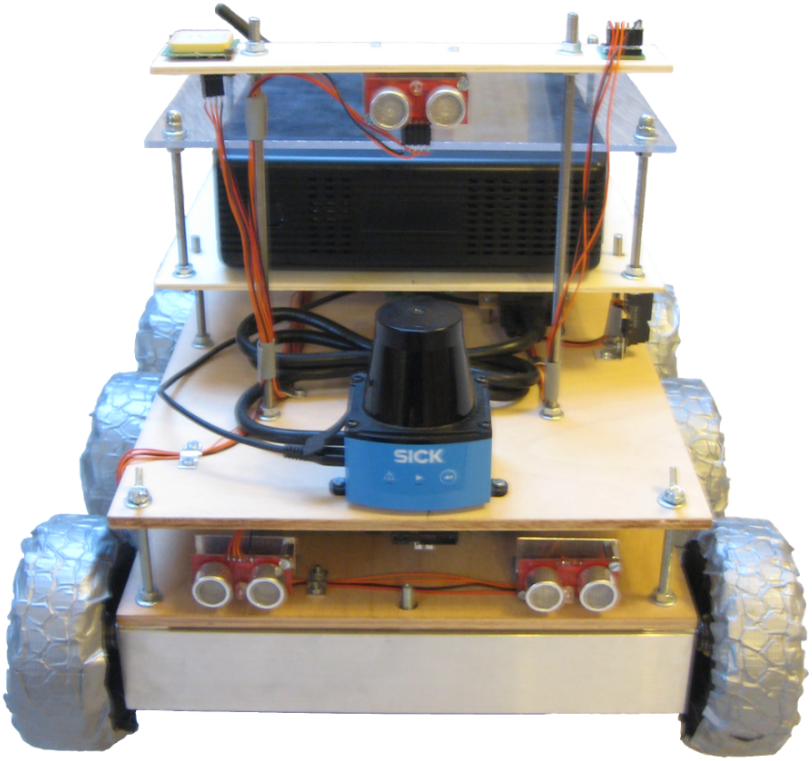
Besondere Lernleistung

Fach Informatik/Physik/Mathematik

Bau und Programmierung eines autonomen Erkundungsroboters



Jannik Springer

Jgst. 12

Betreuender Lehrer: Herr Fassbender

Inhaltsverzeichnis

Abbildungsverzeichnis 4

1. Vorwort 5

2. Einleitung 6

2.1. Themenfindung . . . 6
2.2. Zielsetzung . . . 7
2.3. Zusammenfassung . . . 7

3. Hardware/Mechanik 8

3.1. Gehäuse . . . 8
3.2. Batterien . . . 8
3.3. Motoren . . . 9
3.3.1. PWM . . . 9
3.3.2. Regelung . . . 10

4. Elektronik 13

4.1. Prozessoren . . . 13
4.1.1. AVR . . . 13
4.1.2. ARM . . . 16
4.2. Sensoren . . . 17
4.2.1. Ultraschall . . . 17
4.2.2. Sharp - Entfernungsmesser . . . 18
4.2.3. GPS . . . 20
4.2.4. Kompass . . . 21
4.2.5. Strom . . . 21
4.2.6. Encoder . . . 21

5. Programmierung 23

5.1. Kommunikation . . . 23
5.1.1. TCP/IP - Mainboard . . . 24
5.1.2. I^2C . . . 29
5.1.3. RS-232 . . . 31

5.2. Hauptprogramm 32
5.2.1. Interrupts 34

6. Probleme **36**
6.1. PID - Regler 36
6.2. TCP/IP - Stack 38
6.3. I^2C - Bibliothek 39

7. Fazit **40**

Literaturverzeichnis **42**

A. Anhang **45**
A.1. Inhaltsverzeichnis der CD 45
A.2. Benutzte Software 45

Abbildungsverzeichnis

2.1. Aufgabenverteilung 6

3.1. PWM-Signal [1] 9
3.2. P-Regler Sprungantwort [2] 10
3.3. I-Regler Sprungantwort [3] 11
3.4. D-Regler Sprungantwort [4] 11
3.5. PID-Regler Sprungantwort [5] 12

4.1. SAR ADC [6] 14
4.2. TWI - I^2C Datentransfer [7] 15
4.3. Funktionsweise Sharp [8] 18
4.4. Output des Sensors [9] 19
4.5. GPS Funktionsprinzip [9] 20
4.6. Encoder Oszillogramm [10] 22
4.7. Dioden Oder-Gatter [11] 22

5.1. LPCXpresso Development Board [12] 23
5.2. Vereinfachtes Schichten Modell 24
5.3. Ethernet Frame [13] 25
5.4. TCP/IP Header [14] 26
5.5. TCP-Verbindungsaufbau [15] 26
5.6. Eigenes Protokoll 28
5.7. UART Timing [16] 32
5.8. Grober Programm Ablauf Plan 33

6.1. Dimensionierung nach der Sprungantwort [17] 36
6.2. Sprungantwort des Motors 37
6.3. Pull-Up-Widerstände 39

1. Vorwort

Das besondere Interesse meines Bruders zur Programmierung und meines für die Elektronik, insbesondere Robotik, hat uns beide dazu gebracht ein gemeinsames Projekt zu starten und dieses in der Schule als „Besondere Lernleistung“ anzumelden. Auf diese Weise erhält unser Abitur einen deutlichen naturwissenschaftlichen Schwerpunkt.
Dieses Projekt muss ein Jahr dauern, in Eigenleistung erstellt werden und es sollte eine getrennte Bewertung möglich sein.
Wir haben uns entschlossen einen autonomen Erkundungsroboter zu bauen. Aufgrund unserer verteilten Interessengebiete, haben wir die Aufgabenbereiche sehr gut aufteilen können.
Grob beschrieben hat mein Bruder die Software, durch die der Roboter unbeweglichen und auch mobilen Hindernissen selbständig ausweicht und zu einer vorgegebenen GPS-Koordinate fährt, entwickelt. Außerdem hat er die Schnittstelle zu einem PC, von dem aus man Befehle erteilen kann und die Sensordaten zur grafischen Anzeige bringt, programmiert.
Mein Teil der Aufgaben ist zunächst die sorgfältige Auswahl aller Hardware- und Elektronikkomponenten, die optimal zusammen spielen müssen, der mechanische Aufbau des gesamten Chassis des Roboters, Entwicklung von Platinen und die dazugehörigen Lötarbeiten, sowie die Programmierung der Mikrocontroller, die die Motoren ansteuern und sämtliche Sensordaten auslesen, gewesen.
An dieser Stelle möchte ich meinem Physik- und Informatiklehrer, Herrn Faßbender, danken, da er uns dieses Projekt ermöglicht hat. Weiterer Dank gilt unseren Sponsoren, der Firma SICK und der Stiftung Evolution, ohne die wir kein ausreichendes Budget zur Verfügung gehabt hätten und zuletzt noch unseren Eltern, die uns immer ermutigt und mit Nervennahrung versorgt haben.
Dieser Text wurde mit LaTeX erstellt.

2. Einleitung

2.1. Themenfindung

Durch meine langjährige Erfahrung mit Mikroprozessoren der AVR-Familie, beschäftigt mich das Thema Roboter schon etwas länger. Nachdem mein Bruder und ich erfolgreich an einem Wettbewerb teilgenommen haben, bei dem wir einen Roboter entwickelt haben, der sich mit Hilfe eines EMGs steuern ließ, haben wir uns dazu entschlossen eine „Besondere Lernleistung“ durchzuführen. Obwohl ich schon kleinere autonome Roboter mit Hilfe der oben genannten AVRs entwickelt habe, ist auch dieser Bereich recht neu für mich gewesen, denn bei dem Roboter ist ein ARM-Prozessor zum Einsatz gekommen, wie er auch beispielsweise in iPhones benutzt wird. Die eigentliche Intelligenz ist jedoch auf einem mini-ITX Board implementiert, was den Roboter mit einer enormen Rechenleistung ausstattet. Die Programmierung dieses Mainboards hat mein Bruder übernommen, sodass zwei klare Aufgabenbereiche entstanden sind, die sich jedoch durch eine Schnittstelle miteinander verknüpfen lassen.

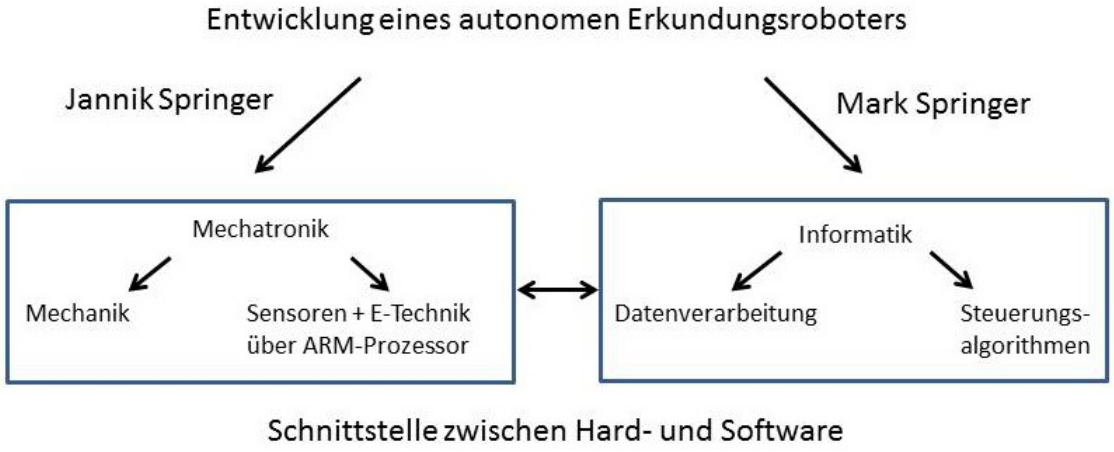


Abbildung 2.1.: Aufgabenverteilung

2.2. Zielsetzung

Ziel der Arbeit ist es einen Roboter zu konstruieren, der autonom nach GPS-Daten fahren kann. Da dies ein sehr ambitioniertes Ziel ist, wird vorausgesetzt, dass das Gelände von dem Roboter gut befahrbar ist. Außerdem soll der Roboter sowohl beweglichen, wie zum Beispiel Menschen, als auch unbeweglichen, wie zum Beispiel einem Baum, Hindernissen ausweichen können. Über WLAN (**W**ireless **L**ocal **A**rea **N**etwork) sollte eine Verbindung vom Roboter zu einem PC hergestellt werden. Dort sollen, mit einer selbst programmieren GUI (**G**rafical **U**ser **I**nerface), wichtige Laufzeitparameter eingestellt und abgefragt werden können.

2.3. Zusammenfassung

Wir wollten also einen Roboter entwickeln, dem man eine GPS Koordinate übermittelt, zu der er dann autonom hinfindet. Bestückt mit mehreren unterschiedlichen Sensoren kann das für einige Bereiche in unserer Gesellschaft interessant sein. Beispielsweise für den militärischen Bereich, da hier Roboter benötigt werden, die zum Beispiel autonom einem Konvoi folgen und Ausrüstungsgegenstände transportieren. Aber auch in anderen Bereichen ist diese Roboterplattform universell einsetzbar.
Meine Aufgabe besteht darin ein geeignetes Chassis zu konstruieren und die Sensoren sowie Aktoren mit dem ARM Prozessor auszulesen bzw. anzusteuern. Danach müssen diese Daten über Ethernet an das Mainboard weitergeleitet werden, damit der ARM daraufhin Motordaten von dem Mainboard entgegennehmen kann. Das Auswerten muss in entsprechender Geschwindigkeit geschehen, was eine besondere Herausforderung darstellt, da die Motoren ebenfalls geregelt werden müssen. Auf der Internetseite mjspringer.wordpress.com haben wir seit Beginn des Projektes unsere Fortschritte dokumentiert.

3. Hardware/Mechanik

3.1. Gehäuse

An das Chassis werden besondere Anforderungen gestellt, denn es muss gleichzeitig stabil und exakt manövrierbar sein. Nachdem zu Beginn mit dem Gedanken gespielt worden ist, ein fertiges Chassis zu kaufen, ist schnell klar geworden, dass ein gekauftes Chassis nicht den Anforderungen gerecht werde und zu teuer sei.
Da wir nicht in der Lage sind Metall zu verarbeiten, müssen wir uns auf Aluminiumprofile beschränken. Diese bilden die Grundstruktur des Roboters und halten die Motoren. Die darüber liegenden Ebenen werden aus Holz gemacht, weil es einfach zu verarbeiten und leicht ist. Insgesamt sind die Ebenen mit Hilfe von Gewindestangen und Muttern fixiert, was eine sehr stabile Konstruktion ergibt.

3.2. Batterien

Auf der Roboterplattform kommen zwei verschiedene Akkumulatoren zum Einsatz. Während für die Motoren ein Akku benötigt wird, der hohe Spitzenströme liefern kann, muss das Mainboard lange Zeit mit einem konstanten Strom versorgt werden. Da sich beide Eigenschaften nicht in einem Akku vereinen lassen und die Handhabung von zwei verschiedenen Akkus einfacher ist, haben wir uns für verschiedene Akkus entschieden.
Deswegen setzen wir für die Motoren einen LiPo-Akku aus dem Modellbau und für das Mainboard und die andere Elektronik, einen externen Notebook-Akku ein.

3.3. Motoren

Der Roboter hat insgesamt 6 Räder, die jeweils von einem Motor angetrieben werden. Bei den Motoren handelt es sich um 12 Volt Gleichstrommotoren mit Getriebe. Diese sind einfach anzusteuern und besitzen das ausreichende Drehmoment, um den 8 Kg schweren Roboter auf bis zu 1,5 m/s zu beschleunigen. Die Motoren haben bereits einen Hall-Sensor als Encoder eingebaut, dieser liefert 1856 Ticks pro Umdrehung.

3.3.1. PWM

PWM steht für **P**uls **W**eiten **M**odulation, das generierte PWM-Signal dient dazu die Motoren kontrolliert zu steuern. Um die Geschwindigkeit der Motoren einzustellen, wird entweder eine analoge Spannung oder ein PWM-Signal benötigt. Bei dem PWM-Signal wird der Motor für eine bestimmte Zeit in einer festgelegten Periode mit derselben Spannung eingeschaltet. Denn für einen Mikroprozessor ist es wesentlich einfacher Pulse mit einem definierten Puls/Pausen Verhältnis zu erzeugen als eine Spannung zu variieren.

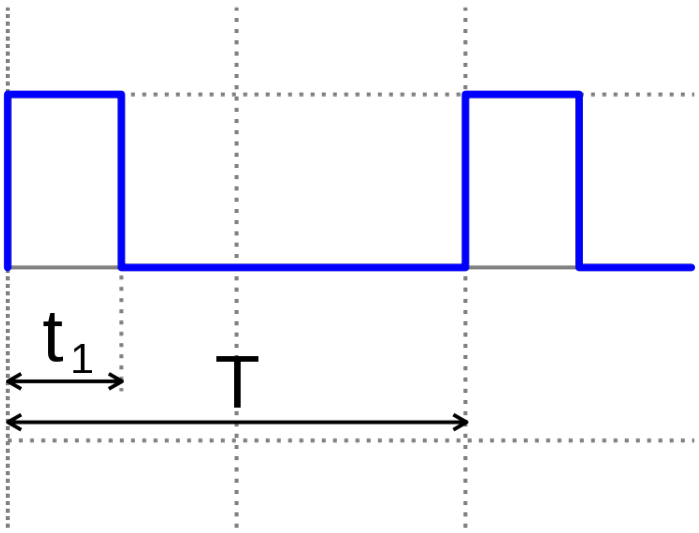


Abbildung 3.1.: PWM-Signal [1]

In Abbildung 3.1 sieht man ein PWM-Signal mit einem Duty-Cycle von 25%, d.h. t_1 ist genau ein viertel so lang wie T. Der ARM Prozessor erzeugt ein 10Bit PWM-Signal, mit einer Frequenz von 1 kHz. Das bedeutet, dass sich die Motoren in 1024 verschiedenen Stufen bzw. Geschwindigkeiten ansteuern lassen. Zu hohe Frequenzen können bei Motoren zu einem Brummen führen, weshalb hier eine Frequenz von 1 kHz gewählt worden ist.

3.3.2. Regelung

Um den Roboter vernünftig navigieren zu können, ist es sinnvoll dafür zu sorgen, dass die Motoren immer mit einer konstanten Geschwindigkeit drehen. Dies ist beispielsweise wichtig, wenn der Roboter auf eine Steigung trifft, ein PID-Regler sorgt dafür, dass die Motoren die Drehzahl beibehalten.

> „Ein Regelkreis dient dazu, eine vorgegebene physikalische Größe (Regelgröße) auf einen gewünschten Wert (Sollwert) zu bringen und dort zu halten, unabhängig von eventuell auftretenden Störungen. Um die Regelungsaufgabe zu erfüllen, muss der Augenblickswert der Regelgröße - der Istwert - gemessen und mit dem Sollwert verglichen werden. Bei auftretenden Abweichungen muss in geeigneter Art und Weise nachgestellt werden.“ [18]

In diesem Fall entspricht die geeignete Art und Weise einem PID-Regler, d.h. der Regler besitzt einen Proportional-, Integral- und Differantialanteil.
Die Stellgröße, d.h. der Ausgang des Reglers wird im Folgenden als $u(t)$ bezeichnet. Die Regelabweichung $e(t)$ berechnet sich aus der Differenz des Sollwertes w und des Istwertes x, als Formel: $e(t) = w - x$. In der analogen Regelungstechnik ist es üblich, die Vorhaltezeit T_{v} und die Nachhaltezeit T_{n} anzugeben, dies ist in der digitalen Regelung nicht notwendig.

P-Regler

Der P-Regler besitzt nur ein P-Glied und reagiert proportional mit der Verstärkung K_{p} zur Regelabweichung. In dem folgenden Bild sieht man die klassische Sprungantwort eines P-Reglers.

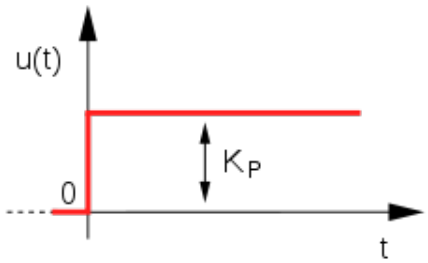


Abbildung 3.2.: P-Regler Sprungantwort [2]

Die Reglergleichung lautet demnach: $u(t) = K_{\mathrm{p}} * e(t)$.

I-Regler

Im Gegensatz zum P-Regler besteht der I-Regler aus einem I-Glied, dieser integralwirkende Regler summiert die Regelabweichung über der Zeit auf und reagiert entsprechend dem Verstärkungsfaktor K_i. Der Regler reagiert also stark auf lange Regelabweichungen, denn je länger eine Regelabweichung vorhanden ist, desto größer wird das Integral und damit die Stellgröße des Reglers. Im folgenden Bild sieht man die klassische Sprungantwort eines I-Reglers.

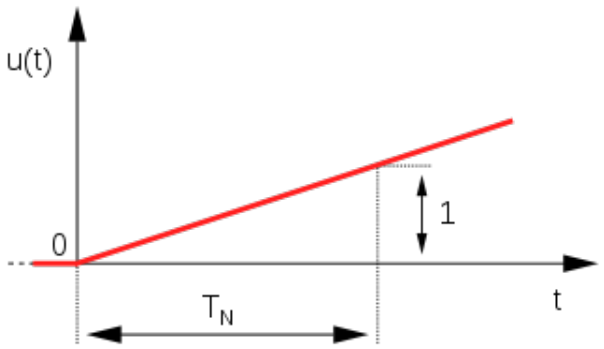


Abbildung 3.3.: I-Regler Sprungantwort [3]

Daraus ergibt sich folgende Reglergleichung: $u(t) = K_i * \int_0^t e(t)dt$.

D-Regler

Der D-Regler besteht aus einem einzelnen D-Glied und wird nur zur Ergänzung von P- und I-Reglern eingesetzt. Im Unterschied zu den beiden vorher genannten Reglern, spricht der D-Regler nicht auf die Größe der Regelabweichung, sondern auf die Änderungsgeschwindigkeit der Regelabweichung an. Im folgenden Bild sieht man die klassische Sprungantwort eines D-Reglers.

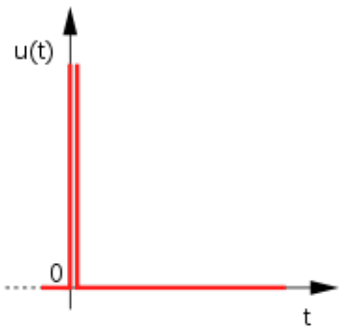


Abbildung 3.4.: D-Regler Sprungantwort [4]

Folgende Gleichung beschreibt das Verhalten des Reglers: $u(t) = K_d * \frac{d}{dt}e(t)$

PID-Regler

Der PID-Regler kombiniert alle Eigenschaften der vorher genannten Regler. Obwohl der Regler sowohl als Parallel-, als auch als Reihenstruktur ausgelegt werden kann, macht es hier keinen Unterschied, ob die verschiedenen Anteile parallel oder in Serie geschaltet werden. Im Vergleich zu anderen Reglerstrukturen reagiert er sehr schnell ohne großes „Überschwingen“ beim Nachregeln. Ein PID-Regler ist jedoch sehr schwer zu dimensionieren, besonders für Unerfahrene.

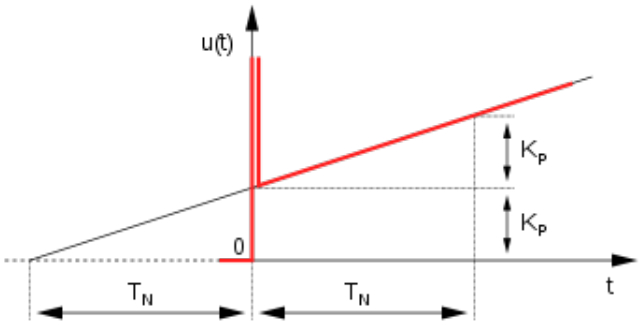


Abbildung 3.5.: PID-Regler Sprungantwort [5]

Die Gleichung für den PID-Regler setzt sich aus allen oben genannten Gleichungen zusammen: $u(t) = K_\mathrm{p} * e(t) + K_\mathrm{i} * \int_0^t e(t)dt + K_\mathrm{d} * \frac{d}{dt}e(t)$.

Die Reglergleichung ist in vereinfachter Form dargestellt und kann, so wie sie ist in Sourcecode umgesetzt werden. Wie man sieht, müssen drei verschiedene Koeffizienten bestimmt werden, damit der Regler vernünftig funktioniert. Dafür haben sich verschiedene Verfahren wie zum Beispiel die Ziegler-Nichols-Methode bewährt.

4. Elektronik

4.1. Prozessoren

Auf dem Roboter kommen zwei verschiedene Prozessor-Familien zum Einsatz, zum einen die AVR-Familie und zum anderen ein ARM-Prozessor, wie er zum Beispiel auch in iPods Verwendung findet. Während die AVRs nur als Slaves eingesetzt werden, koordiniert der ARM alles andere.

4.1.1. AVR

AVR steht laut ATMEL für **A**dvanced **V**irtual **R**ISC (**R**educed **I**nstruction **S**et **C**omputing) und wurde von Alf Egin Bogen und Vegrad Wollan entwickelt. Diese 8-Bit Mikrocontroller werden meist mit 5 Volt betrieben und können, mit bis zu 32 MHz getaktet, circa 110 verschiedene Befehle ausführen. Dazu stehen dem AVR 32 größtenteils gleichwertige Register zur Verfügung. Die Programme auf den Prozessoren sind in der weit verbreiteten Programmiersprache C geschrieben, dabei kommt das AVR-Studio zusammen mit AVR-GCC zum Einsatz.

Aufgaben

Derzeit werden zwei verschiedene Controller auf dem Roboter eingesetzt, einmal ein ATtiny44 und ein ATmega8. Wie schon erwähnt, dienen die AVRs als Slaves und werden wegen ihrer Bertriebsspannung von 5 Volt als AD-Wandler benutzt, da der ARM mit 3,3 Volt arbeitet und so keine Spannungen größer als die Betriebsspannung messen kann. Beide Mikrocontroller werden per I^2C beziehungsweise TWI mit dem ARM-Prozessor verbunden.
Der ATtiny44 wird so programmiert, dass er bis zu fünf analoge Spannungen einlesen kann und diese weiter an den Hauptprozessor gibt, der die digitalisierten Spannungen weiterverarbeitet. Insbesondere liest der ATtiny44 die Sharp Sensoren aus, die eine analoge Spannung proportional zur gemessenen Entfernung ausgeben.
Der ATmega8 sitzt in der Motor-Ebene und überwacht die Spannungen jeder Zelle des LiPo Akkus. Außerdem gibt der Motortreiber analoge Spannungen aus, die proportional zum Motorstrom sind, diese werden ebenfalls von dem AVR überwacht

und über I^2C an den ARM-Prozessor weiter gesendet. Es ist wichtig, die analogen Spannungen nahe der Quelle auszulesen, denn lange Kabel sorgen für viele Störsignale auf dem eigentlichen Signal, deswegen sitzt der ATmega8 in der Motor-Ebene.

ADC

Bei dem ADC beziehungsweise ADW handelt es sich um einen **A**nalog **D**igital **C**onverter/**W**andler, der, wie der Name schon sagt, eine analoge Spannung in einen digitalen Wert umwandelt. Ein ADC benötigt eine Referenzspannung, die bei den AVRs meistens intern bereitgestellt wird.
Der ADC des AVR beruht auf dem Prinzip der Sukzessiven Approximation. Dafür wird ein DAC beziehungsweise DAW (**D**igital **A**nalog **C**onverter/**W**andler) benötigt. Mit Hilfe des SAR (**S**uccessive **A**pproximation **R**egister) und einem Komparator tastet sich der ADC langsam an die angelegte Spannung heran. Das folgende Blockschaltbild beschreibt diesen Aufbau:

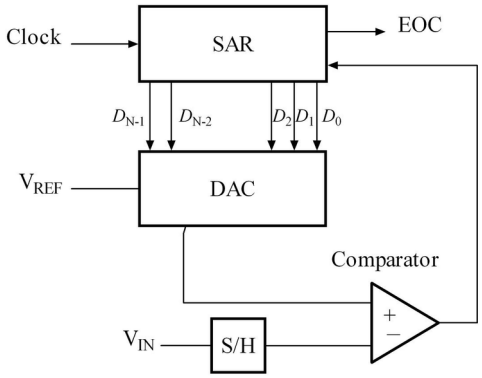


Abbildung 4.1.: SAR ADC [6]

Das Datenwort D ist gleichzeitig der Rückgabewert des ADCs, dieses ist bei den AVRs 10Bit lang, die Mikroprozessoren können damit Spannungen bis zu 4,883 mV auflösen. Das berechnet sich wie folgt $\frac{5V}{2^{10}} = \frac{5V}{1024} \approx 4,883$ mV, liegt am Eingangspin eine Spannung von 0 V an, so liefert der ADC einen Wert von 0. Liegt eine Spannung von 5 V, die der Referenzspannung entspricht an, so liefert der ADC einen Wert von 1023.

TWI - I^2C

TWI steht für **T**wo **W**ire **I**nterface und ist von ATMEL eingeführt worden, um Probleme mit dem Warenzeichen I^2C (**I**nter-**I**ntegratet **C**ircuit) zu vermeiden. Beide Bussysteme sind jedoch kompatibel, obwohl TWI keinen „High speed mode“ besitzt. Dieser wird für diese Anwendung allerdings auch nicht benötigt. Das Besondere bei einem Bus ist, dass sich alle Busteilnehmer die (in diesem Fall) Takt- (SCL) und Datenleitung (SDA) teilen. Da alle TWI-Geräte „Open-Collector-Ausgänge“ besitzen, ist es wichtig, die beiden Leitungen mit Hilfe eines Pull-up-Widerstandes auf den Pegel der Versorgungsspannung (V_{CC}) zu ziehen
TWI ist als Master-Slave-Bus entwickelt worden, das bedeutet, dass eine Übertragung immer vom Master initiiert wird.

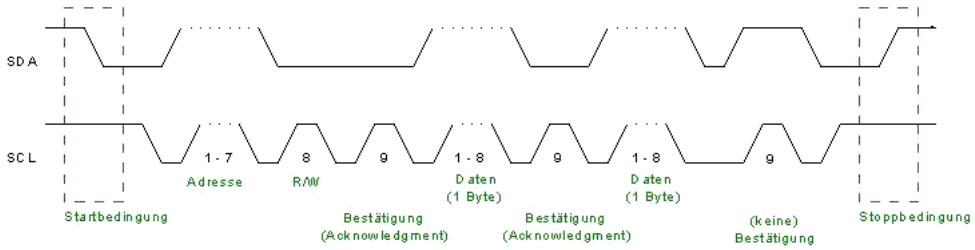


Abbildung 4.2.: TWI - I^2C Datentransfer [7]

Wie in Abbildung 4.2 zu erkennen ist, gibt der Master als erstes eine Startbedingung aus. Danach überträgt er eine 7 Bit lange Slaveadresse und ein R/W Bit, mit der er bestimmen kann, ob er den Slave im Schreibe- oder Lesemodus adressiert. Mit einer 7 Bit langen Adresse können bis zu 127 Slaves adressiert werden, außerdem gibt es noch Multi-Master Systeme, bei denen es mehrere Master gibt. Nach dem der Slave seine Adresse erkannt hat, muss er dies durch das Acknowledgement-Bit bestätigen. Nun hat der Master die Möglichkeit Daten zu senden beziehungsweise zu empfangen, mit der Stoppbedingung signalisiert der Master das Ende eines Transfers.

4.1.2. ARM

Bei den ARM-Prozessoren handelt es sich um so genannte „IP-Cores“, denn die Firma selber stellt keine Prozessoren aus Silizium her, sondern vergibt Lizenzen an Hersteller wie zum Beispiel NXP. Diese Hersteller ergänzen die CPU um Peripherie und Speicher, der hier verwendete LPC1769 besitzt einen Cortex-M3 Prozessor und wurde um 512 kB Flash und 32 kB SRAM ergänzt. Der mit bis zu 120 MHz getaktete Prozessor besitzt eine Vielzahl an Peripherie, unter anderem drei I^2C Interfaces, ein Ethernet MAC Controller (kein PHY), vier UARTS und bis zu 70 GPIO (**G**eneral **P**urpose **I**/**O**). Eine JTAG Schnittstelle dient dem Programmieren und Debuggen und die Betriebsspannung von 3,3 V sorgt für einen moderaten Stromverbrauch.

Aufgaben

Die Aufgabe des LPC1769 besteht darin, alle Sensordaten auszuwerten und über Ethernet an das Mainboard weiter zu senden. Dann muss der Prozessor Motordaten von dem Mainboard entgegen nehmen und diese in ein PWM-Signal für die Motoren umwandeln, dabei kommen mehrere Timer zum Einsatz. Der Prozessor regelt die Motoren mit Hilfe des vorher beschriebenen PID-Reglers und wertet die von den AVRs übermittelten analogen Spannungen aus. Außerdem sind alle unten aufgeführten Sensoren an den Prozessor angeschlossen.

Clock

Im Gegensatz zum AVR besitzt der ARM eine PLL (**P**hase-**L**ocked **L**oop), durch die der ARM zu seiner bis zu fünf Mal höheren Taktrate kommt.

> „Eine Phasenregelschleife, auch als englisch phase-locked loop (PLL) bezeichnet, ist eine elektronische Schaltungsanordnung, die die Phasenlage und damit zusammenhängend die Frequenz eines veränderbaren Oszillators über einen geschlossenen Regelkreis so beeinflusst, dass die Phasenabweichung zwischen einem äußeren Referenzsignal und dem Oszillator oder einem daraus abgeleiteten Signal möglichst konstant ist.“[19]

Obwohl ein externer Quarz mit einer Frequenz von 12 MHz angeschlossen ist, wird die Frequenz mit Hilfe der PLL mit dem Faktor zehn multipliziert. Das Initialisieren der PLL ist das Erste, was in dem Programm gemacht werden muss, danach wird das restliche Programm mit einer Taktfrequenz von 120 MHz ausgeführt.

4.2. Sensoren

Damit der Roboter seine Umgebung wahrnehmen kann, verfügt er über eine Vielzahl von Sensoren. Obwohl der SICK Laserscanner einer der wichtigsten Sensoren ist, wird er unten nicht aufgeführt, weil der Laserscanner über USB direkt an das Mainboard angeschlossen wird.

4.2.1. Ultraschall

Als Ultraschallsensor kommt hier der SRF08 zum Einsatz. Dieser misst die Entfernungen zu Objekten. Insgesamt drei Stück decken die Bereiche über und unter dem Laserscanner ab und verhindern Kollisionen mit Objekten, die nicht von dem Laserscanner erfasst werden können.

Funktionsprinzip

Der Ultraschallsensor besitzt einen Sender und einen Empfänger. Zur Weitenmessung sendet der Sensor 8 Ultraschallbursts aus und empfängt die reflektierten Signale wieder. Aus der Laufzeit lässt sich nun die Entfernung zum Objekt berechnen, dabei gilt: 10 $cm \cong 291\ \mu s$. Weil sich Schall schlecht bündeln lässt, hat der Ultraschallsensor einen Öffnungswinkel von ungefähr 45-55°. Dies hat den Vorteil, dass viele Objekte gut erkannt werden, für bestimmte Aufgaben ist dies jedoch von Nachteil.

Statemachine

Da der Ultraschallsensor bis zu 65 ms für eine Messung benötig und sich die drei verbauten Sensoren gegenseitig beeinflussen würden, muss eine besondere Methode gewählt werden, um die Sensoren auszulesen. Es wird eine „Statemachine“ bzw. „Endlicher Automat“ umgesetzt. Dazu ruft ein Interrupt, alle 60 ms eine Funktion auf, die den aktuellen Status des Automaten überprüft und gegebenenfalls in den nächsten Zustand übergeht.
Durch die Statemachine ist es möglich den Sensor „nicht-blockierend“ abzufragen, dies ist besonders wichtig da der ARM noch viele andere Aufgaben zu erfüllen hat.

4.2.2. Sharp - Entfernungsmesser

Der Sharp misst ebenfalls die Entfernung, arbeitet aber im Gegensatz zum Ultraschallsensor nicht mit Schall sondern Licht. Der Sensor ist für Entfernungen von 10 bis 80 cm ausgelegt und hat nach einem Meter Entfernung einen Spot von ca. 5 mm. Deswegen kann der Sensor sehr genau den Abstand zu einem Punkt messen und wird auf dem Roboter verwendet, um Abgründe zu erkennen und um parallel an einer Wand entlang zu fahren.

Funktionsprinzip

Genau wie der Ultraschallsensor besteht der Sharp Entfernungsmesser aus einem Sender und einem Empfänger. Als Sender dient hier eine IR-LED, die gepulstes Licht aussendet. Der ausgesendete Lichtstrahl wird am Objekt reflektiert und trifft auf ein PSD (**P**osition **S**ensetive **D**evice), der Empfänger setzt den Auftreffpunkt in eine analoge Spannung um. Die Spannung wird dann von dem ATtiny44 ausgelesen und über TWI an den ARM weitergegeben.

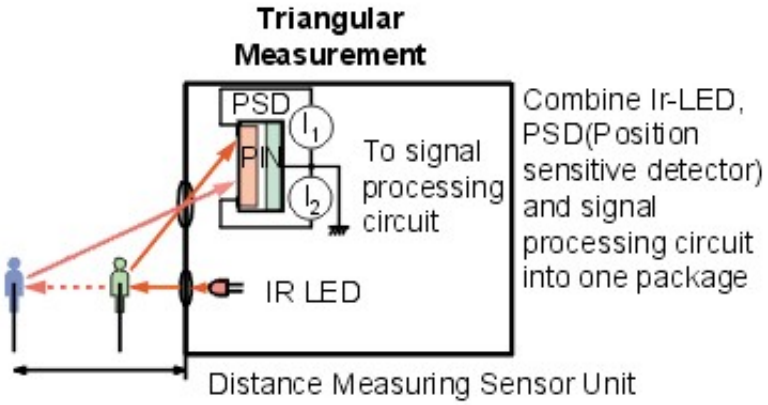


Abbildung 4.3.: Funktionsweise Sharp [8]

Umrechnung der Werte

Das Umrechnen der analogen Spannung in Entfernungen, gestaltet sich als schwierig, da der Sensor nicht linear arbeitet. Eine Funktion für die Kurve lautet:
$f(x) = \frac{A}{x-B}$ [20]. Die Kurve des Sensors sieht wie folgt aus:

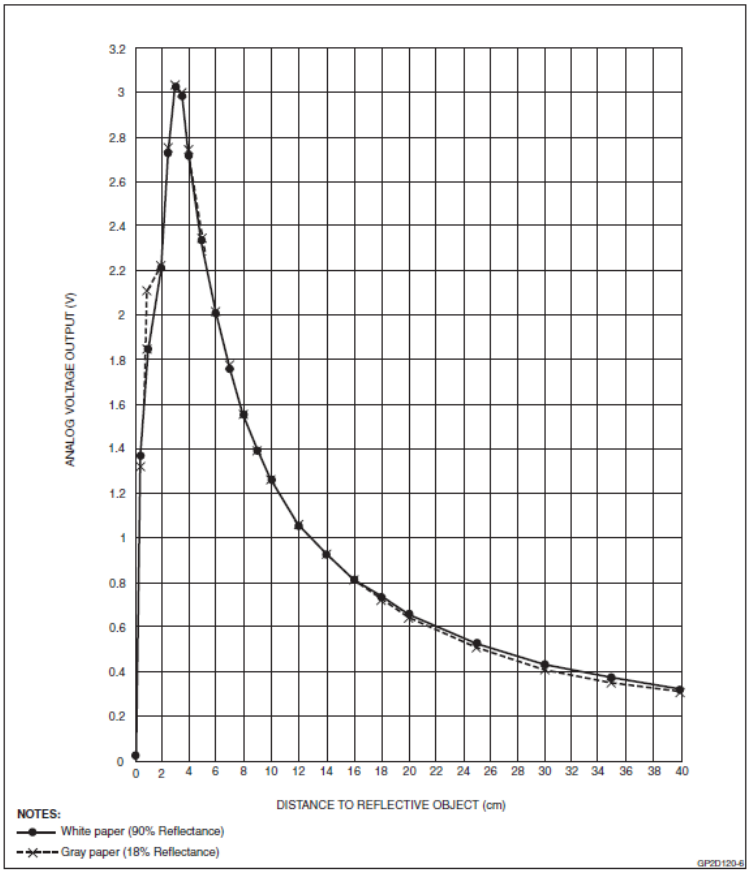


Abbildung 4.4.: Output des Sensors [9]

Hierbei steht $f(x)$ für die Entfernung, x ist der Sensorwert, A ist die Steigung der Kurve $\frac{A}{x}$ und B ist der Offset der Kurve. Zum Bestimmen der Parameter A und B werden bei zwei bekannten Entfernungen die analogen Spannungen mit Hilfe des ADC gemessen. Dann berechnen sich die Parameter wie folgt:

A	B
$\frac{(x'-x)*D'*D}{D-D'}$	$\frac{D'*x'-D*x}{D'-D}$

Tabelle 4.1.: Berechnen der Parameter[20]

4.2.3. GPS

GPS steht für **G**lobal **P**ositioning **S**ystem, dabei handelt es sich um ein System aus Satelliten und Empfängern. Es dient wie der Name schon sagt dem Ermitteln der aktuellen Position. Dies geschieht mit einer Genauigkeit von bis zu 1,5 m. Auf dem Roboter ist das LS20031 von der Firma Locosys verbaut worden. Es kann auf 66 Kanälen empfangen und wird über die serielle Schnittstelle an den ARM angeschlossen.

Funktionsprinzip

Das GPS-Modul empfängt die Position eines Satelliten und die Uhrzeit, zu der die Nachricht gesendet wird. Daraus lässt sich mit der Formel $s = c * \Delta t$ die Entfernung zum Satelliten berechnen. Die eigentliche Berechnung der Position erfolgt mit Hilfe von Trilateration, dazu werden drei Satelliten benötigt.

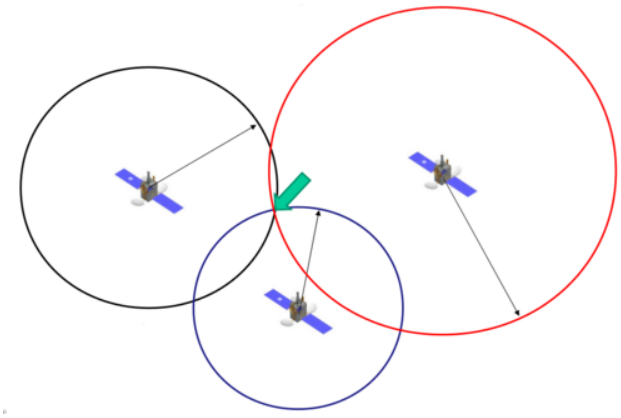


Abbildung 4.5.: GPS Funktionsprinzip [9]

Der Schnittpunkt der Kreise, mit den Entfernungen als Radien, entspricht der Position des GPS-Moduls. In der Praxis werden mehr als nur drei Satelliten benötigt, um Messfehler auszugleichen und die Höhe des GPS-Moduls zu bestimmen. Das hier vereinfacht dargestellte Verfahren funktioniert am besten, wenn die Satelliten möglichst weit von einander entfernt sind.

NMEA - Strings

Bei „NMEA 0183“ handelt es sich um den Standard, mit dem das GPS-Modul über die serielle Schnittstelle kommuniziert. Konkret wird hier der GPGGA-Datensatz ausgewertet. Er enthält die aktuelle Uhrzeit, die Position inklusive der Höhe und die Qualität der Messung. Es gibt noch weitere Datensätze, die zum Beispiel Auskunft über die aktuellen Satelliten geben, diese sind hier aber nicht von Interesse.

4.2.4. Kompass

Da die aktuelle Position nicht ausreicht, um die Richtung zu bestimmten, benötigt der Roboter einen Kompasssensor. Auf dem Roboter ist das Kompassmodul CMPS10 verbaut, es hat eine Genauigkeit von 0.5° und eine Auflösung von 0.1°. Damit lässt sich hinreichend genau die Richtung bestimmten, in die der Roboter fährt.

Funktionsprinzip

In dem Kompassmodul sind drei Magnetfeldsensoren verbaut, dabei handelt es sich um Hall-Sensoren die in Richtung der drei Koordinatenachsen angebracht sind. Diese Hall-Sensoren können das Erdmagnetfeld wahrnehmen und ein Mikroprozessor berechnet aus den gemessenen Werten die absolute Richtung. Gleichzeitig ist ein Beschleunigungssensor verbaut, mit Hilfe dessen das Kompassmodul bei einer Neigung von 60° immer noch eine Genauigkeit von ca. 1° hat.

4.2.5. Strom

Als weitere Kontrollmaßnahme wird der Motorstrom sowie der Strom, der aus dem Mainboard Akku verbraucht wird, gemessen. Dies geschieht einmal über den Motortreiber selbst, der eine analoge Spannung bereitstellt, die proportional zum aktuellen Strom wirkt beziehungsweise über einen Stromsensor, der dieselbe Funktionsweise hat. Die Spannung wird von den AVRs eingelesen und per I^2C an den ARM-Prozessor weitergegeben.

Funktionsprinzip

In dem Stromsensor beziehungsweise Motortreiber sitzt ein Hall-Sensor, dieser kann das Magnetfeld wahrnehmen, das um einen Leiter erzeugt wird. Je stärker der Strom, desto stärker ist das Magnetfeld und desto größer ist die Hall-Spannung. Die Hall-Spannung wird in dem Sensor verstärkt und steht dann an einem Pin zur Verfügung.

4.2.6. Encoder

Die Encoder sind enorm wichtig für den PID-Regler, denn die Encoder sitzen am Motor und messen die aktuelle Drehzahl. Das Abfragen der Encoder ist eine schwere Aufgabe, da diese 1856 Ticks (Flankenwechsel) pro Umdrehung geben.

Funktionsprinzip

Auf der Motorachse ist ein Magnet montiert, der an einem Hall-Sensor vorbei rotiert. Der Hall-Sensor erkennt den Wechsel von Nord- zum Südpol und gibt zwei Signale aus, die um 90° phasenverschoben sind. Auf dem Bild ist das Signal des Encoders aufgenommen:

Abbildung 4.6.: Encoder Oszillogramm [10]

Dioden - Gatter

Der Encoder liefert pro Kanal nur die Hälfte der Ticks, das heißt 928. Um die Anzahl zu verdoppeln, ist es nötig die beiden Kanäle miteinander zu verbinden, das geschieht mit Hilfe einer logischen Verknüpfung. Es ist ein Oder-Gatter gewählt worden, das mit Hilfe von Hochgeschwindigkeitsdioden diskret aufgebaut worden ist.

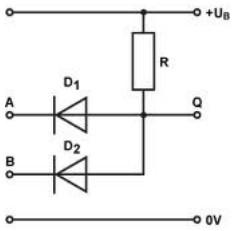


Abbildung 4.7.: Dioden Oder-Gatter [11]

Der Ausgang des Gatters geht an einen besonderen Pin am LPC1769. Der Timer0 beziehungsweise Timer1 kann so konfiguriert werden, dass das Zählregister bei einem Flankenwechsel an dem Pin hochgezählt wird. Dieses Register wird alle 10 ms ausgelesen: Je größer die Zahl im Register, desto höher ist die Drehzahl.

5. Programmierung

Die Programme für den ARM sind mit Hilfe der kostenlosen Software „Code Red“ entwickelt worden. Dieses Programm wird von NXP zur Verfügung gestellt und baut auf Eclipse auf. Es ist speziell für das Programmieren des LPCExpresso Kits ausgelegt. Darüber hinaus ist zu erwähnen, dass bei dem Kit ein JTAG Debugger dabei ist, mit dem sich die Programme komfortabel debuggen und flashen lassen. Das LPCXpresso Board ist für Entwickler gedacht und bietet eine hohe Flexibilität in seiner Anwendung, da sich der „Programmer“ von dem Rest abtrennen lässt. So kann man das Board auch in eigenen Applikationen verwenden.

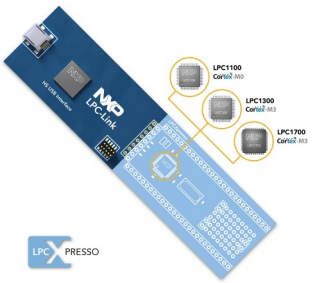


Abbildung 5.1.: LPCXpresso Development Board [12]

5.1. Kommunikation

Auf dem Roboter sind viele Sensoren und Prozessoren untergebracht, die miteinander kommunizieren müssen. Dafür kommen hauptsächlich drei verschiedene Schnittstellen zum Einsatz: Zum einen die Ethernet-Schnittstelle, der I^2C-Bus und natürlich die RS232 (Serielle Schnittstelle). Die Ethernet-Schnittstelle dient einzig der Kommunikation des ARMs mit dem Mainboard. Es wird eine TCP/IP-Verbindung aufgebaut. Dies macht das Abfragen der Sensordaten sehr komfortabel. Der I^2C-Bus wird nur benutzt, um eine Verbindung zwischen den Mikrocontrollern und verschiedenen Sensoren herzustellen. Die RS232 ist die wichtigste Debugging Schnittstelle und wird auch als solche genutzt, der ARM besitzt drei solcher Schnittstellen. Über eine dieser Schnittstellen ist das GPS-Modul angeschlossen.

5.1.1. TCP/IP - Mainboard

Die Anbindung des Ethernet wird in drei Dateien aufgeteilt, die nach einem stark vereinfachten OSI-Modell aufgebaut sind.

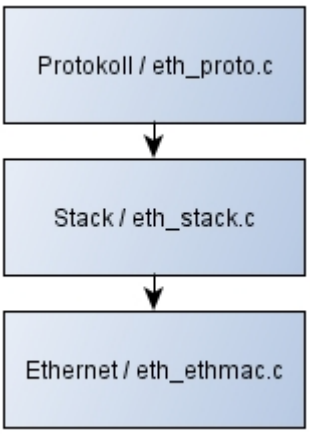


Abbildung 5.2.: Vereinfachtes Schichten Modell

Die Dateien haben den Prefix „eth“, und enthalten die Umsetzung der verschiedenen Schichten in C-Code. Die Datei „eth_stack.c“ wird zum Teil von Ulrich Radig [21] übernommen, jedoch an die Hardware von dem ARM angepasst. Die Datei „eth_ethmac.c“ wird aus einem LPCXpresso Beispiel übernommen, diese ist jedoch fehlerhaft gewesen und musste korrigiert werden.

Ethernet - Low Level

Der LPC1769 besitzt eine „Ethernet MAC“, das über ein RMII (**R**educed **M**edia **I**ndependant **I**nterface) mit dem PHY (physikalische Schnittstelle) kommuniziert. Der Ethernet Block ist entwickelt worden, um optimal mit der „DMA Engine“ (**D**irect **M**emory **A**ccess) zu arbeiten. Dadurch wird weniger Prozessor-Zeit in Anspruch genommen, da der Ethernet Treiber lediglich die Speicheradresse des zu senden Blocks der „DMA Engine“ übergeben muss. Die Speicheradresse und andere wichtige Informationen werden in einem so genannten „descriptor“ gespeichert, die wiederum in einem „Receive descriptor array“ bzw. „Transmit descriptor array“ abgelegt werden. Der Ethernet Block besitzt verschiedene Register, unter anderem das „ProduceIndex“ Register. In diesem Register wird die aktuelle Descriptornummer von der Treibersoftware gespeichert. Im Gegensatz dazu gibt es das „ConsumeIndex“-Register. Dieses Register beinhaltet die Descriptornummer, die als nächstes von der Hardware verarbeitet wird. Grundsätzlich kann entweder die Treibersoftware oder die Hardware einen Descriptor besitzen, dies muss bei der Programmierung berücksichtigt werden.

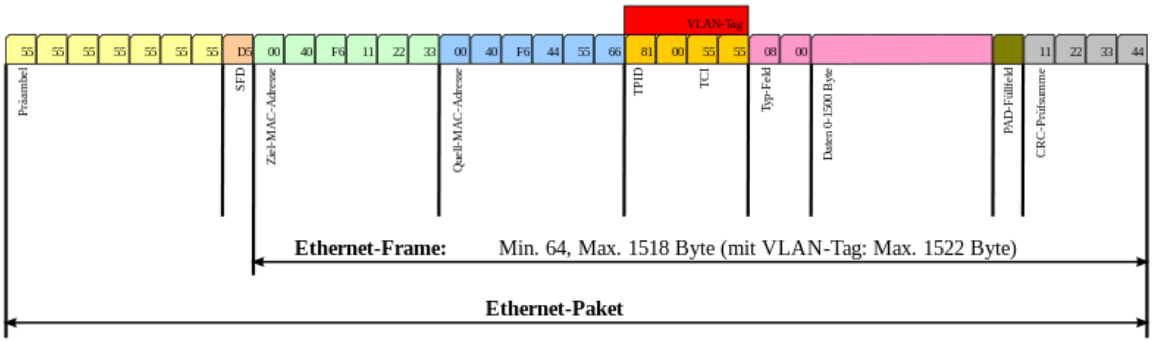


Abbildung 5.3.: Ethernet Frame [13]

Die oben angezeigte Grafik zeigt einen Ethernet Frame, wie er nach IEEE 802.3 üblich ist. Der Ethernet-Header wird zum Teil von der Hardware erstellt, die Präambel und die CRC-Checksumme wird automatisch generiert. Die Source- (grün) und Quell-MAC-Adresse (blau), sowie das Type-Field (rosa) müssen von der Software besetzt werden. Das VLAN-Tag spielt in dieser Applikation keine Rolle und wird weggelassen. Das Type-Field enthält die Information, ob es sich um ein Datenpaket, das heißt es folgt ein IP-Header, oder um einen ARP-Befehl (**A**dress **R**esolution **P**rotocol), handle. Bevor ein Paket versendet werden kann, muss die Software herausfinden, welche MAC-Adresse zu welcher IP-Adresse passt, diese Information wird danach in einem so genannten „ARP-Table“ gespeichert. Der Hardware muss die eigene MAC-Adresse mitgeteilt werden, denn diese besitzt einen Filter, der automatisch alle Pakete mit der eigenen MAC-Adresse an die Treibersoftware weiterleitet.

Stack - TCP/IP

Der TCP/IP-Stack (**T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol) übernimmt das Erstellen der TCP/IP-Header. Der Stack ist minimal gehalten worden, deswegen kann nicht auf alle eventuellen Fehler reagiert werden.
Zunächst wird eine IP-Header erstellt. Dabei müssen Source- und Destination-IP Adressen angegeben werden. Weiter beinhaltet der Header die komplette Länge des IP-Paketes und einen TTL-Zähler (**T**ime **t**o **L**ive). Dieser bekommt einen Startwert und wird von jedem Knoten (z.B. Router), den das Paket passiert, dekrementiert. So wird verhindert, dass IP-Pakete zu lange im Umlauf sind, denn das IP-Protokoll beinhaltet genau so wenig, wie das darunter liegende Ethernet-Protokoll, Informationen über den Status der Verbindung. Für Statusinformationen über die Verbindung ist TCP zuständig. Während das IP- und Ethernet-Protokoll sich nicht darum kümmern, ob ein Paket bei dem Ziel ankommt, sorgt TCP dafür, dass eine Verbindung auf- beziehungsweise abgebaut wird. Außerdem ermöglicht das TCP-Protokoll, durch das Unterscheiden von Ports, das Aufbauen von mehreren Verbindungen zum selben Endgerät.

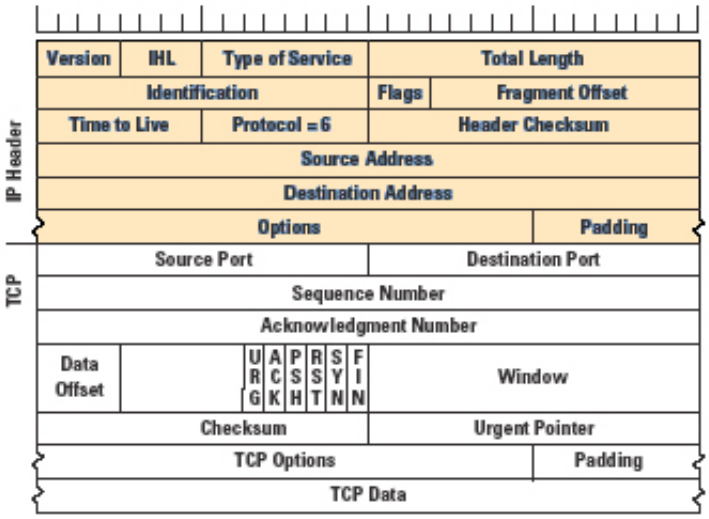


Abbildung 5.4.: TCP/IP Header [14]

In dem Bild sieht man, dass es sich bei dem TCP-Header um Daten des IP-Protokolls handelt. Bei den Daten des TCP-Protokolls (der letzte Punkt im Bild) handelt es sich um die eigentlichen Nutzdaten. Außerdem sind die TCP-Flags zu erkennen, mit dessen Hilfe das Auf- und Abbauen einer Verbindung realisiert wird. Zum Aufbauen einer Verbindung muss das SYN-Flag gesetzt werden, was mit einem SYN- und ACK-Flag bestätigt wird. Nachdem der Client dies mit einem weiteren ACK-Flag bestätigt, ist die Verbindung aufgebaut und es können Daten gesendet werden.

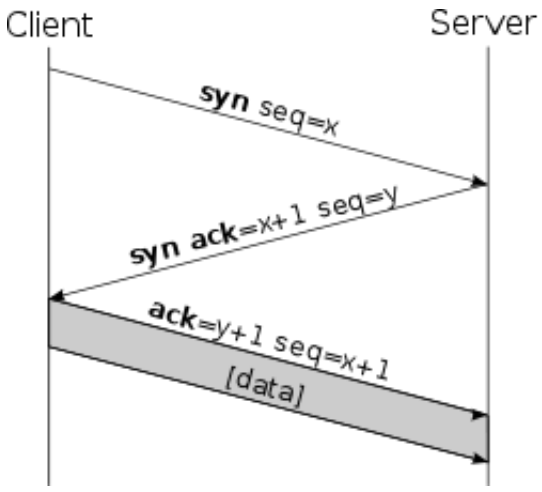


Abbildung 5.5.: TCP-Verbindungsaufbau [15]

Das Abbauen einer Verbindung funktioniert ähnlich, nur, dass diesmal das FIN-Flag gesetzt werden muss. Weiterhin muss jedes mal der Erhalt eines Paketes mit dem ACK-Flag bestätigt werden. Sollte dieses nicht gesetzt sein, wird der entsprechende

Partner versuchen das Paket erneut zu senden.
In dem Programm wird ein Buffer „eth_buffer“ mit der Größe der MTU (**M**aximum **T**ransmission **U**nit) angelegt, dieser Buffer muss nun mit den einzelnen Headern und den Nutzdaten gefüllt werden. DDafür wird für jedes Protokoll ein struct angelegt, das exakt der Größe der entsprechenden Header für die Protokolle entspricht.

```
1  struct TCP_Header {
2    unsigned short  TCP_SrcPort;  //der Quellport
3    unsigned short  TCP_DestPort; //der Zielport
4    unsigned int   TCP_Seqnum;    //numerierter Bytestrom
5    unsigned int   TCP_Acknum;    //numerierter Bytestrom (Gegenrichtung)
6    unsigned char   TCP_Hdrlen;   //4Bit Header
7    unsigned char    TCP_HdrFlags; //4Bit Header Laenge und Flags
8    unsigned short  TCP_Window;   //Fenstergroesse max. 64K Byte
9    unsigned short  TCP_Chksum;   //Pruefsumme
10   unsigned short  TCP_UrgentPtr;//16 Bit Urgent Pointer
11 }__attribute__((packed));
```

Listing 5.1: Struct für TCP-Protokoll

Indem man einen Pointer des entsprechenden Structs anlegt und diese über den „eth_buffer“ legt, kann man sehr komfortabel auf die einzelnen Felder der Header zugreifen:

```
1    struct Ethernet_Header *ethernet;     //Pointer auf Ethernet_Header
2    struct IP_Header       *ip;           //Pointer auf IP_Header
3    struct TCP_Header      *tcp;          //Pointer auf TCP_Header
4
5    ethernet = (struct Ethernet_Header *)&eth_buffer[ETHER_OFFSET];
6    ip       = (struct IP_Header       *)&eth_buffer[IP_OFFSET];
7    tcp      = (struct TCP_Header      *)&eth_buffer[TCP_OFFSET];
8
9    if(tcp->TCP_HdrFlags == SYN_FLAG)
10   {
11     [...]
12   }
```

Listing 5.2: Zugriff auf Structs

In Zeile 12-14 werden die Pointer angelegt und in den Zeilen 16-18 werden die Pointer mit den richtigen Adressen auf die entsprechende Stelle des Buffers gelegt. Der &-Operator, liefert die Speicheradresse einer Variablen. In diesem Fall liefert er die Speicheradresse, des Bytes an der Stelle, an der der Ethernet-, TP- beziehungsweise TCP-Header beginnt. Danach zeigen die Pointer jeweils auf den Anfang der jeweiligen Header. Wie man in der IF-Abfrage sieht, kann man mit Hilfe des Pfeil-Operators („->“) auf ein bestimmtes Element des Structs zugreifen. Speziell in diesem Fall wird

abgefragt, ob das SYN-Flag des TCP-Headers gesetzt ist, genauer gesagt, ob eine Verbindung aufgebaut werden soll.

Protokoll - Sensordaten

Die letzte Datei „eth_proto“ kümmert sich um die Implementierung eines eigenen Protokolls. Immer wenn ein Datenpaket auf Port 5555 empfangen wird, wird eine Funktion aufgerufen, die sich um die Bearbeitung der neuen Daten kümmert. Dabei gibt es hauptsächlich zwei wichtige Structs, mit denen Daten vom ARM beziehungsweise vom Mainboard gesendet werden:

	Data	Length
Header	PräambelA	8
	Sequenz-Nr	8
	Flags	8
	Befehls-Nr	16
	Befehls-Ack	8
Data	Befehls-Nr	16
	Sharp1	64
	Sharp2	64
	Sharp3	64
	Ultrasonic1	64
	Ultrasonic2	64
	Ultrasonic3	64
	ENC1	64
	ENC2	64
	Acceleromete	64
	Acceleromete	64
	Compass	64
	Currentdata	64
	Motorcurren	64
	Motorcurren	64
	GPSdata	136

	Data	Length
Header	PräambelA	8
	Sequenz-Nr	8
	Befehls-Nr	8
Data	MspeedA	64
	MspeedB	64

Abbildung 5.6.: Eigenes Protokoll

Abbildung 5.6 zeigt die zwei verschiedenen Structs. Das linke wird benutzt, damit der ARM mit dem Mainboard kommunizieren kann. Das heißt, dass alle Sensordaten enthalten sind, die aufgenommen worden sind. Mit dem rechten Struct kann das Mainboard dem ARM Befehle für die Motoren übergeben werden, nach einer Präambel werden zwei Geschwindigkeiten für die Motoren links und rechts übermittelt. Bei den Motordaten handelt es sich um Drehzahlen. Es sind Werte zwischen -350 bis 350 rpm (**R**ounds **p**er **M**inute) zulässig, wobei das Vorzeichen gleichzeitig die Richtung angibt. Diese Structs werden 15 mal pro Sekunde hin und her gesendet, was eine Menge Datenverkehr zur Folge hat. Trotz der vielen anderen Aufgaben, die der ARM erfüllen muss, darf er diese Verbindung auf keinen Fall vernachlässigen, deshalb werden die Ethernet-Pakete interrupt gesteuert, empfangen und verarbeitet.

5.1.2. I²C

Die Kommunikation über I²C wird ähnlich dem Ethernet in eine „Low-Level“ und eine „User-Application“ Datei aufgespalten. Die Kommunikation mit der Hardware wird in der „i2c.c“ Datei implementiert und die „User-Applicationen“ sind in den Dateien „i2c0_slaves.c“, „i2c1_slaves.c“ und „i2c2_slaves.c“ zu finden.

I²C - Low Level

Zunächst gibt es auch hier pro Schnittstelle jeweils einen Empfangs- und einen Sendebuffer. Außerdem gibt es jeweils eine Variable, die die Länge der zu lesenden beziehungsweise schreibenden Datensätze beinhaltet. Nachdem die Arrays beschrieben worden sind, kann eine Übertragung gestartet werden, indem ein bestimmtes Flag gesetzt wird:

```
1  void i2c0_start_nonblock(void)
2  {
3    /*--- Issue a start condition ---*/
4    I2C0CONSET = I2CONSET_STA;  /* Set Start flag */
5  }
```

Durch das Setzen dieses Flags wird eine Startcondition auf dem Bus ausgegeben. Jede weitere Kommunikation findet nun in der Interruptroutine für die jeweilige Schnittstelle statt. Besonders wichtig ist das „I2C0STAT-“ beziehungsweise das „I2C1STAT-“ oder das „I2C2STAT-Register“. Hier ist immer der aktuelle Status der Hardware verzeichnet. Im Datenblatt des LPC1769 steht genau beschrieben, was bei welchem Statuscode zu tun ist. Hier ist ein Auszug aus der über 100 Zeilen langen Interrupt Routine:

```
1  void I2C0_Handler(void){
2    unsigned char StatValue = I2C0STAT;
3    switch ( StatValue ){
4    //start condition has been transmitted
5    case 0x08:
6      i2c0_windex = 0;
7      I2C0DAT = i2c0_tbuffer[i2c0_windex++];
8      I2C0CONCLR = (I2CONCLR_SIC | I2CONCLR_STAC);
9      i2c0_state = I2CSTATE_PENDING;
10     break;
11   ...
12   //error
13   default:
14     I2C0CONCLR = I2CONCLR_SIC;
```

```
15          i2c0_error_handler(StatValue);
16        break;
17        }
18    return;
19  }
```

Wie man sieht, besteht die Routine eigentlich nur aus einer großen Switch-Anweisung. In Zeile 2 wird das Status-Register zwischengespeichert. Danach beginnt die Switch-Anweisung. Für den Fall, dass das Statusregister den hexadezimalen Wert „0x08“ hat, ist erfolgreich eine Startcondition gesendet worden. Danach wird der Index für das Schreiben resettet. In Zeile 7 wird das Datenregister mit dem Wert beschrieben, der als nächstes gesendet werden soll. In Zeile 8 wird dafür sorgt, dass keine weiteren Startconditionen mehr gesendet werden, indem es das „I2CONCLR_STAC“ Flag löscht. Zuletzt wird noch der „i2c0_state“ aktualisiert, der aber nur benötigt wird, wenn das Senden blockierend ausgeführt wird.

I²C - Applications

Damit die User Application funktionieren, muss in jeder der drei Dateien folgende Funktionen implementiert werden:

```
1  void i2c1_start_transaction(void){
2    ...
3  }
4
5  void i2c1_transaction_complete_handler (void){
6    ...
7  }
8
9  void i2c1_error_handler (unsigned char err_code){
10   ...
11 }
```

Die Funktionen sind natürlich entsprechend der Schnittstelle anders benannt. Die erste Funktion wird von einem Interrupt alle 60 ms aufgerufen und beginnt eine Transaktion. Mit anderen Worten initialisiert die Funktion den entsprechenden Sendebuffer und andere Variablen. Danach wird die Funktion „i2c1_start_nonblock“ aufgerufen, die wie oben beschrieben, das Senden startet. Die nächsten beiden Funktionen werden automatisch von der beschriebenen Interruptroutine aufgerufen. Die erste Funktion wird immer dann aufgerufen, wenn eine Übertragung erfolgreich gewesen ist. Dies bedeutet, dass alle Daten gesendet beziehungsweise empfangen worden sind. Die letzte Funktion wird immer dann aufgerufen, wenn es zu einem Fehler während der Übertragung kommt, gleichzeitig wird ein Statuscode übergeben, der dafür sorgt, dass die Funktion entsprechend reagiert.

Die Schnittstelle I2C0 ist für die Kommunikation mit den AVRs zuständig, während die zweite I2C1 Schnittstelle ausschließlich den Kompass ausliest. Die letzte Schnittstelle I2C2 hat mit dem Auslesen der drei Ultraschallsensoren am meisten zu tun.

5.1.3. RS-232

Wenn man von einer seriellen Schnittstelle spricht, hat man meistens die RS-232 im Sinn. „Serielle Schnittstelle“ ist jedoch ein Oberbegriff für Schnittstellen, deren Kommunikation nicht parallel, sondern seriell funktioniert, das heißt die Bits werden nacheinander über eine Leitung übertragen. Deswegen gelten USB, Firewire und Ethernet als serielle Schnittstelle, gemeint ist jedoch oft die 9polige RS-232, die über eine Transmit- und eine Receive-Leitung verfügt. Obwohl die RS-232 in Sachen Geschwindigkeit längst überholt ist, ist sie jedoch für debugging Zwecke unverzichtbar. Die Schnittstelle wird auf den Mikrocontrollern auch als UART (**U**niversal **A**synchronous **R**eceiver **T**ransmitter) bezeichnet und arbeitet hier mit 5 V bzw. 3,3 V.
Es werden zwei UARTs des LPC1769 verwendet. Eine dient dem Auslesen des GPS-Moduls, während die andere Informationen an einen PC weiter geben kann.

Baudrate

Wie der Name UART schon sagt, handelt es sich um eine asynchrone Übertragung. Das bedeutet, dass es keine Taktleitung wie zum Beispiel bei I^2C gibt, sondern lediglich eine Datenleitung. Damit die Kommunikation funktioniert, müssen sowohl Sender als auch Empfänger eine bestimmte Baudrate gesetzt haben. Bei der Baudrate handelt es sich um die Geschwindigkeit, mit der Zeichen übertragen werden. Aus der Baudrate lässt sich die Bitdauer ableiten. Übliche Baudraten liegen zwischen 9600bit/s und 115200bit/s.

Datenstrom

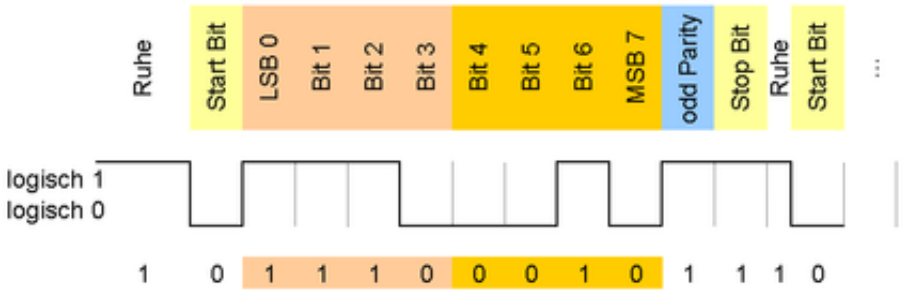


Abbildung 5.7.: UART Timing [16]

Auf dem Bild sieht man die Übertragung des Buchstaben „G“. Die Übertragung wird durch das Startbit eingeleitet. Nach dem Startbit folgen 8 Bit (1 Byte) Daten, zum Schluss wird ein Parity-Bit gesetzt, das oft auch weggelassen wird. Zum Schluss folgt das Stopp-Bit, welches dem Gegenüber sagt, dass die Übertragung beendet ist.

Programmierung

Im Gegensatz zum I^2C ist das Ansteuern des UARTs recht einfach. Einzig die Initialisierung mit der Einstellen der richtigen Baudrate und anderer Parameter ist nicht ganz einfach.

```
1  /* Function sends one Byte over USART0 */
2  void UART0_Transmit (unsigned char data){
3    // wait until receiver ready (Page 307)
4    while (!(U0LSR & 0x20));
5    U0THR = data; // put byte in transmit buffer (Page 301)
6  }
```

Die obige Funktion dient dem Senden eines Bytes über die Serielle Schnittstelle. Zunächst wird mit der While-Schleife gewartet bis kein Byte mehr in dem Transmit-Buffer ist. Wenn der Buffer frei ist, wird ein Flag gesetzt und es wird ein Datenbyte in den Buffer geschrieben. Die Hardware beginnt sofort automatisch damit, dass im Buffer liegende Byte über den UART zu senden. Empfangen wird hier ebenfalls in einer Interruptroutine, da das Programm nicht blockieren darf.

5.2. Hauptprogramm

In der Datei main.c ist das gesamte Hauptprogramm geschrieben. Da viele Programmteile in andere Dateien ausgelagert sind, um die Übersicht und Wiederbenutzbarkeit zu steigern, sieht die „Main Funktion“ wie folgt aus:

```
1  int main(void) {
2    init();
3
4    while(1) {
5      //nothing to do here, everything is done in interrupts
6    }
7
8    return 0;
9  }
```

Wie der Kommentar schon sagt, wird alles in Interruproutinen erledigt und die Hauptroutine ist völlig leer. Da alles in Interrupts erledigt wird, gibt es keinen genauen PAP (**P**rogramm **A**blauf **P**lan) sondern nur einen groben Ablauf:

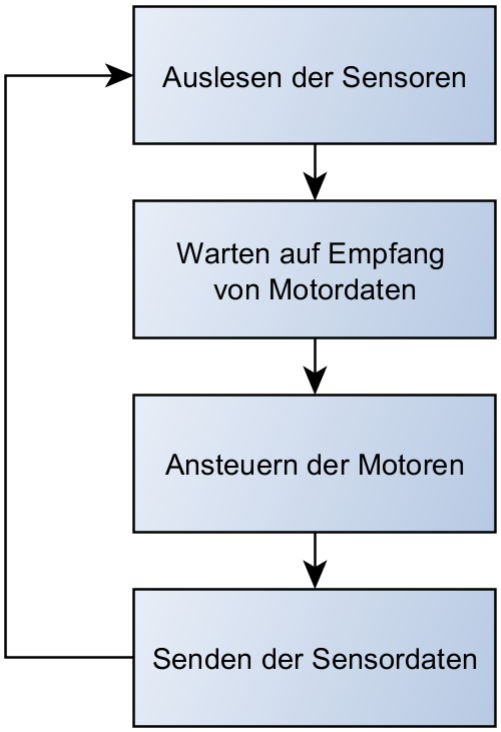


Abbildung 5.8.: Grober Programm Ablauf Plan

Wie man aus dem Diagramm entnehmen kann, werden zuerst alle Sensoren ausgelesen, danach wird auf den Empfang der Motordaten gewartet. Die Motordaten werden exakt 15 mal pro Sekunde gesendet. Nach Empfang werden die Motoren entsprechend angesteuert und die Sensordaten zurück geschickt. Tatsächlich erfolgen diese vier Schritte nahezu parallel, da alles interrupt gesteuert ist.

5.2.1. Interrupts

Interrupts unterbrechen das Ausführen des Hauptprogrammes, was meistens durch ein Hardware-Ereignis ausgelöst wird. Sie dienen zur Behandlung von zeitkritischen Aufgaben, wie zum Beispiel das sofortige Reagieren auf ein empfangenes Ethernet-Paket. Interrupts können den Eindruck von Multitasking beziehungsweise Parallelität erwecken, sind jedoch nicht mit dem Multitasking eines Betriebssystems zu vergleichen.
Der ARM-Prozessor besitzt einen NVIC (**N**ested **V**ectored **I**nterrupt **C**ontroller), dieser ist direkt in dem Prozessorkern eingebaut. Die enge Verbindung von CPU und Interruptcontroller gewährleistet eine kurze Latenz und effiziente Verarbeitung der ankommenden Interrupts.

Um einen Interrupt zu aktivieren, müssen entsprechende Falgs in bestimmten Registern gesetzt werden, außerdem muss die Interruptpriorität definiert werden. Standardmäßig werden alle Interrupts gleich behandelt, man kann jedoch Prioritäten festlegen, demnach wird der Interrupt mit der höchsten Priorität als Erstes bearbeitet.

```
1  void init_interrupts (void){
2
3    NVIC_ENABLE0 |= 0x20;                          //USART0
4    NVIC_PRI1 |= (((4 << 3) & 0xFF) << 8);         //Set Priority of UART0 to 4
5
6    NVIC_ENABLE0 |= (1<<3);                        //Timer2
7    NVIC_PRI0 |= (((1 << 3) & 0xFF) << 8);         //Set Priority of Timer2 to 1
8
9       ...
10
11   NVIC_ENABLE0 |= (1<<28);                       //Ethernet
12   NVIC_PRI7 |= (((0 << 3) & 0xFF) << 0);         //Priority of Ethernet to 0
13
14 }
```

Diese Funktion initialisiert alle Interrupts. Wie man sieht, wird immer erst der Interrupt aktiviert und danach die Priorität festgelegt. In das „Priority Register“ kann man Werte zwischen 0 und 32 schreiben, wobei 0 die höchste Priorität ist. Dabei hat der Ethernet-Interrupt die höchste Priorität, da er die Motordaten entgegennehmen muss, wobei es zu keiner Verzögerung kommen darf. Die eigentlichen Intrerruproutinen sind normale Funktionen, diese müssen jedoch einen besonderen Namen haben. Dieser lässt sich in der Datei „cr_startup_lpc176x.c“ erkennen. Die Funktionen sind mit dem „weak“ Attribut deklariert worden und werden deswegen bei erneuter Deklaration überschrieben. Werden sie nicht überschrieben, so wird ein „Default Handler“ ausgeführt, der das Programm in einer Endlosscheife enden lässt.

6. Probleme

6.1. PID - Regler

Das Einstellen des PID-Reglers hat sich als eine besondere Herausforderung herausgestellt, da ich recht unerfahren auf dem Gebiet der Regelungstechnik bin. Obwohl das Einstellen nach der Ziegler-Nichols-Methode eines der einfachsten Verfahren ist, um einen PID-Regler zu dimensionieren, ist es nicht gelungen vernünftige Werte zu finden:

1. Einstellung des Reglers als reinen P-Regler: $K_i = 0$ und $K_d = 0$
2. Die Reglerverstärkung K_p wird so lange vergrößert, bis sich der geschlossene Regelkreis an der Stabilitätsgrenze befindet und eine Dauerschwingung ausführt.
3. Der dabei eingestellte Wert K_p wird als $K_{p_{krit}}$ bezeichnet.
4. Die Periodendauer der eingestellten Dauerschwingung T_{krit} wird gemessen.

Anhand einer Tabelle [22], die hier nicht aufgeführt ist, lassen sich nun Werte für K_p, K_i und K_d berechnen.
Nachdem es nicht gelungen ist geeignete Werte für den Regler zu finden, wird das Verfahren zur Dimensionierung nach der Sprungantwort herangezogen.

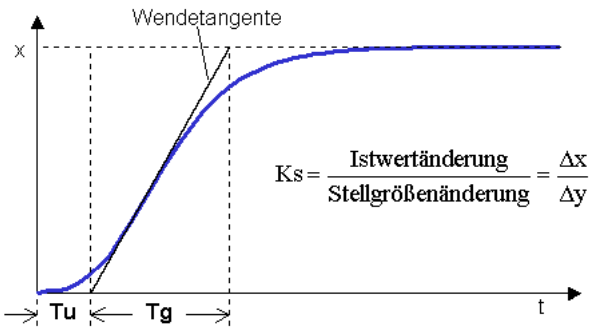


Abbildung 6.1.: Dimensionierung nach der Sprungantwort [17]

Mit Hilfe der herausgefundenen Parameter lassen sich die Faktoren K_p, K_i und K_d berechnen:

Regler	T_n	T_v	K_p	K_i	K_d
P			$\frac{0,3T_g}{K_s * T_u}$		
PI	$1,2 * T_g$		$\frac{0,35T_g}{K_s * T_u}$	$\frac{K_p}{T_n}$	
PID	T_g	$0,5 * T_u$	$\frac{0,6T_g}{K_s * T_u}$	$\frac{K_p}{T_n}$	$K_p * T_v$

Tabelle 6.1.: Dimensionierung nach der Sprungantwort

Das folgende Bild zeigt die aufgenommene Sprungantwort eines Motors, die Tangente und das Maximum sind bereits eingezeichnet:

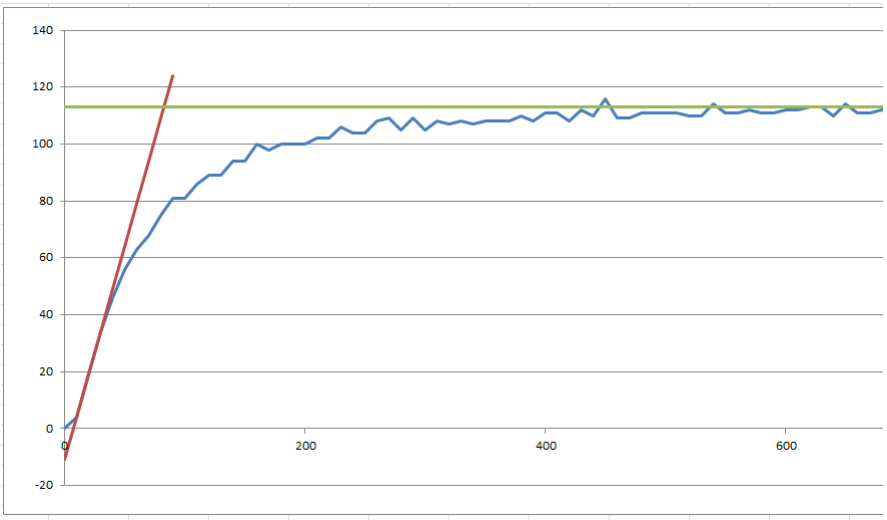


Abbildung 6.2.: Sprungantwort des Motors

Die Werte sind mit Hilfe der seriellen Schnittstelle aufgenommen und in Excel weiter verarbeitet worden. Das obige Bild ist ein Auszug aus der Arbeitsmappe. Für die so aufgenommene Sprungantwort ergeben sich folgende Werte: $K_p = 158$, $K_i = 9$, $K_d = 46$. Mit diesen Werten funktioniert der PID-Regler einwandfrei.

6.2. TCP/IP - Stack

Beim Portieren des TCP/IP-Stacks von Ulrich Radig auf den ARM, hat es einige Probleme gegeben, die gelöst werden müssen. Hauptsächlich müssen die Funktionen ersetzt werden, die dafür zuständig sind, mit der MAC-Schicht zu kommunizieren, wohingegen bei dem AVR nur eine Funktion von Nöten ist, werden hier mehrere gebraucht:

```
1    /* Ablauf fuer das senden eines Ethernet Frames */
2    RequestSend(ARP_REQUEST_LEN);
3    CopyToFrame_EthMAC(eth_buffer, ARP_REQUEST_LEN);
```

Zunächst muss das Senden vorbereitet werden. Danach werden die Daten in der Hardware übergeben. Der Empfang gestaltet sich komplizierter, da hier auf umständlicher Weise herausgefunden werden muss, wie lang das Paket ist:

```
1    /* Ablauf fuer das empfangen eines Ethernet Frames */
2    packet_length = StartReadingFrame();
3    CopyFromFrame_EthMAC(eth_buffer, packet_length);
4    StopReadingFrame();
```

Nachdem die Paketlänge herausgefunden worden ist, können die Daten kopiert werden. Zum Schluss muss der Hardware mitgeteilt werden, dass die Daten abgeholt worden sind und das Descriptor-Array wieder leer ist.
Ein weiteres Problem hat sich beim Setzen des MAC-Filters ergeben, da man der Hardware, wie oben beschrieben, mitteilen muss, welche MAC-Adresse benutzt wird. Die MAC-Adresse wird in bestimmte Register beschrieben. Der Einfachheit halber, wird die 6 Byte lange Adresse mit „defines“ gesetzt. Das Setzten der Adresse sieht wie folgt aus:

```
1  // NOTE - MAC address must be unique on the network!
2  SA0 = (MYMAC_1 << 8) | MYMAC_2; // Station address 0 Reg
3  SA1 = (MYMAC_3 << 8) | MYMAC_4; // Station address 1 Reg
4  SA2 = (MYMAC_5 << 8) | MYMAC_6; // Station address 2 Reg
```

Beim AVR sieht das Speichern der Adresse sehr ähnlich aus, mit dem Unterschied, dass die Bytes die entgegengesetzte Reihenfolge haben. Diesen Fehler zu finden hat einiges an Zeit und Mühe gekostet.

6.3. I²C - Bibliothek

Das Herstellen einer Verbindung zwischen ARM und AVR hat sich als besonders schwierig herausgestellt. Aus Zeitgründen sollte man auf eine fertige I²C-Bibliothek für den ARM zurückgreifen. Leider ist die Suche nach einer Bibliothek zeitaufwändiger, als das Programmieren selber, zum Schluss ist eine Bibliothek von dem Hersteller NXP benutzt worden. Die Bibliothek ist jedoch nicht sofort nutzbar, da sie auf die CMSIS-Bibliothek aufsetzt. Bei der CMSIS-Bibliothek handelt es sich um vorgefertigte Funktionen zur kompletten Hardware der LPC17xx Serie. Da diese Bibliothek nicht verwendet worden ist, muss einiges umgeschrieben werden.
Große Schwierigkeiten hat es auch bei dem Schreiben des Codes für die I²C-Slaves gegeben, auch hier sollte eine fertige Bibliothek verwendet werden, diese ist jedoch fehlerhaft. Obwohl eine Grundfunktionalität bestanden hat, ist die Bibliothek nicht mit einem „Repeated-Start“ zurecht gekommen. Nach einer aufwändigen Suche im Datenblatt des AVRs, hat man die Fehler jedoch beheben können.
Nach dem die Softwarefehler behoben worden sind, ist eine neue Problematik entstanden. Die Kommunikation setzt sporadisch aus.

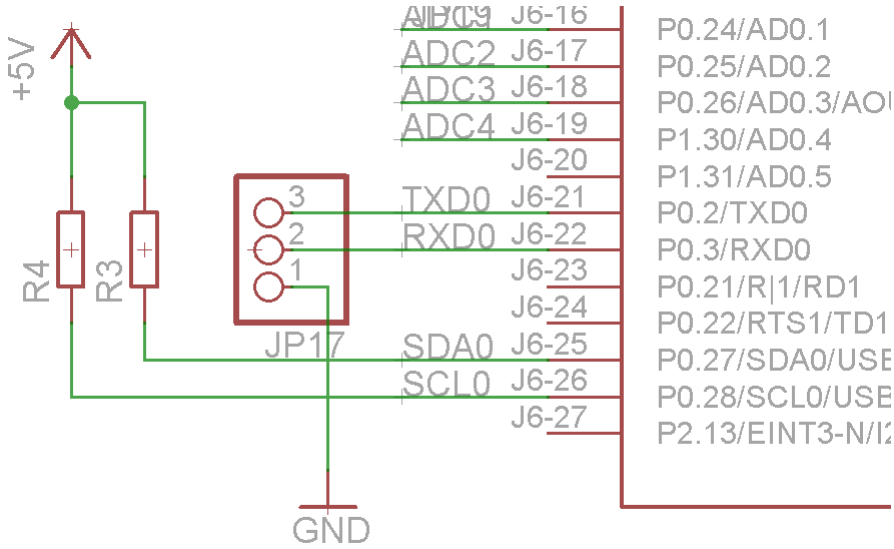


Abbildung 6.3.: Pull-Up-Widerstände

Wie sich herausgestellt hat, sind die Pull-Up-Widerstände zu groß gewählt worden, sie sind von 10 K auf 4,7 K verringert worden. Mit den veränderten Widerständen, läuft die Schaltung nun stabil.

7. Fazit

Am Ende dieses doch sehr außergewöhnlichen Projektes, möchte ich noch ein paar abschließende Worte schreiben. Obwohl dieses Projekt sehr viel Spaß gemacht hat, ist es oft schwierig gewesen gleichzeitig an dem Projekt zu arbeiten und zur Schule zu gehen. Dennoch hat die „Besondere Lernleistung“ mich in vielen Bereichen weiter gebracht, so habe ich Erfahrung gemacht, nach einem Zeitplan zu arbeiten und mit ARM-Prozessoren umzugehen.
Das Projekt begann für mich mit dem eigentlichen Bauen des Roboters, obwohl mir der mechanische Teil viel Spaß gemacht hat, kann ich eindeutig sagen, dass ich dabei auch viele Probleme hatte und diesen Problemen später nicht im Beruf begegnen möchte. Nach dem das Chassis geplant und gebaut war, musste ich mich darum kümmern die Elektronik zu entwickeln, dies bereitete mir sehr viel Freude. Da ich nach dem Abitur E-Technik studieren will, legte ich besonderes Augenmerk auf das Planen und Layouten von Platinen. Nach der Erstellung einer ersten Version, war ich jedoch nicht zufrieden und begann noch einmal. Die doppeltseitigen Platinen wurden von einem Anbieter geätzt und durchkontaktiert. Damit handelt es sich bei den Platinen um keine gewöhnlichen Hobbybastler-Platinen, sondern um recht professionelle Platinen. Im nächsten Schritt mussten die Mikrocontroller programmiert werden, dies gehört zu einer meiner großen Leidenschaften und war wie schon erwähnt eine große Neuerung, da ich mich bisher nur mit den 8 Bit AVRs beschäftigt habe. Zunächst begann ich damit einfache Aufgaben zu implementieren und steigerte mich dann immer weiter bis zu dem jetzigen Zustand. Der Roboter wurde sehr modular aufgebaut, so ist es möglich kleinere Module zu erstellen und diese auf Funktion zu testen. Dies erleichtert die Fehlersuche und gibt die Möglichkeit alles vor dem Zusammensetzen zu testen. Natürlich ist nach dem Zusammensetzen keine Funktion garantiert, es gibt jedoch die Möglichkeit Probleme und Fehler auszuschließen. Nach dem der Roboter fertig entwickelt war, begann ich mit dem Dokumentieren dieser Arbeit. Auch dies war Neuland für mich, da ich vorher noch nie mit LaTeX gearbeitet habe. LaTeX ist sehr vielseitig und ermöglicht das professionelle Erstellen von PDF-Dateien.
Zuletzt möchte ich noch einen kleinen Ausblick geben, denn das Projekt ist noch nicht abgeschlossen. Mit der „Besonderen Lernleistung“, haben wir gerade erst eine Basis geschaffen. Auf dieser Basis können wir nun weiterarbeiten und weitere

Funktionen implementieren. Es handelt sich bei dem Geschaffenen um eine nahezu universell einsetzbare „Roboterplattform“. Als nächstes planen wir eine „Kinect“ an das Mainboard anzuschließen. Die Erweiterung für die Spielkonsole „Xbox“ besitzt eine Kamera und erstellt gleichzeitig zu dem normalen RGB-Bild ebenfalls ein Tiefenbild. Dadurch sind Aufgaben wie zum Beispiel eine Personenverfolgung oder eine Gestensteuerung des Roboters denkbar.

Literaturverzeichnis

[1] *PWM-Signal.* http://de.wikipedia.org/wiki/Pulsweitenmodulation, 2013. – [Online; accessed 25-Jan-2013]

[2] *P-Regler Sprungantwort.* http://de.wikipedia.org/wiki/Regler, 2013. – [Online; accessed 25-Jan-2013]

[3] *I-Regler Sprungantwort.* http://de.wikipedia.org/wiki/Regler, 2013. – [Online; accessed 25-Jan-2013]

[4] *D-Regler Sprungantwort.* http://de.wikipedia.org/wiki/Regler, 2013. – [Online; accessed 25-Jan-2013]

[5] *PID-Regler Sprungantwort.* http://de.wikipedia.org/wiki/Regler, 2013. – [Online; accessed 25-Jan-2013]

[6] *SAR ADC.* http://en.wikipedia.org/wiki/Successive_approximation_ADC, 2013. – [Online; accessed 25-Jan-2013]

[7] *TWI - I2C Datentransfer.* http://www.rn-wissen.de/index.php/Bild:I2C_Datenuebertragung.png, 2013. – [Online; accessed 25-Jan-2013]

[8] *Funktionsweise Sharp.* http://www.rn-wissen.de/index.php/Sensorarten, 2013. – [Online; accessed 25-Jan-2013]

[9] *GPS Funktionsprinzip.* http://google.com, 2013. – [Online; accessed 25-Jan-2013]

[10] *Encoder Oszillogramm.* http://nodna.de/Getriebemotor-12VDC-mit-64-CPR-Encoder-350rpm-6mm-Achse, 2013. – [Online; accessed 25-Jan-2013]

[11] *Dioden Oder Gatter.* http://www.elektronik-kompendium.de/sites/dig/0710091.htm, 2013. – [Online; accessed 25-Jan-2013]

[12] *LPCXpresso Board.* http://ics.nxp.com/lpcxpresso/, 2013. – [Online; accessed 25-Jan-2013]

[13] *Ethernetframe.* http://de.wikipedia.org/wiki/Ethernet, 2013. – [Online; accessed 25-Jan-2013]

[14] *TCPIP - Bild.* http://www.bamed.org/2012/11/10/anatomy-of-a-packet/,

2012. – [Online; accessed 28-Dec-2012]

[15] *TCP-Verbindungsaufbau*. http://de.wikipedia.org/wiki/Transmission_Control_Protocol, 2013. – [Online; accessed 25-Jan-2013]

[16] *UART Timing*. http://de.wikipedia.org/wiki/Universal_Asynchronous_Receiver_Transmitter, 2013. – [Online; accessed 25-Jan-2013]

[17] *Dimensionierung nach der Sprungantwort*. http://www.rn-wissen.de/index.php/Bild:Tu_Tg.gif, 2012. – [Online; accessed 28-Dec-2012]

[18] *Reglungstechnik*. http://www.rn-wissen.de/index.php/Regelungstechnik, 2013. – [Online; accessed 5-Jan-2013]

[19] *Phase-Locked Loop*. http://de.wikipedia.org/wiki/Phasenregelschleife, 2013. – [Online; accessed 28-Dec-2012]

[20] *Berechnung der Parameter*. http://www.rn-wissen.de/index.php/Sensorarten#Formel_zur_Entfernungsberechnung, 2013. – [Online; accessed 28-Dec-2012]

[21] Radig, Ulrich: *Ulrich Radig*. http://www.ulrichradig.de, 2013. – [Online; accessed 28-Dec-2012]

[22] *Tabelle Ziegler-Nichols*. http://www.rn-wissen.de/index.php/Regelungstechnik#Dimensionierung_nach_Einstellregeln, 2013. – [Online; accessed 28-Jan-2013]

[23] *Regelungstechnik*. http://www.rn-wissen.de/index.php/Regelungstechnik, 2013. – [Online; accessed 28-Jan-2013]

[24] *Regelungstechnik*. http://de.wikipedia.org/wiki/Regelungstechnik, 2013. – [Online; accessed 28-Jan-2013]

[25] *Pulsweitenmodulation*. http://de.wikipedia.org/wiki/Pulsweitenmodulation, 2013. – [Online; accessed 28-Jan-2013]

[26] *Pulsweitenmodulation*. http://www.mikrocontroller.net/articles/Pulsweitenmodulation, 2013. – [Online; accessed 28-Jan-2013]

[27] *Atmel AVR*. http://de.wikipedia.org/wiki/Atmel_AVR, 2013. – [Online; accessed 28-Jan-2013]

[28] *AVR*. http://www.mikrocontroller.net/articles/AVR, 2013. – [Online; accessed 28-Jan-2013]

[29] *ARM*. http://de.wikipedia.org/wiki/ARM-Architektur, 2013. – [Online; accessed 28-Jan-2013]

[30] *ARM*. http://www.mikrocontroller.net/articles/ARM, 2013. – [Online; accessed 28-Jan-2013]

[31] *Sensoren*. http://www.rn-wissen.de/index.php/Sensorarten, 2013. – [Online; accessed 28-Jan-2013]

[32] Plate, Prof. J.: *Grundlagen Computernetze*. http://www.netzmafia.de/skripten/netze/, 2013. – [Online; accessed 28-Jan-2013]

[33] *I2C*. http://de.wikipedia.org/wiki/I%C2%B2C, 2013. – [Online; accessed 28-Jan-2013]

[34] *I2C*. http://www.rn-wissen.de/index.php/I2C, 2013. – [Online; accessed 28-Jan-2013]

[35] *CMPS10 - Tilt Compensated Compass Module*. http://www.robot-electronics.co.uk/htm/cmps10doc.htm, 2013. – [Online; accessed 28-Jan-2013]

[36] *Universal Asynchronous Receiver Transmitter*. http://de.wikipedia.org/wiki/Universal_Asynchronous_Receiver_Transmitter, 2013. – [Online; accessed 28-Jan-2013]

[37] *Interrupt*. http://de.wikipedia.org/wiki/Interrupt, 2013. – [Online; accessed 28-Jan-2013]

[38] *Output des Sensors*. http://www.sharpsma.com/webfm_send/1203, 2013. – [Online; accessed 25-Jan-2013]

A. Anhang

A.1. Inhaltsverzeichnis der CD

- Sourcen - In diesem Ordner befinden sich die Sourcen für den ARM.
- Layouts - In diesem Ordner befinden sich die Layouts für die Platinen
- Excel - In diesem Ordner befinden sich Excel-Dokumente.
- Videos - In diesem Ordner befinden sich Videos von dem Roboter.
- Bilder - In diesem Ordner befinden sich Bilder von dem Roboter.
- Webseiten - Hier sind alle benutzten Webseiten gespeichert.

A.2. Benutzte Software

- Code Red - http://lpcxpresso.code-red-tech.com/LPCXpresso/
- AVR Studio 4 - http://www.atmel.com/tools/AVRSTUDIO4.aspx
- PonyProg2000 - http://www.lancos.com/
- HTerm - http://www.der-hammer.info/terminal/
- Dec-C++ - http://www.bloodshed.net/devcpp.html
- TeXnicCenter - http://www.texniccenter.org/
- MiKTeX - http://miktex.org/

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Ort, Datum	Jannik Springer