem Zähler des Moduls G_TIME mitgeführt. Beim periodischen Aufruf der „Wartungsfunktion“ wird jede einzelne LED aktiviert oder deaktiviert, je nach Zustand des Moduls und Modus der LED.

Aufrufschnittstelle

Das Modul bietet im Wesentlichen folgende Funktionen:

Funktion	*Beschreibung*
void led_initialize(void)	Mit dieser Funktion wird das Modul initialisiert. Dies bedeutet vor allem das Schalten der relevanten I/O Ports auf Ausgang und Deaktivieren der LEDs.
void led_setmode(short lednr, short mode)	Mit der Funktion led_setmode können beliebige Module eine bestimmte LED auf einen der Modi LED_OFF, LED_ON, LED_BLINKSLOW, LED_BLINKFAST, LED_PULSE gesetzt werden.
void led_update(void)	Dies ist die Wartungsfunktion, welche von der Interrupt Service Routine des Moduls G_TIME regelmäßig aufgerufen wird.
void led_set(short lednr, short mode)	Diese interne Funktion des Moduls wird verwendet, um eine bestimmte LED sofort zu aktivieren bzw. deaktivieren. Hier wird lediglich das Ausgabesignal auf 0 bzw. 1 gesetzt

Tabelle 4.8: Aufrufschnittstelle des Moduls LED

Verbindung mit anderen Modulen

Das Modul LED wird im Modul MAIN initialisiert und von den verschiedensten Modulen aufgerufen, um mit LEDs einen bestimmten Betriebszustand zu signalisieren.

Konstanten und Externe Referenzen

Das Define NUMBER_OF_LEDS (gesetzt auf 6) wird als Anzahl der Leuchtdioden des Roboters benutzt.
Die Defines LED_MB_RED, LED_MB_YELLOW auf dem Mikrocontrollerboard und LED_SB_RED1, LED_SB_GREEN, LED_SB_RED2, LED_SB_YELLOW auf dem Sensorboard geben den Leuchtdioden Nummer 1-6 sprechende Namen.

Die Defines LED_OFF, LED_ON, LED_BLINKSLOW, LED_BLINKFAST und LED_PULSE repräsentieren alle möglichen Modi einer Led.

Verwendete Ressourcen

Eingesetzt werden lediglich 6 einzelne I/O-Leitungen auf Port 6 bzw. 9. Die Leitungen auf Port 6 aktivieren die LEDS mit logisch 0, jene an Port 9 mit logisch 1. Es wird kein eigener Interrupt, sondern jener von G_TIME verwendet.
Die genaue Konfiguration der T2 und T4-Register ist im Kapitel 4.9 beschrieben.

Verbesserungspotentiale

Das Modul LED sollte aus Performancegründen und zwecks niedrigerer Priorisierung (wieder) durch einen eigenen Timer – unabhängig vom Modul G_TIME - angesteuert werden. Damit ist das Modul wieder abgeschlossen und von anderen Modulen entkoppelt.

Modul KEY

Das Modul KEY bietet eine Funktion zum Auslesen des Tasters von Tinyphoon.

Funktionsweise

Der Taster ist an einer I/O-Leitung des Ports 3 vom xc167 angeschlossen. Das Signal logisch 1 bedeutet, dass keine Taste gedrückt wurde, logisch 0 bedeutet „Taste gedrückt“. Der Status der Taste kann jederzeit abgefragt werden, es wird bei Tastendruck jedoch kein Interrupt bzw. Event ausgelöst.

Aufrufschnittstelle

Das Modul bietet im Wesentlichen folgende Funktionen:

Funktion	*Beschreibung*

void key_initialize(void)	Mit dieser Funktion wird das Modul initialisiert. Es wird hier lediglich der für den Taster verwendete I/O-Port auf Eingang konfiguriert.
short key_pressed(short keynr)	Die Funktion key_pressed liefert für den Taster den aktuellen Zustand zurück: 0 für nicht dedrückt und –1 für gedrückt.

Tabelle 4.9: Aufrufschnittstelle des Moduls KEY

Verbindung mit anderen Modulen

Das Modul KEY wird im Modul MAIN initialisiert und von den Modulen PID und ACC im Rahmen der Kalibrierung der Beschleunigungssensoren aufgerufen.

Konstanten und Externe Referenzen

In diesem Modul werden keine Konstanten oder externe Referenzen verwendet.

Verwendete Ressourcen

Eingesetzt wird lediglich eine digitale Ein/Ausgabeleitung des Ports 3 zum Einlesen des Tasterzustandes. Es wird kein Interrupt eingesetzt.
Die Konfiguration des Port-Registers für Port 3 ist im Kapitel 4.9 beschrieben.

Verbesserungspotentiale

Der Taster sollte bei einem Hardware-Redesign an eine externe Interrupt-Leitung (CAPCOM1) angeschlossen werden. Damit wäre das Auslösen eines Interrupts bei Drücken des Tasters und damit eine eventgesteuerte Verarbeitung möglich.

Modul SPI

Das Modul SPI bietet im wesentlichen Funktionen zum Senden und Empfangen von Daten über die SPI-kompatible synchrone serielle Schnittstelle SSC0. Diese Schnittstelle dient als Kommunikationsschnittstelle zum externen Analog/Digitalwandler.

Funktionsweise

Die synchrone serielle Schnittstelle SSC0 des xc167 ist SPI-kompatibel und wird in Tinyphoon zum Datenaustausch mit dem Analog/Digitalwandler eingesetzt. Die Schnittstelle ist für bidirektionale Kommunikation ausgelegt, die Baudrate beträgt 100kBit/s, übertragen werden jeweils 8 Bits mit niederwertigsten Bit (LSB) zuerst.
Es werden keine Interrupts eingesetzt, sondern blockierend geschrieben bzw. gelesen.

Aufrufschnittstelle

Das Modul bietet im Wesentlichen folgende Funktionen:

Funktion	*Beschreibung*
void spi_Init(void)	Mit dieser Funktion wird das Modul initialisiert. Die Schnittstelle wird auf das eingesetzte Datenformat und Baudrate und die zugehörigen Ein/Ausgabeleitungen konfiguriert.
void spi_SendByte(char data)	Mit dieser Funktion wird das übergebene Datenbyte über die SSC0 geschickt und blockierend auf das Ende der Übertragung gewartet.
void spi_ReceiveByte(void)	Die Funktion spi_ReceiveByte wartet blockierend auf dem Empfang eines Datenbytes.

Tabelle 4.10: Aufrufschnittstelle des Moduls SPI

Verbindung mit anderen Modulen

Das Modul SPI wird im Modul MAIN initialisiert und vom Modul ADC zur Kommunikation mit dem externen Analog/Digitalwandler eingesetzt.

Konstanten und Externe Referenzen

In diesem Modul werden keine Konstanten oder externe Referenzen verwendet.

Verwendete Ressourcen

Eingesetzt wird die synchrone serielle Schnittstelle des xc167 sowie die zugehörigen Ein/Ausgabeleitungen auf Port 3. Es wird kein Interrupt eingesetzt.
Die Konfiguration der SSC0 ist im Kapitel 4.9 beschrieben.

Verbesserungspotentiale

Ein interruptgesteuertes und damit gepuffertes Lesen bzw. Schreiben auf der SSC0 ist zu empfehlen. Damit wird unnötiges Busy-Waiting vermieden und das Modul universell einsetzbar.

Modul ACC

Das Modul ACC liest automatisch die Beschleunigungswerte der beiden Beschleunigungssensoren ein. Die 4 Werte (jeweils X und Y-Beschleunigung pro Sensor) können jederzeit abgerufen werden.

Funktionsweise

Die bei Tinyphoon eingesetzten Beschleunigungssensoren liefern ihre aktuellen Beschleunigungswerte als 4 PWM-Signale. Diese Werte werden vom xc167 mittels CAPCOM

Einheit interruptgesteuert im Modus Capture vermessen, in Variablen abgelegt und können jederzeit von anderen Modulen abgerufen werden.

Wesentlich ist die Funktion der Kalibrierung: Da die Beschleunigungssensoren einerseits stark streuen und es andererseits keinen absoluten Bezugspunkt (Messbereich) gibt, müssen die Sensoren kalibriert werden. Bei der Kalibrierung werden die Beschleunigungswerte für 0 g (waagrechte Roboterposition) und 1 g (senkrechte Roboterposition) aufgenommen. Dabei werden mehrere Kalibrierdurchgänge benutzt und aus den aufgenommenen Werten der Median gebildet.

Aufrufschnittstelle

Das Modul bietet im Wesentlichen folgende Funktionen:

Funktion	*Beschreibung*
void acc_initialize(void)	Mit dieser Funktion wird das Modul initialisiert. Die CAPCOM-Einheit 1 wird entsprechend konfiguriert, Interrupts freigegeben und die zugehörigen Ein/Ausgabeleitungen konfiguriert. Des weiteren werden Konstante für die Burmester-Transformation berechnet.
short acc_get4controlloop(short *acc_trans)	Diese Funktion liefert an die übergebene Feldadresse die 4 bereits transformierten Beschleunigungswerte. Die Funktion acc_get4controlloop wird in der Regelschleife (Modul CTR) aufgerufen
unsigned short acc_calibration(void)	Die Funktion acc_calibration dient zur eingangs erwähnten Kalibrierung der Beschleunigungssensoren auf 0g (waagrechte, ruhende Position) und 1g (senkrechte ruhende Position).

Tabelle 4.11: Aufrufschnittstelle des Moduls ACC

Verbindung mit anderen Modulen

Das Modul ACC wird im Modul MAIN initialisiert und im wesentlichen vom Modul CTR. Die Beschleunigungswerte dienen zur Feinjustierung des Reglers, und zwar wird damit die Steilheit der Stellgrößenrampe eingestellt. Die gelieferten, transformierten Beschleunigungswerte werden vom Modul CTR im Feld acc_trans abgelegt und stehen auch anderen Modulen zur Verfügung.
Das Modul ACC nutzt die Funktionen des Moduls FILTER zur IIR-Filterung der Beschleunigungswerte. Dies ist notwendig, da die Beschleunigungswerte extrem streuen.

Konstanten und Externe Referenzen

Das Define CHANNELS definiert die Anzahl an zu vermessenden PWM-Signalen bzw. Beschleunigungswerten.

Das Define T2_RESISTOR bestimmt den Widerstandswert eines Beschleunigungssensors in kOhm und wird für die Kalibrierung benötigt.

Das Define CAL_COUNTS bestimmt bei der Kalibrierung die Anzahl an Messdurchgänge pro Position. Dieser Wert sollte größer als 4 sein.

Die Defines LBLX, LBLY, LBRX und LBRY definieren den Abstand der Beschleunigungssensoren vom Mittelpunkt in X- und Y-Richtung. Diese Werte werden für die Burmester-Transformation benötigt.

Die Flagvariablen print_flag und filter_flag werden im Modul ACC referenziert. Wenn filter_flag auf 1 gesetzt ist, werden die Beschleunigungswerte mit dem im Modul FILTER realsierten IIR-Tiefpassfilter geglättet.

Die externen Referenzen k_cal, z_cal, x_acc und x_pid werden zwar im Modul ACC im Rahmen der Kalibrierung generiert, aber im Modul FLASH abgelegt. Daher werden sie nicht im Modul ACC sondern im Modul FLASH deklariert.

Verwendete Ressourcen

Eingesetzt werden 4 Kanäle von CAPCOM1 des xc167 sowie die zugehörigen 4 Ein/Ausgabeleitungen auf Port 2.
Die Konfiguration der CAPCOM1 für das Einlesen der Beschleunigungswerte ist im Kapitel 4.9 beschrieben.

Verbesserungspotentiale

Eine sauberere Aufrufschnittstelle für die Beschleunigungsfunktionen ohne externe Referenzen bedeutet eine Verbesserung hinsichtlich Modularität und Wartbarkeit.

Modul FLASH

Das Modul FLASH dient zur Speicherung von prinzipiell beliebigen Daten im internen Flash des xc167. Tinyphoon nutzt das Modul zur Speicherung von Kalibrier- und Konfigurationsdaten.

Funktionsweise

Das interne Flash des xc167 wird nicht nur als Programmspeicher, sondern auch als Datenspeicher eingesetzt. Der Großteil des 128 KByte großen internen Flashes – nämlich der Bereich von 0xC00000-0xC1FFEF dient als Programmspeicher, lediglich die obersten 256

Bytes verwendet Tinyphoon als Datenspeicher. Grundsätzlich kann aber der Speicherbereich jederzeit erweitert werden.

Die zu speichernden Daten werden in der Datei flashdat.c definiert, es wird immer direkt mit den im Flash-Datenbereich liegenden Werten gearbeitet respektive gelesen. Da das direkte Schreiben auf Flashbereiche nicht möglich ist, existiert eine eigene Funktion flash_write zum Schreiben eines Buffers in den Flash-Datenbereich. Es werden genau die ersten 256 Bytes des Flashbuffers in das Flash geschrieben.

Ein wesentliches Problem bei der Programmierung des Flash ist die Tatsache, dass das Flash nicht aus dem Flash selbst heraus programmiert werden kann. Daher muss vor dem eigentlichen Schreiben in das Flash die Flash-Programmierroutine in das RAM kopiert werden. Aus diesem Grund wird die Funktion flash_write nicht direkt aufgerufen, sondern als Aufruffunktion dient die Funktion save_in_flash.

Aufrufschnittstelle

Das Modul bietet folgende Funktion:

Funktion	*Beschreibung*
short save_in_flash (unsigned short far *k_cal_new, unsigned short far *z_cal_new, unsigned short x_acc_new, unsigned short x_pid_new)	Diese Funktion bildet die Aufrufschnittstelle. Übergeben werden die im Flash zu speichernden Kalibrierwerte (teilweise als Pointer auf Strukturen, teilweise direkt). Die Funktion kopiert weiters die eigentliche Flashprogrammierroutine flash_write ins RAM, schaltet alle Interrupts ab und ruft dann die Flashprogrammierroutine mit den übergebenen Daten auf.

Tabelle 4.12: Aufrufschnittstelle des Moduls FLASH

Verbindung mit anderen Modulen

Das Speichern der Kalibrierwerte geschieht ausschließlich nach einer Kalibrierung. Daher wird das Modul lediglich vom Modul ACC benutzt, welches die Kalibrierfunktionen enthält.

Konstanten und Externe Referenzen

Das Define FDATADR definiert, an welche Adresse im internen Flash die Daten gespeichert werden sollen.

Die externen Referenzen k_cal, z_cal, x_acc und x_pid sind die globalen Variablen, in denen die Kalibrierdaten abgelegt sind. Sie werden in diesem Modul lediglich dazu verwendet, um den Flash-Schreibvorgang zu verifizieren.

Verwendete Ressourcen

Das Modul Flash verwendet keine Peripherieeinheiten, es werden lediglich die letzten 128 Bytes des internen Flashes des xc167 verwendet (Adresse 0xC1FF80-0xC1FFFF).

Verbesserungspotentiale

Der Flashbereich könnte auch zum Ablegen anderer Daten wie beispielsweise Konfigurationsdaten, Roboternummer oder Regler-Parameter. Dazu müsste die Aufrufschnittstelle etwas allgemeiner definiert sein.

Modul MISC

Das Modul MISC bietet einige Hilfsfunktionen für andere Module. Es hat keinen Aufgabenbereich im eigentlichen Sinne.

Funktionsweise

Das Modul MISC enthält folgende Hilfsfunktionen:

- Funktion delay zum Verzögern bzw. Warten einer bestimmten Anzahl von ms (busy waiting)
- Ringbufferfunktionen, die von den beiden seriellen Modulen ASC0 und ASC1 verwendet werden.
- Funktion zum Berechnen des Maximums der Absolutwerte zweier übergebener long-Werte
- Timeoutfunktion zum Überwachen von Kommunikationsprozessen
- Funktionen zum Berechnen der Quadratwurzel einer short bzw. einer long Zahl

Aufrufschnittstelle

Das Modul bietet im Wesentlichen folgende Funktionen:

Funktion	*Beschreibung*
void misc_delay(short ms)	Diese Funktion wartet für die angegebene Zeit in ms und verzögert damit den weiteren Programmablauf. Die Funktion arbeitet mit busy-waiting.
void rb_initialize(t_ringbuffer *pringbuffer)	Initialisiert den als Parameter übergebenen Ringbuffer, indem Schreib- und Lesezeiger zurückgesetzt und alle Bufferinhalte auf 0 gesetzt werden.
short rb_isempty(t_ringbuffer *pringbuffer)	Gibt zurück, ob der übergebene Ringbuffer leer ist. Ein Ringbuffer ist dann leer, wenn Schreib- und Lesezeiger auf den gleichen Eintrag stehen.
short rb_isalmostfull(t_ringbuffer	Gibt zurück, ob der übergebene Ringbuffer fast

*pringbuffer)	voll ist. Ein Ringbuffer ist dann fast voll, wenn der Schreibzeiger um 2 kleiner als der Lesezeiger ist, wenn also nur noch 1 Eintrag möglich ist
short rb_isfull(t_ringbuffer *pringbuffer)	Gibt zurück, ob der übergebene Ringbuffer voll ist. Ein Ringbuffer ist dann fast voll, wenn der Schreibzeiger um 1 kleiner als der Lesezeiger ist, wenn also kein Eintrag mehr möglich ist
long misc_absmax(long f1, long f2)	Gibt den größeren der Absolutwerte der beiden übergebenen long-Werte zurück
void misc_settimeout(unsigned short ms, void (*timeoutfunction)())	Mit misc_settimeout wird das Timeout aktiviert. Wird nicht innerhalb der angegebenen Zeit ms (in ms) die Funktion misc_disabletimeout aufgerufen, so wird die übergebene Timeoutfunktion als Callback aufgerufen. Der Programmablauf setzt nach Aufruf dieser Funktion normal fort
void misc_disabletimeout(void)	Diese Funktion deaktiviert ein eventuell laufendes Timeout. Diese Funktion wird im Regelfall dann aufgerufen, wenn die zu überwachenden Aktionen erfolgreich vor dem Ablaufen des Timeouts aufgerufen werden.
unsigned short misc_sqrt(short x,unsigned short n)	Diese Funktion bildet die Quadratwurzel der übergebenen short-Zahl x. Die Anzahl der Iterationen n bestimmt dabei die Berechnungsgenauigkeit.
misc_lsqrt(unsigned long x, unsigned short n)	Diese Funktion bildet die Quadratwurzel der übergebenen long-Zahl x. Die Anzahl der Iterationen n bestimmt dabei die Berechnungsgenauigkeit.

Tabelle 4.13: Aufrufschnittstelle des Moduls MISC

Verbindung mit anderen Modulen

Die genannten Funktionen werden von unterschiedlichsten anderen Modulen eingesetzt

Konstanten und Externe Referenzen

Die globale Konstante FCPU wird von den delay und timeout-Funktionen benötigt.

Verwendete Ressourcen

Die Timeoutfunktionen verwenden Timer 1, ansonsten werden keine Ressourcen eingesetzt.

4.9 Konfiguration des xc167 für Tinyphoon

Da bereits in der Entwurfsphase von Tinyphoon die Anbindung externer Komponenten wie Sensoren, Aktoren und anderer Controller an den xc167 berücksichtigt werden musste, wurden bereits im Vorfeld der Entwicklung die hardwaremäßige Anbindung und die softwaremäßige Konfiguration der einzelnen Peripherieeinheiten und auch der CPU selbst festgelegt.

Dieses Kapitel beschreibt im Detail die gewählten Registerkonfigurationen für CPU und Peripherie der entsprechenden SFR-Register.

Grundlegende CPU-Einstellungen

Alle CPU- und Buskonfigurationsrelevanten Einstellungen werden in der Startdatei START_V2.A66 vorgenommen. Diese Datei wird vom Compilerhersteller mitgeliefert und muss in jedes Projekt eingebunden werden. Sie enthält für alle wichtigen Teile eine Voreinstellung. Im folgenden werden nur die gegenüber der Standardkonfiguration abweichenden Einstellungen beschrieben:

Konfiguration des externen Busses

Als Programmspeicher sind die internen 128 KB des xc167 ausreichend, nicht jedoch der Datenspeicher von 8 KB. Daher wurde für das Board ubx167 ein externes SRAM von 512 KB mit 25ns Zugriffszeit vorgesehen, welches an Chip Select 1 (CS1) angeschlossen ist.

Auf das externe SRAM kann mittels externem Adress- und Datenbus zugegriffen werden. Adressen und Daten sind nicht gemultiplexed, der Datenbus ist 16 Bit breit.

Entsprechend diesen Anforderungen müssen folgende SFR-Register bereits bei der Initialisierung gesetzt werden:

ADDRSEL1: Hier wird das Adressfenster für 512 KB RAM ab Startadresse 10‘0000 festgelegt.
FCONCS1: Hier wird CS1 aktiviert, der Bustyp und die Zugriffsart festgelegt
TCONCS1: Hier wird das Timing für CS1 eingestellt.

Für ADDRSEL1 ergibt sich:

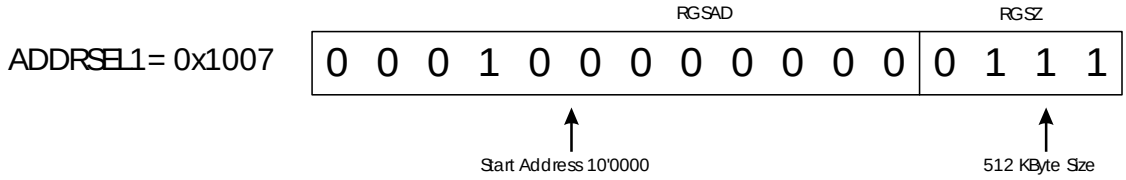

Abbildung 4.27: Konfiguration des externen Busses – Adressbereich für das externe RAM

Aus der Verwendung von CS1, Busmodus 16 Bit demultiplexed Bus ohne Ready ergibt sich für FCONCS1:

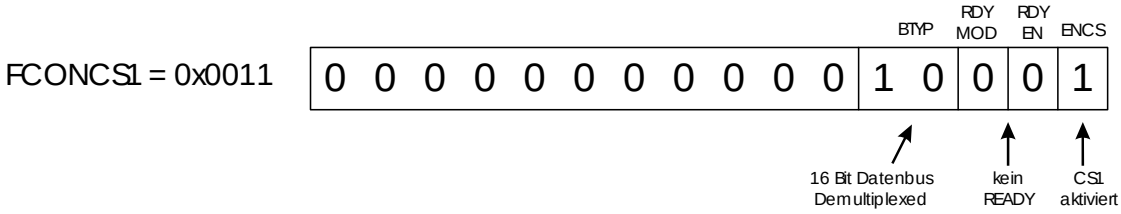

Abbildung 4.28: Konfiguration des externen Busses – Busmodus und Zugriff für das externe RAM

Bleibt noch die Konfiguration des Timings im Register TCONCS1. Hier ergibt sich aus den Datenblättern von xc167 und SRAM bis auf den Timing-Parameter Phase F beim Lesen die jeweils kürzeste und damit schnellste Einstellung. Phase F wird auf 1 Taktzyklus eingestellt. Es ergibt sich:

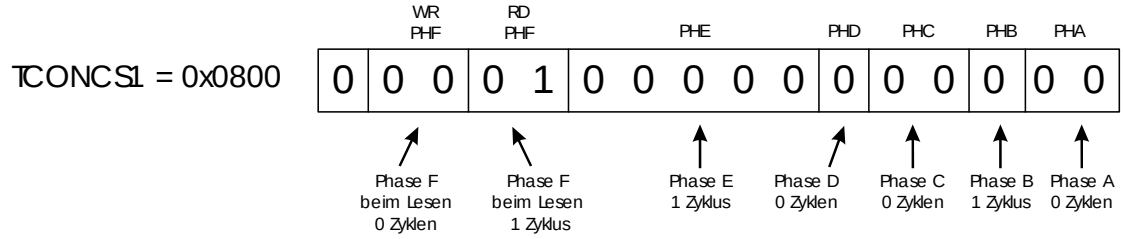

Abbildung 4.29: Konfiguration des externen Busses – Timing für das externe RAM

Taktfrequenz

Die vorgesehene Taktfrequenz für den xc167 ist gleichzeitig die maximal einstellbare, nämlich 40 MHz. Der xc167 bietet die Möglichkeit, das externe Taktsignal (z. B. von Quarz) zu teilen oder/und zu multiplizieren), also nachträglich und softwaremäßig den CPU-Takt zu verändern. Als externer Quarz wurde ein 8 MHz Quarz vorgesehen.

Insgesamt stehen 2 Teiler und 1 Multiplikator zur Verfügung, den vom Quarz extern gelieferten Takt für die CPU zu verändern: PLLIDIV, PLLODIV und PLLMUL. Alle drei haben eingeschränkte Wertebereiche und sind in dem SFR PLLCON einstellbar.

$$f_{CPU} = f_{OSC} * \frac{(PLLMUL + 1)}{(PLLIDIV + 1) * (PLLODIV + 1)}$$

Abbildung 4.30: Berechnung der Taktfrequenz abhängig von unterschiedlichen PLLCON-Parametern

Aus f_{OSC} = 8 MHz und f_{CPU} = 40 MHz ergeben sich unterschiedliche mögliche Kombinationen für PLLMUL, PLLIDIV und PLLODIV.

Gewählt wurde:
PLLMUL = 24, PLLIDIV = 0, PLLODIV = 4
und damit

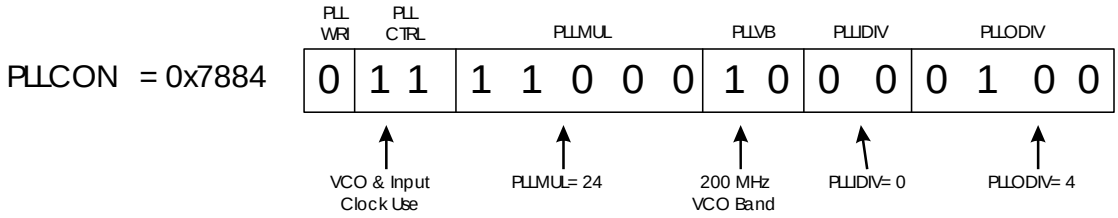

Abbildung 4.31: Konfiguration des Registers PLLCON für Tinyphoon

Eine kurze Überprüfung ergibt:

$$f_{CPU} = 8\ MHz * \frac{(24+1)}{(0+1)*(4+1)} = 40\ MHz$$

Abbildung 4.32: Überprüfung der PLLCON Konfiguration

Intern genutzte Peripherie

Die Software des xc167 nutzt mehrere unabhängige Timer als Zeitgeber für verschiedene Tasks:

Zeitgeber für Regelungschleife: T7

Der diskrete Regler des Moduls CTR wird in äquidistanten Abständen aufgerufen. Dazu wird ein hochpriorer Timer-Interrupt benötigt, der Regler läuft quasi als Interrupt Service Routine. Timer 7 der Capture/Compare Unit 2 bietet sich dafür an. Dieser 16 Bit Timer unterstützt einen automatischen Nachlademodus.

Der Timer wird daher im Nachlademodus betrieben und zählt vom Initialwert hinauf bis 0xFFFF. Beim Überlauf auf 0x0000 wird sowohl ein Interrupt generiert als auch automatisch der Nachladewert wieder in den Timer geladen, der Timer startet von vorne. Der Nachladewert entspricht dem Initialwert.

Für TinyPhoon wurde eine Regler-Periode von 2ms gewählt. Entsprechend ergibt sich in Abhängigkeit der CPU-Frequenz von 40 MHz und des eingestellten Vorteilers 8 eine Timerauflösung von 200ns und damit ein Initial- bzw. Nachladewert von

Initialwert = Nachladewert = 0xFFFF-fcpu/8*ms/1000000 = 0xFFFF-10 = 0xFFF5

Folgende Register werden zur Initialisierung des Timers 7 benötigt:
CC2_T78CON: XX40, Timermodus, automatisches Nachladen, Vorteiler 8
CC2_T7: Initialwert
CC2_T7REL: Nachladewert

Der Interrupt für Timer 7 wird mit folgendem Register konfiguriert:
CC2_T7IC: 0064

Zeitgeber für globale Zeit: T5

Die höchste Priorität kommt dem globalen Zeitgeber im Modul G_TIME zu. Hier wird die global in TinyPhoon benötigte Zeit erzeugt. Der globale Zeitgeber ist vor allem für erhaltene Befehle mit absoluten oder relativen Zeitangaben erforderlich. Timer 5 wird dazu verwendet, einen exakten, regelmäßigen Zeitgebertask als Interrupt Service Routine zu erzeugen.

Des weiteren werden auch die Leuchtdioden über diesen Task angesteuert. Für die LEDs ist deshalb ein Zeitgeber erforderlich, da sie auch Puls- und Blinkmodi unterstützen, die nur einmal angestoßen werden müssen.

Timer 5 wird in Tinyphoon im Timermodus mit Vorteiler 8 betrieben, mit Zählrichtung nach unten.

Folgende Register werden zur Initialisierung des Timers 5 benötigt:
GPT12E_T5CON: 00C1, Timermodus, Vorteiler 8, Zählrichtung nach unten
GPT12E_T5: Initialwert, wird in jeder Interrupt Service Routine nachgeladen.

Initialwert = fcpu/8*ms/1000000 = 10

Der Interrupt für Timer 5 wird mit folgendem Register konfiguriert:
GPT12E_T5IC: 0x007C

Digitale Ein/Ausgabeleitungen

Der xc167 bietet zahlreiche digitale Ein/Ausgabeleitungen. Jede einzelne Leitung kann auf Eingang / Ausgang oder eine Alternativfunktion gesetzt werden. In diesem Kapitel sind alle Ein/Ausgabeleitungen beschrieben, die explizit als solche eingesetzt werden.

Leuchtdioden: Ein/Ausgabeleitungen P6.6, P6.7, P9.2, P9.3, P9.4, P9.5

Insgesamt sind bei Tinyphoon an den xc167 6 Leuchtdioden angeschlossen. Die Leuchtdioden an P6 werden mit logisch 0, jene an P9 mit logisch 1 angesteuert.

Zur Initialisierung müssen die Ein/Ausgabeleitungen P6.6-P6.7 und P9.2-P9.5 auf Ausgang geschaltet werden. Die LEDs werden ausgeschaltet:
DP9: XX3C, Bits 2 bis 5 auf 1, alle anderen Bits bleiben unverändert
DP6: XXCX, Bits 6 und 7 auf 1, alle anderen Bits bleiben unverändert
P9: XXC3, Bits 2 bis 5 auf 0, alle anderen Bits bleiben unverändert
P6: XXCX, Bits 6 und 7 auf 1, alle anderen Bits bleiben unverändert

Anmerkung: Der Zustand der LEDs werden über den Timer 5 der globalen Zeit mit angesteuert, damit Blink- und Pulsmodi möglich sind.

Taster: P3.6

Tinyphoon stellt auch einen Taster zur Verfügung, welcher über den Port P3.6 eingelesen werden kann. Der Taster wird vor allen für die Kalibrierung der Beschleunigungssensoren verwendet.

Da beim Reset des xc167 grundsätzlich alle Ports auf Eingang konfiguriert sind, ist keine separate Einstellung erforderlich. Sicherheitshalber kann die Datenrichtung über das Register DP3 auf Eingang konfiguriert werden:
DP3: XXBX, Bit 6 auf 0, alle anderen Bits bleiben unverändert

DIP-Schalter: P5.0-P5.7

Weiters inkludiert Tinyphoon einen 8fach DIP-Schalter, welcher unterschiedliche Systemkonfigurationen und Anwendungseinstellungen zulässt.

Hier gilt ebenfalls, dass eine Einstellungsänderung der Signalrichtung für P5.0-P5.7 nicht erforderlich ist, da die Standardeinstellung ohnehin „Eingang“ ist. Die Konfiguration für P5 lautet:
DP5: XX00, Bit 0-7 auf 0, alle anderen Bits bleiben unverändert

Externe Komponenten und deren Schnittstellen

Neben internen Zeitgebern sind eine Reihe von externen Komponenten an Peripherieeinheiten des xc167 angeschlossen. Teilweise werden sie mit Interrupts betrieben, teilweise nicht.
Die Peripherieeinheiten werden über die I/O Leitungen des xc167 angeschlossen, daher sind auch die zugehörigen Register der Ein/Ausgabeports zu setzen.

Beschleunigungssensoren: CAPCOM 12-15

Tinyphoon enthält 2 Beschleunigungssensoren mit jeweils 2 Kanälen. Die Daten werden von den Sensoren als PWM-Signale ausgegeben. Um diese PWM-Signale zu vermessen, wird die Capture/Compare Einheit 1 des xc167 verwendet.
Zum Einsatz kommen 4 Kanäle der CAPCOM1 (CC12-CC15) gemeinsam mit Timer 1, welcher mit einer Auflösung von 400ns im betrieben wird. Der Timer durchläuft dabei die volle Periode und wird daher auch nicht nachgeladen. Er löst auch keinen Interrupt aus.
Bei jedem positiven und negativen Flankenwechsel wird der aktuelle Inhalt des Zählers aufgezeichnet und ein Interrupt ausgelöst. Bei positivem Flankenwechsel wird zusätzlich der Differenzwert und daraus der Beschleunigungswert berechnet.

Folgende Register werden zur Initialisierung des Timers 1 und der CAPCOM1 benötigt:
CC1_T01CON: 41XX, Timermodus, Vorteiler 8

CC1_M3: BBBB, jeder der 4 Kanäle benutzt Timer 1, dessen Inhalt sowohl bei positiver als auch negativer Flanke in das zugehörige Capture Register (CC12-CC15) aufgezeichnet wird.

Die Interrupts für die CAPCOM-Kanäle 12-15 werden folgendermaßen initialisiert:
CC1_CC12IC: 0078
CC1_CC13IC: 0079
CC1_CC14IC: 007A
CC1_CC15IC: 007B

Die eingesetzten Ein/Ausgabeleitungen P2.12-P2.15 werden durch folgende Register konfiguriert:
DP2: FXXX
ALTSELOP2: FXXX

Inkrementalgeber: T2 und T4

In den beiden Antriebsmotoren von Tinyphoon sind jeweils Inkrementalgeber integriert. Diese liefern 2048 Impulse pro Radumdrehung. Mittels zweier Signale pro Inkrementalgeber ist auch die Drehrichtung eindeutig feststellbar.
Sowohl Timer 2 als auch Timer 4 des xc167 bieten alternativ zu den obligatorischen Timer- und Countermodi einen Inkrementalgebermodus. Dabei werden die externen Leitungen T2IN, T2EUD, T4IN und T4EUD benutzt
Weder Timer 2 noch Timer 4 löst Interrupts aus. Die aktuellen Zählerstände werden im Regler ausgelesen, welcher regelmäßig aufgerufen wird.

Folgende Register müssen für den Einsatz von T2 und T4 konfiguriert werden:
GPT12E_T2CON: 0173
GPT12E_T4CON: 0173
GPT12E_T2: 8000
GPT12E_T4: 8000

Die Ein/Ausgabeleitungen werden folgendermaßen initialisiert:
P3: XX5X, T2IN und T4IN auf 0
DP3: XX5X, T2IN und T4IN als Eingang
P5: 3XXX, T2EUD und T4EUD auf 0
DP5: 3XXX, T2EUD und T4EUD als Eingang

Beim regelmäßigen Abfragen der Inkrementalwerte durch den Regler ist das quasi-gleichzeitige Abfragen der Registerinhalte von T2 und T4 erforderlich. Daher werden die Timer-Register T2 und T4 im Modul INCR unmittelbar hintereinander abgefragt.

DC-Motor PWM: CAPCOM 8-11

Grundsätzlich ist Capture/Compare Einheit 6 für die Generierung von PWM-Signalen gedacht. Da die zugeordneten Ein/Ausgabeleitungen jedoch als Alternativfunktionen von Port 1 angelegt

sind und dieser bereits als Adressbus eingesetzt wird, kommen 4 Kanäle von CAPCOM 1 gemeinsam mit Timer 0 zum Einsatz. Jeweils 2 Kanäle werden dabei pro Motor eingesetzt, einer für jede Drehrichtung.

Timer 0 dient als Zeitbasis zur Generierung sämtlicher PWM-Signale. Die Auflösung beträgt bei 40 MHz CPU-Takt 50ns, die Periode 50µs. Der Timer wird im automatischen Nachlademodus betrieben, generiert aber keine Interrupts. Für die gewählte hohe Auflösung muss der sogenannte Staggered Mode ausgeschaltet sein.

Die Leitungen CC8-CC11 der Capture/Compare Einheit werden als PWM-Signale genutzt. Zusätzlich kommen P9.0 und P9.1 zum Einsatz, um die beiden Motoren global ein- bzw. auszuschalten.

Timer 0 wird folgendermaßen konfiguriert:
CC1_T01CON: XX41, Timer 0 im automatischen Nachlademodus, Zählrichtung hinauf, Vorteiler 2
CC1_T0REL: PWM-Periode
CC1_IOC: 0004, Staggered Mode ausgeschaltet

Der Wert für die PWM-Periode muss als Timer-Periode gesetzt werden:

PWM-Periode = FFFF- 1000
Den CAPCOM Kanälen 8-11 sind die Ein/Ausgabeleitungen P2.8-P2.11 zugeordnet. Daher müssen diese ebenfalls einmalig initialisiert werden:
P2: XFXX, Port P2.8-P2.11 werden auf 1 gesetzt.
DP2: XFXX, Port P2.8-P2.11 werden auf Ausgang gesetzt.
ALTSELOP2: XFXX, Die Alternativfunktionen für Port P2.8-P2.11 werden aktiviert.

Die CAPCOM-Kanäle 8-11 werden im Register CC1_M2 aktiviert bzw. deaktiviert. Jeweils 4 Bit sind einem Kanal zugeordnet, zum Aktivieren eines Kanales wird 7 in das entsprechende Bitfeld geschrieben, zum Deaktivieren 0.
CC1_M2: Beispiel: 0000, Alle CAPCOM-Kanäle ausgeschaltet
CC1_M2: Beispiel: 0707, CAPCOM 8 und 10 sind aktiviert, 9 und 11 deaktiviert

Der PWM-Wert wird über das entsprechende CAPCOM Register gesetzt. Grundsätzlich kann der Wert zwischen 0 und 999 liegen. Da der Zähler immer hochzählt, muss der PWM-Wert von FFFF abgezogen werden.
CC1_CC8: FFFF-PWM-Wert Kanal 8
CC1_CC9: FFFF-PWM-Wert Kanal 9
CC1_CC10: FFFF-PWM-Wert Kanal 10
CC1_CC11: FFFF-PWM-Wert Kanal 11

Service & Debugschnittstelle: ASC0

Die asynchrone Serielle Schnittstelle ASC0 wird bei Tinyphoon als allgemeine Service- und Debugschnittstelle eingesetzt. Die Baudrate ist für den Betrieb als Service- und Debugschnittstelle fest auf 19200 Baud eingestellt, wenngleich das Modul ASC0 für beliebige Baudraten ausgelegt ist.

Die Initialisierung von ASC0 erfolgt mittels Register ASC0_CON:
ASC0_CON: 8811, Baudratengenrator ein, Empfänger ein, kein Parity, 8 Datenbits, 1 Stopbit, Fractional Divider aktiviert

Die Baudrate ist abhängig von der CPU-Frequenz, von der Vorteiler-Einstellung und natürlich von der gewünschten Baudrate. Es muss einerseits das Baudraten-Nachladezählerregister BG und andererseits das Fractional Divider Register FDV gesetzt werden. Die tatsächliche Baudrate ergibt sich folgendermaßen:

Baudrate = FDV/512*FCPU/(16*(BG+1)

Während mit dem Register BG die Baudrate grob eingestellt wird, kann mit FDV die Abweichung der tatsächlichen Baudrate von der gewünschten minimiert werden. Vor allem bei hohen Baudraten kann mit BG alleine die Abweichung recht hoch sein.
Daher werden 2 Einstellungen benötigt: Register BG und Register FDV. Die beiden Register werden in 2 Stufen initialisiert, zuerst BG, dann FDV.

BG= FCPU / (32*Baudrate)-1

FDV= Baudrate * 512 *16 * (BG+1) / FCPU

ASC0 wird mit Interrupts sowohl für den Empfang als auch für das Senden betrieben. Dies ermöglicht gepuffertes Schreiben und Lesen. Die zugehörigen Interrupt-Register werden folgendermaßen initialisiert:
ASC0_RIC: 0068
ASC0_TIC: 0058

Die asynchrone serielle Schnittstelle nutzt natürlich 2 Ein/Ausgabeleitungen, P3.10 als Sendeleitung und P3.11 als Empfangsleitung. Daher müssen die Ein/Ausgabeleitungen konfiguriert werden:
P3: X4XX, P3.10 auf 1
DP3: X4XX, P3.10 als Ausgang, P3.11 als Eingang
ALTSELOP3: 0C00

Anbindung des Bluetoothmodules: ASC1

Die zweite asynchrone serielle Schnittstelle des xc167, ASC1, wird zur Anbindung des Bluetooth-Moduls verwendet. Die Baudrate ist für den Betrieb fest auf 921600 Baud eingestellt, wenngleich das Modul ASC1 wie jenes ASC0 für beliebige Baudraten ausgelegt ist.

Die Initialisierung von ASC1 erfolgt mittels Register ASC1_CON:
ASC1_CON: 8811, Baudratengenrator ein, Empfänger ein, kein Parity, 8 Datenbits, 1 Stopbit, Fractional Divider aktiviert

Die Baudrate ist abhängig von der CPU-Frequenz, von der Vorteiler-Einstellung und natürlich von der gewünschten Baudrate. Es muss einerseits das Baudraten-Nachladezählerregister BG und andererseits das Fractional Divider Register FDV gesetzt werden. Die tatsächliche Baudrate ergibt sich folgendermaßen:

Baudrate = FDV/512*FCPU/(16*(BG+1)

Während mit dem Register BG die Baudrate grob eingestellt wird, kann mit FDV die Abweichung der tatsächlichen Baudrate von der gewünschten minimiert werden. Vor allem bei hohen Baudraten kann mit BG alleine die Abweichung recht hoch sein.
Daher werden 2 Einstellungen benötigt: Register BG und Register FDV. Die beiden Register werden in 2 Stufen initialisiert, zuerst BG, dann FDV.

BG= FCPU / (32*Baudrate)-1

FDV= Baudrate * 512 *16 * (BG+1) / FCPU

ASC1 wird mit Interrupts sowohl für den Empfang als auch für das Senden betrieben. Dies ermöglicht gepuffertes Schreiben und Lesen. Die zugehörigen Interrupt-Register werden folgendermaßen initialisiert:
ASC1_RIC: 006C
ASC1_TIC: 005C

Die asynchrone serielle Schnittstelle nutzt natürlich 2 Ein/Ausgabeleitungen, P3.0 als Sendeleitung und P3.1 als Empfangsleitung. Daher müssen die Ein/Ausgabeleitungen konfiguriert werden:
P3: XXX1, P3.0 auf 1
DP3: XXX1, P3.0 als Ausgang, P3.1 als Eingang
ALTSELOP3: XXX3, Alternative Funktion für P3.0 und P3.1

Anbindung Externer Analog/Digitalwandler: SSC0

Über die SPI-kompatible synchrone serielle Schnittstelle SSC0 des xc167 ist der externe Analog/Digitalwandler angeschlossen. Dabei wird die SSC0 als Master betrieben, erzeugt also selbst den Takt. Die Baudrate ist fix auf 100 kBaud eingestellt.

Folgende Register werden gesetzt:
SSC0_CON: 4017, SSC0 im Master Mode, MSB wird zuerst gesendet, 8 Bit Datenbreite
SSC0_BR: 0027, Baudrate 100kBaud

Interrupts werden nicht eingesetzt, es gibt daher auch nur die 1 Byte große Bufferung.

Kommunikationsschnittstelle zu anderen Tinyphoon-Komponenten: CAN0

Der xc167 beinhaltet 2 CAN-Module, die entweder unabhängig voneinander arbeiten oder gekoppelt werden können: CAN Node A und CAN Node B.

Da der CAN-Bus als Kommunikationssystem zwischen den einzelnen Komponenten/Baugruppen von Tinyphoon angedacht ist, wird das Peripheriemodul CAN Node A eingesetzt. Im Modul CAN werden alle Initialisierungen für diese Peripherie durchgeführt.

Die Initialisierung geschieht in 2 Teilen: Erstens muss das CAN-Modul grundsätzlich aktiviert, die Baudrate eingestellt und die Interrupts freigeschaltet werden. Im zweiten Schritt werden einzelne Message Objekte für Empfang oder Senden bestimmter Message-Typen bzw. CAN-Message IDs konfiguriert.

Zuerst muss das CAN-Modul A initialisiert werden. Dies geschieht mit dem folgenden Registerzugriff:
CAN_ACR: 0041, CAN Modul startet Initialisierung, Zugriff zu allen Registern

Zum eigentlichen Empfang und Senden von CAN-Nachrichten müssen die sogenannten Message Objekte konfiguriert werden. Dabei handelt es sich im Prinzip um intelligente Buffer mit Filterfunktion. Insgesamt enthält der CAN-Block 32 Message-Objekte, die entweder zum Empfang oder zum Senden einer bestimmten Message ID oder aller Message IDs, die zu einer Bitmaske passen, konfiguriert werden können. In Tinyphoon wird für jede CAN Message ID ein eigenes Message Objekt eingesetzt, im Message Objekt ist immer der letzte Wert der jeweiligen CAN Message ID zu finden, egal ob dieser abgeholt wird oder nicht. Zusätzlich bieten alle Message Objekte eine Buffertiefe von mindestens 2 bis zu 32 Nachrichten – abhängig vom Message Objekt. Natürlich kann jedes Message Objekt auch einen Interrupt auslösen.
Die Message Objekte werden bei Tinyphoon mittels eigener Funktion gesetzt. Da mehrere CAN-Message IDs eingesetzt werden, wird hier die Registerzusammensetzung des CAN-Modules nur allgemein beschrieben (n ist die Message Objekt Nummer):
CAN_MSGDRHn4: Datenbytes D6 und D7 im Message Objekt n
CAN_MSGDRHn3: Datenbytes D4 und D5 im Message Objekt n
CAN_MSGDRHn2: Datenbytes D2 und D3 im Message Objekt n
CAN_MSGDRHn1: Datenbytes D0 und D1 im Message Objekt n
CAN_MSGARHn: Filterregister für CAN Message ID, Bits 16-28 für Message Objekt n
CAN_MSGARLn: Filterregister für CAN Message ID, Bits 0-15 für Message Objekt n

CAN_MSGAMRHn: Maskenregister für CAN Message ID, Bits 16-28 für Message Objekt n, alle Bits 1
CAN_MSGAMRLn: Maskenregister für CAN Message ID, Bits 0-15 für Message Objekt n, alle Bits 1
CAN_MSGCTRHn: Kontrollregister (obere 16 Bit) für Message Objekt n, enthält Rahmenzählerwert
CAN_MSGCTRLn: Kontrollregister (untere 16 Bit) für Message Objekt n, enthält diverse Flags für Ein/Ausschalten des Empfangs/Sende-Interrupt, anstehende Interrupts, anstehende Übertragung und die generelle Aktivierung für dieses Message Objekts.
CAN_MSGCFGHn: Konfigurationsregister (obere 16 Bit) für Message Objekt n, enthält Interrupt Node Id für Senden bzw. Empfangen.
CAN_MSGCFGLn: Konfigurationsregister (untere 16 Bit) für Message Objekt n, enthält Einstellungen, ob Message Objekt zum Senden oder Empfangen eingesetzt wird, ob Standard oder Extended Frames eingesetzt werden, die Datenbreite (1 bis 8 Bytes) und welchem CAN-Modul das Message Objekt zugeordnet ist.

Tinyphoon verwendet explizit die CAN Message IDs 1 und 4 (Standard Frame mit 11 Bit Message ID) mit jeweils 8 Bit Datenbreite. Für den Empfang aller restlichen CAN-Messages wird Message Objekt 15 verwendet, allerdings ist der Empfang aller anderen CAN-Messages derzeit deaktiviert.
Zum Senden von CAN-Messages werden keine Message Objekte vorbelegt, sondern diese bei der erstmaligen Sendeanforderung für eine bestimmte CAN-Message reserviert und konfiguriert. Sind alle Message Objekte bis auf Message Objekt 14 und 15 bereits belegt, so wird Message Objekt 14 für das Senden der CAN Nachricht verwendet. Daher wird in diesem Falle Objekt 14 immer wieder umkonfiguriert.

Für Empfang und auftretende Fehler werden Interrupts verwendet, für das Senden nicht. Das Interruptsystem des CAN-Blockes sieht vor, dass unterschiedlichen Interruptquellen (Senden, Empfangen, Fehler) bis zu 8 verschiedene, unabhängige Interrupt Nodes zugeordnet werden können. Tinyphoon verwendet Interrupt Node 0 zum Empfangen und Interrupt Node 7 für Fehler. Da aber jeder Interrupt einen unterschiedlichen Auslöser haben kann, muss unter Umständen der Auslöser von der Interrupt Service Routine erkannt werden. Diese Interrupt ID steht als Bitfelde INTID im Register CAN_AIR. Für Fehlermeldungen wird keine INTID generiert, da diese ohnehin einem eigenen Interrupt zugeordnet sind (Interrupt Node 7) und der Fehlerstatus aus dem Register CAN_ASR abgefragt werden kann. Dagegen ist für den Empfang von Nachrichten sehr wohl interessant, welches Message Objekt und damit letztendlich welche Message den Interrupt ausgelöst hat.
Für die umfangreiche Interruptkonfiguration müssen folgende Register gesetzt werden:
CAN_AGINP: X0X7, Can Interrupt Node 7 für Fehler, Can Interrupt Node 0 für Senden/Empfang
CAN_AIMRH0: FFFF, Für alle M essage Objekte wird INTID gesetzt
CAN_AIMRL0: FFFF, Für alle Message Objekte wird INTID gesetzt
CAN_AIMR4: 0000, für Fehlermeldungen und generell für Empfangen/Senden kein INTID Eintrag

CAN_0IC: 004C
CAN_7IC: 003F
CAN_AIR: 0000, alle anstehenden Interrupts löschen

Die Baudrate des CAN-Moduls wird aus mehreren Eingangs- und Timingparametern bestimmt: der CPU-Taktfrequenz, TSEG1, TSEG2 und BRP:

baudrate = fcpu/((TSEG1+TSEG2-3)*(BRP+1))

und daher

TSEG1+TSEG2 = fcpu/((brp+1)*baudrate)-3

Zusätzlich müssen folgende Bedingungen erfüllt werden:

Wenn BRP = 0: TSEG1+TSEG2 >= 5
Wenn BRP <>0: TSEG1+TSEG2 >= 3
TSEG1+TSEG2<=22
TSEG1 ~ 2*TSEG2

Die Baudrate wird bei Tinyphoon daher iterativ ermittelt. Dabei wird BRP von 0 bis 63 durchlaufen und jeweils TSEG1+TSEG2 nach obiger Formel ermittelt. Sobald alle Bedingungen erfüllt sind, wird das Register :
CAN_ABTRL und CAN_ABTRH gesetzt:
CAN_ABTRL: BRP+SJW <<6 + TSEG1 << 8 + TSEG2 << 12

Der CAN-Node A kann unterschiedlichen Ports zugeordnet werden. Im Falle von Tinyphoon ist dies Port 7.6.
CAN_PISEL: XXX8, Node A Input Pin ist P7.6
ALTSELOP7: XX8X, Alternativfunktion für Pin P7.7
DP7: XX8X, Leitung P7.7 für Ausgabe

Zu guter Letzt wird die Initialisierung des CAN-Moduls A abgeschlossen:
CAN_ACR: 0018, Ende der Initialisierung, Zugriff zu allen Registern gesperrt, Empfangs- und Fehlerinterrupt aktiviert.

Interrupts und Interruptprioritäten

Allgemeines zum Einsatz von Interrupts bei Tinyphoon

Das Interruptsystem des xc167 bietet 16 unterschiedliche Ausführungs-Prioritäten, und jede Interrupt Service Routine hat ihren eigenen Prozessorkontext (Registerbank). In Tinyphoon wird auf den Einsatz eines Echtzeitbetriebssystemes verzichtet, da alle Aufgaben rein

interruptgesteuert erledigt werden können. Anforderungen an „Echtzeitfähigkeit“ werden bequem mit hochprioren Timer-Tasks gelöst.
Daher läuft praktisch jeder Task von Tinyphoon innerhalb einer Interrupt Service Routine und damit auch auf unterschiedlichem Task-Level. Das Hauptprogramm selbst besteht – wie in vielen kleineren Embedded-Lösungen – aus einer Endlosschleife, in der gar nichts gemacht oder aufgerufen wird. Das Hauptprogramm entspricht also prinzipiell dem Idle-Task mit niedrigster Priorität.

Nachteile dieses Konzepts ergeben sich allerdings: Tasks können nur durch höherpriore Tasks unterbrochen werden. Daher ist grundsätzlich folgende Regel gelten: Je höher die Taskpriorität, desto kompakter sollte die Funktion sein. Ein Dilemma ergibt sich mit dem Regler-Task (CTR): Einerseits soll der Task möglichst präzise zeitgesteuert im Takt der Regelperiode und damit hochprior laufen, andererseits ist gerade dieser Teil relativ umfangreich und teilweise auch rechenintensiv. Da die Reglerperiode bei 2ms und damit in etwa bei der Granularität üblicher Echtzeitsysteme liegt, stellt sich die Situation nicht ganz so dramatisch dar.
Des weiteren würde der Einsatz eines Echtzeitbetriebssystems andere Funktionalitäten wie dynamische Speicherallozierung, Nachrichtensysteme und Prozesssynchronisationsmechanismen bieten. Auf diese Funktionalität wurde bei Tinyphoon daher verzichtet.

Interrupt-Quellen und Prioritäten

Grundsätzlich können in TinyPhoon folgende Peripherieeinheiten Interrupts generieren:
- Timer 7: Zeitgeber für Regelungsschleife
- Timer 5: Zeitgeber für globale Zeit
- Capture/Compare Unit 1 Kanal 12: Beschleunigungssensor 1 Kanal A
- Capture/Compare Unit 1 Kanal 13: Beschleunigungssensor 1 Kanal B
- Capture/Compare Unit 1 Kanal 14: Beschleunigungssensor 2 Kanal A
- Capture/Compare Unit 1 Kanal 15: Beschleunigungssendor 2 Kanal B
- ASC0 Empfang: Gepufferter Empfang und Notification bei Service Schnittstelle
- ASC0 Senden: Gepuffertes Senden bei Service Schnittstelle
- ASC1 Empfang: Gepufferter Empfang und Notification bei Bluetooth-Schnittstelle
- ASC1 Senden: Gepuffertes Senden bei Bluetooth-Schnittstelle
- CAN Empfang: Notification beim Empfang einer CAN Nachricht
- CAN Fehler: Fehlerhandling bei Fehler an CAN Schnittstelle

Um nun diesen Interruptquellen sinnvolle Prioritäten zuzuweisen, wurden folgende Grundregeln herangezogen:
- echtzeitrelevante Tasks hochprior
- Empfangsinterrupts an Schnittstellen hochprior
- Kurze Tasks höherprior als lange
- Sendeinterrupts an Schnittstellen niedrigprior
- Fehlerinterrupts höchstprior

Übersicht über alle genutzten Interrupts

Aus diesen Vorgaben wurden allen genutzten Interrupts folgende Prioritäten zugeordnet:

Quelle	*Modul*	*Priorität*	*IC-Register*
Timer 7	CTR	9(0)	CC2_T7IC = 0x0064
Timer 5	G_TIME	15(0)	GPT12E_T5IC = 0x007C
Capture Compare Einheit 1 – Kanal 12	ACC	14(0)	CC1_CC12IC = 0x0078
Capture Compare Einheit 1 – Kanal 13	ACC	14(1)	CC1_CC13IC = 0x0079
Capture Compare Einheit 1 – Kanal 14	ACC	14(2)	CC1_CC14IC = 0x007A
Capture Compare Einheit 1 – Kanal 15	ACC	14(3)	CC1_CC15IC = 0x007B
Asynchrone serielle Schnittstelle 0 - Empfang	ASC0	10(0)	ASC0_RIC = 0x0068
Asynchrone serielle Schnittstelle 0 - Senden	ASC0	6(0)	ASC0_TIC = 0x0058
Asynchrone serielle Schnittstelle 1 - Empfang	ASC1	11(0)	ASC1_RIC = 0x006C
Asynchrone serielle Schnittstelle 1 - Senden	ASC1	7(0)	ASC1_TIC = 0x005C
CAN Empfang = CAN Interrupt Node 0	CAN	3(0)	CAN_0IC = 0x004C
CAN Fehler = CAN Interrupt Node 7	CAN	15(3)	CAN_7IC = 0x007F

Tabelle 4.14: Übersicht über alle genutzten Interrupts

Alle Interrupt Kontrollregister (IC-Register) haben dabei den gleichen Aufbau:

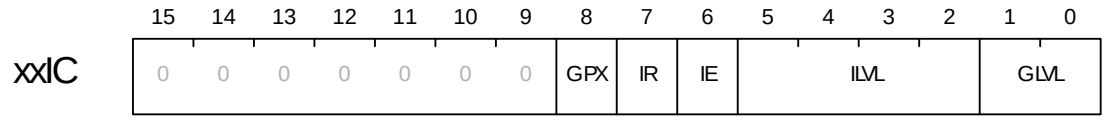

Abbildung 4.33: Grundsätzlicher Aufbau aller Interrupt Control Register

Das Bit IR wird dabei beim Auftreten eines Interrupt-Ereignisses hardwaremäßig gesetzt und bei Sprung in die Interrupt Service Routine auch automatisch gelöscht. Es kann grundsätzlich auch softwaremäßig gelöscht werden (z. B. bei Polling-Betrieb).
Das Bit IE wird softwaremäßig gesetzt, um den Interrupt zu aktivieren. Zusätzlich muss noch das globale Interrupt-Flag IEN im Register PSW gesetzt sein.
Das Bitfeld ILVL bestimmt die Interrupt-Priorität bzw. den „Task"-Level. Höherpriore Interrupts können niedrigpriore unterbrechen.
Das Bitfeld GLVL bestimmt die Sub-Priorität innerhalb einer Interrupt-Priorität. Diese Subpriorität dient lediglich dazu, um bei „gleichzeitigem" Auftreten zweier Interrupts der gleichen Priorität zu entscheiden, welcher zuerst bedient wird. Gleichpriore Interrupts können einander nicht unterbrechen, unabhängig vom Gruppenlevel. Interruptquellen gleicher Priorität sollten immer unterschiedlichen Gruppenlevel haben.

Das Bit GPX erweitert das Bitfeld GLVL um 1 Bit. Dabei ist das Bit GPX das hochwertigste Bit.

Übersicht über genutzte Peripherieeinheiten

Zusammenfassend nochmals alle von Tinyphoon genutzten Peripherieeinheiten:

Peripherieeinheit	*Modul*	*Einsatz*
Timer 7	CTR	Zeitgeber für digitalen Regler, auf 2ms konfiguriert
Timer 5	G_TIME	Zeitgeber für globale Zeit, 1ms Takt
Capture/Compare 1 – Kanal 12-15 mit Timer 1	ACC	Ausmessen der 4 PWM-Signale von den Beschleunigungssensoren
Asynchrone Serielle Schnittstelle 0	ASC0	Service- und Debugschnittstelle, 19200 Baud
Asynchrone Serielle Schnittstelle 1	ASC1	Schnittstelle zum Bluetooth-Modul, 921600 bzw. 115200 Baud
CAN-Modul	CAN	Kommunikationsschnittstelle zu anderen Komponenten von Tinyphoon (z. B. Blackfin)
Timer 2, Timer 4	INCR	Einlesen der Inkrementalgeberwerte der beiden Motoren zur Drehzahlbestimmung
Capture/Compare 1 – Kanal 8-11 mit Timer 0	PWM	Generierung von PWM-Signalen zur Ansteuerung der beiden Motoren
Synchrone Serielle Schnittstelle 0	SPI	Ansteuerung und Datenaustausch externer Analog/Digitalwandler
Ein/Ausgabe Port P6.6-P6.7	LED	Ausgabeports für Leuchtdioden
Ein/Ausgabe Port P9.2-P9.5	LED	Ausgabeports für Leuchtdioden
Ein/Ausgabe Port P3.6	KEY	Einlesen des Tasterzustandes
Ein/Ausgabe Port P5.0-P5.7	-	Einlesen der DIP-Schalter-Stellungen

Tabelle 4.15: Übersicht über die in den einzelnen Modulen genutzte Peripherie

Übersicht über alle genutzten Ein/Ausgabeports

Der xc167 steuert externe Komponenten über Ein/Ausgabeleitungen an. Dabei sind diese entweder als Eingang, als Ausgang, oder in einer Alternativfunktion konfiguriert.

Folgende Ein/Ausgabeleitungen werden von Tinyphoon genutzt:

Port	*I/O*	*Alt.fkt.*	*Modul*	*Einsatz*
P2.8	O	CC8IO	PWM	PWM Signal Motor 0 Drehrichtung 0
P2.9	O	CC9IO	PWM	PWM Signal Motor 0 Drehrichtung 1
P2.10	O	CC10IO	PWM	PWM Signal Motor 1 Drehrichtung 0

P2.11	O	CC11IO	PWM	PWM Signal Motor 1 Drehrichtung 1
P2.12	I	CC12IO	ACC	Capture Signal von Beschleunigungssensor 0 X-Out
P2.13	I	CC13IO	ACC	Capture Signal von Beschleunigungssensor 0 Y-Out
P2.14	I	CC14IO	ACC	Capture Signal von Beschleunigungssensor 1 X-Out
P2.15	I	CC15IO	ACC	Capture Signal von Beschleunigungssensor 1 Y-Out
P3.0	O	TxD1	ASC1	Sendeleitung von Modul ASC1 (für Bluetooth)
P3.1	I	RxD1	ASC1	Empfangsleitung von Modul ASC1 (f. Bluetooth)
P3.5	I	T4IN	INCR	Inkrementalgeber-Signal 0 für Motor 0
P3.6	I	-	KEY	Taster
P3.7	I	T2IN	INCR	Inkrementalgeber-Signal 0 für Motor 1
P3.8	I	MRST0	SPI	Empfangsleitung von Modul SPI (f. ext. ADC)
P3.9	O	MTSR0	SPI	Sendeleitung Modul SPI (f. ext. ADC)
P3.10	O	TxD0	ASC0	Sendeleitung von Modul ASC0
P3.11	I	RxD0	ASC0	Empfangsleitung von Modul ASC0
P3.13	O	SCLK0	SPI	Taktleitung für Modul SPI (f. ext. ADC)
P5.0	I	-	-	DIP switch 0 Eingang
P5.1	I	-	-	DIP switch 1 Eingang
P5.2	I	-	-	DIP switch 2 Eingang
P5.3	I	-	-	DIP switch 3 Eingang
P5.4	I	-	-	DIP switch 4 Eingang
P5.5	I	-	-	DIP switch 5 Eingang
P5.6	I	-	-	DIP switch 6 Eingang
P5.7	I	-	-	DIP switch 7 Eingang
P5.14	I	T4EUD	INCR	Inkrementalgeber-Signal 1 für Motor 0
P5.15	I	T2EUD	INCR	Inkrementalgeber-Signal 1 für Motor 1
P6.6	O	-	LED	Ausgang für LED5
P6.7	O	-	LED	Ausgang für LED6
P7.4	I	CAN2_RXD	CAN	Empfangsleitung CAN Modul 2
P7.5	O	CAN2_TXD	CAN	Sendeleitung CAN Modul 2
P7.6	I	CAN1_RXD	CAN	Empfangsleitung CAN Modul 1
P7.7	O	CAN1_TXD	CAN	Sendeleitung CAN Modul 1
P9.0	O	-	PWM	Aktivierung Motor 0
P9.1	O	-	PWM	Aktivierung Motor 1
P9.2	O	-	LED	Ausgang für LED1
P9.3	O	-	LED	Ausgang für LED2
P9.4	O	-	LED	Ausgang für LED3
P9.5	O	-	LED	Ausgang für LED4

5 Verbesserungspotentiale

High-Performance Prozessor

Einziger Nachteil des Controllers ist die fehlende Fixkomma- oder Fließkommaeinheit für Berechnungen. Daher muss der digitale Regler in Ganzzahlarithmetik realisiert werden. Dies führt auch zu einem erheblichen Nachteil bei der Berechnung der aktuellen Bewegung des Roboters und zu unlösbaren Performanceproblemen bei der Bahnplanung. Aus diesem Grund wird die nächste Generation des xc167er Boards mit einem BlackFin DSP ausgestattet. Der xc167 wird in Zukunft als intelligente Ein/Ausgabeeinheit fungieren. DSP und Mikrocontroller werden über SPI verbunden und alle rechenintensiven Aufgaben, wie etwas Positionsvorhersage, Bahnplanung und High-Advanced Regelung an den DSP ausgelagert.

Funkübertragung

Die verwendete Funkübertragung auf Basis von Bluetooth hat sich insofern als Schwachstelle für den Einsatz als Fußballroboter erwiesen, da der Aufbau einer Kommunikationsverbindung etliche Sekunden dauert. Aufgrund der Häufigkeit der zu sendenden Messages wurden diese broadcasts an alle Teammitglieder übertragen. Kommt es nun zu einem Abriss der Verbindung eines einzigen Empfängers wird beim Neuaufbau der Verbindung die gesamte Funkverbindung zu allen Robotern blockiert. Am besten bewährt haben sich bisher Systeme, die keine stehende Verbindung benötigen.

Bessere Nutzung der Beschleunigungssensoren

Derzeit werden die Informationen der Beschleunigungssensoren lediglich dafür genutzt, die Steilheit der Sollgrößenrampe zu steuern. Damit wurde die Anti-Schlupf-Regelung realisiert. Es ist denkbar, dass mit komplexeren oder alternativen Regelungen die Informationen der Beschleunigungssensoren direkt als Eingangsparameter dienen.

Komplexere Regelung

Der implementierte PID-Regler regelt die Geschwindigkeit des linken und des rechten Antriebsrades unabhängig voneinander. Damit wird im allgemeinen bei vorgegebener Geschwindigkeit und Winkelgeschwindikeit ein Rad vor dem anderen seine Sollgeschwindigkeit erreichen. Dies führt zu einer Richtungsabweichung von der tatsächlich gewünschten Bewegung. Eine potentielle Verbesserung wäre eine komplexere Regelung, bei der die Regelung der beiden Raddrehzahlen gekoppelt ist. Alternativ könnte eine bessere Positioniergenauigkeit erreicht werden, in dem nicht linke und rechte Raddrehzahl, sondern Geschwindigkeit und Winkelgeschwindigkeit geregelt wird.

Fehlerbehandlung

Derzeit inkludiert die Bewegungssteuerungssoftware nur eine rudimentäre Fehlerbehandlung. Zur besseren Diagnose von auftretenden Problemen und saubereren Implementierung könnte die Software um ein durchgängiges Fehlerbehandlungskonzept ergänzt werden. Dabei werden aufgetretene Fehler von allen Softwareschichten zu einer zentralen Fehlerbehandlungsroutine (Error Handler) weitergereicht und dort endgültig behandelt. Dadurch ist auch das Aufzeichnen sämtlicher Fehler oder auch nur bestimmter Fehlerklassen möglich.

Einsatz eines Betriebssystems

Derzeit werden alle Teilaufgaben der Roboter-Bewegungssteuerung mittels Ereignissen auf Interrupt-Basis abgehandelt. Dies hat den Vorteil, dass keinerlei Betriebssystem erforderlich ist und damit kein Performanceverlust durch Taskverwaltung und –umschaltung erlitten wird. Nachteil ist, das Tasks nur dann durch andere unterbrochen werden können, wenn diese höherprior sind. Die Taskabarbeitung ist daher rein prioritätengesteuert. Damit das Gesamtkonzept funktioniert, müssen die Tasks entsprechend schlank sein, zeitkritische Tasks müssen höhere Priorität haben.
Sollte die Aufgaben der Bewegungssteuerungs-Software wachsen, so sollte der Einsatz eines Echtzeitbetriebssystems erwogen werden. Betriebssysteme für embedded Plattformen bieten unterschiedliche Methoden der Taskpriorisierung und –umschaltung. Weitere Möglichkeiten von Echtzeit-Betriebssystemen sind Taskkommunikationsmethoden wie Message Queues oder Mailsysteme, dynamische Speicherverwaltung oder Semaphore.

6 Conclusio

Beim heutigen Stand von Wissenschaft und Technik ist die Robotik noch meilenweit davon entfernt, menschliche Ebenbilder zu schaffen. Es ist gerade einmal möglich, die Nachbildung der Bewegung von Menschen oder anderen Lebewesen einigermaßen effizient nachzubilden. Kommunikation muss mit Hilfe von eigens definierten technischen und einfachen Sprachen (Protokollen) abgewickelt werden.
An ein selbständiges Handeln bei nur grober Vorgabe der Aufgaben ist nicht zu denken. Auch Verstehen von menschlicher Sprache mit ihrer Mehrdeutigkeit, Redundanz und Unklarheit ist nach wie vor ein Traum.

Mobile Roboter werden daher erst durch einen hohen Spezialisierungsgrad und eingeschränkte, genau definierte Aufgaben effizient oder überhaupt erst realisierbar. Auch das Einsatzgebiet respektive die Umgebung, in der sich der Roboter befindet, muss genau definiert und der Roboter darauf optimiert sein.

Dann aber können sie – wie Computer – dem Menschen in der Erledigung einzelner Tätigkeiten überlegen sein: Schnellere Bewegung und Beschleunigung durch Räder, Genauere Ortung und Vermessung der bewältigten Wegstrecke durch genauere Sensorik, höherer Durchsatz bei der Aufnahme und Verarbeitung von Daten durch moderne Kommunikationstechnologien.

Im Falle von Tinyphoon, für den als eine konkrete Aufgabe und als ein konkretes Umfeld das Roboterfussballspiel definiert wurde, bedeuteten dies folgende Vorgaben: Hohe Beschleunigungsfähigkeit und hohe Endgeschwindigkeit bei der Bewegung auf einer geraden Ebene, schnelle Reaktionszeit bei geänderten Aufgaben sowie Möglichkeit der Ballführung und Ausführen von impulsartigen Bewegungen für Schüsse oder Passes.

Unsere Welt mit industrieller Produktion und arbeitskräfteschonenden Dienstleistungen über das Internet hat ohnehin schon einen sehr hohen Automatisierungsgrad erreicht und damit den Menschen grundsätzlich das Leben erleichtert – oder auch nicht.

Das Bild der Menschheit, die das ganze Jahr über Freizeitbeschäftigungen nachgeht, während Heerscharen von Robotern ihre Arbeit erledigen, ist daher ein fragwürdiges. Denn schließlich hat Arbeit neben der produktiven auch eine soziale Komponente: wir Menschen wollen etwas „Nützliches“ schaffen oder anderen nützlich sein, wollen mit anderen Menschen in Kontakt sein. Es nicht davon auszugehen, dass auf diesen Aspekt von Arbeit in Zukunft verzichtet werden kann oder Roboter echtes Sozialverhalten entwickeln können - Daher darf die Vision des Roboters als menschliches Ebenbild oder gar Ersatz ruhig noch länger Science Fiction bleiben.

Literaturverzeichnis

[JongHwan98] Kim, Jong-Hwan*: Lecture Notes on EE006 Robot Soccer System, KAIST, Taejon, Korea, 1998*

[Biribauer2002] Biribauer, Martin*: Ein Bildverarbeitungssystem zur Navigation mobiler Roboter, Diplomarbeit, Technische Universität Wien, März 2002*

[Novak02] Novak, Gregor*: Multi Agent Systems at the Practical Example of Robot Soccer, Dissertation, Technische Universität Wien, April 2002*

[Putz02] Putz, Bernhard*: Antriebsregelung eines Mobilen Kleinroboters, Diplomarbeit, Technische Universität Wien, April 2002*

[Fink03] Fink, Oliver*: Bluetooth Multipointkommunikation mobiler Kleinroboter, Diplomarbeit, Technische Universität Wien, August 2003*

[Infineon2006] Infineon Technologies: *XC167CI-16F Data Sheet, Infineon Technologies AG, Dezember 2006*

[Büttner] Büttner*: Digitale Regelungssysteme, Vieweg Verla , ISBN: 3-528-03041-0*

[Dudek2000] Dudek, Gregory / Jenkin, Michael: *Computational Principles of Mobile Robotics, Cambridge University Press, 2002*

[Nehmzow2000] Nehmzow, Ulrich: *Mobile Robotics: A Practical Introduction, Springer, 2000*

Abbildungsverzeichnis

Abbildung 1.1 Die Miniatur-Roboterplattform Tinyphoon *2*
Abbildung 2.1: Mobile Miniroboter spielen Fussball *6*
Abbildung 3.1:Teilaufgaben eines Roboters *8*
Abbildung 3.2: Elemente der Steuerungshardware *24*
Abbildung 4.1: Roboterfussball als Spielwiese der Wissenschaften *41*
Abbildung 4.2: Blockschaltbild des xc167 ..... *45*
Abbildung 4.3: Systemarchitektur und Register der xc167-CPU *48*
Abbildung 4.4: Speichermodell des xc167 ..... *50*
Abbildung 4.5: Das interne Flash des xc167 ..... *50*
Abbildung 4.6: Belegung des xc167 Speicherbereiches von C000-FFFF *51*
Abbildung 4.7: Beispielkonfiguration für das Register ADCIC *56*
Abbildung 4.8: Beispielkonfiguration für den A/D-Wandler: Register ADC_CON *58*
Abbildung 4.9: Beispielkonfiguration für die asynchrone serielle Schnittstelle: Register ASC0_CON *59*
Abbildung 4.10: Einstellen der Baudrate für die asynchrone Schnittstelle über das Register ASC0_BG *60*
Abbildung 4.11: Beispielkonfiguration für die serielle synchrone Schnittstelle 0 ..... *61*
Abbildung 4.12: Beispielkonfiguration für die Capture/Compare Einheit als ext. Interrupt *63*
Abbildung 4.13: Beispielkonfiguration für Timer 2 ..... *65*
Abbildung 4.14: Tinyphoon in Aktion *67*
Abbildung 4.15: Komponentenaufbau des Tinyphoons *68*
Abbildung 4.16: Mikrocontroller Blockschaltbild *72*
Abbildung 4.17: Die Tinyphoon Mikrocontrollerplatine ubx167 ..... *73*
Abbildung 4.18: Aufgaben der Software der Tinyphoon Bewegungseinheit *77*
Abbildung 4.19: Einfacher Impuls zur Drehzahlmessung *79*
Abbildung 4.20: mehrere Impulse pro Umdrehung zur Drehzahlbestimmung *80*
Abbildung 4.21: Inkrementalgebersignal *80*
Abbildung 4.22: Grundschema des PID - Reglers *82*
Abbildung 4.23: Sollgröße ohne Rampe *82*
Abbildung 4.24: Sollgröße mit Rampe *83*
Abbildung 4.25: Softwaremodule und ihre Kommunikation *88*
Abbildung 4.26: Ablauf der Programminitialisierung *93*
Abbildung 4.27: Konfiguration des externen Busses – Adressbereich für das externe RAM *110*
Abbildung 4.28: Konfiguration des externen Busses – Busmodus und Zugriff für das externe RAM *111*
Abbildung 4.29: Konfiguration des externen Busses – Timing für das externe RAM *111*
Abbildung 4.30: Berechnung der Taktfrequenz abhängig von unterschiedlichen PLLCON-Parametern *111*
Abbildung 4.31: Konfiguration des Registers PLLCON für Tinyphoon *112*

Abbildung 4.32: Überprüfung der PLLCON Konfiguration 112
Abbildung 4.33: Grundsätzlicher Aufbau aller Interrupt Control Register 123

Tabellenverzeichnis

Tabelle 3.1: Vergleich der Roboteraufgaben mit menschlichen Funktionen *9*
Tabelle 3.2: Vergleich der Bewegungsarten *12*
Tabelle 3.3: Einsatzgebiete von Sensortypen für verschiedene Roboteraufgaben *15*
Tabelle 3.4: Vergleich der unterschiedlichen Prozessortypen *27*
Tabelle 3.5: Vergleich der unterschiedlichen Speicherarten *31*
Tabelle 3.6: Anwendungsgebiete von unterschiedlichen Peripherietypen *39*
Tabelle 4.1: Erfüllung der Anforderungen durch die gewählte Plattform *43*
Tabelle 4.2: Mikrocontroller Pinbelegung *76*
Tabelle 4.3: Übersicht über alle Module der TINYPHOON Firmware *91*
Tabelle 4.4: Aufrufschnittstelle des Moduls ASC0 ..... *95*
Tabelle 4.5: Aufrufschnittstelle des Moduls ASC1 ..... *97*
Tabelle 4.6: Aufrufschnittstelle des Moduls PWM *99*
Tabelle 4.7: Aufrufschnittstelle des Moduls INCR *100*
Tabelle 4.8: Aufrufschnittstelle des Moduls LED *102*
Tabelle 4.9: Aufrufschnittstelle des Moduls KEY *103*
Tabelle 4.10: Aufrufschnittstelle des Moduls SPI *104*
Tabelle 4.11: Aufrufschnittstelle des Moduls ACC *105*
Tabelle 4.12: Aufrufschnittstelle des Moduls FLASH *107*
Tabelle 4.13: Aufrufschnittstelle des Moduls MISC *109*
Tabelle 4.14: Übersicht über alle genutzten Interrupts *123*
Tabelle 4.15: Übersicht über die in den einzelnen Modulen genutzte Peripherie *124*