Entwicklung der Steuerelektronik eines Mixed-Reality Roboters unter Beachtung aktueller Fertigungstechniken industrieller Kleinserien

Bachelor-Thesis an der Fachhochschule Kiel
Fachbereich Informatik und Elektrotechnik

Autor

Hannes Eilers

Studiengang

Bachelor of Engineering in Elektrotechnik

Fachrichtung Informationstechnologie

Datum

31. August 2013

Erstprüfer

Prof. Dr. Jens Lüssem

Zweitprüfer

Prof. Dr. Christoph Weber

Kurzfassung (Abstract)

Die Mixed-Reality ist eine Roboterfußball-Liga und verwendet ein halb-virtuelles Spielsystem zur Entwicklung von künstlichen Fußballintelligenzen. Die im System verwendeten Roboter zur Simulation von Fußballspielern sind derweil nicht mehr verfügbar und kompliziert in Aufbau und Betrieb, sodass eine Neuentwicklung eines kostengünstigen Roboters notwendig wurde.

Gegenstand dieser Arbeit ist die Entwicklung der Steuerelektronik eines neuen Mixed-Reality-Roboters unter Betrachten der Produktion in einer industriell gefertigten Kleinserie.

• Inhaltsverzeichnis

Kurzfassung (Abstract)....1
I.Einleitung....4
II.Grundlagen....5
II.1.Künstliche Intelligenz....5
II.2.Roboterfußball....5
II.3.RoboCup....5
II.4.Mixed-Reality-System....6
II.5.Industrielle Leiterplattenfertigung/Leiterplattenbestückung....6
III.Stand der Technik....9
IV.Mixed-Reality Roboter mrShark....10
IV.1.Ziele....10
IV.2.Anforderungen....10
IV.3.Vorgaben....11
IV.4.Namensgebung....11
IV.5.Konzept....12
IV.6.Steuerungsplatine....14
IV.7.Stromversorgung....32
IV.8.Energiebilanz....36
IV.9.Systemschnittstelle....38
IV.10.Bauteilparameter....39
IV.11.Firmware....47
V.Funktionstest....67
VI.Evaluierung....69
VI.1.Zielsetzung und Bewertungsverfahren....69
VI.2.Bewertung Bauteilverfügbarkeit....72
VI.3.Bewertung Bauteilvielfalt....73
VI.4.Bewertung Layout....74
VI.5.Gesamtbewertung mrShark....74
VI.6.Evaluierung der Zielvorgaben....75
VII.Zusammenfassung....77
VIII.Literaturverzeichnis....79

IX.Abbildungsverzeichnis.....82
X.Tabellenverzeichnis.....83
XI.Anhang.....84
Anhang A: Berechnungen.....84
Anhang B: Bauteilbewertungen.....92
Anhang C: Steuerbefehle.....96
Anhang D: Signalverläufe.....99
Anhang E: Schaltplan mrShark.....104
Anhang F: Platinenlayout mrShark.....114
Anhang G: Vorgaben Gehäuse mrShark.....117

I. Einleitung

Diese Arbeit beschreibt die Entwicklung einer Steuerelektronik für einen neuen Roboter für das z.B. im RoboCup eingesetzte Mixed-Reality Roboterfußball-System. Dieses findet insbesondere in der Entwicklung von künstlichen Intelligenzen im Rahmen von Lehr- und Forschungsprojekten Anwendung und stellt einen einfachen Einstieg in die Roboterentwicklung und -programmierung dar.

Nachteil des Systems ist die zur Zeit geringe Verbreitung und der prototypische Aufbau des Systems. Weiterhin sind einzelne Teile, insbesondere der Roboterhardware, schwer bis gar nicht verfügbar, sodass ein Einstieg in das System ohne eigene Entwicklungen nahezu unmöglich ist.

Als Vorteil des Systems ist der besonders einfache programmiertechnische Einstieg auf Basis der weit verbreiteten Programmiersprache Java, sowie die Reduktion von in der Realität typischerweise vorkommenden Roboterproblemen wie Ungenauigkeiten und unvollständigem Umweltwissen, anzusehen. Dies ermöglicht einen einfachen Übergang von reinen Simulationssystem zu hardwarenahen Ansteuerungen von Robotern.

Um den Einstieg in das System zu erleichtern, ist es das Ziel mit dieser Entwicklung ein einfaches, schnell und kostengünstig zu fertigendes Steuersystem für einen Mixed-Reality Roboter zu schaffen, welches die langfristige Verwendung der Roboterhardware ermöglicht. Insbesondere die Beschaffung und Fertigung einzelner Teile der Hardware sollten in Hinsicht auf Kosten und Umsetzungsmöglichkeiten optimiert werden.

Konkret sollte das System durch Hochschulen und Forschungseinrichtungen kostengünstig und unkompliziert zu beschaffen und mit Hilfe üblicher industrieller Standards zu fertigen sein. Dabei musste insbesondere auf die langfristige Verfügbarkeit der verwendeten Komponenten geachtet werden. Als Grundlage wurde das bestehende System verwendet und um neue Fähigkeiten sowohl in der Software als auch in der Hardware erweitert.

II. Grundlagen

Dieses Kapitel umfasst die Grundlagen zu den im Rahmen dieser Arbeit verwendeten Verfahren und Begriffen.

II.1. Künstliche Intelligenz

Eine künstliche Intelligenz (kurz KI) beschreibt in dem Kontext dieser Arbeit eine Ansammlung von Programmen und Algorithmen, welche durch ihre Ausführung das Vorhandensein einer Intelligenz simulieren. Dies besteht hauptsächlich aus Handlungen, welche mit Hilfe der o.g. Programme und Algorithmen errechnet werden und für die Beurteilung als Intelligenz von einem Menschen als rational und nachvollziehbar empfunden werden müssen.

II.2. Roboterfußball

Das Spiel Fußball bietet viele unterschiedliche Aspekte den menschlichen Alltags, wie strategisches Denken, situationsabhängige Handlungen, lernen aus bereits erlebten oder allgemein bekannten Situationen u.v.m.. Dadurch bietet es sich insbesondere für die Entwicklung von verschiedensten Verfahren für künstlicher Intelligenzen an, mit denen das menschliche Verhalten abgebildet werden soll.

Die Interaktion dieser KI mit dem Menschen, z.B. über einen Roboter, lässt sich im Kontext des Fußball besonders leicht umsetzen, da das Spielprinzip den meisten Menschen hinlänglich bekannt ist und in diesem Kontext ein großer Konsens über als rational geltende Handlungen vorliegt. Mit Roboterfußball ist also typischerweise das Spielen des Spiels Fußball durch Roboter, gesteuert durch eine KI, gemeint.

II.3. RoboCup

Der RoboCup ist ein internationaler Wettbewerb zwischen Wissenschaftlern und Studenten im Bereich der Robotik (The RoboCup Federation, 2013). In verschiedenen Wettkämpfen und Disziplinen (Ligen) treten jährlich einzelne Mannschaften gegeneinander an, um typische Aufgaben aus der Robotik, wie z.B. Fußballspiele (siehe 5), Suchaufgaben oder die Interaktion mit Menschen, zu bewältigen.

Die Veranstaltung ist auch als Plattform zum Wissenstransfer gedacht (The RoboCup Federation, 2013) und wird typischerweise von Diskussionsforen und Vorträgen zu Themen aus der Robotik begleitet. In verschiedenen Ländern existieren zudem nationale Vorentscheide. In Deutschland finden hierzu jährlich die RoboCup German Open statt (RoboCup German Open, 2013). Langfristiges Ziel des RoboCup ist es in der Mitte des 21. Jahrhunderts mit einer Mannschaft aus Robotern gegen den amtierenden Weltmeister im Roboterfußball anzutreten (The RoboCup Federation, 2013).

II.4. Mixed-Reality-System

Das Mixed-Reality-System ist ein technisches System zum Austragen von Roboterfußball-Wettbewerben (siehe Roboterfußball). Das System wurde auch schon im Rahmen des RoboCup (siehe RoboCup) eingesetzt.

Das System basiert auf einem teilweise simulierten System, in welchem das Fußballfeld und der Spielball simuliert und auf einem Bildschirm angezeigt werden. Auf diesem Bildschirm werden reale Roboter als Hardwarekomponente eingesetzt. Die Position der Roboter wird durch einen Spielserver über eine optische Bilderkennung bestimmt und über eine Infrarot-Schnittstelle gesteuert.

Die KI interagiert mit diesem Spielserver, welcher auch ein verlässliches Umweltsystem zur Verfügung stellt und die konkrete Ansteuerung der Roboter übernimmt. Die Implementierung der Interaktion der KI mit dem Spielserver basiert auf einem XML-formatierten Protokoll und einer Netzwerkverbindung um die KI auf externe Systeme auslagern zu können.

II.5. Industrielle Leiterplattenfertigung/Leiterplattenbestückung

Die industrielle Fertigung von Leiterplatten beschreibt das Verfahren zur Herstellung von elektronischen Baugruppenträgern durch ganz oder teilweise automatisierte industrielle Fertigungsverfahren. Diese können vereinfacht auch in Hochschulen eingesetzt werden. Zumeist bieten aber externe Unternehmen die Produktion, insbesondere von Kleinserien, gegenüber Hochschulen und anderen Unternehmen und Einrichtungen erheblich kosten- und zeitgünstiger an.

II.5.1. Leiterplattenfertigung

Die Fertigung von Leiterplatten erfolgt in der Industrie typischerweise, über das Fräsen, Ätzen oder Lasern von Leiterbahnen oder dem direkten Aufbringen von Leiterbahnen auf einer Trägerplatte, Basismaterial genannt (Sautter, 1988).

Bei diesen Verfahren werden mit Hilfe von Werkzeugen oder chemischen Prozessen elektrische Leiterbahnen aus einem Basismaterial ausgetragen. Das Basismaterial besteht aus einem nicht-leitenden Trägermaterial, meist aus „Hartpapier, Glasgewebe oder Baumwollpapier“ (DATACOM, 2013, Leiterplatte) und einer aufgetragenen (leitenden) Kupferschicht.

Alle genannten Verfahren haben, je nach konkreter maschineller Umsetzung, unterschiedliche Materialspezifikationen und Anforderungen an die Anordnung und das Layout einzelner Leiterbahnen. Diese sind insbesondere in der Realisierung eines Schaltplanes auf einer Leiterplatte zu beachten.

II.5.2. Leiterplattenbestückung

Unter der Bestückung von Leiterplatten versteht man das Auflöten einzelner elektronischer Baugruppen auf eine Leiterplatte. Im Gegensatz zur Leiterplattenfertigung existieren hier teilweise sehr große Unterschiede zwischen den einzelnen Verfahren in Bezug auf die mögliche Bauteilauswahl und Positionierung.

Mit der Through hole technology (kurz THT) wird die Durchsteckmontage beschrieben, bei der Baugruppen mit Bauteilbeinchen durch die Leiterplatte gesteckt werden um sie mit einer Leiterbahn auf der Leiterplatte zu verbinden (Scheel, 1997).

Hierbei werden die Bauteile zumeist manuell von Hand bestückt. Bei einer automatischen Bestückung müssen die Bauteile zumeist erst sortiert und teilweise die Bauteilbeinchen in die korrekte Form gebogen werden. Jedes Bauteil muss für die automatische Bestückung einer Bestückungsmaschine über einen Bauteilschacht, auch Feeder genannt, zugeführt werden und wird dann mit Hilfe einer Vakuumpipette durch die Maschine automatisch auf der Leiterplatte positioniert. Die bestückte Leiterplatte wird anschließend von Hand gelötet oder mit Hilfe einer Lötwelle gelötet.

Bei dem Einsatz einer Lötwelle fährt die bestückte Leiterplatte über einen aufwallenden flüssigen Berg aus Lot, welcher eine elektrische Verbindung zwischen den Bauteilbeinchen und der Leiterplatte herstellt. Hierbei ist insbesondere die Ausrichtung der Platine beim Fahren über die Lötwelle sowie der Abstand der einzelnen Bauteilbeinchen entscheidend für das Ergebnis des Lötvorgangs. Die nachfolgend genauer erklärten SMT Bauteile auf der Lötseite der Leiterplatte müssen für das Löten mit einer Lötwelle speziell fixiert werden, damit sie nicht abfallen. Daher ist dieses Verfahren nur bedingt bei gemischt bestückten Leiterplatten geeignet.

SMT bezeichnet die Surface mounted technology. In diesem Verfahren werden Surface mounded devices (kurz SMD), also oberflächenmontierte Bauteile, direkt auf einer Seite der Leitplatte befestigt. Dazu wird auf den Flächen, auf denen die Bauteile mit ihren Bauteilbeinchen auf der Leiterplatte angelötet werden, sogenannten Pads, zuerst mit Hilfe einer Schablone eine Lötpaste aufgetragen und die SMD-Bauteile anschließend auf dieser Lötpaste platziert. Durch Erhitzen der Lötpaste wird diese flüssig und stellt nach dem Abkühlen eine dauerhafte leitende Verbindung zwischen der Leitplatte und den SMD-Bauteilen her.

Zum Erhitzen der Leiterplatte wird in der Regel ein Reflow-Lötverfahren angewendet. Bei einem Reflow-Ofen z.B. durchläuft die Leiterplatte verschiedene Temperaturzonen, welche die Leiterplatte langsam erhitzen und nach dem Flüssigwerden der Lötpaste wieder langsam abkühlen (Scheel, 1997). Hierbei ist besonders darauf zu achten, dass beim Durchlaufen der verschiedenen Temperaturzonen Oberflächenspannungen in der Lötpaste auftreten können, was dazu führen kann, dass sich Bauteile während des Prozesses aufrichten (SIGEM Elektronik, 2013, SMD-Design), (Scheel, 1997).

Ein weiteres Reflow-Lötverfahren ist das Dampfphasenlöten, welches ein gasförmiges Trägermaterial in der Luft benutzt um die Leiterplatte zu erhitzen (Scheel, 1997). Dieses Verfahren schließt Probleme durch ungleichmäßige Erhitzung der Leiterplatte aus.

III. Stand der Technik

Das Mixed-Reality-System basiert zur Zeit auf der zweiten Version der ursprünglich vom Uhrenhersteller CITIZEN (RT Lions, 2013) hergestellten Roboter. Diese Roboter besitzen einen komplizierten, an der Gehäuseaußenseite montierten Motor und ein aufwändiges Aluminiumgehäuse. Für die Steuerung existieren verschiedene Platinenversionen, sowohl von CITIZEN, als auch Eigenentwicklungen einzelner Mixed-Reality-Mannschaften.

Die Roboter namens EcoBee werden über Infrarotsignale vom Spielserver gesteuert. Die Roboter besitzen in der Regel einen In-System-Programmer Anschluss (kurz ISP Anschluss), welcher das Aufspielen einer neuen Firmware mittels geeigneter Programmierhardware ermöglicht. Über ein bis zwei Lithium-Polymer (Kurz LiPo) Zellen werden die Roboter mit Strom versorgt, welche über eine einfache Buchsenleiste am Roboter wieder aufgeladen werden können.

Ein besonders großer Nachteil der verwendeten Roboter ist die mangelnde Verfügbarkeit, da CITIZEN die Produktion der Roboter auf Grund mangelnder Nachfrage eingestellt hat. Zudem ist der Preis von ca. 300 EUR pro Roboter nicht attraktiv genug für eine Neuanschaffung. Der komplizierte Aufbau des Gehäuses, des Motors sowie eine Programmierweise, welche Kenntnisse im Bereich Mikrocontrollertechnik voraussetzt, sind weitere Hindernisse im Betrieb des Systems.

IV. Mixed-Reality Roboter mrShark

IV.1. Ziele

Ziel dieser Arbeit ist die Entwicklung eines neuen Mixed-Reality-Roboters, welcher sich leicht in das bestehende System integrieren lässt, jedoch eine kostengünstige Produktion und einen einfacheren Vertrieb erlaubt.

IV.2. Anforderungen

Folgende Anforderungen werden an die Neuentwicklung einer Steuerung eines Mixed-Reality Roboters gestellt:

- Kostengünstige Produktion
- Hochverfügbare Bauteile
- Einfache Handhabung
- Einfache Programmierung
- Einfache Erweiterungsmöglichkeiten
- Lange Laufzeit
- Autonome Stromversorgung
- Kompatibilität mit bestehendem System
- Messung von relevanten Strömen und Spannungen des Systems
- Optimierung in Hinblick auf eine industriell gefertigte Kleinserie

IV.3. Vorgaben

Das Gehäuse des mrShark besteht aus zwei Seitenteilen und zwei Rädern mit integriertem Zahnrad, sowie zwei weiteren Zahnrädern für eine Getriebeübersetzung. Die Seitenteile des mrShark lassen sich zusammenstecken, sodass eine einfache Montage möglich ist. Im Inneren des Gehäuses befinden sich am Boden zwei GM15 DC Getriebemotoren, welche an der Gehäuseaußenseite über ein Getriebe mit den Rädern des mrShark verbunden sind. Über den Motoren im Inneren des mrShark befindet sich Platz für eine Stromversorgung. Auf dem Gehäuse wird die Steuerungselektronik mit Hilfe von vier Schrauben befestigt.

Für das zu Entwickelnde System gelten einige Vorgaben, deren Festlegung nicht Teil dieser Arbeit, für die Entwicklung der Steuerelektronik jedoch obligatorisch sind:

- Platinenaußenmaße 31,0 x 26,0 mm (rechteckig) mit 45° Phase an allen vier Ecken
- Bauraum für Stromversorgung: 22,2 x 21,6 x 13,3 mm (LxBxH) (siehe Anhang G: Vorgaben Gehäuse mrShark)
- 2 GM15 DC Getriebemotoren als Antrieb (SOLARBOTIC, 2013, GM15 datasheet)
- Steuerung via 115,2 kBit/s Infrarotsignal

IV.4. Namensgebung

Für den neuen Roboter wurde durch die betreuende NorthernStars Hochschulgruppe für Robotik an der Fachhochschule Kiel der Name mrShark gewählt. Dieser bildet sich aus dem Kürzel mr für Mixed-Reality und dem Wort Shark (englisch: Hai) als Anlehnung an den maritimen Standort der Fachhochschule Kiel (NorthernStars, 2013, mrShark).

Im Laufe der Entwicklung zeigte sich, dass das Kürzel mr von vielen Personen nicht als Abkürzung für die Mixed-Reality sondern als die englische Anrede Mr. angesehen wurde. Diese Interpretation wurde schlussendlich übernommen, sodass sich der Name mrShark wir Mr. Shark ausspricht.

IV.5. Konzept

IV.5.1. Lastenheft

Aus den oben definierten Anforderungen leiten sich folgende Systemfunktionen ab, welche durch Hard- oder Software zu realisieren sind:

- Auswahl eines Mikrocontrollers für Steuerungsaufgaben
- Auswahl eines Motortreibers für GM15 DC Motoren
- Auswahl einer Technologie für Motorstrom- und Spannungsmessung
- Entwicklung einer 4-Kanal RGB-LED Steuerung
- Auswahl einer Infrarot-Kommunikations-Schnittstelle
- Entwicklung einer Systemschnittstelle für Erweiterungen
- Auswahl und Dimensionierung einer Stromversorgung
- Einbindung der Stromversorgung in die Systemschnittstelle (Ladetechnik-Anschluss)

IV.5.2. Konzept Steuerungsplatine

Für die Steuerungselektronik wird ein Mikrocontroller als zentrale Steuereinheit verwendet (siehe Abbildung 1). Über zwei Motortreiber werden die beiden GM15 DC Getriebemotoren angesteuert, deren Stromaufnahme mit Hilfe eines Analog-Digital-Converters (ADC) gemessen wird. Ein weiterer ADC ist für das Monitoring der Betriebsspannung, sowie der Spannung(-en) der Stromversorgung zuständig.

Über einen Infrarot-Reciever (IR-Reciever) können Daten drahtlos an den mrShark gesendet werden. Ein Infrarot-Decoder (IR-Decoder) wandelt die Daten des IR-Recievers in für den Mikrocontroller kompatible Daten um. Mit Hilfe von 4 RGB-LEDs lässt sich das Gehäuse des mrShark beleuchten, sowie Zustände des mrShark anzeigen.

Eine Systemschnittstelle bietet Zugriff auf einzelne Teile des Systems, wie Stromversorgung, Betriebsspannung, Kommunikationsschnittstellen, etc., sodass der mrShark leicht durch weitere Elektronik in seinem Funktionsumfang erweitert werden kann.

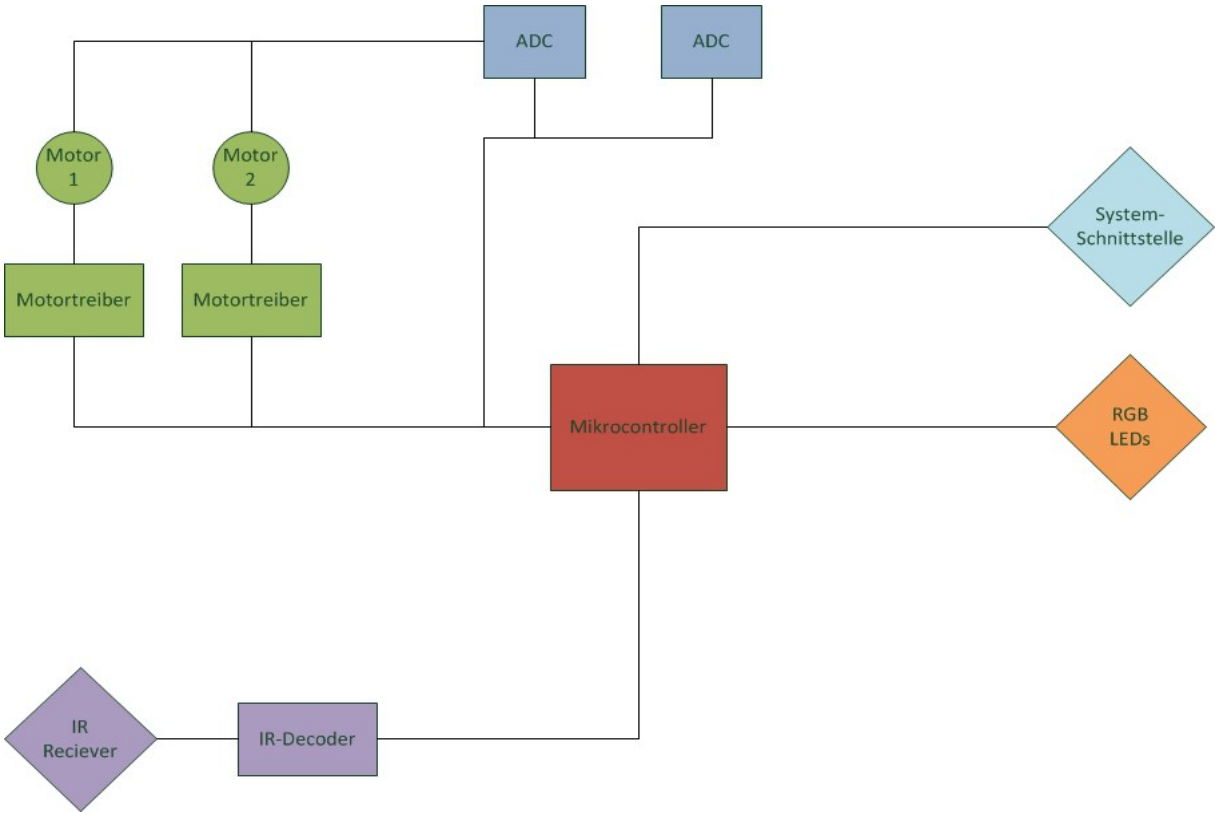


Abbildung 1: Konzept Steuerungselektronik mrShark

IV.5.3. Konzept Stromversorgung

Da für die Steuerelektronik vorwiegend TTL-Pegel verwendet werden, muss die Stromversorgung eine Betriebsspannung von minimal 2,0V bis maximal 5.0V bereit stellen (Wüst, 2003). Da diese autonom sein muss und unterhalb der Steuerplatine verbaut wird, bietet sich die Verwendung von Akkumulatoren (Akkus) an. Diese sind mehrmals wieder aufladbar und müssen daher nur sehr selten ausgewechselt werden.

Um eine möglichst lange Laufzeit des Systems zu erreichen, sollten Akkumulatoren mit möglichst hoher Energiedichte verwendet werden. Der mrShark bietet die Möglichkeit zum Betrieb von zwei parallel geschalteten Akkumulatoren. Dies ermöglicht es, trotz kleinem Bauraum im Gehäuse, hohe Kapazitäten der Akkumulatoren zu erreichen, indem zwei baugleiche Akkumulatoren übereinander verbaut werden.

IV.6. Steuerungsplatine

IV.6.1. Kriterien für Bauteilauswahl

Aus den Anforderungen und Vorgaben an die Steuerelektronik des mrShark gehen einige Kriterien für die Auswahl elektronischer Bauteile für die Steuerplatine hervor.

Insbesondere ist die Auswahl kostengünstiger Bauteile notwendig, um den Preis für einen Roboter möglichst niedrig zu halten. Die Bauteilauswahl bestimmt dabei auch die Kosten die durch die industrielle Fertigung entstehen. So ist die Wahl von möglichst wenig unterschiedlichen Bauteilen sinnvoll, da dies die Rüstkosten der Bestückungsmaschine niedrig hält. Zudem ist zu beachten, dass die Bestückungsmaschinen nur eine begrenzte Anzahl von Bauelementen aufnehmen können. Werden mehr Bauteile benötigt, als die, mit denen die Maschine ausgerüstet werden kann, entstehen zusätzliche Kosten.

Bei der Betrachtung des Bauteilpreises sind jedoch die Mindestabnahmemengen zu beachten, da diese oft erheblich höher liegen, als die Anzahl tatsächlich benötigter Bauteile. Bei sehr häufig verwendeten Standartbauteilen wie Widerständen oder Kondensatoren ist dies kein Problem, da Bestückungsunternehmen diese meist in großen Menge bestellen und Überlieferungen einlagern. Bei keinen sehr häufig verwendeten Bauteilen entstehen hier jedoch hohe Lagerhaltungskosten oder der Preis für die Überlieferung muss durch den Kunden mit getragen werden, ohne dass die Bauteile in der Produktion Verwendung finden.

Auch die Auswahl von Bauteilen aus möglichst nur einem Sortiment eines Bauteillieferanten ist anzuraten, da dadurch Versandkosten sowie Arbeitszeit im Einkauf gespart werden. Zudem sind bei größeren Bestellungen bei vielen Lieferanten Rabatte möglich, wobei bei geringen Abnahmemengen oft Mindermengenzuschläge zu entrichten sind.

Wichtig für einen möglichst langen Produktzyklus[1] ist die Auswahl von möglichst vielen Standartbauteilen, welche bei Lieferanten in großer Menge gelagert sind. Spezialbauteile sind dagegen meist nur nach langer Lieferzeit verfügbar und werden oft nur für eine sehr kurze Zeitspanne hergestellt. Die Verwendung von Standartbauteilen reduziert ebenfalls die

1 Zeitraum, in dem ein Produkt am Markt verfügbar ist, bevor es gegen ein neueres Produkt ausgetauscht wird.

Rüstkosten für die Platinenbestückung, da die Datenbank für die Bestückungsmaschinen oft bereits Daten über diese Bauteile beinhaltet und keine Kosten für die Einrichtung neuer Daten entstehen.

Auf Grund der in den Vorgaben definierten sehr kleinen Platinenfläche sind Bauteile mit geringen Abmaßen vorzuziehen. Diese sind oft nur geringfügig teurer als größere Varianten des Bauteils, lassen sich jedoch erheblich einfacher auf der Platine platzieren und erlauben eine kompakte und einfache Bauweise der Platine, welches sich in geringeren Fertigungskosten für die Leiterplatte widerspiegelt. Die für den mrShark ausgewählten Bauteile wurden daraus folgernd der folgenden Reihenfolge nach mit absteigender Wichtigkeit ausgewählt:

- Verfügbar im Sortiment eines großen weltweiten Lieferanten
- Verfügbar auch in kleinen Mengen
- Geringe Bauform verfügbar
- Günstiger Preis
- Standartbauteil (große Stückzahlen beim Lieferanten lagernd)

IV.6.2. Mikrocontroller

Die Auswahl eines geeigneten Mikrocontrollers für die zentrale Steuereinheit der Steuerelektronik richtet sich stark nach dem Anwendungsgebiet und der weiteren verwendeten Hardware. Da der mrShark über eine Systemschnittstelle erweiterbar sein soll, ist der Einsatz eines Mikrocontrollers mit möglichst vielen verschiedenen Schnittellen sinnvoll. Dadurch lassen sich die unterschiedlichsten Erweiterungen mit dem mrShark verbinden. Durch den autonomen Betrieb, ist ein Mikrocontroller mit möglichst geringem Stromverbrauch zu wählen.

Ebenfalls von zentraler Bedeutung ist die Wahl eines Mikrocontrollers, welcher in der Lage ist einzelne Befehle in möglichst kurzer Zeit auszuführen und sich leicht, am Besten durch eine Hochsprache, programmieren lässt.

Generell gibt es bei Mikrocontrollern im Speziellen, sowie Mikroprozessoren im Allgemeinen verschiedene Systemarchitekturen. Basis für die meisten Architekturen ist die Von-Neumann- oder Harvard-Architektur. Erstere wurde 1946 von Burks, Goldstine und von Neumann entwickelt und basiert auf einer Zentraleinheit mit integriertem Rechen- und Steuerwerk (Beierlein & Hagenbruch, 2011). Über ein Bussystem werden Programmcode und Programmdaten aus/in einem Speicher geladen/geschrieben. Über dieses Bussystem kann zudem ein Eingabe-/Ausgabewerk angesprochen werden (siehe Abbildung 2).

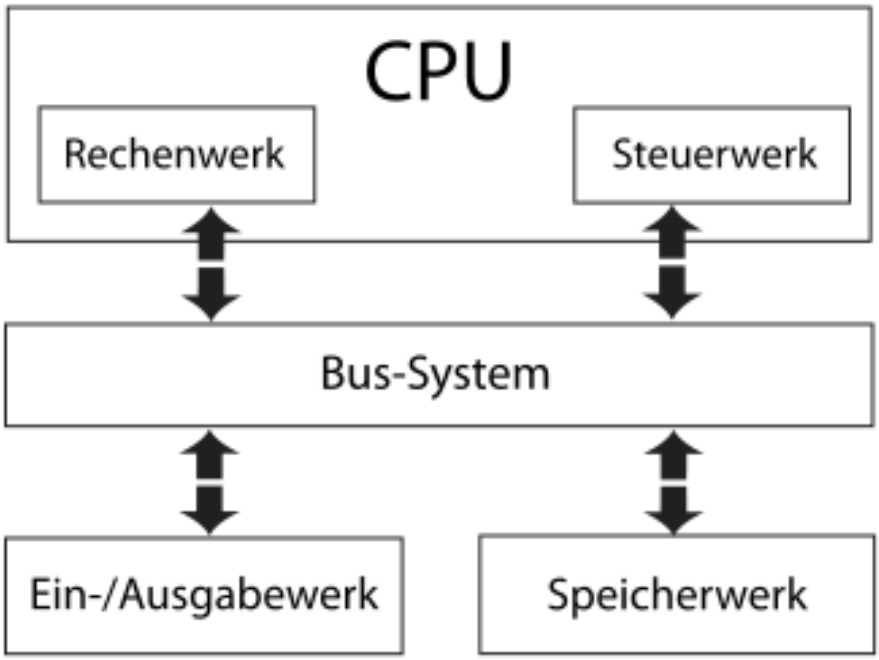


Abbildung 2: Aufbau der Von-Neumann-Architektur (Grossar, 2007)

Eine Variante der Von-Neumann-Architektur ist die Harvard-Architektur. Diese entspricht bis auf einem Punkt der Von-Neumann-Architektur, unterteilt aber Programmcode und Programmdaten in zwei verschiedene, von einander getrennte Speicherbereiche. Die Harvard-Architektur bietet dadurch schnellere Rechenzeiten, da auf Programmcode und Programmdaten gleichzeitig zugegriffen werden kann. Die Ausführungszeit einer Applikation durch einen Mikroprozessor lässt sich laut (Beierlein & Hagenbruch, 2011) Gl. 3.1, S. 72 wie folgt ausdrücken:

$$T_{app} = TPA * OPT * IPO * CPI * TPC \quad (1)$$

T_{app} steht für die Ausführungszeit der Applikation, TPA (Threads per application) für die Anzahl der Threads pro Applikation, OPT (Operations per thread) für die maximale Anzahl an Adressoperationen pro Thread. IPO (Instructions per Operation) beschreibt die mittlere Anzahl von Instruktionen pro Operation, CPI (Clocks per instruction) die mittlere Anzahl von Takten pro Instruktion und TPC (Time per clock cycle) die inverse Taktrate, also die Zeit für einen Systemtakt. Ziel einer Mikroprozessor-Architektur ist es T_{app} zu minimieren.

Die RISC-Architektur optimiert die Parameter CPI und TPC dahingehend, dass sie nur einen begrenzten Befehlssatz aufweist, welche fest in der Chip-Architektur implementiert sind und so eine Ausführungszeit von einem Takt pro Befehl erlauben. RISC-Prozessoren sind damit zwar nur in einem begrenzten Umfeld einsetzbar, bietet dafür aber eine für dieses Umfeld optimierte Programmcodeausführung. RISC-Prozessoren bieten sich daher für fest definierte und übersichtliche Steueraufgaben an.

Mikrocontroller als Untergruppe der Mikroprozessoren haben gegenüber der klassischen Von-Neumann oder Harvard-Architektur den Vorteil, dass sich alle Komponenten wie Rechen- und Steuerwerk, Programmspeicher und Datenspeicher, sowie Ein-/Ausgabeperipherie in einem integriertem Schaltkreis (IC, englisch: Integrated Circuit) befinden und daher gerade für kleine Steueraufgaben in der Elektronik eine Platz- und Kostenersparnis bieten, da weitere Komponenten wie externe Speicher entfallen.

Die einzelnen Mikrocontroller unterscheiden sich maßgeblich in Ihrer Ausstattung an Peripheriemodulen, Speichergrößen und maximalen Systemtakts. Für kleinere Aufgaben und besonders im Hobbybereich beliebt sind Mikrocontroller der Hersteller Atmel und Microchip, welche zu den 5 Herstellern mit den meisten Marktanteilen im Bereich Mikrocontroller gehören (Riemenschneider, 2012).

Atmel bietet mit seinen ATmega und ATtiny Mikrocontrollerfamilien weit verbreitete 8-Bit Mikrocontroller an. Diese verfügen alle über 8-Bit breite Register und sind in vielen, auch energiesparenden Varianten verfügbar. Zudem existieren bereits viele vorgefertigte Programmbibliotheken und eine große Online-community, was das Einsteigen in die Mikrocontrollerprogrammierung vereinfacht. Zudem existieren im Gegensatz zu PIC-Mikrocontrollern des Konkurrenten viele kostenfreie Programme zum Entwickeln und Ausspielen von Programmcodes.

Die weitere Verbreitung, sowie die große Auswahl an Bibliotheken, Hilfen und Entwicklungsprogrammen waren ausschlaggebend für die Wahl eines 8-Bit Mikrocontrollers der Firma Atmel für den mrShark.

Für die Wahl eines Mikrocontrollers der Firma Atmel kommen mehrere verschiedene Modelle in Frage, welche sich durch Ihren maximalen Systemtakt F_{max}, die Größe des Programmspeichers (Flash), des Datenspeichers (SRAM) sowie die verfügbare Peripherie unterscheiden. Die Tabelle 1 gibt eine Übersicht über eine Auswahl an geeigneten Mikrocontrollern.

Mikrocontroller	**F_{max} [MHz]**	**Flash [Kb]**	**SRAM [kB]**	**Peripherie**
ATtiny4313	20	4	0,256	4x PWM, 2 Timer, SPI, I2C, UART,
ATmega8	16	8	1	6-8x ADC, 3x PWM, 3 Timer, SPI, I2C, UART
ATmega48	20	4	0,512	6-8x ADC, 6x PWM, 3 Timer, SPI, I2C, UART
ATmega88	20	8	1	6-8x ADC, 6x PWM, 3 Timer, SPI, I2C, UART
ATmega168	20	16	1	6-8x ADC, 6x PWM, 3 Timer, SPI, I2C, UART
ATmega64	16	64	4	8x ADC, 8x PWM, 4 Timer, SPI, I2C, UART

Tabelle 1: Übersicht AVR Mikrocontroller (Schwarz, 2013, AVR Typen)

Neben den in Tabelle 1 dargestellten Mikrocontrollern gibt es weitere ähnliche Varianten. Die Aufgelisteten sind dabei die Repräsentativsten, welche auch am häufigsten eingesetzt werden. Eine Ausnahme bildet der ATtiny4313, welcher seltener zum Einsatz kommt aber, als einzige Variante der ATtiny-Familie einen ausreichend großen Flash-Speicher besitzt, um die Steuerung des mrShark zu gewährleisten.

Der benötigte Programmspeicher für den mrShark sollte Schätzungsweise[2] mindestens 4 KByte betragen, um die im Lastenheft definierten Funktionen gewährleisten zu können. Die Größe des SRAM ist entscheidend für die von der Steuersoftware generierten Daten. So sind

2 Basierend auf persönlicher Erfahrungen

allein mindestens 256 Byte notwendig, um die CRC-Prüfsummen für das Infrarot-Steuerprotokoll aufzunehmen (siehe Infrarot-Steuerprotokoll). Ein weiteres Kriterium für die Auswahl des Mikrocontrollers ist die Stromaufnahme, welche in Abbildung 3 zu sehen ist.

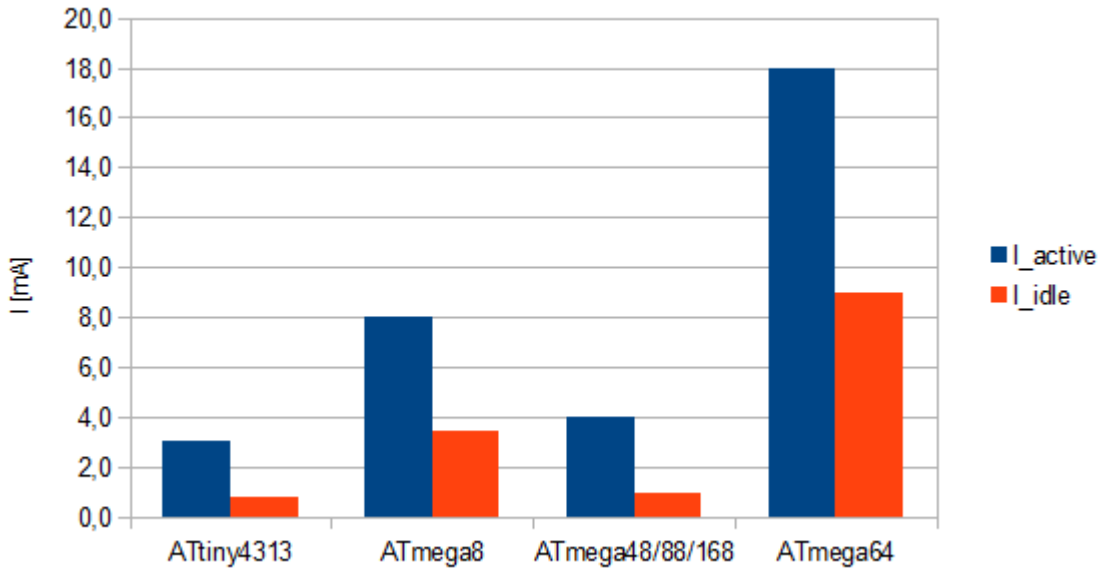


Abbildung 3: AVR Supply-Current (Anhang A: Berechnungen)

Die Abbildung zeigt die Stromaufnahme der verschiedenen Mikrocontroller-Typen in mA mit einem Systemtakt von ca. 11 MHz (für Berechnung siehe Bauteilparameter), sowohl im Active-, als auch im Idle-Mode. Der Active-Mode ist der normale Betriebszustand, in dem alle internen Module des Mikrocontrollers aktiv sind. Im Idle-Mode wird die Ausführung des Hauptprogramms unterbrochen und der Mikrocontroller versorgt nur noch die Peripherie. Dieser Modus reduziert den Stromverbrauch drastisch, erlaubt aber ein schnelles Wiederaufnehmen des Hauptprogramms.

Die Mikrocontroller ATmega48/-88/-168 sind zudem kompatibel, sowohl was die Programmierung, als auch die Gehäuseformen und Pin-Belegungen angeht. Sie können deshalb beliebig gegeneinander ausgetauscht werden, da sie sich nur in der Größe des Flash-Speichers unterscheiden.

Auf Basis der o.g. Daten verwendet der mrShark einen ATMEGA168V Mikrocontroller. Dieser bietet mit 16 kByte Flash-Speicher ausreichend Platz für die Firmware des Roboters und verbraucht dabei ca. 71% - 76% weniger Strom als ein ATmega8, ca. 64% - 86% weniger Strom als ein ATmega64 und bis zu 25% mehr Strom als ein ATtiny4313. Im Active-Mode verbraucht der ATMEGA168V-10MU nur ca. 22% mehr als ein ATtiny4313. Im Gegensatz zu allen anderen in Tabelle 1 gelisteten Mikrocontrollern, kann der ATmega64 bei dem geforderten Systemtakt nur bei einer Betriebsspannung von mindestens 4,5V verarbeiten.

IV.6.3. Motortreiber

Ein Motortreiber ist eine Schaltung, welche die Signale einer Steuereinheit in entsprechende Steuersignale für einen Motor umsetzt. Dies wird notwendig, da Motoren in der Regel einen höheren Strom zum Betrieb benötigen, als digitale Steuerbauteile liefern können.

Die im mrShark verwendeten GM15 Getriebemotoren sind einfache DC-Motoren und können über eine variierende Gleichspannung in ihrer Drehgeschwindigkeit beeinflusst werden. Dabei besteht ein linearer Zusammenhang zwischen der am Motor anliegenden Gleichspannung und der Drehzahl. Über die Polarität der am Motor anliegenden Spannung wird zwischen Links- und Rechts-Betrieb umgeschaltet. Liegt zwischen den Polen des Motors kein Potentialunterschied vor, wird der Motor blockiert. Eine Bewegung wird dadurch ausgebremst und eine erneute Drehung verhindert.

Für die Realisierung eines Motortreibers gibt es verschiedene Ansätze.
Die erste Variante wäre ein Umsetzung der digitalen Steuersignale in eine Analoge Spannung. Dies ließe sich mit einem Digital-Analog-Converter (kurz DAC) realisieren. Da die meisten DAC keine hohen Ströme an ihren Ausgängen bereitstellen (Urbanski, Woitowitz, 2007), wäre eine weitere Transistorschaltung nötig um den Motor anzutreiben. Darüber hinaus müsste noch zwischen Links- und Rechts-Betrieb des Motors umgestellt werden. Hierfür wären weitere Steuerschaltungen notwendig (siehe Abbildung 4).

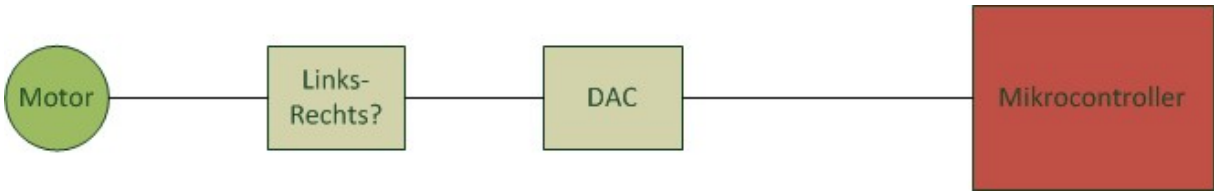


Abbildung 4: Prinzip Motortreiber mit DAC

Um die umständliche Generierung einer analogen Spannung zu umgehen, lässt sich auch ein pulsweitenmoduliertes Signal (kurz PWM-Signal) verwenden. Ein PWM ist ein digitales Signal das nur die zwei Zustände High für die maximale Spannung und Low für die minimale Spannung annimmt. Die beiden Spannungen betragen bei TTL-Bausteinen üblicherweise 5V für High und 0V für Low (Urbanski, Woitowitz, 2007).

Ein PWM-Signal besteht pro Periode T aus einer Zeit t_1 in der das Signal High und einer Zeit t_2 in der das Signal Low ist. Das Verhältnis von t_1 und t_2 in Prozent wird als Tastverhältnis α beschrieben und berechnet sich wie folgt:

$$\alpha = \frac{t_1}{T} * 100 \tag{2}$$

Da der Rotationskörper eines Motors träge ist, verhält sich ein PWM-Signal mit geeigneter Frequenz an einem Motor wie eine Analogspannung mit dem Spannungswert U_{Motor} und der maximalen Spannung U_{max} wie folgt:

$$U_{Motor} = \alpha * U_{max} \tag{3}$$

Eine geeignete Frequenz sollte so hoch sein, dass der Motor bei einem minimalen α noch flüssig dreht und so niedrig, dass der Motor bei minimalem α bereits seine minimale Drehzahl erreicht. Ein Motortreiber mittels PWM bedarf lediglich einer Steuerelektronik für den Links-/Rechts-Betrieb, sowie zur Stromentkopplung des Steuersignals vom Motor. Üblicherweise wird die Funktionalität durch eine sogenannte H-Brücke realisiert. Eine H-Brücke besteht primär aus vier Transistoren oder MOSFETs. Jeweils zwei PWM-Signale werden mit zwei im Schaltplan gegenüberliegenden Transistoren/MOSFETs verbunden. Je nach Dotierung der Transistoren/MOSFETs schalten diese im Low- oder High-Teil des PWM-Signals durch und der Motor kann sich drehen. Das andere PWM-Signal wird, je nach Dotierung der Transistoren/MOSFETs, auf α = 100% oder α = 0% gestellt um die dazugehörigen Transistoren/MOSFETs für den Stromdurchfluss zu sperren.

H-Brücken lassen sich mittels diskreter Bauteile oder als integrierte Schaltkreise (kurz IC) aufbauen. Motortreiber-ICs können dabei viele verschiedene Schnittstellen besitzen, sodass sie sich z.B. über ein Bus-System betreiben lassen. Oft sind ebenfalls bereits

Schutzmaßnahmen wie eine Notstromabschaltung integriert. In diesem Fall wird der Motor abgeschaltet, wenn er z.B. durch eine Blockade einen zu hohen Stromfluss hervorruft. Diese zusätzlichen Funktionen, einfachen Ansteuerungen und bei ICs sehr kleinen Bauformen führten zur Wahl eines Motortreiber-ICs für den mrShark.

Bei der Auswahl eines geeigneten Motortreibers ist besonders die maximale Stromaufnahme des Motors zu beachten. Beim GM15 liegt diese bei maximal 340 mA bei 6V und blockiertem Motor (SOLARBOTIC, 2013, GM15 datasheet). Ein Motortreiber muss mindestens diesen Strom zuzüglich eines Sicherheitspuffers von ca. 10%, also maximal ca. 375 mA treiben können.

Bei der Wahl eines geeigneten Bussystems für den Motortreiber bietet sich der I^2C-Bus an. Dieser besteht nur aus zwei Busleitungen und ist bereits als Peripherie im Mikrocontroller des mrShark vorhanden (siehe Mikrocontroller). Nach einer Onlinerecherche beim weltweiten Elektronikdistributor[3] Farnell[4] wurde für den mrShark der Motortreiber DRV8830 von Texas Instruments ausgewählt.

Der DRV8830 verfügt über eine I^2C-Schnittstelle, eine automatische Strombegrenzung, ist in einer sehr kleinen Bauform gut erhältlich und kann einen Motor mit maximal l 1A treiben. Er erlaubt sowohl den Links- als auch den Rechts-Betrieb, als auch die Bremsung/Blockierung des Motors und arbeitet mit einem PWM-Signal (TI, 2012, DRV8830 datasheet).

Der DRV8830 benötigt einen externen Widerstand zum Einstellen der Strombegrenzung. Dieser berechnet sich laut Datenblatt (TI, 2012, DRV8830 datasheet) wie folgt:

$$R_{ISENSE} = \frac{200\text{mV}}{I_{LIMIT}} \tag{4}$$

R_{ISENSE} ist der Widerstand für die Strombegrenzung und liegt typischerweise unter 1Ω. I_{LIMIT} ist der maximale Strom, bei dessen Überschreiten der Motortreiber den Motor abschaltet. Bei einer Betriebsspannung von 5V kann laut Datenblatt des GM15 Motors ein maximaler Strom von 340 mA fließen (SOLARBOTIC, 2013, GM15 datasheet). Daraus ergibt der Wert für R_{ISENSE} wie folgt:

3 Distributor: Großhändler
4 http://www.farnell.de

$$R_{ISENSE} = \frac{200 * 10^{-3} V}{340 * 10^{-3} A} \approx 0{,}58\,\Omega \quad (5)$$

Bei einer Betriebsspannung von 3V und einem maximalen Strom von 210 mA (SOLARBOTIC, 2013, GM15 datasheet) ergibt sich dagegen ein Wert von 0,95 Ω (siehe Gleichung (6)). Der nächste reguläre verfügbare Widerstandswert zu dem errechneten Wert ist 0,91 Ω.

$$R_{ISENSE} = \frac{200 * 10^{-3} V}{210 * 10^{-3} A} \approx 0{,}95\,\Omega \quad (6)$$

IV.6.4. Analog-Digital-Wandler

Ein Analog-Digital-Wandler (kurz ADC, englisch: analog-digital-converter) wandelt eine analoge Spannung in einen digitalen Wert um. Über einen ADC lassen sich z.B. durch digitale Steuerelektroniken Spannungen und indirekt auch Ströme überwachen.

Ein ADC vergleicht dazu in der Regel eine Messspannung mit einer Referenzspannung und gibt an wie groß die Differenz zwischen beiden Spannungen ist (Herter, Lörcher, 2004).

Um einen Strom mittels eines ADC zu messen, wird ein niederohmiger Widerstand (< 1 Ω), ein sogenannter Shunt-Widerstand in die Stromleitung eingebracht. Die über dem Shunt-Widerstand abfallende Spannung ist jetzt nach dem Ohmschen Gesetz linear abhängig (Moltrecht, 1998, S. 31) vom Stromfluss in der Leitung. Der Strom lässt sich mit Hilfe der Spannungsdifferenz U_{diff} und dem Shunt-Widerstand R_{shunt} wie folgt berechnen:

$$I = \frac{U_{diff}}{R_{shunt}} \quad (7)$$

Beim mrShark sind folgende Spannungen interessant:

- Betriebsspannung
- Spannung Akku 1
- Spannung 2

Die Betriebsspannung sollte während des Betriebs kontinuierlich überwacht werden, da eine Veränderung der Betriebsspannung eine Änderung der Motordrehzahl nach sich zieht. Fällt z.B. die Betriebsspannung, sinkt auch die Motorspannung und damit die Motordrehzahl.

Die Spannungen der einzelnen Batterien für die Stromversorgung (siehe Konzept Stromversorgung) sollte ebenfalls überwacht werden um ein Unterschreiten den minimalen Batteriespannung und damit verbundene Schäden an den Batterien zu vermeiden. Darüber hinaus sind die Stromaufnahmen der beiden Motoren zu überwachen, da diese in grobem Rahmen Rückschluss auf die Motordrehzahl geben (siehe Motor Control Modul).

Insbesondere die Spannungsmessung über dem Shunt-Widerstand ist dabei nicht unkritisch. Da hier nur sehr kleine Spannungen abfallen müssen diese in der Regel noch Verstärkt werden. Dazu lässt sich ein Operationsverstärker als Differenzverstärker verwenden. Dies würde die Anzahl der benötigten Bauelemente und damit den auf der Platine benötigten Platz unnötig erhöhen, da es ebenfalls ADC als ICs gibt.

Hierbei bietet sich, wie bereits beim Motortreiber ein ADC-IC mit I^2C-Schnittstelle an. Farnell listet hier den LTC2990 von Linear Technology. Dieser kann bis zu vier Spannungen oder zwei Strömen, sowie die Temperatur messen und kommt, abgesehen von einem Shunt-Widerstand zur Strommessung, ohne externe Bauteile aus (Linear, 2010, LTC2990 datasheet). Diese Merkmale, ein günstiger Preis und eine gute Verfügbarkeit führten zur Auswahl des LTC2990 als ADC-IC für den mrShark.

Der LTC2990 misst bis zu vier Spannungen. Jeweils zwei der Spannungseingänge lassen sich für eine Strommessung konfigurieren, sodass der IC aus der Differenzspannung automatisch einen dazugehörigen Strom errechnet. Dieser kann sowohl ein positives, als auch negatives Vorzeichen haben, sodass eine Strommessung in beide Stromflussrichtungen möglich ist. Der LTC2990 misst zudem intern die Temperatur und kann ebenfalls über einen externen Transistor eine externe Temperatur messen.

Für die Strommessung wird zusätzlich zum LTC2990 ein Shunt-Widerstand benötigt, welcher sich laut Datenblatt (Linear, 2010, LTC2990 datasheet) wie folgt berechnet:

$$R_{EXTMAX} = \frac{300\text{mV}}{I_{MAX}} \tag{8}$$

Für dir GM15 Getriebemotoren des mrShark ergibt sich bei einer Betriebsspannung von ca. 5V und maximalen Strom I_{MAX} 340 mA (SOLARBOTIC, 2013, GM15 datasheet) ein maximaler Shunt-Widerstand R_{EXTMAX} von:

$$R_{EXTMAX} = \frac{300 * 10^{-3} V}{340 * 10^{-3} A} \approx 88\Omega \tag{9}$$

Bei einer Betriebsspannung von 3V und einem maximalen Strom von 210 mA (SOLARBOTIC, 2013, GM15 datasheet) errechnet sich der Shunt-Widerstand wie folgt:

$$R_{EXTMAX} = \frac{300 * 10^{-3} V}{210 * 10^{-3} A} \approx 1{,}43\,\Omega \tag{10}$$

Für die Messung der Ströme der beiden Motoren des mrShark wird ein LTC2990 benötigt. Ein weiterer LTC2990 misst die Betriebsspannung, sowie die beiden Batteriespannungen. Ein vierter Eingang zur Spannungsmessung des zweiten LTC2990 bleibt zur freien Verfügung frei und ist auf die Systemschnittstelle ausgeführt.

IV.6.5. I²C-Bus

I²C steht für Inter-Integrated-Circuit und wurde in den 1980er Jahren von Philips Semiconductors als serieller Datenbus entwickelt (Beierlein & Hagenbruch, 2011). Atmel führte aus Lizenzgründen für die eigene I²C-Implementierung die Bezeichnung Two-Wire-Interface (kurz TWI) ein. TWI unterscheidet sich jedoch nicht von der I²C-Spezifikation.

Der I²C-Bus basiert auf einem Master-Slave-Prinzip. Ein Teilnehmer des I²C-Bus ist der Master und steuert alle weiteren angeschlossenen Slave-Bausteine. Dazu verwendet der Master ausschließlich zwei Busleitungen, SDA und SCL. SCL steht für Serial-Clock-Line und wird vom Master mit einem Bustakt versorgt. Dieser bestimmt die Übertragungsrate des I²C-Bus und muss von allen Slaves unterstützt werden. SDA steht für Serial-Data-Line und dient der Übertragung der eigentlichen Daten.

Eine Kommunikation kann bei einem I²C-Bus nur vom Master begonnen werden (Beierlein & Hagenbruch, 2011). Möchten Slaves eine Kommunikation starten, wird dieses meist durch zusätzliche Interruptleitungen (siehe Interrupts) realisiert.

Um eine Kommunikation zu starten sendet der Master eine Anfrage an den Client. Dazu sendet der Master eine Anfrage zum Lesen oder Schreiben mit der Adresse des Slaves an den Bus. Antwortet der Slave kann der Master anschließend eine Registeradresse des Slaves senden und erhält bei einer Leseanfrage vom Slave den Registerinhalt oder schickt bei einer Schreibanfrage die Daten zum Beschreiben des gewählten Registers an den Slave.

Der mrShark verfügt über einen internen I^2C-Bus mit dem Mikrocontroller als Master und drei weiteren Slaves (siehe Motortreiber und Analog-Digital-Wandler). Um ein Auslesen der einzelnen Slaves über externe Hardware, wie z.B. eine Ladeschaltung, zu ermöglichen, ist der I^2C-Bus auf die Systemschnittstelle ausgeführt (siehe Systemschnittstelle). Um darüber hinaus auch mehrere mrShark-Roboter über einen I^2C-Bus zusammenzuschließen, muss der interne I^2C-Bus von der Systemschnittstelle entkoppelt werden, da andernfalls zwei Slaves mit identischer Adresse an den I^2C-Bus angeschlossen sind. Dazu verwendet der mrShark den PCA93540 des Herstellers Philips Semiconductors. Dieser entkoppelt die beiden I^2C-Bussysteme von mrShark und Systemschnittstelle, kann sie aber, gesteuert durch den Mikrocontroller, wieder zusammenführen. Der PCA9540 benötigt außer Pull-Up-Widerständen für die I^2C-Leitungen keine externen Bauteile und eignet sich dadurch sehr gut für den Einsatz bei geringem Platz wie dem mrShark.

Die Pull-Up-Widerstände begrenzen den Stromfluss durch den PCA9540 auf maximal 20 mA pro Leitung (Philips, 1999, PCA9540 datasheet). Dieser berechnet sich gemäß dem Ohmschen Gesetz entsprechend der Gleichung (11). R_{PU} ist der minimal zulässige Widerstandswert für den Pull-Up-Widerstand.

$$R_{PU} = \frac{Vcc}{0{,}02\,A} \tag{11}$$

Um in einem Verbund von mehreren mrShark über einen gemeinsamen I^2C-Bus auf Bausteine eines speziellen Roboters zuzugreifen muss der Roboter durch Aktivieren eines Interrupts (siehe Interrupts) in den Slave-Modus geschaltet werden. Danach lässt sich der mrShark über den I^2C-Bus ansteuern und dazu veranlassen die internen Bausteine auf den gemeinsamen I^2C-Bus zu schalten.

IV.6.6. Infrarot-Schnittstelle

Der mrShark empfängt seine Steuerbefehle drahtlos über eine Infrarot-Schnittstelle. Die Infrarot-Schnittstelle empfängt durch Lichtimpulse im infraroten Lichtspektrum übertragene serielle Daten und stellt diese dem Mikrocontroller über eine UART-Schnittstelle zur Verfügung (siehe Remote Control Modul).

Die Infrarotsignale werden durch einen TFBS4711 Transciever-Baustein von Vishey empfangen und an einen TIR1000 Infrarot De-/Encoder von Texas Instruments weitergeleitet. Dieser wandelt die Infrarot-Daten in UART-konforme Daten um und stellt sie für den Mikrocontroller bereit. Der Transciever erlaubt maximale Übertragungsraten von bis zu 115,2 kBit/s. Um eine möglichst große Menge an Daten pro Zeiteinheit übertragen zu können, verwendet der mrShark diese Geschwindigkeit für alle UART-Schnittstellen. Als einziges externes Bauteil benötigt der TFBS4711 einen Vorwiderstand für die Infrarot-Diode zum Senden von Infrarot-Daten. Über einen Shutdown-Eingang kann der TFBS4711 ausgeschaltet werden, sodass er weniger Strom verbraucht.

Der TIR1000 benötigt außer einer Taktversorgung die dem 16-fachen der benutzen Übertragungsgeschwindigkeit entsprechen muss keine weiteren Bauteile (TI, 1995/1999, TIR1000 datasheet). Bei einer Übertragungsgeschwindigkeit von f_{BAUD} = 115,2 kHz beträgt die Taktversorgung $f_{TIR1000}$:

$$f_{TIR1000} = f_{BAUD} * 16 = 115{,}2\,kHz * 16 = 1{,}8432\,MHz \qquad (12)$$

IV.6.7. RGB-LEDs

Um verschiedene Zustände des mrShark zu visualisieren, befinden sich vier RGB-LEDs auf der Steuerplatine. RGB-LEDs sind drei in einem Gehäuse kombinierte Leuchtdioden, welche durch die Mischung der drei Grundfarben Rot, Grün und Blau fast jede Farbe des sichtbaren Lichtspektrums annehmen können.

RGB-LEDs eignen sich besonders gut zur Beleuchtung, da sich ihre Farbe durch den RGB-Farbraum darstellen lässt. Dieser stell üblicherweise Farben über drei 8-Bit breite Werte für die Farben Rot, Grün und Blau dar. Dieses Format eignet sich durch die 8-Bit breiten Werte daher sehr gut für den Einsatz in Mikrocontrollern, da diese oft über 8-Bit breite Register verfügen.

Um eine Farbe mit einer RGB-LED darzustellen müssen die Lichtintensitäten der drei Grundfarben im Verhältnis zueinander angepasst werden. Die lässt sich über analoge Spannung realisieren, oder über eine Pulsweitenmodulation (kurz PWM). Wie bereits im Kapitel Motortreiber beschrieben bietet sich hier die Umsetzung mittels PWM an.

Für die Ansteuerung von vier RGB-LEDs mit jeweils drei Grundfarben wären dafür 12 Signalleitungen notwendig. Diese berechnen sich nach Gleichung (13), welche die Anzahl der benötigten Signalleitungen n_{SIG} durch die Multiplikation von k Farben pro LED mit der Gesamtzahl n_{LED} der LEDs darstellt.

$$n_{SIG} = n_{LED} * k \tag{13}$$

Die benötigten Datenleitungen lassen sich noch weiter reduzieren, indem für alle RGB-LEDs dieselben Signalleitungen für die drei Grundfarben benutzt werden und die RGB-LEDs abwechseln betrieben werden. Dadurch reduziert sich die Anzahl die benötigten Signalleitungen auf die Anzahl der Farben. Dafür wird jedoch pro RGB-LED eine Steuerleitung zum Ein- und Ausschalten der gesamten RGB-LED benötigt. Nach Gleichung (14) ergeben sich für k = 3 Farben und $n_{\_LED}$ = 4 RGB-LEDs jedoch nur n_{SIG} = 5 Leitungen.

$$n_{SIG} = n_{LED} * k \tag{14}$$

Wird zwischen den RGB-LEDs schnell genug umgeschaltet, ist dies für das menschliche Auge nicht mehr sichtbar und es erscheint so, als würden alle LEDs kontinuierlich leuchten. Dabei ist zu beachten, dass eine zu niedrige Frequenz zum Ansteuern der RGB-LEDs zu einem optischen Flackern der LEDs führt. Eine zu hohe Frequenz hingegen resultiert in einem durchgehenden Leuchten der LEDs, da dieses eine bestimmte Zeit brauche, um sich an und ab zu schalten.

Die Steuerung der RGB-LEDs erfolgt dabei in mehreren Schritten. Jede der vier RGB-LEDs wird für 256 Durchläufe aktiviert. In diesem Zeitrahmen kann die RGB-LED durch Steuerung der drei Grundfarben zum leuchten gebracht werden. Die Grundfarben werden dazu entsprechend ihrem RGB-Farbwert für eine bestimmte Zeit der 256 Durchläufe an, oder aus geschaltet. Nach Ablauf der 256-Durchläufe wird die RGB-LED wieder deaktiviert und die nächste RGB-LED aktiviert. Dies wiederholt sich fortlaufend für alle vier RGB-LEDs.

Die Abbildung 5, 6, 7 und 8 zeigen beispielhaft die Signalverläufe der drei Grundfarben für vier RGB-LEDs mit den in Tabelle 2 aufgelisteten Farbanteilen und hexadezimalen Farbwerten, sowie die Zeiten, in denen die vier RGB-LEDs aktiv sind. Die Signale wechseln dabei zwischen den Zuständen 1 für logisch aktiv / HIGH und 0 für logisch inaktiv / LOW.

	Anteil Rot	**Anteil Grün**	**Anteil Blau**	**Farbwert [hex]**
LED1	71,4%	89,0%	0,0%	0xB6E300
LED2	12,5%	65,1%	47,1%	0x20A678
LED3	0,0%	59,6%	88,6%	0x0098E2
LED4	0,0%	0,0%	12,5%	0x000020

Tabelle 2: Farbanteile und-werte RGB-LEDs

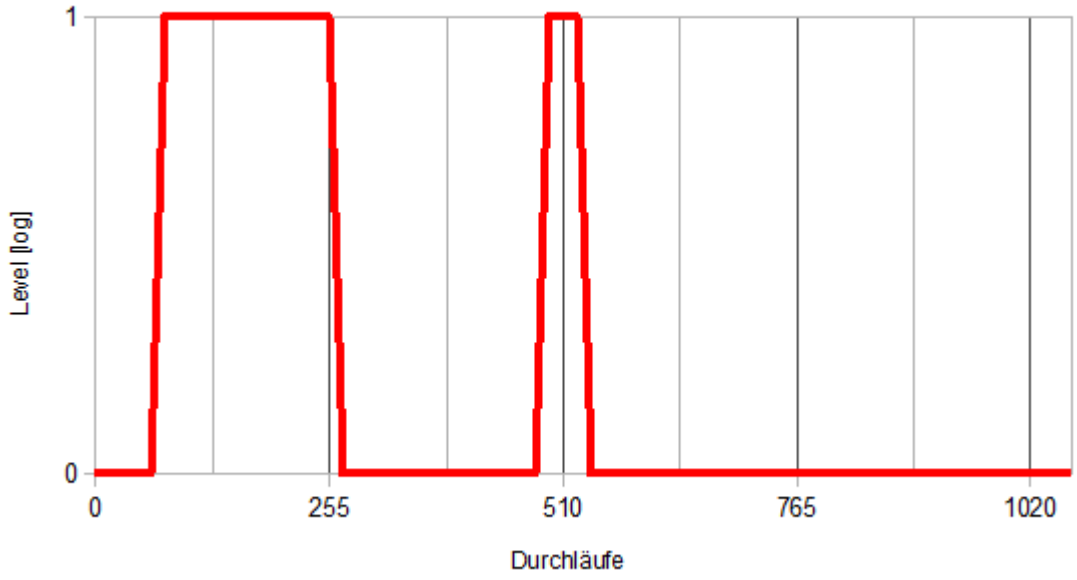


Abbildung 5: Signalverlauf RGB-LEDs Farbe Rot

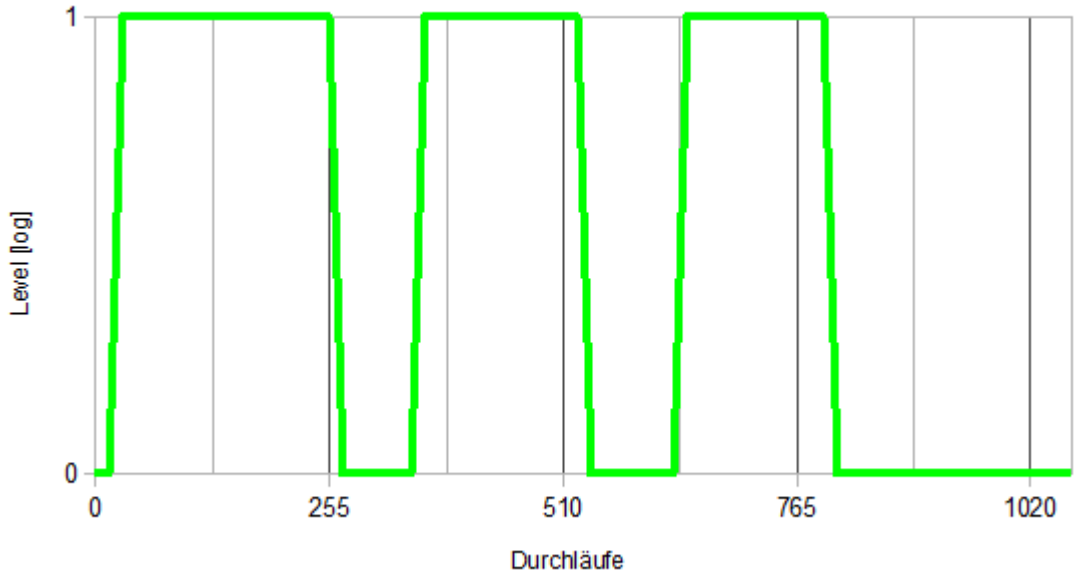


Abbildung 6: Signalverlauf RGB-LEDs Farbe Grün

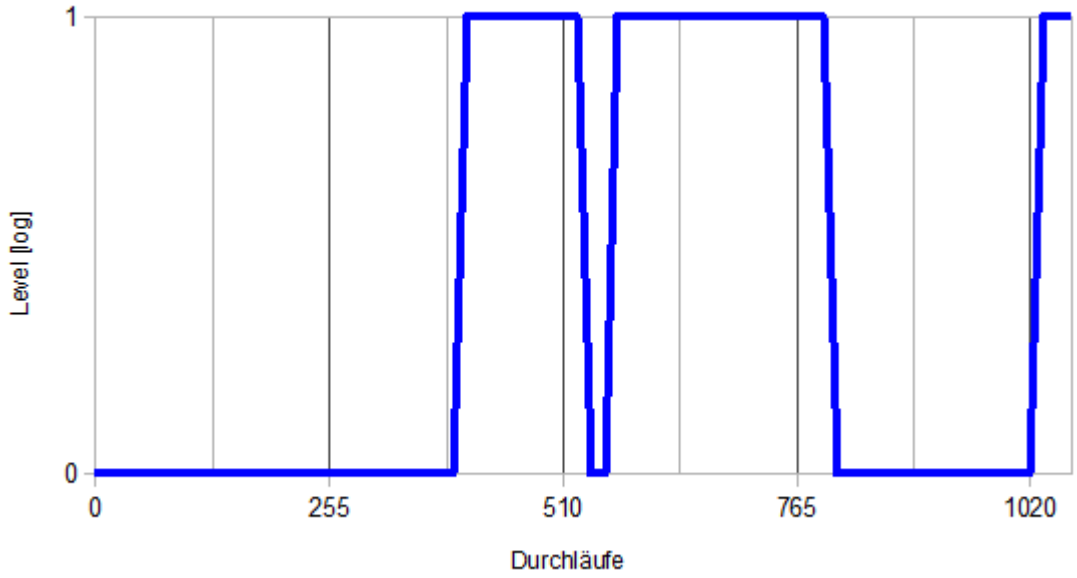


Abbildung 7: Signalverlauf RGB-LEDs Farbe Blau

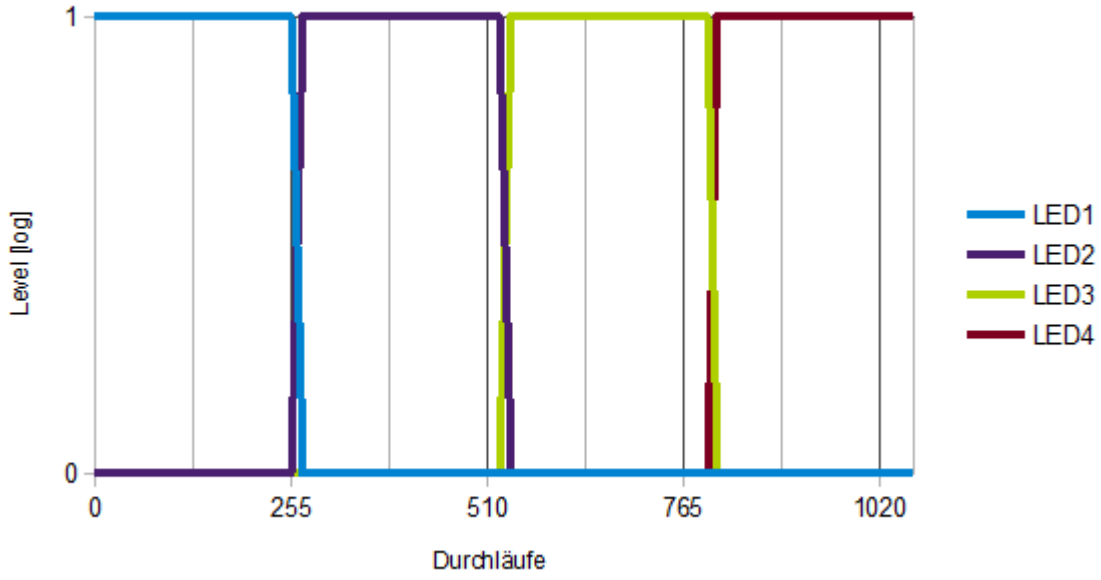


Abbildung 8: Signalverlauf RGB-LED-Steuersignale

Der mrShark verwendet RGB-LEDs vom Typ ASMT-YTD2-0BB02. Diese haben eine kleine Bauform und sind kostengünstig. Die einzelnen Farben weisen bei einem Stromfluss von 20 mA eine gleiche Lichtintensität auf (Avago, 2010, ASMT-YTD2-0BB02 datasheet), was für

die Realisierung des Farbtons weiß entscheidend ist, da bei diesem Farbton alle LED-Farben gleich hell leuchten müssen. Dies ist demnach auch der maximale Strom pro Farbe einer RGB-LED.

IV.6.8. Weitere Hardwarekomponenten

Als Hauptschalter verfügt der mrShark über ein Slide-Switch von C&K Components. Über diesen Schiebeschalter lässt sich die Stromversorgung entweder auf die Systemschnittstelle oder die Betriebsspannung umschalten. Dadurch wird verhindert, dass das mrShark während des Ladevorgangs der Akkumulatoren der Stromversorgung aus eben diesen mit Strom gespeist wird.

An den Mikrocontroller ist eine zusätzliche Status-LED angeschlossen, welche zum Anzeigen einfacher Zustände genutzt werden kann. Darüber hinaus ist eine weitere LED an die Betriebsspannung angeschlossen. Sie leuchtet, sobald der mrShark in Betrieb ist.

Neben diversen Pull-Down-/Pull-Up-Widerständen und Kondensatoren zum Abblocken von Störimpulsen (siehe Bauteilparameter) auf der Betriebsspannung, ist ein 1,8432 MHz Oszillator für die Generierung eines Systemtaktes für die Infrarot-Schnittstelle (siehe Infrarot-Schnittstelle) und ein 11,0592 MHz Quarz (siehe Bauteilparameter) für den Mikrocontroller verbaut.

IV.7. Stromversorgung

IV.7.1. Übersicht gängiger Akkumulatoren

Für die Stromversorgung des mrShark sind zwei parallel geschaltete Akkumulatoren vorgesehen (siehe Konzept Stromversorgung). Bei der Auswahl eines Akkumulators gibt es verschiedene Typen zur Auswahl.

Die erste Generation von Akkumulatoren waren Nickel-Cadmium-Akkus (Ni-Cd-Akkus), welche besonders günstig zu fertigen sind, bei „unsachgemäßer Entsorgung [aber] äußerst giftig“ (Frehner, 2007) sind. Darüber hinaus ist der Memory-Effekt der Ni-Cd-Akkus sehr nachteilig zu bewerten. Ni-Cd-Akkus lassen sich im Gegensatz zu vielen anderen Akkumulatoren auch noch bei sehr niedrigen Temperaturen betreiben.

Nickel-Metallhybrid-Akkumulatoren (Ni-MH-Akkus) sind die Fortführung der Ni-Cd-Akkus und besitzen bei gleicher Bauform eine um bis zu 80% gesteigerte Kapazität. Sie sind zudem umweltverträglich, können aber im Gegensatz zu Ni-Cd-Akkus nicht bei sehr niedrigen Temperaturen eingesetzt werden (Frehner, 2007).

Lithium-Ionen-Akkus (LiIon-Akkus) haben gegenüber Ni-MH-Akkus nochmal eine deutlich gesteigerte Energiedichte und eignen sich „besonders für Geräte mit langen Betriebs- und Standbyzeiten“ (Frehner, 2007). Da LiIon-Akkus eine untypische Spannung von 3,6V pro Zelle haben, können sie nicht direkt andere Akku-Typen ersetzen. LiIon-Akkus sind nur für Temperaturen von 5°C - 30°C einsetzbar, haben dafür aber keinerlei Memory-Effekt (Moraw, 2013). Die Ladetechnik von LiIon-Akkus ist zudem komplexer, die Ladezeiten aber dafür kürzer als bei Ni-Cd- oder Ni-MH-Akkus gleicher Kapazität (Frehner, 2007). Für LiIon-Akkus sind zudem aufwendige Schutzschaltungen notwendig (Hüfner, 2004).

Lithium-Polymer-Akkus (LiPo-Akkus) sind Weiterentwicklungen der LiIon-Akkus und technisch noch nicht voll ausgereift (Frehner, 2007). Sie lassen sich günstig herstellen und sind auch in kleinen Bauformen erhältlich. Die Ladetechnik von LiPo-Akkus entspricht in ihrer Komplexität denen von LiIon-Akkus.

Am längsten am Markt sind Blei bzw. Blei-Gel Akkus vertreten (Hüfner, 2004). Sind jedoch schwer und nur in der Blei-Gel-Variante lageunabhängig zu betreiben. Blei-Akkus weisen darüber hinaus oft eine große Bauform auf und haben ein sehr hohes Gewicht.

IV.7.2. Bewertung und Auswahl von Akkumulatoren für den mrShark

Die folgende Tabelle bewertet die verschiedenen Akkumulator-Typen aus dem Kapitel Übersicht gängiger Akkumulatoren bezüglich ihrer Nutzung im mrShark. Positive Merkmale sind mit einem +, neutrale mit einem o und negative Merkmale mit einem - gekennzeichnet.

Für den mrShark sind insbesondere Merkmale wie Energiedichte und Bauform, sowie kein Memory-Effekt entscheidend. Das Gewicht sollte nicht zu hoch sein, da sehr schwere Akkumulatoren von den Motoren sicher nicht mehr getragen werden könnten. Trotz der komplizierten Lade- und Schutztechnik bei LiIon- und LiPo-Akkus sind diese auf Grund ihrer hohen Energiedichte, der kleinen Bauformen und dem fehlenden Memory-Effekt anderen Akkumulatortypen vorzuziehen.

	Blei	Ni-Cd	Ni-MH	LiIon	LiPo
Energiedichte	-	-	o	+	+
Umweltverträglichkeit	-	-	+	+	+
Lageunabhängig	-	+	+	+	+
Bauformen	-	-	-	+	+
Gewicht	-	o	o	+	+
Memory-Effekt	+	-	o	+	+
Ladetechnik	+	+	+	-	-

Tabelle 3: Bewertung der Eigenschaften verschiedener Akkumulator-Typen (Hüfner, 2004)

IV.7.3. Dimensionierung der Akkumulatoren

Bei der Wahl der Akkumulatoren für den mrShark sind zwei identische Akkumulatoren zu wählen. Dadurch werden Probleme durch Ausgleichsströme vermieden. Diese können auftreten, wenn zwei parallel geschaltete Zellen nicht den selben Ladungszustand aufweisen. Dann kann ein teilweise sehr hoher Strom von der volleren zur leereren Zelle fließen.

Diesem Verhalten kann entgegen gewirkt werden, indem jeweils eine Diode vom positiven Anschluss einer Zelle zur Betriebsspannung geschaltet wird. Dadurch wird ein Stromfluss von einer in die andere Zelle über die positiven Anschlüsse der Zellen vermieden. Entscheidend ist dabei, dass die Diode in Sperrrichtung für den maximal auftretenden Strom zwischen den beiden Zellen ausgelegt ist. Fließt der Ausgleichsstrom zwischen zwei Zellen mit einem gleichen Innenwiderstand R_i und einer Spannungsdifferenz U_{diff}, dann berechnet sich der Ausgleichsstrom nach dem Ohmschen Gesetzt (Moltrecht, 1998, S. 31) wie folgt:

$$I_{ausgl} = \frac{U_{diff}}{2 * R_i} \tag{15}$$

Der Innenwiderstand einer Zelle hängt von der Temperatur und dem altersbedingten Zustand der jeweiligen Zelle ab. Die Spannungsdifferenz bei voll neuen geladenen Zellen beträgt typischerweise ca. 100 mV. Der Innenwiderstand einer neuen Zelle beträgt ca. 4 mΩ (HiSystems, 2013, LiPo-Grundlagen). Damit ergibt sich ein Ausgleichsstrom I_{ausgl} von:

$$I_{ausgl} = \frac{100 * 10^{-3} V}{2 * 4 * 10 - 3 \Omega} = 12{,}5\ A \tag{16}$$

Unter der Annahme von zwei ungleichmäßig geladenen Zellen und einer daraus resultierenden Differenzspannung U_{diff} = 1,0V bei einen Innenwiderstand von ebenfalls 4 mΩ ergibt sich folgender Ausgleichsstrom I_{ausgl}:

$$I_{ausgl} = \frac{1\text{V}}{2 * 4 * 10 - 3 \Omega} = 125\text{A} \tag{17}$$

Die Gleichungen (16) und (17) zeigen, dass bereits zwischen voll geladenen Zellen hohe Ausgleichsströme fließen können.

Für die Wahl eines geeigneten Akkumulators ist ebenfalls der Nennstrom der Zelle entscheidend. Dieser wird in der Regel in der Einheit Coulomb [C] angegeben und ist ein vielfaches der Nennkapazität einer Zelle. So kann eine Zelle mit 150 mAh und einem Nennstrom von 1C mit einem Maximalstrom von 150 mA belastet werden. Werden im mrShark zwei Zellen parallel betrieben, verteilt sich der Strom bei gleichermaßen geladenen Zellen gleichmäßig auf beide Zellen.

IV.7.4. Betriebsspannung

Werden für den mrShark LiPo-Akkumulatoren verwendet, haben diese typischerweise direkt nach einer Vollladung eine Ladeschlussspannung von 4,2V. Nach sehr kurzer Zeit fällt diese auf die Nennspannung von 3,7 V ab. Der Betrieb einer Zelle endet mit einer typischen Entladespannung von 3,0V.

Der mrShark verwendet am positiven Anschluss einer jeden Zelle zum Schutz vor Ausgleichsströmen eine Diode, über welcher eine Spannung von 0,2V abfällt. Die Betriebsspannung des mrShark variiert daraus folgernd, je nach Ladung der LiPo-Zellen, im Bereich von 2,8V - 4,0V. Die mittlere Betriebsspannung liegt daraus folgernd bei 3,4V.

IV.8. Energiebilanz

Der Energieverbrauch des mrShark berechnet sich aus dem Energiebedarf der einzelnen Bauteile des mrShark. Als Betriebsspannung wird für die folgenden Berechnungen ein Wert von 3,3V angenommen. Dieser ist ein typischer Wert für die Betriebsspannung digitaler ICs und entspricht in etwa der mittleren Betriebsspannung (siehe Betriebsspannung).

Deren Stromverbrauch ist in der Tabelle 4 aufgelistet. Er berechnet sich über die Summe der maximalen Stromaufnahmen aller Bauteile. Die Tabelle 5 zeigt den zu erwartenden typischen Strombedarf der Bauteile des mrShark und basiert auf der Annahme, dass nicht alle Bauteile kontinuierlich mit ihrer maximalen Leistung betrieben werden. So ist z.B. bei den RGB-LEDs davon auszugehen, dass im Mittel nur zwei von drei Farben eingeschaltet werden. Am Infrarot-Transciever lässt sich zudem der Transmitter ausschalten. Auch die Motoren werden im Mittel durch Standzeiten etwas weniger als ihren maximalen Stromverbrauch benötigen. Am Mikrocontroller lassen sich durch gezielte Abschaltung von Komponenten und das Aktivieren des Idle-Zustands Energieeinsparungen bewirken.

Der tatsächliche Energieverbrauch des mrShark wird jedoch maßgeblich von der Nutzung der Motoren, RGB-LEDs und dem Infrarot-Transciever beeinflusst. Wobei die Motoren 42% - 56%, die RGB-LEDs 24% - 33% und der Infrarot-Transciever < 1% - 30% der Energiebilanz ausmachen.

Bauteil	Anzahl	Supply-Current [mA]	Stromverbrauch [mA]
ATMEGA168V-10MU	1	4,0	4,0
PCA9306DC	1	?	?
TIR1000PWR	1	?	?
TFBS4711TR1	1	300,0	300,0
DRV8830DRCT	2	1,4	2,8
GM15 Gearmotor	2	210,0	420,0
LTC2990IMSPBF	2	1,8	3,6
ASMT-YTD2-0BB02	4	60,0	240,0
FXO-HC536R-1.8432	1	32,0	32,0
		GESAMT =	1002,4

Tabelle 4: Maximaler Stromverbrauch Bauteile mrShark bei 3,3V-5V

Bauteil	Anzahl	Supply-Current [mA]	Stromverbrauch [mA]
ATMEGA168V-10MU	1	3,000	3,000
PCA9306DC	1	?	?
TIR1000PWR	1	?	?
TFBS4711TR1	1	0,075	0, 075
DRV8830DRCT	2	1,400	2,800
GM15 Gearmotor	2	100,000	200,000
LTC2990IMSPBF	2	1,800	3,600
ASMT-YTD2-0BB02	4	30,000	120,000
FXO-HC536R-1.8432	1	30,000	30,000
		GESAMT =	359,475

Tabelle 5: Erwarteter typischer Stromverbrauch Bauteile mrShark bei 3,3V-5V

IV.9. Systemschnittstelle

Die Systemschnittstelle des mrShark führt einzelne Signale des mrShark auf eine Buchsenleiste aus, sodass über weitere Adapter oder Schaltungen auf diese zugegriffen werden kann.

Das Ziel der Systemschnittstelle ist es einerseits eine Programmierschnittstelle für das Aufspielen einer Firmware für den mrShark zu ermöglichen, andererseits eine Erweiterungsschnittstelle für Systemerweiterungen zu bieten. Daher beinhaltet die Systemschnittstelle Zugriffe auf die wichtigsten Schnittstellen des mrShark.

Die primäre Funktion der Systemschnittstelle als Programmierschnittstelle wird über die Ausführung des Serial Peripheral Interface (kurz SPI) des Mikrocontrollers ermöglicht. Mit Hilfe eines aufsteckbaren Adapters stehen 6- und 10-polige Stiftleisten für den Anschluss eines In-System-Programmers (kurz ISP) zur Verfügung. Über das SPI können zudem weitere elektronische Schaltkreise an den Mikrocontroller angeschlossen werden.

Als Erweiterungsschnittstellen sind neben der Betriebsspannung zwei Interrupts, sowie ein ADC-Eingang des Mikrocontrollers ausgeführt, über welche der Funktionsumfang des Mikrocontrollers erweitert werden kann. Darüber hinaus sind zwei Spannungs-Messe-Eingänge der Analog-Digital-Wandler auf die Systemschnittstelle ausgeführt. Als Kommunikationsschnittstellen enthält die Systemschnittstelle einen Anschluss an den I^2C-Bus welcher mit Hilfe des Mikrocontrollers mit den I^2C-Slaves auf dem mrShark verbunden werden kann. Darüber hinaus ist eine UART-Schnittstelle des Mikrocontrollers ausgeführt, über welche eine direkte Kommunikation mit dem Mikrocontroller ermöglicht werden oder mit Hilfe eines Adapters als Debugging-Schnittstelle dienen kann.

Um das Laden der Akku-Zellen des mrShark ohne Demontage des Gehäuses zu ermöglichen, beinhaltet die Systemschnittstelle zwei Kontakte für die positiven Anschlüsse der beiden Akku-Zellen. Ist der mrShark ausgeschaltet, werden diese über ein Slide-Switch auf die Systemschnittstelle umgelenkt.

IV.10. Bauteilparameter

Dieses Unterkapitel beschäftigt sich mit der Berechnung von Parametern externer Bauteile, welche nicht bereits in den dazugehörigen Kapiteln genannt worden sind.

IV.10.1. Pull-Up- / Pull-Down-Widerstände

Pull-Up- bzw. Pull-Down-Widerstände sollen Anpassungen von Signalpegeln zwischen verschiedenen Bausteinen des mrShark ermöglichen. Sie sorgen zudem für definierte Signalpegel, auch bei hochohmigen Ausgängen der digitalen Schaltkreise.

Ein Pull-Up- bzw. Pull-Down-Widerstand ruft in Abhängigkeit seines ohmschen Widerstandes einen Strom hervor. Ein entsprechender Widerstand ist demnach möglichst hochohmig zu wählen. In der Praxis haben sich Werte zwischen 100 Ω und 1MΩ bewährt. Der konkrete Wert hängt von der zu realisierenden Schaltung ab und ist bei niederfrequenten Schaltungen unkritisch. Um einen möglichst geringen Stromverlust durch Pull-Up- oder Pull-Down-Widerstände zu erhalten, besitzen diese auf dem mrShark einen Wert von 100 kΩ.. Nach dem Ohmschen Gesetzt (vergl. Gleichung (7)) ergibt sich bei einer Betriebsspannung von 3,3V ein Strom von 33 µA. Der mrShark verwendet insgesamt 13 100 kΩ Pull-Up- /bzw. Pull-Down-Widerstände, was zu einem Gesamtstrom von 0,429 mA führt. Für die I^2C-Schnittstelle werden, wie allgemein üblich, Pull-Up-Widerstände mit einem Wert von 1,8 kΩ eingesetzt.

IV.10.2. Entstörkondensatoren

Um kleinere Spannungsspitzen oder Störsignale von der Betriebsspannung zu filtern, wird ein Entstörkondensator eingesetzt, welcher für einen gewissen Frequenzbereich wie ein sehr niederohmiger Widerstand wirkt und dadurch Störsignale filtert. Hierfür verwendet der mrShark einen 100 nF Kondensator. Dessen Widerstand für Wechselspannungen, Blindwiderstand genannt, wird nach der Gleichung (Linder, Brauer, Lehmann, 2008) berechnet.

$$X_C = \frac{1}{2 * \Pi * f * C} \tag{18}$$

Der Verlauf des Blindwiderstandes X_C in Abhängigkeit der Frequenz f für eine Kapazität C = 100 nF ist in Abbildung 9 dargestellt und zeigt, dass für hohe Frequenzen über 2 MHz der Blindwiderstand des Kondensators kleiner als 1 Ω ist. Für Frequenzen über 100 MHz ist der Blindwiderstand nahezu 0 Ω.

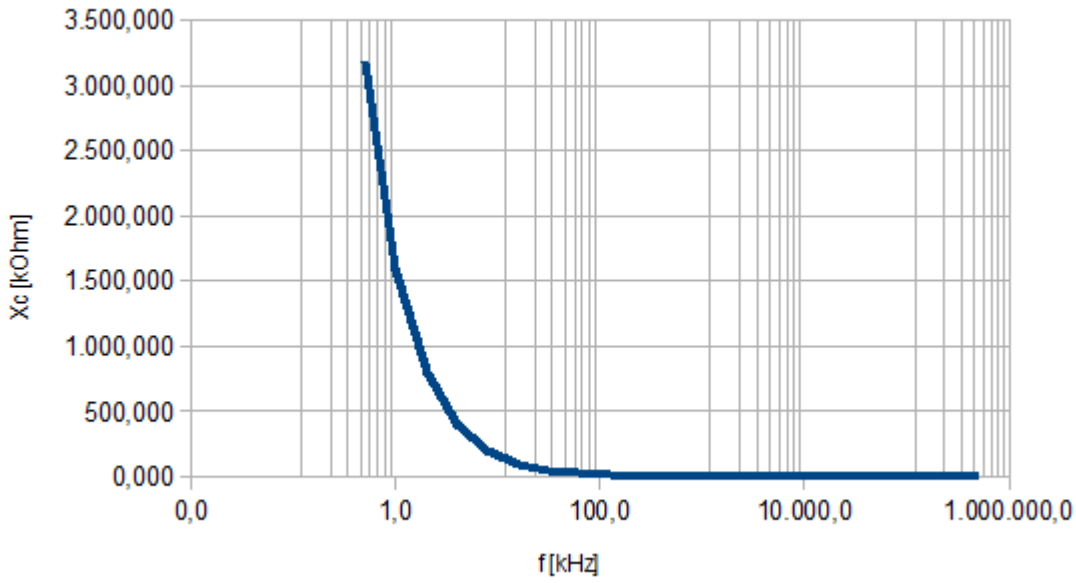


Abbildung 9: Blindwiderstand Xc über Frequenz f (Anhang A: Berechnungen)

IV.10.3. Taktversorgung

Zwei Bausteine des mrShark benötigen eine eigene Taktversorgung. Dies ist einerseits der Infrarot De-/Encoder, dessen Taktfrequenz bereits in Gleichung (12) zu 1,8432 MHz berechnet wurde. Andererseits benötigt der Mikrocontroller ebenfalls eine eigene Taktversorgung, welche von mehreren Parametern abhängt.

Der Mikrocontroller muss in der Lage sein, das serielle Datensignal des Infrarot De-/Encoders zu verarbeiten. Dazu benötigt er eine mindestens 16 mal höhere Taktrate. Für eine Baudrate von 115,2 kBit/s, was einer Frequenz von 115,2 kHz entspricht, benötigt der Mikrocontroller daher einen Systemtakt von mindestens $f_{min} = 16 * 115{,}2\,Hz = 1{,}8432\,MHz$. Dabei ist nicht jede Frequenz größer f_{min} geeignet. Dies resultiert daher, dass der Mikrocontroller seinen Systemtakt mit Hilfe eines ganzzahligen 16 Bit breiten Wertes für die interne Abtastrate des seriellen Signals herunterteilt.

Bei unpassenden Frequenzen entstehen so Abweichungen der Abtastrate. Geeignet sind daher folgende Frequenzen (Atmel, 2004, ATmega 48V/88V/168V datasheet):

- 1,8432 MHz
- 3,6864 MHz
- 7,3728 MHz
- 11,0592 MHz
- 14,7456 MHz
- 18,4320 MHz (bei min. 4,0 V)

Generell sollte ein Systemtakt gewählt werden, welcher möglichst niedrig ist, da der Mikrocontroller umso weniger Strom verbraucht, umso geringer der Systemtakt ist.

Für die Wahl eines Systemtakts muss jedoch auch die Realisierung der PWM-Signale für die RGB-LEDs beachtet werden (siehe RGB-LEDs). Diese werden im Mikrocontroller durch einen Timer realisiert, welcher in regelmäßigen Abständen einen Interrupt (siehe Interrupts) auslöst, welcher die PWM-Pegel der RGB-LEDs einstellt. Ein Interrupt ohne Programmcode benötigt 19 Systemtakte[5] zum aktivieren und anschließenden Verlassen des Interrupts (siehe Interrupts). Damit die PWM-Signale der RGB-LEDs für das menschliche Auge nicht mehr sichtbar sind, sollte die Frequenz der Grundfarben mindestens 100 Hz betragen. Die Frequenz von 100 Hz setzt sich aus der Anzahl *n* der RGB-LEDs sowie der Bit-Breite *b* der RGB-Werte (vergl. Gleichung (19)) zusammen.

$$\begin{aligned} f_{PWM} &= \frac{f_{TIMER}}{n * b} \\ \Rightarrow f_{TIMER} &= f_{PWM} * n * b \end{aligned} \tag{19}$$

Die Frequenz des Timers wird mit Hilfe eines Vorteilers *p* und einem Zählwert *u* vom Systemtakt f_{CPU} generiert (vergl. Gleichung (20)). Der Systemtakt wird dazu mit Hilfe des Vorteilers *p* geteilt. Der Timer zählt jetzt mit Hilfe des neuen Taktes bis zum Zählwert und löst anschließend den Interrupt aus. Dies wiederholt sich fortlaufend.

5 Errechnet aus dem Assembler-Code einer leeren Interrupt-Service-Routine unter Beachtung der Taktzeiten einzelner Befehle (Atmel, 2004, ATmega 48V/88V/168V datasheet).

$$f_{TIMER} = \frac{f_{CPU}}{p * u} \tag{20}$$

Dem Mikrocontroller stehen dabei je nach Bit-Breite des Zählwertes unterschiedliche Vorteiler zur Verfügung wie Tabelle 6 zeigt..

Bit-Breite u	**Mögliche Vorteiler p**
8 Bit	1, 8, 32, 64, 128, 256, 1024
16 Bit	1, 8, 64, 256, 1024

Tabelle 6: Vorteiler p in Abhängigkeit der Bit-Breite des Zählwertes u (Atmel, 2004, ATmega 48V/88V/168V datasheet)

Optimal ist die Wahl eines hohen Vorteilers und des maximalen Zählwertes. Dies würde zu einer maximalen Rechenzeit zwischen den Interrupts führen, benötigt aber einen höheren Systemtakt. Die Wahl eines optimalen Verhältnisses zwischen Vorteiler, Zählwert und Systemtakt ist demnach entscheidend für die Effizienz des Systems in Bezug auf die zwischen den Interrupts verfügbare Rechenzeit und den Stromverbrauch. Hierfür wird zuerst die Frequenz für das Auslösen eines Interrupts f_{INTMIN} benötigt. Diese berechnet sich nach Gleichung (21) mit Hilfe Anzahl n der RGB-LEDs, der Bit-Breite b der RGB-Werte und der Frequenz für das PWM-Signal f_{PWM}.

$$f_{INTMIN} = n * b * f_{PWM} \tag{21}$$

Für n = 4, b = 256 unf f_{PWM} = 100 Hz ergibt sich für $f_{INTMIN.}$ ein Wert von 102,4 kHz. Ein Interrupt für die PWM-Steuerung müsste also mit dieser Frequenz aufgerufen werden.

Die Frequenz für einen Interrupt hängt von den Parameter Vorteiler *p* und Zählwert *u* ab und berechnet sich Gleichung (22).

$$\begin{aligned} f_{INT.} &= \frac{f_{CPU}}{p * u} \\ \Rightarrow u &= \frac{f_{CPU}}{p * f_{INT.}} \end{aligned} \tag{22}$$

Setzt man in Gleichung (22) für $f_{INT.}$ Den Wert von f_{INTMIN} = 102,4 kHz ein, erhält man einen Zählwert in Abhängigkeit des Systemtaktes und des Vorteilers. Zulässige Werte für u sind ausschließlich positive ganze Zahlen, deren Intervall durch die Bit-Breite von u begrenzt ist.

In Tabelle 7 sind Werte für den Zählwert u in Abhängigkeit des Systemtaktes f_{CPU} und dem Vorteiler p angegeben. Besonders wichtig ist die Zeile für den Vorteiler 1. Da hier kein Vorteiler verwendet wird, gibt u die Anzahl an Systemtakten an, nach denen immer ein Interrupt aufgerufen werden müsste. Zu sehen ist, dass bei 1,8432 MHz der Interrupt häufiger aufgerufen werden müsste als der Interrupt ohne Programmcode an Ausführungszeit (19 Takte) benötigt.

p / f_{CPU}	**1,8432 MHz**	**3,6864 MHz**	**11,0592 MHz**	**14,7456 MHz**	**18,4320 MHz**
1	18	36	108	144	180
8	2	5	14	18	22
32	0	1	3	4	5
64	-	0	1	2	2
128	-	-	0	1	1
256	-	-	-	0	0
1024	-	-	-	-	-

Tabelle 7: Zählwert in Abhängigkeit von Vorteiler und Systemtakt

Aus der Tabelle 7 ist ein Systemtakt zu wählen, für den ein möglichst hoher Zählwert bei einem gewählten Vorteiler verfügbar ist. In eine Entscheidung für einen Systemtakt sollte auch der Stromverbrauch mit einbezogen werden, dessen Abhängigkeit von dem Systemtakt in Abbildung 10 dargestellt ist.

Unter Berücksichtigung dieser beiden Faktoren, hohe Taktrate und geringer Stromverbrauch, ist ausgehend von den vorliegenden Daten ein Systemtakt von unter 15 MHz und über 10 MHz zu wählen.

Der mrShark verwendet einen Systemtakt von 11,0592 MHz für den Mikrocontroller, da dieser unter Betrachtung aller vorhergehenden Aspekte die minimal mögliche Frequenz und den geringsten Stromverbrauch aufweist.

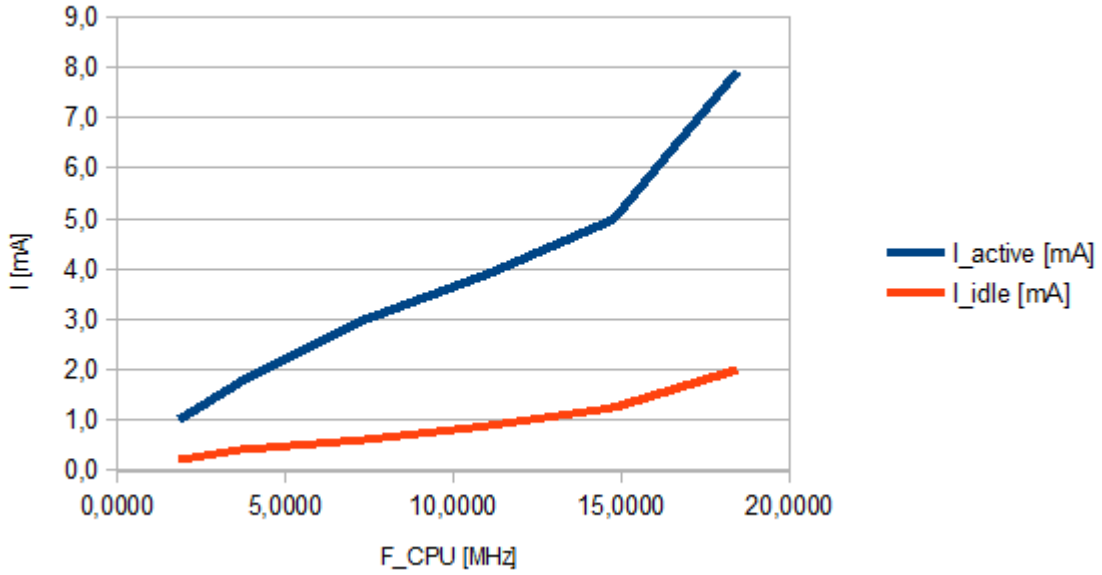


Abbildung 10: ATmega 48V/88V/168V Supply Current vs. Frequency für 3,3V (über 15 MHz für 4,0V) (Anhang A: Berechnungen)

IV.10.4. Transistoren

Da die digitalen Ausgänge des Mikrocontrollers keine hohen Ströme liefern können (Atmel, 2004, ATmega 48V/88V/168V datasheet), werden die RGB-LEDs über Transistoren geschaltet, welche den Signalpegel der RGB-LEDs auf logisch aktiv (HIGH) oder inaktiv (LOW) legen. Dazu werden die Transistoren vom Typ BC817W wie elektrische Schalter betrieben und schalten entweder komplett durch oder nicht. Der BC817W eignet sich insbesondere auf Grund seiner großen Verfügbarkeit, kleinen Bauform, seines günstigen Preises und einem geeigneten maximalen Stromdurchfluss (NXP, 2009, BC817W datasheet).

Der BC817W wird zum Schalten der PWM-Signale für die RGB-LEDs im Sättigungsbereich betrieben, sodass der Transistor die Kollektor-Emitter-Strecke durchschaltet. Dazu muss ein passender Basiswiderstand berechnet werden (Schwarz, 2013, Basiswiderstand). Für die Berechnung des Basiswiderstandes R_B wird die Stromverstärkung h_{FESAT} des Transistors im Sättigungsbereich benötigt. Diese ist aus dem Datenblatt nicht direkt ersichtlich und mit Hilfe der Stromverstärkung außerhalb des Sättigungsbereichs abzuschätzen (Schwarz, 2013,

Basiswiderstand). Dazu ist die minimale Stromverstärkung h_{FE} abzulesen und nach Gleichung (23) durch 2 und 10 zu teilen. Der Mittelwert dieser beiden Werte ist dann ein Schätzwert für die Stromverstärkung im Sättigungsbereich des Transistors.

$$h_{FESAT} = \frac{h_{FE}}{10} + \frac{\frac{h_{FE}}{2} - \frac{h_{FE}}{10}}{2} \tag{23}$$

Des weiteren wird der Strom des Verbrauchers I_c benötigt, welcher geteilt durch die Stromverstärkung h_{FESAT} den benötigten Basisstrom I_b ergibt (vergl. Gleichung).

$$I_b = \frac{I_c}{h_{FESAT}} \tag{24}$$

Theoretisch müsste zu I_b noch der Strom hinzuaddiert werden, welcher durch den verwendeten Pull-Down-Widerstand fließt. Dieser Strom ist jedoch bei einem Pull-Up-Widerstand von 100 kΩ sehr gering (siehe Pull-Up- / Pull-Down-Widerstände und Anhang Berechnung Basiswiderstand) und kann dementsprechend vernachlässigt werden.

Mit Hilfe des Basisstroms I_{bnz} lässt sich der benötigte Basiswiderstand auf Basis des Ohmschen Gesetz nach Gleichung berechnen.

$$R_b = \frac{U_E - U_{BE}}{I_b} = \frac{(U_E - U_{BE}) * h_{FESAT}}{I_c} \tag{25}$$

U_E ist hierbei die Eingangsspannung an der Basis des Transistors und U_{BE} die über der Basis-Emitter-Strecke abfallende Spannung.

Für die Basiswiderstände fließt für jede Farbe durch jede LED ein maximaler Strom von 20 mA (siehe RGB-LEDs). Dies ergibt einen Gesamtstrom I_c von 80 mA. Die Betriebsspannung und damit die Steuerspannung U_E variiert je nach Spannung der Akku-Zellen im Bereich von 3,9 V bis 2,8 V (siehe Betriebsspannung). Die Spannung U_{BE} beträgt beim BC817W 0,7 V (NXP, 2009, BC817W datasheet).

Für h_{FE} ergibt sich nach Gleichung (23) beim BC817W ein Wert von 30 (NXP, 2009, BC817W datasheet) und damit nach Gleichung (24) und (25) ein Vorwiderstand von maximal 1,2 kΩ (siehe Anhang Berechnung Basiswiderstand) pro Transistor zur Farbsteuerung. Der mrShark verwendet entsprechend der Empfehlungen (Schwarz, 2013, Basiswiderstand) einen etwas geringeren Basiswiederstand von 1,0 kΩ.

Für die Steuerung einer RGB-LED beträgt der Strom I_c 60 mA, pro LED drei Farben mit je 20 mA. Daraus ergibt sich ein maximaler Vorwiderstand von 1,6 kΩ (siehe Anhang Berechnung Basiswiderstand). Auch hier verwendet der mrShark einen Wert von 1,0 kΩ, sodass für die Steuerung der vier RGB-LEDs kein zusätzliches Bauteil verwendet werden muss, als für die Steuerung der LED-Farben.

IV.10.5. Vorwiderstände

Die RGB-LEDs haben, je nach Farbe eine unterschiedliche Durchlassspannung. Wird die maximale Durchlassspannung einer LED-Farbe überschritten, führt dies zur Zerstörung der LED. Deshalb sind vor den LED-Farben Vorwiderstände anzubringen, welche die LEDs vor zu hohen Durchlassspannungen schützen. Der Vorwiderstand berechnet sich nach dem Ohmschen Gesetz entsprechend der Gleichung (26) mit Hilfe der Betriebsspannung U_{VCC} der zulässigen Durchlassspannung U_f und dem Durchlassstrom I_f.

$$R_{vor} = \frac{U_{VCC} - U_f}{I_f} \tag{26}$$

Der Wertebereich des Vorwiderstandes wird beim mrShark durch die maximale Betriebsspannung U_{VCC} = 3,9V, den Durchlassstrom der LEDs von I_f = 20 mA bestimmt, sowie die typische und maximale Durchlassspannung U_f (Avago, 2010, ASMT-YTD2-0BB02 datasheet) und ist in Tabelle 8 dargestellt.

U_f / Farbe	**Rot [Ω]**	**Grün [Ω]**	**Blau [Ω]**
typ.	95	40	40
max.	75	10	10

Tabelle 8: Vorwiderstand LED-Farben (siehe Anhang Berechnung Basiswiderstand)

Theoretisch wären Vorwiderstände von 82 Ω für die Farbe Rot und Vorwiderstände von 33 Ω (entsprechend der Mittelwerte und verfügbarer Widerstandswerte) für die Farben Grün und Blau zu wählen.

Für alle Status-LEDs des mrShark werden Vorwiderstände von 130 Ω verwendet, da die genaue Leuchtkraft dieser LEDs nicht optisch relevant ist und dieser Wert für alle gängigen handelsüblichen einfarbigen LEDs geeignet ist.

IV.11. Firmware

Die Firmware ist die zentrale Software des Mikrocontrollers, welche die Funktionen des mrShark steuert. Sie ist in der Programmiersprache C geschrieben und wird mit Hilfe eines speziellen Compilers in für den Mikrocontroller kompatiblen Hexadezimal-Code umgewandelt. Dieser wird mit Hilfe eines Programmieradapters in den Flash-Speicher des Mikrocontrollers geschrieben.

Die Firmware bindet speziell für den eingesetzten Mikrocontroller zugeschnittene Bibliotheken ein, welchen den einfachen Zugriff auf Register des Mikrocontrollers erlauben. Auf die Verwendung dieser Zugriffe im Allgemeinen soll im Rahmen dieser Arbeit nicht eingegangen werden. Weitere Informationen zur prinzipiellen Arbeitsweise mit AVR Mikrocontrollern finden sich z.B. auf der Internetseite www.mikrocontroller.net.

IV.11.1. Modulübersicht

Die Firmware des mrShark besteht aus mehreren einzelnen Komponenten oder Modulen (siehe Abbildung 11). Die einzelnen Module sind alle mit der Main Function[6] verbunden, welche u.a. für die Initialisierung der Module zuständig ist (siehe Hauptroutine).

6 Englisch für: Hauptroutine

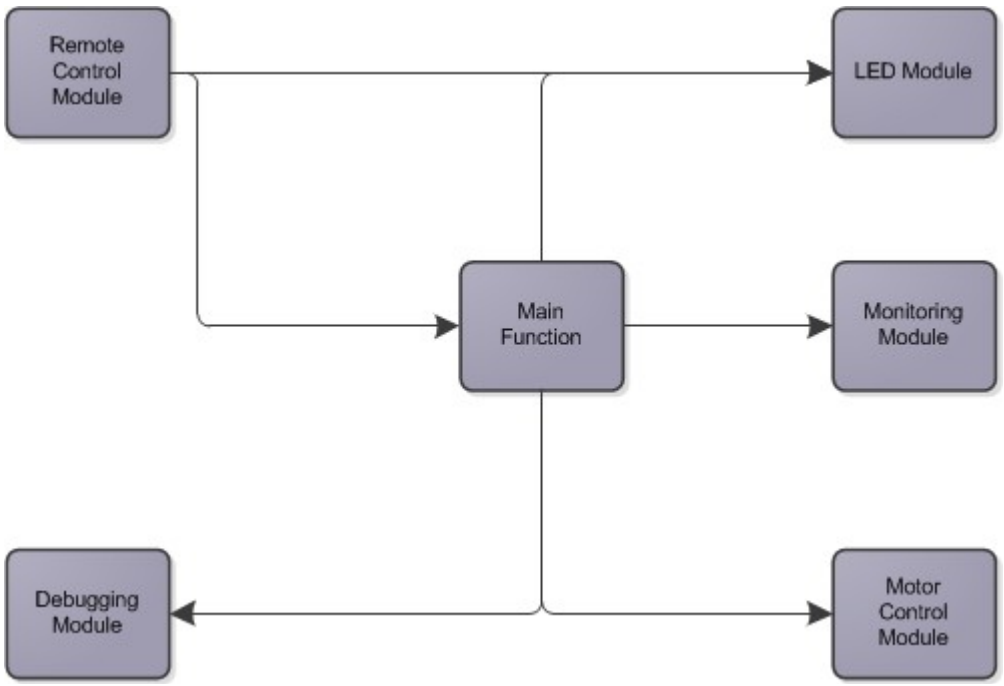


Abbildung 11: Module mrShark Firmware

Über das Remote Control Modul werden Daten der Infrarot-Schnittstelle verarbeitet. Das Modul steuert direkt die LED Funktionen des LED Moduls an und kann die Main Function über Änderungen der Steuerbefehle für das Motor Control Modul informieren. Das Motor Control Modul regelt die Motortreiber des mrShark. Über das Debugging Modul können über die Systemschnittstelle Daten des mrShark abgefragt und manipuliert werden. Zum Abfragen von Strom- und Spannungswerten, sowie der Systemtemperatur kann das Monitoring Modul genutzt werden.

IV.11.2. Hauptroutine

Die Hauptroutine oder Main Function, besteht aus einer Initialisierungsphase und einer Hauptschleife. In der Initialisierungsphase werden direkt nach Programmstart alle benötigten Funktionen des mrShark aktiviert und die dazugehörigen Einstellung gesetzt. In dieser Phase wird auch ein kurzer Motortest durchgeführt, sodass eine simple Funktionsprüfung des mrShark nach dem Einschalten möglich ist.

Die Initialisierung des mrShark wird in der Funktion *sys_init* in der Quellcodedatei *sys.c* geregelt:

```
void sys_init(void){
	// initiate power saving
	sys_power_init();

	// read robot id
	sys_robotID = sys_ee_read_robotID();

	// call libraries and modules init functions
	i2c_init();
	led_init();

	led_on(LED_STATUS);

	motor_init();
	control_init();

	#if !defined(CFG_CODE_LEVEL_AVG) && !defined(CFG_CODE_LEVEL_MIN)
	monitor_init();
	#endif

	#ifndef CFG_CODE_LEVEL_MIN
	debug_init();
	#endif

	sei();

	// send system information
	#ifndef CFG_CODE_LEVEL_MIN
	debug_send_system_info(SYS_NAME, SYS_VERSION, SYS_PUBLISHER);
	#endif

	// set led color green
	led_off(LED_STATUS);
}
```

Die Funktion ruft eine Initialisierungsfunktion zum Energiesparen auf (siehe Energiesparoptionen) und lädt anschließend die ID des Roboters aus dem EEPROM. Dieses speichert die ID auch noch nach dem Ausschalten der Versorgungsspannung.

Anschließend werden Initialisierungsfunktionen der benötigten Bibliotheken und Module aufgerufen. Zuerst wird die Bibliothek für die I2C-Schnittstelle (Fleury, 2006, I2C Master Interface) initialisiert., anschließend das LED Modul, das Motor Control Modul (hier erfolgt auch der Motortest), das Remote Control Modul, das Monitoring Modul und das Debugging Modul.Mit dem Befehl *sei()* werden alle Interrupts (siehe Interrupts) des mrShark aktiviert. Nach der Initialisierung des LED Moduls wird die Status-LED eingeschaltet.

Abschließend werden Firmwareinformationen über das Debugging Modul ausgegeben und anschließend die Status LED ausgeschaltet, wodurch das Ende der Initialisierungsphase angezeigt wird. Im Anschluss an die Initialisierung wird eine Endlosschleife in der Hauptroutine ausgeführt:

```
// main loop
while(1){

	// suspend system
	sys_sleep();

	// check for changed motor values
	if( ctrl_flag_motorL ){
		motor_set_speed(
				MOTOR_ADDR_L,
				control_getMotorSpeed(MOTOR_ADDR_L),
				control_getMotorCommand(MOTOR_ADDR_L));
	}

	if( ctrl_flag_motorR ){
		motor_set_speed(
				MOTOR_ADDR_R,
				control_getMotorSpeed(MOTOR_ADDR_R),
				control_getMotorCommand(MOTOR_ADDR_R));
	}

	// check for changed robot id
	if( ctrl_flag_id )
		sys_ee_set_robotID( control_getRobotID() );

	// process debugging module
	#ifndef CFG_CODE_LEVEL_MIN
	debug_process();
	#endif

}
```

Die Endlosschleife ist notwendig, da der Mikrocontroller seine Programmausführung niemals unterbrechen soll. Sie ruft zuerst eine Funktion zum Energiesparen auf (siehe Energiesparoptionen) und überprüft anschließend, ob neue Steuerbefehle des Remote Control Moduls vorliegen. Sind neue Steuerbefehle eingegangen, werden die entsprechenden Daten an die weiteren Module wie z.B. das Motor Control Modul weitergeleitet. Zuletzt wird die Funktion *debug_process* des Debugging Moduls aufgerufen.

IV.11.3. Interrupts

Ein Interrupt ist ein während des regulären Programmablaufs ausgelöstes Ereignis, welches den Mikrocontroller dazu veranlasst, die Programmabarbeitung sofort zu unterbrechen und eine für den Interrupt fest definierte Funktion aufzurufen, die sogenannte Interrupt Service Routine (kurz ISR). Nach Abarbeitung der ISR springt der Mikrocontroller zurück zu der Stelle im Programmcode, an der er sich vor dem Aufruf der ISR befunden hat. Durch Interrupts kann sofort auf zeitkritische Ereignisse reagiert und diese vorrangig bearbeitet werden (Wüst, 2003).

Die Firmware des mrShark verwendet Interrupts in verschiedenen Modulen, um zeitkritische Aufgaben zu bearbeiten. Eingesetzt werden u.a. Interrupts für das Empfangen von Steuerdaten oder eine in Software implementierte Serielle Schnittstelle.

IV.11.4. Infrarot-Steuerprotokoll

Bei der Verwendung eines Steuerprotokolls zum Senden von Steuerbefehlen via Infrarot-Schnittstelle an den Mixed-Reality-Roboter wurde bewusst ein neues Steuerprotokoll für den mrShark eingeführt. Das auf den alten Robotern eingesetzte Steuerprotokoll basierte auf einer Zeichenkette von Befehlen und Steuerzeichen. Da in der Datenübertragung via Infrarot z.B. durch Streulicht immer wieder Fehler auftreten, verwendet das alte Protokoll die Angabe der Länge der Zeichenkette sowie eine Prüfsumme. Über die Prüfsumme kann sichergestellt werden, dass die Daten korrekt übertragen werden. Sollte die errechnete Prüfsumme der Zeichenkette nicht mit der übertragenen Prüfsumme übereinstimmen, können die empfangenen Daten verworfen werden.

Das Steuerprotokoll des mrShark verwendet für die Befehls- und Steuerdaten keine ASCII-Zeichen, sondern nutzt den kompletten Zahlenraum von 8-Bit Zahlen aus. Zu beachten ist, dass bei der Übertragung von Daten via Infrarot oft ein Bit einer 8-Bit breiten Zahl fehlerhaft empfangen wird. Ist dies der Fall, muss ausgeschlossen werden, dass das fehlerhafte, um ein Bit veränderte Datum einem anderen gültigen Befehl des Steuerprotokolls entspricht.

Befehle des Steuerprotokolls dürfen also keinen neuen Befehl ergeben, wenn ein Bit verändert wird. Um die dieser Einschränkung entsprechenden maximal 8-Bit breiten Zahlen zu ermitteln, kann das nachfolgende Python-Skript verwendet werden:

```
'''
CONFIG
'''
csv_file = "programm-commands.csv" # filepath to store commands as csv
file
write_to_file_enabled= True  # True if to store commands in csv file
log_enabled = True           # True to print log data

valid = []
blocked = []

def log(*args):
     txt = ""
     if log_enabled:
           for arg in args:
                 txt += str(arg) + " "
           print txt

# generate command number
for cmd in range (256):
     log( "checking", cmd)

     # check if command is free
     if cmd not in blocked:

           log( "\tnot blocked" )

           # create corresponding blocked commands
           # by toggling one bit
           err = False
           lst = []
           for x in range(8):
                 newcmd = cmd ^ (1<<x)
                 log( "\tchecking", newcmd )
                 lst.append( newcmd )
                 if newcmd in valid:
                       log( "\tcorresponding command", x, "in valid
list" )
                       err = True

           # update lists
           if not err:
                 log( "\tno error. valid:", cmd, "blocked:", lst )
                 valid.append(cmd)
                 for x in lst:
                       if x not in blocked:
                             blocked.append( x )

log( "VALID", len(valid) )
log( "BLOCKED", len(blocked) )

# write data to file, if enabled
if write_to_file_enabled:
     f = open(csv_file, 'w')
     f.write( "command\tchar\n" )

     for cmd in valid:
           line = str(cmd) + "\t"
           if cmd > 31 and cmd < 127:
```

```
                line += str( chr(cmd) )
            f.write( line + "\n" )

    f.close()
```

Das Skript erstellt eine Liste von gültigen Befehlen und schreibt diese in eine Datei. Daraus ergeben sich 128 zulässige Zahlen. Die Zuordnung der Steuerbefehle des mrShark zu den zulässigen Zahlen kann dem dem Protocol Version 2.0 (siehe Anhang C: Steuerbefehle) entnommen werden.

Die Datenübermittlung nach Protokoll Version 2.0 beginnt mit dem Befehl *PROTOCOL VERSION* gefolgt von der Protokoll Versionsnummer, in diesem Fall *2*. Nachfolgend können einzelne Sektionen von Befehlen übertragen werden. Jede Sektion wird dabei einem Roboter mit einer eindeutigen Identifikationsnummer, genannt Robot-ID, zugewiesen. Eine Sektion beginnt mit dem Steuerbefehl *SECTION SEPERATOR* gefolgt vom Befehl *ROBOT ID* und der ID des Roboters, für den die Sektion bestimmt ist.

In einer Sektion kann nach Nennung der Robot-ID eine beliebige Anzahl an Befehlen folgen. Einzelne Befehle werden dabei durch den Steuerbefehl *SEPERATOR* getrennt. Ein Befehl besteht immer aus zwei Teilen, dem eigentlichen Befehl und einem Wert. Ist für einen Befehl kein Wert vorgesehen, kann ein beliebiger Wert übertragen werden. Am Ende einer Übertragung erfolgt durch *SECTION SEPERATOR* getrennt der Steuerbefehl *TRANSISSION END* gefolgt von einem beliebigen Wert.

Ein Beispiel für eine Übertragung könnte demnach wie folgt aussehen. Die Befehle sind mit ihren hexadezimalen Zahlenwerten angegeben.

```
0x50 0x02 0x2B 0x5A 0x00 0x2E 0x41 0x30 0x2B 0x59 0x00
```

Dies entspräche einer Übertragung nach Protokoll Version 2.0 mit einer Sektion an den Roboter mit der ID 0. Übertragen wird der Befehl *LEFT FWD* mit dem Wert 0x30, was einer Vorwärtsrotation des linken Motors/Rades mit einer Geschwindigkeit von 30% entsprechen würde.

IV.11.5. Energiesparoptionen

Um Energie zu sparen, können über die Funktion *sys_power_init* in der Quellcodedatei *sys.c* einzelne Pheripherie-Module des Mikrocontrollers, wie z.B. Schnittstellen, ausgeschaltet werden um den Stromverbrauch zu senken. Dazu kann in der Datei *config.h* definiert werden, welche Module abgeschaltet werden sollen:

```
// -------- ENEGRY CONFIGURATION --------
// Set all componets to TRUE that aren't used
#define SYS_POWER_DOWN_I2C            FALSE
#define SYS_POWER_DOWN_TIMER0         FALSE
#define SYS_POWER_DOWN_TIMER1         FALSE
#define SYS_POWER_DOWN_TIMER2         TRUE
#define SYS_POWER_DOWN_SPI            TRUE
#define SYS_POWER_DOWN_UART           FALSE
#define SYS_POWER_DOWN_ADC            TRUE
```

Die Firmware des mrShark nutzt den internen Analog-Digital-Wandler, den Timer 2 und die SPI-Schnittstelle nicht und diese werden deshalb abgeschaltet.

Die Funktion *sys_sleep* versetzt den Mikrocontroller in den Idle-Zustand und wird bei jedem Durchlauf von der Hauptroutine aufgerufen. Im Idle-Zustand wird die reguläre Programmausführung unterbrochen und nur die Pheripherie-Module wie Timer, Schnittstellen etc. laufen weiter. Ein von diesen Modulen ausgelöster Interrupt veranlasst den Mikrocontroller dazu die Abarbeitung des Programms fortzusetzen. Im Idle-Zustand wird der Stromverbrauch erheblich gesenkt (siehe Abbildung 3.

IV.11.6. Motor Control Modul

Die Motorsteuerung erfolgt über das Setzen einer Motorspannung in einem Register des Motortreibers (TI, 2012, DRV8830 datasheet). Die Adressen und zugehörigen Register der Motortreiber sind in der Datei *motor.h* hinterlegt:

```
// -------- DEFINES --------
#define MOTOR_ADDR_L            0xC0
#define MOTOR_ADDR_R            0xC4

#define MOTOR_REG_CONTROL       0x00
```

Die Geschwindigkeit eines Motors kann über die Funktion motor_set_speed in die Register der Motortreiber geschrieben werden:

```
void motor_set_speed(uint8_t address, uint16_t speed, uint8_t direction){
     speed = (speed * 100) / 0xff;
     speed  = speed * (MOTOR_SPEED_MAX - MOTOR_SPEED_MIN);
     speed = (speed / 100) + MOTOR_SPEED_MIN;
     i2c_writeData( address, MOTOR_REG_CONTROL, (speed<<2)|direction );
}
```

Die Funktion wandelt den Geschwindigkeitswert zuerst in ein gültiges Intervall um, da der Motortreiber nur ein begrenztes Intervall an Werten für die Motorspannung zulässt. (TI, 2012, DRV8830 datasheet). Die Funktion *i2c_writeData* erweitert die verwendete I^2C-Bibliothek (Fleury, 2006, I2C Master Interface) um eine übergeordnete Funktion zum Senden von Daten.

Das Motor Control Modul ermöglicht über weitere Funktionen auch das Auslesen der Drehrichtung und der Geschwindigkeit eines Motors:

```
void motor_set_speed(uint8_t address, uint16_t speed, uint8_t direction);
void motor_brake(uint8_t address);
void motor_all_brake(void);

uint8_t motor_get_direction(uint8_t address);
uint8_t motor_get_speed(uint8_t address);
```

Um eine Beschädigung der Motoren zu vermeiden, schaltet der Motortreiber die Motoren z.B. beim Überschreiten eines durch einen externen Widerstand festgelegten Stroms ab (siehe Motortreiber). Das Auftreten eines Fehler speichert der Motortreiber intern in einem Register, welches über die folgenden Funktionen abgefragt und zurückgesetzt werden kann:

```
uint8_t motor_get_fault(uint8_t address);
void motor_clear_fault(uint8_t address);
```

Mögliche Fehler sind ein zu hoher Stromverbrauch des Motors, das Unterschreiten der minimalen Betriebsspannung, Überschreitung der maximalen Gesamtstromaufnahme des Motortreibers oder Überschreiten der Betriebstemperatur.

Die Funktion motor_test in der Quellcodedatei motor.c kann zum Ausführen eines Motortests über das Debugging Modul genutzt werden. Dazu werden die Fehlerspeicher der beiden Motortreiber gelöscht, die Motoren sowohl links, als auch rechts herum gedreht und

anschließend die Werte der Fehlerspeicher erneut eingelesen und zurückgegeben. Dieser Motortest wird auch bei der Initialisierung des Motor Control Moduls ausgeführt, jedoch ohne Analyse der Fehlerwerte.

IV.11.7. Remote Control Modul

Das Remote Control Modul ist für die Verarbeitung der Steuersignale der Infrarot-Schnittstelle zuständig (siehe Debugging Modul). Es verwendet dazu die eingebaute serielle Schnittstelle des Mikrocontrollers. Diese wird in der Initialisierungsfunktion des Moduls eingerichtet. Des weiteren wird der Interrupt für das Empfangen eins Zeichens aktiviert:

```
// initiate uart
uart_init();

// enable uart interrupt
UCSR0B |= (1<<RXCIE0);
```

Die ISR *ISR(USART_RX_vect)* wird anschließend immer aufgerufen, wenn ein neues Zeichen über die serielle Schnittstelle empfangen wurde. Um die empfangenen Steuerbefehle zu interpretieren, ändert die ISR je nach empfangenen Daten den Zustand des Remote Control Moduls (siehe Anhang Protocol States).

Wird ein Steuerbefehl für die Status-LED oder RGB-LEDs empfangen, wird dieser sofort durch einen Aufruf der entsprechenden Funktion des LED Moduls ausgeführt. Alle anderen Steuerbefehle sowie deren Werte werden zwischengespeichert, da ihre Ausführungszeit länger dauert. Als Beispiel sind insbesondere die Steuerbefehle für die Motoren zu nennen. Da diese über den I^2C-Bus übertragen werden müssen, würde ihre Ausführung zu viel Zeit in der ISR beanspruchen.

Durch ein Flag[7] zeigt das Remote Control Modul an, dass ein neuer Befehl eingetroffen ist. Die Hauptroutine kann anschließend darauf reagieren und die entsprechenden Daten abrufen und weiterverarbeiten, um z.B. die Steuerbefehle für die Motoren auszuführen. Dazu stellt das Modul Funktionen zum Zugriff auf die Steuerbefehle und -daten bereit:

```
uint8_t control_getMotorCommand(uint8_t motor);
uint8_t control_getMotorSpeed(uint8_t motor);
uint8_t control_getRobotID();
```

7 Englisch für: Fahne / Wimpel

IV.11.8. Debugging Modul

Das Debugging Modul stellt ein Interface zur Fehlersuche und zur Abfrage von Systemdaten bereit. Dazu implementiert es eine serielle Schnittstelle, welche auf die Systemschnittstelle ausgeführt ist. So lässt sich der mrShark z.B. über ein serielles Datenkabel mit einem PC verbinden. Da die interne serielle Schnittstelle des Mikrocontrollers bereits vom Remote Control Modul verwendet wird, implementiert das Debugging Modul eine entsprechende Schnittstelle mit Hilfe eines Timers.

Sowohl die im Remote Control Modul, als auch die in diesem Modul implementierte serielle Schnittstelle empfängt asynchrone Daten. Die Daten werden über eine einzelne Leitung vom Sender zum Empfänger übertragen. Da keine Leitung für den Basistakt der Datenübertragung vorliegt, müssen Empfänger und Sender vor der Inbetriebnahme der Schnittstelle auf eine einheitliche Datenrate, auch Baudrate genannt, sowie ein einheitliches Datenformat eingestellt werden. Ist ein bidirektionaler Betrieb vorgesehen, sind zwei separate Datenleitungen notwendig.

Für die Datenübertragung ist es üblich, dass die Datenleitung im inaktiven Zustand einen logischen HIGH-Pegel führt. Durch ein Absenken des Signal auf einen LOW-Pegel für einen Takt wird der Beginn einer Übertragung gekennzeichnet. Anschließend wird, beginnend mit dem niederwertigsten Bit, eine fest definierte Anzahl von Bits mit einem logischen HIGH-Pegel für eine digitale 1 und einem logischen LOW-Pegel für eine digitale 0 übertragen. Auf Übertragung der Daten kann noch ein sogenanntes Parity-Bit[8] folgen, welches angibt, ob der binäre Wert der übertragenen Daten eine gerade oder ungerade Zahl repräsentiert. Die Übertragung endet mit mindestens einem Stop-Bit, welches durch einen HIGH-Pegel die Datenübertragung beendet (Beierlein & Hagenbruch, 2011). Der mrShark verwendet für die serielle Schnittstelle des Debugging Moduls acht Datenbits, kein Parity-Bit und ein Stop-Bit (Kurzschreibweise: 8N1).

Eine in der Software implementierte serielle Schnittstelle muss die Signal-Pegel auf der Empfangs-Datenleitung der Schnittstelle analysieren, eine beginnende Datenübertragung erkennen und die einzelnen übertragenen Bits zu einem binären Wert zusammensetzen.

8 Englisch für: Paritäts-Bit

Dazu tastet der Mikrocontroller mit Hilfe eines Timers das Signal der Empfangs-Datenleitung an einem seiner digitalen IOs ab. Damit eine sichere Aussage über den Signal-Pegel getroffen werden kann, verwendet der Mikrocontroller ein Oversampling[9]. Dazu wird das Signal mit einem vielfachen der eigentlichen Taktrate der Schnittstelle abgetastet (siehe auch Funktionstest). Dadurch werden Zeitverschiebungen durch ungenaue Systemtakte des Empfängers oder Senders kompensiert (Jetzek, 2013).

Das Debugging Modul verwendet dazu die suart Bibliothek mit den Quellcodedateien *suart.h* und *suart.c*. Die Initialisierungsfunktion *suart_init* startet den Timer 1 des Mikrocontrollers:

```
// Enable timer
TCCR1B |= (1<<WGM12) | (1<<CS10);     // ctc mode, no prescaler
OCR1AH = SUART_COMP_VAL >> 8;         // set compare match value for baudrate
OCR1AL = SUART_COMP_VAL & 0xFF;
TIMSK1 |= (1<<OCIE1A);                      // enable compare match interrupt
```

Die benötigten Werte werden in der Quellcodedatei *uart.h* berechnet:

```
#define SUART_BAUD              300UL        // Baudrate

#define SUART_THRESHOLD         50                  // Threshold for rx level in %
#define SUART_OVERSAMPLING      8                   // oversampling factor


// Calculation
#define SUART_COMP_VAL   (F_CPU / (SUART_OVERSAMPLING * SUART_BAUD))
            // compare match value rounded
#define SUART_BAUD_REAL ((F_CPU * 100) / (SUART_OVERSAMPLING * SUART_COMP_VAL))
            // real baudrate * 100
#define SUART_BAUD_ERR   (SUART_BAUD / 100)
            // 1% of baudrate

#if ( (SUART_BAUD_REAL + SUART_BAUD_ERR) < SUART_BAUD_REAL || (SUART_BAUD_REAL -
SUART_BAUD_ERR) > SUART_BAUD_REAL)
  #error Software UART Baudrate error > 1%
#endif

#define SUART_TH   ((SUART_OVERSAMPLING * (100-SUART_THRESHOLD)) / 100)
```

Die Präprozessordirektiven berechnen auf Basis des Systemtaktes des Mikrocontrollers, der gewünschten Baudrate sowie einem Oversampling-Faktor einen Wert, bis zu welchem der Timer zählen muss, um die benötigte Oversampling-Taktrate zu erreichen. Da für diesen Wert des Timers nur ganze Zahlen zulässig sind, entsteht ggf. durch Rundungsfehler eine

9 Englisch für: Überabtastung.

Abweichung der Taktrate. Die nachfolgenden Präprozessordirektiven berechnen deshalb diesen Fehler und geben eine Fehlermeldung beim Kompilieren des Programmcodes aus, sollte die Baudrate durch Rundungsfehler mehr als ein Prozent vom gewünschten Wert abweichen.

Die aktivierte ISR des Timer 1 des Mikrocontrollers wird nun mit der eingestellten Taktrate zyklisch aufgerufen und fragt den Signal-Pegel der Empfangsleitung ab. Der Signal-Pegel wird dazu mit Hilfe des Oversamplings über mehrer Takte gespeichert. Der Schwellwert *SUART_THRESHOLD* bestimmt die Grenze für die Erkennung des Signalspegels. Liegt der logische Signalwert im Mittel über dem eingestellten Grenzwert, wird er als logisches HIGH-Signal, andernfalls als logisches LOW-Signal interpretiert. Erkennt die ISR ein Start-Bit für einen Takt der Basistaktrate der Baudrate, werden anschließend acht Datenbits eingelesen und danach auf ein Stop-Bit gewartet. Die eingelesenen Bits werden zu einem Byte zusammengefasst und ein Flag signalisiert den Eingang neuer Daten.

Für das Senden von Daten verwendet die suart Bibliothek ebenfalls den Timer 1. Soll ein Zeichen übertragen werden, speichert die Bibliothek dieses zwischen und signalisiert der ISR mit Hilfe eines Flag, dass neue Daten für die Übertragung bereits stehen. Die ISR liest dazu das Zeichen ein, löscht das entsprechende Flag, generiert ein Start-Bit und überträgt anschließend die einzelnen Bits des zu übertragenen Zeichens. Die Übertragung wird mit einem vier-fach verzögerten Stop-Bit beendet. Durch das vierfach längere Stop-Bit wird sichergestellt, dass auch bei Fehlern die Übertragung korrekt beendet wird. Für den Zugriff auf den Zwischenspeicher von empfangenen und zu sendenden Daten stellt das Debugging Modul folgende Funktion zur Verfügung:

```
uint8_t suart_getc(void);
uint8_t suart_read(void);
void suart_putc(char data);
void suart_putc_wait(char data);
void suart_puts(char *s);
```

Die Funktion *suart_getc* wartet, bis ein Zeichen auf der seriellen Schnittstelle eingetroffen ist und gibt dieses zurück. Die Funktion *suart_read* gibt dagegen den hexadezimalen Wert *0x00* zurück, falls keine Daten vorliegen. Hier ist zu beachten, dass der Rückgabewert ebenfalls *0x00* ist, wenn der empfangene Wert ebenfalls *0x00* ist.

Über die Funktion *suart_putc* kann ein Zeichen zum Senden hinzugefügt werden. Ist bereits ein anderes Zeichen zum Senden vorhanden, wird das aktuelle Zeichen verworfen. Diese Funktion eignet sich, wenn sichergestellt werden kann, dass zwischen ihrem Aufruf und dem nächsten Aufruf der Funktion genug Zeit liegt, damit die ISR des Timers die Daten aus dem Zwischenspeicher lesen kann. Um Daten sicher zu senden, können die Funktionen *suart_putc_wait* und *suart_puts* verwendet werden. Sie erlauben das Senden eines einzelnen Zeichens oder einer Zeichenkette und warten solange, bis das vorherige Zeichen von der ISR eingelesen wurde.

Das Debugging Modul kann über einen Aufruf der Funktion *debug_process* einfache Steuerbefehle verarbeiten. Die Funktion liest dazu ein Zeichen von der seriellen Schnittstelle des Moduls ein und führt ggf. einen dazugehörigen Befehl aus (siehe Anhang C: Steuerbefehle). Im Gegensatz zu den Steuerbefehlen der Infrarot-Schnittstelle bietet das Debugging Modul nur rudimentären Zugriff auf die RGB-LEDs und die Motoren. Dafür können jedoch Fehlerdaten und Firmwareinformationen abgefragt oder ein Motortest durchgeführt werden. Da auf der verwendeten seriellen Schnittstelle auf Grund der festen Verdrahtung keine größeren Übertragungsfehler zu erwarten sind, kann der Befehlssatz des Debugging Moduls jedes beliebige Zeichen verarbeiten.

IV.11.9. Monitoring Modul

Das Monitoring Modul bietet Zugriffsfunktionen für die Analog-Digital-Wandler des mrShark. Über sie können die Motorströme, die Betriebsspannung, die Spannung der einzelnen Akkumulatoren, zwei Spannungen der Systemschnittstelle und die Systemtemperatur gemessen werden.

Die Funktion *monitor_init* initialisiert die beiden Analog-Digital-Wandler, und startet die Erfassung der Messwerte. Diese werden fortan von den Analog-Digital-Wandlern laufend neu berechnet und über Register mit Zugriff über den I2C-Bus zur Verfügung gestellt. Das Monitoring Modul bietet über folgende Funktionen Zugriff auf die Messwerte:

```
uint16_t monitor_read_temp(uint8_t address);
uint16_t monitor_read_voltage(uint8_t address, uint8_t voltage);
uint16_t monitor_read_current(uint8_t address, uint8_t current);
```

Als Adresse ist den Funktionen die I²C-Adresse des Analog-Digital-Wandlers zu übergeben, sowie ggf. eine der folgenden Werte zur Angabe, welcher Messwert ausgelesen werden soll:

```
#define MONITOR_U1       1
#define MONITOR_U2       2
#define MONITOR_U3       3
#define MONITOR_U4       4
#define MONITOR_UVCC     5

#define MONITOR_I1       1
#define MONITOR_I2       3
```

Die Temperatur gibt das Modul als 13 Bit breites Zweierkomplement zurück, mit einer LSB-Wertigkeit von 0,0625 °C. Die Temperatur berechnet sich mit Hilfe des Messwertes *mw* nach Gleichung (27) (Linear, 2010, LTC2990 datasheet).

$$T = mw * 0{,}0625\,°C \tag{27}$$

Ströme und Spannungen gibt das Modul als 14 Bit breites Zweierkomplement zurück. Ströme besitzen eine LSB-Wertigkeit von 19,42 µV / R_{ISENSE} (siehe Analog-Digital-Wandler). Daher berechnen sich die Stromwerte aus dem Messwert *mw* entsprechend Gleichung (28) (Linear, 2010, LTC2990 datasheet). Alle Spannungen außer der Betriebsspannung berechnen sich auf Basis des Messwertes *mw* und einer LSB-Wertigkeit von 305,18 µV entsprechend Gleichung (29) (Linear, 2010, LTC2990 datasheet). Für die Berechnung der Betriebsspannung aus dem Messwert *mw* wird dagegen die Gleichung (30) (Linear, 2010, LTC2990 datasheet) verwendet.

$$I = \frac{mw * 19{,}42\,\mu V}{R_{ISENSE}} \tag{28}$$

$$U = mw * 305{,}18\,\mu V \tag{29}$$

$$U_{VCC} = 2{,}5 + mw * 305{,}18\,\mu V \tag{30}$$

IV.11.10. LED Modul

Die PWM-Signale für die Steuerung der RGB-LEDs, erfolgt über einen Timer. Dieser steuert die digitalen IOs[10] des Mikrocontroller an, um ein entsprechendes PWM-Signal zu erzeugen. Dazu enthält die Quellcodedatei *led.h* Präprozessordirektiven für die von den PWM-Signalen verwendeten IOs:

```
// -------- DEFINES --------
#define LED_STAT_DDR     DDRB
#define LED_STAT_PORT    PORTB

#define LED_STAT   PB2

#define LED1_DDR   DDRD
#define LED1_PORT  PORTD

#define LED2_DDR   DDRD
#define LED2_PORT  PORTD

#define LED3_DDR   DDRC
#define LED3_PORT  PORTC

#define LED4_DDR   DDRC
#define LED4_PORT  PORTC

#define LED1               PD6
#define LED2               PD5
#define LED3               PC3
#define LED4               PC2

#define LED_N              1            // LED1
#define LED_O              2            // LED2
#define LED_S              3            // LED3
#define LED_W              4            // LED4
#define LED_STATUS         5            // LED_STATUS

#define LED_COLOR_R_DDR  DDRD
#define LED_COLOR_R_PORT PORTD

#define LED_COLOR_G_DDR  DDRB
#define LED_COLOR_G_PORT PORTB

#define LED_COLOR_B_DDR  DDRB
#define LED_COLOR_B_PORT PORTB

#define LED_COLOR_R             PD7
#define LED_COLOR_G             PB0
#define LED_COLOR_B             PB1
```

10 Englisch: Input- Outputs für Ein- Ausgänge

Neben den Angaben der IOs müssen auch die dazugehörigen Port- und Data-Directory-Register angegeben werden, welche die Datenrichtung der IOs (Ein- oder Ausgang) sowie deren Zustand (logisch HGH oder LOW) festlegen.

Um in der Firmware einzelne LEDs zu steuern, sind die LEDs nach Himmelsrichtungen benannt (LED_N, LED_O, LED_S, LED_W). Diese Bezeichnungen entsprechen den LEDs eins bis vier (LED1, LED2, LED3, LED4). Die Farbe, und ob die LED an oder aus ist, wird über Variable vom Strukturtyp *led_color* festgelegt:

```
// -------- VARIABLES --------
struct led_color{
     uint8_t led_on;
     uint8_t color_r;
     uint8_t color_g;
     uint8_t color_b;
};
extern volatile struct led_color led_color_N;
extern volatile struct led_color led_color_O;
extern volatile struct led_color led_color_S;
extern volatile struct led_color led_color_W;
```

Durch Ändern der Attribute der Variablen *led_color_N*, *led_color_O*, *led_color_S* und *led_color_W* können die Farben der RGB-LEDs angepasst werden. Die folgenden Funktionen bieten einen komfortablen Zugriff auf die Variablen und deren Attribute:

```
void led_on(uint8_t led);
void led_off(uint8_t led);

void led_all_on(void);
void led_all_off(void);

void led_set_colors(uint8_t r, uint8_t g, uint8_t b, uint8_t led);
void led_set_color_red(uint8_t r, uint8_t led);
void led_set_color_green(uint8_t g, uint8_t led);
void led_set_color_blue(uint8_t b, uint8_t led);

void led_set_allcolors(void);
void led_set_nocolors(void);
```

Mit Hilfe der o.g. Funktionen lassen sich alle oder einzelne LEDs ein- oder ausschalten sowie deren Farben bestimmen. Zu beachten ist, dass die Status LED (LED_STATUS) nur ein- oder ausgeschaltet werden kann, da sie eine einfarbige LED ist.

Der 8-Bit Timer 0 des Mikrocontrollers steuert die PWM-Signale. Dazu wird dieser in der Initialisierungsfunktion für die LEDs mit den im Abschnitt Taktversorgung errechneten Werten gestartet:

```
void led_init_timer(void){
	TCCR0A |= (1<<WGM01);			// CTC-Mode
	TCCR0B |= (1<<CS00);			// prescaler 1
	TIMSK0 |= (1<<OCIE0A);			// eable compare match A interrupt
	OCR0A = 0x6c;				// set compare match value to 108 (0x6c) >
100 Hz PWM
}
```

Die dazugehörige ISR wechselt alle 255 Takte die RGB-LED, die angesteuert werden soll. Ist die RGB-LED aktiv, wird sie angeschaltet. Alle anderen RGB-LEDs sind zu diesem Zeitpunkt aus. Anschließend werden in Abhängigkeit der eingestellten Farbwerte PWM-Signale für die Farben Rot, Grün und Blau erzeugt:

```
ISR( TIMER0_COMPA_vect ){
	static uint8_t pwm = 0x00;		// pwm counter
	static uint8_t led = 0;			// led counter
	struct led_color cur_led;		// current led

	// reset leds
	LED1_PORT &= ~(1<<LED1);
	LED2_PORT &= ~(1<<LED2);
	LED3_PORT &= ~(1<<LED3);
	LED4_PORT &= ~(1<<LED4);

	// enable led
	if( led == 0 && led_color_N.led_on ){
		LED1_PORT |= (1<<LED1);
		cur_led = led_color_N;
	}
	else if( led == 1 && led_color_O.led_on ){
		LED2_PORT |= (1<<LED2);
		cur_led = led_color_O;
	}
	else if( led == 2 && led_color_S.led_on ){
		LED3_PORT |= (1<<LED3);
		cur_led = led_color_S;
	}
	else if( led == 3 && led_color_W.led_on ){
		LED4_PORT |= (1<<LED4);
		cur_led = led_color_W;
	}

	// set color pwm
	if( cur_led.led_on ){
		if(pwm < cur_led.color_r)
			LED_COLOR_R_PORT |= (1<<LED_COLOR_R);
		else
```

```
                LED_COLOR_R_PORT &= ~(1<<LED_COLOR_R);

        if(pwm < cur_led.color_g)
                LED_COLOR_G_PORT |= (1<<LED_COLOR_G);
        else
                LED_COLOR_G_PORT &= ~(1<<LED_COLOR_G);

        if(pwm < cur_led.color_b)
                LED_COLOR_B_PORT |= (1<<LED_COLOR_B);
        else
                LED_COLOR_B_PORT &= ~(1<<LED_COLOR_B);
    }
    else{
        LED_COLOR_R_PORT &= ~(1<<LED_COLOR_R);
        LED_COLOR_G_PORT &= ~(1<<LED_COLOR_G);
        LED_COLOR_B_PORT &= ~(1<<LED_COLOR_B);
    }

    //manage led counter
    if( led > 3 )
        led = 0;
    if( !pwm )
        led++;

    // increase pwm counter
    pwm++;
}
```

Dieser Vorgang wiederholt sich endlos, sodass ein kontinuierliches Leuchten der RGB-LEDs erzeugt wird.

Die Status-LED wird nicht über den Timer gesteuert, da sie einfach über eine der o.g. Funktionen an oder ausgeschaltet werden kann und für sie kein PWM-Signal vorgesehen ist.

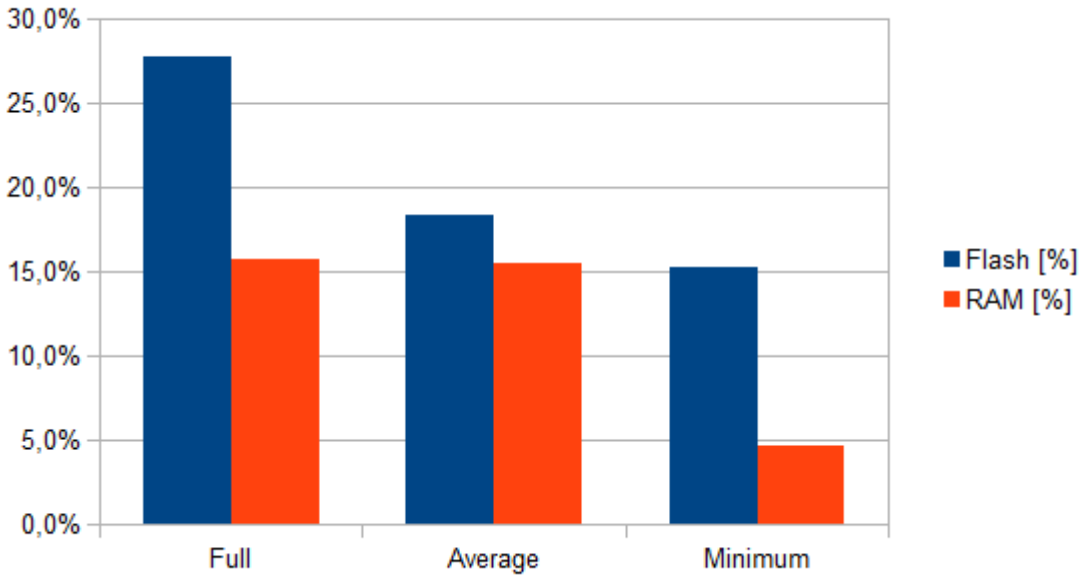
IV.11.11. Programmgröße

Die Firmware des mrShark belegt 25,9% (4,2 kB) des Programmspeichers (Flash-Speicher) und 15,7% (0,2 kB) des Datenspeichers (RAM) (siehe Anhang Code Level vs. Program Size und Abbildung 12). Um den Programmcode auch auf Mikrocontrollern mit weniger als 4 kByte Flash-Speicher einsetzen zu können, kann der Funktionsumfang der Firmware in der Quellcodedatei *config.h* angepasst werden:

```
// uncomment ONE of the following lines to reduce code size
// and to disable features
//#define CFG_CODE_LEVEL_AVG
//#define CFG_CODE_LEVEL_MIN
```

Durch Auskommentieren der Präprozessordirektiven kann der Funktionsumfang der Firmware in zwei Stufen reduziert werden, um Speicherplatz zu sparen. Die Option *CFG_CODE_LEVEL_AVG* (Code Level Average) schaltet die Funktionen für die RGB-LEDs und die Abfrage der Fehlerspeicher der Motortreiber ab. Zudem werden einige Befehle des Debugging Moduls deaktiviert. Dies reduziert den benötigten Flash-Speicher auf 18,3% (3,0 kB) und den benötigten RAM auf 15,5% (0,15 kB).

Die Option *CFG_CODE_LEVEL_MIN* (Code Level Minimum) schaltet darüber hinaus das gesamte Debugging Modul ab, was den benötigten Flash-Speicher auf 15,3% (2,5 kB) und den RAM auf 4,6% (46 Byte) reduziert.



V. Funktionstest

Um die Funktionen des mrShark zu testen wird die Firmware auf den Mikrocontroller des Roboters übertragen. Im Folgenden wurden die Signalverläufe einzelner digitaler Signale des mrShark analysiert, um einen Rückschluss auf die korrekte Funktionsweise von Funktionen der Firmware zu erhalten. Als Messinstrument wurde das digitale Speicheroszilloskop DSO6014A der Firma Agilent Technologies eingesetzt.

Zuerst wurde der Systemtakt überprüft, da dieser die Basis für eine zeitlich korrekte Abarbeitung der Programmbefehle darstellt. Mit einem Messwert von 11,05 MHz weicht der Systemtakt nicht von dem zu erwartenden Wert ab (siehe Abbildung 15).

Der Signalverlauf und das Oversampling des Debugging Moduls wurde getestet, indem von einem PC aus ein Byte an den Mikrocontroller gesendet wurde. Die ISR des Moduls wurde so verändert, dass sie die Status-LED in jedem Takt kurz an und wieder ausschaltet. Dieses Verhalten wurde zudem auf das Einlesen der Datenbits beschränkt. Dadurch konnte die zeitliche Relation des Datensignals und des Abtastsignals dargestellt werden (siehe Abbildung 16).

Zu sehen ist, dass das Oversampling bereits einen Takt vor dem eigentlichen Daten-Bit beginnt und entsprechend einen Takt früher endet (siehe Abbildung 17). Hierdurch wird der Zeitversatz zwischen dem Takt des Datensignals und dem Takt des verwendeten Timers des Mikrocontrollers deutlich.

Die Analyse eines mit dem Debugging Modul gesendeten Bytes zeigt, dass dieser Versatz sehr gering ist, da die Frequenz des Timers mit 300,3 Hz nur 0,3 Hz von der verwendeten Baudrate von 300 Bit/s abweicht (siehe Abbildung 18).

Beim Vergleich des Infrarot-Signals mit dem vom Infrarot-Decoder erzeugten seriellen Datensignals für das Remote Control Modul, zeigt sich neben den unterschiedlichen Signalverläufen von Infrarot-Signalen und seriellen Signalen, dass das serielle Datensignal mit einer Übertragungszeit eines Bits von 8,68 µs pro Bit nicht vom errechneten Wert abweicht (siehe Abbildung 20).

Zu sehen ist jedoch, dass der Anteil der Signalübergänge zwischen LOW- und HIGH-Pegeln zur Gesamtzeit eines Bits größer ist, als beim Debugging Modul. Dies ist auf die höhere Baudrate von 115, kBit/s zurückzuführen.

Bei der Betrachtung der Signalverläufe der Motortreiber zur Steuerung der Motoren, ist deutlich die Drehrichtung der Motoren durch den positiven bzw. negativen Signalverlauf sichtbar (siehe Abbildung 19 und Abbildung 22). Beide Signale wurden mit ca. 39% Geschwindigkeit aufgenommen. Das Steuersignal für die Motoren weist, sowohl in vorwärts als auch rückwärtiger Drehrichtung jedoch nur ein Tastverhältnis von ca. 33% auf. Diese Abweichung ist auf die Skalierung des übertragenen Geschwindigkeitswertes auf das für den Motortreiber gültige Intervall zurückzuführen.

Die Aufnahme der Steuersignale der RGB-LEDs 1 und 2 zeigen, dass die Frequenz zur Ansteuerung der LEDs mit 74,3 Hz stark von den berechneten 100 Hz abweicht (siehe Abbildung 21). In diesem Zusammenhang ist zu beobachten, dass die Firmware zu keinem Zeitpunkt den Programmcode der Hauptroutine ausführt und daher keine Steuerdaten zur Motorsteuerung verarbeitet werden. Dieses Verhalten lässt den Schluss zu, dass der Mikrocontroller ausschließlich mit der Ausführung von Interrupts beschäftigt ist und keine Rechenzeit für die Hauptroutine zur Verfügung steht.

Unter der Annahme, dass eine minimale Frequenz von ca. 50 Hz für die Ansteuerung der RGB-LEDs gerade noch ausreichend ist, berechnet sich nach Gleichung (22) ein neuer Zählwert für den Timer 0. Eine Messung mit diesem Wert zeigt, dass die erreichte Frequenz von 49,8 Hz der RGB-LEDs nicht mehr vom errechnet Wert abweicht (siehe Abbildung 23). Darüber hinaus ist bei dieser Frequenz eine Ausführung der Hauptroutine zu beobachten.

VI. Evaluierung

VI.1. Zielsetzung und Bewertungsverfahren

Ziel der Evaluierung soll die Bewertung der Tauglichkeit des vorangegangenen Konzepts und die Realisierung in Bezug auf eine industrielle Kleinserienfertigung sein. Dazu untersucht diese Evaluierung drei zentrale Aspekte des mrShark.

Zuerst werden die Bauteile auf ihre Verfügbarkeit und Bauform bewertet. Anschließend betrachtet die Evaluierung die Bauteilvielfalt, also wie viele unterschiedliche Bauteile es gibt, und bewertet diese. Zuletzt wird die Umsetzung des Stromlaufplans in ein Platinenlayout unter Betrachtung einer maschinellen Bestückung und einem Reflow-Lötverfahren überprüft und bewertet.

Die Bewertung beruht auf einem Punkteverfahren und Multiplikatoren. Die maximale Punktzahl für ein zu überprüfendes Item sind 100 Punkte. Der maximale Multiplikator beträgt 1,0. Sowohl Punkte und Multiplikatoren sind ausschließlich positiv und aus der Menge der natürlichen Zahlen.

Jeder der drei Aspekte Bauteilverfügbarkeit (P_1), Bauteilvielfalt (P_2) und Layout (P_3) wird nach Gleichung (31) mit gleichen Gewichtungen für die Ermittlung einer Gesamtbewertung $P_{mrShark}$ auf Basis des Quotienten der erreichten (P_1 bis P_3) zu den maximal erreichbaren Punkten (P_{GES}), mit $P_{GES} \in \{0, \ldots, 100\}$ herangezogen.

$$P_{mrShark} = \frac{P_1 + P_2 + P_3}{P_{GES}} * 100\ Punkte \tag{31}$$

Für die Evaluierung wird von einer Kleinserie von 150 Robotern ausgegangen, was dem Umfang von drei eigenständigen Teams entspricht. Jedes Team besitzt demnach 50 Roboter, welche zwei Sätze mit je 25 Robotern bilden. Dadurch lassen sich pro Team zwei parallele Spiele mit je 22 Spielern und 3 Auswechselspielern realisieren.

Als Datum für die Erhebung der Verfügbarkeiten der einzelnen Bauteile wurde der 18. August 2013 gewählt. Nicht in die Bewertung einbezogen sind Standartbauteile wie Widerstände mit Werten > 1 Ω oder Kondensatoren, da diese in der Regel bereits beim

Bestückungsunternehmen vorrätig sind. Ausgenommen hiervon sind Bauteile mit ungewöhnlichen Werten oder Bauformen. Die Liste er benötigten Bauteile ist damit ein Ausschnitt der Bauteilliste des mrShark, welche aus dem Stromlaufplan des mrShark erstellt werden kann.

Alle Anzahlen von Bauteilen stammen aus der Bauteilliste (siehe Anhang Part Informations) für die Bestellung eines Leiterplattennutzen von mrShark-Robotern in Verbindung mit weiterer Hardware, welche nicht Bestandteil dieser Arbeit sind.

VI.1.1. Bewertungsverfahren der Bauteilverfügbarkeit

Entscheidend für die Kosten einer Kleinserie ist die Verfügbarkeit von Bauteilen. Können Bauteile über einen Distributor bezogen werden, entfallen nur einmalig Versandkosten. Sind Bestellungen bei mehreren Distributoren nötig, sind diese entsprechend negativer zu bewerten. Bestellungen im europäischen Ausland sind, auch bei hoher Bauteilverfügbarkeit teurer da, zusätzliche Versand- und/oder Zollkosten auftreten können. Eine innereuropäische Bestellung ist in diesem Falle vorzuziehen. Sind Bauteile nur direkt über den Hersteller zu beziehen, reduziert dies die Auswahl von Bezugsquellen und führt dazu, dass der Bauteilpreis einzig und allein vom Hersteller bestimmt wird. Dies ist ebenfalls negativ zu bewerten. Als Distributor ist ein möglichst breit aufgestelltes Unternehmen zu wählen, da dieses einen großen Sortimentumfang und hohe Stückzahlen für lagernde Artikel verspricht. Als europäischer Distributor für Elektronikbauteile ist z.B. Farnell zu nennen.

Für die Bauteilverfügbarkeit wird die beim Distributor Farnell lagernde Menge des benötigten Bauteils herangezogen. Die Punkte P eines Bauteils berechnen sich nach Gleichung (32) über den Quotienten aus der benötigten Anzahl des Bauteils n und der lagernden Menge k des Bauteils beim Distributor. Diese Punktzahl wird mit dem Multiplikator m multipliziert.

Der Multiplikator m ist 1,0 falls das Bauteil über den Distributor Farnell ohne Mehrkosten wie Zollgebühren etc. aus einem europäischen Lager bezogen werden kann. Ist das Bauteil bei Farnell nur mit Mehrkosten aus einem außer-europäischen Lager zu beziehen, ist der Multiplikator $m = 0{,}75$. Ist das Bauteil nur über einen anderen Distributor als Farnell zu beziehen, ist der Multiplikator $m = 0{,}5$. Sollte es nur direkt beim Hersteller verfügbar sein, ist

der Multiplikator *m = 0,25* und sollte es nirgends zu beziehen sein, ist der Multiplikator *m = 0,0*. Ist der Quotient aus k und n größer als 1,0 ist dieser auf den Wert 1,0 zu setzen, sodass keine Punktzahl größer 100 Punkte möglich ist.

$$P = \frac{k}{n} * m * 100\ Punkte \tag{32}$$

VI.1.2. Bewertungsverfahren Bauteilvielfalt

Die Anzahl der unterschiedlichen Bauteile oder die Bauteilvielfalt wirkt sich auf die Fertigungsmethode einer Leiterplatte aus. Eine automatische Bestückungsmaschine verfügt nur über eine begrenzte Anzahl von Feedern. Werden also mehr Bauteile benötigt, als Feeder zur Verfügung stehen, müssen diese während der Bestückung gewechselt werden, was zusätzliche Arbeitszeit kostet. Auch wenn die maximale Anzahl von Feedern nicht erreicht ist, benötigt die Vorbereitung der Bestückungsmaschine umso mehr Zeit, je größer die Bauteilvielfalt ist. Eine geringe Bauteilvielfalt ist daher Vorteilhafter als eine große Bauteilvielfalt. Für die Bewertung der Bauteilvielfalt wird eine einzelne Punktzahl nach Gleichung (33) berechnet. *v* ist hierbei die Vielfalt und n die Gesamtzahl aller Bauteile.

$$P = \left(1 - \frac{v}{n}\right) * 100\ Punkte \tag{33}$$

VI.1.3. Bewertungsverfahren Layout

Das Layout einer Leiterplatte beeinflusst teilweise sowohl eine automatische Fertigung in einer Kleinserienfertigung als auch die anschließende Benutzung (Scheel, 1997). Umso gleichmäßiger ein Leiterplattenlayout ist, umso effizienter kann die dazugehörige Leiterplatte bestückt werden. Dies zeigt sich z.B. in der Ausrichtung einzelner Bauteile. Weisen diese eine große Anzahl unterschiedlicher Ausrichtungen auf, muss das jeweilige Bauteil immer wieder gedreht werden, bevor es auf der Leiterplatte platziert werden kann. Dies erhöht die Produktionszeit, die eine Bestückungsmaschine für die Bestückung einer Leiterplatte benötigt. Eine große Variation in der Ausrichtung von Bauteilen führt ebenfalls zu Problemen wie dem

Grabsteineffekt in einem Reflow-Lötverfahren (siehe Leiterplattenbestückung). Die einzelnen Bauteile einer Leiterplatte sollten daraus folgernd möglichst ein und dieselbe Ausrichtung aufweisen. Dies erhöht zudem die Lesbarkeit der Bauteilwerte bei der Fehlersuche o.ä..

Zur Vereinfachung der Fehlersuche trägt auch eine einheitliche Platzierung einzelner Bauteilgruppen auf ein und dieselbe Leiterplattenseite bei. So lässt sich z.B. gezielter nach einem Widerstand mit einem bestimmten Wert suchen, wenn seine Position nur auf einer Leiterplattenseite liegen kann. Dies erhöht ebenfalls die Effizienz einer ggf. nötigen Fehlersuche.

Für die Bewertung des Layouts wird die Anordnung der Bauteile auf der Leiterplatte betrachtet und bewertet. Optimal ist die Platzierung von gleichen Bauteilen auf nur einer Seite der Leiterplatte mit derselben Ausrichtung. Bauteile mit gleichem Bauteilwert und gleicher Bauform werden zu einer Gruppe zusammengefasst. Die Punkte für eine Bauteilgruppe P_{GROUP} berechnen sich nach Gleichung (34). n_{MAIOR} ist dabei die Anzahl der Bauteile, welche sich auf der Leiterplattenseite befinden, auf der sich die meisten Bauteile dieser Gruppe befinden. n_{MINOR} ist demnach die Anzahl der Bauteile, welche sich auf der Leiterplattenseite befinden, auf denen sich die wenigsten Bauteile dieser Gruppe befinden. Befinden sich alle Bauteile auf einer Leiterplattenseite ist der Quotient aus n_{MAJOR} und $n_{MINOR} = 1{,}0$

$$P_{GROUP} = \frac{n_{MINOR}}{n_{MAIOR}} * m * 100\,Punkte \tag{34}$$

Der Multiplikator *m* ist 1,0 wenn alle Bauteile dieselbe Ausrichtung haben. Für jede Anzahl an Ausrichtungen > 1 einer Gruppe wird der Multiplikator um 0,25 verringert. So ist er z.B. bei zwei unterschiedlichen Ausrichtungen 0,75. Bei drei unterschiedlichen Ausrichtungen 0,5 usw.

VI.2. Bewertung Bauteilverfügbarkeit

Es zeigt sich, dass die Bauteilverfügbarkeit zu 91% (entspricht 91 Punkten), entsprechend der Tabelle Bauteilverfügbarkeit (siehe Anhang Bauteilbewertungen) und Abbildung 13, geeignet für eine industrielle Kleinserienfertigung ist, da die Mehrheit der Bauteile bei dem Distributor Farnell zu beschaffen ist. Drei Bauteile müssten über einen anderen Distributor beschafft

werden und nur ein Bauteil ist direkt beim Hersteller zu beziehen. Alle Bauteile sind in ausreichender Menge verfügbar, sodass eine Kleinserienfertigung mit 150 mrShark-Robotern in Bezug auf die Bauteilverfügbarkeit möglich ist.

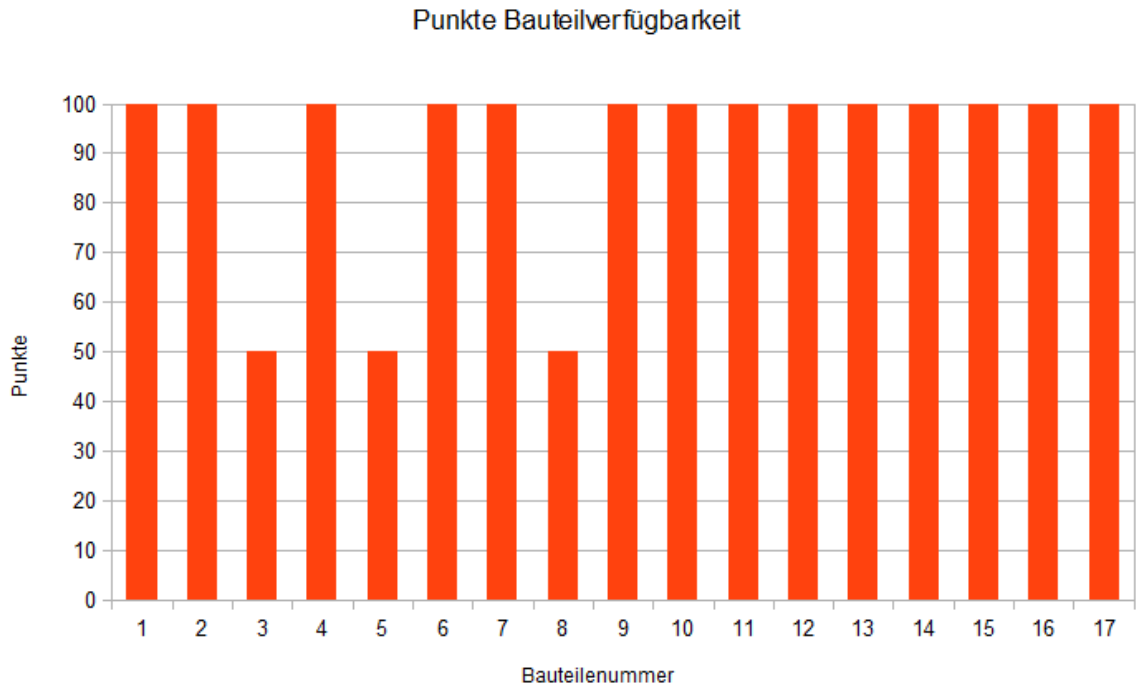


Abbildung 13: Punkteverteilung Bauteilbewertung (Anhang B: Bauteilbewertungen)

VI.3. Bewertung Bauteilvielfalt

Der mrShark verfügt insgesamt über 66 Bauteile, davon 26 unterschiedliche (siehe Anhang B: Bauteilbewertungen). Nach Gleichung (33) ergibt sich daraus eine Punktzahl von

$$P = \left(1 - \frac{v}{n}\right) * 100\ Punkte = \left(1 - \frac{26}{66}\right) * 100\ Punkte = \mathbf{61\ Punkte}$$

Dies entspricht einer Eignung der Bauteilvielfalt für eine industrielle Kleinserienfertigung von 61%. Die absolute Anzahl von 26 unterschiedlichen Bauteilen ist zudem gering und kann damit voraussichtlich von einer Bestückungsmaschine bedient werden, sodass keine weiteren Produktionskosten durch das Umrüsten der Bestückungsmaschine entstehen.

VI.4. Bewertung Layout

Das Layout des mrShark in der Version 1.0 ist zu 87% (entspricht 87 Punkten), entsprechend der Tabelle Layoutbewertung (siehe Anhang Bauteilbewertungen) und Abbildung 14, geeignet für eine industrielle Kleinserienfertigung, da sich die meisten Bauteile einer Gruppe nur auf einer Leiterplattenseite befinden (siehe Abbildung 24, 25, 26 und 27). In 5 von insgesamt 26 Gruppen sind die Bauteile in zwei unterschiedlichen Ausrichtungen und in 3 weiteren Gruppen in mehr als zwei unterschiedlichen Ausrichtungen angeordnet.

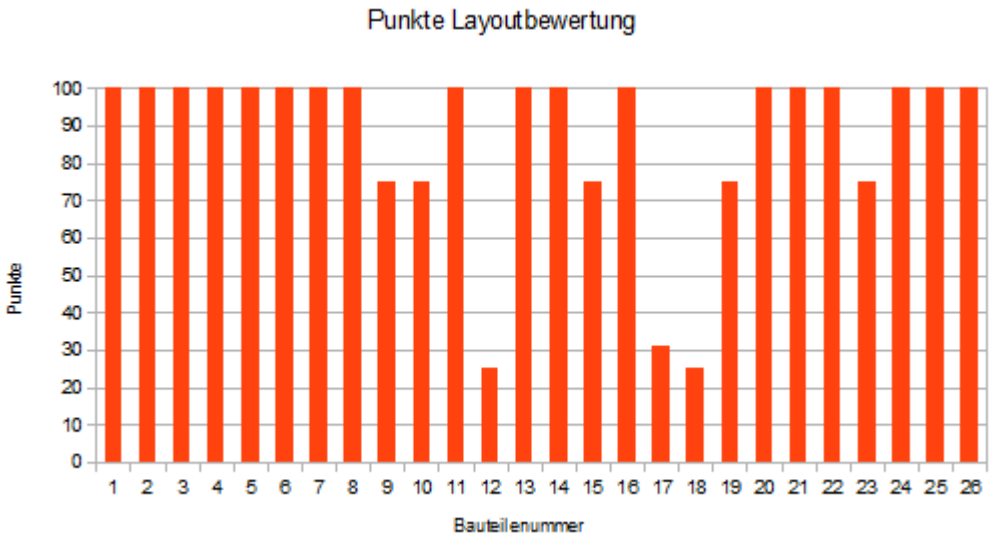


Abbildung 14: Punkteverteilung Layoutbewertung (Anhang B: Bauteilbewertungen)

VI.5. Gesamtbewertung mrShark

Für die Gesamtbewertung des mrShark ergeben sich nach Gleichung (31) und den Einzelpunktzahlen der drei Aspekte Bauteilverfügbarkeit (87 Punkte), Bauteilvielfalt (61 Punkte) und Layout (87 Punkte) eine Gesamtpunktzahl von

$$P_{mrShark} = \frac{P_1 + P_2 + P_3}{P_{GES}} * 100\ Punkte = \frac{91 + 61 + 87}{300} * 100\ Punkte = \mathbf{80\ Punkte}$$

Dies entspricht einer Eignung des mrShark für eine industrielle Kleinserienfertigung von 80%. Dieser Wert kann als ausreichend angenommen werden, da im Durchschnitt mehr als drei Viertel der Leiterplatte des mrShark für eine effiziente Kleinserienfertigung geeignet ist. Bedingt durch die kleinen Platinenmaße (siehe Vorgaben) ist jedoch keine effizientere Auswahl an Bauteilen oder deren Platzierung mehr möglich, sodass die erreichte Gesamtbewertung als gutes Ergebnis in Bezug auf die Eignung des mrShark für eine Kleinserienfertigung angesehen werden kann.

VI.6. Evaluierung der Zielvorgaben

Dieses Kapitel soll die in den Zielen definierten Vorgaben auf ihre Umsetzung im mrShark überprüfen (siehe Tabelle 9).

Ziel	Bewertung
Kostengünstige Produktion	Die Kosten für eine Produktion sind ohne eine Konkrete Anfrage an ein Fertigungsunternehmen schwer abzuschätzen. Da jedoch viele Standartbauteile wie Transistoren und keine besonderen integrierten Schaltkreise verwendet werden, sollte eine kostengünstige Produktion gegeben sein.
Hochverfügbare Bauteile	Die Bewertung der Bauteilverfügbarkeit, ergab eine hohe Verfügbarkeit der Bauteile. Die meisten Bauteile sind zudem auch in sehr großen Stückzahlen verfügbar.
Einfache Handhabung	Die Handhabung des mrShark ist nicht kompliziert. Die Installation der Steuerplatine benötigt jedoch Lötkenntnisse. Für die Programmierung sind Kenntnisse über Mikrocontrollertechniken vorausgesetzt. Da die Firmware aber leicht anzupassen ist, sollte ein Einstieg hier nicht schwer fallen.
Einfache Programmierung	Die Firmware ist in der weit verbreiteten Programmiersprache C programmiert. Dies ermöglicht es einer breiten Bandbreite an Programmieren die Firmware zu modifizieren und an ihre Bedürfnisse anzupassen. Mikrokontrollerkenntnisse vorausgesetzt.
Einfache Erweiterungsmöglichkeiten	Die Systemschnittstelle bietet unter anderem verschiedene Kommunikationsschnittstellen zur Erweiterung des mrShark. Hierdurch ist eine Vielzahl von Erweiterungen möglich.
Lange Laufzeit	Die Laufzeit des mrShark lässt sich nicht exakt vorhersagen und hängt auf Grund des großen Anteils der Motoren an der Energiebilanz stark von deren Benutzung ab. Unter der Voraussetzung, dass zwei 240 mAh LiPo-Akkus verwendet werden und der Stromverbrauch bei erwarteten 360 mA (siehe

	Energiebilanz) liegt, ist eine Laufzeit von ca. 1 Stunde zu erwarten.
Autonome Stromversorgung	Eine autonome Stromversorgung kann durch bis zu zwei parallel geschaltete Akkumulatoren gewährleistet werden.
Kompatibilität mit bestehendem System	Die Infrarot-Steuerung des mrShark ist kompatibel mit der bisher eingesetzten technischen Einrichtung des Mixed-Reality-System. Das neue Steuerprotokoll in der Version 2.0 muss für eine Abwärtskompatibilität ausgewechselt oder angepasst werden.
Messung von relevanten Strömen und Spannungen des System	Der mrShark kann alle relevanten Ströme und Spannungen über die am I^2C-Bus beteiligten Analog-Digital-Wandler messen.
Optimierung in Hinblick auf eine industriell gefertigte Kleinserie	Mit einer überdurchschnittlichen Bewertung in der Bauteilvielfalt und einer hohen Bewertung des Layouts (siehe Bewertung Bauteilvielfalt und Bewertung Layout), eignet sich der mrShark sehr gut für eine industrielle Kleinserienfertigung,

Tabelle 9: Evaluierung der Zielvorgaben

VII. Zusammenfassung

Die Steuerelektronik für den Mixed-Reality-Roboter mrShark bietet primär eine Steuerung für zwei Gleichstrommotoren via Infrarot-Schnittstelle. Hierfür wurde ein neues Steuerprotokoll entwickelt, welches eine einfache und sichere Datenübertragung ermöglicht. Der Funktionsumfang des Steuerprotokolls wurde gegenüber dem bisher verwendeten Protokoll erweitert und bietet jetzt z.B. die Möglichkeit die Identifikationsnummer des Roboters dauerhaft zu verändern.

Darüber hinaus können vier RGB-LEDs zur Beleuchtung des Robotergehäuses eingesetzt werden. Eine Systemschnittstelle ermöglicht die Erweiterung der Elektronik um zusätzliche Funktionen und bietet Zugriff auf interne Bussystem und Schnittstellen.

Die Systemschnittstelle ermöglicht auch das Laden der in den Roboter integrierten Akkumulatoren, ohne diese aus dem Roboter entfernen zu müssen. Dadurch wäre das Laden mehrere Roboter in einer externen Ladestation möglich. Für diesen Zweck entkoppelt die Systemschnittstelle die Akkumulatoren von der Steuerelektronik, was den Betrieb des Roboters über eine externe Stromversorgung und das parallele Laden der Akkumulatoren ermöglichen würde.

Über eine serielle Schnittstelle lassen sich auf einfache Weise Daten des integrierten Mikrocontrollers abfragen und manipulieren. Dies erleichtert die Fehlerdiagnose und den Test des Systems. Hierüber wäre auch die Kommunikation mit Erweiterungen der Steuerelektronik in Verbindung mit der Systemschnittstelle möglich.

Der integrierte I2C-Bus kann bei Bedarf mit der Systemschnittstelle verbunden werden, sodass ein Zugriff auf die integrierten Schaltkreise des mrShark durch externe Hardware möglich ist. In diesem Kontext wäre z.B. eine Ladezustandskontrolle durch eine externe Ladestation möglich. Denkbar wäre auch eine Vernetzung mehrere Roboter über den I^2C-Bus, um Daten unter den unterschiedlichen Einheiten auszutauschen.

Die Steuerelektronik bietet durch Analog-Digital-Wandler die Möglichkeit Spannungen und Ströme des Systems zu messen. Dadurch kann auf Änderungen der Betriebsspannung oder der Motorströme reagiert werden.

Der eingesetzte Mikrocontroller ermöglicht eine energiesparende Arbeitsweise, bei gleichzeitig ausreichender Rechen- und Speicherkapazität. Er ist darüber hinaus Pin-kompatibel zu weiteren Mikrocontrollern, sodass er einfach durch einen anderen Typ ersetzt werden kann. Dies ermöglicht unterschiedlich bestückte Platinenversionen, ohne dass ein neues Platinenlayout notwendig ist.

Die Auswahl der verwendeten Bauteile der Platine ist optimiert für die Verwendung des mrShark in einer industriellen Kleinserienfertigung. Die hohe Bauteilverfügbarkeit ermöglicht zudem eine kostengünstige Produktion, einen langen Produktzyklus und die kosteneffiziente Bestellung der benötigten Bauteile bei wenigen Distributoren.

Der Energieverbrauch des mrShark hängt stark vom Einsatz der Motoren ab. Negativ fällt hier zudem die Infrarot-Schnittstelle ins Gewicht, da diese bei aktiviertem Transmitter einen sehr hohen Strombedarf aufweist. Idealer wäre hier die Verwendung eines reinen Infrarot-Empfängers, was aber durch den kleinen Bauraum auf der Platine nicht möglich ist.

Um Strom zu sparen, reguliert die Firmware den Energieverbrauch des Mikrocontrollers, indem nicht benötigte Komponenten abgeschaltet werden und der Mikrocontroller so oft wie möglich in den Ruhezustand geht.

Bei der Analyse der Funktionen des mrShark wurde deutlich, dass der Mikrocontroller viel Rechenzeit für die Ausführung von Interrupt-Service-Routinen benötigt und daher wenig Rechenzeit für die Hauptroutine zur Verfügung steht. Dies beeinträchtigt maßgeblich die Steuerung der RGB-LEDs. Hier ist die Verwendung eines höheren Systemtaktes anzuraten, auch wenn dieser eine höhere Stromaufnahme durch den Mikrocontroller zur Folge hätte.

Als Erweiterungen des mrShark wäre eine Regelung der Motordrehzahl über die Strommessung der Analog-Digital-Wandler denkbar. So ließen sich Drehzahlunterschiede der Motoren ausgleichen. Über einen Bootloader und die Systemschnittstelle könnte die Firmware unkompliziert geändert werden. Dies würde ein Systemupgrade auch ohne spezielle Hardware möglich machen.

Der mrShark stellt durch seine neuen Funktionen und seine Ausrichtung für eine industrielle Kleinserienfertigung eine kostengünstige Alternative zum bestehenden Roboter dar.

VIII. Literaturverzeichnis

Atmel Cooperation (2004). ATmega 48V/88V/168V datasheet (Revision 2545D–AVR–07/04). San Jose, CA

Avago Technologies (2010). ASMT-YTD2-0BB02 datasheet (Revision AV02-2592EN).

Beierlein T. & Hagenbruch O. (2011). Taschenbuch Mikroprozessortechnik (4. Auflage). München: Carl Hanser Verlag

DATACOM Buchverlag GmbH (2013). Leiterplatte. Verfügbar unter http://www.itwissen.info/definition/lexikon/Leiterplatte-PCB-printed-circuit-board-LP.html [Abrufdatum 04.08.2013]

Fleury P. (2006). I2C Master Interface. Verfügbar unter http://homepage.hispeed.ch/peterfleury/avr-software.html [Abrufdatum 29.08.2013]

Frehner M. (2007). Alles über Akkus. Verfügbar unter http://www.funkcom.ch/akkuinfos.htm [Abrufdatum 06.08.2013]

Grossar L. (2007). Aufbau der Von-Neumann Architektur. Verfügbar unter https://de.wikipedia.org/w/index.php?title=Datei:Von-Neumann_Architektur.svg [Abrufdatum 06.08.2013]

Herter E., Lörcher W. (2004). Nachrichtentechnik (9. Auflage). München, Wien: Carl Hanser Verlag

HiSystems GmbH (2013). LiPo-Grundlagen. Verfügbar unter http://www.mikrokopter.de/ucwiki/LiPo-Grundlagen [Abrufdatum 06.08.2013]

Hüfner U. (2004). Akkutypen. Verfügbar unter http://www.akkumagic.de/html/akkutypen.html [Abrufdatum 06.08.2013]

Jetzek U. (2013). Vorlesungsskript ASIC-Design / VHDL BK 107, Unit 7: UART (Rev. PA2). Kiel: University of Applied Sciences Kiel - Institute for Communications Technology and Microelectronics

Linder H., Brauer H. Lehmann C. (2008). Taschenbuch der Elektrotechnik und Elektronik (9. Auflage). München: Car Hanser Verlag

Linear Technology (2010). LTC2990 datasheet (LT 1211 REV C). Milpitas, California

Moltrecht E. K. W. (1998). Amateurfunk-Lehrgang Teil I: Elektrotechnik, Elektronik. Baden-Baden: Verl. für Technik und Handwerk

Moraw K. (2013). Die kleine Akku-Übersicht. Verfügbar unter http://www.flyheli.de/akku.htm [Abrufdatum 06.08.2013]

NorthernStars, Fachhochschule Kiel (2013). mrShark. Verfügbar unter http://northernstars-wiki.wikidot.com/projects:mrshark [Abrufdatum 04.08.2013]

NXP (2009). BC817; BC817W; BC337 datasheet (Rev. 06).

Philips Semiconductors (1999). PCA9540 datahseet (Revision SCPS113J). Sunnyvale, California

Riemenschneider F. (2012). Die größten Mikrocontroller-Hersteller 2011: Atmel und Infineon rücken vor. Verfügbar unter http://www.elektroniknet.de/halbleiter/sonstiges/artikel/86934/ [Abrufdatum 04.08.2013]

RoboCup German Open (2013). RoboCup. Verfügbar unter http://www.robocupgermanopen.de/de/robocup [Abrufdatum 04.06.2013]

RT Lions, Hochschule Reutlingen Fakultät Technik (2013). FAQ zur Mixed-Reality. Verfügbar unter http://www.tec.reutlingen-university.de/robocup/mixed-reality/faq.html [Abrufdatum 03.08.2013]

Sautter R. (1988). Leiterplatten mit oberflächenmontierten Bauelementen - 1. Auflage. Würzburg: Voegl Buchverlag

Scheel W. (1997). Baugruppentechnologie der Elektronik. Berlin: Verlag Technik

Schwarz A. (2013). Basiswiderstand. Verfügbar unter https://www.mikrocontroller.net/articles/Basiswiderstand [Abrufdatum 20.08.2013]

Schwarz A. (2013). AVR Typen. Verfügbar unter https://www.mikrocontroller.net/articles/AVR_Typen [Abrufdatum 04.08.2013]

SIGEM Elektronik (2013). SMD-Design. Verfügbar unter http://www.sigem-elektronik.de/elektro/cad/eagle/biblio/smdesign.htm [Abrufdatum 04.08.2013]

SOLARBOTICS (2013). GM15 Tiny Planetary Geared Pager Motor. Calgary, Canada

Texas Instruments (1995/1999). TIR1000 datasheet (Revision SLLS228F). Dallas, Texas

Texas Instruments (2012). DRV8830 datasheet (Revision SLVSAB2F). Dallas, Texas

The RoboCup Federation (2013). Objective RoboCup. Verfügbar unter http://www.robocup.org/about-robocup/objective [Abrufdatum 04.06.2013]

Urbanski K., Woitowitz R. (2007). Digitaltechnik (5. Auflage). Berlin: Springer-Verlag

Wüst K. (2003). Mikroprozessortechnik (Ausgabe 2). Wiesbaden: Vieweg & Sohn Verlag

IX. Abbildungsverzeichnis

Abbildung 1: Konzept Steuerungselektronik mrShark....13
Abbildung 2: Aufbau der Von-Neumann-Architektur (Grossar, 2007)....16
Abbildung 3: AVR Supply-Current (Anhang A: Berechnungen)....19
Abbildung 4: Prinzip Motortreiber mit DAC....20
Abbildung 5: Signalverlauf RGB-LEDs Farbe Rot....30
Abbildung 6: Signalverlauf RGB-LEDs Farbe Grün....30
Abbildung 7: Signalverlauf RGB-LEDs Farbe Blau....31
Abbildung 8: Signalverlauf RGB-LED-Steuersignale....31
Abbildung 9: Blindwiderstand Xc über Frequenz f (Anhang A: Berechnungen)....40
Abbildung 10: ATmega 48V/88V/168V Supply Current vs. Frequency für 3,3V (über 15 MHz für 4,0V) (Anhang A: Berechnungen)....44
Abbildung 11: Module mrShark Firmware....48
Abbildung 12: Code Level vs. Program Size (Anhang A: Berechnungen)....66
Abbildung 13: Punkteverteilung Bauteilbewertung (Anhang B: Bauteilbewertungen)....73
Abbildung 14: Punkteverteilung Layoutbewertung (Anhang B: Bauteilbewertungen)....74
Abbildung 15: Signalverlauf Systemtakt....97
Abbildung 16: 10 Bit UART RX Daten (gelb) und Oversampling (rot), Baudrate = 300 Bits/s, 8N1....98
Abbildung 17: 1 Bit UART RX Daten (gelb) und Oversampling (rot), Baudrate = 300 Bit/s, 8N1....98
Abbildung 18: UART TX Daten (0x4d), Baudrate 300 Bit/s, 8N1....99
Abbildung 19: UART RX Daten (gelb) und IR Daten (rot), Baudrate = 115,2 kBit/s, 8N1....99
Abbildung 20: Motor Control PWM, forward, speed = 39%....100
Abbildung 21: Motor Control PWM, backward, speed = 39%....100
Abbildung 22: RGB-LED 1 (gelb) und 2 (rot) Control Signal, Compare Match Value = 0x6c 101
Abbildung 23: RGB-LED 1 (gelb) und 2 (rot) Control Signal, COmpare Match Value = 0xd8 101
Abbildung 24: Platinenlayout mrShark Layer Bottom....112
Abbildung 25: Platinenlayout mrShark Layer Top....112
Abbildung 26: Bestückungsplan mrShark Bottom....114
Abbildung 27: Bestückungsplan mrShark Top....114

X. Tabellenverzeichnis

Tabelle 1: Übersicht AVR Mikrocontroller (Schwarz, 2013, AVR Typen).....18
Tabelle 2: Farbanteile und-werte RGB-LEDs.....29
Tabelle 3: Bewertung der Eigenschaften verschiedener Akkumulator-Typen (Hüfner, 2004). 34
Tabelle 4: Maximaler Stromverbrauch Bauteile mrShark bei 3,3V-5V.....37
Tabelle 5: Erwarteter typischer Stromverbrauch Bauteile mrShark bei 3,3V-5V.....37
Tabelle 6: Vorteiler p in Abhängigkeit der Bit-Breite des Zählwertes u (Atmel, 2004, ATmega 48V/88V/168V datasheet).....42
Tabelle 7: Zählwert in Abhängigkeit von Vorteiler und Systemtakt.....43
Tabelle 8: Vorwiderstand LED-Farben (siehe Anhang Berechnung Basiswiderstand).....46
Tabelle 9: Evaluierung der Zielvorgaben.....76

XI. Anhang

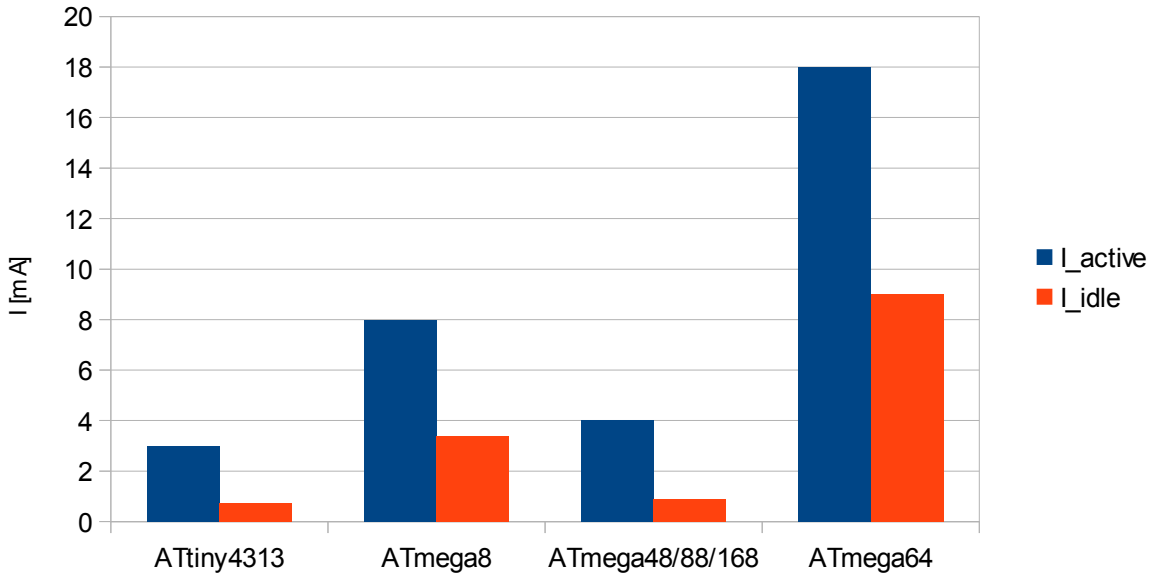
Anhang A: Berechnungen

AVR Supply Current

Autor: H. Eilers (2013)

F_CPU =	11,0 Mhz
U_Supply =	3,3 V

Microcontroller	I_active [mA]	I_idle [mA]
ATtiny4313	3,0	0,750
ATmega8	8,0	3,400
ATmega48/88/168	4,0	0,900
ATmega64	18,0	9,000 @ 4,5V



Quellen:

Atmel Corporation (2011). ATtiny2313A/4313 datasheet Revision 8246B – 10/11, Fig. 23-50 & Fig. 23-55. San Jose: Atmel Cooperation

Atmel Corporation (2013). ATmega8(L) datasheet Revision 2486AA- 02/2013, Fig. 119 & Fig. 126. San Jose: Atmel Cooperation

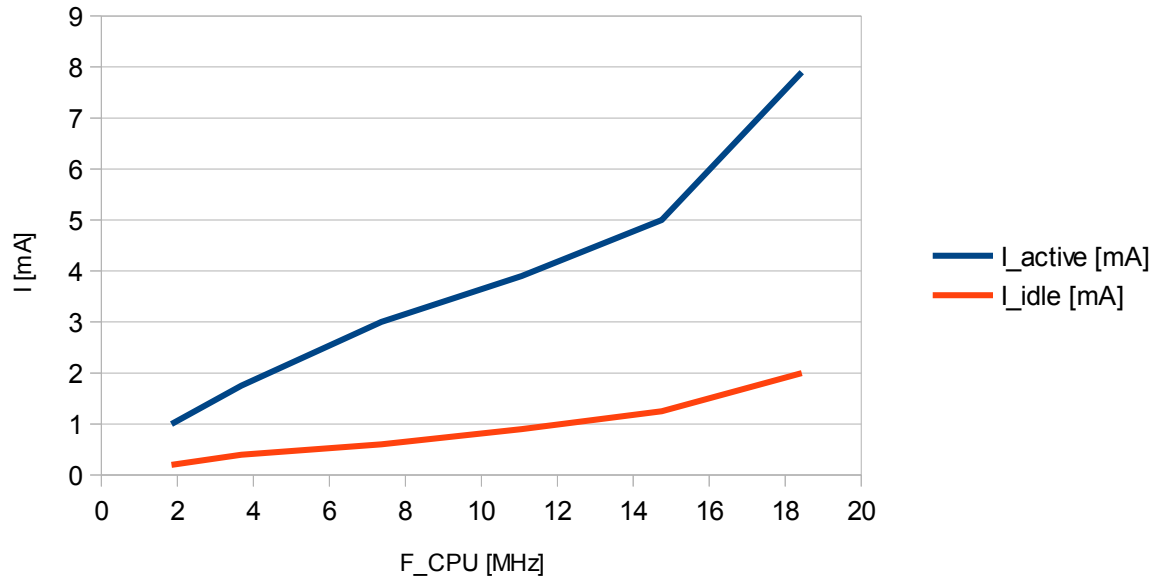
Atmel Corporation (2004). ATmega48V/88V/168V datasheet Revision2545D–AVR–07/04, Fig. 137 & 143 . San Jose: Atmel Cooperation

Atmel Corporation (2013). ATmega64(L) datasheet Revision 2490R-02/13, Fig. 164 & Fig. 171. San Jose: Atmel Cooperation

Supply Current vs. Frequency

Device: ATmega168

F_CPU [MHz]	I_active [mA]	I_idle [mA]
1,8432	1,0	0,2
3,6864	1,8	0,4
7,3728	3,0	0,6
11,0592	3,9	0,9
14,7456	5,0	1,3
18,4320	7,9	2,0



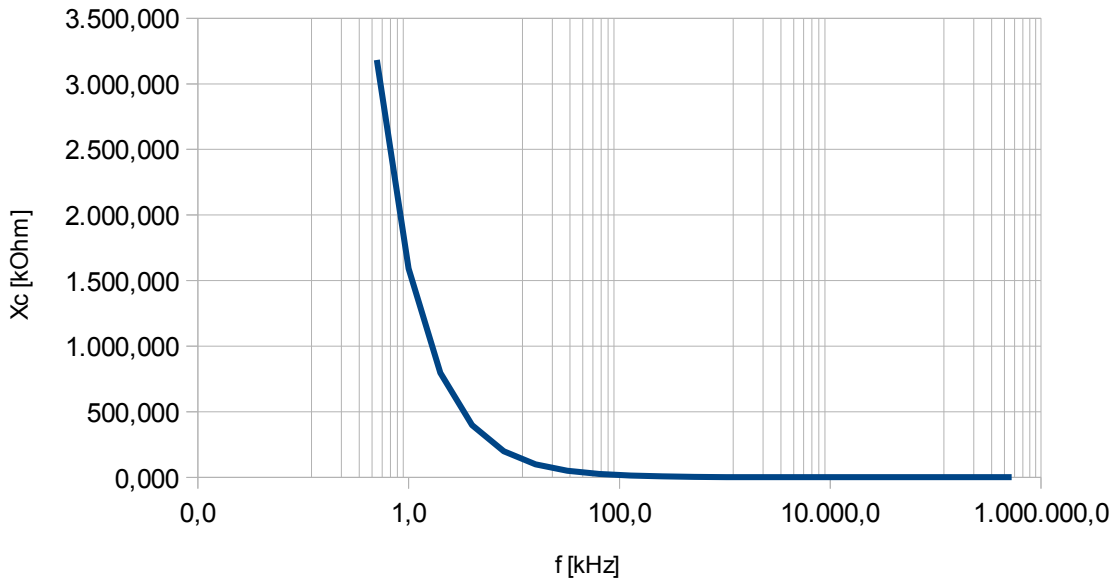
Quelle:

Atmel Corporation (2004). ATmega48V/88V/168V datasheet Revision2545D–AVR–07/04, Fig. 137 & 143 . San Jose: Atmel Cooperation

Blindwiderstand Xc

C = **1,00E-007 F**

f [kHz]	Xc [kOhm]
0,5	3.183,099
1,0	1.591,549
2,0	795,775
4,0	397,887
8,0	198,944
16,0	99,472
32,0	49,736
64,0	24,868
128,0	12,434
256,0	6,217
512,0	3,108
1.024,0	1,554
2.048,0	0,777
4.096,0	0,389
8.192,0	0,194
16.384,0	0,097
32.768,0	0,049
65.536,0	0,024
131.072,0	0,012
262.144,0	0,006
524.288,0	0,003



Maximum Timer Timings

clk_timer = 19 clks (without code)

f_cpu [Hz]	t_clk [s]	t_timer_max [s]	f_timer_max [Hz]	Supply Current [mA]
1.843.200	5,425E-07	1,031E-05	97.011	0,80
3.686.400	2,713E-07	5,154E-06	194.021	1,80
7.372.800	1,356E-07	2,577E-06	388.042	3,00
11.059.200	9,042E-08	1,718E-06	582.063	4,00
14.745.600	6,782E-08	1,289E-06	776.084	5,00
18.432.000	5,425E-08	1,031E-06	970.105	7,00 @4V

LED Timer

f_pwm_ges = 100 Hz
Resolution = 256 Bit
Num of LEDs = 4
f_int = 102.400 Hz
t_int = 9,766E-06 s

Software-UART Timer

Baudrate = 9600 Bit/s
Oversampling = 8
f_int = 76.800 Hz
t_int = 1,302E-05 s

Compare Match Value

prescaler p	f_cpu [Hz] 1843200	3686400	11059200	14745600	18432000
1	18	36	108	144	180
8	2	5	14	18	23
32	1	1	3	5	6
64	0	1	2	2	3
128	0	0	1	1	1
256	0	0	0	1	1
1024	0	0	0	0	0

Berechnung Basiswiderstand

	Ubat		Usys	
Umax =	4,2 V		**Umax =**	4,0 V
Utyp =	3,7 V		**Utyp =**	3,5 V
Umin =	3,0 V		**Umin =**	2,8 V
Udiode =	0,2 V			
n LEDs =	4			

	Red	**Blue**	**Green**	**SUMME**
DC Current Ic =	0,02	0,02	0,02	0,06 A
Typ. fwd. Voltage Vf,typ =	2,1	3,2	3,2	V
Max. fwd. Voltage Vf,max =	2,5	3,8	3,8	V
Ic_ges =	0,08	0,08	0,08	0,06 A

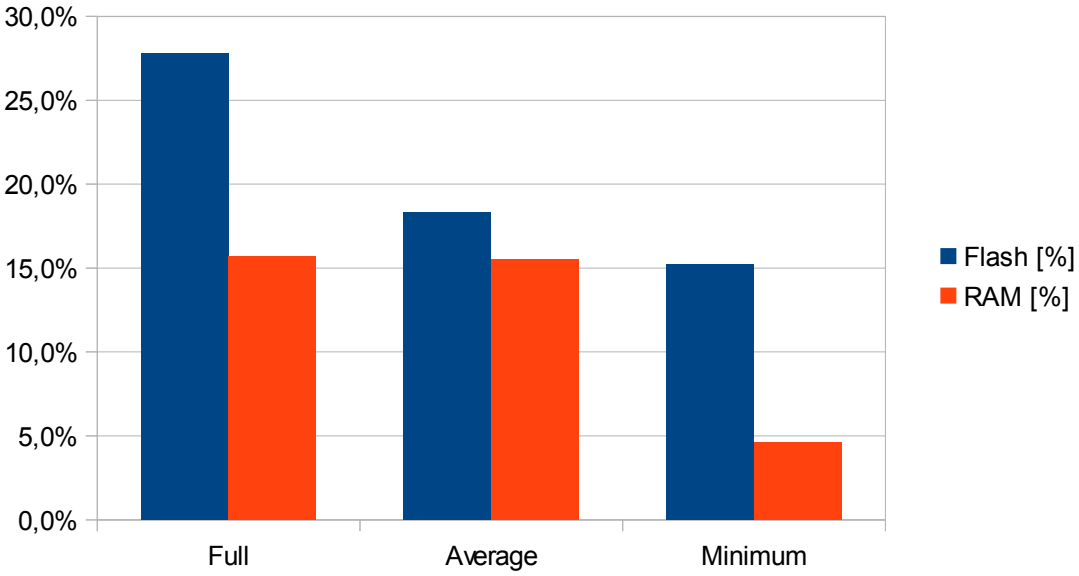
Rpulldown =	100.000 Ohm			
Ipulldown =	2,8	-	4 µA	
Npulldow =	7			
Igespulldown =	19,6	-	28 µA	
Hfesat =	10	-	50	
Hfesat,arm =	30			
Ib =	2,67	2,67	2,67	2,00 mA
Ib,pulldown =	2,67	2,67	2,67	2,00 mA
Ube =	0,7 V			
Rb,min =	0,79	0,79	0,79	1,05 kOhm
Rb,max =	1,24	1,24	1,24	1,65 kOhm
Rb =	**1,00**	**1,00**	**1,00**	**1,00 kOhm**
Rvor,typ =	95,00	40,00	40,00	Ohm
Rvor,min =	75,00	10,00	10,00	Ohm
Rvor =	**82,00**	**33,00**	**33,00**	**Ohm**

GM15 Current Power Test

Uq [V]	Im [mA]	Pm [mW]
0,5	22,0	11,0
0,6	25,0	15,0
0,7	28,0	19,6
0,8	30,0	24,0
0,9	32,0	28,8
1,0	35,0	35,0
1,1	37,0	40,7
1,2	39,0	46,8
1,3	41,0	53,3
1,4	43,0	60,2
1,5	44,0	66,0
1,6	44,5	71,2
1,7	45,5	77,4
1,8	47,0	84,6
1,9	48,0	91,2
2,0	49,0	98,0
2,1	50,0	105,0
2,2	53,0	116,6
2,3	56,5	130,0
2,4	50,0	120,0
2,5	65,0	162,5
2,6	70,0	182,0
2,7	75,0	202,5
2,8	79,0	221,2
2,9	83,0	240,7
3,0	87,5	262,5
3,1	92,0	285,2
3,2	97,0	310,4
3,3	102,5	338,3
3,4	107,5	365,5
3,5	113,0	395,5
3,6	123,0	442,8
3,7	129,0	477,3
3,8	130,0	494,0
3,9	134,0	522,6
4,0	138,5	554,0
4,1	142,5	584,3

Code Level vs. Program Size

Mode	Flash [bytes]	RAM [bytes]	Flash [%]	RAM [%]
Full	4.548	161	27,8%	15,7%
Average	2.998	159	18,3%	15,5%
Minimum	2.494	47	15,2%	4,6%



Anhang B: Bauteilbewertungen

Bauteilverfügbarkeit

Document: mrShark 10
Last update: 18 August 2013

Nr.	Anzahl	Bauteilwerte	Wert	Gehäuse	Anzahl lagernd	Multiplikator	Punkte	Fehlend
1	150	CON_PROG	DF12(5.0)-20DP-0.5V	DF12(5.0)-20DP-0.5V	1.312	1,00	100	0
2	300	D1, D2	PMEG2010EA	SOD323-R	4.319	1,00	100	0
3	150	IC1	ATMEGA168V-10MU	MLF32	783	0,50	50	0
4	150	IC2	PCA9306DC	VSSOP8	2.885	1,00	100	0
5	150	IC3	TIR1000PWR	PW_R-PDSO-G08	1.962	0,50	50	0
6	150	IC4	TFBS4711TR1	TFBS4711	960	1,00	100	0
7	300	IC5, IC7	DRV8830DRCT	S-PDSO-N10	451	1,00	100	0
8	300	IC6, IC8	LTC2990IMS#PBF	SOP50P490X110-10N	464	0,50	50	0
9	150	LED1	green	CHIPLED_0603	6.924	1,00	100	0
10	600	LED2, LED3, LED4, LED5	ASMT-YTD2-0BB02	PLCC6K	13.391	1,00	100	0
11	150	LED6	red	CHIPLED_0603	6.741	1,00	100	0
12	150	Q1	7X-1.8432MBB-T	3.2x2.5mm	202	1,00	100	0
13	1050	Q2, Q3, Q4, Q5, Q6, Q7, Q8	BC817W	SOT323-BEC	7.609	1,00	100	0
14	150	Q9	11.0592MHZ	5.0x3.2mm	942	1,00	100	0
15	300	R29, R31	0.95R	R0603	4.564	1,00	100	0
16	300	R30, R32	0R2	R0603	21.950	1,00	100	0
17	150	SWITCH_PWR	AYZ0202AGRLC	AYZ0202AGLRC	6.347	1,00	100	0
						GESAMT =	**91 Punkte**	
						PROZENT =	**91%**	

Layoutbewertung

Document: mrShark 10
Last update: 18 August 2013

Nr.	Anzahl	Bauteilwerte	Wert	Gehäuse	Anzahl Top	Anzahl Bottom	Anzahl Ausrichtungen	Multiplikator	Punkte	MINOR	MAIOR
1	300	C1, C2	100n	C0201	0	2	1	1,00	100	0	2
2	300	C3, C4	18p	C0201	0	2	1	1,00	100	0	2
3	150	CON_PROG	DF12(5.0)-20DP-0.5V	DF12(5.0)-20DP-0.5V	1	0	1	1,00	100	0	1
4	300	D1, D2	PMEG2010EA	SOD323-R	2	0	1	1,00	100	0	2
5	150	IC1	ATMEGA168V-10MU	MLF32	0	1	1	1,00	100	0	1
6	150	IC2	PCA9306DC	VSSOP8	1	0	1	1,00	100	0	1
7	150	IC3	TIR1000PWR	PW_R-PDSO-G08	1	0	1	1,00	100	0	1
8	150	IC4	TFBS4711TR1	TFBS4711	1	0	1	1,00	100	0	1
9	300	IC5, IC7	DRV8830DRCT	S-PDSO-N10	2	0	2	0,75	75	0	2
10	300	IC6, IC8	LTC2990IMS#PBF	SOP50P490X110-10N	2	0	2	0,75	75	0	2
11	150	LED1	green	CHIPLED_0603	1	0	1	1,00	100	0	1
12	600	LED2, LED3, LED4, LED5	ASMT-YTD2-0BB02	PLCC6K	0	4	4	0,25	25	0	4
13	150	LED6	red	CHIPLED_0603	1	0	1	1,00	100	0	1
14	150	Q1	FXO-HC536R-1.8432	3.2x2.5mm	1	0	1	1,00	100	0	1
15	1050	Q2, Q3, Q4, Q5, Q6, Q7, Q8	BC817W	SOT323-BEC	0	7	2	0,75	75	0	7
16	150	Q9	11.0592MHZ	5.0x3.2mm	0	1	1	1,00	100	0	1
17	1950	R1, R2, R3, R4, R5, R9, R14, R15, R16, R17, R18, R23, R24	100k	R0201	5	8	3	0,50	31	5	8
18	900	R6, R21, R22, R27, R28, R33	130R	R0201	2	4	3	0,50	25	2	4
19	600	R7, R8, R34, R35	1k8	R0201	4	0	2	0,75	75	0	4
20	150	R10	27R	R0201	1	0	1	1,00	100	0	1
21	300	R11, R13	2k	R0201	0	2	1	1,00	100	0	2
22	150	R12	1k	R0201	0	1	1	1,00	100	0	1
23	600	R19, R20, R25, R26	560R	R0201	0	4	2	0,75	75	0	4
24	300	R29, R31	0.95R	R0201	2	0	1	1,00	100	0	2
25	300	R30, R32	0R2	R0201	2	0	1	1,00	100	0	2
26	150	SWITCH_PWR	AYZ0202AGRLC	AYZ0202AGLRC	1	0	1	1,00	100	0	1
								GESAMT =	**87 Punkte**		
								PROZENT =	**87%**		

Anhang C: Steuerbefehle

Protocol Version 2.0

nr	hex	char	command	nr	hex	char	command	nr	hex	char	command	nr	hex	char	command
0	0x00		RESERVED	65	0x41	A	LEFT FWD	129	0x81			192	0xC0		
3	0x03			66	0x42	B	LEFT BWD	130	0x82			195	0xC3		
5	0x05			68	0x44	D	RIGHT FWD	132	0x84			197	0xC5		
6	0x06			71	0x47	G	RIGHT BWD	135	0x87			198	0xC6		
9	0x09			72	0x48	H	STOP	136	0x88			201	0xC9		
10	0x0A			75	0x4B	K	SET ID	139	0x8B			202	0xCA		
12	0x0C			77	0x4D	M	LEDS ON	141	0x8D			204	0xCC		
15	0x0F			78	0x4E	N	LEDS OFF	142	0x8E			207	0xCF		
17	0x11			80	0x50	P	PROTOCOL VERSION	144	0x90			209	0xD1		
18	0x12			83	0x53	S		147	0x93			210	0xD2		
20	0x14			85	0x55	U	LED STAT ON	149	0x95			212	0xD4		
23	0x17			86	0x56	V	LED STAT OFF	150	0x96			215	0xD7		
24	0x18			89	0x59	Y	TRANSMISSION END	153	0x99			216	0xD8		
27	0x1B			90	0x5A	Z	ROBOT ID	154	0x9A			219	0xDB		
29	0x1D			92	0x5C	\		156	0x9C			221	0xDD		
30	0x1E			95	0x5F	_		159	0x9F			222	0xDE		
33	0x21	!		96	0x60	`		160	0xA0			225	0xE1		
34	0x22	“		99	0x63	c	LED1 RED	163	0xA3			226	0xE2		
36	0x24	$		101	0x65	e	LED1 GREEN	165	0xA5			228	0xE4		
39	0x27	'		102	0x66	f	LED1 BLUE	166	0xA6			231	0xE7		
40	0x28	(		105	0x69	i	LED2 RED	169	0xA9			232	0xE8		
43	0x2B	+	SECTION SEPERATOR	106	0x6A	j	LED2 GREEN	170	0xAA			235	0xEB		
45	0x2D	-		108	0x6C	l	LED2 BLUE	172	0xAC			237	0xED		
46	0x2E	.	SEPERATOR	111	0x6F	o	LED3 RED	175	0xAF			238	0xEE		
48	0x30	0		113	0x71	q	LED3 GREEN	177	0xB1			240	0xF0		
51	0x33	3		114	0x72	r	LED3 BLUE	178	0xB2			243	0xF3		
53	0x35	5		116	0x74	t	LED4_RED	180	0xB4			245	0xF5		
54	0x36	6		119	0x77	w	LED4_GREEN	183	0xB7			246	0xF6		
57	0x39	9		120	0x78	x	LED4_BLUE	184	0xB8			249	0xF9		
58	0x3A	:		123	0x7B	{	LED ON	187	0xBB			250	0xFA		
60	0x3C	<		125	0x7D	}	LED OFF	189	0xBD			252	0xFC		
63	0x3F	?		126	0x7E	~		190	0xBE			255	0xFF		

mrShark Debugging Commands

Command	Description	Command	Description
f	Both motors forward with half speed	M	Do motor test an return result
b	Both motors backward with half speed	T	Return internal temperature values
z	Brake both motors	U	Return voltage 1
g	All RGB-LEDs on (white color)	V	Return voltage 2
h	All RGB-LEDs off	W	Return voltage BAT1
s	Status LED on	X	Return voltage BAT2
r	Status LED off	I	Return current left motor
c	Turn clock wise with half speed	J	Return current right motor
a	Turn counter clock wise with half speed	A	Return supply voltages
m	Return current left motor speed		
n	Return current right motor speed		
o	Return current left motor command		
p	Return current right motor command		
i	Return current bot id		
j	Return firmware publisher		
k	Return firmware version		
l	Return firmware product name		
d	Return left motor faults		
e	Return right motor faults		
u	Clear left motor faults		
v	Clear right motor faults		

Anhang D: Signalverläufe

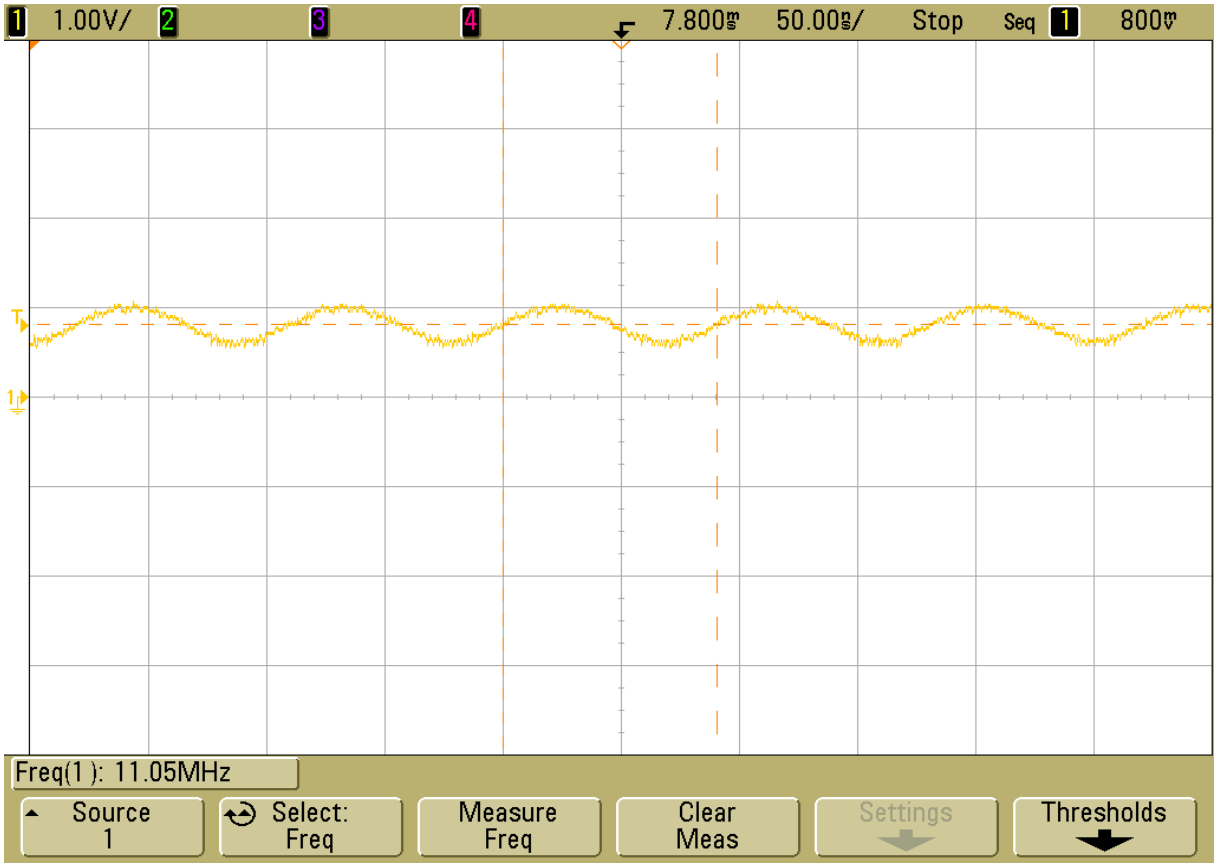


Abbildung 15: Signalverlauf Systemtakt

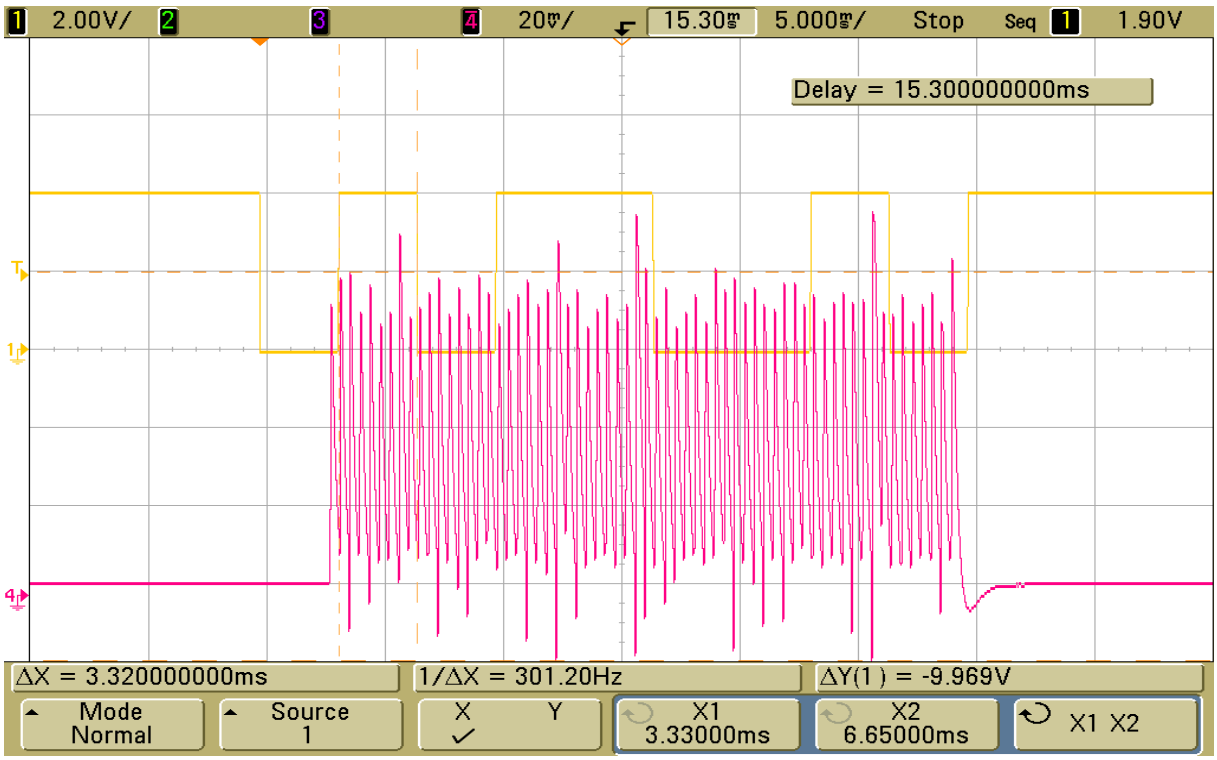


Abbildung 16: 10 Bit UART RX Daten (gelb) und Oversampling (rot), Baudrate = 300 Bits/s, 8N1

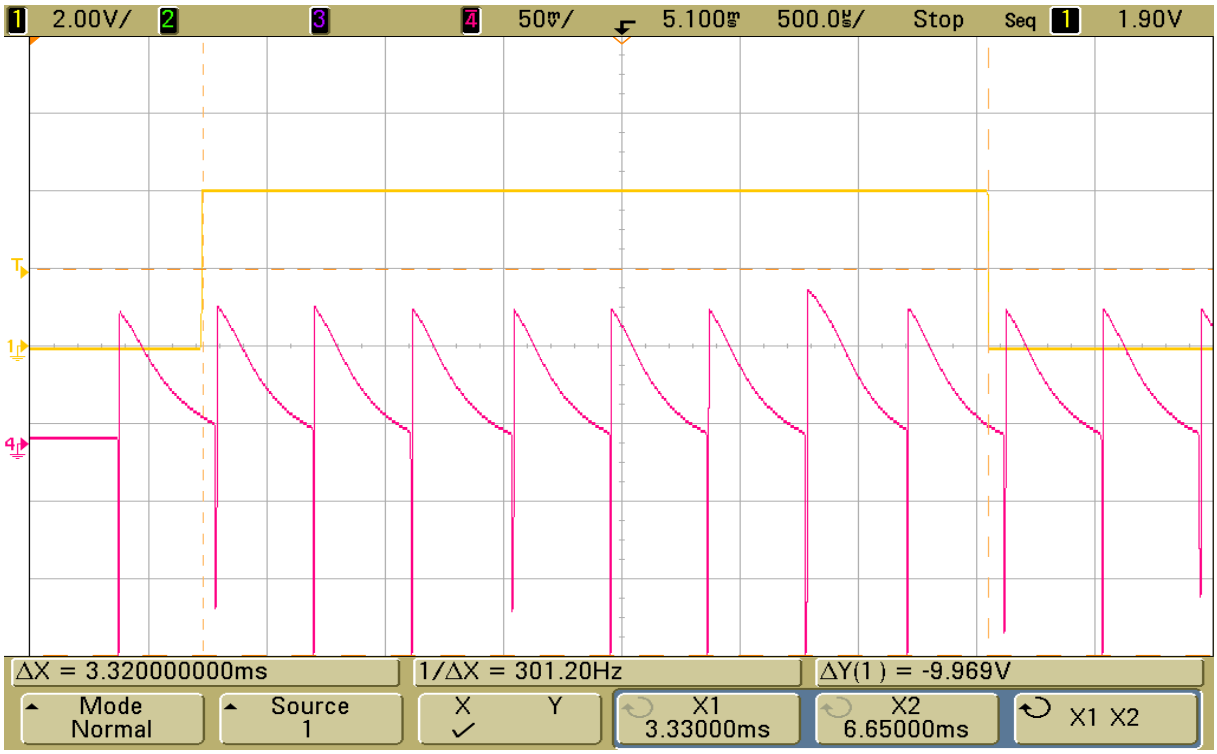


Abbildung 17: 1 Bit UART RX Daten (gelb) und Oversampling (rot), Baudrate = 300 Bit/s, 8N1

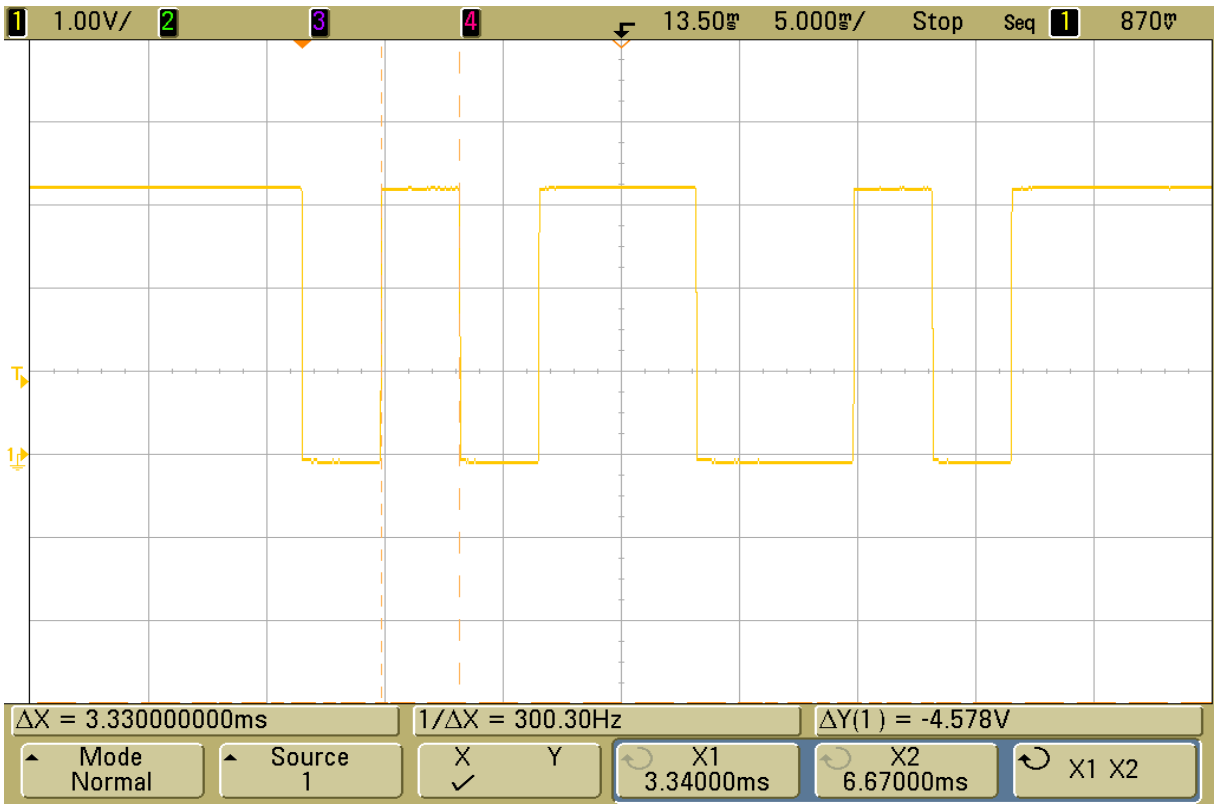


Abbildung 18: UART TX Daten (0x4d), Baudrate 300 Bit/s, 8N1

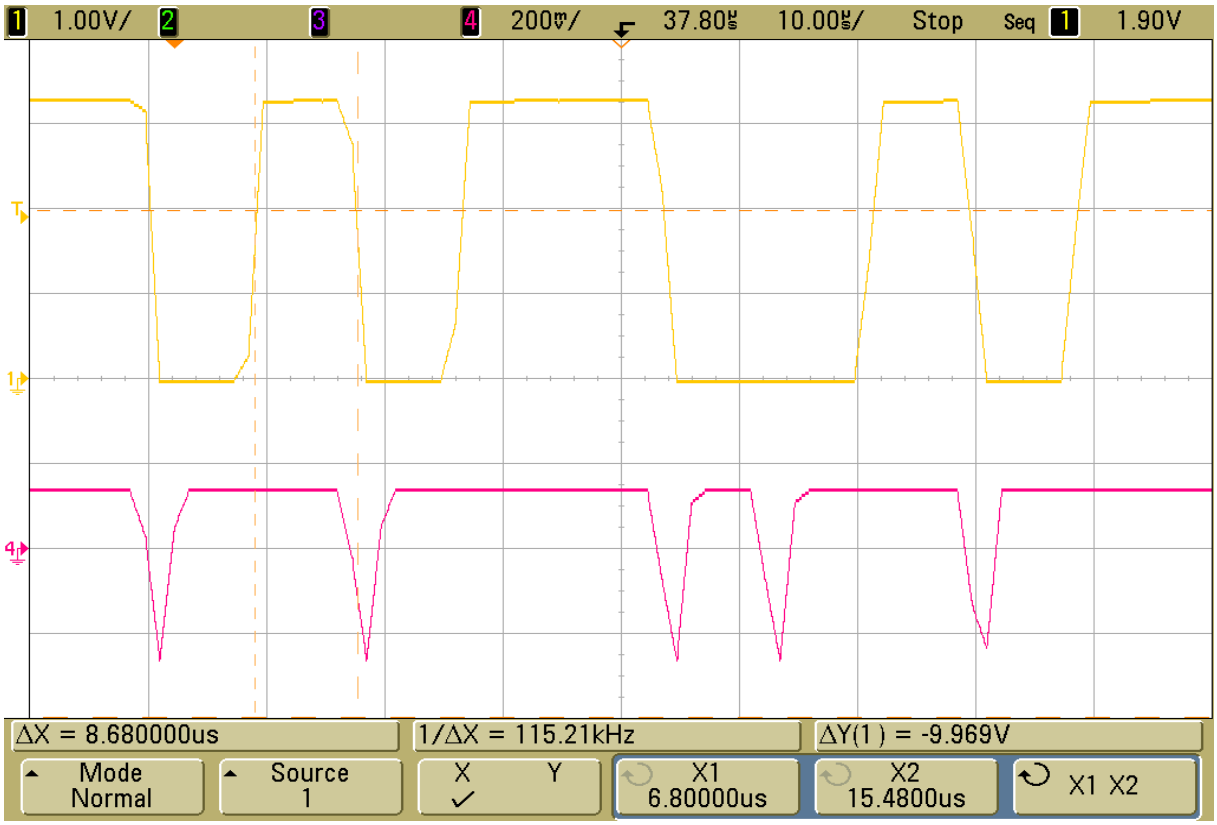


Abbildung 19: UART RX Daten (gelb) und IR Daten (rot), Baudrate = 115,2 kBit/s, 8N1

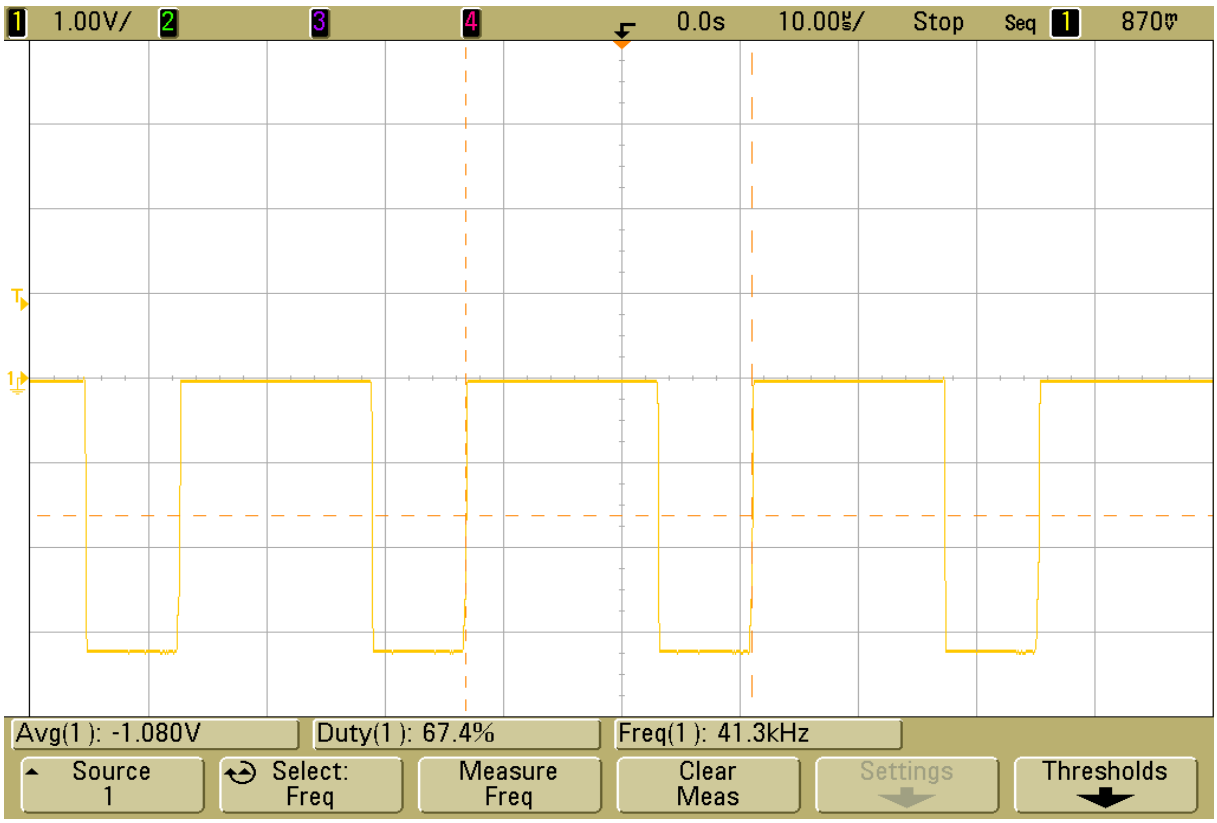


Abbildung 20: Motor Control PWM, forward, speed = 39%

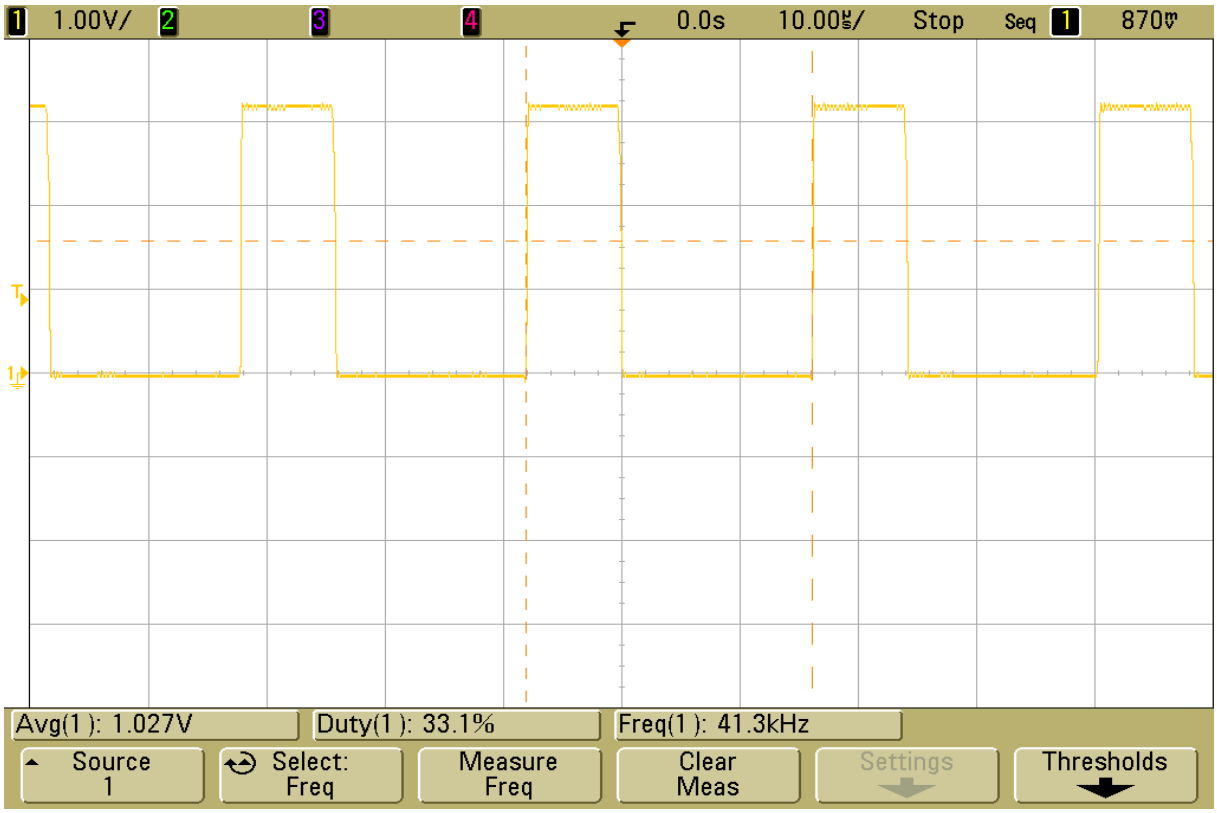


Abbildung 21: Motor Control PWM, backward, speed = 39%

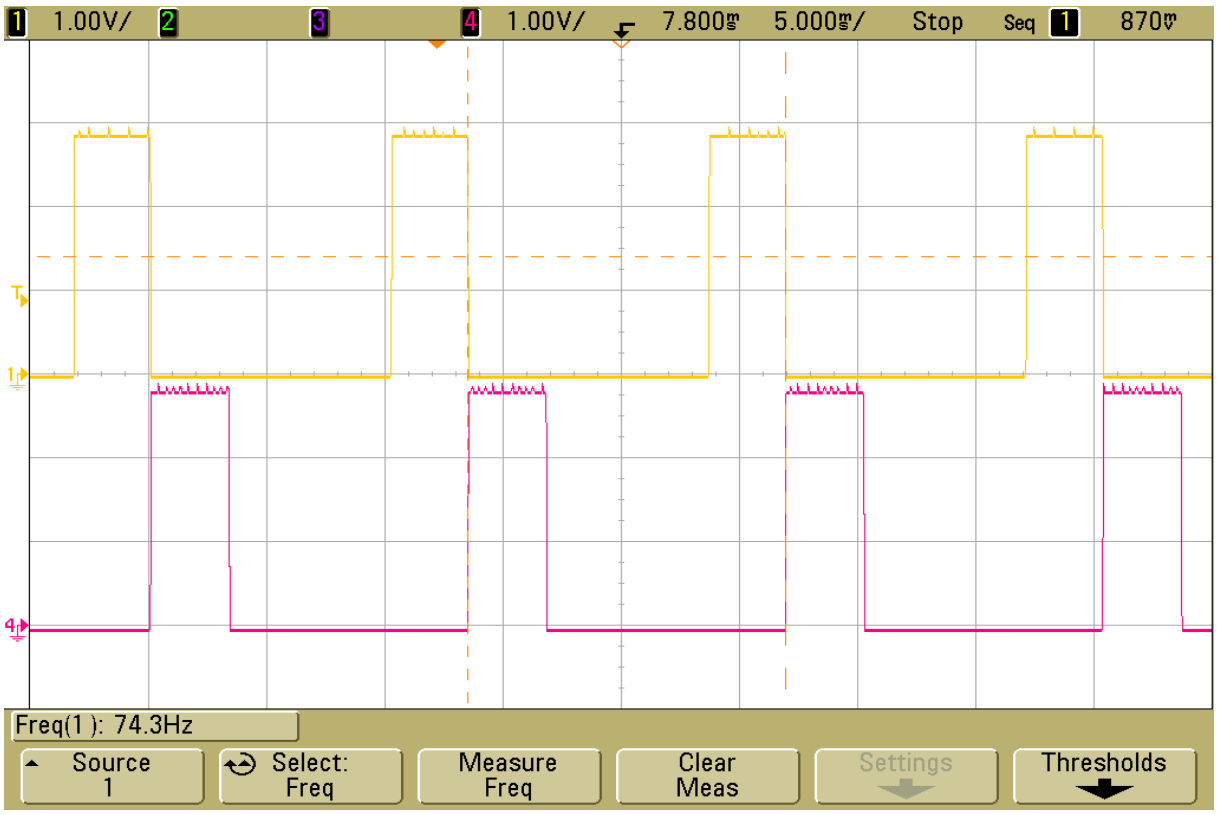


Abbildung 22: RGB-LED 1 (gelb) und 2 (rot) Control Signal, Compare Match Value = 0x6c

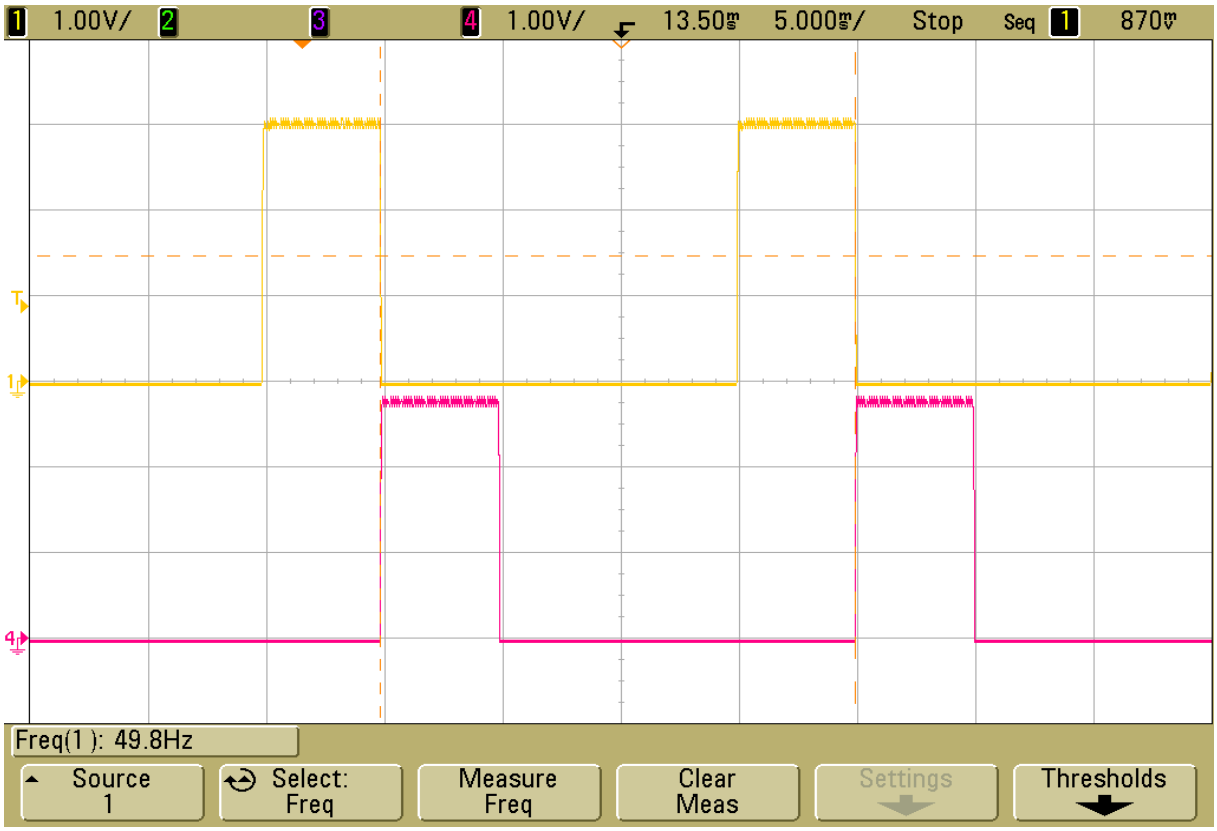
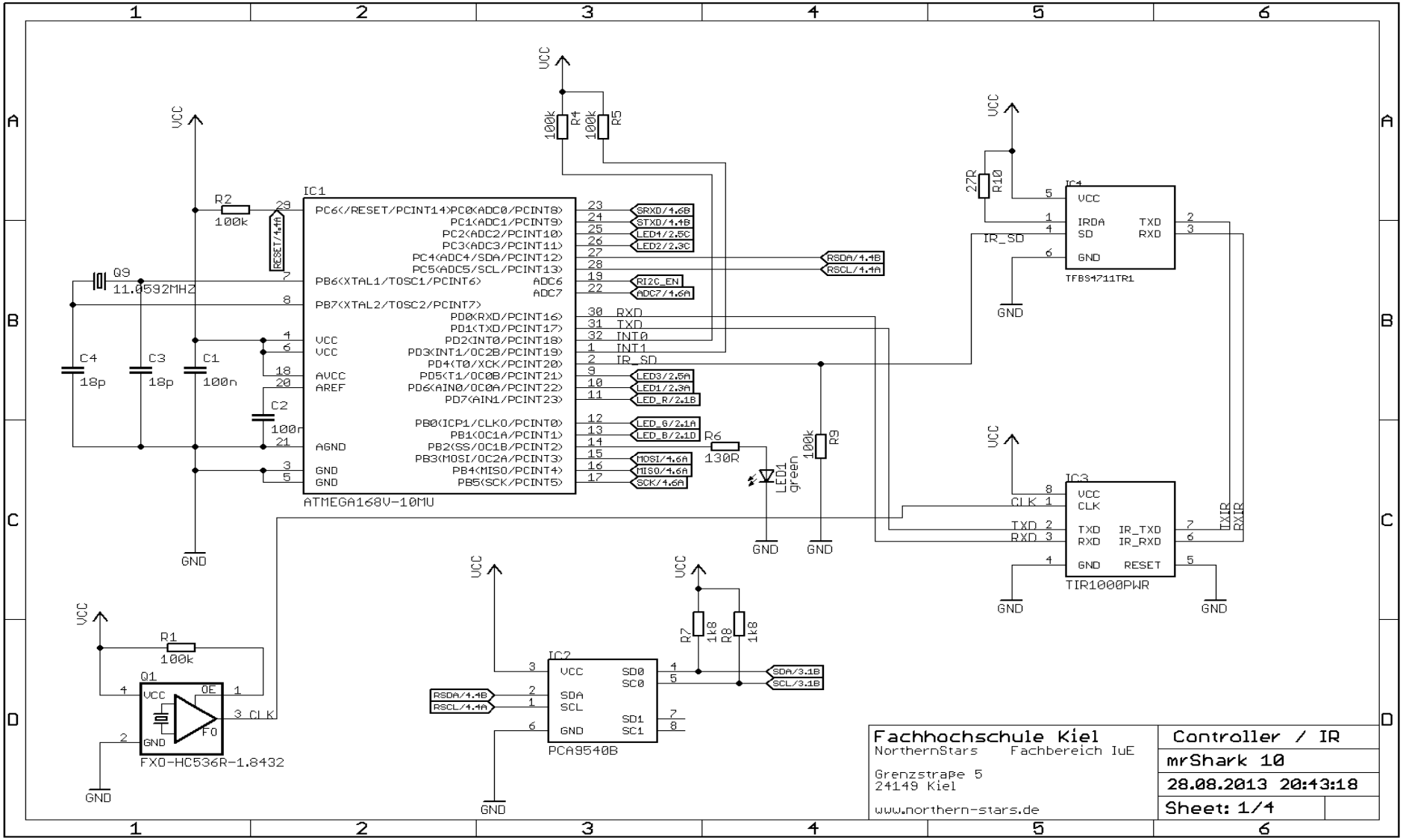
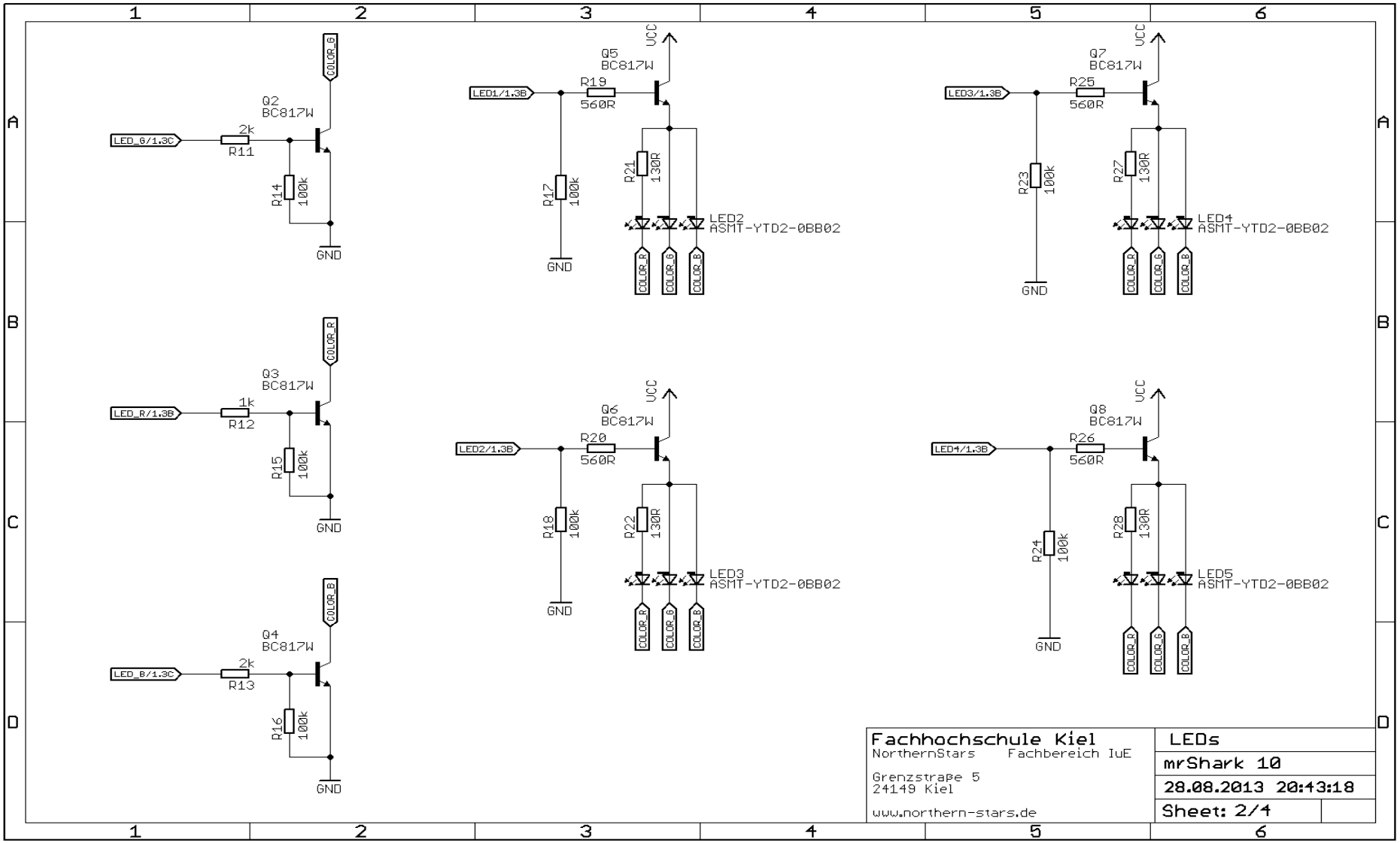
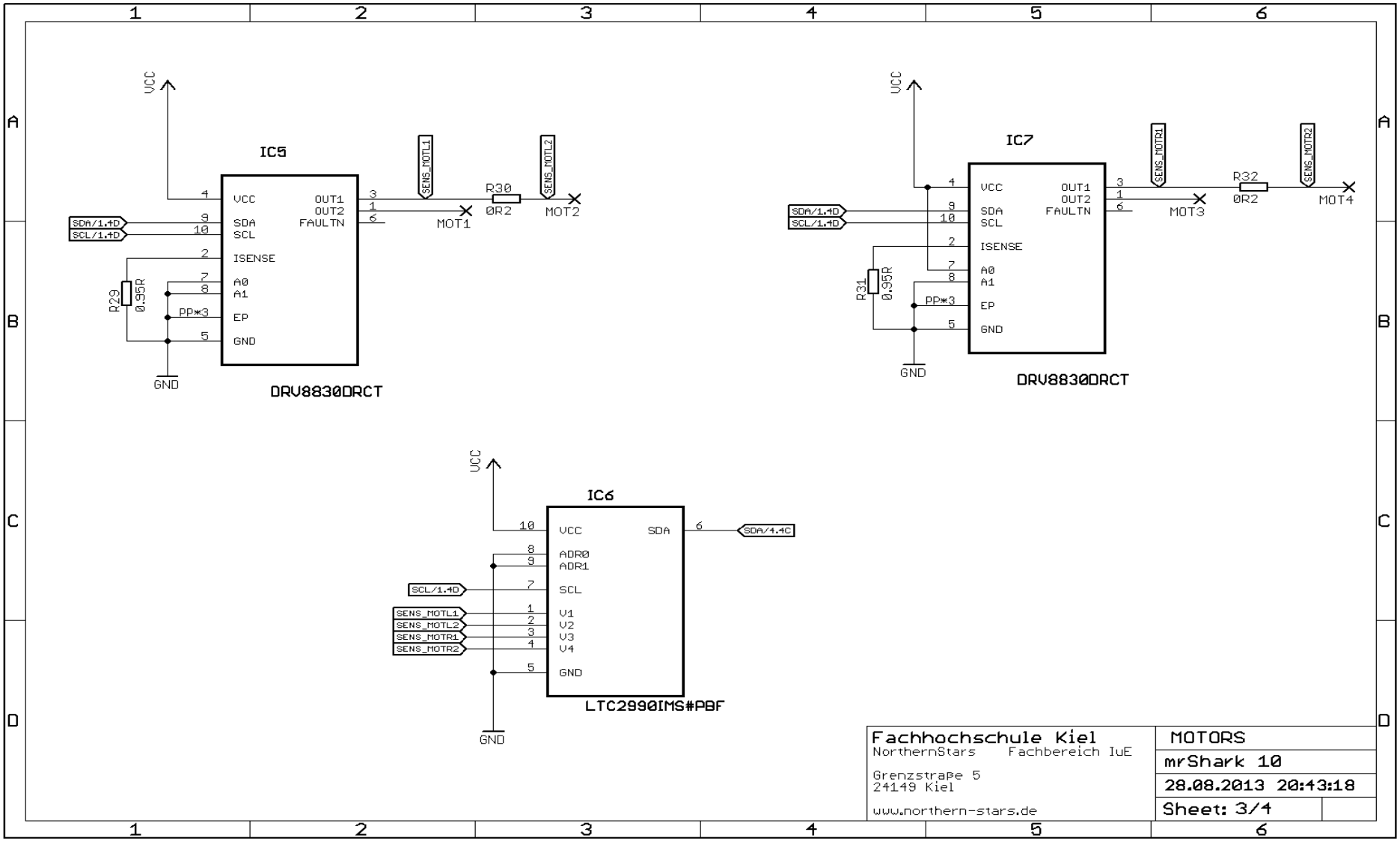


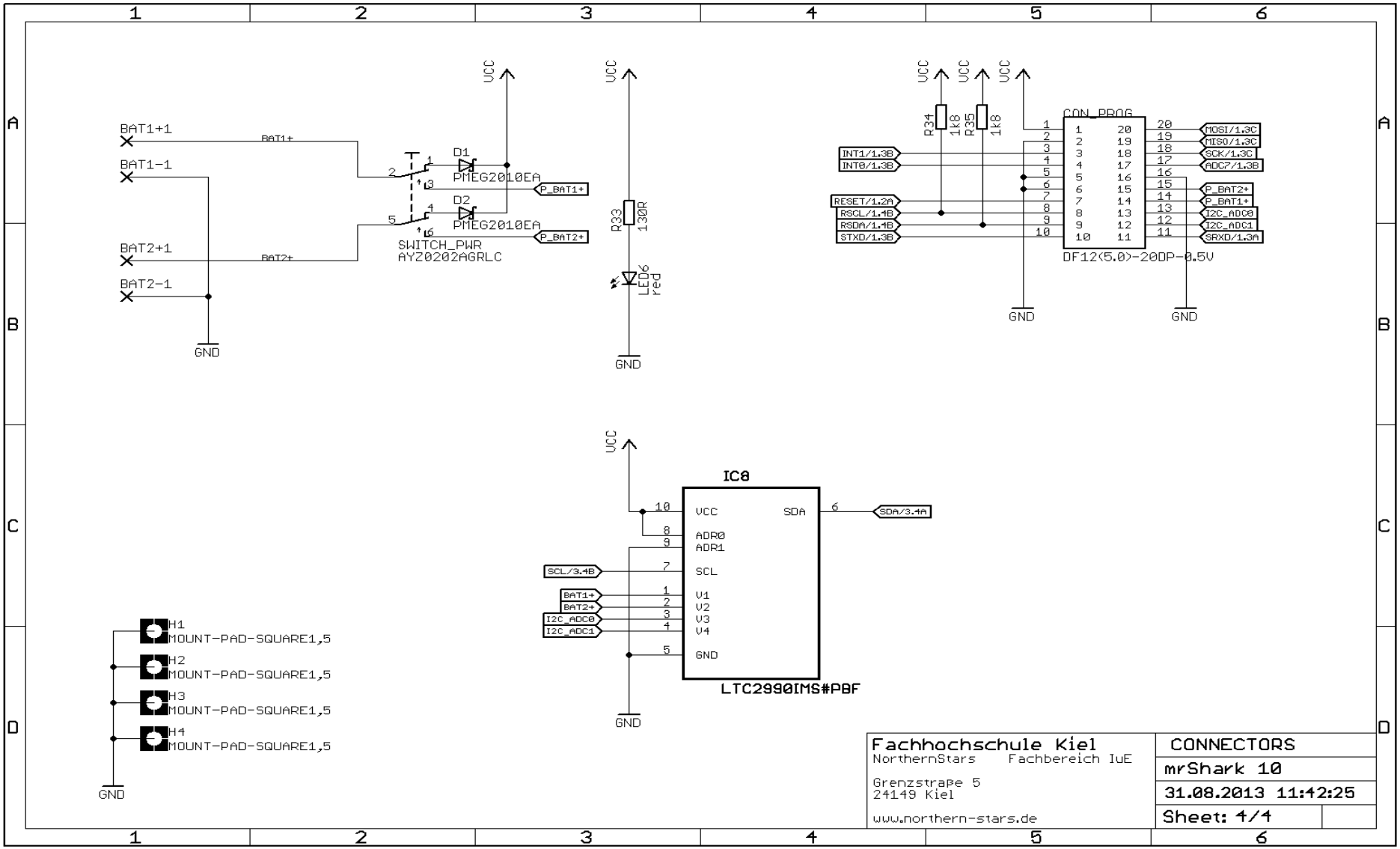
Abbildung 23: RGB-LED 1 (gelb) und 2 (rot) Control Signal, COmpare Match Value = 0xd8

Anhang E: Schaltplan mrShark


IC1
ATMEGA168V-10MU
PC6(/RESET/PCINT14)
PC0(ADC0/PCINT8)
PC1(ADC1/PCINT9)
PC2(ADC2/PCINT10)
PC3(ADC3/PCINT11)
PC4(ADC4/SDA/PCINT12)
PC5(ADC5/SCL/PCINT13)
PB6(XTAL1/TOSC1/PCINT6)
PB7(XTAL2/TOSC2/PCINT7)
PD0(RXD/PCINT16)
PD1(TXD/PCINT17)
PD2(INT0/PCINT18)
PD3(INT1/OC2B/PCINT19)
PD4(T0/XCK/PCINT20)
PD5(T1/OC0B/PCINT21)
PD6(AIN0/OC0A/PCINT22)
PD7(AIN1/PCINT23)
PB0(ICP1/CLKO/PCINT0)
PB1(OC1A/PCINT1)
PB2(SS/OC1B/PCINT2)
PB3(MOSI/OC2A/PCINT3)
PB4(MISO/PCINT4)
PB5(SCK/PCINT5)
ADC6
ADC7
AVCC
AREF
AGND
VCC
GND
RESET/4.4A
SRXD/4.6B
STXD/4.4B
LED4/2.5C
LED2/2.3C
RSDA/4.4B
RSCL/4.4A
RI2C_EN
ADC7/4.6A
RXD
TXD
INT0
INT1
IR_SD
LED3/2.5A
LED1/2.3A
LED_R/2.1B
LED_G/2.1A
LED_B/2.1D
MOSI/4.6A
MISO/4.6A
SCK/4.6A
R2 100k
R4 100k
R5 100k
R6 130R
R9 100k
C1 100n
C2 100n
C3 18p
C4 18p
Q9 11.0592MHZ
LED1 green
IC4
TFBS4711TR1
VCC
IRDA
SD
TXD
RXD
GND
R10 27R
IR_SD
IC3
TIR1000PWR
CLK
TXD
RXD
IR_TXD
IR_RXD
RESET
TXIR
RXIR
IC2
PCA9540B
SD0
SC0
SD1
SC1
SDA
SCL
R7 1k8
R8 1k8
SDA/3.1B
SCL/3.1B
Q1
FXO-HC536R-1.8432
OE
FO
R1 100k
Fachhochschule Kiel
NorthernStars Fachbereich IuE
Grenzstraße 5
24149 Kiel
www.northern-stars.de
Controller / IR
mrShark 10
28.08.2013 20:43:18
Sheet: 1/4


1
2
3
4
5
6
A
B
C
D
Q2
BC817W
2k
R11
R14
100k
LED_G/1.3C
COLOR_G
GND
Q5
BC817W
VCC
LED1/1.3B
R19
560R
R17
100k
R21
130R
LED2
ASMT-YTD2-0BB02
COLOR_R
COLOR_G
COLOR_B
Q7
BC817W
LED3/1.3B
R25
560R
R23
100k
R27
130R
LED4
ASMT-YTD2-0BB02
Q3
BC817W
1k
R12
R15
100k
LED_R/1.3B
COLOR_R
Q6
BC817W
LED2/1.3B
R20
560R
R18
100k
R22
130R
LED3
ASMT-YTD2-0BB02
Q8
BC817W
LED4/1.3B
R26
560R
R24
100k
R28
130R
LED5
ASMT-YTD2-0BB02
Q4
BC817W
2k
R13
R16
100k
LED_B/1.3C
COLOR_B
Fachhochschule Kiel
NorthernStars Fachbereich IuE
Grenzstraße 5
24149 Kiel
www.northern-stars.de
LEDs
mrShark 10
28.08.2013 20:43:18
Sheet: 2/4


1
2
3
4
5
6
A
B
C
D
VCC
IC5
SENS_MOTL1
SENS_MOTL2
R30
0R2
MOT2
MOT1
SDA/1.4D
SCL/1.4D
VCC
SDA
SCL
ISENSE
A0
A1
PP*3
EP
GND
OUT1
OUT2
FAULTN
R29
0.95R
GND
DRV8830DRCT
IC7
SENS_MOTR1
SENS_MOTR2
R32
0R2
MOT4
MOT3
R31
0.95R
DRV8830DRCT
IC6
SDA/4.4C
ADR0
ADR1
SCL/1.4D
SENS_MOTL1
SENS_MOTL2
SENS_MOTR1
SENS_MOTR2
V1
V2
V3
V4
LTC2990IMS#PBF
Fachhochschule Kiel
NorthernStars Fachbereich IuE
Grenzstraße 5
24149 Kiel
www.northern-stars.de
MOTORS
mrShark 10
28.08.2013 20:43:18
Sheet: 3/4


BAT1+1
BAT1-1
BAT2+1
BAT2-1
BAT1+
BAT2+
GND
VCC
D1
PMEG2010EA
D2
PMEG2010EA
P_BAT1+
P_BAT2+
SWITCH_PWR
AYZ0202AGRLC
R33
130R
LED6
red
R34
1k8
R35
1k8
CON_PROG
DF12(5.0)-20DP-0.5V
INT1/1.3B
INT0/1.3B
RESET/1.2A
RSCL/1.4B
RSDA/1.4B
STXD/1.3B
MOSI/1.3C
MISO/1.3C
SCK/1.3C
ADC7/1.3B
I2C_ADC0
I2C_ADC1
SRXD/1.3A
IC8
LTC2990IMS#PBF
VCC
SDA
ADR0
ADR1
SCL
V1
V2
V3
V4
SDA/3.4A
SCL/3.4B
H1
H2
H3
H4
MOUNT-PAD-SQUARE1,5
Fachhochschule Kiel
NorthernStars Fachbereich IuE
Grenzstraße 5
24149 Kiel
www.northern-stars.de
CONNECTORS
mrShark 10
31.08.2013 11:42:25
Sheet: 4/4

Anhang F: Platinenlayout mrShark

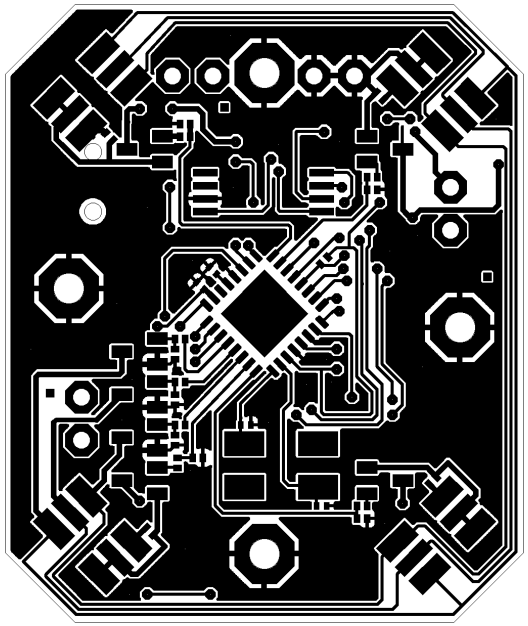


Abbildung 24: Platinenlayout mrShark Layer Bottom

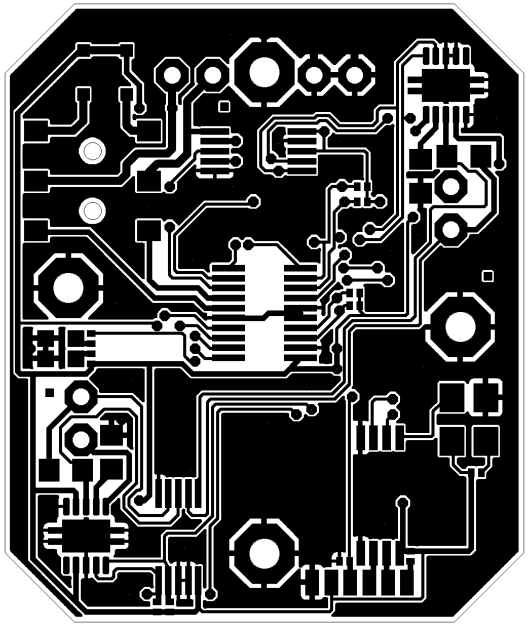


Abbildung 25: Platinenlayout mrShark Layer Top

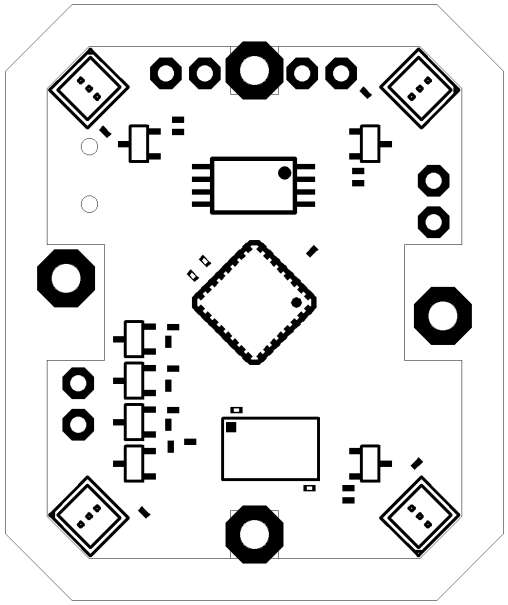


Abbildung 26: Bestückungsplan mrShark Bottom

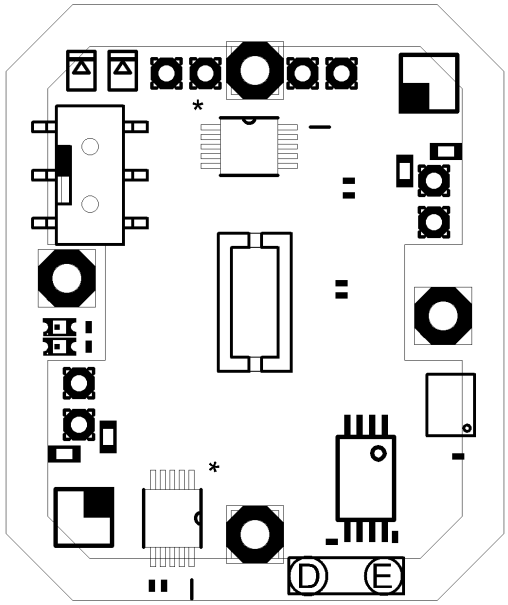
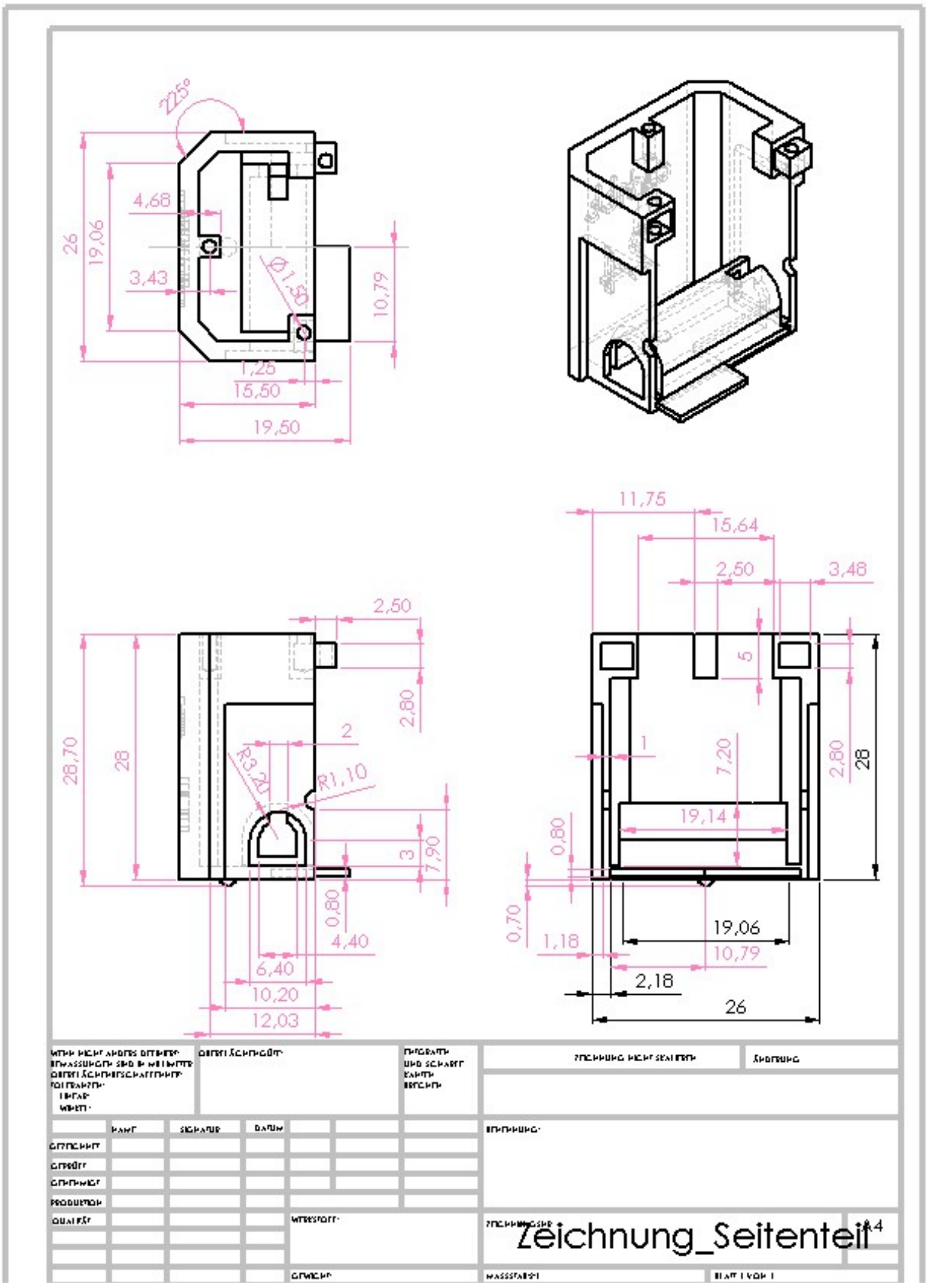


Abbildung 27: Bestückungsplan mrShark Top

Anhang G: Vorgaben Gehäuse mrShark



Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Weiterhin versichere ich, dass diese Arbeit noch nicht als Abschlussarbeit an anderer Stelle vorgelegen hat.

Datum, Unterschrift