Projektgruppe Smart Systems 3: Submarine Exploring

Endbericht

Dr. Frank Slomka
Frank Bodmann
Matthias Behrens
Claas Diederichs
Jörg Gesen
Henning Hoffmann
Christian Myrtus
Daniel Schulte
Robert Tunnell
Gibeon Wijlaars

11. Januar 2007

Inhaltsverzeichnis

Kapitel 1

Einleitung

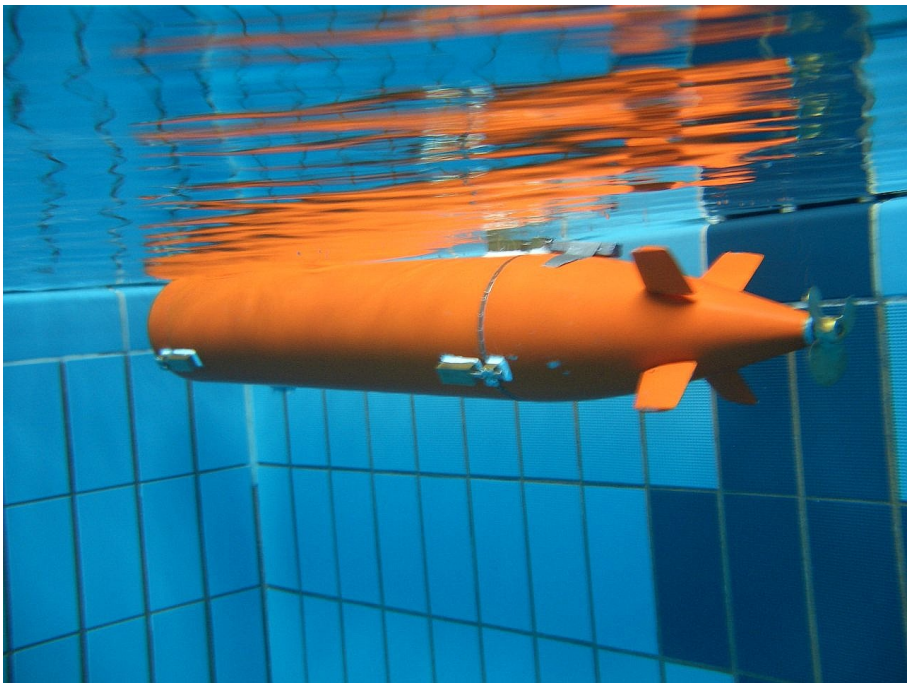


Abbildung 1.1: Das Uboot unter Wasser

1.1 Motivation

In dieser Projektgruppe soll ein autonom fahrender Unterwasserroboter, hier Uboot genannt, gebaut werden. Die Aufteilung erfolgt in diesen Themenbereichen:

1. Rumpfbau
2. Inertialnavigation
3. Hardware Plattform
4. Regelung
5. Hardware - Software Architektur
6. Kommunikation
7. Hostsystem

1.2 Danksagungen

Folgen Institutionen und Personen wollen wir danken für ihre Unterstützung:

- H. Behrens GmbH & Co. KG, Am Steenöver 15, 27777 Ganderkesee
- Internationaler Modell-U-Boot Verein SONAR e.V
- Alcatel SEL AG Components Division Dunkermotoren, Allmendstrasse 11, 79848 Bonndorf/ Schwarzwald

Kapitel 2

Rumpfbau

Abbildung 2.1: Forschungsuboot

2.1 Motivation und Konzeptfindung

Das Ziel unserer Projektgruppe ist es, einen autonomen Tauchroboter zu bauen, der in einem Schwimmbad operieren soll. Seine Aufgabe besteht darin, Gegenstände auf dem Grund zu erkennen, zu kartographieren und diese einzeln anzufahren. Desweiteren soll es möglich sein Fotos und Videos vom Roboter aus zu machen und diese zu übertragen. Wir haben uns sofort dafür entschieden den Roboter in Form eines U-Bootes zu bauen, wobei wir anfänglich viele verschiedene Formen diskutiert und in Erwägung gezogen haben.
Auf den ersten Treffen im September 2005 wurde beschlossen die Außenhülle aus PMMA (Polymethylmethacrylat), auch unter dem Handelsnamen Plexiglas bekannt, zu bauen. Ein Hauptrohr, zwei Rohre für die Gondeln, sowie Halbkugeln für Bug und Heck waren zu bekommen. Die Konstruktion mit zwei Gondeln haben wir gewählt, um eine gute Manövrierfähigkeit zu erreichen. Die Halbkugeln und das Hauptrohr sollten mit Bajonettverschlüssen aus Aluminium verbunden werden. Allerdings waren die Kosten sehr hoch. Wir haben schnell festgestellt, dass neben den Kosten dieser Vorschlag nicht die Machbarkeit bzw. spätere Probleme bei der Fertigung so wie eventuelle grundsätzliche Eigenschaften wie Strömungs- und Ansteuerverhalten berücksichtigt. Obwohl wir später einen anderen Entwurf realisiert haben, möchten wir im Folgenden die drei hauptsächlich diskutierten Grundformen vorstellen die im Laufe unserer Planungsphase in Betracht kamen.

- Komplette Steuerung über beweglichen Düsen
  Dieser Entwurf ist an ein Forschungsuboot angelehnt.

  Aufgrund der vielen nötigen Durchbrüche und den diversen Motoren und Servos und der daraus zu realisierenden komplizierten Ansteuerung, haben wir diesen Vorschlag sehr schnell verworfen. Dazu kam, dass uns der Rumpfaufbau aufgrund der vielen Ansteuerdüsen und der dazu nötigen Anschlüsse, vor ein scheinbar nicht lösbares Problem stellte.

- Torpedoform
  Bei der von uns später umgesetzten Torpedoform, kamen zwei Konzepte in die nähere Auswahl:

  1. Torpedoform mit zwei außen anliegenden Antriebsgondeln die jeweils einen Antriebsmotor und eine Korddüse zur Steuerung beinhalten sollten.
     Der eigentliche Rumpf sollte somit hauptsächlich die Technik sowie die Tauchtanks enthalten.
  2. Torpedoform mit Kreuzruder (bzw. X-Ruder) Steuerung mit Heckschraubenantrieb.
     Bei diesem Konzept befinden sich alle Technischen Komponenten sowie der Antrieb und die restlichen Komponenten im Hauptrohr.

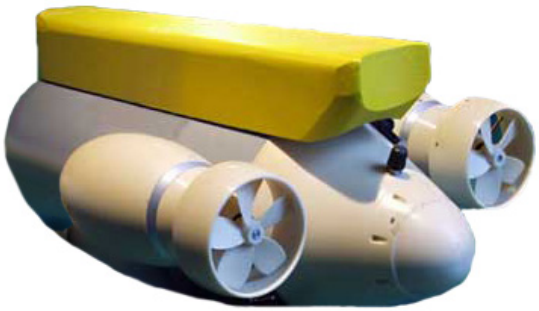


Abbildung 2.2: U-Boot mit Gondeln

Abbildung 2.3: Rohtorpedoform

Um eine Entscheidung über die Bauform zu treffen, mussten wir noch weitere Informationen einholen. Über die Recherchen im Internet und aus dem Buch [Brü92] hinaus versuchten wir Informationen von Personen zu sammeln, die schon einschlägig Erfahrungen im Modellbau gesammelt haben. Dabei mussten wir feststellen, dass gerade der Ubootmodellbau aus einer kleinen Gemeinde von Bastlern besteht. Diese wenigen Ubootmodellbauer unterteilen sich dazu noch in zwei Lager.
Zum einen diese, die nur Fertigmodelle zusammenbauen und solche die von der Erstellung des Konzeptes, Herstellung einzelner Komponenten sowie dem Zusammenbau alles selber in die Hand nehmen. Bei einem Besuch der Modellbaumesse in Hamburg im Oktober lernten wir Lothar Mentz, den Vorsitzenden des Internationalen U-Boot Modellbauclubs Sonar EV, kennen. Er riet uns von der Konstruktion der Gondeln ab und meinte, dass ein Boot auch mit einem Antrieb manövrierfähig genug sei. Anstatt des PMMA würde er ein handelsübliches PVC Abflussrohr nehmen und Bug und Heck aus GFK anfertigen. Desweiteren riet er uns vom Prinzip eines Bajonettverschlusses ab, da dieser mit einem Durchmesser von 20cm zu schwer zu öffnen und zu schließen sei. Stattdessen sollten wir einen normalen Verschluss nehmen, den wir in das Rohr drücken und dann von außen oder innen fixieren können. Wir haben uns entschlossen auf seine Vorschläge einzugehen, zumal er uns auch die Formen für Bug, Heck und auch Ruder zur Verfügung stellen wollte. Bei einem der Besuche bei Lothar Mentz haben wir das X-Ruderprinzip kennen gelernt und uns entschieden dieses auch in unser Konzept umzusetzen. Neben den von Lothar Mentz genannten Gründen haben wir auch noch weitere Gründe gehabt die einfache Torpedoform zu realisieren:

- Bei dem Gondelentwurf fallen zusätzliche Durchbrüche zu den Gondeln an, die wie wir später feststellen sollten, ziemlich viele Probleme mit sich bringen. Die Form ohne Gondeln reduziert die Durchbrüche durch den Rumpf auf ein Minimum und somit auch spätere potentielle undichte Stellen.

- Das Prinzip der Korddüse ist zwar auf den ersten Blick sehr brillant, aber bei genauerer Betrachtung stellt sich kein so viel besseres Steuerverhalten ein, als das sich der recht komplizierte Bau rentieren würde. Dies resultiert vor allem daraus,s das es sich bei der Korddüsensteuerung, im Gegensatz zu dem Steuerdüsen Prinzip eines Forschungsubootes, um eine Umleckung des Antriebstrahls handelt. Aus mechanischen und strömungstechnischen Gründen kann der Strahl der Korddüse nicht einmal um 60° umgelenkt werden, der Bewegungsgrad der Steuerdüse an sich ist jedoch nur von der Aufhängung abhängig.

- Das Strömungsverhalten eines Bootes mit Außengondeln ist schlechter.

- Die Kosten für den Bau mit zwei Antriebsgondeln sind um ein einiges höher.

- Die Steuerung mit X-Rudern hat einen höheren Wirkungsgrad als die Steuerung mit Kreuzrudern, da alle vier Ruder zu 71% wirken, anstatt das zwei Ruder zu 100% wirken. Es ist mit X-Rudern möglich das Motormoment auszugleichen und die Ruder auch zum Bremsen zu benutzen. Ein weiterer Vorteil ist, dass die Ruder nicht über die Breite und Höhe des Druckkörpers hinausragen und deshalb die Gefahr einer Beschädigung verringert wird.

2.2 Entwurf

Der Rumpf besteht aus drei Hauptteilen, Heck, Mittelteil und Bug, die später mit großen Aluminiumverschlüssen zusammen die Außenhülle bilden sollen. Am Heck befestigt, befindet sich der Schlitten der die später montierten Ansteuerplatinen sowie auch das FPGA, Akkus und sonstige Technik tragen soll. Der Schlitten wird mit dem Heck in das Mittelrohr geschoben bis der Aluminiumverschluss mit dem Rohr abschließt und mit Hilfe eines Dichtringes für einen wasserdichten Verschluss sorgt.
Bei der Konstruktion des Rumpfes und der Anordnung der Technik wurde darauf geachtet das alle Durchbrüche, so z.B. Ansaugstutzen für die Tauchtanks oder Drucksensoranschluss, im Heck aus dem Rumpf treten. Durch die Vermeidung von Durchbrüchen im Mittel und Bugteil ist es möglich das komplette Technikgerüst aus dem Rumpf zu ziehen ohne irgendwelche Anschlüsse zu trennen.
Die Außenhülle, sowie sämtliche mechanische Komponenten wurden hauptsächlich bei Maschinenbau

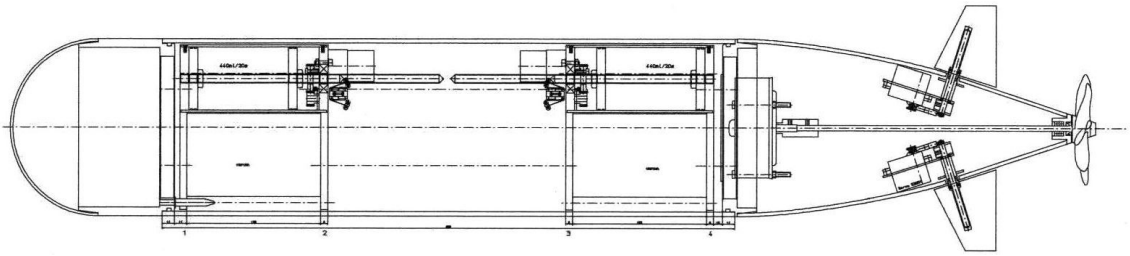


Abbildung 2.4: Zeichnung Konzept

Behrens in Schierbrook gefertigt. Bis auf die Teile die mit einer Drehbank hergestellt wurden (z. B. Aluminiumverschlüsse) und ein paar Schweißarbeiten, haben wir alles selbst angefertigt. Im folgenden Kapitel werden wir die einzelnen Komponenten sowie ihre Herstellung und einige Details beschreiben.

2.3 Einzelne Komponenten

2.3.1 Rumpf

Bei der Rumpfkonstruktion stellte sich natürlich die Frage wie groß das Hauptrohr werden soll. Diese Entscheidung sollte sehr wohl überlegt sein, um auf der einen Seite das U-Boot so klein wie möglich zu halten und auf der anderen Seite genügend Platz für die geplante Technik zu haben und auch den Freiraum zu schaffen, für eventuell später geplante Erweiterungen. Dabei ist die eigentliche Länge des Bootes nicht der entscheidende Faktor. Durch das schon vorher kurz vorgestellte Schlittenprinzip kann sowohl der Schlitten als auch der Rumpf recht schnell der gewünschten Länge angepasst werden. Viel mehr ist es entscheidend, und auch dementsprechend für uns schwieriger gewesen, den Durchmesser für unser Boot zu bestimmen. In diese Entscheidung sollten folgende Aspekte einfluss haben:

- Materialbeschaffung
- Anordnung der einzelnen Bauteile z.B. Akkus, Motor, elektronische Steuerung
- Beschaffung einer geeigneten Form zur spätere GFK-Formung von Bug und Heck

Die Abmaße des FPGAs als auch die von Lothar Mentz uns zur Verfügung gestellten Negativformen zur GFK-Formung, haben uns einen Durchmesser von 20 cm wählen lassen. Bei der späteren Integration der einzelnen Komponenten erwies sich der Einbau gerade im Heck als sehr schwierig, obwohl unsere Wahl schon auf die größte angedachte Variante von 20 cm gefallen war.

Heck und Bug

Heck- und Bugteil sind von der Form her sicherlich die am kompliziertesten herzustellenden Teile, da diese sowohl aus aerodynamischen als auch aus Stabilitätsgründen besondere Anforderungen erfüllen müssen. Bei dem Material haben wir uns für GFK entschieden, da dieses Material leicht zu modellieren ist und auch schon bei geringer Wandstärke unseren Druckanforderungen standhält.
Um einen möglichst guten CW-Wert zu erhalten ist der Bug in Form einer Halbkugel. Das Heck hat eine Zylinderform. Zur GFK-Modellierung haben wir die von Lothar Mentz zur Verfügung gestellten Formen benutzt:

Herstellung des GFK-Bugs und -Hecks

Es werden drei bis vier Schichten Trennwachs in die Bugform gestrichen. Pro Schicht werden dazwischen ca. zwei Stunden Trockenzeit benötigt. Mit einem Fön haben wir die Trockenzeiten jedoch ungefähr um

Abbildung 2.5: Buggussform und fertiges Teil

Abbildung 2.6: Heckgussform und fertiges Teil

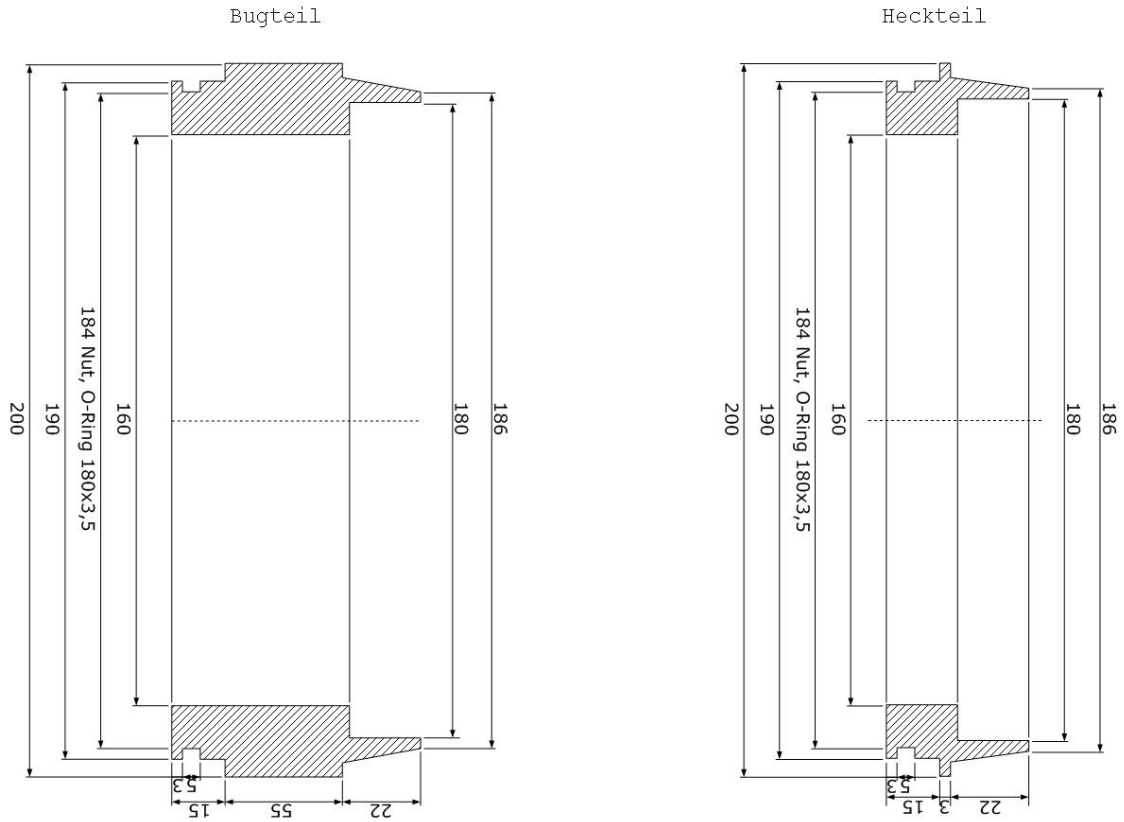


Abbildung 2.7: Zeichnung Aluminiumverschlüsse

die Hälfte verringern können. Im nächsten Schritt werden noch zwei Schichten Trennlack aufgetragen. Diese Schichten sorgen alleine dafür, dass es später möglich ist die Form mit Hilfe von Druckluft und ein paar leichten Hammerschlägen vom Guss zu trennen. Nachdem die Trennschichten alle getrocknet sind, kann mit der eigentlichen GFK-Modellierung begonnen werden. Hierzu wird eine Schicht Lack auf die Innenseite der Form aufgetragen um dann im darauf folgenden Schritt auf diesen vorbereiteten Grund die GFK-Matten zu legen. Die Matten saugen sich mit dem Lack voll und bilden so eine Schicht. Zur bestmöglichen Festigkeit sollten sich die GFK-Matten immer um ca. 2-3 cm überlappen. Dieser Schritt wird so oft wiederholt bis die gewünschte Dicke der Außenwand erreicht ist. Die Hecksektion hat eine Wandstärke von sechs Schichten. In der Bugsektion haben wir extra 12 Schichten zur besseren Stabilität einlaminiert.Bei einem Frontalaufprall des Bootes sollte es daher keine Schäden geben.
Für den Bug sowie auch für das Heck sind aus einem Rundaluminiumblock zwei Verschlüsse gedreht worden.
Der Verschluss der Bug-Sektion ist im Mittelteil etwas länger da dort später die Anschlüsse für Bugstrahlruder geplant sind. Ansonsten sind die beiden Verschlüsse baugleich. Die beiden fertigen GFK-Teile werden nun in dem Bereich auf den die Verschlüsse geschoben werden mit 6-8 kleinen Löchern versehen. Diese Löcher dienen dazu, dass mit Hilfe einer Spritze Harz in den verbleibenden Zwischenraum von Verschluss und GFK-Teil gespritzt werden kann. Dieser härtet dort aus und verbindet so auf recht einfache Weise die beiden Bauteile. Damit ist der Bug bis auf die Montage der Außenverschlüsse soweit schon fertig. In das Heck müssen im folgenden noch Koker und Servos, Motoren und Motorhalterung, sowie die Durchführung der Schläuche für den Tauchtank und den Drucksensor eingebaut werden.

Montage der Koker für die X-Ruder
Die Koker führen die Ruderwellen von außen ins Boot. Sie haben an beiden Enden eine Vertiefung für

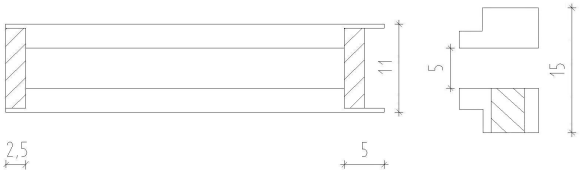


Abbildung 2.8: Zeichnung Koker und Kokerverschluss

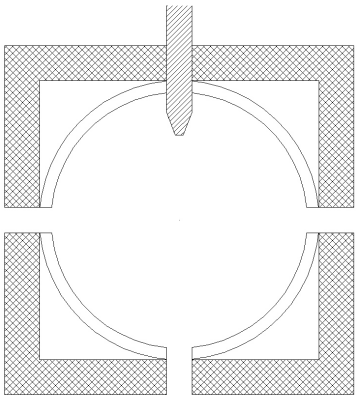


Abbildung 2.9: Zeichnung Montagevorrichtung für Koker

einen Dichtring. Innen sitzt dort dann der Ruderhebel auf und außen haben wir ein kleines Verschlussteil aufgesetzt. Dieses Teil sorgt dafür das der äußere Dichtring nicht heraus rutschen kann. Damit mit die Koker später möglichst genau platziert sind, haben wir in erster Instanz ein Holzgestell gebaut. In dieses Holzgestell konnte an definierten Punkten so gebohrt werden, das die Abstände von den einzelnen Löchern zueinander exakt stimmen.

Im nächsten Schritt wurde in einen Holzklotz ein 5 mm dickes Loch exakt senkrecht gebohrt und in dieses die Ruderstange gesteckt. Die Ruderstange wird nun durch das vorher gebohrte Loch gesteckt und dadurch, dass das Loch exakt senkrecht in den Klotz gebohrt wurde und dieser auf der Außenwand aufliegt, erhält der nun aufgesteckte Koker einen genauen 90° Winkel zur Außenwand. Diese Konstruktion mag auf den ersten Blick etwas umständlich erscheinen, aber durch die leicht gewölbte Außenwand lässt sich kaum eine exaktere Ausrichtung erzielen.

Nach der Platzierung der Koker im Heck, werden sie mit etwas Heißkleber fixiert damit sie bei dem anschließenden Einlaminieren nicht verrutschen. Damit eine höhere Stabilität garantiert werden kann werden an jedem Koker zwei Kunststoffstreifen als Steg angebracht. Über diesen Steg und um den Koker wird dann laminiert. Nach der Trockenphase kann dann der Holzklotz mit der Ruderstange entfernt werden.
Zur Abdichtung der Ruderwellen ist an beiden Seiten des Kokers eine Vertiefung in der sich jeweils ein O-Ring zur Abdichtung befindet.

Befestigung der Servos und Anschluss an die Ruderstange
Um die Servos im Heck zu befestigen haben wir für jeden Servo eine Halterung aus V4-Stahl gebaut. Die Halterung besteht auf der Seite aus zwei Flacheisen die dem Servo seitlich Stabilität geben, sowie vier Löchern mit dem der Servo auf der Halterung befestigt werden kann. An der anderen Seite der Halterung sind drei Löcher so angebracht, dass die Halterung an dem Verschlussring befestigt werden kann.

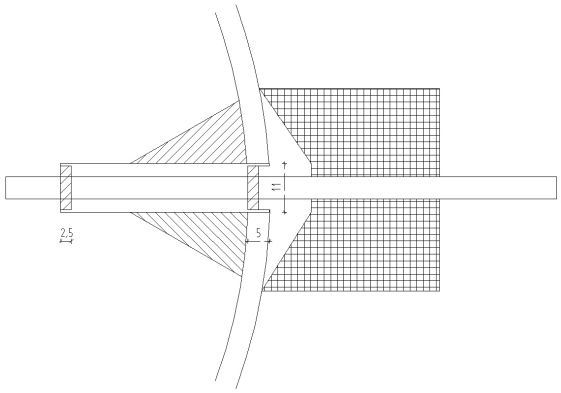


Abbildung 2.10: Kokerfixierung

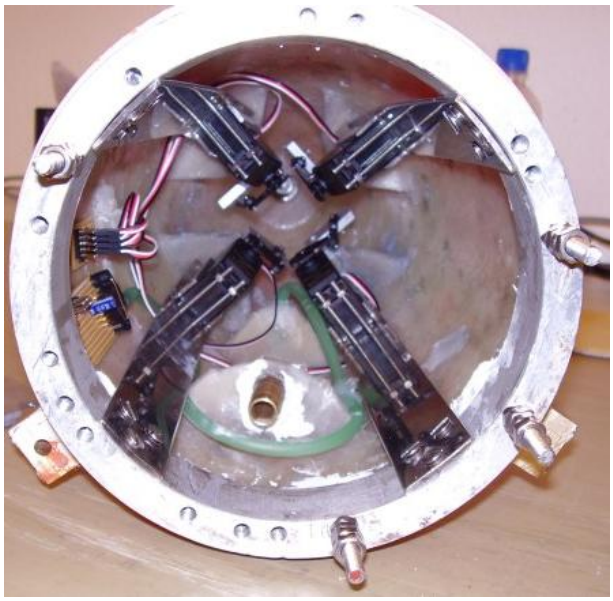


Abbildung 2.11: Heck mit montierten Servos und Ruderansteuerung

Aus Aluminium haben wir einen Hebel gefertigt der auf die Ruderstange montiert wird. Dieser Hebel stellt das Gegenstück zu dem Servoarm dar. Mithilfe einer M3-Gewindestange mit beidseitigem Kugelkopfgelenk wird die Kraft des Servosarms auf den Hebel übertragen und bewegt somit das Ruder. Der Hebel ist mit Hilfe einer Madenschraube auf der Ruderstange montiert. Der Vorteil gegenüber einer Festmontage liegt darin, dass sich der Hebel immer wieder nachjustieren lässt und bei einer ungewollt hohen Belastung (Berührung des Beckenrands oder Bodens) auf die Ruder die plötzlich auftretende Kraft nicht voll auf den Servos und die Koker schlägt.

Schlauchdurchführungen für Tauchtank und Drucksensor

Bei den Durchführungen sind wir ähnlich wie bei der Einlaminierung der Koker vorgegangen. Nachdem die Löcher in die Unterseite des Hecks gebohrt wurden, haben wir die den Drucksensorschlauch und den Winkelansaugstutzen für den Tauchtankschlauch mit Hilfe von Heißkleber fixiert. Mit Hilfe von Kunststoffteilen wurden dann wieder jeweils zwei "Flügelängebracht und dann die Bauteile einlaminiert.

Ventil

Um im U-Boot einen Überdruck zu erzeugen, haben wir ein normales Fahrradventil ins Heck eingesetzt.

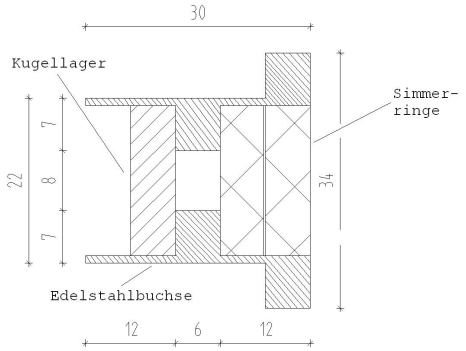


Abbildung 2.12: Zeichnung Wellenführung

Dazu mussten wir ein Loch ins Heck bohren und das Ventil (inkl. Ventilschaft) hineinschrauben. Danach haben wir Die Ränder mit Sikaflex abgedichtet. Der erhöhte Innendruck gibt dem PVC-Rohr mehr Stabilität, da es dadurch nicht so leicht vom Wasserdruck verformt werden kann. Des weiteren strömt bei kleinen Lecks zuerst lange Zeit Luft aus dem Boot, bevor Wasser eindringen kann. Wir können diesen Vorgang über einen Innendrucksensor überwachen.

Antriebsmotor und Schraube
Das U-Boot wird mit einem Boschmotor angetrieben, welcher aus einer Ameise, einem elektrischen Hubwagen, stammt. Den Motor haben wir gebraucht von Lothar Mentz gekauft. Er wird mit 12 V angesteuert, kann jedoch auch mit 24 V laufen.
Die Drehzahl des Motors wird mit Hilfe eines Reedkontaktes gemessen. Dieser ist am Motorgehäuse befestigt und auf der Lüfterscheibe ist ein Magnet angebracht. Nach den ersten Trockentest sind die gelieferten Werte exakt genug für uns.
Eine Antriebsschraube (ca. 12 cm Durchmesser) haben wir zuerst von Lothar Mentz bekommen. Er wollte diese allerdings wieder haben, da es seine erste selbst gelötete Schraube ist. Wir sahen uns selbst nicht in der Lage eine solche Schraube zu löten. Schrauben in dieser Größe fertig zu kaufen ist viel zu teuer, da im Schiffsmodellbau nur Schrauben bis ca. 6 cm Durchmesser üblich sind. Nach langem Suchen sind wir im Souvenirbereich fündig geworden. Die Schraube (Messing, 3-Blatt, ca. 12 cm Durchmesser) war sehr günstig, allerdings viel zu groß aufgebohrt für unsere Welle. Karl-Heinz Behrens hat uns ein Stück Messingrundstange mit Silberlöten in unserer Schraube befestigt, ein 6mm Loch für die Welle gebohrt und eine Kerbe auf der Innenseite gefräst. In die Kerbe passt ein Fixierstift, der durch ein in die Welle gebohrtes Loch geht. Jetzt haben wir die Schraube auf die Welle geschoben und am Ende mit dem Fixierstift und einer Stoppmutter befestigt.

Motorbefestigung und Lagerung der Antriebswelle
Am Heckverschluss wurde eine Aluminiumplatte angebracht. Die M6-Gewindestangen die an den Motor montiert sind, können somit an der Platte mit Hilfe von Schrauben befestigt werden. Somit ist die Aufhängung des Motors zur Verschlussseite gesichert, aber um ein Aufschwingen des Motors zu verhindern, muss dieser auch noch an der anderen Seite gelagert werden. Hierzu haben wir uns die Austrittstelle der Antriebswelle am Heck ausgesucht. Dazu haben wir ein Bauteil fertigen lassen, welches wir in das Hinterteil des Hecks einlaminiert haben.
Auf der Innenseite ist ein Kugellager zur Führung der Welle eingepresst und auf der Außenseite sind zwei Simmerringe eingelassen. Somit dient dieses Bauteil neben der Lagerung der Welle im hinteren Segment noch zur Abdichtung der rotierenden Welle.

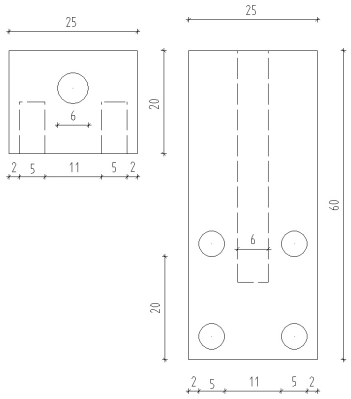


Abbildung 2.13: Zeichnung Außenverschlussteil für Hauptrohr

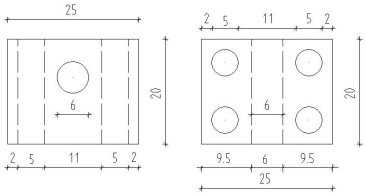


Abbildung 2.14: Zeichnung Außenverschlussteil für Bug und Heck

Mittelteil

Um unsere Aluminiumverschlüsse von außen schließen zu können, haben wir uns überlegt am Rohr, sowie an Bug und Heck Metallblöcke zu befestigen, die wir mit Schrauben aneinander ziehen können. Zuerst haben wir als Material für die Blöcke V4A-Stahl und für die Schrauben Messing gewählt. Nachdem die Bearbeitung der Blöcke (Löcher und Gewinde bohren) sich aber als viel zu problematisch herausstellte, haben wir die Blöcke aus Messing gefertigt und die Schrauben aus Stahl.

Nun haben wir am Anfang und am Ende des Rohres jeweils drei Blöcke mit gleichen Abständen befestigt. Die Befestigung erfolgte mit vier Schrauben, die von innen durch das Rohr, in die sich in den Blöcken befindlichen Gewinde geschraubt wurden. In diese sechs Blöcke ist jeweils ein Gewinde gebohrt, in das die Schraube zum Verschließen geschraubt werden kann. In den anderen Blöcken, die wir nun an Bug und Heck befestigen mussten, ist nur ein Loch durch das die Schraube passt.
Zur Befestigung dieser Blöcke haben wir jeweils eine Verschlussschraube durch einen Block an der entsprechenden Halterung am Rohr befestigt und Bug, bzw. Heck auf das Rohr gesetzt. Jetzt haben wir mit einer Handbohrmaschine durch die Blöcke in das GFK und in den sich darunter befindlichen Aluminiumverschluss jeweils vier Löcher gebohrt, und mit Gewindeschraube und Mutter fixiert. Zum Schluss haben wir die Auflagekanten der Blöcke, sowie die Innen liegenden Schraubenköpfe mit Sikaflex abgedichtet.
Die Verschlüsse sind zwar strömungstechnisch ungünstig, jedoch erfüllen sie vorerst ihren Zweck. Probleme bereitet allerdings die Dichtigkeit, da durch das Herausziehen und Reinschieben des Schlittens das Sikaflex an den Schraubenköpfen stark belastet wird. Der Überdruck im Bootsinneren ist jedoch so groß, dass wir bisher noch keinen Wassereinbruch hatten. In nächster Zeit ist jedoch geplant, das Boot mit Hilfe eines Elektromagneten zu verschließen und damit auf Außenverschlüsse und die dadurch entstandenen Durchbrüche verzichten zu können.

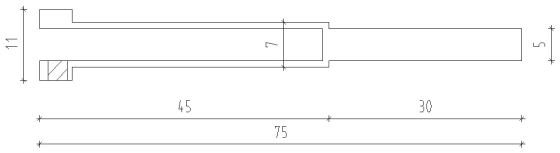


Abbildung 2.15: Zeichnung Ruderstangenführung

Abbildung 2.16: Bild fertiges Ruder

2.3.2 Ruder

Für unser angedachtes X-Ruderprinzip haben wir, wie schon zuvor, Formen von Lothar Menz zur Verfügung gestellt bekommen. Dadurch ist uns die aufwendige Modellierung der Negativformen erspart geblieben und zugleich haben wir von Anfang an die richtigen Größenverhältnisse der Ruder im Vergleich zum Heck und dem Rest unserer gewählten Form.

Die Ruderstangenführungen
In eine 11 mm dicke Rundmessingstange wird ein 5 mm dickes Loch gebohrt. Dieses Loch wird später die Ruderstange führen. An der dem Uboot zugewendeten Seite behält die Stange ihre volle Dicke von 11 mm, da dort eine Madenschraube zur späteren Fixierung der Ruderstange eingelassen wird. Der Rest der Stange wurde auf der vollen Länge auf 9 mm und ganz an der Spitze 5 mm dünner gedreht, um die Form der Führung etwas der Form des Ruders anzupassen und nicht zu dünne Wandstärken im unteren Bereich des Ruders zu erzeugen.
Die Herstellung der Ruder war im Prinzip sehr einfach. Zuerst müssen die Formen mit Trennlack und Trennwachs vorbereitet werden. Im nächsten Schritt wird die Gussmasse angerührt, diese besteht pro Ruder aus ca. einem Esslöffel Baumwollflocken und aus 40 ml Harz-Härtergemisch. Die Baumwollflocken sorgen dafür, dass das Material härter und weniger brüchig wird. In die Form wird mit Hilfe einer selbstgebauten Halterung die Ruderstangenführung gestellt und mit eingegossen.

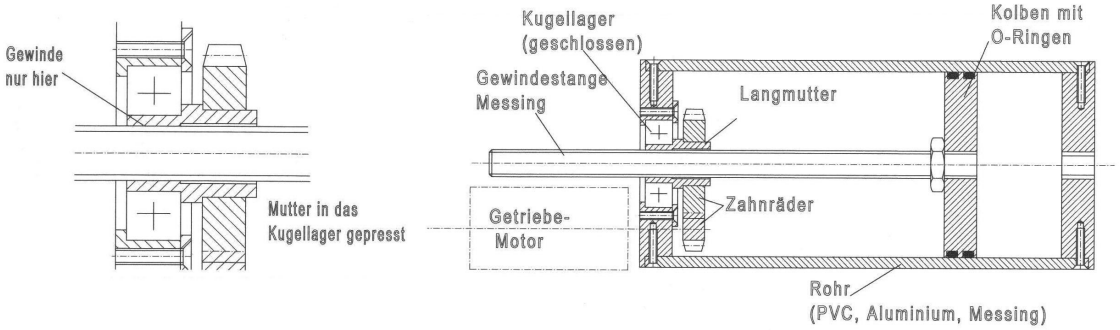


Abbildung 2.17: Zeichnung Tauchtankkonzept

2.3.3 Tauchtank

Bei den Tauchtanks haben wir uns von Anfang an für Kolbentanks entschieden, weil diese am genausten anzusteuern sind. Wenn die Umdrehung des Motors gemessen wird, kann der Füllstand genau berechnet werden. Bei allen anderen Tauchsystemen werden Pumpen oder Druckluft verwendet, und zur Ansteuerung muss der Durchfluss berechnet werden. Für ein Druckluftsystem hätten wir außerdem einen Drucktank bauen müssen und das schien uns als zu hohe Anforderung, da dieser auch vom TÜV abgenommen werden muss. Ein Tauchsystem bei dem eine Pumpe Wasser in einen Gummisack rein und raus pumpt war uns allerdings von den Bauanforderungen zu niedrig. Fertige Tanks zu kaufen war uns zu teuer und nachdem wir von Lothar Mentz auch noch Material sowie Konstruktionsskizzen bekommen haben, war der Eigenbau der Kolbentanks beschlossen.
Als Rohr haben wir ein Aluminiumrohr genommen, welches wir auch von Lothar Mentz bekommen haben. Zuerst wollten wir ein PVC-Rohr benutzen, aber diese sind von Innen nicht rund genug. Der Kolben ist aus PVC und mit zwei Dichtringen abgedichtet. Am Kolben ist eine M10 Gewindestange aus Messing befestigt, welche über eine Zahnraduntersetzung von einem Motor angetrieben wird. Direkt auf der Gewindestange läuft eine speziell angefertigte Langmutter aus Stahl auf die ein Kunststoffzahnrad gepresst ist. Diese Langmutter ist aus Stahl, da bei der Eisenverarbeitung immer ein hartes (Stahl) und ein weiches (Messing) Material verwendet werden sollte.Somit wird verhindert, dass die relativ aufwendig herzustellende Langmutter abnutzt. Abgeschlossen wird der Tank auf der einen Seite mit einem 10 mm starken PVC-Deckel in welchen eine Schlauchtülle eingeschraubt ist. Den Deckel am Rohr und das Tüllengewinde im Deckel haben wir mit Sikaflex geklebt, bzw. abgedichtet. Auf der Antriebsseite des Tauchtanks haben wir den Deckel aus 12mm starkem Stahl drehen lassen wobei, wie auch beim PVC-Deckel, 2 mm des Deckels dem Außendurchmesser des Rohres entsprechen. In diesen Deckel haben wir Gewinde gedreht und ihn mit zwei Schrauben am Rohr befestigt.
Als wir mit den Materialien und den Konstruktionsskizzen (Abb. Tauchtank) zu Karl-Heinz Behrens gefahren sind, mussten wir erstmal ein paar Details überarbeiten. Die größten Probleme waren die Befestigung des Kunststoffzahnrades auf der Langmutter und die Lagerung der Langmutter in dem vorgesehenen Kugellager. Der Befestigungsteil des Kunststoffzahnrades muss soweit aufgebohrt werden, dass nur noch eine 1,5 mm starke Hülle übrig bleibt. Diese lässt sich nur befestigen, wenn sie auf die Langmutter geschoben und darüber einen Stahlring spannt wird. Diese Lösung ist nicht besonders gut, aber da es in der Größe keine Zahnräder mit größerem Befestigungsteil gibt, mussten wir es dabei belassen. Das Kugellager hatte nach Einschätzung von Karl-Heinz Behrens einen zu kleinen Innendurchmesser um eine ausreichend starke Langmutter zu drehen, die dort eingebaut werden kann. Er riet uns dann gleich anstatt Kugellager Sinterbuchsen zu verwenden, da diese unempfindlich gegenüber Wasser sind. Sinterbuchsen bestehen aus speziellem Sintermessing, in dem Stahlwellen besonders gut laufen. Sie werden als Gleitlager bezeichnet.
Nun kam hinzu, dass wir noch Inkrementalgeber an den Tauchtanks anbauen wollten, welche nicht mehr auf die Motorwelle gepasst hätten, da diese zu kurz und einen zu geringen Abstand zur Gewindestange gehabt hätten. Deshalb musste eine weitere Zahnraduntersetzung außerhalb des Tanks mit einem Abstandshalter aufgesetzt werden. Das hatte zur Folge, dass wir eine zweite Welle brauchten und diese auch im

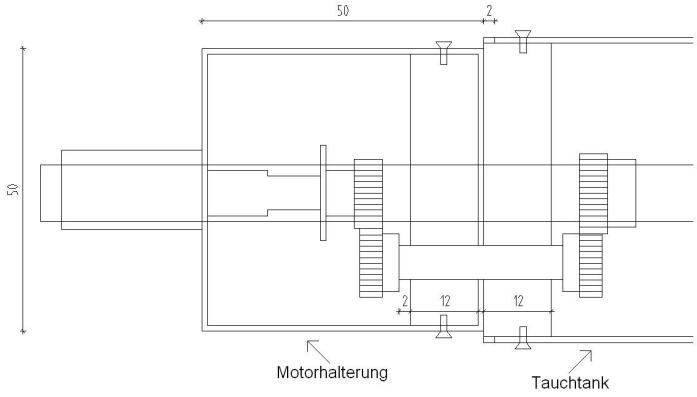


Abbildung 2.18: Zeichnung Tauchtankantrieb

Tauchtankdeckel lagern mussten. Hier sollte ebenfalls eine Sinterbuchse verwendet werden. Desweiteren wurden von einem Teil der Gruppe, sowie von Frank Slomka, neue Motoren vorgeschlagen, die die Tanks in knapp 10-15 Sekunden, statt 45 Sekunden fluten oder lenzen sollten.
Nun wurden diese Tanks realisiert, wobei wir die Abstandshalter leider um 1-2mm zu klein fertigten und die Inkrementalgeber nicht mehr dazwischen passten. Die neuen Motoren hatten allerdings an beiden Enden eine Welle, so dass wir die Inkrementalgeber zur Not auch dort hätten anbringen können. Allerdings liefen die Tanks nicht ordentlich. Nach langem Überlegen wo der Fehler sein könnte, fiel uns auf, dass die Sinterbuchsen nicht nur eine radiale, sondern auch eine axiale Kraft an der Langmutter abfangen müssen und dafür nicht geeignet sind. Außerdem schienen uns die Motoren, trotz des berechneten ausreichenden Drehmomentes dieses nicht aufbringen zu können.
Somit begann die nächste Konstruktionsänderung: Wieder Kugellager an den Langmuttern statt Sinterbuchsen und eine höhere Untersetzung (15:30 statt 20:25) um die Tanks dann in ca. 20 Sekunden fluten oder lenzen zu können. Die Kugellager haben gut funktioniert, allerdings waren die Langmuttern durch die erneute Bearbeitung nicht mehr rund und mussten erneuert werden. Außerdem waren wir uns sicher, dass die schnelleren Motoren auch mit der neuen Untersetzung nicht stark genug sein würden um einen Stempel im Tank unter Last zu bewegen.
Also haben wir die Untersetzung wieder auf die ursprünglichen Werte (20:25) zurückgesetzt, die alten Motoren genommen und neue Abstandshalter gefertigt. Nun konnten wir auch die Inkrementalgeber in die Abstandshalter integrieren und die Tauchtanks endlich fertig stellen. Dazu mussten die zusätzlichen Wellen mit einem Messingblock gelagert werden, da diese zu viel Spiel hatten. Allerdings mussten wir feststellen, dass die Tanks immer noch nicht so liefen, wie wir uns das vorgestellt hatten. Bei einem Tauchtank saß das Plastikzahnrad nicht fest genug auf der Langmutter, da uns die eine Langmutter zu dünn gedreht wurde. Bei höherer Belastung (Über-/Unterdruck im Tank) drehte sich nur noch die Mutter. Aber auch der zweite Tank lief bei dieser Belastung nicht zuverlässig genug. Es gab immer wieder Aussetzer und der Kolben blieb sogar stehen.
Obwohl und auch gerade weil wir so viel Zeit in die Tanks investiert hatten, haben wir uns entschlossen nicht daran weiter zu bauen, sondern zwei Kolbentanks (je 500ml) der Firma Engel zu kaufen. Zusätzlich haben wir noch Ersatzzahnräder für die Motorwellen bestellt um die Originalmotoren durch unsere schnellen Motoren aus der Eigenkonstruktion zu ersetzen, da wir sonst keine Inkrementalgeber hätten anbringen können. Diese Version der Tauchtanks wurde nun in den Schlitten integriert. Die Flut-/ Lenzzeit beträgt ca. 12 Sekunden.
Fazit: Durch die von uns benötigten Änderungen an den Originalplänen von Lothar Mentz, den Einbau der Inkrementalgeber und den Wunsch nach höherer Geschwindigkeit, ergaben sich größere Probleme als

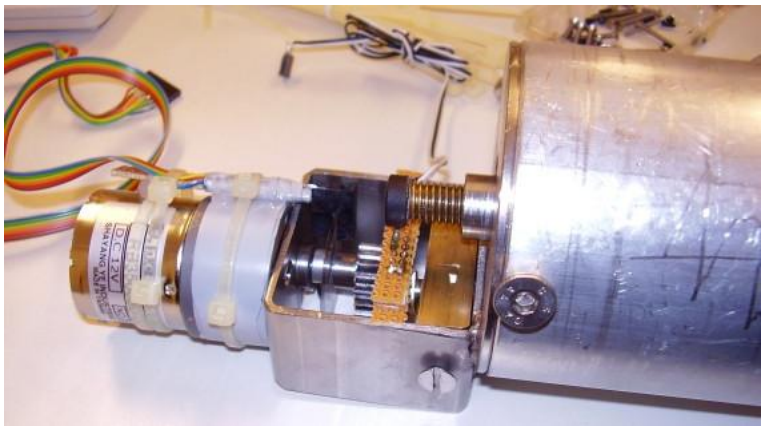


Abbildung 2.19: fertiger Tauchtank

erwartet. Hätten wir die Tauchtanks ohne diese Änderungen hergestellt, da sind wir uns sicher, hätten diese auch zuverlässig funktioniert. Allerdings hätten wir uns mit unseren Anforderungen doch besser dafür entscheiden sollen Tauchtanks zu kaufen, da uns der Bau der eigenen Tanks mit Abstand am meisten Zeit beim Bau des gesamten Rumpfes gekostet hat. Außerdem sind die Materialkosten unserer Eigenkonstruktion auch ständig gestiegen, da für jede Änderung wieder neue Teile benötigt wurden. Jedem, der einen Eigenbau machen möchte, können wir nur raten eine viel höhere Untersetzung zu wählen als wir es getan haben. Dann ist es auch möglich einen schnellen Motor zu verwenden, wie wir bei den gekauften Tanks gut sehen konnten.

2.3.4 Technikschlitten

Um die Technik im Inneren des U-Boots leicht zugängig zu machen, haben wir uns das Prinzip eines Technikschlittens überlegt. Sechs PVC-Scheiben mit dem Innendurchmesser des Rohres werden mit Gewindestangen zusammengehalten und am Heck mit dem Aluminiumverschluss verbunden. Somit kann das gesamte Innenleben aus der Außenhülle herausgezogen und bequem daran gearbeitet werden. Der Schlitten ist in drei Teile unterteilt: Vorne und hinten jeweils einen Teil für einen Tauchtank und zwei Akkus, sowie einen Mittelteil mit einer Stahlplatte auf der die Platinen befestigt werden. Die Gewindestangen verbinden jeweils einen Teil und versetzt sind weitere Stangen angebracht um die Teile zusammenzuhalten. Somit lassen sich die Teile gut trennen. Für die Tauchtanks haben wir Löcher in die Scheiben gesägt, damit diese genau fixiert werden können. Bei den Akkus haben wir eine Vertiefung in die Scheiben gefräst, um sie einfach zwischen die Scheiben zu spannen.

2.3.5 Trimmblei

Da wir wussten, dass wir den Auftrieb des Bootsrumpfes nicht nur mit den an Bord befindlichen mechanischen und elektronischen Teilen ausgleichen können, haben wir Trimmblei hergestellt. Dafür haben wir bei einem Schrotthändler Blei gekauft und zusätzlich noch Reste von Karl-Heinz Behrens bekommen. Wir haben uns einen Schmelztiegel und eine Form aus Eisen zusammengeschweißt und damit unsere Bleibarren hergestellt. Hierfür konnten wir den Ofen bei Maschinenbau Behrens nutzen. Die Barren sind ca. 2 cm breit und hoch mit einer Länge von ca. 30 cm. Die Form haben wir oben etwas breiter gemacht, damit wir das erkaltete Blei leichter herausbekommen konnten.

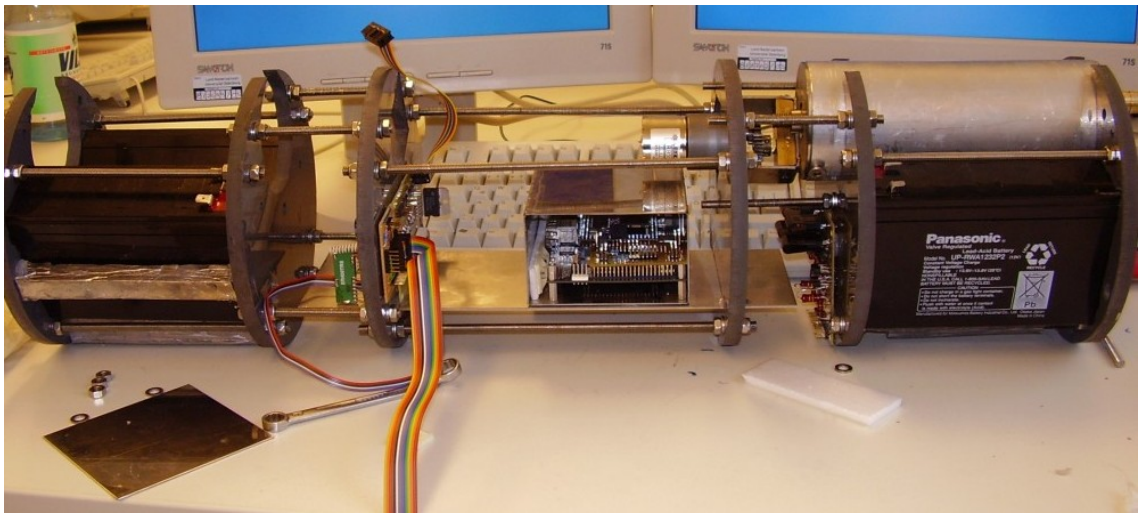


Abbildung 2.20: Bild Technikschlitten

2.4 Zusammenbau und Integration

Beim endgültigen Zusammenbau des Bootes gab es keine technischen Probleme. Der Bug wurde einfach mit den Verschlüssen am Hauptrohr befestigt und es ging nur noch um den Schlitten und das Heck, sowie die Integration der Teile auf dem Schlitten. Zuvor haben wir allerdings noch Bug, Heck und Mittelteil angeschliffen, grundiert und mit Autolack lackiert.

2.4.1 Verbindung von Heck und Schlitten

Das Heck sollte, wie geplant, mit dem Schlitten verbunden werden und so in einem Stück in das Hauptrohr geschoben werden. Dazu wurden die restlichen Teile ins Heck integriert (siehe 2.4.2-2.4.3) und der Motor mit der Halterungsplatte am Aluminiumverschluss festgeschraubt. In den Verschluss haben wir ebenfalls Gewinde gedreht, wodurch wir den Schlitten mit Gewindestangen am Heck verschrauben konnten.

2.4.2 Anschluss der Tauchtankschläuche

Im Heck mussten die Kabel für die Ruderservos und den Drucksensor mit Steckern angeschlossen, sowie die Schläuche vom Ansaugstutzen zu den Tauchtanks verlegt werden. Direkt am Ansaugstutzen haben wir ein T-Verteilerstück mit einem kurzen Stück Schlauch gesetzt. Bei den selbstgebauten Tauchtanks haben wir einen Schlauchanschluss von 12 mm Durchmesser eingebaut und daher auch den Ansaugstutzen mit diesem Maß benutzt. Die gekauften Tauchtanks haben allerdings nur einen Schlauchanschluss von 8 mm Durchmesser, weshalb wir Verbindungsstücke gekauft haben was diesen Unterschied ausgleicht. Vom vorderen Tauchtank läuft der dünne Schlauch fast bis zum Heck, wo das Verbindungsstück ihn mit dem kurzen Stück Schlauch verbindet, was vom Verteiler kommt. Beim hinteren Tauchtank haben wir das Problem, dass der Schlitten direkt an die Motorhalterungsplatte stösst, und wir den Tauchtankschlauch deshalb durch ein Loch in der Platte laufen lassen müssen. Ansonsten ist die Verbindung identisch mit der des vorderen Tauchtanks. Die Schläuche können an den Verbindungsstücken leicht getrennt werden, wenn wir bei Arbeiten das Heck vom Schlitten trennen oder den Schlitten selbst zerlegen wollen.

2.4.3 Integration der Platinen

Auf dem Schlitten waren bereits die Akkus und die Tauchtanks montiert, so fehlten nur noch die elektronischen Komponenten. Da das FPGA-Board und die Sensoren für die Navigation (Gyroskope und Beschleunigungssensoren) empfindlich gegenüber elektromagnetischen Störungen sind, haben wir einen Kasten

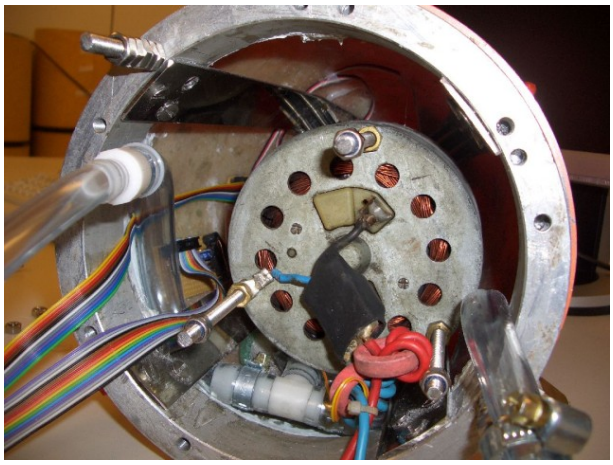


Abbildung 2.21: Heck mit Motor und Tauchtankschläuchen

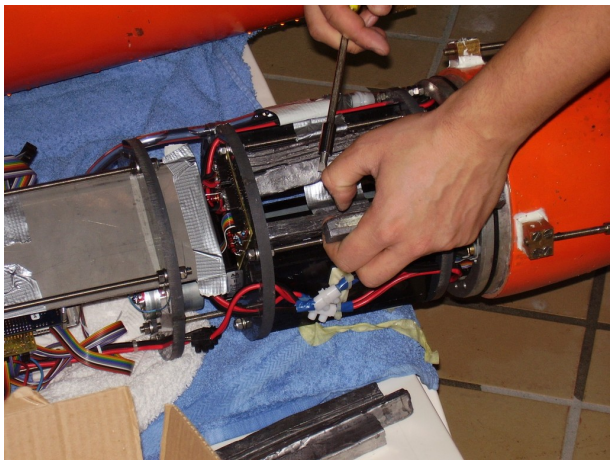


Abbildung 2.22: Austrimmung des Bootes

aus Blech fertigen lassen, der nur auf einer Seite geöffnet ist. Diesen Kasten haben wir in der Mitte des Schlittens auf die Halterungsplatte geklebt. Alle weiteren Platinen wurden hauptsächlich mit Heißkleber im Schlitten fixiert und so angeordnet, dass sie sich nicht gegenseitig stören und möglichst keinen Platz verschwenden.

2.4.4 Austrimmen des Bootes

Um das U-Boot in einem Schwebezustand unter Wasser zu halten, muss das Auftriebsvolumen gleich dem Gesamtgewicht des Bootes sein. Dazu haben wir ersteinmal grob abgewogen wieviel Blei wir dafür ins Boot legen mussten. Im Schwebezustand sollen die Tauchtanks jeweils mit ca. 300 ml Wasser gefüllt sein. Unsere Bleibarren haben wir so zersägt, dass wir das Hauptgewicht unter der Metallplatte im Mittelteil fixieren konnten. Wir haben mehrere Barren mit Gewebeband zusammengeklebt und zwischen die PVC-Scheiben gepresst. So lässt sich das Blei auch immer problemlos wieder herausnehmen. Das restliche Trimmblei haben wir zwischen den vorderen und hinteren Akkus platziert. Da das Heck einen viel größeren Auftrieb als der Bug hat, befindet sich zwischen den hinteren Akkus viel Blei und zwischen den vorderen Akkus ist nach entgültigem Austrimmen nur noch ein kleines Stück Blei geblieben.
Bei der Feintrimmung haben wir kleine Bleistücke so auf dem im Wasser schwimmenden Boot platziert, dass das Boot vorne und hinten gleich hoch aus dem Wasser ragte. Nachdem auch diese Bleistücke im

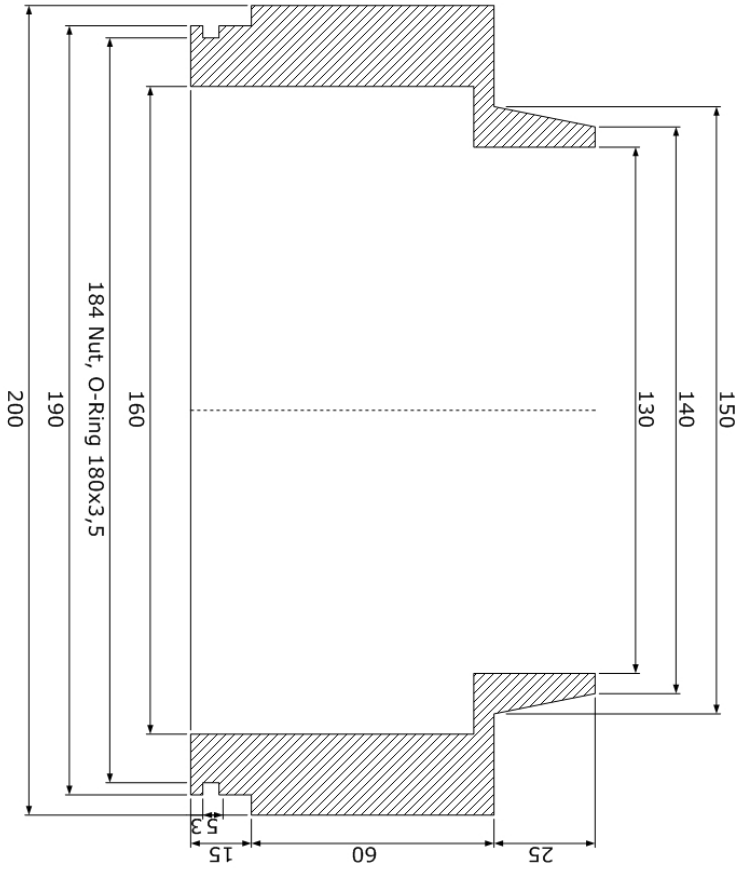


Abbildung 2.23: Zeichnung Aluminiumring für neuen Bug

Schlitten mit Heißkleber befestigt wurden, konnten wir mit den ersten Tauchfahrten beginnen.

2.5 Erweiterungen

Wie geplant, haben wir im zweiten Semester Erweiterungen am Boot durchgeführt. Das Boot hat einen neuen Bug und einen neuen Antriebsmotor bekommen. Außerdem haben wir das Verschlusssystem geändert. Da sich auch die Anforderungen an den FPGA geändert haben, wurde ein neues Board eingesetzt und der Blechkasten, der nun zu klein war, herausgenommen. Die elektromagnetischen Störungen haben sich im Laufe des Projektes als nicht so stark herausgestellt, also haben wir auch keinen neuen Kasten gebaut. Nach den Änderungen mussten wir natürlich das Boot neu austrimmen.

2.5.1 Bug

Um in unser Boot eine Kamera integrieren zu können, haben wir uns dazu entschieden einen neuen Bug zu bauen. Dazu haben wir uns einen neuen, längeren Aluminiumring von Herrn Behrens drehen lassen und dort eine Halbkugel aus PMMA mit Kunstharz befestigt. Wir haben die Kuppel von außen mit Heißkleber am Bug befestigt und von innen mit einer Spritze den Harz in den Freiraum gespritzt.
Zur Befestigung der Kamera, welche zur Beweglichkeit auf einem Servo montiert ist, haben wir eine PVC-Scheibe über vier Gewindestangen mit der Deckplatte des Bug verbunden. Auf diese Scheibe ist der Servo geklebt und somit können wir die Kamera samt der Halterung aus dem Bug ziehen.
Als nächstes haben wir ein Stück Schlauch für einen weiteren Drucksensor in den Aluminiumring einlaminiert und diesen daran angeschlossen. Den Sensor haben wir an der PVC-Scheibe mit Heißkleber befestigt.

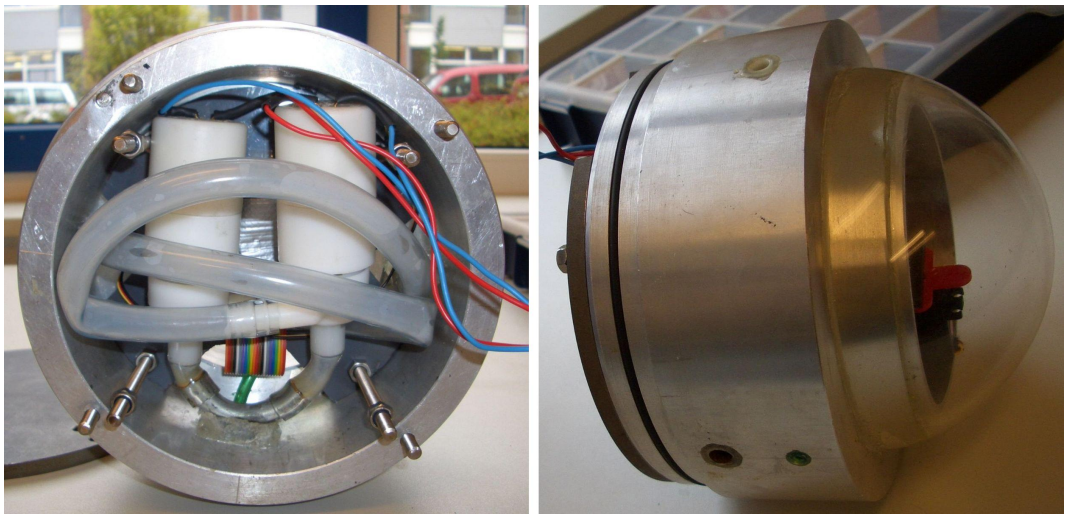


Abbildung 2.24: Bild offener Bug und Seitenansicht

Bugstrahlruder

Schon früh hatten wir die Idee unser U-Boot mit Bugstrahlrudern auszustatten um das Boot auf engem Raum ohne Motor drehen zu können. In den Alumiumring haben wir jeweils ein Loch links und ein Loch rechts, sowie eines unten gebohrt. Links und rechts haben wir Winkelstutzen einlaminiert die bei den Pumpen dabei waren. Dann haben wir uns einen Y-Verteiler aus einem Stück Rohr zusammengeschweißt und diesen unten in den Ring einlaminiert. Die Pumpen haben wir nun zwischen die Halterungsscheibe für die Kamera und die Deckplatte gesetzt und über Schläuche mit den einlaminierten Stutzen verbunden. Befestigen mussten wir die Pumpen ansonsten nicht, da der Platz im Bug so begrenzt ist, dass ein Verrutschen nicht möglich ist.

2.5.2 Magnetverschluss

Bei den Außenverschlüssen des Bootes gab es immer wieder Probleme mit der Dichtigkeit. Daher haben wir überlegt, wie wir das Boot von außen öffnen und schließen können ohne so viele Durchbrüche wie vorher zu haben.
Wir kamen auf die Idee das Boot mit einem Elektromagneten zu verschließen und diesen durch anlegen einer Spannung an zwei einlaminierten Kontakten wieder zu öffnen. Der Magnet sollte Bug und Schlitten zusammen halten, da wir an dieser Stelle nur Kabelverbindungen haben.
Der von uns gewählte Magnet hat einen Durchmesser von 90 mm, wiegt 1,7 kg und hat eine Haftkraft von 1200 N. Dieser reicht aus um unser Boot auch bei einem überdruckt von 0,5 Bar problemlos zusammen zu halten.
Im Heck haben wir zwei Gewindestangen im Aluminiumring mit Kunstharz einlaminiert. An diese sind die Kabel die zum Magneten führen angeschlossen. Die alten Außenverschlüsse haben wir abgeschraubt und die Löcher mit Harz gefüllt.
Um den Magneten im Boot unterzubringen, haben wir einfach eine weitere Sektion vorne an den Schlitten gesetzt und ein längeres Rohr besorgt. Die Sektion besteht aus zwei 20 mm dicken PVC-Scheiben. Die hintere Scheibe ist 10 mm tief mittig mit einem Durchmesser von 90 mm ausgefräßt. Außerdem haben wir ein Loch für den Stromanschluss und ein Loch zum Festschrauben gebohrt, da der Magnet auf der Rückseite eine Gewindebohrung hat. Die vordere Scheibe ist komplett mit einem Durchmesser von 90 mm ausgefräßt und so stabilisiert sie den Magneten, der vorne noch 10 mm herausragt.

Die Deckplatte am Bug besteht aus 10 mm Eisen. Zuerst hatten wir eine 3 mm Platte, doch auf dieser

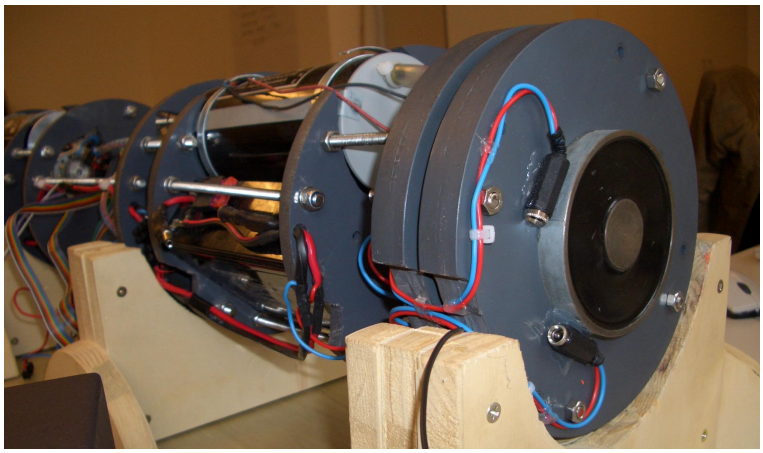


Abbildung 2.25: Bild Erweiterter Schlitten mit Elektromagneten

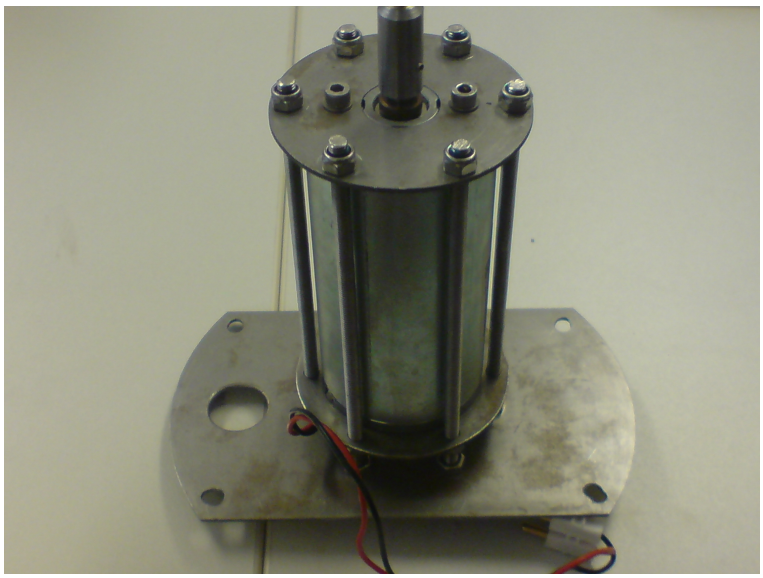


Abbildung 2.26: Bild Neue Motorhalterung mit Motor

hatte der Magnet zu wenig Kraft. Laut Hersteller benötigt der Magnet eine Mindestbelegungsdicke von 7,5 mm. Außerdem ist da darauf zu achten keinen Edelstahl zu benutzen, da dieser keine, bzw. schlechte magnetische Eigenschaften besitzt.

2.5.3 Antrieb

Unser Antriebsmotor hat leider im Verlaufe des Projektes, vermutlich aufgrund seines Alters, versagt. Ersatz haben wir bei der Firma Dunker gefunden, die uns freundlicherweise gleich vier Motoren gespendet hat. Da der neue Motor sich in Länge, Durchmesser und Wellendurchmesser von dem alten Motor stark unterschieden hat, haben wir eine neue Halterung gebaut und uns eine neue Welle drehen lassen. Auf die Messung der Motorgeschwindigkeit, wie beim alten Motor, haben wir mittlerweile verzichtet.
Die Halterung besteht aus zwei Aluminiumscheiben, in denen Löcher für die Welle und den Stromanschluss sind, sowie am Rand sechs Löcher um den Motor mit Gewindestangen zwischen die beiden Scheiben zu spannen. Diese Halterung haben wir an die ebenfalls erneuerte Aluminiumscheibe am Heck geschraubt.

2.6 Fazit und Ausblick

Der Rumpfbau gehörte mit Sicherheit zu den zeitintensivsten und umfangreichsten Bereichen unserer Projektgruppe. Gerade in den ersten Monaten gab es viel zu tun, denn schnell war der Punkt erreicht an dem alle anderen Bereiche so weit waren das sie den Rumpf benötigten. Wir hatten geplant das Boot Anfang Dezember 2005 erstmals ins Wasser zu lassen, aber unser Ziel erst Mitte Februar 2006 erreicht. Eine Zeitersparnis von ungefähr 4 Wochen wäre sicherlich möglich gewesen, wenn wir uns frühzeitig zum Kauf von Tauchtanks entschieden hätten. Ansonsten mussten wir feststellen, dass es nicht leicht ist ein U-Boot zu konstruieren, und da sehr viele Überlegungen auch bei kleinen Bauteilen von wichtig sind. Im zweiten Semester standen beim Thema Rumpfbau nur noch die Erweiterungen des Bootes an, die uns aber nicht so viel Zeit gekostet haben. Wir konnten auf die Erfahrungen aus dem ersten Semester zurückgreifen und haben so bei vielen Arbeitsschritten schneller arbeiten können. Desweiteren waren natürlich noch einige Reparaturen durchzuführen und es mussten beispielsweise die Ruder des öfteren nachjustiert werden.
In der Hauptzeit der Bauphase haben wir uns in der Regel 3 Tage pro Woche in der Werkstatt von Karl-Heinz Behrens aufgehalten und dort an unseren Teilen selbstständig gearbeitet. Das Laminieren der Bug- und Hecksektion sowie das Gießen der Ruder haben wir bei Robert im Keller durchgeführt. Den anschließenden Zusammenbau und die Integration haben wir dann in unserem Projektraum gemacht. Es ist allerdings für alle Beteiligten besser, wenn es einen Raum zum Programmieren und einen für den Bau gibt. Die Arbeiten an Metall und Kunststoff machen viel Lärm und vor allem Schmutz, der bei den empfindlichen Elektronikbauteilen möglichst zu vermeiden ist.
Mit denen von uns gesammelten Erfahrungen würden wir mittlerweile das U-Boot anders bauen. Da es eine Nachfolgeprojektgruppe gibt, haben wir noch einen Entwurf erstellt. Auf Tauchtanks würden wir mittlerweile verzichten, da sie die Opertationstiefe des Bootes einschränken und wir im Laufe des Projektes immer mehr Probleme mit der Ermittlung des Füllstandes hatten. Da wir auf der Robocup Messe 2006 die U-Boote der Firma Atlas Elektronik gesehen hatten und von deren Tauchsystem begeistert waren, übernahmen wir das Konzept. Hierbei hat das Boot eine Mittelsektion in der sich horizontal ein Hubpropeller befindet, der das Boot hoch und runter bewegt. Bei diesem Konzept kann das Boot entweder so getrimmt, dass werden das ein ganz geringer Restauftrieb besteht und zum Tauchen der Propeller immer langsam laufen muss. Sicherheitstechnisch ist das die beste Lösung. Energiesparender ist es allerdings das Boot so auszutrimmen, dass es sich ständig im Schwebezustand befindet, da der Propeller dann nur bei gewünschter Tiefenänderung laufen muss. Falls aber vorgesehen ist, dass das Boot die meiste Zeit über Wasser fährt, ist diese Trimmung nicht vorteilhaft.
Der Mittelteil ist mit je einem Magnetverschluss mit dem Schlittenteil des Bugs, bzw. des Hecks verbunden. Desweiteren würden wir Strahlruder auch in das Heck einbauen um die Manövrierfähigkeit des Bootes weiter zu erhöhen.
Wir haben im ersten Semester sehr viel im Bereich Metall und Kunststoffverarbeitung gelernt und auch viel Spaß bei der Arbeit gehabt. Allerdings ist der Informatikanteil kurz gekommen und wir waren froh uns im zweiten Semester mit der Anbindung des Festspeichers, sowie der Kamera beschäftigen zu können.
Zum Schluss bleibt noch zu sagen, dass wir ohne die Unterstützung von Lothar Mentz, Karl-Heinz Behrens und der Uni-Werkstatt den Bau garantiert nicht hätten bewerkstelligen können. Also vielen Dank noch einmal.

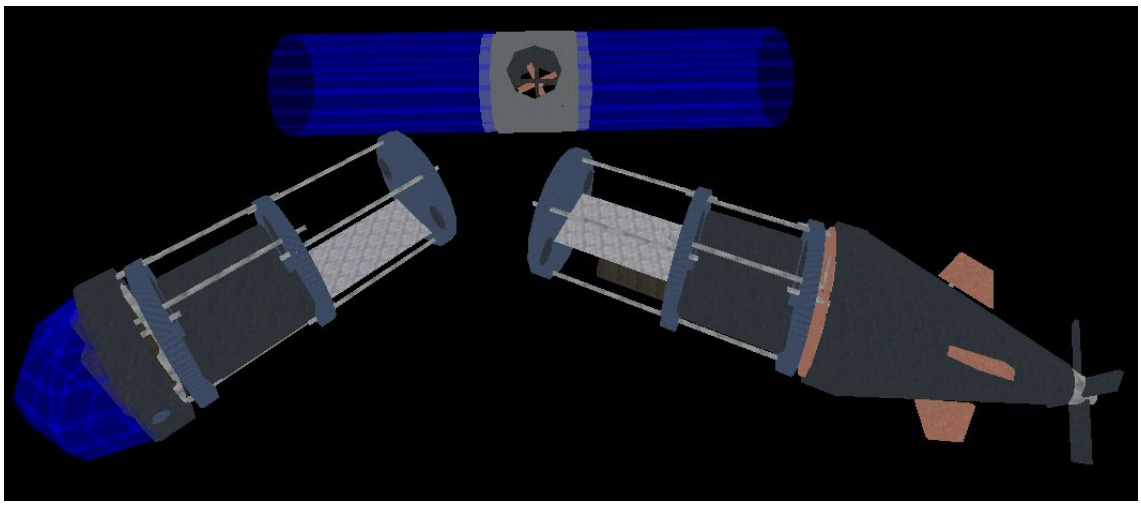


Abbildung 2.27: Bild Konzept für neues U-Boot

Kapitel 3

Inertialnavigation

3.1 Einleitung

Die Navigation beschäftigt sich im allgemeinen mit der Standortbestimmung zu Lande, im Wasser und in der Luft. Es gibt unterschiedliche Arten von Navigation, wie die Navigation nach Sicht, anhand der Gestirne oder auch der Satellitennavigation wie dem GPS.

Für die Bestimmung der Position wird ein Inertialnavigationssystem eingesetzt. Dieses hat den Vorteil, dass für die Messung keine externen Komponenten außerhalb des Uboots erforderlich sind. Zudem kann die absolute Position bestimmt werden, weil sämtliche Bewegungsänderungen registriert werden.

Dieses Thema ist zwei Teile, hier Meilensteine genannt, aufgeteilt. Im ersten Meilenstein werden die Geschwindigkeiten und die Lagen bestimmt. Es werden analoge Sensoren verwendet und die Implementierung erfolgt in C++. Beim zweiten Meilenstein wird noch zusätzlich die Position bestimmt, digitale Sensoren verwendet und die Implementierung erfolgt in der Programmiersprache C. Diese Meilensteine waren jedoch im Laufe des Projekts nicht klar getrennt und haben sich teils uberschnitten.

3.2 Motivation

Andere Navigationsmethoden als die Inertialnavigation wären schwierig oder gar nicht realisierbar. So funktioniert die GPS-Navigation unter der Wasseroberfläche gar nicht, da sich die Funkwellen dort schlecht ausbreiten können. Auch die Koppelnavigation mittels einem Impeller reicht nicht aus, weil die Strömung im Wasser nicht mitgemessen werden kann.

3.2.1 Koppelnavigation

Die Koppelnavigation ist eine recht alte Methode die Position zu bestimmen. Anhand des gefahrenen Kurses und der Geschwindigkeit wird der zurückgelegte Weg bestimmt. Der Kurs wird durch einem Kompass und die Geschwindigkeit durch einen Impeller (Logge) ermittelt. Dies ist dann Geschwindigkeit und Kurs über Wasser. Für den zurückgelegten Weg muss noch die Strömung über eine Vektoraddition mit einberechnet werden.

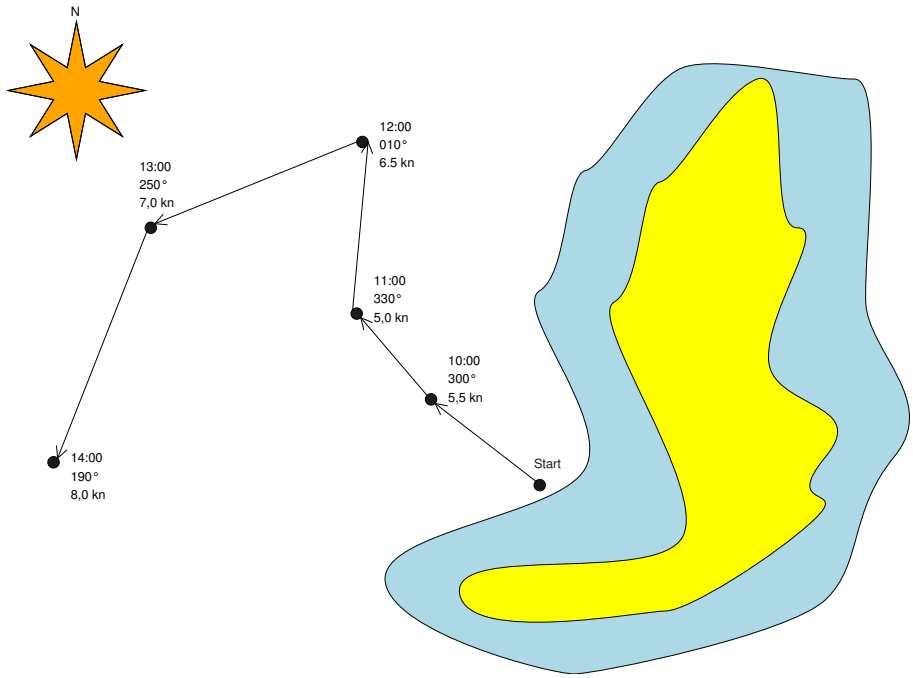


Abbildung 3.1: Diese Karte zeigt eine Fahrt, in der mit der Koppelnavigation navigiert wurde

Abbildung 3.1 zeigt eine Beispielfahrt. Dort wurde zu jeder vollen Stunde der Kurs und die Geschwindigkeit eingetragen. So wurde bis um 10:00 Uhr eine Geschwindigkeit von 5,5 Knoten (Seemeilen pro Stunde) und einen Kurs über Grund von 300 Grad gefahren. Somit ist dann der aktuelle Standort 5,5 Seemeilen mit einen Kurs von 300 Grad vom vorher eingetragenen Standort entfernt (hier ist das der Startpunkt).

Zum Thema Koppelnavigation sei auf [Beh05] verwiesen.

3.2.2 Aufbau der Inertialnavigation

Das Prinzip der Inertialnavigation ist so ähnlich wie bei der Koppelnavigation. Aus der Beschleunigung wird das erste Integral, die Geschwindigkeit, und dann daraus das zweite Integral, die Position bestimmt. Es ist aber eigentlich ein kontinuierliches Aufaddieren von Sensorwerten, da, wie später erklärt wird, numerisch integriert werden muss.

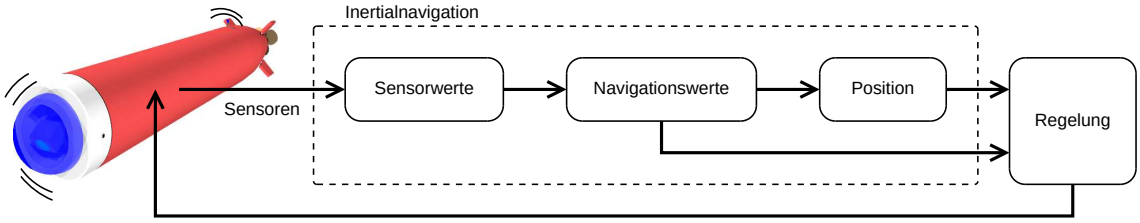


Abbildung 3.2: Aufbau der Inertialnavigation und Einsatz im Uboot

Aus den Sensorwerten werden die Navigationswerte wie Geschwindigkeit, Kurs und Tiefe bestimmt. Aus diesen Daten kann dann eine Position bestimmt werden. Die Navigationswerte und die Position werden an die Regelung übergeben. Diese steuert dann das Uboot. Die Reaktion des Uboots wird wieder von den Sensoren gemessen. Abbildung 3.2 zeigt die Inertialnavigation in diesen Regelkreis.

Im Folgenden wird zwischen Sensorwerten und Navigationswerte unterschieden. Die Sensorwerte sind die direkt ausgelesenen 12 Bit-Werte aus dem AD-Wandler. Diese werden dann mit x gekennzeichnet. Anhand der Kalibrierwerte, wie der Empfindlichkeit und dem Offset, werden die Navigationswerte berechnet. Dazu gehören die Winkeländerungen, der Kurs, die Geschwindigkeit und auch die Position.

Eine Alternative zur Inertialnavigation wäre die Ortung durch Ultraschallsatelliten. Dabei wird die Laufzeit der Schallwellen vom Ultraschallgeber zum Uboot bestimmt. Daraus lässt sich dann eine Position bestimmen. Diese Methode ist zwar genauer, aber eventuell für einen späteren Zeitpunkt geplant.

3.3 Konzeptfindung / Theorie

Weil sich das Uboot unter Wasser im dreidimensionalen Raum bewegen kann, sind alle 6 räumlichen Freiheitsgrade zu bestimmen:

- Translatorische:
  - x-Achse: Längs des Rumpfes, in Richtung des Bugs
  - y-Achse: Breitseite des Rumpfes,in Richtung Steuerbord
  - z-Achse: Tiefe des Bootes
- Rotatorische:
  - α : Krängung des Bootes (Rotation um die x-Achse)
  - β : Tiefenlage des Bootes (Rotation um die y-Achse)
  - γ : Kurs der Bootes (Rotation um die z-Achse)

Die Rotationen erfolgen im Uhrzeigersinn, mit Blick in Richtung der Achse. Die Achsen sind in den Abbildungen 3.3, 3.4 und 3.5 zu sehen.

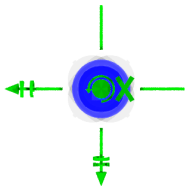


Abbildung 3.3: X-Achse vom Uboot (Ansicht von Vorn)

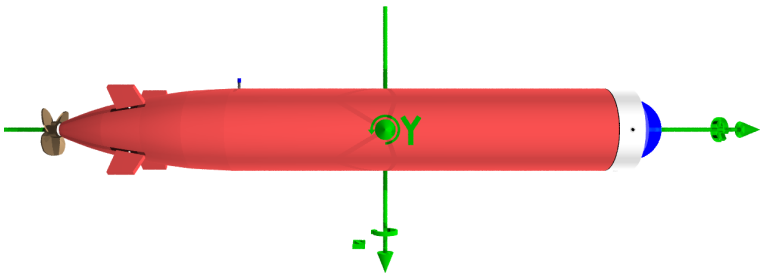


Abbildung 3.4: Y-Achse vom Uboot (Ansicht von der Steuerbordseite)

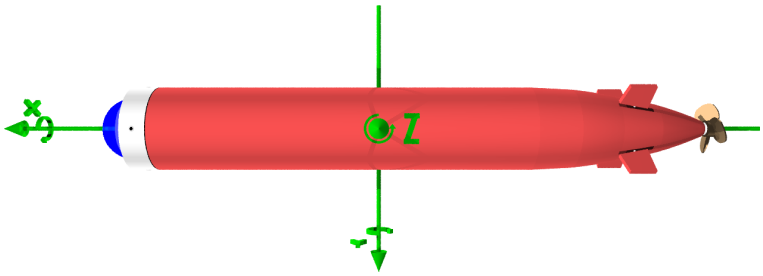


Abbildung 3.5: Z-Achse vom Uboot (Ansicht von Unten)

3.3.1 1. Meilenstein: Fernsteuerbare Tauchplattform mit Tiefen,- Lage-, Fahrtenregelung

Hier soll das Uboot mit einer Fernsteuerung gesteuert werden und eine einfache Tiefen- und Lageregelung haben. Es soll der Kurs, die Geschwindigkeit, die horizontale Lage und die Tiefe bestimmt werden.

Die Tiefe lässt sich direkt aus dem Drucksensor linear berechnen. Die Geschwindigkeit ist eine fortlaufende Addition der Beschleunigungswerte. Dasselbe wird mit der Lage und dem Kurs anhand der Gyroskope gemacht.

Bestimmung des Berechnungsintervalls

Das Intervall zwischen zwei Positionsbestimmungen sollte möglichst klein sein. Ist es zu groß, so können zum Beispiel Geschwindigkeitsänderungen im Intervall nicht erfasst werden (Abbildung 3.6). Dabei gilt, je größer die Geschwindigkeitsänderung, desto mehr Informationen gehen verloren und desto größer wird die Abweichung. Theoretisch wäre ein unendliches Intervall optimal. Dies ist aber aufgrund der zeitlich begrenzten Rechengeschwindigkeit nicht möglich. Außerdem können bei zu kleinen Intervallen zu geringe Aufaddierungen entstehen , so dass numerische Auslöschungen auftreten.

Die untere Intervallgrenze ist auch durch die Rauschunterdrückung beschränkt. Es müssen, wie in 3.4.3 beschrieben, 50 Werte eingelesen werden. Das minimale Intervall, in der ein Task auf dem Nios ausgeführt werden kann, beträgt 1 ms. Um auch andere Tasks (zum Beispiel die Regelung) noch ausführen zu können, wird nur alle 5 ms ein Sensorwert eingelesen. Das heißt, dass das Intervall t_i mindestens

$$t_i = 5ms * 50 = 250ms \tag{3.1}$$

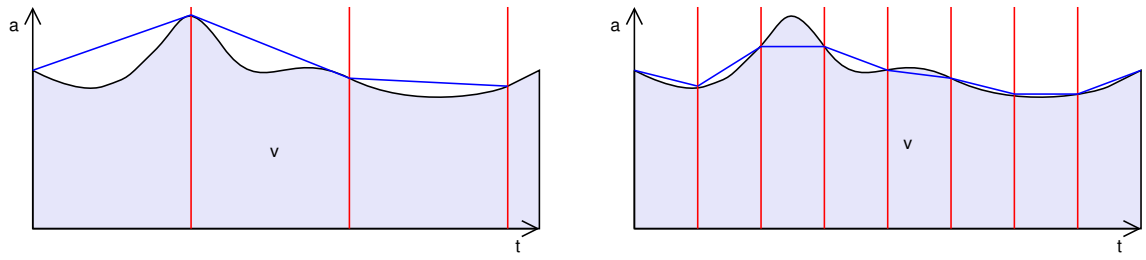


Abbildung 3.6: Unterschied der Genauigkeit zwischen einen größeren Intervall (links) und einen kleineren Intervall(rechts)

betragen muss.

Beim Kalman-Filter (Abschnitt 3.4.3) brauchen vorher keine Werte eingelesen werden. Deshalb ist das Berechnungsintervall nur durch die Rechenleistung begrenzt.

Bestimmung der Geschwindigkeit

Im erstem Meilenstein wird die Geschwindigkeit in die X-Richtung bestimmt. Der Geschwindigkeitsvektor ist entlang der x-Achse. Das heißt es wird die "Fahrt durch Wasser"gemessen. Oder anders ausgedrückt, steht das Uboot senkrecht nach oben und hat eine Geschwindigkeit von 1 m/s, so fährt es mit 1 m/s nach oben.

Die Geschwindigkeit ist eigentlich das 1. Integral der Beschleunigung:

$$v(t) = \int_0^t a(s)ds \tag{3.2}$$

Nun ist $a(s)$ unbekannt, da es von den äußeren Bedingungen abhängt, noch ist unklar, ob es sich leicht integrieren lässt. Deshalb muss $a(s)$ numerisch integriert werden. Das geschieht, in dem alle gemessenen Sensorwerte addiert werden. Wie oben beschrieben, liegt das Optimum bei einen möglichst kleinen Intervall.

Die Geschwindigkeit ist dann im jeden Intervall t_I wie folgt berechenbar:

$$v_x(i) = v_x(i-1) + a_x * t_I \tag{3.3}$$

$v_x(i)$ ist die aktuelle Geschwindigkeit und $v_x(i-1)$ die vorige berechnete. a_x ist die gemessene Beschleunigung in die x-Richtung. Die Einheiten sind in m/s, bzw m/s^2 und s.

Diese Formel ist nur gültig, wenn das Boot horizontal liegt. Auf den Sensor wirkt neben der Beschleunigung auch die Gravitation g ein (Abbildung 3.7). Das hat zur Folge, dass wenn der Sensor oder das Uboot senkrecht steht, ein g abgezogen werden muss. Ist es waagerecht, wirkt die Gravitation darauf nicht ein. Dieser Fehler hängt kreisförmig von der orthogonalen Lage β ab, so dass man $g * sin(\beta)$ von a_x subtrahieren muss [Ana04a].

$$a_x = a_{gemessen\_x} - g * sin(\beta) \tag{3.4}$$

Dieselben Regeln gelten auch für die Geschwindigkeit in die y-Richtung.

$$v_y(i) = v_y(i-1) + a_y * t_I \tag{3.5}$$

$$a_y = a_{gemessen\_y} - g * sin(\alpha) \tag{3.6}$$

Bestimmung der Rotationen

Wie bei der Geschwindigkeit, ist der Kurs γ, der ein Winkel ist, das Integral der Winkeländerung ω_z:

$$\gamma(t) = \int_0^t \omega_z(s)ds \tag{3.7}$$

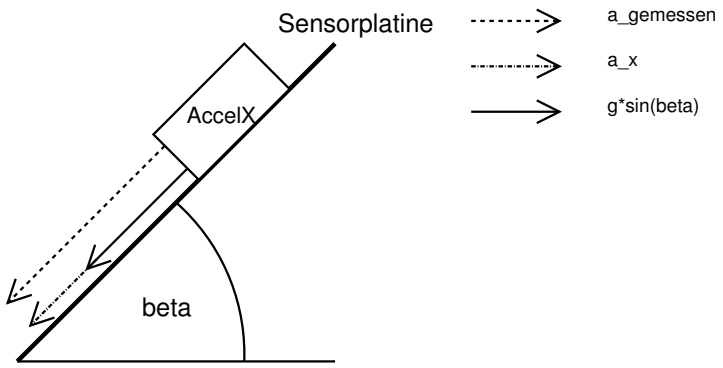


Abbildung 3.7: Einfluss der Gravitation auf die Sensorplattform

Hier muss auch numerisch integriert werden, da dieselben Bedingungen vorhanden sind, wie bei der Geschwindigkeit.

$$\gamma(t) = \gamma(t-1) + \omega_z * t_I \tag{3.8}$$

Für die horizontale Lage β und die vertikale Lage α gilt dasselbe:

$$\alpha(t) = \alpha(t-1) + \omega_x * t_I \tag{3.9}$$
$$\beta(t) = \beta(t-1) + \omega_y * t_I \tag{3.10}$$

Bestimmung des Sinus und Cosinus

Bei der Berechnung der Beschleunigung wird die Sinusfunktion benötigt (siehe Formel 3.4). Diese ist in den Standardbibliotheken auf dem Nios nicht vorhanden. Deshalb muss sie selber implementiert werden.

Der Sinus kann auch als Taylorreihe geschrieben werden:

$$sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \tag{3.11}$$
$$= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - ... \tag{3.12}$$

In der Programmiersprache C implementiert, würde es so aussehen:

```
1  double z = x, y; // y = sin(x)
2  y  = x; z *= x*x; //1!
3  y -= z/6; z *= x*x; //3!
4  y += z/120; z *= x*x; //5!
5  y -= z/5040; z *= x*x; //7!
6  y += z/362880; z *= x*x; //9!
7  y -= z/39916800; z *= x*x; //11!
8  y += z/6227020800.0; z *= x*x; //13!
9  y -= z/1307674368000.0; z *= x*x; //15!
10 y += z/355687428096000.0; z *= x*x; //17!
11 y -= z/121645100408832000.0; z *= x*x; //19!
12 ...
```

Dabei ist zu beachten, dass die Einheit von x noch nicht in Grad ist, sondern den des Einheitskreises (rad) entspricht. x muss deshalb umgeformt werden:

$$x = \frac{\pi}{360} x \tag{3.13}$$

Je größer die Iteration wird, desto genauer wird die Funktion für im Betrag große Werte. Die hier verwendeten Winkel liegen im Bereich von -180° bis 180°. In der Praxis werden sie aber nie über $\pm$ 90° hinaus gehen. Deshalb reichen hier 3 Iteration. Diese haben eine Genauigkeit von 3 Stellen hinter dem Komma im Bereich von $\pm$90°.

Für die Positionsbestimmung ist auch der Cosinus nötig. Man könnte den Winkel um 90° nach rechts verschieben und dann die oben genutzte Sinusfunktion verwenden:

$$cos(x) = sin(x + 90) \tag{3.14}$$

Jedoch haben die Cosinuswerte bei 180° einen Fehler von ungefähr 2,5. Deshalb wird hier auch die Taylereihe für den Cosinus angewendet:

$$cos(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} \tag{3.15}$$

$$= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - ... \tag{3.16}$$

Die C-Code sieht entsprechend so aus:

```
1 double z = x*x, y = 1;// y = cos(x)
2 y -= z/2; z *= x*x; //2!
3 y += z/24; z *= x*x; //4!
4 y -= z/720; z *= x*x; //6!
5 y += z/40320; z *= x*x; //8!
6 y -= z/3628800; z *= x*x; //10!
7 y += z/479001600.0; z *= x*x; //12!
8 y -= z/87178291200.0; z *= x*x; //14!
9 ...
```

3.3.2 2. Meilenstein: Autonomes Fahren

Nach der Tiefen- und Lageregelung soll eine autonome Wegefindung implementiert werden. Dazu ist die Bestimmung der aktuellen Position nötig.

Hier soll die Anwendung von 4x4 Matrizen (nach Denavit-Hardenburg), wie in der Robotik ([Hei04]) bekannt, stattfinden, um die Position im Raum zu bestimmen. Mit einer kontiniuierlichen Matrizenmultiplikation kann man so alle Positionen und Lagen bestimmen.

Desweiteren wird ein Simulationsprogramm entwickelt, welches anhand der gemessenen Sensorwerte den zurückgelegten Weg berechnet. Durch den Vergleich mit dem realen zurückgelegten Weg können so die Sensorwerte feiner kalibriert werden.

Positionsbestimmung

Für die Positionsbestimmung ist ein globales Koordinatensystem, auch Basiskoordinatensystem (BKS) genannt, nötig. Damit wird angegeben, wie die Dimensionen definiert sind. Die Dimensionen sind mit x,y und z gekennzeichnet.

Die Tiefe ist die z-Achse. Die x- und y-Koordinaten geben die Positon planar über den Boden an. Abbilung 3.8 zeigt wie das Uboot im Basiskoordinatensystem aussieht.

Die Anfangsposition wird beim Start vorgeben. Neben den Koordinaten müssen auch die Lagen angegeben werden. Diese sind in Abschnitt 3.3 definiert. Damit besteht die Positonsangabe aus einen 6-dimensionalen Vektor:

$$P = (x, y, z, \alpha, \beta, \gamma) \tag{3.17}$$

Die x- und y-Koordinaten werden über das zweite Integral der Beschleunigungen, die z-Koordinate über den Drucksensor und die Lagen über die Gyroskope bestimmt. Leider können die Elemente des Vektors nicht unabhängig voneinander berechnet werden. Hat zum Beispiel das Uboot einen Kurs von 90°, so wird die y-Koordinate über den x-Beschleunigungssensor bestimmt.

In Abbildung 3.9 zeigt ein Beispiel, wie sich die Achsen des Uboots während der Fahrt im BKS bewegen können. Die Verbindung zwischen den Ubooten kann man als lokale Positionsänderung zwischen zwei Berechnungsintervallen sehen. Dies kann auch als Positionsangabe beschrieben werden. Es wird dann zwischen lokaler Postion, also der Positionsänderung, und der globalen Postionen, die im Basiskoordinatensystem liegt, unterschieden.

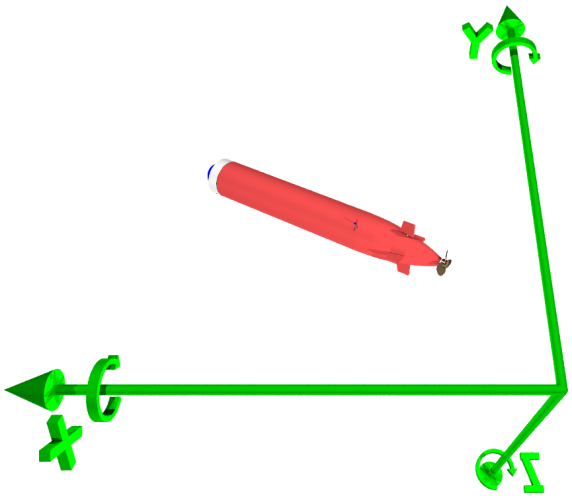


Abbildung 3.8: Das Uboot im Basiskoordinatensystem

Nun muss ein Verfahren gefunden werden, welches von einer gegeben globalen Postion P_k und der lokalen Postion L_k auf die neue globale Postion P_{k+1} schließen lässt:

$$P_{k+1} = P_k \oplus L_k \tag{3.18}$$

Die Positionsangaben lassen auch als 4x4-Matrizn beschreiben. Diese haben die Form

$$\begin{pmatrix} R_{xx} & R_{xy} & R_{xz} & P_x \\ R_{yx} & R_{yy} & R_{yz} & P_y \\ R_{zx} & R_{zy} & R_{zz} & P_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.19}$$

, wobei R die Stellungen der Achsen des Objekts als Normalvektoren und P_i die Koordinaten beschreibt. Der Vorteil ist, dass mit Matrizenmultiplikationen Objekte von einen Koordinatensystem ins andere transferiert werden können. Zum Beispiel ist Objekt A im BKS ($^{BKS}T_A$) und Objekt B in Relation zu A ($^{A}T_B$) gegeben (Abbildung 3.10). Um $^{BKS}T_B$ zu bestimmen, werden beide Matrizen multipliziert:

$$^{BKS}T_B = {}^{BKS}T_A \cdot {}^{A}T_B \tag{3.20}$$

In Abbildung 3.10 beträgt beispielsweise

$$^{BKS}T_A = \begin{pmatrix} 1 & 0 & 0 & 4 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix}, {}^{A}T_B = \begin{pmatrix} 0 & 0 & -1 & 2 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.21}$$

Dann ist

$$^{BKS}T_B = {}^{BKS}T_A \cdot {}^{A}T_B \tag{3.22}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 4 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & -1 & 2 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.23}$$

$$= \begin{pmatrix} 0 & 0 & -1 & 6 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.24}$$

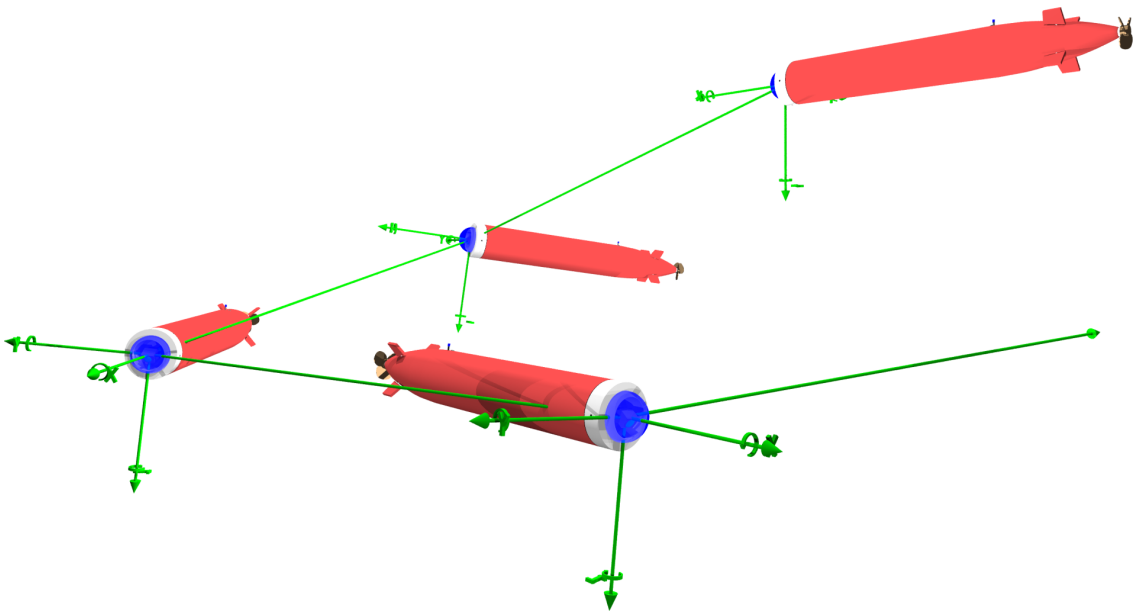


Abbildung 3.9: Eine Fahrt im BKS

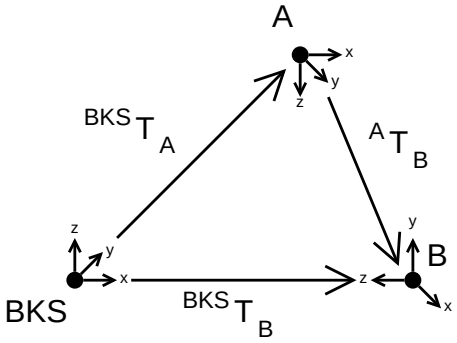


Abbildung 3.10: Transformation von Objekt B im Koordinatensystem A zum BKS

Das kann man auch zwischen lokaler Positionsänderung und globaler Positon machen:

$$P_{k+1} = P_k \cdot L_k \tag{3.25}$$

Damit kann man fortlaufend bei gegebener Startposition P_0 die Postion im jeden Berechnungsintervall bestimmen:

$$P_{k+1} = P_0 \cdot L_1 \cdot L_2 \cdot ... \cdot L_k \tag{3.26}$$

In Abbildung 3.11 sind diese Positonsangaben.zu sehen. Die lokale Position ist die Änderung der Lage und der Position in einem Berechnungsintervall.

Für die 4x4-Matriz muss noch der R-Anteil bestimmt werden. Da die Neigungen für jede Achse einzeln bestimmt werden können, wird nach Yaw-Pitch-Roll bestimmt. Für jede Lage kann man die Rotationsmatrizen definieren:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{pmatrix} \tag{3.27}$$

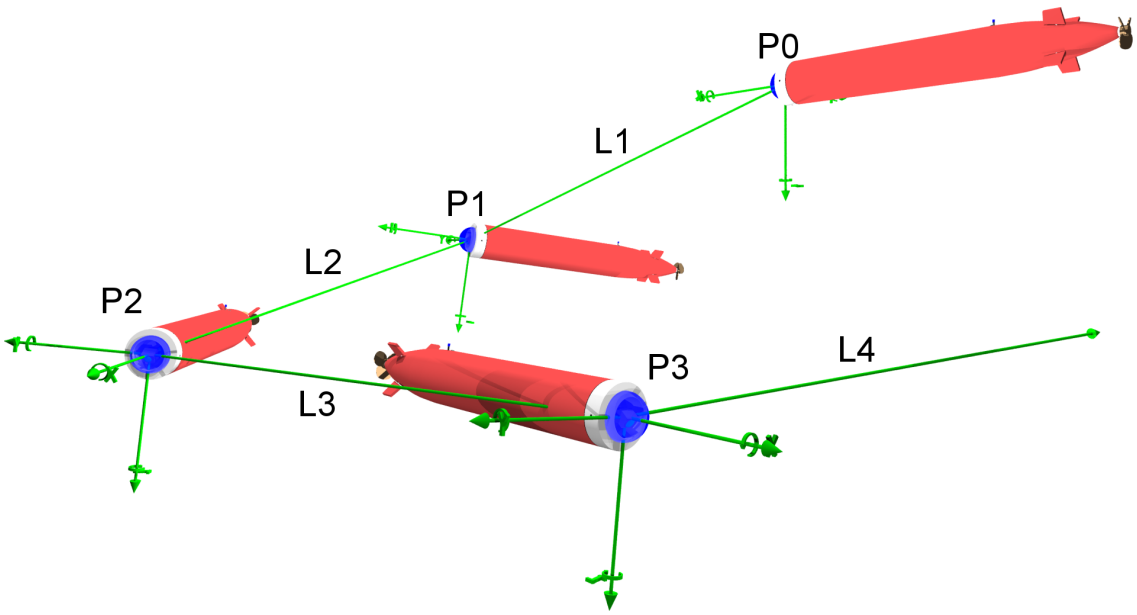


Abbildung 3.11: Berechnung der Fahrt im BKS

$$R_y = \begin{pmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{pmatrix} \tag{3.28}$$

$$R_x = \begin{pmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.29}$$

Zusammen ergibt das

$$R_{ges} = R_x R_y R_z \tag{3.30}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{pmatrix} \begin{pmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{pmatrix} \begin{pmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.31}$$

$$= \begin{pmatrix} cos(\beta) & 0 & sin(\beta) \\ sin(\alpha)sin(\beta) & cos(\alpha) & -sin(\alpha)cos(\beta) \\ -cos(\alpha)sin(\beta) & sin(\alpha) & cos(\alpha)cos(\beta) \end{pmatrix} \begin{pmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.32}$$

$$= \begin{pmatrix} cos(\beta)cos(\gamma) & -cos(\beta)sin(\gamma) & sin(\beta) \\ sin(\alpha)sin(\beta)cos(\gamma) + cos(\alpha)sin(\gamma) & -sin(\alpha)sin(\beta)sin(\gamma) + cos(\alpha)cos(\gamma) & -sin(\alpha)cos(\beta) \\ -cos(\alpha)sin(\beta)cos(\gamma) + sin(\alpha)sin(\gamma) & cos(\alpha)sin(\beta)sin(\gamma) + sin(\alpha)cos(\gamma) & cos(\alpha)cos(\beta) \end{pmatrix} \tag{3.33}$$

Wird die lokale Positionsänderung so definiert:

$$L = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} & P_x \\ R_{yx} & R_{yy} & R_{yz} & P_y \\ R_{zx} & R_{zy} & R_{zz} & P_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.34}$$

dann berechnen sich die Werte entsprechend:

$$
\begin{aligned}
R_{xx} &= cos(\beta)cos(\gamma) && (3.35)\\
R_{xy} &= -cos(\beta)sin(\gamma) && (3.36)\\
R_{xz} &= sin(\beta) && (3.37)\\
R_{yx} &= sin(\alpha)sin(\beta)cos(\gamma) + cos(\alpha)sin(\gamma) && (3.38)\\
R_{yy} &= -sin(\alpha)sin(\beta)sin(\gamma) + cos(\alpha)cos(\gamma) && (3.39)\\
R_{yz} &= -sin(\alpha)cos(\beta) && (3.40)\\
R_{zx} &= -cos(\alpha)sin(\beta)cos(\gamma) + sin(\alpha)sin(\gamma) && (3.41)\\
R_{zy} &= cos(\alpha)sin(\beta)sin(\gamma) + sin(\alpha)cos(\gamma) && (3.42)\\
R_{zz} &= cos(\alpha)cos(\beta) && (3.43)\\
P_x &= \Delta s_x && (3.44)\\
P_y &= \Delta s_y && (3.45)\\
P_z &= 0 && (3.46)
\end{aligned}
$$

P_0 aus Formel 3.26 wird beim Start vorgeben (Initiale Position).

Bestimmung der Tauchgeschwindigkeit

Beim statischen Tauchen braucht die Regelung auch die Angabe der Tauchgeschwindigkeit v_z. Diese ist erste Ableitung der Tiefe d:

$$v_z = \frac{\delta d}{\delta t} \tag{3.47}$$

Diese Operation muss auch hier numerisch durchgeführt werden. Es ist die Differenz zwischen der vorigen Tiefe $d(i-1)$ und der aktuellen Tiefe $d(i)$ geteilt durch das Intervall t_I:

$$v_z(i) = \frac{d(i) - d(i-1)}{t_I} \tag{3.48}$$

3.4 Aufbau

Damit die benötigten Sensorwerte gemessen werden können, bedarf es natürlich auch Sensoren. Diese werden auf einer Platine angebracht und zentral im Uboot eingebaut. Sie wird dann an den FPGA angeschlossen. Aufgebaut ist die Navigationsplattform als ein *Strapdown System* [Law01], das heißt es ist fest mit dem Uboot verbunden und nicht kardanisch aufgehängt. Das erspart ein mechanischen Aufbau und dem damit verbundenen Platz. Die zusätzlichen Berechnungen fallen durch den FPGA nicht so sehr ins Gewicht.

3.4.1 Verwendete Bauteile

Um die Tiefe zu messen wird ein absoluter Drucksensor verwendet. Die Bewegungen in x- und y-Richtung werden mit einem Beschleunigungssensor gemessen. Für die Neigung ist für jede Achse je ein Gyroskop zuständig. Man könnte auch die z-Achse (Tiefe) mit einem Beschleunigungssensor messen. Da diese jedoch direkt aus dem Druck abgeleitet werden, wird aufgrund der größeren Genauigkeit auf dem zusätzlichen Beschleunigungssensor verzichtet.

Aufbau und Position der Sensoren und der Achsen siehe Abbildung 3.12. Im ersten Meilenstein wurden analoge Sensoren eingesetzt. Deren Werte wurden dann von einem AD-Wandler in digitale Signale umgewandelt. Dagegen wurden im zweiten Meilenstein digitale Sensoren verwendet. Diese haben einen eingebauten AD-Wandler. Dadurch wird eine weniger große Empfindlichkeit gegenüber äußeren Störquellen und ein geringeres Rauschen erhofft.

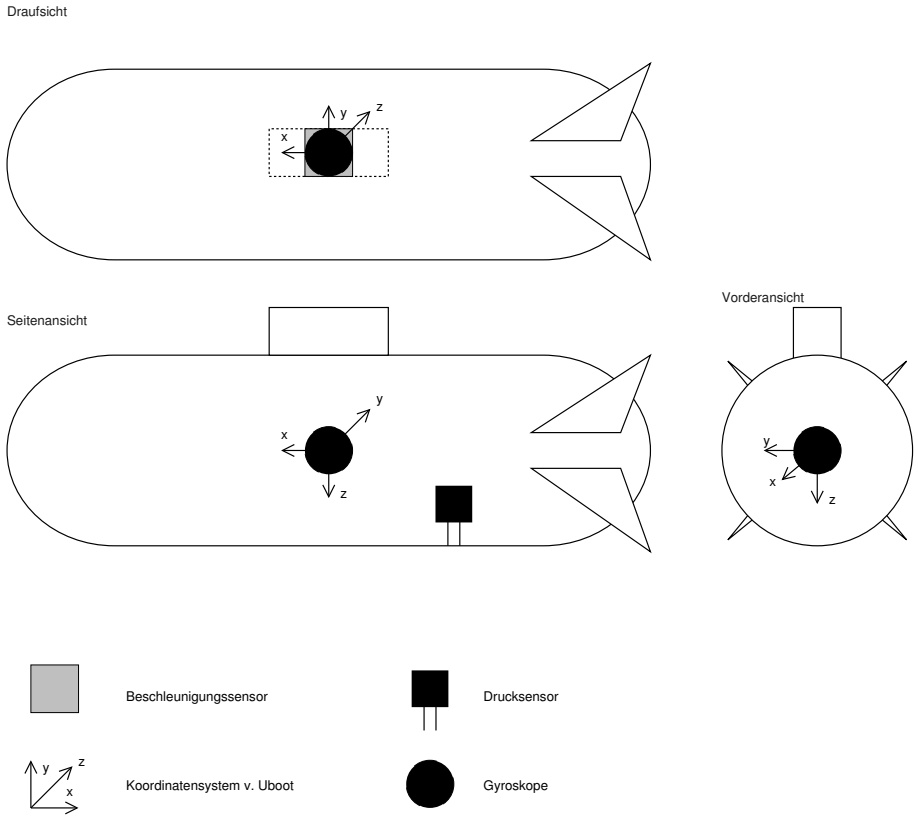


Abbildung 3.12: Positionen der analogen Sensoren im Uboot

Beschleunigungssensoren

Da die Abweichung der berechneten Position zur wirklichen Position mit der Zeit durch das zweifache Integrieren quadratisch zunimmt, sollten die Beschleunigungssensoren eine möglichst hohe Genauigkeit haben. Das heißt, es muss der richtige Messbereich gewählt werden. Ist dieser zu groß, werden kleinste Beschleunigungen nicht mehr erfasst. Ist er zu klein, so werden zu große Beschleunigungen (zum Beispiel durch einen starken Motor oder einer Kollision) auch nicht mehr berücksichtigt.

Es wird erstmal davon ausgegangen, dass maximal 2 g auf das Boot im Normalbetrieb einwirken. Somit wurde als Beschleunigungssensor der Analog Device ADXL203CD [Ana04a] ausgewählt. Er hat einen Messbereich von $\pm$ 1,7 g und kann die Beschleunigung in zwei Richtungen, x und y, messen. Der Ausgang ist Analog, so dass ein AD-Wandler nötig ist. Die Ausgangsspannung beträgt 0-5V.

Gepackt ist der Sensor in einen 8-poligen CLCC (Ceramic Leadless Chip Carrier) mit Ausmaßen von 5 x 4,5 mm. Diese Größe lässt sich mit herkömmlichen Methoden schwer einlöten. Es gibt zwar ein Entwicklerboard im DIL-Format ([Ana04b]), dieses war jedoch zum Zeitpunkt der Bestellung noch nicht verfügbar gewesen. Dennoch konnte es als Referenz für unseren Entwurf verwendet werden.

Als digitaler Sensor wird der ADIS16003 von Analog Devices ([Ana05]) verwendet. Er sitzt auf dem Evaluation Board ADIS16003PCB ([Ana06a]). Er hat den selben Messbereich wie der analoge Sensor, jedoch ist hier ein AD-Wandler miteingebaut.

Leider gabs bei diesen Sensor auch einige Probleme. So klappte das Auslesen der Sensorwerte nicht auf Anhieb. Erst mit einer weiteren Anschlussbelegung gelang es. Nun liegt ein Wert von 12 Bit Länge vor. Jedoch haben die Werte einen Abstand von 32 LSB bei gleichen Messbereich wie die analogen Sensoren. Dies ist nicht ausreichend für eine genaue Geschwindigkeitbestimmung. Hier liegt wahrscheinlich noch ein elektronisches Problem vor.

Gyroskope

Die rotatorischen Sensorwerte werden mit Hilfe von drei Gyroskopen gemessen. Eingesetzt werden Gyroskope von Typ Analog Devices ADXRS300EB [Ana04c]. Diese messen Winkeländerungen von bis zu 300 °/s und haben eine Ausgangsspannung von 0-5 V.

Sie sind in einen 32-Lead Chip Scale Grid Array (32-Lead BGA) gepackt. Der Chip hat eine Breite von nur 7 mm. Glücklicherweise ist dafür, im Gegensatz zum Beschleunigungssensor, das Entwicklerboard ([Ana03]) verfügbar. Es ist eine kleine Platine im Format 32-DIL. Auf ihr sind alle benötigten Kondensatoren vorhanden, so dass man direkt die Werte messen kann.

Der Gyro für die z-Achse liegt in der Mitte der Platine, bzw. in der Mitte des Uboots. Da die Gyroskope für die x- und y- Achse zum rechten Winkel zu einander und der Platine sein müssen, werden gewinkelte Sockel eingesetzt, die zum Beispiel auch für LED-Anzeigen verwendet werden.

Hier ist der digitale Sensor ein ADIS16100 von Analog Devices ([Ana06b]) auf dem Evaluation Board ADIS16100PCB ([Ana06c]). Auch ist hier der Messbereich derselbe, wie beim analogen Sensor. Es gab auch hier ähnliche Probleme wie beim digitalen Beschleunigungssensor. Diese konnten aber zufriedenstellend gelöst werden.

Drucksensor

Um die Tiefe zu messen, wird der Drucksensor Motorola MPX 4250AP [Mot98] verwendet. Dieser ist ein Absolutdrucksensor mit einen Messbereich von 0,2 bis 2,5 Bar. Verbunden mit dem Wasser nach draußen wird er über einen Schlauch, der mit Luft gefüllt ist. Durch die Kappilareffekte bleibt immer eine Luftblase am Drucksensor. Die Ausgangsspannung liegt im Bereich von 0-5V, so dass der Drucksensor direkt am AD-Wandler angeschlossen werden kann.

Der Drucksensor wird aber auf einer extra Platine gebaut. Es bringt den Vorteil, dass er räumlich von anderen Geräten getrennt werden kann. Durch die Verbindung nach draußen ist ein Risiko des Wassereinbruchs vorhanden, wenn zum Beispiel der Schlauch abrutscht. Die Eingangsspannung wird auf der Platine noch stabilisiert.

Im zweiten Meilenstein wird ein digitaler Drucksensor von Typ SMI SM-5822-030-A-B [Sil05] angeschlossen. Dieser wird über einem I^2C-Bus angeschlossen. Es ist ein Absolutdrucksensor mit einem Messbereich von 0- 2,068 Bar. Der Anschluss an den Schlauch erfolgt genauso wie beim analogen Drucksensor, nur ist hier der Schlauch mit 3 mm Durchmesser etwas kleiner. Um den Innendruck zu messen, wird noch ein Sensor gleichen Typs eingesetzt.

AD-Wandler

Die analogen Sensoren geben nur Ausgangssignale im Bereich von 0 bis 5 V aus. Damit der FPGA die Sensorwerte auswerten kann, muss noch ein AD-Wandler dazwischen geschaltet werden. Hier kommt der MAX 186 [Max96] in Betracht. Er hat 8 Eingänge und eine Auflösung von 12 Bit. Die Eingänge sind wie folgt belegt (Tabelle 3.1):

Eingang	Signal	Bemerkung	Variable	Controlbyte
0	PRESSURE	Drucksensor	SENSOR_PRESSURE	0x8E
1	GYRO_Z_OUT	Gyroskop an der z-Achse	SENSOR_GYRO_Z	0xCE
2	ACCEL_Y_OUT	Beschleunigung in der y-Achse	SENSOR_ACCEL_Y	0xDE
3	ACCEL_X_OUT	Beschleunigung in der x-Achse	SENSOR_ACCEL_X	0x9E
4	GYRO_Y_OUT	Gyroskop an der y-Achse	SENSOR_GYRO_Y	0xAE
5	EXT_IN_2	Externer Eingang	SENSOR_IN_2	0xEE
6	EXT_IN_1	Externer Eingang	SENSOR_IN_1	0xBE
7	GYRO_X_OUT	Gyroskop an der x-Achse	SENSOR_GYRO_X	0xFE

Tabelle 3.1: Belegung der Eingänge am AD-Wandler

Die Verbindung zum FPGA erfolgt über den SPI-Bus. Um die Werte aufzurufen, wird die Funktion *alt_32 subEx_getSensorData(alt_u8 channel)* bereitgestellt. Für den Parameter *channel* gibt es Präprozessorvariablen, die die Controlbytes enthalten. Die Funktion sendet das Controlbyte an den AD-Wandler und erhält die beiden Receivebytes, welche den Sensorwert in einer Auflösung von 12 Bit enthält.

Der Zugriff auf die digitalen Sensoren erfolgt im zweiten Meilenstein über die Funktionen *Sensors_get<Bezeichnung des Sensors>()*, da diese hier an unterschiedliche Bussen angeschlossen sind. Die Gyroskope haben je einen eigenen SPI-Kanal, während die Beschleunigungssensoren ihr eigenes Sendewort haben, um die Daten anzufordern. Zudem sind die Drucksensoren am I^2C-Bus angeschlossen, was einen ganz anderen Zugriff erfordert.

3.4.2 Aufbau Sensoreinheit

Abbildung 3.13 zeigt den Schaltplan der Hauptplatine und Abbildung 3.14 das Layout.

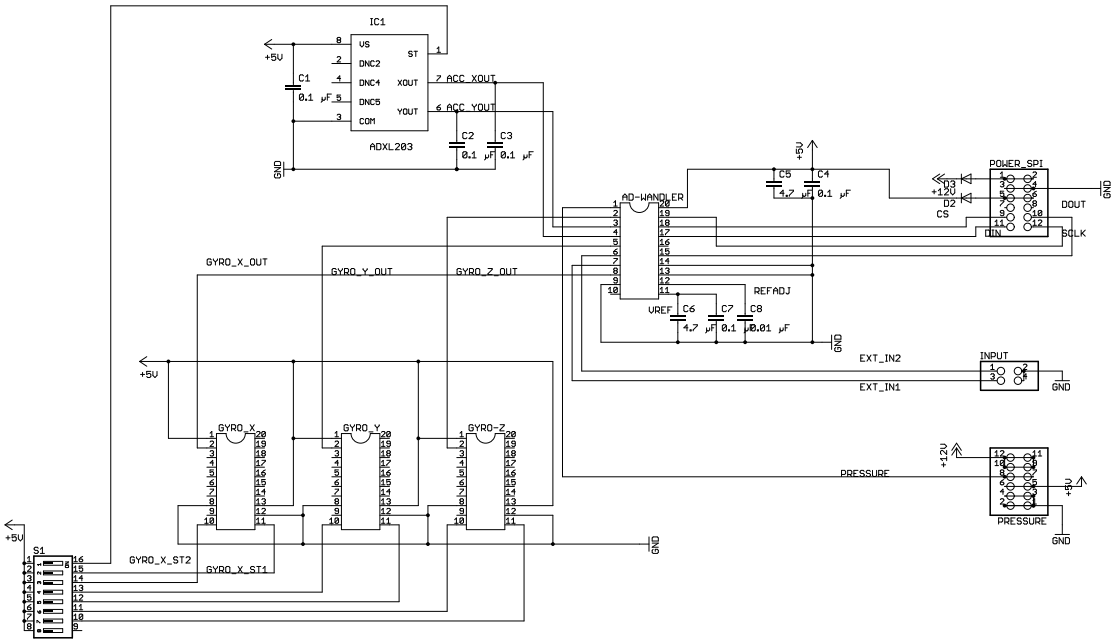


Abbildung 3.13: Schaltplan der Hauptplatine für die analogen Sensoren

Für den zweiten Meilenstein müssen die Signale stabilisiert werden. Dazu war der Einbau von zwei parallelen Kondensatoren mit 100 nF und 10 μF zur Stabilisierung der Eingangsspannung in Nähe jedes Sensors geplant. An den sogenannten *Self-Test*-Eingängen der Gyroskope kommen zwei Widerstände zu je 1kOhm. Die Versorgungsspannung kann mit einen Spannungsregler vom Typ UA7805 geregelt werden. Durch die digitalen Sensoren ist es nicht mehr nötig. Es sind nur noch Schutzdioden nötig, um die Sensoren vor falsche Spannung zu schützen.

Wichtig für den Aufbau ist, dass die Achsen miteinander orthogonal sind. Das heißt, das zentral der Gyro für die Z-Achse und der Beschleunigungssensor angebracht sind. Die Gyroskope für die x- und y-Achse gehen rechtwinklig von dem Mittelpunkt ab. (Abbildung 3.15).

Dadurch entfallen komplexe Umrechnungen von einen anderen nicht orthogonalen Koordinatensystem. Der Beschleunigungssensor und der z-Gyro sollten sich idealerweise an derselben Stelle befinden. Da dies räumlich nicht möglich ist, ist der Beschleunigungssensor auf der Rückseite der Platine angelötet.

Im Uboot sollte die Platine so angebracht werden, dass sich der z-Gyro in der Mitte des Uboots befindet. Der y-Gyro ist dann in der Mitte der Höhe vom Uboot angebracht.

Beim Drucksensor sollte beachtet werden, dass der Schlauchausgang nach unten zeigt. Damit kann der Wasserdruck auch im aufgetauchten Zustand gemessen werden.

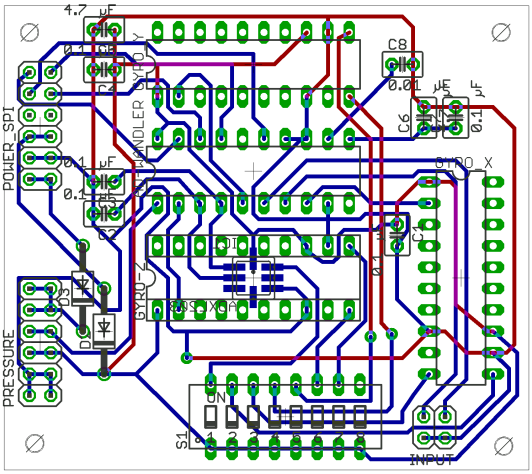


Abbildung 3.14: Layout der Hauptplatine für die analogen Sensoren

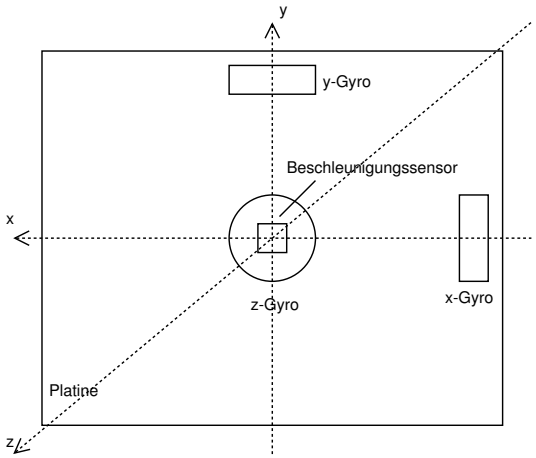


Abbildung 3.15: Aufbau der Platine mit den Gyroskopen und dem Beschleunigungssensor

Die digitalen Sensoren haben eine eigene Platine und können direkt an dem FPGA angeschlossen werden, Deshalb ist eine zentrale Platine nicht notwendig. Die Sensoren können je nach den Platzverhältnissen im Uboot eingebaut werden, Dabei sollten die Sensoren aber entlang ihrer Achsen liegen (Abbildung 3.16)

Der x-Gyro ist in der Mitte des zweiten vorderen Schotts angebracht. Die anderen Sensoren sind auf einer Brücke befestigt, die über dem Board liegt (siehe Abbildung 3.17). Dabei befindet sich der z-Gyro und der Beschleunigungssensor oben, während der y-Gyro seitlich befestigt ist. Der äußere Drucksensor befindet sich im Heck. Der Innendrucksensor ist über der Brücke angebracht.

3.4.3 Rauschunterdrückung

Leider zeigten erste Tests, dass auf den Sensorsignalen ein für die Anwendung zu starkes Rauschen vorhanden ist. Weil die Sensorwerte für die Navigation kontinuierlich aufaddiert werden, entsteht mit der Zeit immer ein größerer Fehler. Es ist ein kontinuierliches Wegdriften von realen Wert. So lag die Ungenauigkeit zum Beispiel beim Z-Gyro bei 6 °/s.

Zuerst wurden zwei Bits vom Sensorwert abgeschnitten. Das hat aber den Nachteil, dass die Empfindlichkeit weniger wird und es war trotzdem noch ein geringes Rauschen vorhanden. Außerdem wird das

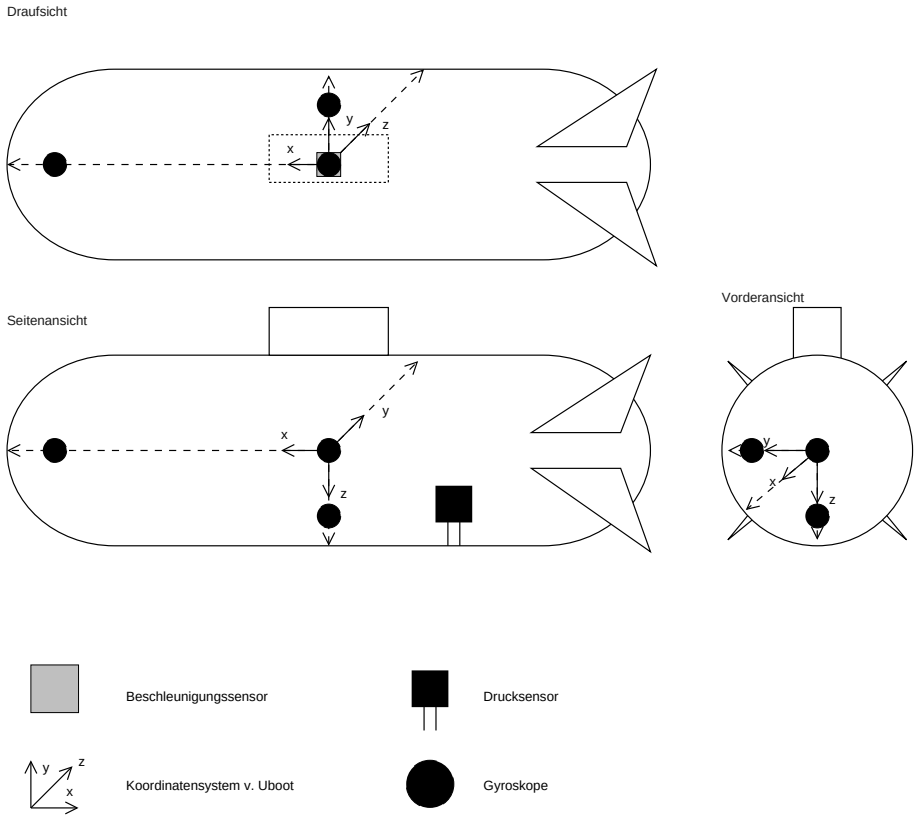


Abbildung 3.16: Positionen der digitalen Sensoren im Uboot

Signal vom Rauschen überlagert, also addiert, somit wirkt diese Methode nur einem bestimmten Bereich. Also muss eine andere Methode eingesetzt werden.

Eine Analyse des Rauschens am Oszilloskop zeigt, dass das Rauschen nicht genau spezifizierbar ist. Es ist ein Gemisch verschiedenster Störquellen. Dieses Rauschen war auch vorhanden, als direkt vom Akku gemessen wurde. Deshalb sollte die Sensorplatine in ein Metallgehäuse eingebaut werden, um eine möglichst gute Abschirmung zu erreichen.

Zusätzlich wurden im Laufe des Projekts zwei Filter implementiert: ein Mittelwert-Filer und ein Kalman-Filter.

Mittelwert-Filter

Da das Rauschen überwiegend im Hochfrequenzbereich liegt, wird ein einfacher Tiefpassfilter eingesetzt. Dabei wird eine bestimmte Anzahl n von Sensorwerten eingelesen, und daraus dann das arithmetische Mittel genommen.

$$x = \frac{1}{n} \sum_{n}^{1} x \tag{3.49}$$

Bei der Programmausführung wird in einem möglichst kleinen Intervall fortlaufend die gemessenen Sensorwerte x zu x_g addiert.

$$x_g = x_g + x \tag{3.50}$$

Nach der bestimmten Anzahl n gemessener Sensorwerte, wird die Summe x_g durch n geteilt.

$$x = \frac{x_g}{n} \tag{3.51}$$

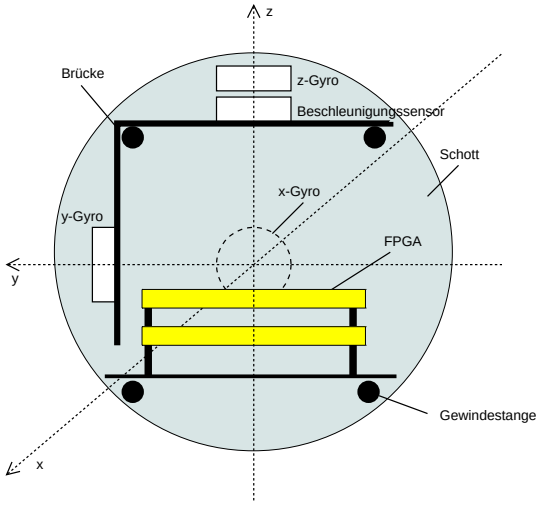


Abbildung 3.17: Aufbau der Brücke für die digitalen Sensoren (Ansicht nach Vorne)

Je mehr Werte eingelesen werden, desto glatter wird das Signal. Tests zeigen dass mindestens 50 Werte eingelesen werden müssen, um ein relativ stabiles Signal zu bekommen. Dieser Filter ist relativ intuitiv implementiert, ist jedoch kein richtiger Tiefpass. Da noch andere Prozesse verarbeitet werden müssen, können nur alle 5 ms die Werte eingelesen werden. Somit können auch nur alle 250 ms die Navigationswerte berechnet werden. Zudem bleibt der Offset abhängig von der Betriebsspannung, so dass hier eine sehr genaue Spannungsregelung erforderlich ist.

Dieser Filter wurde im ersten Meilenstein eingesetzt.

Kalman-Filter

Für den zweiten Meilenstein wird der Kalmanfilter eingesetzt [Kal60],[WB06]. Dieser wird im überwiegend in Inertialnavigationssystemen eingesetzt. Er erzeugt ein sehr stabiles Signal, wobei sogar der Offset in etwa dem Anfangswert entspricht.

Der Kalmanfilter ist ein fortlaufender itertiver Prozess, bei nur die vorige Instanz mitbetrachtet wird. Dadurch kann das Berechnungsintervall verkleinert werden. Ein weiterer Vorteil ist, dass der Filter leicht in Hardware implementiert werden kann.

Es gibt zwei Klassen von Variablen, feste Variablen (Tabelle 3.2), die meistens konstant sind, bei Bedarf können diese auch verändert werden, und Zustandsvariablen (Tabelle 3.3), die bei jeden Schritt neu berechnet werden. Diese Variablen sind Matrizen mit den Dimensionen l, m und n. l ist die Anzahl der optionalen Regelungeingänge, die hier aber nicht verwendet werden. m ist die Anzahl der betrachteten Sensorwerte. n ist die Anzahl der betrachteten Zustande, hier sind es die Navigationswerte.

A	$\in$	$\mathbb{R}^{n\times n}$	Relation zwischen vorigen und jetzigen Zustand x_k
B	$\in$	$\mathbb{R}^{n\times l}$	Relation zwischen Regeleingang u_k und x_k
Q	$\in$	$\mathbb{R}^{n\times n}$	Covarianz des Prozessrauschens
H	$\in$	$\mathbb{R}^{n\times l}$	Relation zwischen Messung z_k und Zustand x_k
I	$\in$	$\mathbb{R}^{n\times n}$	Einheitsmatrix
R	$\in$	$\mathbb{R}^{m\times m}$	Covarianz des Messrauschens

Tabelle 3.2: Feste Variablen im Kalmanfilter

Beim Start werden $\hat{x}_{k-1}$ und P_{k-1} gesetzt. $\hat{x}_{k-1}$ ist der geschätzte Anfangswert, P_{k-1} der a posteriori

$\hat{x}_k^-$	$\in$	$\mathbb{R}^n$	geschätzter Zustand a priori
P_k^-	$\in$	$\mathbb{R}^{n\times n}$	a priori Fehler-Kovarianz
K_k	$\in$	$\mathbb{R}^{n\times m}$	Gain, minimiert P_k
$\hat{x}_k$	$\in$	$\mathbb{R}^n$	geschätzter Zustand a posteriori P_k
P_k	$\in$	$\mathbb{R}^{n\times n}$	a posteriori Fehler-Kovarianz
z_k	$\in$	$\mathbb{R}^m$	Messung
u_k	$\in$	$\mathbb{R}^l$	Regeleingang (optional)

Tabelle 3.3: Zustandsvariablen im Kalmanfilter

Fehler. Daraus werden die a priori Werte berechnet (Time Update).

$$1. \quad \hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{3.52}$$
$$2. \quad P_k^- = AP_{k-1}A^T + Q \tag{3.53}$$
$$\tag{3.54}$$

Danach erfolgt das Measurement Update. Hier werden auch die Sensorenwerte (Messung) mit einbezogen.

$$3. \quad K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{3.55}$$
$$4. \quad \hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{3.56}$$
$$5. \quad P_k = (I - K_k H)P_k^- \tag{3.57}$$

Berechnet werden die neuen posteriori Werte und der Gain K_k. Diese zwei Blöcke Time Update (Zustand schätzen) und Measurement Update (Zustand mit Messung vergleichen) werden in einer Schleife fortlaufend wiederholt (Abbildung 3.18).

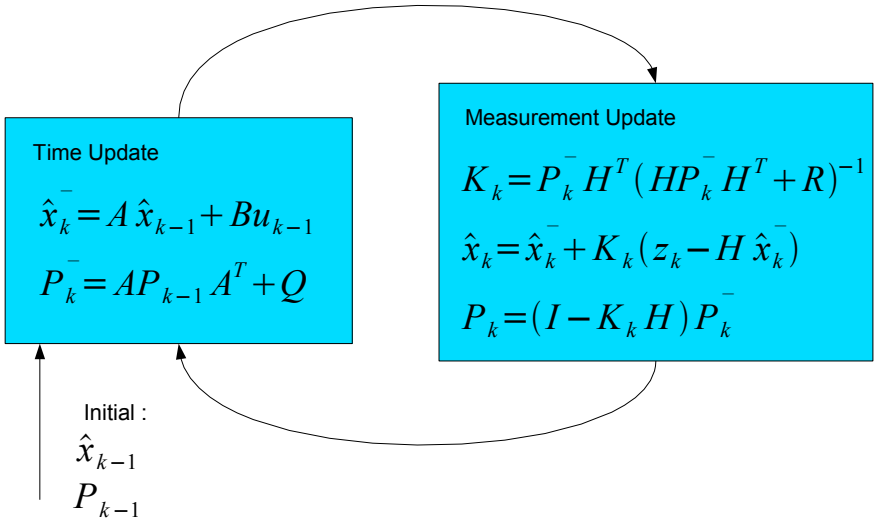


Abbildung 3.18: Blöcke des Kalmanfilters

Hier wird der Kalmanfilter nur für jeden Sensor einzeln angewendet. Das heißt, es wird anstatt mit Matrizen mit Werten aus dem realen Zahlenbereich gearbeitet. Der Kalmanfilter hat eine höheres Potenzial, wenn alle Sensoren miteinander in einen Kalmanfilter eingebunden sind. Es war jedoch aus zeitlichen Gründen nicht möglich, das zu realisieren. Es gibt noch den erweiterten Kalmanfilter, mit dem man auch die Positionsbestimmung und die Neigungen der Beschleunigungssensoren miteinbauen kann.

Das Diagramm in Abbildung 3.19 zeigt ein Beispiel wie die gefilterten Werte von Kalmanfilter und vom Mittelwert aussehen für einen Vergleich. Hier sieht man, dass beim Mittelwertfilter erst nach 50 eingelesenen Werten ein gefilterter Wert wieder neu vorhanden ist. Zudem wird der Ausschlag des Wertes deutlich gedämpft.

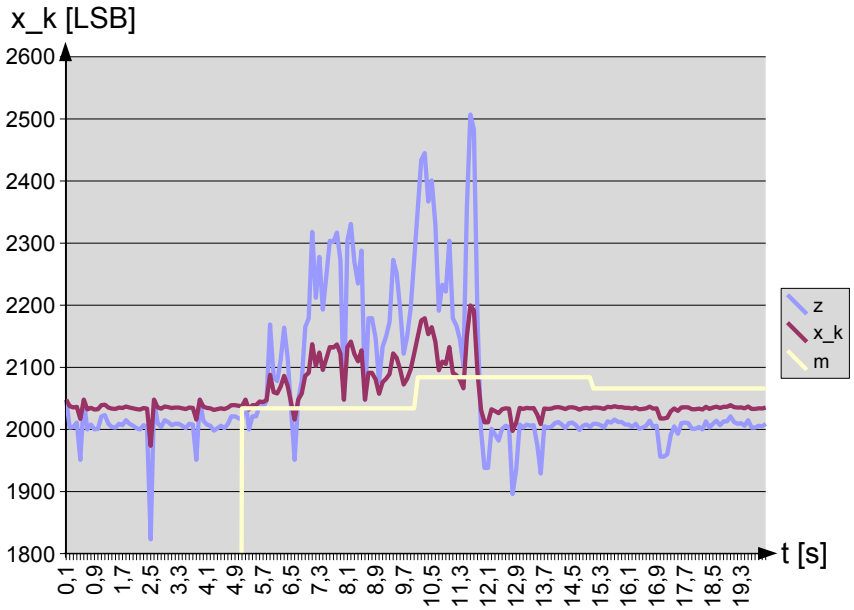


Abbildung 3.19: Vergleich der Ergebnisse vom Kalmanfilter (x_k) und dem Mittelwertfilter (m) mit Sensorwerten (z) über einen Zeitraum von 20 Sekunden

3.4.4 Kalibrierung

Von den Sensoren kommen Spannungswerte von 0 bis 5 V. Die werden von dem AD-Wandler in diskrete Werte zwischen 0 und 4096 LSB umgewandelt. LSB steht für *Least Significant Bit* und ist hier Maßeinheit für die digitalen Werte aus dem AD-Wandler. Mit diesen Werten kann man noch nicht viel anfangen. Sie müssen entsprechend den Sensorkennlinien in physikalische Werte umgewandelt werden.

Für den ersten Meilenstein wird von linearen Kennlinien ausgegangen. Dazu muss der Offset und die Empfindlichkeit bestimmt werden. Der Navigationswert soll wie folgt berechnet werden :

$$y = s(x - h) \tag{3.58}$$

y ist der Navigationswert, x der Wert vom AD-Wandler mit der Rauschunterdrückung (siehe 3.4.3), s die Empfindlichkeit und h der Offset oder Nullwert. Für die Kalibrierung wird x in der Ruheposition gemessen. Durch das Rauschen ergibt sich eine Reihe von Werten. Drei Werte sind dabei interessant

$$x_0 \quad Durchschnittlicher\ Wert \tag{3.59}$$

$$x_{0min} \quad Minimaler\ Wert \tag{3.60}$$

$$x_{0max} \quad Maximaler\ Wert \tag{3.61}$$

Sofern möglich, wird dieses auch bei anderen Sensorwerten gemacht. x_{0min} und x_{0max} sollten möglichst nahe am x_0 sein. Der Offset h entspricht dem Wert der Ruhelage:

$$h = x_0 \tag{3.62}$$

Falls weitere Werte bestimmt werden konnten, so kann die Empfindlichkeit auch bestimmt werden, ansonsten muss sie aus nachfolgenden Berechnungen geschätzt werden. Ist sie bekannt (durch Messversuche oder aus dem Datenblatt), so lässt sich die Empfindlichkeit s am AD-Wandler ermitteln. Der Spannungsbereich des AD-Wandlers liegt 0-5V, mit 12 Bit-Ausgabe, also $2^{12} = 4096\ LSB$. Die Empfindlichkeit für die Spannung liegt bei

$$s_V = \frac{5V}{4096\ LSB} \tag{3.63}$$
$$= 1,2207 * 10^{-3} V\ LSB^{-1} \tag{3.64}$$

Für die Sensorempfindlichkeit s am AD-Wandler gilt dann

$$s = \frac{s_V}{s_{analog}} \tag{3.65}$$

Bei den weiteren Meilensteinen wird auch die Position bestimmt. Da der Fehler quadratisch anwächst, müssen die Sensorkennlinien weiter präzisiert werden. Dazu gehören auch Nichtlinearitäten, die Hysterese und Temperaturabhängigkeiten [Law01]. Dort wird die Kalibrierung überwiegend auch über das Simulationsprogramm ermittelt (siehe Abschnitt 3.6.1).

Beschleunigungssensor

Nach dem Datenblatt [Ana04a] liegt die Ausgangsspannung der Ruheposition (0 g) bei 2.4 - 2.6 V, also im digitalen Bereich von 1966 bis 2130. Im Normalfall liegt die Spannung bei 2.5V, also ist der Offset für beide Richtungen

$$h_{accel} = 2048\ LSB \tag{3.66}$$

Die Empfindlichkeit liegt bei 940mV/g bis 1060mV/g, im Normalfall bei 1000mV/g. Da hier in $\frac{m}{s^2}$ und nicht in g gemessen wird, muss die Empfindlichkeit noch umgerechnet werden. Als 1g wird die Normgravitation angenommen.

$$1g = 9,80665\frac{m}{s^2} \tag{3.67}$$
$$s_{accel\_analog} = \frac{s_{analog\_1g}}{1g} \tag{3.68}$$
$$= \frac{1000mVg^{-1}}{9,80665g(ms^{-2})^{-1}} \tag{3.69}$$
$$= 101,97\frac{mV}{ms^{-2}} \tag{3.70}$$

Nach 3.65 gilt

$$s_{accel} = \frac{s_V}{a_{accel\_analog}} \tag{3.71}$$
$$= \frac{1,2207 * 10^{-3} V LSB^{-1}}{101,97\frac{mV}{ms^{-2}}} \tag{3.72}$$
$$= 11,971 * 10^{-3}\frac{m}{s^2 LSB} \tag{3.73}$$

Im Idealfall müsste sich die Beschleunigung für beide Richtungen so berechnen:

$$a = s_{accel}(x_{accel} - h_{accel}) \tag{3.74}$$
$$= 11,971 * 10^{-3}(x_{accel} - 2048)\frac{m}{s^2} \tag{3.75}$$

Um die Empfindlichkeit des Sensors zu messen, wurde der Sensor senkrecht aufgestellt, so dass die Schwerkraft auf ihn einwirkt. Damit kann eine Beschleunigung von $\pm 1g$ simuliert werden. Bei $1g$ war der

Durchschnittswert in der x-Richtung $x_1 = 2884,93 LSB$ und bei $-1g$ $x_2 = 1280,57 LSB$. Die Empfindlichkeit ist dann

$$s_{accelX} = \frac{2 * g}{x_2 - x_1} \tag{3.76}$$
$$= \frac{2 * 9,80665}{2884,93 - 1280,57} \frac{m}{s^2 LSB} \tag{3.77}$$
$$= 12,225 * 10^{-3} \frac{m}{s^2 LSB} \tag{3.78}$$

In der y-Richtung beträgt $x_1 = 2852,79$ und $x_2 = 1260,12$. Die Empfindlichkeit ist hier

$$s_{accelY} = \frac{2 * g}{x_2 - x_1} \tag{3.79}$$
$$= \frac{2 * 9,80665}{2852,79 - x_2 = 1260,12} \frac{m}{s^2 LSB} \tag{3.80}$$
$$= 12,3147 * 10^{-3} \frac{m}{s^2 LSB} \tag{3.81}$$

Der Offset kann dagegen nicht so leicht bestimmt werden. Durch das Einwirken der Schwerkraft und der Situation, dass das Uboot nie ganz gerade ist, wird es als gerade definiert, wenn die Inertialnavigation startet. Der Offset ist der erste eingelesene Wert.

Mit diesen Kalibrierungen werden dann die Beschleunigungen so berechnet

$$a_{accelX} = 12,225 * 10^{-3}(x_{accelX} - h_{accelX}) \frac{m}{s^2} \tag{3.82}$$
$$a_{accelY} = 12,3147 * 10^{-3}(x_{accelY} - h_{accelY}) \frac{m}{s^2} \tag{3.83}$$

h_{accelX} und h_{accelY} werden beim Start eingelesen.

Die digitalen Beschleunigungssensoren haben zur Zeit folgende Empfindlichkeiten eingestellt:

$$s_{accelX,digital} = 1.22147 * 10^{-2} \frac{m}{s^2 LSB} \tag{3.84}$$
$$s_{accelY,digital} = 1.23148 * 10^{-2} \frac{m}{s^2 LSB} \tag{3.85}$$

Gyroskop

Hier ist der Offset nach dem Datenblatt [Ana04c] bei 2,5V, als ist

$$h_{gyro} = 2048 LSB \tag{3.86}$$

Die Empfindlichkeit beträgt

$$s_{gyro\_analog} = 5mV/°/s \tag{3.87}$$

Nach 3.65 gilt

$$s = \frac{s_v}{s_{gyro\_analog}} \tag{3.88}$$
$$= \frac{1,2207 * 10^{-3} V}{5mV/°/sLSB} \tag{3.89}$$
$$= 0,24414°/sLSB \tag{3.90}$$

Die Empfindlichkeit lässt sich aufrund der nicht vorhandenen Versuchsgeräte nicht messen. Das heißt, anhand der weiteren Berechnungen (Lage, Kurs) muss die Empfindlichkeit bestimmt werden. Der Offset ist

einfach zu bestimmen. Unter verschiedenen Positionen wird die Ruheposition gemessen. Der Durchschnitt ist der Offsetwert:

$$h_{gyroX} = 2553LSB \tag{3.91}$$
$$h_{gyroY} = 2464LSB \tag{3.92}$$
$$h_{gyroZ} = 2340LSB \tag{3.93}$$

Die Steigung wird experimentell bestimmt, indem in einen bestimmten Winkel gedreht wird. Folgende Empfindlichkeiten haben die Sensoren :

$$s_{gyroX} = 0,45593°/sLSB \tag{3.94}$$
$$s_{gyroY} = 0,45593°/sLSB \tag{3.95}$$
$$s_{gyroZ} = 0,44105°/sLSB \tag{3.96}$$

Damit gilt

$$\omega_x = 0,45593(x_{gyroX} - 2553)°/s \tag{3.97}$$
$$\omega_y = 0,45593(x_{gyroY} - 2464)°/s \tag{3.98}$$
$$\omega_z = 0,44105(x_{gyroZ} - 2340)°/s \tag{3.99}$$

Die digitalen Gyroskope haben den gleichen Messbereich und zur Zeit folgende Kalibrierung eingestellt:

$$h_{gyroX,digital} = 0LSB \tag{3.100}$$
$$h_{gyroY,digital} = 2021,5LSB \tag{3.101}$$
$$h_{gyroZ,digital} = 2019,95LSB \tag{3.102}$$

$$s_{gyroX,digital} = 0°/sLSB \tag{3.103}$$
$$s_{gyroY,digital} = 0,24194°/sLSB \tag{3.104}$$
$$s_{gyroZ,digital} = 0,24194°/sLSB \tag{3.105}$$

Das X-Gyroskop ist momentan defekt, deswegen sind die Werte null und damit die abgeleiteten Werte auch.

Da die Lage die Uboots von dem Y-Gyroskop abhängt, und eine Analyse des genauen Sensorverhaltens bis zum ersten Meilenstein zeitlich nicht möglich ist, wird die Lage direkt aus dem Beschleunigungssensor gemessen. Das ermöglicht dann auch die Lageregelung. Die Geschwindigkeit kann aber dann nicht mehr bestimmt werden.

Auf dem Beschleunigungssensor wirkt auch die Gravitation ein. Nach dem Datenblatt [Ana04a] beträgt die Lage

$$\beta = arcsin(\frac{a_x}{g}) \tag{3.106}$$
$$a_x = Beschleunigung\ in\ X - Richtung \tag{3.107}$$

Da, wie bei dem Sinus, die trigonometrischen Funktion nicht in der Standardbibliothek vorhanden sind, muss der Arcussinus auch implementiert werden. Hierfür bietet sich wieder eine Taylorreihe an. Wenn man sich aber den Funktionsgraphen ansieht, so erkennt man, dass für kleinere Winkel die Funktion annähernd linear verläuft (Abbildung 3.20).

Mit einer Steigung von 58 beträgt die Abweichung bei 18° ungefähr $\frac{1}{100°}$ und bei 30° 1°. Dies sollte für die Anwendung des Uboots ausreichen. Damit gilt dann:

$$\beta = \frac{58}{g}a_x \tag{3.108}$$

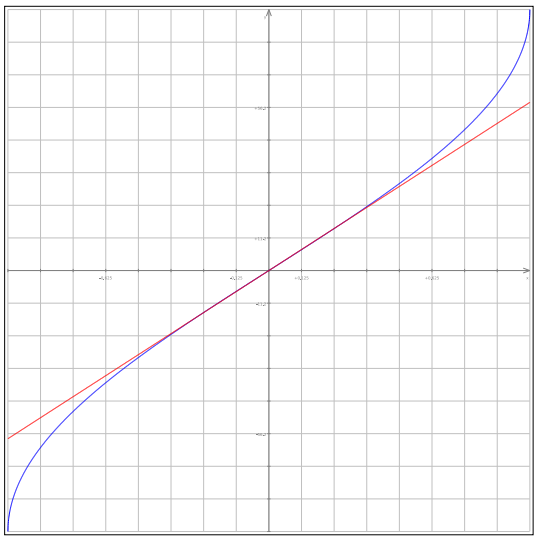


Abbildung 3.20: Arscussinus (blau) und die linearisierte Form (rot)

Hinzu kommen noch die Sensorparameter für den Beschleunigungssensor

$$\beta = \frac{58 * s_{accelX}}{g}(x_{accelX} - h_{accelX})^\circ \tag{3.109}$$

$$= \frac{58 * 11,971 * 10^{-3}}{9,80665}(x_{accelX} - h_{accelX})^\circ \tag{3.110}$$

$$= 70,801 * 10^{-3}(x_{accelX} - h_{accelX})^\circ \tag{3.111}$$

Kalibriert wurde die Neigung mit $72,201 * 10^{-3}$, h_{accelX} wird beim Start eingelesen.

Drucksensor

Mit den Drucksensor soll die Tiefe d des Uboots bestimmt werden. Da die Tiefe linear von dem Druck p abhängt, sollte die Tiefe wie der Druck so berechnet werden können:

$$d = s_{depth}(x_{depth} - h_{depth}) \tag{3.112}$$

$$p = s_{pressure}(x_{depth} - h_{pressure}) \tag{3.113}$$

Wenn die Inertialnavigation gestartet wird und das Uboot im aufgetauchten Zustand ist, so setzt sich der Startwert x_0 aus dem Luftdruck p_{Luft} und dem Sensoroffset h_{Sensor} zusammen. Der Drucksensor hat baubedingt eine Mindesttiefe d_0 (Abbildung 3.21).

Der Druck wächst unter Wasser mit $9,807kPa$ pro Meter. Dafür wird die Konstante $c_{WS} = 9,807\frac{kPa}{m}$ verwendet. Die Tiefe berechnet sich dann mit

$$d = \frac{1}{c_{WS}}(p - p_{Luft}) + d_o \tag{3.114}$$

p wird durch den Drucksensor gemessen (3.113). Eingesetzt ergibt es

$$d = \frac{1}{c_{WS}}(s_{pressure}(x_{depth} - h_{pressure}) - p_{Luft}) + d_o \tag{3.115}$$

Nun wird es nach 3.112 passend umgeformt:

$$d = \frac{1}{c_{WS}}(s_{pressure}x_{depth} - s_{pressure}h_{pressure} - p_{Luft}) + d_o \tag{3.116}$$

$$= \frac{s_{pressure}}{c_{WS}}(x_{depth} - (h_{pressure} + \frac{p_{Luft}}{s_{pressure}} - \frac{c_{WS}d_o}{s_{pressure}})) \tag{3.117}$$

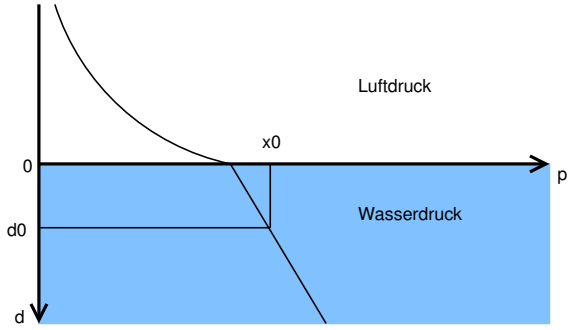


Abbildung 3.21: Verlauf des Drucks unter Wasser mit der Starttiefe d_0 und dem Startdruck x_0

Damit gibt es eine Empfindlichkeit für die Tiefe

$$s_{depth} = \frac{s_{pressure}}{c_{WS}} \tag{3.118}$$

$$\Leftrightarrow s_{pressure} = s_{depth} * c_{WS} \tag{3.119}$$

und einen Offset

$$h_{depth} = h_{pressure} + \frac{p_{Luft}}{s_{pressure}} - \frac{c_{WS} d_o}{s_{pressure}} \tag{3.120}$$

$$= h_{pressure} + \frac{p_{Luft}}{s_{depth} c_{WS}} - \frac{c_{WS} d_o}{s_{depth} c_{WS}} \tag{3.121}$$

$$= h_{pressure} + \frac{p_{Luft}}{s_{depth} c_{WS}} - \frac{d_o}{s_{depth}} \tag{3.122}$$

Der Startwert x_0 setzt sich,wie oben gesagt, aus dem Luftdruck, dem Sensoroffset und der Starttiefe d_0 zusammen. Um die Starttiefe mit einzubeziehen, muss sie vom Offset abgezogen werden.

$$h_{depth} = x_0 - \frac{d_o}{s_{depth}} \tag{3.123}$$

$$x_0 = h_{pressure} + \frac{p_{Luft}}{s_{depth} c_{WS}} \tag{3.124}$$

Die Empfindlichkeit des Sensors s_{analog} beträgt nach dem Datenblatt [Mot98] $20mV/kPa$. Die digitale Sensorempfindlichkeit $s_{pressure}$ beträgt dann

$$s_{pressure} = \frac{s_v}{s_{analog}} \tag{3.125}$$

$$= \frac{1,2207 * 10^{-3} V LSB^{-1}}{20mV/kPa} \tag{3.126}$$

$$= 61,035 * 10^{-3} \frac{kPa}{LSB} \tag{3.127}$$

$$= 61,035 * 10^{-2} \frac{mBar}{LSB} \tag{3.128}$$

$$\tag{3.129}$$

Da die Tiefe berechnet werden soll, muss c_{WS} eingefügt werden:

$$s_{depth} = \frac{s_{pressure}}{c_{WS}} \tag{3.130}$$

$$s_{depth} = \frac{61,035 * 10^{-3} kPa}{9,807 \frac{kPa}{m} LSB} \tag{3.131}$$

$$s_{depth} = 6,2235 * 10^{-3} \frac{m}{LSB} \tag{3.132}$$

Das Rauschen muss hier nicht berücksichtigt werden, da Tiefenwerte nicht weiter navigatorisch verarbeitet werden und kaum ein Rauschen vorhanden ist, da die Eingangsspannung stabilisiert ist. Kalibriert ist der Drucksensor mit

$$s_{depth} = 8,8907 * 10^{-3} \frac{m}{LSB} \tag{3.133}$$

Die Tiefe berechnet sich dann mit

$$d = 8,8907 * 10^{-3}(x_{depth} - h_{depth})m \tag{3.134}$$

$$h_{depth} = x_{0depth} - \frac{0.2}{s_{depth}} LSB \tag{3.135}$$

Der digitale Drucksensor im zweiten Meilenstein hat jedoch ein zu starkes Rauschen. Hier liegt die Ungenauigkeit der Tiefe von $\pm$ 20 cm. Im Vergleich dazu lag die Ungenauigkeit beim analogen Sensor bei $\pm$ 5 cm. Deswegen kommt hier ein Kalmanfilter im Einsatz mit folgenden Parametern:

$$\hat{x}_{k-1} = 2048 \tag{3.136}$$

$$P = 1 \tag{3.137}$$

$$A = 1 \tag{3.138}$$

$$B = 0 \tag{3.139}$$

$$H = 1 \tag{3.140}$$

$$Q = 0,1 \tag{3.141}$$

$$R = 10 \tag{3.142}$$

$$\tag{3.143}$$

Die Empfindlichkeit des digitalen Außendrucksensors ist

$$s_{depth,digital} = 0.0107 \frac{m}{LSB} \tag{3.144}$$

Um den Innendruck zu messen, wird ein zweiter Drucksensor eingebaut. Der Druck berechnet sich nach 3.113 mit dem Sensorwert $x_{pressure}$:

$$p = s_{pressure}(x_{pressure} - h_{pressure}) \tag{3.145}$$

$s_{pressure}$ ist der Wert aus Formel 3.128. $h_{pressure}$ sollte im Idealfall null betragen. Dennoch wird der Sensor mit folgenden Werten kalibriert:

$$s_{pressure} = 8,8907 * 10^{-4} \frac{Bar}{LSB} \tag{3.146}$$

$$h_{pressure} = 0LSB \tag{3.147}$$

Der Innendruck ist dann

$$p = 8,8907 * 10^{-4}(x_{pressure} - 0)Bar \tag{3.148}$$

Der digitalen Innendrucksensor hat die gleiche Empfindlichkeit, jedoch eine anderen Offset. Das liegt daran, dass die Kennlinie nach dem Datenblatt bei 0,5 V anfängt.

$$h_{pressure,digital} = 900LSB \tag{3.149}$$

Auswertung der Log-Dateien

Bei den einzelen Tauchgängen wurde, soweit es möglich war, die Navigationswerte aufgezeichnet. Dies erfolgte über das Syslog. Die Auswertung erfolgt mit den Simulationsprogramm *SimuNav* (Abschnitt 3.6). Mit den aufgezeichneten Sensorwerten kann dann mit unterschiedlichen Kalibrierwerten die Fahrt simuliert werden, ohne dass bei geänderter Kalibrierung wieder eine Testfahrt aufgenommen werden müsste.

Der Aufbau der Logdatei gestaltet sich folgendermaßen:

Die erste Spalte ist die Zeitangabe. Bei der Software im Uboot ist das die Uhrzeit. Bei einigen Tests, die außerhalb des Uboots gefahren wurden, zum Beispiel das Einrichten eines einzelen Sensors, ist es die Zeit in Millisekunden nach dem Start. Die zweite Spalte ist der Name des Tasks, hier *navigation*.

Nun folgen die Werte. Da der Schreibpuffer des Syslogs nicht ausreicht, um alle Werte gleichzeitig zu schreiben, werden die Werte auf mehrere Zeilen verteilt. Es gibt zwei Versionen.

Der erste Version ist die ältere. Die erste Zeile hat die Sensorwerte und die zweite Zeile die berechneten Navigationswerte. Die Werte sind Fixkommazahlen mit zwei Stellen hinter dem Komma als Genauigkeit. Hier ein Auschnitt aus der Logdatei:

```
1  00:55:30:063 navigation |AXB 2466|AYB 2430|GXB 2430|GYB 2465|GZB 2263|OPB 1585|IPB 1920|
2  00:55:30:063 navigation |AX 4670|AY −18732|GX 0|GY 0|GZ −793|D 41|IP 170|VX 0|VY 0|IX 0|IY 158|IZ 0|
```

In der zweiten Version kamen mehr Werte hinzu. In der ersten Zeile sind die ungefilterten Sensorwerte, in der zweiten Zeile die gefilterten Sensorwerte. Die dritte Zeile hat die direkt berechneten Navigationswerte. Die vierte Zeile die integrierten Navigationswerte.

```
1  14:23:20:964 navigation |AXR 2079|AYR 2047|GXR 2002|GZR 2017|GYR 2017|OPR 2159|IPR 2106|
2  14:23:20:964 navigation |AXB 2106|AYB 2090|GXB 1996|GZB 2017|GYB 2018|OPB 2159|IPB 2106|
3  14:23:20:965 navigation |AX 41|AY 98|GX −42|GZ −21|GY −21|D −6|IP 107|
4  14:23:20:965 navigation |VX 402|VY 681|VZ −14|IX −8451|IZ −15732|IY −5094|PX 22926|PY 14495|
```

Die Beschreibung der Abkürzungen ist in der Javaklasse *main.logevents.NavigationDataElementOrder*. Der erste Buchstabe gibt den Sensor, beziehungsweise Art des Wertes, an, *A* für Beschleunigung, *G* für Gyroskop, *V* für Geschwindigkeit. Der zweite Buchstabe beschreibt die Achse, in der der Wert wirkt. Der dritte optionale Buchstabe steht bei *R* für ungefilterten und bei *B* für einen gefilterten Sensorwert.

Bei der Auswertung der analogen Sensoren ist aufgefallen, dass die Offsets zu stark schwanken bei unterschiedlichen Fahrten. Die Werte wichen im Bereich von mehreren zehn LSB ab. Für zuverlässige Werte sind aber Abweichungen von höchsten ein LSB nötig. Dies lag vorallem daran, dass die Sensorwerte auch von der Betriebsspannung abhingen. Dies war ein Problem der analogen Gyroskope. Bei den Beschleunigungssensoren wird der Offset beim Start eingelesen.

Die digitalen Sensoren haben eine eingebaute Spannungsreglung. Dadurch weicht der Offset nur ein bis zwei LSB ab. Um die zunehmende Abweichung zu veringern ist der Offset auf zwei Stellen hinter dem Komma genau kalibriert. Damit kann bei dem Gyroskopen eine Abweichung von höchstens $3°/sec$ ereicht werden.

Die Geschwindigkeit bleibt nur wenige Minuten in einen realistischen Wert. Deshalb sind Grenzwerte definiert, wenn diese überschritten werden, wird eine Variable gesetzt, die besagt, ob die Navigationswerte zurückgesetzt werden sollten. Das kann dann über das Hostsystem erfolgen. Dort kann die Kalibrierung auch nachträglich angepasst werden.

Eine stabile Inertialnavigation während des gesamten Tauchgangs von 90 Minuten Dauer ist leider nicht möglich. Die Werte bleiben nur wenige Minuten, im besten Fall zehn Minuten, stabil. Deshalb sind weitere Navigationshilfen notwendig.

3.5 Implementierung

Nun muss die Navigation programmiert werden. Es soll die Berechnung für die Navigationswerte und Zugriff der Daten für andere Anwendungen implementiert werden. Im ersten Meilenstein soll das Programm in C++ programmiert werden. Dabei dienen die Klassen nicht so sehr der Objektorientierung, sondern mehr der Strukturierung des Programmcodes. In dem weiteren Meilenstein ist der Programmcode in C umgeschrieben.

3.5.1 1. Meilenstein: Programmierung in C++

Die Navigation wird in der Klasse *Navigation* bereitgestellt. Zu jeder Klasse gibt die C++-Datei *.cpp und die Headerdatei *.hpp. In der Headerdatei stehen die Signaturen, Klassenfunktionen und die Klassenvariablen. Abbildung 3.22 zeigt das Klassendiagramm, mit den für die Inertialnavigation relevanten Klassen.

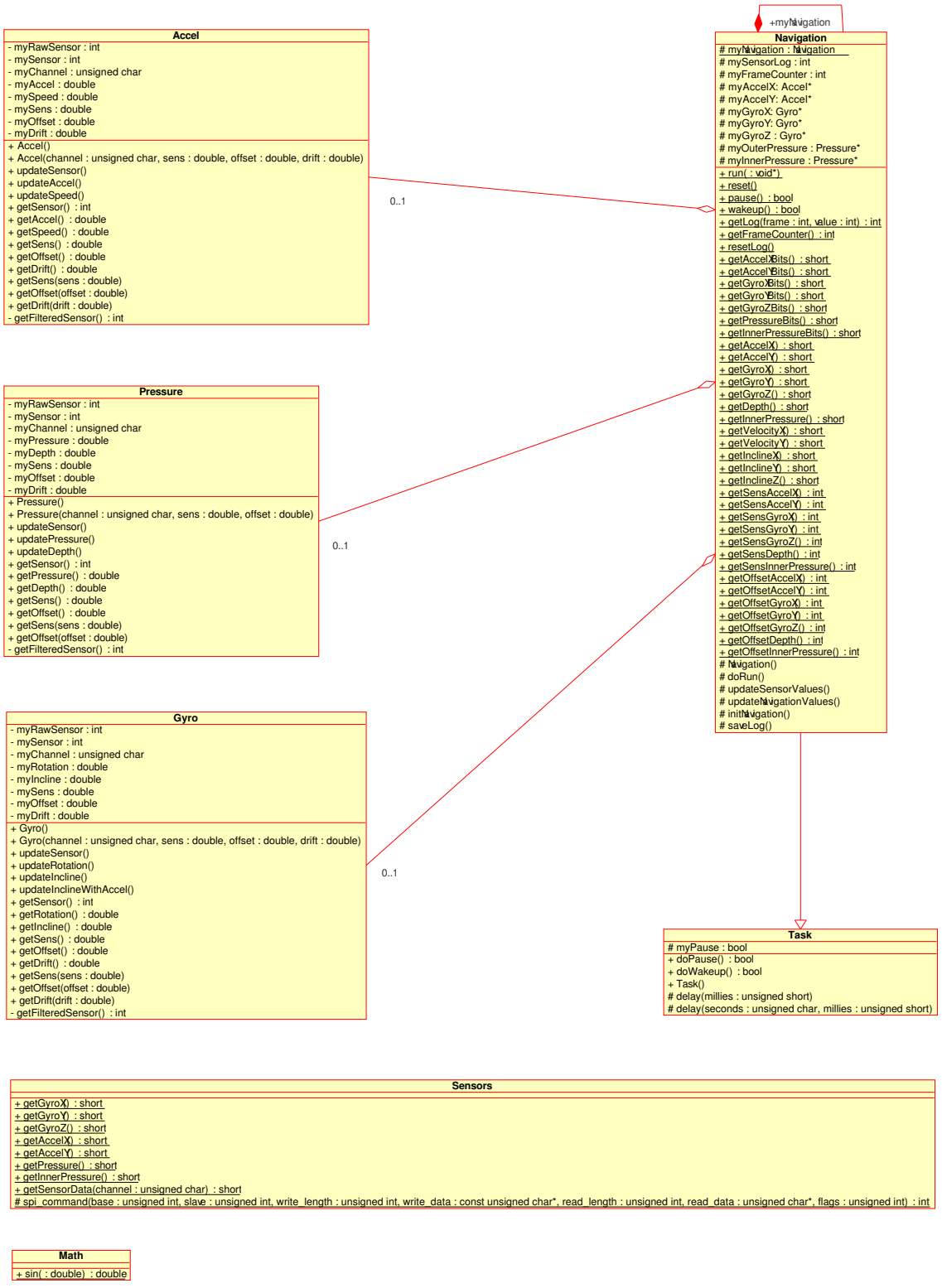


Abbildung 3.22: Klassendiagramm der Navigation

Klassen

Die wichtigste Klasse ist die *Navigation*. Diese enhält die Ausführungen der Inertialnavigation, die Sensoren als Objekte und die Funktionen um auf die Sensor- und Navigationswerte zuzugreifen. Sie ist abgeleitet

von der Klasse *Task* und damit ausführbares Programm. Folgende Funktionen sind relevant:

- *Navigation()*: Der Konstruktor, er ruft die Funktion *initNavigation()* auf.
- *void run(void*)*: wird vom Betriebsystem aufgerufen, ruft dann selber *doRun()* auf.
- *void reset()*: Setzt die Navigation zurück, ruft also dann *initNavigation()* auf.
- *bool pause()*: lässt die Navigation pausieren.
- *bool wakeup()*: weckt die Navigation wieder auf.
- *int getLog(int frame, int value)*: gibt den Sensorwert aus dem Logbuch in dem *frame* und dem Sensorindex *value* zurück.
- *int getFrameCounter()*: gibt die aktuelle Position im Logbuch zurück.
- *void resetLog()*: Setzt die Position im Logbuch wieder auf null zurück.
- *void doRun()*: Die Navigation wird hier mit einer For-Schleife abgearbeit. Die For-Schleife dient dazu *NAVIGATION_MEASUREMENTS_COUNT*-mal *updateSensorValues()* aufzurufen. Der zeitliche Abstand beträgt steht in *NAVIGATION_MEASUREMENTS_INTERVALL*. Nach der For-Schleife werden die Navigationswerte berechnet (*updateNavigationValues()*) und ins Logbuch eingetragen (*saveLog()*). Abbildung 3.23 zeigt den Ablauf der Navigation.
- *void updateSensorValues()*: Da das Rauschen gefiltert werden muss, rufen hier die Sensoren ihre Funktion *updateSensor()* auf.
- *void updateNavigationValues()*: Hier werden die Navigationswerte berechnet. Dazu werden bei den Sensoren die Funktionen *update<Navigationswert>()* aufgerufen.
- *void initNavigation()*: Hier wird die Initialnavigation initialisiert. Sie wird normalerweise beim Start den Uboots aufgerufen. Es wird der Zeiger auf das Logbuch auf null gesetzt, die Offsets der Beschleunigungssensoren und des äußeren Drucksensors bestimmt und die Sensoren erstellt.
- *void saveLog()*: Speichert die Navigationswerte und die Sensorwerte in das Logbuch.

Dazu gibt es noch die Funktionen, die die Sensor- und Navigationswerte zurückgeben. Diese werden später erläutertet. Folgende Klassenvariablen gibt es auch noch:

- *Navigation myNavigation*: Die Klasse als eigenes Objekt. Dadurch werden statische Funktionen ermöglicht.
- *int mySensorLog*: Das Logbuch. Es ist ein zweidimensionales Integer-Array. Das erste Feld ist der Frame, in der die Navigationswerte berechnet wurden. Das zweite Feld gibt die Nummer des Navigations- und Sensorwerts an. Die Werte haben das Format, welches über die get-Funktionen ausgeben wird. Zur Zeit hat das Array eine Größe von 2400 Frames mal 20 Werte.
- *int myFrameCounter*: Der Zeiger im Logbuch, der ausagt, welcher Frame bearbeitet wird. Er wird wieder auf null gesetzt, wenn der maximale Wert von 2400 ereicht wird.
- *Accel* myAccelX*: Der Beschleunigungssensor in der x-Richtung von der Klasse *Accel*.
- *Accel* myAccelY*: Der Beschleunigungssensor in der y-Richtung von der Klasse *Accel*.
- *Gyro* myGyroX*: Das Gyroskop in der x-Richtung von der Klasse *Gyro*.
- *Gyro* myGyroY*: Das Gyroskop in der y-Richtung von der Klasse *Gyro*.
- *Gyro* myGyroZ*: Das Gyroskop in der z-Richtung von der Klasse *Gyro*.
- *Pressure* myOuterPressure*: Der äußere Drucksensor von der Klasse *Pressure*.

- *Pressure* myInnerPressure*: Der innere Drucksensor von der Klasse *Pressure*.

Die Sensoren sind in den Klassen *Accel*, *Gyro* und *Pressure* vertreten. Der Aufbau ist zwar sehr ähnlich, aber es wurde für das Erste auf eine Vererbung abgesehen. Sie haben folgende Funktionen und Variablen:

- *<Konstruktor>(unsigned char channel, double sens, double offset, double drift)*: Erstellt den Sensor mit dem Kanal auf dem AD-Wandler, der Empfindlichkeit *sens*, dem Offset und der Adrift des 1. Integrals.
- *void updateSensor()*: Addiert zu *myRawSensor* den aktuellen Sensorwert.
- *void update<Navigationswert>()*: Berechnet den Navigationswert, den der Sensor repräsentiert. Bei *Accel* ist das die Beschleunigung, bei *Gyro* die Winkeländerung und bei *Pressure* der Druck.
- *void update<1. Integral>()*: Berechnet das erste Integral des Sensors. Bei *Accel* ist das die Geschwindigkeit, bei *Gyro* der Winkel und bei *Pressure* die Tiefe.
- *int getSensor()*: Gibt den gefilterten rohen Sensorwert zurück.
- *double get<Navigationswert>()*: Gibt den repräsentierten Navigationswert zurück.
- *double get<1. Integral>()*: Gibt das erste Integral vom Navigationswert zurück.
- *double getSens()*: Gibt die Empfindlichkeit zurück.
- *double getOffset()*: Gibt den Offset zurück.
- *double getDrift()*: Gibt die Abdrift des ersten Integrals zurück.
- *void setSens(double sens)*: Ändert die Empfindlichkeit.
- *void setOffset(double offset)*: Ändert den Offset.
- *void setDrift(double drift)*: Ändert die Abdrift des ersten Integrals.
- *int getFilteredSensor()*: filtert, zur Zeit durch das Mitteln, den Sensorwert und gibt ihn aus.
- *int myRawSensor*: direkter Wert von Sensor
- *int mySensor*: gefiterter Sensorwert
- *unsigned char myChannel*: der Kanel des Sensors am AD-Wandler
- *double my<Navigationswert>*: der repräsentierte Navigationswert
- *double my<1. Integral>*: das erste Integral des Navigationswertes
- *double mySens*: die Empfindlichkeit des Sensors
- *double myOffset*: der Offset
- *double myDrift*: die Abdrift des ersten Integrals

In der Klasse *Gyro* existiert noch die Funktion *void updateInclineWithAccel()*. Mit ihr wird, wenn als Kanal ein Beschleunigungssensor gewählt wurde, die Neigung von diesem Sensor direkt gemessen.

Um die Sensorwerte auslesen zu können, wird die Funktion *short Sensors::getSensorData(unsigned char channel)* von der Klasse *Sensors* verwendet. Es gibt zwar die Funktionen wie *getAccelX()*, diese werden aber nicht verwendet, aus Gründen der Parametrisierbarkeit. Zu guter Letzt gibt es noch die Klasse *Math*. Sie enthält mathematische Funktionen, die standardmäßig nicht vorhanden sind. Zur Zeit ist dort nur die Funktion *double sin(double)* vorhanden.

In der Abbildung 3.23 ist der Ablauf der Navigation aufgezeigt. Hier ist der Quellcode für die Funktion initNavigation():

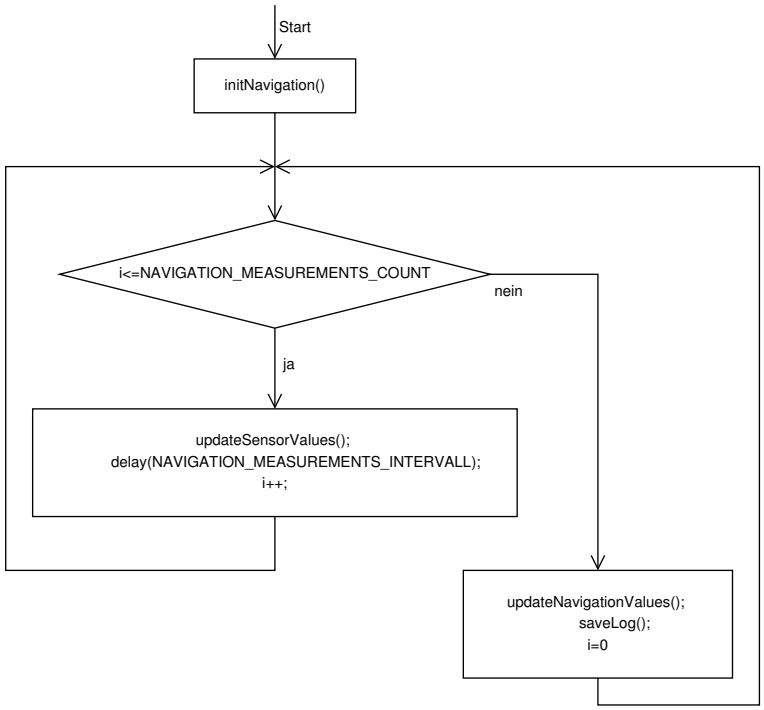


Abbildung 3.23: Ablauf der Navigation

```
1  void Navigation::initNavigation(){
2    //Zeiger fuer das Logbuch
3    myFrameCounter = 0;
4
5    //Da das Uboot nie ganz gerade ist, ist es als gerade definiert, wenn es gestartet wird
6    int offsetAccelX = Sensors::getAccelX();
7    int offsetAccelY = Sensors::getAccelY();
8    //Da der Luftdruck sich aendert, ist er im Offset mit drin
9    double offsetDepth = Sensors::getPressure() - DEPTH_INITIAL/SENS_DEPTH;
10
11   //Sensoren erstellen
12   myAccelX = new Accel(SENSOR_ACCEL_X, SENS_ACCEL_X , offsetAccelX , 0);
13   myAccelY = new Accel(SENSOR_ACCEL_Y, SENS_ACCEL_Y , offsetAccelY , 0);
14   //Messungen als Neigungssensor
15   myGyroX = new Gyro(SENSOR_ACCEL_Y, SENS_GYRO_X_WITH_ACCEL_Y , offsetAccelY , 0);
16   myGyroY = new Gyro(SENSOR_ACCEL_X, SENS_GYRO_Y_WITH_ACCEL_X , offsetAccelX , 0);
17   myGyroZ = new Gyro(SENSOR_GYRO_Z, SENS_GYRO_Z , OFFSET_GYRO_Z , DRIFT_COURSE);
18   myOuterPressure = new Pressure(SENSOR_PRESSURE, SENS_DEPTH , offsetDepth);
19   myInnerPressure = new Pressure(SENSOR_IN_1, SENS_INNER_PRESSURE , OFFSET_INNER_PRESSURE);
20 }
```

und die Funktion updateNavigationValues():

```
1  void Navigation::updateNavigationValues(){
2    //Vertikale Lage
3    myGyroX->updateInclineWithAccel();
4    //myGyro->updateRotation(); nicht anschalten, wenn mit Accel gemessen
5    //Horizontale Lage
6    myGyroY->updateInclineWithAccel();
7    //Kurs des Ubootes
8    myGyroZ->updateRotation();
9    myGyroZ->updateIncline();
10   //Werte in der X-Achse
11   myAccelX->updateAccel();
12   myAccelX->updateSpeed();
13   //Werte in der Y-Achse
14   myAccelY->updateAccel();
15   myAccelY->updateSpeed();
16   //Tiefe des Ubootes
17   myOuterPressure->updateDepth();
18   //Innendruck
19   myInnerPressure->updatePressure();
```

20 }

Globale Variablen

Für die Kalibrierung der Sensoren werden folgende konstanten Variablen verwendet:

- *int NAVIGATION_MEASUREMENTS_COUNT*: Anzahl der Messungen pro Berechnung
- *int NAVIGATION_MEASUREMENTS_INTERVALL*: Abstand zwischen zwei Messungen in mSekunden
- *double NAVIGATION_INTERVALL_IN_SECONDS*: Das Intervall zwischen zwei Berechnungen in Sekunden.
- *double GRAVITATION*: Die Normgravitation.
- *double SENS_ACCEL_X*: Empfindlichkeit des Beschleunigungssensors in X-Richtung
- *double SENS_ACCEL_Y*: Empfindlichkeit des Beschleunigungssensors in Y-Richtung
- *double OFFSET_GYRO_X*: Offset vom X-Gyro
- *double SENS_GYRO_X*: Empfindlichkeit des X-Gyroskops
- *double SENS_GYRO_X_WITH_ACCEL_Y*: Messung mit AccelY
- *double OFFSET_GYRO_Y*: Offset vom Y-Gyro
- *double SENS_GYRO_Y*: Empfindlichkeit des Y-Gyroskops
- *double SENS_GYRO_Y_WITH_ACCEL_X*: Messung mit AccelX
- *double OFFSET_GYRO_Z*: Offset vom Z-Gyro
- *double SENS_GYRO_Z*: Empfindlichkeit des Z-Gyroskops
- *double DRIFT_COURSE*: Abdrift von dem realen Wert
- *double DEPTH_INITIAL*: Tiefe des Sensor unter Wasser im aufgetauchten Zustand in Meter
- *double SENS_DEPTH*: Empfindlichkeit des Sensors für die Tiefe in Meter
- *double OFFSET_INNER_PRESSURE*: Offset des äußeren Drucksensors
- *double SENS_INNER_PRESSURE*: Empfindlichkeit des äußeren Drucksensors

Sie stehen in der *defines.h*.

Schnittstellen

Um der Regelung den Zugriff auf die Navigationswerte zu ermöglichen, gibt es folgende get-Funktionen in der Klasse *Navigation*. Das sind die rohen gefilterten Sensorwerte:

- *short getAccelXBits()*: vom Beschleunigungssensor in x-Richtung
- *short getAccelYBits()*: vom Beschleunigungssensor in y-Richtung
- *short getGyroXBits()*: vom Gyroskop in x-Richtung
- *short getGyroYBits()*: vom Gyroskop in y-Richtung
- *short getGyroZBits()*: vom Gyroskop in z-Richtung

- *short getPressureBits()*: vom äußeren Drucksensor
- *short getInnerPressureBits()*: vom inneren Drucksensor

Dann folgen die Navigationswerte:

- *short getAccelX()*: Beschleunigung in x-Richtung
- *short getAccelY()*: Beschleunigung in y-Richtung
- *short getGyroX()*: Winkeländerung in x-Richtung
- *short getGyroY()*: Winkeländerung in y-Richtung
- *short getGyroZ()*: Winkeländerung in z-Richtung
- *short getDepth()*: Tiefe des Uboots
- *short getInnerPressure()*: Druck im Inneren des Uboots
- *short getVelocityX()*: Geschwindigkeit in x-Richtung
- *short getVelocityY()*: Geschwindigkeit in x-Richtung
- *short getInclineX()*: Winkel in x-Richtung (Krängung)
- *short getInclineY()*: Winkel in y-Richtung (Lage)
- *short getInclineZ()*: Winkel in z-Richtung (Kurs)

Die Rückgabewerte sind Festkommazahlen mit zwei Stellen hinter dem Komma im Dezimalsystem. Die Kalibrierung kann man auch kontrollieren:

- *int getSensAccelX()*: die Empfindlichkeit des Beschleunigungssensor in x-Richtung
- *int getSensAccelY()*: die Empfindlichkeit des Beschleunigungssensor in y-Richtung
- *int getSensGyroX()*: die Empfindlichkeit des Gyroskops in x-Richtung
- *int getSensGyroY()*: die Empfindlichkeit des Gyroskops in y-Richtung
- *int getSensGyroZ()*: die Empfindlichkeit des Gyroskops in z-Richtung
- *int getSensDepth()*: die Empfindlichkeit des äußeren Drucksensors
- *int getSensInnerPressure()*: die Empfindlichkeit des inneren Drucksensors
- *int getOffsetAccelX()*: der Offset des Beschleunigungssensor in x-Richtung
- *int getOffsetAccelY()*: der Offset des Beschleunigungssensor in y-Richtung
- *int getOffsetGyroX()*: der Offset des Gyroskops in x-Richtung
- *int getOffsetGyroY()*: der Offset des Gyroskops in y-Richtung
- *int getOffsetGyroZ()*: der Offset des Gyroskops in z-Richtung
- *int getOffsetDepth()*: der Offset des äußeren Drucksensors
- *int getOffsetInnerPressure()*: der Offset des inneren Drucksensors

Bei der Empfindlichkeit sind die Rückgabewerte Festkommazahlen mit 6 Stellen hinter dem Komma im Dezimalsystem.

3.5.2 2. Meilenstein: Programmierung in C

In den zweiten Meilenstein soll vor allen die Positionsbestimmung implementiert werden. Dazu wurden die homogenen 4x4 Matrizen hinzugefügt. Außerdem wurde der Quellcode auf C umgestellt. Dies bedeutete eine andere Art der Zugriffe auf die Funktionen und den Daten.

Der Quellcode liegt in den Ordner *navigation* des Sourcen-Verzeichnisses. Die Struktur ist ähnlich wie beim C++-Code (Abbildung 3.22). Jede Datei mit dem C-Code und ihrer Headerdatei bildet eine „Klasse“. Die Dateien, die die Sensoren abbilden, sind im Ordner *sensors*. Zu jeder C-Datei gibt es eine Headerdatei, die die Signaturen der Funktionen hat.

Es gibt folgende Dateien mit ihren Funktion, Variablen und Strukturen:

Berechnung der Navigation

- *Navigation.c* Die wichtigste Datei. Sie enthält die Struktur *navigation* mit den Attributen
  - *Accel myAccelX* Beschleunigungssensor in X-Richtung
  - *Accel myAccelY* Beschleunigungssensor in Y-Richtung
  - *Gyro myGyroX* Gyroskop in X-Richtung
  - *Gyro myGyroY* Gyroskop in Y-Richtung
  - *Gyro myGyroZ* Gyroskop in Z-Richtung
  - *Pressure myOuterPressure* Außendrucksensor
  - *Pressure myInnerPressure* Innendrucksensor
  - *int shouldDataReset* wird auf 1 gesetzt, wenn die Grenzwerte überschritten wurden.

  Daneben gibt es noch folgende Funktionen:

  - *void Navigation_initNavigation()* Initialisiert die Navigation. Die Sensoren werden mit ihrer Kalibrierungen initialisiert.
  - *void Navigation_updateNavigationValues()* Diese Funktion wird periodisch aufgerufen. Für jeden Sensor wird die Funktion *updateValues* ausgeführt. Zwischen den Beschleunigungssensoren liegt eine Pause von 10 ms aufgrund aufgetretener Zeitprobleme. Die Werte würden sonst korrelieren. Am Ende wird die Funktion *Position_update()* ausgeführt. Der Ablauf ist in Abbildung 3.24 zu sehen. Zur Zeit wird *Navigation_saveLog()* aufgerufen, wenn vier mal *void Navigation_updateNavigationValues()* (j<=4) aufgerufen wurde. Dies ist abhängig von der Geschwindigkeit, die das Speichermedium schreiben kann. Hier sind das die Werte für eine Compact-Flash-Karte.
  - *double Navigation_setOffsetAccels(int (*channel)())* Bestimmt den Offset des Beschleunigungssensors, der durch den Funktionspointer *channel* vorgegeben ist. Dabei werden 50 Sensorwerte eingelesen, über den Kalmanfilter gefiltert und der Mittelwert gebildet.
  - *double Navigation_setOffsetOuterPressure()* Bestimmt den Offset des äußeren Drucksensors nach (3.122).
  - *void Navigation_saveLog()* Speichert die aktuellen Sensor- und Navigationswerte über das Syslog. Das Format ist in Abschnitt 3.4.4 beschrieben.
  - *void Navigation_saveCalibrationLog()* Speichert die aktuelle Kalibrierung über das Syslog.
  - *void Navigation_printData()* Gibt die Navigationsdaten über die Konsole aus. Dies dient unter anderen für die Fehlersuche, wird aber im normalen Betrieb nicht verwendet.
  - *void Navigation_setData(int speedX, int speedY, int inclineX, int inclineY, int inclineZ, int posX, int posY)* Hier kann man die Geschwindigkeiten, die Winkel und die Position setzen. Kann zum Beispiel über das Hostsystem beim Start aufgerufen werden.
  - *void Navigation_resetData()* Hier werden die Geschwindigkeiten, die Winkel und die Position auf null zurückgesetzt. Kann aufgerufen werden, wenn die Grenzwerte überschritten wurden (wenn shouldDataReset==1 ist).

Es gibt folgende Variablen:

- *char * logDesc[]* Abkürzungen der Sensor- und Navigationswerte (Erläuterung in Abschnitt 3.4.4).
- *typedef enum NavDesc* Reihenfolge der Sensor- und Navigationswerte. Wird bei Speichern der Werte in das Syslog gebraucht.

- *Position.c* Hier wird die Position bestimmt.
  - *double Position_local[4][4]* Lokale Positionsänderung in einem Berechnungsintervall als 4x4-Matrix.
  - *double Position_global[4][4]* Globale Position.
  - *void Position_init(double x, double y)* Setzt die Anfangsposition auf x und y und die lokale Positionsänderung auf die Einheitsmatrix.
  - *void Position_update()* Berechnet die Position nach dem in 3.3.2 vorgestellten Schema.

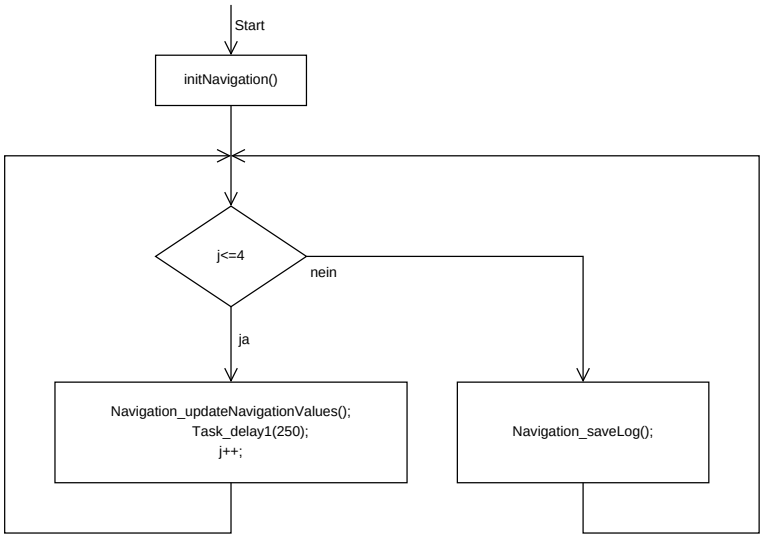


Abbildung 3.24: Ablauf der Navigation im zweiten Meilenstein

Sensoren

- *sensors/Accel.c* Repräsentiert einen Beschleunigungssensor. Die Struktur Accel hat folgende Elemente:
  - *myRawSensor* roher Sensorwert, der direkt ausgelesenen wurde.
  - *mySensor* gefilterter Wert über dem Kalmanfilter.
  - *int (*myChannel)()* Funktion, um den Sensorwert zu bekommen. Es werden hier Funktionen wie *getAccelX()* verwendet. Um Programmcode nicht mehrmals zu wiederholen, werden hier Funktionspointer verwendet.
  - *double myAccel* berechneter Sensorwert (m/s^2).
  - *double mySpeed* aktuelle Geschwindigkeit (in m/s).
  - *double myWay* zurückgelegte Strecke in einem Updateintervall.
  - *double mySens* die Empfindlichkeit des Sensors.

  - *double myOffset* der Offset des Sensors.
  - *int myAccelFix* Beschleunigung als Fixkommazahl.
  - *int mySpeedFix* Geschwindigkeit als Fixkommazahl.
  - *Kalman myKalman* der verwendeter Kalmanfilter.
  - *int myMinAccel* minimale Beschleunigung, die auftreten kann.
  - *int myMaxAccel* maximale Beschleunigung, die auftreten kann.
  - *int myMinSpeed* minimale Geschwindigkeit, die auftreten kann.
  - *int myMaxSpeed* maximale Geschwindigkeit, die auftreten kann.

  Die Funktion:

  - *void Accel_Accel(Accel* accel, int (*channel)(), double sens, double offset)* Initialisiert den Sensor *accel* mit der aufzurufenen Funktion *channel* für den Sensorwert, die Empfindlichkeit *sens* und den Offset *offset*. Geschwindigkeit und Beschleunigung sind am Anfang null.
  - *void Accel_updateValues(Accel* accel)* Berechnet die Navigationswerte. Die Sensorwerte werden ausgelesenen und mit dem Kalmanfilter gefiltert. Danach werden die Beschleunigung, Geschwindigkeit und diese geprüft, ob sie die Grenzwerte überschritten haben. Zum Schluss werden die Fixkommawerte gesetzt.

- *sensors/Gyro.c* Repräsentiert ein Gyroskop. Die Struktur *Gyro* hat folgende Elemente:

  - *int myRawSensor* roher Sensorwert.
  - *int mySensor* gefilterter Wert über dem Kalmanfilter.
  - *int (*myChannel)()* Funktionen, um den Sensorwert zu bekommen.
  - *double myRotation* berechneter Sensorwert (Winkeländerung in °/s).
  - *double myIncline* aktueller Winkel (in °).
  - *double myAngle* aktuelle Winkeländerung pro Updateintervall (in °).
  - *double mySens* die Empfindlichkeit des Sensors.
  - *double myOffset* der Offset des Sensors.
  - *int myRotationFix* berechneter Sensorwert (Winkeländerung in °/s) als Fixkommazahl.
  - *int myInclineFix* aktueller Winkel (in °) als Fixkommazahl.
  - *Kalman myKalman* verwendeter Kalmanfilter.
  - *int myMinGyro* minimale.Winkeländerung, die auftreten kann.
  - *int myMaxGyro* maximale Winkeländerung, die auftreten kann.
  - *int myMinIncline* minimaler Winkel, der auftreten kann.
  - *int myMaxIncline* maximaler Winkel, der auftreten kann.

  Die Funktionen:

  - *void Gyro_Gyro(Gyro* gyro, int (*channel)(), double sens, double offset)* Initialisiert den Sensor *gyro* mit der aufzurufenen Funktion *channel* für den Sensorwert, der Empfindlichkeit *sens* und den Offset *offset*. Der Winkel und die Winkeländerung sind am Anfang null.
  - *void Gyro_updateValues(Gyro* gyro)* Berechnet die Navigationswerte. Die Sensorwerte werden ausgelesen und mit den Kalmanfilter gefiltert. Dann werden die Winkeländerung und der Winkel berechnet. Diese Werte auf die Grenzwerte geprüft. Zum Schluss werden die Fixkommawerte gesetzt.

- *sensors/Pressure.c* Repräsentiert einen Drucksensor. Die Struktur *Pressure* hat folgende Elemente:

  - *int myRawSensor* roher Sensorwert.
  - *double mySensor* gefilterter Wert über dem Kalmanfilter.
  - *int (*myChannel)()* Funktion, um den Sensorwert zu bekommen.
  - *double myPressure* Berechneter Sensorwert (bar).
  - *double myDepth* aktuelle Tiefe (in m).
  - *double myDepthOld* Tiefe der alten Messung (in m).
  - *double myDepthSpeed* aktuelle Tauchgeschwindigkeit (in m).
  - *double mySens* die Empfindlichkeit des Sensors.
  - *double myOffset* der Offset des Sensors.
  - *int myPressureFix* Druck als Fixkommazahl.
  - *int myDepthFix* Tiefe als Fixkommazahl.
  - *int myDepthSpeedFix* Tauchgeschwindigkeit als Fixkommazahl.
  - *Kalman myKalman* verwendeter Kalmanfilter.

  Die Funktionen:

  - *void Pressure_Pressure(Pressure* pressure, int (*channel)(), double sens, double offset)* Initialisiert den Sensor *pressure* mit der aufzurufenen Funktion *channel* für den Sensorwert, der Empfindlichkeit *sens* und den Offset *offset*. Die Tiefe und die Tauchgeschwindigkeit sind am Anfang null.
  - *void Pressure_updatePressure(Pressure* pressure)* Berechnet die Navigationswerte. Die Sensorwerte werden ausgelesen und mit den Kalmanfilter gefiltert. Dann wird der Druck berechnet und als Fixkommawert gesetzt.
  - *void Pressure_updateDepth(Pressure* pressure)* Berechnet die Navigationswerte. Die Sensorwerte werden ausgelesen, mit den Kalmanfilter gefiltert. Dann wird die Tiefe und aus der Differenz des alten Tiefe die Tauchgeschwindigkeit berechnet. Zum Schluss werden die Fixkommawerte gesetzt.

- *kalman.c* Repräsentiert einen Kalmanfilter. Dieser wird von jeden Sensor verwendet. Die Struktur *Kalman* hat folgende Elemente:

  - *double A* Relation zwischen x_m_k_1 und x_m_k
  - *double A_T* Transponierte von A
  - *double B* Relation zwischen u und x_m_k (optional)
  - *double H* Relation zwischen Messwert z und Zustand x_m_k
  - *double H_T* Transponierte von H
  - *double I* Einheitsmatrix
  - *double Q* Kovarianzmatrix des Prozessrauschens
  - *double R* Kovarianzmatrix des Messrauschens
  - *double u* Reglereingang (optional)
  - *double z* Messwert
  - *double x_m* a priori Zustand
  - *double x* a posteriori Zustand
  - *double K* Gain oder blending Faktor
  - *double P_m* a priori Fehler
  - *double P* a posteriori Fehler

Die Funktionen:

- *void Kalman_init(Kalman* k)* Initialisiert den Kalmanfilter mit folgenden Standardparametern:

$$A = 1 \quad (3.150)$$
$$B = 0 \quad (3.151)$$
$$H = 1 \quad (3.152)$$
$$I = 1 \quad (3.153)$$
$$Q = 0 \quad (3.154)$$
$$R = 0 \quad (3.155)$$
$$u = 1 \quad (3.156)$$

- *void Kalman_update(Kalman* k)* Führt eine Iteration des Kalmanfilters *k* aus (Abbildung 3.18)

Hilfsfunktionen und Definitionen

- *defines.h* Hier sind die Definitionen der Programmvariablen. Dazu gehören Konstanten, Zeitintervalle, Kalibrierwerte und Grenzwerte. Hier sind es die derzeitigen Werte.
  - *NAVIGATION_MEASUREMENTS_COUNT 50* Anzahl der Messungen pro Berechnung,
  - *NAVIGATION_MEASUREMENTS_INTERVALL 5* Abstand zwischen zwei Messungen in Millisekunden.
  - *NAVIGATION_INTERVALL_IN_SECONDS 0.25* Zeit zwischen zwei Berechnungen in Sekunden
  - *GRAVITATION 9.80665* Gravation der Erde (hier Norm-Gravitation (Wikipedia))
  - *OFFSET_GAP 0.1* Steigerung/Minimierung des Offsets, wenn ein Sensor die Grenzwerte überschritten hat

  Der Beschleunigungssensor in die x-Richtung hat folgende Einstellungen:

  - *SENS_ACCEL_X 1.22147E-2* Empfindlichkeit des Sensors
  - *MIN_ACCEL_X -10.0* Minimale Beschleunigung
  - *MAX_ACCEL_X 10.0* Maximale Beschleunigung
  - *MIN_SPEED_X -10.0* Minimale Gewchwindigkeit
  - *MAX_SPEED_X 10.0* Maximale Gewchwindigkeit

  Der Beschleunigungssensor in die y-Richtung hat folgende Einstellungen:

  - *SENS_ACCEL_Y 1.23148E-2* Empfindlichkeit des Sensors
  - *MIN_ACCEL_Y -20.0* Minimale Beschleunigung
  - *MAX_ACCEL_Y 20.0* Maximale Beschleunigung
  - *MIN_SPEED_Y -20.0* Minimale Gewchwindigkeit
  - *MAX_SPEED_Y 20.0* Maximale Gewchwindigkeit

  Das Gyroskop in die x-Richtung hat folgende Einstellungen:

  - *OFFSET_GYRO_X 0* Offset vom X-Gyro, der Wert ist null, da der Sensor zur Zeit ausgefallen ist.
  - *SENS_GYRO_X 0* Empfindlichkeit des Sensors
  - *MIN_GYRO_X -90.0* Minimale Rotation
  - *MAX_GYRO_X 90.0* Maximale Rotation

  - *MIN_INCLINE_X -45.0* Minimaler Winkel
  - *MAX_INCLINE_X 45.0* Maximaler Winkel

  Das Gyroskop in die y-Richtung hat folgende Einstellungen:

  - *OFFSET_GYRO_Y 2021.5* Offset vom Y-Gyro
  - *SENS_GYRO_Y 0.24194* Empfindlichkeit des Sensors
  - *MIN_GYRO_Y -20.0* Minimale Rotation
  - *MAX_GYRO_Y 20.0* Maximale Rotation
  - *MIN_INCLINE_Y -45.0* Minimaler Winkel
  - *MAX_INCLINE_Y 45.0* Maximaler Winkel

  Das Gyroskop in die z-Richtung hat folgende Einstellungen:

  - *SENS_GYRO_Z 0.24194* Empfindlichkeit des Sensors
  - *OFFSET_GYRO_Z 2019.95* Offset vom Z-Gyro
  - *MIN_GYRO_Z -20.0* Minimale Rotation
  - *MAX_GYRO_Z 20.0* Maximale Rotation
  - *MIN_INCLINE_Z -1000.0* Minimaler Winkel
  - *MAX_INCLINE_Z 1000.0* Maximaler Winkel

  Der Außendrucksensor hat folgende Einstellungen:

  - *DEPTH_INITIAL 0.0* Tiefe des Sensor unter Wasser im aufgetauchten Zustand in Meter
  - *SENS_DEPTH 0.0107* Empfindlichkeit des Sensors für Tiefe in Meter
  - *DEPTH_KALMAN_R 10* Einstellung für Kalmanfilter

  Der Innendrucksensor hat folgende Einstellungen:

  - *OFFSET_INNER_PRESSURE 900* Offset des Innendrucksensors
  - *SENS_INNER_PRESSURE 8.8907E-4* Empfindlichkeit des Sensors

- *Math.c* Hier sind die Sinus- und Kosinusfunktionen für die Bestimmung der Position enthalten:
  - *double Math_sin(double)* Berechnet des Sinus.
  - *double Math_cos(double)* Berechnet den Kosinus.

Zugriff auf die Navigationwerte

Im ersten Meilenstein waren die folgenden Funktionen in der Klasse *Navigation*. Der übersichtlichkeitshalber sind sie jetzt in den folgenden Dateien ausgelagert.

- *NavigationCalibration.c* Enthält die Funktionen für die Kalibrierung der Sensoren. Die Kalibrierung erfolgt über Fixkommazahlen.

  Die Auflösungen der Fixkommazahlen:

  - *OFFSET_RESOLUTION 100.0* Auflösung des Offsets
  - *SENS_RESOLUTION 1000000.0* Auflösung der Empfindlichkeit

  Funktionen, die die Kalibrierung zurückgeben:

  - *int getSensAccelX()* Gibt die Empfindlichkeit des Beschleunigungssensors in die x-Richtung zurück.

- *int getSensAccelY()* Gibt die Empfindlichkeit des Beschleunigungssensors in die y-Richtung zurück.
- *int getSensGyroX()* Gibt die Empfindlichkeit des Gyroskop in die x-Richtung zurück.
- *int getSensGyroY()* Gibt die Empfindlichkeit des Gyroskop in die y-Richtung zurück.
- *int getSensGyroZ()* Gibt die Empfindlichkeit des Gyroskop in die z-Richtung zurück.
- *int getSensOuterPressure()* Gibt die Empfindlichkeit des Außendrucksensors zurück.
- *int getSensInnerPressure()* Gibt die Empfindlichkeit des Innendrucksensors zurück.
- *int getOffsetAccelX()* Gibt den Offset des Beschleunigungssensors in die x-Richtung zurück.
- *int getOffsetAccelY()* Gibt den Offset des Beschleunigungssensors in die y-Richtung zurück.
- *int getOffsetGyroX()* Gibt den Offset des Gyroskops in die x-Richtung zurück.
- *int getOffsetGyroY()* Gibt den Offset des Gyroskops in die y-Richtung zurück.
- *int getOffsetGyroZ()* Gibt den Offset des Gyroskops in die z-Richtung zurück.
- *int getOffsetOuterPressure()* Gibt den Offset des Außendrucksensors zurück.
- *int getOffsetInnerPressure()* Gibt den Offset des Innendrucksensors zurück.

Funktionen, die die Kalibrierung setzen:

- *void setSensAccelX(int x)* Setzt die Empfindlichkeit des Beschleunigungssensors in die x-Richtung.
- *void setSensAccelY(int x)* Setzt die Empfindlichkeit des Beschleunigungssensors in die y-Richtung.
- *void setSensGyroX(int x)* Setzt die Empfindlichkeit des Gyroskops in die x-Richtung.
- *void setSensGyroY(int x)* Setzt die Empfindlichkeit des Gyroskops in die y-Richtung.
- *void setSensGyroZ(int x)* Setzt die Empfindlichkeit des Gyroskops in die z-Richtung.
- *void setSensOuterPressure(int x)* Setzt die Empfindlichkeit des Außendrucksensors.
- *void setSensInnerPressure(int x)* Setzt die Empfindlichkeit des Innendrucksensors.
- *void setOffsetAccelX(int x)* Setzt den Offset des Beschleunigungssensors in die x-Richtung.
- *void setOffsetAccelY(int x)* Setzt den Offset des Beschleunigungssensors in die y-Richtung.
- *void setOffsetGyroX(int x)* Setzt den Offset des Gyroskops in die x-Richtung.
- *void setOffsetGyroY(int x)* Setzt den Offset des Gyroskops in die y-Richtung.
- *void setOffsetGyroZ(int x)* Setzt den Offset des Gyroskops in die z-Richtung.
- *void setOffsetOuterPressure(int x)* Setzt den Offset des Außendrucksensors.
- *void setOffsetInnerPressure(int x)* Setzt den Offset des Innendrucksensors.

- *NavigationData.c* Enthält die Funktionen für den Zugriff auf die Navigationsdaten. Diese werden von anderen Tasks und beim Speichern der Daten ins Syslog verwendet, Die Rückgabewerte sind Fixkommazahlen mit einer Auflösung *DATA_RESOLUTION* 100, also zwei Stellen hinter dem Komma.

  - *int getNavigationAccelXRaw()* Gibt den ungefilterten Sensorwert des Beschleunigungssensors in die x-Richtung zurück.
  - *int getNavigationAccelYRaw()* Gibt den ungefilterten Sensorwert des Beschleunigungssensors in die y -Richtung zurück.
  - *int getNavigationGyroXRaw()* Gibt den ungefilterten Sensorwert des Gyroskops in die x-Richtung zurück.
  - *int getNavigationGyroYRaw()* Gibt den ungefilterten Sensorwert des Gyroskops in die y-Richtung zurück.

  - *int getNavigationGyroZRaw()* Gibt den ungefilterten Sensorwert des Gyroskops in die z-Richtung zurück.
  - *int getNavigationOuterPressureRaw()* Gibt den ungefilterten Sensorwert des Außendrucksensors zurück.
  - *int getNavigationInnerPressureRaw()* Gibt den ungefilterten Sensorwert des Innendrucksensors zurück.
  - *int getNavigationAccelXBits()* Gibt den gefilterten Sensorwert des Beschleunigungssensors in die x-Richtung zurück.
  - *int getNavigationAccelYBits()* Gibt den gefilterten Sensorwert des Beschleunigungssensors in die y-Richtung zurück.
  - *int getNavigationGyroXBits()* Gibt den gefilterten Sensorwert des Gyroskops in die x-Richtung zurück.
  - *int getNavigationGyroYBits()* Gibt den gefilterten Sensorwert des Gyroskops in die y-Richtung zurück.
  - *int getNavigationGyroZBits()* Gibt den gefilterten Sensorwert des Gyroskops in die z-Richtung zurück.
  - *int getNavigationOuterPressureBits()* Gibt den gefilterten Sensorwert des Außendrucksensors zurück.
  - *int getNavigationInnerPressureBits()* Gibt den gefilterten Sensorwert des Innendrucksensors zurück.
  - *int getNavigationAccelX()* Gibt die Beschleunigung in die x-Richtung zurück.
  - *int getNavigationAccelY()* Gibt die Beschleunigung in die y-Richtung zurück.
  - *int getNavigationGyroX()* Gibt die Winkeländerung in die x-Richtung zurück.
  - *int getNavigationGyroY()* Gibt die Winkeländerung in die y-Richtung zurück.
  - *int getNavigationGyroZ()* Gibt die Winkeländerung in die z-Richtung zurück.
  - *int getNavigationDepth()* Gibt die Tiefe zurück.
  - *int getNavigationInnerPressure()* Gibt den Innendruck zurück.
  - *int getNavigationVelocityX()* Gibt die Geschwindigkeit in die x-Richtung zurück.
  - *int getNavigationVelocityY()* Gibt die Geschwindigkeit in die y-Richtung zurück.
  - *int getNavigationVelocityZ()* Gibt die Geschwindigkeit in die z-Richtung (Tauchgeschwindigkeit) zurück.
  - *int getNavigationInclineX()* Gibt die Neigung der x-Achse zurück.
  - *int getNavigationInclineY()* Gibt die Neigung der y-Achse zurück.
  - *int getNavigationInclineZ()* Gibt die Neigung der z-Achse zurück.
  - *int getNavigationPositionX()* Gibt die Position in der x-Achse zurück.
  - *int getNavigationPositionY()* Gibt die Position in der y-Achse zurück.
  - *int getNavigationPositionZ()* Gibt die Position in der z-Achse zurück.

- *navigationWrapper.c*, *pcNavigation.c* Damit können bei Bedarf die Navigationswerte über das Hostsystem für Regelung gesetzt werden. Diese Werte haben aber kein Einfluss auf die Werte, die hier in das Syslog eingetragen werden.

3.6 Simulationsprogramm

Um die Sensorkennlinien weiter bestimmen zu können, wurde ein Testprogramm in Java entwickelt. Der Name des Programms ist *SimuNav* (**Simu**late **Nav**igation). Mit diesen kann die Fahrt des Uboots, anhand gegebener Sensorwerte, simuliert werden. Mit den aufgezeichneten Daten und den in der Wirklichkeit gefahrenen Weg lassen sich daraus entsprechende Rückschlüsse ziehen. Diese Daten werden schon im ersten Meilenstein erfasst.

3.6.1 Handbuch

Das Hauptfenster besteht aus vier Komponenten: Das Programmmenü, der Toolbar, der Bereich für die Unterfenster und der Statusleiste (Abbildung 3.25). Über das Programmmenü sind alle Funktionen des Programms aufrufbar. Die wichtigsten Funktionen wie Öffnen, Speichern von Dateien, Starten, Stoppen der Simulation sind in der Toolbar direkt aufrufbar. In den Unterfenstern können verschiedene Daten angezeigt, verändert und Einstellungen gemacht werden. Diese Unterfenster können je nach Anwendung beliebig verschoben, geschlossen und deren Größe verändert werden. Die Statusleiste zeigt wichtige Informationen an. Das ist der Fortschritt beim Laden von Dateien, Zeit in der Simulation und ob gerade Daten berechnet werden.

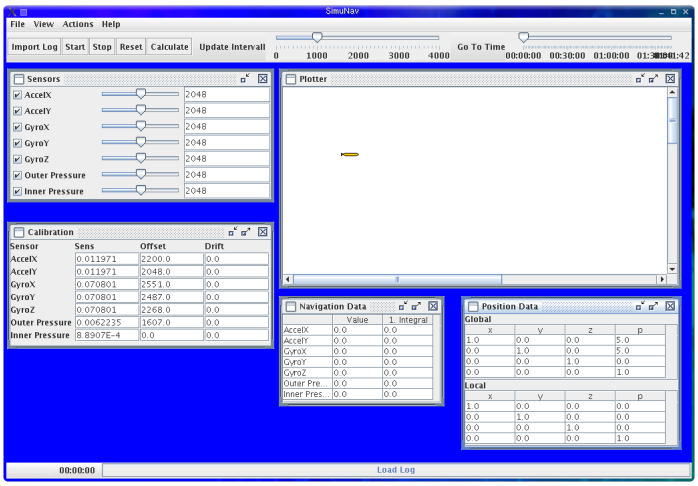


Abbildung 3.25: Hauptfenster nach dem Start von SimuNav

Im folgenden werden einige wichtige Anwendungen erklärt:

Simulation der Inertialnavigation

Eine Möglichkeit in dem Programm ist es, die Inertialnavigation zu simulieren. Über *Actions* → *Start Simulation* wird die Simulation gestartet. Nun können im Fenster *Sensors* (siehe Abbildung 3.26) die Sensorwerte eingestellt werden. In dem Fenster *Calibration* (Abbildung 3.27) kann die derzeitige Kalibrierung vorgenommen werden.

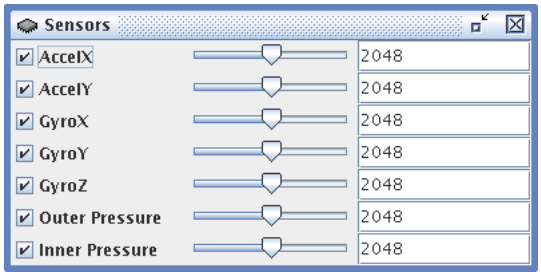


Abbildung 3.26: Einstellung der Sensorwerte

Stellt man zum Beispiel den Beschleunigungssensor in die X-Richtung über dem Offset, so wird das Uboot beschleunigt und es bewegt sich nach vorne. So kann dann simuliert werden, wie die sich die Position auf die Sensorwerte verhält. Die anderen Fenster, *Navigation Data* (Abbildung 3.28) und *Position Data* (Abbildung 3.29), für die Navigationswerte zeigen dabei auch ihre momentanen Werte an. Während der Simulation stellt man fest, dass sich die Beschleunigungssensoren wie ein Schub und die Gyroskope sich wie die Ruderstellungen in die jeweilige Richtung verhalten.

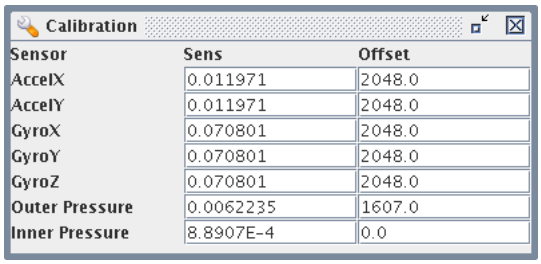


Abbildung 3.27: Kalibrierung der Sensoren

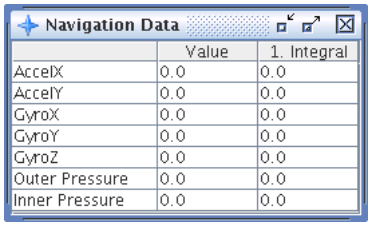


Abbildung 3.28: Momentane Navigationswerte, die Sensoren, ihr Navigationswert (z.B. Beschleunigung) und das erste Integral (z.B. Geschwindigkeit)

Gestoppt wird die Simulation mit dem Befehl *Actions* → *Stop Simulation*. Die momentanen Navigationswerte und die Position bleiben erstmal erhalten. Bei einem erneuten Start wird die Fahrt dann fortgesetzt. Mit *Actions* → *Reset Simulation* können die Werte zurückgesetzt werden.

Auswertung von Log-Dateien

Die andere Möglichkeit die das Programm bietet, ist es die Werte aus dem Logbuch auszuwerten. Unter *File* → *Import Log* kann eine Logdatei ausgewählt werden. Nach dem Importieren werden, anhand der aktuellen eingestellten Kalibrierung, die Navigationswerte berechnet. Diese sind dann in den einzelnen Fenstern zu sehen. Wird die Kalibrierung geändert, so werden die Werte sofort aktualisiert. Somit ist es möglich, wenn zum Beispiel der gefahrene Weg bekannt ist, die Sensoren nachträglich zu kalibrieren. Das gleiche gilt für die Parameter des Kalmanfilters. Die kann man auch einstellen (Abildung 3.30). Über ein Drop-Down-Feld ist der entsprechende Sensor auszuwählen. Damit der Kalmanfilter bei der Berechnung berücksichtigt wird, muss die Option *Options* → *Use Raw Sensor Values* gesetzt sein.

Diese ganzen Einstellungen können auch gespeichert werden. Mit *File* → *Save Navigation Data* werden die Daten aus dem importierten Logbuch, die Kalibrierung und die Parameter des Kalmanfilters in eine Navigationsdatei gespeichert. Diese kann auch wieder unter *File* → *Load Navigation Data* geöffnet werden.

Man kann auch die Fahrt auch simulieren. Dieses funktioniert analog zur Simulation, nur werden die Sensorwerte anhand des Logbuchs bestimmt. *Actions* → *Start* startet die Fahrt, A*ctions* → *Stop* stoppt sie und *Actions* → *Reset* setzt sie zum Anfang zurück.

Die Navigationswerte kann man sich auch als Tabellen anzeigen lassen (Abbildung 3.31). Das gleiche gilt für die Kalibrierwerte, die zu jenen Zeitpunkt eingestellt waren (Abbildung 3.32). In Abbildung 3.33 ist eingetragen, wann das Board resettet oder die Navigationswerte zurückgesetzt wurden. Daneben werden in Abbildung 3.34 nicht zugeordnete Einträge im Logbuch aufgeführt.

Da eine Tabellenform nicht immer gerade anschaulich ist, gibt es noch die Möglichkeit die Navigationswerte als Diagramm anzuzeigen (siehe Abbildung 3.35). Das Diagramm hat zwei Achsen, einmal die X-Achse mit dem Zeitbereich und die Y-Achse mit den Werten. Welche Werte angezeigt werden, können in der Leiste links neben dem Diagramm ausgewählt werden. Hier stehen die Abkürzungen der Navigationswerte. Oberhalb des Diagramms ist die Toolbar. Mit *Draw* wird das Diagramm neu gezeichnet. In den Textfeldern daneben kann der angezeigte Werte- und Zeitbereich eingetragen werden. Alternativ kann der

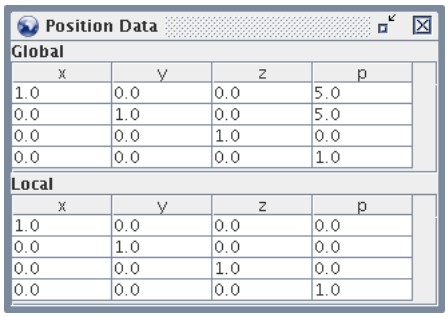


Abbildung 3.29: Momentane Position in Matrizenform, die lokale Positionsänderung (unten) und die globale Position (oben)

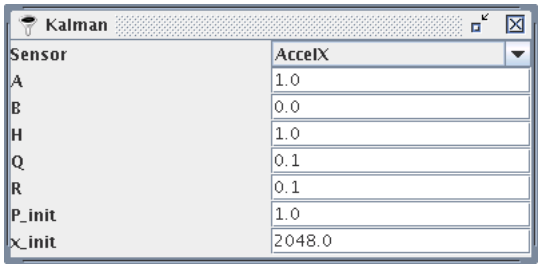


Abbildung 3.30: Parameter für den Kalmanfilter

Bereich mit der Maus im Diagramm auch ausgewählt werden. Mit *Zoom* wird das Diagramm mit den neuen Bereich gezeichnet. Mit *Zoom In* und *Zoom Out* kann der Bereich verkleinert oder vergrößert werden.

Dann kann man sich auch die Fahrt auf einer Karte anschauen (siehe Abbildung 3.36). In diesem Plotter ist die Draufsicht des Uboots zu sehen, wie die Fahrt nach der Inertialnavigation aussehen würden. Links oben im Plotter ist ein Maßstab und ein rotes Kreuz mit seiner Position eingezeichnet. Das schwarze Kreuz stellt die Nullposition dar. Der Startpunkt ist ein blauer Punkt mit der Beschriftung *Start*. Über dem Plotter befindet sich auch eine Toolbar. Mit *Zoom In* und *Zoom Out* wird die Ansicht vergrößert und verkleinert. *Zoom All* verkleinert die Ansicht so, dass die ganze Fahrt zu sehen ist. *Zoom Select* vergrößert auf dem mit der Maus ausgewählten Bereich und mit *Zoom Reset* wird die Ansicht auf dem Startzustand zurückgesetzt. Mit der Pan-Funktionen kann die Ansicht verschoben werden.

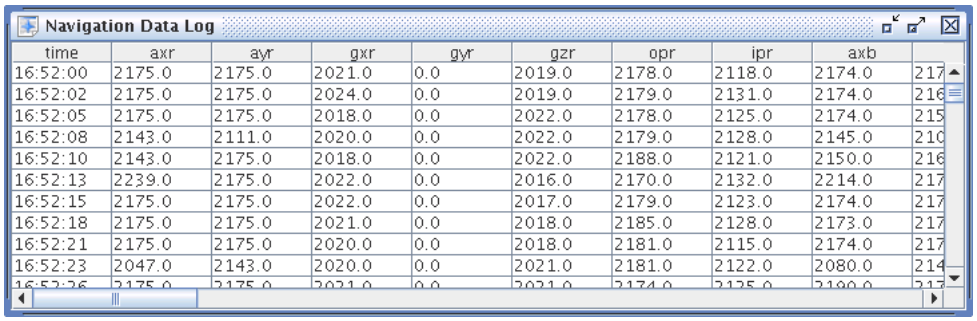
Funktionsbeschreibung

Im Hauptmenü sind alle Funktionen des Programms erreichbar. Im Dateimenü (*File*) sind die Funktionen für das Speichern und Öffnen von Datei untergebracht. Die Unterpunkte sind

- *Load Navigation Data* Lädt eine Navigationsdatei mit dem gelogten Werten und der Kalibrierung (in der Toolbar unter ).
- *Save Navigation Data* Speichert die aktuell geladene Navigationsdatei (in der Toolbar unter ).
- *Save As Navigation Data* Speichert die Navigationsdatei unter einer anderen Datei
- *Import Log* Importiert eine vorhandene Logdatei (in der Toolbar unter ).
- *Quit* Beendet das Programm.

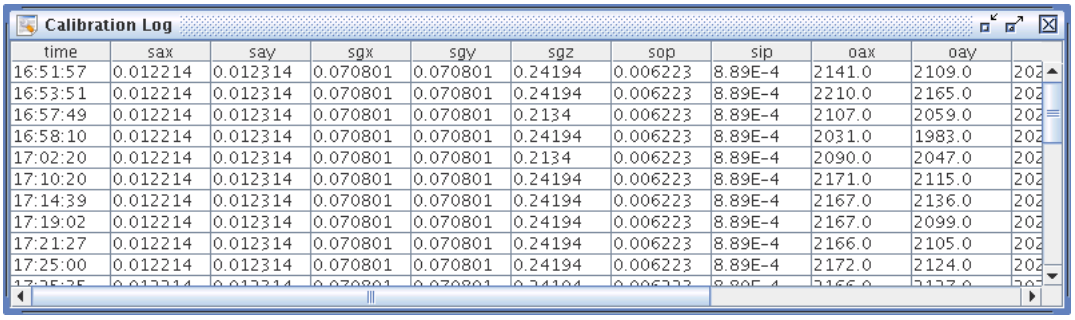
Im Menü *Windows* werden die einzelnen Unterfenster aufgerufen.

- *View Navigation Log* Zeigt das Fenster mit den aufgezeichneten Navigationswerten an (Abbildung 3.31).



time	axr	ayr	gxr	gyr	gzr	opr	ipr	axb	
16:52:00	2175.0	2175.0	2021.0	0.0	2019.0	2178.0	2118.0	2174.0	217
16:52:02	2175.0	2175.0	2024.0	0.0	2019.0	2179.0	2131.0	2174.0	216
16:52:05	2175.0	2175.0	2018.0	0.0	2022.0	2178.0	2125.0	2174.0	215
16:52:08	2143.0	2111.0	2020.0	0.0	2022.0	2179.0	2128.0	2145.0	210
16:52:10	2143.0	2175.0	2018.0	0.0	2022.0	2188.0	2121.0	2150.0	216
16:52:13	2239.0	2175.0	2022.0	0.0	2016.0	2170.0	2132.0	2214.0	217
16:52:15	2175.0	2175.0	2022.0	0.0	2017.0	2179.0	2123.0	2174.0	217
16:52:18	2175.0	2175.0	2021.0	0.0	2018.0	2185.0	2128.0	2173.0	217
16:52:21	2175.0	2175.0	2020.0	0.0	2018.0	2181.0	2115.0	2174.0	217
16:52:23	2047.0	2143.0	2020.0	0.0	2021.0	2181.0	2122.0	2080.0	214

Abbildung 3.31: Die aufgezeichneten Navigationswerte



time	sax	say	sgx	sgy	sgz	sop	sip	oax	oay	
16:51:57	0.012214	0.012314	0.070801	0.070801	0.24194	0.006223	8.89E-4	2141.0	2109.0	202
16:53:51	0.012214	0.012314	0.070801	0.070801	0.24194	0.006223	8.89E-4	2210.0	2165.0	202
16:57:49	0.012214	0.012314	0.070801	0.070801	0.2134	0.006223	8.89E-4	2107.0	2059.0	202
16:58:10	0.012214	0.012314	0.070801	0.070801	0.24194	0.006223	8.89E-4	2031.0	1983.0	202
17:02:20	0.012214	0.012314	0.070801	0.070801	0.2134	0.006223	8.89E-4	2090.0	2047.0	202
17:10:20	0.012214	0.012314	0.070801	0.070801	0.24194	0.006223	8.89E-4	2171.0	2115.0	202
17:14:39	0.012214	0.012314	0.070801	0.070801	0.24194	0.006223	8.89E-4	2167.0	2136.0	202
17:19:02	0.012214	0.012314	0.070801	0.070801	0.24194	0.006223	8.89E-4	2167.0	2099.0	202
17:21:27	0.012214	0.012314	0.070801	0.070801	0.24194	0.006223	8.89E-4	2166.0	2105.0	202
17:25:00	0.012214	0.012314	0.070801	0.070801	0.24194	0.006223	8.89E-4	2172.0	2124.0	202

Abbildung 3.32: Die aufgezeichnete Kalibrierung

- *View Calibration Log* Zeigt das Fenster mit der aufgezeichneten Kalibrierung an (Abbildung 3.32).
- *View Reset Log* Zeigt das Fenster mit den Resetereignissen an (Abbildung 3.33).
- *View Other Log* Zeigt das Fenster mit den restlichen Ereignissen aus dem Logbuch an (Abbildung 3.34).
- *View Diagrams* Zeigt das Fenster mit dem Diagrammen an (Abbildung 3.35).
- *View All Navigation Data* Zeigt das Fenster mit den momentanen Navigationswerten an (Abbildung 3.28).
- *View Position Data* Zeigt das Fenster mit den momentanen Positionsangaben an (Abbildung 3.29).
- *View Sensors* Zeigt das Fenster mit den Sensoren an (Abbildung 3.26).
- *View Calibration* Zeigt das Fenster mit der Kalibrierung an (Abbildung 3.27).
- *View Kalman* Zeigt das Fenster mit Kalmanfilter an (Abbildung 3.30).
- *View Plotter* Zeigt den Plotter an (Abbildung 3.36).

Das Menü *Plotter* steuert die Ansicht im Plotter. Die Befehle befinden sich auch in der Toolbar des Plotters.

- *Zoom In* Vergrößert die Ansicht ().
- *Zoom Out* Verkleinert die Ansicht ().
- *Zoom All* Verkleinert die Ansicht, so dass alle berechneten Positionen sichtbar sind ().

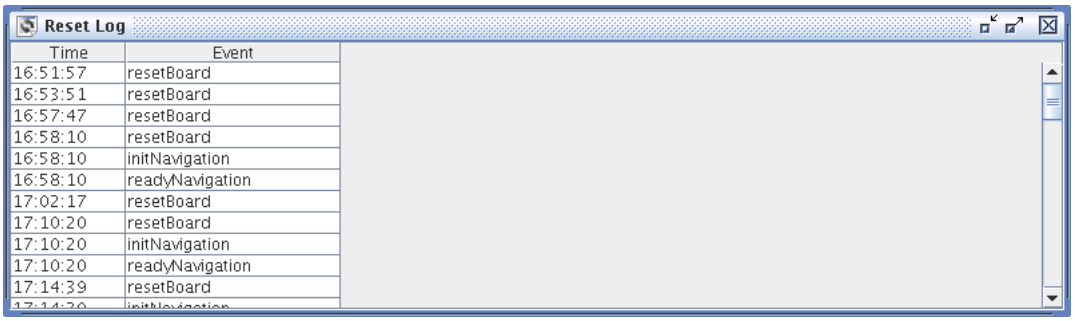


Abbildung 3.33: Zeiten, in der das Board resetet wurde

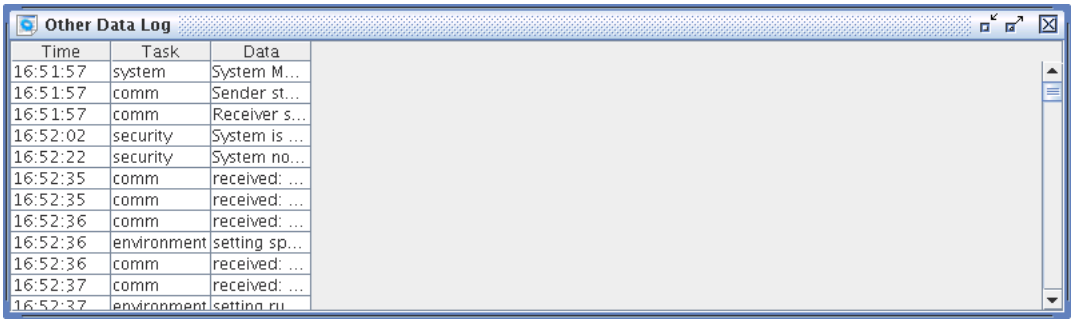


Abbildung 3.34: Andere aufgezeichnete Werte

- *Zoom Selection* Vergrößert auf den ausgewählten Bereich ( ).
- *Pan Right* Verschiebt die Ansicht nach rechts ( ).
- *Pan Left* Verschiebt die Ansicht nach links ( ).
- *Pan Up* Verschiebt die Ansicht nach oben ( ).
- *Pan Down* Verschiebt die Ansicht nach unten ( ).

Dieses Menü namens *Actions* kontrolliert die Simulation.

- *Start* Startet die Simulation anhands des Logbuchs (in der Toolbar unter).
- *Stop* Stopt die Simulation anhand des Logbuchs (in der Toolbar unter).
- *Reset* Setzt diese Simulation zurück (in der Toolbar unter).
- *Calculate Navigation* Berechnet die Navigationswerte im voraus (in der Toolbar unter).
- *Start Simulation* Startet die Simulation.
- *Stop Simulation* Stopt die Simulation.
- *Reset Simulation* Setzt die Simulation zurück.

Im Menü *Options* können Programmeinstellungen vorgenommen werden.

- *Set AccelX with InclineY* Bei der Berechnung der Navigationswerte wird die Neigung des Beschleunigungssensors in die X-Richtung mitberücksichtigt.

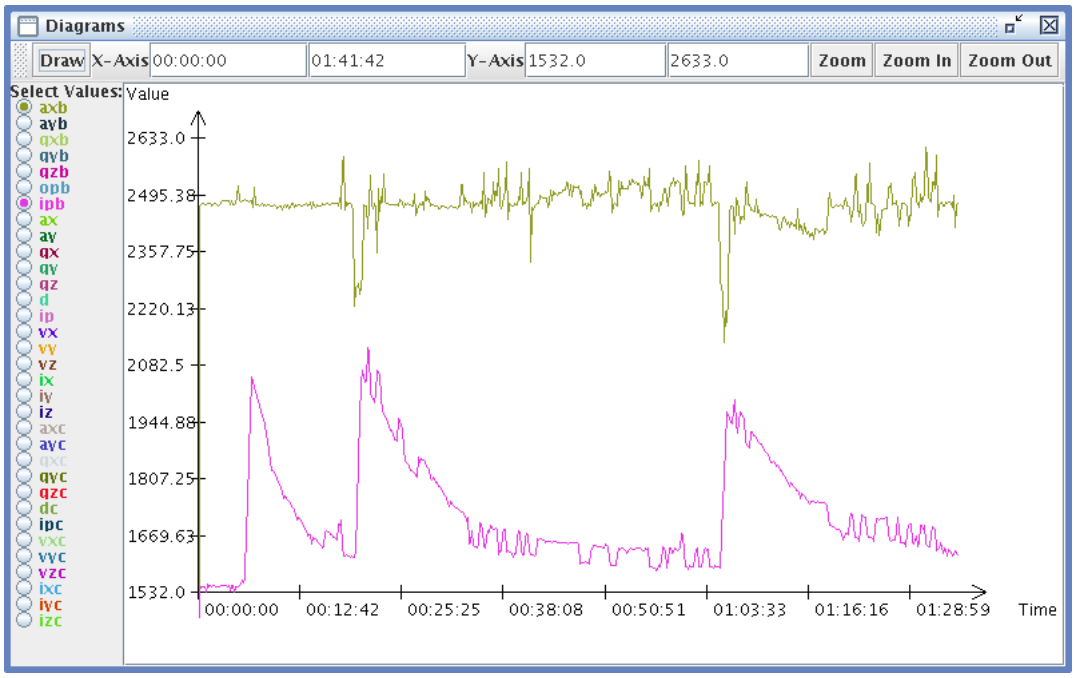


Abbildung 3.35: Anzeige der Navigationswerte in einem Diagramm

- *Set AccelY with InclineX* Berücksichtigt auch die Neigung des Beschleunigungssensors in die Y-Richtung.
- *Use Raw Sensor Values* Nimmt für die Berechnung der Navigationswerte die rohen Sensorwerte statt die gefilterten. Der Kalmanfilter wird damit mitsimuliert.
- *Set Angel in Circle Intervall* Begrenzt den Bereich der Winkel im Bereich von -180° bis 180°.
- *Show Board Reset* Zeigt im Diagramm als rote Linie an, wann das Board resetet wurde.
- *Show Reset Data* Zeigt im Diagramm als grüne Linie an, wann die Navigationswerte zurückgesetzt wurden.
- *Show Set Data* Zeigt im Diagramm als blaue Linie an, wann die Navigationswerte über das Hostsystem gesetzt wurden.

3.6.2 Klassenbeschreibung

Hier folgt nun eine Beschreibung der Klassen des Programms. Für eine ausführlichere Beschreibung mit den Attributen und den Methoden sei auf die Javadocs des Programms verwiesen. Das Klassendiagramm ist in Abbildung 3.37 zu sehen.

- *gui* Dieses Paket beinhaltet die alle Klassen, die die Elemente der Benutzeroberfläche bereitstellen.
  - *MainWindow* Das Hauptfenster, dort liegen fast alle Funktionen bereit. Es besteht aus einer Menuleiste und einer Toolbar oben und einer Statusleiste unten. Die einzelnen Komponenten werden als Unterfenster angezeigt.
  - *Util* Enthält häufig verwendete Funktionen. Dazu gehören die Datenpfade, Speicherdialoge, Umwandlung von Zeitformaten, Runden von Werten und das Laden der Programmsymbole.
- *gui.internalFrames* Hier sind die Klassen für die Unterfenster:

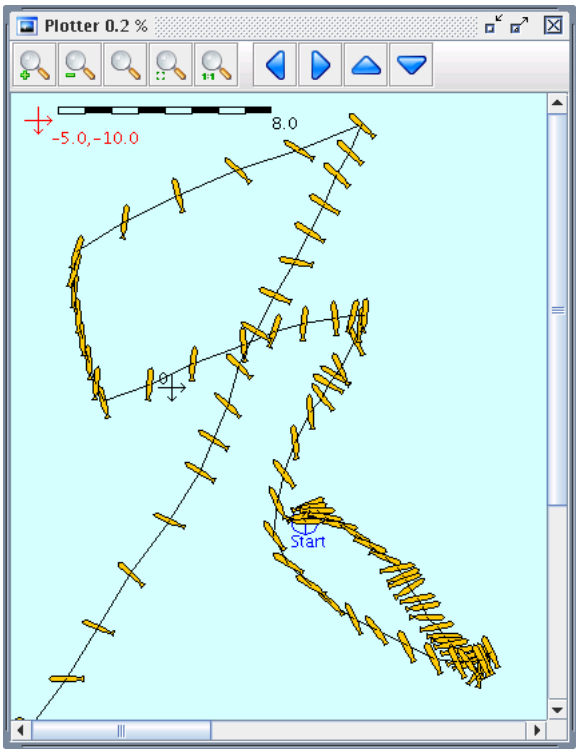


Abbildung 3.36: Fahrt des Uboots im Plotter

- *AllNavigationDataFrame* Zeigt alle Navigationsdaten an. Dabei wird der Sensor, die erste und die zweite Ableitung angezeigt. (Abbildung 3.28)
- *CalibrationDataLogFrame* Zeigt die Kalibrierwerte des Logbuchs als Tabelle an. Die Klasse ist abgeleitet von *DataLogFrame*. (Abbildung 3.32)
- *CalibrationFrame* Einstellungen für die Kalibrierung der Sensoren. Hier werden die aktuelle Kalibrierung angezeigt. Diese kann hier geändert werden. (Abbildung 3.27)
- *DataLogFrame* Zeigt die ausgewählten Daten des Logbuchs an. Dient als Vorlage für spezielle Daten, wie zum Beispiel die Navigationsdaten. Die Anzeige erfolgt in Tabellenform. Dies eine abstrake Klasse.
- *DiagramFrame* Zeigt ein Fenster, in dem die Sensorwerte graphisch als Diagramme ausgewertet werden können. Es enthält eine Toolbar mit wichtigen Funktionen, wie Zoom, Redraw, und eine Leiste, in der die Daten ausgewählt werden können. (Abbildung 3.35)
- *KalmanFrame* Zeigt die Parameter des Kalmanfilter der Sensoren an. Diese können hier verändert werden. (Abbildung 3.30)
- *NavigationDataLogFrame* Zeigt die Navigationsdaten des Logbuchs an. Diese Klasse ist von *DataLogFrame* abgeleitet. (Abbildung 3.31)
- *OtherDataLogFrame* Zeigt die restlichen Daten des Logbuchs an, die nicht geparst worden sind. Diese Klasse ist von *DataLogFrame* abgeleitet. (Abbildung 3.34)
- *PlotterFrame* Ein Fenster, in dem Positon als Plotter dargestellt wird. (Abbildung 3.36)
- *PositionDataFrame* Zeigt die Postion und die Postitionsänderung als Matrixdarstellung an. (Abbildung 3.29)
- *ResetLogFrame* Zeigt die ResetEvents des Logbuchs an. Diese Klasse ist von *DataLogFrame* abgeleitet. (Abbildung 3.33)

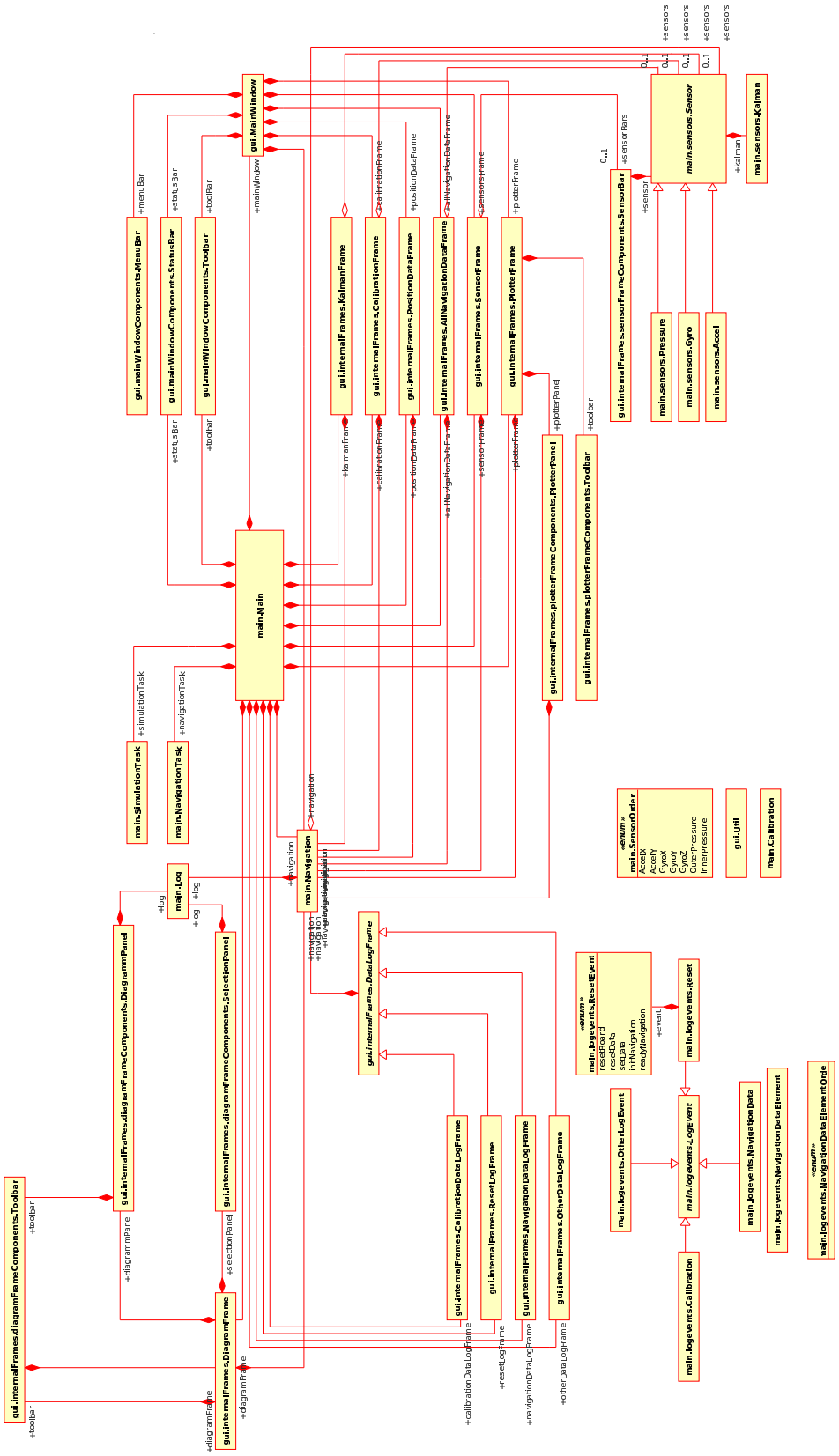


Abbildung 3.37: Klassendiagramm des Simulationsprogramms

  - *SensorFrame* Enthält die Einstellungen für die Sensorwerte. Bei der Simulation können hier die Sensorwerte eingestellt werden. Bei Fahrten aus dem Logbuch werden hier die Sensorwerte angezeigt. Wenn die Sensorwerte per Hand geändert werden, wird der Wert unmittelbar zur zugehörigen Klasse *Sensor* geleitet. (Abbildung 3.26)

- *gui.internalFrames.diagramFrameComponents* Die Komponenten von *DiagramFrame*.
  - *DiagrammPanel* Hier wird das Diagramm gezeichnet. Beschreibung siehe Abschnitt 3.6.3.
  - *SelectionPanel* Auswahl der Werte, die im Diagramm angezeigt werden.
  - *Toolbar* Toolbar mit Buttons für das Diagramm.

- *gui.internalFrames.plotterFrameComponents* Die Komponenten von *PlotterFrame*.
  - *PlotterPanel* Zeigt die Fahrt des Uboots in Richtung der Z-Achse (also von oben). Wie das Uboot gezeichnet wird, ist in Abschnitt 3.6.3 beschrieben.
  - *Toolbar* Die Toolbar für den Plotter. Enthält Funktion für den Plotter.

- *gui.internalFrames.sensorFrameComponents* Die Komponenten von *SensorFrame*.
  - *SensorBar* Einstellung eines einzelnen Sensorwertes.

- *gui.mainWindowComponents* Die Komponenten des Hauptfensters *MainWindow*.
  - *MenuBar* Die Menüleiste, enthält alle Funktionen des Programms.
  - *StatusBar* Die Statusleiste, zeigt wichtige Informationen an.
  - *Toolbar* Enthält die wichtigsten Funktionen des Programms. Die Symbole basieren von [KDE]

- *main* Die Klassen, die nicht zur Benutzeroberfläche gehören. Sie dienen zur Verarbeitung und Berechnung der Daten.
  - *Calibration* Enthält die Standardwerte der Sensorkalibrierung. Diese werden beim Programmstart eingelesen und gesetzt.
  - *Log* Das Logbuch, das dient dazu, aufgezeichnete Daten zu verwalten. Damit können Fahrten simuliert werden und die Sensoren kalibriert werden. Hier werden die Logdateien und Navigationsdaten geladen und geparst. Die Daten sind alle vom Typ *LogEvent*. Der Zugriff erfolgt über die Indizies der Reihenfolge in der Logdatei.
  - *Main* Zentrale Klasse, führt das Programm aus. Hier werden auch alle Funktionen des Programms aufgerufen.
  - *Navigation* Berechnet die Navigationswerte, führt die Simulation (der Sensorwerte) und die Navigation (Fahrt vom Logbuch) aus. Lädt auch die Navigationsdateien.
  - *NavigationTask* Führt die Navigation (Fahren nach dem Logbuch) periodisch aus.
  - *SimulationTask* Führt die Simulation periodisch aus.
  - *SensorOrder* Reihenfolge der Sensoren, wie sie bearbeitet werden.

- *main.logevents* Ereignisse aus dem Logbuch.
  - *Calibration* Einstellung der Kalibrierung im Logbuch.
  - *LogEvent* Ein Ereignis im Logbuch, kann eine Liste von Sensorwerten sein, Regeleingaben oder anderes sein. Dies ist eine abstrakte Klasse.
  - *NavigationData* Enthält die Navigationsdaten zu einen bestimmten Zeitpunkt.
  - *NavigationDataElement* Enthält einen Wert der Klasse NavigationData, zum Beispiel Beschleunigung in eine Richtung oder der Kurs
  - *OtherLogEvent* Andere Ereignisse im Logbuch, die noch nicht zugeordnet werden können.

  - *Reset* Zeiten, in dem das Board resettet wurde oder die Navigationswerte zurückgesetzt wurden.
  - *NavigationDataElementOrder* Reihenfolge der Navigationsdaten im Logbuch, wie sie gespeichert werden.
  - *ResetEvent* Mögliche Ereignisse, die Einfluss auf die Navigationswerte haben.

- *main.sensors* Die Sensoren. Diese berechnen ihre weiteren Werte, wie bei der Implementierung auf dem Board.
  - *Accel* Ein Beschleunigunssensor. Diese Klasse ist von *Sensor* abgeleitet.
  - *Gyro* Ein Gyroskop. Diese Klasse ist von *Sensor* abgeleitet.
  - *Kalman* Der Kalmanfilter für einzelne Sensoren.
  - *Pressure* Ein Drucksensor. Enthält die Kalibrierung und berechnet die darauf abgeleiteten Daten wie Tiefe und Druck. Diese Klasse ist von *Sensor* abgeleitet.
  - *Sensor* Ein Navigationssensor Enthält die Kalibrierung und die Berechnung davon abgeleiteter Werte.

3.6.3 Beschreibung von Algorithmen

Nicht immer deckt die Klassenbeschreibung und die Javadocs eine Erläuterung vollständig ab. Deshalb werden einige Eigenschaften hier erklärt.

Ablauf der Simulation

Die Simulation kann entweder über das Programmmenü oder über die Toolbar gestartet werden. In beiden Fällen wird *Main.startSimulation()* aufgerufen. Abbildung 3.38 zeigt das Sequenzdiagramm dazu.

Zuerst wird die Kalibrierungen über *calibrationFrame.update()* eingelesen. Dabei werden die aktuell gesetzten Werte der Textfelder in den Sensoren gesetzt (*setSens* und *setOffset*). Danach wird der Inhalt des Plotters geleert, das heißt eventuelle schon eingezeichnete Positionen werden entfernt. An Ende der Initialisierung wird der Timer *simulationTimer* gestartet (*schedule()*).

Dieser führt jetzt periodisch, in Abhängigkeit von *updateIntervall*, *simulationTask.run()* aus. Dieser ruft dann *Main.updateSimulation* auf. Dort werden die Navigationswerte berechnet. Mit *updateIntervall* wird *updateValue()*, *updateFirstIntegral()* und *updateSecondIntegral()* der Sensoren aufgerufen. Zum Schluss werden alle relevanten Anzeigen (Plotter, AllNavigationDataFrame, usw.) aktualisiert (*updateViews()*).

Während der *simulationTimer* läuft, können die Sensorwerte geändert werden. Wird ein Sensorwert geändert, so wird sofort der entsprechende Wert aus in der dazugehörigen Klasse *Sensor* geändert. Das geschieht mit einem *ChangeListener*. Damit brauchen nicht jedes mal die Werte von *SensorFrame* aufgerufen werden. Die Werte kann man dann auch von anderen Quellen her einstellen.

Gestoppt wird der *simulationTimer* mit einem einfachen *cancel()*. Die aktuellen Navigationswerte bleiben erhalten und die Simulation kann fortgesetzt werden.

Die Simulation der aufgezeichneten Logdaten, hier Navigation genannt, verläuft ähnlich. Hier gibt es ein *navigationTimer* und ein *navigationTask*. Die Funktion *Main.startNavigation()* verläuft wie die Funktion *Main.startSimulation()* ab.

Bei *updateNavigation()* werden jedoch die Werte aus dem Syslog gelesen und in der dazugehörigen *sensorBar* gesetzt. Dadurch wird der Sensorwert angezeigt und bei *updateSimulation()*, welches von *updateNavigation()* aufgerufen wird, gesetzt. Der Zeitraum zwischen zwei Aktualisierungen ist der Zeitunterschied zwischen zwei aufgezeichneten Navigationswerten. Abbildung 3.39 zeigt das dazugehörige Sequenzdiagramm.

Plotten der Position

In der Klasse *PlotterFrame* wird das Position des Uboots eingezeichnet. Es wird eine Draufsicht des Uboots gezeichnet, und das Uboot zeigt in Richtung seines Kurses. Die Figur des Uboots besteht aus einem Polygon mit neun Punkten:

$$F = (\vec{p}_0, \vec{p}_1, \vec{p}_2, \vec{p}_3, \vec{p}_4, \vec{p}_5, \vec{p}_6, \vec{p}_7, \vec{p}_8) \tag{3.157}$$

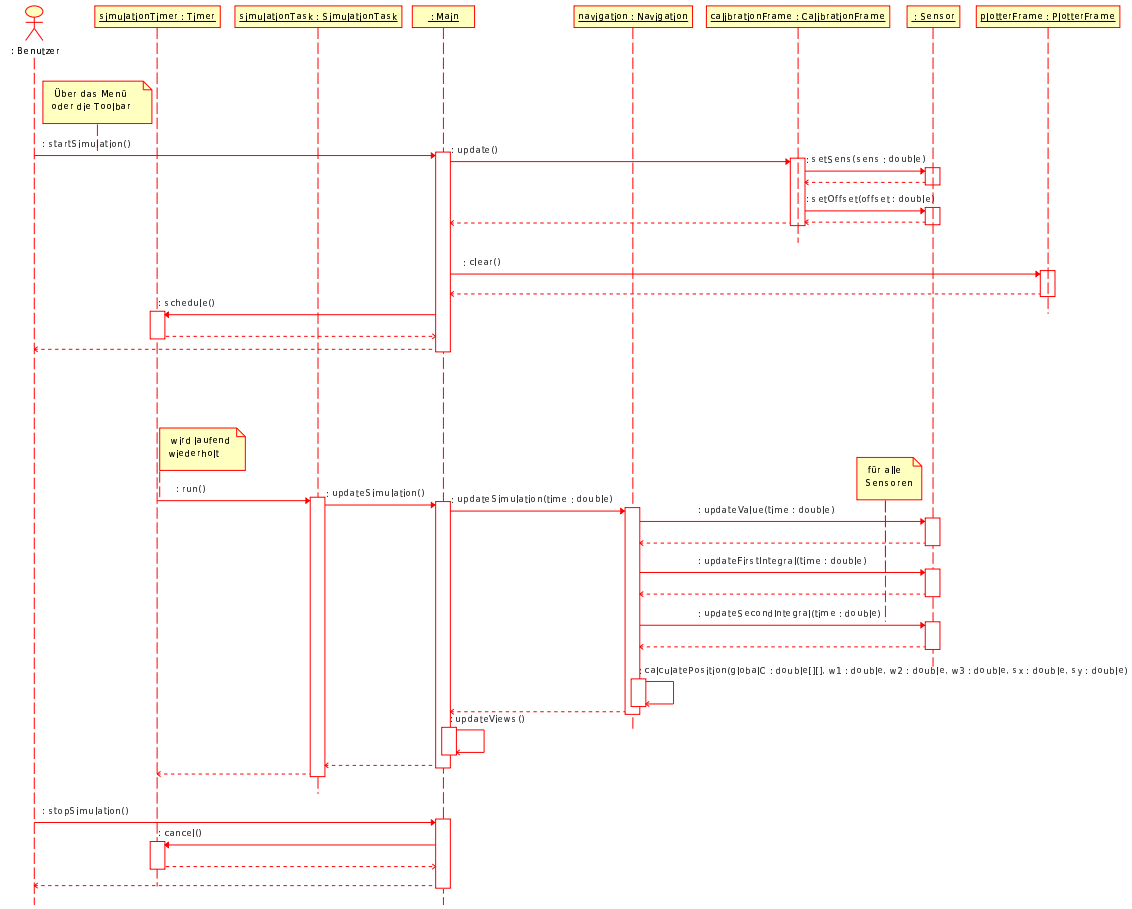


Abbildung 3.38: Sequenzdiagramm der Simulation

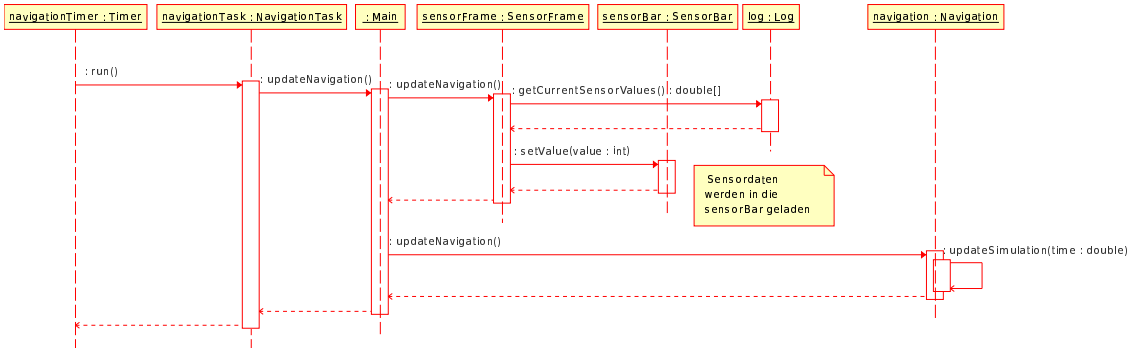


Abbildung 3.39: Sequenzdiagramm der Navigation

Die globale Position ist als 4x4-Matrix gegeben:

$$P = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} & P_x \\ R_{yx} & R_{yy} & R_{yz} & P_y \\ R_{zx} & R_{zy} & R_{zz} & P_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.158}$$

Für die Draufsicht sind nur die x,y-Koordinaten und der Kurs notwendig. Um die Position der Punkte

zu bestimmen, wird die Vektorrechnung angewandt. Die x,y-Koordinaten sind der Punkt $\vec{p}$:

$$\vec{p} = (P_x, P_y) \tag{3.159}$$

Der Kurs wird durch die zwei Normalvektoren $\vec{x}$ und $\vec{y}$ bestimmt:

$$\vec{x} = (R_{xx}, R_{xy}) \tag{3.160}$$
$$\vec{y} = (R_{yx}, R_{yy}) \tag{3.161}$$

Abbildung 3.40 zeigt die Vektoren im Polygon.

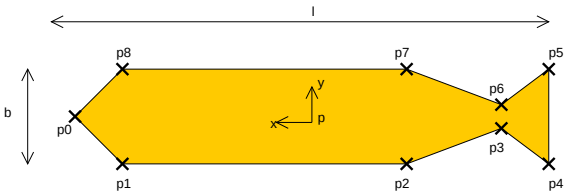


Abbildung 3.40: Polygon des Uboots mit den Punkten und der Position

Das Uboot hat die Länge l und die Breite b. Die Punkte werden nun wie folgt berechnet:

$$\vec{p_0} = \vec{p} + 0,5l\vec{x} \tag{3.162}$$
$$\vec{p}_{1,8} = \vec{p} + 0,4l\vec{x} \pm 0,5b\vec{y} \tag{3.163}$$
$$\vec{p}_{2,7} = \vec{p} - 0,2l\vec{x} \pm 0,5b\vec{y} \tag{3.164}$$
$$\vec{p}_{3,6} = \vec{p} - 0,4l\vec{x} \pm 0,16b\vec{y} \tag{3.165}$$
$$\vec{p}_{4,5} = \vec{p} - 0,5l\vec{x} \pm 0,5b\vec{y} \tag{3.166}$$

In der Funktion *drawSubmarine* werden die Punkte berechnet. Dort sind sie die Felder *xPoints* und *yPoints*. Diese können dann mit *drawPolygon()* gezeichnet werden.

Zeichnen des Diagramms

Die Klasse *DiagrammPanel* zeichnet das Diagramm zur Anzeige der aufgezeichneten Navigationsdaten. Die Größe der Fläche zum Zeichen ist *SIZE* mit den Ausmaßen 1000 mal 600 Pixel. Die Variable *GAP_HORIZONTAL* (gh) bestimmt den horizontalen Abstand der Achsen von Rand der Fläche, die Variable *GAP_VERTICAL* (gv) den vertikalen Abstand. In Abbildung 3.41 ist die Konstruktion des Diagramms zu sehen.

Die waagerechte X-Achse zeigt die Zeit an, die senkrechte Y-Achse die Werte der ausgewählten Navigationsdaten. Der anzuzeigene Werte- und Zeitbereich wird mit entsprechend *minX*, *minY*, *maxX* und *maxY* angegeben. Beim Zeichnen der Kurve wird jedes Pixel entlang der X-Achse (Länge ist xl) durchgegangen. Die Zeit, die zu dem Pixel x gehört, wird so berechnet:

$$time = \frac{maxX - minX}{xl} * x + minX \tag{3.167}$$

Mit *log.getNavigationValues(time, selectedValues)* werden die zu dem Zeitpunkt aufgezeichneten Werte abgerufen. Aus dem erhaltenen Navigationswert $value$ wird die Pixelposition auf der y-Achse berechnet:

$$y = \frac{yl}{maxY - minY} * (value - minY) \tag{3.168}$$

yl ist die Länge der Y-Achse. x und y sind die Position auf dem Zeichfeld und können deshalb direkt eingezeichnet werden. Dadurch, dass nur für jedes Pixel auf der x-Achse der Wert eingetragen wird und nicht für jeden aufgezeichneten Navigationswert, ist die Geschwindigkeit der Anzeige unabhängig von der Größe des Logbuchs.

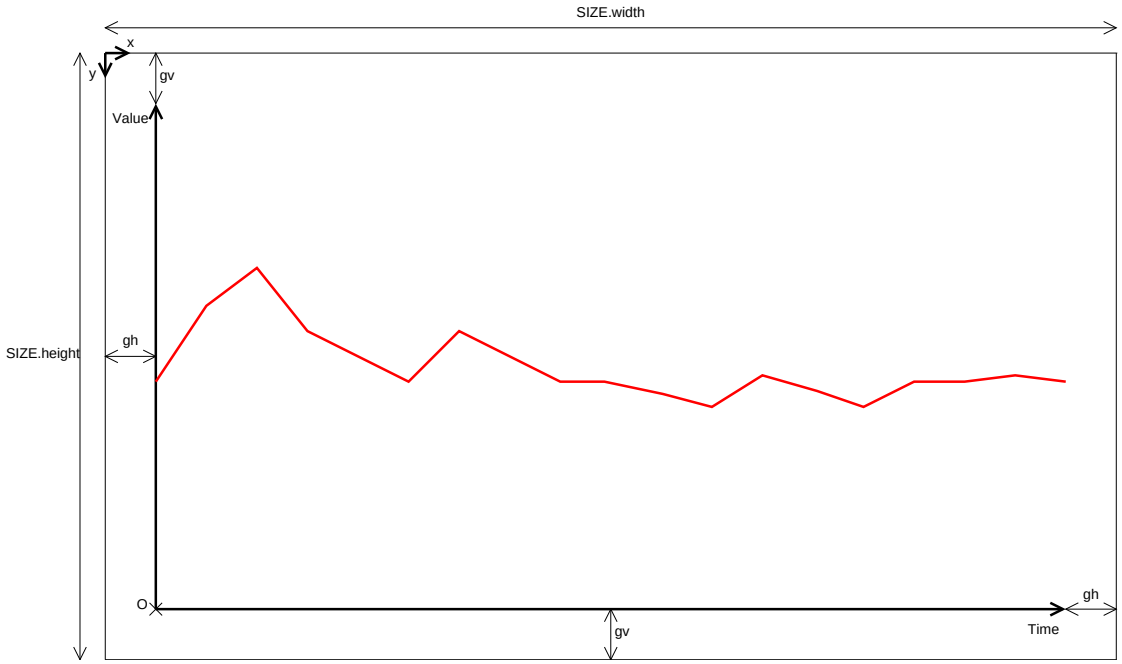


Abbildung 3.41: Konstruktion des Diagramms

Erweiterung

Im Laufe des Projekts können neue Sensoren, neue Arten von Einträgen ins Logbuch oder auch neue Typen von Navigationswerten hinzukommen. Dieses Programm ist soweit wie möglich auf diese Erweiterungen ausgelegt. Im Folgenden wird erklärt, wie das Programm um solche Dinge erweitert wird.

Hinzufügen eines neuen Sensors *NewSensor*:

1. Zuerst wird eine neue Klasse *NewSensor* in Package *main.sensors* erstellt. Diese ist von der abstraken Klasse *Sensor* abgeleitet. Die abstraken Funktionen müssen implementiert werden.
2. *main.SensorOrder* einen Eintrag für den Sensor hinzufügen.
3. Beim Konstruktor *main.Navigation.Navigation(double)* den neuen Sensor initialisieren.
4. Die Berechnung der zu dem neuen Sensor zugehörigen Navigationswerte in *main.Navigation.updateSimulation()* und *main.Navigation.calculate()* vornehmen.

Hinzufügen eines neues Navigationswertes im Logbuch:

1. In *main.logevents.NavigationDataElementOrder* den neuen Eintrag hinzufügen.
2. Im Konstruktor *main.logevents.NavigationData.NavigationData(double,...)* als neuen Parameter hinzufügen. Eventuell für *main.Navigation.calculate()* get- und set-Funktionen erstellen.
3. In *main.Log.parseNavigationData()* entsprechend den Eintrag hinzufügen.
4. Zu *main.Navigation.calculate()* den Eintrag wie bei den anderen Einträgen einfügen.

Hinzufügen eines neuen aufgezeichneten Kalibrierwertes:

1. In *main.logevents.Calibration* für jede Methode den Eintrag hinzufügen. Hier ist noch keine Auflistung wie bei *main.logevents.NavigationData* vorhanden.
2. *main.Log.parseCalibration()* anpassen.

Eine neue Art von Logeinträgen *NewLogEvent* hinzufügen:

1. Die Klasse *NewLogEvent* im Package *main.logevents* abgeleitet von *LogEvent* implementieren.

2. Parser hinzufügen: Bei *main.Log.parse()* vor *else{ OtherLogEvent otherLogEvent = ...* den Parser hinzufügen, Es ist Empfehlenswert eine neue Funktion *parseNewLogEvent(Time time, String data)* zu implementieren. Von dort aus kann dann der Konstruktor für *NewLogEvent* aufgerufen werden. Zum Schluss mit *logEvents.add(newLogEvent)* das neue Ereignis hinzufügen.

3. *main.Log.getNewLogEventData()*, welche nur diese neuen Ereignisse als Vektor zurück gibt, implementieren. Als Beispiel kann *getNavigationData()* genommen werden.

4. Für ein Unterfenster, welches die Einträge in Tabellenform anzeigt, muss nur die Klasse *gui.internalFrames.NewLogEvent* von *gui.internalFrames.DataLogFrame* abgeleitet werden. Im Konstruktor braucht nur der Titel des Fensters und die Funktion *log.getNewLogEventData()* angepasst werden. Als Beispiel kann man sich an *OtherDataLogFrame* orientieren.

3.7 Arbeitstechniken

In diesen Kapitel werden die Arbeitstechniken, Probleme, deren Lösungen und Alternativen beschrieben. Bekanntlich läuft in dem meisten Projekten nicht alles perfekt. So gab es auch hier verschiedene Entwürfe des Aufbaus und einige Probleme mit den Sensoren (Rauschen).

Für die Kalibrierung entstanden eine Reihe von Testergebnissen. Diese wurden in einer Tabellenkalkulation gespeichert und mit einfachen Statistiken ausgewertet. Dabei kam das Programm OpenOffice.org zum Einsatz. Jedoch wurde die Darstellung als Diagramm immer langsamer, je mehr Daten angezeigt werden sollten. Deshalb erfolgten die komplexeren Auswertungen im Simulationsprogramm. Geschrieben wurde es in Java. Dadurch ist von der Plattform unabhängig und kann somit auf fast jeden Computer gestartet werden. Mit der Objektorientierung wurde das Entwickeln komplexere Implementierungen einfacher.

3.7.1 Alternativen

Wie in den großen Ubooten, soll hier auch Inertialnavigationssystem eingesetzt werden. Der Unterschied dazwischen besteht darin, dass bei den großen Ubooten hochpräzise, teils mechanische Sensoren eingesetzt werden. Diese wären zu groß und teuer für unser Uboot. Somit musste eine preiswerte und integrierte Lösung entworfen werden.

Eins von den drei Gyroskopen, die hier verwendet werden, kostet knapp hundert Euro. Um das Budget nicht zu überschreiten, sind mehrere Alternativen in Betracht gezogen worden. Eine Alternative war die, zwei Beschleunigungssensor im Bug und im Heck einzubauen. Durch die unterschiedlichen Beschleunigungen zwischen Bug und Heck sollte eine Drehung um die z-Achse bestimmt werden können. Die Neigungen um die x- und y-Achse sollten mit einem Neigungssensor bestimmt werden (Abbildung 3.42).

Die zweite Alternative war der jetzigen recht ähnlich. Die z-Achse wurde mit einem Gyroskop gemessen. Damit war nur ein Beschleunigungssensor für die x- und y-Richtung nötig.

Es waren leider kaum passende Neigungssensoren zu finden. Wenn, dann waren diese teurer als die Gyroskope oder stellten sich als Kippschalter heraus. So wurden dann doch alle Achsen mit den Gyroskopen gemessen.

3.7.2 Probleme und Lösungen

Die meisten Probleme gab es bei den Sensoren. Diese haben einen analogen Ausgang. Dadurch wird nicht nur der eigentliche Sensorwert übermittelt, sondern auch eine Reihe weiterer, unnützer und schwer filterbare Informationen. Das sind die Stabilität der Versorgenspannung, Vibrationen, Einflüsse benachbarter ungeschirmter Kabel (zum Beispiel die Servoverbindungen) oder sonstiges allgemeines Rauschen. Das Rauschen wurde in Kapitel 3.4.3 schon behandelt.

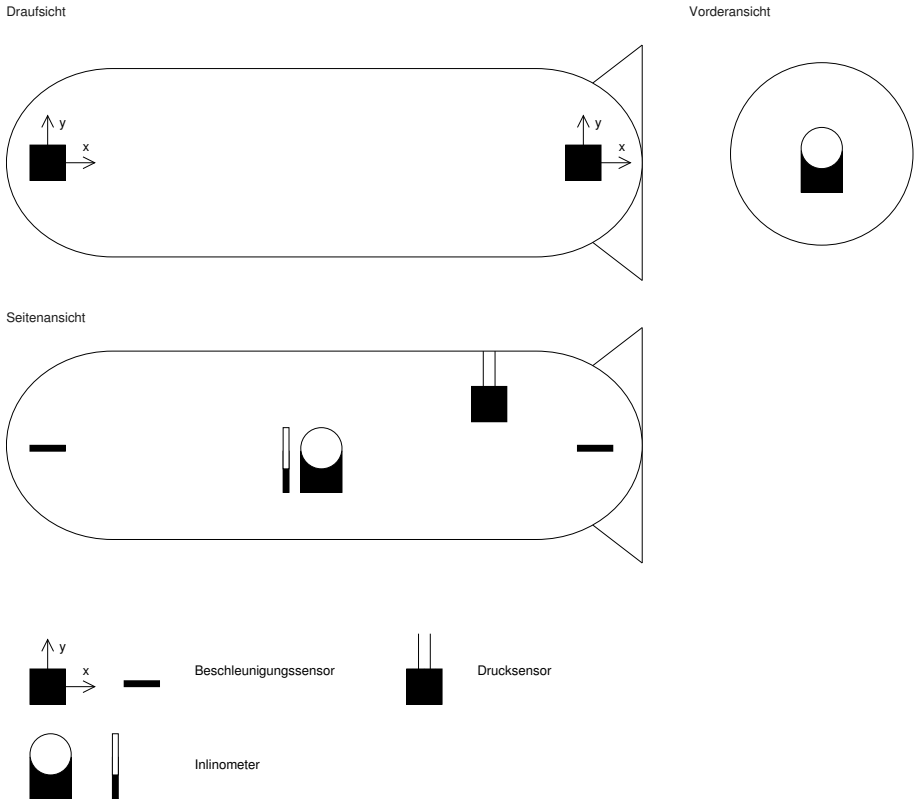


Abbildung 3.42: Ein alternativer Aufbau für die Navigation

Eine schwankende Versorgungsspannung stellt ein weiteres Problem dar. Es zeigte sich, das sich die Sensorparameter, insbesondere der Offset, änderten, wenn der Drucksensor angeschlossen wurde. Auch zwischen den Betrieb am USB-Port und im Uboot sind Unterschiede festzustellen.

Zuerst sollte der Offset beim Start dynamisch bestimmt werden, um eine flexible Anpassung zu erhalten. Bei der Implementierung zeigte sich aber ein nicht deterministisches Verhalten. Mal war der eingelesene Offset zu klein oder zu groß. Außerdem war die Empfindlichkeit nicht dynamisch bestimmbar. So wurde die Idee dann verworfen. Über das Hostsystem sollen aber die Parameter nachträglich im Betrieb justiert werden können. Somit ist ein stabile Spannungsversorgung notwendig. Dieses ist dann Aufgabe der Elektrik. Bei den digitalen Sensoren traten diese Probleme nicht mehr so stark auf.

Weitere Probleme gab es beim Testen. Das Board hat natürlich während der Fahrt keine direkte Ausgabe. Um zu wissen, welche Navigationswerte es gab, muss eine Möglichkeit implementiert werden, diese Daten auszulesen. Die erste Möglichkeit ist es, alle Daten über das Hostsystem auszulesen. Dies ist jedoch nur möglich, wenn ein Funkkontakt besteht und, aufgrund der langsamen Datenübertragung, nur wenn es explizit aufgerufen wird. Was zwischen zwei Aufrufen passierte, kann nur spekuliert werden.

Im ersten Meilenstein wurden die Daten in ein Array gespeichert, denn ein Festspeicher für das Board gab es damals noch nicht. Das Array hatte einige Nachteile. Die Kapazität war auf 2400 Frames beschränkt. Wird diese überschritten, so werden die ältesten Einträge überschrieben. Stürzte das Board ab oder wurde die Spannungsversorgung unterbrochen, so gingen die aufgezeichneten Daten verloren, da diese nur im RAM sind. Erst mit der Compact-Flash und der SD-Karte gab es ein Syslog und es war möglich, Navigationsdaten zu sammeln. Mit dem Simulationsprogramm konnte man die Daten nun vernünftig analysieren.

Die Zuverlässigkeit der Elektrik war auch Problem. Von Zeit zu Zeit fiel irgendeine Komponente aus. Ärgerlich ist, wenn ausgerechnet das Board ausfällt, so dass es sich darauf nicht mehr arbeiten lässt. Das Board ist aber nötig, besonders am Ende des Projekts, da hier für die Kalibrierung mit realen Werten und möglichst in der realen Umgebung (Spannungsversorgung) gearbeitet werden muss. Am Ende des Projekts fiel dann noch das y-Gyroskop aus. Dieses wurde durch das x-Gyroskops ersetzt. Die Neigung der x-Achse

ist stabil und wird nicht geregelt.

3.7.3 Ausblick

Die Initialnavigation hier kann auch weiter fortgeführt werden. Einige geplante Ziele konnten aus Zeitgründen nicht mehr erreicht werden. Die gesamte Berechnung, die hier anfällt, kann mit einem erweiterten Kalmanfilter realisiert werden. Die Parameter dafür müssen aber noch über eine Simulation gefunden werden. Es wäre interessant, wie sich das System im Vergleich zu hier verhalten würde.

Auf dem Board gibt es keine Fließkommaeinheit. Berechnungen mit Fließkomma müssen hier simuliert werden. Man kann versuchen, die Berechnungen auf Basis von Festkommazahlen umzustellen. Dabei muss getestet werden, wieviel Stellen nötig sind, um die gewünschte Genauigkeit zu bekommen. Zu viele Stellen dürfen es aber auch nicht sein, da es sonst zu einem Überlauf der vorhanden Stellen kommen kann. Dann ist der Zustand der Initialnavigation undefiniert und es müsste neu gestartet werden.

Die gesamte Implementierung kann auch in Hardware, zum Beispiel in VHDL, geschrieben werden. Dann können zum Beispiel die Navigationswerte in einen festen Takt immer wieder berechnet werden. Die Berechnung erfolgt dann parallel und wird nicht durch andere Tasks unterbrochen.

Kapitel 4

Hardware Plattform

4.1 Motivation

Um die die Steuerung der Tauchplattform mittels des verwendeten FPGAs zu realisieren, bedarf es einer Umformung der verwendeten Steuerimpulse, da Motoren bzw. Steuerglieder mit anderen Spannungen und höheren Strömen arbeiten. Darüber hinaus müssen die mit Hilfe von Sensoren ermittelten analogen Signale in digitaler Form zur Weiterverarbeitung im FPGA zur Verfügung gestellt werden. Es muss eine drahtlose Kommunikation zur Tauchplattform für die Übermittlung von Steuerinformationen und von der Tauchplattform zur Host-Software zur Datenübermittlung gewährleistet werden. Eine digitale Kamera soll Bilder an das verwendete FPGA weiterleiten können. Außerdem muss eine dauerhafte Datenspeicherung innerhalb der Tauchplattform möglich sein. Alle elektronisch relevanten Teile müssen auf geeignete Weise innerhalb der Plattform miteinander verbunden werden.

Folgende Teile wurden zu diesem Zweck realisiert:

- Antriebsmotorsteuerung, die mit Hilfe digitaler Signale den Hauptantrieb in Drehgeschwindigkeit und -richtung steuert
- Tauchtankmotorsteuerung, die mit Hilfe digitaler Signale die verwendeten Tauchtanks füllt bzw. leert
- Drehzahlermittlung für alle verwendeten Motoren und digitale Aufbereitung des Signals
- Ermittlung des fließenden Antriebsmotorstroms und digitale Aufbereitung des Signals
- Erstellen einer Plattform für die verwendeten Sensoren und digitale Aufbereitung der auflaufenden Signale
- Erzeugung einer stabilen Versorgungsspannung für alle elektronischen Bauteile
- Realisation einer drahtlosen Kommunikation mit der Host-Software
- Integration eines nichtflüchtigen Datenspeichers in die Tauchplattform
- Anbindung einer digitalen Kamera an das FPGA zur Aufnahme von Standbildern

Diese Baugruppen wurden während des Projektes ständig an neue Anforderungen angepasst und erweitert. Ein besonderer Schwerpunkt lag dabei in der Miniaturisierung der nötigen Baugruppen, da die verwendete Tauchplattform nur ein begrenztes Platzangebot hat. Darüber hinaus wurden einige der Baugruppen zusammengefasst, um noch mehr Platz sparen zu können und um Störungen durch elektomagnetische Einflüsse möglichst gering zu halten. So wurde z.B. die zunächst realisierte Sensorplatine durch die Verwendung von digitalen Sensoren, die sich direkt an das verwendete FPGA anschließen ließen, überflüssig.

4.2 Einleitung

Zur Realisation der verschiedenen Baugruppen wurde das Platinenlayoutprogramm 'Eagle' verwendet. Bei einigen Platinen wurde jedoch auch auf eine Lochrasterverdrahtung zurückgegriffen, um die Entwicklungszyklen möglichst kurz halten zu können und den Projektverlauf bei kurzfristigen Anderungen nicht unnötig zu verzögern. Geätzte Platinen wurden dabei grundsätzlich zweilagig ausgeführt, um den Platzverbrauch der einzelnen Komponenten möglichst gering halten zu können. Alle logischen Verbindungen sowie Spannungsversorgungsleitungen, die nur geringe Leistungen zu übertragen hatten, wurden mit Hilfe von Flachbandleitungen realisiert. Alle anderen Spannungsversorgungsleitungen wurden als Einzeladern ausgeführt.

4.3 Grundlagen Motorsteuerung

4.3.1 Spannung und Strom

Da in der Halbleitertechnik nur kleine Spannungen zum Weiterleiten von Steuersignalen verwendet werden, ist es nicht möglich, einen leistungsstarken Motor direkt an einem FPGA, wie er in unserer Plattform Verwendung findet, zu betreiben. Außerdem sind die maximalen Ströme, die durch einen FPGA fließen können, sehr begrenzt, da der Strom nur zur Übermittlung logischer Signale (Ja/Nein, An/Aus) verwendet wird, wozu allerdings kaum ein Stromfluß nötig ist. Daher müssen die vom FPGA erzeugten logischen

Signale auf geeignete Weise verstärkt werden, um Motoren ansteuern zu können.

4.3.2 PWM-Signal

Um den prozentualen Anteil zu beschreiben, mit dem ein Motor in Bezug auf seine Gesamtleistung betrieben werden soll, nutzen wir ein PWM(PulsWeitenModulation) Signal. Dieses Signal pulst analog zum gewünschten prozentualen Leistungsnivau (Z.B. für 100 Prozent ein Dauersignal, für 50 Prozent ein Signal mit gleich langen Ein- und Ausphasen). Der große Vorteil eines solchen Signals ist die einfache Möglichkeit der Weiterverarbeitung innerhalb einer Leistungsstufe, da ein Elektromotor auf genau die selbe Weise angesteuert werden kann. Durch die Trägheit seiner rotierenden Masse verhält sich ein Elektromotor, der kurzzeitig mit Pulsen voller Spannung betrieben wird, sehr ähnlich wie ein Motor, der mittels einer veränderbaren Spannung in seiner Leistung geregelt wird.

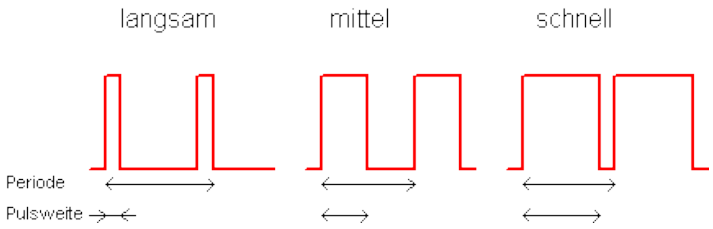


Abbildung 4.1: Beispiel PWM Signal

Die zu entwickelnde Leistungselektronik muss also nur die von der Steuerung übermittelten Signale entsprechend der verwendeten Motorspannung aufbereiten und einen geeignet hohen Stromfluß ermöglichen.

4.3.3 H-Brücke

Die verwendeten Motoren müssen nicht nur in ihrer Leistung regelbar sein, es ist darüber hinaus nötig, ihre Drehrichtung ändern zu können. Zu diesem Zweck nutzen wir eine so genannte H-Brücken Schaltung, die über vier schaltbare Verbindungen, von der jeweils zwei geöffnet und zwei geschlossen sind, die Polung des Motors umkehren kann.

4.3.4 Motorstrom

Um den fließenden Motorstrom mit Hilfe eines Analog/Digital-Wandlers messen zu können, muss der Strom über eine Spannung gemessen werden, die sich analog zum Stromfluss ändert, da Analog-Digital-Wandler nur Spannungen erfassen können. Die Ermittlung des Motorstroms dient zum Einen der möglichst schnellen Erfassung einer Überlastung, noch bevor eine Sicherung auslösen muss, zum Anderen kann sie der Regelung zur Einschätzung der abgegebenen Motorleistung dienen.

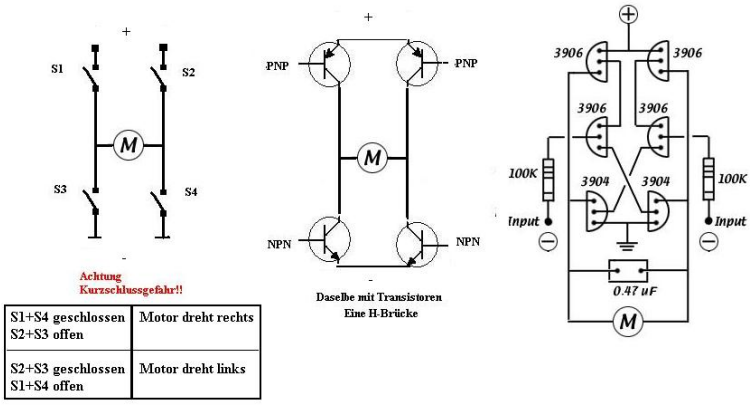


Abbildung 4.2: Beispiel H-Brücke

4.3.5 Motordrehzahl

Die Drehzahl eines Elektromotors ändert sich nicht analog zu der angelegten Spannung, wie man zunachst vermuten könnte. Vielmehr hängt die die Drehzahl entscheidend von der Belastung des Motors ab. Da man aus diesem Grund nicht anhand der an den Motor angelegten Spannung auf die Drehzahl schließen kann, muss diese gemessen werden, um für Regelungszwecke die abgegebene Leistung zu ermitteln. Bei der Ansteuerung der Tauchtanks ist eine Erfassung der Drehzahl außerdem nötig, um den Füllstand der Tanks möglichst genau ermitteln zu können.

4.4 Realisierung

4.4.1 Antriebsmotorsteuerung

Die Steuerung des Antriebsmotors erfolgt vom FPGA mit Hilfe eines PWM-Signals und eines Signals zur Steuerung der Drehrichtung. Da der verwendete Motor mit einer Spannung von 12V angesteuert wird und und einen maximalen Anlaufstrom von 30A besitzt, musste die verwendete Schaltung entsprechend ausgelegt werden. Verwendung gefunden haben hier MosFET Transistoren, die mit einer Steuerspannung nahezu stromlos geschaltet werden können und aufgrund ihres geringen Innenwiderstandes nur sehr geringe Abwärme erzeugen. Die verwendeten Transistoren können bei einer Spannung von 12V Ströme bis zu 60A schalten. Die Schaltung beinhaltet also noch genug Sicherheitsreserven. Um mit Hilfe der H-Brücken Schaltung eine Drehrichtungsumkehr erreichen zu können, wurden vier MosFET Transistoren verwendet. Da der geringe Innenwiderstand von MosFET Transistoren nur dann erreicht wird, wenn die Steuerspannung möglichst der Lastspannung entspricht, wurde jedem MosFET Transistor ein Bipolarer Transistor vorgelagert, welcher mit Hilfe eines Optokopplers angesteuert wurde. Durch die Optokoppler ist die Schaltung einerseits galvanisch von der Steuerelektronik getrennt, zum Anderen erleichterten diese Bausteine eine Anpassung an den logischen Spannungspegel von 3.3V des verwendeten FPGA. Die Drehrichtung kann einfach durch das Schalten einer Masse auf den entsprechenden Optokoppler geregelt werden. Sollten durch eine Fehlfunktion beide Optokoppler zeitgleich geschaltet werden, so bleibt der Motor aufgrund des fehlenden Potentialunterschieds stehen. Ein Kurzschluss ist nicht möglich. Dieses Verhalten der Schaltung kann auch dafür genutzt werden, um eine so genannte Motorbremse zu implementieren. Die Hauptantriebssteuerung wurde nach ihrer Fertigstellung nicht noch einmal überarbeitet, da sie sofort zufriedenstellend funktionierte. Auch der Wechsel des verwendeten Antriebsmotors führte aufgrund der großzügig ausgelegten Leistungsreserven zu keinerlei Problemen.

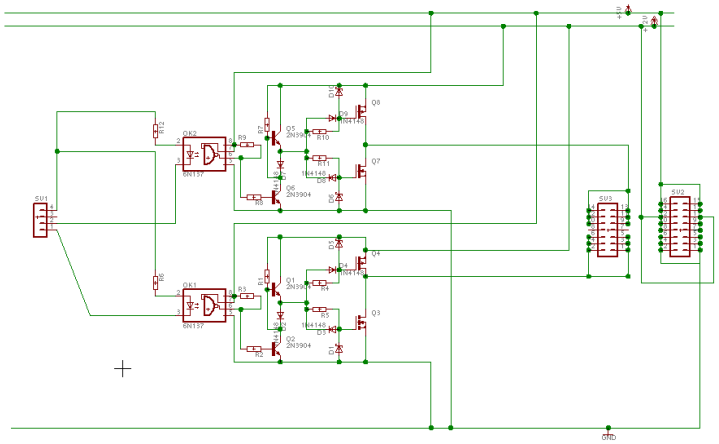


Abbildung 4.3: Schaltplan Antriebssteuerung

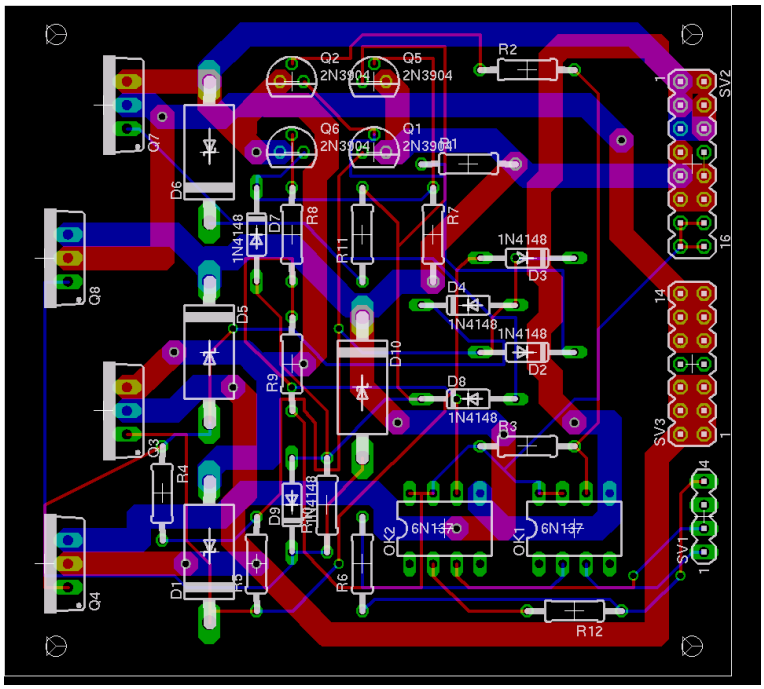


Abbildung 4.4: Layout Antriebssteuerung

4.4.2 Tauchtankmotorsteuerung

Auch bei der Tauchtanksteuerung war für die verwendeten Motoren gefordert, dass sie sich mit Hilfe eines PWM-Signales in der Drehzahl und durch ein Steuersignal in der Drehrichtung beeinflussen lassen können müssen. Da die zur Benutzung geplanten Motoren eine maximale Stromaufnahme von 1A aufwiesen, bot sich die Verwendung eines integrierten Bausteins an, der bereits zwei vollständige H-Brücken implementiert. Dieser Baustein wurde durch acht Freilaufdioden ergänzt, die die Zerstörung des Bausteins durch rückinduzierte Spannungen aus dem Motor verhindern sollen. Die erste Revision der Platine enthielt noch keine Optokoppler, da diese aufgrund der Kurzschlußfestigkeit zwischen Steuer- und Lastkreis des verwendeten Bausteins nicht notwendig erschienen. Allerdings wurde der verwendete Baustein baubedingt nur mit einer Steuerspannung von 3.3V aus dem FPGA versorgt. Dies führte zu einer erheblich beschleunigten Alterung des Bausteins, da sein Innenwiderstand wesentlich über dem Arbeitsbereich lag. Die daraus resul-

tierende Erwärmung sorgte für eine Zerstörung des Bausteins innerhalb von nur 10 Betriebsstunden. Durch die Verwendung von Optokopplern konnte dieses Problem beseitigt werden. Darüber hinaus wurden einige Eingangssignale für den H-Brücken Baustein in der neuen Platinenrevision durch einen Nand-Baustein aus anderen Eingangssignalen erzeugt, um Signalpins am FPGA einsparen zu können. Um den Platzverbrauch weiter zu reduzieren, wurden außerdem sämtliche Signale, die die Tauchtanks zur Verfügung stellen (Drehzahlmesser, Endschalter) über diese Platine zum FPGA geleitet. Dies wurde vorher von einer externen Lochrasterplatine übernommen. Leider stellte sich auch die neue Revision der Tauchtankplatine als

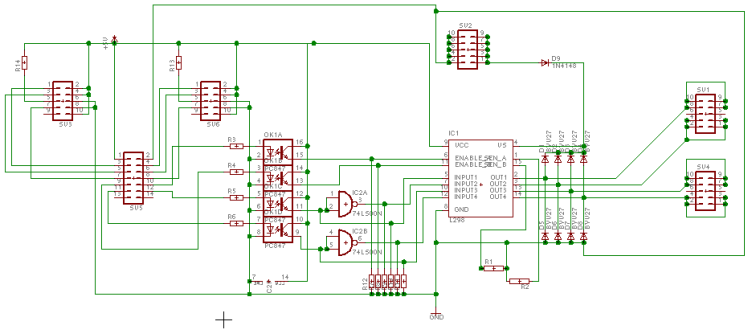


Abbildung 4.5: Schaltplan Tauchtanksteuerung

problembehaftet heraus. Die verwendeten H-Brücken-Bausteine fielen sehr oft aus nicht nachvollziehbaren Gründen aus. Eine Analyse der anliegenden Steuersignale mit Hilfe eines Oszilloskops zeigte dann, dass die Regelung der Tauchplattform unter bestimmten Vorraussetzungen ein Signal erzeugte, welches die Drehrichtung der Motoren mehr als 100 Mal in der Sekunde wechselte. Durch die Trägheit der Motoren blieben diese stehen und verursachten einen Stromfluß durch die Bausteine, der einem Vielfachen ihrer Nennbelastbarkeit entsprach. Nachdem innerhalb des FPGA ein Sicherungsmechanismus integriert wurde, welcher maximal einen Wechsel der Drehrichtung pro Sekunde zulässt, funktionierte die Tauchtankplatine einwandfrei. Da sich eine H-Brücke auf MosFET-Transistor Basis bei der Hauptantriebssteuerung als sehr zuverlässig erwiesen hatte, tauschten wir die verwendeten integrierten H-Brücken Bausteine in einer letzten Revision der Tauchtankplatine gegen Bausteine solcher Bauart aus. Da diese Bausteine darüber hinaus Freilaufdioden enthielten, konnten diese auf der Platine entfallen. Die Platine wurde dadurch noch kompakter, obwohl die verwendeten MosFET Bausteine nur noch eine H-Brücke enthielten und daher doppelt ausgelegt werden mussten.

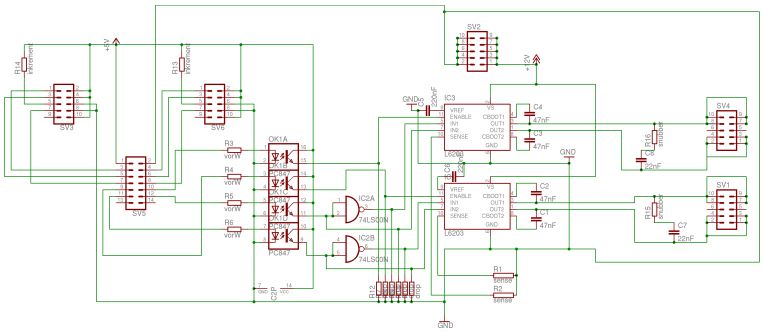


Abbildung 4.6: Schaltplan Tauchtanksteuerung finale Revision

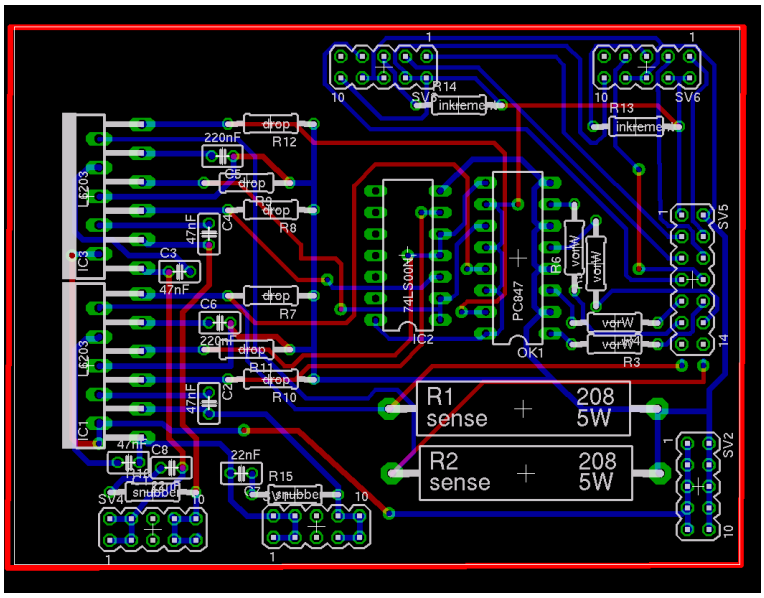


Abbildung 4.7: Layout Tauchtanksteuerung finale Revision

4.4.3 Drehzahlermittlung Hauptantrieb

Die Realisierung der Drehzahlermittlung des Hauptantriebs sollte zunächst mit Hilfe eines optischen Inkrementalgebers erfolgen, der eine auf der Antriebsachse angebrachte klare Kunststoffscheibe, die mit schwarzen Streifen versehen wurde, abtastet. Das vom Inkrementalgeber erzeugte Pulssignal hätte direkt an den FPGA weitergeleitet und dort zur Berechnung der Drehzahl genutzt werden können. Leider stellte sich heraus, das für diesen Ansatz nicht genug Platz im Rumpf vorhanden war und darüber hinaus der gewählte Sensor Probleme mit den vom Hauptantrieb ausgehenden Vibrationen hatte.
Da die Anforderungen an die Genauigkeit der Drehzahlerfassung am Hauptantrieb nicht sehr hoch sind, wurde statt einem fertigen Inkrementalgeber ein einfacher Reed-Kontakt Schalter verwendet. Reed-Kontakt Schalter schalten in dem Moment, wo sie in den Einflussbereich eines Magnetfeldes gelangen. Der Reed-Kontakt wurde am Motor befestigt, so dass ein am Lüftungsrad des Motors angebrachter Dauermagnet bei jeder vollständigen Umdrehung einen kurzen Schaltimpuls auslöst. Dieses Signal konnte direkt durch den FPGA ausgewertet werden. Da die Hauptantriebsdrehzahl innerhalb der Regelung nicht verwendet wurde,

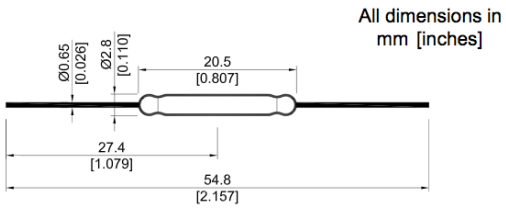


Abbildung 4.8: Reed Kontakt

wurden die zur Erfassung nötigen Bauteile nach dem Wechsel des Antriebsmotors nicht wieder eingebaut.

4.4.4 Drehzahlermittlung Tauchtanks

Die Ermittlung der Drehzahl war bei den Tauchtanks erheblich schwieriger, da dort ein sehr genauer Wert benötigt wird, um den korrekten Füllstand der Tauchtanks berechnen zu können. Zu diesem Zweck wurden optische Inkrementalgeber eingesetzt, die eine Auflösung von 120 Impulsen pro Umdrehung haben. Um

die für die Inkrementalgeber nötigen Kunststoffscheiben an den Motoren der Tauchtanks anbringen zu können, wurden die Motoren durch ein Modell mit durchgehender Achse ersetzt.

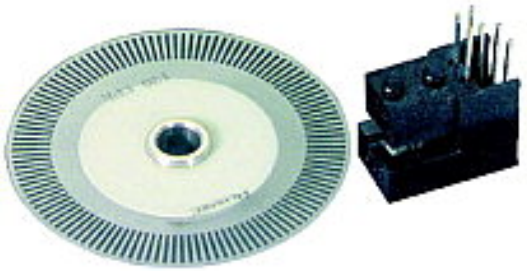


Abbildung 4.9: Inkrementalgeber mit Taktscheibe

4.4.5 Motorstromermittlung

Um den Motorstrom im Hauptantriebsmotor und in den Tauchtankmotoren ermitteln zu können, wurde in den Lastkreis der Motoren ein sehr kleiner Hochlastwiderstand integriert. Bei den Tauchtankmotoren befindet sich dieser Widerstand auf der Steuerplatine, beim Hauptantrieb wurde er in die Verkabelung zum Motor eingeschliffen. Da in einer Reihenschaltung der Strom innerhalb aller Verbraucher gleich groß ist, kann man anhand der am Widerstand abfallenden Spannung auf den fließenden Strom schließen. Dabei sind die Spannung und der Strom am Widerstand linear proportional. Da die Ströme der Motoren innerhalb der Regelung keine Verwendung fanden und sich keine analogen Sensoren mehr in der Tauchtankplattform befinden, die einen A/D Wandler erfordern würden, werden diese Daten nicht mehr vom FPGA erfasst.

4.4.6 Sensorplattform

Die verwendeten Sensoren wurden weitestgehend auf einer Platine angeordnet, zusammen mit dem für die Auswertung nötigen Analog-Digital-Wandler. Da die verwendeten Gryoskope und der Beschleunigungssensor Spannungen innerhalb des Arbeitsbereiches des AD-Wandlers ausgeben, war eine Implementierung relativ einfach.

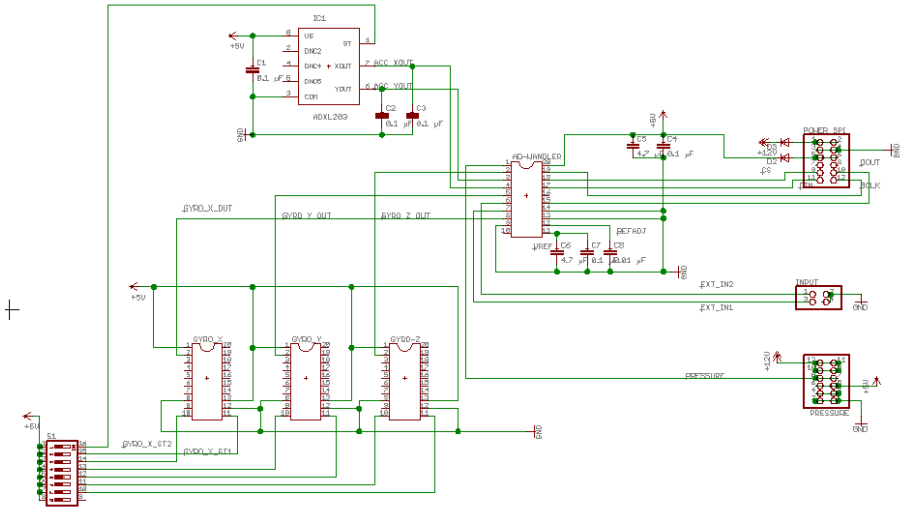


Abbildung 4.10: Schaltplan Sensorplatine

Die Ausgangsspannung des zunachst verwendeten Drucksensors musste hingegen mit Hilfe eines Operationsverstarkers angepasst werden. Außerdem wurde der Drucksensor auf einer gesonderten Platine montiert, da er im Heck der Tauchplattform Platz finden musste. Leider stellte sich der verwendete Drucksensor später als ungeeignet heraus und wurde durch eine neue Variante ersetzt. Dadurch wurde auch die Verwendung eines Operationsverstärkers überflüssig und auf das Design einer neuen Platine zugunsten einer Lochrasterverdrahtung verzichtet.

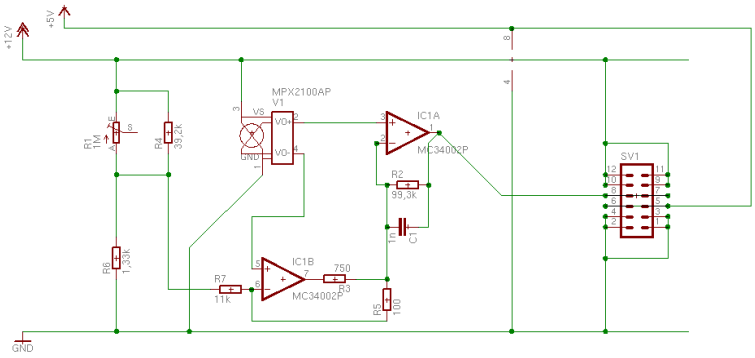


Abbildung 4.11: Schaltplan Drucksensorplatine

Da sich die Verarbeitung der Signale aus den analogen Sensoren zunehmend als problematisch herausstellte, wurden diese durch digitale Pedanten ersetzt. Dies hatte zur Folge, dass eine aufwendige Schaltung zur digitalisierung analoger Sensorsinale nicht mehr notwendig war und die entsprechende Platine entfallen konnte. Statt dessen wurde die Tochterplatine, die zwischen dem FPGA-Board und den einzelnen Platinen geschaltet war, um Anschlüsse für digitale Sensoren erweitert.

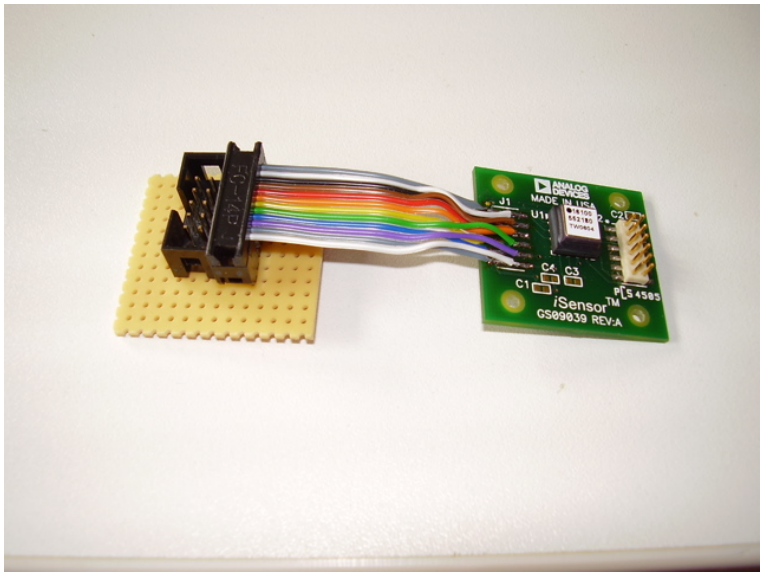


Abbildung 4.12: Digitaler Sensor

4.4.7 Tochterplatine

Um alle elektronischen Komponenten der Tauchplattform mit dem FPGA zu verbinden, wurde eine Tochterplatine realisiert, die auf der einen Seite sämtliche Anschlussleitungen aufnimmt und die entsprechenden

Signale an die I/O Pins des FPGA weiterleitet, aber auf der anderen Seite auch die Versorgungsspannungen für einzelne Komponenten zur Verfügung stellen kann. Bei der ersten Revision der Tochterplatine war diese noch ziemlich klein und sehr spezifisch für die angeschlossenen Komponenten ausgelegt. Dies resultierte vor Allem daraus, dass zunächst alle elektronischen Komponenten direkt mit den I/O Pins des FPGA-Boards verbunden wurden, um möglichst schnell deren Funktionalität testen zu können. Dies wurde aber auf lange Sicht zu unübersichtlich und die Gefahr einer Falschverdrahtung war recht hoch. Die erste Revision der Tochterplatine wurde genau für die bisher an das FPGA-Board angeschlossenen Komponenten entwickelt.

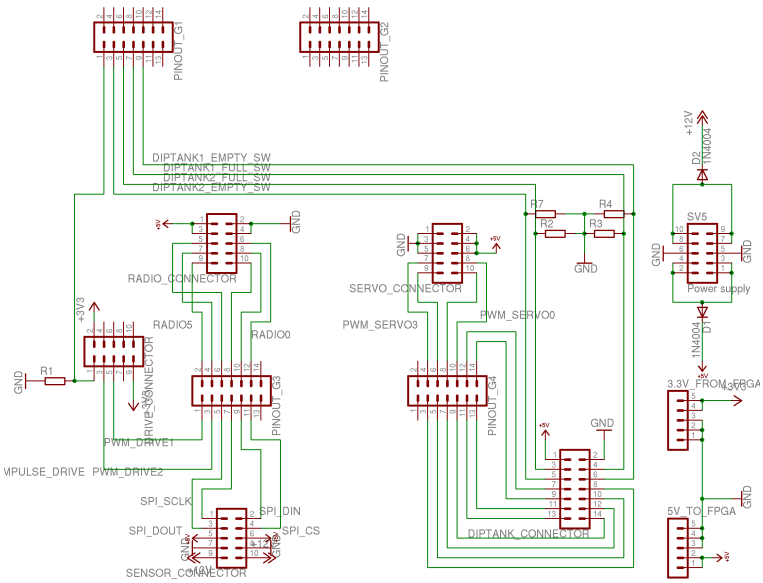


Abbildung 4.13: Tochterplatine 1. Revision Schaltplan

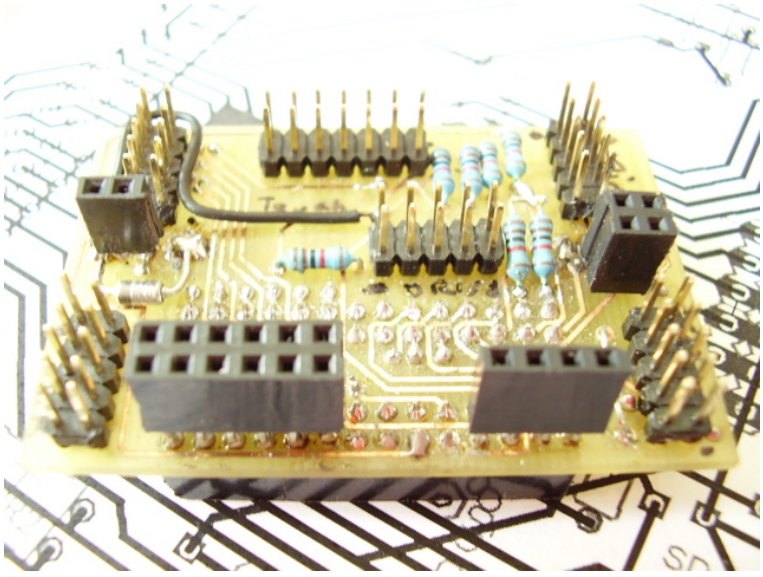


Abbildung 4.14: Tochterplatine 1. Revision Bild

Der Einsatz eines neuen, mit mehr I/O Pins bestückten FPGA-Boards machte auch die Entwicklung einer

neuen Tochterplatine nötig. Dabei wurde auf dieser Tochterplatine nur für die Komponenten, die keiner weiteren Anderung mehr bedurften, eine zur ersten Revision der Tochterplatine kompatible Steckverbindung integriert. Für alle anderen Komponenten, insbesondere für die neuen digitalen Sensoren, wurden standarisierte Ports realisiert, durch die es möglich wurde, auch zu einem späteren Zeitpunkt weitere Komponenten zu integrieren und vorhandene Komponenten ohne Neuverdrahtung an andere Ports anzuschließen.
Die Erfahrung mit digitalen Sensoren hatte gezeigt, dass diese immer nur eines von zwei Standardprotokollen verwenden. Daher wurden für die Protokolle I2C und SPI standarisierte Steckverbindungen auf der Tochterplatine geschaffen. Um die Möglichkeit von Falschverdrahtungen weitgehend ausschließen zu können wurden auf der neuen Tochterplatine verpolungssichere Flachband-Pfostenstecker oder aber Aussparungen in Pinleisten verwendet.
Da die Tochterplatine durch die gesteigerten Anschlussmöglichkeiten sowie die höhere Anzahl von I/O Pins auf dem neuen FPGA-Board eine Größe erreicht hatte, die sich mit dem Layout-Programm 'Eagle' in der Freeware-Version nicht mehr erstellen ließ, teilten wir die Platine in zwei Teilbereiche, die dann aber nebeneinander auf eine einzelne Platine geätzt wurden.

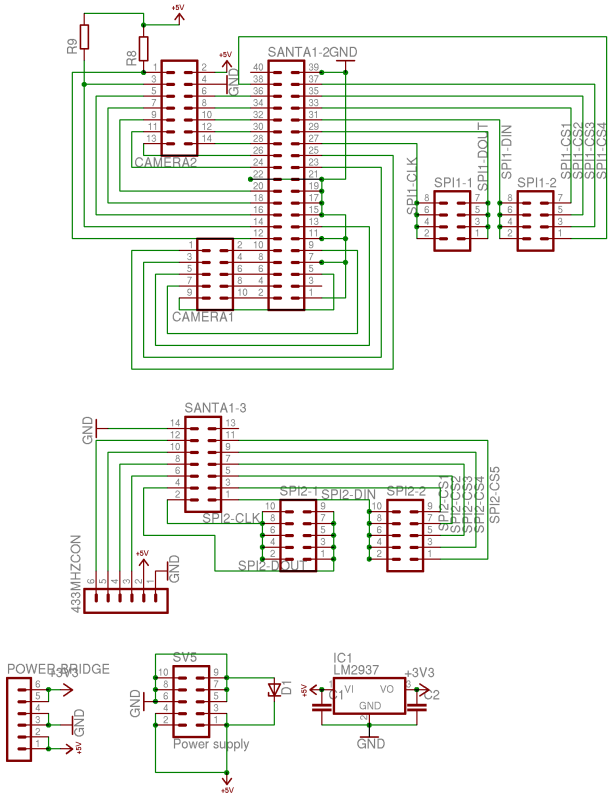


Abbildung 4.15: Tochterplatine finale Revision Schaltplan Teil A

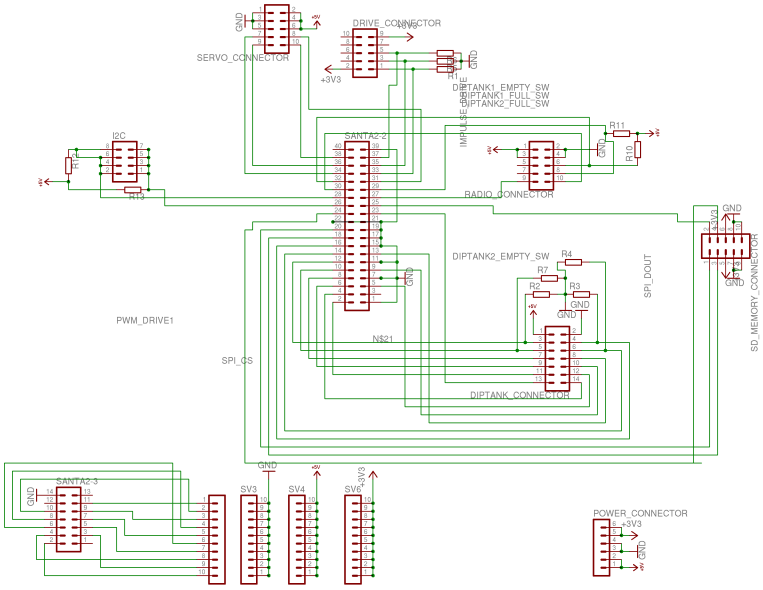


Abbildung 4.16: Tochterplatine finale Revision Schaltplan Teil B

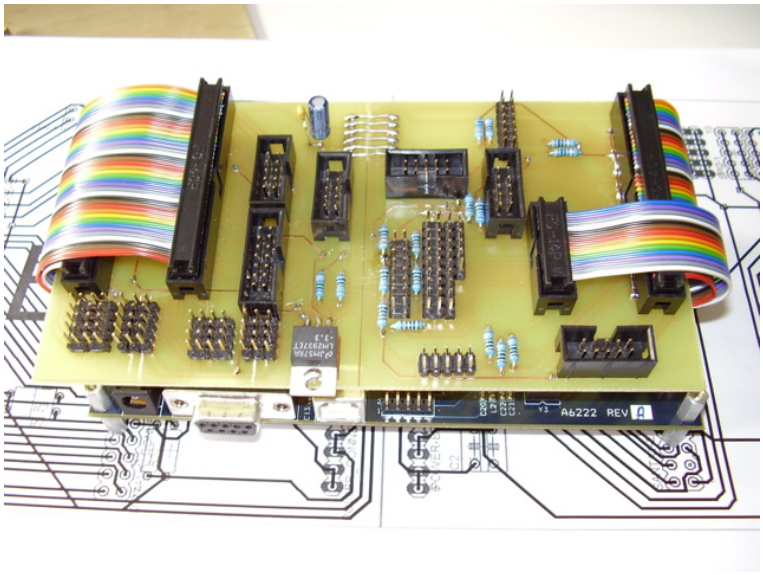


Abbildung 4.17: Tochterplatine finale Revision Bild

4.4.8 Versorgungsspannung

Da die Spannungsversorgung innerhalb der Plattform mit Hilfe von 12V BleiAkkus realisiert wurde, musste ein Weg gefunden werden, eine stabile 5V Versorgung für den FPGA, die Sensoren, sowie die Servomotoren zur Verfügung stellen zu können. Dies wurde zunächst mit mehreren Linearreglern realisiert, allerdings stellte sich schnell heraus, das diese bei stark schwankenden Eingangsspannungen, verursacht hauptsächlich durch die ungleichmäßige Belastung der Akkus durch die Motoren, keine ausreichend stabile Spannungsversorgung garantieren konnten. Vor Allem die Genauigkeit der Sensorik leidet bereits bei kleinen Schwankungen der Spannungsversorgung. Desshalb wurden die Linerarregler durch ein Schaltnetzteil ersetzt, welches auch bei hoher Last eine maximale Spannungsschwankung von 0,1V garantiert.

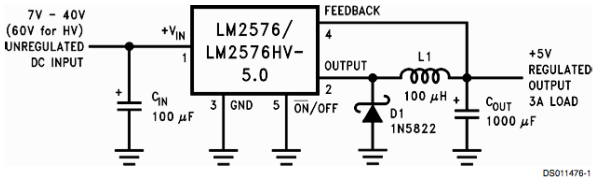


Abbildung 4.18: Schaltplan Schaltnetzteil

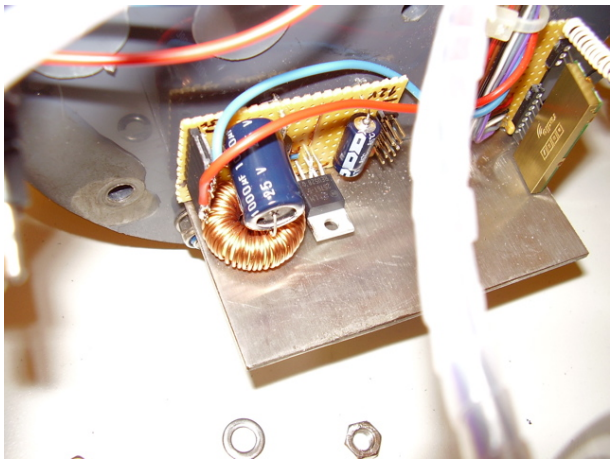


Abbildung 4.19: Bild Schaltnetzteil

4.4.9 Verdrahtung

Die Verbindung zwischen den einzelnen elektronischen Komponenten wurde mit Hilfe von Flachbandkabel vorgenommen. Dies wurde auch zur Verteilung der benötigten Versorgungsspannungen für die einzelnen Komponenten verwendet, wenn dies die maximale Stromaufnahme der Komponenten erlaubte. Alle anderen Spannungsversorgungsleitungen wurden als Einzeladern ausgeführt, wobei Leitungen mit einem Querschnitt von 1 und 2,5 mm zum Einsatz kamen. Alle Motoren wurden mit Hilfe von KFZ-Sicherungen abgesichert, ebenso das 5V Schaltnetzteil und jede Batterie.

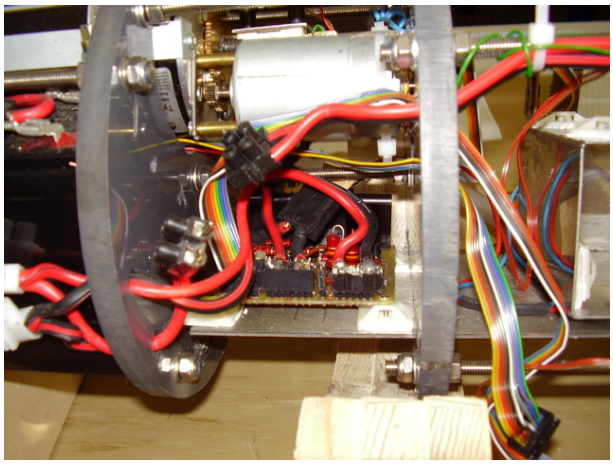


Abbildung 4.20: Bild Verdrahtung

4.4.10 Funkmodule

Zur Kommunikation mit der Tauchplattform kamen zunächst zwei verschiedene Funksysteme zum Einsatz. Zum einen wurde eine im Modellbau übliche 40Mhz Fernbedienung mit der passenden Empfangseinheit verwendet. Diese Empfangseinheit ließ sich relativ einfach in die Tauchplattform integrieren, da sie PWM-Signale erzeugt, die das verwendete FPGA direkt weiterverarbeiten kann.
Darüber hinaus wurden Funkmodule im 433Mhz Band verwendet, um eine serielle Datenkommunikation zwischen Host-Software und Tauchplattform zu realisieren. Die verwendeten Module beinhalten dabei bereits ein RS232 kompatibles Protokoll, welches mit TTL Pegeln arbeitet. Daher konnte das Modul auf der Tauchplattformseite problemlos an das FPGA angeschlossen werden, da dieser ebenfalls mit TTL Pegeln arbeitet. Auf der PC Seite mussten die Signalpegel des Funkmoduls mit Hilfe eines RS232 Pegelwandlerbausteins auf den PC Spannungspegel angehoben werden. Diese Schaltung wurde aus Zeitgründen und da

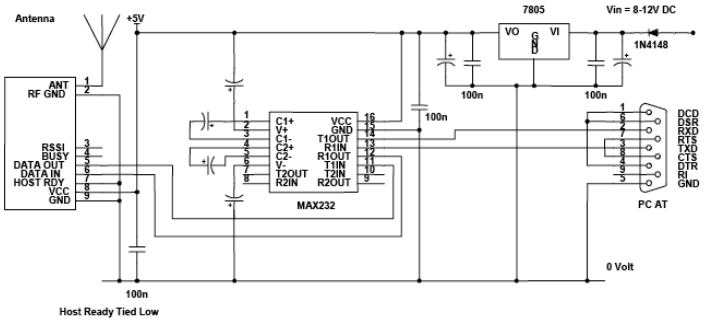


Abbildung 4.21: Schaltplan Funkmodul PC

sie nur eine Zwischenlösung darstellte, auf einer Lochrasterplatine implementiert.
Die verwendete 40Mhz Fernbedienung wurde, um alle Steuerungsfunktion mit Hilfe der Host-Software realisieren zu können, zunächst deaktiviert. Es wurden alle Steuersignale als Datenpaket über das 433Mhz Modul gesendet. Da eine Unterwasserkommunikation innerhalb dieses Frequenzbands nicht möglich ist, wurde eine 27 Mhz Funkverbindung von der Host-Software zur Tauchplattform implementiert.
Für dieses Frequenzband existieren keine Bausteine, die eine Datenkommunikation auf direktem Wege erlauben. Daher verwendeten wir eine handelsübliche Funkfernbedienung als Sender, die mit Hilfe von D/A Wandlern und einem weiteren FPGA an einen PC angeschlossen wurde. Zu diesem Zeitpunkt wurde auch die 433Mhz Kommunikation an dieses FPGA angeschlossen, so dass man über das FPGA Informationen über beide Frequenzbänder an die Tauchplattform senden kann. Der Empfänger für die 27Mhz Kommunikation konnte innerhalb der Tauchplattform wieder sehr einfach implementiert werden, da er, genau wie bei der 40Mhz Kommunikation, PWM-Signale erzeugt, die direkt durch das FPGA ausgewertet werden können.

4.4.11 Festspeicheranbindung

Um während der Fahrt Daten dauerhaft speichern zu können, musste eine externe Speicherlösung implementiert werden, da das FPGA keinen zur Laufzeit beschreibbaren nichtflüchtigen Speicher enthält. Zunächst wurde dies mit Hilfe eines Interfaces realisiert, welches FAT formatierte CompactFlash-Karten mit Hilfe von AT-Befehlen, ähnlich wie bei einem Modem, auf einem seriellen Bus ansteuern kann. Diese Anbindung hatte den Vorteil, dass sie nur wenige Anschlüsse auf dem FPGA benötigte und leicht zu implementieren war. Eine direkte Anbindung einer CompacFlash-Karte an das FPGA würde ein Vielfaches der nötigen Pins erfordern. Leider ist diese Art der Anbindung nicht sehr schnell, war allerdings für die Speicherungen von Status- und Log-Dateien ausreichend.
Da die Speicheranbindung des CompactFlash-Moduls nicht schnell genug zur Speicherung von Bildern aus einer Kamera war, diese jedoch eingebaut werden sollte, musste eine schnellere Speicherlösung gefunden werden. Da SD-Karten das SPI Protokoll beherrschen, welches wir bereits verwendet hatten und für das bereits Steckverbindungen auf der neuen Tochterplatine vorhanden waren, implementierten wir einen

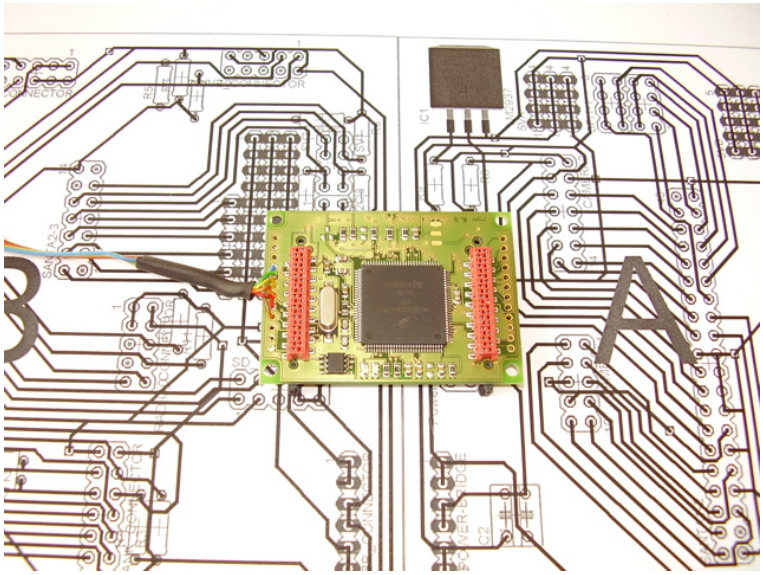


Abbildung 4.22: CompactFlash-Modul

Kartenslot für dieses Format. Dieser ließ sich direkt an unsere Tochterplatine anschließen.

4.4.12 Kameranbindung

Um Bilder während eines Tauchgangs aufzeichnen zu können, musste eine digitale Kamera an das FPGA angeschlossen werden. Da das zuerst verwendete FPGA nicht mehr genug Resourcen für eine Implementierung zur Verfügung stellte, mussten das FPGA-Board und die Tochterplatine ausgetauscht werden.
Die Kamera stellt einen 16Bit breiten Bus zur Verfügung, welcher zeitgleich die Farb- und Helligkeitswerte eines Bildpunktes überträgt. Darüber hinaus werden die Signale HREF, VSYNC und PCLOCK verwendet, um ein Bild zeilenweise zu übertragen.
Durch die Länge der verwendeten Flachbandleitung waren die Signale aufgrund externer Einstreuungen vom FPGA teilweise nicht mehr eindeutig auswertbar. Leider konnten die Signale nicht durch eine Filterschaltung bereinigt werden, da das Timingverhalten der einzelnen Signale zueinander eine große Rolle beim Bildaufbau spielt und dieses durch die Filterung zu stark beeinflusst wurde. Daher haben wir von analogen Filtergliedern abgesehen und die Filterung der Signale innerhalb des FPGA vorgenommen.

4.4.13 Sonar

Zur Kartographierung und zur Ortung von Hindernissen sollte ein Sonarsystem implementiert werden. Leider war eine Implementierung innerhalb der Projektzeit nicht mehr möglich.

Kapitel 5

Regelung

Motivation

Unser U-boot soll in der Lage sein autonom an der Oberfläche so wie im getauchten Zustand navigieren zu können. Die Navigation ermöglicht uns innerhalb des Raumes Positionen zu bestimmt und eine Route zu einem gewünschten Punkt zu ermitteln. Allerdings benötigen wir eine sichere und genaue Steuerung des Bootes, auch ohne Eingriff des Benutzers oder des Hostsystems, um eine solche Route dann tatsächlich ansteuern zu können. Eine Regelung die auf dem Nios II FPGA-Board implementiert ist und die verschiedenen Größen überwacht, kann dies sicherstellen. Die Regelung ist damit das Bindeglied zwischen Sensorik und Aktorik, Informationen aus der Umwelt werden genutzt um Antrieb, Ruder und Tauchtanks sinnvoll anzusteuern.

5.1 Einleitung

In diesem Kapitel geht es um die Simulation eines U-Bootmodells in Matlab Simulink. Genauer werden die einzelnen zu regelden Systeme modelliert. Ein Modell dient zur Veranschaulichung und Test der Regelung.

Diese Modelle zu entwickeln, ist nicht immer trivial oder möglich, da teilweise nichtlineare Eigenschaften vorhanden sind oder die Erfassung einfach zu komplex wäre. Als Besipiel sei hier die thermischen Strömungen im Schwimmbad erwähnt.

Je präziser das Modell entwickelt wird, desto näher kommt es an die Realität heran. Es dient aber auch dazu, die physikalischen Gestztmäßigkeiten zu erforschen und das zu händelnde Objekt somit besser zu verstehen. So lassen sich nachher Anforderung an die Umwelterfassung, mit Hilfe verschiedenster Sensoren, leichter formulieren.

5.2 Konzeptfindung

5.2.1 Einfaches Modell des Bootes

Um die verschiedenen Größen und Effekte zu diskutieren die auf das Boot wirken wird zuerst eine Definition des Modells durchgeführt. Ein Koordinatensystem wird in das U-Boot gelegt die x-Achse zeigt hierbei in positiver Richtung nach vorn, die y-Achse nach links und die z-Achse nach unten. Die Kräfte die auf das Boot in x,y,z-Richtung wirken sind hier jeweils F_x, F_y, F_z. Die Winkel die das Boot um die Jeweiligen Achsen einnimmt, bennenen wir α, β, γ, wobei α der Winkel um die x-Achse, β der Winkel um die y-Achse und γ der Winkel um die z-Achse ist. Der Winkel α wird auch als Krängung, der Winkel β als Lagewinkel und der Winkel γ als Kurswinkel des Bootes bezeichnet. Auf das Boot wirken Drehmomente welche benannt werden M_x, M_y, M_z entsprechend den Winkel und Achsen um die sie wirken. Damit ergibt sich folgende Sizze des Bootes mit Koordinatensystem.

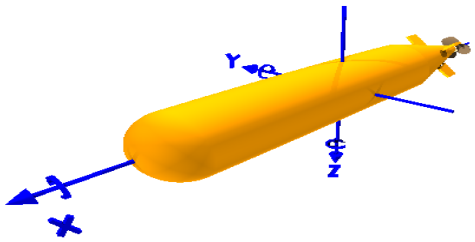


Abbildung 5.1: U-Bootkoordinatensystem

5.3 Einführung in die Regelungstechnik

Nach DIN 19226 ist eine Regelung definiert als:
Das Regeln, die Regelung, ist ein Vorgang, bei dem eine Größe, die zu regelnde Größe (Regelgröße),fortlaufend erfasst, mit einer anderen Größe, der Führungsgröße, verglichen und im Sinne einer Angleichung der Führungsgröße beeinflusst wird.
Abb. 5.2 wird das Funktionsprinzip eines Regelkreises im allg. Fall verdeutlicht. Als Regler werden folgende Standardregler verwendet:

- P-Glied:
  Hierbei handelt es sich um ein verzögerungsfreies statisches Übertragungsglied. Das Verhalten lässt

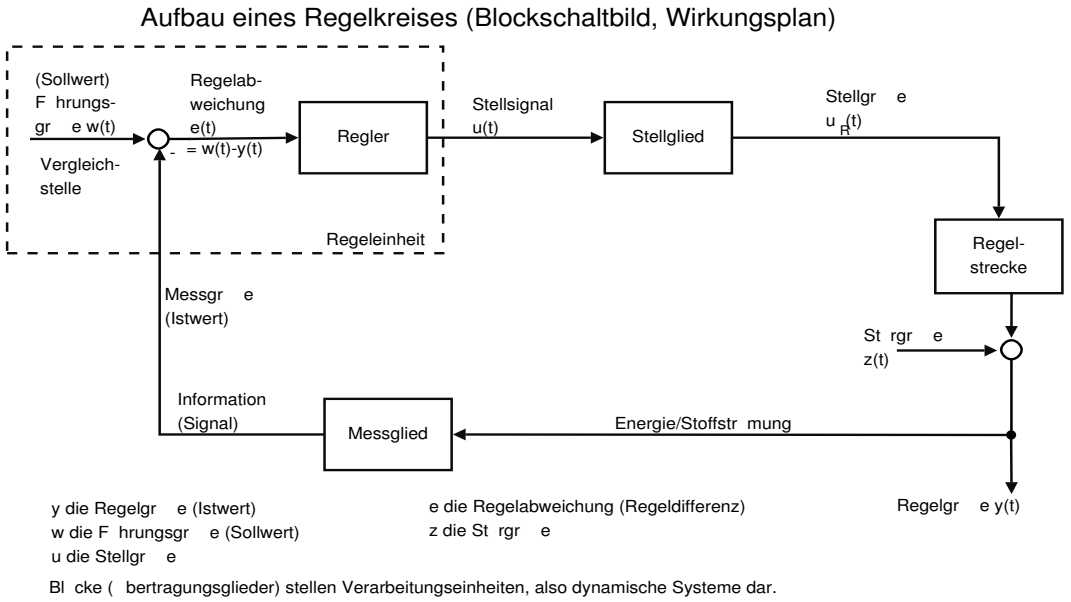


Abbildung 5.2: Allg. Blockschaltbild eines Regelkreises

sich durch folgende Abb. beschreiben.

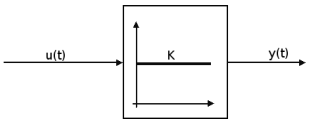


Abbildung 5.3: Proportionalglied

$$x_a(t) = K \cdot x_e(t)$$

K ist der Übertragungsbeiwert oder Verstärtkungsfaktor, welcher positiv als auch negativ sein kann. Somit wird die Regelgröße durch $y = k \cdot u$ verändert.

- I-Glied:
  Hierbei handelt es sich um ein dynamisches Übertragungsglied. Der integralwirkende Regler summiert die Regelabweichung über der Zeit auf und multipliziert die Summe (d.h. das Integral) mit dem Faktor T_i.

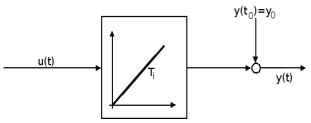


Abbildung 5.4: Integralglied

$$x_a = \frac{1}{T_i} \cdot \int_t^0 x_e(x)\,\mathrm{d}x + x_a(0)$$

Je länger eine Regelabweichung ansteht, desto größer wird die Stellgröße des I-Reglers.

- D-Glied: Hierbei handelt es sich um ein dynamisches Übertragungsglied. Die Eingangsgröße kontinuirtlich differenziert.

$$x_a(t) = T_D \cdot \frac{d}{dt} x_e(t)$$

D-Glieder sind Übertragungsglieder, deren Ausgang durch Verändrung der Eingangsgröße bestimmt wird. Bei konstanter Eingnagsgröße folgt eine Ausgangsgröße, welche gegen 0 strebt. Die Regelgrö-

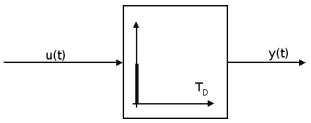


Abbildung 5.5: Differentialglied

ße y wird durch $y = T_D \cdot u$ verändert. Doch ist dieses Standardglied verzögerungsfrei physikalisch nicht realisierbar.

- Übersicht über Vor- und Nachteile der einzelnen Standardglieder

Reglertyp	Geschwindigkeit	Genauigkeit
P	schnell	ungenau
I	langsam	genau
PI	sehr schnell	ungenau

- PID-Regler: Der PID-Regler ist der Zusammenschluss der oben erklärten Standardglieder, wobei sich die Wirkung additive verhält.

$$x_a(t) = (K \cdot x_e(t)) + (\frac{1}{T_i} \cdot \int_t^0 x_e(x)) + (T_D \cdot \frac{d}{dt} x_e(t))$$

5.4 Was soll geregelt werden?

Um diese Frage beantworten zu können muss man sich frage, was das U-Boot eigentlich in seinen Grundfunktionen alles machen soll. Es soll selbstständig tauchen können. Was heißt das? Im Grunde genommen eigentlich tauchen, fahren und navigieren.

- Tauchen:
  Wir unterscheiden zwischen statischen und dynamischen Tauchen. Beim statischen Tauchen werden lediglich Gewichtsveränderung mit Hilfe der Tauchtanks benutzt um die Lage des U-Boot in z-Richtung zu verändern.
  Beim dynamischen Tauchen werden die Ruder verwendet. Durch den Motor wird ein Vortrieb erzeugt, wodurch die Ruder umströmt werden. Durch richtiges Ausrichten der Ruder wird eine Kraft erzeugt, die dafür sorgt, dass das U-Boot seine Position in z-Richtung verändert.

- Fahren:
  Hiermit ist im Grunde genommen die Antriebsregelung gemeingt, d. h. das richtige beschleunigen auf eine vorgegebene Geschwindigkeit.

- Navigieren:
  Selbstverständlich will man auch in y-Richtung navigieren können, sprich eine Richtungänderung vornehmen. Dieses geschieht auch mit Hilfe der Ruder.

Somit müssen die beiden Tauchtanks, der Hauptantrieb in Form eines Bosch Elektromotors und die vier Ruder, angetrieben durch jweils einem Servomotor, geregelt werden.

5.5 Funktionsprinzip der X-Ruder

5.5.1 Höhenruder

Bei der x-Ruderanordnung sind die senkrecht gegenüberliegenden Achsen mechanisch oder elektronisch miteinander verbunden, in unserem Fall elektronisch. Dadurch schlagen die Ruder oben rechts und oben links bzw. unten links und unten rechts in die entgegengesetzte Richtung aus. Im Gegensatz zum normalen +-Ruder braucht man für eine Richtungsänderung in einer Ebene (Seiten oder Tiefenruder) aber immer beide Ruderpaare. Will man z.B. nach untern fahren, muss man das eine Ruderpaar nach unten links und das andere nach unten rechts auslenken. Die Summe aller wirkenden Kräfte ergibt dann die Resultierende, die das Heck nach oben drückt und somit den Bug nach unten zeigen lässt.
Anhand der hier aufgezeigten Theorie wird einem auch schnell klar, warum das x-Ruder bei gleicher Ruderfläche und Seitenrichtungswechsel dem herkömmlichen +-Ruder überlegen ist. Wenn man das +-Ruder betrachtet, sind bei vollem Seitenruderausschlag zwei Ruder im Eingriff, also 2x100% Ruderwirkung. Beim x-Ruder zählt nur die Kraftkomponente in der waagerechten, die anderen Kräfte heben sich gegenseitig auf. Deshalb hat jedes Ruder nur einen wirksamen Anteil von 1/Wurzel=71%. Aber 4x71% beim x-Ruder stehen 2x100% beim +-Ruder gegenüber, die x-Anordnung hat also bei gleicher Ruderfläche eine um 41% bessere Wirksamkeit.

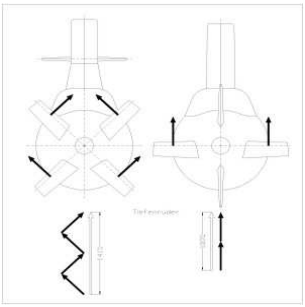


Abbildung 5.6: Funktionsweise des Höhenruder

5.5.2 Seitenruder

Bei der x-Ruderanordnung sind die diagonal gegenüberliegenden Achsen mechanisch oder elektronisch miteinander verbunden. Dadurch schlagen die Ruder oben rechts und unten links bzw. oben links und unten rechts in die gleiche Richtung aus. Im Gegensatz zum normalen +-Ruder braucht man für eine Richtungsänderung in einer Ebene (Seiten oder Tiefenruder) aber immer beide Ruderpaare. Wenn man nun den Kurs nach Backbord wechseln will, zeigt das eine Ruderpaar nach unten links und das andere nach oben links. Da nun durch die Anströmung der Ruder eine Kraft auf die Ruderachsen wirkt, kann man anhand der Kräftesumme schön die Resultierende ermitteln, die nur nach rechts zeigt und damit das Boot nach Backbord dreht.
Anhand der hier aufgezeigten Theorie wird einem auch schnell klar, warum das x-Ruder bei gleicher Ruderfläche und Seitenrichtungswechsel dem herkömmlichen +-Ruder überlegen ist. Wenn man das +-Ruder betrachtet, sind bei vollem Seitenruderausschlag zwei Ruder im Eingriff, also 2x100% Ruderwirkung. Beim x-Ruder zählt nur die Kraftkomponente in der waagerechten, die anderen Kräfte heben sich gegenseitig auf. Deshalb hat jedes Ruder nur einen wirksamen Anteil von 1/Wurzel=71%. Aber 4x71% beim x-Ruder stehen 2x100% beim +-Ruder gegenüber, die x-Anordnung hat also bei gleicher Ruderfläche eine um 41% bessere Wirksamkeit.

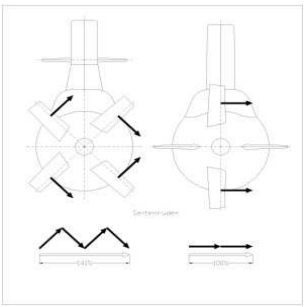


Abbildung 5.7: Funktionsweise des Seitenruder

5.6 Kaskadierte Regelung

Um eine Regelung abstrakt darzustellen, benutzt man in der Regel Blockdiagramme. Eine allgemeine Darstellung kann man in Abb. 5.2 betrachten.

Wie man dem Blockschaltbild entnehmen kann, benötigt man zur Ermittlung der Regelabweichung den momentanen Ist-Wert, der mit Hilfe von Sensoren ermittelt wird. Die für die Regelung notwendigen Sensoren sind:

- Beschleunigungssensor für die Beschleunigung in x-Richtung
- Gyroskope für die Dehung um die x-, y-, und z-Achse. Für jede Achse ist ein Gyroskop vorhanden.
- Drucksensor, der den Wasserdruck ermittelt und man so die Ist-Tiefe ermitteln kann.

Genauere Beschreibungen über die Funktionsweisen der entsprechenden Sensoren finden Sie Teil über Sensorik des U-Bootes.
Um eine Änderung des Zustands unsere U-Bootes zu bewirken, stehen uns folgende Aktoren zur Verfügung:

- Hauptantriebsmotor (Eletromotor) mit welchem wir eine, über eine Welle verbundene, Schraube rotieren lassen. Diese Rotation bewirkt eine Wasserverdrängung. Diese entsteht aufgrund der Form und der Ausrictung der Propellerflügel, wodurch die Schraube schräg oder asymmetrisch umströmt wird, was einen Vor- oder Rüchschub, je nach Rotationrichtung, zur Folge hat.

- Über Servomotoren können wir bewegliche Ruder ausrichten. Wärend der Fahrt durch das Wasser werden die Ruder von Wasser umströmt. Durch eine Ausrichtung kommt es nun zu einer Zunahme des Strömungswiderstandes und somit zu einer Richtungs-/ Zustandsänderung. Im oberen Teil LiteraturFunktionsprinzip der X-Ruder wird das Ansteuerungsprinzip verdeutlicht, mit welchem man eine gezielte Richtungsänderung vornehmen kann. Somit ist eine Kursänderung (Rotation um die z-Achse) und eine Tiefenänderung (Rotation um die y-Achse) möglich, welche die dynamische Zusatandsänderung unseres U-Bootsystems repräsentieren.

- Statische Zustandsänderungen lassen sich über unsere Tauchtanks hervorrufen. Ein Tauchtank ist ein Kolbensystem, welches über das Verschieben eins Stempels im Inneren einen Unterdruck oder einen Überdruck erzeugt. Das Verschieben des Stempels geschieht über eine Gewindestange, welche durch eine rotierende Mutter bewegt wird. Über einen angeschlossenen Schlauch ist ein Druckausgleich mit der Außenwelt möglich. Dieser sieht bei einem Unterdruck das Fluten des Innenraums mit Wasser vor und bei einem Überdruck das Lenzen des Kolbens. Dadurch lässt sich gezielt die Masse des U-Bootes variieren. Im Schwebenzustand sind die Tauchtanks zu 2/3 mit Wasser gefüllt. Durch weiteres Fluten kann das U-Boot weiter absinken; durch Lenzen wieder auftauchen.

Wenden wir doch diese Art der Darstellung auf die oben beschriegenen Bereiche an.

- Tauchtanks:
  Wie man der Abb. 5.8 entnehmen kann enthält die Tauchtankregelung ein PID-Regler. Der Elektromotor wird immer mit voller oder keiner Spannung angesteuert, quasi an- oder ausgeschaltet. Ein Inkrementalgeber ist an der Motorwelle befestigt mit welchem man die Anzahl der Umdrehungen der Motorwelle auslesen kann. Die Navigation übergibt der Regelung einen Solltiefenwert. Der aktuelle Tiefenwert wird mittels eines Drucksensors ermittelt.

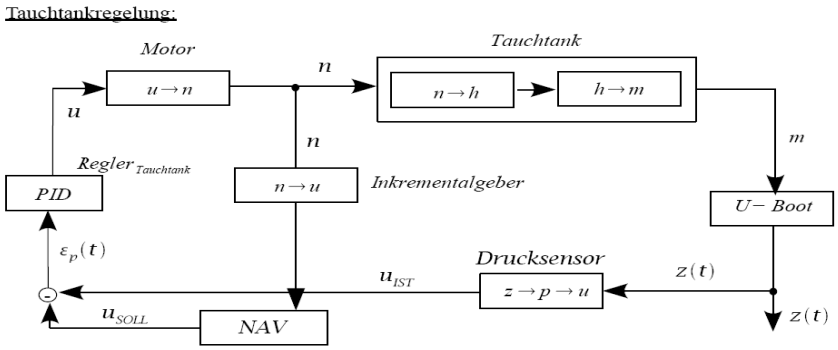


Abbildung 5.8: Tauchtankregelung

- X-Ruder:
  Bei der Ansteuerung der Ruder existiert jeweils ein PID-Regler. Einer ist zuständig für die Tiefe-/Lageänderung und der andere für die Kursänderung. Wie in der Abb. 5.9 gezeigt werden die Signale dann an die einzelnen Ruder übertragen. Das Prinzip der Ruderansteuerung kann Abschnitt LiteraturFunktionsprinzip der X-Ruder entnommen werden. Die Gesamtruderwirkung ergibt dann die gewollte Zustandsänderung.
  In der aktuellen Version existiert nur ein P-Regler der auf die Eingaben der Funkfernbedienung reagiert. Der Grund dafür liegt in den noch nicht ausreichend genau funktionierenden Sensoren.

- Antrieb:
  Ganz wichtig ist die Antiebstregelung. Sie gewährleistet das Erreichen bzw. Halten einer vorgegebenen Geschwindigkeit. Insbesondere das richtige Abbremsen in Notfallsituationen ist eine äußerst

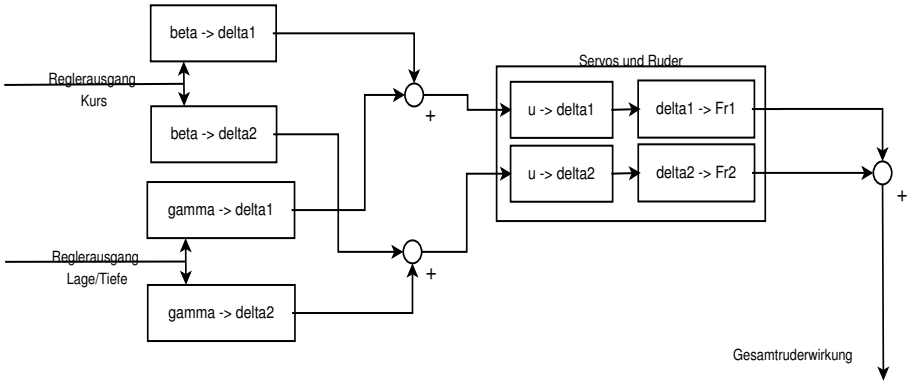


Abbildung 5.9: Regelkreis für X-Ruder

kritische Anforderung. Zudem ist die Ruderwirkung immer quadratisch proportional zur Geschwindigkeit, was für die Lage-/ und Tiefenregelung sehr entscheidend ist.
Zum momentanen Zeitpunkt ist eine solche Geschwindikeitsregelung noch nicht in Software implementiert. Über eine Fernbedienung werden entsprechende Signale dem Motor zugeführt. Grund dafür sind die noch nicht exakt arbeitenden Sensoren.

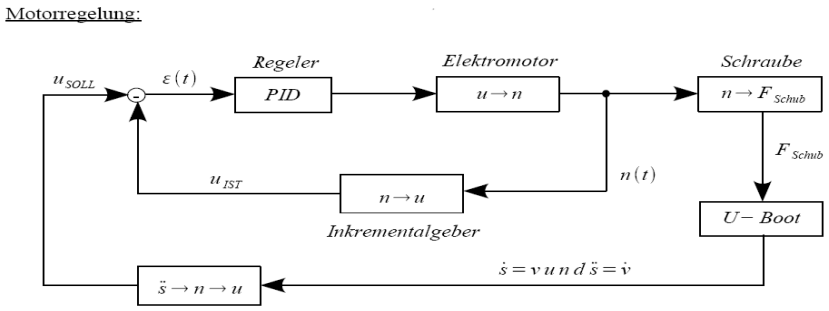


Abbildung 5.10: Motorregelkreis

Vereinigt ergibt sich folgende kaskadierte Regelung:

5.7 Physikalische Gesetzmäßigkeiten der Umwelt

Um die physikalischen Gesetzmäßigkeiten zu vedeutlich, müssen wir erst einmal nach auftretende Kräfte suchen.

Die Kraft im physikalischen Sinne ist die Fähigkeit etwas zu bewirken. Kraft hat die Fähigkeit die Bewegung eines Körpers zu ändern (Richtungsänderung oder Beschleunigung) oder einen Körper zu verformen. Eine Kraft ist immer eine gerichtete vektorielle Größe, die erst durch die Angabe von Zahlenwert, Einheit und Richtung festgelegt wird. Das Formelzeichen für die phsikalische Kraft ist F. Um den vektoriellen Charakter im Formelzeichen zu verdeutlichen wird über dem F ein Pfeil angefügt ($\vec{F}$). Aber wo treten bei dem U-Boot Kräfte auf? Die folgenden Abb. 5.12 sollen dies ein wenig verdeutlichen. Abb 5.12 verdeutlicht, dass man es schon bei einer Lageänderung mit Hebelkräften, Reibungskräften und Trägheitskäften zutun bekommt. Diese entstehen als Folge der gerichteten Kräfte der Ruder, die angesteuert vom U-Boot eine Richtungsänderung bewirken sollen. Abb. 5.13 verdeutlicht die Kräfte bei einer Beschleuni-

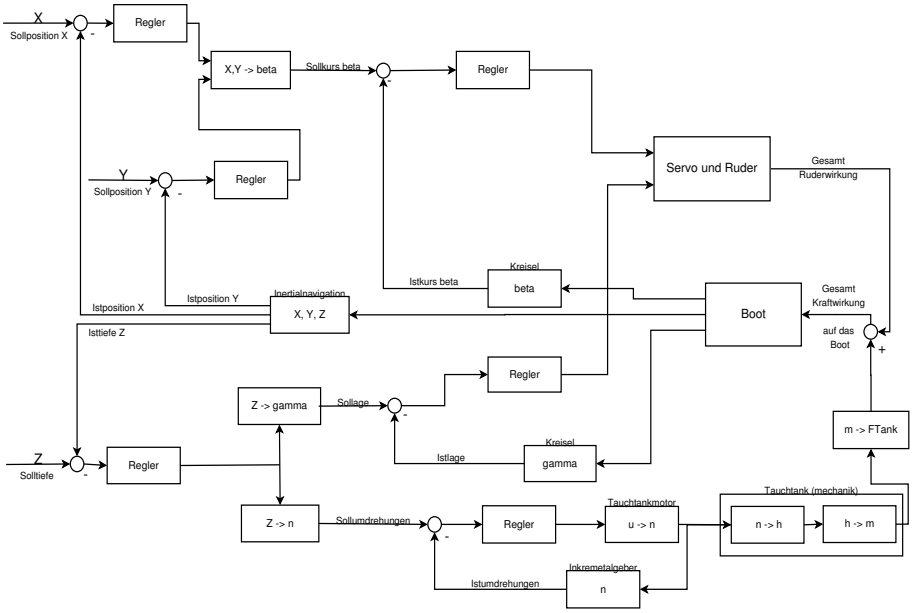


Abbildung 5.11: Kaskadierter Regelkreises

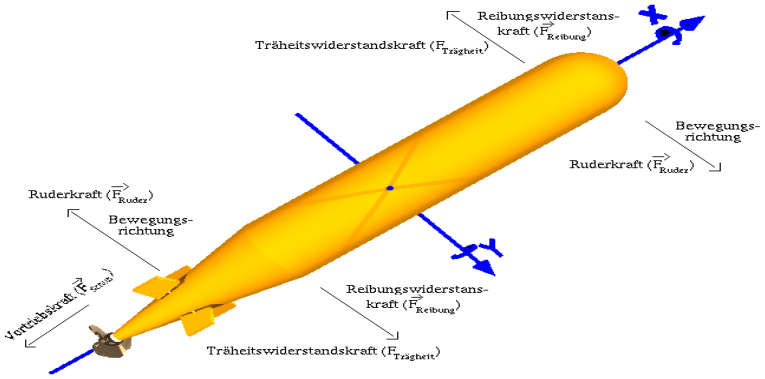


Abbildung 5.12: Kräfte bei einer Kursveränderung

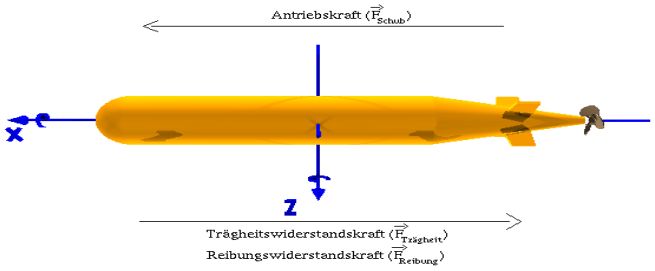


Abbildung 5.13: Kräfte bei der Beschleunigung 3D

gung in x-Richtung durch den Hauptantiebsmotor. Diese geichtete Beschleunigung hat die Reibungskräfte und die Trägheitskraft zur Folge.

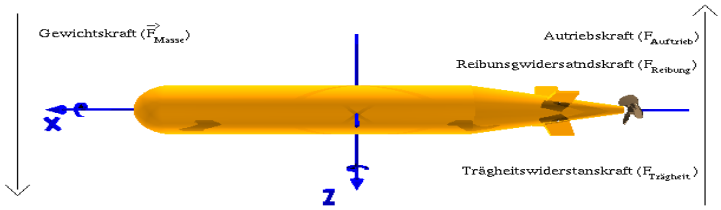


Abbildung 5.14: Kräfte beim statischen Tauchen

5.7.1 Reibungswiderstandskraft / Strömungswiderstandskraft

Zwischen der U-Bootoberfläche und dem Medium Wasser entsteht bei einer Bewegung eine Reibung. Man kann im folgendem zwischen einer laminaren und einer turbulenten Strömung unterscheiden:

Laminare Strömung

Laut Stockschen Gesetzt gilt für kugelförmige Körper:
Formel:

$$F_{Reibung_{laminar}} = 6 \cdot \pi \cdot \eta v \cdot r$$

$\eta =$ Viskosität des Mediums; die bei Wasser 1 beträgt
$v =$ Geschwindigkeit
$r =$ Radius des U-Boots

Turbulente Strömung

Formel:

$$F_{Reibung_{turbulent}} = \frac{1}{2} \cdot \rho \cdot c_W \cdot A \cdot v^2$$

$c_W =$ Widerstandbeiwert
$\rho =$ Dichte des Mediums (in unserem Fall 1000)
$A =$ Fläche senkrecht zum Widerstandskraft (Querschnittfläche/Projektionsfläche)
$v =$ Anströhmgeschwindigkeit

Da bei einer Bewegung in x-Richtung größtenteils der vordere Rumpf dem Wasser einen Widerstand entgegensetzt, verwenden wir bei der Reibunswiderstandskraft für die Antriebsregelung die Formel zur laminaren Strömung.

Hingegen bei der Bewegung um eine Achse als Reaktion auf eine Ruderaktion verwenden wir die Forlmel zur turbulenten Strömung. [Wikd]

5.7.2 Trägheitskraft

Isaac Newton postulierte 1687 in seinem ersten Axiom: Ëin Körper verharrt in seinem Zustand der Ruhe oder der gleichförmigen geradlinigen Bewegung, solange die Summe aller auf ihn einwirkenden Kräfte

Null ist.¨
Aufgrund der Trägheitskraft verharren Körper in ihrem Bewegungszustand. Dieser Zustand hält auch solange an, solange keine äußere Kraft auf sie einwirkt. Quatifiziert wird die Trägheit durch seine Masse. Eine einwirkende Kraft hat proportional zur Masse des Körpers Einfluss auf diesen. Hat also ein Körper eine größe Masse, ist mehr äußere Kraft notwendig um Einfluss auf den Zustand des Körpers zu nehmen. In unserem Fall wirkt die Trägheitskraft immer entgegen einer vom U-Boot durch Aktoren ausgehenden Zustandsänderung. Soll das U-Boot auf eine Geschwindigkeit beschleunigt werden muss anfangs ein Kraft gößer als die Trägheitkraft aufgebracht werde um das U-Boot zu beschleunigen. Die Berechnung bzgl. der Lage- und Kursregelung haben wir mit Hilfe der Hauptträgheitsmomenter einfacher geometrischer Körper durchgeführt. Demnach gilt für einen Zylinder, er um eine Achse rotiert, die senkrecht zur Symmetrieachse liegt, dier Formel:

$$M_{Traegheit_{Zylinder}} = \frac{1}{2} \cdot m \cdot r^2 + \frac{1}{12} \cdot m \cdot l^2$$

$m =$ U-Bootmasse
$l =$ U-Bootlänge
[Wike] In diesem Zusammenhang haben wir das U-Boot als einen Vollzylinder angesehen und die geometrische Form des Bug und des Heck vernachlässigt.

5.7.3 Auftriebskraft

Unter dem Begriff der Auftriebskraft versteht man eine Kraft, die der Gewichtskraft eines Körpers entgegenwirkt. Man unterscheidet zwischen statischer und dynamischer Auftreibskraft, die im Folgenden kurz mal vorgestellt werden.

Statische Auftriebskraft

Formel zur Vedeutlichung des statischen Auftriebs:

$$F_{A_{statisch}} = \rho \cdot V \cdot g$$

Exakt müsste es heißen:

$$F_{A_{st}} = V_{Koerper} \cdot g \cdot (\rho_{aussen} - \rho_{innen})$$

$\rho =$ Dichte des Mediums (in unserem Fall 1000)
$V =$ verdrängtes Volumen
$g =$ Gewichtskraft
Als statischen Auftrieb bezeichnet man also den Einfluss auf die Gewichtskraft eines Körpers, der entsteht, wenn der Körper eine andere Dichte als das ihn umgebende Medium aufweist. Als bekantestes Beispiel zur Verdeutlichung des statischen Auftriebs ist ein Heißluftballon, dessen erwärmte Luft eine andere Dichte hat als die Umgebungsluft.

Dynamische Auftriebskraft

Als dynamischen Auftrieb bezeichnet man den Auftreib, der beim Umströmen von Körpern entsteht. Bekantestes Beispiel ist das Flugzeug. Hierbei wird die Tragfläche von Luft umströmt. Durch die speizielle Profilform, die auf der oberen Seite eine Wölbung aufweist, müssen die Lufteilchen auf der Oberseite eine größere Strecke in gleicher Zeit wie die Luftteilchen, die die Unterseiten umströmen, zurücklegen. Dadurch ensteht ein Unterdruck an der Oberseite und durch die Verlangsamung der Luftteilchen an der unteren Profilseite ein Überdruck an der Unterseite. Durch diese Druckunterschiede entsteht eine Auftriebskraft, welche auf das Profil bzw. die Tragfläche/den Körper wirkt. Formel zur Vedeutlichung des statischen Auftriebs:

$$F_{A_{dynamisch}} = \frac{1}{2} \cdot \rho \cdot c_A \cdot A \cdot v^2 \cdot S$$

c_A = Auftreibsbeiwert
ρ = Dichte des Mediums
A = Fläche senkrecht zum Widerstandskraft (Querschnittfläche/Projektionsfläche)
v = Anströhmgeschwindigkeit
S = Fläche

5.7.4 Gewichtskraft

Die Formel zur Bestimmung der Gewichtskraft kann man entnehmen, dass die Gewichtskraft sich aus dem Produkt aus Masse des Körper mal Gravitation. Die Masse des Körper ist abhängig von seine spezifischen Dichte. Sie kann man z.B. dem Periodensystem entnehmen

$$\vec{G} = m \cdot \vec{g}$$

m = Masse des Körpers
$\vec{g}$ = zum Erdmittelpunkt gerichtete Schwerebeschleunigung (Gravitation = 9,80665)

5.7.5 Hebelkraft

Der Hebel dient als mechanisches Kraftübertragungssystem. Zudem leigen Ursachen und Wirkung (Kraft und Last) in einer Ebene, nicht aber auf einer Linie. Der meist stabförmige, starre Körper ist in der Regel um eine Achse (Drehpunkt oder Hypomochlion) drehbar. Er befindet sich im Gleichgewicht, wenn die Summe der (Dreh-)Momente aller an ihm angreifenden Kräfte Null ist. Wirken Kräfte so auf den Körper, sodass eine Drehbewegung beschleunigt oder verzögert wird, so ist das (Dreh-)Moment immer ungleich null. Im ebenen Fall (alle Kräfte wirken in einer Ebene) sind alle Momentenvektoren rechtwinklig zur Ebene orientiert. Hierbei werden links- und rechtsdrehende (Dreh-)Momente unterschieden.
Somit dienen Hebel der Kraftübertragung und ermöglichen große Kraftwirkungen mit geringem Aufwand. (Je größer dieser Abstand, desto größer die Drehwirkung der angreifenden Kraft).

$$F_K \cdot l_K = F_L \cdot l_L$$

Wichtig ist, das nur die Kräfte, welche im Winkel von 90° den Hebel angreifen eine Wirkung erzielen. Ist der Winkel verschieden von 90°, so müssen die Kräfte in die einzelnen Komponenten zerlegt werden. Ausschleißlich die Komponente, die rechtwinklig vom jeweiligen Arm wegzeigt, geht in die Rechnung ein. In unserem Fall bewegt das U-Boot sich um den Gewichtsschwerpunkt. Da dieser nicht in der Mitte

Abbildung 5.15: Hebelwirkung

liegt sondern eher etwas weiter hintern ist (siehe Abb. 5.16), ist z.B. die Wirkung der vorderen Tauchtanks höher, sodass das U-Boot vorne schneller auf- und abtaucht.

5.7.6 Zentriefugalkraft

Die Zentrifugalkraft ist eine physikalische Kraft, die an einem Körper angreift, der sich auf einer kreisförmigen Bahn bewegt. Sie wirkt auf den Körper so, dass er von seiner Kreisbahn nach außen hin sich bewegt. Somit steht die Kraft senkrecht auf der Oberfläche des Körpser und ist nach außen weg vom Kreismittelpunkt bzw. zur Drehachse gerichtet. Sie wird auch Fliehkarft genannt und ist eine Trägheitskraft bzw.

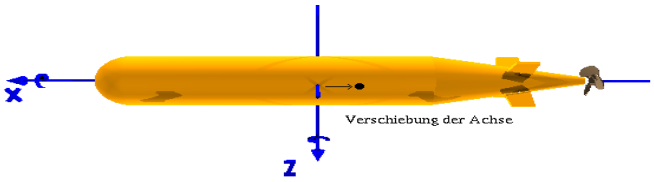


Abbildung 5.16: Gewichtsschwerpunkt bei unserem U-Boot

Scheinkraft.
Berechnung:
Für einen Körper der Masse m (in kg), der sich im Abstand r (in Meter) mit der Geschwindigkeit v (in Meter pro Sekunde) auf einer Kreisbahn bewegt, ist der Betrag der Zentrifugalkraft:

$$F_Z = \frac{m \cdot v^2}{r}$$

Als Pondon dazu existiert die Zentripetalkraft, welche in Betrag gleich der Zentrifugalkraft ist aber nach innen gerichtet.

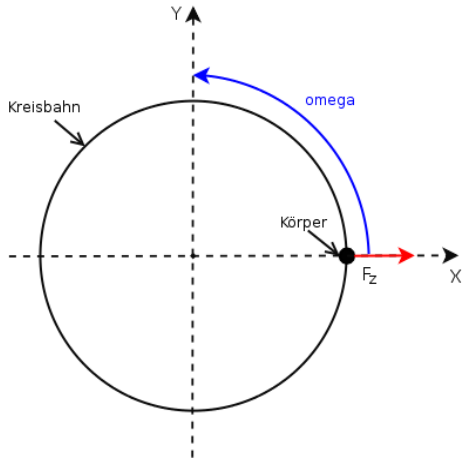


Abbildung 5.17: Vedeutlichung der Zentrifugalkraft

5.7.7 Corioliskraft

Bewegte Körper werden in einem rotierenden Bezugssystem aus Sicht eines mitrotierenden Beobachters abgelenkt. Diese Ablenkung wird der Corioliskraft zugeschrieben, die nach dem französischen Physiker Gaspard Gustave de Coriolis benannt ist.

Die Corioliskraft ist eine so genannte Scheinkraft, da sie im ruhenden Bezugssystem nicht vorhanden ist. Dort sind alle kräftefreien Bewegungen geradlinig. Die Corioliskraft tritt nur in rotierenden Bezugssystemen auf. Sie stellt eine Beschleunigung senkrecht zur Bewegungsrichtung dar, die dazu führt, dass kräftefreie Bewegungen vom rotierenden Bezugssystem aus gekrümmt erscheinen. Aus diesem Grund ist auch die Bezeichnung ‘Corioliskraftírreführend. Besser wäre der Begriff Ćoriolis-Effekt.

Die Corioliskraft tritt zusätzlich zur Zentrifugalkraft auf. Sie ist nur bei - im Bezug auf das rotierende Bezugssystem - bewegten Körpern vorhanden. Während die Zentrifugalkraft nur vom Ort Ihres Messkörpers abhängig ist, hängt die Corioliskraft zusätzlich von der Geschwindigkeit des Meßkörpers relativ zum rotierenden Bezugssystem ab.
Veranschaulichung an einem Bsp.:
Eine Kugel rollt auf einer Schiene in x-Richtung. Relativ zu einem äußeren Beobachter rotiert außerdem eine Scheibe mit der Winkelgeschwindigkeit $\vec{\omega}$ Für einen äußeren Beobachter bewegt sich die Kugel ge-

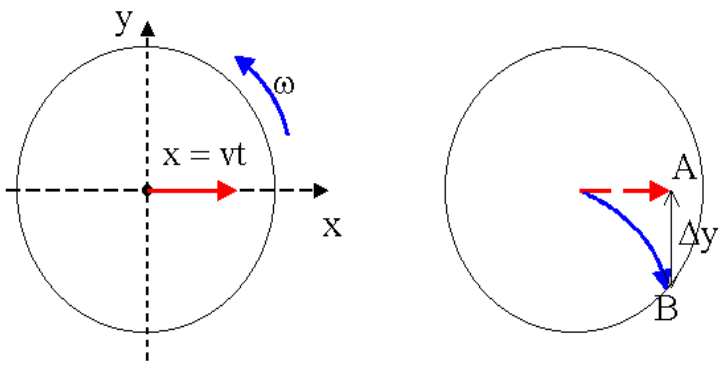


Abbildung 5.18: Corioliskraft

radlinig und gleichförmig nach außen. Ein Beobachter auf der rotierenden Scheibe sieht die Kugel auf sich zurollen, wobei sie sich aber während der Flugzeit um den Winkel $\alpha = \omega \cdot t$ dreht. Dieses stell für den bewegten Beobachter eine Ablenkung nach rechts (bzw. unten) dar. [Ver06] [Wika]

5.8 Physikalische Eigenschaften des U-Boots

In der folgenden Tabelle sind die physikalischen Eigenschaften und die physikalischen Gesetzmäßigkeiten, die das U-Boot ausgesetzt sind, aufgeführt:

Eigenschaften / Gesetzmäßigkeiten:	Wert:
Gravitation (g)	9.81
Dichte des Wasser (ρ)	1000
cw-Wert des Rumpfes in x-Richtung:	0.7
cw-Wert des Rumpfes in z-Richtung:	0.8
U-Boot Volumen (V):	0.0382 m^3
U-Boot Masse (m):	38.23 kg
U-Boot Radius (r):	0.1 m
U-Boot Länge (l):	1.45 m
U-Boot Ruderfläche (F):	0.0204 m^2
U-Boot Rumpfquerschnitt:	0.31 m^2
U-Bootquerschnitt:	0.03 m^2
U-Boot Metazentrische Höhe:	0.02 m
U-Boot Wellenradius:	0.01 m
U-Boot Wellenmasse:	0.2 kg
U-Boot Motorradius:	0.1 m
U-Boot Motormasse:	1.0 kg
U-Boot Rotorfläche der Schraube:	0.0101 m^2
Schraubenrumpfradius:	0.02916 m
Schraubenrumpfmasse:	0.15 kg
Schraubenblattlänge:	0.04737 m
Schraubenblattbreite:	0.05369 m

Die Werte der Tabelle entsprechen dem Standard des SI-Einheitensystems.

5.9 Mathematisches Modell / Aufstellen der Differentialgleichungen

Die durch die o. a. physikalischen Gesetzte und die physikalischen Beschränkung unsere Aktoren entstehenden Differentialgleichungen, die eine mathematisches Modell bilden. Es werden also die einzelnen Kräfte und Momente, die besprochen wurde, in einen Zusammenhang gebracht, der unserem Boot entspricht. Im folgenden ergeben sich somit 6 Differentialgleichungen, jeweils eine Kraft in x,y,z Richtung sowie je ein Moment um die x,y,z-Achse.

$$F_x = F_{Antrieb} - F_W$$

$$F_y = F_{A_{dyn}} - F_W$$

$$F_z = F_{A_{dyn}} - F_W$$

$$M_x = M_{Antrieb} - M_W$$

$$M_y = M_{Ruder} - M_W$$

$$M_z = M_{Ruder} - M_W$$

Mit F_W sind hier die jeweiligen Widerstände gemeint. Speiziell wird in den jeweiligen Regelung auf die entsprechenden Kräfte und Momente eingegangen.

Dieses erstellen wir mit Matlab Simulink. Es werden somit folgende Regelkreise simuliert:

- Statisches Tauchen
- Dynamisches Tauchen

- Antieb
- Kursveräderung
- Trimmung

5.10 Simulation in Matlab/Simulink

Nachdem die Differentialgleichungen für das Uboot Modell bekannt sind, ist der n?hste Schritt das Aufstellen einer Simulation. Hierbei werden fr die verschiedenen Anwendungen erst Einzelsimulationen erstellt und diese dann in einer Gesamtsimulation zusammengefgt. Mittels dieser Simulation kann zum einen das erarbeitete Differentialgleichungsmodell getestet und zum anderen gnstige Reglerwerte ermittelt werden. Als Umgebung für dieses Modell wurde Matlab/Simulink gewählt, da es ein einfaches erstellen und testen ermöglicht.

5.10.1 Kurswinkel- und Lagewinkelregelung

Jeweils für die Kurswinkel- als auch für die Lagewinkelregelung gelten folgende Gesetzmäßigkeiten, wobei sie bei den jeweiligen Regelungen in verschiedene Richtungen wirken.

$$M_{Reibungsmoment} = \int \frac{1}{2} \cdot \rho \cdot c_W \cdot A_{Rumpf} \cdot \omega$$

$$M_{Ruder_{dyn}} = \frac{1}{2} \cdot \rho \cdot c_W \cdot A_{Ruder} \cdot v^2 \cdot r_{Schwerpunkt}$$

$\rho =$ Dichte des Mediums (hier Wasser)
$c_W =$ dimensionsloser Widerstandsbeiwert
$\omega =$ Winkelbeschleunigung um die jeweilige Achse
$v =$ Geschwindigkeit der U-Boots in x-Richtung
$A_{Ruder} =$ Ruderfächer
$A_{Rumpf} =$ Fläche des U-Bootrumpfes
Nicht zu vergessen sind die Auftreibs- und die Gewichtskraft, die aber in dieser Situation gleich sind und somit weggelassen werden können.

Zur metazentrischen Höhe: Sie beschreibt die Schwimmlage eines Körpers, der sich in einer Flüssigkeit befindet. Für uns ist die Schwimmlage des U-Bootes im Wasser gemeint.
Formel:

$$h_m = \frac{I_y}{V_a} \cdot s$$

$I_y =$ Flächenträgheitsmoment um die Drehachse des U-Boots
$V_a =$ Volumen des U-Boots
$s =$ Abstand vom Körperschwerpunkt zum Auftriebskörperschwerpunkt (= positiv, wenn der Körperschwerpunkt oberhalb des Auftriebskörperschwerpunkts liegt, sonst negativ)
Bei den hier vorhandenen Flächenträgheitsmomenten handelt es sich um axiale Flächenträgheitsmomente. Sie lassen sich durch fogende Gleichungen beschreiben:

$$I_y = \int_A z^2 \; dA, Einheit : [m^4]$$

$$I_z = \int_A y^2 \; dA, Einheit : [m^4]$$

Die Schwimmlage wird als:

- stabil, wenn $h_m > 0$

- indifferent, wenn $h_m = 0$
- labil, wenn $h_m < 0$

[Wikc] [Wikb]

Kurswinkelregelung

Die Kurswinkelregelung, hierbei wird der Winkel γ um die z-Achse geregelt, stellt den Kurs dar, den das U-Boot fährt. Wie die restlichen Modelle auch besteht dieser Regelkreis hauptsächlich aus einem Aufsummierblock der verschiedenen auf das Boot wirkeneden Momente sowie einer mehrfachen Integration des resultierenden Moments. In diesem Fall sind dies das Reibungsmoment $M_{Reibung}$, das Drehmoment der Ruder $M_{Ruder_{dyn}}$ sowie dem Trägheitsoment (siehe 5.7.2). Zuerst zur Aufsummierung, hier werden die verschiedensten Momente jeweils mit Berücksichtigung der Wirkrichtung aufaddiert. Zum einen das Reibungsmoment welches durch eine Bewegung im Wasser am Rumpf entsteht und berechnet sich über die Läge des Rumpfes. Dieses wirkt der Bewegung des Bootes entgegen und fließt daher negativ ein.

Das Drehmoment der Ruder wird positiv addiert da es die Richtung der Drehbewegung im Normalfall vorgibt. Dieses läßt sich nach der Auftriebskraft an den Rudern und der Entfernung der Ruder zum Gewichtsschwerpunkt bestimmen.

Nachdem diese Größen aufaddiert wurden erhöht man als Ergebnis das Gesamtmoment das auf den Rumpf wirkt, dieses kann durch das Trägheitsmoment $M_{Traegheit}$ des Rumpfkörpers , welches definiert ist als

$$M_{Traegheit} = \frac{1}{4} \cdot m \cdot r^2 + \frac{1}{12} \cdot m \cdot l^2$$

geteilt werden. Macht man dies so erhöht man die Winkelbeschleunigung welche integriert die Winkelgeschwindigkeit und diese nochmals integriert den Kurswinkel liefert. Der so gewonnene Winkel stellt den Kurswinkel dar und kann als Rückführungsgröße genutzt werden um mittels eines PID-Reglers das Modell zu regeln.

Die Umsetztung lässt sich in Abb. 5.19 betrachten, in welcher man die oben geschilderten Stationen der Momente noch einmal nachverfolgen kann.

Eine wichtige Voraussetzung zur Berechnung des Rudermoments ist, dass sich die Geschwindigkeit des U-Boots sehr stark auf das resultierende Rudermoment auswirkt. Deshalb spielt die Geschwindigkeitsregelung schon in dieser Regelung eine große Rolle.

Lagewinkelregelung über die Ruder

Entsprechend der Kurswinkelregelung ist die Lagewinkelregelung aufgebaut. Es handelt sich um den Lagewinkel β um die y-Achse, wobei wieder durch die Ruder der Winkel beeinflusst werden kann. Aufgebaut ist diese Regelung wiederrum aus einer Aufsummierung von Momenten. Wir haben wieder ein Reibungsmoment $M_{Reibung}$, das auf den Rumpf wirkt. Zusätzlich wirkt allerdings noch das $M_{Traegheit}$ und selbstveständlich ist die Geschwindigkeitsregelung ein Teil des Regelmodells.

Die Vorgehensweise erklärt sich wie folgt: Durch die Ruder wird ein Moment erzeugt, welches das Heck um die Drehachse des U-Boots in eine gerichtete Richtung zwingt. Dagegen wirkt das Reibungsmoment. Durch teilen durch das Trägheitsmoment erhält man anschließend die Winkelbeschleunigung, welche integriert die Winkelgeschwindigkeit und nochmals integriert den aktuellen Winkel ergibt. Dieser wird mit dem Soll-Wert verglichen und der PID-Regler bestimmt anhand des Fehlers das neue Moment der Ruder. Dieses wind in Abb. 5.20 verdeutlicht.

Regelung des dynamisches Tauchen

Selbstverständlich kann man die Ruder auch dazu verwenden, um die Lage des U-Boots, sprich den Winkel β, die Drehung um die y-Achse, zu verändern. Dabei wird durch die Ausrichtungsänderung der Ruder (siehe dazu 5.5.1) ein Kraft induziert, die eine Drehung um die y-Achse zur Folge hat. Die Kraft kann aber nur dadurch entstehen, dass die Ruder von Wasser umstörmt werden, d.h. das U-Boot bewegt sich in

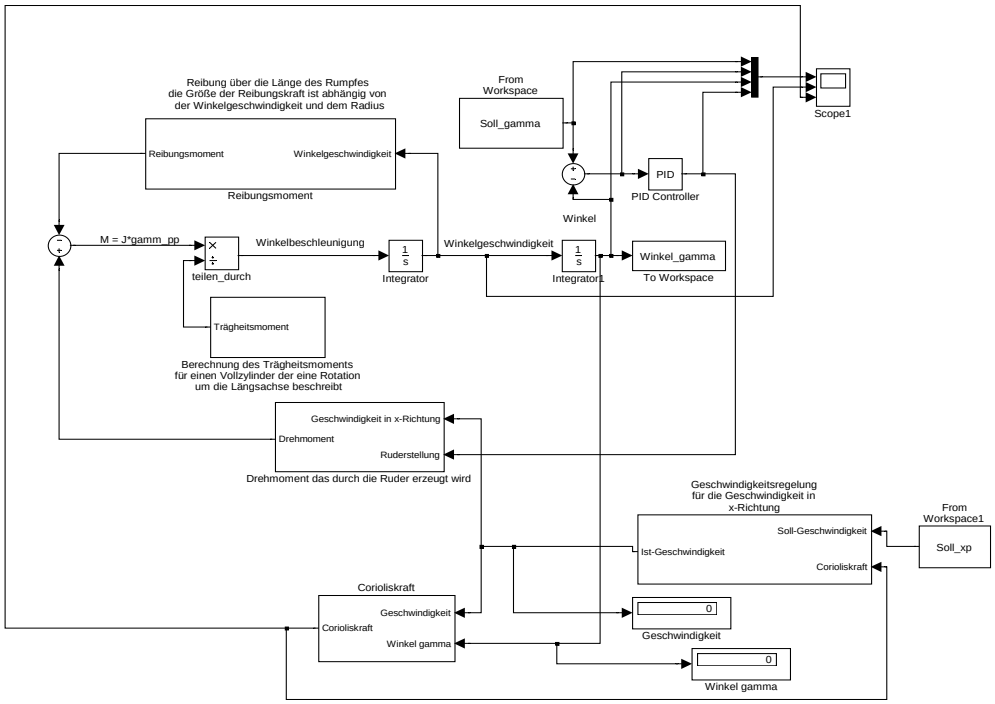


Abbildung 5.19: Kurswinkelregelung

x-Richtung. Sollten wir keine Bewegung des U-Boots in x-Richtung haben, wäre das Ruderprinzip nutzlos. Deshalb sprechen wir hier auch vom dynamischen Tauchen.

Aus diesem Grund, ist die Antiebsregelung (siehe: 5.10.3) ein Teil der dynamischen Tauchregelung.

Abb. 5.21 stellt den Regelkreis dar. Wie man gut erkennen kann, sind die Antriebsregelung, wie schon obern erwähnt, und die Lagewinkelregelung in dem Regelkreis zum dynamsichen Tauchen enthalten. Entscheidend ist nun die Berechnund der z-Position. Wir haben ja durch die Lagewinkelregelung nur einen Winkel und durch die Geschwindigkeitsregelung nur eine gerichtete Geschwindigkeit gegeben. Ziel ist es aber eine Regelung zu erstellen, der man eine Soll-Tiefe vorgibt und durch die Bestimmung der aktuellen Tiefe einen Vergleich mittels eines PID-Reglers durchführen kann.
Wie kann man nun die z-Position bestimmen? Geholfen hat uns die Veranschaulichung der Problems mittels eine Skizze 5.22.

Ein positiver Winkel β entspricht dem Abtauchen und ein negativer dem Auftauchen. Um, wie in der Skizze dargestellt, die Strecke c zu berechnen, integriere wir die Geschwindigkeit, die wir als Ausgang der Geschwindigkeitsregelung erhalten. Dieses entspricht der Position in x-Richtung bei verändertem Winkel β. Anschließend kann man mit der Formel:

$$a = sin(\alpha) \cdot c$$

die temporäre Tiefe berechnen. Abb. zeigt die Umsetztung in Matlab Simulink.

Hinter dem Subsystem zur Berechnung der z-Position werden die z-Positionveränderung in einem Speicher addiert. Leider ist es zur korrekten Berechnung notwenig, das bei jeder Geschwindigkeitsänderung der Integrator wieder auf null gesetzt wird, um die neue x-Position zu bestimmen. Diese ist aber in Matlab so nicht möglich. Man kann den Integrator neu starten lassen, doch kann man dieses nicht mit dem Geschwindigkeitswechsel synchronisieren. Deshalb funktioniert die Berechnung der z-Position nur bei konstanter Geschwindigkeit. Hier sind noch Erweiterungen nötig, um die realistische Berechnung der z-Position zu bestimmen.

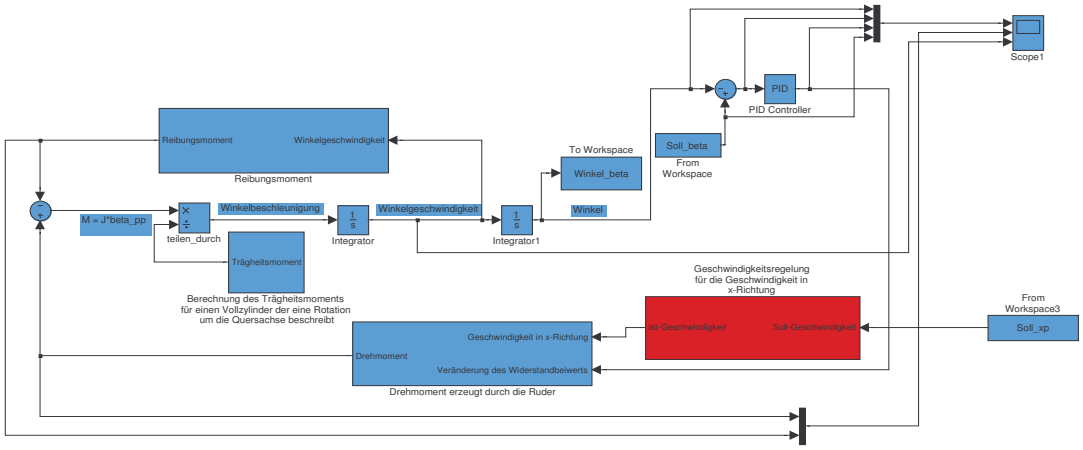


Abbildung 5.20: Lagewinkelregelung

5.10.2 Regelung des statischen Tauchens

Beim Modell der statischen Tiefenregelung, also der Tiefenveränderung, durch die Befüllung der Tauchtanks, handelt es sich um eine Aufsummierung von Kräften. Hier wird aus den verschiedenen wirkenden Kräfte die Gesamtkraft in z-Richtung berechnet. Im Modell wirken der Strömungswiderstand F_W, die statische Auftriebskraft $F_{A_{stat}}$ und die Gewichtskraft F_G, welche wie folgt definiert sind:

$$F_{A_{stat}} = V \cdot \rho \cdot g$$

$$F_G = m \cdot a = (m_{statisch} + \Delta m) \cdot a$$

$$F_W = \frac{1}{2} \cdot \rho \cdot c_W \cdot A_{Ruder} \cdot v^2$$

Ähnlich den Momenten in den Vorhergehenden Modellen werden die Kräfte hier addiert je nach Wirkrichtung, als Ergebnis erhalten wir die in z-Richtung wirkende Gesamtkraft des Bootes. Die statische Auftriebskraft und die Reibungwiderstandskraft wirken hierbei in negative Richtung und die Gewichtskraft in positive Richtung. Es werden die einzelnen Kräfte addiert und die resultierende Kraft F_z durch die aktuelle Masse des Bootes geteilt. Da $F = m \cdot a$ gilt, erhält man die Beschleunigung $\ddot{z}$ des Bootes in z-Richtung, diese Integriert ergibt die Geschwindigkeit $\dot{z}$ und diese wiederrum integriert ergibt die aktuelle z-Koordinate, also die Tiefe. Einer Beschleunigung ungleich 0 wirkt noch ein Trägheitsmoment entgegen:

$$M_{Traegheit} = \frac{1}{2} \cdot m \cdot r^2$$

Die Tiefe wird nun als Rückführungsgröße genutzt und mit Hilfe eines PID-Glieds zu einem Regelkreis vollendet. Dieses kann man in Abb. 5.24 sehr gut nachverfolgen.

Im Laufe der Entwicklung und der Tests im Schwimmbad erkannten wir, dass durch die hohe Trägheit im Wasser ein starkes Schwinngen um den Soll-Wert stattfand. Weitere Gründe waren das ständige

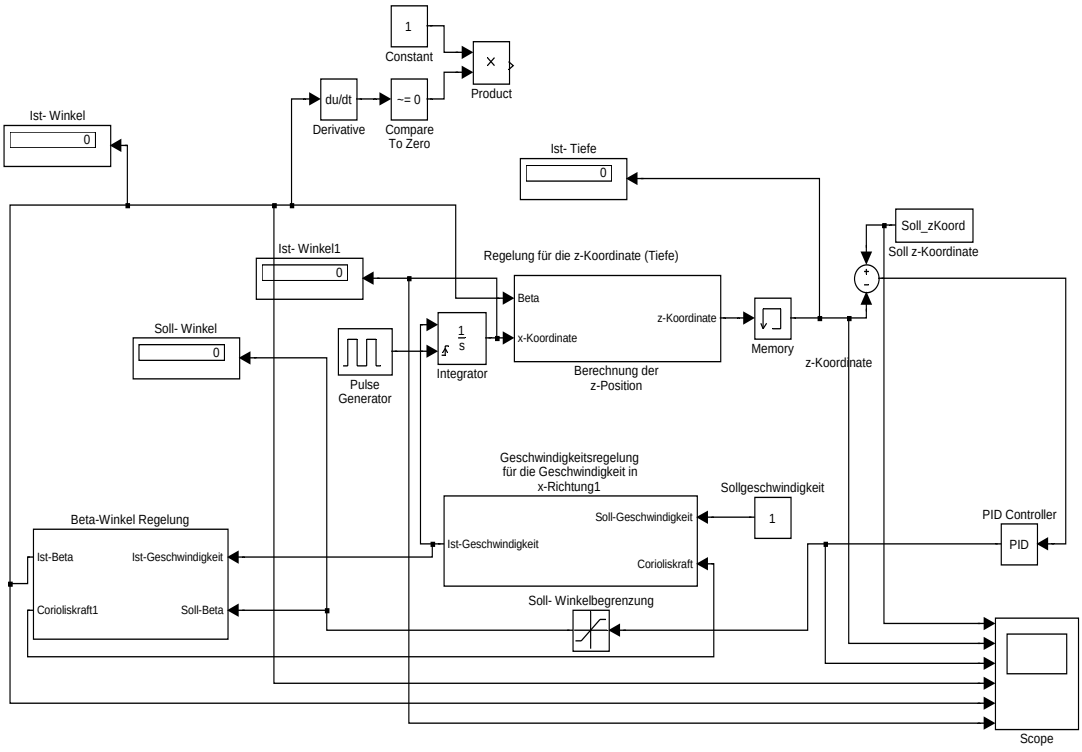


Abbildung 5.21: dynamische Tauchregelung

Abreisen der Funkverbindung und dem dadurch induzierten Auftauchen des U-Boots. Eine Lösung des Problems wäre eine Geschwindigkeitsregelung der Tauchtankmotoren, die beim Nähern des Soll-Wertes die Geschwindigkeit des Füllens der Tauchtanks verringern und somit das Erreichen des Soll-Wertes ermöglichen. Dieses wurde aber auf Grund anderer Probleme nach Hinten gestellt und nur bedingt getestet. Ein Versuch der Modellierung ist in unserem SVN zu finden. Es ist noch nicht vollständig, da sowohl keine Angaben zu den Motoren als auch keine Informationen zur Übersetzung vorhanden sind. Im Sinne einer Weiterentwicklung des statischen Tauchens mittels Tauchtanks wäre ein solches Modell noch in Betracht zu ziehen und dann zu vervollständigen.

5.10.3 Antriebsregelung

Wie man es in Abb. 5.13 sehen kann, benötigen wir bei der Simulation mit Matlab Simulink folgende Kräfte:

- Trägheitswiderstandskraft: (siehe 5.7.2)
  Die Trägheitswiderstandskraft tritt nur bei einer Beschleunigung des U-Boots auf.

$$F_{Traegheit} = \frac{1}{2} \cdot r^2 \cdot m$$

  r = Radius des U-Boots
  m = Masse des U-Boots

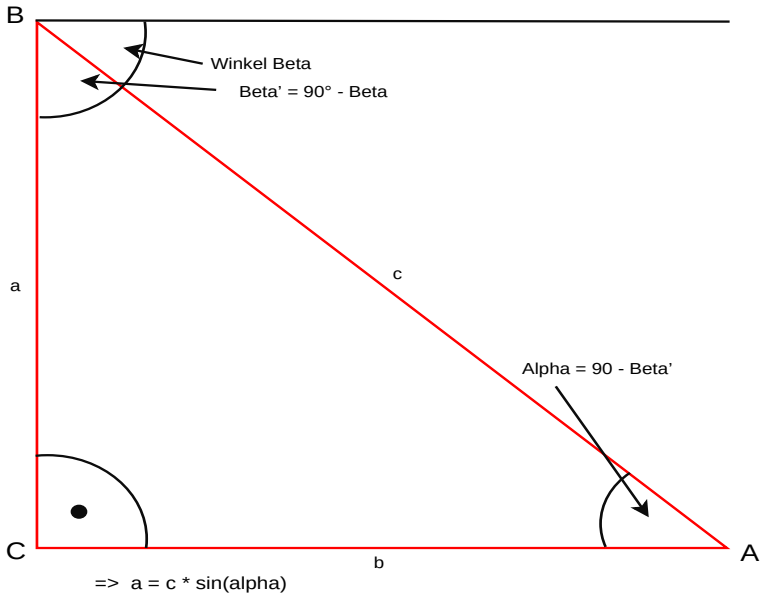


Abbildung 5.22: Berechnung der z-Position

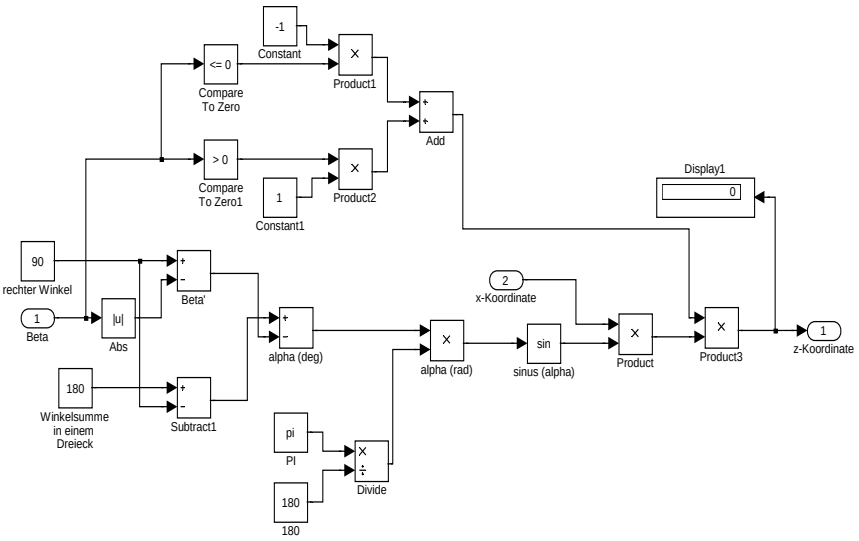


Abbildung 5.23: Berechnung der z-Position in Matlab

- Strömungswiderstandskraft/ Reibungswiderstandskraft: (siehe 5.7.1)
  Bei der gegebenen Bauform benutzen wir die laminaren Strömungseigenschaften. Somit wird der Strömungswiderstand nur durch die innere Reibung des Mediums verursacht. Diese Formel (Stokessches Gesetz) lässt sich leicht in Matlab Simulink umsetzen (siehe Abb. 5.26) und als Subsystem in

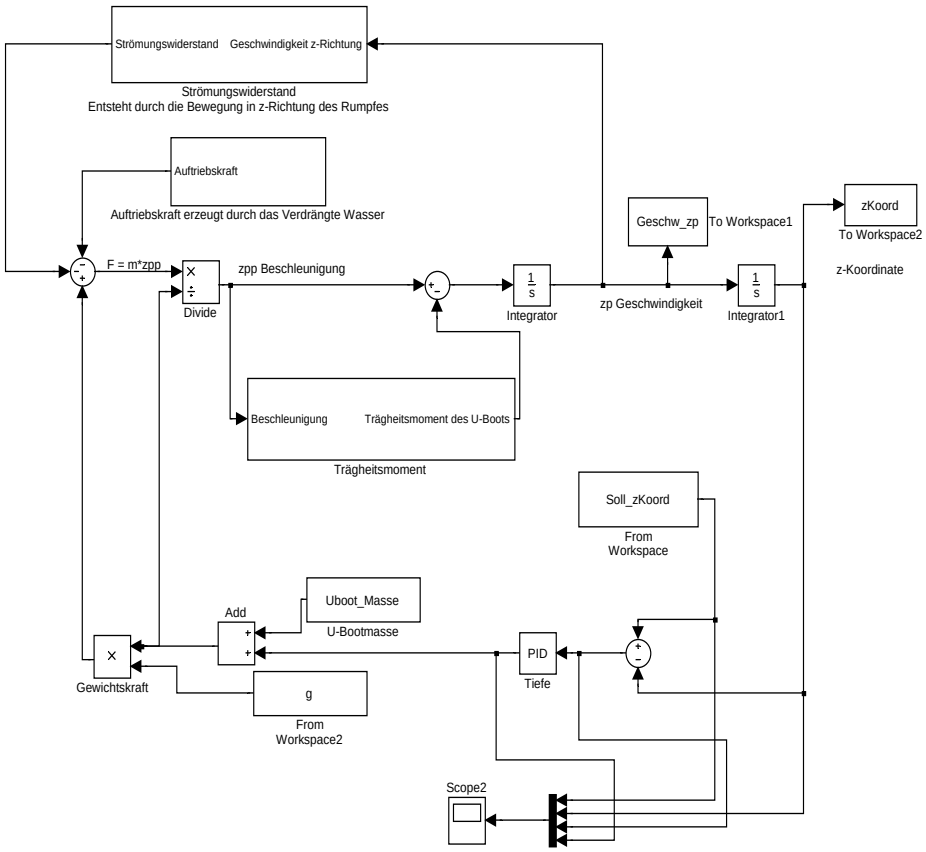


Abbildung 5.24: Regelkreis des statischen Tauchens

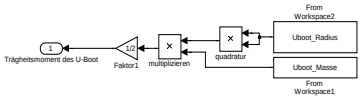


Abbildung 5.25: Trägheitsmoment des U-Boots

die Antreibsregelung einbinden.

$$F_{Reibung} = \pi \cdot \eta \cdot v \cdot r$$

η = Viskosität des Mediums; bei Wasser entspricht es dem Wert 1.
v = Geschwindigkeit des U-Boots

r = Radius des U-Boots

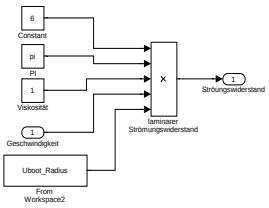


Abbildung 5.26: Strömungswiderstand des U-Boots

- Antriebskraft:
  Die Antriebskraft zu simulieren gestaltet sich dagegen ein wenig aufwändiger. Wenn wir die Antriebskraft verändern wollen, lässt sich diese am einfachsten über die Drehzahl des Antriebsmotors gestalten, was am realen Modell auch einfach durch die Ansteuerung des Motors über ein PWM-Signal geschieht.
  Um nun den tatsächlichen Schub zu ermitteln, der das U-Boot in x-Richtung beschleunigt, benötigen wir einige wichtige Formeln:

$$P_{Welle} = M \cdot \omega = M \cdot n \cdot \frac{\pi}{30}$$

$M = DrehmomentdesMotors$
$\omega = Winkelgeschwindigkeit$
$n = UmdrehungenderWelleproSekunde$
Hierbei handelt es sich um die Wellenleistung. Sie wird benötigt um unsere Schraube/Propeller zu drehen. Die direkte Umsetzung ist in Abb. 5.27 zu sehen. Da wir keine Übersetzung zwischen Motor und Welle haben, ist die Welle nur eine Verlängerung der Motorwelle und dementsprechend liegt die gleich Drehzahl und das gleiche Drehmoment an der Antriebswelle wie an der Motorwelle an. Mit Hilfe des Datenblattes über den Antreibsmotor (siehe Anhang) und in dem zu findenen Leistungskurven lässt sich einfach zu einer gewissen Drehzahl das dementsrechende Drehmoment bestimmen.

$$M_{Motor} = n \cdot -0,0353 + 2$$

n Umdrehungen der Welle pro Sekunde.
Wie man in Abb. 5.28 sehen kann, spielt die Drehrichtung eine wichtige Rolle. Wir gehen hier von einer positiven Drehrichtung aus, wenn das U-Boot sich vorwärts bewegt und negative Drehrichtung, wenn sich das U-Boot rückwärts bewegt. Mit Hilfe dieser Werte lässt sich der Schub berechnen:

$$S_{Antrieb} = \sqrt[3]{2 \cdot \rho \cdot F \cdot (\varsigma \cdot P)^2}$$

ρ = Dichte des Mediums
F = Fläche der Rotoren
ς = Gütegrad der Welle
Zum Gütegrad lässt sich noch folgendes anmerken; möchte man einen gewissen Schub erziehlen,

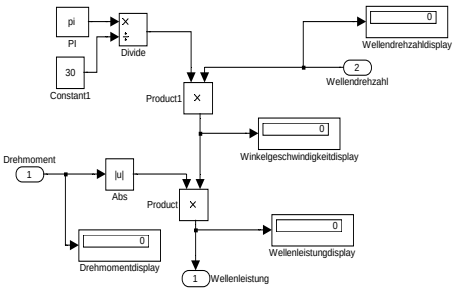


Abbildung 5.27: Wellenleistung

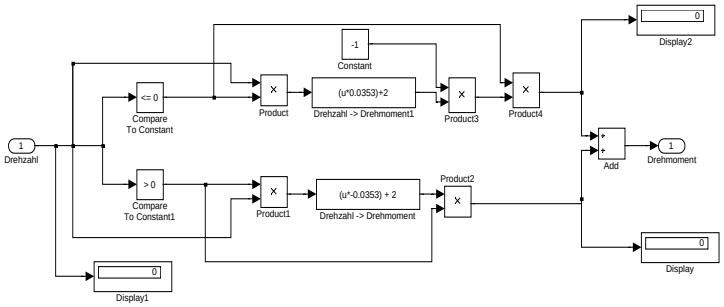


Abbildung 5.28: Drehmoment des Hauptantriebs

wird man feststellen, dass die dazu nötige Wellenleistung nicht mit der tatsächlichen benötigten Wellenleistung übereinstimmt. Die Grunde dafür sind die verschiedenen Bauarten von Schrauben/Propellern bzw. der Reibungsverlust der Welle. Der Gütegrad lässt sich nun wie folgt berechen:

$$\varsigma = \frac{P_{id}}{P}$$

P_{id} entspricht hier einer idealen Wellenleistung ohne Verluste. Dementsrechend kann man die For-

mel so umstellen, dass man die Wellenleistung herausbekommt:

$$P_{Welle} = \varsigma \cdot (M \cdot \omega) = \varsigma \cdot (M \cdot n \cdot \frac{\pi}{30})$$

Wie man in der Abb. 5.29 sehen kann, wir durch einen Test überprüft, ob die Wellenleistung positive

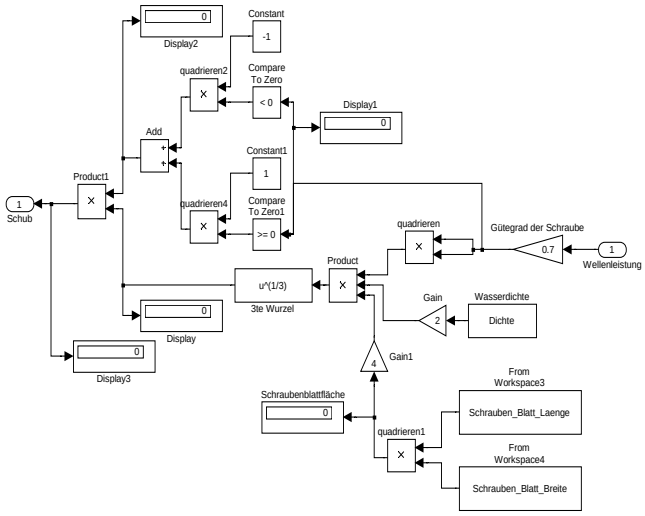


Abbildung 5.29: Schub

oder negative ist. Dieses spiegelt das positive oder negative Drehen der Welle wieder. Würde man diese Abfrage nicht machen, wäre nicht zu ersehen, ob das U-Boot rückwärts oder vorwärts fährt und zudem könnte die Simulation nicht die Geschwindigkeit veringern. Gegen den Schub wirkt der Strömungs- /Reibungswiderstand. Abzüglich diesem erhalten wir unseren Schub, der das U-Boot in x-Richtung beschleunigt. Um die Beschleunigung zu ermitteln teilt man einfach den Schub durch die U-Bootmasse. Bei jeder Beschleunigung haben wir aber noch das entgegenwirkende Trägheitsmoment. Bereinigen wir unsere Beschleunigung um diesen Faktor können wir anschließend durch Intrgration die Geschwindigkeit berechnen, die für die Berechnung des Strömungs- /Reibungswiderstand erforderlich ist.
In einigen Fällen kommt es vor, dass sich negative oder positive auf den Schub noch die Corioliskraft (siehe 5.18) hinzuaddiert. Dieses ist z.B. bei der Lage- und Kursregelung der Fall.

Diese einzelnen Kräfte und Moment können so zu einer Regelschleife zusammengesetzt werden, wie es in Abb. 5.30 verdeutlicht wird. Wie man erkennen kann, bekommt der PID-Regler einen Sollwert in Form einer Konstanten. Dieser wird mit dem Ist-Wert vergleichen und dementsprechend die Umdrehungzahl der Welle angepasst. Über die Wellenleistung kann man den Schub berechnen, welcher bereinigt, sich weiter durch Integration in die aktuelle Geschwindigkeit umrechnen lässt. Für die Berechnung des Strömunswidersatnds benötigen wir die aktuelle Geschwindigkeit. Diese ist aber zu Anfang 0. Deshalb wird ein ”Unit Delay” dazwischen platziert, welches den Anfangswert 0 durchpropagiert.

Wenn man nun die Regelung laufen ließe, würde auch das Erreich fantastischen Umdrehungszahlen des Motors nicht unmöglich sein. Darum wurden Begrenzungsblöck (Saturation) eingebaut, die die Drehzahl auf einen Bereich von $\pm$ 50 Umdrehungen pro Sekunde beschränken.

Auf die PID-Werte und die Simulationsergebnisse wird unter Punkt ”Test am Modell” eingegangen.

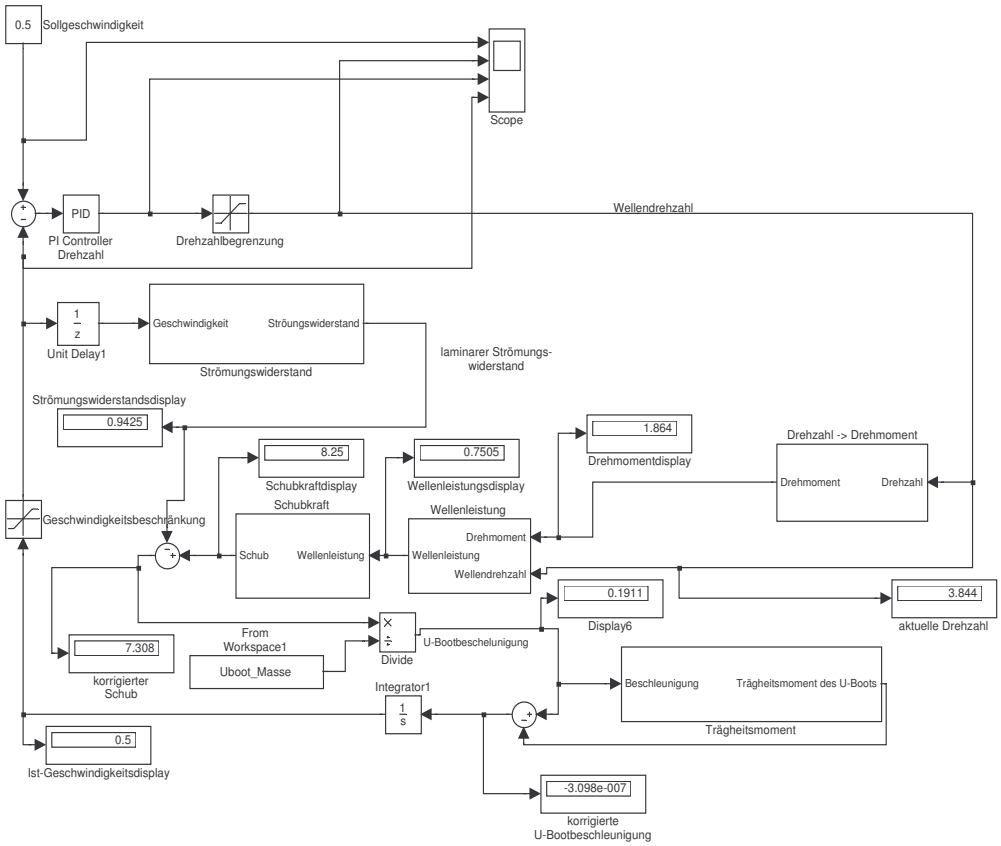


Abbildung 5.30: Antreibsregelung

5.10.4 Trimmung mittels Tauchtanks

Eingangs hatten wir geplant die Trimmung mittels Tauchtanks zu gewährleisten. Die grundlegende Idee ist, dass die Tauchtanks unterschiedlich befüllt werden und da sie ja an verschiedenen Positionen im U-Boot eingebaut wurden, eine direkte Lageveränderung als Drehung um die y-Achse erzeugen können. Zusätzlich kann eine Sollfüllung der Tauchtanks, entsprechend einer gewissen Tiefe, übergeben und die Regelung versucht ausschließlich durch Verlagerung des Gewichts zwischen den beiden Tauchtanks, die Trimmung zu gewährleisten. Möglich wäre es, die statische Tauchregelung zu integrieren und somit den Füllstand zu ermitteln und zu übergeben. Hierzu steht auch ein Modell zur Verfügung (siehe 5.31). Wir haben uns aber schnell dafür entschieden, dieses Feature nicht weiter zu verfolgen. Dies hatte mehrere Gründe; zum einen stellten wir fest, dass die Tauchtanks ein sehr träges Verhalten aufwiesen. Ein Aufschwinngen oder nur langes Einschwingen war nicht zu vermeiden. Zudem hatte das U-Boot durch eine lange Austarhierungsphase eine schon sehr gute Lage im Wasser. Die dynamsiche Lagewinkelregelung ist für weitere Änderungen wesentlich präziser und schneller geeignet.

Für die Implementierung gab es zudem das Problem, wärend dieser Entscheidungsphase, ob wir dieses Feature implementieren oder nicht, dass die Sensorik nicht ordnungsgemäß funktionierte und zudem kein zweiter Außendrucksensor für die Ermittlung des Ist-Wertes zur Verfügung stand.

5.10.5 Gesamtregelung

Wie man erkennen kann, sind hier alle Regelungen als Subsysteme enthalten. Zudem wurde eine Schaltung eingebaut, mit welcher man die noch nicht wirklich funktionierende dynamische Tauchregelung ausschal-

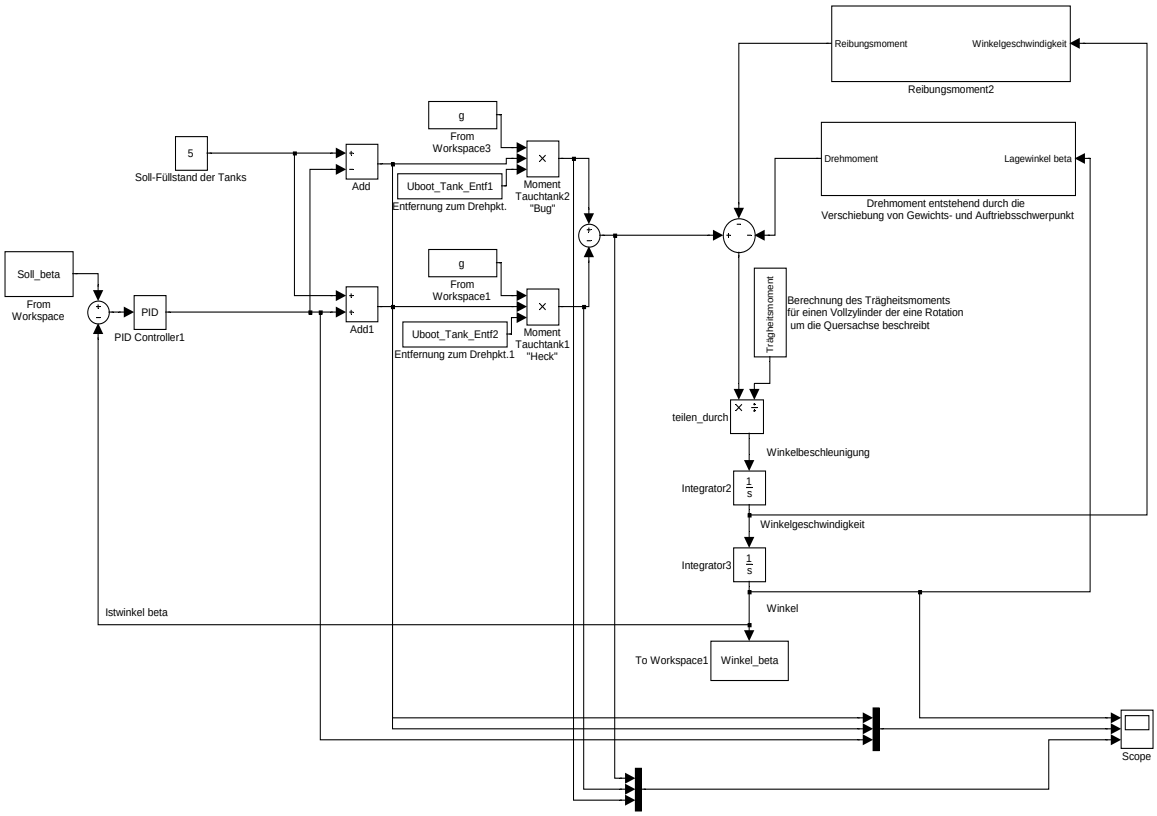


Abbildung 5.31: Trimmungsregelung

ten und die Positionierung lediglich über die statische Tauchregelung durchführen kann. Dieses geilt aber lediglich für das Modell.
Der Rest funktioniert sehr gut und auch zuverlässig.

5.10.6 Test am Modell

Hiermit ist die Simulation der Modelle in Matlab Simulink gemeint. Dadurch sollen die PID-Werte getestet werden um mögliche Verbesserung anzustreben.

In der Entwickling der einzelnen Reglermodelle, haben wir sie erst autonom simuliert und optimiert. Doch nachdem wir sie zu einer kaskadierten Regelung zusammengesetzt haben, fiel auf, dass die Simulationen in jedem Fall ein schlechteres Ergebnis präsentierten, wenn nicht sogar ein instabiles Verhalten aufwiesen. Das musst noch einmal optimiert werden. Ein Beispiel hierfür ist die Kurswinkelregelung. Sie vereint eine Geschwindigkeitsregelung und dir eigentliche Kurswinkelregelung.
In folgender Tabelle findet man einen Überblick über die PID-Werte der einzelnen PID-Regler die in Regelung verwendet wurden:

Regler:	P-Wert:	I-Wert	D-Wert:
Antrieb	6,5	2,5	3,5
Lagewinkel	3,5	0,25	10
Kurswinkel	3,5	0,25	10
Statisch Tauchen	20	0,4	0,7

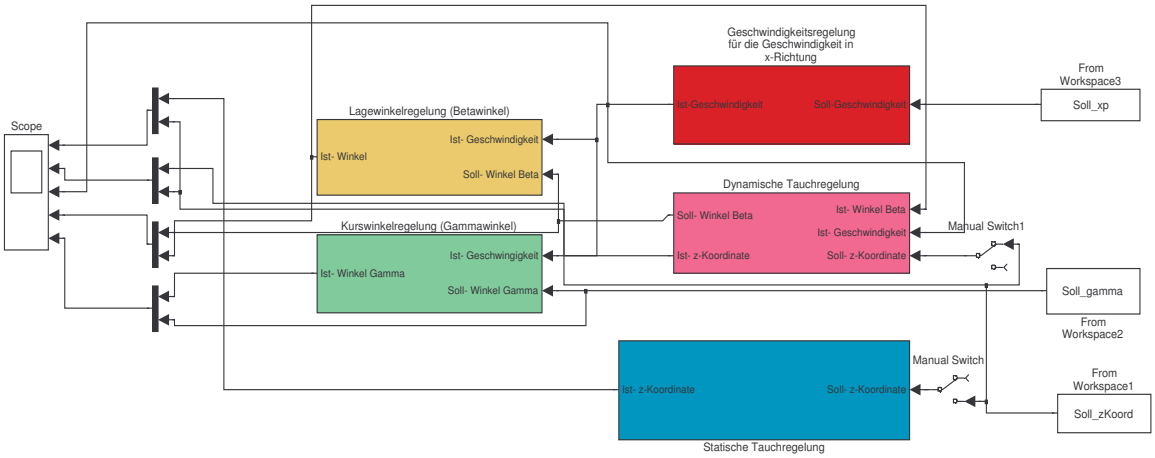


Abbildung 5.32: Zusammengesetzte Gesamtregelung

Letztendlich sind folgende Regelungmodelle funktionstüchtig und getestet:

- statische Tauchregelung,
- Lagewinkelregelung,
- Kurswinkelregelung,
- Antriebsregelung

Wie oben schon geschildert, ist die dynamische Tauchregelung als auch die Trimmregelung nicht vollständig implementiert und getestet.

Simulationsergbnisse als Diagramme

Im Folgendem sind die Simulationergenisse als Diagramme aufgeführt, aus denen man die Güte der Regelungen entnehmen kann

5.11 Modell vs. Realität

In Testreihen, die wir im Universitätsschwimmbad dürchführten, zeigte sich, dass die PID-Werte, die für die Modelle optimal waren, für die eigentliche Regelung unbrauchbar waren. Dies stellte sich insbesondere bei der statischen Tauchregelung heraus.

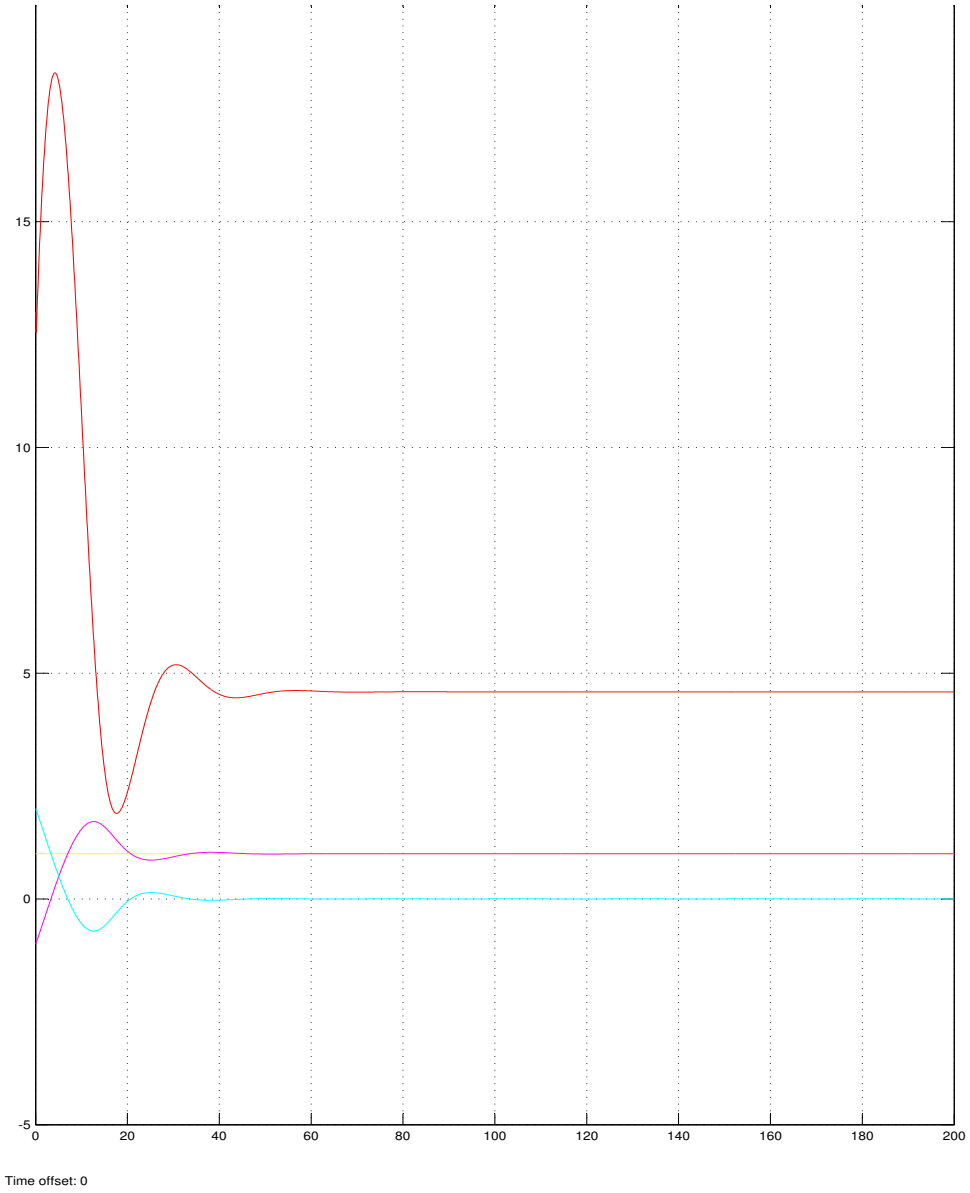


Abbildung 5.33: Simulationergebnis zur Antriebsregelung

Dafür gibt es verschiedene Ursachen, wobei man sie in zwei Klassen grob unterteilen kann:

- Nicht zu modellierende Umwelteigenschaften:
  - Thermische Strömungen im Schwimmbecken
  - Strömungen, die durch die Wasserfilteranlage erzeugt werden
- Technische Entwicklingsprobleme:
  Ein großer Nachteil war, dass die von uns verwendete 433 MHz Funkverbindung bei einer Tiefe von ca. 20 cm abbrach, die 40 MHz bei einer Tiefe von ca. 40 cm. Bei der statischen Tauchregelung war dies ein Grund, welcher das Testen fast unmöglich machte, da wir, bezüglich der Sicherheitsanforderungen, einen Fail-Safe-Mode implementiert haben, der das U-Boot nach Abriss der Funkverbindung wieder auftauchen lässt. In der letzten Phase wurde dann die 27 MHz Funkverbindung implementiert.

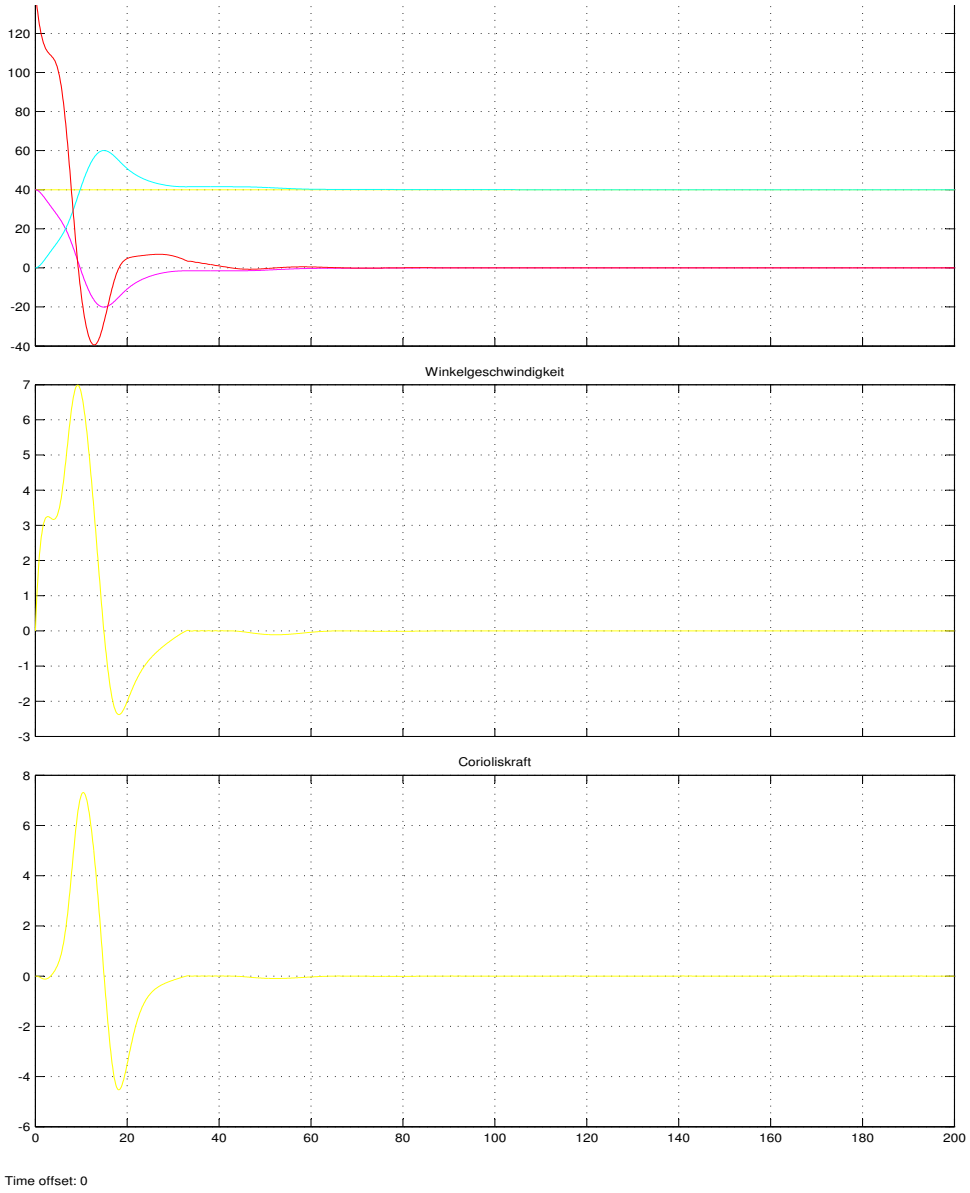


Abbildung 5.34: Simulationergebnis zur Kurswinkelregelung

Danach zeigte sich eine sehr sehr verzögerte Reaktion des U-Boots auf Befehle. Zudem erreichten wir nur eine ”stabile” Funkverbindung bis in eine Wassertiefe von ca. 1,2 m.
In machen Situationen musste wir über einige Werte Schätzungen anstellen, z.B. bei $c_W - Wert$ unseres neuen Bugsegments. Die Berechnung wäre an dieser Stelle viel zu komplex geworden. Als weiteres Beispiel lässt sich noch die Annahme über die Strömungsverhältnisse geben; sind sie laminar oder turbolent. In solchen Fällen haben wir nach unserem Verständnis Werte geschätzt oder Eigenschaften angenommen. Diese Entscheidungen wurden aber an entsprechender Stelle auch begründet. Trotzdem stimmen sie nicht unbedigt mit den exakten Werten oder Eigenschaften überein sondern entsprechen Näherungswerten.

Das Resultat war, dass wir um Schwimmbad sämtliche Regelungen erneut optimieren mussten und uns nur als Ausgangswerte an den, in den Simulationen der Modelle, verwendeten Werte orientiert haben.

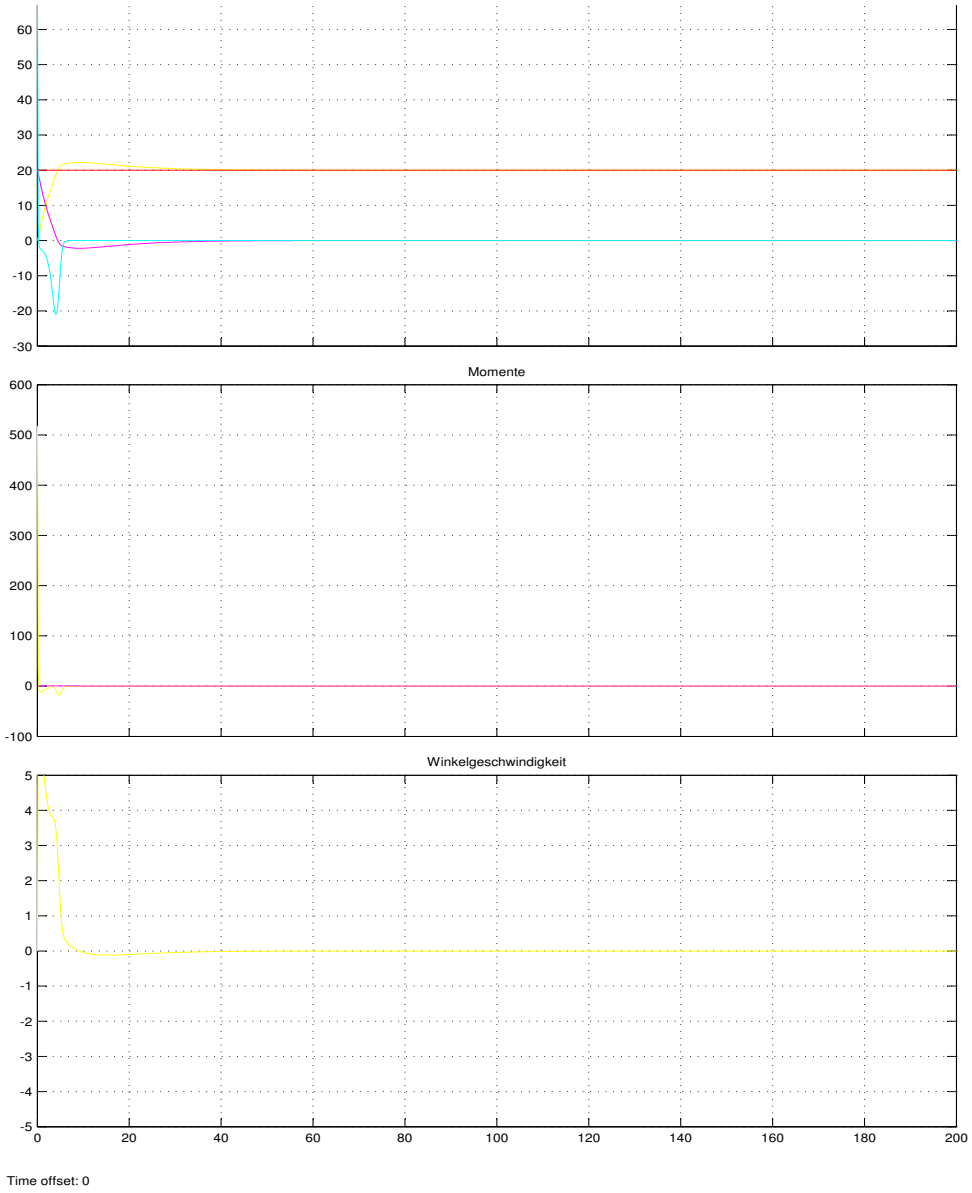


Abbildung 5.35: Simulationergebnis zur Lagewinkelregelung

5.12 Regelung in Software

Das entwickelte Konzept für die Regelung soll zuerst in Software umgesetzt und getestet werden, dies aufgrund der einfachen Möglichkeit Software testen und ändern zu können. Angedacht war danach eine Implementierung in Hardware, welche aber wieder vorworfen bzw. nicht aufgenommen wurde.

Die PID-Regler für die verschiedenen Komponnenten wurden zuerst in Software geschrieben. Hierzu wurde eine Funktion die einen PID-Regler und mehrere Funktionen die die einzelnen Regler bzw. Regelkreise implementieren. Die verschiedenen Funktionen werden in der Regelungstask abgearbeitet und ergeben damit das Verhalten einer kaskadierten Regelung. Auf diese Art und Weise kann jeder Regelkreis einzeln implementiert, eingestellt und getestet werden. Zu dem lassen sich komplette Regler leicht ersetzten oder einfügen falls Änderungen am Boot vorgenommen werden. Um angepasst an die restliche Software

zu arbeiten werden sämtliche Berechnungen in einer Festkommaarythmetik vorgenommen.

5.12.1 PID-Regler in C

Die Funktion `Pid` stellt einen einfachen PID-Regler zur Verfügung den weitere Funktionen für die einzelnen Regelungen nutzen.

Der eigentliche Regler verbirgt sich hinter der Methode
`int Pid_step(Pid *pid, int xIst,int xSoll)`
anhand der letzen fünf Regelabweichungen, dem Sollwert sowie dem angestrebten Istwert berechnet sie den aktuellen Regleroutput. Hierzu wurde eine einfache Rechenforschrift genutzt, der Form:

$$u_{(n)} = K \cdot e_{(n)} + (e_{(n-1)} + e_{(n-2)} + e_{(n-3)} + e_{(n-4)} + e_{(n-5)}) \cdot T_i + (e_{(n-1)} - e_{(n)}) \cdot T_d$$

Dies ist eine vereinfachte Form der PID-Reglerfunktion die ein schnelles und annähernd genaues berechnen eines Reglerwertes ermöglicht. Die aktuelle Regelabweichung wird berechnet und wie die letzten fünf Regelabweichungen in einem Array gespeichert, danach aus Soll- und Istwert sowie den diversen Regelabweichungen der neue Regeloutput berechnet und zurückgegeben.

Als Beispiel hier der C-Code:

```
int Pid_step(Pid* pid, int xIst,int xSoll)
{
  pid->myXDiff = xSoll - xIst;

  // P Anteil
  Pid_mul(pid->myXDiff,pid->myProportional)

  // D Anteil
  + Pid_mul(pid->myXDiff - pid->myLastXDiff[0],pid->myDerivate)

  // I Anteil
  + Pid_mul(pid->myLastXDiff[5] + pid->myLastXDiff[4]+ pid->myLastXDiff[3]
  + pid->myLastXDiff[2] + pid->myLastXDiff[1]
  + pid->myLastXDiff[0], pid->myIntegral);

  for(i=5;i>0;i--){
  pid->myLastXDiff[i] = pid->myLastXDiff[i-1];}
  pid->myLastXDiff[0] = pid->myXDiff;

  return controlOutput;
  }
```

Nachdem die aktuelle Regelabweichung berechnet ist werden die drei Anteile des Regleroutputs berechnet und addiert. Einige Multiplikationen sind dazu nötig die über eine eigene Funktion realisiert wurde, da es sich um eine Festkommamultiplikation handelt. Die einzelnen Berechnungsschritte entsprechen der oben erwähnten Rechenforschrift.

5.12.2 Einzelregler

Die verschiedenen einzelnen Regler nutzen diese Funktion und erweitern sie jeweils für den speziellen Regelkreis den sie betreiben um z.B. verschiedene Sensorinformationen abzufragen.

Lagewinkel um x-Achse

```
int ArcBeta_step(ArcBeta* arcBeta, int xSetpoint){
```

```
    int xIs = NavigationWrapper_getInclineY();

    controlOutput = Pid_step(arcBeta, xIs,xSetpoint);

    return controlOutput;
}
```

Die Funktion beinhaltet den Regler für den Lagewinkel β um die y-Achse. Als Sensor wird ein Neigungssensor genutzt, die Einwirkung auf das Boot findet über die Ansteuerung der Ruder statt. Hier wird zunächst der aktuelle Neigungswinkel von der Navigation angefordert und als Istwert genutzt. Die Datenstruktur arcbeta beeinhaltet neben den Reglerwerten, die letzten berechneten Regelabweichungen und einige Hilfsvariablen. Grundsätzlich sind alle Regler in dieser Form aufgebaut und unterscheiden sich nur durch verschiedene Sensoren die abgefragt werden sowie durch diverse Randbedinungen die eingehalten werden müssen. Anschaulicher ist die Strutktur auf Abbildung 36 zu erkennen.

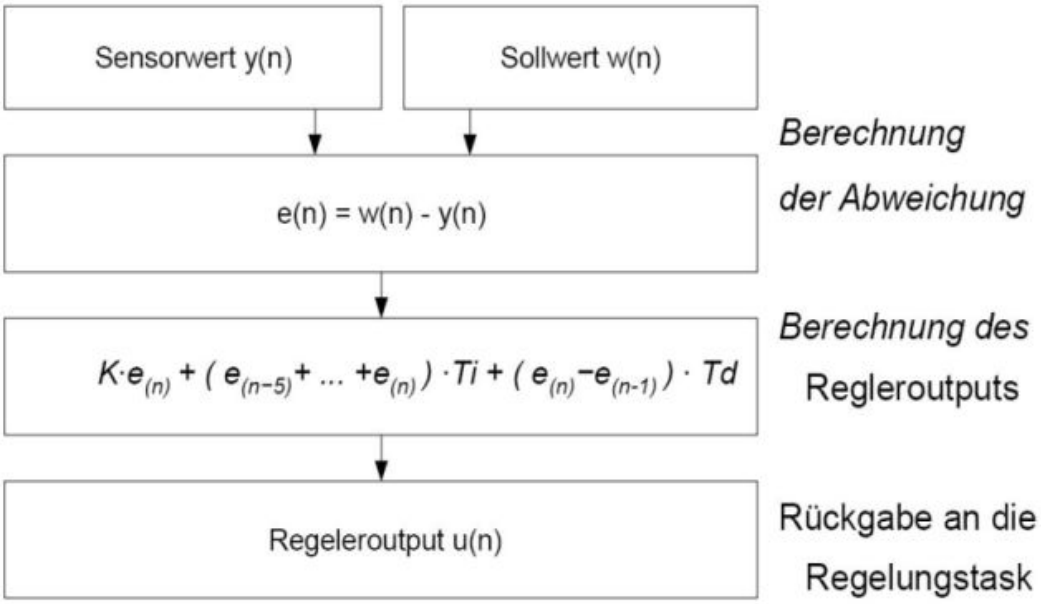


Abbildung 5.36: Struktur der Regler

Lagewinkel um die y-Achse (Tauchtanks)

Ähnlich der Regelung des Lagewinkels über die Ruder nur werden anstatt der Ruder die Tauchtanks unterschiedlich befüllt und damit der Lagewinkel manipuliert. Die verwendete Sensorik stimmt mit der vorhergehenden Regelung über ein.

```
int ArcBetaDiptank_step(ArcBetaDiptank* arcBetaDiptank, int xSetpoint){

    int xIs = NavigationWrapper_getInclineY();

     controlOutput = Pid_step(arcBetaDiptank, xIs,xSetpoint);

     return controlOutput;
}
```

Kurswinkelregelung

Der Kurs, also der Winkel um die z-Achse wird einzig durch die Ruder beeinflußt als Messglied dient ein Gyroskope. Der Soll-Kurswinkel der an das Boot gesendet wurde, wird allerdings als Kursänderung interpretiert. Sollte die gewünschte Kursänderung 0 betragen wird der letzte zwischengespeicherte Kurswinkel als Sollwert angenommen. Dies hat den Vorteil das das Boot versucht den Kurs zu halten falls keine Eingabe von aussen erfolgt.

```
int ArcGamma_step(ArcGamma* arcGamma, int xSetpointChange){
     ...
    if(xSetpointChange!=0)
     {
       xSetpoint = xIs + xSetpointChange;
       arcGamma->myLastValue = xIs;
     }
     else{
       xSetpoint = arcGamma->myLastValue;
     }
       ...
    }
```

Tiefenregelung über die Tauchtanks (statisch)

```
int DepthDiptank_step(DepthDiptank* depthDiptank, int xSetpointChange)
```

Das statische Tauchen erfolgt über die Tauchtanks welche abhängig von der gewünschten Tiefen gefüllt oder gelenzt werden. Die Tiefe wird über Drucksensoren ermittelt. Diese Regelungsfunktion sowie die Ansteuerung der Tauchtanks wird noch erstetzt durch ein verändertes System.
Die derzeitige Ansteuerung erfolgt über die Vorgabe eines Füllstandes den der Tauchtank anfährt. Da dies Träge und Ungenau ist werden die Tauchtanks über eine Geschwindigkeitsansteuerung der Motoren angsteuert werden. Dies hat den Vorteil das die Regelung sich beim Einschwingen nahe des Sollwertes mit langsamen Motorbewegungen an den optimalen Füllstand antasten kann. Das erhoffte Ergebnis ist ein verbessertes Einschwingverhalten.

Tiefenregelung über die Ruder (dynamisch)

```
int DepthRudder_step(DepthRudder* depthRudder, int xSetpointChange)
```

Neben dem statischen Tauchen besteht die Möglichkeit dynamisch zu Tauchen über das erzeugen vom Abtrieb am Rumpf. Das dynamische Tauchen erfodert es das sich der Rumpf um die y-Achse neigt. Um dies zu erreichen gibt die Tiefenregelung der Lagewinkelregelung einen Sollwert vor welcher dann angefahren werden kann. Als Sensorik dient hier der Druckesensor um die Tiefe zu ermitteln.

Geschwindigkeitsregelung

Die Geschwindigkeit des Bootes in x-Richtung wird durch einen Beschleunigungssensor ermittelt. Der Hauptantrieb wird über ein PWM-Signal angesteuert welches über die Regelung kontrolliert wird. Ähnlich der Kurswinkelregelung wird der übertragene Wert als Geschwindigkeitsveränderung interpretiert und versucht die aktuelle Geschwindigkeit zu halten falls keine Eingabe von der Hostsoftware kommt.

```
int SpeedEngine_step(SpeedEngine* speedEngine, int xSetpointChange){
    ...
     if(xSetpointChange!=0){
        xSetpoint = xIs + xSetpointChange;
        speedEngine->myLastValue=xIs;
     }
```

```
 else{
   xSetpoint = speedEngine->myLastValue;
 }
 ...
controlOutput = Pid_step(speedEngine, xIs,xSetpoint);
 ...
}
```

5.12.3 Gesamtregelung

Da nun PID-Regler zur Verfügung stehen kann das Prinizp der kaskadierten Regelung umgesetzt werden. Das angedachte System aus mehreren Regelungen die teilweise ineinander greifen läßt sich einfach in Software umsetzten. Es wird hierbei ein Task auf dem FPGA-Board gestartet welche periodisch Aufgerufen, die Regelung des Ubootes übernimmt. Innerhalb des Tasks werden sequenziell die verschiedenen Reglerfunktionen abgearbeitet, neue Regeloutputs berechnet und zuletzt die Aktorik angesteuert. Eine kurze Übersicht über den Ablauf der abgearbeiteten Funktionen, siehe Bild.

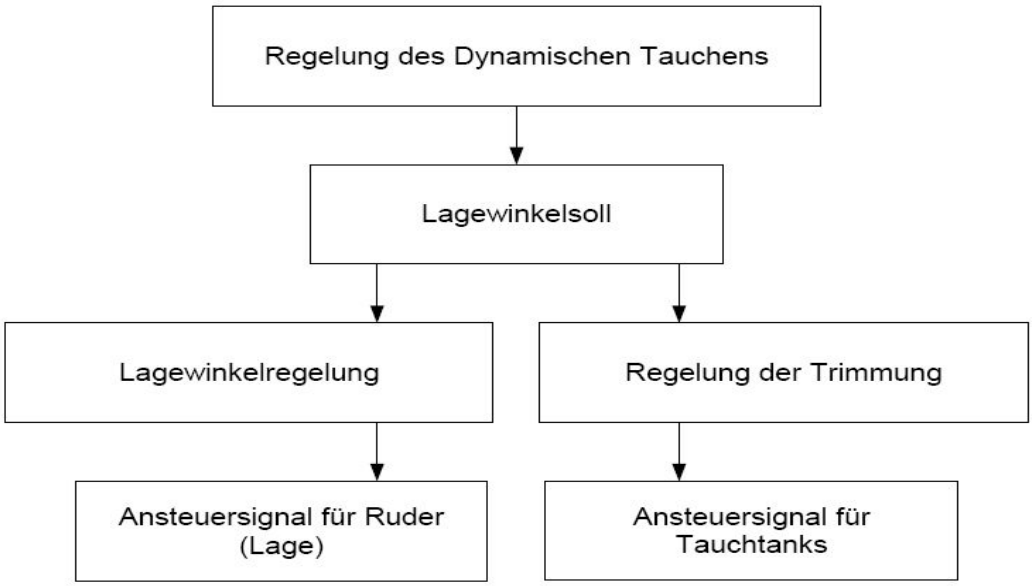


Abbildung 5.37: Ablauf der Regelungstask1

Die Regelungen für die Kurswinkelregelung, statisches Tauchen und die Geschwindigkeitsregelung werden unbeeinflußt voneinander ausgeführt. Die Regelung des dynamischen Tauchens gibt den von ihr berechneten Output weiter. Dieser wird als Sollwert für die Lagewinkelregelung über die Ruder sowie die Tauchtanks genutzt, welche ihrerseits versuchen eine Lagewinkelveränderung des Bootes zuerreichen. Im Anschluss werden dann die diversen Aktoren angesteuert.

Methoden der Gesamtregelung

Einige Methoden der Regelungstask die verschiedene Funktionalität implementieren.

`void Control_changePid(int controller, int p, int i, int d)`
Diese Methode ermöglicht es die Reglerwerte P,I und D eines bestimmten Reglers zu ändern, wobei mit Controller die Nummer des Regelkreises angegeben wird.

`void Control_enablePid(int pid)`

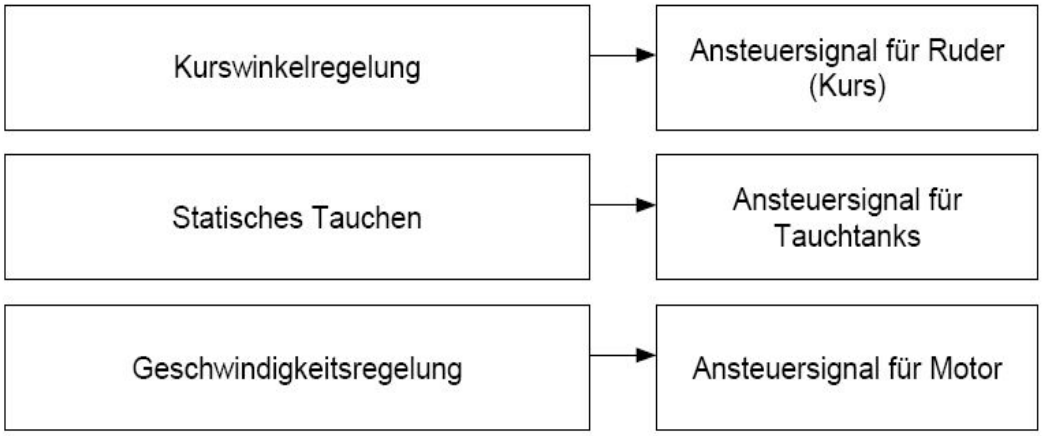


Abbildung 5.38: Ablauf der Regelungstask2

Diese Methoder schaltet die Regler von einfachen P-Reglern um auf PID-Regler. Setzt also die I und D-Werte der Regler. Ein einfacher P-Regler erlaubt eine einfache Ansteuerung über die Fernbedienung.

`void Control_doRun()`
Die eigentliche Regelung, hier werden sämtliche Regler abgearbeitet und Aktorik gesetzt. Sie muss einmal aktiviert werden und läuft dann als Task unendlich weiter. Während die Regelungstask pausiert wurde ist ein Teil der Aktorik in Ruheposition, dies bedeutet das z.B. die Tauchtanks in eine Nulllage fahren. Die Ansteuerung der Servos erfolgt als „harter“ X-Mischer, die einzelnen Ansteuersignale für die Ruder und Tauchtanks werden jeweils aufaddiert und damit die Aktorik angesteuert.

`int Control_doPause()`
Diese Methode ermöglicht es die Regelungstask pausieren zulassen. Eine wakeup-Methode lässt die Task nachdem sie pausiert wurde weiter arbeiten.

```
void Control_setUseSpeed(int value)
void Control_setUseBeta(int value)
```
usw.
Diese Methoden ermöglich es die einzelnen Regelungen ein- oder auszuschalten und entsprechenden mit der entsprechenden Get-Methoden den Status der Regelung abzufragen.

5.13 Fazit

5.13.1 Modell vs. Realität

In Testreihen, die wir im Universitätsschwimmbad dürchführten, zeigte sich, dass die PID-Werte, die für die Modelle optimal waren, für die eigentliche Regelung unbrauchbar waren. Dies stellte sich insbesondere bei der statischen Tauchregelung heraus.

Dafür gibt es verschiedene Ursachen, wobei man sie in zwei Klassen grob unterteilen kann:

- Nicht zu modellierende Umwelteigenschaften:
  - Thermische Strömungen im Schwimmbecken
  - Strömungen, die durch die Wasserfilteranlage erzeugt werden
- Technische Entwicklingsprobleme:
  Ein großer Nachteil war, dass die von uns verwendete 433 MHz Funkverbindung bei einer Tiefe von

ca. 20 cm abbrach, die 40 MHz bei einer Tiefe von ca. 40 cm. Bei der statischen Tauchregelung war dies ein Grund, welcher das Testen fast unmöglich machte, da wir, bezüglich der Sicherheitsanforderungen, einen Fail-Safe-Mode implementiert haben, der das U-Boot nach Abriss der Funkverbindung wieder auftauchen lässt. In der letzten Phase wurde dann die 27 MHz Funkverbindung implementiert. Danach zeigte sich eine sehr sehr verzögerte Reaktion des U-Boots auf Befehle. Zudem erreichten wir nur eine ”stabile” Funkverbindung bis in eine Wassertiefe von ca. 1,2 m.
In machen Situationen musste wir über einige Werte Schätzungen anstellen, z.B. bei cw-Wert unseres neuen Bugsegments. Die Berechnung wäre an dieser Stelle viel zu komplex geworden. Als weiteres Beispiel lässt sich noch die Annahme über die Strömungsverhältnisse geben; sind sie laminar oder turbolent. In solchen Fällen haben wir nach unserem Verständnis Werte geschätzt oder Eigenschaften angenommen. Diese Entscheidungen wurden aber an entsprechender Stelle auch begründet. Trotzdem stimmen sie nicht mit den exakten Werten oder Eigenschaften überein sondern entsprechen Näherungswerten.

Das Resultat war, dass wir um Schwimmbad sämtliche Regelungen erneut optimieren mussten und uns nur als Ausgangswerte an die in den Simulationen der Modelle verwendeten Werte orientiert haben.

5.13.2 Zusammenfassung

Das Boot verfügt über diverse Regelungen die es ermöglichen die Größen Kurs, Lagewinkel, Geschwindigkeit sowie Tiefe zu regeln. Lagewinkel sowie Tiefe können über jeweils zwei verschiedene Aktoren beeinflußt werden. Das Boot wurde in Matlab modelliert und daran ein Regelungssystem entwickelt. Anschließend die Regler in C-Code implementiert und auf das FPGA übertragen um dort als Task zuarbeiten. Bei Test im Schwimmbad mussten wir allerdings feststellen das die im Modell gefundenen Regelwerte nicht auf das reale Boot übertragen werden konnten. Daraus folgte das die gesamte Regelung neueingestellt werden musste, dies ist bislang noch nicht abgeschlossen. Darüberhinaus konnten das Trimmen über die Tauchtanks sowie das dynamische Tauchen noch nicht eingehend getestet werden.
Über dies bleibt noch die Frage in wie weit Modell und Realität sich voneinander unterscheiden, in diese Fragestellung fällt auch der Vergleich des Reglers in C-Code mit dem in Matlab vorhandenem Regler.

Kapitel 6

Hardware - Software Architektur

6.1 Motivation

Das System soll autonom tauchen können. Um dieses zu ermöglichen, muss eine Applikation alle Belange des Bootes steuern und kontrollieren können. Über Sensoren müssen Daten über die Umwelt gesammelt und ausgewertet werden, anhand dieser Daten und vom Benutzer kommenden Befehlen müssen Aktoren des Systems angesteuert werden. Die Aktoren verlangen verschiedene Techniken. Da das System autonom taucht und nicht immer vom Benutzer kontrolliert werden kann, ist Sicherheit ein besonderer Aspekt, den das System erfüllen muss.

Zu den Hauptaufgaben der Applikation gehören:

- das Berechnen der momentanen Position und Lage im Raum anhand von Sensorwerten (Navigation)
- Das Steuern der Aktoren auf Basis von Zielposition und momentaner Position (Regelung)
- Das Entgegennehmen von Befehlen des Benutzers über Funk und das Übermitteln von Daten an den Benutzer (Kommunikation)
- Das Überwachen von Kommunikation und Sensorwerten, um eine Notsituation zu erkennen (Sicherheit)
- Das Erstellen von Bildern mit zugehöriger Position zur nachträglichen Auswertung (Umweltwahrnehmung)

Es müssen also verschiedene Aufgaben durchgeführt werden, die sich abwechseln. Ein Betriebssystem muss dafür sorgen, das dies möglich ist. Da das System autonom taucht, ist es wichtig, dass das Verhalten der Software (insbesondere des Betriebssystems) vorhersagbar ist. Dies ist bei einem Echtzeitbetriebssystem möglich.

6.2 Einleitung

Das System wird zum Teil in Hardware und zum Teil in Software implementiert. Möglich wird dies durch das Benutzen eines FPGA (Field Programmable Gate Array), einer frei konfigurierbaren Hardwareplattform. Das System ist mit Hilfe der Entwurfsumgebungen Altera Quartus und Altera SOPC Builder implementiert. Dieses System bietet bereits viele Hardwareelemente als fertige Bausteine an, die man in seinem System benutzen kann.

Ebenfalls ist bereits ein für FPGAs programmierter 32 Bit Prozessor enthalten. Der Altera HAL (Hardware Abstraction Layer) erleichtert die Kommunikation zwischen Software und Hardwarekomponenten. Als Betriebssystem kommt der Echtzeitkernel MicroC/OSII zum Einsatz. Dieser ermöglicht das Ausführen mehrerer Tasks (siehe 6.3.4).

6.3 Grundlagen

6.3.1 Field Programmable Gate Array (FPGA)

FPGAs sind Chips, deren Struktur noch nicht festgelegt ist. Ein FPGA bietet die Möglichkeit, Logikzellen wie AND,OR usw. frei miteinander zu verknüpfen. Auf diese Weise können von einfachen Schaltungen bis zu komplexen 32 Bit Prozessoren beliebige Schaltungen realisiert werden und sind nur durch die Größe des FPGA's begrenzt. Das Programmieren eines FPGA ist nicht begrenzt, es kann beliebig oft durchgeführt werden.

6.3.2 Very High Speed Integrated Circuit Hardware Description Language (VHDL)

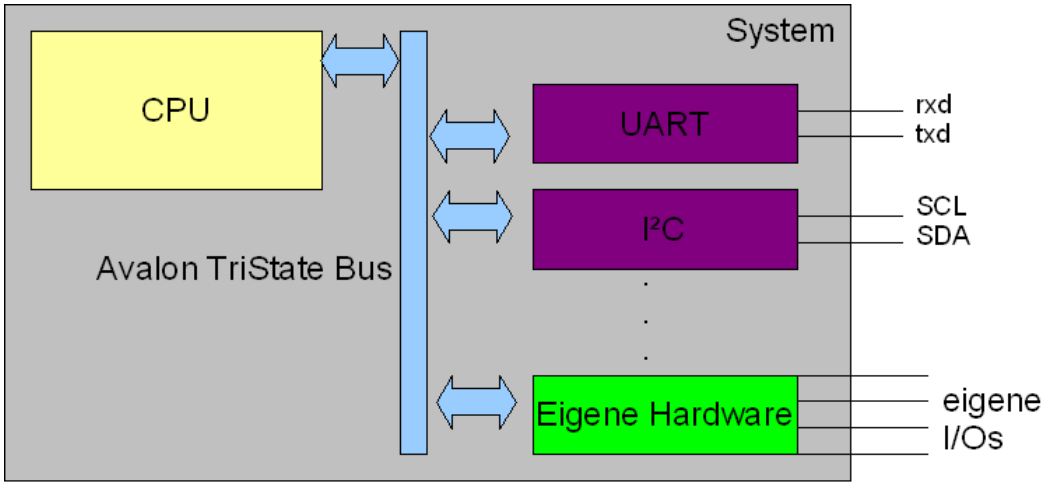
VHDL ist eine Hardwarebeschreibungssprache, die ähnlich einer Programmiersprache ist. Mit VHDL ist es möglich, komplexe digitale Systeme zu beschreiben. In VHDL wird das gewünschte Verhalten einer Schaltung beschrieben, die von Programmen synthetisiert und in eine Netzliste übersetzt werden. Diese Netzliste wird dann benutzt, um einen FPGA zu programieren.

6.3.3 Der Nios II Prozessor

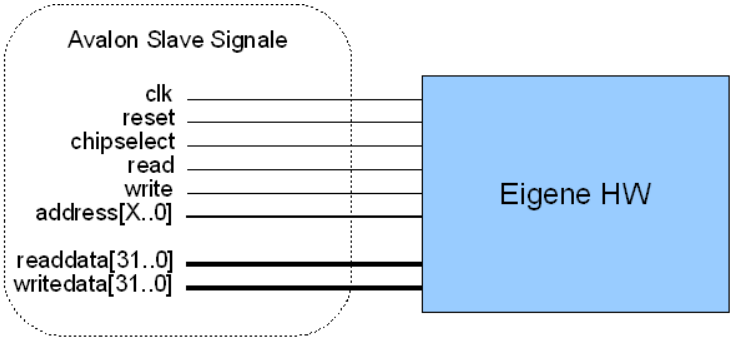
Der NiosII Prozessor ist ein speziell für FPGAs entwickelter 32 Bit RISC Prozessor von Altera. Er liegt als Hardwarebeschreibung vor und kann so auf beliebigen FPGAs eingesetzt werden.

Avalon Bus

Der Avalon Bus ist ein von Altera eingeführtes Interface für die einfache Kommunikation zwischen Peripheriehardware eines Nios Prozessors und der Software, die auf auf dem Prozessor ausgeführt wird. Die Spezifikation beschreibt die Signale, die die Hardware benutzt, und ihr Verhalten. Fast alle für den Nios verfügbare Hardwarelogik benutzt das Avalon-Interface.



Um eigene Hardware an den Avalon Bus anzubinden, müssen bestimmte Signale in dem Interface der eigenen Hardware definiert werden. Der Avalon Bus benutzt diese Signale nach einem festen Muster.



clk	Das System Clock Signal
reset:	Das System reset Signal (Optional)
chipselect:	Der Avalon Bus Signalisiert, dass er die Hardware benutzen will.
read:	Der Avalon Bus Signalisiert, dass er Daten von der Hardware erwartet.
write:	Der Avalon Bus Signalisiert, dass er Daten an die Hardare überträgt.
address[X..0]	Signalisiert, welche Information übertragen wird/werden soll (Optional).
writedata[31..0]	An die Hardware übertragene Daten.
readdata[31..0]	Kanal, über den die Hardware Daten über den Avalon Bus sendet.

Die Hardware muss diese Signale auswerten. Setzt der Avalon Bus chipselect und read, muss die Hardware die Adressleitung auswerten (falls vorhanden) und sofort die zu sendenden Informationen auf dem readdata Bus anlegen. Setzt der Avalon Bus das chipselect und write Signal, muss die Hardware die Adressleitung auswerten und den auf writedata anliegenden Wert auswerten.

Benötigt eine Hardware nur eine der beiden Zugriffsarten, kann die andere weggelassen werden.

Die Hardware kann jetzt über die Entwurfssoftware von Altera in das System eingebunden werden. Nicht-Avalon-Signale werden aus dem System herausgeführt. Ist das Hardwaresystem eingebunden, kann es in der Software sehr einfach angesteuert werden. Die Altera Entwurfssoftware publiziert alle Hardware-Adressen in einem Headerfile über Macros, die den Namen der eigenen Hardware enthalten. Mit Hilfe der Macros IORD (Lesen) und IOWR (Schreiben) kann über diese Hardware-Adresse einfach kommuniziert werden.

Der Befehl IORD(MY_HW_BASE,0); wird auf dem Avalon Bus so umgesetzt, dass die Adressleitung auf den Wert 0 gesetzt wird, chipselect und read werden aktiviert und der von der Hardware auf den readdata Bus angelegte Wert wird als Rückgabewert von IORD an die Software zurückgegeben.

6.3.4 Echtzeitbetriebsysteme

Echtzeitbetrieb ist der Betrieb eines Rechnersystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zufälligen, zeitlichen Verteilung oder zu bestimmten Zeitpunkten auftreten. (DIN44300)

Ein Echtzeitbetriebssystem oder auch RTOS (real-time operating system) ist ein Betriebssystem mit zusätzlichen Echtzeit-Funktionen für die Einhaltung von Zeitbedingungen und die Vorhersagbarkeit des Prozessverhaltens. (Wikipedia)

Ein Echtzeitbetriebssystem muss mehrere Eigenschaften erfüllen:

- Rechtzeitigkeit - Es muss zur richtigen Zeit reagieren
- Gleichzeitigkeit - Es muss auf mehrere Dinge gleichzeitig reagieren können
- Verlässlichkeit - Es muss zuverlässig und sicher sein
- Vorhersagbarkeit - Alle Reaktionen müssen planbar sein.

Ein Echtzeitbetriebssystem plant Ereignisse, sogenannte Tasks. Es gibt verschiedene Klassen von Tasks:

- periodische Tasks

  Ereignisse, die in einem festen Zeitintervall immer wieder auftreten. Zum Beispiel das Planen des Kurses.

- Aperiodische Tasks

  Ereignisse, die plötzlich auftreten (z.B. eingehender Befehl)

Dabei gibt es mehrere Planungsalgorithmen, falls zwei Tasks zur selben Zeit auftreten (Scheduler). Der einfachste ist ein prioritätenbasierter Scheduler, bei dem jeder Task eine Priorität zugeordnet bekommt. Der Task mit der höchsten Priorität wird zuerst ausgeführt.

Bei manchen Echtzeitbetriebssystemen sind Tasks unterbrechbar. Wird zum Beispiel ein Task mit niedriger Priorität ausgeführt, und ein höherer tritt auf, wird der Task mit der niedrigen Priorität in seiner Ausführung unterbrochen und der höher-priorisierte Task wird ausgeführt.

MicroC/OSII

MicroC/OSII (kurz ucosII) ist ein Echtzeitbetriebssystem für Mikroprozessoren. Der Code ist in C geschrieben und veröffentlicht. UcosII bietet prioritätenbasiertes Scheduling für bis zu 255 Tasks und viele weitere Eigenschaften eines Echtzeitbetriebsystems. UcosII ist für Universitäten kostenlos verfügbar.

6.3.5 Peripheriekontrolle

Pulsweitenmodulierte Signale

Bei der Pulsweitenmodulation (PWM) wechselt die Spannung zwischen zwei Werten in schneller Abfolge. Dies hat zur Folge, dass ein Signal das Regelmäßig zwischen Null Volt und zwölf Volt wechselt im Mittel als sechs Volt benutzt werden kann. Ist der Zeitraum, in de mdas Signal auf zwölf Volt ist länger als der Zeitraum, in dem das Signal auf Null Volt ist, verschiebt sich der Mittelwert. Das Verfahren wird zum Ansteuern von Motoren verwendet, um diese mit unterschiedlicher Drehzahl zu betreiben (6.1).

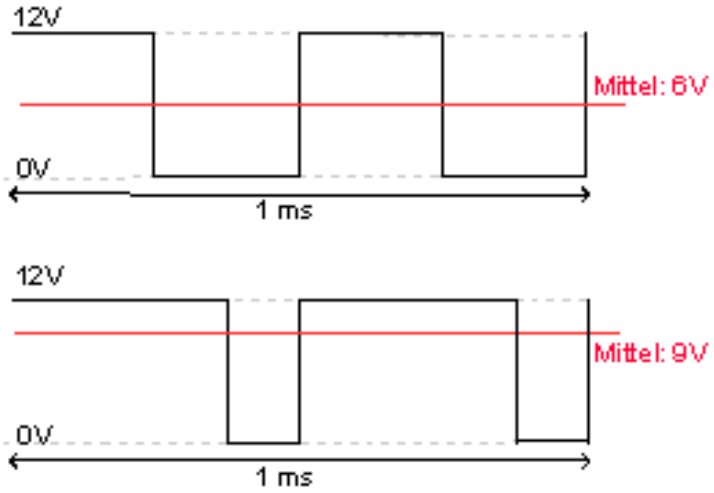


Abbildung 6.1: PWM Signale zur Motorsteuerung

Im Modellbau werden PWM Signale bei der Funkübertragung benutzt, allerdings anders als Motorsteuerungssignale. Das Signal ist für 0,8-1,2 ms auf high (i.d.R. 3,3V), und fällt dann auf low ab. Ein solches Signal wird alle 20 ms wiederholt. Die Zeit, die das Signal auf high ist wird durch den Ausschlag des Knüppels auf der Fernsteuerung bestimmt. 0,8ms bedeutet -100%, 1ms Nullstellung und 1,2ms 100% Ausschlag (6.2). (Die genauen Zeiten variieren bei verschiedenen Herstellern).

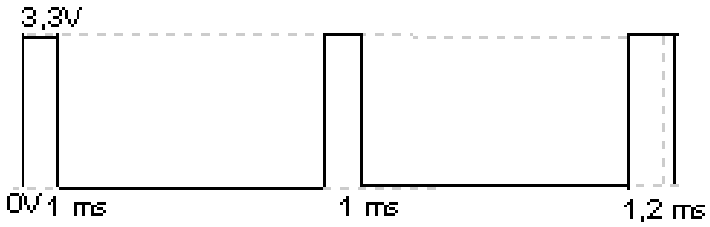


Abbildung 6.2: PWM Signale im Modellbau

Serial Peripheral Interface (SPI)

Das Serial Peripheral Interface ist ein Übertragungsprotokoll zwischen digitalen Schaltkreisen. Es ist ein synchroner serieller Datenbus, der von Motorola entwickelt wurde. Es ist ein Master-Slave Protokoll, bei dem genau ein Master mit beliebig vielen Slaves verbunden ist. Der Bus beinhaltet drei Leitungen, an den alle Teilnehmer parallel angeschlossen sind: ein vom Master generiertes serielles Clock-Signal, ein „Master Out Slave In“ Signal (MOSI) und ein „Master In Slave Out“ Signal (MISO).

Weiterhin wird jeder Slave mit einer Slave Select Leitung mit dem Master verbunden. Ein Slave sendet nur Daten auf den Bus, wenn er vom Master über das Slave Select Signal aktiviert wurde.

Das SPI Protokoll erlaubt Taktfrequenzen im Khz und Mhz Bereich.

Serielle Schnittstelle (EIA-232)

Die EIA-232 ist eine asynchrone serielle Kommunikation. Es gibt kein Taktsignal, die Synchronisation geschieht über die Datenleitung. Beide Teilnehmer müssen auf eine feste Geschwindigkeit eingestellt sein. Es gibt vier Leitungen: zwei Datenleitungen Transmit Data (TxD) und Read Data (RxD) sowie zwei Steuerleitung Ready to Send (RTS) und Clear To Send (CTS). Es gibt noch Weitere Steuersignale, die aber nur für analoge Modems gebraucht werden.

Es sind Geschwindigkeiten zwischen 9600 Bits/sec und 115200 Bits/sec in festen Abstufungen erlaubt.

6.4 Hardwareinterface der Aktoren

Alle in Hardware realisierten Elemente wurden über den Avalon-Slave Bus angeschlossen. Über diesen Bus kann die auf dem Nios Prozessor laufende Software über spezielle Befehle direkt auf die Hardware zugreifen.

In der Hardware selbst ist hierzu die Implementierung eines Adress- sowie Datenbusses und der zugehörigen Signale (chipselect, read,write) nötig. Die Auswertung des Adressbusses bleibt der Hardware überlassen. Die von der Software gesendeten/angeforderten Daten laufen über den Datenbus.

Das Boot verfügt über vier verschiedene Hauptaktoren, die über verschiedene Techniken gesteuert werden müssen:

6.4.1 Antrieb

Der Antrieb wird über pulsweitenmodulierte Signale gesteuert. Es kann die Richtung sowie die Geschwindigkeit gesteuert werden. Über die Länge der Pulse der PWM Signale wird die Drehgeschwindigkeit des

Motors bestimmt. Für die Software wird einfaches Interface zur Verfügung gestellt, in dem der Antrieb mit Werten von -100% bis +100% Leistung angesteuert wird. Die Werte werden dann intern in die PWM Signale umgewandelt.

6.4.2 Ruder

Die Servos der Ruder müssen über Modelbau-PWM Signale angesteuert werden. Der maximale Stellwinkel der Ruder beträgt baubedingt 35° in jede Richtung. Über verschieden lange PWM Signale kann der Servo in jeden beliebigen Drehwinkel geschaltet werden. Für die Software wird ein einfaches Interface zur Verfügung gestellt, in der jedes Ruder mit einem Winkel von -35° bis +35° gesetzt werden kann.

6.4.3 Tauchtanks

Die Tauchtanks werden je über einen Motor gefüllt bzw. geleert. Diese Motoren werden wie der Antriebsmotor mit einem PWM Signal angesteuert. Es müssen weiterhin Endabschalter berücksichtigt werden, die signalisieren, dass der Tauchtank vollständig gefüllt bzw. geleert ist. Der Software wird ein Interface zur Verfügung gestellt, in dem jeder Tauchtank mit einem Wert von -100% bis +100% Leistung angesteuert werden kann. Es können weiterhin die Signale der Endabschalter ausgelesen werden.

6.4.4 Bugstrahlruder

Die Bugstrahlruder werden über je einen Ausgang geschaltet. Es darf nur eines der Bugstrahlruder aufgrund der hohen Stromaufnahme zur Zeit aktiv sein. Die Bugstrahlruder dürfen aus dem selben Grund nicht länger als acht Sekunden eingeschaltet sein, bevor sie für vier Sekunden abgeschaltet sein müssen. Die Einhaltung dieser Grenzen wird von der Hardware übernommen. Für die Software wird ein Interface zur Verfügung gestellt, das die Bugstrahlruder in die Modi Links/Rechts/Aus schalten kann.

6.4.5 Sicherheit

Die Aktorensteuerung beinhaltet zwei Sicherheitlevel: Zum einen bietet sie ein Interface für die Software, die das System in vier unterschiedliche Stufen schalten kann.

- Normaler Modus:

  Alle Aktoren können betrieben werden.

- Sicherheitsmodus:

  Hauptantrieb und Bugstrahlruder werden abgeschaltet.

- Fail-Safe Modus:

  Wie Sicherheitsmodus , zusätzlich werden die Tauchtanks vollständig geleert.

- Reset:

  Das gesamte System wird neu gestartet (Hardware und Software)

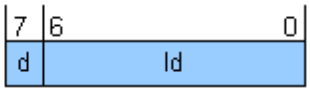
Als zweites bietet sie eine Ausfallerkennungsfunktion: Wird über zwei Sekunden der Sicherheitsstufe nicht aktualisiert, schaltet das System automatisch in den Sicherheitsmodus, nach zehn Sekunden in den Fail-Safe Modus.

6.4.6 Registerbeschreibung

Name	Modus	Größe	Adresse	Default
Rudder1	RW	8	0x00	0
Rudder2	RW	8	0x01	0
Rudder3	RW	8	0x02	0
Rudder4	RW	8	0x03	0
CamServo	RW	8	0x04	0
DriveLoad	RW	8	0x08	0
DriveSpeed	R	8	0x09	???
FrontTankLoad	RW	8	0x10	0
FrontTankLevel	R	16	0x11	???
TankSwitches	R	4	0x29	???
RearTankLoad	RW	8	0x18	0
RearTankLevel	R	16	0x19	???
Thruster	R	4	0x20	0
Security	RW	4	0x28	0

- RudderX/CamServo Register

  Hier wird das Ruder gesetzt und gelesen. Das oberste Bit setzt die Richtung des Ausschlags, die restlichen sieben Bit die Intensität von 0% bis 100%.

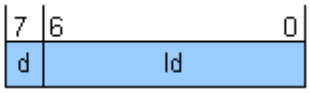
  

  d: Richtung

  ld: Intensität

- DriveLoad Register

  Die Leistung des Hauptantriebs. Das oberste Bit bestimmt die Drehrichtung, die restliche sieben Bit die Leistung von 0% bis 100%.

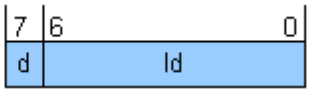
  d: Richtung

  ld: Leistung

- Drive Speed Register

  Enthält die aktuelle Drehzahl als Umdrehung pro Sekunde.

- XXXXtankLoad Register

  Die Leistung der Tauchtanks. Das oberste Bit bestimmt dir Richtung, die restlichen sieben Bit die Füllgeschwindigkeit von 0% bis 100%.

  d: Richtung

  ld: Leistung

- XXXXTankLevel Register

  Der aktuelle Füllstand der Tauchtanks in Prozent, ungenau.

- TankSwitches Register

  Die Schalter der Tauchtanks

  fE Vorderer Tank leer

  fF Vorderer Tank voll

  rE Hinterer Tank leer

  rF Hinterer Tank voll

- Thruster Register

  Informationen über den Zustand der Bugstrahlruder

  R Reserviert

  en Bugstrahlruder ist aktiv/Bugstrahlruder aktivieren

  t Das aktive Bugstrahlruder

  d Zeigt, ob das Bugstrahlruder gerade eine Sicherheitspause macht.(Nur lesen)

- Sicherheit Register

  R Reserviert

  s Sicherheitsmodus

  f FailSafe Modus

  r Reset Modus

6.4.7 Softwarebeschreibung

- unsigned int Environment_getFrontTankLevel();

  Liefert den aktuellen Füllstand des vorderen Tanks in Prozent (ungenau).

- int Environment_getFrontTankLoad();

  Liefert die aktuelle Ansteuerung des vorderen Tauchtanks von -100% bis +100%.

- void Environment_setFrontTankTarget(int load);

  Setzt die neue Ansteuerung des vorderen Tauchtanks von -100% bis +100%

- unsigned int Environment_getRearTankLevel();

  Liefert den aktuellen Füllstand des hinteren Tanks in Prozent (ungenau).

- int Environment_getRearTankLoad();

  Liefert die aktuelle Ansteuerung des hinteren Tauchtanks von -100% bis +100%.

- void Environment_setRearTankTarget(int load);

  Setzt die neue Ansteuerung des hinteren Tauchtanks von -100% bis +100%

- unsigned int Environment_getDriveRps();
  Liefert die aktuelle Drehzahl des Hauptantriebs in Umdrehungen pro Sekunde
- int Environment_getDriveLoad();
  Liefert die aktuelle Ansteuerung des Hauptantriebs von -100% bis +100%
- void Environment_setDriveLoad(int load);
  Setzt die neue Ansteuerung des Hauptantriebs von -100% bis +100%
- int Environment_getCamArc();
  Liefert den aktuellen Stellwinkel der Kamera in Grad
- void Environment_setCamArc(int arc);
  Setzt den neuen Stellwinkel der Kamera in Grad von -90° bis +90°
- int Environment_getTLRudderArc();
  Liefert den aktuellen Winkel des oberen Backbordruders in Grad.
- int Environment_getTRRudderArc();
  Liefert den aktuellen Winkel des oberen Steuerbordruders in Grad.
- int Environment_getBLRudderArc();
  Liefert den aktuellen Winkel des unteren Backbordruders in Grad.
- int Environment_getBRRudderArc();
  Liefert den aktuellen Winkel des unteren Steuerbordruders in Grad.
- void Environment_setTLRudderArc(Environment *env,int arc);
  Setzt den neuen Winkel des oberen Backbordruders im Bereich von -25° bis +25°
- void Environment_setTRRudderArc(Environment *env,int arc);
  Setzt den neuen Winkel des oberen Steuerbordruders im Bereich von -25° bis +25°
- void Environment_setBLRudderArc(Environment *env,int arc);
  Setzt den neuen Winkel des unteren Backbordruders im Bereich von -25° bis +25°
- void Environment_setBRRudderArc(Environment *env,int arc);
  Setzt den neuen Winkel des unteren Steuerbordruders im Bereich von -25° bis +25°
- void Environment_enableLeftBowThruster();
  Aktiviert das rechte Bugstrahlruder. Das linke wird gegebenenfalls deaktiviert.
- void Environment_enableRightBowThruster();
  Aktiviert das linke Bugstrahlruder. Das rechte wird gegebenenfalls deaktiviert.
- void Environment_disableBowThruster();
  Beide Bugstrahlruder werden deaktiviert.

6.5 Software

6.5.1 Softwarearchitektur

Die Softwarearchitektur ist in mehrere Bereiche gegliedert:

- System

  Systemdienste und Sicherheitstools.

- Navigation

  Berechnen von Position und Lage anhand von Sensorwerten.

- Regelung

  Steuerung der Aktoren auf Basis von Navigation.

- Kommunikation

  Entgegennehmen von Befehlen und Senden von Statusinformationen.

- Umwelterfassung

  Das Speichern von Bildern und Positionen zur anschließenden Analyse.

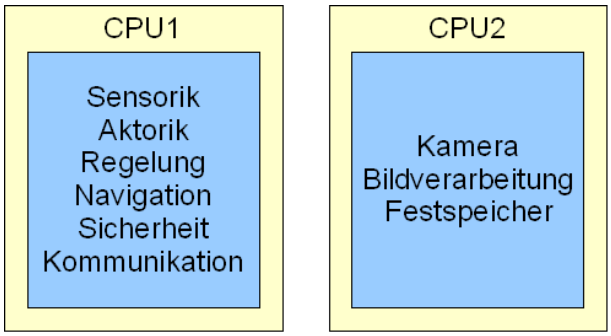
- Festspeicher

  Speichern von Bildern und Logdateien.

Jeder dieser Bereiche hat verschiedene Tasks, die periodisch aktiviert werden. Die höchste Priorität hat der Sicherheitsmanager, darunter befindet sich die eingehende Kommunikation, dann die Regelung, die Navigation, dann ausgehende Kommunikation.

Da mit der Zeit die Rechenleistung der verschiedenen Aufgaben angestiegen ist, wurden zum Ende des Projektes die verschiedenen Aufgaben auf zwei Prozessoren aufgeteilt. Die Software auf den beiden Prozessoren kommunizieren über Mailboxen.

Auf den zweiten Prozessor wurden zeitintensive, aber nicht sicherheitskritische Aufgaben ausgelagert. Hier werden die Tasks von Kameraerfassung, Bildverarbeitung und Festspeicheransteuerung ausgeführt.



System

Das System besteht aus zwei Programmen:

- SecurityManager

  Überwacht den Zustand des Systems. Benutzt den Sicherheitsmodus der Aktorik. Überwacht Umweltwerte und Kommunikation, und versetzt das System bei Bedarf ein einen der Sicherheitsmodi.

- Command Dispatcher

  Dekodiert Befehle, die von den verschiedenen Kommunikationsmodulen kommen, und reagiert entsprechend. Setzt Variablen für die Regelung, setzt Variablen über den Kommunikationsstatus.

Navigation

Die Navigation besteht aus einem Prozess, der periodisch Sensorwerte auswertet und daraus Informationen für die Regelung bereitstellt. Genauere Informationen sind unter dem Hauptabschnitt Navigation zu finden.

Regelung

Die Regelung besteht aus einem Prozess, der periodisch Navigationswerte auswertet und aufgrund dieser Informationen die Aktorik steuert. Genauere Informationen sind unter dem Hauptabschnitt Regelung zu finden.

Kommunikation

- 27MhzProzess

  Der Prozess überwacht den Eingangspuffer der 27Mhz Kommunikation. Eingehende Befehle werden an den Command Dispatcher übergeben. Sendet außerdem abwechselnd die wichtigsten Umweltwerte.

- 433Mhz Receiver

  der Empfänger lauscht auf der Eingangsleitung der 433Mhz Kommunikation. Ankommende Befehle werden an den CommandDispatcher übergeben. Über diesen Kommunikationsweg werden weniger zeit- und sicherheitskritische Daten übertragen.

- 433Mhz Sender

  Puffert zu sendende Daten zwischen, um diese dann periodisch gesteuert zu übertragen. Dies verhindert, dass andere Tasks auf die Kommunikation warten müssen und sich eventuell gegenseitig blockieren. Der Prozess hat eine geringe Priorität, da sicherheitskritische Umweltwerte über die 27Mhz Leitung übertragen werden.

Umwelterfassung

Die Umwelterfassung besteht aus einem Prozess, der periodisch Kamerabilder in den Speicher überträgt, diese Bilder weiterverarbeitet und anschließend auf dem Festspeicher ablegt. Genauere Informationen sind unter dem Hauptabschnitt Kamera zu finden.

Festspeicheransteuerung

Neben Bildern werden auch Systeminformationen auf dem Festspeicher abgelegt. Dafür ist der Prozess Syslog zuständig. Über die Funktion syslog können an jedem Punkt der Software Informationen zum Schreiben des Syslogs übergeben werden. Diese Informationen werden zunächst im RAM gepuffert, um dann vom Syslog Prozess periodisch auf den Festspeicher geschrieben zu werden. Informationen zur Implementierung der Festspeicheransteuerung sind unter dem Hauptabschnitt SD-Karte zu finden.

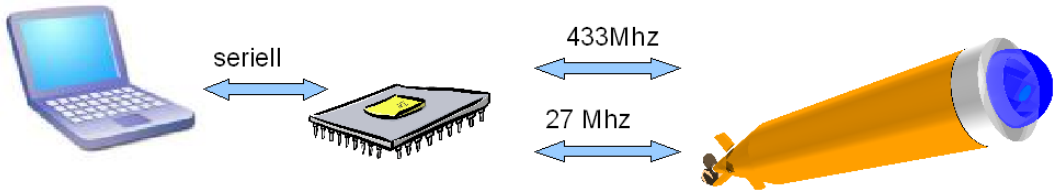
Kapitel 7

Kommunikation

7.1 Motivation

Gerade in der Entwicklungs- und Testphase, aber auch im späteren autonomen Betrieb ist es wichtig, mit dem getauchten Uboot zu kommunizieren. Dabei werden Steuersignale zum Boot und Umgebungswerte vom Boot übertragen. In Wasser ist die Dämpfung einer elektromagnetischen Welle etwa proportional zum Quadrat ihrer Frequenz. Höhere Frequenzen jedoch erlauben höheren Datendurchsatz.

7.2 Einleitung



Es wurde eine duale Kommunikation entworfen, die über verschiedene Frequenzen funkt. Steuersignale und wichtige Sensorwerte werden mit 27Mhz Frequenz übertragen, der Datendurchsatz ist sehr gering, dafür ist die Reichweite im Wasser auch für Tauchfahrten geeignet. Größere Datenmengen werden über 433Mhz übertragen, dies funktioniert nur im aufgetauchten Zustand bzw. bis zu einer geringen Wasser tiefe (wenige Zentimeter).

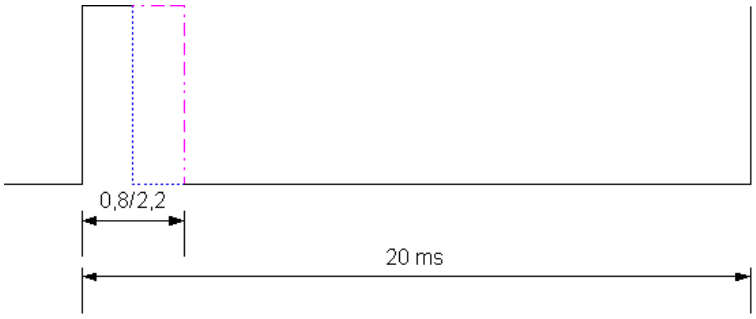
Für die 433Mhz Kommunikation wurden „Easy Radio“ Module verwendet, das sind fertige Bausteine, die eine UART Schnittstelle sowie integrierte CRC Funktionen bieten. Es werden Transreceiver eingesetzt, diese können sowohl senden als auch empfangen. Der Datendurchsatz der Module ist je nach Verbindungsqualität bis zu 19200 Baud.

Im 27Mhz Bereich gibt es keine fertigen Lösungen, es müssen auch gesetzliche Bestimmungen beachtet und eingehalten werden. Aus diesem Grund wurde eine Lösung entworfen, die als Kern eine 27Mhz Modellbaufernsteuerung beinhaltet, deren Ein- bzw. Ausgänge digital angesteuert bzw. ausgewertet werden.

7.3 27Mhz Kommunikation

Die 27Mhz Modellbaufernsteuerung, die den Kern des entwickelten Systems darstellt, verfügt über zwei Kanäle. Auf jedem Kanal werden Steuersignale als PWM Signal übertragen. Um innerhalb der gesetzlichen Bestimmungen zu bleiben dürfen die Signale nicht grundlegend verändert werden.

Also müssen die zu übertragenden Daten als PWM Signal dargestellt werden. Um eine bestimmte Anzahl von Bits pro Puls zu übertragen müssen alle möglichen Kombinationen dieser Bits als Pulslänge dargestellt werden.



Beispiel:

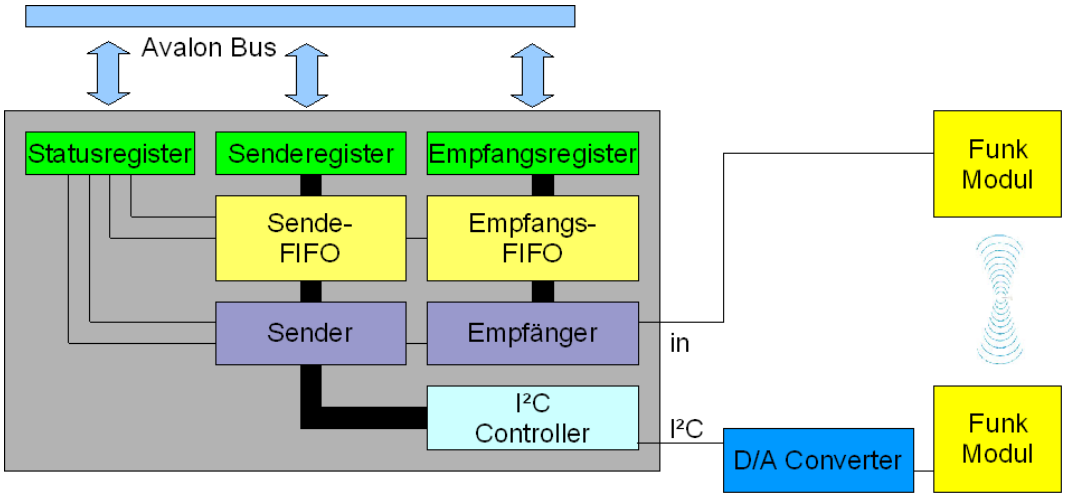
Idle	0,8ms
00	1,2ms
01	1,6ms
10	2,0ms
11	2,2ms

Der erste Ansatz sollte vier Bit in einem Puls übertragen, es müssen 16 verschiedene Pulslängen unterschieden werden. Dies zeigte sich in der Praxis als nicht möglich, da die Signale bei längerem Übertragungsweg durchs Wasser „verwischen“. Auch drei Bit (neun Pulslängen) konnten nicht mit ausreichender Genauigkeit unterschieden werden, so das am Ende nur noch zwei Bit pro Puls (vier Pulslängen) unterschieden wurden.

Bei einem Intervall von 20 ms und einer Übertragung von vier Bit pro Intervall (je zwei Bit pro Kanal) ergibt sich eine Transferrate von 200bit/Sekunde. Für die Übertragung werden 20bit Wörter benutzt, diese bieten die Möglichkeit, 16 Bit Werte plus vier Bit Identifizierung zu übertragen. Bei der Übertragung werden weitere vier Bit pro Wort für eine CRC Prüfsumme verwendet. Um den Anfang eines neuen Wortes zu erkennen muss für ein Intervall ein „Leerlauf“ Signal übertragen werden. Die tatsächliche Übertragung ist also 28 Bit pro Wort. Es können also nur 7,1 Wörter pro Sekunde übertragen werden.

Der Eingang der 27Mhz Funkfernsteuerung ist ein analoger Eingang, es werden die Bereiche von ein bis vier Volt in verschieden lange Pulse umgewandelt.

7.3.1 Architektur



Die Hardwarearchitektur gliedert sich in drei Teile:

- Statusregister beinhalten Informationen über den Zustand des Moduls, ob es eingeschaltet ist, wie viele Wörter empfangen wurden, wie viele Wörter darauf warten gesendet zu werden, wie viele Fehler bei der Übertragung aufgetreten sind, wie viele Wörter insgesamt übertragen wurden.

- Der Sender sendet Wörter, falls solche in der Sende-FIFO bereit sind. Ist ein Wort vorhanden, wird es aus der FIFO genommen. Es wird zunächst eine CRC Prüfsumme über das Wort berechnet. Dann werden alle 20 ms auf jedem Kanal je zwei Bit übertragen. Dafür werden über ein I2C Bus die Ausgänge eines externen Digital-Analog-Wandlers (D/A Wandler) geschaltet. Die Ausgänge des D/A Wandlers sind mit dem Eingang der 27Mhz Funkfernsteuerung verbunden. Am Ende des Wortes wird die vier Bit CRC Prüfsumme übertragen. Anschließend wird ein einen Puls langes Leerlaufsignal gesendet, das das Ende eines Wortes bzw. den Beginn eines neuen Wortes markiert. Soll kein

neues Wort übertragen werden, werden Leerlaufsignale gesendet bis ein neues Wort übertragen werden soll.

- Der Empfänger empfängt Wörter, sobald sich das Eingangssignal vom Leerlauf Signal unterscheidet. Wenn auf beiden Kanälen das PWM Signal beendet ist, wird die Länge der Signale mit gespeicherten Längen verglichen und das PWM Signal wieder in eine Bitfolge zurückverwandelt. Kann ein PWM Signal keiner Bitfolge zugeordnet werden, wird das Wort als fehlerhaft markiert und nicht in die Empfänger FIFO geschoben, stattdessen wird ein Fehlerzähler erhöht. Am Ende wird die empfangene CRC Prüfsumme auf das Wort angewandt. Tritt hier ein Fehler auf, wurde eines der PWM Signale fehlerhaft interpretiert und das Wort wird wieder als fehlerhaft behandelt, jetzt wird zusätzlich zu dem Fehlerzähler auch ein CRC Fehlerzähler erhöht.

7.3.2 Kalibrierungsmodus

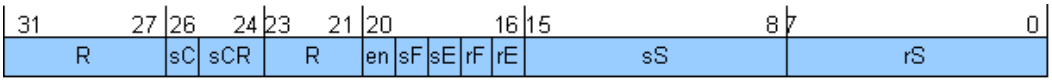
Neben dem Standardmodus gibt es noch einen Kalibrierungsmodus. Um diesen zu aktivieren, muss im Control Register sendCalibrationEnable Bit auf eins gesetzt werden, das sendCalibrationChannel auf den entsprechenden Wert des Kanals für 00, 01,10 und 11. Der Sender sendet dann die PWM Frequenz für dieses Signal durchgehend, ohne CRC und ohne Leerlaufsignale. Am Empfänger können dann die Register currentSignalA und currentSignalB ausgelesen werden. Anschließend können die Register limitXXLow und LimitXXHigh korrigiert werden, wobei XX den entsprechenden Kanal widerspiegelt.

Anschließend muss der Sender durch das Setzen des sendCalibrationEnable Bits aus 0 in den normalen Betriebsmodus gebracht werden.

7.3.3 Registerbeschreibung

Name	Modus	Größe	Adresse	Standartwert
Status	R	32	0x00	0x50000
Control	W	8	0x00	
Receive	R	20	0x01	0
Send	W	20	0x01	
receivedWords	R	32	0x02	0
errorCount	R	32	0x03	0
crcCount	R	32	0x04	0
currentSignalChannelA	R	10	0x05	???
currentSignalChannelB	R	10	0x06	???
timerDivisor	RW	16	0x07	2000
limit00Low	RW	10	0x08	113
limit00High	RW	10	0x09	153
limit01Low	RW	10	0x0A	157
limit01High	RW	10	0x0B	197
limit10Low	RW	10	0x0C	216
limit10High	RW	10	0x0D	256
limit11Low	RW	10	0x0E	268
limit11High	RW	10	0x0F	308
currentSenderWord	R	24	0x10	0
currentReceiverWord	R	24	0x11	0
signalIntervall	R	16	0x12	???

- Statusregister:



Das Statusregister kombiniert alle Informationen über den Zustand des Systems. Es beinhaltet Informationen über den Betriebsmodus und über den Zustand der FIFO's.

R: Reserverd

sC: sendCalibartionEnable

sCR: sendCalibrationRegister

en: Receiver Enabled

sF: Sender Fifo Full

sE: Sender Fifo Empty

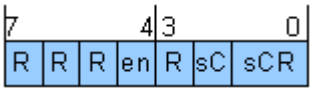
rF: Receiver Fifo Full

rE: Receiver Fifo Empty

sS: Sender Fifo Size

rS: Receiver Fifo Size

- Control Register:



Das Controlregister bündelt alle Einstellmöglichkeiten des Systems.

R: Reserverd

sC: sendCalibartionEnable

sCR: sendCalibrationRegister

en: Receiver Enable

- Receive Register:

  Wird das ReceiveRegister ausgelesen, wird das nächste Wort aus der LeseFIFO geholt, die FIFO wird debei um ein Wort kleiner. Ist die FIFO leer, wird das zuletzt empfangene Wort zurückgegeben.

- Send Register:

  Wird das SendRegister gesetzt, wird das gesetzte Wort in an die Sende FIFO angehängt. Ist die FIFO bereits voll, wird das Wort verworfen.

- ReceivedWords Register:

  beinhaltet die Anzahl der seit dem Einschalten empfangenen Wörter, auch fehlerhafte Wörter.

- ErrorCount Register:

  beinhaltet die Anzahl der seit dem Einschalten als fehlerhaft erkannte Wörter.

- CrcCount Register:

  Beinhaltet die Anzahl der seit dem Einschalten durch CRC erkannten fehlerhaften Wörter.

- CurrentSignalA Register:

  Beinhaltet die Länge des zuletzt auf Kanal A empfangenen PWM Signals. Zum Debuggen und Einstellen des Systems.

- CurrentSignalB Register:

  Beinhaltet die Länge des zuletzt auf Kanal B empfangenen PWM Signals. Zum Debuggen und Einstellen des Systems.

- TimeDivisor Register:

  Zum Einstellen des 20ms Timers. Durch die verschiedenen Frequenzen der Quarze sind die genauen Wiederholraten nicht immer 20ms. Das Register wird als ms Wert mit 2 Kommastellen eingestellt, für 20ms muss das Register auf 2000 gesetzt werden.

- LimitXXLow Register:

  untere Grenze, in der ein Signal als der Wert XX erkannt wird.

- LimitXXHigh Register:

  obere Grenze, in der ein Signal als der Wert XX erkannt wird.

- CurrentSenderWord Register:

  Beinhaltet die Teile des Wortes, das der Sender gerade überträgt, die noch nicht übertragen sind. Nur zu Debug Zwecken.

- CurrentRecieverWord Register:

  Beinhaltet die bereits empfangenen Bits des zu erkennenden Wortes. Nur zu Debug Zwecken.

- SignalIntervall Register:

  Beinhaltet den Abstand zwischen zwei Signalen, spiegelt den tatsächlichen 20ms Intervall des Senders wieder.

7.3.4 Softwareinterface

Vordefinierte Konstanten:
Konstanten zum Vergleichen des Status Registers

- RADIO27_RECEIVER_EMPTY_MSK
- RADIO27_RECEIVER_FULL_MSK
- RADIO27_SENDER_EMPTY_MSK
- RADIO27_SENDER_FULL_MSK
- RADIO27_RECEIVER_ENABLE_MSK
- RADIO27_SENDER_CALIBRATE_MSK

Konstanten zum Setzen des Kontrollregisters

- RADIO27_RECEIVER_ENABLE_BIT

Konstanten zum erreichen der Register

- RADIO27_STATUS_REG
- RADIO27_CONROL_REG
- RADIO27_SEND_REG
- RADIO27_RECEIVE_REG
- RADIO27_WORDCOUNT_REG
- RADIO27_ERRORCOUNT_REG
- RADIO27_CRC_ERRORCOUNT_REG

- RADIO27_CUR_LEN_A_REG
- RADIO27_CUR_LEN_B_REG
- RADIO27_TIME_DIVISOR
- RADIO27_00LOW_REG
- RADIO27_00HIGH_REG
- RADIO27_01LOW_REG
- RADIO27_01HIGH_REG
- RADIO27_10LOW_REG
- RADIO27_10HIGH_REG
- RADIO27_11LOW_REG
- RADIO27_11HIGH_REG
- RADIO27_SENDER_CURRENT_REG
- RADIO27_RECEIVER_CURRENT_REG
- RADIO27_RECEIVER_SIGNAL_INTERVALL_REG

Funktionen:

- unsigned long radio27_getStatus();
  liefert den Inhalt des Status Registers
- void radio27_setCotrol(unsigned long status);
  Setzt das Kontrollregister
- int radio27_receiverEmpty();
  Testet, ob der Empfangsuffer leer ist
- int radio27_receiverFull();
  Testet, ob der Empfangsuffer voll ist.
- int radio27_senderEmpty();
  Testet, ob der Sendepuffer leer ist.
- int radio27_senderFull();
  Testet, ob der Sendepuffer voll ist.
- int radio27_receiverEnabled();
  Testet, ob der Empfänger aktiviert ist.
- void radio27_enableReceiver();
  Aktiviert den Empfänger, andere Bits werden nicht verändert.
- void radio27_setChannelCalibration(int enable,int channel);
  Startet bei enable=1 den Kalibrierungsmodus auf dem übergebenen Kanal
- void radio27_send(unsigned long word);
  Fügt ein Wort an den Sendepuffer an

- unsigned long radio27_receive();
  Liest ein empfangenes Wort aus dem Empfangspuffer
- void radio27_set00ChannelConstrains(unsigned int calib);
  Setzt die neuen Grenzen für Kanal 00 (Grenzen= calib-20 bis calib+20)
- void radio27_set01ChannelConstrains(unsigned int calib);
  Setzt die neuen Grenzen für Kanal 01
- void radio27_set10ChannelConstrains(unsigned int calib);
  Setzt die neuen Grenzen für Kanal 10
- void radio27_set11ChannelConstrains(unsigned int calib);
  Setzt die neuen Grenzen für Kanal 11
- unsigned int radio27_getSendTimeDivisor();
  Liefert den aktuellen Wert des TimeDivisor Registers.
- void radio27_setSendTimeDivisor(unsigned int length);
  Setzt einen neuen Wert für das TimeDivisor Register.
- unsigned long radio27_getReceivedWords();
  Liefert den Inhalt des receivedWords Registers.
- unsigned long radio27_getReceivedErrors();
  Liefert den Inhalt des errorCount Registers.
- unsigned long radio27_getReceivedCrcErrors();
  Liefert den Inhalt des crcCount Registers.

7.4 PC Anschlussbox

Die PC Anschlussbox stellt ein einfaches Kommunikationsinterface für die Hostsoftware zur Verfügung und übernimmt die Aufteilung auf die beiden Kommunikationskanäle. Der Hostsoftware wird ein einfaches String-basiertes Frage-Antwort Protokoll zur Verfügung gestellt.

7.4.1 Hardware

Die Kommunikationsbox besteht aus einen kleinen FPGA mit den beiden Kommunikationsmodulen sowie einer RS232 Schnittstelle zur Kommunikation mit dem PC. Da Teile der 27 Mhz Fernsteuerung zwölf Volt Versorgungsspannung benötigen, ist die Versorgungsspannung der Box zwölf Volt. In der Box stellt für den FPGA sowie das 433 Mhz Modul ein Spannungsregler eine fünf Volt Spannungsversorgung bereit.

Ein MAX232 Konverter sorgt für die Umwandlung in PC-COM Port kompatible Signale. Der Digital Analog Wandler benötigt eine zwölf Volt Spannungsversorgung und eine fünf Volt Referenzspannung. Die acht analogen Ausgänge werden über ein I2C Protokoll auf einen prozentualen Wert zwischen Masse und Referenzspannung gesetzt. Zwei Ausgänge sind mit den Eingängen der 27Mhz Funkfernsteuerung verbunden. Der 27Mhz Empfängerbaustein wird mit fünf Volt versorgt, die Datensignale werden mit Eingängen des FPGA verbunden. Der 433Mhz Transreceiver wird ebenfalls mit fünf Volt versorgt, das Dateninterface ist mit dem FPGA verbunden.

7.4.2 Protokollbeschreibung

Mit der Box wird über einen RS232 Port mit der Geschwindigkeit 115200 Baud kommuniziert. Die Box versteht eine Reihe von Befehlen, auf die es unterschiedlich antwortet. Jede Kommunikation muss vom PC gestartet werden, die Box antwortet nur auf Anfragen. Die Befehle an die Box sind in der Regel zwei bzw. dreistellige Buchstabenkombinationen, optional mit einem Wert. Es gibt zwei Arten von Befehlen, Befehle zum manipulieren des Bootes und Befehle zum Erhalten von Informationen über den Zustand des Bootes.

Auf die Manipulationsbefehle antwortet die Box immer mit einer Wiederholung des Befehls und einem OK oder FAIL. Für jeden Informations-Abfrage Befehl ist das Antwortformat festgelegt.

Befehle zur Steuerung des Bootes

Die Befehle beginnen mit „R“, was für Remote Control steht. Die Befehle steuern das Boot direkt.

- RD (Wert)

  Setzen der Solltiefe in Meter

- RR (Wert)

  Setzen der Fahrtrichtung (rechts/links) in Prozent von -100% bis +100%

- RS (Wert)

  Setzen der Geschwindikeit von -100% (Rückwärts) bis +100%

- RB[L|R|0]

  Aktivieren/Deaktieren der Bugstrahruder

Systembefehle

Die Befehle beginnen mit „S“, was für System steht. Sie sind zur Manipulation des internen System des Bootes gedacht.

- ST (Wert)

  Setzen der Systemzeit (Format: Jahr Monat Tag Stunde Minute Sekunde) durch Leerzeichen getrennt.

- SG

  Abfragen der Systemzeit Format wie ST.

- SR

  Startet das System neu.

Befehle zum Steuern der Reglerparameter

Die Befehle beginnen mit „C“, was für Control steht. Sie sind zur Manipulation der Regler und Reglerparameter gedacht.

- CM[0|1]

  Gesamtregelung ein- bzw. ausschalten

- CS[0|1]

  Geschwindikgkeitsregelung ein- bzw. ausschalten

- CR[0|1]

  Regelung für Tiefenruder ein- bzw. ausschalten

- CB[0|1]

  Neigungsregler Betawinkel ein- bzw. ausschalten

- CT[0|1]

  Tauchtank Neigungsregler ein- bzw. ausschalten

- CD[0|1]

  Tauchtank Tiefenregler ein- bzw. ausschalten

- CG[0|1]

  Richtungsregler Gammawinkel ein- bzw. ausschalten

- CP[0|1]

  PID Regelungsmodus ein- bzw. ausschalten

- CC ID P I D

  Setzen eines PID Parameters für den Regler ID

Es gibt noch weitere Befehle, insbesondere für Testmodi der Intertialnavigation, die hier nicht aufgeführt werden.

Kapitel 8

SD-Karte

8.1 Einleitung

Zur dauerhaften Speicherung von Daten im Uboot ist es notwendig ein Festspeichermedium einzubinden. Hauptsächlich sollen Sensordaten zur späteren Weiterverarbeitung und Auswertung, sowie Kamerabilder gespeichert werden. Diese beiden Anwendungsfälle setzen eine möglichst hohe Zugriffsgeschwindigkeit voraus. Im Falle der Sensordaten ist es wichtig so viele Daten pro Zeiteinheit wie möglich speichern zu können, um eine breite und detaillierte Basis für nachfolgende Verarbeitungsschritte zur Verfügung zu haben. Die Speicherungsgeschwindigkeit der Bilder bestimmt maßgeblich die FPS (Frames per Second) und die daraus resultierende Bilddichte, bzw. die Möglichkeit aus den Einzelbildern einen Stream zu erstellen. Zur Auswahl standen mehrere Speichermedien. So z.B. Festplatten und diverse Kartentypen bis hin zum einfachen USB-Stick. Die Speicherung der Daten auf Festplatte und USB-Stick haben wir sehr schnell verworfen, da die dazu nötigen VHDL-Komponenten für ein IDE-Interface bzw. für eine USB-Anbindung nicht kostenlos zu bekommen sind. In der ersten Version haben wir die Daten auf eine Compactflashkarte gespeichert. Sehr schnell stellte sich heraus, dass die Zugriffsgeschwindigkeit nicht unseren Anforderungen genügen würde. Zwar ist es prizipell möglich sehr schnelle Zugriffe auf Compactflash zu realisieren, aber aufgrund der begrenzten Anzahl an Pinouts auf unserem FPGA-Board konnten wir nur die serielle Anbindung über SPI umsetzen. Daraufhin haben wir uns nach einem anderem Speichermedium umgeschaut, welches seriell angesprochen werden kann und uns trotzdem höhere Zugriffszeiten ermöglicht. Wir haben uns daraufhin für die Anbindung einer SD-Karte entschieden. Nach der Wahl des Speichermediums mussten wir uns für eine Speicherstruktur entscheiden. Wir haben sehr schnell davon abgesehen eine eigene von uns entwickelte Speicherstruktur zu implementieren. Dies wäre zwar in der Umsetzung einfacher und schneller zu realisieren gewesen, aber das Abrufen der Daten auf anderen Systemen hätte wieder eine von uns eigens darauf zugeschnittene Software verlangt. Uns war es aber vor allem wichtig einen möglichst leichten und plattformunabhängigen Zugriff auf die Daten zu ermöglichen. Dies konnte nur bedeuten das wir uns für ein standardisiertes Dateisystem entscheiden mussten.

8.2 SD-Karte und Anbindung

Der Vorteil der SD-Karten liegt vor allem darin, dass wir problemlos die Anbindung über die SPI-Schnittstelle realisieren konnten, welche dieser Kartentyp standardmäßig unterstützt. Bei der von uns verwendeten Karte handelt es sich um eine der Klasse 2 mit einer Kapazität von 1GB und einer maximalen physikalischen Zugriffsgeschwindigkeit, bei serieller Anbindung, von ca. 2MB pro Sekunde.

PIN	
1	Chip Select(CS)
2	CMD/DI
3	GND
4	VCC
5	CLK/SCLK
6	GND
7	DAT/DO
8	NC
9	NC

Die SD-Karte ist über einen USB-Kartenleser an das System angeschlossen. Dabei ist jegliche Logik zur USB-Schnittstelle aus dem Kartenleser entfernt worden, da nur die SPI-Signale CS(Chipselect), Clk,Datain und Dataout durchgeführt werden müssen und die Anbindung über USB vorher schon verworfen wurde. Somit dient der Kartenleser nur zum einfachen Wechseln der SD-Karte, denn prinzipiell hätten auch die nötigen SPI-Signale sowie sie Spannungsversorgung direkt auf der Karte angebracht werden können.

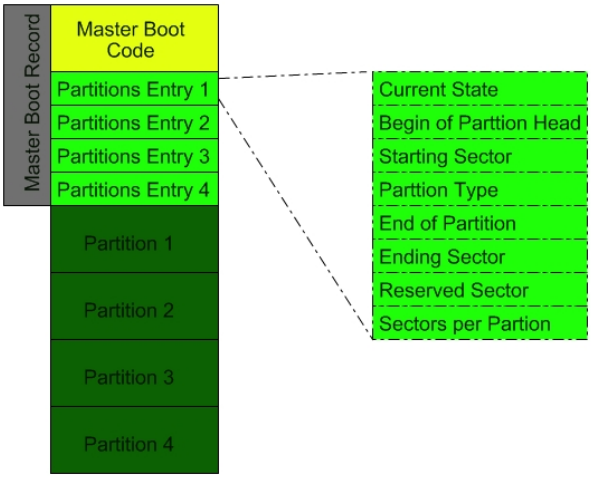
8.3 FAT 16 Datei-System

Wir haben uns aus folgenden Gründen für ein FAT 16 Dateisystem entscheiden:

- Es ist eines der weit verbreitesten Dateisysteme und bietet deshalb von fast jedem Betriebssystem (so z.B Linux, Windows)lesend sowie schreibend Zugriff auf die Daten. Sicherlich gibt es bessere Dateisysteme (so z.B. NTFS, Raiser) aber diese sind meistens für ein bestimmtes Betriebssystem entworfen worden.
- FAT 16 ist zudem das Standard-Dateisystem für Speicherkarten mit einer Kapazität unter 2 GB.
- Die Lebensdauer einer SD-Karte ist von der Anzahl der Speicherzugriffe abhängig und gerade in diesem Gesichtpunkt ist das FAT 16 gegenüber anderen Dateisystemen optimaler. Eine handelsübliche Karte ist für ca 10.000 Speicherzugriffe ausgelegt.
- Die Implementierung des FAT 16 Dateisystems ist einfacher zu realisieren im Vergleich zu neueren Dateisystemen.

8.3.1 Aufbau eines FAT-16

Die Abbildung 1 zeigt die Grundstruktur eines FAT 16 Dateisystems. Diese besteht immer aus dem Master Boot Record gefolgt von den 4 Partitionsbereichen.



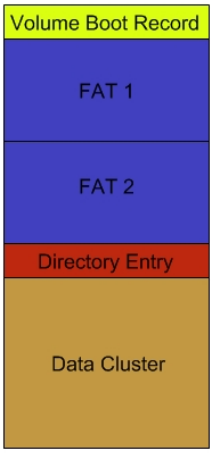
Master Boot Record

Der erste Bereich ist immer für den MBR (Master Boot Record) reserviert. Dieser besteht im ersten Teil aus einem ausführbaren x86-Maschinencode und im Hauptteil aus den 4 Partitionseinträgen. Diese enthalten im wesentlichen Informationen über den Startsektor und Endsektor, Anzahl der gesamten Sektoren, sowie die Beschreibung und den Status der jeweiligen Datenpartition. Bei einigen Formatierungen einer SD-Karte, kann diese auch direkt mit den 4 Partitionseinträgen beginnen.

Partition

Eine Partition besteht aus folgenden Bereichen:

1. Volumen Boot Record
2. Jeweils 1 Bereich pro Kopie der File Allocation Table. In der Regel sind dies 2 FAT Bereiche
3. Root Directoy Entry
4. Data Cluster



Im folgenden werden wir die einzelnen Bereiche genauer beschreiben.

Volumen Boot Record
Am Anfang jeder Partition befindet sich der Volumen Boot Record. Dieser enthält detaillierte Informationen zu der Partition. Die wichtigsten davon sind:

- Name der Partition
- Anzahl der FAT Kopien
- maximale Anzahl an Root Directory Entrys
- Anzahl der Bytes pro Sector
- Anzahl an Sektoren pro Cluster

Alle weitere Information sowie ihre genauen Adressen im Volumen Boot Record sind aus der folgenden Tabelle zu entnehmen.

Adresse	Beschreibung	Größe
00h	Sprunganweisung + NOP	3 Bytes
03h	OEM Name	8 Bytes
0Bh	Anzahl der Bytes pro Sektor auf der Festplatte	1 Word
0Dh	Die Anzahl der Sektoren pro Cluster	1 Byte
0Eh	Anzahl der reservierten Sektoren	1 Word
10h	Anzahl der Kopien der FAT, üblicherweise 2	1 Byte
11h	Anzahl der möglichen Einträge im Hauptverzeichnis: 512 bei FAT16	1 Word
13h	Anzahl der Sektoren in kleineren Partitionen	1 Word
15h	Hexadezimalwert F8 für Festplatten; F0 für Disketten	1 Byte
16h	Sectors Per FAT	1 Word
18h	Sectors Per Track	1 Word
1Ah	Number of Heads	1 Word
1Ch	Reservierte Sektoren am Beginn der Festplatte	1 Double Word
20h	Gesamtzahl der Sektoren in der Partition	1 Double Word
24h	Laufwerksnummer. Die erste Festplatte trägt 80h, die zweite 81h usw	1 Word
26h	Erweiterte Boot-Signatur (29h)	1 Byte
27h	Seriennummer des Datenträgers	1 Double Word
2Bh	Datenträgerbezeichnung	11 Bytes
36h	FAT Name (FAT16)	8 Bytes
3Eh	Executable Code	448 Bytes
1FEh	Signatur (55h AAh)	2 Bytes

Aus diesen Angabe lassen sich zum Beispiel die Startadressen der anderen Bereiche bestimmen.

- File Allocation Table = Volumen Boot Record Startsektor + Anzahl an Reservierten Sektoren(0eh)
- Root Directory Entry = File Allocation Table + Anzahl an FAT-Kopien(10h)*Sektoren pro FAT(16h)
- Data Cluster = Root Directory Entry + Anzahl der möglichen Root Directory Entrys(11h) * 32(Größe eines Root Directory Entrys / Anzahl an Bytes pro Sektor(0Bh)

Data Cluster
Im Data-Cluster-Bereich liegen die eigentlichen Daten zu Dateien und Verzeichnissen.Es gibt zwei Typen von Daten die in den Data Cluster Bereich abgelegt werden:

- Verzeichniseinträge
- Datenbereiche

Verzeichniseinträge
Diese 32 Byte großen Blöcke beschreiben jeweils eine Datei oder ein Verzeichnis. Die wichtigsten Angaben sind Datei- bzw Verzeichnisname und Endung, Größe, Start-Cluster des Datenbereichs sowie Attribute der Datei bzw. des Verzeichnisses. Alle weiteren Informationen, sowie die genauen Positionen im Verzeichniseintrag kann der folgenden Tabelle entnommen werden.

Adresse	Beschreibung	Größe
00h	Status der Partition: 0 wenn inaktiv, 80h wenn aktiv.	1 Byte
01h	Der Sektor, auf dem die Partition beginnt (Bits 0-5) und der Zylinder, auf dem die Partition beginnt (Bits 6-16).	1 Byte
02h	Zylinder, auf dem die Partition beginnt	1 Word
04h	Partitionstyp (1h-FAT12, 4h-FAT16<32Mb, 6h-FAT16>32Mb, Bh-FAT32)	1 Byte
05h	Der Kopf, auf dem die Partition endet.	1 Byte
06h	Der Sektor, auf dem die Partition endet (Bits 0- 5) und der Zylinder, auf dem die Partition endet (Bits 6-16).	1 Word
08h	Abstand zwischen MBR-Sektor und dem ersten Sektor der Partition in Sektoren	1 Double Word
0Ch	Gesamtzahl der Sektoren in der Partition	1 Double Word

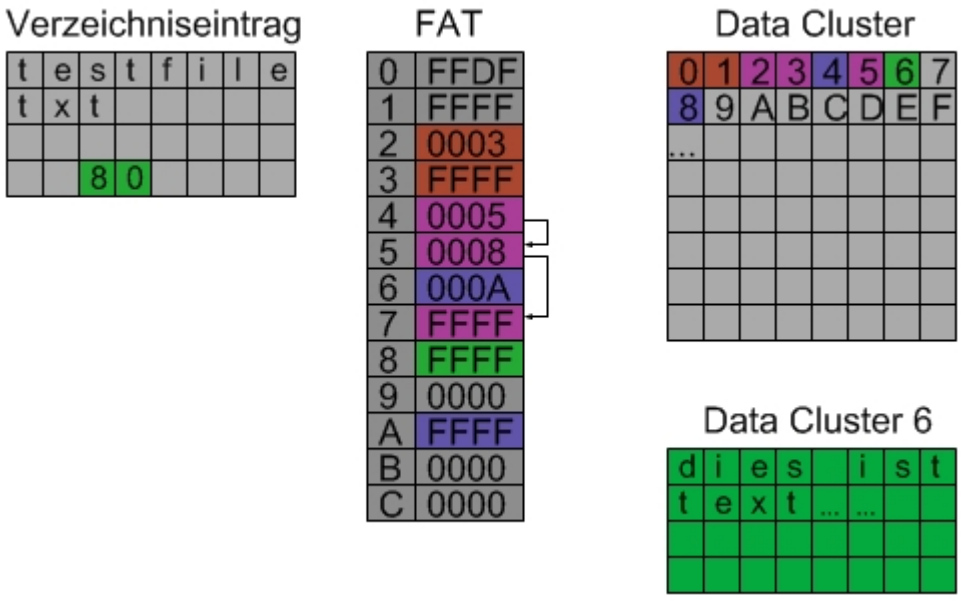
Root Directory Entry
Hier befinden sich die Verzeichniseinträge des Rootverzeichnisses der Partition. Zu allen weiteren Unterverzeichnissen werden die Verzeichniseinträge in Daten-Clustern abgelegt.

File Allocation Table
Die FAT ist eine Tabelle mit jeweils 2 Byte großen Einträgen. Daher stammt auch der Zusatz des FAT-System, da jeder FAT-Eintrag genau 16 Bit groß ist. Somit sind maximal 2^{16} also 65536 Einträge möglich. Die Datenbereiche von Dateien und die Verzeichniseinträge werden in einer geordneten Kette von Daten-Clustern abgelegt. Genau diese Cluster-Ketten werden im FAT beschrieben. Die Position des FAT-Eintrages bestimmt dabei den zugehörigen Daten-Cluster. Dem FAT-Eintrag an der Position A ist dabei der Daten Cluster an der Position A - 2 zugeordnet. Dieser Offset von -2 ergibt sich daraus, dass die beiden ersten Einträge im FAT immer reserviert sind, aber die Daten-Cluster immer bei Null beginnen. Der FAT-Eintrag selbst kann folgende Werte annehmen:

- 0000 : Es handelt sich um einen unbenutzten Cluster
- 1 bis Cluster-Anzahl : Der Verweis auf den darauf folgenden FAT-Eintrag und somit auf den darauf folgenden nächsten Cluster in der Kette
- FFF7 : Defekter Cluster
- FFF8h - FFFFh : Letzter Cluster einer Datei bzw. der Verzeichnisseinträge

Eine beschädigte bzw. unvollständige FAT-Tabelle hätte zur Folge, dass die einzelnen Datenbereiche nicht mehr richtig zusammen gesetzt werden könnten, bzw. dass diese falsch zusammen gesetzt werden. Aus diesem Grund existieren mehrere FAT pro Partition, in den meisten Fällen sind es zwei. Die FAT sind identisch und dienen als Sicherheitskopien zu einander.

Die folgende Grafik verdeutlicht den Zusammenhang zwischen den Verzeichniseinträgen, der FAT und den Data-Clustern.



Im Beispiel ist der Verzeichniseintrag der Datei „testfile.txt“ zu sehen. Die grün markierte Position verweist auf die FAT-Tabelle. In diesem Fall befindet sich der FAT-Eintrag an der Stelle 8 im FAT. An dem Wert = „FFFF“ an der Position 8 im FAT lässt sich erkennen das dieser der letzte FAT-Eintrag in der

Verweisungskette dieser Datei ist. Somit besteht unsere Datei „testfile.txt“ nur aus einem Cluster. Aufgrund des schon vorher erwähnten Offsets von -2 befinden sich die Daten zu der Datei an der Position 6 im Daten-Cluster-Bereich. Im Daten-Cluster stehen dann die Daten selbst,in unserem Beispiel ein kleiner Text. An den violett eingefärbten Feldern in der FAT-Tabelle ist zu erkennen wie mehrere Einträge auf ihr nächstes Glied in der Kette zeigen, in diesem Fall von Feld 4 auf 5 und von Feld 5 auf 7. Die dazu gehörigen Cluster sind dem Offset entsprechend wieder 2, 3 und 5.

8.3.2 Implementierung und Umsetzungsschwierigkeiten

Die Implementierung bestand in den größten Teilen daraus, zu der zu lesenden oder zu schreibenden Datei den jeweilig richtigen Verzeichniseintrag zu finden bzw. zu erzeugen um dann im nächsten Schritt den Daten-Cluster und den FAT zu manipulieren. Wichtig bei der Umsetzung war eine minimale Anzahl an Schreib- und Lesezugriffen zu gewährleisten um eine hohe Zugriffsgeschwindigkeit zu erreichen. Aktuelle Sektoren werden z.B. im Speicher behalten. Somit wird ein erneutes Schreiben desselben Sektors minimiert und meistens erst beim Zugriff auf einen neuen Sektor notwendig. Desweiteren wird die FAT-Tabelle beim Initialisieren der Karte komplett in einen Puffer geschrieben. In den weiteren Schritten wird immer auf dem Puffer gearbeitet und dieser erst dann zurückgeschrieben falls ein flush- oder ein close-Befehl einer Datei vorliegt. Dies verhindert auch, dass bei einem unvorhergesehenem Abbruch der Dateioperation, z.B. Abziehen der Karte, die FAT falsche Einträge bekommt. So haben wir von der ersten funktionsfähigen Version bis zur finalen Version einen Geschwindigkeitszuwachs vom Faktor 3 realisieren können. Bei den zur Verfügung stehenden Funktion haben wir versucht uns an den Standard von Dateimanipulationsfunktionen zu richten um dem Benutzer den Umgang zu erleichtern. Um Inkonsistenzen zu vermeiden war es notwendig Sicherheitsmechanismen einzubauen. So wird zur Laufzeit z.B. immer eine Liste der momentan manipulierten Dateien geführt um beispielsweise ein erneutes Öffnen, zum Schreiben einer schon geöffneten Datei, zu verhindern.

Die häufigsten Fehler bzw. Umsetzungsprobleme

- Bei den meisten Operationen die implementiert werden mussten, handelte es sich um Adressberechnungen. Bei der Beseitigung eines Problems hat sich oft die Berechnungart oder sogar die ganze Basisadresse eines Kontextes geändert was dann schwerwiegende Konsequenzen zur Folge hatte, die einem hin und wieder aber nicht immer sofort im vollen Umfang klar wurden.
- Nach jeder Operation musste sicher gestellt werden das nicht falsche Datenbereiche überschrieben wurden. So konnte z.B. ein Überschreiben des Master Boot Records erst bei dem nächsten FAT-Init auffallen und somit nicht mehr ganz nachvollzogen werden, welche Änderung diesen Fehler genau verursacht hat. Eine ständige Sicherung von getesteten Versionen ist selbst bei kleinsten Änderungen notwendig gewesen.
- Die Literatur zu diesem Thema ist zwar reichlichem Umfang vorhanden, aber oftmals werden in den verschiedenen Quellen dem selben Begriff verschiedene Bedeutungen zugewiesen. Das erschwerte das Verständnis für den Sachverhalt und führte daraus resultierend zu einigen falschen Implementierungen.
- Desweiteren ist die Literatur auch oftmals unvollständig oder ungenügend um das FAT zu implementieren. So wird z.B. der Offset von 2 der zwischen FAT-Postion und Daten-Clusterposition existiert oftmals nicht erwähnt. Alleine dieses Problem hat uns unnötig Zeit gekostet.

Dazu kommen natürlich noch die üblichen Fehler die im Umgang mit einer neuen Programmiersprache zustande kommen und insbesondere Flüchtigkeitsfehler, die bei den Adressoperationen sehr schnell zu schwer nachvollziehbaren Fehlern führen können, sowie einige Feinheiten der FAT Implementierung z.B. Behandlung von Dateinamen etc., die den Rahmen hier aber sprengen würden.

8.3.3 Zugriffsbefehle auf die SD-Karte

- mmc_init() - initialisiert die SD-Karte
- fs_fat_init() - initialisiert das FAT-Dateisystem, liest z.B. Basisadressen aus und kopiert die FAT in einen Buffer
- fs_mkdir(unsigned char* dirname) - legt ein Verzeichnis mit dem Namen 'dirname' an
- fs_fopen(const char* fileName, unsigned char* modus) - öffnet eine bestehende Datei, bzw. erzeugt eine Datei mit dem Namen 'fileName' in dem Modus 'modus'. Der Modus kann r = read, a = append oder w = write sein und liefert ein Struct vom Typ fs_file_struct zurück. Dieser Struct enthält verschiedenste Informationen über die geöffnete Datei. So z.B. aktuelle Position, Name, Endung, Start-Cluster, Aktuelles-Cluster
- fs_fgetc(fs_file_struct* sd_file_data) - liefert den char der aktuellen Position der geöffneten Datei 'sd_file_data' zurück und erhöht die aktuelle Position um eins
- fs_fgets(char* s,int n,fs_file_struct* sd_file_data) - schreibt in den char 's' den String der Länge 'n' der aktuellen Position der geöffneten Datei 'sd_file_data' und erhöht die aktuelle Position um 'n' Stellen
- fs_fputc(fs_file_struct* sd_file_data,unsigned char c) - schreibt an der aktuellen Position der Datei 'sd_file_data' den char 'c'und erhöht die aktuelle Position um eins
- fs_fputs(fs_file_struct* sd_file_data, unsigned char* s) - schreibt den String 's' an die aktuellen Position der geöffneten Datei 'sd_file_data' und erhöht die aktuelle Position um 'n' Stellen
- fs_fflush(fs_file_struct* sd_file_data) - speichert die bis dahin noch nicht gesicherten Sektoren der Datei 'sd_file_data' und schreibt die sich geänderten FAT-Sektoren auf die SD-Karte
- fs_fclose(fs_file_struct* sd_file_data) - führt ein fs_flush() aus und löscht sd_file_data Struct
- fs_rmdir(unsigned char* dirname) - entfernt das Verzeichnis mit dem Namen 'dirname' falls das Verzeichnis leer ist
- fs_remove(unsigned char* filename) - entfernt die Datei mit dem Namen 'filename'
- fs_openDir(unsigned char* dirname) - liefert ein Struct vom Typ fs_directory_struct zurück, in diesem ist zum Beispiel die Position des Daten-Clusters angegeben in dem sich die Verzeichniseinträge befinden
- fs_readDir(fs_directory_struct* Directory,fs_entry_struct* entry) - schreibt in den Struct vom Typ fs_entry_struct den Namen sowie den Typ und die Größe des aktuellen Verzeichniseintrages und erhöht die Position des aktuellen Verzeichniseintrages bis zum letzten Eintrag
- fs_closeDir(fs_directory_struct* Directory) - löscht das fs_directory_struct Struct
- fs_fileSize(unsigned char* filename) - gibt die Dateigröße der Datei 'filename' zurück falls diese existiert, sonst wird -1 zurückgegeben
- fs_isFile(unsigned char* filename) - gibt 1 zurück falls die Datei'filename' existiert sonst 0
- fs_existDirectory((unsigned char* dirname) - gibt 1 zurück falls das Verzeichnis'dirname' existiert sonst 0

Alle hier angegebenen Namen für Dateien und Verzeichnisse müssen den kompletten Pfad enthalten.

Ein kleines Beispiel zur Erzeugung eines Verzeichnisses, einer Datei und des Schreibens eines kurzen Strings in die neu angelegte Datei:

```
....
fs_mkdir("/TestVerzeichnis");
fs_file_struct* TestStruct = fs_fopen("/TestVerzeichnis/TestDatei.TXT","w");
fs_fputs(TestStruct,“Ich bin ein kleiner Testtext ");
fs_fclose(TestStruct);
....
```

8.4 Fazit

Die Entscheidung ein bekanntes Dateisystem zu implementieren hat auf der einen Seite die erhoffte Flexibilität gebracht und auf der anderen Seite den Vorteil, dass es zu jeder Zeit genügend Quellen und Beispiele zu finden gab, auch wenn diese hin und wieder fehlerhaft bzw. unvollständig waren. Die von uns an das Speichermedium gestellten Anforderungen konnten zur Zufriedenheit umgesetzt werden, auch wenn die Übertragungsgeschwindigkeit nicht ganz für einen flüssigen Videostream reichen wird. In diesem Teilprojekt haben wir vor allem erkannt, wie wichtig kontinuierliche Datensicherung und umfangreiches Testen ist. Wir sind dazu übergegangen jede kleine Änderung einzeln zu testen und diese gleich zu sichern wenn sie funktioniert hat. Bei dem Versuch gleich mehrere Änderungen auf einmal umzusetzen haben wir keine Chance gehabt den/die Fehler zu finden, da falsche Operationen auf der FAT nur schwer nachzuvollziehen sind. Zusammenfassend können wir diesen Projektteil als erfolgreich abgeschlossen betrachten.

Kapitel 9

Kamera

9.1 Motivation

Die Anforderung an unser Projekt war, dass der Tauchroboter Unterwasser Bilder und Videos machen kann. Da eine Übertragung aufgrund der niedrigen Bandbreite sehr schlecht möglich ist, haben wir uns überlegt diese Daten auf einer SD-Speicherkarte zu sichern. Die so gewonnenen Bilder sollen auf der einen Seite zur visuellen Rückkopplung und auf der anderen Seite teils in eine noch nicht implementierte Objekterkennung eingehen.Somit haben wir im zweiten Semester mit der Implementierung und Anbindung der Kamera an das Uboot begonnen.

9.2 Kameramodul

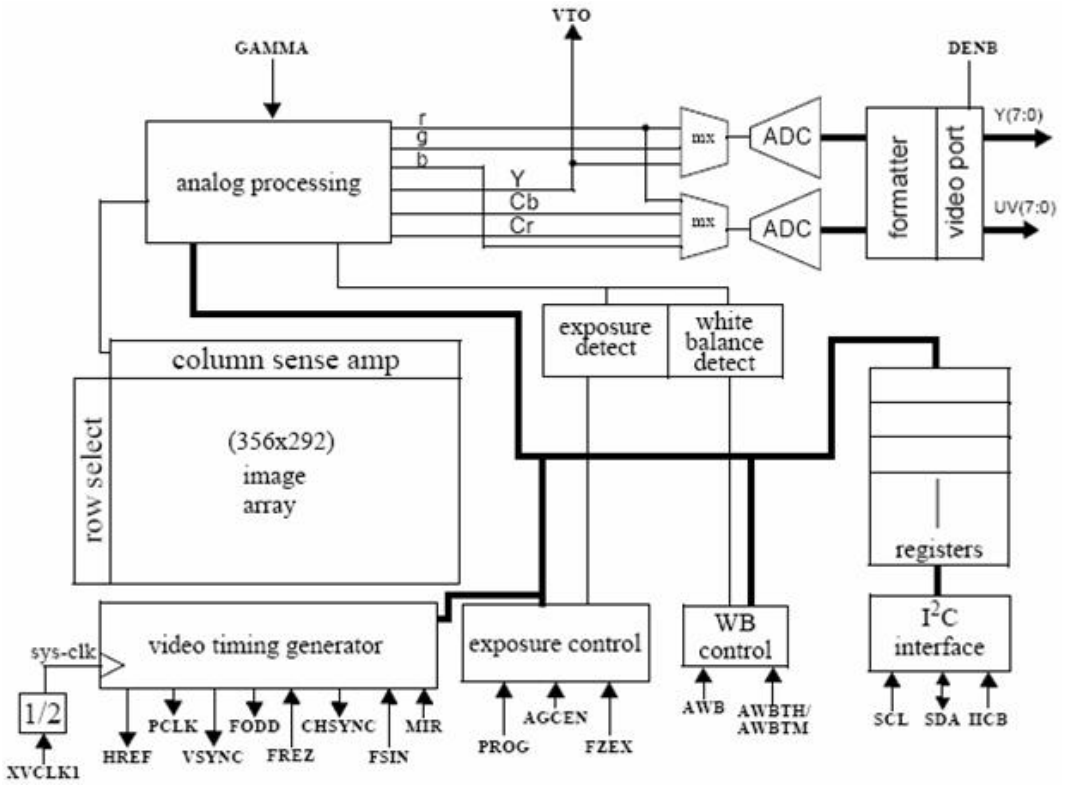
9.2.1 Wahl der Kamera

Die Wahl der Kamera hat bei uns nicht besonders lange gedauert. Zuerst hatten wir die Idee eine fertige Digitalkamera ins Boot einzubauen und nur deren Ansteuerung selbst zu implementieren. Diese Umsetzung ist im Rahmen eines Informatik-Projektes zu trivial und aufgrund dessen haben wir diesen Lösungsansatz nicht weiter verfolgt. Bei unseren Recherchen haben wir in Erfahrung gebracht, das die Smart Systems Projektgruppen aus dem letzten Jahr ebenfalls Kameras benutzt haben. Eine Neuanschaffung barg das Risiko das wir von unserem knappen Budget einen Teil fehl investieren würden. Somit konnten wir mit Erlaubnis der Abteilung unsere ersten Versuche mit dem schon vorhandene Kameramodul machen. Neben dem finanziellen Aspekt konnten wir von der Dokumentation und dem Programmcode der Smart Systems Gruppen lernen, auch wenn wir für unser Vorhaben die größten Teile neu implementiert haben.

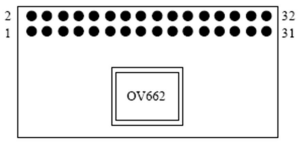
9.2.2 Omnivision OV6620 Kamera

Bei der Kamera handelt es sich um den OV6620 Kamerachip von Omnivision. Dieser Kamerachip ist weit verbreitet und dadurch gut dokumentiert. Seine häufigsten Einsatzgebiete sind in Webcams oder in vielen Individual-Lösungen wie z.B. Forschungsrobotern wie unserer.

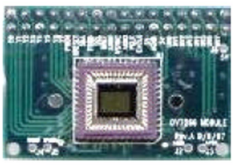
Die Kamera arbeitet basierend auf der CMOS Technologie und hat eine Auflösung von 352x288 Pixeln. Die Daten werden dem Benutzer über zwei 8 Bit-Breiten Bussen für Chrominanz und Luminanz zur Verfügung gestellt. Das Auslesen der Bilder ist mit einer maximalen Geschwindigkeit von 60 FPS möglich. Zusätzlich zu dem digitalen Bildausgang verfügt die Kamera über einen Video Analog Ausgang. Dieser hat es uns oftmals ermöglicht mit Hilfe eines Fernsehers einfache Bildtests und Kamerajustierungen durchzuführen ohne extra Auswertelogik anzubinden. Der Kamerachip verfügt leider nicht über einen internen Speicher. Dies hat zur Folge das ein Bild nur solange gehalten wird, bis es zu einer neuen Belichtung des CMOS-Feldes kommt. Die folgende Zeichnung zeigt die Architektur des OV6620 Chips.



Dieser Fakt wird im späteren Verlauf noch einige unserer Designentscheidungen beeinflussen. Der OV6620 Chip sitzt auf der C3088 Kamera Platine. Diese führt im wesentlichen nur die vom Chip zur Verfügung gestellte Signale durch und stellt sie über Pins zur Verfügung. Leider werden nicht alle vorhandenen Signale bereitgestellt. Die Verwendung dieser Kameraplatine schrenkt daher schon die vom Chip bereitgestellten Funktionalitäten ein, was aber für unsere Designentscheidungen keine Rolle spielen sollte.



PCB Layout (Top side)



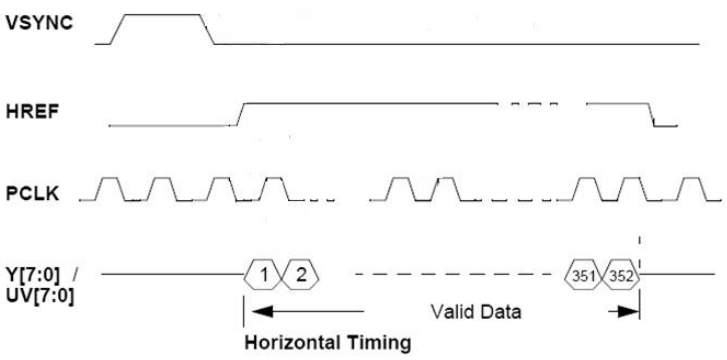
Bei einem Vergleich der Chiparchitektur mit der des Kameramodules wird schnell klar das hier nur eine minimierte Anzahl an Signalen zur Verfügung steht. Das Konfigurieren der Kamera ist über ein I^2C-Interface möglich. Neben diversen Einstellung der Bildqualität, so z.B. Helligkeit, Farbsättigung, etc. ist es hier möglich die Kamera in verschiedenen Betriebsmodi laufen zu lassen.

9.2.3 Betriebsmodi und Bildsynchronisation

Im folgenden wird der Ablauf bei der Bildsynchronisation beschrieben. Der normale Betriebsmodus beschreibt den Standardablauf, von dem allen weiteren Modi nur eine Abwandlung darstellen.

- **Normaler Modus**
  Für Synchronisation eines Bildes sind 3 Signale zuständig.
  - **Vsync** wird zu Anfang und zum Ende eines Bildes auf high gesetzt. Während der Übertragung gültiger Pixel zu einem Bild ist Vsync wieder auf Null zurückgesetzt
  - **Href** wird bei der Übertragung jeder Zeile auf 1 gesetzt. Alle übertragenen Pixel während Href auf 1 gesetzt ist werden als gültige Pixel angenommen.
  - **Pclk** gibt den Takt für die Kamera an. Bei gesetztem Href und einem Event der Pixelclock liegt ein gültiges Pixel am Ausgang an.

  Im normalen Modus wird während der Bildübertragung für jede Zeile 288 mal Href auf 1 gesetzt und die Pixelclock sollte 352 gültige Takte pro Zeile besitzen. Dies führt zu der Bildauflösung von 288x352 Pixeln. Die untere Zeichnung verdeutlicht den Signalverlauf bei der Übertragung eines Bildes.



- **Slave Modus**
  Im Slave Modus werden die 3 Signale Href, Vsync und Clk von dem Master extern gesetzt und die Kamera agiert als Slave. Der Vorteil dabei ist, dass das System das Timing der Bildsynchronisation bestimmen kann. Der große Nachteil liegt in dem Aufbau der Kamera. Da diese, wie schon oben erwähnt, keinen internen Speicher hat werden die Photozellen automatisch ohne mechanischen Shutter wieder belichtet und das aktuelle Bild überschrieben. Somit ist die externe Beeinflussung der Bildsynchronisation stark durch die Architektur des Chips eingeschränkt.

- **Frames Exposure Modus**
  Ähnlich dem Slave Mode wird hier über das Setzen eines externen Signals das Timing der Bildsynchronisation bestimmt. Das dazu verwendete Signal ist Frex. Falls Frex auf 1 gesetzt ist, wird ein neues Bild gelesen und beim Zurücksetzen auf null wird das Bild, wie im normalen Modus, zur Verfügung gestellt. Dies bedeutet im Besonderen das in diesem Modus Frex extern gesetzt wird und weiterhin Vsync, Href und Pclk von der Kamera gesetzt werden. Auch hier tritt wieder das Problem des notwendigen mechanischen Shutters auf um diesen Modus im vollen Umfang nutzen kann.

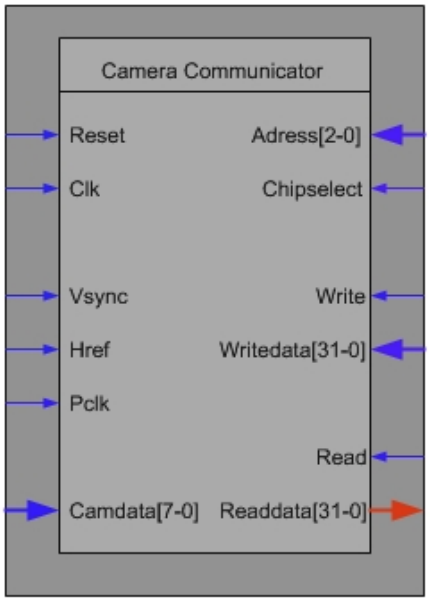
- **QCIF Modus**
  In diesem Modus wird die Bildauflösung halbiert und somit die Übertragungszeit drastisch verkürzt und die Datenmenge minimiert.

Für unsere Zwecke kommen daher nur der Slave Modus und der QCIF Modus in Frage, da wir von Anfang an von der Montage eines mechanischen Shutters aus Platzgründen in der Kuppel abgesehen haben.

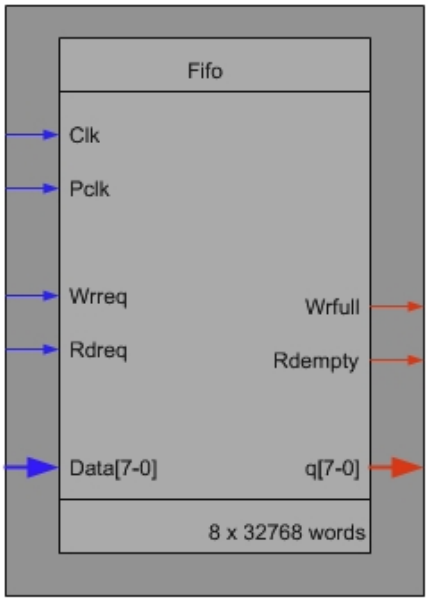
9.3 Anbindung

9.3.1 VHDL Anbindung

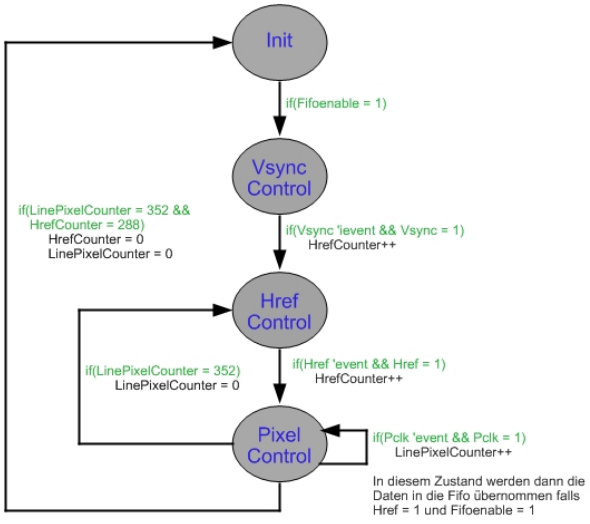
Bei der Implementierung und Anbindung in VHDL konnten wir auf die von den Smart Systems Gruppen erstellten Dateien zugreifen. Bei der Umsetzung unserer VHDL-Anbindung wurde schnell klar, dass ein Recyceln und Anpassen der schon erstellten Datei mehr Zeit kosten würde als eine Neuimplementierung. Die anderen Smart System Gruppen haben ebenfalls eine DMA-Controller benutzt zum Ansprechen des Speichers. Wir haben uns gegen einen DMA-Controller und für ein 2-Prozessor-System entschieden. Der Vorteil ist, dass neben der reibunglosen Bildspeicherung, die später geplante Bildkomprimierung ebenfalls auf diesem nur für Bildverarbeitung vorgesehenen Prozessor Platz finden würde.
An dem obigen Design lässt sich erkennen das wir nur die Bilddaten für Luminanz speichern. Dies hat



den Vorteil das eine Speicherung von Schwarzweißbildern den Speicherbedarf um fast die Hälfte reduziert und mehrere Bilder aufgenommen werden können. Zwar wurde uns im Laufe unseres Projekts eine DCT-Implementierung zur Bildkompression angeboten, aber aus uns nicht ersichtlichen Gründen ist das dazugehörige individuelle Projekt nie vollendet worden, bzw haben wir nichts mehr davon gehört. Somit haben wir uns gegen einen weitere Zeitverzögerung entschieden und eine Bildkompression vorerst nicht eingebaut. Neben den 8 bit Luminazdaten hat unsere VHDL-Anbindung die 3 Steuersignale PClk, Href und Vsync, die Systemclock und die 5 Signal zur Kommunikation über den Avalonbus. Diese sind Read, Write, Chipselect, Readdata und Writedata. Über die beiden letzten 32 Bit breiten Signale kann mit dem VHDL-Modul kommuniziert werden.



In dem VHDL ist neben der Bildsynchronisationslogik noch eine FIFO eingebettet. Diese buffert ca. $1/4$ des Bildes um bei Verzögerungen beim Auslesen der Pixel einem Datenverlust vorzubeugen. Die folgende Zeichnung an Hand des gezeigten Zustandsautomaten die Logik unseres VHDL-Modules erleutern.



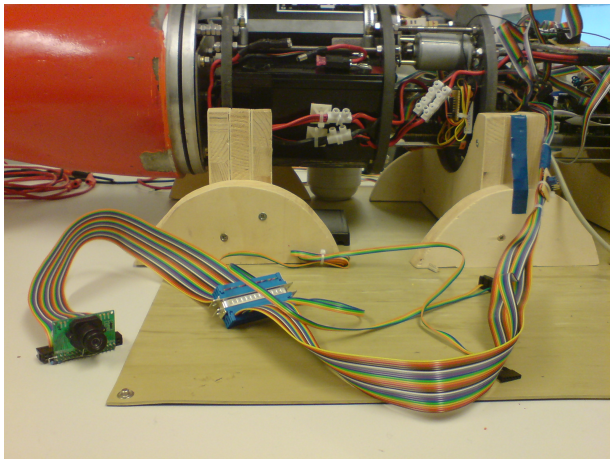
Im Grunde wurde das unter normalen Modus vorgestellte Modell auf diesen Zustandsautomaten übertragen und in VHDL umgesetzt. Neben der VHDL-Anbindung wurde noch der Zugriff über den Avalonbus modifiziert. Standardmäßig wird dem lesenden Zugriff über den Avalonbus ein Waitzyklus hinzugefügt. Dies resultiert daraus, dass es einen Clockzyklus dauert bis die Adresse am VHDL-Modul anliegt. Im VHDL-Modul selber muss darüber hinaus noch ein readrequest an die FIFO angelegt werden um Daten auszulesen. Nach Anlegen des Readrequests liegen die Daten ebenfalls erst nach einem Clockzyklus an. Darauf hin mussten wir statt einem Waitzylkus zwei Waitzyklen beim lesen über dem Avalonbus im SOPC-Builer voraussehen.

9.3.2 C-Code Einbindung

Aus der FIFO wird über den Avalon-Bus im C-File Pixel für Pixel ausgelesen und direkt mit Hilfe der SD-Kartenbefehle als BMP-Format gespeichert. Zusätzlich kann die Kamera über eine Reihe von vordefinierten I^2C-Befehlen konfiguriert werden. Die dazu nötigen Adressen und zu setzenden Werte zur Konfigurierung können aus den Datenblättern zur Kamera entnommen werden.

9.3.3 Einbau im Technikgerüst

Die Kamera ist über ein 50 cm 21 poliges Flachbandkabel mit der FPGA-Anschlussplatine verbunden.



Die extreme Kabellänge ergibt sich aus der Position der Kamera und dem FPGA. Dieses Kabel ermöglicht den Zugriff auf die zwei Signalbusse für Chrominanz und Luminanz, Href, Vsync, Pclk und der Stromversorgung. Die Kamera ist im Gegensatz zum FPGA nicht mittig im Uboot, sondern vorne in der Glaskuppel montiert. Durch einen Servo lässt sich der Neigungswinkel der Kamera einstellen um leichter Bilder vom Bodenbereich bzw. von Objekten zu machen die sich über dem Uboot befinden.
Die letzte Grafik soll noch einmal eine Übersicht über die Integration der Kamera im System geben.

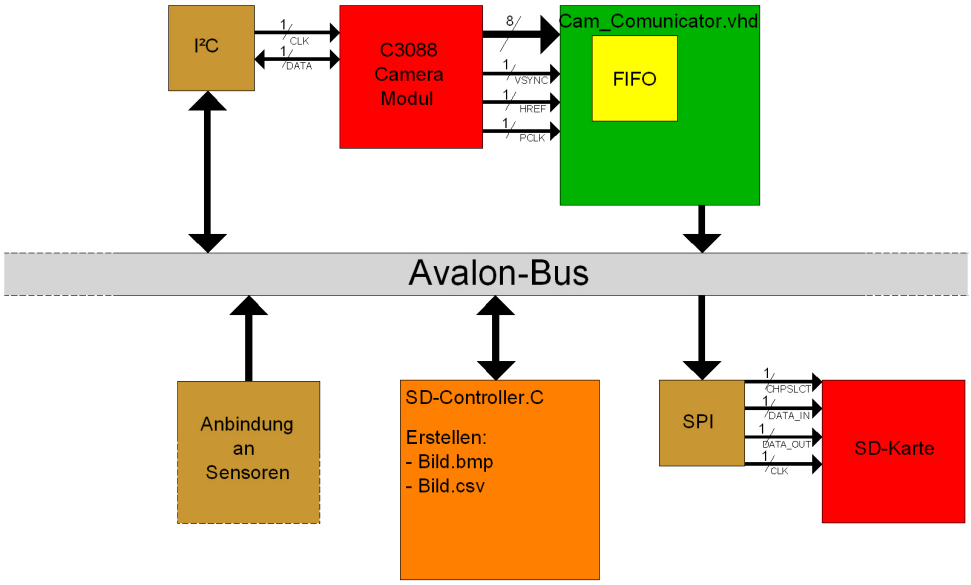
9.4 Implementierungsprobleme und Fazit

Bei der Anbindung der Kamera sind hauptsächlich technische Probleme aufgetreten. Bei der Datenverarbeitung und Abfrage über den Avalonbus traten mehrere Timingprobleme auf. Diese waren oftmals darauf zurück zu führen das bei der Timinganalyse die notwendigen Read- und Write-Wartezyklen zur Übernahme der Daten nicht berücksichtigt wurden. Diese Probleme konnten aber schnell beseitigt werden und traten nach einer korrigierten Timinganalyse und den daraus resultierenden Änderungen nicht mehr auf.

Auf Grund der extremen Kabellänge und dem Uboot spezifischen Platzmangel konnten wir nicht extrem stark isolierte Kabel verwenden. Darauf hin wurden immer wieder Signale von der Kamera gestört oder verfälscht. Gerade bei der Bildsynchronisation hat diese Störungsquelle zu einigen Seitenefekten geführt. Zur Beseitigung dieser Effekte wurde hauptsächlich das VHDL angepasst. Einige immer wieder auftretende Fehler waren:

1. Das Vsync-Signal wurde während einer Bildsynchronisation durch Überlagerung auf High gezogen. In dem alten VHDL-Modul führte dies zum Abbruch der Bildübertragung in die FIFO, da bei ungestörtem Signalverlauf ein erneutes Vsync = 1 ein neues Bild bedeuten würde.

2. Ein weiterer Fehler wurde durch eine Verfälschung des Pclk-Signals verursacht. Pro Bild gab es dadurch in einigen Zeilen 1 bis 2 Pixel mehr als die Auflösungsbedingten 352 Pixel es vorsahen.



Dies hatte zur Folge, dass das Bild in sich verdreht war. Das Problem haben wir durch ein Sampeln der Pclk versucht zu beseitigen.

Letztendlich war dieser Projektteil eine eher hardwaretechnische Herausforderung als eine Softwareproblematik. Ein loser Pullabwiederstand oder ein loses Kabel, Timingprobleme und Störsignale waren eigentlich die Probleme die es hauptsächlich zu meistern galt. Diese waren leider oftmals nicht auf dem ersten Blick zu erkennen. Die meiste Zeit verging dabei diese Probleme zu finden, Lösung fiel uns dann oftmals nicht schwer.

Kapitel 10

Hostsystem

10.1 Einleitung (Motivation)

Das Hostsystem soll die Überwachung, Steuerung und Kommunikation mit dem UBoot während einer Tauchfahrt ermöglichen. Telemetriedaten werden kontinuierlich vom Boot zur Hostsoftware geschickt und können über selbige dargestellt und überwacht werden. Darüberhinaus kann die Regelung de-/aktiviert, sowie verschiedene Parameter der Regelung verändert werden. Die Steuerung des Bootes kann vollständig über das Hostsystem erfolgen so das Antrieb, Tauchtanks und Ruder über Tastatur gesteuert werden können.

Einige Anforderungen konnten allerdings nicht erreicht werden, wie z.B. das Anfahren bestimmter Position im Becken welche über eine Eingabemaske vorgegeben werden. Teilweise ist die Funktionalität implementiert und kann dank des modularen Aufbaus schnell eingebunden werden.

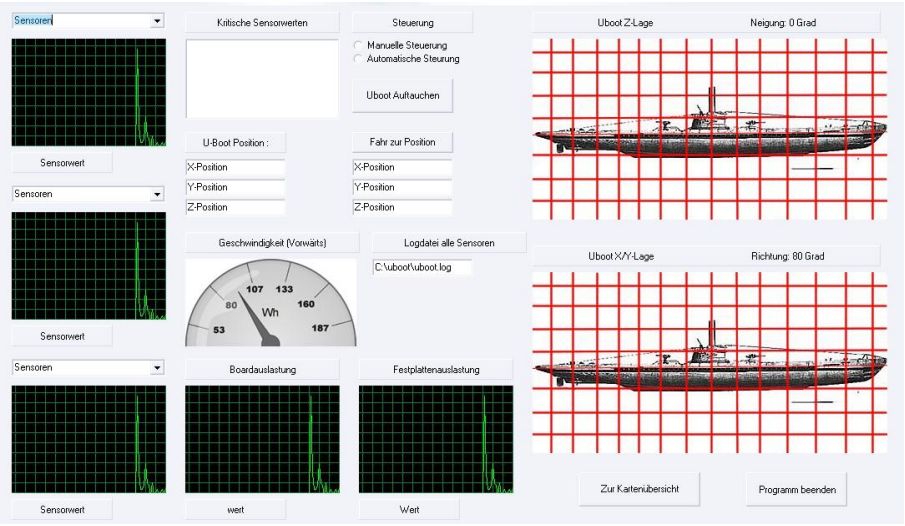


Abbildung 10.1: Erste Vorstellung vom Hostprogramm

10.2 Zielbestimmung

10.2.1 Muss-Kriterien

Die Hostsoftware sollte als Überwachungssystem für unser Uboot dienen. Ohne das Hostsystem würden wir nicht wissen was innerhalb unseres Ubootes passiert. Da wir aber kein Empfang haben sobald das Uboot unter Wasser ist, brauchen wir die Möglichkeit bestimmte Logfiles zu erstellen und auszulesen, damit wir Daten auswerten können bei einem Tauchgang der tiefer ist als die Empfangstiefe unserer Kommunikationsmodul (433 Mhz mit 27 Mhz Rückfallfrequenz in einer Richtung). Wenn das Uboot aber in

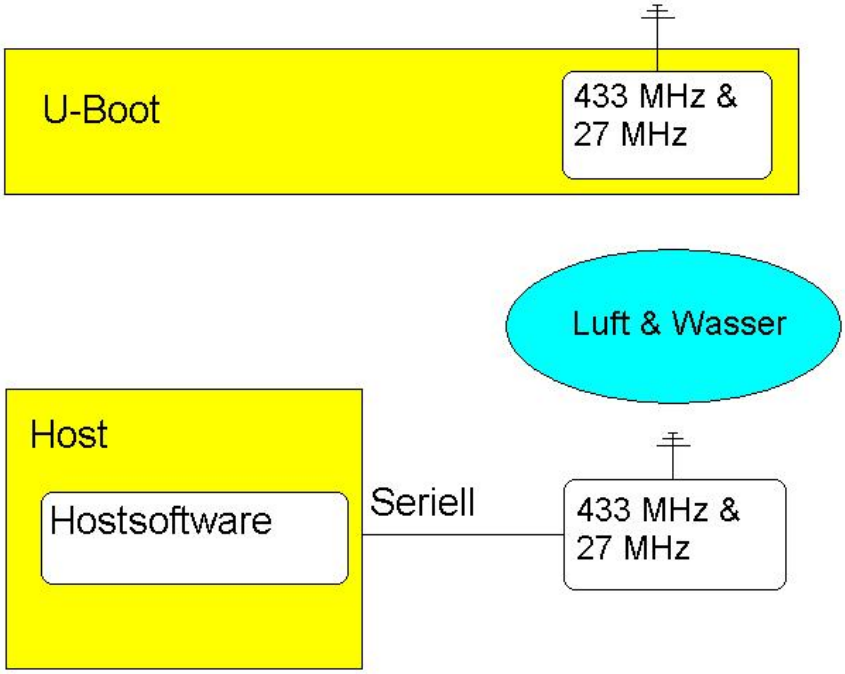


Abbildung 10.2: Kommunikationsanforderung

Empfangsweite ist muss auch die Möglichkeit behalten bleiben die Werte direkt auszugeben. Ein weiteres Soll-Kriterium ist die Eingabe einer Tiefe am Uboot, wo es sich nachher aufhalten soll. Das Uboot soll sich selber in die gewählte Tiefe einregeln können und bei weiterfahrt auf diese Tiefe halten können ohne große Abweichungen.

Das Programm sollte geteilt werden in Bereiche für die einfache Benutzung und in ein zuschaltbaren Teil für die erweiterte Benutzung, damit neue Benutzer nicht überladen werden mit Informationen. Dabei handelt es sich um Optionen, die man nicht unbedingt braucht wenn man nur das Uboot fahren will. Man ist dabei unabhängig von diese Optionen.

10.2.2 Wünschkriterien

Darüber hinaus gibt es viele Wunschkriterien die erfüllt werden können, die nicht in erster Linie dazu gedacht waren um über unser Hostsystem ausgegeben zu werden. So würde es sich zum Beispiel anbieten unsere Regelung anzusteuern und zu verbessern. Weiter ist auch die Steuerung unser Uboot über ein angeschlossenes „Gamepad“ am Rechner wünschenswert. Dieses würde dann als Ersatz dienen für unsere Funkfernbedienung. Die Logfiles vom Uboot abholen ist ein Muss, aber die Weiterverarbeitung zum Beispiel in einer SQL-Datenbank wäre wünschenswert, da die Möglichkeit zur schnelleren Auswertung

besteht. Eine Ausgabe von Statistiken wie Boardauslastung, wieviel Speicher noch frei ist sowie eine Ansteuerung der Kamera gehören ebenfalls dazu.
Im Moment indem die Verbindung abgebrochen wird zum Uboot, sollten alle Schaltflächen der Steuerung deaktiviert werden. Das wäre hilfreich, so dass man nicht etwas abschicken kann, wenn es auch kein Zweck mehr hat. Der Benützer bekommt über diesen Weg schneller mit ob es überhaupt noch eine Verbindung existiert.
Eine Sprachenänderung im Programm ist dann hilfreich, da viele Funktionen auf Englisch definiert sind und dieses Programm von Deutsche Benutzer benützt wird. Deswegen die Wahl zwischen Deutsch und Englisch im Hostprogramm. Andere Sprachen einfügen ist aber auch Möglich.

10.3 Produktumgebung

10.3.1 Software

Zum Kompilieren wird das neue JDK 1.5 gebraucht, da bestimmte Funktionen hierin enthalten sind die sich nicht in der vorherigen JDK-Version befinden. Zum Kompilieren braucht man allerdings extra Pakete, wie das comm32 für die serielle Schnittstelle und einmal das jjstick.dll für die Unterstützung vom Gamepad. Für die serielle Schnittstellen sollten die Treiber installiert sein wenn man mit ein USB zu Comm Kabel arbeiten will. Wenn aber eine serielle Schnittstelle am Rechner vorhanden ist, wird empfohlen diese zu bevorzügen.
Der Betrieb der Hostsoftware müsste unter alle bekannten Betriebssysteme ohne Einschränkung möglich sein. Allerdings sollte man darauf achten das mindestens ein Java Virtual-Machine installiert ist. Unter die meist bekannte Betriebssysteme ist dies schon der Fall.

10.3.2 Hardware

Zum Betrieb des Hostsystems ist eine serielle Schnittstelle erforderlich, ohne diese Schnittstelle ist der Anschluss unseres Kommunikationsboxes nicht möglich. Dazu braucht man eine 12V Stromquelle für die Kommunikationsbox.
Wenn man das Uboot nicht nur mit der Tastatur ansteuern will, aber mit ein Joystick oder ein Gamepad, dann sollte dieser vor Programmstart angeschlossen werden am Rechner.

10.4 Funktionen

10.4.1 Serielle Kommunikation

Das Hostsystem sollte immer ein „ping“ senden und überwachen ob die Kommunikationsbox antwortet. Dazu sollten im Hostsytem die mögliche Commports des Rechners abwechselnd genutzt werden, damit das Programm nicht abhängig ist von einer bestimmten Commport-Einstellung. Der Kommunikationsbox sollte dabei innerhalb 2 sekunden antworten. Die Pings werden in Abschnitten von 2 Sekunden zum Kommunikationsbox gefunkt. Empfängt das Hostsystem nach 4 mal „anpingen“ keine Antwort wechselt das Programm auf den nächsten Commport. Wenn eine Verbindung da ist mit dem Kommunikationsbox und der Kommunikationsbox mehr als 10 sekunden nicht antwortet, fängt das Programm an wieder die einzelne Commports zu scannen, die Verbindung wird eingestuft als „Connection lost“. Ist eine Verbindung gelungen, schickt der Rechner ein „GSV“ (Sensorwertabfrage) Kommando und werden die erste Sensorwerte geliefert wenn eine 433Mhz Kommunikation existiert.

- Senden: Zum Senden wird eine Warteschlange (Vector) aufgebaut, wobei das unterste Element immer als erstes abgeschickt wird. Da unsere Kommunikation kein grosser Last haben kann. Müssen wir soviel wie möglich sparen beim Abschicken unsere Befehle. Unnötige Befehle werden zum Beispiel rausgenommen aus der Warteschlange.
  Beispiel: Wir schicken ein Befehl um vorwärts zu fahren. In der Warteschlange befinden sich aber noch zwei andere Befehle für den Motor, Einmal Rückwärts und einmal Anhalten. Wenn wir diese nicht rausnehmen, würde sich der Motor erst nochmal Rückwärts bewegen, dann anhalten und danach erst vorwärst fahren. Das ist nicht ein gewünschtes Verhalten und machen in dem Fall genau folgendes. Wenn der Befehl Motor vorwärts bewegen oben in der Liste eingefügt wird, durchsuchen wir zuerst die Liste ob es da noch Befehle gibt, die auf dem Motor zugreifen wollen. Wenn ja, werden alle diese Befehle aus der Liste rausgenommen.

  Da wegen die 27Mhz Kommunikation unser Datendurchsatz noch weiter verringert und nur bestimmte Befehle bei 27Mhz gesendet werden, verändert sich die Priorität der Befehle. In dem Moment, das Steuerungsbefehle abgeschickt werden sollten, werden diese nicht wie gewohnt oben in der Liste eingefügt, aber unten in der Liste, so das diese gleich als nächstes weggeschickt werden. Dieses ist Möglich, da alle andere Steurungsbefehle erst gelöscht werden aus der Liste.

- Empfangen: Im Moment das Daten anliegen an der serielle Schnittstelle wird alles Zeilenweise ausgelesen und wiedergegeben in der Konsole: „Answer: (Daten die reingekommen sind)“
  Sind es zum Beispiel Daten der Sensoren werden diese über ein Listener weitergeleitet. Dieser String wird auseinander genommen und die jeweilige Werte als Variabele weitergegeben. In das Programm, wo ein Listener eingebaut ist, werden diese neue Werte eingesetzt.

Alle mögliche Befehle sind in Abbildung 10.3 zu finden. Diese können auch von Hand eingegeben werden auf dem „Controlpanel“ für Testzwecke.

```
public static String SENSOR_UPDATE = "GSV";
public static String CONTROL_UPDATE = "GCV";
public static String NAVIGATION_UPDATE = "GNV";

public static String NAVIGATION_DEBUG_ENABLE = "NM1";
public static String NAVIGATION_DEBUG_DISABLE = "NMO";
public static String NAVIGATION_DEBUG_VALUES = "ND";
public static String NAVIGATION_DEBUG_START = "NS";

public static String PING = "PNG";

public static String CONTROL_MASTER = "CM";
public static String CONTROL_SPEED = "CS";
public static String CONTROL_BETA = "CB";
public static String CONTROL_BETA_DIPTANK = "CT";
public static String CONTROL_GAMMA = "CG";
public static String CONTROL_DEPTHRUDDER = "CR";
public static String CONTROL_DEPTHDIPTANK = "CD";
public static String CONTROL_PID = "CP";
public static String CONTROL_CHANGE = "CC";
public static String CONTROL_REQ = "CVV";
public static String CONTROL_PC = "PCC";
public static String CONTROL_SELFTEST = "TST";
public static String CONTROL_EMERGENCY = "CE";

public static String CTRL_SPEED = "RS";
public static String CTRL_RUDDER = "RR";
public static String CTRL_DEPTH = "RD";

//public static String GET_SENSOR_LOG = "SL";
public static String GET_NAVIGATION_CALIBRATION = "GNC";
public static String SET_NAVIGATION_CALIBRATION = "NC";
public static String SET_NAVIGATION_WRAP = "SNW";
public static String GET_SENSOR_LOG_RESET = "SLR";
public static String NAVIGATION_RESET = "NR";
public static String START_CAM = "SAC";
public static String STOP_CAM = "SOC";
public static String TAKE_PICTURE = "TP";
public static String SEND_SUB = "SS";
public static String SET_TIME = "ST";
```

Abbildung 10.3: Befehleliste

10.4.2 SensorUpdate und SensorValueHolder

Die Sensorwerte werden jede Sekunde abgeholt und ausgegeben auf dem Sensorpanel und Sensorseitenpanel. Desweiteren werden von jedem Sensor der höchste und niedrigste Wert gespeichert um eine schnellere Bewertung der aktuellen Messwerte zu ermöglichen. Inzwischen wird an unser String ganz hinten ein Wert angehängt der aussagt ob wir über eine 433Mhz Kommunikation verfügen. Wenn dies Nicht der Fall ist (der letzte Wert im Sensorwertstring ist dabei 0), bekommt man immer der letzte Sensorwertstring zugeschickt vom Kommunikationsbox der da gespeichert worden ist.

10.4.3 ControlStatusUpdate und ControlStatusValueHolder

Die Kontrollwerten sind dafür um zu sehen ob eine bestimmte Kontrolle im Uboot überhaupt gesetzt worden ist. Diese Kontrolle passiert jede 6 Sekunde einer Abfrage und setzt gegebenfalls die jeweilige „Toglebutton“ um auf dem ControlPanel. Durch die Verzögerung beim senden ist es Möglich, das man ein Kontrollwert gerade sendet indem die Verbindung abreißt. In diesem Fall wird sich zum Beispiel das Programm die Kontrollwerten auf dem Controlpanel einstellen nachdem wieder eine Verbindung existiert.

10.4.4 ControlValueUpdate und ControlValueHolder

Mit diese Werten kann man die Regelungswerten für die unterschiedliche Regelungen anfragen vom Board im Uboot. Der selektierte Regelung im Controlpanel bekommt dabei seine Werten für P, I und D-Anteil zugeschickt und werden wiedergegeben in den Textfenster im Panel.

10.4.5 ConnectionListener

Dieser Variabele wird gesetzt als allgemeiner Wert für das gesammte Programm ob es eine Verbindung existiert mit dem Uboot. Der Wert ist die letzte Variabele in den Sensorstring die an der serielle Port rein kommt. Es wird damit entscheidet ob Buttons auf die jeweilige Panel angezeigt werden und zugleich der Statusanzeige auf dem Sensorseitenpanel.

10.4.6 CalibrationValueUpdate und CalibartionValueHolder

Um nach das Flashen vom FPGA die Möglichkeit zu bieten die Einstellungen der Sensoren zu ändern kann man die Sensor und Offset Werten das Uboot zuschicken. Dazu kann man auch die Kalibrierungswerten der Sensoren die im System sind abfragen. Insgesamt kann man dann die Werten abfragen, bearbeiten und zuschicken in einer Durchgang ausführen.

10.4.7 SensorUpdateTimer

Im Fall das wir eine 433Mhz Kommunikation haben wird das „GSV“ und „GCV“ Zeitgesteuert gesendet. In der unten abgebildete Code ist zu sehen wie groß die Zeitintervalle sind zwischen das Abschicken der einzelne Befehle (updateIntervall).

```
static int updateIntervall = 1250; // in ms
static boolean fastConnection = true; //433 Mhz Verbindung
```

Diese Einstellungen werden Initial gesetzt und bleiben bestehen in dem Fall einer 433Mhz Kommunikation existiert. Da wir über 433Mhz auch Antworten zurück bekommen können, wird einmal pro 6 durchläufe eine „GCV“ geschickt um die Status der Buttons abzufragen.
Wenn wir nur über eine 27Mhz Kommunikation verfügen und der 433Mhz Verbindung abgerissen ist, können wir nur Daten zum Uboot senden. Es können in dem Fall keine Daten zurück gesendet werden wenn es geht um Sensorwerte oder Kontrolwerten. Die 27Mhz ist von Brandbreite her wesentlich kleiner, was uns nur die Möglichkeit gibt nur noch die wichtigste Daten zu senden die benötigt werden wie Steuerung. Damit die Kommunikation immer noch richtig ablaufen kann, wird dabei das Intervall für die Abfrage der Sensorwerte erhöht und die Abfrage der Kontrolwerte unterbunden. Wir können nur über den String der Sensorwerte erkennen ob eine 433Mhz verbindung wieder existiert und können deshalb die Abfrage der Sensorwerte nicht unterbinden oder auf ein zu grosses Intervall vergrössern.

```
updateIntervall=5000;
fastConnection = false;
```

Die Variabele „fastConnection“ sagt aus ob eine 433Mhz Verbindung existiert. Wenn nicht, wird diese Variabele auf „false“ gesetzt damit keine Kontrollwertabfragen mehr zugeschickt werden. Unser Abfrageintervall der Sensorwerte wird dabei vergrössert auf 5 Sekunden.

10.4.8 commandSender

Der CommandSender regelt die ansteuerung des Uboots. Bei ein bestimmter Tastendruck sollte sich Aktorik bewegen. Da allerdings das Problem existiert, dass auf unerwünschte Momente die Verbindung abreisst, wird die Ansteuerung teils „Event-triggered“ als auch „Time-triggered“ geregelt. Bekommt das Uboot nach 5 Sekunden kein Status zugeschickt von der GUI aus, wird die ansteuerbare Aktorik ausgeschaltet.

- Time-triggered: Jede 5 Sekunden wird für alle ansteuerbare Aktorik der Status zugeschickt. (Für Motor, Ruder und Bugstrahlruder)

- Event-triggered: In den Fall, dass eine Taste gedrückt wird, wird einmal die änderung zum Uboot geschickt und zugleich gespeichert, damit auch der Teil der jede 5 Sekunden zum Uboot geschickt wird, die gewünschte Zustand entspricht.

10.4.9 Joystickanbindung

Im Programm ist ein Joystick angebunden. Beim Starten vom Programm wird das Programm sehen ob ein Joystick/Gamepad angebunden ist oder nicht. Wenn ja, wird folgendes angezeigt in der Console:

```
Joystick/Gamepad found
```

Wenn kein Joystick gefunden wird, wird es folgende Ausgabe geben in der Console:

```
No Joystick/Gamepad found
```

Die Steuerung vom Motor und die Bugstrahlruder sind auf bestimmte Tasten auf das Gamepad weggelegt. Jede Taste auf das Gamepad ist mit ein bestimmten Wert codiert. Beim drücken einer bestimmte Taste wird ein „joystickButtonChanged“ Ereignis ausgelöst. Wenn es eine Taste ist der eine Funktion zugewiesen ist, wird ein Tastendruckereignis simuliert im Programm und den Ereignis weiter geleitet an den „CommandSender“. Das Richtungskreuz auf das Gamepad wird benützt für das rechts und links steuern des Uboots. Das Richtungskreuz kann auf 360 Grad 8 unterschiedliche Werten zurückgeben. Im Programm sind dabei die Ereignisse zu 270 Grad und 90 Grad implementiert.
Alle letzgedrückten Werten vom Richtungskreuz und der Tasten werden gespeichert, damit erkannt wird ob eine Taste wieder losgelassen worden ist.

10.5 Benutzungsoberfläche

Das Hauptprogramm setzt sich im Hauptfenster aus folgenden Komponenten zusammmen:

- Oben gibt es eine Menübalken mit den Menü „File“ für Hauptfunktionen. Damit man die Sensoren zurücksetzen kann auf 0 im Programm und um den erweiterten Modus freizuschalten(Abbildung 10.4). Das Zweite Menü ist für eine Sprachenänderung wobei bis jetzt die Möglichkeit besteht zwischen Englisch und Deutsch. Im dritten Menü ist eine kleine Hilfe eingebaut.

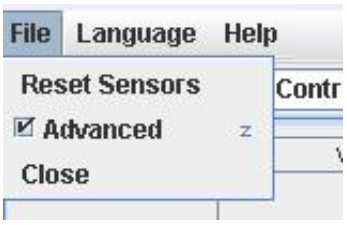


Abbildung 10.4: Menu

- Das Hauptpanel zeigt standartmäßig das Submarine-Logo. Verschiedene Panele können für unterschiedliche Funktionalität ausgewählt werden. Die einzelnen Panele werden unten weiter spezialisiert:
  - Das Hauptpanel („Mainpanel“) dient Präsentationszwecke. Das Bild ist unser Logo der Projektgruppe mit darunter eine Versionsangabe der GUI.

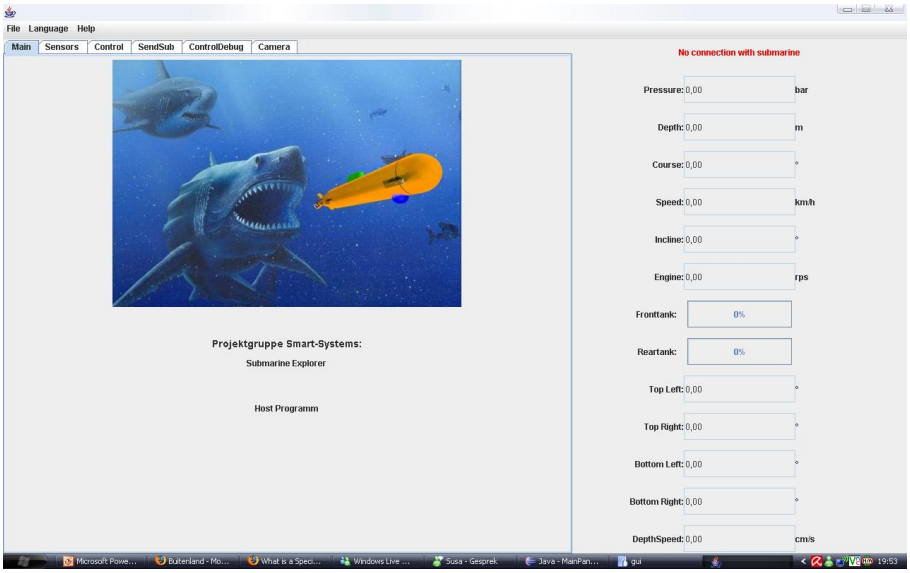


Abbildung 10.5: Hauptpanel

– Im „Sensorpanel“ werden die Werten der meiste Sensoren wiedergegeben. Es gibt daneben eine extra Ausgabe für die maximal und minimal gemessene Sensorwerten. Damit können wir schnell sehen ob sich die Werten der Sensoren in Ordnung sind oder ob das Uboot sich in Ausnahmezustand befindet.

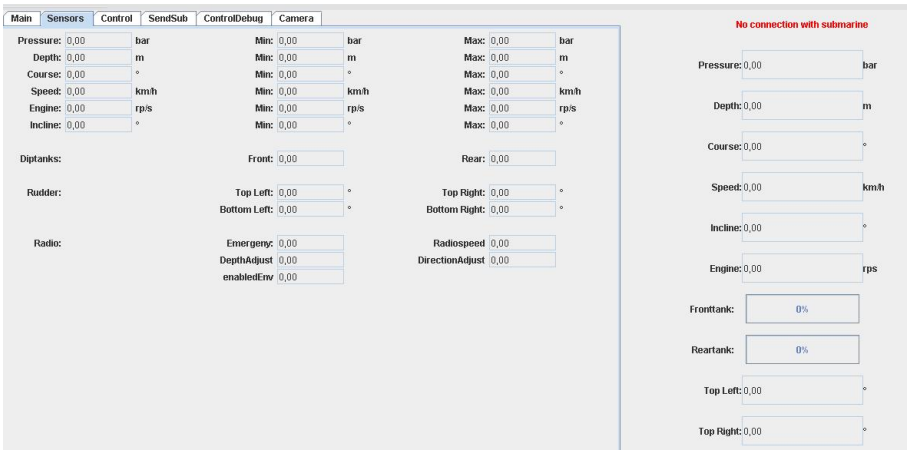


Abbildung 10.6: Sensorpanel

– Der „Controlpanel“ ist für das Ein- und Ausschalten der einzelnen Steuerungen und daneben gibt es die Regelungswertesteuerung, Tiefenangabe und ein Eingabefeld für Kommandos. Das Ein- und Ausschalten ermöglicht uns nur bestimmte Aktorik freizugeben. In der 27Mhz Kommunikationszustand funktioniert allerdings nur das Ein- und Ausschalten der Hauptkontrolle.

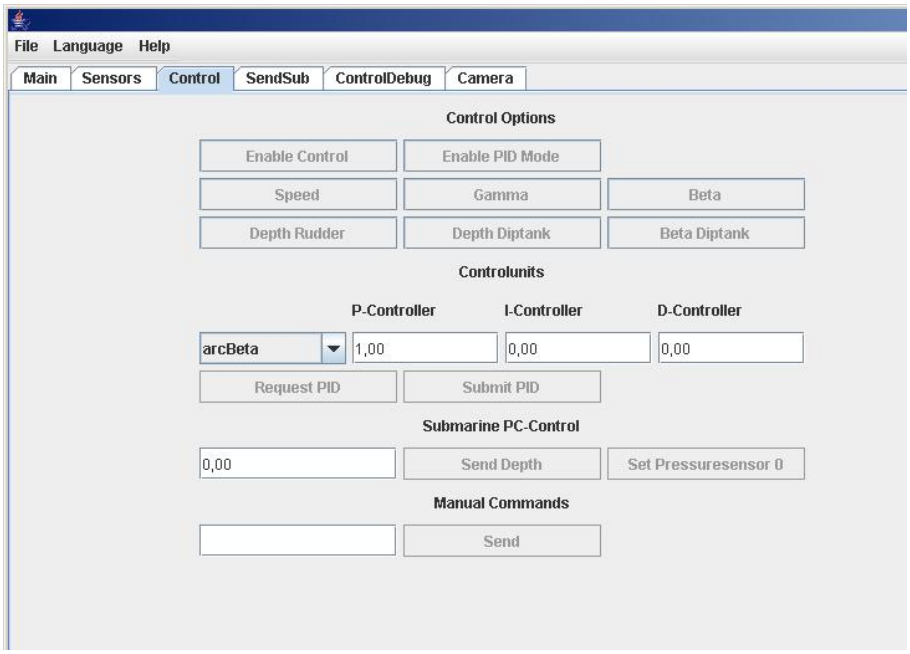


Abbildung 10.7: Controlpanel

– Der „Controldebugpanel“ gibt uns ein grafischen Überblick über den Unterschied zwischen Ist- und Sollwert. Es wird auf diesen Panel ein Graph gezeignet mit den angesteuerte Tiefe und die wirkliche Tiefe (aus dem Drucksensor) wo er sich gerade befindet. Nebenbei gibt es die Möglichkeit sich den Graphen zu löschen. Gleich danach fängt der Graph beim Eintreffen von Sensorwerten wieder an aufzuzeichnen.

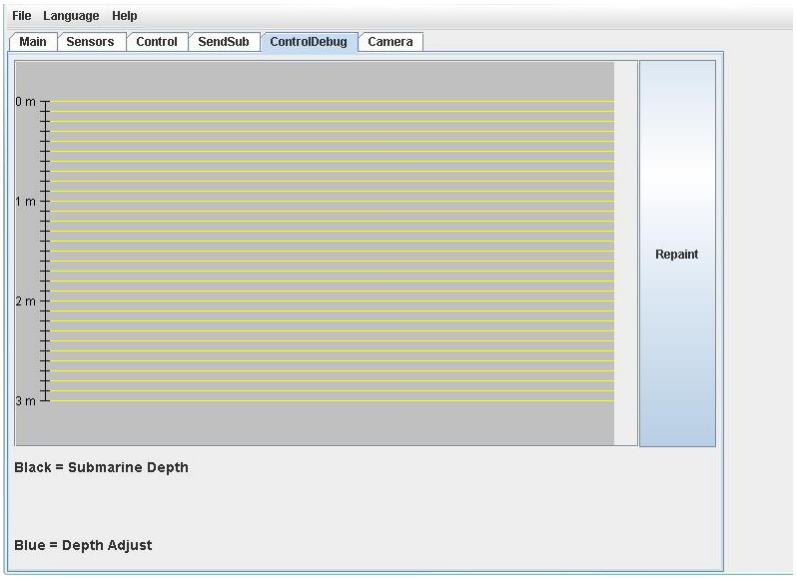


Abbildung 10.8: Controldebugpanel

– Auf dem „Camerapanel“ sind 3 Buttons. Einmal für das machen eines Bildes mit dem Kamera und einmal für das Aufnehmen einer Film. Der Status der Buttons ist soweit implementiert, dass beim Filmen automatisch der Button für das machen eines Bildes ausgeschaltet wird und man den Kamera nur sagen kann, dass er anhalten soll mit Filmen.

Abbildung 10.9: KameraPanel

– Der „Calibrationpanel“ hat einige Textfelder wo man Werte einsetzen kann um die Sensoren zu kalibrieren. Es gibt die Möglichkeit sich die Initialwerte der Kalibrierung abzufragen aus dem Uboot. Diese werden dann in den jeweiligen Textfeld angezeigt. Um andere Werteschemas zu benützen hat man die Möglichkeit vorher gespeicherte Werte in ein Tekstdatei zu speichern und solche Dateien auch wieder öffnen zu können. Dieser Panel ist nur erreichbar unter den erweiterten Modus.

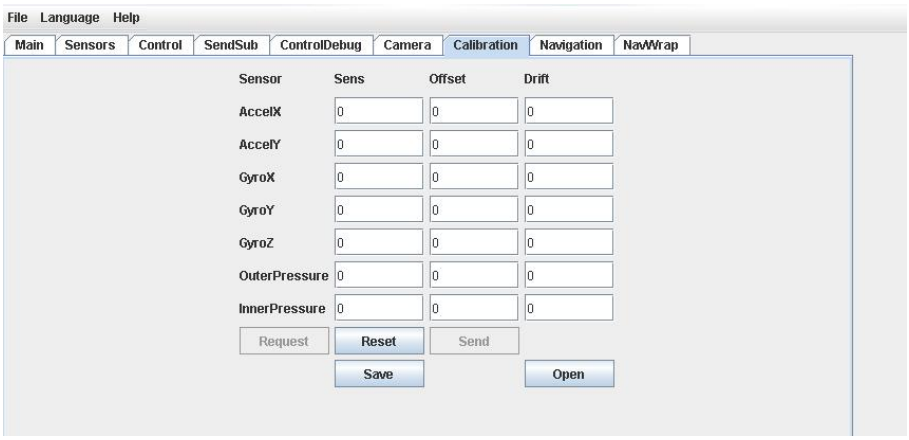


Abbildung 10.10: Calibrationpanel

– Im „Navigationpanel“ werden alle Navigationswerte angezeigt. Diese Werte werden allerdings erst angezeigt indem man eine Anfrage am Uboot schickt. („request“) Bei einer Navigationsreset werden alle Navigationswerte im Uboot auf „0“ gesetzt. Dieser Panel ist nur erreichbar über den erweiterten Modus.

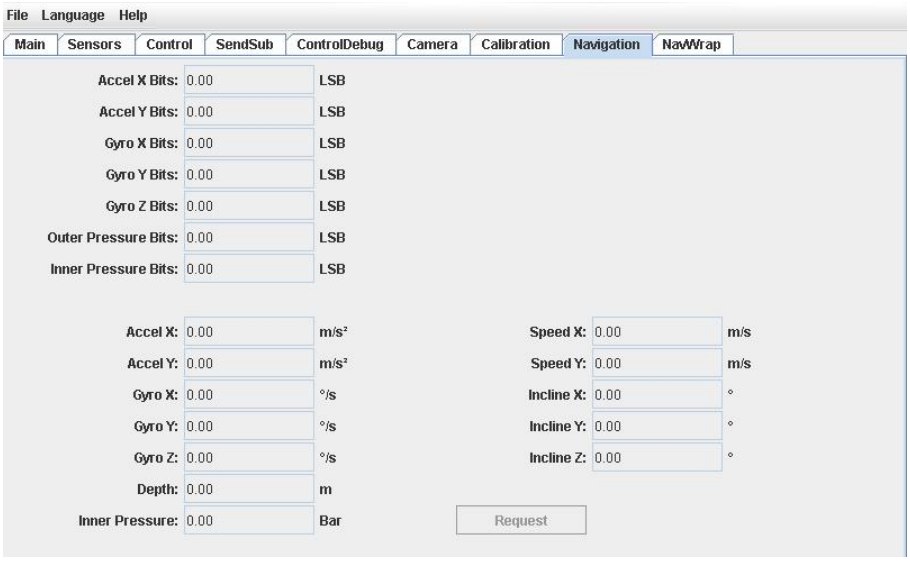


Abbildung 10.11: Navigationpanel

– Das „NavWrapPanel“ dient zum simulieren von Sensorwerten. Wir können zu bestimmte Sensoren (zu sehen in Abbildung 10.12) ein Werteverlauf simulieren.

Abbildung 10.12: navWrap

Will man simulieren, sollte man erst die Debugwerten einschalten. Man sollte sich dabei im Klaren sein, dass von alle Sensoren keine realen Werte mehr geliefert werden. Es werden dabei standartmäßig nur noch 0‘en zurück geliefert. Es wird die Möglichkeit geboten ein Startwert, Schrittwert (Anzahl der Schritte), Schrittzeit und die Änderung pro Schritt einzutragen. Hat man diese eingegeben, kann man diese in das Feld oben im Panel eintragen mit den Button „add“. Man kann jetzt mehrere Teilabschnitte machen zu den gewünschte Werteverlauf. Wenn man fertig ist, werden diese über den Button „Transfer“ abgeschickt zum Uboot mit ein Wiederholungswert, wie oft der Verlauf wiederholt werden sollte.

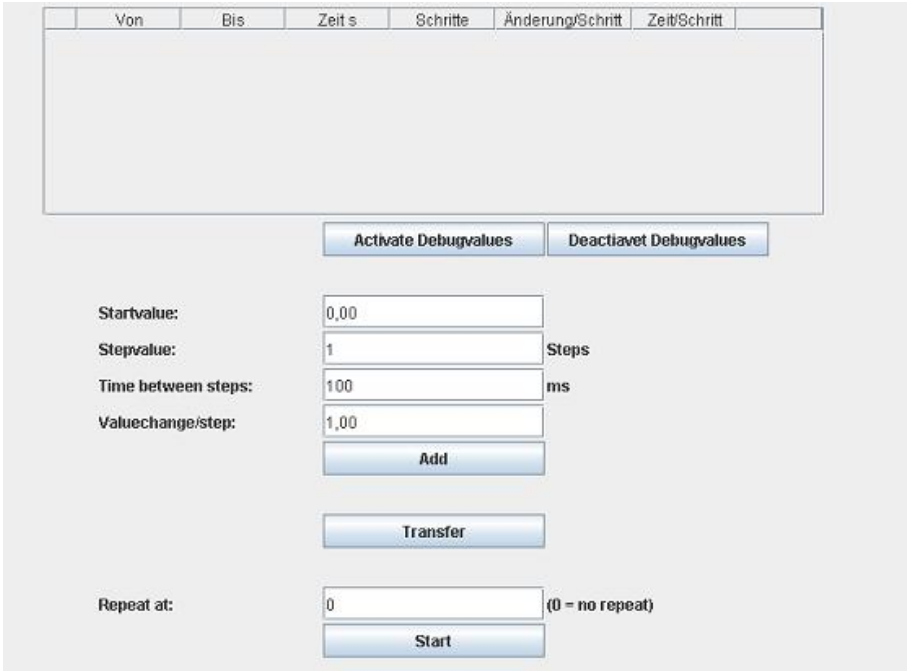


Abbildung 10.13: navWrap2

- Rechts vom Hauptpanel gibt es ein Feld mit den Standarsensoren und da drüber eine kleine Anzeige ob eine Verbindung zum Uboot über 433Mhz existiert. Die Sensoren die sich auf diesem Panel befinden müssen ständig im Auge behalten werden. Bei Ausgabe von rot angezeignete Werten sollte das Uboot aus dem Wasser geholt werden oder ist der Sensor nicht in Ordnung.

10.6 Ergänzungen

10.6.1 Vorschriften, Installation, usw.

Das Programm wird bei der Benützung und nicht vorhandene Verbindung die seriellen Ports (wenn vorhanden) scannen bis ein Kontakt mit der Plattform gefunden ist. Wenn kein seriellen Port gefunden wurde, wird es folgende Ausgabe geben:

```
No Connection trying again in 2 seconds
```

Bei der Benützung des Kommunikationsboxes sollte es allerdings mit eine externen Stromquelle versorgt sein. Bei keiner Verbindung mit dem Hostsystem ist die Benützung des Uboots nicht möglich, da die Steuerungen zuerst freigeschaltet werden muss. Man sollte im Controlpanel im Auge behalten ob ein Kommando richtig erfast wurde vom Uboot. Beim senden eines Kommandos im Moment indem die Verbindung unterbrochen wird, kann man davon ausgehen, dass dieses Kommando nicht empfangen wurde.

Bei Benützung einer Gamepad oder Joystick sollte der bevor das Starten des Programms angeschlossen werden. Ist dies nicht der Fall ist ein späteres anschliessen für das Programm nicht mehr möglich. Wenn man das Gamepad in dem Fall benutzen will, sollte man das Programm neustarten.

Symbolverzeichnis

Backbord linke eines Schiffes, von hinten aus gesehen

Bug Vordere Teil eines Schiffes

C++ Objektorientierte Erweiterung der Programmiersprache C

FPGA Field-Programmable Gate-Array, ein Gatterfeld, welches frei programmiert werden kann.

Heck Hintere Teil eines Schiffes

I^2C-Bus Ein weiteres Bussystem für digitale Schaltkreise

Java Plattformunabhängige Programmiersprache

Javadoc System, um aus den Kommentaren im Javaquellcode Dokumente zu erstellen

Logbuch Tagebuch für ein Schiff. Dort wird zum Beispiel die gefahrenen Kurse und Position, besondere Ereignisse eingetragen

Logge Die Logge besteht aus eine Impeller, mit dem das vorbeiströmende Wasser gemessen wird. Damit können Schiffe ihre Geschwindigkeit bestimmen.

LSB Least Significant Bit, Hier kleinste digitale Einheit, die gemessen werden kann

orthogonales Koordinatensystem Koordinatensystem, in dem die Basisvektoren senkrecht zueinander stehen

Schott Wand, die ein Schiff in mehrere Bereichen unterteilt.

SPI-Bus Serial Peripheral Interface, ein einfaches Bussystem für digitale Schaltkreise

Steuerbord rechte eines Schiffes, von hinten aus gesehen

USB-Port Schnittstelle zwischen dem FPGA und dem Computer, auf dem der FPGA programmiert wird

VHDL Hardwarebeschreibungsprache mit der zum Beispiel der FPGA programmiert wird

Literaturverzeichnis

[Ana02] Analog Devices Inc. *AD7490 16-Channel, 1 MSPS, 12-Bit ADC with Sequencer in 28-Lead TSSOP Data Sheet (REV. A)*, 2002.

[Ana03] Analog Devices Inc. *ADXRS300 300 °/s Single Chip Rate Gyro Evaluation Board Data Sheet (REV. 0)*, 2003.

[Ana04a] Analog Devices Inc. *ADXL103 ADXL203 Precision 1.7 g Single-Dual Axis Accelerometer Data Sheet (REV. 0)*, 2004.

[Ana04b] Analog Devices Inc. *ADXL203EB Dual Axis Accelerometer Evaluation Board Data Sheet (REV. 0)*, 2004.

[Ana04c] Analog Devices Inc. *ADXRS300 300 °/s Single Chip Yaw Rate Gyro with Signal Conditioning Data Sheet (REV. B)*, 2004.

[Ana05] Analog Devices Inc. *Dual-Axis 1.7 g Accelerometer with SPI Interface (REV. 0)*, 2005.

[Ana06a] Analog Devices Inc. *1.7g Dual-Axis Accelerometer, SPI Interface Evaluation Board (REV. PrA)*, 2006.

[Ana06b] Analog Devices Inc. *300°/s Yaw Rate Gyro with SPI Interface (REV. 0)*, 2006.

[Ana06c] Analog Devices Inc. *300°/sec Yaw Rate Gyro, SPI Interface Evaluation Board (REV.PrA)*, 2006.

[Beh05] Matthias Behrens. Einführung in die Inertialnavigation. In: *Projektgruppe Smart Systems 3: Submarine Exploring Seminar*, 2005.

[Brü92] Norbert Brüggen. *Technik der U-Boot-Modelle: Fernsteuerung, Tauchtechnik, Lageregelung, Sonderfunktionen*. Verlag für Technik und Handwerk, 3 Ausgabe, 1992.

[Hei04] Prof. Dr.-Ing. Andreas Hein. *Vorlesung Robotik*. Abteilung für Automatisierungs- und Messtechnik (AMT), Universität Oldenburg, 2004.

[Ine05] *Inertiales Navigationssystem Wikipedia*. http://de.wikipedia.org/wiki/Inertiales Navigationssystem, 10 2005.

[Kal60] Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[KDE] *K Desktop Environment*. http://www.kde.org/.

[Kop05] *Navigation Wikipedia*. http://de.wikipedia.org/wiki/Koppelnavigation, 10 2005.

[Kur05] *Kurs (Navigation) Wikipedia*. http://de.wikipedia.org/wiki/Kurs (Navigation), 10 2005.

[Law01] Anthony Lawrence. *Modern Inertial Technology*. Springer, 2 Ausgabe, 2001.

[Log05] *Logge Wikipedia*. http://de.wikipedia.org/wiki/Logge, 10 2005.

[Max96] Maxim Integrated Products. *MAX186/MAX188 Data Sheet*, 1996.

[Mot98] Motorola. *MPX4250 Series Data Sheet (REV 3)*, 1998.

[Mot02] Motorola. *MPX2100 Data Sheet*, 2002.

[Sil05] Silicon Microstructures, Inc. *SM5822/SM5872 Co-Integrated Pressure Sensor (REV 1.3 6_05*, 2005.

[Sun04] Sun Microsystems, Inc., http://java.sun.com/j2se/1.5.0/docs/api/. *JavaTM 2 Platform Standard Edition 5.0 API Specification*, 2004.

[Ull05] Christian Ullenboom. *Java ist auch eine Insel*. Galileo Computing, http://www.galileocomputing.de/openbook/javainsel5/, 5. Ausgabe, 2005.

[Ver06] *Versuch zur Corioliskraft*. http://htc.physik.hu-berlin.de/ mitdank/dist/scriptenm/coriolis2.htm, 2006.

[WB06] Greg Welch und Gary Bishop. An Introduction to the Kalman Filter. Technischer Bericht, University of North Carolina at Chapel Hill, 2006.

[Wika] *Corioliskraft*. http://de.wikipedia.org/wiki/Corioliskraft.

[Wikb] *Flaechentraegheitsmoment*. http://de.wikipedia.org/wiki/Flaechentraegheitsmoment.

[Wikc] *Metazentrum*. http://de.wikipedia.org/wiki/Metazentrum.

[Wikd] *Stroemungswiderstand*. http://de.wikipedia.org/wiki/Stroemungswiderstand.

[Wike] *Traegheitsmoment*. http://de.wikipedia.org/wiki/Traegheitsmoment.

[Wik05] Wikipedia, http://de.wikipedia.org/wiki/Navigation. *Navigation*, 10 2005.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang - Datenblätter

Dual-Axis ±1.7 *g* Accelerometer with SPI Interface

ADIS16003

FEATURES

Dual-axis accelerometer
SPI® digital output interface
Internal temperature sensor
Highly integrated; minimal external components; bandwidth externally selectable
1 m*g* resolution at 60 Hz
Externally controlled electrostatic self-test
3.0 V to 5.25 V single-supply operation
Low power: <2 mA
3500 *g* shock survival
7.2 mm × 7.2 mm × 3.6 mm package

APPLICATIONS

Industrial vibration/motion sensing
Platform stabilization
Dual-axis tilt sensing
Tracking, recording, analysis devices
Alarms, security devices

GENERAL DESCRIPTION

The ADIS16003 is a low cost, low power, complete dual-axis accelerometer with an integrated serial peripheral interface (SPI). An integrated temperature sensor is also available on the SPI interface. The ADIS16003 measures acceleration with a full-scale range of ±1.7 *g* (minimum), and it can measure both dynamic acceleration (vibration) and static acceleration (gravity).

The typical noise floor is 110 µg/√Hz, allowing signals below 1 m*g* (60 Hz bandwidth) to be resolved.

The bandwidth of the accelerometer is set with optional capacitors C_X and C_Y at the XFILT and YFILT pins. Selection of the two analog input channels is controlled via the serial interface.

An externally driven self-test pin (ST) allows the user to verify the accelerometer functionality.

The ADIS16003 is available in a 7.2 mm × 7.2 mm × 3.6 mm, 12-terminal LGA package.

FUNCTIONAL BLOCK DIAGRAM

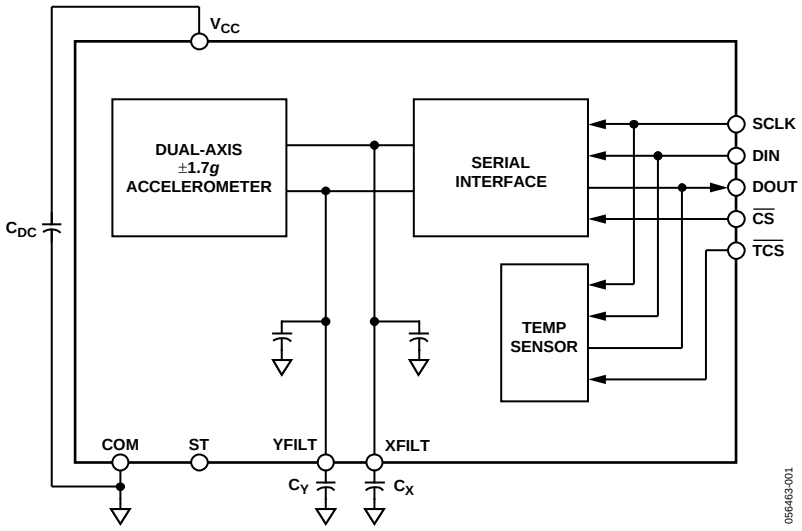


Figure 1.

Rev. 0

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.461.3113

TABLE OF CONTENTS

REVISION HISTORY

10/05—Revision 0: Initial Version

SPECIFICATIONS

T_A = −40°C to +125°C, V_{CC} = 5 V, C_X, C_Y = 0 µF, acceleration = 0 *g*, unless otherwise noted. All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.

Table 1.

Parameter	Conditions	Min	Typ	Max	Unit
ACCELEROMETER SENSOR INPUT	Each axis				
Measurement Range[1]		±1.7			*g*
Nonlinearity	% of full scale		±0.5	±2.5	%
Package Alignment Error			±1.5		degrees
Alignment Error	X sensor to Y sensor		±0.1		degrees
Cross Axis Sensitivity			±2	±5	%
ACCELEROMETER SENSITIVITY	Each axis				
Sensitivity at XFILT, YFILT		769	820	885	LSB/*g*
Sensitivity Change due to Temperature[2]	Delta from 25°C		±8		LSB
ZERO *g* BIAS LEVEL	Each axis				
0 *g* Voltage at XFILT, YFILT		1905	2048	2190	LSB
0 *g* Offset vs. Temperature			±0.14		LSB/°C
ACCELEROMETER NOISE PERFORMANCE					
Noise Density	@25°C		110		µ*g*/√Hz rms
ACCELEROMETER FREQUENCY RESPONSE[3]					
C_X, C_Y Range[4]		0		10	µF
R_{FILT} Tolerance		24	32	40	kΩ
Sensor Resonant Frequency			5.5		kHz
ACCELEROMETER SELF-TEST					
Logic Input Low				0.2 × V_{CC}	V
Logic Input High		0.8 × V_{CC}			V
ST Input Resistance to COM		30	50		kΩ
Output Change at X_{OUT}, Y_{OUT}[5]	Self-Test 0 to Self-Test 1	323	614	904	LSB
TEMPERATURE SENSOR					
Accuracy	V_{CC} = 3 V to 5.25 V		±2		°C
Resolution			10		Bits
Update Rate			400		µs
Temperature Conversion Time			25		µs
DIGITAL INPUT					
Input High Voltage (V_{INH})	V_{CC} = 4.75 V to 5.25 V	2.4			V
	V_{CC} = 3.0 V to 3.6 V	2.1			V
Input Low Voltage (V_{INL})	V_{CC} = 3.0 V to 5.25 V			0.8	V
Input Current	V_{IN} = 0 V or V_{CC}	-10	1	10	µA
Input Capacitance			10		pF
DIGITAL OUTPUT					
Output High Voltage (V_{OH})	I_{SOURCE} = 200 µA, V_{CC} = 3.0 V to 5.25 V	V_{CC} – 0.5			V
Output Low Voltage (V_{OL})	I_{SINK} = 200 µA			0.4	V
POWER SUPPLY					
Operating Voltage Range		3.0		5.25	V
Quiescent Supply Current	F_{SCLK} = 50 kSPS		1.5	2.0	mA
Power Down Current			1.0		mA
Turn-On Time[6]	C_x, C_y = 0.1 µF		20		Ms

[1] Guaranteed by measurement of initial offset and sensitivity.
[2] Defined as the output change from ambient to maximum temperature or ambient to minimum temperature.
[3] Actual bandwidth response controlled by user-supplied external capacitor (C_x, C_y).
[4] Bandwidth = 1/(2π x 32 kΩ x (2200 pF + C)). For C_x, C_y = 0, bandwidth = 2260 Hz. For C_x, C_y = 10 µF, bandwidth = 0.5 Hz. Min/max values not tested.
[5] Self-test response changes as the square of V_{cc}.
[6] Larger values of C_x, C_y increase turn-on time. Turn-on time is approximately 160 x (0.0022 µF + C_x + C_y) + 4 ms, where C_x, C_y are in µF.

TIMING SPECIFICATIONS

T_A = −40°C to +125°C, acceleration = 0 *g*, unless otherwise noted.

Table 2.

Parameter[1, 2]	V_{CC} = 3.3	V_{CC} = 5	Unit	Description
f_{SCLK}[3]	10	10	kHz min	
	2	2	MHz max	
$t_{CONVERT}$	14.5 t_{SCLK}	14.5 t_{CSLK}		
t_{ACQ}	1.5 t_{SCLK}	1.5 t_{SCLK}		Throughput time = $t_{CONVERT}$ + t_{ACQ} = 16 t_{SCLK}
t_1	10	10	ns min	$\overline{TCS}$/$\overline{CS}$ to SCLK setup time
t_2[4]	60	30	ns max	Delay from $\overline{TCS}$/$\overline{CS}$ until DOUT three-state disabled
t_3[4]	100	75	ns max	Data access time after SCLK falling edge
t_4	20	20	ns min	Data setup time prior to SCLK rising edge
t_5	20	20	ns min	Data hold time after SCLK rising edge
t_6	0.4 × t_{SCLK}	0.4 x t_{SCLK}	ns min	SCLK high pulse width
t_7	0.4 × t_{SCLK}	0.4 x t_{SCLK}	ns min	SCLK low pulse width
t_8[5]	80	80	ns max	$\overline{TCS}$/$\overline{CS}$ rising edge to DOUT high impedance
t_9	5	5	µs typ	Power-up time from shutdown

[1] Guaranteed by design. All input signals are specified with tr and tf = 5 ns (10% to 90% of V_{CC}) and timed from a voltage level of 1.6 V. The 3.3 V operating range spans from 3.0 V to 3.6 V. The 5 V operating range spans from 4.75 V to 5.25 V.

[2] See Figure 3 and Figure 4.

[3] Mark/space ratio for the SCLK input is 40/60 to 60/40.

[4] Measured with the load circuit in Figure 2 and defined as the time required for the output to cross 0.4 V or 2.0 V with V_{CC} = 3.3 V and time for an output to cross 0.8 V or 2.4 V with V_{CC} = 5.0 V.

[5] t_8 is derived from the measured time taken by the data outputs to change 0.5 V when loaded with the circuit in Figure 2. The measured number is then extrapolated back to remove the effects of charging or discharging the 50 pF capacitor. This means that the time, t_8, quoted in the timing characteristics is the true bus relinquish time of the part and is independent of the bus loading.

CIRCUIT AND TIMING DIAGRAMS

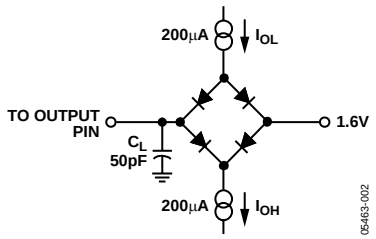


Figure 2. Load Circuit for Digital Output Timing Specifications

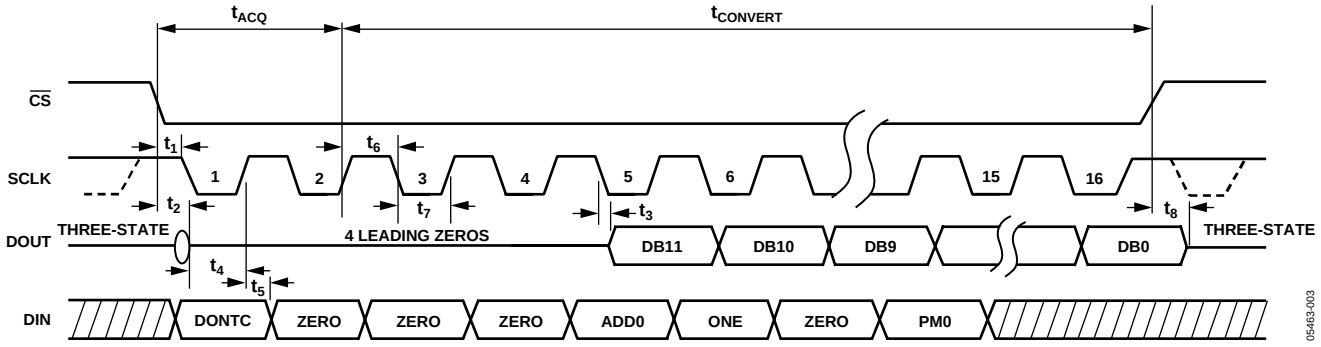


Figure 3. Accelerometer Serial Interface Timing Diagram

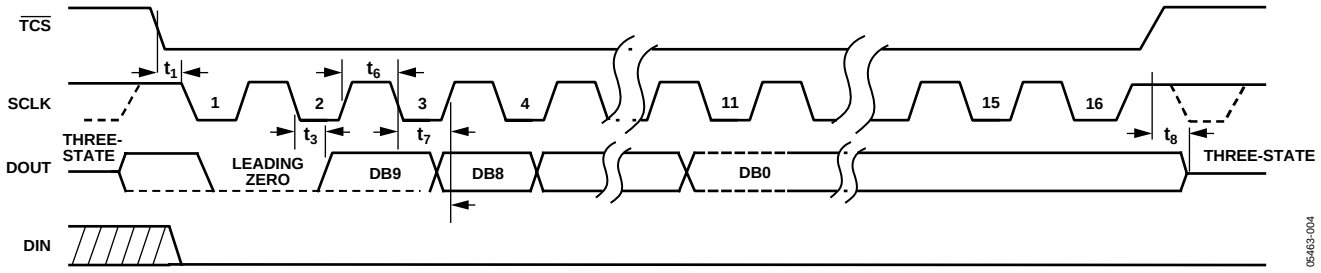


Figure 4. Temperature Serial Interface Timing Diagram

ABSOLUTE MAXIMUM RATINGS

Table 3.

Parameter	Rating
Acceleration (Any Axis, Unpowered)	3,500 *g*
Acceleration (Any Axis, Powered)	3,500 *g*
V_{CC}	–0.3 V to +7.0 V
All Other Pins	(COM – 0.3 V) to (V_{CC} + 0.3 V)
Output Short-Circuit Duration (Any Pin to Common)	Indefinite
Operating Temperature Range	–40°C to +125°C
Storage Temperature	–65°C to +150°C

Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Table 4. Package Characteristics

Package Type	θ_{JA}	θ_{JC}	Device Weight
12-Terminal LGA	200°C/W	25°C/W	0.3 grams

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Electrostatic charges as high as 4000 V readily accumulate on the human body and test equipment and can discharge without detection. Although this product features proprietary ESD protection circuitry, permanent damage may occur on devices subjected to high energy electrostatic discharges. Therefore, proper ESD precautions are recommended to avoid performance degradation or loss of functionality.

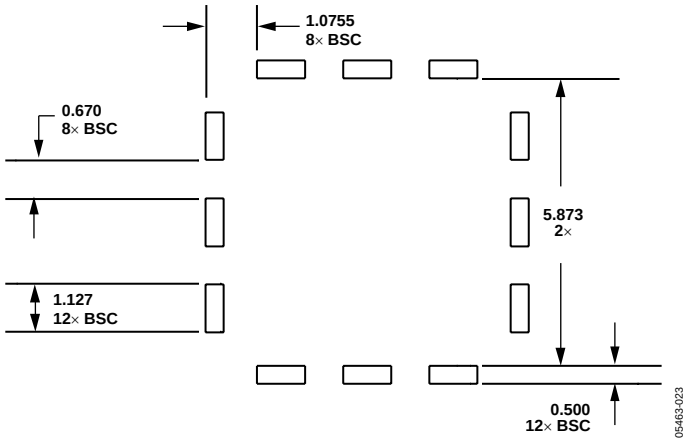


Figure 5. Second-Level Assembly Pad Layout

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

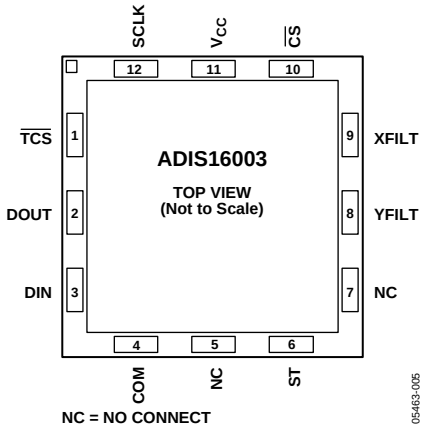


Figure 6. Pin Configuration

Table 5. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	$\overline{TCS}$	Temperature Chip Select. Active low logic input. This input frames the serial data transfer for the temperature sensor output.
2	DOUT	Data Out, Logic Output. The conversion of the ADIS16003 is provided on this output as a serial data stream. The bits are clocked out on the falling edge of the SCLK input.
3	DIN	Data In, Logic Input. Data to be written into the ADIS16003's control register is provided on this input and is clocked into the register on the rising edge of SCLK.
4	COM	Common. Reference point for all circuitry on the ADIS16003.
5, 7	NC	No Connect.
6	ST	Self-Test Input. Active high logic input. Simulates a nominal 0.75 *g* test input for diagnostic purpose.
8	YFILT	Y Channel Filter Node. Used in conjunction with an optional external capacitor to band-limit the ac signal from the accelerometer.
9	XFILT	X Channel Filter Node. Used in conjunction with an optional external capacitor to band-limit the ac signal from the accelerometer.
10	$\overline{CS}$	Chip Select. Active low logic input. This input provides the dual function of initiating the accelerometer conversions on the ADIS16003 and frames the serial data transfer for the accelerometer output.
11	V_{CC}	Power Supply Input. The V_{CC} range for the ADIS16003 is from 3.0 V to 5.25 V.
12	SCLK	Serial Clock, Logic Input. SCLK provides the serial clock for accessing data from the part and writing serial data to the control register. This clock input is also used as the clock source for the ADIS16003's conversion process.

TYPICAL PERFORMANCE CHARACTERISTICS

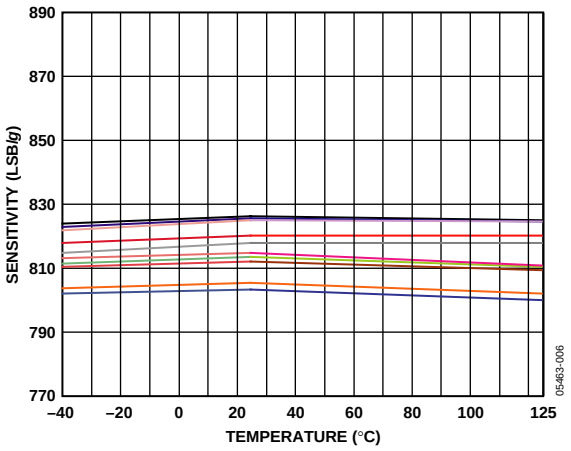


Figure 7. Sensitivity vs. Temperature (AD16003 Soldered to PCB)

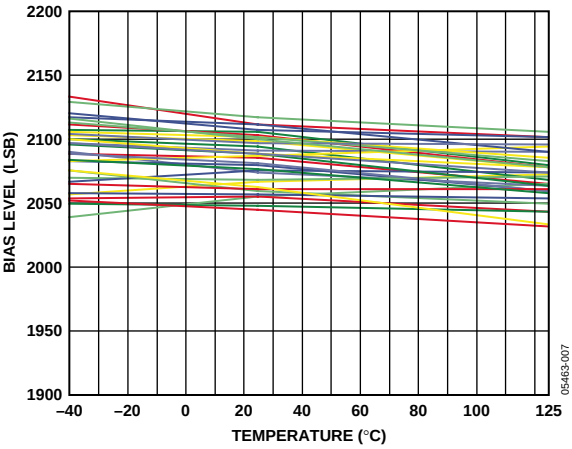


Figure 8. Zero g Bias vs. Temperature

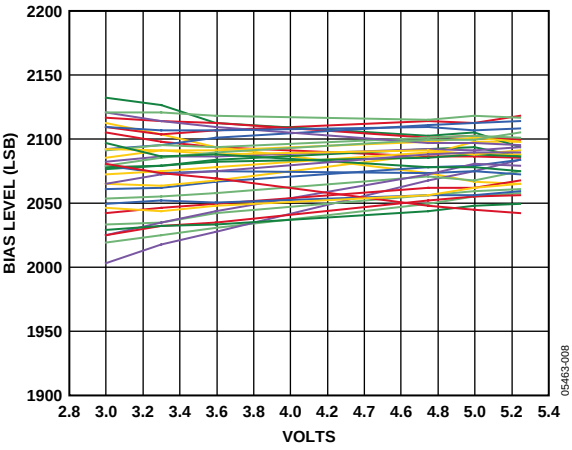


Figure 9. Zero g Bias vs. Supply

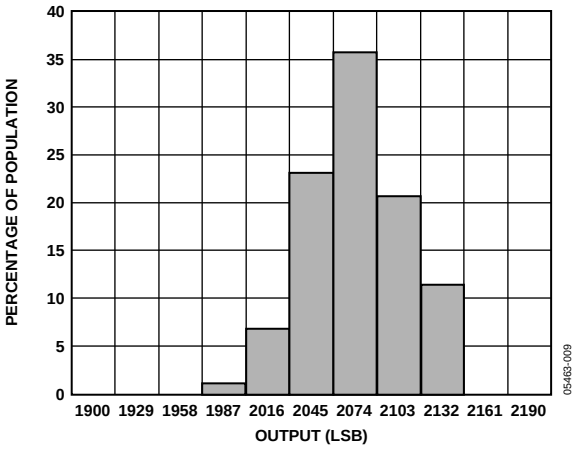


Figure 10. X-Axis Zero g Bias at 25°C

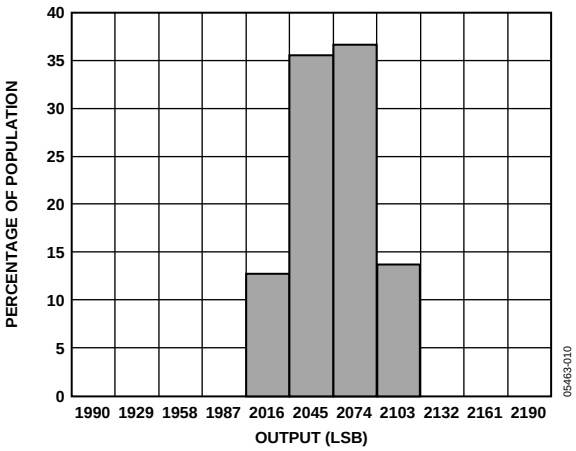


Figure 11. Y-Axis Zero g Bias at 25°C

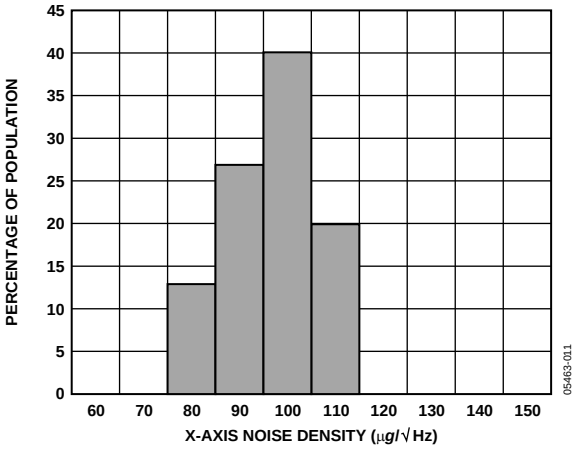


Figure 12. X-Axis Noise Density at 25°C

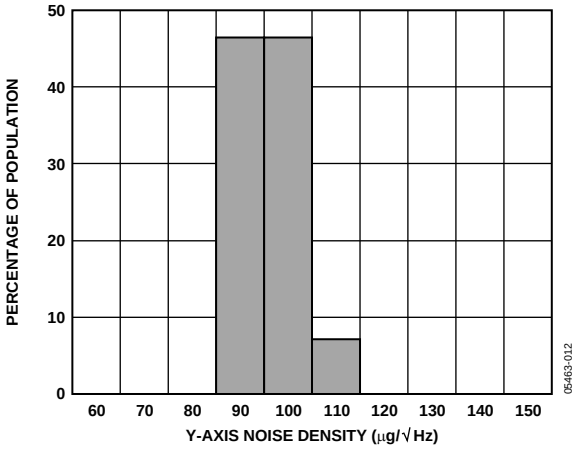


Figure 13. Y-Axis Noise Density at 25°C

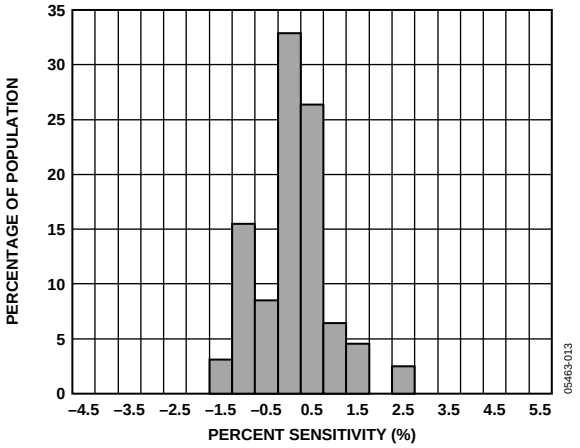


Figure 14. Z vs. X Cross-Axis Sensitivity

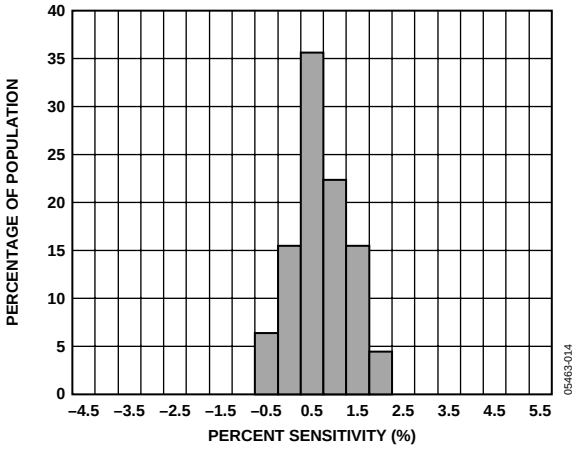


Figure 15. Z vs. Y Cross-Axis Sensitivity

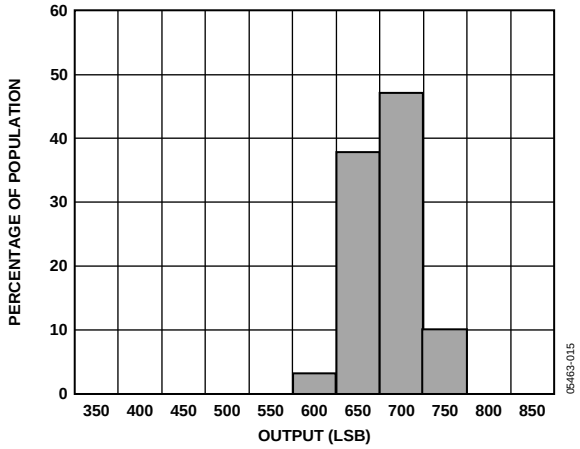


Figure 16. Self-Test at 25°C, V_{CC} at 5.0 V

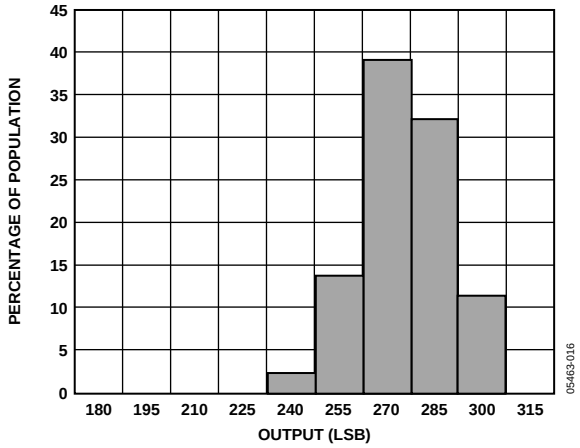


Figure 17. Self-Test at 25°C, V_{CC} at 3.3 V

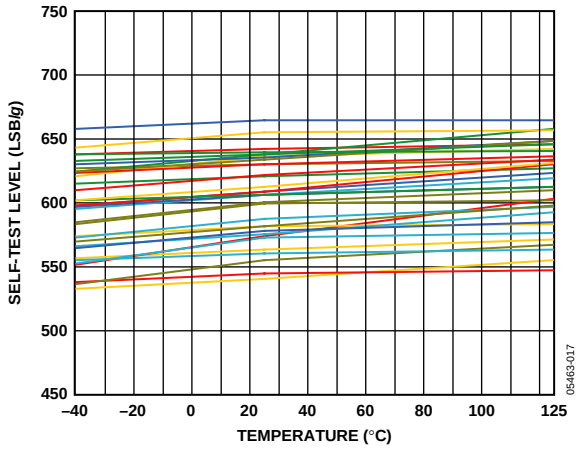


Figure 18. Self-Test vs. Temperature V_{CC} at 5.0 V

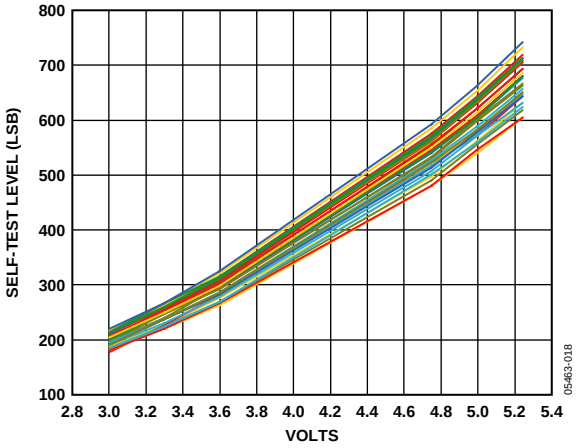


Figure 19. Self-Test vs. Supply Voltage

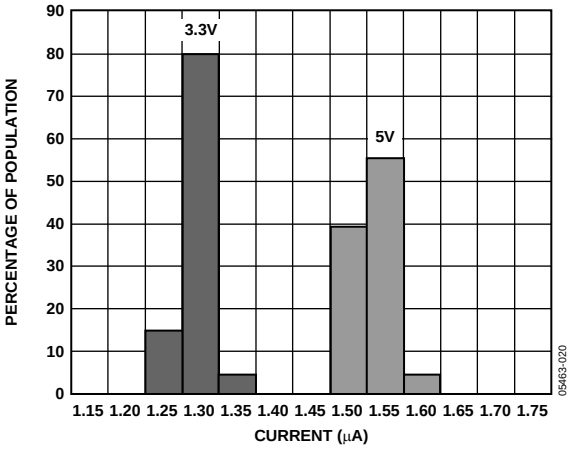


Figure 21. Supply Current at 25°C

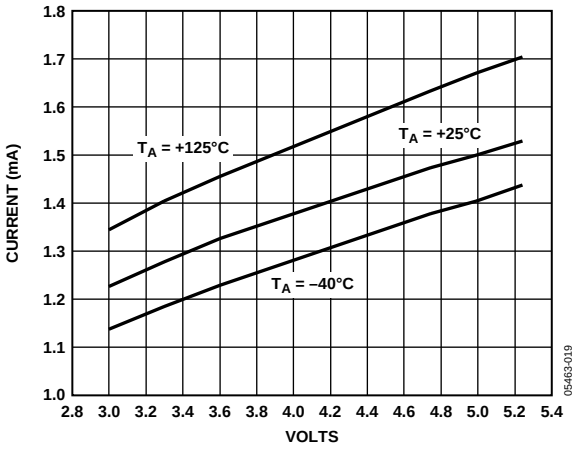


Figure 20. Supply Current vs. Supply Voltage

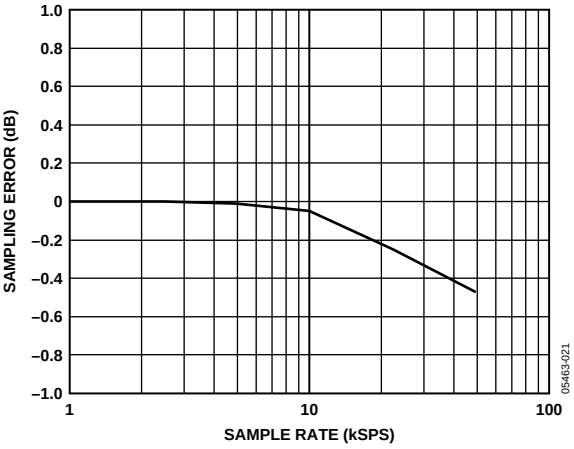


Figure 22. Sampling Error vs. Sample Rate

THEORY OF OPERATION

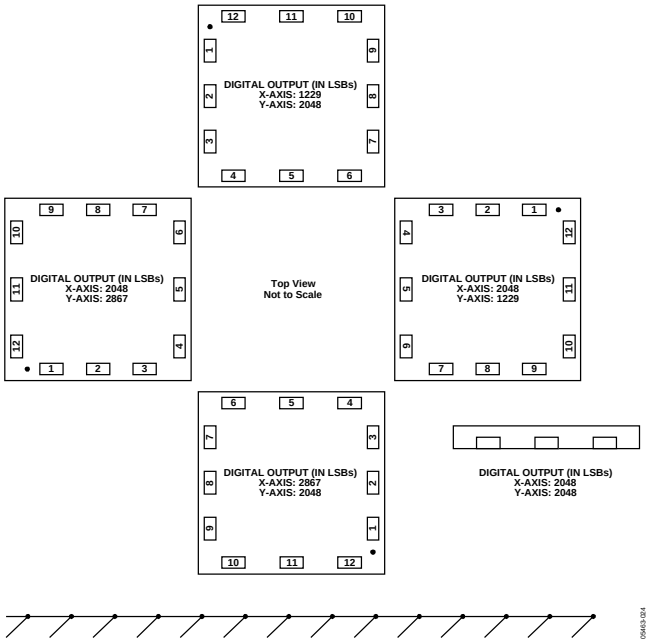


Figure 23. Output Response vs. Orientation

The ADIS16003 is a low cost, low power, complete dual-axis accelerometer with an integrated serial peripheral interface (SPI) and an integrated temperature sensor whose output is also available on the SPI interface. The ADIS16003 is capable of measuring acceleration with a full-scale range of ±1.7 *g* (minimum). It can also measure both dynamic acceleration (vibration) and static acceleration (gravity).

SELF-TEST

The ST pin controls the self-test feature. When this pin is set to V_{CC}, an electrostatic force is exerted on the beam of the accelerometer. The resulting movement of the beam allows the user to test if the accelerometer is functional. The typical change in output is 750 mg (corresponding to 614 LSB) for V_{CC} = 5.0 V. This pin may be left open-circuit or connected to common in normal use. The ST pin should never be exposed to voltage greater than V_{CC} + 0.3 V. If the system design is such that this condition cannot be guaranteed (for example, multiple supply voltages present), a low V_F clamping diode between ST and V_{CC} is recommended.

SERIAL INTERFACE

The serial interface on the ADIS16003 consists of five wires, $\overline{CS}$, $\overline{TCS}$, SCLK, DIN, and DOUT, with the temperature sensor's serial interface in parallel with the accelerometer's serial interface. The $\overline{CS}$ and $\overline{TCS}$ are used to select the accelerometer or temperature sensor outputs, respectively. $\overline{CS}$ and $\overline{TCS}$ cannot be active at the same time.

The SCLK input accesses data from the internal data registers.

ACCELEROMETER SERIAL INTERFACE

Figure 3 shows the detailed timing diagram for serial interfacing to the accelerometer in the ADIS16003. The serial clock provides the conversion clock. $\overline{CS}$ initiates the data transfer and conversion process and frames the serial data transfer for the accelerometer output. The accelerometer output is sampled on the second rising edge of the SCLK input after the falling edge of the $\overline{CS}$. The conversion requires 16 SCLK cycles to complete. The rising edge of $\overline{CS}$ puts the bus back into three-state. If $\overline{CS}$ remains low, the next digital conversion is initiated. The details for the control register bit functions are shown in Table 6.

Accelerometer Control Register

MSB							LSB
DONTC	ZERO	ZERO	ZERO	ADD0	ONE	ZERO	PM0

Table 6. Accelerometer Control Register Bit Functions

Bit	Mnemonic	Comments
7	DONTC	Don't care. Can be one or zero.
6, 5, 4	ZERO	These bits should be held low.
3	ADD0	This address bit selects the x-axis or y-axis outputs. Zero selects the x-axis; one selects the y-axis.
2	ONE	This bit should be held high.
1	ZERO	This bit should be held low.
0	PM0	This bit selects the operation mode for the accelerometer; set to zero for normal operation and one for power down mode.

Power Down

By setting PM0 to one when updating the accelerometer control register, the ADIS16003 goes into a shutdown mode. The information stored in the control register is maintained during shutdown. The ADIS16003 changes modes as soon as the control register is updated. If the part is in shutdown mode and PM0 is changed to zero, then the part powers up on the sixteenth SCLK rising edge.

ADD0

By setting ADD0 to zero when updating the accelerometer control register, the x-axis output is selected. By setting ADD0 to one, the y-axis output is selected.

ZERO

ZERO is defined as the logic low level.

ONE

ONE is defined as the logic high level.

DONTC

DONTC is defined as don't care; can be a low or high logic level.

Accelerometer Conversion Details

Every time the accelerometer is sampled, the sampling function discharges the internal C_X or C_Y filtering capacitors by up to 2% of their initial values (assuming no additional external filtering capacitors have been added). The recovery time for the filter capacitor to recharge is approximately 10 µs. Thus, sampling the accelerometer at a rate of 10 kSPS or less does not induce a sampling error. However, as sampling frequencies increase above 10 kSPS, one can expect sampling errors to attenuate the actual acceleration levels.

TEMPERATURE SENSOR SERIAL INTERFACE

Read Operation

Figure 4 shows the timing diagram for a serial read from the temperature sensor. The $\overline{TCS}$ line enables the SCLK input. Ten bits of data and a leading zero are transferred during a read operation. Read operations occur during streams of 16 clock pulses. The serial data is accessed in a number of bytes if 10 bits of data are being read. At the end of the read operation, the DOUT line remains in the state of the last bit of data clocked out until $\overline{TCS}$ goes high, at which time the DOUT line from the temperature sensor goes three-state.

Write Operation

Figure 4 also shows the timing diagram for the serial write to the temperature sensor. The write operation takes place at the same time as the read operation. Data is clocked into the control register on the rising edge of SCLK. DIN should remain low for the entire cycle.

Temperature Sensor Control Register

MSB							LSB
ZERO	ZERO	ZERO	ZERO	ZERO	ZERO	ZERO	ZERO

Table 7. Temperature Sensor Control Register Bit Functions

Bit	Mnemonic	Comments
7 to 0	ZERO	All bits should be held low.

ZERO

ZERO is defined as the logic low level.

Output Data Format

The output data format for the temperature sensor is twos complement. Table 8 shows the relationship between the digital output and the temperature.

Temperature Sensor Conversion Details

The ADIS16003 features a 10-bit digital temperature sensor that allows an accurate measurement of the ambient device temperature to be made.

The conversion clock for the temperature sensor is internally generated so no external clock is required except when reading from and writing to the serial port. In normal mode, an internal clock oscillator runs the automatic conversion sequence.

A conversion is initiated approximately every 350 µs. At this time, the temperature sensor wakes up and performs a temperature conversion. This temperature conversion typically takes 25 µs, at which time the temperature sensor automatically shuts down. The result of the most recent temperature conversion is available in the serial output register at any time. Once the conversion is finished, an internal oscillator starts counting and is designed to time out every 350 µs. The temperature sensor then powers up and does a conversion. Note that if the $\overline{TCS}$ is brought low every 350 µs (±30%) or less, then the same temperature value is output onto the DOUT line every time without changing. It is recommended that the $\overline{TCS}$ line not be brought low every 350 µs (±30%) or less. The ±30% covers process variation. The $\overline{TCS}$ should become active (high to low) outside this range.

The device is designed to auto convert every 350 µs. If the temperature sensor is accessed during the conversion process, an internal signal is generated to prevent any update of the temperature value register during the conversion. This prevents the user from reading back spurious data. The design of this feature results in this internal lockout signal being reset only at the start of the next auto conversion. Therefore, if the $\overline{TCS}$ line goes active before the internal lockout signal is reset to its inactive mode, the internal lockout signal is not reset. To ensure that no lockout signal is set, bring $\overline{TCS}$ low at a greater time than 350 µs (±30%). As a result, the temperature sensor is not interrupted during a conversion process.

In the automatic conversion mode, every time a read or write operation takes place, the internal clock oscillator is restarted at the end of the read or write operation. The result of the conversion is typically available 25 µs later. Reading from the device before conversion is complete provides the same set of data.

Table 8. Temperature Sensor Data Format

Temperature	Digital Output (DB9 … DB0)
–40°C	11 0110 0000
–25°C	11 1001 1100
–0.25°C	11 1111 1111
0°C	00 0000 0000
+0.25°C	00 0000 0001
+10°C	00 0010 1000
+25°C	00 0110 0100
+50°C	00 1100 1000
+75°C	01 0010 1100
+100°C	01 1001 0000
+125°C	01 1111 0100

POWER SUPPLY DECOUPLING

For most applications, a single 0.1 µF capacitor (C_{DC}) adequately decouples the accelerometer from noise on the power supply. However, in some cases, particularly where noise is present at the 140 kHz internal clock frequency (or any harmonic thereof), noise on the supply may cause interference on the ADIS16003 output. If additional decoupling is needed, ferrite beads may be inserted in the supply line of the ADIS16003. Additionally, a larger bulk bypass capacitor (in the 1 µF to 22 µF range) may be added in parallel to C_{DC}.

SETTING THE BANDWIDTH USING C_{XFILT} AND C_{YFILT}

The ADIS16003 has provisions for band-limiting the accelerometer. Capacitors can be added at the XFILT and YFILT pins to implement further low-pass filtering for antialiasing and noise reduction. The equation for the 3 dB bandwidth is

$$F_{-3dB} = 1/(2\pi(32\ k\Omega) \times (C_{(XFILT,\ YFILT)} + 2200\ pF))$$

or more simply,

$$F_{-3dB} = 5\ \mu F/(C_{(XFILT,\ YFILT)} + 2200\ pF)$$

The tolerance of the internal resistor (R_{FILT}) can vary typically as much as ±25% of its nominal value (32 kΩ); thus, the bandwidth varies accordingly.

A minimum capacitance of 0 pF for C_{XFILT} and C_{YFILT} is allowable.

Table 9. Filter Capacitor Selection, C_{XFILT} and C_{YFILT}

Bandwidth (Hz)	Capacitor (µF)
1	4.7
10	0.47
50	0.10
100	0.047
200	0.022
400	0.01
2250	0

SELECTING FILTER CHARACTERISTICS: THE NOISE/BANDWIDTH TRADE-OFF

The accelerometer bandwidth selected ultimately determines the measurement resolution (smallest detectable acceleration). Filtering can be used to lower the noise floor, which improves the resolution of the accelerometer. Resolution is dependent on the analog filter bandwidth at XFILT and YFILT.

The ADIS16003 has a typical bandwidth of 2.25 kHz with no external filtering. The analog bandwidth may be further decreased to reduce noise and improve resolution.

The ADIS16003 noise has the characteristics of white Gaussian noise, which contributes equally at all frequencies and is described in terms of µg/√Hz (that is, the noise is proportional to the square root of the accelerometer's bandwidth). The user should limit bandwidth to the lowest frequency needed by the application in order to maximize the resolution and dynamic range of the accelerometer.

With the single pole roll-off characteristic, the typical noise of the ADIS16003 is determined by

$$rmsNoise = (110\ \mu g/\sqrt{Hz}) \times (\sqrt{(BW \times 1.6)})$$

At 100 Hz, the noise is

$$rmsNoise = (110\ \mu g/\sqrt{Hz}) \times (\sqrt{(100 \times 1.6)}) = 1.4\ mg$$

Often, the peak value of the noise is desired. Peak-to-peak noise can only be estimated by statistical methods. Table 10 is useful for estimating the probabilities of exceeding various peak values, given the rms value.

Table 10. Estimation of Peak-to-Peak Noise

Peak-to-Peak Value	Percentage of Time that Noise Exceeds Nominal Peak-to-Peak Value
2 × rms	32%
4 × rms	4.6%
6 × rms	0.27%
8 × rms	0.006%

APPLICATIONS

DUAL-AXIS TILT SENSOR

One of the most popular applications of the ADIS16003 is tilt measurement. An accelerometer uses the force of gravity as an input vector to determine the orientation of an object in space. An accelerometer is most sensitive to tilt when its sensitive axis is perpendicular to the force of gravity, that is, parallel to the earth's surface. At this orientation, its sensitivity to changes in tilt is highest. When the accelerometer is oriented on axis to gravity, near its +1 *g* or −1 *g* reading, the change in output acceleration per degree of tilt is negligible. When the accelerometer is perpendicular to gravity, its output changes nearly 17.5 m*g* per degree of tilt. At 45°, its output changes at only 12.2 m*g* per degree, and resolution declines.

Converting Acceleration to Tilt

When the accelerometer is oriented so both its x-axis and y-axis are parallel to the earth's surface, it can be used as a 2-axis tilt sensor with a roll axis and a pitch axis. Once the output signal from the accelerometer has been converted to an acceleration that varies between −1 *g* and +1 *g*, the output tilt in degrees is calculated as follows:

$$PITCH = A\sin(A_X/1\ g)$$

$$ROLL = A\sin(A_Y/1\ g)$$

Be sure to account for overranges. It is possible for the accelerometers to output a signal greater than ±1 *g* due to vibration, shock, or other accelerations.

SECOND-LEVEL ASSEMBLY

The ADIS16003 may be attached to the second-level assembly board using SN63 (or equivalent) or lead-free solder. Figure 24 and Table 11 provide acceptable solder reflow profiles for each solder type. Note: These profiles may not be the optimum profile for the user's application. In no case should 260°C be exceeded. It is recommended that the user develop a reflow profile based upon the specific application. In general, keep in mind that the lowest peak temperature and shortest dwell time above the melt temperature of the solder results in less shock and stress to the product. In addition, evaluating the cooling rate and peak temperature can result in a more reliable assembly.

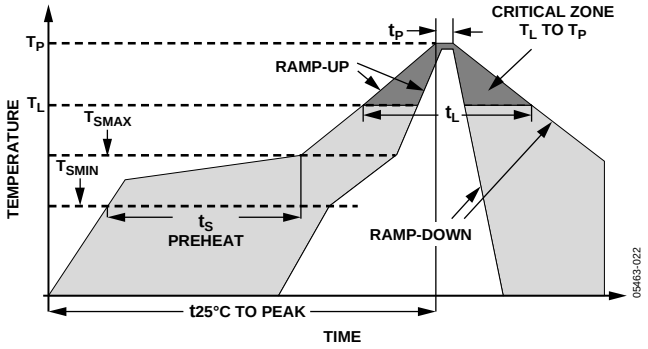


Figure 24. Acceptable Solder Reflow Profiles

Table 11.

	Condition	
Profile Feature	**Sn63/Pb37**	**Pb-free**
Average Ramp Rate (T_L to T_P)	3°C/sec max	3°C/sec max
Preheat		
Minimum Temperature (T_{SMIN})	100°C	150°C
Maximum Temperature (T_{SMAX})	150°C	200°C
Time (T_{SMIN} to T_{SMAX}) (t_S)	60 sec to 120 sec	60 sec to 150 sec
T_{SMAX} to T_L		
Ramp-Up Rate	3°C/sec	3°C/sec
Time Maintained Above Liquidous (T_L)		
Liquidous Temperature (T_L)	183°C	217°C
Time (t_L)	60 sec to 150 sec	60 sec to 150 sec
Peak Temperature (T_P)	240°C + 0°C/−5°C	260°C + 0°C/−5°C
Time Within 5°C of Actual Peak Temperature (t_P)	10 sec to 30 sec	20 sec to 40 sec
Ramp-Down Rate	6°C/sec max	6°C/sec max
Time 25°C to Peak Temperature	6 min max	8 min max

OUTLINE DIMENSIONS

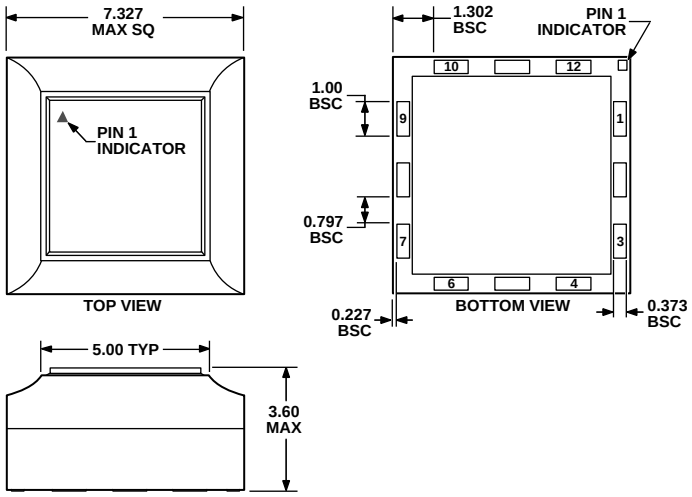


Figure 25. 12-Terminal Land Grid Array [LGA]
(CC-12)
Dimensions shown in millimeters

ORDERING GUIDE

Model	Temperature Range	Package Description	Package Option
ADIS16003CCCZ [1]	−40°C to +125°C	12-Terminal Land Grid Array (LGA)	CC-12
ADIS16003/PCB		Evaluation Board	

[1] Z = Pb-free part.

NOTES

D05463-0-10/05(0)

±1.7g Dual-Axis Accelerometer, SPI Interface Evaluation Board

Preliminary Technical Data

ADIS16003/PCB

GENERAL DESCRIPTION

The ADIS16003/PCB is an evaluation board that has been designed to provide simple access to the ADIS16003 using standard connectors. These connectors can be accessed using a variety of cable options, including standard 0.1" ribbon cables. While this evaluation board has been designed to fit into the ADIS-EVAL-PCB system, it can also be mounted directly to a system's platform as well. All of the components are on the top side and four mounting holes (sized for 2-56 or 2mm screws) have been provided to secure the board during evaluation.

CIRCUIT DESCRIPTION

The schematic, layout and parts list for the ADIS16003/PCB can be found in Figure 1, Figure 2, and Table 1.

The ADIS16003's digitized outputs can be accessed using the 4-wire serial port interface (SPI) signals on J1: SCLK, CS, DOUT and DIN. For specific information on using the ADIS16003's SPI interface, refer to the ADIS16003 datasheet. C1 is not populated but provides opportunity to add a filtering capacitor for helping filter noisy power supply inputs. C2 and C3 are not installed either but provide the ability to reduce the bandwidth on each axis. See the ADIS16003 datasheet for more details.

Table 1 – ADIS16003/PCB Parts List

Reference Designator	Part Description
U1	ADIS16003CCCZ
J1,J2	Connector, 12-pin, dual row, 2mm
C1	Filtering external power supply noise. Not installed
C2, C3	Additional Filtering of the output response, Not Installed

SPECIAL NOTES ON HANDLING

Note that the ADIS16003/PCB is not reverse polarity protected. Reversing the power supply or applying inappropriate voltages to any pin (outside the Absolute Maximum Ratings in the ADIS16003 data sheet) may damage the ADIS16003/PCB.

Table 2 – Power Supply Levels

Vcc	*+3.0V to +5.25V*

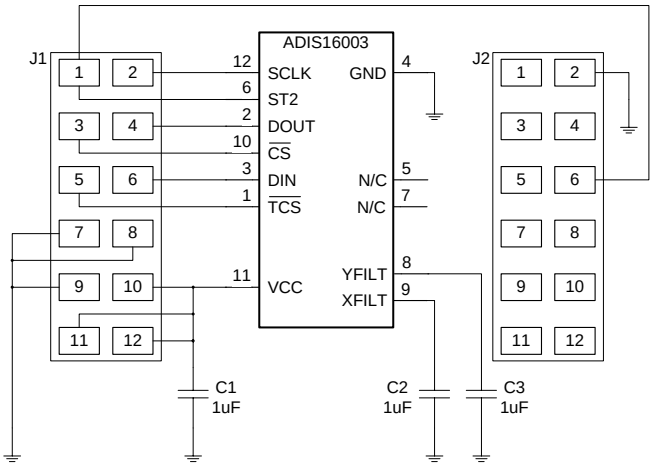


Figure 1 - ADIS16003/PCB Schematic

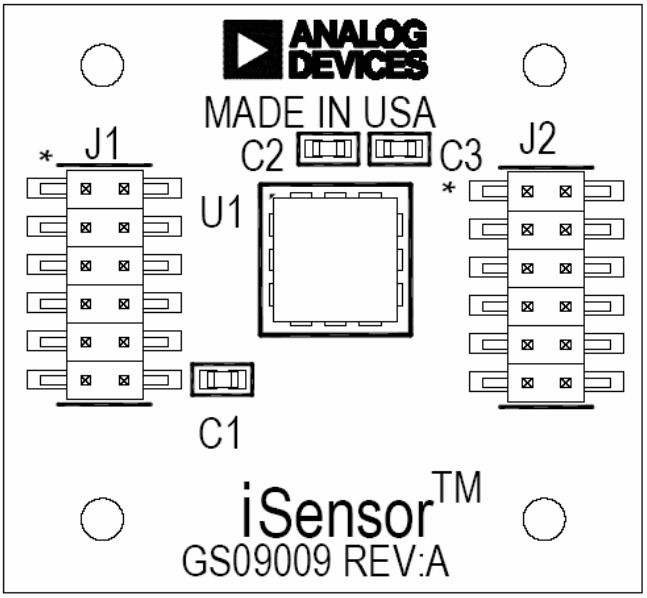


Figure 2 - ADIS16003/PCB Layout (Top View)

ORDERING GUIDE

Model	Package Description
ADIS16003/PCB	Evaluation Board

Rev. PrA **2/13/2006**

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.326.8703

±300°/s Yaw Rate Gyro with SPI Interface

ADIS16100

FEATURES

Complete angular rate gyroscope
Z-axis (yaw rate) response
SPI® digital output interface
High vibration rejection over wide frequency
2000 *g* powered shock survivability
Externally controlled self test
Internal temperature sensor output
Dual auxiliary 12-bit ADC inputs
Absolute rate output for precision applications
5 V single-supply operation
8.2 mm × 8.2 mm × 5.2 mm package

APPLICATIONS

Platform stabilization
Image stabilization
Guidance and control
Inertial measurement units

GENERAL DESCRIPTION

The ADIS16100 is a complete angular rate sensor (gyroscope) that uses Analog Devices' surface-micromachining process to make a functionally complete angular rate sensor with an integrated serial peripheral interface (SPI).

The digital data available at the SPI port is proportional to the angular rate about the axis normal to the top surface of the package (see Figure 19). A single external resistor can be used to increase the measurement range. An external capacitor can be used to lower the bandwidth.

Access to an internal temperature sensor measurement is provided, through the SPI, for compensation techniques. Two pins are available to the user to input analog signals for digitization. An additional output pin provides a precision voltage reference. Two digital self-test inputs electro-mechanically excite the sensor to test operation of the sensor and the signal conditioning circuits.

The ADIS16100 is available in an 8.2 mm × 8.2 mm × 5.2 mm 16-terminal, peripheral land grid array (LGA) package.

FUNCTIONAL BLOCK DIAGRAM

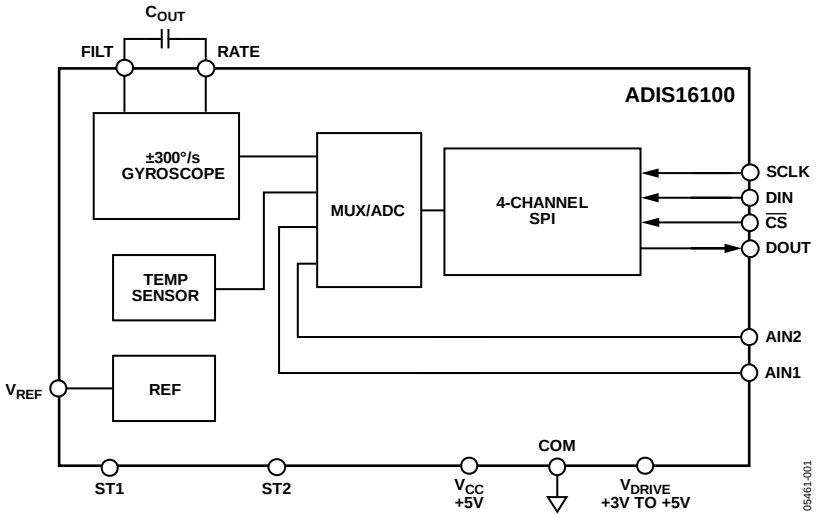


Figure 1.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 www.analog.com
Fax: 781.461.3113

TABLE OF CONTENTS

REVISION HISTORY

1/06—Revision 0: Initial Version

SPECIFICATIONS

T_A = 25°C, V_{CC} = V_{DR} = 5 V, angular rate = 0°/sec, C_{OUT} = 0 µF, ±1 *g*, unless otherwise noted.

Table 1.

Parameter	Conditions	Min[1]	Typ	Max[1]	Unit
SENSITIVITY	Clockwise rotation is positive output				
Dynamic Range[2]	Full-scale range over specifications range	±300			°/s
Initial	@25°C	3.68	4.1	4.52	LSB/°/s
Change over Temperature[3]	V_{CC} = V_{DR} = 4.75 V to 5.25 V		±10		%
Nonlinearity	Best fit straight line		0.15		% of FS
NULL					
Initial Null		1843	2048	2253	LSB
Change Over Temperature[3]	V_{CC} = V_{DR} = 4.75 V to 5.25 V		±205		LSB
Turn-On Time	Power on to ±½°/s of final		75		ms
Linear Acceleration Effect	Any axis		0.82		LSB/*g*
Voltage Sensitivity	V_{CC} = V_{DR} = 4.75 V to 5.25 V		4.1		LSB/V
NOISE PERFORMANCE	0.1 Hz to 40 Hz		3.25		LSB rms
Rate Noise Density	f = 100 Hz		0.43		LSB rms/√Hz
FREQUENCY RESPONSE					
3 dB Bandwidth (User-Selectable)[4]	C_{OUT} = 0 µF		40		Hz
Sensor Resonant Frequency			14		kHz
SELF-TEST INPUTS					
ST1 RATEOUT Response[5]	ST1 pin from Logic 0 to Logic 1	−121	−221	−376	LSB
ST2 RATEOUT Response[5]	ST2 pin from Logic 0 to Logic 1	+121	+221	+376	LSB
Logic 1 Input Voltage	Standard high logic level definition	3.3			V
Logic 0 Input Voltage	Standard low logic level definition			1.7	V
Input Impedance	To common		50		kΩ
TEMPERATURE SENSOR					
Reading at 298°K			2048		LSB
Scale Factor	Proportional to absolute temperature		6.88		LSB/°K
2.5 V REFERENCE					
Voltage Value		2.45	2.5	2.55	V
Load Drive to Ground	Source		100		µA
Load Regulation	0 µA < I_{OUT} < 100 µA		5.0		mV/mA
Power Supply Rejection	V_{CC} = V_{DR} = 4.75 V_{CC} to 5.25		1.0		mV/V
Temperature Drift	Delta from 25°C		5.0		mV
LOGIC INPUTS					
Input High Voltage, V_{INH}		0.7 × V_{DRIVE}			V
Input Low Voltage, V_{INL}				0.3 × V_{DRIVE}	V
Input Current, I_{IN}	Typically 10 nA	−1		+1	µA
Input Capacitance, C_{IN}			10		pF
ANALOG INPUTS[6]	All at T_A = −40°C to +85°C				
Resolution			12		Bits
Integral Nonlinearity[6]		−2		+2	LSB
Differential Nonlinearity		−2		+2	LSB
Offset Error		−8		+8	LSB
Gain Error		−2		+2	%FSR
Input Voltage Range		0		VREF × 2	V
Leakage Current		−1		+1	µA
Input Capacitance			20		pF
Full Power Bandwidth			8		MHz

Parameter	Conditions	Min[1]	Typ	Max[1]	Unit
DIGITAL OUTPUTS					
Output High Voltage (V_{OH})	I_{SOURCE} = 200 µA	V_{DRIVE} − 0.2			V
Output Low Voltage (V_{OL})	I_{SINK} = 200 µA			0.4	V
CONVERSION RATE					
Conversion Time	16 SCLK cycles with SCLK at 20 MHz			800	ns
Throughput Rate				1	MSPS
POWER SUPPLY	All at T_A = −40°C to +85°C				
V_{CC}		4.75	5	5.25	V
V_{DRIVE}		2.7		5.25	V
V_{CC} Quiescent Supply Current	V_{CC} @ 5 V, f_{SCLK} = 50 kSPS		7.0	9.0	mA
V_{DRIVE} Quiescent Supply Current	V_{DRIVE} @ 5 V, f_{SCLK} = 50 kSPS		70	500	µA
Power Dissipation	V_{CC} and V_{DRIVE} @ 5 V, f_{SCLK} = 50 kSPS		40		mW

[1] All minimum and maximum specifications are guaranteed. Typical specifications are neither tested nor guaranteed.
[2] Dynamic range is the maximum full-scale measurement range possible, including output swing range, initial offset, sensitivity, offset drift, and sensitivity drift at 5 V supplies.
[3] Defined as the output change from ambient to maximum temperature or ambient to minimum temperature.
[4] Frequency at which the response is 3 dB down from dc response. Bandwidth = $1/(2 \times \pi \times 180\ k\Omega \times (22\ nF + C_{OUT}))$. For C_{OUT} = 0, bandwidth = 40 Hz. For C_{OUT} = 1 µF, bandwidth = 0.87 Hz.
[5] Self-test response varies with temperature.
[6] For $V_{IN} < V_{CC}$.

TIMING DIAGRAM

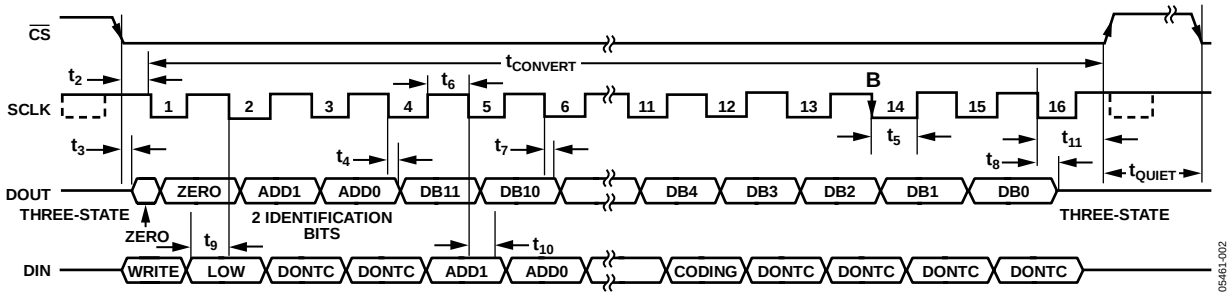


Figure 2. Gyroscope Serial Interface Timing Diagram

The DIN bit functions are outlined in the following table (see the Control Register section for additional information).

Table 2. DIN Bit Functions

MSB (11)											**LSB (0)**
WRITE	LOW	DONTC	DONTC	ADD1	ADD0	HIGH	HIGH	DONTC	DONTC	LOW	CODING

TIMING SPECIFICATIONS

T_A = 25°C, angular rate = 0°/sec, unless otherwise noted.[1]

Table 3.

Parameter	$V_{CC} = V_{DR} = 5$	Unit	Description
f_{SCLK}[2]	10	kHz min	
	20	MHz max	
$t_{CONVERT}$	16 × t_{SCLK}		
t_{QUIET}	50	ns min	Minimum quiet time required between $\overline{CS}$ rising edge and start of next conversion
t_2	10	ns min	$\overline{CS}$ to SCLK setup time
t_3[3]	30	ns max	Delay from $\overline{CS}$ until DOUT three-state disabled
t_4[3]	40	ns max	Data access time after SCLK falling edge
t_5	0.4 × t_{SCLK}	ns min	SCLK low pulse width
t_6	0.4 × t_{SCLK}	ns min	SCLK high pulse width
t_7	10	ns min	SCLK to DOUT valid hold time
t_8[4]	15/35	ns min/max	SCLK falling edge to DOUT high impedance
t_9	10	ns min	DIN setup time prior to SCLK falling edge
t_{10}	5	ns min	DIN hold time after SCLK falling edge
t_{11}	20	ns min	16th SCLK falling edge to $\overline{CS}$ high
t_{12}	1	µs max	Power-up time from full power-down/auto shutdown modes

[1] Guaranteed by design. All input signals are specified with t_r and t_f = 5 ns (10% to 90% of V_{CC}) and timed from a voltage level of 1.6 V. The 5 V operating range spans from 4.75 V to 5.25 V.
[2] Mark/space ratio for the SCLK input is 40/60 to 60/40.
[3] Measured with the load circuit in Figure 3 and defined as the time required for the output to cross 0.4 V or 0.7 V × V_{DRIVE}.
[4] t_8 is derived from the measured time taken by the data outputs to change 0.5 V when loaded with the circuit in Figure 3. The measured number is then extrapolated back to remove the effects of charging or discharging the 50 pF capacitor. This means that the time, t_8, quoted in the timing characteristics is the true bus relinquish time of the part and is independent of the bus loading.

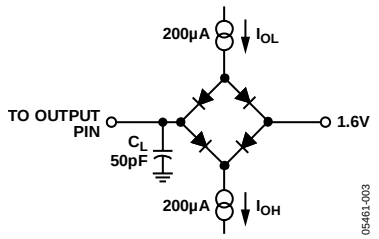


Figure 3. Load Circuit for Digital Output Timing Specifications

ABSOLUTE MAXIMUM RATINGS

Table 4.

Parameter	Rating
Acceleration (Any Axis, Unpowered, 0.5 ms)	2000 *g*
Acceleration (Any Axis, Powered, 0.5 ms)	2000 *g*
$+V_{CC}$ to COM	−0.3 V to +6.0 V
$+V_{DRIVE}$ to COM	−0.3 V to V_{CC} + 0.3 V
Analog Input Voltage to COM	−0.3 V to V_{CC} + 0.3 V
Digital Input Voltage to COM	−0.3 V to +7.0 V
Digital Output Voltage to COM	−0.3 V to V_{CC} + 0.3 V
STx Input Voltage to COM	−0.3 V to V_{CC} + 0.3 V
Operating Temperature Range	−40°C to +85°C
Storage Temperature Range	−65°C to +150°C

Stresses above those listed under the Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Drops onto hard surfaces can cause shocks of greater than 2000 *g* and exceed the absolute maximum rating of the device. Care should be exercised in handling to avoid damage.

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Electrostatic charges as high as 4000 V readily accumulate on the human body and test equipment and can discharge without detection. Although this product features proprietary ESD protection circuitry, permanent damage may occur on devices subjected to high energy electrostatic discharges. Therefore, proper ESD precautions are recommended to avoid performance degradation or loss of functionality.

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

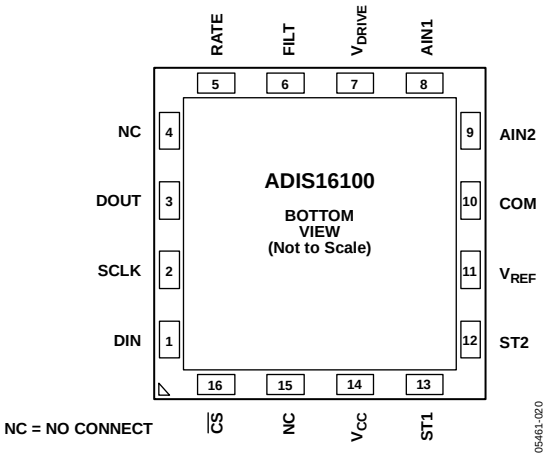


Figure 4. Pin Configuration

Table 5. Pin Function Descriptions

Pin No.	Mnemonic	Type[1]	Description
1	DIN	I	Data In. Data to be written to the control register is provided on this input and is clocked in on the falling edge of the SCLK.
2	SCLK	I	Serial Clock. SCLK provides the serial clock for accessing data from the part and writing serial data to the control registers. Also used as a clock source for the ADIS16100 conversion process.
3	DOUT	O	Data Out. The data on this pin represents data being read from the control registers and is clocked on the falling edge of the SCLK.
4	NC		No Connect.
5	RATE	O	Buffered analog output representing the angular rate signal.
6	FILT	I	External capacitor connection to control bandwidth.
7	V_{DRIVE}	S	Power to SPI. The voltage supplied to this pin determines the voltage at which the serial interface operates.
8	AIN1	I	External Analog Input Channel 1. Single-ended analog input multiplexed into the on-chip track-and-hold according to the setting of the ADD0 and ADD1 address bits.
9	AIN2	I	External Analog Input Channel 2. Single-ended analog input multiplexed into the on-chip track-and-hold according to the setting of the ADD0 and ADD1 address bits.
10	COM	S	Common. Reference point for all circuitry in the ADIS16100.
11	V_{REF}	O	Precision 2.5 V Reference.
12	ST2	I	Self Test Input 2.
13	ST1	I	Self Test Input 1.
14	V_{CC}	S	Analog Power.
15	NC		No Connect.
16	$\overline{CS}$	I	Chip Select. Active low. This input frames the serial data transfer and initiates the conversion process.

[1] I = Input; O = Output; S = Power supply.

TYPICAL PERFORMANCE CHARACTERISTICS

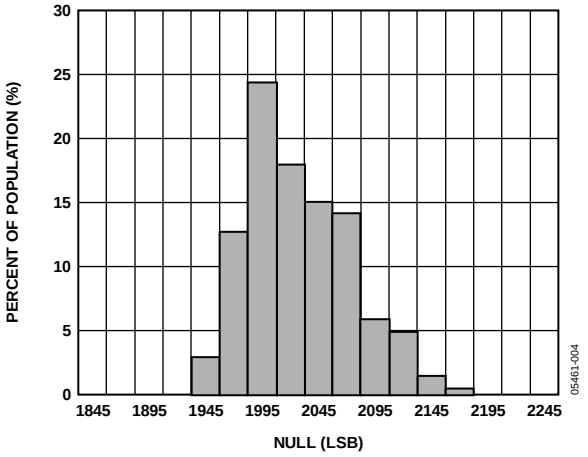


Figure 5. Initial Null Histogram

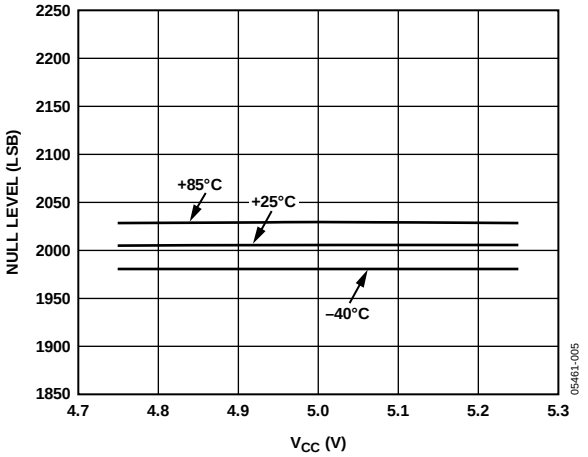


Figure 6. Null Level vs. Supply Voltage

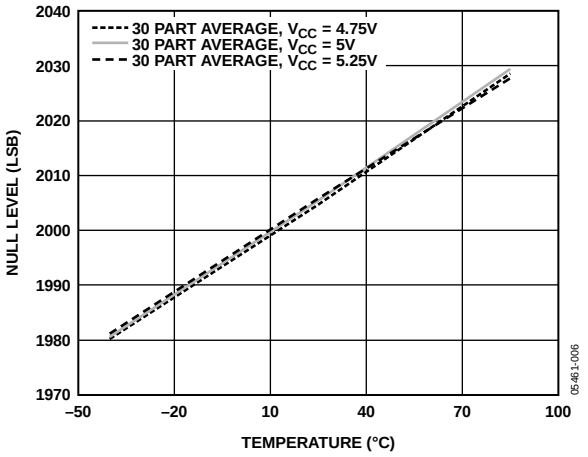


Figure 7. Null Level vs. Temperature

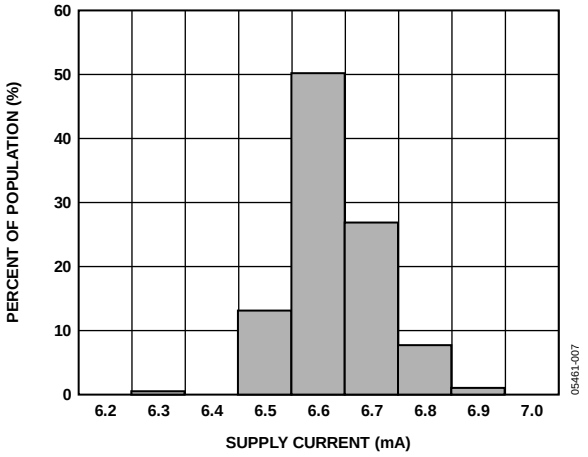


Figure 8. Supply Current Histogram

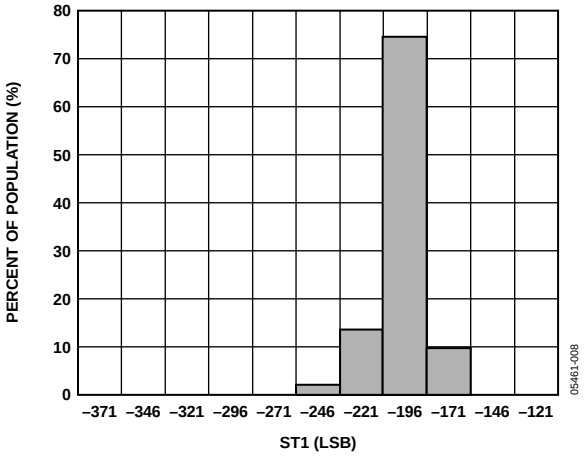


Figure 9. Self Test 1 Histogram

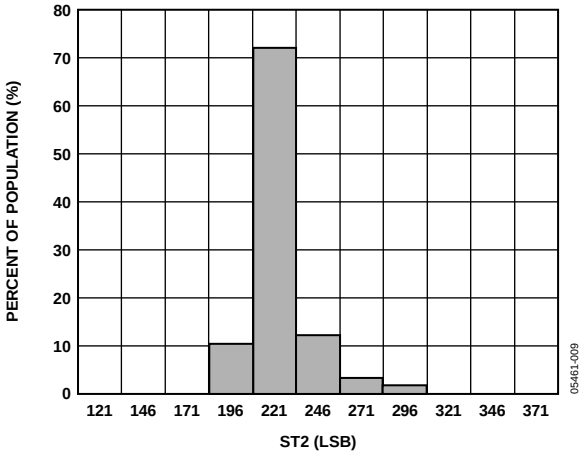


Figure 10. Self Test 2 Histogram

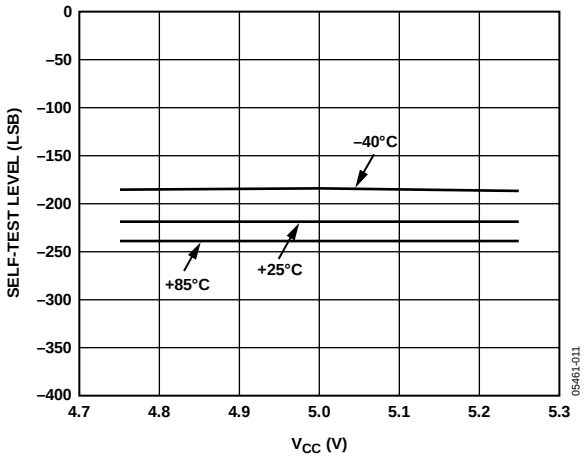


Figure 11. Self Test 1 vs. Supply Voltage

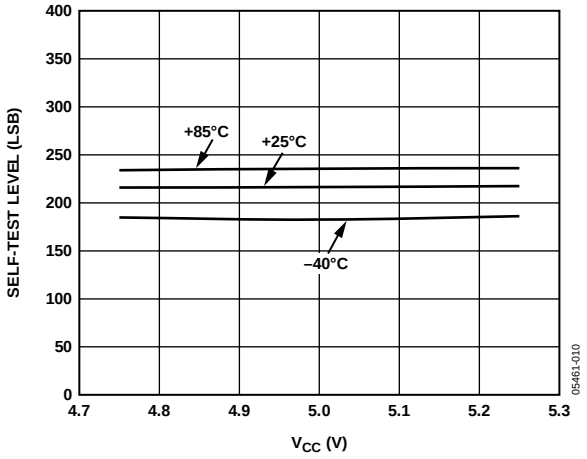


Figure 12. Self Test 2 vs. Supply Voltage

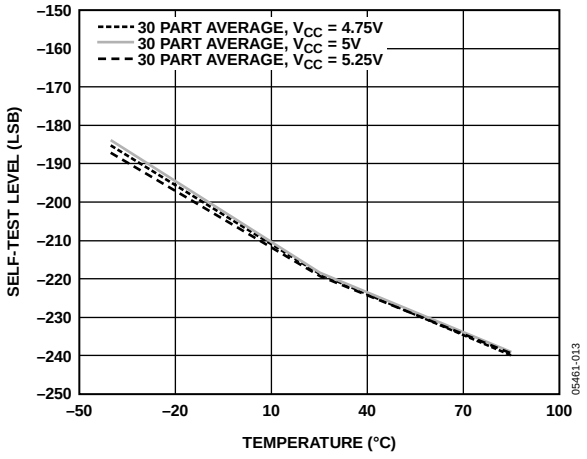


Figure 13. Self Test 1 vs. Temperature

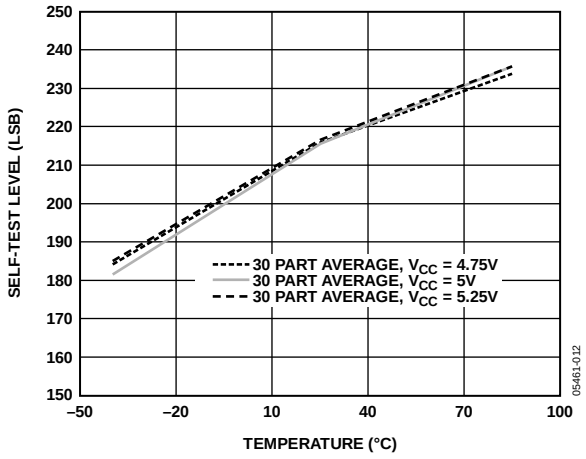


Figure 14. Self Test 2 vs. Temperature

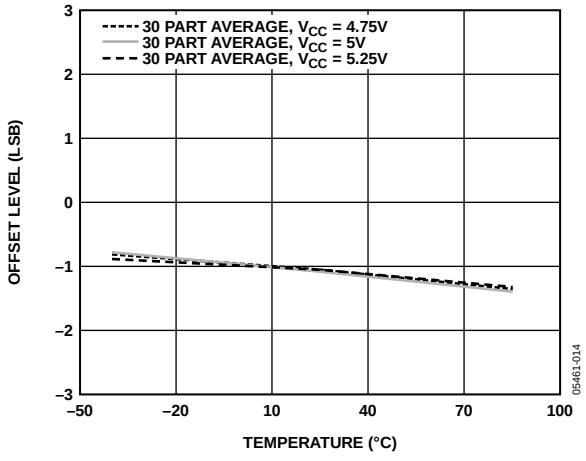


Figure 15. ADC Offset vs. Temperature and Supply Voltage

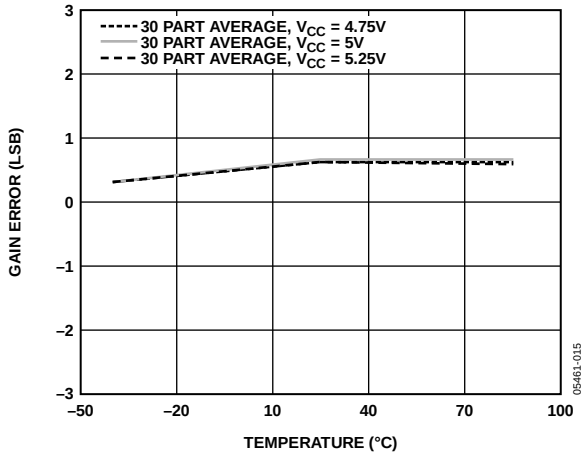


Figure 16. ADC Gain Error vs. Temperature (Excluding V_{REF})

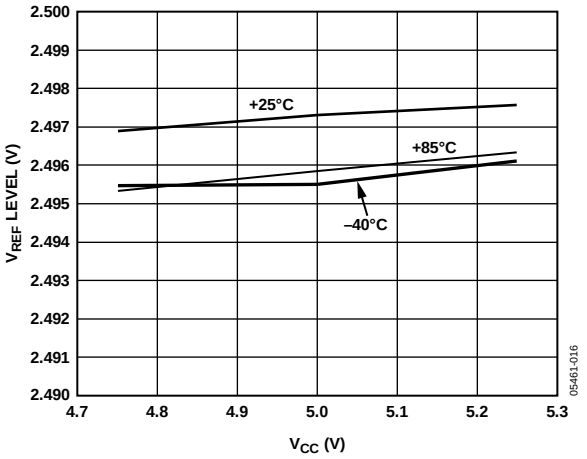


Figure 17. V_{REF} vs. Supply Voltage

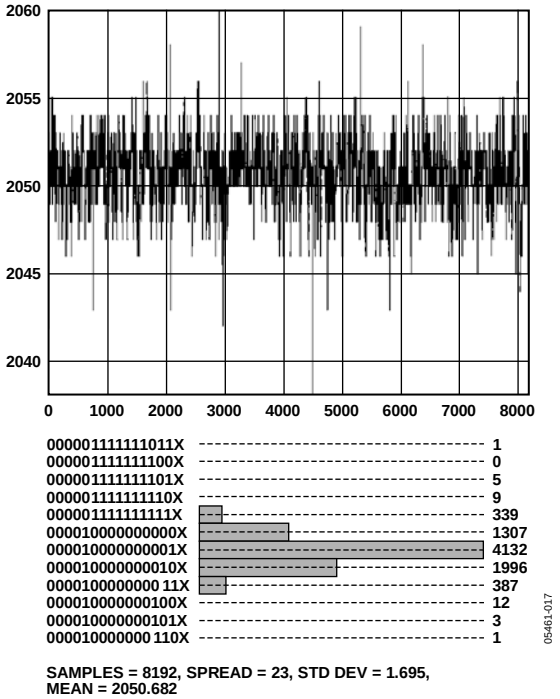


Figure 18. Noise Histogram

THEORY OF OPERATION

The ADIS16100 operates on the principle of a resonator gyro. Two polysilicon sensing structures each contain a dither frame, which is electrostatically driven to resonance. This produces the necessary velocity element to produce a Coriolis force during angular rate. At two of the outer extremes of each frame, orthogonal to the dither motion, are movable fingers that are placed between fixed pickoff fingers to form a capacitive pickoff structure that senses Coriolis motion. The resulting signal is fed to a series of gain and demodulation stages that produce the electrical rate signal output. The rate signal is then converted to a digital representation of the output on the SPI pins. The dual-sensor design rejects external *g*-forces and vibration. Fabricating the sensor with the signal conditioning electronics preserves signal integrity in noisy environments.

The electrostatic resonator requires 14 V to 16 V for operation. Since only 5 V is typically available in most applications, a charge pump is included on-chip.

After the demodulation stage, there is a single-pole, low-pass filter included on-chip that is used to limit high frequency artifacts before final amplification. A second single-pole, low-pass filter is set up via the bandwidth limit capacitor, C_{OUT}. This pole acts as the primary filter within the system (see the Increasing Measurement Range section).

SUPPLY AND COMMON CONSIDERATIONS

Power supply noise and transient behaviors can influence the accuracy and stability of any sensor-based measurement system. When considering the power supply for the ADIS16100, it is important to understand that the ADIS16100 provides 0.2 µF of decoupling capacitance on the V_{CC} pin. Depending on the level of noise present in the system power supply, the ADIS16100 may not require any additional decoupling capacitance for this supply. The analog supply, V_{CC}, and the digital drive supply, V_{DRIVE}, were segmented to allow multiple logic levels to be used in receiving the digital output data. V_{DRIVE} is intended for the down-stream logic power supply and supports standard 3.3 V and 5 V logic families. The V_{DRIVE} supply does not have internal decoupling capacitors.

INCREASING MEASUREMENT RANGE

The full-scale measurement range of the ADIS16100 is increased by placing an external resistor between the RATE pin and FILT pin, which would parallel an internal 180 kΩ, 1% resistor. For example, a 330 kΩ external resistor gives ~50% increase in the full-scale range. This is effective for up to a 4× increase in the full-scale range (minimum value of the parallel resistor allowed is 45 kΩ). The internal circuitry headroom requirements prevent further increase in the linear full-scale output range.

The trade-off associated with increasing the full-scale range are potential increase in output null drift (as much as 2°/s over temperature) and introducing initial null bias errors that must be calibrated.

SETTING BANDWIDTH

An external capacitor can be used in combination with an on-chip resistor to create a low-pass filter to limit the bandwidth of the ADIS16100's rate response.

The −3 dB frequency is defined as

$$f_{OUT} = 1/(2 \times \pi \times R_{OUT} \times (C_{OUT} + 0.022\ \mu F))$$

where R_{OUT} represents an internal impedance that was trimmed during manufacturing to 180 kΩ ± 1%.

Any external resistor applied between the RATE pin and the FILT pin results in

$$R_{OUT} = (180\ k\Omega \times R_{EXT})/(180\ k\Omega + R_{EXT})$$

With C_{OUT} = 0 µF, a default −3 dB frequency response of 40 Hz is obtained based upon an internal 0.022 µF capacitor implemented on-chip.

SELF-TEST FUNCTION

The ADIS16100 includes a self-test feature that actuates each of the sensing structures and associated electronics in the same manner as if subjected to angular rate. It provides a simple method for exercising the mechanical structure of the sensor, along with the entire signal processing circuit. It is activated by standard logic high levels applied to inputs ST1, ST2, or both. ST1 causes a change in the digital output equivalent to typically −221 LSB, and ST2 causes an opposite +221 LSB change. The self-test response follows the viscosity temperature dependence of the package atmosphere, approximately 0.25%/°C.

Activating both ST1 and ST2 simultaneously is not damaging. Since ST1 and ST2 are not necessarily closely matched, actuating both simultaneously can result in an apparent null bias shift.

CONTINUOUS SELF TEST

As an additional failure detection measure, power-on self test can be performed. However, some applications can warrant continuous self test while sensing rate.

CONTROL REGISTER

The control register on the ADIS16100 is a 12-bit, write-only register. Data is loaded from the DIN pin on the falling edge of SCLK. The data is transferred on the DIN line at the same time that the conversion result is read from the part. The data transferred on the DIN line dictates the configuration for the next conversion. This requires 16 serial clocks for every data transfer. Only the information provided on the first 12 falling clock edges (after $\overline{CS}$ falling edge) is loaded to the control register.

MSB denotes the first bit in the data stream. Table 8 shows the analog input channel selection options.

Table 6. Channel Selection

ADD1	ADD0	Analog Input Channel
0	0	Gyroscope
0	1	Temperature sensor
1	0	AIN1 input
1	1	AIN2 input

Table 7. The DIN Bit Stream

MSB (11)											LSB (0)
WRITE	LOW	DONTC	DONTC	ADD1	ADD0	HIGH	HIGH	DONTC	DONTC	LOW	CODING

Table 8. Analog Input Channel Selection Options

Bit	Mnemonic	Comment
11	WRITE	The value written to this bit of the control register determines whether the following 11 bits are loaded to the control register or not. If this bit is a 1, the following 11 bits are written to the control register. If it is a 0, the remaining 11 bits are not loaded to the control register and it remains unchanged.
10	LOW	This bit should be held low.
9, 8	DONTC	Don't care.
7, 6	ADD1, ADD0	These two address bits are loaded at the end of the present conversion sequence and select which analog input channel is to be converted in the next serial transfer. The selected input channel is decoded as shown in Table 6. The address bits corresponding to the conversion result are output on DOUT prior to the 12 bits of data. The next channel to be converted is selected by the mux on the 14th SCLK falling edge.
5, 4	HIGH	These pins should be held high.
3, 2	DONTC	Don't care.
1	LOW	This bit should be held low.
0	CODING	This bit selects the type of output coding used for the conversion result. If this bit is set to 0, the output coding for the part is twos complement. If this bit is set to 1, the output coding from the part is straight binary (for the next conversion).

SERIAL INTERFACE

Figure 2 shows the detailed timing diagram for the serial interface to the ADIS16100. The chip select signal, $\overline{CS}$, frames the entire data transfer, because it must be kept in a Logic 0 state to communicate with the ADIS16100. The serial clock, SCLK, provides the conversion clock and controls the transfer of information to and from the ADIS16100 during each conversion cycle. The data input, DIN, provides access to critical control parameters in the control register and the output signal, DOUT, provides access to the ADIS16100's output data.

The ADIS16100 offers an efficient data transfer function by supporting simultaneous READ and WRITE cycles. A data transfer cycle is started when the $\overline{CS}$ transitions to a Logic 0 state. If DIN is in Logic 1 state during the first falling edge of the SCLK, then the next 11 SCLK cycles fill the control register with the contents on the DIN pin. The appropriate bit definitions for DIN can be found in Table 7 and Table 8. If the DIN is in a Logic 0 state during the first falling edge of the SCLK, then contents of the control register remain unchanged. Since the control register is only 12-bits wide, the contents on the DIN pin during the last 4 SCLK cycles are ignored.

During this same cycle, the digital output data is clocked out on the DOUT pin, with the bit transitions occurring shortly after the SCLK falling edges. DOUT's bit sequence is characterized in Table 9 and Table 10. On the 16th falling edge of SCLK, the DOUT line goes back into a tri-state mode. If the rising edge of $\overline{CS}$ occurs before 16 SCLKs have elapsed, the DOUT line goes back into tri-state mode and the control register is not updated. Otherwise, DOUT returns to a tri-state mode on the 16th SCLK falling edge, as shown in Figure 2.

RATE SENSITIVE AXIS

This is a z-axis rate-sensing device that is also called a yaw rate sensing device. It produces a positive going output voltage for clockwise rotation about the axis normal to the package top, that is, clockwise when looking down at the package lid.

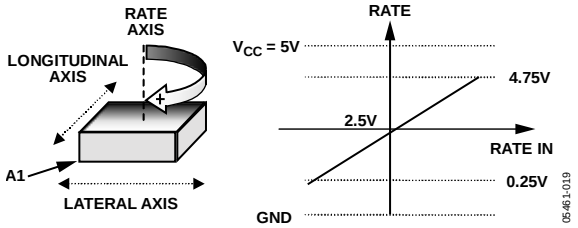


Figure 19. Rate Signal Increases with Clockwise Rotation

Table 9. DOUT Bit Stream

SCLK1															SCLK16
LOW	LOW	ADD0	ADD1	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0

Table 10. DOUT Bit Functions

SCLK	Mnemonic	Comment
1, 2	LOW	The outputs are low for SCLK1 and SCLK2.
3, 4	ADD0, ADD1	The address bits corresponding to the conversion result are output on DOUT prior to the 12 bits of data. See Table 6 for the coding of these address bits.
5	DB11	Data Bit 11 (MSB).
6 to 15	DB10 to DB1	Data Bit 10 to Data Bit 1.
16	DB0	Data Bit 0 (LSB).

SECOND-LEVEL ASSEMBLY

The recommended pad geometries for the ADIS16100 are displayed in Figure 20. The ADIS16100 can be attached to printed circuit boards using Sn63 or an equivalent solder. Figure 21 and Table 11 provide recommended solder reflow profiles for each solder type. Note: These profiles may not be the optimum profile for the user's application. In no case should the temperature exceed 260°C. It is recommended that the user develop a reflow profile based upon the specific application. In general, keep in mind that the lowest peak temperature and shortest dwell time above the melt temperature of the solder results in less shock and stress to the product. In addition, evaluating the cooling rate and peak temperature can result in a more reliable assembly.

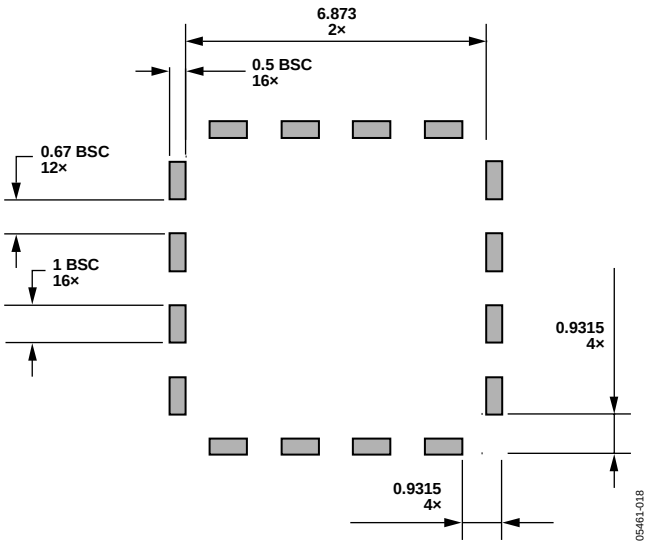


Figure 20. Second Level Assembly Pad Layout

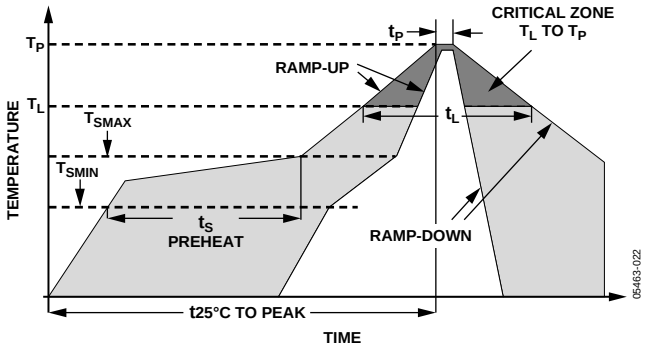


Figure 21. Recommended Solder Reflow Profiles

Table 11. Solder Profile Characteristics

Profile Feature	Sn63/Pb37
Average Ramp Rate (T_L to T_P)	3°C/sec max
Preheat	
Minimum Temperature (T_{SMIN})	100°C
Maximum Temperature (T_{SMAX})	150°C
Time (T_{SMIN} to T_{SMAX}) (t_S)	60 sec to 120 sec
T_{SMAX} to T_L	
Ramp-Up Rate	3°C/sec
Time Maintained Above Liquidous (T_L)	
Liquidous Temperature (T_L)	183°C
Time (t_L)	60 sec to 150 sec
Peak Temperature (T_P)	240°C + 0°C/–5°C
Time Within 5°C of Actual Peak Temperature (t_p)	10 sec to 30 sec
Ramp-Down Rate	6°C/sec max
Time 25°C to Peak Temperature	6 min max

OUTLINE DIMENSIONS

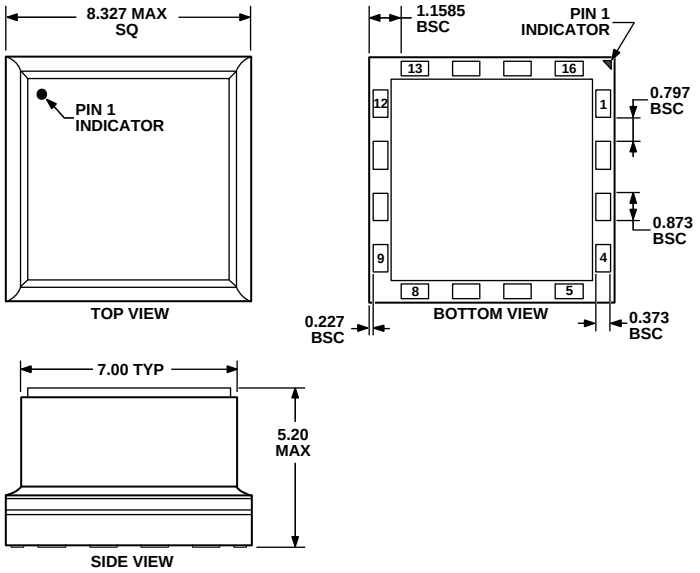


Figure 22. 16-Terminal Land Grid Array [LGA]
(CC-16-2)
Dimensions shown in millimeters

ORDERING GUIDE

Model	Temperature Range	Package Description	Package Option
ADIS16100ACC	−40°C to +85°C	16-Terminal Land Grid Array (LGA)	CC-16-2
ADIS16100/PCB		Evaluation Board	

NOTES

D05461-0-2/06(0)

±300°/sec Yaw Rate Gyro, SPI Interface Evaluation Board

Preliminary Technical Data

ADIS16100/PCB

GENERAL DESCRIPTION

The ADIS16100/PCB is an evaluation board that has been designed to provide simple access to the ADIS16100 using standard connectors. These connectors can be accessed using a variety of cable options, including standard 0.1" ribbon cables. While this evaluation board has been designed to fit into the ADIS-EVAL-PCB system, it can also be mounted directly to a system's platform as well. All of the components are on the top side and four mounting holes (sized for 2-56 or 2mm screws) have been provided to secure the board during evaluation.

CIRCUIT DESCRIPTION

The schematic, layout and parts list for the ADIS16100/PCB can be found in Figure 1, Figure 2, and Table 1.

The ADIS16100's digitized outputs can be accessed using the 4-wire serial port interface (SPI) signals on J1: SCLK, CS, DOUT and DIN. For specific information on using the ADIS16100's SPI interface, refer to the ADIS16100 datasheet. Auxiliary functions, such as the two 12-bit ADC inputs, can be accessed using J2. C1 and C4 provide additional filtering for the two different power supply inputs (Vdrive and Vcc). The ADIS16100's reference voltage is filtered by C2.

Table 1 – ADIS16100/PCB Parts List

Reference Designator	Part Description
U1	ADIS16100XXX
J1,J2	Connector, 12-pin, dual row, 2mm
C1, C4	Power supply filtering, not installed
C2	Vref filtering, not installed
C3	Bandwidth reduction, not installed

SPECIAL NOTES ON HANDLING

Note that the ADIS16100/PCB is not reverse polarity protected. Reversing the power supply or applying inappropriate voltages to any pin (outside the Absolute Maximum Ratings in the ADIS16100 data sheet) may damage the ADIS16100/PCB.

Table 2 – Power Supply Levels

Vcc	*+4.75V to +5.25V*
Vdrive	*+2.7V to +5.25V*

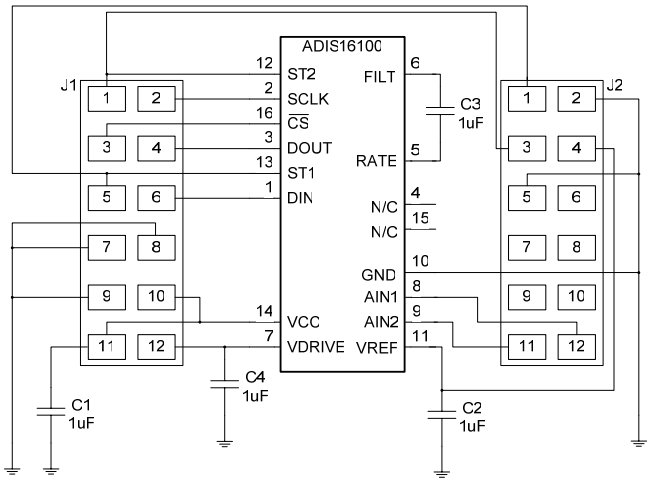


Figure 1 - ADIS16100/PCB Schematic

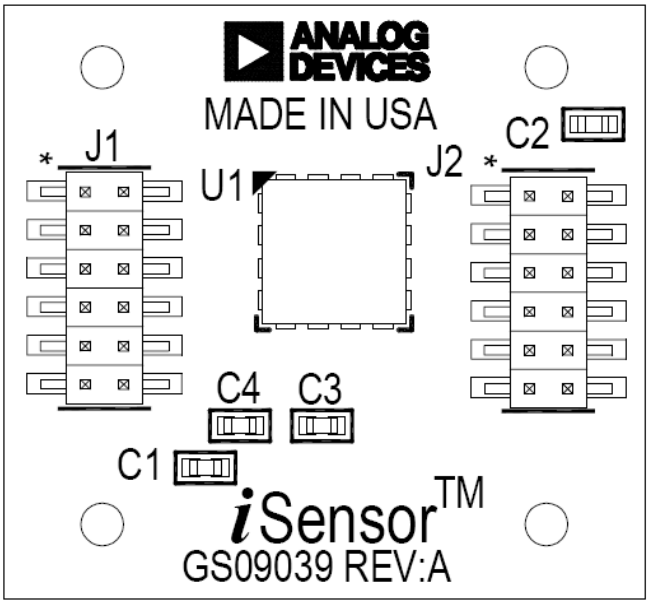


Figure 2 - ADIS16100/PCB Layout (Top View)

ORDERING GUIDE

Model	Package Description
ADIS16100/PCB	Evaluation Board

Rev. Pr.A **2/13/2006**

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.326.8703

Precision ±1.7 *g* Single/Dual Axis Accelerometer

ADXL103/ADXL203

FEATURES

High performance, single/dual axis accelerometer on a single IC chip
5 mm × 5 mm × 2 mm LCC package
1 m*g* resolution at 60 Hz
Low power: 700 µA at V_S = 5 V (typical)
High zero *g* bias stability
High sensitivity accuracy
–40°C to +125°C temperature range
X and Y axes aligned to within 0.1° (typical)
BW adjustment with a single capacitor
Single-supply operation
3500 *g* shock survival

APPLICATIONS

Vehicle Dynamic Control (VDC)/Electronic Stability Program (ESP) systems
Electronic chassis control
Electronic braking
Platform stabilization/leveling
Navigation
Alarms and motion detectors.
High accuracy, 2-axis tilt sensing

GENERAL DESCRIPTION

The ADXL103/ADXL203 are high precision, low power, complete single and dual axis accelerometers with signal conditioned voltage outputs, all on a single monolithic IC. The ADXL103/ADXL203 measures acceleration with a full-scale range of ±1.7 *g* . The ADXL103/ADXL203 can measure both dynamic acceleration (e.g., vibration) and static acceleration (e.g., gravity).

The typical noise floor is 110 µg/√Hz, allowing signals below 1 m*g* (0.06° of inclination) to be resolved in tilt sensing applications using narrow bandwidths (<60 Hz).

The user selects the bandwidth of the accelerometer using capacitors C_X and C_Y at the X_{OUT} and Y_{OUT} pins. Bandwidths of 0.5 Hz to 2.5 kHz may be selected to suit the application.

The ADXL103 and ADXL203 are available in 5 mm × 5 mm × 2 mm, 8-pad hermetic LCC packages.

FUNCTIONAL BLOCK DIAGRAM

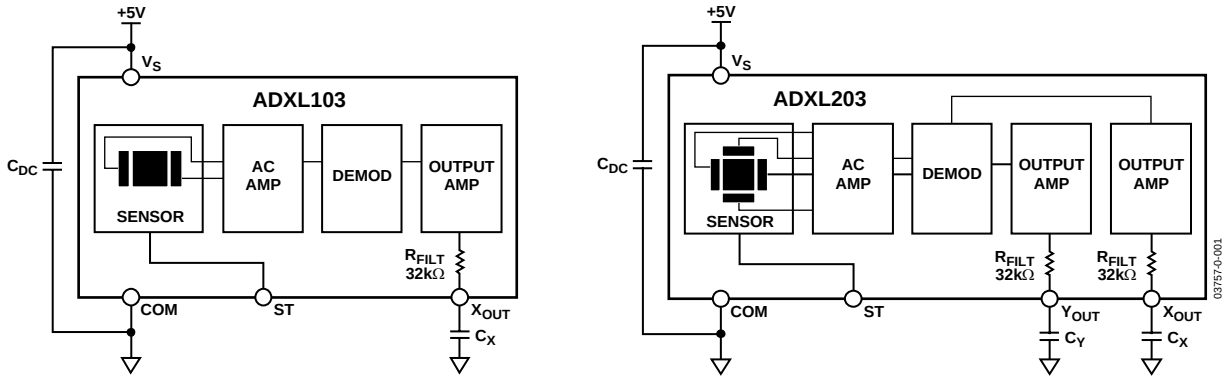


Figure 1. ADXL103/ADXL203 Functional Block Diagram

Rev. 0

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.326.8703

TABLE OF CONTENTS

REVISION HISTORY

Revision 0: Initial Version

SPECIFICATIONS

Table 1. T_A = −40°C to +125°C, V_S = 5 V, C_X = C_Y = 0.1 µF, Acceleration = 0 *g*, unless otherwise noted.

Parameter	Conditions	Min	Typ	Max	Unit
SENSOR INPUT	Each Axis				
Measurement Range[1]		±1.7			*g*
Nonlinearity	% of Full Scale		±0.5	±2.5	%
Package Alignment Error			±1		Degrees
Alignment Error (ADXL203)	X Sensor to Y Sensor		±0.1		Degrees
Cross Axis Sensitivity			±2	±5	%
SENSITIVITY (Ratiometric)[2]	Each Axis				
Sensitivity at X_{OUT}, Y_{OUT}	V_S = 5 V	940	1000	1060	mV/*g*
Sensitivity Change due to Temperature[3]	V_S = 5 V		±0.3		%
ZERO *g* BIAS LEVEL (Ratiometric)	Each Axis				
0 *g* Voltage at X_{OUT}, Y_{OUT}	V_S = 5 V	2.4	2.5	2.6	V
Initial 0 *g* Output Deviation from Ideal	V_S = 5 V, 25°C		±25		m*g*
0 *g* Offset vs. Temperature			±0.1		m*g*/°C
NOISE PERFORMANCE					
Output Noise	< 4 kHz, V_S = 5 V, 25°C		1	6	mV rms
Noise Density	@25°C		110		µ*g*/√Hz rms
FREQUENCY RESPONSE[4]					
C_X, C_Y Range[5]		0.002		10	µF
R_{FILT} Tolerance		24	32	40	kΩ
Sensor Resonant Frequency			5.5		kHz
SELF TEST[6]					
Logic Input Low				1	V
Logic Input High		4			V
ST Input Resistance to Ground		30	50		kΩ
Output Change at X_{OUT}, Y_{OUT}	Self Test 0 to 1	400	750	1100	mV
OUTPUT AMPLIFIER					
Output Swing Low	No Load		0.3		V
Output Swing High	No Load		4.5		V
POWER SUPPLY					
Operating Voltage Range		3		6	V
Quiescent Supply Current			0.7	1.1	mA
Turn-On Time[7]			20		ms

[1] Guaranteed by measurement of initial offset and sensitivity.
[2] Sensitivity is essentially ratiometric to V_S. For V_S = 4.75 V to 5.25 V, sensitivity is 186 mV/V/*g* to 215 mV/V/*g*.
[3] Defined as the output change from ambient-to-maximum temperature or ambient-to-minimum temperature.
[4] Actual frequency response controlled by user-supplied external capacitor (C_X, C_Y).
[5] Bandwidth = 1/(2 × π × 32 kΩ × C). For C_X, C_Y = 0.002 µF, Bandwidth = 2500 Hz. For C_X, C_Y = 10 µF, Bandwidth = 0.5 Hz. Minimum/maximum values are not tested.
[6] Self-test response changes cubically with V_S.
[7] Larger values of C_X, C_Y will increase turn-on time. Turn-on time is approximately 160 × C_X or C_Y + 4 ms, where C_X, C_Y are in µF.

All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.

ABSOLUTE MAXIMUM RATINGS

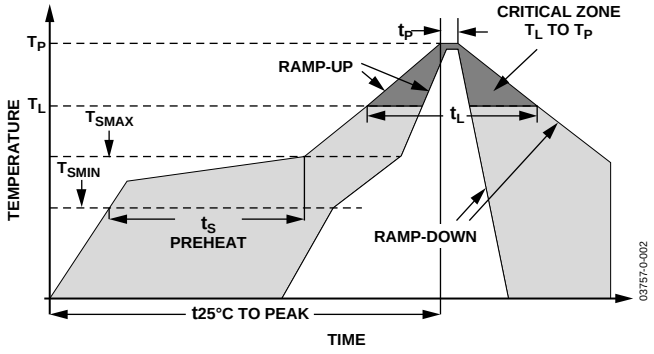
Table 2. ADXL103/ADXL203 Stress Ratings

Parameter	Rating
Acceleration (Any Axis, Unpowered)	3,500 *g*
Acceleration (Any Axis, Powered)	3,500 *g*
Drop Test (Concrete Surface)	1.2 m
V_S	–0.3 V to +7.0 V
All Other Pins	(COM – 0.3 V) to (V_S + 0.3 V)
Output Short-Circuit Duration (Any Pin to Common)	Indefinite
Operating Temperature Range	–55°C to +125°C
Storage Temperature	–65°C to +150°C

Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Table 3. Package Characteristics

Package Type	θ_{JA}	θ_{JC}	Device Weight
8-Lead CLCC	120°C/W	20°C/W	<1.0 gram



Profile Feature	Condition	
	Sn63/Pb37	Pb Free
Average Ramp Rate (T_L to T_P)	3°C/second Max	
Preheat		
• Minimum Temperature (T_{SMIN})	100°C	150°C
• Minimum Temperature (T_{SMAX})	150°C	200°C
• Time (T_{SMIN} to T_{SMAX}) (t_S)	60–120 seconds	60–150 seconds
T_{SMAX} to T_L		
• Ramp-Up Rate	3°C/second	
Time Maintained above Liquidous (T_L)		
• Liquidous Temperature (T_L)	183°C	217°C
• Time (t_L)	60–150 seconds	60–150 seconds
Peak Temperature (T_P)	240°C +0°C/–5°C	260°C +0°C/–5°C
Time within 5°C of Actual Peak Temperature (t_P)	10–30 seconds	20–40 seconds
Ramp-Down Rate	6°C/second Max	
Time 25°C to Peak Temperature	6 minutes Max	8 minutes Max

Figure 2. Recommended Soldering Profile

TYPICAL PERFORMANCE CHARACTERISTICS

(V_S = 5 V for all graphs, unless otherwise noted.)

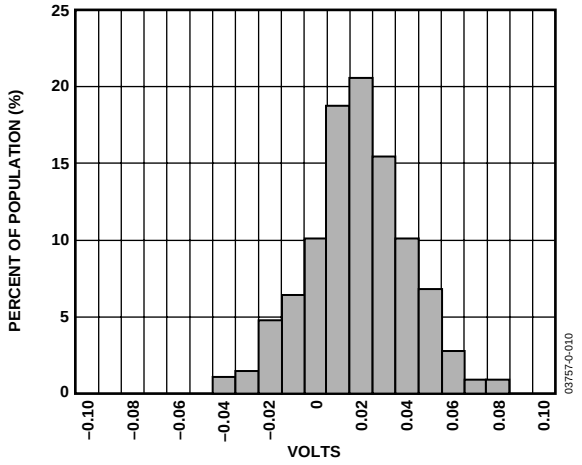


Figure 3. X Axis Zero g Bias Deviation from Ideal at 25°C

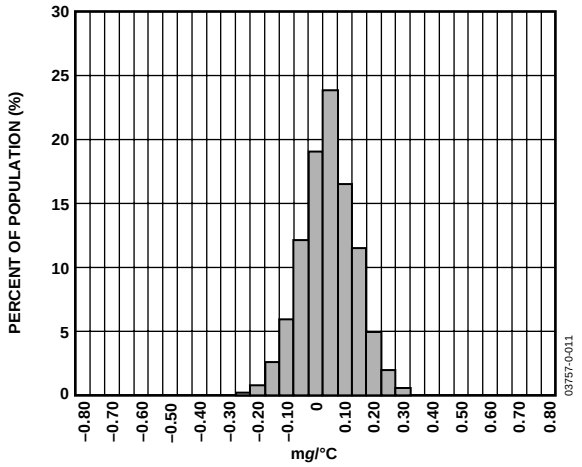


Figure 4. X Axis Zero g Bias Tempco

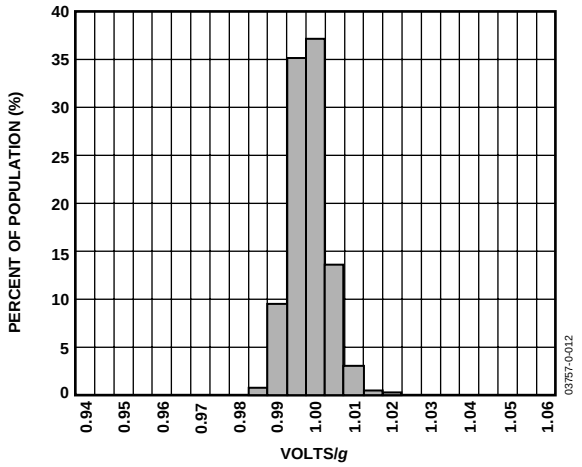


Figure 5. X Axis Sensitivity at 25°C

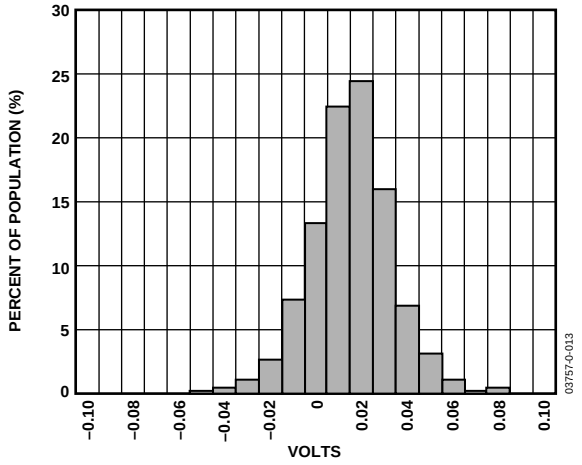


Figure 6. Y Axis Zero g Bias Deviation from Ideal at 25°C

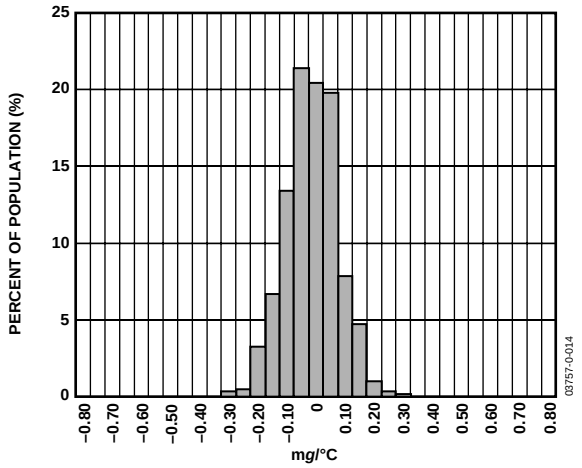


Figure 7. Y Axis Zero g Bias Tempco

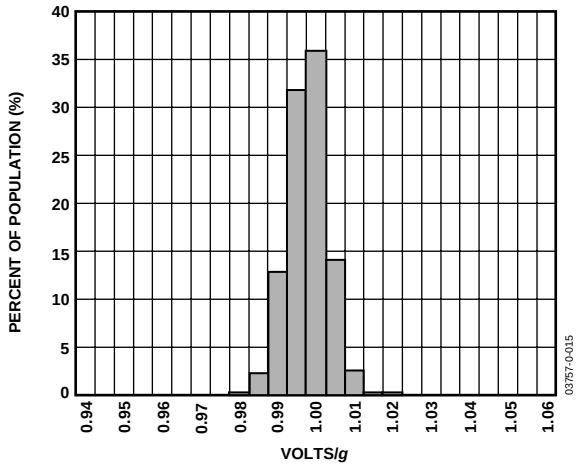


Figure 8. Y Axis Sensitivity at 25°C

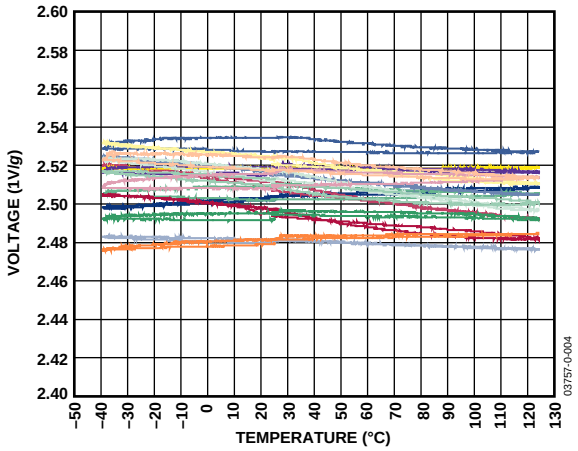


Figure 9. Zero g Bias vs. Temperature – Parts Soldered to PCB

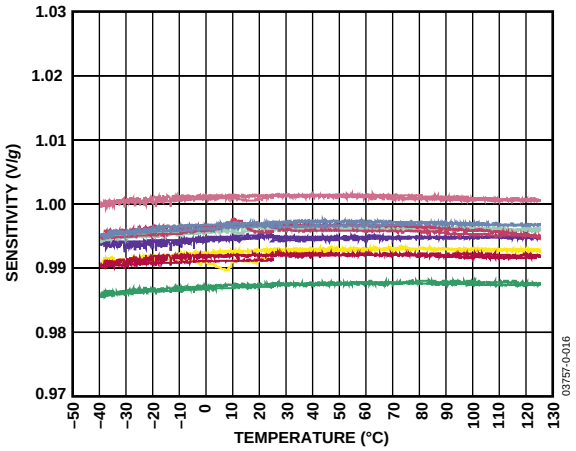


Figure 12. Sensitivity vs. Temperature – Parts Soldered to PCB

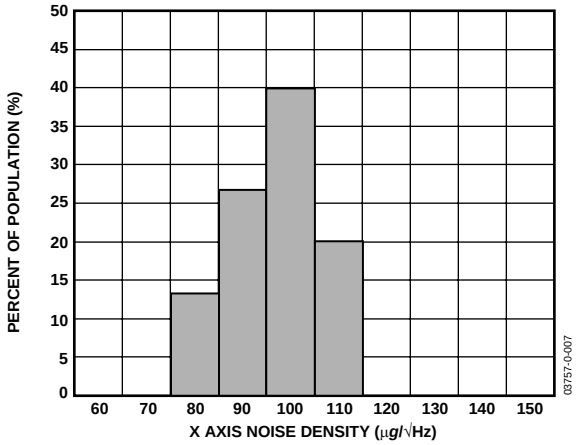


Figure 10. X Axis Noise Density at 25°C

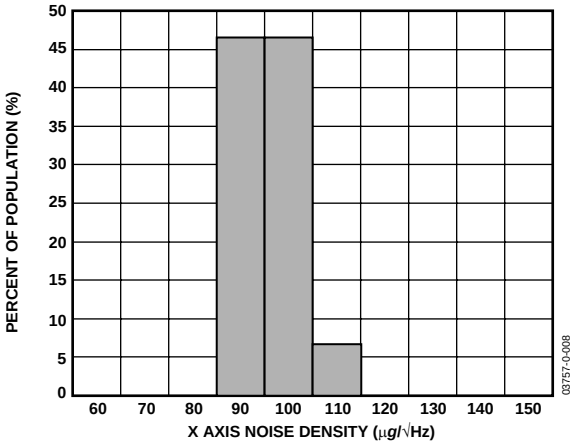


Figure 13. Y Axis Noise Density at 25°C

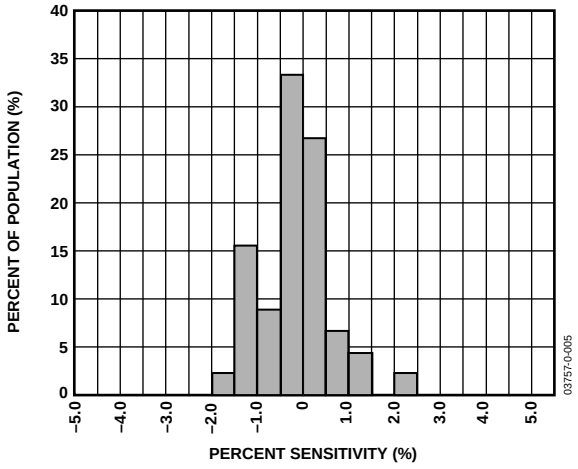


Figure 11. Z vs. X Cross-Axis Sensitivity

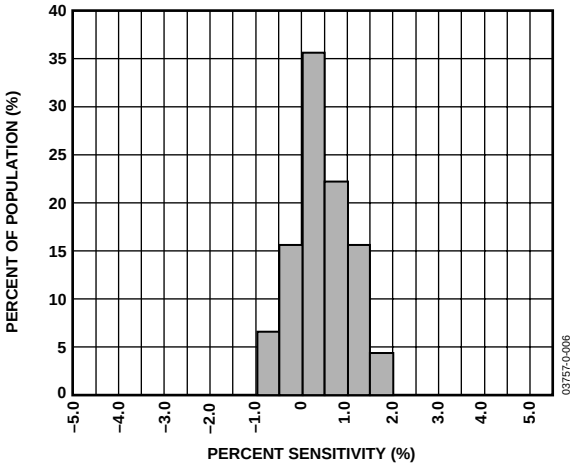


Figure 14. Z vs. Y Cross-Axis Sensitivity

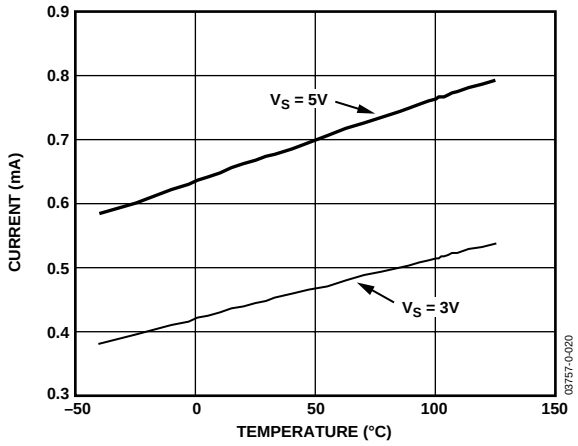


Figure 15. Supply Current vs. Temperature

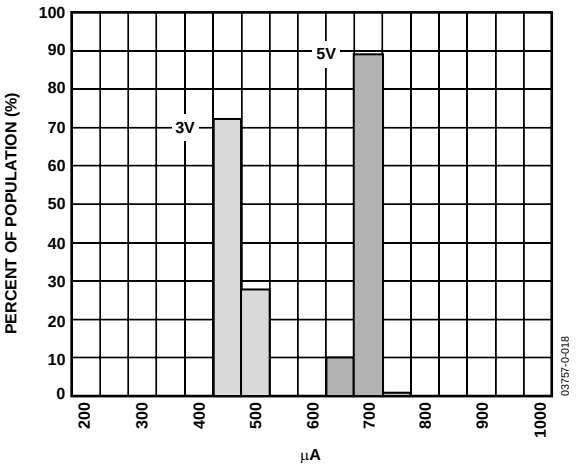


Figure 18. Supply Current at 25°C

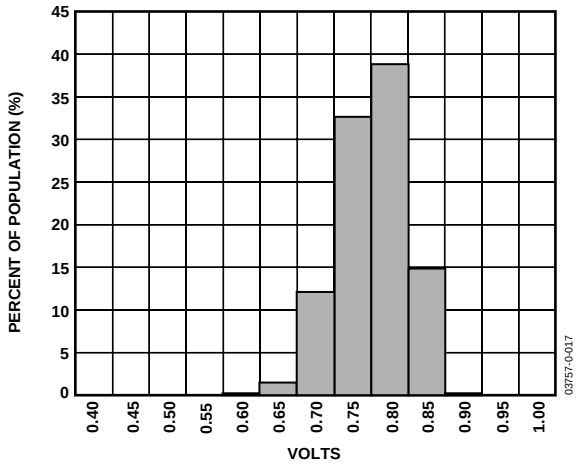


Figure 16. X Axis Self Test Response at 25°C

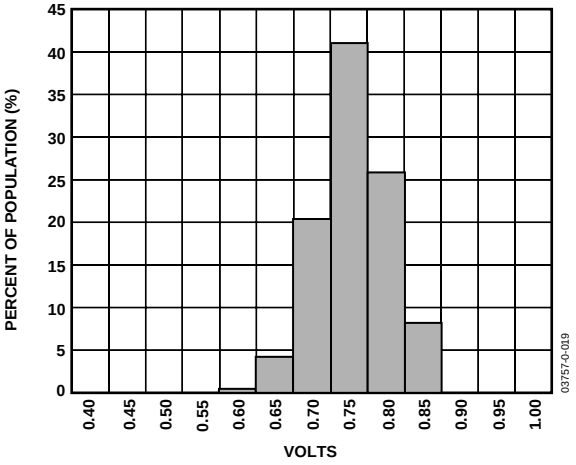


Figure 19. Y Axis Self Test Response at 25°C

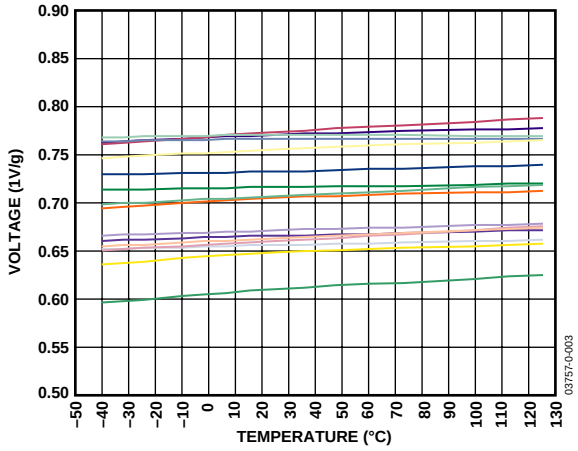


Figure 17. Self Test Response vs. Temperature

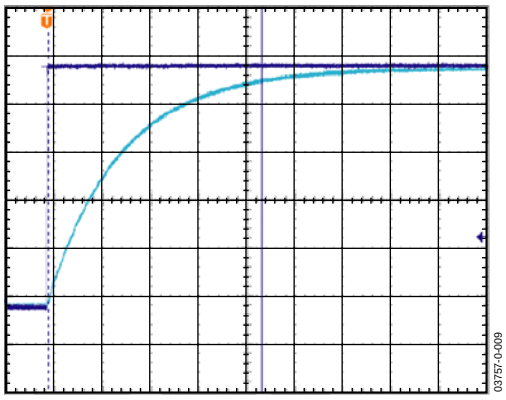


Figure 20. Turn-On Time – C_X, C_Y = 0.1 µF, Time Scale = 2 ms/div

THEORY OF OPERATION

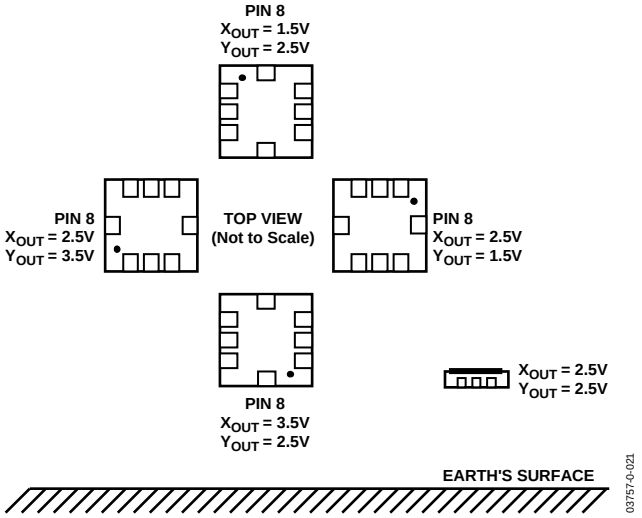


Figure 21. Output Response vs. Orientation

The ADXL103/ADXL203 are complete acceleration measurement systems on a single monolithic IC. The ADXL103 is a single axis accelerometer, while the ADXL203 is a dual axis accelerometer. Both parts contain a polysilicon surface-micromachined sensor and signal conditioning circuitry to implement an open-loop acceleration measurement architecture. The output signals are analog voltages proportional to acceleration. The ADXL103/ADXL203 are capable of measuring both positive and negative accelerations to at least ±1.7 *g*. The accelerometer can measure static acceleration forces such as gravity, allowing it to be used as a tilt sensor.

The sensor is a surface-micromachined polysilicon structure built on top of the silicon wafer. Polysilicon springs suspend the structure over the surface of the wafer and provide a resistance against acceleration forces. Deflection of the structure is measured using a differential capacitor that consists of independent fixed plates and plates attached to the moving mass. The fixed plates are driven by 180° out-of-phase square waves. Acceleration will deflect the beam and unbalance the differential capacitor, resulting in an output square wave whose amplitude is proportional to acceleration. Phase sensitive demodulation techniques are then used to rectify the signal and determine the direction of the acceleration.

The output of the demodulator is amplified and brought off-chip through a 32 kΩ resistor. At this point, the user can set the signal bandwidth of the device by adding a capacitor. This filtering improves measurement resolution and helps prevent aliasing.

PERFORMANCE

Rather than using additional temperature compensation circuitry, innovative design techniques have been used to ensure high performance is built in. As a result, there is essentially no quantization error or non-monotonic behavior, and temperature hysteresis is very low (typically less than 10 m*g* over the −40°C to +125°C temperature range).

Figure 9 shows the zero *g* output performance of eight parts (X and Y axis) over a −40°C to +125°C temperature range.

Figure 12 demonstrates the typical sensitivity shift over temperature for V_S = 5 V. Sensitivity stability is optimized for V_S = 5 V, but is still very good over the specified range; it is typically better than ±1% over temperature at V_S = 3 V.

APPLICATIONS

POWER SUPPLY DECOUPLING

For most applications, a single 0.1 µF capacitor, C_{DC}, will adequately decouple the accelerometer from noise on the power supply. However in some cases, particularly where noise is present at the 140 kHz internal clock frequency (or any harmonic thereof), noise on the supply may cause interference on the ADXL103/ADXL203 output. If additional decoupling is needed, a 100 Ω (or smaller) resistor or ferrite beads may be inserted in the supply line of the ADXL103/ADXL203. Additionally, a larger bulk bypass capacitor (in the 1 µF to 22 µF range) may be added in parallel to C_{DC}.

SETTING THE BANDWIDTH USING C_X AND C_Y

The ADXL103/ADXL203 has provisions for bandlimiting the X_{OUT} and Y_{OUT} pins. Capacitors must be added at these pins to implement low-pass filtering for antialiasing and noise reduction. The equation for the 3 dB bandwidth is

$$F_{-3\,dB} = 1/(2\pi(32\ \text{k}\Omega) \times C_{(X,\,Y)})$$

or more simply,

$$F_{-3\,dB} = 5\ \mu\text{F}/C_{(X,\,Y)}$$

The tolerance of the internal resistor (R_{FILT}) can vary typically as much as ±25% of its nominal value (32 kΩ); thus, the bandwidth will vary accordingly. A minimum capacitance of 2000 pF for C_X and C_Y is required in all cases.

Table 4. Filter Capacitor Selection, C_X and C_Y

Bandwidth (Hz)	Capacitor (µF)
1	4.7
10	0.47
50	0.10
100	0.05
200	0.027
500	0.01

SELF TEST

The ST pin controls the self-test feature. When this pin is set to V_S, an electrostatic force is exerted on the beam of the accelerometer. The resulting movement of the beam allows the user to test if the accelerometer is functional. The typical change in output will be 750 m*g* (corresponding to 750 mV). This pin may be left open-circuit or connected to common in normal use.

The ST pin should never be exposed to voltage greater than V_S + 0.3 V. If the system design is such that this condition cannot be guaranteed (i.e., multiple supply voltages present), a low V_F clamping diode between ST and V_S is recommended.

DESIGN TRADE-OFFS FOR SELECTING FILTER CHARACTERISTICS: THE NOISE/BW TRADE-OFF

The accelerometer bandwidth selected will ultimately determine the measurement resolution (smallest detectable acceleration). Filtering can be used to lower the noise floor, which improves the resolution of the accelerometer. Resolution is dependent on the analog filter bandwidth at X_{OUT} and Y_{OUT}.

The output of the ADXL103/ADXL203 has a typical bandwidth of 2.5 kHz. The user must filter the signal at this point to limit aliasing errors. The analog bandwidth must be no more than half the A/D sampling frequency to minimize aliasing. The analog bandwidth may be further decreased to reduce noise and improve resolution.

The ADXL103/ADXL203 noise has the characteristics of white Gaussian noise, which contributes equally at all frequencies and is described in terms of µg/√Hz (i.e., the noise is proportional to the square root of the accelerometer's bandwidth). The user should limit bandwidth to the lowest frequency needed by the application in order to maximize the resolution and dynamic range of the accelerometer.

With the single pole roll-off characteristic, the typical noise of the ADXL103/ADXL203 is determined by

$$rmsNoise = (110\mu\text{g}/\sqrt{Hz}) \times (\sqrt{BW \times 1.6})$$

At 100 Hz, the noise is

$$rmsNoise = (110\mu\text{g}/\sqrt{Hz}) \times (\sqrt{100 \times 1.6}) = 1.4m\text{g}$$

Often, the peak value of the noise is desired. Peak-to-peak noise can only be estimated by statistical methods. Table 5 is useful for estimating the probabilities of exceeding various peak values, given the rms value.

Table 5. Estimation of Peak-to-Peak Noise

Peak-to-Peak Value	% of Time That Noise Will Exceed Nominal Peak-to-Peak Value
2 × RMS	32
4 × RMS	4.6
6 × RMS	0.27
8 × RMS	0.006

Peak-to-peak noise values give the best estimate of the uncertainty in a single measurement. Table 6 gives the typical noise output of the ADXL103/ADXL203 for various C_X and C_Y values.

Table 6. Filter Capacitor Selection (C_X, C_Y)

Bandwidth(Hz)	C_X, C_Y (µF)	RMS Noise (m*g*)	Peak-to-Peak Noise Estimate (m*g*)
10	0.47	0.4	2.6
50	0.1	1.0	6
100	0.047	1.4	8.4
500	0.01	3.1	18.7

USING THE ADXL103/ADXL203 WITH OPERATING VOLTAGES OTHER THAN 5 V

The ADXL103/ADXL203 is tested and specified at V_S = 5 V; however, it can be powered with V_S as low as 3 V or as high as 6 V. Some performance parameters will change as the supply voltage is varied.

The ADXL103/ADXL203 output is ratiometric, so the output sensitivity (or scale factor) will vary proportionally to supply voltage. At V_S = 3 V the output sensitivity is typically 560 mV/*g*.

The zero *g* bias output is also ratiometric, so the zero *g* output is nominally equal to V_S/2 at all supply voltages.

The output noise is not ratiometric but is absolute in volts; therefore, the noise density decreases as the supply voltage increases. This is because the scale factor (mV/*g*) increases while the noise voltage remains constant. At V_S = 3 V, the noise density is typically 190 µ*g*/√Hz.

Self-test response in g is roughly proportional to the square of the supply voltage. However, when ratiometricity of sensitivity is factored in with supply voltage, self-test response in volts is roughly proportional to the cube of the supply voltage. So at V_S = 3 V, the self-test response will be approximately equivalent to 150 mV, or equivalent to 270 m*g* (typical).

The supply current decreases as the supply voltage decreases. Typical current consumption at V_{DD} = 3 V is 450 µA.

USING THE ADXL203 AS A DUAL-AXIS TILT SENSOR

One of the most popular applications of the ADXL203 is tilt measurement. An accelerometer uses the force of gravity as an input vector to determine the orientation of an object in space.

An accelerometer is most sensitive to tilt when its sensitive axis is perpendicular to the force of gravity, i.e., parallel to the earth's surface. At this orientation, its sensitivity to changes in tilt is highest. When the accelerometer is oriented on axis to gravity, i.e., near its +1 *g* or −1 *g* reading, the change in output acceleration per degree of tilt is negligible. When the accelerometer is perpendicular to gravity, its output will change nearly 17.5 m*g* per degree of tilt. At 45°, its output changes at only 12.2 m*g* per degree and resolution declines.

Dual-Axis Tilt Sensor: Converting Acceleration to Tilt

When the accelerometer is oriented so both its X axis and Y axis are parallel to the earth's surface, it can be used as a 2-axis tilt sensor with a roll axis and a pitch axis. Once the output signal from the accelerometer has been converted to an acceleration that varies between −1 *g* and +1 *g*, the output tilt in degrees is calculated as follows:

$$PITCH = ASIN(A_X/1\ g)$$

$$ROLL = ASIN(A_Y/1\ g)$$

Be sure to account for overranges. It is possible for the accelerometers to output a signal greater than ±1 *g* due to vibration, shock, or other accelerations.

PIN CONFIGURATIONS AND FUNCTIONAL DESCRIPTIONS

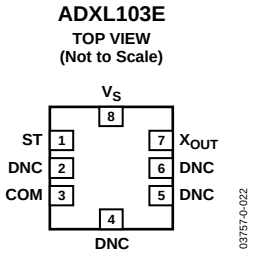


Figure 22. ADXL103 8-Lead CLCC

Table 7. ADXL103 8-Lead CLCC Pin Function Descriptions

Pin No.	Mnemonic	Description
1	ST	Self Test
2	DNC	Do Not Connect
3	COM	Common
4	DNC	Do Not Connect
5	DNC	Do Not Connect
6	DNC	Do Not Connect
7	X_{OUT}	X Channel Output
8	V_S	3 V to 6 V

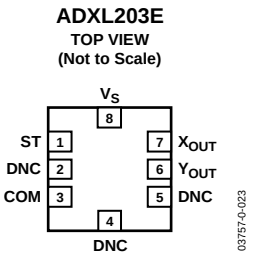


Figure 23. ADXL203 8-Lead CLCC

Table 8. ADXL203 8-Lead CLCC Pin Function Descriptions

Pin No.	Mnemonic	Description
1	ST	Self Test
2	DNC	Do Not Connect
3	COM	Common
4	DNC	Do Not Connect
5	DNC	Do Not Connect
6	Y_{OUT}	Y Channel Output
7	X_{OUT}	X Channel Output
8	V_S	3 V to 6 V

OUTLINE DIMENSIONS

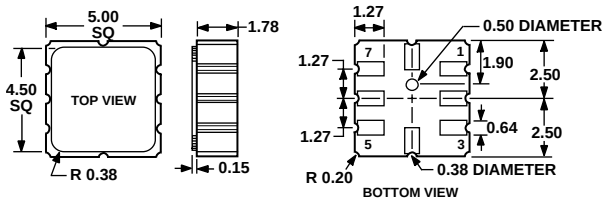


Figure 24. 8-Terminal Ceramic Leadless Chip Carrier [LCC]
(E-8)
Dimensions shown in millimeters

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Electrostatic charges as high as 4000 V readily accumulate on the human body and test equipment and can discharge without detection. Although this product features proprietary ESD protection circuitry, permanent damage may occur on devices subjected to high energy electrostatic discharges. Therefore, proper ESD precautions are recommended to avoid performance degradation or loss of functionality.

ORDERING GUIDE

ADXL103/ADXL203 Products	Number of Axes	Specified Voltage (V)	Temperature Range	Package Description	Package Option
ADXL103CE[1]	1	5	–40°C to +125°C	8-Lead Ceramic Leadless Chip Carrier	E-8
ADXL103CE–REEL[1]	1	5	–40°C to +125°C	8-Lead Ceramic Leadless Chip Carrier	E-8
ADXL203CE[1]	2	5	–40°C to +125°C	8-Lead Ceramic Leadless Chip Carrier	E-8
ADXL203CE–REEL[1]	2	5	–40°C to +125°C	8-Lead Ceramic Leadless Chip Carrier	E-8
ADXL203EB Evaluation Board				Evaluation Board	

[1] Lead finish—Gold over Nickel over Tungsten.

D03757–0–4/04(0)

ANALOG DEVICES

www.analog.com

Dual Axis Accelerometer Evaluation Board

ADXL203EB

GENERAL DESCRIPTION

The ADXL203EB is a simple evaluation board that allows quick evaluation of the performance of the ADXL203 dual axis ±1.7 *g* accelerometer. The ADXL203EB has a 5-pin 0.1 inch spaced header for access to all power and signal lines that the user can attach to a prototyping board (breadboard) or wire using a standard plug. Four holes are provided for mechanical attachment of the ADXL203EB to the application.

The ADXL203EB is 20 mm × 20 mm, with mounting holes set 15 mm × 15 mm at the corners of the PCB.

CIRCUIT DESCRIPTION

The schematic and parts list of the ADXL203EB are shown in Figure 1. Analog bandwidth can be set by changing capacitors C2 and C3. See the ADXL203 data sheet for a complete description of the operation of the accelerometer.

The part layout of the ADXL203EB is shown in Figure 2. The ADXL203EB has two factory-installed 100 nF capacitors (C2 and C3) at X_{OUT} and Y_{OUT} to reduce the bandwidth to 50 Hz. Many applications require a different bandwidth, in which case the user can change C2 and C3, as appropriate.

SPECIAL NOTES ON HANDLING

The ADXL203EB is not reverse polarity protected. Reversing the +V supply and ground pins can cause damage to the ADXL203.

Dropping the ADXL203EB on a hard surface can generate several thousand *g* of acceleration and might exceed the data sheet absolute maximum limits. See the ADXL203 data sheet for more information.

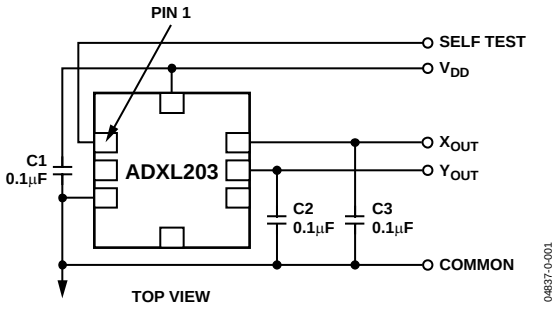


Figure 1. ADXL203EB Schematic

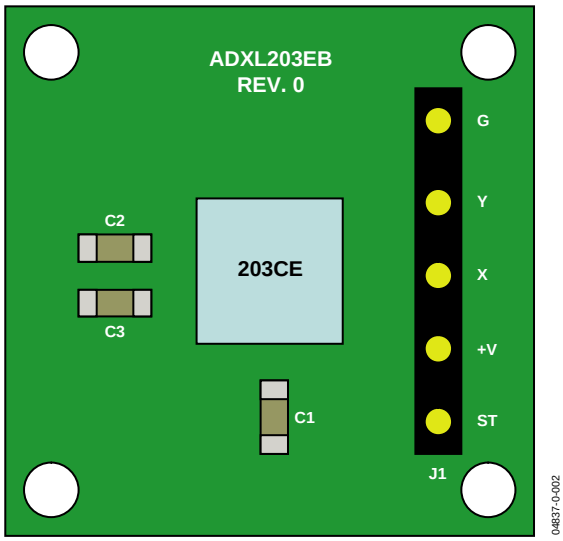


Figure 2. ADXL203EB Physical Layout

ORDERING GUIDE

Model	Package Description
ADXL203EB	Evaluation Board

Rev. 0

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.326.8703

D04837–0–4/04(0)

±300°/s Single Chip Yaw Rate Gyro with Signal Conditioning

ADXRS300

FEATURES

Complete rate gyroscope on a single chip
Z-axis (yaw rate) response
High vibration rejection over wide frequency
2000 *g* powered shock survivability
Self-test on digital command
Temperature sensor output
Precision voltage reference output
Absolute rate output for precision applications
5 V single-supply operation
Ultrasmall and light (< 0.15 cc, < 0.5 gram)

APPLICATIONS

Vehicle chassis rollover sensing
Inertial measurement units
Platform stabilization

GENERAL DESCRIPTION

The ADXRS300 is a complete angular rate sensor (gyroscope) that uses Analog Devices' surface-micromachining process to make a functionally complete and low cost angular rate sensor integrated with all of the required electronics on one chip. The manufacturing technique for this device is the same high volume BIMOS process used for high reliability automotive airbag accelerometers.

The output signal, RATEOUT (1B, 2A), is a voltage proportional to angular rate about the axis normal to the top surface of the package (see Figure 4). A single external resistor can be used to lower the scale factor. An external capacitor is used to set the bandwidth. Other external capacitors are required for operation (see Figure 5).

A precision reference and a temperature output are also provided for compensation techniques. Two digital self-test inputs electromechanically excite the sensor to test proper operation of both sensors and the signal conditioning circuits. The ADXRS300 is available in a 7 mm × 7 mm × 3 mm BGA chip-scale package.

FUNCTIONAL BLOCK DIAGRAM

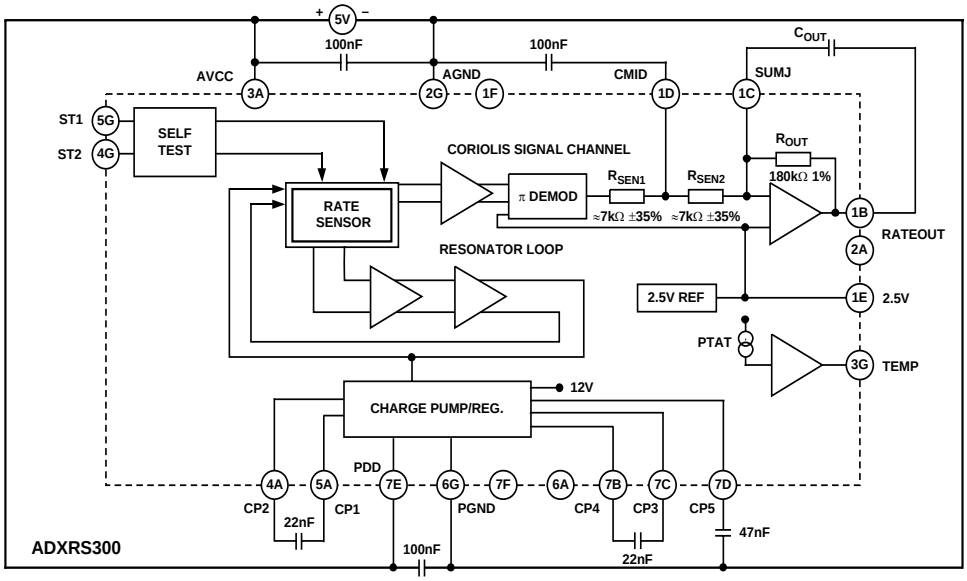


Figure 1.

Rev. B

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.326.8703

TABLE OF CONTENTS

REVISION HISTORY

3/04—Data Sheet Changed from Rev. A to Rev. B

Updated Format ..... Universal
Changes to Table 1 Conditions 3
Added Evaluation Board to Ordering Guide 8

3/03—Data Sheet Changed from Rev. 0 to Rev. A

Edit to Figure 3 5

SPECIFICATIONS

@T_A = 25°C, V_S = 5 V, Angular Rate = 0°/s, Bandwidth = 80 Hz (C_{OUT} = 0.01 µF), ±1*g*, unless otherwise noted.

Table 1.

		ADXRS300ABG			
Parameter	**Conditions**	**Min[1]**	**Typ**	**Max[1]**	**Unit**
SENSITIVITY	Clockwise rotation is positive output				
Dynamic Range[2]	Full-scale range over specifications range	±300			°/s
Initial	@25°C	4.6	5	5.4	mV/°/s
Over Temperature[3]	V_S = 4.75 V to 5.25 V	4.6	5	5.4	mV/°/s
Nonlinearity	Best fit straight line		0.1		% of FS
NULL					
Initial Null		2.3	2.50	2.7	V
Over Temperature[3]	V_S = 4.75 V to 5.25 V	2.3		2.7	V
Turn-On Time	Power on to ±½°/s of final		35		ms
Linear Acceleration Effect	Any axis		0.2		°/s/*g*
Voltage Sensitivity	V_{CC} = 4.75 V to 5.25 V		1		°/s/V
NOISE PERFORMANCE					
Rate Noise Density	@25°C		0.1		°/s/√Hz
FREQUENCY RESPONSE					
3 dB Bandwidth (User Selectable)[4]	22 nF as comp cap (see the Setting Bandwidth section)		40		Hz
Sensor Resonant Frequency			14		kHz
SELF-TEST INPUTS					
ST1 RATEOUT Response[5]	ST1 pin from Logic 0 to 1	–150	–270	–450	mV
ST2 RATEOUT Response[5]	ST2 pin from Logic 0 to 1	+150	+270	+450	mV
Logic 1 Input Voltage	Standard high logic level definition	3.3			V
Logic 0 Input Voltage	Standard low logic level definition			1.7	V
Input Impedance	To common		50		kΩ
TEMPERATURE SENSOR					
V_{OUT} at 298°K			2.50		V
Max Current Load on Pin	Source to common			50	µA
Scale Factor	Proportional to absolute temperature		8.4		mV/°K
OUTPUT DRIVE CAPABILITY					
Output Voltage Swing	I_{OUT} = ±100 µA	0.25		V_S – 0.25	V
Capacitive Load Drive		1000			pF
2.5 V REFERENCE					
Voltage Value		2.45	2.5	2.55	V
Load Drive to Ground	Source		200		µA
Load Regulation	0 < I_{OUT} < 200 µA		5.0		mV/mA
Power Supply Rejection	4.75 V_S to 5.25 V_S		1.0		mV/V
Temperature Drift	Delta from 25°C		5.0		mV
POWER SUPPLY					
Operating Voltage Range		4.75	5.00	5.25	V
Quiescent Supply Current			6.0	8.0	mA
TEMPERATURE RANGE					
Specified Performance Grade A	Temperature tested to max and min specifications	–40		+85	°C

[1] All minimum and maximum specifications are guaranteed. Typical specifications are not tested or guaranteed.
[2] Dynamic range is the maximum full-scale measurement range possible, including output swing range, initial offset, sensitivity, offset drift, and sensitivity drift at 5 V supplies.
[3] Specification refers to the maximum extent of this parameter as a worst-case value of T_{MIN} or T_{MAX}.
[4] Frequency at which response is 3 dB down from dc response with specified compensation capacitor value. Internal pole forming resistor is 180 kΩ. See the Setting Bandwidth section.
[5] Self-test response varies with temperature. See the Self-Test Function section for details.

ABSOLUTE MAXIMUM RATINGS

Table 2.

Parameter	Rating
Acceleration (Any Axis, Unpowered, 0.5 ms)	2000 *g*
Acceleration (Any Axis, Powered, 0.5 ms)	2000 *g*
$+V_S$	–0.3 V to +6.0 V
Output Short-Circuit Duration (Any Pin to Common)	Indefininte
Operating Temperature Range	–55°C to +125°C
Storage Temperature	–65°C to +150°C

Stresses above those listed under the Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Applications requiring more than 200 cycles to MIL-STD-883 Method 1010 Condition B (–55°C to +125°C) require underfill or other means to achieve this requirement.

Drops onto hard surfaces can cause shocks of greater than 2000 *g* and exceed the absolute maximum rating of the device. Care should be exercised in handling to avoid damage.

RATE SENSITIVE AXIS

This is a Z-axis rate-sensing device that is also called a yaw rate sensing device. It produces a positive going output voltage for clockwise rotation about the axis normal to the package top, i.e., clockwise when looking down at the package lid.

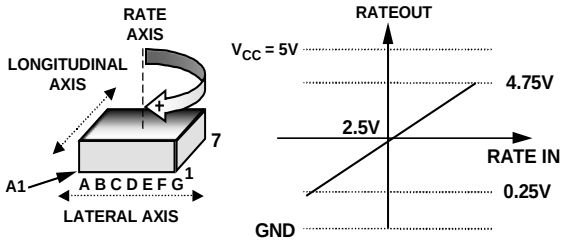


Figure 2. RATEOUT Signal Increases with Clockwise Rotation

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Electrostatic charges as high as 4000 V readily accumulate on the human body and test equipment and can discharge without detection. Although this product features proprietary ESD protection circuitry, permanent damage may occur on devices subjected to high energy electrostatic discharges. Therefore, proper ESD precautions are recommended to avoid performance degradation or loss of functionality.

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

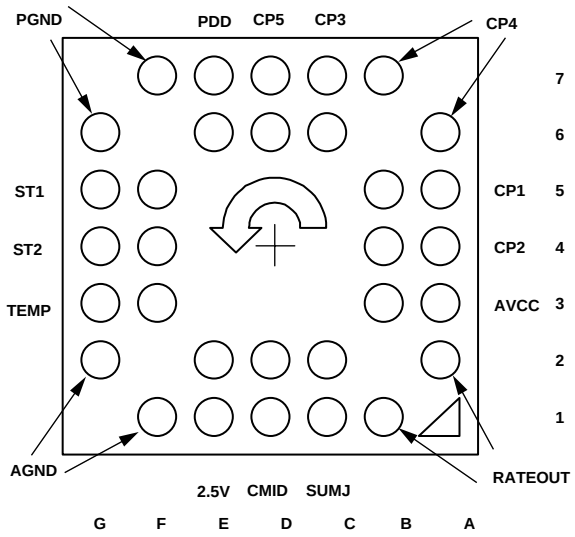


Figure 3. 32-Lead BGA (Bottom View)

Table 3. Pin Function Descriptions

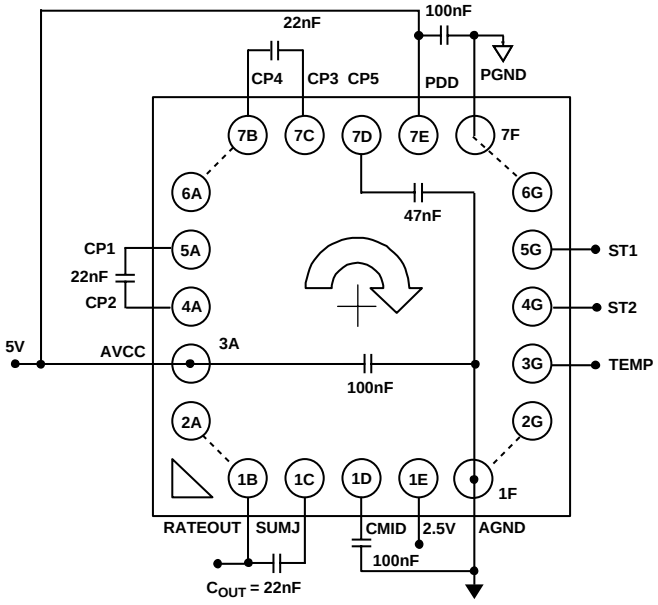
Pin No.	Mnemonic	Description
6D, 7D	CP5	HV Filter Capacitor—47 nF
6A, 7B	CP4	Charge Pump Capacitor—22 nF
6C, 7C	CP3	Charge Pump Capacitor—22 nF
5A, 5B	CP1	Charge Pump Capacitor—22 nF
4A, 4B	CP2	Charge Pump Capacitor—22 nF
3A, 3B	AVCC	+ Analog Supply
1B, 2A	RATEOUT	Rate Signal Output
1C, 2C	SUMJ	Output Amp Summing Junction
1D, 2D	CMID	HF Filter Capacitor—100 nF
1E, 2E	2.5V	2.5 V Precision Reference
1F, 2G	AGND	Analog Supply Return
3F, 3G	TEMP	Temperature Voltage Output
4F, 4G	ST2	Self-Test for Sensor 2
5F, 5G	ST1	Self-Test for Sensor 1
6G, 7F	PGND	Charge Pump Supply Return
6E, 7E	PDD	+ Charge Pump Supply

THEORY OF OPERATION

The ADXRS300 operates on the principle of a resonator gyro. Two polysilicon sensing structures each contain a dither frame, which is electrostatically driven to resonance. This produces the necessary velocity element to produce a Coriolis force during angular rate. At two of the outer extremes of each frame, orthogonal to the dither motion, are movable fingers that are placed between fixed pickoff fingers to form a capacitive pickoff structure that senses Coriolis motion. The resulting signal is fed to a series of gain and demodulation stages that produce the electrical rate signal output. The dual-sensor design rejects external *g*-forces and vibration. Fabricating the sensor with the signal conditioning electronics preserves signal integrity in noisy environments.

The electrostatic resonator requires 14 V to 16 V for operation. Since only 5 V is typically available in most applications, a charge pump is included on-chip. If an external 14 V to 16 V supply is available, the two capacitors on CP1–CP4 can be omitted and this supply can be connected to CP5 (Pin 7D) with a 100 nF decoupling capacitor in place of the 47 nF.

After the demodulation stage, there is a single-pole low-pass filter consisting of an internal 7 kΩ resistor (R_{SEN1}) and an external user-supplied capacitor (CMID). A CMID capacitor of 100 nF sets a 400 Hz ±3 5% low-pass pole and is used to limit high frequency artifacts before final amplification. The bandwidth limit capacitor, C_{OUT}, sets the pass bandwidth (see Figure 5 and the Setting Bandwidth section).



NOTE THAT INNER ROWS/COLUMNS OF PINS HAVE BEEN OMITTED FOR CLARITY BUT SHOULD BE CONNECTED IN THE APPLICATION.

Figure 4. Example Application Circuit (Top View)

SUPPLY AND COMMON CONSIDERATIONS

Only power supplies used for supplying analog circuits are recommended for powering the ADXRS300. High frequency noise and transients associated with digital circuit supplies may have adverse effects on device operation.

Figure 4 shows the recommended connections for the ADXRS300 where both AVCC and PDD have a separate decoupling capacitor. These should be placed as close to the their respective pins as possible before routing to the system analog supply. This minimizes the noise injected by the charge pump that uses the PDD supply.

It is also recommended to place the charge pump capacitors connected to the CP1–CP4 pins as close to the part as possible. These capacitors are used to produce the on-chip high voltage supply switched at the dither frequency at approximately 14 kHz. Care should be taken to ensure that there is no more than 50 pF of stray capacitance between CP1–CP4 and ground. Surface-mount chip capacitors are suitable as long as they are rated for over 15 V.

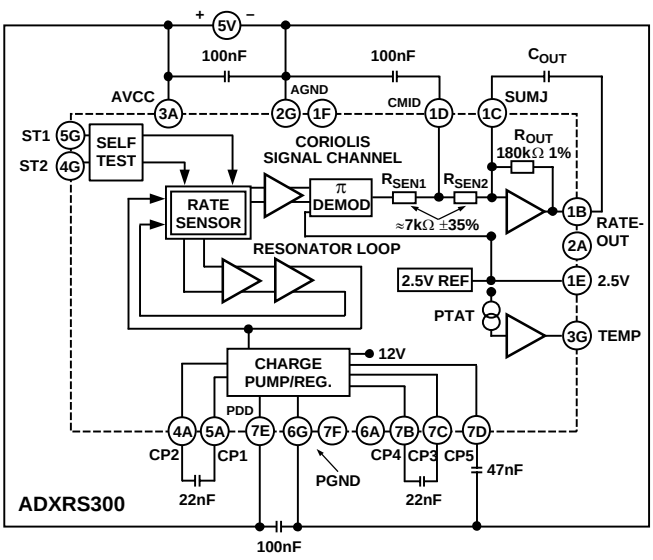


Figure 5. Block Diagram with External Components

SETTING BANDWIDTH

External capacitors CMID and C_{OUT} are used in combination with on-chip resistors to create two low-pass filters to limit the bandwidth of the ADXRS300's rate response. The –3 dB frequency set by R_{OUT} and C_{OUT} is

$$f_{OUT} = 1/(2 \times \pi \times R_{OUT} \times C_{OUT})$$

and can be well controlled since R_{OUT} has been trimmed during manufacturing to be 180 kΩ ± 1%. Any external resistor applied between the RATEOUT (1B, 2A) and SUMJ (1C, 2C) pins results in

$$R_{OUT} = (180\ \text{k}\Omega \times R_{EXT})/(180\ \text{k}\Omega \times R_{EXT})$$

The –3 dB frequency is set by RSEN (the parallel combination of R_{SEN1} and R_{SEN2}) at about 3.5 kΩ nominal; CMID is less well controlled since R_{SEN1} and R_{SEN2} have been used to trim the rate sensitivity during manufacturing and have a ±35% tolerance. Its primary purpose is to limit the high frequency demodulation artifacts from saturating the final amplifier stage. Thus, this pole of nominally 400 Hz @ 0.1 µF need not be precise. Lower frequency is preferable, but its variability usually requires it to be about 10 times greater (in order to preserve phase integrity) than the well-controlled output pole. In general, both –3 dB filter frequencies should be set as low as possible to reduce the amplitude of these high frequency artifacts and to reduce the overall system noise.

INCREASING MEASUREMENT RANGE

The full-scale measurement range of the ADXRS300 can be increased by placing an external resistor between the RATEOUT (1B, 2A) and SUMJ (1C, 2C) pins, which would parallel the internal R_{OUT} resistor that is factory-trimmed to 180 kΩ. For example, a 330 kΩ external resistor will give ~50% increase in the full-scale range. This is effective for up to a 4× increase in the full-scale range (minimum value of the parallel resistor allowed is 45 kΩ). Beyond this amount of external sensitivity reduction, the internal circuitry headroom requirements prevent further increase in the linear full-scale output range. The drawbacks of modifying the full-scale range are the additional output null drift (as much as 2°/sec over temperature) and the readjustment of the initial null bias (see the Null Adjust section).

USING THE ADXRS300 WITH A SUPPLY-RATIOMETRIC ADC

The ADXRS300's RATEOUT signal is nonratiometric, i.e., neither the null voltage nor the rate sensitivity is proportional to the supply. Rather they are nominally constant for dc supply changes within the 4.75 V to 5.25 V operating range. If the ADXRS300 is used with a supply-ratiometric ADC, the ADXRS300's 2.5 V output can be converted and used to make corrections in software for the supply variations.

NULL ADJUST

Null adjustment is possible by injecting a suitable current to SUMJ (1C, 2C). Adding a suitable resistor to either ground or to the positive supply is a simple way of achieving this. The nominal 2.5 V null is for a symmetrical swing range at RATEOUT (1B, 2A). However, a nonsymmetric output swing may be suitable in some applications. Note that if a resistor is connected to the positive supply, then supply disturbances may reflect some null instabilities. Digital supply noise should be avoided, particularly in this case (see the Supply and Common Considerations section).

The resistor value to use is approximately

$$R_{NULL} = (2.5 \times 180{,}000)/(V_{NULL0} - V_{NULL1})$$

V_{NULL0} is the unadjusted zero rate output, and V_{NULL1} is the target null value. If the initial value is below the desired value, the resistor should terminate on common or ground. If it is above the desired value, the resistor should terminate on the 5 V supply. Values are typically in the 1 MΩ to 5 MΩ range.

If an external resistor is used across RATEOUT and SUMJ, then the parallel equivalent value is substituted into the preceding equation. Note that the resistor value is an estimate since it assumes V_{CC} = 5.0 V and V_{SUMJ} = 2.5 V.

SELF-TEST FUNCTION

The ADXRS300 includes a self-test feature that actuates each of the sensing structures and associated electronics in the same manner as if subjected to angular rate. It is activated by standard logic high levels applied to inputs ST1 (5F, 5G), ST2 (4F, 4G), or both. ST1 causes a voltage at RATEOUT equivalent to typically –270 mV, and ST2 causes an opposite +270 mV change. The self-test response follows the viscosity temperature dependence of the package atmosphere, approximately 0.25%/°C.

Activating both ST1 and ST2 simultaneously is not damaging. Since ST1 and ST2 are not necessarily closely matched, actuating both simultaneously may result in an apparent null bias shift.

CONTINUOUS SELF-TEST

The one-chip integration of the ADXRS300 gives it higher reliability than is obtainable with any other high volume manufacturing method. Also, it is manufactured under a mature BIMOS process that has field-proven reliability. As an additional failure detection measure, power-on self-test can be performed. However, some applications may warrant continuous self-test while sensing rate. Application notes outlining continuous self-test techniques are also available on the Analog Devices website.

OUTLINE DIMENSIONS

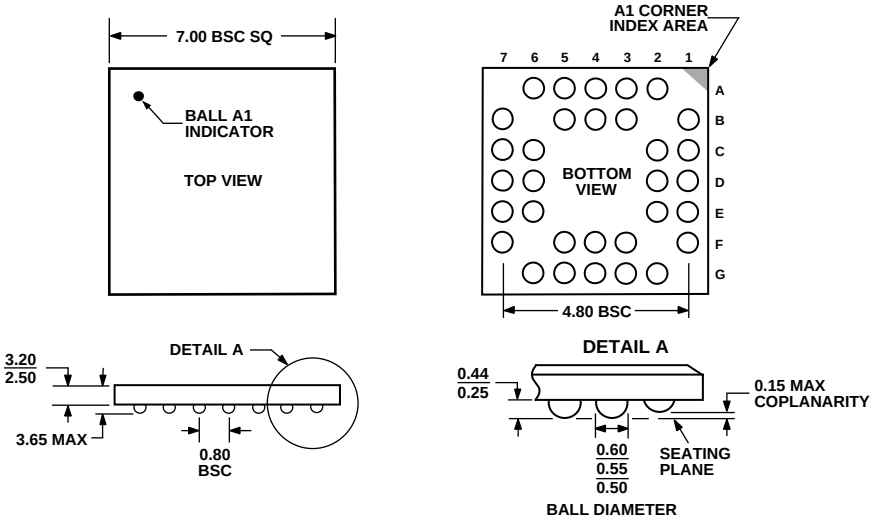


Figure 6. 32-Lead Chip Scale Ball Grid Array [CSPBGA]
(BC-32)
Dimensions shown in millimeters

ORDERING GUIDE

Model	Temperature Range	Package Description	Package Outline
ADXRS300ABG	–40°C to +85°C	32-Lead BGA	BC-32
ADXRS300ABG-Reel	–40°C to +85°C	32-Lead BGA	BC-32
ADXRS300EB		Evaluation Board	

C03227–0–3/04(B)

±300°/s Single Chip Rate Gyro Evaluation Board

ADXRS300EB

GENERAL DESCRIPTION

The ADXRS300EB is a simple evaluation board that allows the user to quickly evaluate the performance of the ADXRS300ABG yaw rate gyro. No additional external components are required for operation. The ADXRS300EB has a 20-lead dual-in-line (0.3 inch width by 0.1 inch pin spacing) interface that allows the user to easily prototype products without having to deal with BGA soldering. The 0.4 square inch outline of the ADXRS300EB is still among the smallest gyros available today.

CIRCUIT DESCRIPTION

The schematic of the ADXRS300EB is shown in Figure 1. It is identical to the suggested application shown in the ADXRS300ABG data sheet.

The analog and power grounds (AGND and PGND) have separate ground planes and are joined at one point. The user may cut this trace if separate ground schemes are desired.

Note that the analog supply voltage and charge pump supply voltage (AVCC and PDD) are not connected on the ADXRS300EB, and the user must connect these as appropriate to the application.

The parts layout of the ADXRS300EB is shown in Figure 2, and the part list for the ADXRS300EB is shown in Table I. As delivered, the ADXRS300EB is set for 40 Hz bandwidth (C_{OUT} = 22 nF). The user may add an additional external capacitor to further reduce the bandwidth and improve the noise floor.

SPECIAL NOTES ON HANDLING

Note that the ADXRS300EB is not reverse polarity protected. Reversing the power supply, or applying inappropriate voltages to any pin (outside the ADXRS300 data sheet's Absolute Maximum Ratings), may damage the ADXRS300EB.

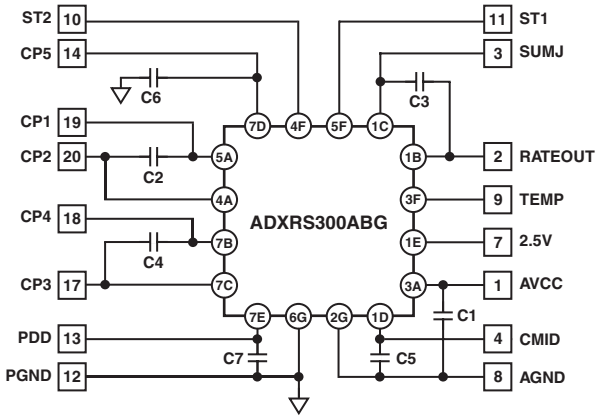


Figure 1. ADXRS300EB Schematic

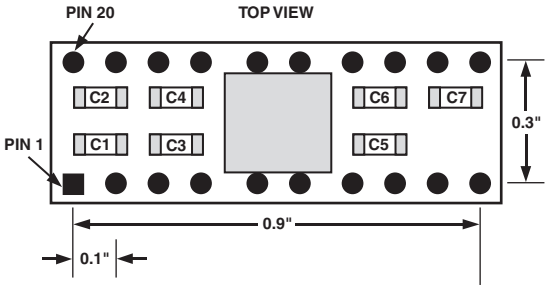


Figure 2. ADXRS300EB Parts Layout

Table I. ADXRS300EB Component Values

Component	Values (nF)
C1	100
C2	22
C3	22
C4	22
C5	100
C6	47
C7	100

C03540–0–1/03(0)

PRINTED IN U.S.A.

REV. 0

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781/329-4700 www.analog.com
Fax: 781/326-8703

Digital Communication with SM5800 Series Parts

OVERVIEW

The SM5800 series pressure product offers the corrected pressure output in both analog and digital formats. Accessing the analog output is an easy process requiring a simple voltage measurement. But accessing the digital output requires an understanding of the digital communication protocols and memory locations containing the digital output data. The purpose of this note is to provide the information needed to establish a connection with a SM5800 part and then read the digital output of the part.

INTRODUCTION

The SM5800 series pressure product utilizes the Inter Integrated Circuit (I^2C) Bus interface for all digital communication. Using the I^2C bus the corrected digital pressure and digital temperature values are accessed from on board memory registers.

DIGITAL COMMUNICATION BASICS

The I^2C bus interface utilizes a two-wire method for carrying information back and forth between integrated circuit devices connected to the bus. The two-wire method is comprised of a bi-directional, 8-bit serial data (SDA) line providing the path for data transfers and a serial clock (SCL) line for synchronizing the data transfers.

A Master device in the form of a personal computer or microcontroller initiates and terminates data transfers on the SDA line and generates the clock signal on the SCL line. A Slave device (which represents the SM5800 series product) receives requests from the Master and transmits data according to the request. When multiple Slave devices are connected to the bus, each Slave must have a unique address and the Master must use this unique address when performing data transfers with a particular Slave. A diagram of the I^2C bus with a Master and multiple Slaves (or SM5800 parts) is shown in Figure 1.

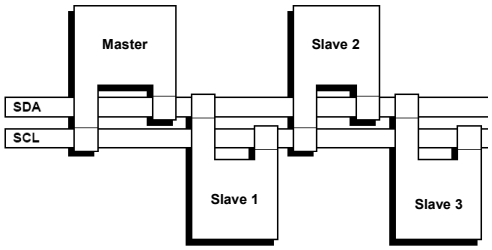


Figure 1. Multiple SM5800 series devices connected to the I^2C bus

DATA TRANSFER

The data transfer process begins with the Master issuing a START (S) command, which alerts all Slave devices of the pending transfer request. The Master then specifies which Slave device to communicate with and how the data will flow between the Master and Slave. Next, the Master waits for the addressed Slave to respond. When the desired Slave responds, the Master and Slave begin the process of transmitting data back and forth. The transfer process ends with the Master issuing a STOP (P) command. Figure 2 provides a diagram showing a simple data transfer sequence.

START & STOP CONDITIONS

Initiated by the Master, a START conditioned is identified by a HIGH to LOW transition of the SDA line while maintaining a stable HIGH condition on the SCL line. All data transfer must begin with a START condition and the bus lines are considered busy after a START condition is issued. To end the transfer of data the Master initiates a STOP condition, which is identified by a LOW to HIGH transition of the SDA line while maintaining a stable HIGH condition on the SCL line.

Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

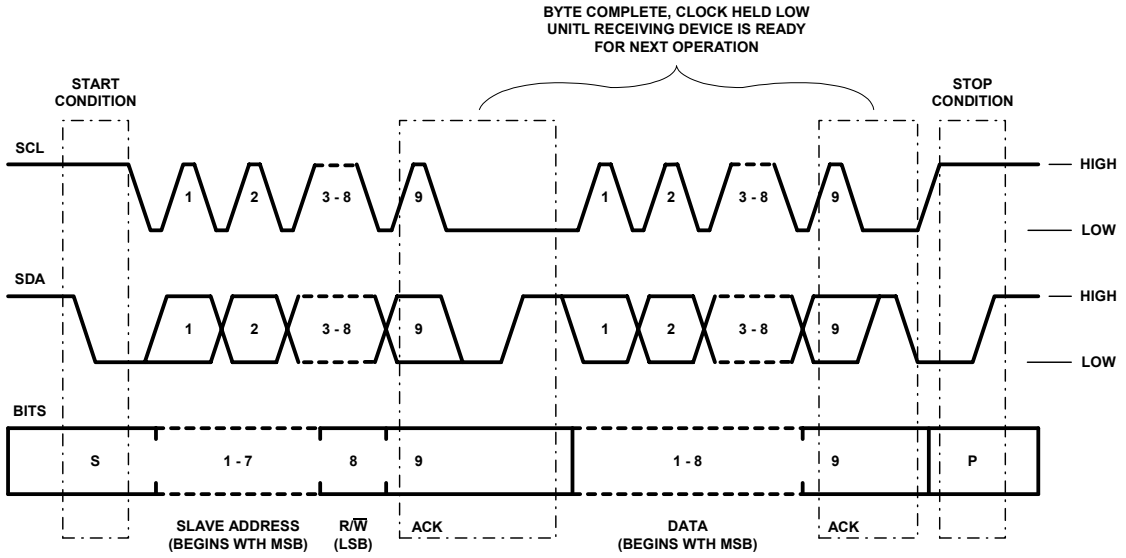


Figure 2. Diagram of a simple data transfer sequence

DATA FORMAT & VALIDITY

The SDA line is an 8-bit digital line and this defines the size of each byte transferred. An acknowledge bit starts with the Transmitter releasing the SDA line HIGH at the start of the clock cycle immediately following a byte transfer. The Receiver must then pull the SDA line LOW during the HIGH period of the same clock cycle. The set-up and hold times for the SDA and SCL lines are provided in APPENDIX A.

Every byte transferred from Transmitter to Receiver starts with the most significant bit (MSB). Valid data transfers only occur during the HIGH period of the SCL clock. If a Receiver needs time to carry out a request before receiving another byte, it can hold the SCL line LOW. To do this the Receiver performs an internal interrupt to hold the SCL LOW, which forces the Transmitter to wait before sending the next byte. When ready, the Receiver releases the SCL line and data transfer continues.

ADDRESSING A SLAVE DEVICE

All Slave devices must have a unique identification or address. A Slave address cannot exceed a length of seven bits. The Slave address shares space in the address byte with a read/write bit and the maximum length of the address byte is eight bits. To construct the address byte, place the Slave address at the MSB of the address byte and place the read/write bit at the least significant bit (LSB).

The address byte always follows a START condition. When an address byte is sent over the SDA line, each Slave compares the MSB of this byte with its own unique address. The Slave with the matching address becomes active and responds by sending an acknowledge bit along the SDA line to the Master. How the Slave responds to the next byte transmitted by the Master following the acknowledgement depends on the read/write bit. When the LSB of the address byte is “1” the Slave transmits data for the Master to read. When the LSB of the address byte is ”0” the Slave waits for the Master to write data to the Slave.

Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

EEPROM SPECIFIC READ OPERATIONS

The Slave devices with onboard EEPROM contain an internal byte register, which will store the register address of the EEPROM register recently accessed by the Master. To set the value of the internal byte register a *Random Address Read* operation must be performed. Once the internal byte register is set then the value of the EEPROM register corresponding to the address found in the internal byte register can be read repeatedly through the use of the *Current Address Read* operation. The two read operations are detailed in the following:

Random Address Read

Following the initial START condition, the Master performs a dummy write operation to load the EEPROM register address into the internal byte register. The Slave acknowledges this dummy write operation. The Master then sends a Repeated START (Sr) condition. Following the repeated START condition, the Master again sends the Slave address byte with the read/write bit set to "1". The Slave acknowledges this and outputs the EEPROM register data corresponding to the address stored in the internal byte register. The Master does not acknowledge having received the byte output of the Slave, but instead terminates the data transfer session with a STOP condition. Pseudo code detailing the steps of the random address read operation can be found in APPENDIX B. Figure 3 shows a diagram of the random address read operation.

Current Address Read

Following a START condition, the Master transmits the first byte with the read/write set to "1". The Slave acknowledges this and outputs the EEPROM register data corresponding to the address stored in the internal byte register. The Master does not acknowledge having received the byte output of the Slave, but instead terminates the data transfer session with a STOP condition. Figure 4 shows a diagram of the current address read operation.

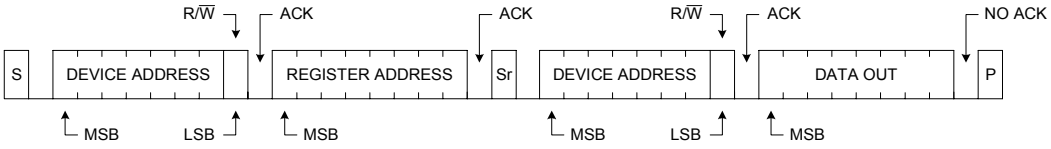


Figure 3. EEPROM specific Random Address Read operation

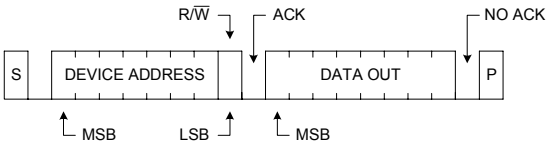


Figure 4. EEPROM specific Current Address Read operation

Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

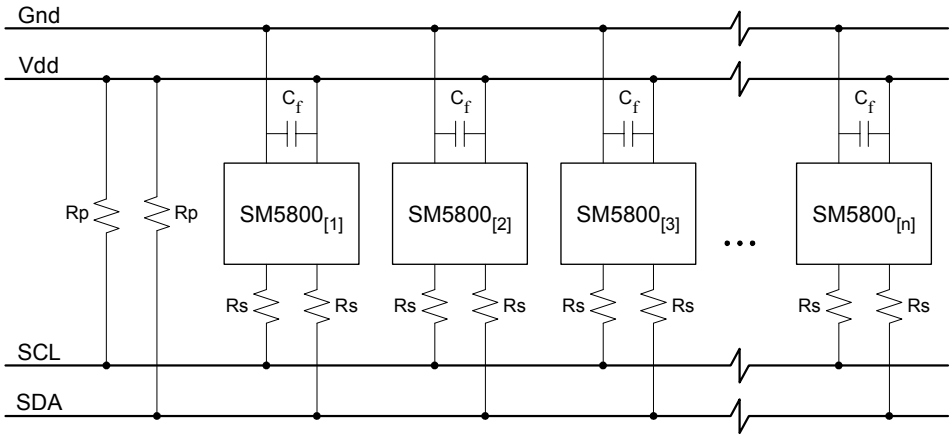


Figure 5. Implementation of Rp and Rs resistors along the I²C-bus

PRECAUTIONS

To protect devices connected to the SCL and SDA lines, series resistors must be utilized. The series resistors protect against high-voltage spikes traveling along the bus lines and help reduce interference and ringing. In addition to the series resistors, the SCL and SDA lines must be connected to the positive supply voltage (Vdd) using either a current-source or a pull-up resistor. A current-source or pull-up resistor helps maintain the HIGH state of the SCL and SDA lines and ensures that the signals are pulled from the LOW to HIGH state within the set rise time. For bus lines with a low capacitive load (<100pF) use pull-up resistors and when dealing with a high capacitive load (>100pF) use an external current-source pull-up. Figure 5 provides a diagram showing how to implement series and parallel resistors in the I^2C bus.

ACCESSING THE DIGITAL OUTPUT

A total of four digital values are accessible through the digital output of the SM5800. These digital values are:

- Band-gap voltage (11-bit)
- Temperature (11-bit)
- Uncorrected pressure (11-bit)
- Corrected pressure (12-bit)

The digital output values are stored in addressable memory registers and updated with each clock cycle. The length of the digital output values range from eleven to twelve bits. A single memory register has a maximum capacity of eight bits. Since a single digital output value exceeds the storage capacity of a single register a total of two consecutive memory registers are reserved for each digital output value. Each digital value is split in half with the LSB placed into the first register and the MSB placed into the second register. Table 1 provides a list of the register addresses reserved for the LSB and MSB of each digital output value.

Table 1. Digital Output Memory Mapping

REGISTER	DESCRIPTION
128	Corrected pressure, LSB [5:0]
129	Corrected pressure, MSB [11:6]
130	Temperature, LSB [5:0]
131	Temperature, MSB [11:6]
132	Un-corrected pressure, LSB [5:0]
133	Un-corrected pressure, MSB [11:6]
134	Band-gap voltage, LSB [5:0]
135	Band-gap voltage, MSB [11:6]

Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

CONCLUSIONS

SM5800 series parts are now capable of digital communication through an I^2C bus interface. The two-wire I^2C bus allows multiple SM5800 series parts to be accessed. Different SM5800 models and pressure ranges can be connected to the same I^2C bus as long as each SM5800 series part has a unique bus address. Please consult the factory to obtain SM5800 series parts programmed with addresses other than the factory default value of *95*.

The digital interface provides access to the digital band-gap voltage, digital temperature, uncorrected digital pressure, and corrected digital pressure values of a SM5800 series part. Now a SM5800 series part can be used in a purely digital system thus eliminating the need of an analog-to-digital converter for the corrected analog pressure output.

APPENDIX A. *I^2C Bus Characteristics*

PARAMETER	MIN	TYP	MAX	UNITS
SCL frequency	0	–	100	kHz
LOW period of the SCL clock	4.7	–	–	µs
HIGH period of the SCL clock	4.0	–	–	µs
Rise time for SCL and SDA bus signals	–	–	1000	ns
Fall time for SCL and SDA bus signals	–	–	300	ns
Set-up time for repeated START condition	4.7	–	–	µs
Set-up time for Data	250	–	–	ns
Set-up time for STOP condition	4.0	–	–	µs
SDA transfer rate	–	–	100	kbits/s
LOW level input voltage for each bus line	-0.5	–	0.3 * Vdd	V
HIGH level input voltage for each bus line	0.7 * Vdd	–	–	V
Supply Voltage, Vdd	–	5	–	V
Supply Voltage Filter Capacitance, Cf		0.1		pF
Capacitive load per bus line, Cb	–	–	400	pF
Parallel Resistor, Rp	–	10	–	kΩ
Series Resistor, Rs	–	100	–	Ω
Noise limit at the LOW level for each device connected	0.1 * Vdd	–	–	V
Noise limit at the HIGH level for each device connected	0.2 * Vdd	–	–	V

Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

APPENDIX B. *Random Address Read* Pseudo Code

```
//********** MAIN PROGRAM **********
START
  START_CONDITION;
  DEVICE_ADDRESS;                          //Set the read/write bit to "0" (= write)
  ACKNOWLEDGEMENT;
  REGISER_ADDRESS;
  ACKNOWLEDGEMENT;
  RE-START_CONDITION:
  DEVICE_ADDRESS;                          //Set the read/write bit to "1" (= read)
  ACKNOWLEDGEMENT;
  DATA_OUT;
  NO_ACKNOWLEDGEMENT;
  STOP_CONDITION;
END

//********** SUB-ROUTINES **********
START_CONDITION
  SDA(W)=0;                                //Set SDA bus line "low"
  DELAY;
  SCL(W)=1;                                //Set SCL bus line "high"
  DELAY;

DEVICE_ADDRESS
  n=7;
  FOR i=0 to n-1
    SDA(W)=DEVICE_ADD(i);                  //Writing ith bit of the Device Address
    DELAY;
    SCL(W)=0;                              //Set SCL bus line "low"
    DELAY;
    SCL(W)=1;                              //Set SCL bus line "high"
    DELAY
    i=i+1;
  END;
  SDA(W)=R_or_W;                           //Set the read/write bit
  DELAY;
  SCL(W)=0;                                //Set SCL bus line "low"
  DELAY;
  SCL(W)=1;                                //Set SCL bus line "high"
  DELAY;

ACKNOWLEDGEMENT
  SDA(W)=1;                                //Set SDA bus line "high"
  DELAY;
  SCL(W)=0;                                //Set SCL bus line "low"
  DELAY;
  ACK=SDA(R);                              //Reading ACK bit
  DELAY;
  SCL(W)=1;                                //Set SCL bus line "high"
  DELAY;

REGISTER_ADDRESS
  n=8;
  FOR i=0 to n-1
    SDA(W)=REGISTER_ADD(i);                //Writing ith bit of the Register Address
    DELAY;
    SCL(W)=0;                              //Set SCL bus line "low"
    DELAY;
    SCL(W)=1;                              //Set SCL bus line "high"
    DELAY
    i=i+1;
  END;
```


Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

```
RE-START_CONDITION
  SDA(W)=1;                              //Set SDA bus line "high"
  DELAY;
  SCL(W)=0;                              //Set SCL bus line "low"
  DELAY;
  SDA(W)=0;                              //Set SDA bus line "low"
  DELAY;
  SCL(W)=1;                              //Set SCL bus line "high"
  DELAY;

DATA_OUT
  n=8;
  FOR i=0 to n-1
    DATA(i)=SDA(R);                      //Reading i^th bit of Data Out
    DELAY;
    SCL(W)=0;                            //Set SCL bus line "low"
    DELAY;
    SCL(W)=1;                            //Set SCL bus line "high"
    DELAY
    i=i+1;
  END;

NO_ACKNOWLEDGEMENT
  SDA(W)=1;                              //Set SDA bus line "high"
  DELAY;
  SCL(W)=0;                              //Set SCL bus line "low"
  DELAY;
  SCL(W)=1;                              //Set SCL bus line "high"
  DELAY;

STOP_CONDITION
  SDA(W)=0;                              //Set SDA bus line "low"
  DELAY;
  SCL(W)=0;                              //Set SCL bus line "low"
  DELAY;
  SDA(W)=1;                              //Set SDA bus line "high"
  DELAY;
```

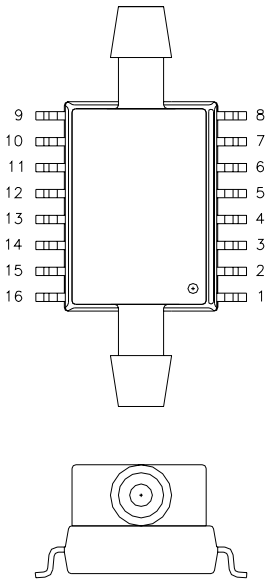

Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

APPENDIX C. Glossary of Terms

ACKNOWLEDGE (ACK) BIT :	LOW signal sent by the over the SDA line during the 9th clock pulse indicating a successful data transfer.
ADDRESS :	7-bits used to identify a specific SLAVE on the I^{2}C-bus and found in the MSB of the first byte transmitted after a START condition.
BUS CAPACITENCE (Cb) :	Total capacitance of the wire, connections, and pins found on the bus.
I^{2}C Bus :	Inter Integrated Circuit, a two-wire communication bus consisting of a bi-directional serial data line and a clock line. (Also known as *Inter IC.*)
MASTER :	A desktop computer or microcontroller. Generates the clock signal and initiates/terminates the data transfer process with a Slave.
PARALLEL RESISTOR (Rp) :	Resistor placed between the bus and supply voltage lines.
READ (R) :	8th bit of the ADDRESS byte with a value of "1".
RECEIVER :	Obtains data transmitted over the SDA line.
REPEATED START CONDITION (Sr) :	See START Condition. Occurs during the data transfer process.
SERIAL CLOCK (SCL) :	Line used to synchronize data transfers between a Master and Slave device.
SERIAL DATA (SDA) :	Bi-directional line used to transfer data between a Master and Slave device.
SERIES RESISTOR (Rs) :	Resistor placed between the bus line and a Slave device.
START CONDITION (S) :	Transitioning from HIGH to LOW on the SDA line while the SCL line remains HIGH.
STOP CONDITION (P) :	Transitioning from LOW to HIGH on the SDA line while the SCL line remains HIGH.
SUPPLY VOLTAGE (Vdd) :	Voltage powering I^{2}C devices.
SLAVE :	A SM5800 series part. Receives/sends data when addressed by a Master.
TRANSMITTER :	Sends data over the SDA line.
$WRITE(\overline{W})$:	The 8th bit of the ADDRESS byte with a value of "0".

Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

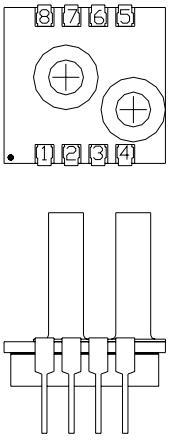
APPENDIX D. SM5800 Package Pin-Out Diagrams



PIN NO.	PIN DESCRIPTION
1	NC
2	NC
3	SERIAL DATA (SDA)
4	SERIAL CLOCK (SCL)
5	NC
6	NC
7	NC
8	NC
9	NC
10	NC
11	NC
12	GROUND
13	Vexc = 5.000 VDC
14	NC
15	NC
16	ANALOG OUTPUT

NOTE: DO NOT GROUND NC PINS.
NC PINS MUST FLOAT.

Figure 6. SM5822/SM5872 package pin-out diagram



PIN NO.	PIN DESCRIPTION
1	NC
2	GROUND
3	NC
4	SERIAL DATA (SDA)
5	SERIAL CLOCK (SCL)
6	NC
7	Vexc = 5.000 VDC
8	ANALOG OUTPUT

NOTE: DO NOT GROUND NC PINS.
NC PINS MUST FLOAT.

Figure 7. SM5812/SM5852 package pin-out diagram

Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

SILICON MICROSTRUCTURES INCORPORATED

SM5822/SM5872
Co-Integrated Pressure Sensor

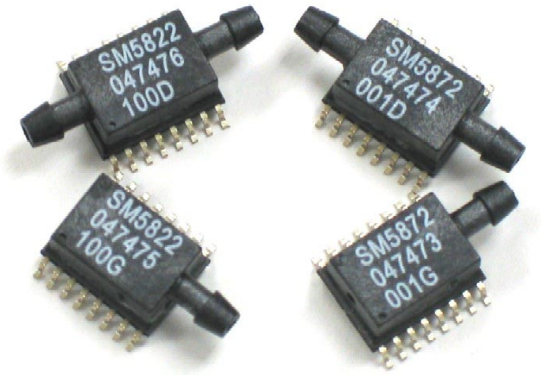
- ***FULLY INTEGRATED CMOS PRESSURE SENSOR SYSTEM IN A STANDARD SO-16 PACKAGE***
- ***FACTORY CALIBRATED SINGLE-CHIP INTEGRATED SENSOR***
- ***IMPROVED ACCURACY COMPARED TO COMPETITIVE PRODUCTS***
- ***LOWER COST ASSOCIATED WITH SENSOR INTEGRATION AND PACKAGE DESIGN***

DESCRIPTION

The Silicon Microstructures **SM5822** and **SM5872** series of OEM integrated pressure sensors combine state-of-the-art pressure sensor technology with on-chip signal processing technology to produce an amplified, fully conditioned, multi-order pressure and temperature compensated sensor die in a low cost SO-16 plastic package.

The sensor utilizes a CMOS digital signal processor and on-chip EEPROM for storage of calibration and compensation data. This allows the device to be factory calibrated including the effects from packaging stress. It also permits multi-order temperature and pressure compensation to achieve increased accuracy compared to conventional amplified pressure systems.

The **SM5822/SM5872** series pressure sensor is a factory calibrated ready-to-use sensor system. The model **SM5822** is designed for operating pressure ranges from 0-5 PSI to 0-100 PSI full-scale in gauge, absolute and differential pressure configurations. The model **SM5872** is designed for operating low-pressure ranges from 0-0.6 PSI up to 0-1.5 PSI full-scale for gauge differential and single-ended differential pressure applications. For both models, the sensor output is ratiometric with the supply voltage.



FEATURES

- Amplified, calibrated, fully signal conditioned output span of 4.0 VDC FS (0.5 to 4.5 V signal)
- Digital temperature and calibrated pressure available through I^2C interface
- Output ratiometric with supply voltage
- Multi-order correction for pressure non-linearity (factory programmed)
- Multi-order correction for temperature coefficient of span and offset (factory programmed)
- Gage, differential, and absolute versions

TYPICAL APPLICATIONS

- Heating, Ventilation and Air Conditioning (HVAC)
- Automotive pressure sensing
- Barometric measurement
- Medical instrumentation
- Pneumatic control
- Gas flow
- Respirators and ventilators

Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

THEORY OF OPERATION

The operation of the signal processor is depicted in the block diagram below. The internal pressure sensor is a piezoresistive bridge. This transduces the applied pressure into an electronic signal, which is inputted into the integrating amplifier.

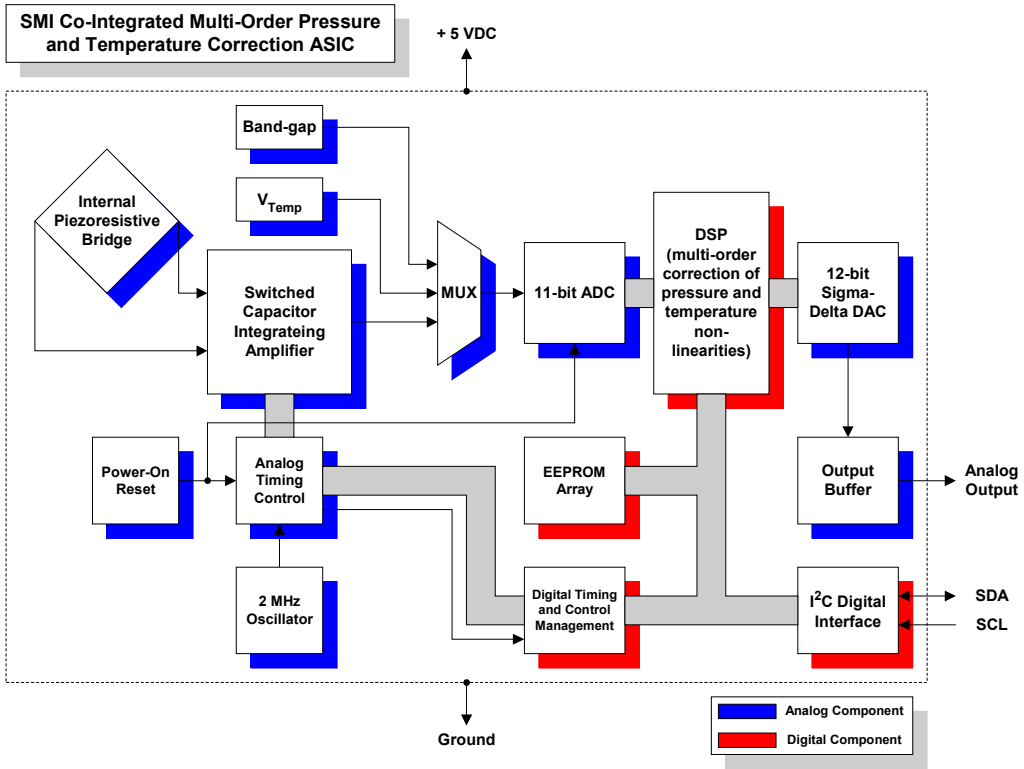
During the amplification step an offset correction factor is added in order to allow maximum gain for a given pressure while minimizing the offset error.

The signal is then passed to an 11-bit analog to digital converter (ADC). The ADC samples the signal multiple times and uses the sum of those samples as a 13-bit word.

A digital signal processor (DSP) is then used to correct and calibrate the pressure signal. The DSP provides multi-order correction of both pressure and temperature non-linearity through the use of factory-programmed coefficients. A combined total of twenty coefficients are available for correcting pressure and temperature non-linearity. The unique coefficients are determined during a calibration process performed at the factory. Factory calibration is the last step performed which means the effect of the package on the pressure signal will also be taken into account. This provides a great advantage over conventional laser-trimming approaches.

The DSP outputs a corrected digital word, which travels to a 12-bit digital to analog converter (DAC) to provide a calibrated analog output. In addition to the analog output, the corrected pressure signal is accessible through an I^2C digital interface.

See SMI application note AN05-001 for a detailed description of how to read out the digital corrected pressure signal using the I2C bus interface.



Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

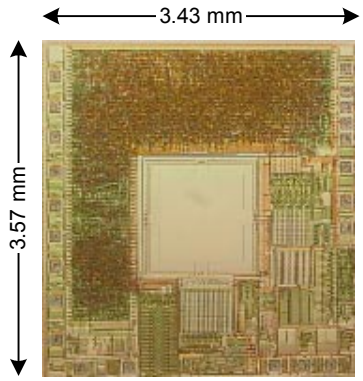
CHARACTERISTICS FOR SM5822/SM5872 – *SPECIFICATIONS*

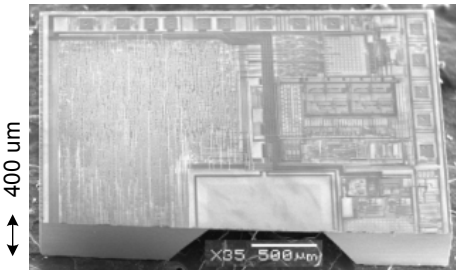
All parameters are measured at room temperature while applying 5.000V supply, unless otherwise specified.

	Absolute[1], Gage & Single[2]			Differential				
	MIN	TYP	MAX	MIN	TYP	MAX	UNITS	NOTES
Zero output (absolute and gauge)	0.42	0.50	0.58				V	3
Zero output (differential)				2.42	2.50	2.58	V	3
Output Span	3.92	4.00	4.08	1.96	2.00	2.04	V FS	3, 4
Total Error			1			1	%FS	
Response Time	2			2			msec	
Supply voltage	4.75	5.00	5.25	4.75	5.00	5.25	V	3, 7
Current consumption			10			10	mA	
Overpressure		>5X			>5X		%FS	6
Operating temperature range	-40	25	+125	-40	25	+125	°C	
Compensated temperature range	0	25	+70	0	25	+70	°C	
Storage temperature range	-55	25	+135	-55	25	+135	°C	
Media compatibility								8

Notes:

1. Absolute parts are only offered in the SM5822 Series.
2. Single-ended parts (Pressure Type - S) have 2 ports and are for higher gain differential applications where the differential pressure is **always** positive.
3. Sensor output is ratiometric to supply.
4. Full-scale (FS) is defined as zero pressure to rated pressure; differential parts can be used ±FS. Absolute and Gauge zero output is 0.5 V typical and full-scale output is 4.5 V. Span is the difference between Full-scale output and zero output, (4 V). For Differential parts, the negative full-scale is typically at 0.5 V, zero is typically 2.5 V, and positive full-scale is 4.5 volts to give a span of ±2.0 V.
5. Total accuracy is defined as departure from ideal response at all temperatures and pressures as a percentage of full-scale. This includes temperature error and pressure non-linearity errors.
6. Or 225 PSI, whichever is less. Output amplifier will saturate at about 0.25 V for applied pressure below the rated Zero and at about 4.75 V for applied pressure above the rated Full-scale.
7. A 100 nF filter capacitor must be placed between Vsupply and Ground.
8. Clean, dry gas compatible with wetted materials. Wetted materials include Pyrex glass, silicon, Kovar™, epoxy, RTV, gold, aluminum, Ultem™ plastic, Duraplast™ plastic.

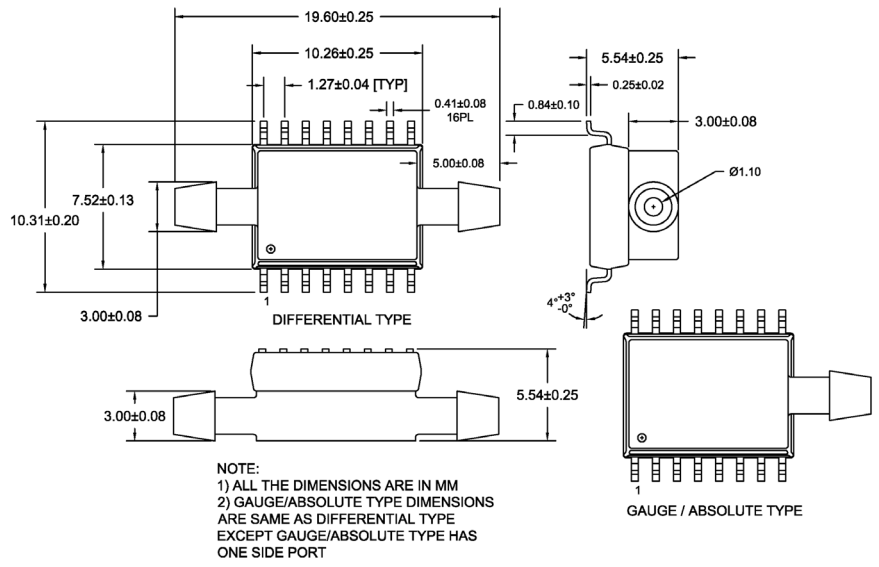




Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

ADDITIONAL INFORMATION

Package Dimensions & Pin-Out



NOTE:
1) ALL THE DIMENSIONS ARE IN MM
2) GAUGE/ABSOLUTE TYPE DIMENSIONS ARE SAME AS DIFFERENTIAL TYPE EXCEPT GAUGE/ABSOLUTE TYPE HAS ONE SIDE PORT

PIN	DESCRIPTION
1	NC
2	NC
3	**SERIAL DATA (SDA)**
4	**SERIAL CLOCK (SCL)**
5	NC
6	NC
7	NC
8	NC
9	NC
10	NC
11	NC
12	**GROUND**
13	**V_{exc} = 5.000 VDC**
14	NC
15	NC
16	**ANALOG OUTPUT**

NOTES:
- Do **not** connect to NC pins.
- External connections to NC pins will cause part malfunction.

Ordering Information

Pressure Type
A : Absolute (1 port)
D : Differential (2 ports)
G : Gauge (1 port)
S : Single-ended (2 ports)

Cap Type
B : Horizontal barbed ports

Model Number — **Pressure Type**

SM5822 - XXX - Y - Z

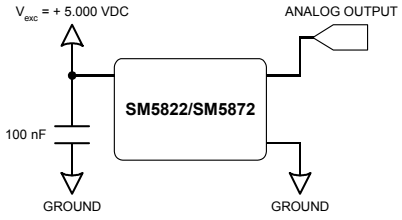
Pressure Range — **Cap Type**

Notes
1. Absolute configuration only available in SM5822.
2. Single-ended configuration is for higher gain differential applications where differential pressure is **always** positive.
3. Other configurations available on large orders. Consult SMI for details.

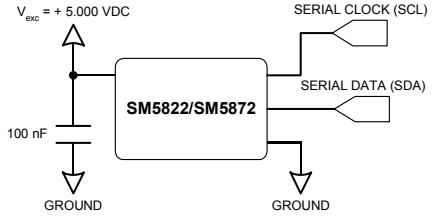
Full-Scale Pressure Ranges

SM5822	PSI	[kPa]	SM5872	PSI	[kPa]
005	5	[34.5]	006	0.6	[4.1]
015	15	[103.4]	015	1.5	[10.3]
030	30	[206.8]			
060	60	[413.7]			
100	100	[689.5]			

Wiring Diagrams



Wiring diagram for analog output



Wiring diagram for digital output

Silicon Microstructures, Inc. ♦ 1701 McCarthy Blvd. ♦ Milpitas, CA 95035 ♦ USA
Tel: 408-577-0100 ♦ Fax: 408-577-0123 ♦ Sales@si-micro.com ♦ www.si-micro.com

HARDWARE DOKUMENTATION

Avisaro Module (WLAN und CF Memory)

- Spannungsversorgung, Beschaltung IO-Pins
- Schnittstellen: Belegung und Timing (RS232, I2C, Parallel)
- Pinbelegung der Avisaro Modul
- Abmessungen
- Programmierbeispiele

Version 3.1

Änderungshistorie

2004-06-24		Erstellung
2004-09-27		Erweiterungen nach Kundenfeedback
2004-10-31		Erweiterungen Compact Flash Memory Modul
2005-02-03		Viele Detailergänzungen
2005-05-01		Schnittstellen Dokumentation hinzugefügt. Programmierbeispiele hinzugefügt.
2005-06-01		Fehler beim Parallel-Protokoll beseitigt

Kontakt:

Avisaro AG
Vahrenwalderstr. 7 (tch)
30165 Hannover

Telefon:
+49-(0)511-9357411
Telefax:
+49-(0)721-151 254826

eMail:
support@avisaro.com

INHALTSVERZEICHNIS

STROMVERSORGUNG UND IO-PINS

STROMVERSORGUNG

Das Avisaro Modul wird mit 3.3 V (+/- 0,1V) betrieben. Der verwendete Prozessor kann auch bei 5V betrieben werden – da jedoch fast alle Compact Flash Medien nur für 3.3 V spezifiziert sind, empfiehlt es sich bei 3.3 V Spannungsversorgung zu bleiben. Ein gewöhnlicher 3.3V Festspannungsregler hat sich bewährt, um das Modul in einer 5V – Umgebung zu betreiben.

Der Stromverbrauch schwankt je nach Applikation. Bei wireless LAN Karten liegt dieser bei ca. 300mA, mit eingeschaltetem Power Save Mode bei 80 mA. Bei Memory Modulen liegt er bei ca. 100mA.

IO-PINS

Die digitalen Eingänge des Avisaro Moduls sind 5V tolerant. D.h. ein Eingang des Avisaro Moduls kann mit 5V angesprochen werden, auch wenn das Modul sonst mit 3.3V arbeitet. So ist eine Integration in vorhandene 5V Systeme einfach.

Die digitalen Ausgänge haben eine Belastbarkeit von 50mA.

Lassen Sie alle nicht benötigten Leitung offen. Pull-Up oder Pull-Down Widerstände sind nicht notwendig.

SERIELLE SCHNITTSTELLE - RS232

Die RS232 Schnittstelle ist ausreichend dokumentiert, sodass hier nicht das Timing wiederholt wird. Das Avisaro Modul unterstützt Baudraten von 1.200 bis 115.200 Baud in den üblichen Abstufungen. Es werden verschiedene Parity unterstützt. Es werden nur 8 Datenbits unterstützt. Zur Flusskontrolle wird „keine“, „Software“ und „Hardware“ unterstützt.

Entsprechende Signale liegen wie folgt am Modul an:

Pin	MicroMatch	Stiftleiste	I/O	Beschreibung
RXD0	OST 6	OST 4	I	RS232 – RX
TXD0	OST 7	OST 5	O	RS232 – TX
SCL/CTS	OST 8	OST 8	O	CTS Flusskontrolle ’1’ = Stop Sende ’0’ = Ok zum Senden
SDA/RTS	OST 9	OST 9	I	RTS Flusskontrolle ’1’ = Stop Sende ’0’ = Ok zum Senden

Da „Rx“ und „Tx“ von der Betrachtung abhängen, sei die Datenrichtung klargestellt: RX – Das Avisaro Modul empfängt Daten. TX – Das Avisaro Modul sendet Daten. CTS – Das Avisaro Modul signalisiert Empfangsbereitschaft. RTS – Das Avisaro Modul bekommt Sendeerlaubnis.

SERIELLE SCHNITTSTELLE – I2C (SLAVE)

Die I²C-Bus Implementation im Avisaro-Modul entspricht einem I²C-Bus Slave d.h. das Avisaro-Modul kann nicht von sich aus kommunizieren und muß deshalb von einem Master-Device ständig gepollt werden. Es wird das I²C-Bus Modul des Controllers verwendet. Die Taktfrequenz ist Standard-I²C (100 kHz).

Die Default Adresse auf dem I2C Bus ist die ‚73'.

Pin	MicroMatch	Stiftleiste	I/O	Beschreibung
SCL/CTS	OST 8	OST 8	I	SCL Leitung (I2C Clock)
SDA/RTS	OST 9	OST 9	I/O	SDA Leitung (I2C Daten – In und Out)

Der I2C Bus ist ausreichend z.B. im Internet dokumentiert, so dass hier nicht das Timing wiederholt werden muss.

EINSTELLUNGEN

Die einzige Einstellung, die der Benutzer ändern kann, ist die I2C-Bus Adresse des Moduls. I2C-Bus Adressen gehen von 0...127 wobei '0' die Broadcast-Adresse (General Call Address) ist. Das Avisaro-Modul kennt keine 'Reservierten Adressen' (siehe I2C-Bus Specification, Seite 16), sondern reagiert nur auf seine eigene (Slave) Adresse.

Ein jungfräuliches Modul hat die die Adresse '73'. Wenn Sie ein Flash-Update mit einem Modul vor dem 1. April 2005 gemacht haben, dann steht die Adresse auf ‚128'.

Um die Adresse zu ändern, muß der Benutzer das AT-Kommando:

AT+I2C ADDRESS <xxx>

eingeben, wobei <xxx> ein beliebiger Wert zwischen 0 und 127 sein darf. Zum Anzeigen der Adresse gibt es das Kommando:

AT+STATUS I2C

Um das 'Huhn und Ei' Problem zu lösen (Sie müssen die I2C Adresse ändern, um überhaupt Befehle an das Modul schicken zu können), können Sie eine Compact Flash Speicherkarte benutzen, um das Avisaro Modul zu konfigurieren. Sie dazu das Benutzerhandbuch. (Kurz: erzeugen Sie eine Datei avi_config.txt mit den at-Befehlen – diese wird nach dem Einschalten wie eine Skriptdatei ausgeführt).

EINFACHER TEST

Das Avisaro-Modul speichert kurz nach dem Einschalten die Ausgabe der AT-Maschine OK (einschliesslich CR/LF, also 4 Bytes) in seinem Ausgangspuffer. Wenn der I2C Busmaster sich diese Daten abholen kann, funktioniert die I2C-Kommunikation.

DATENAUSTAUSCH MIT DEM AVISARO-MODUL ÜBER I²C

Das Avisaro-Modul hat jeweils einen Eingangs- und Ausgangspuffer (FIFO). Der Eingangspuffer wird von einem externen I2C -Gerät gefüllt und vom Avisaro-Modul sehr schnell verarbeitet. Der Ausgangspuffer enthält Daten vom Avisaro-Modul, die sich ein I2C Busmaster abholen kann. Sollte der Busmaster längere Zeit keine Daten aus dem Modul auslesen (im Falle einer offenen TCP-Verbindung und wenn die Gegenstelle etwas sendet), blockiert das Modul. Der blockierende Zustand kann durch Senden der Escape-Sequenz (+<pause>+<pause>+) aufgehoben werden. Die über TCP empfangenen Daten gehen dabei allerdings verloren. Besser ist es also, man holt die Daten ab.

DATEN ZUM AVISARO-MODUL SENDEN

Daten werden als "Stream" zum Avisaro-Modul gesendet. Der Ablauf aus Sicht des Busmasters ist folgender:

1. Setze die I2C START Bedingung.
2. Sende die I2C Adresse des Avisaro-Moduls um eine Stelle nach links geshiftet als Byte (8 Bits), Bit 0 muß gelöscht sein (I2C Adressen belegen die Bits 1...7, Bit 0 ist das Read/Write Bit).
3. Warte auf ACK vom Avisaro-Modul (Bit 9).
4. Sende ein Byte der Daten (8 Bits).
5. Warte auf ACK vom Avisaro-Modul (Bit 9).
6. Gehe zu Schritt 4 bis alle Daten verschickt wurden.
7. Setze die I2C STOP Bedingung.

DATEN VOM AVISARO-MODUL EMPFANGEN

Anders als beim Versenden der Daten läuft der Empfang ab. Die ersten beiden Bytes, die der Busmaster beim Empfang liest, geben an wieviele Daten im Ausgangspuffer des Avisaro-Moduls bereit stehen. Das erste Byte ist dabei das MSB (höherwertige Teil) und das LSB kommt direkt danach. Dieser 16-Bit Wert gibt an, wieviele Daten in Folge aus dem Modul gelesen werden können. Der Busmaster kann, muß aber nicht, die erhaltene Anzahl an Daten sofort auslesen. Wichtig dabei ist: Beim letzten gelesenen Byte darf der Busmaster kein ACK senden (siehe I2C-Bus Specification Seite 14). Sollte er es doch tun, dann geht ein Byte verloren. (Das Avisaro-Modul würde ein weiteres Byte senden, dass der Busmaster aber nicht abholen würde).

Der Ablauf beim Empfang ist folgender:

1) Setze die I2C START Bedingung.
2) Sende die I2C Adresse des Avisaro-Moduls um eine Stelle nach links geshiftet, bei gesetztem Bit 0, als Byte (8 Bits), Bit 0 muß gesetzt sein (I2C Adressen belegen die Bits 1...7, Bit 0 ist das Read/Write Bit).
3) Warte auf ACK vom Avisaro-Modul (Bit 9).

4) Lese das erste Byte aus dem Avisaro-Modul (8 Bits).
5) Setze ACK (Bit 9).
6) Lese das zweite Byte aus dem Avisaro-Modul (8 Bits).
7) Verknüpfe beide Bytes zu einem 16-Bit Wert, Anzahl = Byte1<<8 | Byte2
8) Ist dieser Wert 0 (keine Daten) oder 0xffff (Kein Kontakt zum Modul) gehe zu Schritt 15
9) Setze ACK (Bit 9).
10) Lese ein Byte der Daten aus dem Avisaro-Modul (8 Bits).
11) Setze ACK (Bit 9).
12) Bis Anzahl-1 oder ein Byte weniger als gewünscht: gehe zu Schritt 10.
13) Lese letztes Byte der Daten aus dem Avisaro-Modul (8 Bits),
14) Setze kein ACK (Bit 9).
15) Setze die I2C STOP Bedingung.

SOURCE-CODE BEISPIEL

Siehe Kapitel „I2C Schnittstelle“ auf Seite 23.

PARALLELE SCHNITTSTELLE

Der Parallel-Bus besteht aus 8 Datenleitungen und 4 Steuerleitungen und benutzt die ganze "WEST"-Seite des Avisaro-Moduls. Die Datenrichtung der Steuerleitungen ist festgelegt, aber die Datenrichtung der Datenleitungen ist änderbar um einen abwechselnden Datenaustausch zu erreichen (Bidirektional, Halbduplex). Der Bus arbeitet asynchron – eine Clock-Leitung gibt es daher nicht. Alle Operationen werden durch „Valid" und „Ack" angekündigt bzw. bestätigt. So kann der Bus auch mit sehr langsamen Clients arbeiten.

Pin	MicroMatch	Stiftleiste	I/O	Beschreibung
ATD0	WEST 1	WEST 1	I	Client Bus Request
ATD1	WEST 2	WEST 2	I	Client Data Valid / Data Acknowledge
DAC0	WEST 3	WEST 3	O	Master Bus Acknowledge
DAC1	WEST 4	WEST 4	O	Master Data Valid / Data Acknowledge
IO0	WEST 13	WEST 5	I/O	Bidirectional Data 0
IO1	WEST 14	WEST 6	I/O	Bidirectional Data 1
IO2	WEST 15	WEST 7	I/O	Bidirectional Data 2
IO3	WEST 16	WEST 8	I/O	Bidirectional Data 3
IO4	WEST 17	WEST 9	I/O	Bidirectional Data 4
IO5	WEST 18	WEST 10	I/O	Bidirectional Data 5
IO6	WEST 19	WEST 11	I/O	Bidirectional Data 6
IO7	WEST 20	WEST 12	I/O	Bidirectional Data 7

SOURCE CODE

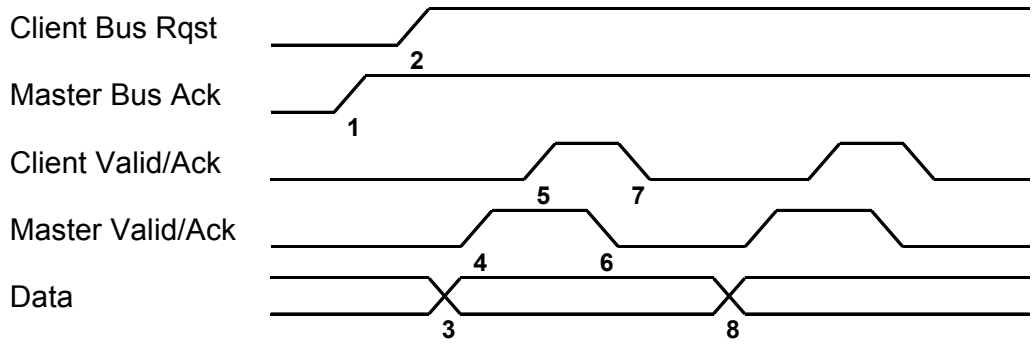
Siehe Kapitel „Parallele Schnittstelle" auf Seite 25.

BUSTIMING

Bei dem Bustiming gibt es zwei Mechanismen: Die Richtung des Datenbusses muss ausgehandelt werden und die Daten müssen angekündigt und bestätigt werden. Die folgenden Diagramme beschreiben die verschiedenen Szenarien. Kollisionen am Bus sind nicht möglich, wenn sich Client und Master richtig verhalten.

MASTER SENDET DATEN ZUM CLIENT

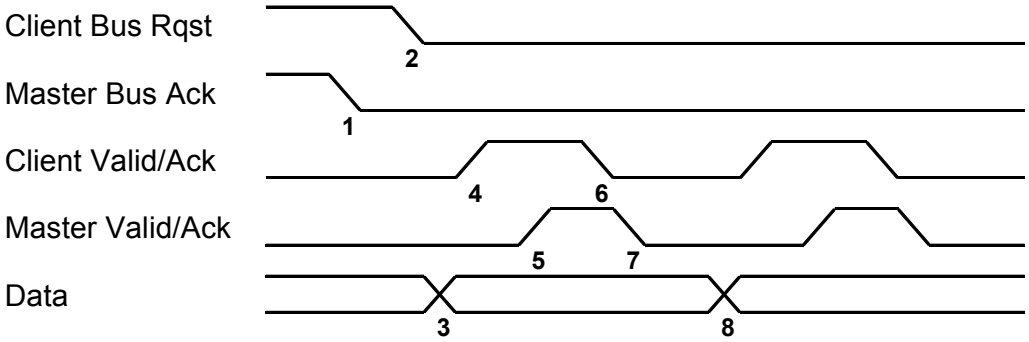
1) Der Master fordert den Bus an, indem die Bus Ack Leitung auf High gesetzt wird.
2) Der Client bestätigt die Busanforderung indem die Bus Rqst Leitung auf High gesetzt wird. Nur wenn sowohl Bus Ack als auch Bus Rqst Leitung auf High sind, kann der Master zum Client Daten senden. So lange der Client die Busumschaltung nicht bestätigt, kann er weiter Daten zum Master senden. Der Master ist also abhängig von der Cooperation des Clients – dies macht jedoch Sinn, da so der Client z.B. ein Datenpacket fertig übertragen kann.
3) Der Master legt Daten an den Bus
4) Der Master setzt Data Valid
5) Der Client bestätigt die Daten
6) Der Master setzt Data Valid zurück
7) Der Client setzt Data Valid zurück
8) Der Master legt neue Daten auf den Bus und der Vorgang beginnt neu. Der Bus bleibt die ganze Zeit im selben Zustand.



CLIENT SENDET DATEN ZUM MASTER

1) Der Master gibt den Bus frei, dass der Client senden kann, indem die Bus Leitung auf Null gesetzt wird. Der Client muss also so lange mit dem Senden warten.
2) Der Client bestätigt die Umschaltung.
3) Der Client legt Daten an den Bus an.
4) Der Client setzt Data Valid um die Gültigkeit der Daten zu signalisieren
5) Der Master quitiert mit Data Ack
6) Der Client setzt Data Valid wieder zurück
7) Der Master setzt Data Ack zurück

8) Der Client beginnt mit einer neuen Datenübertragung. Der Bus bleibt die ganze Zeit in die selbe Richtung geschaltet.



MASTER MÖCHTE DATEN SENDEN, ABER CLIENT HAT NOCH DATEN

1) Eine Datenübertragung von Client an Master ist im Gang (Bus Ack und Bus Request Leitung auf High). Master fordert den Bus an um Daten zum Client zu senden.

2) Der Client hat noch Daten und möchte diese an den Master senden. Der Client legt die Daten an den Bus.

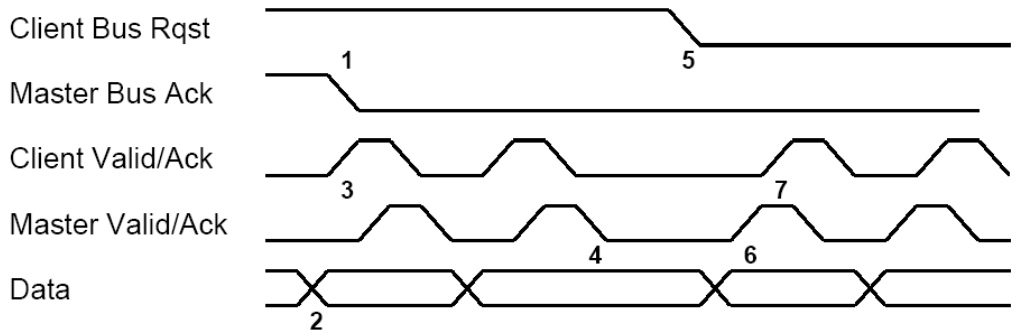
3) Der Client fährt mit der Datenübertragung fort und validiert die Daten per Data Valid. Der Master akzeptiert die Daten und quittiert.

4) Der Client ist mit der Datenübertragung fertig.

5) Der Client bestätigt die Busumschaltung

6) Der Master legt Daten an den Bus und setzt das Valid Signal

7) Der Client empfängt die Daten und bestätigt.



PIN BELEGUNG – ALLGEMEIN

Das Avisaro Modul hat zwei Stecksysteme – ein Micro Match Stecker und eine Lochleiste zur Aufnahme von z.B. Stiftleisten. Der MicroMatch bietet etwas mehr Funktionen aufgrund der höheren Polzahl.

Die Signale und Stromversorgungsleitung der beiden Stecker sind direkt miteinander verbunden. Haben Sie also z.B. auf der Stiftleiste 3,3 V angelegt, benötigen Sie dies nicht noch einmal auf dem MicroMatch Stecker zu tun.

Mit ‚West' und ‚Ost' sind die beiden Seiten bezeichnet. Auf den unten gezeigten Fotos ist die Bezeichnung herausgestellt.

Pin	MicroMatch	Stiftleiste	I/O	Beschreibung
ATD0	WEST 1	WEST 1	I	Analog zu Digital – Kanal 0
ATD1	WEST 2	WEST 2	I	Analog zu Digital – Kanal 1
DAC0	WEST 3	WEST 3	O	Digital zu Analog – Kanal 0
DAC1	WEST 4	WEST 4	O	Digital zu Analog – Kanal 1
NC2	WEST 5	-	-	Nicht belegt 2
RXD1	WEST 6	-	I	RS232 Kanal 1 (sekundär) - Empfang
TXD1	WEST 7	-	O	RS232 Kanal 1 (sekundär) – Senden
PWM0	WEST 8	OST 11	I/O	Puls Weiten Modulator 0 / GIO
PWM1	WEST 9	OST 12	I/O	Puls Weiten Modulator 1 / GIO
GND	WEST 10	OST 7	PWR	Versorgung: Masse
VCC	WEST 11	OST 6	PWR	Versorgung: 3,3 V
NC3	WEST 12	-	-	Nicht belegt 3
IO0	WEST 13	WEST 5	I/O	Benutzer Digital IO 0
IO1	WEST 14	WEST 6	I/O	Benutzer Digital IO 1
IO2	WEST 15	WEST 7	I/O	Benutzer Digital IO 2
IO3	WEST 16	WEST 8	I/O	Benutzer Digital IO 3
IO4	WEST 17	WEST 9	I/O	Benutzer Digital IO 4
IO5	WEST 18	WEST 10	I/O	Benutzer Digital IO 5
IO6	WEST 19	WEST 11	I/O	Benutzer Digital IO 6

IO7	WEST 20	WEST 12	I/O	Benutzer Digital IO 7
BKGD	OST 1		-	Debug / Programmier Modus
RESET	OST 2	OST 3	I	Reset (Microprozessor)
MODA	OST 3		-	Debug / Programmier Modus
MODB	OST 4		-	Debug / Programmier Modus
ECLK	OST 5		-	Debug / Programmier Modus
RXD0	OST 6	OST 4	I	RS232 Kanal 0 (primär) – Empfangen
TXD0	OST 7	OST 5	O	RS232 Kanal 0 (primär) – Senden
SCL/CTS	OST 8	OST 8	O	I2C Bus – SCL Leitung | CTS Flusskontr.
SDA/RTS	OST 9	OST 9	I	I2C Bus – SDA Leitung | RTS Flusskontr.
GND	OST 10	OST 7	PWR	Versorgung: Masse
VCC	OST 11	OST 6	PWR	Versorgung: 3,3 V
CS_EXT	OST 12		O	Chip Select – externe Komponente
NC0	OST 13		-	Nicht belegt 0
NC1	OST 14		-	Nicht belegt 1
GIO_0	OST 15	OST 10	I/O	Benutzer Digital IO – General Purpose
WE#	OST 16		O	Write Enable – externe Komponente
ADDR0	OST 17		O	Adressbus 0 – externe Komponente
ADDR1	OST 18		O	Adressbus 1 – externe Komponente
ADDR2	OST 19		O	Adressbus 2 – externe Komponente
ADDR3	OST 20		O	Adressbus 3 – externe Komponente

PIN BELEGUNG „CF MEMORY MODUL“

Zusätzlich zur Schnittstelle werden einige Statusinformation gegeben. Diese Statusinformationen sind nur bei den Modulen ‚RS232' und ‚I2C' gültig, da der parallele Port die selben Pins benutzt. Die Software "Memory Modul" belegt folgende Pins mit den beschriebenen Funktionen:

Pin	MicroMatch	Stiftleiste	I/O	Beschreibung
GND	WEST 10	OST 7	PWR	Versorgung: Masse
VCC	WEST 11	OST 6	PWR	Versorgung: 3,3 V
IO0	WEST 13	WEST 5	O	Anzeige: „Betrieb / Power on“ (grüne LED)
IO1	WEST 14	WEST 6	O	Anzeige: „Lese – Zugriff“ (grüne LED)
IO2	WEST 15	WEST 7	O	Anzeige: „Schreib – Zugriff“ (rote LED)
IO3	WEST 16	WEST 8	O	Anzeige: „Error“ (rote LED)
IO4	WEST 17	WEST 9	I	Taster: „Operation beenden“
GND	OST 10	OST 7	PWR	Versorgung: Masse
VCC	OST 11	OST 6	PWR	Versorgung: 3,3 V

Die Belegung für die jeweilige Schnittstelle – siehe oben in diesem Dokument.

Alle LEDs werden gegen GND geschaltet.

Lassen Sie alle nicht benötigten Leitung offen. Pull-Up oder Pull-Down Widerstände sind nicht notwendig.

PIN-BELEGUNG „WLAN RS232 / I2C“

Im WLAN RS232 Modul kann die Verwendung der I/O Leitungen geändert werden. Zur Auswahl steht: ‚Box', ‚Embedded' und ‚I/O'. So stehen verschiedene Statusinformation zur Verfügung. Die parallele Schnittstelle verwendet die selben IO Leitungen – daher stehen in diesem Fall keine Statusinformationen zur Verfügung. Die Software im Modul WLAN – RS232 benutzt folgende Pinbelegung zusätzlich zur Datenschnittstelle

Pin	MicroMatch	Stiftleiste	I/O	Beschreibung
GND	WEST 10	OST 7	PWR	Versorgung – Masse
VCC	WEST 11	OST 6	PWR	Versorgung: 3,3 V
GND	OST 10	OST 7	PWR	Versorgung - Masse
VCC	OST 11	OST 6	PWR	Versorgung: 3,3 V
RESET	OST 2	OST 3	I	Reset des Mikroprozessors. Aktiv low. (Interner Pullup zu high)
IO0	WEST 13	WEST 5	O	Box: Verbindung zum Access Point ok Embedded: Verbindung zum Access Point ok (high: Verbindung ok, low: nicht ok) I/O: Benutzerdefiniert
IO1	WEST 14	WEST 6	O	Box: Datenverbindung (TCP) besteht Embedded: Datenverbindung (TCP) besteht (high: Verbindung besteht, low: ~nicht) I/O: Benutzerdefiniert
IO2	WEST 15	WEST 7		Box: Daten Empfang und Gesendet Embedded: Daten Empfang und Gesendet I/O: Benutzerdefiniert
IO3	WEST 16	WEST 8		Box: Error Embedded: Error (high: Fehlerfall liegt vor, low: alles ok) I/O: Benutzerdefiniert
IO4	WEST 17	WEST 9	I	Box & Embedded: Verbindung aufbauen / abbauen. Übergang von high zu low: 1) Wenn keine Verbindung besteht (siehe IO1) wird diese aufgebaut gemäß Einstellungen „Verbindung aufbauen zu IP/Port“. 2) Wenn eine Verbindung besteht (siehe IO1) wird diese abgebaut. (Interner Pullup zu high) I/O: Benutzerdefiniert
IO5	WEST 18	WEST 10		Box & Embedded: keine Bedeutung I/O: Benutzerdefiniert
IO6	WEST 19	WEST 11		Box & Embedded: keine Bedeutung I/O: Benutzerdefiniert

				I/O: Benutzerdefiniert
IO7	WEST 20	WEST 12	I	Box: keine Bedeutung Embedded: Reset zu Default-Werten ‚Overwrite' [1] (Interner Pullup zu high) I/O: Benutzerdefiniert

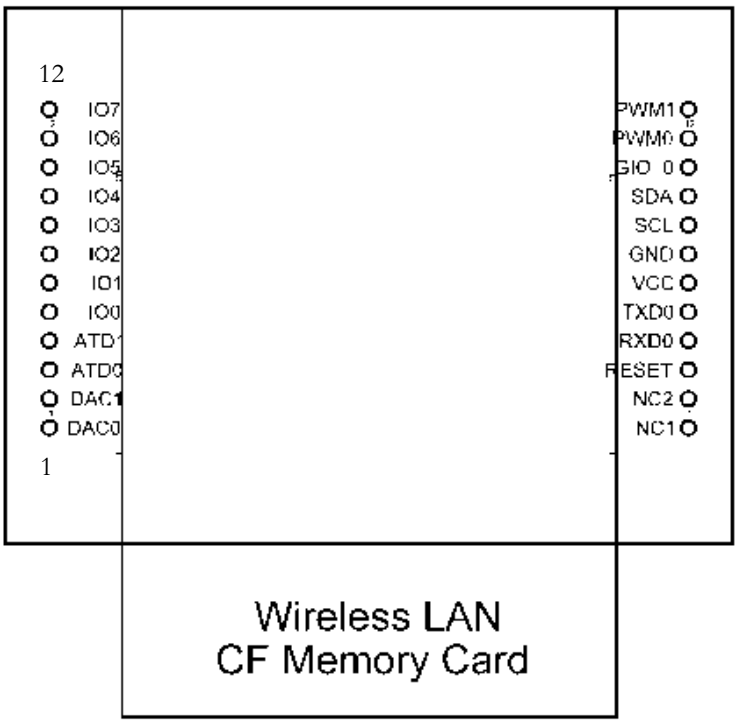
1) Normalerweise werden Default-Werte benutzt nachdem das Modul ohne eingelegte Karte eingeschaltet wird. Mit diesem Pin wird diese Eigenschaft abgeschaltet. Dazu wird der PIN schon beim Einschalten des Moduls auf Masse gezogen (z.B. über 2,2kOhm). Nun wird das Modul nicht in den Default Modus geschaltet, auch wenn keine Karte eingelegt ist. Um das Modul in den Default-Zustand zurück zu setzten, wird der PIN bei eingeschaltetem Modul kurz (min. 0,5 sec auf VCC) getoggelt. Beim nächsten Neustart des Moduls werden nun Default-Werte verwendet.
Wird der Pin nicht beschaltet, bleibt das Feature aktiv.

PIN LAYOUT STIFTLEISTE

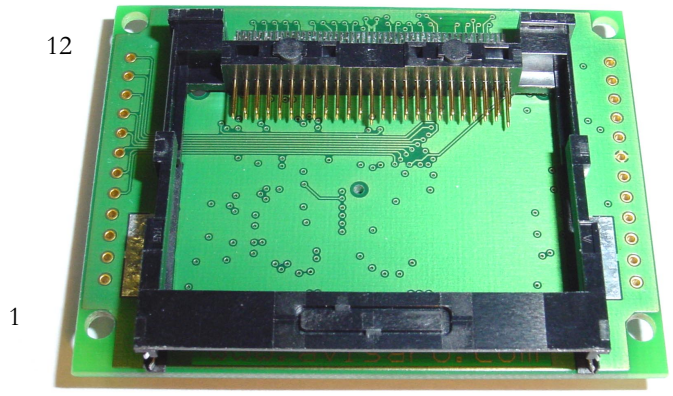
Das Modul verfügt über Stiftleisten im 2.54mm Rastermaß. So können Sie ohne Spezialstecker das Modul in Ihre Anwendung integrieren – oder einfach auch Drähte anlöten, um schnell einen Testaufbau zu realisieren.

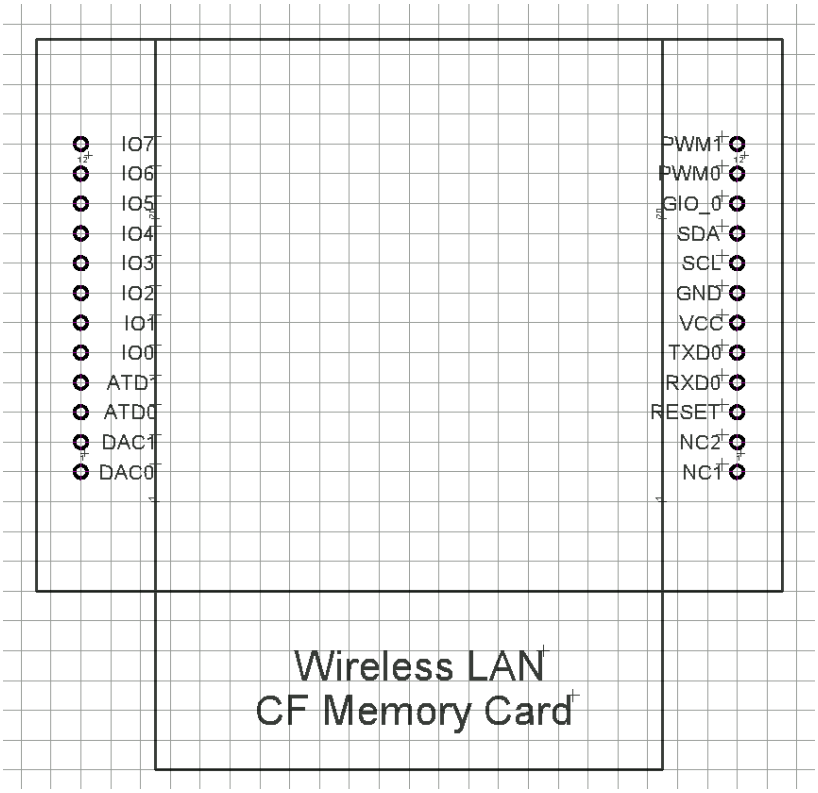
Die Zeichnungen zeigen das Modul in der Aufsicht. Das Foto dient zu Ihrer Orientierung – es ist in der gleichen Perspektive wie die Zeichnungen aufgenommen. In der dritten Zeichnung ist das Modul mit einem 2.54 Rastermaster abgebildet – dies hilft Ihnen ggf. beim Ausrichten eines Steckers auf einer Trägerplatine.

„WEST"-Seite



„OST"-Seite

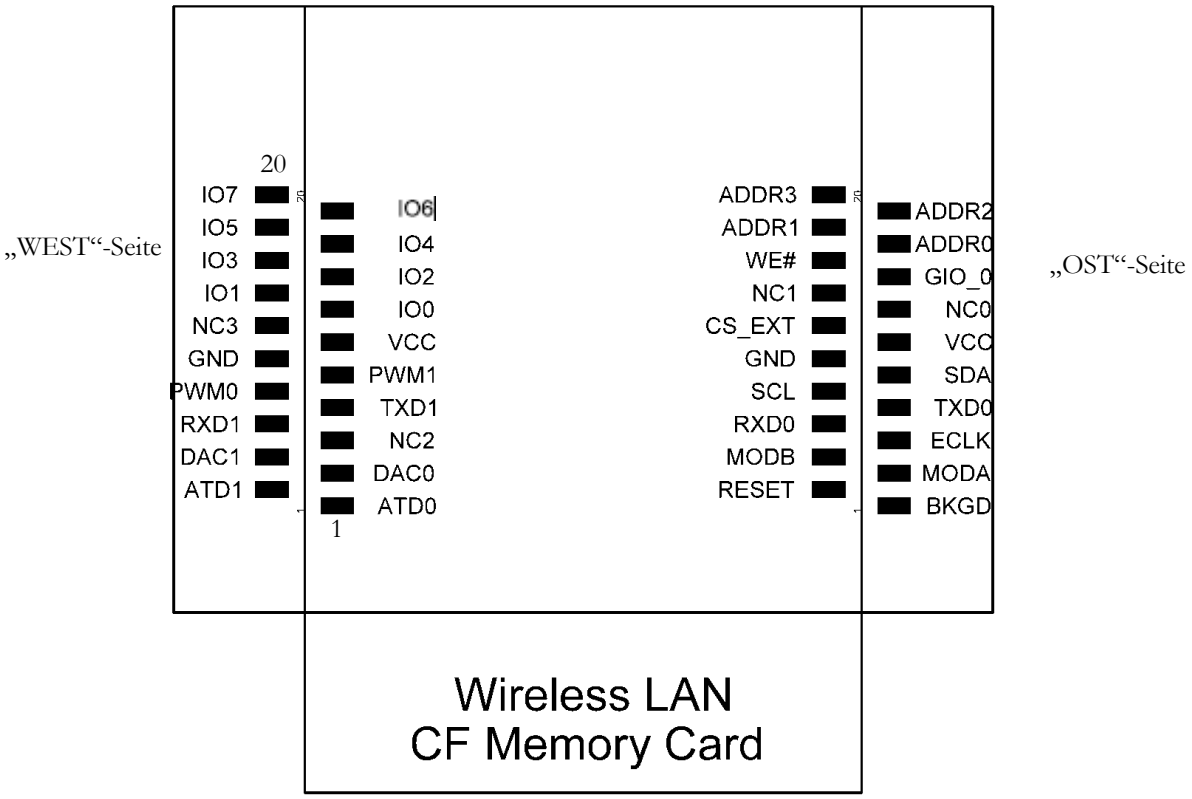


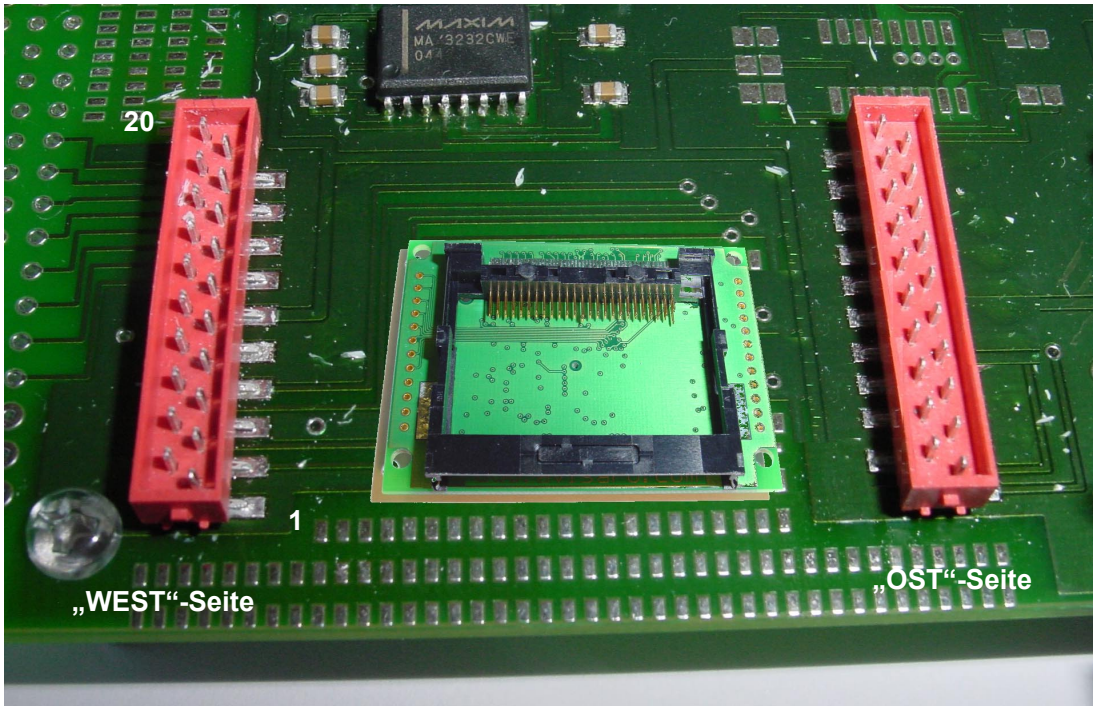

IO7
IO6
IO5
IO4
IO3
IO2
IO1
IO0
ATD
ATDC
DAC1
DAC0
PWM1
PWM0
GIO_0
SDA
SCL
GND
VCC
TXD0
RXD0
RESET
NC2
NC1
Wireless LAN
CF Memory Card

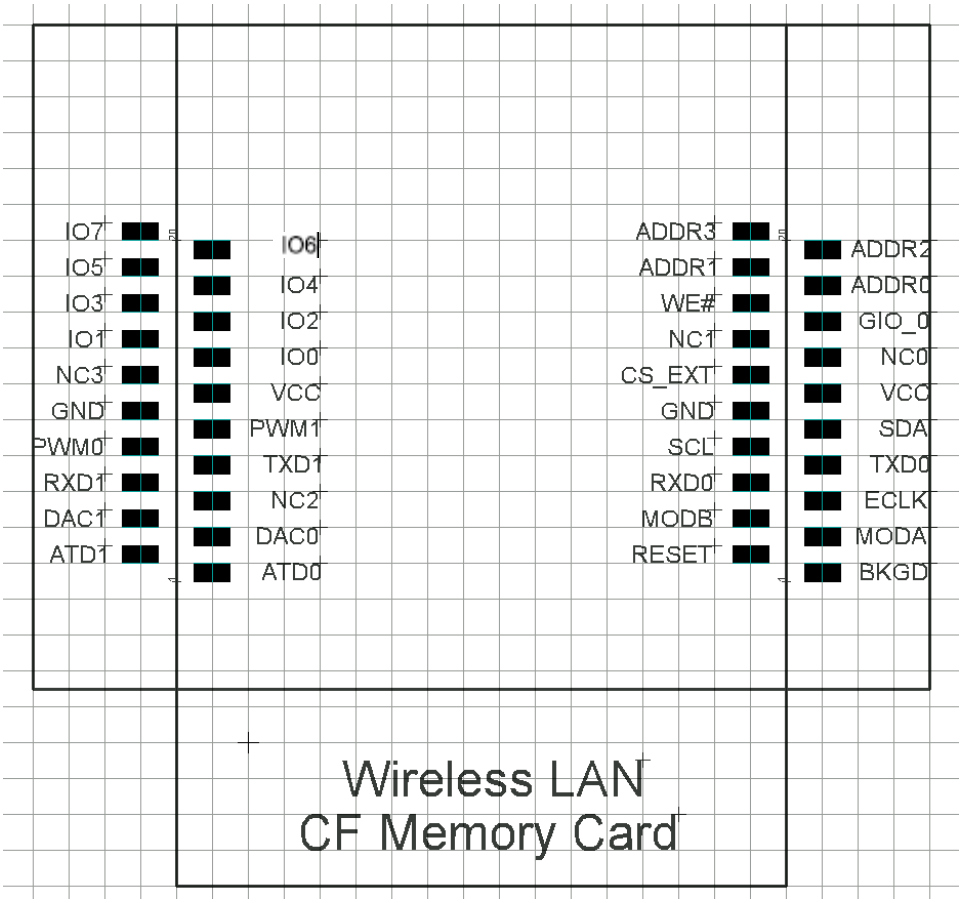
PIN LAYOUT MICRO MATCH STECKER

Das Modul verfügt über ein MicroMatch Stecksystem. Dieses Stecksystem ermöglicht Ihnen auf einer Trägerplatine SMD bestückbare Sockelsteckter zu verwenden, oder auch Flachbandkabel mit aufgequetschtem Stecker zu verwenden – für fast alle Anwendungen finden Sie den richtigen Stecker aus dem MicroMatch System.

Die Zeichnungen zeigen das Modul in der Aufsicht. Das Foto einer Trägerplatine (mit kleinem Modul als Fotomontage) dient zu Ihrer Orientierung – es ist in der gleichen Perspektive wie die Zeichnungen aufgenommen. In der dritten Zeichnung ist das Modul mit einem 2.54 Rastermaster abgebildet – dies hilft Ihnen ggf. beim Ausrichten eines Steckers auf einer Trägerplatine.



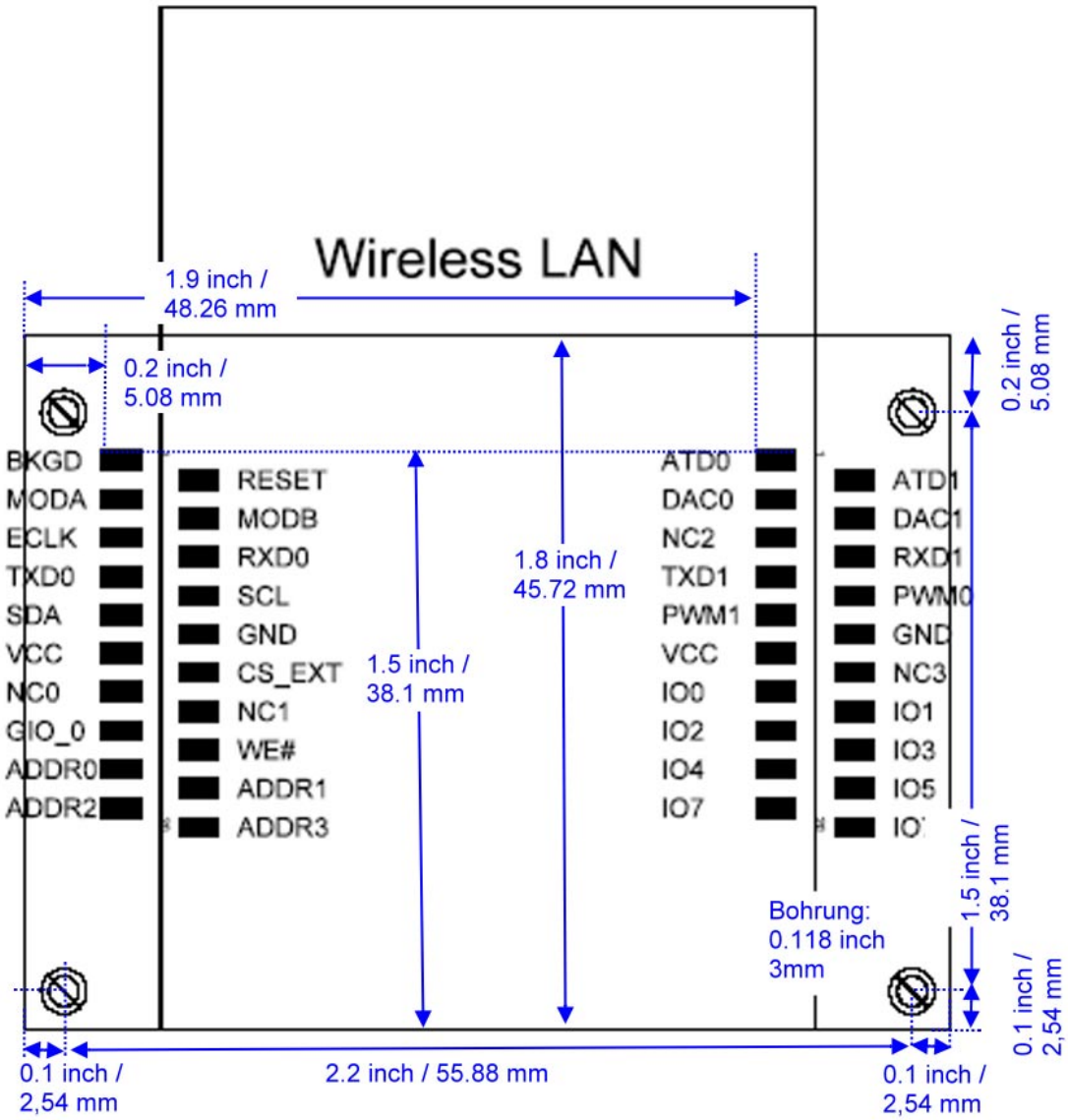

20
1
„WEST“-Seite
„OST“-Seite


IO7
IO5
IO3
IO1
NC3
GND
PWM0
RXD1
DAC1
ATD1
IO6
IO4
IO2
IO0
VCC
PWM1
TXD1
NC2
DAC0
ATD0
ADDR3
ADDR1
WE#
NC1
CS_EXT
GND
SCL
RXD0
MODB
RESET
ADDR2
ADDR0
GIO_0
NC0
VCC
SDA
TXD0
ECLK
MODA
BKGD
Wireless LAN
CF Memory Card

ABMESSUNGEN

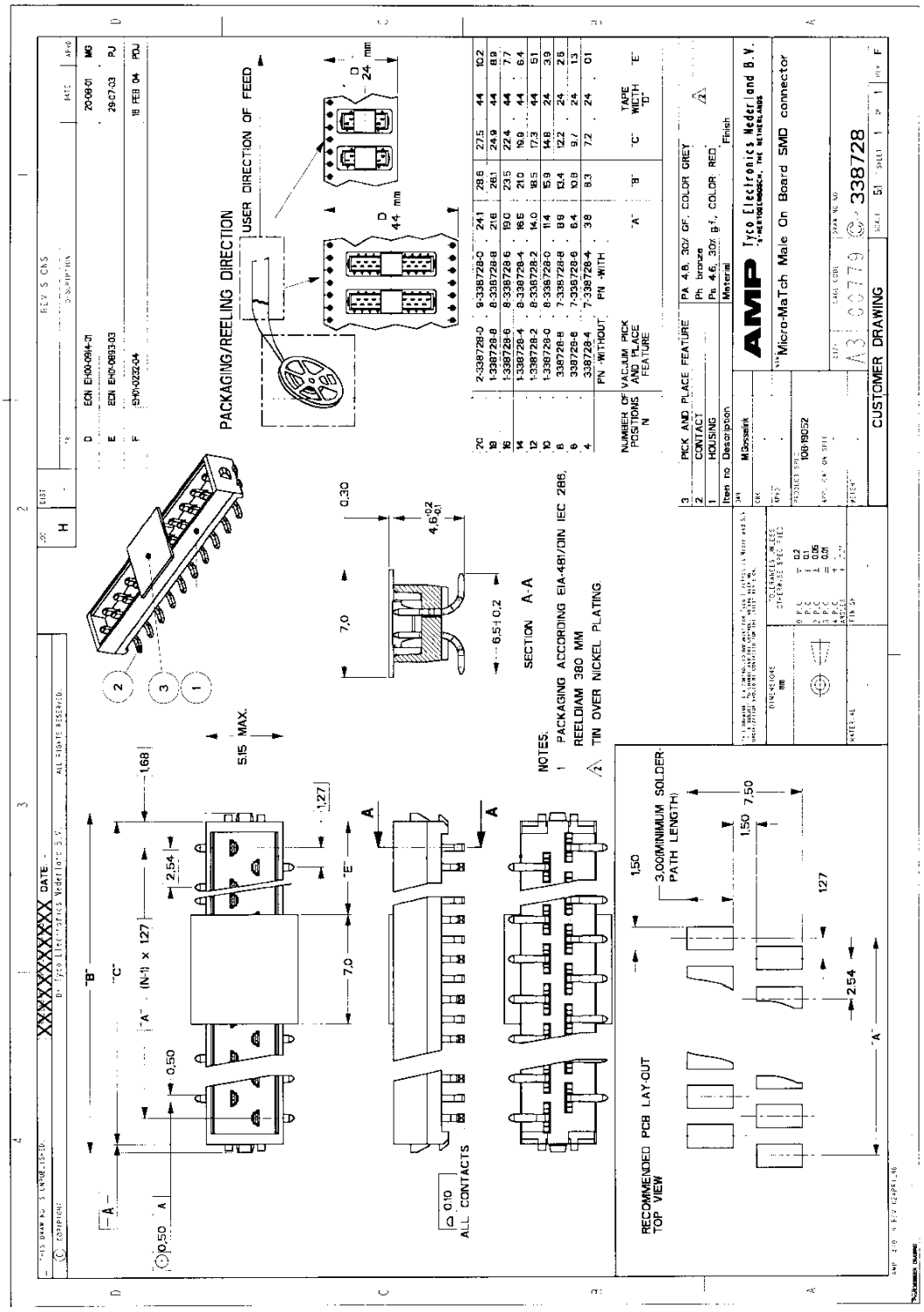
Die Abmessungen für den Steckplatz für das Avisaro Modul sind in der folgenden Darstellung dargestellt.

Die MikroMatch Stecker haben eine ausreichende Haltekraft, um das Modul zu halten. Die zusätzlichen Löcher für eine Verschraubung des Moduls sind nur notwendig, wenn Vibrationen (z.B. bei Verwendung in Baufahrzeugen) auftauchen.



MICROMATCH CONNECTOR (SMD)

Für Ihre Trägerplatine verwenden Sie den MicroMatch 9-338728-0 von AMP-Tyco.



PROGRAMMIERBEISPIELE

I2C SCHNITTSTELLE

Um den I2C Busmaster zu progrmmieren, kann folgender Code ggf. weiterhelfen:

```
// Diese Funktion sendet Daten an das Avisaro-Modul
// -----------------------------------------------
// Input:
// device_address  ---> I2C-Adresse des Avisaro-Moduls
// *data           ---> Zeigt auf den Anfang der zu versendenden Daten
//           size           ---> Anzahl der Daten die zu versenden sind
void I2C_Send (UINT8 device_address, UINT8 *data, int size)
{
   // Setze START Bedingung
   I2C_start();
   // Sende die Geräteadresse des Moduls
   I2C_out_byte (device_address<<1);
   // Warte Antwort ab
   I2C_wait_ack();
   // Für alle Daten...
   while (size--)
   {
     // Sende ein Byte
     out_byte (*data);
     data++;
     // Warte Antwort ab
     I2C_wait_ack();
   }
   // Fertig, setze STOP Bedingung
   I2C_stop();
}



// Diese Funktion empfängt Daten vom Avisaro-Modul
// -----------------------------------------------
// Der Benutzer gibt einen Puffer und die Grösse des Puffers an, in den
// die Daten kopiert werden sollen.
// Die Funktion überträgt empfangene Daten bis zur Größe des Puffers
// Input:
//    device_address ---> Geräteadresse des Avisaro-Moduls
//    *dest          ---> Adresse des Empfangspuffers
//    how_much       ---> Maximale Anzahl (Grösse des Empfangspuffers)
// Return:
//    Der Rückgabewert ist die Anzahl tatsächlich übertragener Daten
UINT16 I2C_Receive (UINT8 device_address, UINT8 *dest, UINT16 how_much)
{
   UINT16 size;
   UINT16 cnt = 0;

   // Sende START Bedingung
   I2C_start();
   // Sende Geräteadresse mit gesetztem Bit 0 (READ)
   I2C_out_byte (device_address<<1 | 1);
```

```
    // Warte Bestätigung ab
    I2C_wait_ack();
    // Hole Anzahl Bytes im Puffer des Moduls (High-Byte)
    size = I2C_in_byte();
    // Sende Bestätigung
    I2C_send_ack();
    // Hole Anzahl Bytes im Puffer des Moduls (Low-Byte) und
    // berechne Gesamtgrösse
    size <<= 8;
    size |= I2C_in_byte();
    // Fehler oder keine Daten im Puffer des Moduls?
    if (size == 0 || size == 0xffff)
       // Ein Takt weiter (ohne Bestätigung)
       I2C_nop();
    // Es sind Daten da !!!
    else
    {
       // Sende ACK für das 2 Byte der Länge
       I2C_send_ack();
       // Bis entweder alle Daten gelesen wurden oder der Puffer voll ist
       while (size && how_much)
       {
          // Hole ein Byte und übertrage es in den Puffer
          *dest = I2C_in_byte();
          // Ist es das letzte Byte?
          if (size == 1 || how_much == 1)
             // Sende kein ACK
             I2C_nop();
          // Für alle anderen Bytes...
          else
             // Sende ACK
             I2C_send_ack();
          // Alle Zähler und Pointer weiterzählen
          dest++;
          cnt++;
          size--;
          how_much--;
       }
    }
    // Setze I2C STOP Bedingung
    I2C_stop();
    // Rückgabe: Anzahl in den Puffer übertragener Zeichen
    return cnt;
}
```

PARALLELE SCHNITTSTELLE

Der folgende Code ist ein Beispiel für eine Client Implementierung der Parallelen Schnittstelle

```
// Diese Funktion sendet ein Byte an das Avisaro-Modul
// ---------------------------------------------------
// Input:
//          b ---> Das zu versendende Byte
// Return:
//          TRUE, wenn es geklappt hat, sonst FALSE
BOOL tester_tx (UINT8 b)
{
   unsigned long timeout;

   // Ist der Bus frei?
   if (DATA_ACK_NOT_SET)
   {
      // Daten auf den Bus legen
      WRITE_DATA_LINES(b);
      // Dem Avisaro-Modul signalisieren dass Daten da sind
      DATA_VALID_ON;
      // Auf Bestätigung warten
      timeout = PPR_GET_COUNTER_VALUE() + 2000;
      while (DATA_ACK_NOT_SET)
      {
         if (timeout < PPR_GET_COUNTER_VALUE())
         {
            // Timeout, Fehler
            DATA_VALID_OFF;
            return FALSE;
         }
      }
      // OK, Data-Valid wieder zurücknehmen
      DATA_VALID_OFF;
      // Darauf warten, dass Avisaro-Modul das ACK zurücknimmt
      timeout = PPR_GET_COUNTER_VALUE() + 2000;
      while (DATA_ACK_SET)
      {
         if (timeout < PPR_GET_COUNTER_VALUE())
            // Timeout, Fehler
            return FALSE;
      }
      // OK, Byte versendet
      return TRUE;
   }
   // Bus nicht frei, Fehler
   return FALSE;
}
```

```
// Diese Funktion empfängt ein Byte vom Avisaro-Modul
// --------------------------------------------------
// Input:
```

```
//      *b ---> Zeigt auf eine Adresse, in der das empfangene Byte
gespeichert wird
// Return:
//      TRUE wenn alles geklappt hat.
//      FALSE bei Fehler oder wenn keine Daten da waren.
BOOL tester_rx (UINT8 *b)
{
   unsigned long timeout;

   // Sind Daten da?
   if (DATA_ACK_SET)
   {
      // Ja, Datenwort lesen und speichern
      *b = READ_DATA_LINES;
      // Bestätigen
      DATA_VALID_ON;
      // Warten, bis das Avisaro-Modul das ACK zurücknimmt
      timeout = PPR_GET_COUNTER_VALUE() + 2000;
      while (DATA_ACK_SET)
      {
         if (timeout < PPR_GET_COUNTER_VALUE())
         {
            // Timeout, Fehler
            DATA_VALID_OFF;
            return FALSE;
         }
      }
      // Bestätigung wieder zurücknehmen
      DATA_VALID_OFF;
      // Alles OK
      return TRUE;
   }
   // Keine Daten vorhanden
   return FALSE;
}
```

```
// Globale Variable speichert die Busrichtung
// 0 == Undefiniert
// 1 == Client empfängt
// 2 == Client darf senden
int direction = 0;
```

```
// Steuert die Client-seitige Busumschaltung
void tester_tick (void)
{
    // Master möchte senden, Client noch nicht auf Empfang
    if (BUS_ACK_IS_TX && direction != 1)
    {
       // Neue Richtung merken
       direction = 1;
       // Client Datenrichtung auf Empfang umschalten
       BUS_DIRECTION_RX;
       // Beim Master bestätigen
       BUS_REQUEST_TX;
    }
```

```
    // Master möchte empfangen, Client noch nicht auf Sendung
    else if (BUS_ACK_IS_RX && direction != 2)
    {
       // Neue Richtung merken
       tdir = 2;
       // Client Datenrichtung auf Empfang umschalten
       BUS_DIRECTION_TX;
       // Beim Master bestätigen
       BUS_REQUEST_RX;
    }
}
```

```
// Hauptprogramm
// Simuliert einen Parallel Bus Client auf einem Microcontroller
int main (void)
{
    // I/O-Ports initialisieren
    tester_init();

    // Endlosschleife
    while (1)
    {
    // Auf Busumschaltung reagieren
       tester_tick();

       if (direction == 0)
       {
           // Undefiniert, Startbedingung
       }

       // Der Client darf Empfangen
       if (direction == 1)
       {
           UINT8 b;
           if (TRUE == tester_rx(&b))
           {
                // Empfangene Daten verarbeiten
                // .....
           }
       }

       // Der Client darf Senden
       if (direction == 1)
       {
           UINT8 b;
           // Wenn Daten zum Versenden da sind dann jetzt senden
           if (get_data_for_tx(&b))
               tester_tx (b);
       }
    }
}
```

TCP/IP (WIN-SOCKET) PROGRAMMIERUNG

```
/*
    Simple client example using WINSOCK
    ------------------------------------
    Link with ws2_32.lib (MSVC) or libws2_32.a (GCC/MinGW).

    This program connects to a web server and displays all received bytes
    (including HTTP headers) in the console window.

    You may also visit these sites:
    *   http://msdn.microsoft.com/library/default.asp?url=/library/en-
us/winsock/winsock/getting_started_with_winsock.asp
    *   http://www.hal-pc.org/~johnnie2/winsock.html
    *   http://tangentsoft.net/wskfaq/
    *   http://www.vijaymukhi.com/
    *   http://cs.ecs.baylor.edu/~donahoo/practical/CSockets/winsock.html
    *   http://www.sockets.com/

    Have fun,
       -> peter@avisaro.com
*/

// This is the webserver address
// www.yahoo.akadns.net (yahoo.com)
#define IPADDRESS "216.109.117.204"
// This is the port number (HTTP)
#define PORT 80

// For 'printf' etc.
#include <stdio.h>
```

```
// Also include socket definitions
#include <windows.h>

// Macro which prints an error message
// and exits the program
#define ERR(_x_){printf("Could not %s\n",_x_);Sleep(INFINITE);exit(1);}

// This is the simplest HTTP GET request
#define HTTP_REQUEST "GET / HTTP/1.0\r\n\r\n"

// Let's go
int main()
{
    // Structure gets filled by WSAStartup
    WSADATA wsa;
    // Our socket handle
    SOCKET sock;
    // Structure holding the complete network address
    SOCKADDR_IN sockaddr;
    // Temporary return value from socket functions
    int result;

    // Initialize the socket library
    if (0 != WSAStartup (0x0202, &wsa))
       ERR ("initialize socket library");

    // Get a socket instance
    sock = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock == INVALID_SOCKET)
       ERR ("create socket instance");

    // Connect to the remote service
```

```
sockaddr.sin_family = AF_INET;
sockaddr.sin_port = htons (PORT);
sockaddr.sin_addr.s_addr = inet_addr (IPADDRESS);
result = connect (sock, (SOCKADDR*)&sockaddr, sizeof (SOCKADDR_IN));
if (result == SOCKET_ERROR)
   ERR ("connect");
printf ("Connected to %s\n", IPADDRESS);

// Now send an HTTP request
result = send (sock, HTTP_REQUEST, sizeof(HTTP_REQUEST)-1, 0);
if (result == SOCKET_ERROR)
    ERR ("send HTTP request");

// Display all received data
while (1)
{
    // Receive buffer
    char buff[256];
    // Temporary pointer for displaying received bytes
    char *temp;

    // Try to get some data
    // The return value is the number of received bytes.
    result = recv (sock, buff, 256, 0);
    // Leave the loop if either the connection was closed
    // or there was an error. The web server closes the
    // connection when all data was send.
    if (result == 0 || result == SOCKET_ERROR)
        break;
    // Something has been received - show it now.
    temp = buff;
    while (result--)
```

```
            printf ("%c", *temp++);
        }
        printf ("\nDisconnected\n");

        // Finally close the socket
        closesocket (sock);

      // Ready
      return 0;
   }
```

BENUTZERHANDBUCH

Avisaro Compact Flash Box

"WISSEN WAS LÄUFT, SENDEN WAS LAUFEN SOLL“

Version 1.3

Änderungshistorie

2004-10-01		Erstellung
2004-10-25		At-Kommandos
2004-11-04		Fehlermeldungen, Flash-Update, Quick-Start
2004-11-18		Erweiterungen der Befehle und Geschwindigkeit Serielle Schnittstelle
2004-12-13		Erweiterungen des Befehlssatzes. Insbesondere der Paketmodus ist hinzugekommen. Automatischer Lernmodus, ob bei der Eingabe <cr><lf> verwendet wird. Statusabfrage über AT+STATUS. Flusskontrolle (Hardware und Software)

Kontakt:

Avisaro AG
Vahrenwalderstr. 7 (tch)
30165 Hannover

Telefon:
+49-511-9357411
Telefax:
+49-0721-151 254826

eMail:
support@avisaro.com

INHALTSVERZEICHNIS

EINLEITUNG

Mit der Avisaro Compact Flash Box haben Sie die Möglichkeit große Datenmengen auf handelsüblichen Speicherkarten zu speichern und ohne Umwege am PC zu lesen. Die Avisaro CF Box schließen Sie über eine übliche serielle Schnittstelle (RS232) an Ihre Anwendung an.

Um mit den Speichermedien zu arbeiten senden Sie Befehle über die serielle Schnittstelle. Zum „ausprobieren" können Sie auch die CF Box an einen PC anschließen und dann an einem Terminal Programm die Befehl von Hand eingeben.

Der Ablauf um Daten zu schreiben oder zu lesen gliedert sich in drei Schritte:

1) Datei zum Schreiben oder Lesen vorbereiten (at+create, at+append, at+open)

2) Daten übermitteln (at+datastream, at+writepacket, at+readpacket)

3) Vorgang abschließen (at+close, Stop-Sequenz)

Darüber hinaus stehen noch Befehle zur Verwaltung des Datenträgers (Speichergröße, freier Speicherplatz) und zur Verwaltung des Moduls zur Verfügung (Uhrzeit).

SICHERHEITSMAßNAHMEN

Beachten Sie alle üblichen Sicherheitsmaßnahmen bei dem Betrieb von elektrischen Geräten.

Öffnen Sie niemals das Produkt. Führen Sie niemals Reparaturen oder Modifikationen selbst durch – überlassen Sie dies ausschließlich der Avisaro AG.

HAFTUNGSAUSSCHLUSS

Die Avisaro AG haftet nicht für Schäden die durch den Einsatz der Avisaro Compact Flash Box entstehen. Insbesondere für den Ausfall des Produkts an sich und für Folgeschäden wie z.B. Betriebsunterbrechnungen oder Schäden an angeschlossenen Anlagen.

Ausgeschlossen ist auch die Haftung für die Vollständigkeit der Daten beim Schreiben und Lesen auf Speichermedien.

BETRIEB

INBETRIEBNAHME

Schließen Sie die Avisaro Compact Flash Box über ein serielles Kabel an Ihre Anwendung an. Zur Stromversorgung benutzen Sie ein geeignetes Steckernetzteil, dass 5V bis 6V Spannung und mindestens 200mA Strom liefert. Die Avisaro CF Box ist sofort einsatzbereit und immer ‚eingeschaltet' so lange sie mit Strom versorgt wird.

Die serielle Schnittstelle wird im Auslieferungszustand mit 9600 Baud betrieben.

SPEICHERKARTEN EINSTECKEN UND ENTNEHMEN

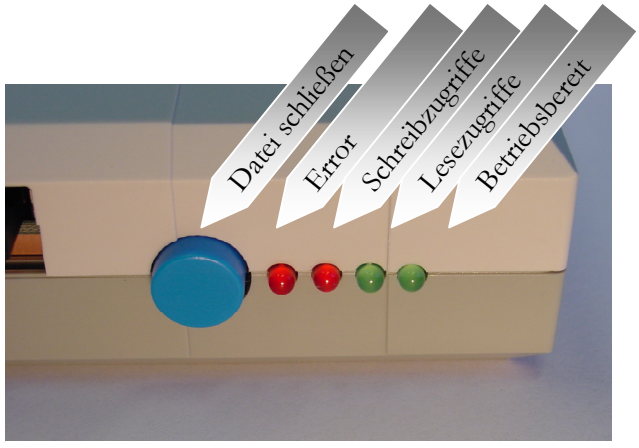
Speicherkarten können im eingeschaltetem Zustand eingesteckt werden. Die Avisaro Compact Flash Box erkennt automatisch eine neue eingesteckte Karte.

Die Karte kann im eingeschaltetem Zustand entnommen werden. **Dabei müssen Sie darauf achten, dass keine Schreiboperation stattfindet.** Sie erkennen aktive Schreiboperationen an der Leuchtdiode am Gehäuse.

Je nach Anwendung empfiehlt es sich mit dem „at+cardpresent" Befehl zu überprüfen, ob eine Karte eingesteckt ist bevor mit Schreib- und Lesebefehlen begonnen wird.

LAUFENDER BETRIEB

Im Betrieb werden über die serielle Schnittstelle Befehle zum lesen und schreiben von Dateien gegeben. Leuchtdioden am Gehäuse zeigen Schreib- und Leseoperationen an.



ANZEIGE: BETRIEBSBEREIT

Mit der Leuchtanzeige „Betriebsbereit“ wir angezeigt, dass das Avisaro Modul mit Strom versorgt und betriebsbereit ist. Wird das Avisaro Modul mit Strom versorgt, wird ein Selbsttest durchgeführt – deshalb leuchtet die Anzeige etwas verzögert auf.

ANZEIGE: LESEZUGRIFFE

Mit der grünen Leuchte werden Lesezugriffe angezeigt. Wird eine Datei zum Lesen geöffnet, leuchtet die Anzeige dauerhaft. Sie erlischt kurz bei jedem Lesezugriff. Wird die Datei geschlossen, erlischt die Anzeige dauerhaft.

Auch wenn Lesezugriffe die Integrität des Dateisystems nicht gefährden, sollte die Speicherkarte nicht entnommen werden wenn diese Anzeige leuchtet.

ANZEIGE: SCHREIBZUGRIFFE

Mit der roten Leuchtanzeige werden Schreibzugriffe auf die Speicherkarte angezeigt. Wird eine Datei zum Schreiben geöffnet, leuchtet die Anzeige dauerhaft. Sie erlischt kurz bei jedem Schreibzugriff. Beim Schreiben werden auch Daten gelesen – deshalb flackert die Anzeige für Lesezugriffe ebenfalls. Wird die Datei geschlossen, erlischt die Anzeige dauerhaft.

Schreibzugriffe beeinflussen die Integrität des Dateisystems. So lange die rote Anzeige leuchtet, sollte die Speicherkarte auf keinen Fall entnommen werden. Mit der blauen Taste kann die Datei manuell geschlossen werden.

ANZEIGE: FEHLER / ERROR

Ist ein Fehler aufgetreten, leuchtet diese Anzeige dauerhaft. Fehler können z.B. falsche AT-Befehle oder ein voller Datenträger sein. Die Anzeige leuchtet so lange bis mit einem ‚at-readerror’ der Fehler quittiert worden ist.

BEDIENELEMENT: DATEI SCHLIEßEN

Mit dem blauen Taster wird die laufende Operation abgeschlossen und eventuell noch offene Dateien geschlossen. Vor der Entnahme der Speicherkarte sollte weder die Anzeige für Schreib- oder Lesezugriffe leuchten.

BESONDERHEITEN

Folgende Besonderheiten sollten Sie beim Betrieb der Avisaro Compact Flash Box beachten:

- Die Avisaro Compact Flash Box hat keine eigene Uhr. Sie können jedoch eine Uhrzeit und ein Datum mit AT-Befehlen einstellen. Diese wird dann z.B. beim Erzeugen von Dateien verwendet.

- Verwenden Sie für Ihre Anwendung nie den Dateinamen „aviup001.s19“. Ein solche Datei wird verwendet, um neue Software in das Modul zu laden („Flash-Update“).

- Verwenden Sie für Ihre Anwendung nicht den Dateinamen „outdata.log“ und „indata.log“. Diese Dateien werden verwendet, wenn der automatische Datalogging und Datareplay Modus verwendet wird.

- Das Avisaro Modul ‚lernt' ob eine Zeile mit <cr><lf> oder nur mit <lf> abgeschlossen wird. Dabei wird angenommen, dass die Eingabe des ersten Befehls dem verwendeten Schema entspricht. Die Information des Zeilenabschlusses ist für das Modul wichtig – insbesondere bei der Verwendung des Paketmodus.

DIE SERIELLE SCHNITTSTELLE

Die serielle Schnittstelle lässt sich im Bereich von 1.200 bis 115.200 Baud konfigurieren.

Hinweis auf möglichen Datenverlust:

Die Schreibgeschwindigkeit zur Compact Flash hängt von verschiedenen Faktoren ab. Zum einen gibt es Qualitätsunterschiede zwischen Herstellern von Speichermedien. Wesentlicher ist jedoch die Menge der auf der Speicherkarte vorhandenen Dateien und wie stark diese „Fragmentiert" sind. Eine frisch formatierte Speicherkarte ist schneller als eine mit vielen Dateien.

Bei hohen Geschwindigkeit (z.B. 115.200 Baud) besteht daher die theoretische Gefahr, dass es zu Datenverlust kommt. Wenn die Daten auf der seriellen Schnittstelle schneller gesendet werden, als sie zur Karte geschrieben werden, gehen Bytes verloren. Das Avisaro CF Modul hat einen sehr großen Eingangsspeicher, um dies möglichst zu vermeiden. Sollte es dennoch vorkommen, so wird dieser Fehler erkannt und durch die rote LED am Modul angezeigt. Mit dem Befehl at+readerror kann überprüft werden, ob es sich um einen Speicherüberlauf (gleichbedeutend einem Datenverlust) handelt.

Maßnahmen, um einen Datenverlust zu vermeiden:

- Benutzen Sie leere und frisch formatierte Speicherkarten.
- Machen Sie kurze Pausen beim Senden. Diese entstehen häufig bereits automatisch, da die meisten Applikationen eine gewissen Rechenzeit haben, bevor Daten geschrieben werden.
- Benutzen Sie niedrige Baudraten.
- Benutzen Sie Software oder Hardware Flusskontrolle. Dies bietet 100% Schutz vor Bufferüberlauf.
- Benutzen Sie den Paketmodus. Dies bietet 100% Schutz vor Bufferüberlauf.

KOMMANDOMODUS, DATENMODUS UND PACKETMODUS

Es werden drei Modi unterschieden:

1) Im Kommandomodus werden Befehle eingegeben. D.h. in diesel Modus sind die 'AT'-Befehl gültig und somit können alle Dateioperationen ausgeführt werden

2) Im Datenmodus werden die Daten übertragen. Der Datenmodus wird durch den Befehl at+datastream eingeschaltet und mit der Stop-Sequenz (drei Plus-Zeichen im zeitlichem Abstand) wieder ausgeschaltet.

3) Im Packetmodus werden ebenfalls Daten übertragen. Durch den Befehl at+writepacket wird für eine wohldefinierte Anzahl von Bytes in den Datenmodus geschaltet und danach automatisch wieder zurück in den Befehlsmodus.

Die Unterscheidung der drei Modi ist sinnvoll, da Daten nicht als Befehle interpretiert werden dürfen.

KOMMANDOMODUS

- Im Kommandomodus können Befehle beginnend mit AT+ eingegeben werden.
- Ein Kommando endet entweder mit einem CR, einem LF oder zwei Zeichen, die entweder CR oder LF sein können. Das erste Kommando nach dem Einschalten oder Neustart hat eine Verzögerung von drei Sekunden, während der auf ein zweites Endekennzeichen gewartet wird (entweder CR oder LF). Dadurch merkt sich das CF-Modul, mit welchen Zeichen der Benutzer seine Kommandos abschliesst.
- Kommandos können gross- oder klein geschrieben werden.
- Kommandos mit einer Länge über 64 Zeichen (nach dem AT+) werden mit einem Fehler abgebrochen.
- Nach der Ausführung eines Kommandos wird ein OK<CR+LF> (Erfolgreich) oder ERROR<CR+LF> zurückgesendet.
- Nach der Ausführung eines Kommandos befindet sich das CF-Modul wieder im Kommandomodus.

DATENMODUS

- In den Datenmodus wird AT+DATASTREAM gewechselt.
- In den Datenmodus kann nur gewechselt werden, wenn eine Datei geöffnet ist (entweder zum Lesen oder Schreiben).
- Das CF-Modul sendet OK<CR+LF> zurück.
- Ist eine Datei zum Schreiben geöffnet, dann können jetzt Rohdaten zum CF-Modul gesendet werden.
- Ist eine Datei zum Lesen geöffnet, dann wird augenblicklich der Inhalt der Datei zum Benutzer-Terminal gesendet.
- Beim Schreiben in eine Datei bricht der Datenmodus ab, wenn ein Filesystem-Fehler aufgetreten ist. In diesem Fall wird ERROR<CR+LF> ausgegeben und die Datei geschlossen.
- Beim Lesen aus einer Datei bricht der Datenmodus ab, wenn die Datei komplett übertragen wurde oder wenn ein Filesystem-Fehler aufgetreten ist. Im Fehlerfall wird ERROR<CR+LF> ausgegeben und die Datei geschlossen.

- Beim Schreiben in eine Datei kann der Datenmodus nur durch die Eingabe von +++ (mit mindestens 0.5 Sekunden Abstand) abgebrochen werden (wenn kein Fehler aufgetreten ist).
- Das CF-Modul befindet sich nach dem Abbruch des Datenmodus' wieder im Kommandomodus.

PAKETMODUS

- In den Paketmodus wird mit AT+WRITEPACKET gewechselt.
- In den Paketmodus kann nur gewechselt werden, wenn eine Datei zum Schreiben geöffnet ist.
- Bei aktivem Software Flow Control (XON/XOFF) wird Paketmodus abgelehnt.
- AT+WRITEPACKET gibt OK<CR+LF> zurück wenn in den Paketmodus gewechselt wurde.
- Nach dem Wechsel in den Paketmodus muß das Paket gesendet werden. Das Paket ist wie folgt aufgebaut: 2 Bytes Längenangabe, 1...1536 Bytes Daten, 2 Bytes CRC-Wert.
- Alle 2-Byte Werte sind 'Little Endian' (Low Byte zuerst)
- Die Längenangabe bezieht sich auf die Daten d.h. Länge und CRC werden nicht mitgezählt.
- Der CRC-Wert ist optional. Ein CRC-Wert von 0 bedeutet dass kein CRC-Wert angegeben wurde. In diesem Fall wird das Paket ohne CRC-Check akzeptiert.
- Nach dem CRC-Check wird OK<CR+LF> oder ERROR<CR+LF> zurückgesendet. Genauere Angaben bekommt man im Fehlerfall mit AT+READERROR.
- Nachdem das Paket gesendet wurde befindet sich das CF-Modul wieder im Kommandomodus d.h. für ein weiteres Paket muss erneut AT+WRITEPACKET und ein Paket gesendet werden.

DATENSTROM UND PAKETMODUS

Es stehen zwei Möglichkeiten zur Verfügung Daten zur Karte zu schreiben bzw. Daten von der Karte zu lesen. Beim „Datastream“ werden die Daten kontinuierlich zur Avisaro Box geschrieben, während sie beim „Paketmodus“ in Blöcken übertragen werden.

Vorteil des Paketmodus ist insbesondere die garantierte Datenübertragung. Es kann kein Datenverlust durch einen Bufferüberlauf entstehen, ebenso kann kein Datenverlust durch Fehler auf z.B. dem seriellen Kabel entstehen.

Welcher Modus einfacher zu Verwenden ist, hängt von der Anwendung ab. Wenn immer eine bekannte Datenmenge (z.B. ein Datagramm) übertragen wird, dann bietet sich der Packetmodus an,

da keine Stop-Sequenz (drei Plus-Zeichen mit Pause) generiert werden muss. Werden kontinuierlich Daten unterschiedlicher Menge geschrieben werden müssen, bietet sich der Datenstrom Modus an.

	at+writepacket	**at+datastream**
Vorteil	• Datensicherung durch CRC • Datenverlust ausgeschlossen	• Einfache Handhabung bei variablen Datenmengen
Nachteil	• Die Datenmenge muss vor dem Schreiben bekannt sein • CRC Datensicherung erfordert Rechenaufwand	• Eine Stopsequenz muss erzeugt werden, um wieder in den Befehlsmodus zurück zu wechseln • Keine besondere Datensicherung

TIPPS UND TRICKS (FAQ)

Bei Anregungen und Problemen wenden Sie sich bitte jederzeit an uns. Am besten erreichen Sie uns über E-Mail: support@avisaro.com . Wünschen Sie einen Rückruf, geben Sie bitte Ihre Telefonnummer und eine geeignete Rückrufzeit an.

Vielleicht helfen auch folgende Anregungen:

ANTWORTZEITEN

Die Speichergrößen von Compact Flash Speicherkarten sind sehr groß geworden. Wenn Sie viele Daten gespeichert haben, werden die Antwortzeiten z.B. auf ein „at-delfile“ Befehl mitunter recht lang. Wenn Sie z.B. eine 60 Megabyte große Datei löschen, dauert dies einige Minuten.

BESCHÄDIGTES DATEISYSTEM REPARIEREN

Wenn Sie ein Speichermedium aus der Avisaro CF Box entnehmen während ein Schreib- oder Lesevorgang in Betrieb ist, kann das Dateisystem beschädigt werden. Dies kann recht leicht wieder ‚repariert' werden:

Stecken Sie die Speicherkarte in Ihren Windows PC. Die Speicherkarte finden Sie unter einem Laufwerksbuchstaben im „Arbeitsplatz“. Mit rechte Maustaste , „Eigenschaften“ und dann die Lasche „Extras“ bekommen Sie die Möglichkeit zur „Fehlerüberprüfung“. Klicken Sie auf „Jetzt prüfen“ um den Datenträger zu überprüfen und zu reparieren.

.

BEISPIELE

Hier finden Sie häufig verwendete Dateioperationen als Beispiele erläutert.

Zur Erklärung:

- <cr><lf> bedeutet „Carriage Return“ und „Line Feed“ – wird normalerweise übertragen, wenn Sie die ‚Enter’ Taste an Ihrem Terminal drücken. Wenn Sie eine programmierbare Steuerung haben, müssen Sie die Zeichen generieren. In der Programmiersprache „C“ sähe dies so aus: printf(„at+create logfile.txt \r\n“). Wenn Sie die Zeichen einzeln übertragen, wäre dies die Zahl 13 (oder hex 0x0d) für <cr> und die Zahl 10 (oder hex 0x0a) für <lf>.
- Die Avisaro Modul sind ‚tolerant’. Wenn sie anstelle von <cr><lf> nur eins von beiden senden, funktioniert dies auch.

DATEI SCHREIBEN

Um eine neue Datei (z.B. „logfile.txt“) zu erzeugen und mit Daten zu füllen, gehen Sie wie folgt vor:

Die Datei wird zunächst erzeugt mit:

Befehl: **at+create logfile.txt** <cr><lf>

Antwort: **OK** <cr><lf>

Anschließend wird die Datenübertragung vorbereitet mit:

Befehl: **at+datastream** <cr><lf>

Antwort: **OK** <cr><lf>

Sie können nun Daten senden (binär oder ASCII):

The quick brown fox jumps over lazy dog. <cr><lf>

Die Avisaro Compact Flash Box ist klasse. <cr><lf>

Testende.

Um die Dateiübertragung zu beenden, senden Sie die Stop-Sequenz (drei ‚+’ Zeichen mit jeweils mehr als 0,5 Sekunden Pause dazwischen):

Befehl: **+ + +**

Antwort: **OK** <cr><lf>

Die Datei wurde automatisch geschlossen und die Operation ist beendet.

DATEI LESEN

Um eine Datei (z.B. „logfile.txt“) zu lesen, gehen Sie wie folgt vor:

Die Datei wird zunächst erzeugt mit:

Befehl: **at+open logfile.txt** <cr><lf>

Antwort: **OK** <cr><lf>

Anschließend wird die Datenübertragung vorbereitet mit:

Befehl: **at+datastream** <cr><lf>

Antwort: **OK** <cr><lf>

Der Dateiinhalt wird nun automatisch an Ihre Anwendung übertragen:

The quick brown fox jumps over lazy dog. <cr><lf>

Die Avisaro Compact Flash Box ist klasse. <cr><lf>

Testende.

Sie erkennen das Datenende daran, dass keine Daten innerhalb von 0,5 Sekunden mehr kommen. Um den Vorgang abzuschließen senden Sie die Stop-Sequenz:

Befehl: **+ + +**

Antwort: **OK** <cr><lf>

Die Datei wurde automatisch geschlossen und die Operation ist beendet.

WARTUNG (SOFTWAREUPDATE)

Sie können die Software der Avisaro Compact Flash Box recht einfach auf dem neusten Stand halten. So rüsten Sie neue Befehle nach oder korrigieren eventuelle Fehler in der Software.

Die neuste Software finden Sie im Internet unter:

http://www.avisaro.com/html/speicher_box.html

So funktioniert der Software-Update:

- Laden Sie sich das gewünschte File auf Ihren PC. Entpacken Sie die Datei. Die Updates Files heissen alle "aviup001.s19". Am besten bewahren Sie also das ZIP-File auf, weil dieses einen bedeutsameren Dateinamen hat.
- Speichern Sie die Datei ("aviup001.s19") auf eine Compact Flash Speicherkarte. **Verwenden Sie ausschließlich frisch formatierte und leere Speicherkarten.** Die Datei sollte im obersten Verzeichnis liegen.
- Schalten Sie das Avisaro Modul aus und ziehen Sie stecken die Speicherkarte mit dem Updatefile hinein.
- Schalten Sie das Modul wieder ein.
- **Warten Sie mindestens 2 Minuten**. In dieser Zeit lädt das Avisaro Modul das Updatefile. Schalten Sie in dieser Zeit das Modul nicht aus. Es gibt keine besondere Rückmeldung, wenn der Updatevorgang fertig ist - sie können sich aber voll und ganz auf die 2 Minuten verlassen (länger schadet nicht).
- Schalten Sie das Modul aus, entnehmen die Speicherkarte mit dem Updatefile - das war's. (Vergessen Sie nicht die „aviup001.s19" Datei auf der Speicherkarte zu löschen, sonst starten Sie den Updatevorgang immer wieder).

ÜBERSICHT AT-KOMMANDOS

Allgemeine Hinweise:

- Alle AT-Befehle werden mit „Carriage Return“ (<cr>) und „Line Feed“ (<lf>) abgeschlossen. Der Befehl zum erzeugen einer Datei lautet also: at+create test.txt <cr><lf>. In der nachfolgenden Aufstellung wird auf das <cr><lf> verzichtet. Siehe auch die ASCII Tabelle am Ende dieser Dokumentation.

- Alle Antworten (z.B. ok, error, 32000) werden ebenfalls mit <cr><lf> abgeschlossen. Antworten wie z.B. die Anzahl der freien Speicherplätze wird im ASCII – Format zurück gegeben (’3’,’2’,’0’,’0’,’0’).

- Groß und Kleinschreibung ist bei den Befehlen nicht relevant. Alle Antworten vom Modul werden immer in Großbuchstaben gegeben.

- Dateinamen haben das Format 8.3 (8 Zeichen Name und 3 Zeichen Kennung)

- Parameter werden mit „Leerzeichen“ von einander getrennt.

DATEIEN ZUM SCHREIBEN UND LESEN VORBEREITEN		
Befehl	**Antwort**	**Beschreibung**
Datei zum Lesen öffnen: at+open *dateiname* at+open *directory**dateiname*	OK, ERROR	Öffnet eine vorhandene Datei zum Lesen. Ist die Datei nicht vorhanden, wird ‚ERROR’ zurück gegeben. Um die Datenübertragung zu starten, wird der Befehl at+datastream verwendet. Um den Vorgang abzuschließen, z.B. um eine andere Datei zu lesen, wird der Befehl at+close verwendet. TIP: Der at+open Befehl kann auch verwendet werden, um zu testen ob eine Datei vorhanden ist: ‚OK’: Datei ist da, ‚ERROR’: Datei ist nicht da. Es muss nur nach einem ‚OK’ der at+close Befehl geschickt werden, wenn die Datei nicht weiter verwendet werden soll. **Voraussetzung:** Es darf keine Datei geöffnet sein. **Mögliche Rückgabewerte von at+readerror:** - FILE IS OPEN

		- I AM HAPPY - Filesystem Fehler (Nummer 0x40 bis 0x4d)
Neue Datei erzeugen: at+create *dateiname* at+create *directory**dateiname*	OK, ERROR	Erzeugt eine Datei auf dem Datenträger mit dem Namen „Dateiname“. Wird nur ein Dateiname angegeben, wird die Datei im Wurzelverzeichnis angelegt. Mit „\directory\dateiname“ wird eine Datei im angegebenen Verzeichnis angelegt. Dies muss vorhanden sein. Ist eine Datei mit dem selben Namen bereits vorhanden, so wird diese gelöscht. Nach dem Erzeugen ist die Datei geöffnet. Soll sie nicht weiter verwendet werden, muss sie mit „at+close“ geschlossen werden. **Voraussetzung:** Es darf keine Datei geöffnet sein. **Mögliche Rückgabewerte von at+readerror:** - FILE IS OPEN - I AM HAPPY - Filesystem Fehler (Nummer 0x40 bis 0x4d)
Daten an Datei anhängen: at+append *dateiname* at+append *directory**dateiname*	OK, ERROR	Es wird die vorhandene Datei “Dateiname” zum schreiben geöffnet. Alle Daten werden an das Ende der Datei angehängt. Um Daten zu senden, kann der Befehl at+datastram verwendet werden. **Voraussetzung:** Es darf keine Datei geöffnet sein. **Mögliche Rückgabewerte von at+readerror:** - FILE IS OPEN - I AM HAPPY - Filesystem Fehler (Nummer 0x40 bis 0x4d)

DATEN ÜBERTRAGEN		
Befehl	**Antwort**	**Beschreibung**
Datenstrom senden: at+datastream *daten*	OK, ERROR	Die Avisaro CF Box wird in den Daten-Modus versetzt. Im Datenmodus wird ein kontinuierlicher Datenstrom verarbeitet. Es muss zuvor eine Datei zum Schreiben geöffnet worden sein. Alle gesendeten Daten werden in die geöffnete Datei geschrieben. Um den Vorgang zu beenden wird die „Stop-Sequenz“ ('+','+','+') verwendet. Die Stop-Sequenz wird mit OK oder ERROR quittiert. Dabei wird die Datei geschlossen. Ein at+close ist nicht notwendig. Wenn ‚zwischendurch' ein Fehler auftritt (z.B. Datenträger voll) wird ein „ERROR“ gesendet. Daten können sowohl ASCII, wie auch binäres Format haben. **Voraussetzung:** Es muss eine Datei zum Schreiben oder Lesen geöffnet sein. **Mögliche Rückgabewerte von at+readerror:** - FILE NOT OPEN - I AM HAPPY
Datenstrom lesen: at+datastream	OK, ERROR Datenstrom	Die Avisaro CF Box wird in den Daten-Modus versetzt. Im Datenmodus wird ein kontinuierlicher Datenstrom verarbeitet. Es muss zuvor eine Datei zum Lesen geöffnet worden sein. Die Daten aus der Datei werden ausgegeben. Um den Datenstrom anzuhalten kann „Software Flusskontrolle (XON, XOFF)“ oder „Hardware Flusskontrolle (RTS, CTS)“ verwendet werden. Um den Vorgang abzubrechen, kann die „Stop-Sequenz“ geschrieben werden. Der Vorgang ist beendet, wenn alle Daten aus der Datei übermittelt worden sind. Dieses Ende kann erkannt werden, wenn – bei abgeschalteter Flusskontrolle – keine Daten innerhalb von 0.5 Sekunden mehr kommen. Die Datei wird automatisch geschlossen. Es ist also kein at+close Befehl notwendig, nachdem die Datei vollständig übertragen worden ist.

		Der at+datastream Befehl wird mit einem ‚OK' oder ‚ERROR' <cr><lf> bestätigt. Die Nutz-Daten aus der Datei beginnen demnach nach dem <cr><lf>. Daten können sowohl ASCII, wie auch binäres Format haben.
Daten als Paket schreiben: at+writepacket <cr><lf> *packet (length, data, crc)*	OK, ERROR	Anstelle eines kontinuierlichen Datenstroms werden Daten packetweise zum Schreiben übermittelt. Eine Datei muss vorher zum Schreiben geöffnet worden sein. Das Moduls antwortet auf at+writepacket mit OK oder ERROR. Dann kann das Paket übertragen werden. Das Format ist: • 2 Bytes: Datenlänge (nur Daten, ohne CRC). Das niederwertige Byte wird zuerst übertragen. • x Bytes: Daten (Anzahl gemäß Länge, Maximal: 1536 Bytes.) • 2 Bytes: CRC Prüfsumme. Die CRC Überprüfung kann ausgeschaltet werden, indem zwei Bytes mit einer 0 übertragen werden. Das Avisaro Modul antwortet mit OK oder ERROR. Bei ‚ERROR' (z.B. falsche Prüfsumme) wurde das Paket nicht auf das Speichermedium geschrieben. Die Übertragung kann mit der Stop Sequenz abgebrochen werden. Die maximale Paketlänge ist durch das Modul vorgegeben (meist 512 Bytes). **Voraussetzung:** Funktioniert nicht, wenn XON/XOFF aktiv ist. Eine Datei muss vorher geöffnet worden sein. **Mögliche Rückgabewerte von at+readerror:** - IMPROPER FLOW CONTROL METHOD - FILE NOT OPEN - I AM HAPPY

Daten paketweise lesen: at+readpacket demnächst verfügbar	packet	Anstelle eines kontinuierlichen Datenstroms werden Daten paketweise gelesen. Eine Datei muss vorher zum Lesen geöffnet worden sein. Das Modul antwortet mit Daten im Paketformat: 2 Bytes: Datenlänge (ohne CRC) x Bytes: Daten (Anzahl gemäß Länge) 2 Bytes: CRC Prüfsumme Mit einem wiederholten at+readpackt Befehl werden die nächsten Daten aus der geöffneten Datei angefordert. Die maximale Paketgröße lässt sich konfigurieren.
Daten wiederholen (lesen): at+repeatpacket demnächst verfügbar	packet	Das letzte Datenpaket wird wiederholt. Dies wird verwendet, um z.B. bei einer fehlerhaften CRC Prüfsumme die Daten erneut anzufordern. Das Modul antwortet mit Daten im obigen Paketformat:

OPERATIONEN ABSCHLIEßEN UND BEENDEN		
Befehl	**Antwort**	**Beschreibung**
Operation abschließen: at+close	OK, ERROR	Schließt eine geöffnete Datei. Dieser Vorgang ist wichtig, um die Integrität des Dateisystems zu gewährleisten. Der Befehl bezieht sich immer auf die aktuelle Datei, es werden also keine Parameter benötigt. **Voraussetzung:** Es muss eine Datei zum schreiben oder lesen geöffnet sein. **Mögliche Rückgabewerte von at+readerror:** - FILE NOT OPEN - I AM HAPPY - Filesystem Fehler (Nummer 0x40 bis 0x4d)
Stop-Sequenz: '+' .. Pause > 0.5 sec .. '+' .. Pause > 0.5 sec .. '+'	Je nach aktivem Befehl	Mit dieser Sequenz werden laufende Vorgänge beendet. Insbesondere wird so der Datenmodus beendet. Die Sequenz setzt sich zusammen aus: drei ‚+' Zeichen mit jeweils einer Pause von mindestens 0.5 sec (höchstens 5 sec). Wird ein anderes Zeichen zwischendurch gesendet, wird die

		Sequenz abgebrochen. Nach der Stop-Sequenz wird die Datei geschlossen. Sollen weitere Daten geschrieben/gelesen werden muss die Datei wieder geöffnet (at+append) werden. Beim Schreiben (at+datastream) werden die + Zeichen nicht mit in die Datei geschrieben.
XON = Hex-Wert 0x11	keine	Setzt den Datenstrom fort. Aktiv beim Auslesen einer Datei und nachdem der Datenstrom mit XOFF pausiert wurde.
XOFF = Hex-Wert 0x13	keine	Hält den Datenstrom an. Aktiv beim Auslesen einer Datei. Bei besetztem XOFF ist die Stop-Sequenz noch aktiv.

DATENTRÄGER UND DATEIINFORMATIONEN		
Befehl	**Antwort**	**Beschreibung**
Freier Speicherplatz: at+freespace	Anzahl der freien Bytes OK, ERROR	Ermittelt den freien Speicherplatz auf dem Datenträger. Als Antwort wird die Zahl der freien Bytes zurück gegeben. **Voraussetzung:** Es darf keine Datei geöffnet sein. **Mögliche Rückgabewerte von at+readerror:** - FILE IS OPEN - I AM HAPPY - Filesystem Fehler (Nummer 0x40 bis 0x4d)
Größe des Datenträgers: at+diskspace	Anzahl der Bytes auf dem Datenträger OK, ERROR	Ermittelt die Größe des Datenträgers. Ist keine Speicherkarte eingelegt, wird ‚error' zurück gegeben
Dateigröße: at+filesize	Dateigröße in Bytes OK, ERROR	Ermittelt die Dateigröße. Die Datei muss vorher mit dem at+open Befehl geöffnet worden sein. **Voraussetzung:** Es muss eine Datei geöffnet sein. **Mögliche Rückgabewerte von at+readerror:**

		- FILE NOT OPEN - I AM HAPPY
Dateidatum: at+filedate	Datum und Zeit der Datei im Format: jjjj/mm/tt hh:mm:ss	Ermittelt das Datum und die Zeit der Datei. Die Datei muss vorher mit dem at+open Befehl geöffnet worden sein. Die Angaben folgen im Abstand mit einer Leerzeichen („Space“). **Voraussetzung:** Es muss eine Datei geöffnet sein. **Mögliche Rückgabewerte von at+readerror:** - FILE NOT OPEN - I AM HAPPY

VERZEICHNISSE UND DATEIEN VERWALTEN		
Befehl	**Antwort**	**Beschreibung**
Verzeichnis erzeugen: at+createdir *directory*	OK, ERROR	Es wird ein Verzeichnis mit dem Namen “Directory” angelegt. Ist das Verzeichnis bereits vorhanden, so wird eine Fehlermeldung ausgegeben. **Voraussetzung:** Es darf keine Datei geöffnet sein. **Mögliche Rückgabewerte von at+readerror:** - FILE IS OPEN - I AM HAPPY - Filesystem Fehler (Nummer 0x40 bis 0x4d)
Inhaltsliste darstellen: at+dir \ at+dir *directory*	Dateiname, Größe, Datum <cr><lf> Dateiname, Größe, Datum <cr><lf> ok,error<cr><lf>	Zeigt den Inhalt des Wurzelverzeichnisses oder eines Unterverzeichnisses an. Es wird eine Liste mit dem Format Dateiname Größe Datum zurückgegeben. Die Liste wird mit einem ‚OK’ oder ‚ERROR’ <cr><lf> abgeschlossen. **Voraussetzung:** Es darf keine Datei geöffnet sein. **Mögliche Rückgabewerte von at+readerror:** - FILE IS OPEN

		- I AM HAPPY - Filesystem Fehler (Nummer 0x40 bis 0x4d)
Datei / Verzeichnis löschen: at+delete *dateiname* at+delete *directory*	OK, ERROR	Die Datei „dateiname“ oder das Verzeichnis „directory“ wird gelöscht. Ein zu löschendes Verzeichnis muss leer sein. **Voraussetzung:** Es darf keine Datei geöffnet sein. **Mögliche Rückgabewerte von at+readerror:** - FILE IS OPEN - I AM HAPPY - Filesystem Fehler (Nummer 0x40 bis 0x4d)
Datei verschieben: at+move d*ateiname dateiname*	OK, ERROR	

BESONDERE BEFEHLE		
Befehl	**Antwort**	**Beschreibung**
Karte vorhanden ? at+cardpresent	OK, ERROR	Es wird überprüft, ob eine Speicherkarte eingesteckt ist (OK) oder nicht (ERROR). **Voraussetzung:** Funktioniert unabhängig davon, ob eine Datei geöffnet ist oder nicht. **Mögliche Rückgabewerte von at+readerror:** - MEMORY CARD NOT PRESENT - I AM HAPPY
Status abfragen at+status	Liste von Statusinformtionen OK	Listet diverse interne Statusinformationen: - Copyrightmeldung, Rev.Nr. etc. - Eingabeerkennung zwei/ein Zeichen - Art der aktiven Flow Control Methode - Auslastung des Eingangspuffers - Ob Flow-Control wirksam war - Ob Bytes verschluckt wurden **Voraussetzung:** Funktioniert immer.

		Mögliche Rückgabewerte von at+readerror: - I AM HAPPY
Verfügbare Befehle / HELP at+commands	Liste der Befehle mit der Anzahl der erwarteten Parameter in Klammern. OK	Listet alle verfügbaren Befehle und die Anzahl der erwarteten Parameter auf. Die Ausgabe erfolgt ohne das führende „at+“. **Voraussetzung:** Funktioniert immer. **Mögliche Rückgabewerte von at+readerror:** - I AM HAPPY
Mögliche Fehlermeldungen at+errorlist	Liste der Fehlermeldungen OK	Listet alle verfügbaren Rückgabemeldungen des Befehls at+readerror. **Voraussetzung:** Funktioniert immer. **Mögliche Rückgabewerte von at+readerror:** - I AM HAPPY
Gesamt Speicherplatz at+diskspace	Speichergröße OK, ERROR	Ermittelt die Größe des Datenträgers. Ist keine Speicherkarte eingelegt, wird ‚error' zurück gegeben.
Fehlermeldungen lesen: at+readerror	Fehlernummer Fehlertext	Liest Detailinformationen zu einer Fehlermeldung aus. Es wird eine Fehlernummer und eine Fehlermeldung im Klartext ausgegeben. Abgeschlossen wird die Antwort mit <cr><lf>. Es wird kein OK nach dem Fehlercode ausgegeben. **Voraussetzung:** Funktioniert unabhängig davon, ob eine Datei geöffnet ist oder nicht.
Datum setzen: at+setdate *datum*	OK, ERROR	Setzt das Datum im Avisaro Modul. Dieser Befehl ist immer nach dem Einschalten notwendig, wenn das Avisaro Modul über keine batteriegepufferte Uhr verfügt. So lange das Modul eingeschaltet ist, wird Zeit und Datum weitergezählt. Das Datum wird verwendet, wenn Dateien erzeugt werden. Das Datum wird im Format: jjjj mm tt

		übertragen (Leerzeichen dazwischen) **Voraussetzung:** Eingaben müssen numerisch sein d.h. dürfen keine Buchstaben haben. Tag und Monat werden auf gültigen Bereich geprüft. **Mögliche Rückgabewerte von at+readerror:** - INVALID PARAMETER - PARAMETER OUT OF RANGE - I AM HAPPY
Zeit setzten: at+settime *time*	OK, ERROR	Setzt die Zeit im Avisaro Modul. Dieser Befehl ist immer nach dem Einschalten notwendig, wenn das Avisaro Modul über keine batteriegepufferte Uhr verfügt. Die Zeit wird verwendet, wenn Dateien erzeugt werden. Die Zeit wird im Format: hh mm ss übertragen. (Leerzeichen dazwischen). **Voraussetzung:** Eingaben müssen numerisch sein d.h. dürfen keine Buchstaben haben. Stunde, Minute und Sekunde werden auf gültigen Bereich geprüft. **Mögliche Rückgabewerte von at+readerror:** - INVALID PARAMETER - PARAMETER OUT OF RANGE - I AM HAPPY
Modul neu starten: at+restart		Das Modul wird neu gestartet. So werden z.B. die neu eingestellten Parameter für die serielle Schnittstelle übernommen. **Voraussetzung:** Funktioniert nur, wenn keine Datei geöffnet ist. Reset des Controllers über Watchdog-Timer. Ausgabe von OK nach dem Neustart. **Mögliche Rückgabewerte von at+readerror:** keine
Daten eines Sectors: at+dump *sector*		Listet die Daten eines Sectors aus. Diese Funktion dient zur Diagnostik im Fehlerfall. Detaillierte Kenntnisse von Dateisystemen sind

		erforderlich. **Voraussetzung:** Funktioniert nur, wenn keine Datei geöffnet ist. Gibt einen Sektor der FlashDisk aus (512 Bytes). **Mögliche Rückgabewerte von at+readerror:** - FILE IS OPEN - I AM HAPPY

KONFIGURATION		
Befehl	**Antwort**	**Beschreibung**
Packetlänge: at+paketlength demnächst verfügbar	OK, ERROR	Setzt die maximale Paketlänge beim Lesen einer Datei.
<u>RS232 Paramter:</u> at+linepar *speed parity stop fluss* **(Beachten Sie bei hohen Baudraten die Hinweise in Kapitel „Die serielle Schnittstelle“ auf Seite 7).**	OK, ERROR	Setzt die Parameter für die serielle Schnittstelle. Um diesen Befehl zu senden, muss mit den gegenwärtig eingestellten Parametern gearbeitet werden. Die Einstellung wird dauerhaft bis zum nächsten at+linepara gespeichert. Nach dem Ein-/Ausschalten bzw. nach „at+restart“ werden die neuen Parameter aktiv. *Speed:* Baudrate: 1200, 2400,9600, 19200,38400,57600,115200 *Parity:* Parity: n (none), e (even), o (odd) *Stop:* Anzahl Stop Bits: 1, 2 *Fluss:* Flusskontrolle: none, sw (software), hw (hardware) Defaulteinstellungen sind: 9600 n 1 sw <u>Z. Z. wird nur der „Speed“ Parameter und die Flusskontroller ausgewertet.</u> **Voraussetzung:** Es werden nur erlaubte Parameter akzeptiert. Funktioniert unabhängig davon, ob eine Datei geöffnet ist oder nicht. **Mögliche Rückgabewerte von at+readerror:** - INVALID PARAMETER - I AM HAPPY

FEHLERMELDUNGEN

Nr.	Fehlertext	Beschreibung
0x00	I AM HAPPY	Es sind keine Fehler aufgetreten
0x01	INPUT BUFFER OVERFLOW	Es wurde ein überlanger Befehl eingegeben (mehr als 64 Zeichen). Verwenden Sie korrekten Syntax um dies zu vermeiden (z.B. schließen Sie Kommandos mit <cr><lf>.
0x02	PARAMETER COUNT MISMATCH	Es wurde ein Parameter vergessen. Z.B. bei at+open wurde der Dateiname vergessen. Wenn zu viele Parameter übergeben worden sind, so wird auch diese Fehlermeldung ausgegeben (z.B. wurde ein „Space“ zu viel eingegeben).
0x03	INVALID PARAMETER	Der übergebene Parameter stimmt nicht bzw hat ein falsches Format. Z.B. bei at+settime wurde ein fehlerhaftes Format verwendet.
0x04	PARAMETER OUT OF RANGE	Der übergebene Parameter ist zu groß oder zu klein.
0x05	COMMAND NOT IMPLEMENTED	Der verwendete Befehl steht noch nicht zur Verfügung. Schauen Sie auf der Web-Seite nach Software-Updates.
0x06	UNKNOWN COMMAND	Der verwendete Befehl ist unbekannt. Überprüfen Sie die Schreibweise.
0x07	FILE IS OPEN	Es ist eine Datei geöffnet und der Befehl kann nicht ausgeführt werden, so lange diese Datei offen ist. Z.B. ein at+open file1 gefolgt von einem at+open file2. Schließen Sie die Datei mit at+close und führen Sie den Befehl erneut aus.
0x08	NO OPEN FILE	Es wurde ein Befehl übertragen, der eine geöffnete Datei benötigt. Z.B. at+datastream benötigt einen at+open file Befehl vorab.
0x09	MEMORY CARD NOT PRESENT	Es ist keine Speicherkarte gesteckt, oder die gesteckte Speicherkarte wurde nicht als Speicherkarte erkannt.
0x0A	LOW RS232 INPUT BUFFER	Der Eingangsspeicher zur Pufferung von Daten war gefährlich niedrig. Es ist zwar noch keine Datenverlust vorgekommen, aber es ist

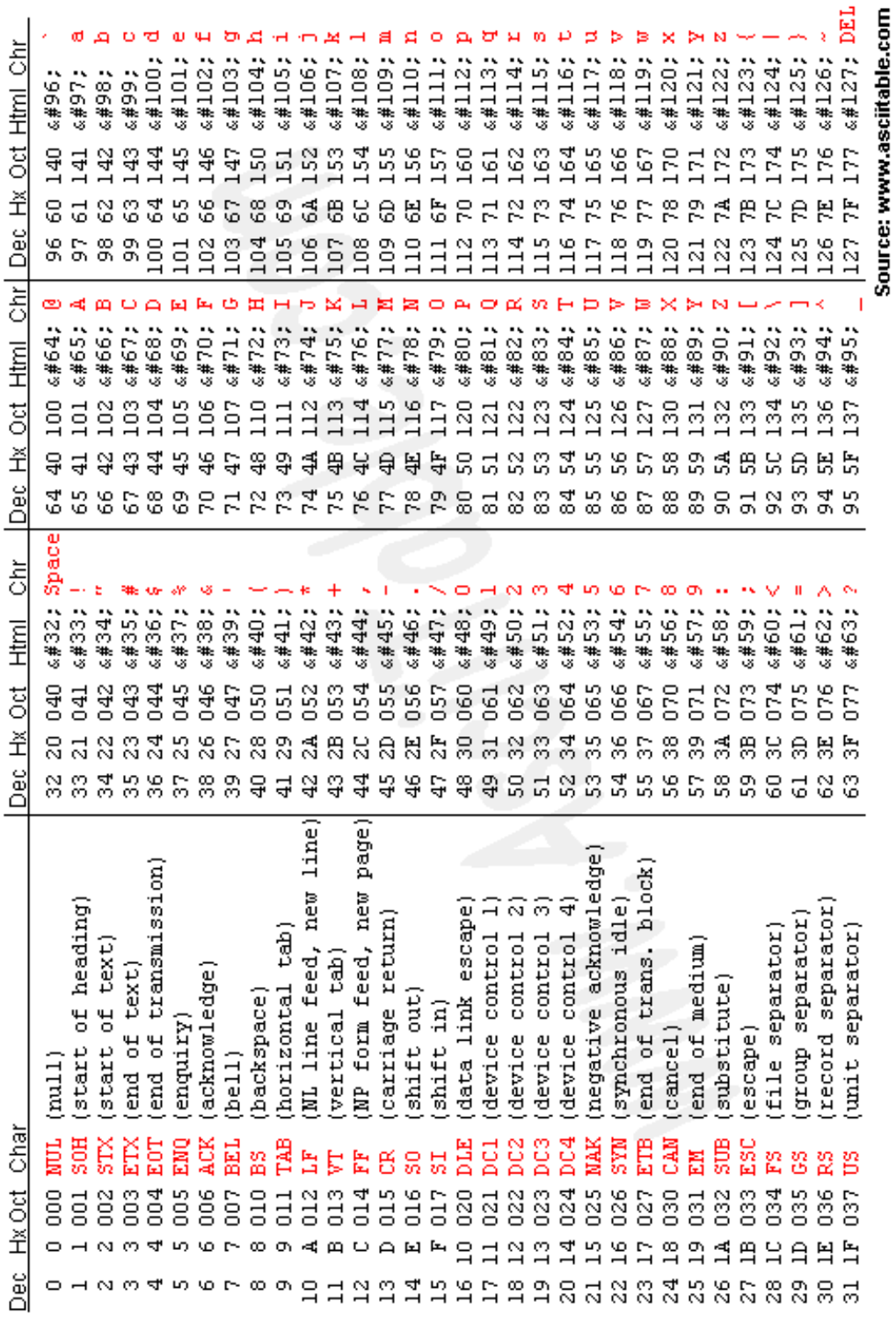
		damit zu rechnen, dass dies passieren wird.
0x0B	RS232 INPUT BUFFER OVERFLOW	Der Eingangsspeicher ist übergelaufen. Die Daten konnten nicht schnell genug zur Speicherkarte geschrieben werden. Es ist zum Datenverlust gekommen.
0x0C	WRONG PACKET LENGTH	Paketmodus: Die Längenangabe ist kleiner als 1 oder größer als 1536 Bytes
0x0D	CRC ERROR	Paketmodus: Der CRC-Wert ist falsch oder ungleich 0 (kein CRC-Wert).
0x0E	IMPROPER FLOW CONTROL METHOD	Paketmodus: Software Flow Control (XON/XOFF) ist aktiv. Der Paketmodus akzeptiert nur Hardware Flow Control (RTS/CTS) oder kein Flow Control.

Nr.	**Fehlertext**	**Beschreibung**
0x40	ACCESS DENIED	Der Zugriff ist wurde verweigert. Z.B. ein at+create file , wärend file schreibgeschützt ist.
0x41	NAME CONFLICTS WITH SUBDIRECTORY	Es wurde versucht eine Datei zu erzeugen, deren Name als Verzeichnissname vorhanden ist. Wählen Sie einen anderen Namen.
0x42	INSUFFICIENT DIRECTORY SPACE	Es ist kein Speicherplatz mehr vorhanden.
0x43	DISK IMPROPERLY FORMATTED	Die Formatierung des Speichermediums ist fehlerhaft. Formatieren Sie das Medium auf einem PC neu, oder korrigieren Sie mit „chkdsk“ den Fehler.
0x44	ERROR IN READING OR WRITING DISK	Es konnte nicht geschrieben oder gelesen werden. Vielfältige Fehlerursachen.
0x45	CORRUPT FILE ALLOCATION TABLE	Das Dateisystem ist beschädigt. Fehlerursache ist meist, wenn die Karte herausgezogen wurde, während Daten geschrieben worden sind.
0x46	DIRECTORY DOES NOT EXIST	Das bezeichnete Verzeichnis ist nicht vorhanden.
0x47	INVALID DRIVE OR FILE NAME	Es können mehrere Partitionen auf dem Speichermedium sein. Dies wird zur Zeit noch nicht unterstützt. Dieser Fehler wird auch ausgegeben, wenn ein ungültiger Pfad angegeben wurde. Wenn z.B.

		Verzeichnisse mit „/“ anstelle von „\“ getrennt eingegeben wurden.
0x48	INVALID MODE	Interner Softwarefehler. Es wurde ein falscher Modus verwendet. Bitte berichten Sie diesen Fehler an Avisaro (support@avisaro.com).
0x49	FILE NOT FOUND	Es wurde versucht auf eine Datei zuzugreifen, die nicht existiert.
0x4a	FILE NAME OR DIRECTORY ALREADY EXISTS	Es wurde versucht ein Verzeichnis zu erzeugen, dass bereits existiert. Bitte wählen Sie einen anderen Namen.
0x4b	SEEK OUT OF RANGE	Interner Softwarefehler. Der Dateizeiger hat einen falschen Wert. Bitte berichten Sie diesen Fehler an Avisaro (support@avisaro.com).
0x4c	THE FILE CAN NOT BE RESTORED	Interner Softwarefehler. Der Dateizeiger hat einen falschen Wert. Bitte berichten Sie diesen Fehler an Avisaro (support@avisaro.com).
0x4d	UNDEFINED FILE SYSTEM ERROR	Allgemeiner Fehler. Bitte wenden Sie sich an Avisaro zur Usachenforschung.

ASCII ZEICHEN TABELLE

Unter „www.asciitable.com“ finden Sie folgende und weitere nützliche ASCII Tabellen:



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

PROGRAMMIERBEISPIELE

QUELLCODE CRC BERECHNUNG

Im Paketmodus kann eine CRC Prüfsumme verwendet werden, um Daten bei der Übertragung zu sichern. Die Berechnung des CRCs kann nach folgendem Schema erfolgen:

```
static const unsigned int crc_tab[16] =
{
  0x0000, 0x1081, 0x2102, 0x3183,
  0x4204, 0x5285, 0x6306, 0x7387,
  0x8408, 0x9489, 0xA50A, 0xB58B,
  0xC60C, 0xD68D, 0xE70E, 0xF78F
};


unsigned short crc_update (unsigned short crc, unsigned char c)
{
    crc = (((crc >> 4) & 0x0FFF) ^ crc_tab[((crc ^ c) & 0x000F)]);
    crc = (((crc >> 4) & 0x0FFF) ^ crc_tab[((crc ^ (c>>4)) & 0x000F)]);
    return crc;
}
```

- Eine Variable (16 Bits, der CRC-Wert) wird mit 0xffff initialisiert:
  unsigned short crc_wert = 0xffff;

- Für jedes Byte des Pakets wird die Variable verändert, indem man die obenstehende Funktion aufruft (in einer Schleife):
  crc_wert = crc_update (crc_wert, aktuelles_byte);

- Nachdem das letzte Byte verarbeitet wurde, wird die Variable negiert:
  crc_wert = ~crc_wert;

QUELLCODE „WRITE PACKET“

Das folgende Beispiel illustriert, wie eine Anwendung ein Paket generieren könnte, das zum Avisaro Modul übertragen wird.

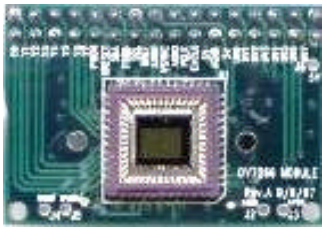
```
void CommWritePacket (HANDLE hcomm, char *data, short size)
{
    unsigned short crc = 0xffff;

    CommWrite (hcomm, size & 0xff);
    CommWrite (hcomm, size >> 8);
    while (size--)
    {
        CommWrite (hcomm, *data);
        crc = crc_update (crc, *data);
        data++;
    }

    //CommWrite (hcomm, 0); //diese Zeilen verwenden, wenn kein
    //CommWrite (hcomm, 0); //CRC verwendet werden soll
    crc = ~crc;
    CommWrite (hcomm, crc & 0xff); // diese Zeilen weglassen, wenn
    CommWrite (hcomm, crc >> 8);   // kein CRC verwendet werden soll
}
```

C3088
1/4" Color Camera Module
With Digital Output

General Description



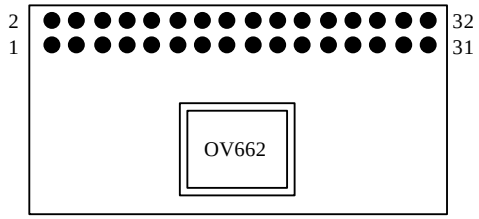
The C3088 is a 1/4" color camera module with digital output. It uses OmniVision's CMOS image sensor OV6620. Combining CMOS technology together with an easy to use digital interface makes C3088 a low cost solution for higher quality video image application.

The digital video port supplies a continuous 8/16 bit-wide image data stream. All camera functions, such as exposure, gamma, gain, white balance, color matrix, windowing, are programmable through I^2C interface.

In combine with OV511+, USB controller chip, it will be easily form a USB camera for PC application.

Features:

101,376 pixel, CIF/QCIF format
Small size : 40 x 28 mm
Lens: f=4.9mm (Optional)
8/16 bit video data : CCIR601, CCIR656, ZV port
Read out - progressive
Data format -YCrCb 4:2:2, GRB 4:2:2, RGB
I^2C interface
Wide dynamic range, anti blooming, zero smearing
Electronic exposure / Gain / White balance control
Image enhancement - brightness, contrast, gamma, saturation, sharpness, window, etc
Internal / external synchronization scheme
Frame exposure / line exposure option
Single 5V operation
Low power consumption (<100mW)
Monochrome composite video signal output(50 Hz)



PCB Layout (Top side)

Application Example

- Video Conferencing
- PC Multimedia
- Video Phone
- Video Mail
- Still Image
- Machine Vision
- Process control

Note: Evaluation Board is available for C3088

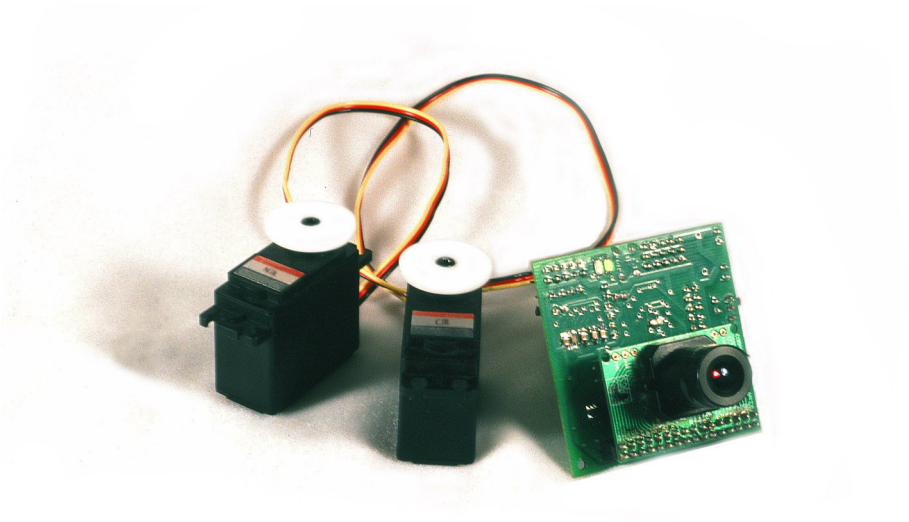
Specification

Imager	OV6620, CMOS image sensor
Array Size	356x 292 pixels
Pixel size	9.0 x 8.2 µm
Scanning	Progressive
Effective image area	3.1mm x 2.5mm
Electronic Exposure	500:1
Gamma Correction	0.45/0.55/1.0
S/N Ratio	48dB
Min Illumination	3lux @F1.2
FPN	<0.03% Vp-p
Dark current	<0.2 nA/cm^2
Dynamic Range	72dB
Operation Voltage	5 VDC
Operation Current	80mW Active 30 µW Standby
Lens (Optional)	F4.9mm, F2.8

Pin Description

1~8	Y0~Y7	Digital output Y Bus.
9	PWDN	Power down mode
10	RST	Reset
11	SDA	I^2C Serial data
12	FODD	Odd Field flag
13	SCL	I^2C Serial clock input
14	HREF	Horizontal window reference output
15	AGND	Analog Ground
16	VSYN	Vertical Sync output
17	AGND	Analog Ground
18	PCLK	Pixel clock output
19	EXCLK	External Clock input (remove crystal)
20	VCC	Power Supply 5VDC
21	AGND	Analog Ground
22	VCC	Power Supply 5VDC
23~30	UV0-UV7	Digital output UV bus.
31	GND	Common ground
32	VTO	Video Analog Output (75Ω monochrome)

CMUcam2 Vision Sensor



User Guide

* Servos not included.

Contents

This Icon will link you to pages where more detailed general information can be found.

This icon will warn you about common mistakes.

This icon will point you to pages where commands used in the text are described.

This icon will suggest a generic tip from your friend the yellow dart.

This is the CMUcam2 Manual v1.05 for the CMUcam2 v1.0 firmware.
For more information go to http://www.cs.cmu.edu/~cmucam or contact us at cmucam@cs.cmu.edu

Edited by Charles Rosenberg and Illah Nourbakhsh

Introduction

The CMUcam2 consists of a SX52 microcontroller (http://www.ubicom.com/products/sx/sx.html) interfaced with an OV6620 or OV7620 Omnivision CMOS camera (http://www.ovt.com) on a chip that allows simple high level data to be extracted from the camera's streaming video. The board communicates via a RS-232 or a TTL serial port and has the following functionality:

- Track user defined color blobs at up to 50 Frames Per Second*
- Track motion using frame differencing at 26 Frames Per Second
- Find the centroid of any tracking data
- Gather mean color and variance data
- Gather a 28 bin histogram of each color channel
- Manipulate Horizontally Pixel Differenced Images
- Transfer a real-time binary bitmap of the tracked pixels in an image
- Arbitrary image windowing
- Adjust the camera's image properties
- Dump a raw image (single or multiple channels)
- Up to 160 x 255 Resolution**
- Supports Multiple Baudrates: 115,200 57,600 38,400 19,200 9,600 4,800 2,400 1,200
- Control 5 servo outputs
- Slave parallel image processing mode off of a single camera bus
- Automatically use servos to do two axis color tracking
- B/W Analog video output (PAL or NTSC)**
- Flexible output packet customization
- Multiple pass image processing on a buffered image
- Works with the OV7620 or OV6620 module

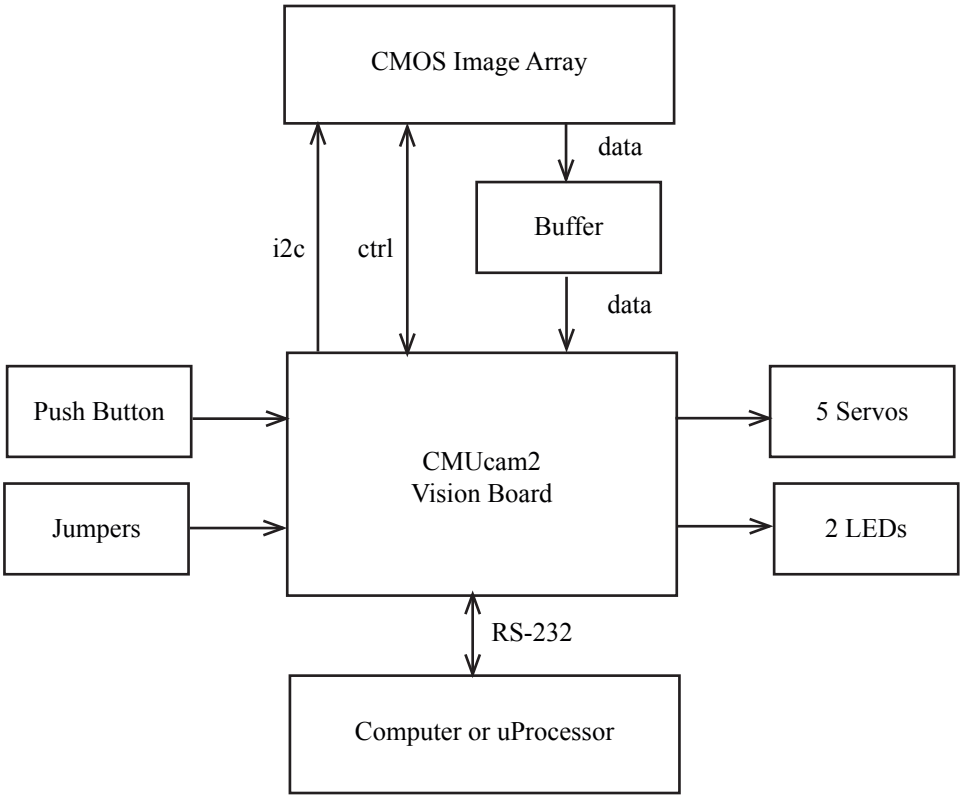
*Frame Rate Depends on Window Size
**Camera Properties Depend on Camera Module

Typical Configurations and Uses

Typical Uses

One of the primary uses of the CMUcam2 is to track or monitor color. The best performance can be achieved when there are highly contrasting and intense colors. For instance, it can easily track a red ball on a white background, but it would be hard to differentiate between different shades of brown in changing light. Tracking colorful objects can be used to localize landmarks, follow lines, or chase a moving beacon. Using color statistics, it is possible to monitor a scene, detect a specific color or do primitive motion detection. If the camera detects a drastic color change, then chances are something in the scene changed. Using "line mode," the CMUcam2 can act as an easy way to get low resolution binary images of colorful objects. This can be used to do more sophisticated line following that includes branch detection, or even simple shape recognition. These more advanced operations would require custom algorithms that would post process the binary images sent from the CMUcam2.

See line mode on page 35.



Typical Configuration

The most common configuration for the CMUcam2 is to have it communicate to a master processor via a standard RS232 serial port. This “master processor” could be a computer, PIC, Basic Stamp, Handy Board, Brainstem or similar microcontroller setup. The CMUcam2 is small enough to add simple vision to embedded systems that cannot afford the size or power of a standard computer based vision system. Its communication protocol is designed to accommodate even the slowest of processors. If your device does not have a fully level shifted serial port, you can also communicate to the CMUcam2 over the TTL serial port. This is the same as a normal serial port except that the data is transmitted using non-inverted 0 to 5 volt logic. The CMUcam2 supports various baud rates to accommodate slower processors. For even slower processors, the camera can operate in “poll mode”. In this mode, the host processor can ask the CMUcam2 for just a single packet of data. This gives slower processors the ability to more easily stay synchronized with the data. It is also possible to add a delay between individual serial data characters using the “delay mode” command. Due to the communication delays, both poll mode and delay mode will lower the total frame rate that can be processed. Frame resolutions are not affected by delay mode or baud rate as they were in the original CMUcam.

See poll mode on page 46.

See delay mode on page 33.

Operational Explanation

How does an image get converted into a series of pixels?

The CMOS image sensor is the heart of what actually gathers the information. It is a silicon chip that contains a grid of boxes, each of which are sensitive to different colors of light. After light passes through the lens, it stimulates these boxes, generating a different voltage proportional to the amount of light. This voltage gets converted into a single numerical value for each channel. In the case of the CMUcam2, this value is in the range of 16 to 240. There is a red channel, a blue channel and two green channels, each of which are only sensitive to that particular color of light. The extra green channel helps fill in the grid so that each pixel can be evenly spaced across the sensor. The extra green information also more closely approximates the human eye which is more sensitive to the color green. For the purpose of simplification, the CMUcam2 ignores the second green value.

Camera Sensor Output Pixel Mapping

It is sometimes useful to understand more percisely how the data from the camera sensor is translated into pixels. Here we explain it for the OV6620 sensor, but the same basic layout applies to the OV7620 sensor.

The sensor has 356 columns and 292 rows of light sensitive cells arranged on a grid. Each location can detect a single color: red, green or blue. Here is the sensor layout of the first four rows:

Row 1: B(1,1) G(1,2) B(1,3) G(1,4) B(1,5) G(1,6) ...B(1,355) G(1,356)
Row 2: G(2,1) R(2,2) G(2,3) R(2,4) G(2,5) R(2,6) ...G(2,355) R(2,356)
Row 3: B(3,1) G(3,2) B(3,3) G(3,4) B(3,5) G(3,6) ...B(3,355) G(3,356)
Row 4: G(4,1) R(4,2) G(4,3) R(4,4) G(4,5) R(4,6) ...G(4,355) R(4,356)

The camera module takes the data from two sensor rows at a time to generate each line output from the camera module:

Row 1: B(1,1) G(2,1) R(2,2) G(1,2) B(1,3) G(2,3) R(2,4) G(1,4) ...
Row 2: B(3,1) G(2,1) R(2,2) G(3,2) B(3,3) G(2,3) R(2,4) G(3,4) ...

The CMUcam2 takes this data and outputs following pixel data:

Row 1: [R(2,2):G(1,2):B(1,1)] [R(2,4):G(1,4):B(1,3)] ...
Row 2: [R(2,2):G(3,2):B(3,1)] [R(2,4):G(3,4):B(3,3)] ...

What is tracking a color and how does the CMUcam2 do it?

Color tracking is the ability to take an image, isolate a particular color and extract information about the location of a region of that image that contains just that color. As an example, assume that you are given a photograph that contains a red ball sitting on a dirt road. If someone were to ask you to draw a box around anything that was the color red in the image, you would quite easily draw a rectangle around the ball. This is the basic idea behind color tracking. You did not need to know that the object was a ball. You only needed to have a concept of the color red in order to isolate the object in the picture. In this section we will briefly address how the CMUcam2 actually uses the information in a camera image to perform color tracking.

(Photograph Courtesy of Jim Reed)

In order to specify color, you need to define a minimum and maximum allowable value for each of those three color channels. Every unique color is represented by a red, green, and blue value that indicates how much of each channel is mixed into that final color. The tricky part about specifying a color is that you need to define a range of allowable values for all three color channels. Since light is not perfectly uniform and the color of an object is not perfectly uniform, you need to accommodate for these variations. However, you don't want to relax these bounds too much, or many unwanted colors will be accepted. Since, in the case of the CMUcam2, each color channel is converted into a number between 16 and 240, you can bound each channel with two numbers, an upper and lower limit. If you have two limits for each of the three channels, this means that six values can be used to constrain the entire color space that you wish to track. If you imagine the colors being represented by a cube where each side is a different color channel (red, green and blue) then the six values used to select your color would draw a three dimensional box inside that cube that defines your desired set of colors.

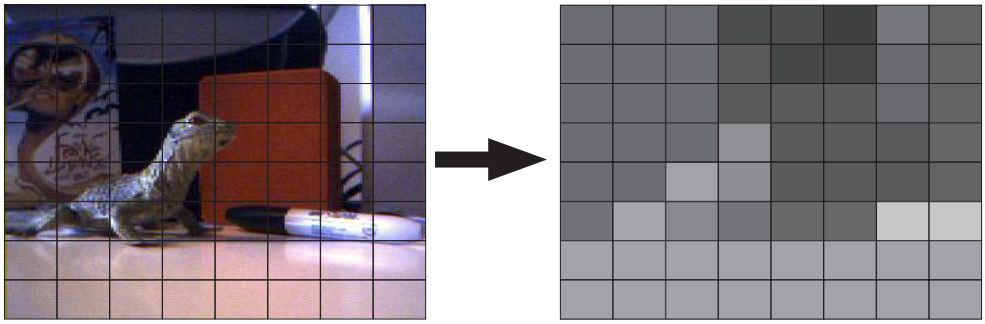
Once you have a bound for the color you wish to track, the CMUcam2 takes these bounds and processes the image. There are many ways to track colors in an image that can be quite complex. The CMUcam2 uses a simple one pass algorithm that processes each new image frame from the camera independently. It starts at the top left of the image and sequentially examines every pixel row by row. If the pixel it is inspecting falls inside the range of colors that the user specified, it marks that pixel as being tracked. It also examines the position of the current tracked pixel to see if it is the top most, bottom most, left most or right most position of all the tracked pixel found thus far in the image. If it finds that the pixel is outside of the current bounding box of the tracked region, it grows the bounding box to contain this new pixel. Because the location of even a single tracked pixel can change the bounding box, the bounding box can sometimes fluctuate quite a bit from frame to frame. Noise filtering (see next paragraph) can be used to reduce some of that fluctuation. The only other major piece of information that is stored is a sum of the horizontal and vertical coordinates of the tracked pixels. At the end the image you can take the horizontal sum and the vertical sum of the tracked pixels and divide each by the total number of tracked pixels, you get a value that shows where the middle of the tracked object is located. Because each tracked pixel only contributes a small part to the final horizontal and vertical sums the middle (often called the centroid) of the tracked pixels is typically a much more stable measurement than the bounding box. Once all of the pixels in the image have been checked, the total number of tracked pixels can also be used in conjunction with the area of the bounding box to calculate the confidence of the tracked object.

Noise filtering allows us to make the color tracking ranges larger so we can accommodate larger variations in the image pixel values without causing other random variations in the image to be tracked. The idea behind noise filtering is that we only want to consider a pixel to be of the tracked color if it is part of a group of pixels that are within the color tracking bounds. Again in the CMUcam2 we implement this in a way that only requires a single pass over the image. While processing the pixels in an image the CMUcam2 maintains a counter which keeps of track of how many sequential pixels in the current row, before the current pixel were within the tracked color bounds. If that value is above the noise filter value then the current pixel is marked as a tracked pixel.

How does the CMUcam2 do Frame Differencing?

Frame differencing is a method of identifying changes in a series of images. Given multiple images at different times from the same or similar view points, it is possible to compare them in order to isolate objects that may have moved. Using the CMUcam2's frame differencing functionality is a good way to detect and track such motion in a scene. Instead of storing an entire image, the CMUcam2 stores an abstraction of the image. Using a similar process to color tracking, the CMUcam2 will generate or compare the image on a line by line basis as it receives the data.

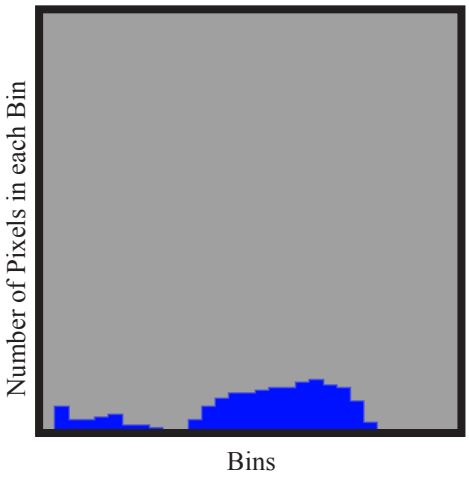
The CMUcam2 internally represents a reference image as an array of 8 by 8 bytes. Each element of this array stores the average of a corresponding region on the main camera image. The default setting uses the green or intensity channel, but this can be changed for situations where one channel clearly shows more variation than the others. When a new image is read in, it is also converted into an array of 8x8 bytes. To look for a change, each block in the 8x8 grid is subtracted from the corresponding reference image block. If there is more than a specified threshold, a change is flagged. The rest of the data, such as middle mass, is calculated in an almost identical manner to the way it is in color tracking.



What is a histogram and what is it good for?

A histogram is a type of chart that displays the frequency and distribution of data. In the case of the CMUcam2, the histograms show the frequency and distribution of color values found in an image. Each bar represents a range of color values for a specific channel. The CMUcam2 can divide the possible color values from 16 to 240 into up to 28 different bins. Each bin contains the number of pixels found in the image that fall within those color bounds. So a large value in one particular bin, means that many of those colors where found in the image. Each histogram only represents one select channel of color. Using buffer mode it is possible to quickly grab three histograms, one for each channel.

Histograms are a way of abstracting the contents of an image. They have many uses such as primitive object recognition, thresholding or color balancing. They are particularly useful for distinguishing between different textures. Try pointing the CMUcam2 with auto-gain turned off at two different textured surfaces and notice the difference in their color distributions. This effect could be used to distinguish floor surfaces or detect obstacles. When used in conjunction with pixel differencing a histogram can tell you about the strength of the edges visible to the camera.

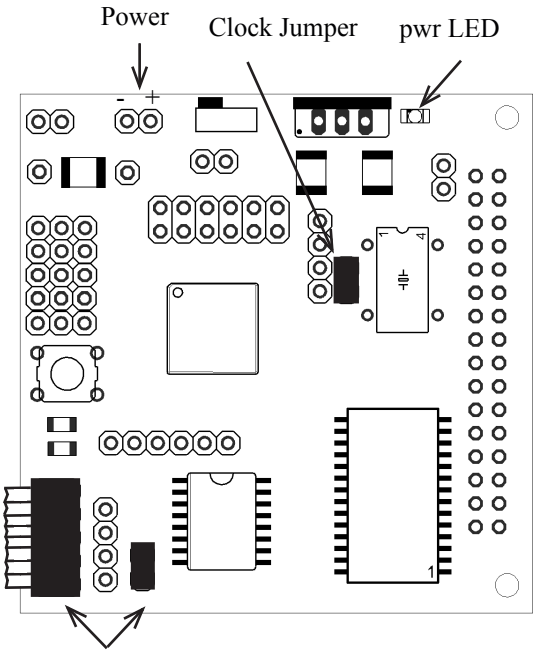


Getting Started

Setting Up the Hardware

In order to initially test your CMUcam2, you will need a serial cable, a power adapter and a computer. The CMUcam2 can use a power supply which produces anywhere from 6 to 15 volts of DC power capable of supplying at least 200mA of current. This can be provided by either an AC adapter (possibly included) or a battery supply. These should be available at any local electronics store. The serial cable should have been provided with your CMUcam2. Make sure that you have the CMOS sensor board connected to the CMUcam2 board so that it is in the same orientation as the picture shows on the cover of this manual.

See page 59 for startup troubleshooting.



First, connect the power. Make sure that the positive side of your power plug is facing away from the main components on the board. If the camera came with an AC adapter, make sure that the connector locks into the socket correctly.

Now that the camera has power, connect the serial link between the camera and your computer. This link is required initially so that you can test and focus your camera. The serial cable should be connected so that the ribbon part of the cable faces away from the board. You must also connect the serial pass through jumper.

Check to make sure that the clock jumper is connected. This allows the clock to actively drive the processor.

Once everything is wired up, try turning the board on. The power LED should illuminate green and only one LED should remain on. Both LEDs turn on upon startup, and one turns off after the camera has been sucessfully configured.

Make Sure Clock and Serial Jumper are in place.

Testing the Firmware

Once you have set the board up and downloaded the firmware, a good way to test the system is to connect it to the serial port of a computer.

Step 1: If one does not already exist, build a serial and/or power cable

Step 2: Plug both of them in.

Step 3: Open the terminal emulator of your choice.

See page 62, for more detailed terminal software information.

Step 4: Inside the terminal emulator set the communication protocol to 115,200 Baud, 8 Data bits, 1 Stop bit, no parity, local echo on, no flow control and if possible turn on "add line feed" (add \n to a received \r). These setting should usually appear under "serial port" or some other similar menu option.

Step 5: Turn on the CMUcam2 board; the Power LED should light up and only one of the two status LEDs should remain on.

Step 6: You should see the following on your terminal emulator:

```
CMUcam2 v1.0 c6
:
```

If you have seen this, the board was able to successfully configure the camera and start the firmware.

Step 7: Type gv followed by the enter key. You should see the following:

```
:gv
ACK
CMUcam2 v1.0 c6
:
```

See get version on page 37.

This shows the current version of the firmware. If this is successful, your computer's serial port is also configured correctly and both transmit and receive are working.

Focusing with the CMUcam2 Graphical User Interface (GUI)

When you first run your CMUcam2, the lens will most likely not be in focus. In order to focus the camera you need to look at some dumped images. The easiest way to do this is using a graphical user interface that can display the CMUcam2 frame dump packets. One option is to use the CMUcam2GUI, a Java program that can be found on the CMUcam2 website.

Step 1: Testing if you have java installed

The CMUcam2GUI needs java version 1.4.0 or higher.

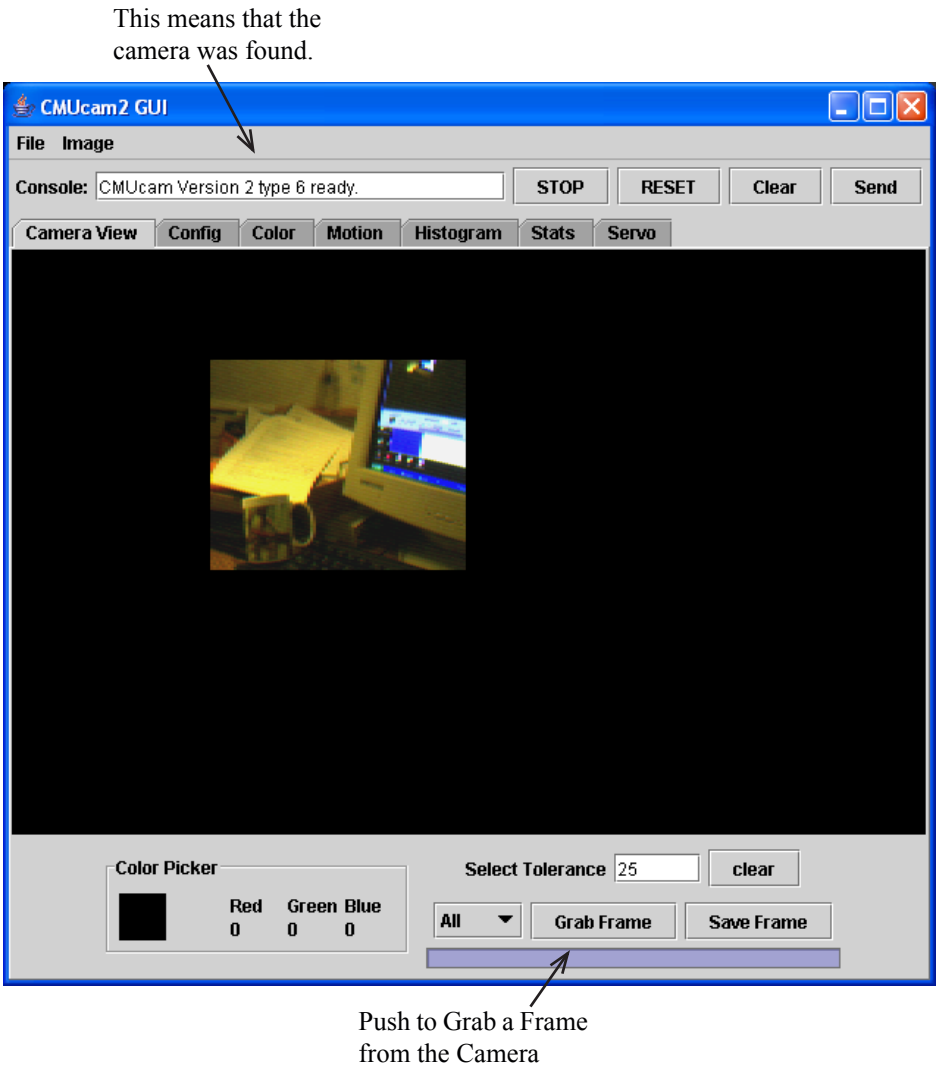
The first step is to determine if your computer already has java installed. The easiest way to do this is go to the "start menu" in windows and select "run". Inside the run dialog, type "command" to get a dos prompt. (In unix or later versions of the Mac OS, open up a shell.) Now try typing "Java -version" into your command line. If a message that says "Java version "1.x.xx" appears then java is installed. If instead you get "command not found" or some similar message, then you need to go to java.sun.com and download a copy of Java (J2SE, JDK, JRE are all valid things to install). Sun should have platform specific instructions on how to install java. Also be sure that your version of Java is 1.4.0 or newer. If it is not, then you will need to download a new copy of Java.

Step 2: Running the CMUcam2GUI

Once you have Java installed, download a copy of the latest CMUcam2GUI Java program. Unzip the CMUcam2GUI.zip file. Open up the stand_alone folder. In Windows double click on the CMUcam2GUI jar file. In unix, navigate to the CMUcam2GUI directory and type "java -jar CMUcam2GUI" to execute the GUI.

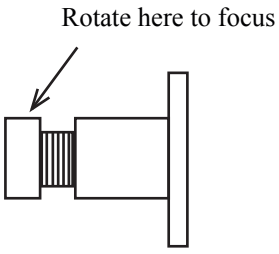
Step 3: Grabbing a Frame

You should now see a dialog box that asks you to select the correct serial COM port. In windows, type in the number of the COM port that the CMUcam is connected to and press the "Ok" button. In unix, make sure that the path to your com port is correct and then press "Ok". The CMUcamGUI should now open and display the message "CMUcam version 2 type X ready." in the "Console" box. That means that the CMUcam2GUI found and was able to communicate with the camera. Once this works, select "Send Frame". After a few seconds you should see an image appear in the window.



Step 4: Focusing

Once you have the ability to grab frames from the camera, you should be able to rotate the front part of the CMUcam lens and see the image change. Try to get the picture to be as sharp as possible by dumping frames and changing the position of the lens a small amount each time. Usually the camera is in focus when the lens is a few rotations away from the base. (Once you have focused the lens you may find it useful to use some electrical tape to keep it in place)



CMUcam Lens Mount

Step 5: Other things to try once the camera is focused

Now take a quick look at the Config tab. When you change Color Space, White Balance, etc. (except for Noise Filter), it will automatically get sent and configured to the CMUcam.

Now go to the Color tab. This has the TrackWindow button. Place a uniform, highly color-saturated object in front of the camera and click this button to track. To stop it use the "STOP" button top right. Try it with line mode by setting Config line mode on.

Go to Motion tab. Position the camera so it is looking at something static (non-moving) and hit Load Frame. Then immediately hit Frame Diff and continuous frame differencing to the loaded frame begins. Move a small object like a pencil across the camera FOV(field of view) to test. When done hit Stop.

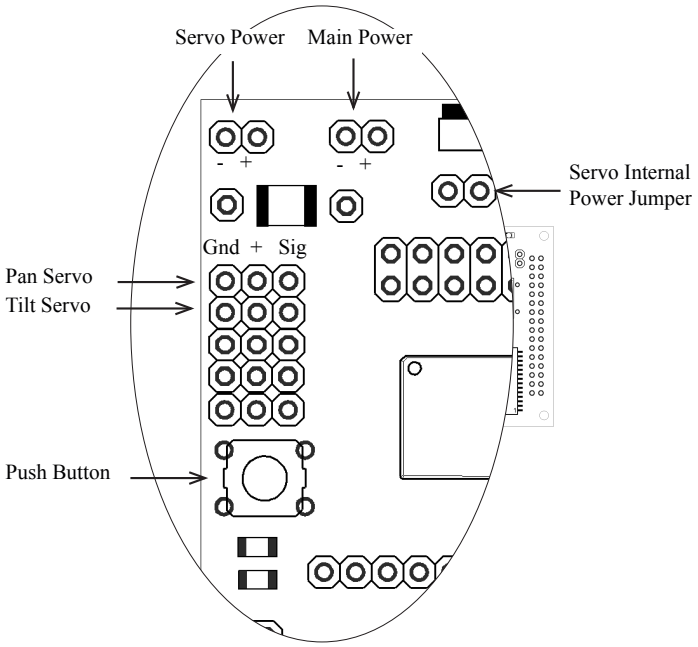
Now go to Histogram tab. In here you can do 1D histograms of each color channel individually. Left to right, the histogram shows amount of '0' at the left extreme (no intensity in that color) and '255' at the right extreme (high intensity in that color). It's continuous once you hit Get Histogram. Try something black, homogenously colored, something with varied color. Again, use STOP to finish.

On the Stats page, once you hit GetMean, there is a very nice continuous update of the mean color seen across the camera's window.

Demo Mode

Demo mode causes the camera to call track window and then drive two standard hobby servos towards the object being tracked. This can be initiated autonomously at startup. First you need to plug a pan and/or tilt servo into servo ports 0 and 1. Servo port 0 is for the pan, while 1 is for the tilt. Next, make sure that the servos are being powered by either the internal servo power jumper or by an external power source. While holding down the push button, turn the camera on. The tracking LED should begin rapidly blinking. Immediately release the push button and wait for the LED to stop blinking. Next, point the camera at a colored object and press the push button again. This should grab the color of the object and begin automatically servoing towards it. If the servos appear to be driving in the reverse direction, add the appropriate servo direction jumper. During the period when the LED is blinking, the camera is adjusting to the light conditions in the room. Try not to hold the object in front of the camera while this is occurring. Experiment with different colors and lighting. You will notice that some work much better than others.

See page 21, for pan and tilt servo reverse jumpers.

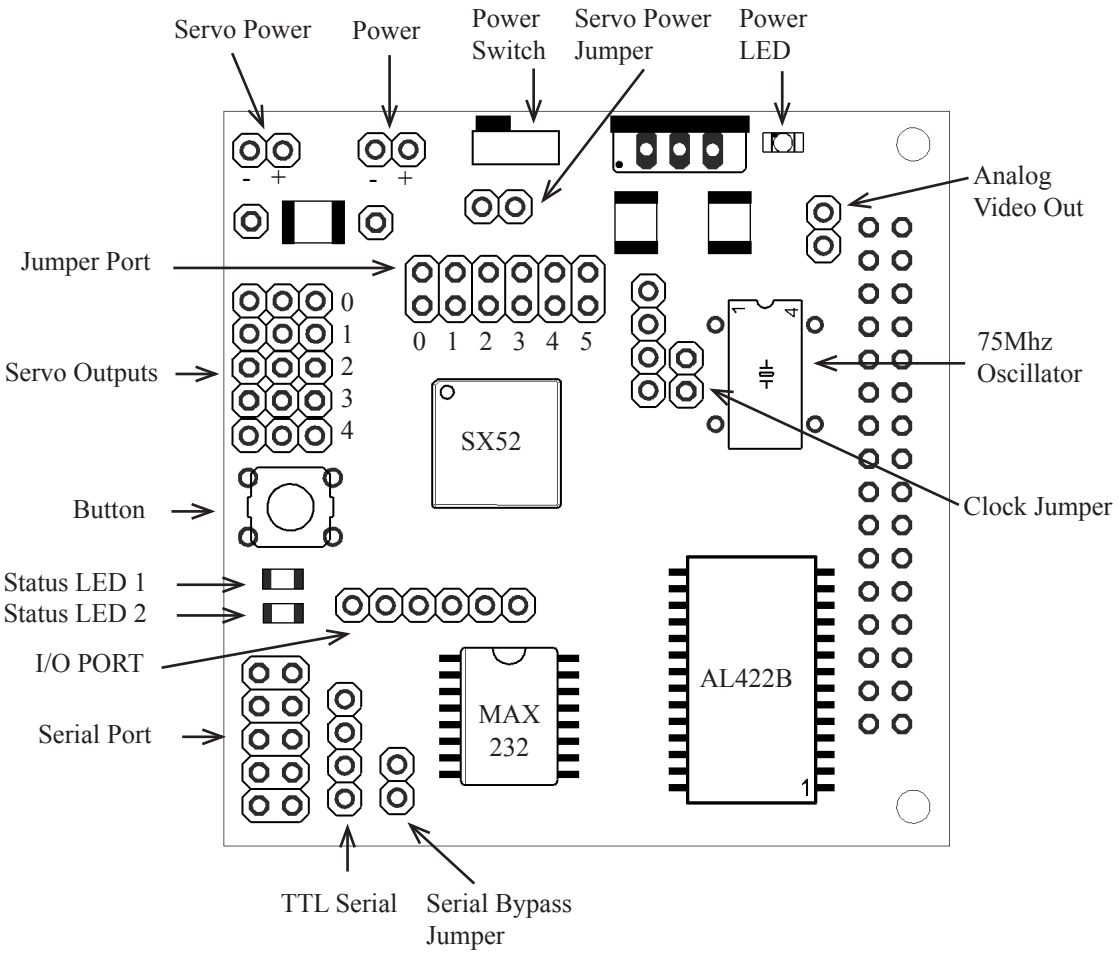


The following steps are performed during power up in demo mode:

1. RS is sent to the camera
2. Pause 5 seconds while blinking the LED to allow the camera to stabilize
3. The Camera register string “CR 18 32 19 32” is sent.
4. Auto Servo Mode is enabled.
5. TW is called.

See RS on page 49.
See CR on page 31.
See SM on page 51.
See TW on page 54.

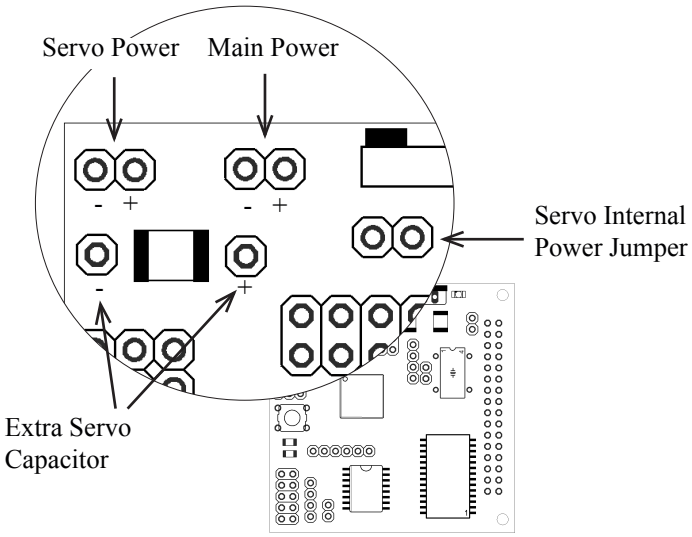
Board Layout



Ports

Power

The input power to the board goes through a 5 volt regulator. It is ideal to supply the board with between 6 and 15 volts of DC power that is capable of supplying at least 200 milliamperes of current.



Do not connect external servo power while the servo jumper is in place

If the servos are jittering or the camera does not properly power up, try soldering a 100uF external capacitor to the extra servo cap pads.

The servos can either be powered by internal power, or by the external servo power connector. To run them off of internal power, connect a jumper across the “servo internal power jumper”. To run them off of external power, leave the jumper open, and connect another 5volt supply to the servo power connector. Do not connect an external servo supply while the servo power jumper is in place. If the servos are drawing too much power or seem to be noisy, try soldering a large valued capacitor across the “Extra Servo Capacitor” connections. The external servo power is not switched by the main power switch.

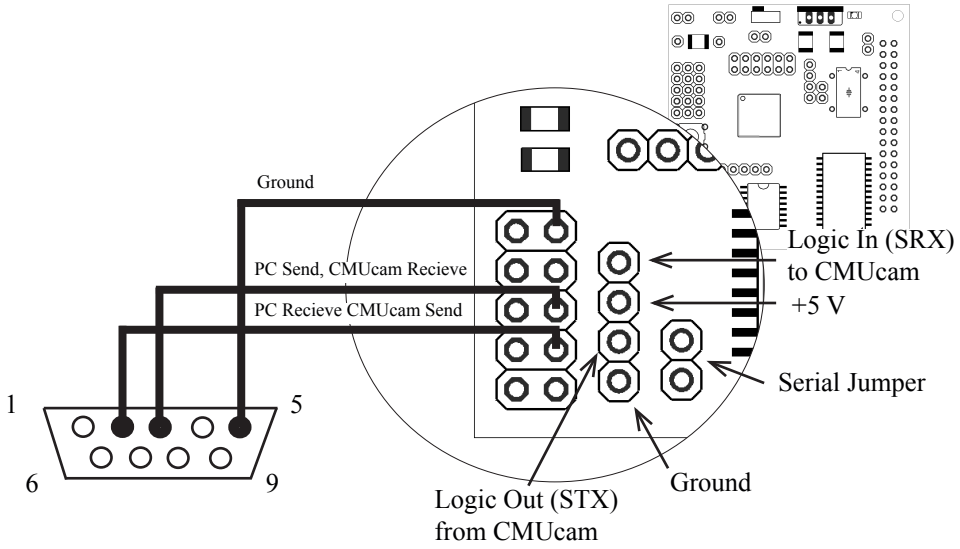
Serial Port

The CMUcam2 has a standard level shifted serial port to talk to a computer as well as a TTL serial port for talking to a microcontroller.

If the standard serial port does not work, try plugging in the serial connector the opposite way.

The level shifted serial port only uses 3 of the 10 pins. It is in a 2x5 pin configuration that fits a standard 9 pin ribbon cable clip-on serial sockets and 10 pin female clip on serial headers that can both attach to a 10 wire ribbon cable. If this initially does not work, try flipping the direction that the ribbon cable connects to the CMUcam2 board. Make sure the serial jumper is in place when you use this mode.

The TTL connector can be used to talk to a micrcontroller without the use of a level shifting chip. It operates between 0 and 5 volts. Remove the Serial Jumper when you use this mode.

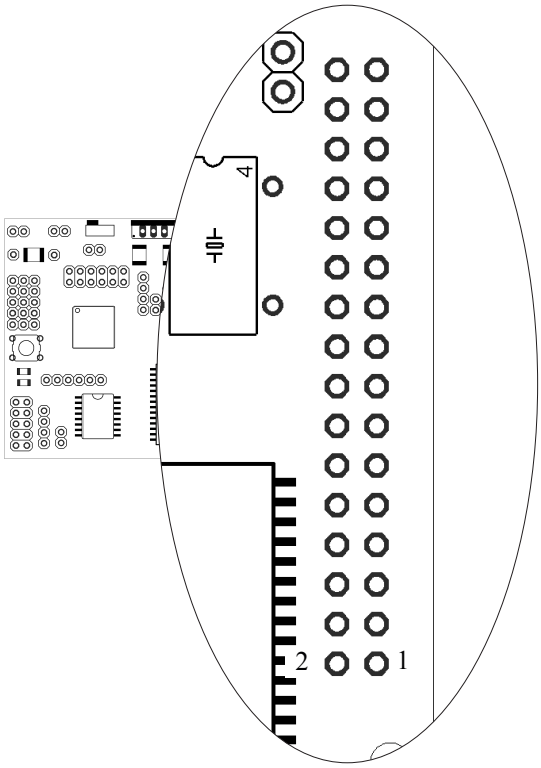


The Trapezoidal serial connector shown is what the serial connector on your computer should look like if drawn in an annoying line art drawing program.

Camera Bus

This bus interfaces with the CMOS camera chip. The CMOS camera board is mounted parallel to the processing part of the board and connects starting at pin 1. The female camera header should be soldered on the back of the board.

See page 25 for more information on the CMOS camera chips.



1-8	Y0-Y7	Digital Output Y Bus
9	PWDN	Power Down Mode
10	RST	Reset
11	SDA	I2C Serial Data
12	FODD	Odd Field Flag
13	SCL	I2C Serial Clock
14	HREF	Horizontal Ref
15	AGND	Analog Ground
16	VSYNC	Vertical Sync
17	AGND	Analog Ground
18	PCLK	Pixel Clock
19	EXCLK	External Clock
20	VCC	+5 VDC
21	AGND	Analog Ground
22	VCC	+5 VDC
23-30	UV0-UV7	Digital Output UV BUS
31	GND	Common Ground
32	VTO	Video Out (75Ohm)

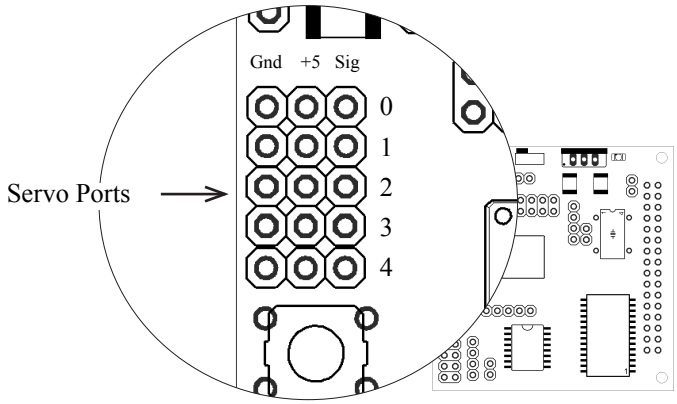
See the picture on the cover of the manual to make sure that you have the CMOS sensor connected correctly.

Servo Port

The CMUcam2 has the ability to control 5 servos. This can be useful if you do not wish to use a separate servo controller. The servo port can also be used as a general purpose digital outputs.

See SV servo command on page 53.

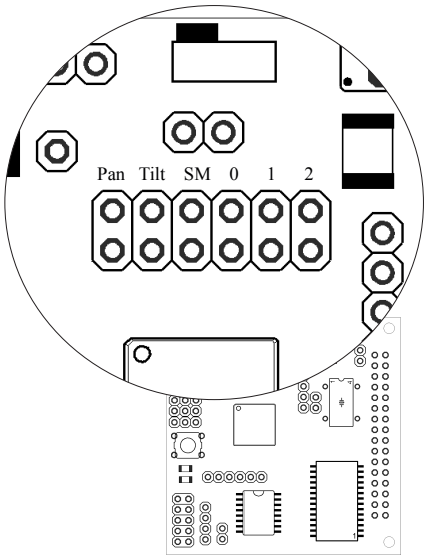
See SO servo command on page 51.



See page 17 for more information on servo power.

Configuration Jumpers

The jumpers can be used to set the camera's baudrates or configure various modes of operation.



Jumpers 0 1 and 2 set the camera into the following baudrates:

Baud Rate	Pin 0 1 2
115,200 Baud	_ _ _
57,600 Baud	_ _ X
38,400 Baud	_ X _
19,200 Baud	_ X X
9,600 Baud	X _ _
4,800 Baud	X _ X
2,400 Baud	X X _
1,200 Baud	X X X

X - jumper closed
_ - jumper open

Pan and Tilt Reverse Jumpers

During Auto Servo Mode, or demo mode it may be necessary to reverse the direction of the pan or tilt servo. Connecting the pan and/or tilt jumper will cause auto servo mode to send the opposite commands to each servo. Note, this only works for auto servo mode, and not for normal servo operations.

Slave Mode Jumper

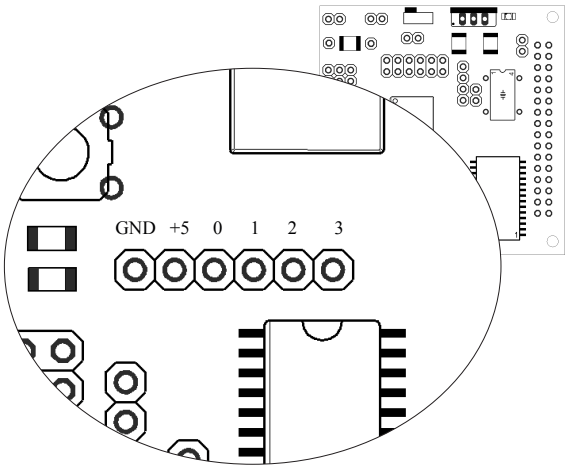
The CMUcam2 supports a mode of operation that allows multiple boards to process data from the same camera. If a PC104 style pass-through header is used instead of the standard double row female header, it is possible to rack multiple boards along the same camera bus. Upon startup, if the "SM" jumper is set, the camera becomes a slave. Slave mode stops the camera board from being able to configure or interfere with the CMOS camera's settings. Instead it processes the format setup by the master vision board. When linking the buses together you must only have one master; all other boards should be setup to be in slave mode. In this current version of the system there is no message passing between boards other than the image data from the camera bus. This means you have to communicate to each slave board via a separate serial link. This communication to the board should be identical to using a single CMUcam2. For example, you could have the master board tracking some color while the slave board could be told to get mean color data. Each board runs independently of one another and only the master can control camera registers.

See CT command on page 32.

Axuliary I/O

The CMUcam has 4 auxiliary Input Output ports that can be used for reading data from external devices. Note, that pin 3 is used for the sleep deeply command.

See GI command on page 35.
See SD command on page 49.



Analog Video Port

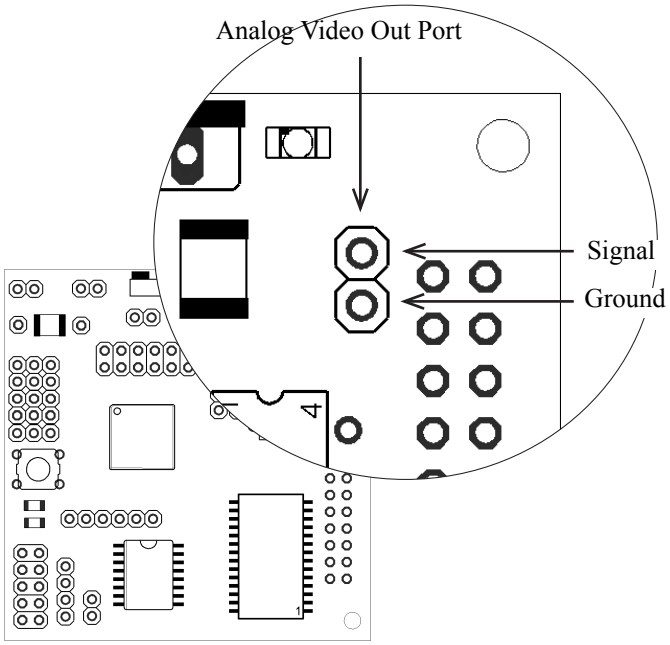
Using the OV6620 camera module, you will be able to get a PAL video signal from the analog port of the CMUcam2. This would sync up with any PAL monitor, but will not work with a standard NTSC monitor.

The OV7620 camera module will output a standard black and white NTSC video signal.

Make Sure Camera is operating at full frame rate and in YCrCb mode.

To use this output, it is necessary to keep the camera at its maximium frame rate (the default) and switch it into YCrCb mode in order to see the image on a monitor.

See CR command on page 31 for info on how to switch to YCrCb mode.



Notes on Better Tracking

Auto-gain and White Balance

Auto-gain is an internal control that adjusts the brightness level of the image to best suit the environment. It attempts to normalize the lights and darks in the image so that they approximate the overall brightness of a hand adjusted image. This process iterates over many frames as the camera automatically adjusts its brightness levels. If for example a light is turned on and the environment gets brighter, the camera will try and adjust the brightness to dim the overall image.

The camera module requires Auto-gain to be enabled to utilize white balance.

White balance on the other hand attempts to correct the camera's color gains. The ambient light in your image may not be pure white. In this case, the camera will see colors differently. The camera begins with an initial guess of how much gain to give each color channel. If active, white balance will adjust these gains on a frame-by-frame basis so that the average color in the image approaches a gray color. Empirically, this "gray world" method has been found to work relatively well. The problem with gray world white balance is that if a solid color fills the camera's view, the white balance will slowly set the gains so that the color appears to be gray and not its true color. Then when the solid color is removed, the image will have undesirable color gains until it re-establishes its gray average.

When tracking colors, like in demo mode, you may wish to allow auto-gain and white balance to run for a short period and then shut them off. While on for a period of about 5 seconds, the camera can set its brightness gain and color gains to what it sees as fit. Then turning them off will stop the camera from unnecessarily changing its settings due to an object being held close to the lens or shadows etc. If auto-gain and white balance where not disabled and the camera changed its settings for the RGB values, then the new measured values may fall outside the originally selected color tracking thresholds.

YCrCb Color Space

YCrCb is a different color space definition from the more commonly known RGB space. In YCrCb the pixel illumination data is stored in the Y channel. Because of this property, in YCrCb mode the camera may be more resistant to changes in illumination. Because it is a different color space, images in YCrCb do not look like standard RGB images when directly mapped by a frame dump program. The RGB channels map to CrYCb. So in YCrCb mode, the value returned as the red parameter is actually Cr, the green parameter is Y and the blue parameter is Cb. So if you wish to track a red object, you need to look at a dumped frame to see what that object's colors map to in YCrCb. It should then be possible to find the Cr and Cb bounds while giving a very relaxed Y bound showing that illumination is not very important. Below are the transformations used by the camera to convert RGB into YCrCb:

Notice that the RGB channels map to give you CrYCb, not YCrCb.

RGB -> CrYCb
Y=0.59G + 0.31R + 0.11B
Cr=0.713x (R-Y)
Cb=0.564x (B-Y)

When using YCrCb, make sure you take into account that in terms of all CMUcam I/O, Red maps to Cr, Green to Y and Blue to Cb.

About the CMOS Camera Modules

From power up, the camera can take up to 5 seconds to automatically adjust to the lighting conditions. Drastic changes in the environment, such as lights being turned on and off, can induce a similar readjustment time. When using the camera outside, due to the sun's powerful IR emissions, even on relatively cloudy days, it will probably be necessary to use either an IR filter or a neutral density camera filter to decrease the ambient light level. The field of view depends on the lens attached to the camera. It is possible to special order the camera with wider or narrower lenses. Individual lenses can be purchased separately.

The functions provided by the camera board are meant to give the user a toolbox of color vision functions. Actual applications may greatly vary and are left up to the imagination of the user. The ability to change the viewable window, grab color / light statistics and track colors can be interwoven by the host processor to create higher level functionality.

One notable property of the CMOS sensor array is that it returns values between 16 and 240 for each pixel. This effect is noticeable when the camera is tracking colors, getting mean color data or dumping a frame. This limited range on the data does not depend on the mode of the camera and still applies in YCrCb mode.

Serial Commands

The serial communication parameters are as follows:

- 1,200 to 115,200 Baud
- 8 Data bits
- 1 Stop bit
- No Parity
- No Flow Control (Not Xon/Xoff or Hardware)

All commands are sent using visible ASCII characters (123 is 3 bytes “123”). Upon a successful transmission of a command, the ACK string should be returned by the system. If there was a problem in the syntax of the transmission, or if a detectable transfer error occurred, a NCK string is returned. After either an ACK or a NCK, a \r is returned. When a prompt (’\r’ followed by a ‘:’) is returned, it means that the camera is waiting for another command in the idle state. White spaces do matter and are used to separate argument parameters. The \r (ASCII 13 carriage return) is used to end each line and activate each-command. If visible character transmission causes too much overhead, it is possible to use varying degrees of raw data transfer.

See Raw Mode on page 48 for information on configuring ascii vs raw text packets.

Functionally Grouped Command Listing

Buffer Commands

BM	Buffer Mode	30
RF	Read Frame	47

Camera Module Commands

CR	Camera Register	31
CP	Camera Power	32
CT	Camera Type	32

Data Rate Commands

DM	Delay Mode	33
PM	Poll Mode	46
PS	Packet Skip	47
RM	Raw Mode	48
PF	Packet Filter	46
OM	Output Packet Mask	45

Servo Commands

SV	Servo Position	53
SP	Servo Parameters	52
GS	Get Servo Position	36
SM	Servo Mask	51
SO	Servo Output	51

Image Windowing Commands

SF	Send Frame	50
DS	Down Sample	33
VW	Virtual Window	55
FS	Frame Stream	34
HR	HiRes Mode	38
GW	Get Window	38
PD	Pixel Difference	46

Auxiliary I/O Commands

GB	Get Button	35
GI	Get Auxiliary I/O	35
L0(1)	LED control	39

Color Tracking Commands

TC	Track Color	53
TI	Track Inverted	53
TW	Track Window	54
NF	Noise Filter	44
LM	Line Mode	40
GT	Get Tracking Parameters	37
ST	Set Tracking Parameters	52

Histogram Commands

GH	Get Histogram	35
HC	Histogram Config	38
HT	Histogram Track	39

Frame Differencing Commands

FD	Frame Difference	34
DC	Difference Channel	32
LF	Load Frame	39
MD	Mask Difference	44
UD	Upload Difference	55
HD	HiRes Difference	38
LM	Line Mode	40

Color Statistics Commands

GM	Get Mean	36
LM	Line Mode	40

System Level Commands

SD	Sleep Deeply	49
SL	Sleep	50
RS	Reset	49
GV	Get Version	37

Alphabetical Command Listing

\r

This command is used to set the camera board into an idle state. Like all other commands, you should receive the acknowledgment string “ACK” or the not acknowledge string “NCK” on failure. After acknowledging the idle command the camera board waits for further commands, which is shown by the ‘:’ prompt. While in this idle state a \r by itself will return an “ACK” followed by \r and : character prompt. This is how you stop the camera while in streaming mode.

Example of how to check if the camera is alive while in the idle state:

```
:
ACK
:
```

BM active \r

This command sets the mode of the CMUcam’s frame buffer. A value of 0 (default) means that new frames are constantly being pushed into the frame buffer. A value of 1, means that only a single frame remains in the frame buffer. This allows multiple processing calls to be applied to the same frame. Instead of grabbing a new frame, all commands are applied to the current frame in memory. So you could get a histogram on all three channels of the same image and then track a color or call get mean and have these process a single buffered frame. Calling **RF** will then read a new frame into the buffer from the camera. When **BM** is off, **RF** is not required to get new frames.

See RF on page 47 to read a new frame when buffer mode is enabled.

Example of how to track multiple colors using buffer mode:

Processing on an already buffered image is much faster than processing a new image.

```
:BM 1
ACK
:PM 1
ACK
:TC 200 240 0 30 0 30
ACK
T 20 40 10 30 30 50 20 30
:RF
ACK
:TC 0 30 200 240 0 30
ACK
T 30 50 20 40 40 60 22 31
```

CR [reg1 value1 [reg2 value2 ... reg16 value16]]\r

This command sets the **C**amera's internal **R**egister values directly. The register locations and possible settings can be found in the Omnivision CMOS camera documentation. All the data sent to this command should be in decimal visible character form unless the camera has previously been set into raw mode. It is possible to send up to 16 register-value combinations. Previous register settings are not reset between CR calls; however, you may overwrite previous settings. Calling this command with no arguments resets the camera and restores the camera registers to their default state. This command can be used to hard code gain values or manipulate other low level image properties.

See page 25 for more information on YCrCb color space.

Register		Value	Effect
5	Contrast	0-255	
6	Brightness	0-255	
18	Color Mode		
		36	YCrCb Auto White Balance On
		32	YCrCb Auto White Balance Off
		44	RGB Auto White Balance On
		40	*RGB Auto White Balance Off
17	Clock Speed		
		0	*50 fps
		1	26 fps
		2	17 fps
		3	13 fps
		4	11 fps
		5	9 fps
		6	8 fps
		7	7 fps
		8	6 fps
		10	5 fps
19	Auto Exposure		
		32	Auto gain off
		33	*Auto gain on

* indicates the default state

Example of switching into YCrCb mode with White Balance off

```
:CR 18 32
ACK
:
```

CP boolean \r

This command toggles the **C**amera module's **P**ower. A value of 0, puts the camera module into a power down mode. A value of 1 turns the camera back on while maintaining the current camera register values. This should be used in situations where battery life needs to be extended, while the camera is not actively processing image data. Images in the frame buffer may become corrupt when the camera is powered down.

See SL and SD on pages 49 and 50 to decrease camera power consumption even more.

CT boolean \r

This command toggles the **C**amera **T**ype while the camera is in slave mode. Since the CMUcam2 can not determine the type of the camera without communicating with the module, it is not possible for it to auto-detect the camera type in slave mode. A value of 0, sets the CMUcam2 into ov6620 mode. A value of 1 sets it into ov7620 mode. The default slave mode startup value assumes the ov6620.

See slave mode on page 22.

DC value \r

This command sets the **C**hannel that is used for frame **D**ifferencing commands. A value of 0, sets the frame differencing commands LF and FD to use the red (Cr) channel. A value of 1 (default) sets them to use the green (Y) channel, and 2 sets them to use the blue (Cb) channel.

See LF and FD on page 39 and page 34.

DM value \r

This command sets the **D**elay **M**ode which controls the delay between characters that are transmitted over the serial port. This can give slower processors the time they need to handle serial data. The value should be set between 0 and 255. A value of 0 (default) has no delay and 255 sets the maximum delay. Each delay unit is equal to the transfer time of one bit at the current baud rate.

DS x_factor y_factor \r

This command allows **D**own **S**ampling of the image being processed. An x_factor of 1 (default) means that there is no change in horizontal resolution. An x_factor of 2, means that the horizontal resolution is effectively halved. So all commands, like send frame and track color, will operate at this lower down sampled resolution. This gives you some speed increase and reduces the amount of data sent in the send frame and bitmap linemodes without clipping the image like virtual windowing would. Similarly, the y_factor independently controls the vertical resolution. (Increasing the y_factor downsampling gives more of a speed increase than changing the x_factor.) The virtual window is reset to the full size whenever this command is called.

Example of down sampling the resolution by a factor of 2 on both the horizontal and vertical dimension.

```
:DS 2 2
ACK
:GM
ACK
S 89 90 67 5 6 3
S 89 91 67 5 6 2
```

FD threshold \r

See LF on page 39 to load a new baseline frame to difference off of.

This command calls **F**rame **D**ifferencing against the last loaded frame using the **LF** command. It returns a type T packet containing the middle mass, bounding box, pixel count and confidence of any change since the previously loaded frame. It does this by calculating the average color intensity of an 8x8 grid of 64 regions on the image and comparing those plus or minus the user assigned *threshold*. So the larger the threshold, the less sensitive the camera will be towards differences in the image. Usually values between 5 and 20 yield good results. (In high resolution mode a 16x16 grid is used with 256 regions.)

See MD on page 44 to see how to reduce motion noise.

FS boolean \r

This command sets the **F**rame **S**treaming mode of the camera. A value of 1, enables frame streaming, while a 0 (default) disables it. When frame streaming is active, a send frame command will continuously send frames back to back out the serial connection.

See SF on page 50.

GB \r

See Demo Mode on page 15.

This command **G**ets a **B**utton press if one has been detected. This command returns either a 1 or a 0. If a 1 is returned, this means that the button was pressed sometime since the last call to Get Button. If a 0 is returned, then no button press was detected.

GH <channel> \r

See HC and HT commands on pages 38 and 39.

This command **G**ets a **H**istogram of the *channel* specified by the user. The histogram contains 28 bins each holding the number of pixels that occurred within that bin's range of color values. So bin 0 on channel 0 would contain the number of red pixels that were between 16 and 23 in value. If no arguments are given, get histogram uses the last channel passed to get histogram. If get histogram is first called with no arguments, the green channel is used. The value returned in each bin is the number of pixels in that bin divided by the total number of pixels times 256 and capped at 255.

GI \r

This command **G**ets the auxiliary I/O **I**nput values. When get inputs is called, a byte is returned containing the values of the auxiliary IO pins. This can be used to read digital inputs connected to the auxiliary I/O port. The aux I/O pins are internally lightly pulled high. See page 22 for pin numbering. Note that the pins are pulled up internally by the processor.

Example of how to read the auxiliary I/O pins. (in this case, pins 0 and 1 are high, while pins 2 and 3 are low).

```
:GI
3
ACK
:
```

GM \r

This command will **G**et the **M**ean color value in the current image. If, optionally, a subregion of the image is selected via virtual windowing, this function will only operate on the selected region. The mean values will be between 16 and 240 due to the limits of each color channel on the CMOS camera. It will also return a measure of the average absolute deviation of color found in that region. The mean together with the deviation can be a useful tool for automated tracking or detecting change in a scene. In YCrCb mode RGB maps to CrYCb.

See page 45 to see how the OM command can create a custom S Packet.

This command returns a Type S data packet that by default has the following parameters:

S Rmean Gmean Bmean Rdeviation Gdeviation Bdeviation\r

Example of how to grab the mean color of the entire window:

```
:SW 1 1 40 143
ACK
:GM
ACK
S 89 90 67 5 6 3
S 89 91 67 5 6 2
```

GS servo \r

This command will **G**et the last position that was sent to the **S**ervos.

See SV command on page 53.

Example of how to use get servo:

```
:GS 1
ACK
128
:
```

GT \r

This command **G**ets the current **T**rack color values. This is a useful way to see what color values track window is using.

This example shows how to get the current tracking values:

```
:TW
 ACK
 T 12 34 ....
:GT
ACK
200 16 16 240 20 20
:
```

GV \r

This command **G**ets the current **V**ersion of the firmware and camera module version from the camera. It returns an ACK followed by the firmware version string. c6 means that it detects an OV6620, while c7 means that it detected an OV7620.

Example of how to ask for the firmware version and camera type:

```
:GV
ACK
CMUcam2 v1.00 c6
```

GW \r

This command **G**ets the current virtual **W**indowing values. This command allows you to confirm your current window configuration. It returns the x1, y1, x2 and y2 values that bound the current window.

HC #_of_bins scale \r

See GH on page 35 to see how to get histograms.

This command lets you **C**onfigure the **H**istogram settings. The first parameter takes one of three possible values. A value of 0 (default) will cause GH to output 28 bins. A value of 1 will generate 14 bins and a value of 2 will generate 7 bins. The scale parameter (default 0) allows you to better examine bins with smaller counts. Bin values are scaled by 2^{scale} where scale is the second parameter of the command.

#_of_bins

Input	Bins
0	28
1	14
2	7

HD boolean \r

See LF and FD on pages 39 and 34 to see how to use the more basic frame differencing commands.

This command enables or disables **H**iRes frame **D**ifferencing. A value of 0 (default) disables the high resolution frame differencing mode, while a value of 1 enables it. When enabled, frame differencing will operate at 16x16 instead of 8x8. The captured image is still stored internally at 8x8. The extra resolution is achieved by doing 4 smaller comparisons against each internally stored pixel. This will only yield good results when the background image is relatively smooth, or has a uniform color.

HR state \r

This sets the camera into **HiR**es mode. This is only available using the OV6620 camera module. A *state* value of 0 (default) gives you the standard 88x143, while 1 gives you 176x287. HiRes mode truncates the image to 176x255 for tracking so that the value does not overflow 8 bits.

HT boolean \r

See GH on page 35 to see how to get a histogram.

This command enables or disables **H**istogram **T**racking. When histogram tracking is enabled, only values that are within the color tracking bounds will be displayed in the histograms. This allows you to select exact color ranges giving you more detail, and ignoring any other background influences. A value of 0 (default) will disable histogram tracking, while a value of 1 will enable it. Note that the tracking noise filter applies just like it does with the TC and TW commands.

L0 boolean \r
L1 boolean \r

These commands enable and disable the two tracking **L**EDs. A value of 0 will turn the LED off, while a value of 1 will turn it on. A value of 2 (default) will leave the LED in automatic mode. In this mode, LED 1 turns on when the camera confidently detects an object while tracking and provides feedback during a send frame. In automatic mode, LED 0 does nothing, so it can be manually set.

LF \r

See FD on page 34.

This command **L**oads a new **F**rame into the processor's memory to be differenced from. This does not have anything to do with the camera's frame buffer. It simply loads a baseline image for motion differencing and motion tracking.

LM type mode \r

This command enables **L**ine **M**ode which transmits more detailed data about the image. It adds prefix data onto either **T** or **S** packets. This mode is intended for users who wish to do more complex image processing on less reduced data. Due to the higher rate of data, this may not be suitable for many slower microcontrollers. These are the different types and modes that line mode applies to different processing functions:

Type	Mode	Effected Command	Description
0	0	TC TW	Default where line mode is disabled
0	1	TC TW	Sends a binary image of the pixels being tracked
0	2	TC TW	Sends the Mean, Min, Max, confidence and count for every horizontal line of the tracked image.
1	0	GM	Default where line mode is disabled
1	1	GM	Sends the mean values for every line in the image
1	2	GM	Sends the mean values and the deviations for every line being tracked in the image
2	0	FD	Default where line mode is disabled
2	1	FD	Returns a bitmap of tracked pixels much like type 0 mode 0 of track color
2	2	FD	Sends the difference between the current image pixel value and the stored image. This gives you delta frame differenced images.
2	3	LF FD	This gives you the actual averaged value for each element in a differenced frame. It also returns these values when you load in a new frame. This can be used to give a very high speed gray scale low resolution stream of images.

Note, that the “mode” of each “type” of linemode can be controlled independently.

Line Mode Type 0: Track Color

Mode 1: Bitmap of tracked region

When the linemode type is 0 and the mode is set to 1, TC or TW will send a binary bitmap of the image as it is being processed. It will start this bitmap with an 0xAA flag value (hex value AA not in human readable form) followed by the Xsize and Ysize of the binary image. The value 0xAA will not occur in the data stream. This is followed by bytes each of which contains the binary value of 8 pixels being streamed from the top-left to the bottom-right of the image. The bits for each row are padded with zeros to fill an integral number of bytes. The binary bitmap is terminated by two 0xAA's. This is then followed by the normally expected standard **T** data packet.

See TC on page 53.

Example of TC with line mode on:

```
:LM 0 1
ACK
:TC
ACK
(raw data: AA Xsize Ysize XX XX …. XX XX XX AA AA) T 55 90 45 72 65 106 18 51
(raw data: AA Xsize Ysize XX XX …. XX XX XX AA AA) T 55 90 46 72 65 106 18 52
```

Mode 2: Per row statistics in the tracked region

When the linemode type is 0 and the mode is set to 2, TC or TW will send various statistics about each row that is being tracked. It sends the minimium x value, the maximium x value, the average x value, the count of tracked pixels on that line and the confidence. This can be especially useful for line following applications since you can essentially get a trace of the middle of the line. Like other linemode options, this new data is sent as a prefixed packet. The packet starts with an 0xFE, followed by the number of rows (the y-size) that it will send. The packet will then contain, the xLineMean, xLineMin, xLineMax, line pixel count, and line confidence for each row. These will all be sent as raw values. The packet terminates with a 0xFD followed by a normal T packet.

See OM on page 45 to see how to mask these line mode data packets.

0xFE y-size xLineMean xLineMin xLineMax LineCount Conf ... 0xFD Tpacket

Line Mode Type 1: Get Mean

Mode 1: Per line statistics

See GM on page 36.

When the linemode type is 1 and the mode is set to 1, GM will send a raw (not human readable) mean value of every line being processed. These packets start with an 0xFE. The data is sent in the following raw format rLineMean, gLineMean, bLineMean, and terminate with an 0xFD.

0xFE Rmean Gmean Bmean ... 0xFD Mpacket

Example of GM with line mode on

```
:LM 1 1
ACK
:GM
ACK
(raw data: FE XX XX XX … XX XX XX FD) M 45 56 34 10 15 8
(raw data: FE XX XX XX … XX XX XX FD) M 45 56 34 10 15 8
```

See OM on page 45 to see how to mask these line mode data packets.

Mode 2: More per line statistics

When the linemode type is 1 and the mode is set to 2, GM will send a raw (not human readable) mean value and deviation for every line being processed. These packets are started with an 0xFE. The data is sent in the following raw format rLineMean, gLineMean, bLineMean, rDeviation, gDeviation, bDeviation and terminate with an 0xFD.

0xFE Rmean Gmean Bmean Rdev Gdev Bdev ... 0xFD Mpacket

Line Mode Type 2: Frame Differencing

Mode 1: Bitmap for tracked pixels

See FD on page 34.

When the linemode type is 2 and the mode is set to 1, FD will send a binary bitmap of the image as it is being processed. It will start this bitmap with an 0xAA flag value (hex value AA not in human readable form) followed by the Xsize and Ysize of the binary image. The value 0xAA will not occur in the data stream. This is followed by bytes each of which contains the binary value of 8 (or 16) pixels being streamed from the top-left to the bottom-right of the image. The binary bitmap is terminated by two 0xAA's. This is then followed by the normally expected standard **T** data packet.

Example of TC with line mode on:

```
:LM 2 1
ACK
:FD 10
ACK
(raw data: AA XX XX XX … XX XX XX AA AA) T 5 10 4 9 8 100
(raw data: AA XX XX XX … XX XX XX AA AA) T 5 10 4 9 8 100
```

Mode 2: Deltas between reference frame

When the linemode type is 2 and the mode is set to 2, FD will send the values of the differences between the current image and the original saved frame. The packet starts with 0xFC followed by the xSize and ySize of the image buffer that is to be sent. A single value for each pixel is transmitted and the packet ends with an 0xFD. The delta values are capped at +/- 112 with 128 added to the delta, so 128 means zero difference. This forces the values to remain in the 16-240 range.

```
:LM 2 2
ACK
:FD 10
ACK
(raw data: FC xSize ySize  XX XX XX … XX XX XX FD)
(raw data: FC xSize ySize  XX XX XX … XX XX XX FD)
```

Mode 3: Deltas between reference frame

When the linemode type is 2 and the mode is set to 3, LF and FD will send a binary bitmap of the internally stored image that they are operating on. This image is stored in the same format as mode 2 of frame differencing.

MD threshold \r

See FD on page 34.

This command is almost identical to FD except that it **M**asks the first frame it **D**ifferences on. Any motion detected on the first frame is masked out, so that areas with high amounts of noise are ignored. Basically, if you call frame differencing and there is always an area of the frame that is moving, then MD will mask out that portion of the image so subsequent calls to FD will ignore that portion of the image. Calling the LF command will clear any masked pixels.

NF threshold \r

This command controls the **N**oise **F**ilter setting. It accepts a value that determines how many consecutive active pixels before the current pixel are required before the pixel should be detected (default 2). The range is between 0 and 255.

Example of how to turn off noise filtering:

```
:NF 0
ACK
:
```

OM packet mask \r

This command sets the **O**utput **M**ask for various packets. The first argument sets the type of packet:

#	**Tracking Type**	**Packet**
0	Track Color	T
1	Get Mean	S
2	Frame Difference	T
3	Non-tracked packets*	T
4	Additional Count Information**	T, H
5	Track Color Line Mode 2	T
6	Get Mean Line Modes 1 and 2	S

The *mask* should be a single byte that represents the bitwise mask of the tracking packet. So a value 255 would allow all the parameters to be printed, while a value of 3 would only allow the first two parameters to be printed. Each mask for each paket type is stored separately and remains set until the camera is reset.

*Non-Tracked packets are packets that are printed when the object being tracked is not detected. If this is set to 0, then no packet is printed when the object is not found. If this is set to 1, then just a "T 0" is sent when no object is found. If this is set to 2 (default) then the packet is identical to a tracked packet of that type.

**The additional count information flag lets you get access to the full 16 bit count values for color tracking or histogramming. A value of 0 (default) disables the 16 bit values. A value of 1, adds the 16 bit count of tracked pixels in 2 separate bytes, the first for the LSB and the second for the MSB. A value of 2 will add a 16 bit count of all pixels used to generate a histogram as the first two bytes following the H in the histogram packet. A value of 3, enables both modes simultaneously.

Example of how to only show Mx and My in a T packet:

```
:OM 0 3
ACK
:TC 200 230 0 30 0 30
T 23 45
```

PD boolean \r

See TI on page 53 to find out how to use inverse tracking for better edge following.

This command enables the **P**ixel **D**ifference mode. By default, the mode is off. A value of 1 causes the difference between the current pixel and the previous pixel to be used by all processing commands instead of the original pixel value. This essentially does a horizontal edge detecting convolution on the image. So the intensity of the remaining lines in each channel is proportional to the sharpness of an edge found in that channel. The best way to understand this command is to try enabling pixel differencing and try sending a frame. Notice what types of lines appear stronger than others. You can then track these edges based on their intensity using track color etc. The difference values are capped at +/- 112 with 128 added to each delta so a value of 128 indicates a 0 difference. This forces the values to remain between 16 and 240. This command applies to all commands.

PF boolean \r

This command enables the **P**acket **F**iltering mode. By default, the mode is off. A value of 1 makes it so that only the first empty packet when a tracked object disappears from the screen is displayed. No packets will be transmitted until the object returns into view. This command can help in situations where empty packets may unnecessarily tax the host processor.

PM mode \r

This command puts the board into **P**oll **M**ode. Setting the mode parameter to 1 engages poll mode while 0 (default) turns it off. When poll mode is set to 0, a continous stream of packets is returned from a processing function. When poll mode is set to a value of 1, only one packet is returned when an image processing function is called. If mode is set to a value of 2, then poll mode will wait until an object is tracked and then return. This could be useful if you would like to rapidly change parameters or if you have a slow processor that can’t keep up with a given frame rate.

Example of how to get one packet at a time:

```
:PM 1
ACK
:TC 50 20 90 130 70 255
ACK
C 38 82 53 128 35 98
:
```

PS number \r

This command controls if **P**ackets should be **S**kipped or not. The default value is 0, which means that all packets will be transmitted. A value of 1 means that every other packet will be skipped. A value of 2 means that only every second packet will be displayed etc. This is useful if you need to slow down the data rate so that your processor can keep up with the data stream when poll mode is enabled.

RF \r

See BM on page 30.

This command **R**eads a new **F**rame into the buffer. This should only be used to get new data when using buffer mode (**BM**). The frame buffer is what allows multiple pass image processing on a single frame. While in buffer mode, you are constantly reprocessing the same frame until read frame is called. Under normal non-buffer mode operation, a new frame is loaded right before a processing function is called.

RM bit_flags \r

This command is used to engage the **R**aw serial transfer **M**ode. It reads the bit values of the first 3 (lsb) bits to configure settings. All bits cleared sets the default visible ASCII mode. If bit 0 is set, then all output from the camera is in raw byte packets. The format of the data packets will be changed so as not to include spaces or be formatted as readable ASCII text. Instead you will receive a 255 valued byte at the beginning of each packet, the packet identifying character (i.e. C for a color packet) and finally the packet data.There is no \r sent after each packet, so you must use the 255 to synchronize the incoming data. Any 255 valued bytes that may be sent as part of the packet are set to 254 to avoid confusion. If bit 1 is set, the “ACK\r” and “NCK\r” confirmations are not sent. If bit 2 is set, input will be read as raw byte values, too. In this mode, after the two command byte values are sent, send 1 byte telling how many arguments are to follow. (i.e. DF followed by the raw byte value 0 for no arguments) No \r character is required.

bit_flags = B2 B1 B0

B0	Output from the camera is in raw bytes
B1	“ACK\r” and “NCK\r” confirmations are suppressed
B2	Input to the camera is in raw bytes

Example of the new packet for Track Color with Raw Mode output only:

```
:RM 1
ACK
:TC 50 100 30 90 0 30
ACK
T>#$%KFDSAG@#$
```

RS \r

This command **R**e**S**ets the vision board. Note, on reset the first character is a \r. Also keep in mind that all register values are reset to their default state.

Example of how to reset the camera:

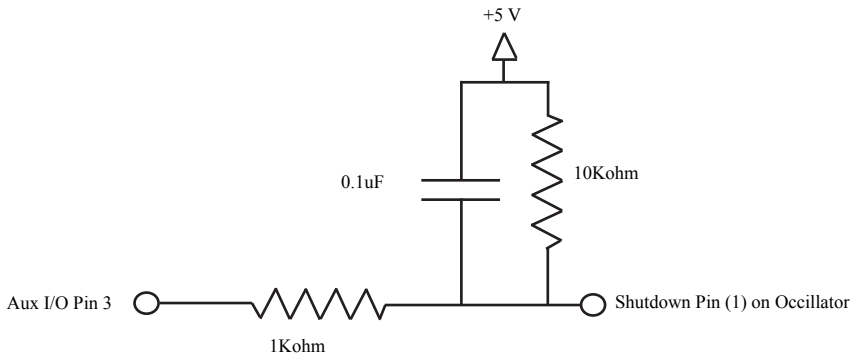
```
:RS
ACK

CMUcam2 v1.0 c6
:
```

SD \r

This command **S**leeps the camera **D**eeply to save power. This command puts the processor to sleep just as the SL command does and aditionally uses one of the auxilary I/O pins to sleep the oscillator. Wakeup from this mode is achieved by sending any character to the module, typically '\r'. The oscillator needs to shut off slightly later than the processor to ensure that that processor powers down correctly. To achieve this delay, it is neccessary to add a pullup resistor on the enable line of the oscillator and then have a resistor and capacitor in series with each other before being connected to auxiliary I/O pin 3. You will need to connect 1K series resistor between the oscillator's enable pin and the aux IO pin 3. You then need to connect a 10K resistor in parallel with a 0.1uF capacitor between the enable pin on the oscillator and +5 volts. See diagram below.

See SL on page 50, for a faster, more basic sleep command.



SF [channel] \r

See FS on page 34, to find out how to stream frames.
See DS on page 33, to find out how to reduce data sent by send frame.

This command will **S**end a **F**rame out the serial port to a computer. This is the only command that will by default only return a non-visible ASCII character packet. It dumps a type F packet that consists of the raw video data row by row with a frame synchronize byte and a column synchronize byte. (This data can be read and displayed by the CMUcam2GUI java application.) To get the correct aspect ratio, double each column of pixels. Since the image is being read from a buffer, the image resolution is not dependent on baud rate. The baud rate just controls how fast the image will be transmitted. Optionally, a channel (0-2) can be added to the command which causes send frame to only send that channel. This will effectively transmit one third of the data.

Type F data packet format
1 - new frame followed by X size and Y size
2- new col
3 - end of frame
RGB (CrYCb) ranges from 16-240
1 xSize ySize 2 r g b r g b ... r g b r g b 2 r g b r g b ... r g b r g b 3

SL active \r

For greater power saving see the SD and CP commands on pages 49 and 32.

This command enables **SL**eep mode by putting the processor to sleep. Sleep mode can be used when the camera is not needed in order to save power. Sending any character wakes the camera back up after a delay of up to 10ms. It is best to use '\r' to wake the camera up since this will ensure that no unforeseen command gets executed. Sleeping will disable the servo outputs.

SM bit_flags \r

This command sets the **S**ervo **M**ask on the CMUcam. The servo mask controls which automatic servo axes are active and which ones should report their values at the end of tracking packets. Pan and Tilt enable / disable turn off the respective automatic servo function while tracking. The servo reporting is added after all of the normal outputs in the Tracking packet, but before the final "\r". Note that automatic control only operates with 'T' packets returned by TC and TW commands.

bit_flags = B3 B2 B1 B0

B0	Pan Control Enable
B1	Tilt Control Enable
B2	Pan Report Enable
B3	Tilt Report Enable

Example of how to enable both pan and tilt automatic servoing and both pan and tilt reporting. In this case, since it doesn't see the object, the servos stay at position 128:

```
:SM 15
ACK
:TC
ACK
T 0 0 0 0 0 0 0 0 128 128
```

SO servo_number level \r

This command sets a **S**ervo **O**utput on the CMUcam to be either a constant high or low value. This essentially converts the servo outputs to be standard TTL digital outputs. The servo number (0-4) selects which servo you want to control, and a level value of either 1 or 0 switches between 5 and 0 volts. If a servo is connected and the output is set to 0, the servo is effectively turned off.

SP [pan_range_far pan_range_near pan_step
tilt_range_far tilt_range_near tilt_step] \r

This command sets the **S**ervo **P**arameters that are used by the automatic tracking control law. Changing these values can help you tune your tracking for a particular servo setup. The automatic servoing uses a two stage "bang-bang" control law. When the pixel value is greater than the "far" range, the related servo will move by the step amount. When the pixel value is between the near and far range, the servo will move by half of the step amount. Any value smaller than the near value is part of the dead zone and will not trigger any servo motion.

See page 21 for pan tilt direction jumpers.

See the SM command on page 51.

Variable	Description	Default
pan_range_far	Pixel distance needed to do a large pan step	16
pan_range_near	Pixel distance needed to do a small pan step	8
pan_step	Servo position change of a long pan step	5
tilt_range_far	Pixel distance needed to do a large tilt step	30
tilt_range_near	Pixel distance needed to do a small tilt step	15
tilt_step	Servo position change of a long tilt step	5

ST Rmin Rmax Gmin Gmax Bmin Bmax \r

This command allows you to **S**et **T**racking parameters without actually calling track color. These values can then be stored until you might call TC with no arguments later.

Example of how to use ST:

```
:ST 200 0 0 250 20 20
ACK
:TC
ACK
T 6 55 2 40 12 60 10 70
T 6 55 2 41 12 61 11 70
```

SV servo position \r

See SO on page 51 to learn how to disable the servos.

This command lets you set the position of one of five **S**er**V**os. The servos have an active region of between 46 and 210. A value of 128 is the center and generates a 1500 us pulse. The pulse increments by 8.68us and covers a range from 400 us to 1820 us.

Example to set servo 1 to position 200:

```
:SV 1 200
ACK
:
```

TC [Rmin Rmax Gmin Gmax Bmin Bmax] \r

See page 45 to see how the OM command can create a custom S Packet.

Using VW on page 55 to decrease the vertical resolution will allow 50 fps tracking.

This command begins to **T**rack a **C**olor . It takes in the minimum and maximum RGB (CrYCb) values and outputs a type T packet. This packet by default returns the middle mass x and y coordinates, the bounding box, the number of pixels tracked, and a confidence value. The packet can be masked using the **OM** output mask function. Remember that the color values from the CMOS camera will range from between 16 and 240. If TC is called with no arguments it will track with the precious set of tracking parameters.

Default Type T packet
T mx my x1 y1 x2 y2 pixels confidence\r

Example of how to Track a Color with the default mode parameters:

```
:TC 130 255 0 0 30 30
ACK
T 50 80 38 82 53 128 35 98
T 52 81 38 82 53 128 35 98
```

TI boolean \r

This command activates **T**rack **I**nverted mode. When track inverted mode is enabled, the camera will track colors that are outside of the user defined color range instead of inside. This is good for either tracking edges, or tracking any object that shows up against a homogenous background.

TW \r

This command will **T**rack the color found in the central region of the current **W**indow. After the color in the current window is grabbed, the track color function is called with those parameters and on the full image window. This can be useful for locking onto and tracking an object held in front of the camera. Since it actually calls track color, it returns the same type T track packet. Note, the current virtual window setting will only be used for grabbing the color to track and then the window will return to its maximum size.

The following internal steps are performed when "TW" is called:

1. Shrink the window to 1/2 the size (in each dimension) of the current window centered on the current window. (sw 30 54 50 90)

2. Call the get mean command but do not display the output. (gm)

3. Restore the window to the full image size. (sw 1 1 88 143)

4. Set the min and max value for each color channel to be the mean for that channel +/- 30.

Example of how to use Track Window:

```
:TW
ACK
T 6 55 2 40 12 60 10 70
T 6 55 2 41 12 61 11 70
```

UD <64 raw bytes> \r

This command allows you to **U**pload a **D**ifference frame buffer. The command waits for 64 raw byte values that fill up the 8 by 8 internal frame difference buffer. A '\r' cancels the transfer. A value of 0 indicates that the region should be masked and not detect motion. With this command in combination with line mode type 2, it is possible to download and upload different reference frames for frame differencing.

See LM on page 40 for instructions on downloading a difference buffer.

VW [x y x2 y2] \r

Do not try VW 0 0 88 144, this is outside of the 1 1 88 143 bounds.

This command sets the **V**irtual **W**indow size of the camera. It accepts the x and y Cartesian coordinates of the upper left corner (1,1) followed by the lower right of the window you wish to set. The origin is located at the upper left of the field of view. **VW** can be called before an image processing command to constrain the field of view. Without arguments it returns to the default full window size of for the current combination of camera type, downsampling and resolution mode. Note that reducing the vertical window size can be used to speed up processing time to achieve higher frame rates with the track color command. 50 fps can be achieved with a vertical dimension of 65 or less.

See GW on page 37 to find out how to check your window configuration.

Example of setting the camera to select the mid portion of the view:

```
:VW 35 65 45 75
ACK
```

Data Packet Description

When raw mode is disabled all output data packets are in ASCII viewable format except for the F frame and prefix packets.

ACK

This is the standard acknowledge string that indicates that the command was received and fits a known format.

NCK

This is the failure string that is sent when an error occurred. The only time this should be sent when an error has not occurred is during binary data packets.

Type **F** data packet format:

This packet does NOT begin with an "F" and it only contains raw data.

1 2 r g b r g b ... r g b r g b 2 r g b r g b ... r g b r g b 3

1 - new frame 2 - new row 3 - end of frame
RGB (CrYCb) ranges from 16 - 240
RGB (CrYCb) represents two pixels color values. Each pixel shares the red and blue.
176 cols of R G B (Cr Y Cb) packets (forms 352 pixels)
144 rows
To display the correct aspect ratio, double each column so that your final image is 352x144

Type **H** packet:

H bin0 bin1 bin2 bin3 ... bin26 bin27 \r

This is the return packet from calling get histogram (**GH)**. Each bin is an 8 bit value that represents the number of pixels that fell within a set range of values on a user selected channel of the image.

Bin0 – number of pixels between 16 and 23
Bin1 – number of pixels between 24 and 31
.
.
.
Bin27 – number of pixels between 232 and 240

Type **T** packet:

T mx my x1 y1 x2 y2 pixels confidence\r

This is the return packet from a color tracking or frame differencing command.

mx - The middle of mass x value
m*y* - The middle of mass y value
x1 - The left most corner's x value
y1 - The left most corner's y value
x2 - The right most corner's x value
y2 -The right most corner's y value
pixels –Number of Pixels in the tracked region, scaled and capped at 255: (pixels+4)/8
confidence -The (# of pixels / area)*256 of the bounded rectangle and capped at 255

Type **S** data packet format:

S Rmean Gmean Bmean Rdeviation Gdeviation Bdeviation \r

This is a statistic packet that gives information about the camera's view

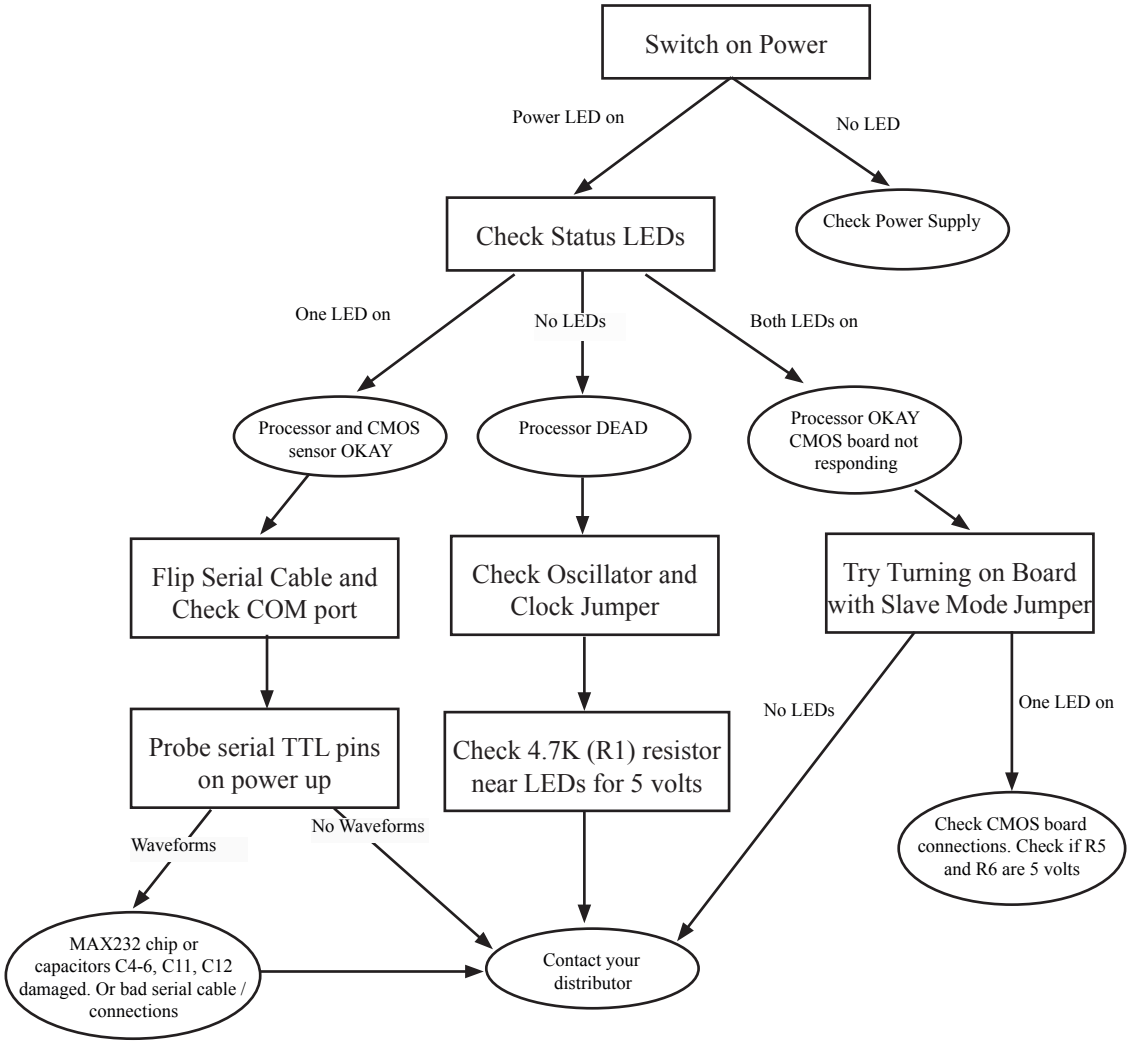
Rmean - the mean Red or Cr (approximates r-g) value in the current window
Gmean - the mean Green or Y (approximates intensity) value found in the current window
Bmean - the mean Blue or Cb (approximates b-g) found in the current window
R*deviation* - the *deviation of red or Cr found in the current window
G*deviation*- the *deviation of green or Y found in the current window
B*deviation*- the *deviation of blue or Cb found in the current window

*deviation: The mean of the absolute difference between the pixels and the region mean.

Troubleshooting

Diagnostic Fault Tree

The diagram below shows a few quick steps to help you diagnose a hardware problem with the CMUcam2.



General

In Demo Mode, the light turns on for a second and then everything stops:

When both the camera and servo are active, the power required is greater. Try using a battery or voltage source rated at a higher current.

The power LED does not glow:

The board either has a fault, or your power supply is not generating enough power. Check the power supply and look over all of the soldier connections. Try unplugging all of the cables except power and turn it on again.

I get garbage output from the camera:

Try turning the camera off and unplugging it for 10 seconds. Then plug it back in and try again. Also, make sure that the baud rate is set correctly.

I get wavy lines or a distorted black and white image when I call dumpframe:

This is most likely due to power. Make sure that you have a high enough voltage and that you are getting a clean signal. Running the camera off of fresh batteries (not an AC adaptor) is a good way to test if this is the problem.

See page 46 for poll mode and page 33 for delay mode.

My processor can not keep up with the serial data stream:

Try running the camera in poll mode and setting a delay mode value.

I don't seem to get any serial data:

Make sure that the serial cable is connected on the CMUcam side correctly. If in doubt, try reversing it.

Why does VW keep giving me a NCK?

Make sure you are within the VW 1 1 88 143 bounds.

I see the CMUcam startup message, but then nothing happens:

Check to make sure the transmit line on your serial cable is connected correctly.

CMUcam2 GUI

When I run java I get: Exception in thread "main" java.lang.NoClassDefFoundError CMUcam2GUI:

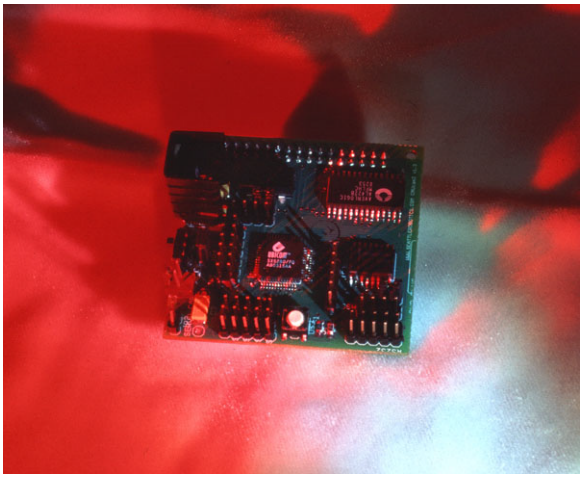
Chances are you are not in the CMUcam2GUI directory. Type "dir" at the command line prompt and make sure that you see the CMUcam2GUI.class file. Also check to make sure an old version of Quicktime did not set your CLASSPATH variable (there should be no CLASSPATH variable in new versions of java).

I see CMUcam2GUI.java but I don't see the CMUcam2GUI.class file:

You should download a new copy of the GUI, because the .class files should be included. If you really need to recompile them, type "javac *.java" .

I get: 'java' is not recognized as an internal or external command, operable program or batch file:

This means that java is not correctly installed in your path. Try re-installing java and reading Sun's installation documentation.



3rd Party Software Information

Java

The CMUcam2 GUI should not need to be recompiled!

The CMUcam2 GUI requires java's JRE version 1.4.0 or newer. The latest version of java can be downloaded for free at http://java.sun.com. JRE stands for java runtime environment and contains all that is needed to run a java program. JDK stants for java development kit and does everything that the JRE does, plus it allows you to compile java programs into byte code. Since the CMUcam2 GUI is given to you already compiled, you should be able to run it on any type of computer without having to recompile.

Terminal Emulation Programs

The following are free terminal emulation programs that can be found as shareware on the internet:

Windows:
- HyperTerm - built into windows, but tends to be confusing
- TeraTerm - Fast and easy to use

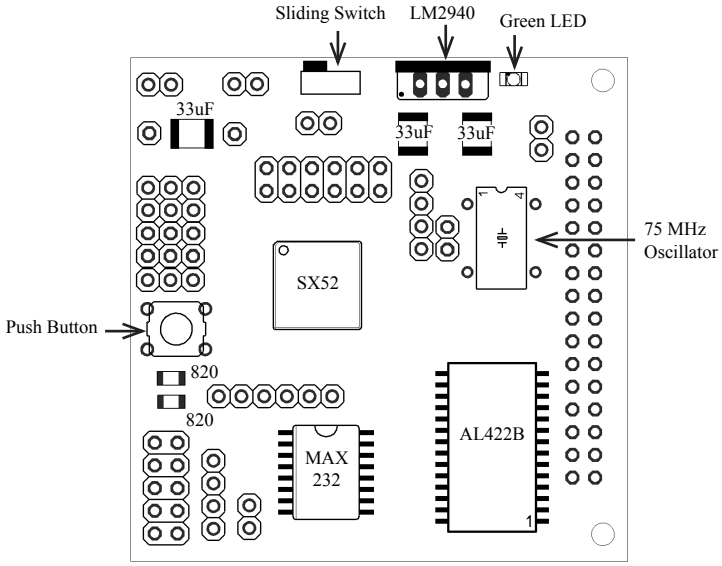
Macintosh OS 7,8,9:
- Zterm - fast, free, easy to use

Mac OS X
- Port Term

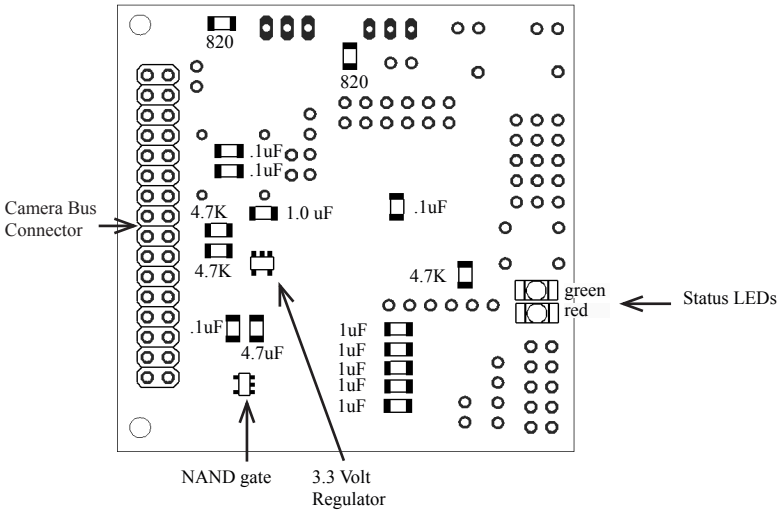
Unix / Linux:
- Minicom - standard easy to use com program
  - alt-a adds line feeds to \r
  - alt-e turns on local echo
  - alt-s lets you configure the serial port

Components and Schematic

Top Components



Bottom Components

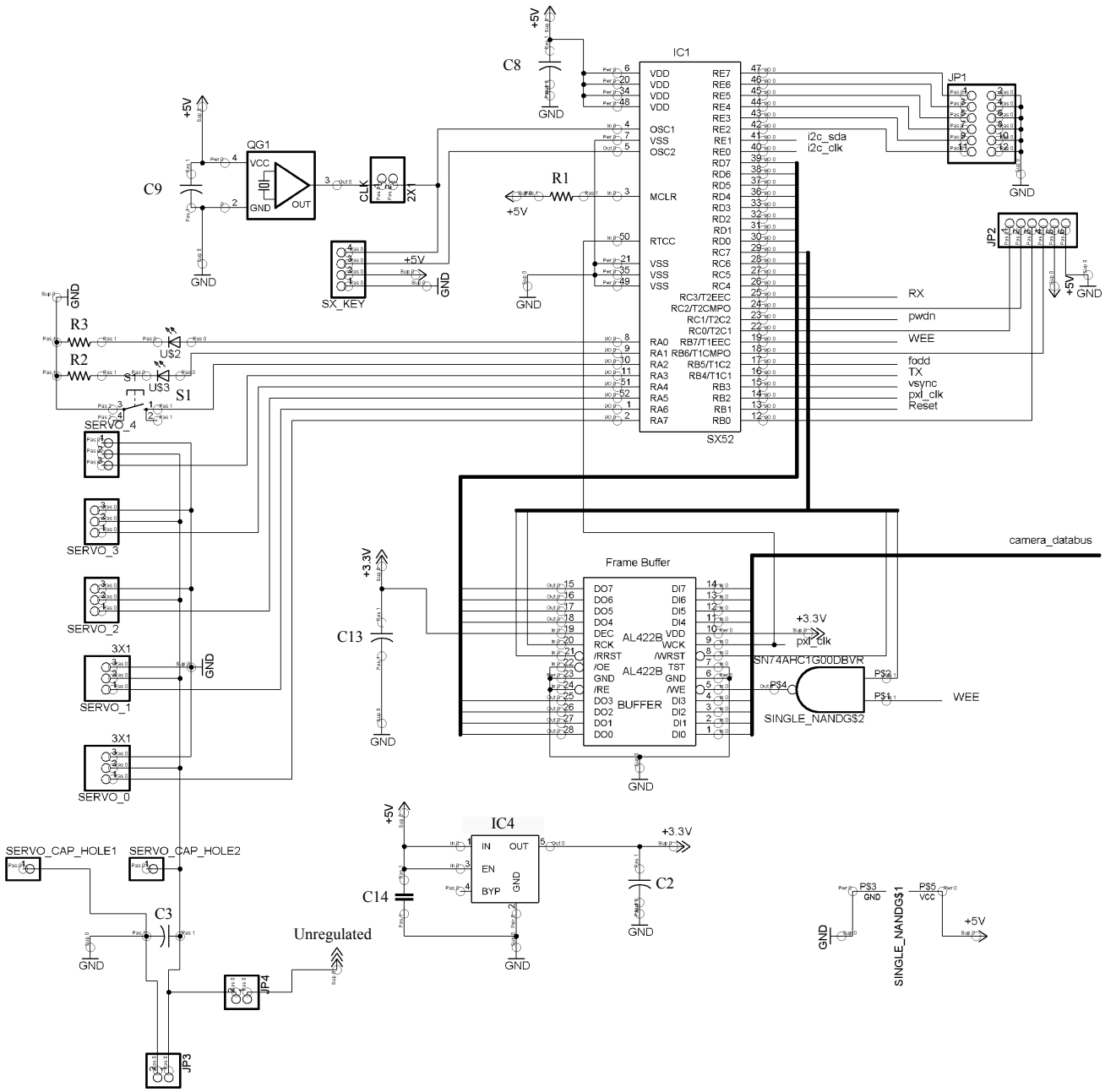


This image is a bottom view of the components. Note that the camera connector is on the left side.

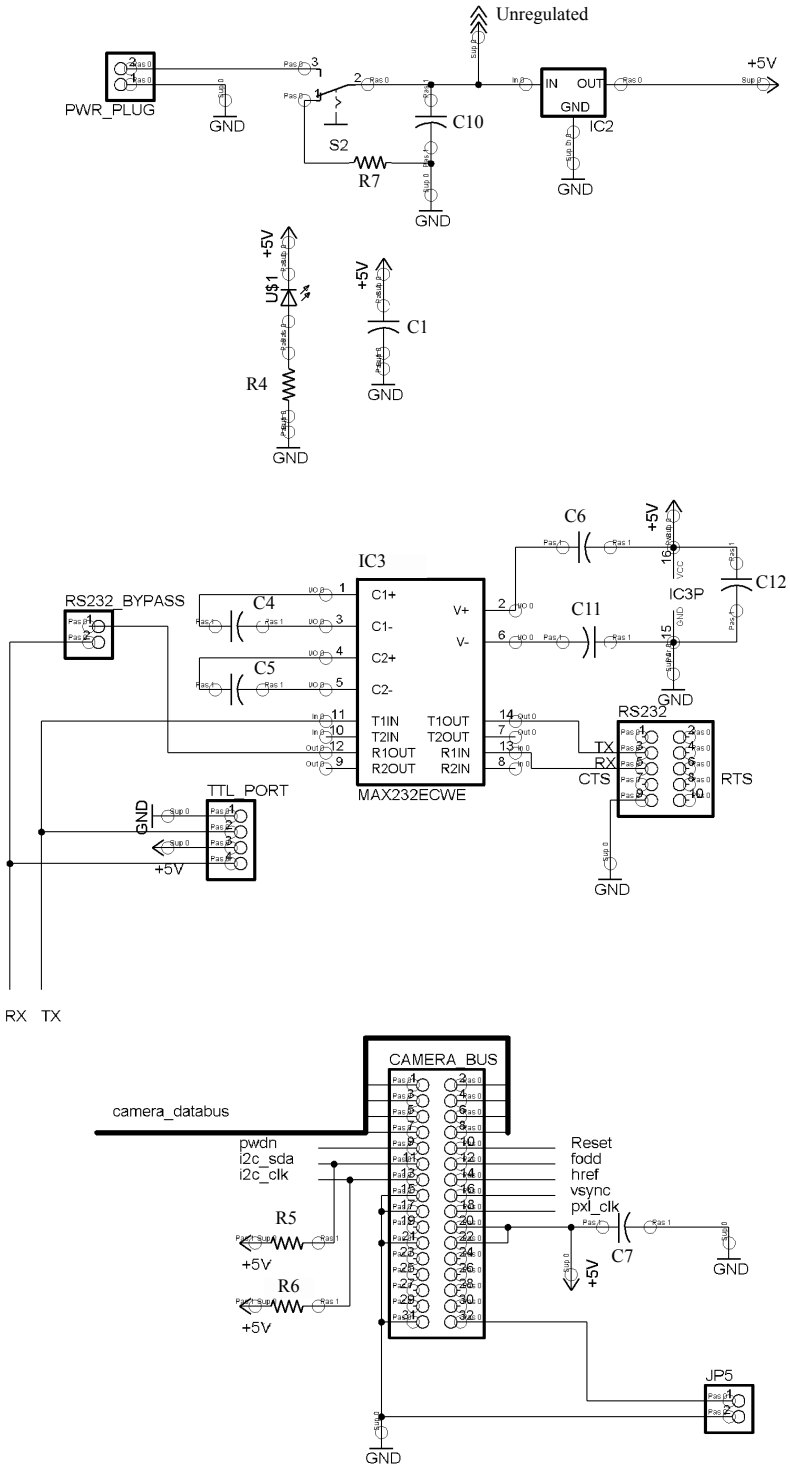
Parts List

Part	Digikey Part Number	Qty.	Schematic
High Brightness Red Led	160-1405-1-ND	1	U$2
High Brightness Green Led	160-1404-1-ND	2	U$3,U$1
820 Ohm Resistor	311-820ACT-ND	4	R2-4, R7
4.7 K resistor	311-4.7KATR-ND	3	R1,R5-6
NAND Gate	296-1087-1-ND	1	SINGLE_NANDG$2
Max 232 chip	296-13095-1-ND	1	IC3
	MAX232CWE-ND	alternate	
75 Mhz Oscillator	SG-8002DC-SHC-ND	1	QG1
5v Regulator	LM2940CT-5.0-ND	1	IC2
3.3v Regulator	LP2985IM5-3.3CT-ND	1	IC4
Averlogic AL422B		1	AL422B
Ubicom SX52		1	IC1
Slide Switch	EG1847-ND	1	S2
Push Switch	SW400-ND	1	S1
0.1uF Cap	311-1141-1-ND	4	C7-9,C13
33uF Cap	399-1634-1-ND	2	C1,C3
33uF Cap or 10uF 25V Cap	399-1634-1-ND 399-1599-1-ND	1	C10
1.0uF Cap	PCC2249CT-ND	5	C4-6,C11,C12
2.2uF Cap	PCC1923CT-ND	1	C2
Heatsink	HS333-ND	optional	
Double Female Header	929852-01-36-ND	1	CAMERA_BUS
Single Male Header	929647-09-36-ND	3	CLK, SX-KEY, RS232, PWR_PLUG, TTL_ PORT, RS232_BYPASS, JP1-2, JP4-5
Polarized 2 pin Terminal Housing	WM2700-ND	2	
Crimp Terminals	WM2200-ND	4	
Polarized 2 pin terminal header	WM2000-ND	2	PWR_PLUG, JP3
Female serial ribbon cable head	AFS09G-ND	1	
Serial Ribbon Cable Socket Connector	ASC10G-ND	1	

Schematic - Main Body



Schematic - Peripheral



Disclaimer

No warranties, either expressed or implied, are made regarding the operation, use or results of this hardware. This product is meant for educational purposes only. Any resemblance to real persons, living or dead is purely coincidental. CMUcam2 void where prohibited with some assembly required. Batteries and servos not included. Contents may settle during shipment so only use as directed. No other warranty expressed or implied. Do not use while operating a motor vehicle or heavy equipment. Apply CMUcam2 only to affected area. If condition persists, consult your physician. May be too intense for some viewers and for recreational use only. Do not disturb CMUcam2 during boot process. All models over eighteen years of age. No user-serviceable parts inside. Freshest if used before date on carton. Subject to change without notice. Many CMUcam2 times approximate and many pictures simulated. Breaking seal constitutes acceptance of agreement. This product is known to the state of California to cause birth defects. As seen on TV one size fits all. My man the yellow darts comments not actually generated by the yellow dart. Contains a substantial amount of non-tobacco ingredients. Colors may, in time, fade. Slippery when wet. If CMUcam2 acts up; keep cool; process promptly. Not responsible for direct, indirect, incidental or consequential damages resulting from any defect, error or failure to perform. Substantial penalty for early withdrawal. Keep away from fire or flame. Replace with same type. Some of the trademarks mentioned in this product appear for identification purposes only. No animals were hurt in the production of this device. This supersedes all previous notices.

Microtronix

Product Starter Kit

Revision A

Datasheet

9-1510 Woodcock St.
London, ON
Canada N6H 5S1
www.microtronix.com

This datasheet provides information regarding the Product Starter Kit. The following table shows the document revision history.

Document Revision History	
Date	**Description**
May 2005	Initial Release – Version 1.0

How to Contact Microtronix

E-mail

Sales Information: sales@microtronix.com

Support Information: support@microtronix.com

Website

General Website: http://www.microtronix.com

Nios Forum Website: http://www.niosforum.com

Phone Numbers

General: 519-690-0091

Fax: 519-690-0092

Typographic Conventions

`Path/Filename`	A path/filename
`[SOPC Builder]$ <cmd>`	A command that should be run from within the Cygwin Environment.
`…sdk/<path>`	A file that is relative to the sdk directory.
`Code`	Sample code.
↵	Indicates that there is no break between the current line and the next line.

Table of Contents

Features

Firefly Module

- Altera EP1C12F324C8 or EP1C4F324C8 Cyclone FPGA
- Altera EPCS4 or EPCS1 Active Serial Configuration Device
- Supports Nios II soft core processor
- Supports self-reconfiguration of FPGA from logic in FPGA
- On-board 24MHz oscillator
- 8 Mb Flash (one 8M x 16-bit chip)
- 16 Mb SDRAM (PC-133 compatible, one 4M x 32-bit chip)

Configuration

- On-board USB Blaster JTAG configuration circuit
- Active Serial configuration header

Communication and Clocks

- 4-wire RS-232 with female DB9 connector
- Socketed 24MHz oscillator

Prototyping Area

- Santa Cruz Header (46 I/Os, .1” pin-pin spacing, 3.3V I/O, access to regulated 3.3VDC, 5.0VDC, and unregulated power from the board)
- Santa Cruz Header, reduced I/O (41 I/Os, .1” pin-pin spacing, 3.3V I/O, access to regulated 3.3VDC, 5.0VDC, and unregulated power from the board)

General Description

The Microtronix Product Starter Kit provides a hardware platform for developing embedded systems based on the Microtronix Firefly Module featuring Altera Cyclone FPGAs. The board is based on an Altera EP1C12F324C8 Cyclone FPGA with 12,060 Logic Elements (LEs) and 239,616 bits of embedded on-chip memory.

The development board offers a great deal of flexibility, providing two Santa Cruz headers for add-on boards. It also allows for basic communication with the FPGA through both a RS-232 serial port and an on-board USB Blaster JTAG interface.

The Microtronix Product Starter Kit contains enough memory and supporting architecture to act as an excellent platform for embedded software development for various IP based soft-core processors (such as the Altera Nios II).

The Firefly Module makes use of the low-cost and low-power Active Serial Programming Device (Altera EPCS4 or EPCS1) to program the SRAM based Cyclone FPGA.

The Microtronix Product Starter Kit comes pre-loaded with an Altera Nios II design that connects all of the components on the board to an

Avalon Bus, which is an Altera soft-core bus architecture allowing designers to create Systems On Programmable Chip (SOPC) designs based on a Nios II soft-core processor. Software designers can use the pre-programmed design on the board to begin prototyping software immediately, with full access to all communications interface and memory components on the board.

Default Reference Design

When the Microtronix Product Starter Kit is powered on, the Cyclone FPGA configures itself by pulling configuration data from the Active Serial Configuration Device on the board. Once configuration of the Cyclone completes, the Cyclone issues a Power On Reset to itself, after which the Linux default Nios II reference design starts up and begins booting Linux from the FLASH device. Communication can be established through the USB JTAG interface using Altera's nios2-terminal utility.

Loading Designs to the Board

The board can be loaded with any design compiled for the Cyclone Device, including re-loading the default reference design, using one of the following two methods:

1. Using the on-board USB Blaster circuit to download a .sof file created by Quartus II during the compile process (see Altera Documentation for details on compiling designs and creating .sof download files). This downloads a design directly to the FPGA, but the design is lost when power is removed.
2. Using the Active Serial header (**J3,** Figure 2) to download a .pof file created by Quartus II during the compile process (see Altera Documentation for details on compiling designs and creating .pof download files). This downloads a design to the Active Serial Device, which the Cyclone uses to get a design from after power is applied to the board. The Active Serial Device contains serial flash memory, so the FPGA core is saved even when power is removed. When a design is loaded into the Active Serial Device, it does not become active on the Cyclone FPGA until after a power sequence to the FPGA occurs (either power off/on the board, hit the reconfigure button (**SW2,** Figure 2) which acts as a board reset and request for reconfiguration of the Cyclone Device, or toggle the dedicated Reconfiguration Request signal tied to an I/O line of the FPGA itself).

See the Microtronix Product Starter Kit Quick Start Guide for details on how to set up the Product Starter Kit board for operation.

Block Diagram

Figure 1 shows a block diagram of the Product Starter Kit board.

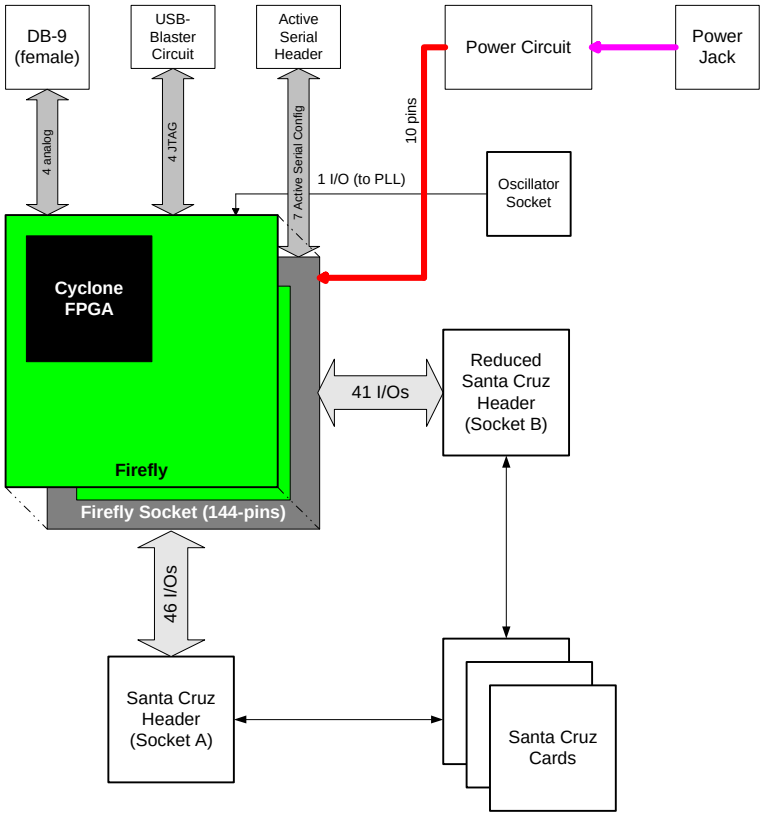


Figure 1: Block Diagram

Board Components

This section describes the various components and sub-circuits of the Microtronix Product Starter Kit. For each component or circuit, this section explains: basic function, performance limits, I/O requirements, and how to interface to the Avalon Bus in a Nios II based system (including custom interface components). Figure 2 shows the location of all the Product Starter Kit components mentioned in this section.

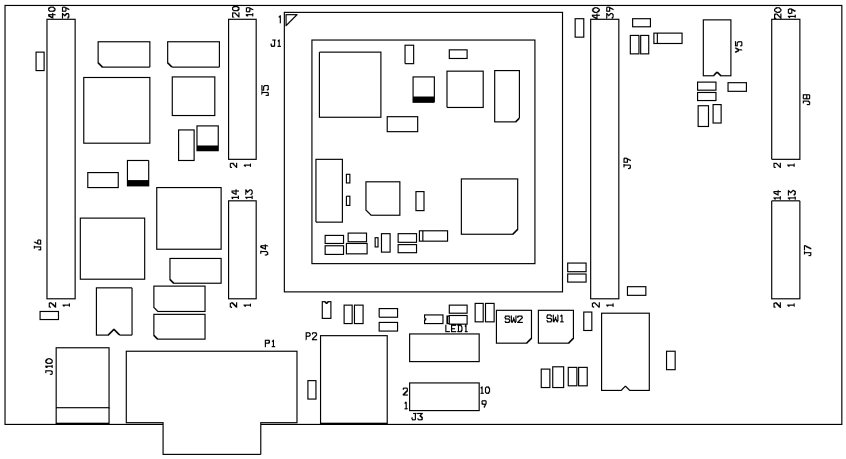


Figure 2: Product Starter Kit Components

USB Blaster

Included on the Product Starter Kit is Altera's USB Blaster circuit. A simple USB cable will allow a PC installed with Altera's development environment to download designs to the board. The USB Blaster circuit functions just like a stand-alone USB Blaster connected to the FPGA's JTAG port. In addition to downloading designs, it can be used to program the flash and configuration device, debug software and also communicate to a Nios II system through a JTAG UART.

Santa Cruz Headers

There are two Santa Cruz headers on the Product Starter Kit. Header A is a full implementation, while header B is a slightly reduced implementation (due to I/O constraints).

Santa Cruz A

J4, J5 and **J6** in Figure 3 make up Santa Cruz A. These headers provide 41 general I/O, 2 control I/O and 3 clock I/O connections to the Cyclone FPGA on the Firefly Module. In addition, the headers provide access to 3.3 and 5.0VDC regulated power as well as raw 12VDC from the wall supply.

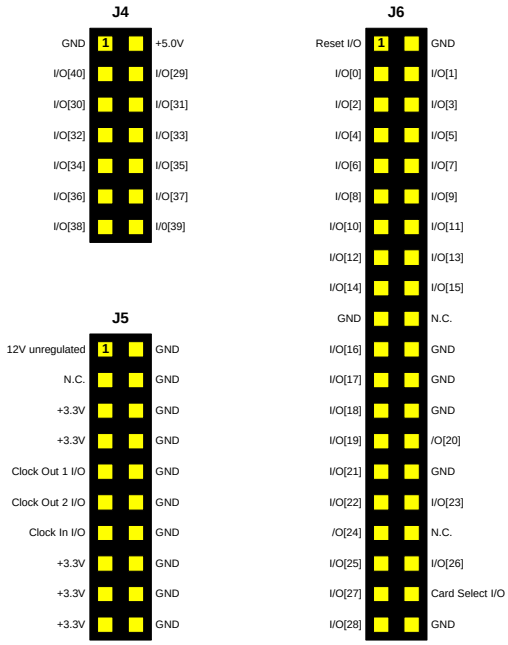


Figure 3: Santa Cruz A

Santa Cruz B

J7, J8 and **J9** in Figure 4 make up Santa Cruz B, a partial Santa Cruz header implementation. These headers provide 39 general I/O, 2 control I/O and 2 PLL (one in, one out) connections to the Cyclone FPGA on the Firefly Module. In addition, the headers provide access to 3.3 and 5.0VDC regulated power as well as raw 12VDC from the wall supply.

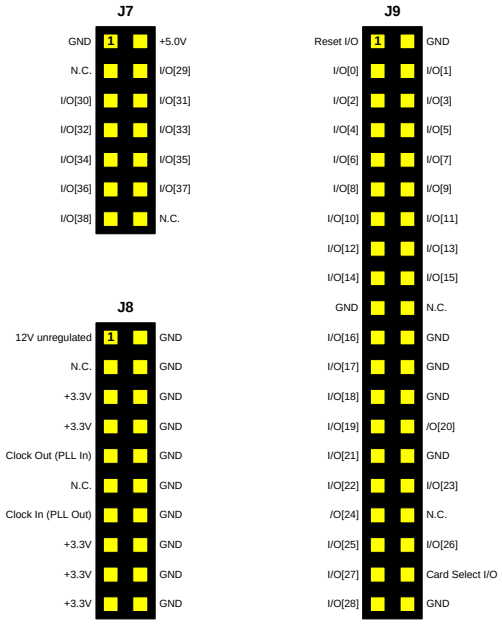


Figure 4: Santa Cruz B

Buttons

SW1 in Figure 2 is connected to pin C11 on the Cyclone device. Its intended use is as a reset for a Nios II system. It can, however, be used for any purpose.

SW2 in Figure 2 is used to trigger a reconfiguration of the FPGA. When this button is pressed, its current configuration is discarded and it starts to reload the configuration found in the serial configuration device.

Clock

Y5 in Figure 2 is a 4-pin DIP socket for standard half-size DIP oscillators. The input is connected to PLL 2 of the Cyclone through pin J15. The external output through pin A6 is connected to the CLK in signal of Santa Cruz B. A 24MHz oscillator is included with the board.

Firefly Module

The Product Starter Kit supports both the 1C12 and 1C4 versions of the Firefly Module. The Firefly module is inserted into **J1** in Figure 2. Figure 5 shows the location of all Firefly components mentioned in this subsection.

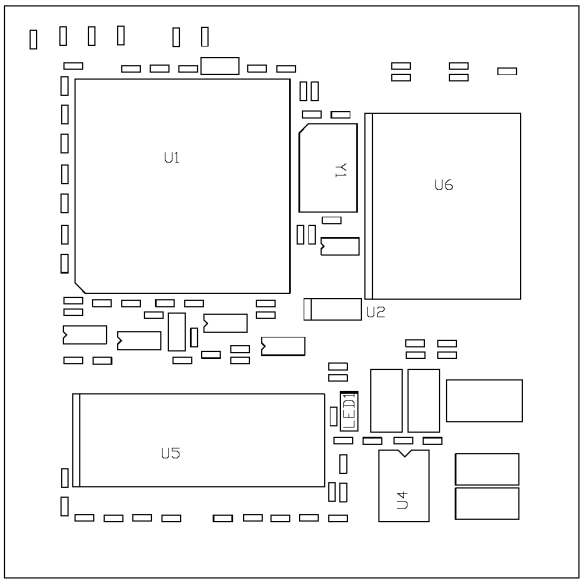


Figure 5: Firefly Components

Cyclone FPGA

U1 in Figure 5 is an Altera Cyclone EP1C12F324C8 or EP1C4F324C8 FPGA device in a 324 pin FineLine BGA® package. Table 1 lists the Cyclone device features.

Table 1: Cyclone Device Features

	EP1C12F324C8	EP1C4F324C8
LEs	12,060	4,000
M4K RAM Blocks	52	17
Total RAM Bits	239,616	78,336
PLLs	2	2
Maximum User I/Os	249	249
Maximum Clock Speed	275MHz	275MHz

The Cyclone I/O driver power for the FPGA is 3.3VDC supplied from a linear regulator fed by the 12VDC source. The core power (VCCINT) is 1.5VDC from another regulator also supplied by the 12VDC source.

The Cyclone FPGA on the Product Starter Kit can be configured using three methods:

1) Download a core via the USB Blaster on the board. The download file format is .sof. The core in the FPGA will be lost when power is removed from the board or the board is reset.

2) Download a core to the Active Serial Device using the Active Serial programming header on the board (**J3**, Figure 2), and an Altera download cable on the host PC. The file format is .pof. The core downloaded this way does not take effect in the FPGA until a power off/on sequence or board level reset (**SW2**, Figure 2) has been issued. The program remains in the flash portion of the Active Serial device even after power has been removed from the board. See the Active Serial Configuration Device section of this data sheet for more details. On board power-up, the Active Serial Device will attempt to configure the Cyclone FPGA.

3) Use the Active Serial Interface SOPC component in an existing core (or add one to your own core and load it via method 1) above), and Altera pre-defined functions for accessing the Active Serial Programming Device to load a binary file or ASCII text file into the Active Serial Device. Then force a reconfiguration of the Cyclone FPGA by having the code running on your SOPC design toggle a PIO output connected to the RECONFIGn signal (I/O pin E10 of the Cyclone FPGA). For more information on building SOPC systems, and on the PIO component, refer to Altera documentation.

See the Altera Cyclone literature page for Cyclone related documentation at http://www.altera.com/literature/lit-cyc.jsp.

Active Serial Configuration Device

U2 in Figure 5 is an Active Serial Configuration Device EPCS4 (for the 1C12 module) or EPCS1 (for the 1C4 module). This device is a serial

flash device with additional state machine logic for handshaking with the Active Serial Programming feature of the Cyclone FPGA. After a Power-On or Board Reset event, the Cyclone FPGA and Active Serial Programming Device work together to load a design from the flash memory of the configuration device into Cyclone FPGA SRAM cells. Once configuration is complete, the new core starts running.

Detailed information about the configuration device, as well as design file sizes, can be found in Table 2 below.

Table 2: Active Serial Configuration Devices

	EPCS4	EPCS1
Flash Memory Size (bits)	4,194,304	1,048,576
Max Serial Clock Frequency	20MHz	20MHz
Max Uncompressed Design Size (bits)	2,326,528	924,512
Min area left for general use (bits)	1,867,776	124,064
Compression Performance	35-60%	35-60%
Power On Reset Time	100ms	100ms

The minimum area left for general use in the flash memory of the configuration device can be used by software running on a softcore processor in the Cyclone FPGA (such as an Altera Nios II) to store configuration information, programs, other compressed core designs, etc. If an Altera Nios II based system is running on the FPGA, the EPCS Serial Flash Controller SOPC component, along with software routines provided by the HAL can be used to access the portions of EPCS flash memory that are not being used to hold the core design.

The Active Serial Programming Device also supports compression of the designs being held within it. The compression is enabled from the Quartus II design tool when compiling a logic design. The Active Serial Programming Device will send a compressed data bit stream to the Cyclone FPGA, which will decompress the data locally in real time and use the uncompressed data to program its SRAM based logic cells.

See the Altera configuration literature page for EPCS related documentation at http://www.altera.com/literature/lit-config.jsp.

Flash

U6 in Figure 5 is a 8MB FLASH chip (4M x 16-bit) chip with a 120ns access time. The FLASH sits on its own 16-bit dedicated bus.

Interfacing the FLASH with a SOPC based system can be easily accomplished using the “Flash Memory (Common Flash Interface)” SOPC component available that comes with Altera’s Nios II distribution. For proper operation, use the following parameters with the SOPC component: address width 22 bits, data width 16 bits, setup 20ns, wait 100ns, hold 45ns. This component also requires it’s own “Avalon Tri-

State Bridge" to connect to the CPU. See n2cpu_nii51013.pdf in the Nios II documents subdirectory for further details.

SDRAM

U5 in Figure 5 is a 16Mbyte SDRAM chip (4M x 32-bit) chip with a 7ns access time (PC133 compatible SDR SDRAM). The SDRAM exists as a 32-bit wide device on its own dedicated bus.

Interfacing the SDRAM with a SOPC based system can be easily accomplished using the Altera SDRAM SOPC component (see n2cpu_nii51005.pdf in the Nios II documents subdirectory for further details). The SDRAM chip used on the µKit board is available pre-defined in the SDRAM SOPC component. Choose "single Micron MT48LC4M32B2-7 chip" from the "Presets:" list.

Along with the SOPC builder component, the SDRAM requires a clock signal. This signal should be provided by a PLL megafunction in Quartus II. Pin J3 should be used as the input clock (24MHz on-module oscillator) and pin K4 should be used as the output to the SDRAM. The frequency of the output should match the frequency of the SOPC Builder system.

RS-232

U4 in Figure 5 is a Maxim Semiconductor 4-wire UART transceiver (2 input, 2 output) part MAX3232, capable of 115,200 baud transfer rate (max).

The UART transceiver is connected through 4 wires to P1 (female DB-9 connector) on the base board. The 4 wires used are: TXD, RXD, CTS, and RTS. The P1 connection pinout is a hybrid combination of DCE and DTE pinouts. The TXD and RXD pins are connected in DCE fashion, while the CTS and RTS pins are connected in DTE fashion. This configuration is needed to support the Split-Y cable for dual serial port from single DB-9 connector communication with the host PC.

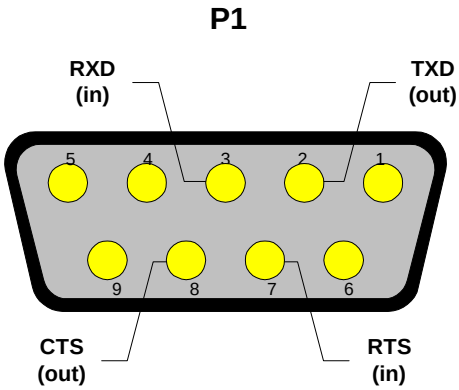


Figure 6: RS-232 Pinout

Interfacing the UART transceiver with a SOPC based system can be easily accomplished using the Altera UART SOPC component. See n2cpu_nii51010.pdf in the Nios II documents subdirectory for further details.

Clock

Y1 in Figure 5 is a 24MHz oscillator. It is connected to the Cyclone PLL 1 through pin J3. The external output of this PLL is pin K4 that is connected to the clock of the SDRAM device.

Appendix A: Cyclone FPGA Pinouts

Firefly Module

The following table lists all of the pin connections to the Cyclone device from the components on the Firefly Module.

Pin #	Signal	Connected to…	Comments
R9	FLASH_A0	FLASH (U6) *Address*	
V10	FLASH_A1		
U10	FLASH_A2		
T10	FLASH_A3		
R8	FLASH_A4		
V9	FLASH_A5		
U9	FLASH_A6		
T9	FLASH_A7		
R5	FLASH_A8		
V6	FLASH_A9		
U6	FLASH_A10		
T6	FLASH_A11		
V4	FLASH_A12		
U5	FLASH_A13		
T5	FLASH_A14		
U4	FLASH_A15		
U16	FLASH_A16		
R7	FLASH_A17		
V8	FLASH_A18		
U8	FLASH_A19		
U7	FLASH_A20		
T7	FLASH_A21		
U3	FLASH_A22	FLASH (U6) *Address*	Used for optional 16 and 32 MB FLASH
R4	FLASH_A23	FLASH (U6) *Address*	Used for optional 16MB FLASH
V11	FLASH_DQ0	FLASH (U6) *Data (16-bit)*	
T12	FLASH_DQ1		
V12	FLASH_DQ2		
T13	FLASH_DQ3		
V13	FLASH_DQ4		
R14	FLASH_DQ5		
R13	FLASH_DQ6		
U15	FLASH_DQ7		
R10	FLASH_DQ8		
U12	FLASH_DQ9		
R11	FLASH_DQ10		
U13	FLASH_DQ11		
R12	FLASH_DQ12		
T14	FLASH_DQ13		
T15	FLASH_DQ14		
V15	FLASH_DQ15		
U11	FLASH_OEn	FLASH (U6) *Control*	
V7	FLASH_WEn		
T11	FLASH_CEn		
T8	FLASH_ACC		FLASH_ACC should be tied to ground for normal operation
T4	FLASH_WPn		FLASH_WPn must be high to enable writing to the FLASH. A low signal will enable write-protect.
R6	FLASH_RSTn		

Pin #	Signal	Connected to…	Comments
E10	RECONFIGn	CONFIGURATION CIRCUIT	Toggling this line will force the Cyclone FPGA to reconfigure itself using the Data in the Active Serial Configuration Device (U15)
M4	SDRAM_A0	SDRAM (U5) *Address*	
M5	SDRAM_A1		
M6	SDRAM_A2		
L3	SDRAM_A3		
L2	SDRAM_A4		
H1	SDRAM_A5		
H2	SDRAM_A6		
H3	SDRAM_A7		
G3	SDRAM_A8		
G2	SDRAM_A9		
L4	SDRAM_A10		
L7	SDRAM_A11		
D4	SDRAM_DQ0	SDRAM (U5) *Data (32-bit)*	
E4	SDRAM_DQ1		
E5	SDRAM_DQ2		
F4	SDRAM_DQ3		
F5	SDRAM_DQ4		
F6	SDRAM_DQ5		
G4	SDRAM_DQ6		
G5	SDRAM_DQ7		
F2	SDRAM_DQ8		
F3	SDRAM_DQ9		
E3	SDRAM_DQ10		
E2	SDRAM_DQ11		
D1	SDRAM_DQ12		
D2	SDRAM_DQ13		
D3	SDRAM_DQ14		
C2	SDRAM_DQ15		
N6	SDRAM_DQ16		
N5	SDRAM_DQ17		
N4	SDRAM_DQ18		
N3	SDRAM_DQ19		
P4	SDRAM_DQ20		
P3	SDRAM_DQ21		
R3	SDRAM_DQ22		
T3	SDRAM_DQ23		
T2	SDRAM_DQ24		
R2	SDRAM_DQ25		
R1	SDRAM_DQ26		
P2	SDRAM_DQ27		
N2	SDRAM_DQ28		
N1	SDRAM_DQ29		
M1	SDRAM_DQ30		
M2	SDRAM_DQ31		
F7	SDRAM_DQM0	SDRAM (U5) *Byte Enables*	
F1	SDRAM_DQM1		
N7	SDRAM_DQM2		
M3	SDRAM_DQM3		
L6	SDRAM_BA0	SDRAM (U5) *Bank Address*	
L5	SDRAM_BA1		
H5	SDRAM_RASn	SDRAM (U5) *Control*	
H6	SDRAM_CASn		
G6	SDRAM_WEn		
H4	SDRAM_CSn		
G1	SDRAM_CKE		
K4	SDRAM_CLK	SDRAM (U5) *Clock*	

Pin #	Signal	Connected to…	Comments
P10	RS232_TXD	RS-232 (U4, P1)	
P9	RS232_RXD		
P12	RS232_CTS		
P13	RS232_RTS		
J3	FPGA_PLL_IN_A	Oscillator (Y1)	24 MHz

Product Starter Kit Base Board

The following table lists all of the pin connections to the Cyclone device from the components on the base board.

Pin #	Firefly Pin	Signal	Connected to…	Comments
L16	B8	SC_A_RSTn	Santa Cruz A (J6-1)	1kΩ pull-down
G12	X1	SC_A_SELn	Santa Cruz A (J6-39)	10kΩ pull-up
J13	Y2	SC_A_IO0	Santa Cruz A (J6-3)	
F12	V1	SC_A_IO1	Santa Cruz A (J6-4)	
H13	X2	SC_A_IO2	Santa Cruz A (J6-5)	
F13	T2	SC_A_IO3	Santa Cruz A (J6-6)	
G16	W2	SC_A_IO4	Santa Cruz A (J6-7)	
G15	P2	SC_A_IO5	Santa Cruz A (J6-8)	
H14	U2	SC_A_IO6	Santa Cruz A (J6-9)	
E16	L1	SC_A_IO7	Santa Cruz A (J6-10)	
G13	U1	SC_A_IO8	Santa Cruz A (J6-11)	
E15	K1	SC_A_IO9	Santa Cruz A (J6-12)	
G14	T1	SC_A_IO10	Santa Cruz A (J6-13)	
E17	G2	SC_A_IO11	Santa Cruz A (J6-14)	
F14	R2	SC_A_IO12	Santa Cruz A (J6-15)	
D17	F2	SC_A_IO13	Santa Cruz A (J6-16)	
E14	P1	SC_A_IO14	Santa Cruz A (J6-17)	
C17	E1	SC_A_IO15	Santa Cruz A (J6-18)	
H16	N2	SC_A_IO16	Santa Cruz A (J6-21)	
G17	N1	SC_A_IO17	Santa Cruz A (J6-23)	
H17	L2	SC_A_IO18	Santa Cruz A (J6-25)	
F16	K2	SC_A_IO19	Santa Cruz A (J6-27)	
D18	D1	SC_A_IO20	Santa Cruz A (J6-28)	
F15	H2	SC_A_IO21	Santa Cruz A (J6-29)	
D15	G1	SC_A_IO22	Santa Cruz A (J6-31)	
F18	C1	SC_A_IO23	Santa Cruz A (J6-32)	
D16	F1	SC_A_IO24	Santa Cruz A (J6-33)	
F17	D2	SC_A_IO25	Santa Cruz A (J6-35)	
H18	B2	SC_A_IO26	Santa Cruz A (J6-36)	
G18	C2	SC_A_IO27	Santa Cruz A (J6-37)	
M16	A2	SC_A_IO28	Santa Cruz A (J6-39)	
T17	B18	SC_A_IO29	Santa Cruz A (J4-4)	
R18	A16	SC_A_IO30	Santa Cruz A (J4-5)	
R17	B16	SC_A_IO31	Santa Cruz A (J4-6)	
R16	A15	SC_A_IO32	Santa Cruz A (J4-7)	
R15	B15	SC_A_IO33	Santa Cruz A (J4-8)	
P17	B13	SC_A_IO34	Santa Cruz A (J4-9)	
P16	B14	SC_A_IO35	Santa Cruz A (J4-10)	
N16	B12	SC_A_IO36	Santa Cruz A (J4-11)	
N18	A13	SC_A_IO37	Santa Cruz A (J4-12)	
M17	B10	SC_A_IO38	Santa Cruz A (J4-13)	
N17	A12	SC_A_IO39	Santa Cruz A (J4-14)	
T16	A17	SC_A_IO40	Santa Cruz A (J4-3)	
L17	A9	SC_A_CLKin	Santa Cruz A (J5-13)	
M18	A10	SC_A_CLKout1	Santa Cruz A (J5-9)	
L18	B9	SC_A_CLKout2	Santa Cruz A (J5-11)	

Pin #	Firefly Pin	Signal	Connected to…	Comments
B14	Y4	SC_B_RSTn	Santa Cruz B (J9-1)	1kΩ pull-down
A9	X6	SC_B_SELn	Santa Cruz B (J9-38)	10kΩ pull-up
E7	R20	SC_B_IO0	Santa Cruz B (J9-3)	
D7	P19	SC_B_IO1	Santa Cruz B (J9-4)	
E8	N20	SC_B_IO2	Santa Cruz B (J9-5)	
D8	N19	SC_B_IO3	Santa Cruz B (J9-6)	
D9	M20	SC_B_IO4	Santa Cruz B (J9-7)	
D10	M19	SC_B_IO5	Santa Cruz B (J9-8)	
E11	K20	SC_B_IO6	Santa Cruz B (J9-9)	
D11	K19	SC_B_IO7	Santa Cruz B (J9-10)	
E13	H20	SC_B_IO8	Santa Cruz B (J9-11)	
D12	H19	SC_B_IO9	Santa Cruz B (J9-12)	
B5	Y16	SC_B_IO10	Santa Cruz B (J9-13)	
E6	T20	SC_B_IO11	Santa Cruz B (J9-14)	
C5	X15	SC_B_IO12	Santa Cruz B (J9-15)	
B4	Y15	SC_B_IO13	Santa Cruz B (J9-16)	
A4	X14	SC_B_IO14	Santa Cruz B (J9-17)	
A6	Y14	SC_B_IO15	Santa Cruz B (J9-18)	
C8	X13	SC_B_IO16	Santa Cruz B (J9-21)	
C6	Y12	SC_B_IO17	Santa Cruz B (J9-23)	
B8	X12	SC_B_IO18	Santa Cruz B (J9-25)	
A7	X11	SC_B_IO19	Santa Cruz B (J9-27)	
B6	Y11	SC_B_IO20	Santa Cruz B (J9-28)	
B9	Y9	SC_B_IO21	Santa Cruz B (J9-29)	
B7	Y8	SC_B_IO22	Santa Cruz B (J9-31)	
C7	X9	SC_B_IO23	Santa Cruz B (J9-32)	
A8	X8	SC_B_IO24	Santa Cruz B (J9-33)	
A10	Y6	SC_B_IO25	Santa Cruz B (J9-35)	
C9	X7	SC_B_IO26	Santa Cruz B (J9-36)	
B10	Y5	SC_B_IO27	Santa Cruz B (J9-37)	
C10	X5	SC_B_IO28	Santa Cruz B (J9-39)	
L13	F19	SC_B_IO29	Santa Cruz B (J7-4)	
N13	G19	SC_B_IO30	Santa Cruz B (J7-5)	
N14	E19	SC_B_IO31	Santa Cruz B (J7-6)	
M13	F20	SC_B_IO32	Santa Cruz B (J7-7)	
L14	C19	SC_B_IO33	Santa Cruz B (J7-8)	
P14	E20	SC_B_IO34	Santa Cruz B (J7-9)	
M15	B19	SC_B_IO35	Santa Cruz B (J7-10)	
P15	D20	SC_B_IO36	Santa Cruz B (J7-11)	
L15	A19	SC_B_IO37	Santa Cruz B (J7-12)	
N15	B20	SC_B_IO38	Santa Cruz B (J7-13)	
K15	A6	SC_B_CLKin	Santa Cruz B (J8-13)	Connected to PLL2 output
J16	A7	SC_B_CLKout1	Santa Cruz B (J8-9)	Connected to PLL2 input
J15	J15	BRD_CLK	SOCKETED OSCILLATOR (Y5)	Connected to PLL2 input
C11	X3	CPU_RESETn	BUTTON (SW1)	

GR 63x55, 100 W

Versions of GR 63x55 / Ausführungen GR 63x55	P./S.
With gearbox / Als Getriebemotor	37
With brake / Als Bremsmotor	48
With controller / Mit Regelelektronik	54
With tacho generator / Mit Tachogenerator	50
With magnetic pulse generator / Mit magnetischem Impulsgeber	51
With incremental encoder / Mit Inkrementalgeber	52

Standard/Standard On request/auf Anfrage

- General information about the characteristics of our commutated motors, see page 8
- the standard version has leads (300 mm)
- special and high voltage windings available on request
- on request different shaft lengths and diameters or shaft on both sides are available as per our program.
- Protection class IP 50, higher class available on request.
- motor shaft with ball bearing

- Allgemeine Informationen über die Eigenschaften unserer Kollektormotoren siehe S. 8
- Der Motor wird standardmäßig mit Litzen (300 mm) geliefert
- Sonder- und Hochspannungswicklungen auf Anfrage erhältlich
- Auf Anfrage verschiedene Wellenlängen und -durchmesser bzw. beidseitige Wellen gemäß unserem Programm lieferbar
- Schutzart IP 50, auf Anfrage auch höher
- Die Motorwelle ist kugelgelagert

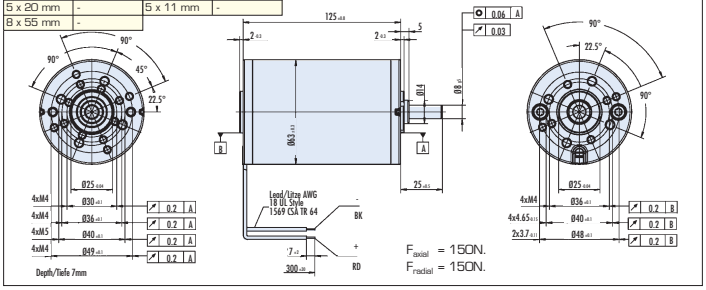
Data / Leistungsdaten		GR 63x55			
Rated voltage/ Nennspannung		12 VDC	24 VDC	40 VDC	60 VDC
Continuous rated speed/ Nenndrehzahl	rpm*)	3000	3350	3450	3350
Continuous rated torque/ Nenndrehmoment	Ncm*)	24	27	27	28.5
Continuous current/ Nennstrom	A*)	8.7	4.9	3	2
Starting torque/ Anlaufmoment	Ncm**)	202	211	210	200
Starting current/ Anlaufstrom	A**)	64	40	28.6	19.7
No load speed/ Leerlaufdrehzahl	rpm**)	3500	3650	3600	3600
No load current/ Leerlaufstrom	A**)	0.8	0.4	0.28	0.2
Demagnetization current/ Entmagnetisierstrom	A**)	66	33	20	13
Rotor inertia/ Trägheitsmoment	gcm²	750	750	750	750
Weight of motor/ Motorgewicht	g	1700	1700	1700	1700

*) $\Delta\vartheta_W$ = 100 K; **) ϑ_R = 20 °C

GR 63x55, 100 W

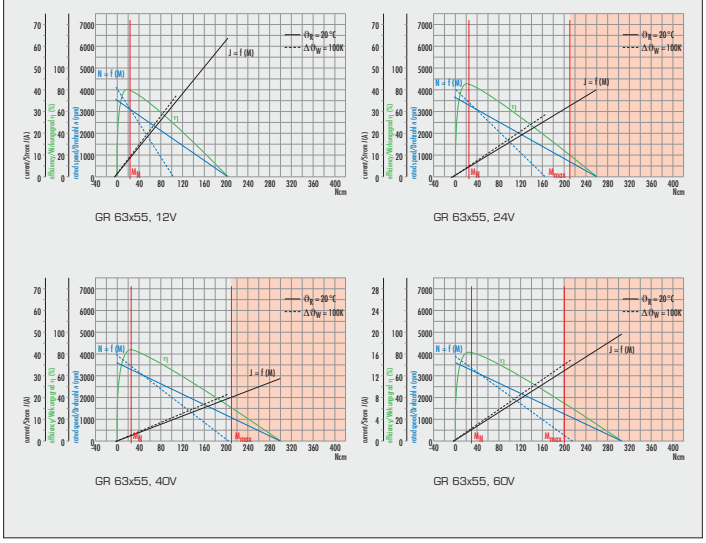
Dimensions in mm / Maßzeichnung in mm

Welle vorne Shaft front	Welle hinten Shaft back	Welle vorne Shaft front	Welle hinten Shaft back
8 x 25 mm	-	8 x 55 mm	8 x 55 mm
5 x 20 mm	-	5 x 11 mm	-
8 x 55 mm	-		



Characteristic diagram / Belastungskennlinien

In accordance with EN 60034
Belastungskennlinien gezeichnet nach VDE 530



Zilog Ethernet Camera

Matthew Condit

CS 339

5/5/2005

Design Overview

The purpose of this project is to create a CMOS camera module interface over Ethernet. An ez80 Acclaim! Zilog microcontroller is used to read information from a camera module and then send the raw picture data to a PC. The microcontroller is also responsible for accepting camera configuration commands and then setting the camera register to the appropriate values using the I^2C bus. A GUI on the PC facilitates the display of the captured pictures as well as camera configuration settings.

The proposed requirements that the project must meet are:

- Interact with PC over Ethernet interface.
- Support TCP/IP
- Be able to grab a single frame from camera module.
- PC must be able to modify camera settings.
- Microcontroller will do basic formatting of image data before sending back to PC

Hardware Modules

The project design consists of two main hardware modules: the ez80 Acclaim! Modular Development Board and the C3088 CMOS Camera.

eZ80F91 Modular Development Kit

The EZ80F91 modular development kit allows the user to design and evaluate projects using the EZ80F91 microcontroller. The kit contains an EZ80F91 module, which contains the EZ80F91 device running at 50 MHz, with 256 Kbytes of Flash memory and 8 Kbytes of internal SRAM. Off chip, the processor has access to 128 Kbytes of external SRAM. This module also contains an Ethernet Media Access Controller (EMAC) and Ethernet port. In the Zilog literature, this module is referred to as the mini-module because it plugs into a larger modular development adaptor board through 56 pin mini-module connectors.

The adapter board contains an RS-232 connector circuit for UART0 of the EZ80F91. Furthermore, it contains JTAG and Zilog's proprietary ZDI for debugging and programming. This board is actually made to plug into yet a larger board through two 60-pin headers on the bottom of the adapter board. This project does not need the larger board, so the header pins are used to directly interface with the microcontroller instead. There is also a footprint on the adapter board for a GPRS modem. Finally, the adapter board provides voltage regulation from 9V to 5V and 3.3V respectively.

C3088 CMOS Camera Module

The C3088 is a ¼" color camera module with digital output. It uses the Omnivision OV6620 image sensor. Together, the module provides a continuous stream of 8 or 16 bit-wide image data through a digital video port. All the camera functions, including gain, brightness, white balance, image size, etc, are configurable over the I^2C bus.

The C3088 supports multiple data formats using different channels (RGB, UVY). The format used in this design is the simplest one possible: YUV 4:2:2. For the project, only the Y channel (intensity) is used so that each pixel is represented by 8-bits and corresponds to a grayscale value. The UV channel (chrominance) represents the color aspects of the picture, but this project leaves them unconnected. This information is sent continuously from the camera and the VSYNC, HREF, and PCLK signals are used to synchronize the picture. The assertion of the VSYNC signal indicates the start of a new frame. The HREF signal is used for horizontal synchronization and indicates when data is valid. The PCLK signal is used to clock out picture data. The period of PCLK can be configured over the I^2C bus. The default period of the camera is 17.76 MHz.

Figure 1 shows the timing signals used in the C3088.

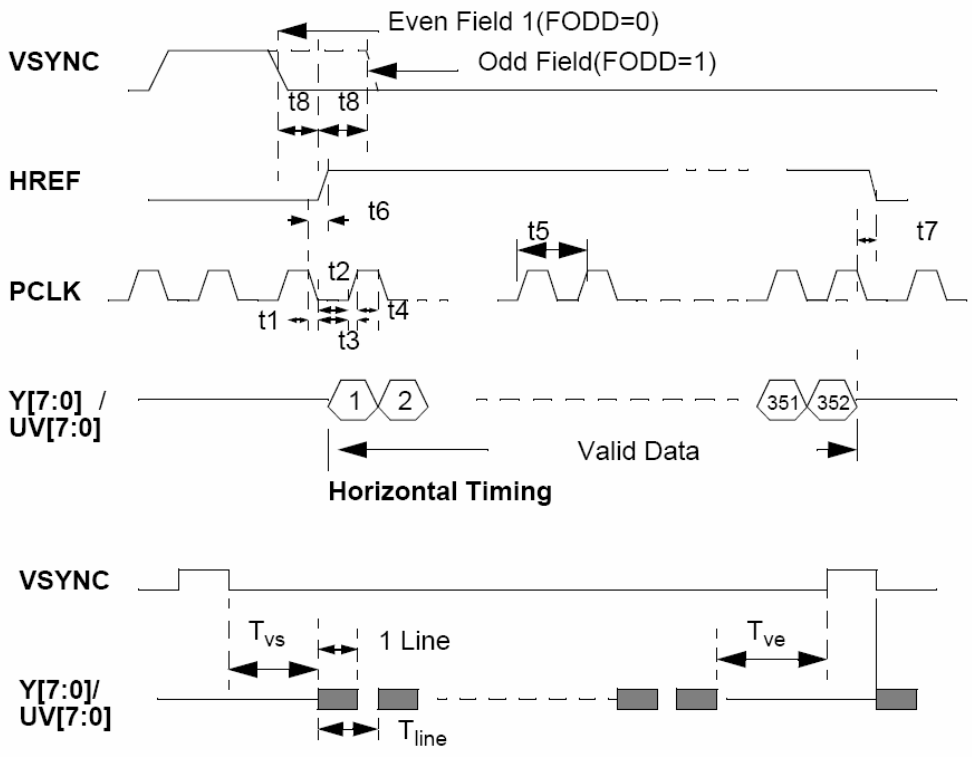


Figure 1. Default C3088 Timing

Here, the default C3088 timing is shown. The transition of VSYNC from high to low, signals the start of a new frame. A frame consists of a number of rows. In this default case, there are 288 rows of 352 pixels each. After the start of a frame, HREF will become asserted. While HREF is high, on the rising edge of each PCLK, picture data can be sampled from the Y and UV buses. In the default case, there should be 352 PCLK cycles while HREF is asserted. Once a complete row has been sent, HREF is set to low. Once the start of the next row begins, HREF is asserted once again. In the default case, this process takes place 288 times.

Table 1 shows the pinout of the C3088.

Table 1. C3088 Pin Description

Pins	Name	Description	Design Comment
1-8	Y0-Y7	Digital Y output	
9	PWDN	Power Down	Grounded
10	RST	Reset	Grounded
11	SDA	I^2C Serial Data	
12	FODD	Odd Field Flag	Not Used
13	SCL	I^2C Serial Clock	
14	HREF	Horizontal Reference	
15, 17, 21	AGND	Analog Ground	
16	VSYNC	Vertical Sync	
18	PCLK	Pixel Clock	
19	EXCLK	External Clock	Not Used
20, 22	VCC	5V Power Supply	
23-30	UV0-UV7	Digital UV output	Not Used
31	GND	Common Ground	Not Used
32	VTO	Video Analog Output	Not used

The main signals used in the project at the Y bus, I^2C bus signals, VSYNC, HREF, PCLK, VCC, and AGND.

Hardware Interfacing

The interfacing of the microcontroller to the camera module is relatively straightforward. Only 13 pins from the C3088 directly connect to the microcontroller. Figure 2 shows the schematic for the design.

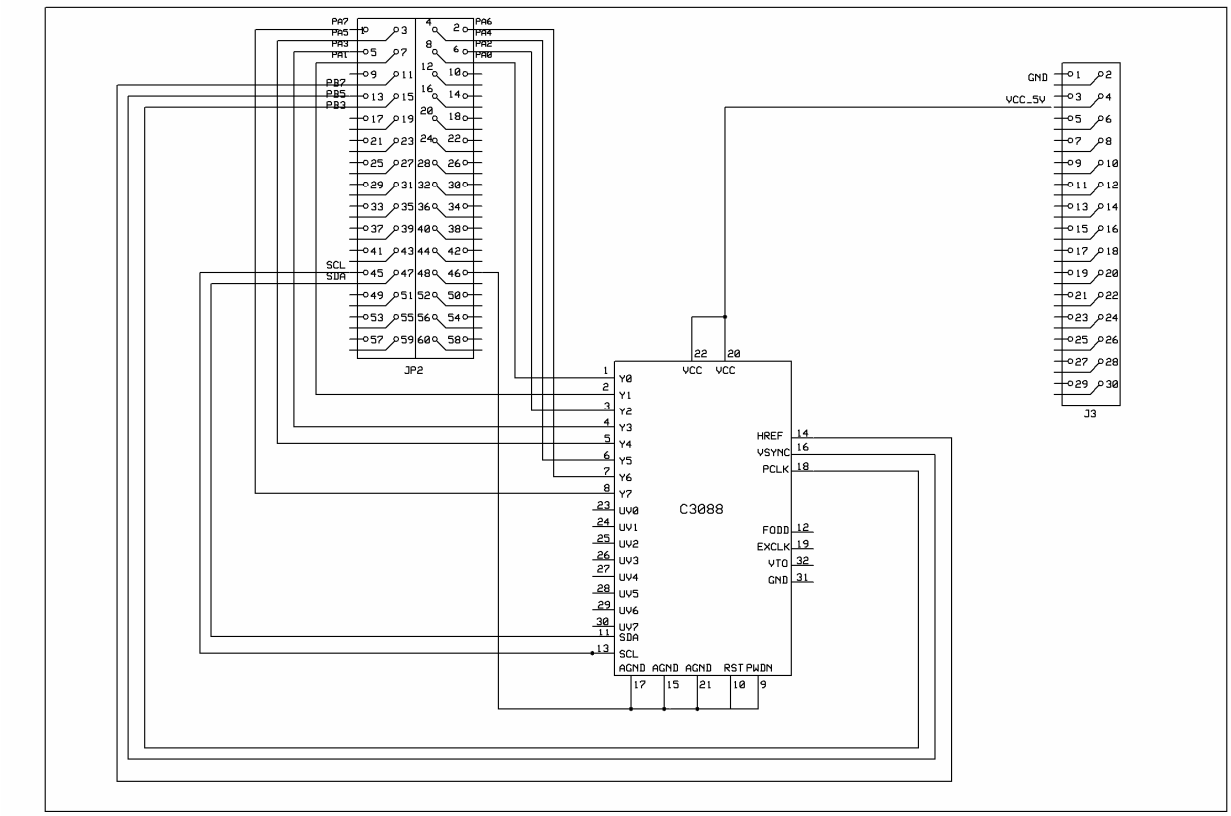


Figure 2. System Schematic

The JP2 header is present on the bottom of the eZ80F91 Adapter board. This header provides access to the microcontroller's GPIO pins and is used to connect the C3088 to the processor. The J3 header is for a GPRS modem, however, in this case the 5V power pin is used to power the C3088. The EZ80F91 runs on a voltage of 3.3V while the camera needs 5V. The only easily accessible voltage of 5V is from the J3 socket. Table 2 summarizes the pin connections.

Table 2. Pin Summary

Camera Pins	Pin Description	Board Pin Number/Header	Name
1-8	Y0-Y7	JP3: 7-0	PA0-PA7
11	SDA	JP3: 47	SDA
13	SCL	JP3: 45	SCL
14	HREF	JP3: 11	PB7
16	VSYNC	JP3: 13	PB5
18	PCLK	JP3: 15	PB3
9	PWDN	JP3: 46	GND
10	RST	JP3: 46	GND
15	AGND	JP3: 46	GND
17	AGND	JP3: 46	GND
21	AGND	JP3: 46	GND
22,28	VCC	J3: 4	VCC_5V

Parts List

Table 3 contains the part numbers used for the project as well as cost.

Table 3. Parts List

Description	Part Number	Cost
EZ80F91 Dev Board	EZ80F910100KIT	$99
C3088 CMOS Camera	C3088-4928IR	$55
15 pin flex cable x2	A9BAG-1508F-ND	$15.88
16 pin flex cable x2	A9BAG-1608F-ND	$16.64

Software

The software design of the project is divided into two distinct halves: the microcontroller camera interface and the client-side GUI. The majority of the work took place in the microcontroller firmware, so it warrants a detailed examination.

File Listing:

Application Files

- main.c – main entry point of the program. Initializes UART and camera. Launches main application thread
- cam.c, cam.h – camera specific functions
- i2c_cam.c, i2c_cam.h - I^2C functions used to read and write bytes

- network_transfer.c, network_transfer.h – main thread of the application. Handles requests from PC over a socket as well as initiates frame grabs.

RZK Files

- config.c – memory map for processor
- eZ80eval.c – devboard specific functions
- ez80HWConf.c – device driver configuration file
- F91PhyInit.c - handles initialization of Ethernet port
- RZKStartup.s – assembly code for the startup of the OS
- tty_conf.c – defines number of TTY devices to support
- XTLInit.c – launches the main.c file
- XTLSysgen.c – sets initial defines for memory
- XTLZDevice.c – defines the device drivers loaded into OS
- ZTPConfig.c – sets up IP addressing, DHCP requests, etc.

Firmware

RZK/ZTP Framework

The eZ80F91 development board comes with its own real time operating system (RTOS) called RZK. Specifically, RZK is used to support Zilog's own TCP/IP stack called ZTP. For this project, RZK version 1.1.2 and ZTP version 1.4.2 are used. The OS provides facilities for thread creation and management, memory allocation and partitioning, built in support for serial communication via a UART, among many other things. The TCP/IP stack has many built in features such as telent, ftp, http, SSL, and standard sockets. Within this framework of design, the project was created.

Compiling the project

Note: To properly compile the project, ZDS II for the ez80 Acclaim! version 4.8.0 must be used. Newer versions of the IDE do not support RZK version 1.1.2 and ZTP 1.4.2.

In order to successfully compile the project, the necessary project options had to be deteremined. The basis for the settings for this project come from the ZTPDemo_F91_Mini sample project provided with ZTP. Please see Appendix A for project compilation settings.

Serial Communication

Simple debugging was accomplished using the processor's UART and a terminal program running on a PC. Within RZK as a default, UART0 is selected for serial communication. After using RZK's open command on SERIAL0, UART0 is properly configured. After this, standard printf() statements can be used to convey debug information. This was the exact scheme used in the project to help diagnose problems.

I^2C Communication

RZK has built-in support for I^2C, however, their libraries never worked properly in the design. Instead, a set of I^2C functions was created for the project that used the processor's I^2C resources accessed by using the I2C_CTL, I2C_CCR, and I2C_DR registers. These functions include I2C_Init(), I2C_Write(), and I2C_Read() among some others. These functions can be found in the i2c_cam.c and i2c_cam.h files.

The functions in these files are application specific in that they automatically place the proper device address onto the bus. A write cycle begins by issuing a start condition. The next step is to place the write address of the camera (0xC0) onto the bus. Next, the register to write to is placed. Following this, the byte to write is issued. A read cycle is slightly more complex. The read will always return the contents of the last write. To select a different register, a dummy write must be enacted where only the register is selected and nothing written. After the dummy write, the read address of the camera is placed on the bus (0xC1). After this, the camera sends two bytes back. The first is the register address, and the second is the actual value.

One major hurdle that had to be overcome was to reliably get the I^2C functions to work properly. As it turns out, after the camera has been reset, the next I^2C instruction must be repeated twice for the value to be properly written. To account for this, the wrapper functions found in the cam.c file always read the value back after a write. If the value was not correct, it will write it again. These functions will be further elaborated later.

Sockets Programming

One advantage to using ZTP 1.4.2 is its support for standard sockets programming conventions that have been used in C programs for years. After the main application thread is launched, a socket is opened and binded. After this, the thread listens for an incoming connection on the appropriate port. Once the connection is accepted, the thread waits for data to be sent. It is here then that it implements the camera protocol defined by the design.

Protocol Implementation

The project supports a number of commands to be sent from PC to the Zilog chip. These commands are detailed in Table 3.

Command Name	Hex Value	Description
GRAB_FRAME	0xbb	Initiates the transfer of one frame. After sending the packet, an ACK will be sent back. Following the ACK, the raw data bytes will be sent. Currently only 1 picture size is supported (176x144). Future version will implement the 352x288 picture size.
TERMINATE_CONNECTION	0xcc	Terminates the connection. This will inform the camera interface to shut down the connection socket and accept new connections.
ADJUST_BRIGHTNESS	0x01	Changes the brightness setting of the camera. The camera default is 0x80. 0xFF is the highest setting, 0x00 is the smallest.
ADJUST_CONTRAST	0x02	Changes the contrast setting of the camera. The camera default is 0x48. 0xFF is the highest setting, 0x00 is the smallest.
ADJUST_SHARPNESS	0x03	Changes the sharpness setting of the camera. The default setting is 0xC6. The upper 4 bits set the threshold of sharpness. The lower 4 bits provide the sharpness control.
ADJUST_CLOCK_SPEED	0x04	Changes the clock speed of the camera. The default camera setting is 0x00. Currently, the fastest the picture data can be sampled is when the clock speed is set as 0x14. The slowest clock rate is 0x3f.
ADJUST_SATURATION	0x05	Changes the saturation settings of the camera. The default value is 0x80. OxFF is the highest setting, 0x00 is the smallest.
RESET_CAMERA	0x06	Resets the camera to its default state
MIRROR_IMAGE	0x07	Mirrors the image.
UNMIRROR_IMAGE	0x08	Unmirrors the image.
BIG_PICTURE_SIZE	0x09	Configures the camera to capture pictures with a resolution of 352x288. This is currently not supported.
SMALL_PICTURE_SIZE	0x0A	Configures the camera to capture pictures with a resolution of 176x144.
BLACK_WHITE_MODE	0x0B	Sets the camera into black and white only mode. This is currently the only option supported.
COLOR_MODE	0x0C	Sets the camera into color mode. Currently, this mode is not supported.

After each command is parsed, the appropriate wrapper function from the cam.c file is called. In general, these commands set register values in the camera over the I^2C bus. There are a few that do not, however. TERMINATE_CONNECTION closes the session and deletes the current socket. The program begins to listen for a new connection at this point. GRAB_FRAME causes the camera to obtain picture data and transfer it over the Ethernet connection. This process is further elaborated in the following section.

Obtaining Picture Data

The core function behind the entire project is the grab_frame() function found in cam.c. This is the function that actually reads raw picture data from the camera. To do this, the function takes in a pointer to a buffer in memory large enough to hold an entire picture. Currently, this means that the buffer size is 176x144 in length for a total of 25344 bytes. This is roughly 1/5 of the available SRAM. In its current configuration, RZK will not allocate a buffer much bigger than this. This is one reason why the larger resolution of 352x288 is not supported in the project.

The function begins by waiting for the VSYNC signal to go high and then for it to go low. This signals the start of a new frame. After this, it begins a loop that will iterate 144 times – one for each row. It first waits for HREF to assert. It then begins another loop that continues looping while HREF is high. Once in this loop, it waits for the rising edge of PCLK. Once high, the program samples the 8-bits of the Y channel and increments the buffer counter. Once HREF drops low, 176 pixels should have been sampled. The loop then waits for HREF to go high again and the process repeats itself. Once the outer loop has iterated the requisite number of times, the function returns and the raw data is sent over the socket connection to the PC.

There were many issues getting this function to work. The first problem that was encountered was the fact RZK operating system kept preempting the main application thread while picture data was being sampled. It took some work determining that this in fact was the problem. This problem was solved by preventing other threads from preempting the application thread whenever grab_frame() was called. Also, the thread priority was increased as well as the thread time slice. This ensures that the application will have preferential treatment of the processor.

Another problem involved finding the correct speed at which to set the camera clock in order to read from it fast enough. The camera clock has to be significantly slowed down to around 135 KHz. This is a direct result of the way the VSYNC, HREF, and PCLK signals are sampled. To increase speed, a detailed timing analysis should be conducted. Interrupts should be used to signal the presence of VSYNC and HREF. Assembly language should then be written that will exactly match the number of cycles to read in pixels. At this time, the current code is not agile enough to properly sample data points. Once the processor can more effectively sample data, the clock speed of the camera can be increased.

Initially, the big picture size (352x288) was to be supported. As was stated previously, only about ¼ of the total amount of memory needed could be allocated. To get around this problem, multiple loops were used to send a quarter of the image at a time. The problem with this approach was the time it needed to do this. 10 total frames had to be read (read the first quarter, then it must wait for the first quarter to pass and grab the next quarter, etc). Because of the delay caused by sending the data, the image quality suffered. Each quarter of the image was clearly visible. Because of this poor performance, the big picture size was disabled in the project.

GUI

The Java GUI is implemented in two parts: Camera.java encapsulates interaction with the camera itself and CameraGUI.java contains the requisite java code to control the GUI. The GUI itself is rather straightforward and won't be discussed further, but the Camera.java class bears looking into further.

Camera.java implements the protocol discussed earlier. Each command is mapped to its own specific method each keeping track of any GUI related information it may contain. For example, the adjust saturation, contrast, brightness, and sharpness scales the value into a percentage that is displayed on the slider bars on the GUI.

The most important method in the class, however, is the getImage() method. This method borrows heavily on the java.awt.image APIs. After all of the pixel information has been read into a byte array, DataBufferByte instance is created from the read-in information. Next, a component sample model must be created that specifies how the pixel information is to be interpreted. In this case, every byte represents one pixel. The image height is 144 rows, and the image width is 176 pixels. After the sample model is created, a component color model must be created; this holds the color information for the picture. In this case, a grayscale model is used. Once the color model is created, a writable raster must be created that takes in the sample model and the pixel buffer as arguments. Finally, an instance of image can be create which uses the writable raster and color model as inputs. This image class can now be displayed in the gui or even used to save the image in a variety of formats (jpeg, bmp, png, etc).

GUI Operation

The GUI performs several simple tasks. First it takes in the IP address of the camera to connect to. This project loads in a default IP of 192.168.0.50 for the camera. Once the IP address has been entered, the user must initiate a connect. Once connected, the options at the bottom of the GUI become accessible.

The Capture Image button initiates a picture transfer and displays it in the GUI. The image itself has been expanded to better fill the GUI window. Saturation, sharpness, contrast, and brightness are all configurable by the slider bars. Unfortunately in the black and white mode setting of the camera, only brightness changes the image; the other three do not produce anything significant. Finally, the Mirror Image check box toggles whether or not the image should be mirrored or not. Figure 3 shows a screenshot of the GUI.

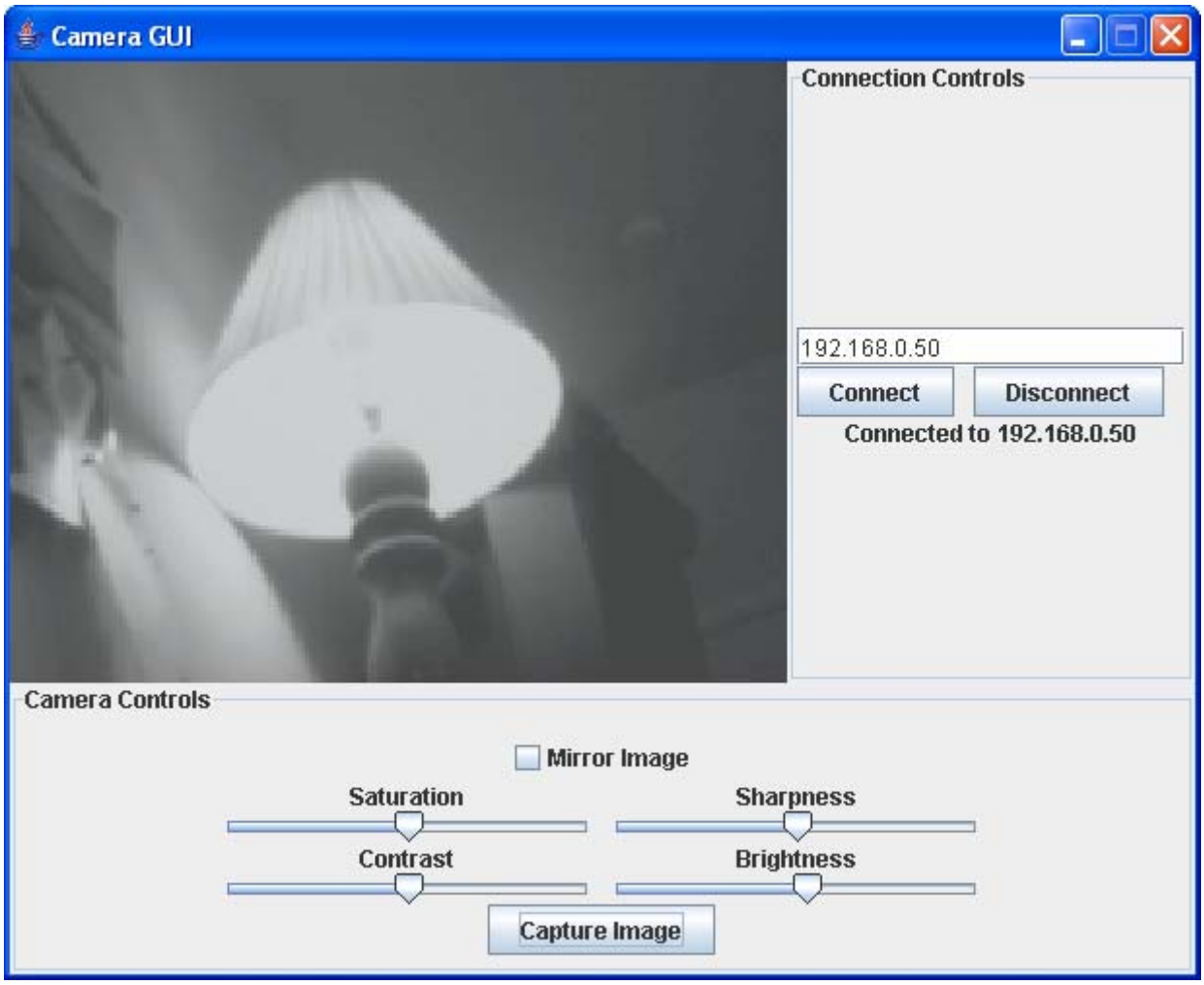


Figure 3. GUI Screenshot

Future Additions and Changes

The results of the project are by no means perfect. There are many parts that can be improved and many features could be easily added without too much difficulty. Some possible ideas are discussed below.

Faster Pixel Capturing

The most important part of the design that needs improving is the pixel-capturing algorithm. By improving this part of the code, an entire picture can be read much quicker which would allow for faster frames per second. To scan an entire picture, it currently takes roughly 188 ms; if the camera clock period could be increased to somewhere around 2.2MHz, an entire frame could be captured in approximately 12 ms.

In order to accomplish this, the pixel data capturing code should be rewritten using hand-coded assembly language. The assembly instructions should be designed keeping in mind the number of cycles to execute and how they relate to the incoming data. For instance, if the camera clock runs at 2 MHz and the processor runs at 50 MHz, every 25 cycles will correspond to one clock cycle. Using a scheme such as this along with external interrupts will allow a tight interfacing between components and summarily increase the speed at which data can be read from the camera.

Another way to increase pixel capture time is to shorten the transmissions lines between the processor and the camera. In the current design, the image data must travel over two ribbon cables connected through a protoboard. By designing a custom PCB where both modules can snap into, propagation delay and clock skew will have less influence on the quality of the data being transferred.

Larger RAM

One of the biggest limitations in the current design is the inability to obtain a large enough block of memory from the operating system. Even though the system has 128Kbytes of SRAM, only one fifth of this can be allocated.

To obtain a larger memory buffer, there are several options that could be explored. The first is simply to increase the amount of memory in the system. The SRAM could be increased from 128Kbyte to 256 Kbyte or 512Kbyte which should easily allow an entire picture with resolution of 352x288 and 16-bit pixels to be stored in RAM. Another option is to optimize the operating system by minimizing the amount of RAM it consumes. Things such as thread stack sizes as well as loaded module would have to be taken into account. This might necessitate a change of operating systems and even processors if memory requirements cannot be met.

Color Pictures

Color pictures could be added relatively easily to the design. The UV channel bus of the C3088 would need to be connected to the microprocessor. The camera supports a variety of picture formats such as YUV 4:2:2 and RGB. A suitable picture format should be chosen so that the proper pixel interpretation could be implemented because each format transfers pixels differently. There are several issues when color is added to pictures, however. First, is that adding color roughly doubles the memory requirements of a picture because each pixel is now expressed as a 16-bit value as opposed to a 8-bit grayscale value. Because the picture is now a larger amount of information, this will impact Ethernet transfer time. The eZ80F91 has a maximum TCP transfer time of approximately 2 MBps. A new Ethernet controller and processor may have to be used in order to transfer information faster.

Data Compression

Another method to reduce transfer time is to implement a data compression scheme before the data is sent. This could significantly reduce the number of bits needed to be sent which will help increase the frames per second that can be displayed. The amount of time to compress the raw pixel data, however, must be operate quickly so that the net time savings for compression plus transmission time is meaningful. Adding compression is not a trivial task.

Image Processing

The processor can accomplish a variety of image processing before ever sending it to the PC. This could include color tracking, motion detection, image windowing, etc. Doing this, however, could add significant processing overhead that could affect the number of supported frames per second. This might be acceptable depending on the application.

Appendix A: Project Settings

General Tab

- CPU Family: eZ80 Acclaim!
- EZ80F91

C Tab - > Category Preprocessor

- Preprocessor Definitions: HELL,I2C,_EZ80F91,EVB_F91_MINI,EMACF91_MINI, XTL=1,UARTDEV0,TCPDEVICE,UDPDEVICE,EMAC,DHCP_REQ,_EZ80ACCLAIM!,RZKMACRODEBUG,RZK_PRIORITYINHERITANCE
- Include Paths: ..\..\Inc\TCPIPCore
- User: ..\..\..\RZK\eZ80F91\Inc\Core;..\..\..\RZK\eZ80F91\Inc\Bsp;..\..\..\RZK\eZ80F91\Inc\FS;..\..\Inc\AppProto;..\..\Inc\Common;..\..\Inc\XTL;

Assembler

- Defines: ZDS1=0,RAM_MAP_ASSEMBLY=1,_INITALIZE_F91_PLL

Linker

- Object/library modules: ..\..\..\RZK\eZ80F91\Lib\Core\RZKDebugPI.lib,..\..\..\RZK\eZ80F91\Lib\Bsp\EMACF91MiniLib.lib,..\..\..\RZK\eZ80F91\Lib\Bsp\UARTF91lib.lib,..\..\..\RZK\eZ80F91\Lib\FS\NOFileSystem.lib,..\..\Lib\common.lib,..\..\Lib\NOdhcp.lib,..\..\Lib\NOdns.lib,..\..\Lib\NoFtpServer.lib,..\..\Lib\nohttp.lib,..\..\Lib\NOigmp.lib,..\..\Lib\NOppp.lib,..\..\Lib\NOrarp.lib,..\..\Lib\NOsmtp.lib,..\..\Lib\NOsnmp.lib,..\..\Lib\telnet.lib,..\..\Lib\NOtftp.lib,..\..\Lib\tty.lib,..\..\Lib\xc.lib,..\..\Lib\XTL.lib,..\..\Lib\ZTPCoreF91Mini.lib, ..\..\Lib\Shell.lib

Target

- Char: 8
- Short: 16
- Float: 32
- Long: 32
- Int: 24
- Double: 32
- Bitfield: 24
- ROM: 0000-3FFFF
- RAM: C00000-C1FFFF
- ExtIO: 0-FFFF
- IntIO: 0-FF
- FlashInfo: 0-FF

Debugger

- Driver: Serial Driver
- Target: EZ80DEVPLATFORM/F91
    - Configure
        - Program Counter: 00000
        - SPL Stack Pointer: FFFFF
        - SPS Stack Pointer: FFFF
        - Chip Select Register: CS0
        - Bound: 000000-03FFFF
        - Control Register: E8
        - Bus Mode 02
        - Start in ADL mode: check
        - Enable Data RAM: check
        - Enable EMAC RAM: check
            - Address Upper Byte: 0
        - Enable Flash:
            - Address Upper Byte: 0
        - Phase-locked loop: check
        - System Clock Frequency: 50000000
        - Oscillator Frequency: 5000000
        - Charge Pump Current: 500uA
        - Lock Criteria: 8 cycles
        - Change ZDI clock upon reset: check

Appendix B: Project Pictures

Figure 4. System overview

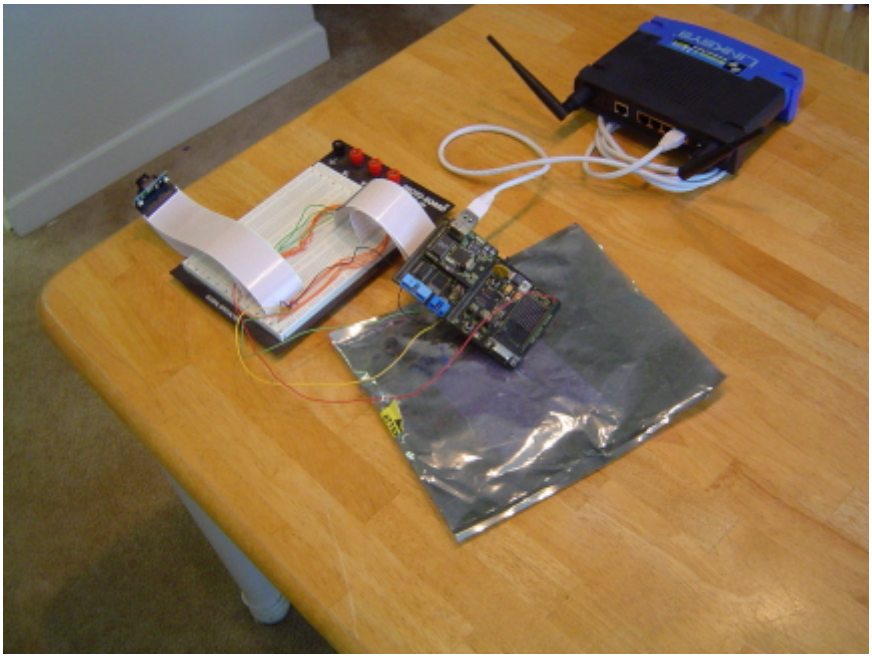


Figure 5. Another system overview

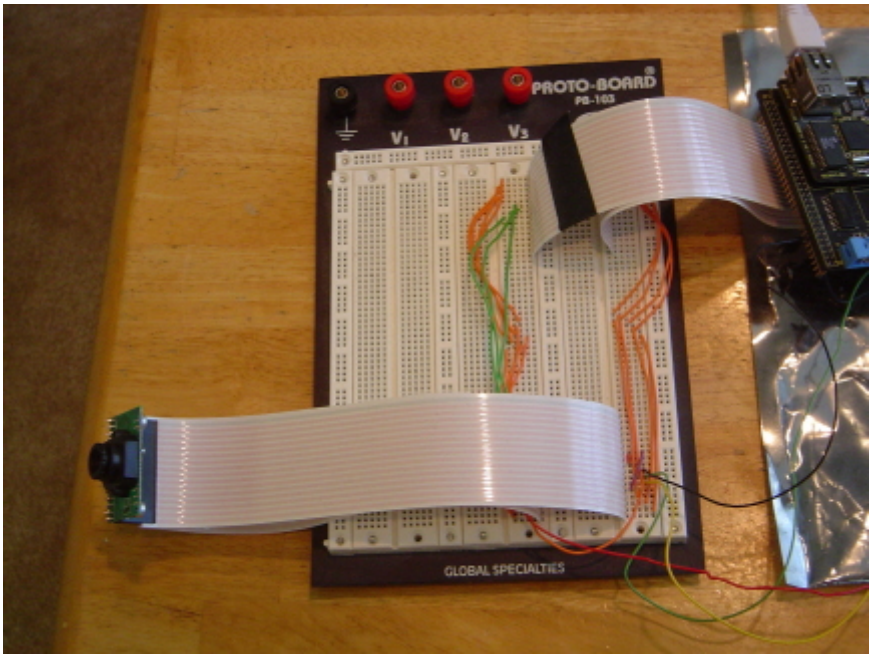


Figure 6. Camera breadboard interface

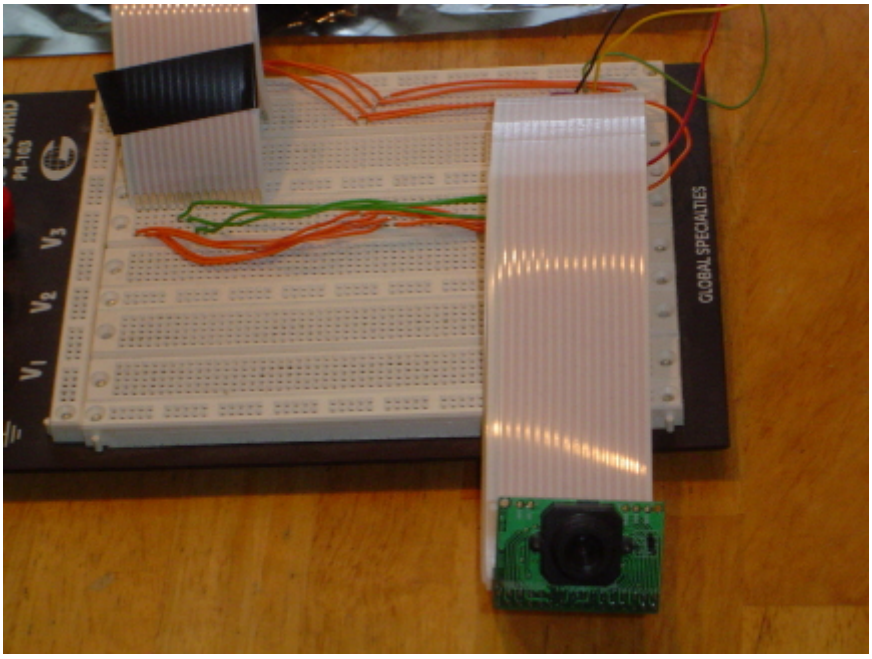


Figure 7. C3088 and breadboard

Figure 8. EZ80F91 to breadboard wiring

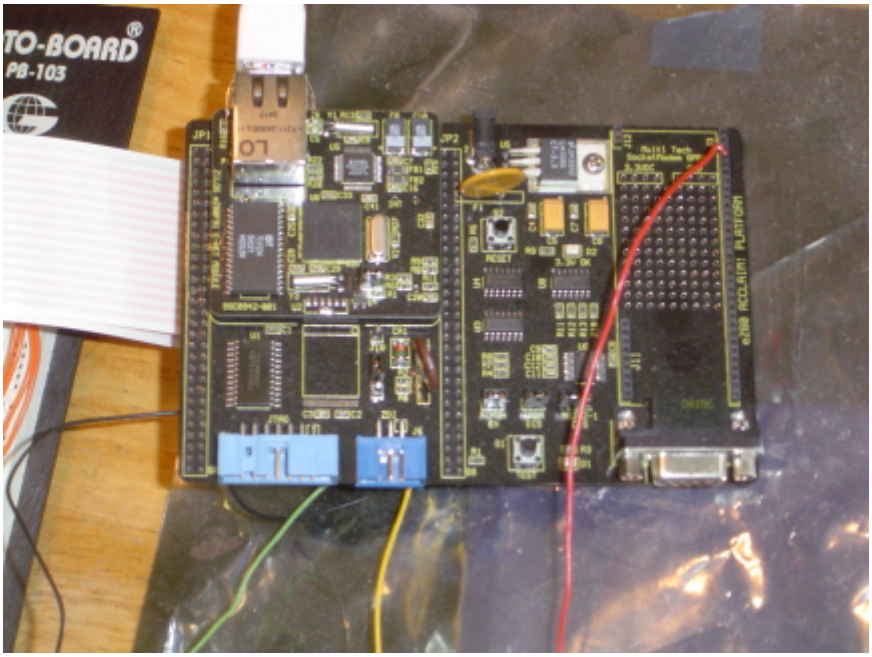


Figure 9. EZ80F91 development board

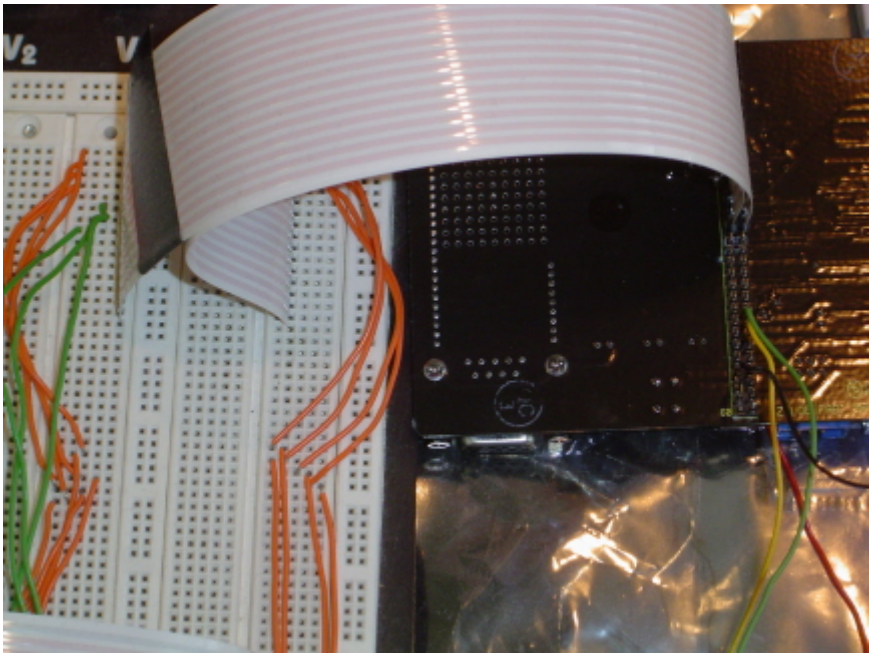


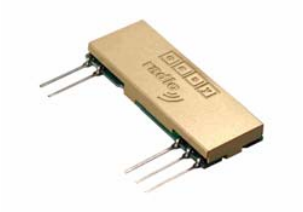
Figure 10. Bottom of the EZ80F91 development board

Low Power Radio Solutions Ltd

"ERx00-02 Series Data Sheet (Rev 2.3).pdf"

Modules Included:

- **ER400TS-02**
- **ER400RS-02**
- **ER400TRS-02**
- **ER900TS-02**
- **ER900RS-02**
- **ER900TRS-02**



Contents

Disclaimer

Low Power Radio Solutions Ltd has an on going policy to improve the performance and reliability of their products; we therefore reserve the right to make changes without notice. The information contained in this data sheet is believed to be accurate however we do not assume any responsibility for errors or any liability arising from the application or use of any product or circuit described herein. This data sheet neither states nor implies warranty of any kind, including fitness for any particular application.

EasyRadio modules are a component part of an end system product and should be treated as such. Testing to fitness is the sole responsibility of the manufacturer of the device into which easyRadio products are fitted, as is also the deployment into the field.
Any liability from defect or malfunction is limited to the replacement of product ONLY, and does not include labour or other incurred corrective expenses.

Using or continuing to use these devices hereby binds the user to these terms.

Introduction to easyRadio

The easyRadio (ER) ERx00TS Transmitter, ERx00RS Receiver and ERx00TRS transceiver incorporate 'easyRadio' technology to provide high performance, simple to use radio devices that can transfer data over a range of up to 250 metres Line Of Sight (LOS). Furthermore 'Easy-Radio' technology allows frequency, data rate and power output to be optimised for customer specific applications. The embedded software reduces design and development time significantly.
ER modules are available in two frequency versions: ER400 series (433-434MHz) & ER900 series (869.85MHz & 902-928MHz). For purposes in this document they will be referred to as ERx00.

2nd Generation Modules (please READ)

This Data Sheet covers the new hardware/firmware revisions of the Easy Radio TS, RS & TRS modules. Their Part numbers are the same as before with the addition of "-02" (eg. ER400TRS-02).
The "02" modules address issues that have been identified in previous hardware. The primary objectives for its development was to solve two main issues:

- Unintentional Loss of user configured settings:
  - The previously used micro-controller (µC) was susceptible to EEPROM corruption when a 'brown-out' condition occurred. Particularly when power cycling the module. This had the effect of triggering the automatic *Reset to Default* behaviour of the module.
- Frequency Stability:
  - Previous data sheets clearly show a frequency tolerance of +-50ppm. Despite operating within this specification, over some batches of product this has caused some unreliable communication to occur, as frequencies have been too far away from each other. If you are having issues in this area, please contact us.
- Back Compatibility:
  - New Modules can be set to be compatible with any frequency variations in the past. Please contact technical for details or consult evaluation software help file.

What improvements are there in '**02**'?

1. The µC has now been changed to one which also has a low-voltage brown-out-detect (BOD) circuit built in. Extensive testing has been done on this, and therefore power cycling techniques to conserve power are now easily achieved without the loss of programmed settings. The new µC also benefits from greater ROM size and peripheral features which means more functionality is being packed into the Easy Radio Modules.
2. The Crystal that controls the RF is now as low as +-10ppm, which guarantees consistent frequency stability over batches and over an extended temperature range of –40°C to +85°C.

Of course some specifications have been effected to allow these hardware changes, but we feel it has been a necessary move to improve the product. LPRS are committed to the continued development and enhancement of Easy Radio and enhancements may be made without prior notice. A list of development hardware and software known issues and changes are listed at the end of this document. Please always make sure you are reading the latest document for update information.

This data sheet describes the electrical and physical characteristics of the device. Operation of the Easy Radio software and Timing Specifications are described later in this document. Further information is available in the 'Easy-Radio Demonstration Kit & Programming Software' guide, which should be read in conjunction with this data sheet.

General Features

Crystal controlled synthesiser for frequency accuracy
High sensitivity receiver – typically –103dBm @ 19.2 Kbps
Up to 10mW Transmit Power (at 434MHz)
Low operating Voltage – 2.5-5.5 Volts – Single Lithium Cell
Low power consumption: Receiver - 21mA
Transmitter - 25mA
Sleep – 120µA (V2.01.6 + later)
User programmable: Frequency of operation
Data Rate
Output Power

Applications

Handheld Terminals
Environmental Sense & Control
Vehicle to Base Station Data Transfer
Remote Data Acquisition
Electronic Point of Sale equipment
Etc

- User Customisable RS232 BAUD rate
  - Literally add any required BAUD rate in the new Easy Radio Evaluation Software and select your new BAUD using the new ER_CMD#U**0** command. *
- RS232 Parity **
  - Odd/even parity is supported**
- FAST ACK feature
  - Reduce the time to issue 'ER' commands, using the HEX06 ASCII ACK character.
- Default Power Settings
  - Set your own power defaults for each channel using lowercase 'p' command.
- Customisable Data ID **
  - Set a 16-bit ID to prevent communication with other Easy Radio Users.
- Programmable Encryption **
  - Secure your data with a user defined 16-bit seed.
- User programmable frequencies
  - Change the channel frequencies to personalised settings. (Via PC Software)
- DCS (Digital Channel Selection)
  - Prevent crosstalk between adjacent channels
- Repeater Mode **
  - Extend transmission range
- 16-bit CRC **
  - Increased error checking

* Note some BAUD rates may require two stop bits. (See Evaluation Software for details on specific BAUD)

** Available on firmware versions 2.01.6 and above

ERx00TS-02 Transmitter

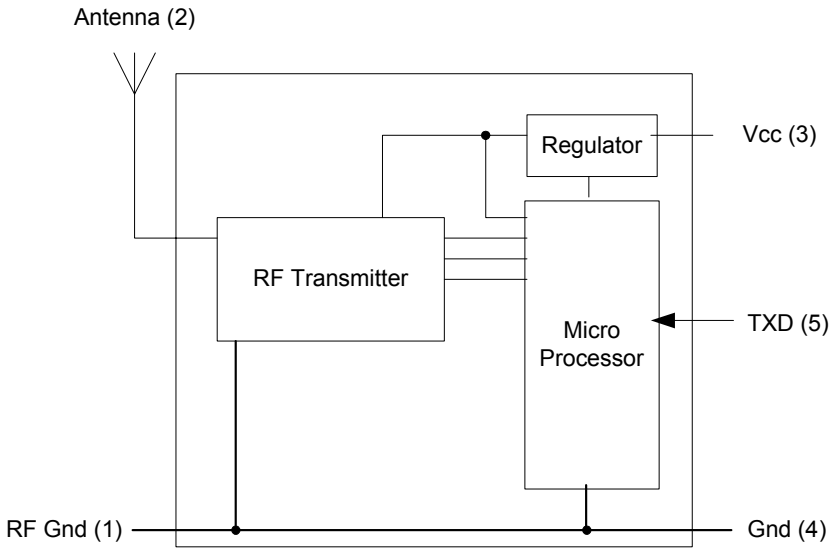


Figure 1 Block Diagram

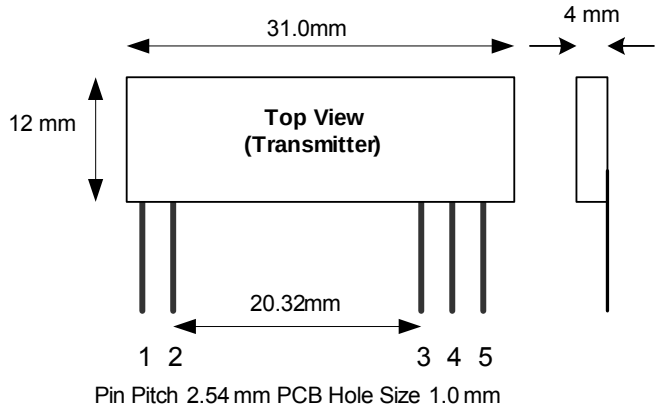


Figure 2 Physical Dimensions

Pin No	Name	Description	Notes
1	RF Gnd	RF ground. Connect to antenna ground (coaxial cable screen braid) and local ground plane. Internally connected to Pin 4	
2	RF Out	50Ω RF output. Connect to suitable antenna	
3	Vcc	Positive supply pin. +2.5 to +5.5 Volts. This should be a 'clean' noise free supply with less than 25mV of ripple	
4	Gnd	Supply 0 Volt and Ground Plane	
5	TXD	Transmit Data Digital Input (SDI)	

Notes

1. The module operates internally from an on board 3.3 Volt low drop regulator.
2. TXD input will be correctly driven by logic operating at 5 Volts (CMOS & TTL logic levels). Input should not be driven by an analogue source.

ERx00RS Receiver

The ERx00RS-02 Receiver is a complete sub-system that combines a high performance low power RF receiver, a 'flash' programmable microcontroller and a voltage regulator (Figure 3). The microcontroller programmes the functions of the RF receiver and provides the interface to the host system via a data output. It also contains programmable EEPROM memory that holds configuration data for the various receiver-operating modes. The microcontroller also relieves the host from the intensive demands of searching for signals within the noise, recovering the received data and then presenting it to the host. A Received Signal Strength Indicator output can be optionally used to measure received signal levels. The module connects to a 50Ω antenna such as a whip, helical or PCB loop.

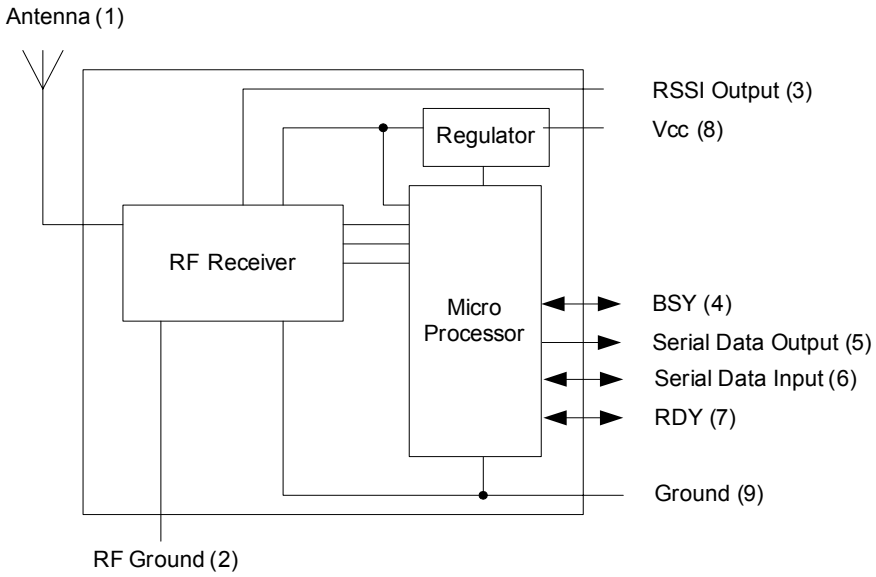


Figure 3 Easy-Radio Receiver Block Diagram

The ERx00RS-02 will receive and decode any Easy-Radio transmission within range and on the same frequency and deliver clean RS232 data to the host system for further processing.

The Serial Data Output operates at programmable standard Baud Rates (default/typical 19,200Baud).

Key parameters (frequency, power output, serial baud rate etc.) of the module may be programmed using the Easy-Radio PC Software via SDI pin 6. (Easy Radio Configuration Command Set, later in this document)

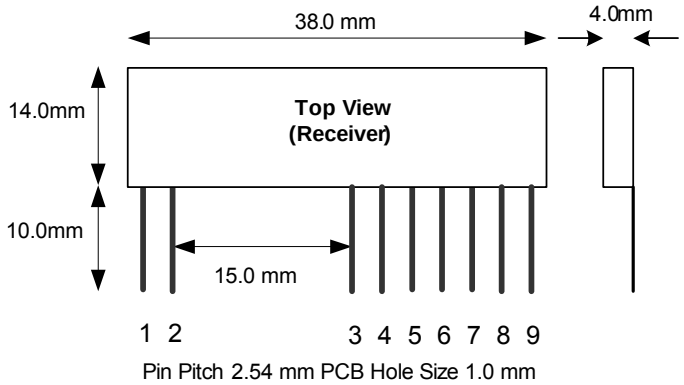


Figure 4 Physical Dimensions

Pin No	Name	Description	Notes
1	Antenna	50Ω RF input/output. Connect to suitable antenna.	
2	RF Ground	RF ground. Connect to antenna ground (coaxial cable screen braid) and local ground plane. Internally connected to other Ground pins.	
3	RSSI	Received Signal Strength Indication - Analogue	
4	BSY	Output (Low - Ready for data from Host) (High - Not Ready)	CTS function
5	Data Out	Received Data Output	SDO
6	Data In	ER command Input	SDI
7	RDY	Input (Low – Host Ready to receive data) (High – Not Ready)	RTS function
8	Vcc	Positive supply pin. +2.5 to +5.5 Volts. This should be a 'clean' noise free supply with less than 25mV of ripple.	
9	Ground	Connect to supply 0 Volt and ground plane	

Checklist

1. The module operates internally from an on board 3.3 Volt low drop regulator. The logic levels of the input/output pins are therefore between 0 Volt and 3.3 Volts. (See RS Performance Data).

2. All digital inputs and outputs are intended for connection to low voltage logic devices. Do not connect any of the inputs or outputs directly to an RS232 port. The receiver module may be permanently damaged by the voltages (+/- 12V) present on RS232 signal lines. See Application Circuit (Figure 11) for typical connection to an RS232 port via MAX232 interface IC.

3. Outputs will drive logic operating at 5 Volts provided the switching levels are less than 3V.

4. If in handshaking mode, pin (7) of the ERx00RS-02 module should be 0V for data to be delivered.

Application & Operation ERx00TS-02 & RS-02

Figure 5 shows a typical system block diagram comprising hosts (user's application) connected to Easy-Radio Transmitters and Receivers. Host (A) will be monitoring (collecting data) and Host (B) will be receiving and processing this data.

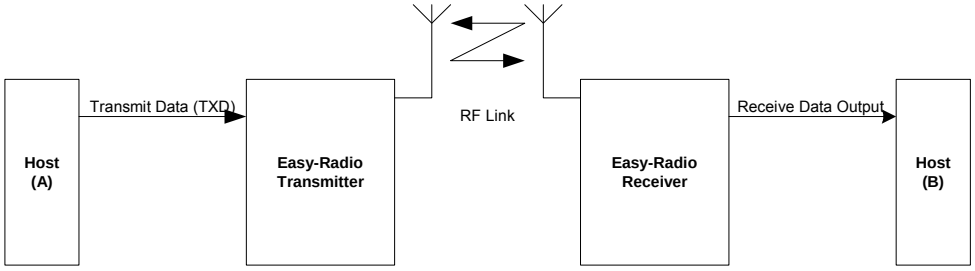


Figure 5 Typical System Block Diagram

The Host (A) should provide the serial data input (up to a maximum 180 characters per packet) to the Easy-Radio transmitter. The data should be sent in 'bursts' therefore allowing adequate time for transmission and reception over the RF link (See Figure 6). The receiver, upon reception and decoding of the RF transmission immediately sends serial data to the Host B.

Data is sent and received in standard 'RS232' serial format (logic level only) and there is no restriction on the characters that may be sent. (HEX 00 – FF)

A. Host (A) sends serial data to the Easy-Radio Transmitter (A). The data must be continuously streamed at the selected baud rate and it loads an internal transmit buffer until either it is full or a gap of two bytes is detected.
B. After detecting either the 'End of Data' gap or the 'Buffer Full' condition the controller enables RF transmit and sends the data in the buffer using Manchester coding for efficient transmission across the RF link. Any Easy-Radio receivers within range that 'hear' the transmission will simultaneously decode the data and place it into their receive buffers.
C. After checking the data for integrity, the Data within the receive buffer of Easy-Radio Receiver (B) is then sent continuously to the host at the selected baud rate.

There is no 'RF handshaking' provided at either the transmitter or receiver. The user should therefore ensure that sufficient time is allowed for the completion of transmission and reception of data. The Timing Specifications detail these requirements (see page 9). Transmitter Host (A) must allow time for the 'Over Air' transmission and for the receiving Host (B) to unload (and process) the data before sending any more new data. The receiver Host (B) must always be 'ready and waiting' for data to arrive. It should be possible to use fast response 'interrupts' without any loss of data.

With such a 'one-way' (simplex) system there is no confirmation of the satisfactory reception of the data and for added reliability it is recommended that the data be sent, perhaps, repetitively several times. For increased reliability the use of transceivers (which can acknowledge packet reception) is recommended. Easy-Radio services do not provide automatic acknowledgement (or re-tries) but these can be provided by the users application.

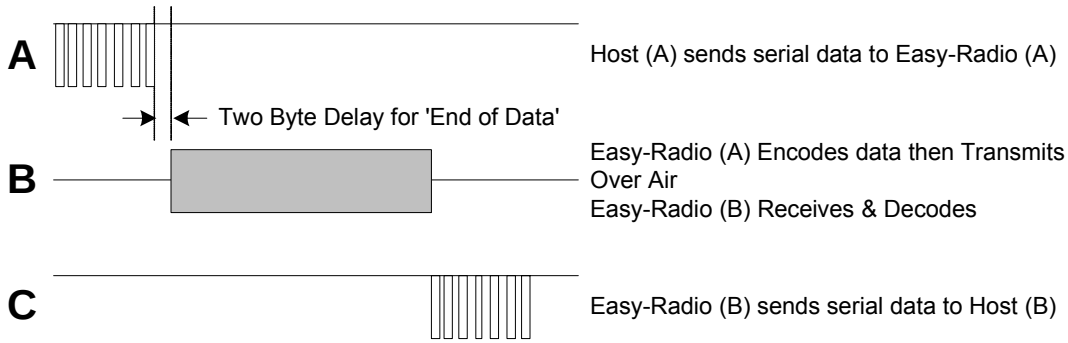


Figure 6 Serial Data

ERx00TRS-02 Transceiver Description

The Easy-Radio Transceiver is a complete sub-system that combines a high performance very low power RF transceiver, a microcontroller and a voltage regulator (Figure 7).

.

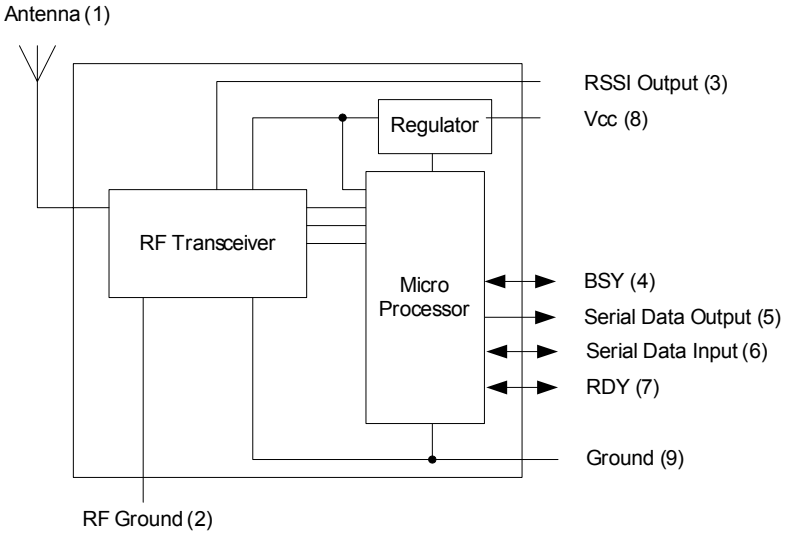


Figure 7 **Easy-Radio Transceiver Block Diagram**

The Serial Data Input and Serial Data Output operate at the standard 19,200 Baud and the two handshake lines provide optional flow control to and from the host. The Easy-Radio Transceiver can accept and transmit up to 180 bytes of data, which it buffers internally before transmitting in an efficient over-air code format.

Any other Easy-Radio Transceiver within range that 'hears' the transmission will decode the message and place the recovered data within a receive buffer that can then be unloaded to the receiving host for processing and interpretation. Transmission and reception are bi-directional half duplex i.e. transmit OR receive but not simultaneously.

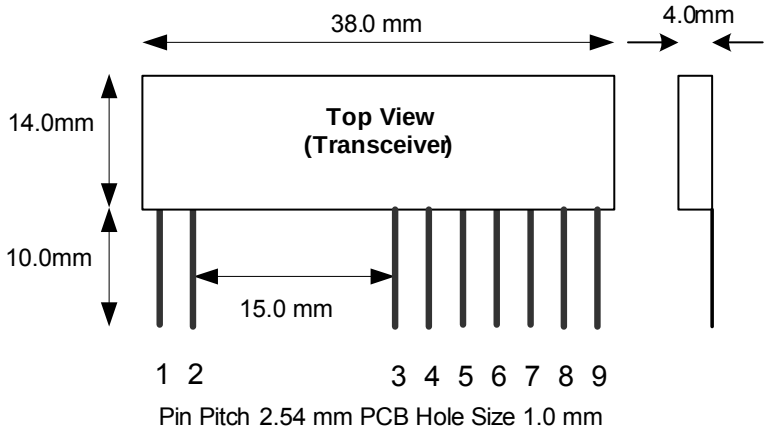


Figure 8 Physical Dimensions

Pin No	Name	Description	Notes
1	Antenna	50Ω RF input/output. Connect to suitable antenna.	
2	RF Ground	RF ground. Connect to antenna ground (coaxial cable screen braid) and local ground plane. Internally connected to other Ground pins.	
3	RSSI	Received Signal Strength Indication	
4	Busy Output	Digital Output to indicate that transceiver is ready to receive serial data from host.	CTS function
5	Serial Data Out	Digital output for received data to host	
6	Serial Data In	Digital input for serial data to be transmitted	
7	Host Ready Input	Digital Input to indicate that Host is Ready to receive serial data from transceiver	RTS function
8	Vcc	Positive supply pin. +2.5 to +5.5 Volts. This should be a 'clean' noise free supply with less than 25mV of ripple.	
9	Ground	Connect to supply 0 Volt and ground plane	

Checklist

1. The module operates internally from an on board 3.3 Volt low drop regulator. The logic levels of the input/output pins are therefore between 0 Volt and 3.3 Volts. (See specifications/performance data).
2. The serial inputs and outputs are intended for connection to a UART or similar low voltage logic device. Do not connect any of the inputs or outputs directly to an RS232 port. The transceiver module may be permanently damaged by the voltages (+/- 12V) present on RS232 signal lines. See Application Circuit (Figure 11) for typical connection to an RS232 port via MAX232 interface IC.
3. The 'Host Ready Input' should be tied to 0 Volt (Ground) if not used, when handshaking is enabled.
4. The 'Serial Data In' should be tied to Vcc if not used. (Receive mode only).
5. Outputs will drive logic operating at 5 Volts and inputs will be correctly driven by logic operating at 5 Volts (CMOS & TTL logic levels).

Application & Operation ERx00TRS-02

Figure 9 shows a typical system block diagram comprising hosts (user's application) connected to Easy-Radio Transceivers. The hosts (A & B) will be monitoring (collecting data) and/or controlling (sending data) to some real world application.

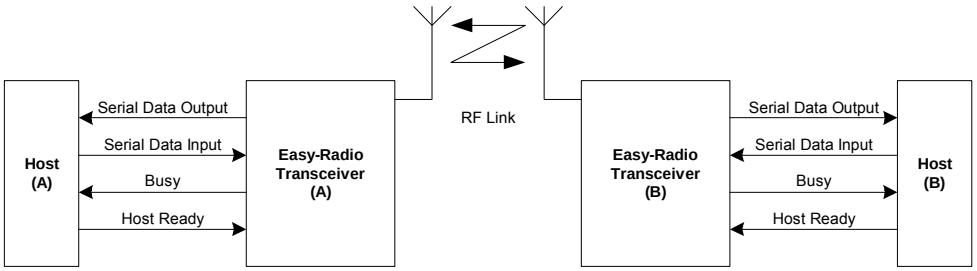


Figure 9 Typical System Block Diagram

The hosts provide serial data input and output lines and two 'handshaking' lines that control the flow of data to and from the Easy-Radio Transceivers. The 'Busy' output line, when active, indicates that the transceiver is undertaking an internal task and is not ready to receive serial data. The 'Host Ready' input is used to indicate that the host is ready to receive the data held in the buffer of the Easy-Radio Transceiver.

The host should check before sending data that the 'Busy' line is not high, as this would indicate that the transceiver is either transmitting or receiving data over the radio link. It should also pull the 'Host Ready' line low and check that no data appears on the Serial Data Output line.

Detailed operation of interfacing, handshaking (including timing) is described in the 'Easy-Radio Software Guide'.

Timing Specifications – Applies to all Easy-Radio Modules.

Parameters	Min	Units	Notes
Host Serial Input/Output	2400, 4800, 9600, 19200, 38400 + Custom	baud	1
Host Character Format	1 Start, 8 Data, No Parity, 1 Stop	Bits	2 &2a
End of Data Delay	2 x BAUD BYTE Duration	mS	3
RF Transmit	13.2 + (n Bytes X 0.8)	mS	4
Buffer Size	1-180	Bytes	5

Notes

1. Data is inverted i.e. Start Bit is logic low. The inputs are intended for direct connection to a microcontroller UART or to RS232 inputs and outputs via an RS232 Level translator such as a Maxim MAX232, which invert the logic of the RS232 signals. This allows direct connection to, for example a Microcontroller UART. The data rate is user programmable (Default 19200 baud) and may differ between individual units within a system. (See Application Circuit diagram for logic level to RS232 interface figure 11).
2. 1 start, 8 data, 1 stop = 10 bits @ 104uS/bit = 0.52mS/character at 19200 Baud. (Default)
    a. Some Custom BAUD rates require 2 stop bits, otherwise some characters may be lost.
    b. If parity is used, substitute in the above calculation using 11 bits.
3. The 'End of Data' delay is fixed at twice the character time.
4. A fixed package overhead of 13.2mS is added to all packets.
5. The buffer size is limited to 180 bytes. Sending more than 180 bytes will cause loss of data.
    a. CTS pin will go high 2 bytes before the buffer is full. This allows characters already sent to be accepted by the ER module.

Absolute Maximum Ratings ERx00TS-02, ERx00RS-02 and ERx00TRS-02

Operating Temperature Range	-40° C to +85° C
Storage Temperature Range	-40° C to +85° C
Vcc	- 0.3 to + 6 Volts
All Other Pins (N.B.)	- 0.3 to 3.3 Volts
Antenna	50V p-p @ < 10MHz

Performance Data: ERx00TS-02 Transmitter Supply +5.0 Volt ± 5%, Temperature 20° C

DC Parameters	Pin	Min	Typical	Max	Units	Notes
Supply Voltage (Vcc)	3	2.5	5.0	5.5	Volts	
Supply current	3		23		mA	1
Interface Levels						
Data Input Logic 1		2.0			Volts	
Data Input Logic 0				0.2	Volts	
Input Impedance			100		KΩ	
RF Parameters	**Pin**	**Min**	**Typical**	**Max**	**Units**	**Notes**
Antenna Impedance	2		50		Ohms	
RF Frequency		433 868 902	434 869.85 915	434 870 928	MHz MHz MHz	See ER Configuration Command set
RF Power Output	2	-5 -5	+10 +7	+10 +7	dBm (434MHz) dBm (869MHz)	50Ω load (depends on frequency)
Frequency accuracy			±10		ppm	Overall
FM deviation			64		KHz	
Harmonics		-43			dBc	Below fundamental
Data Rate		2.4	19.2 (Default)	38.4	Kbps	Custom Over Air BAUD rates can be set via software
Logic Timing	**Pin**	**Min**	**Typical**	**Max**	**Units**	**Notes**
Power Up Time			13		mS	2
Mechanical						
Size		31 x 12 x 4		mm		
Pin Pitch			2.54		mm	Standard 0.1 Inch
Weight			2.5		gms	

Notes

Measure on full RF power. Current will reduce as power decreases.
Time required to 'lock' synthesiser from power up. If not previously calibrated or after a channel change, power up time is increased to 70mS.

Performance Data: ERx00RS-02 Receiver Supply +5.0 Volt ± 5%, Temperature 20° C

DC Parameters	Pin	Min	Typical	Max	Units	Notes
Supply Voltage (Vcc)	8	2.5	5.0	5.5	Volts	
Receive supply current	8		19.5		MA	
Sleep Mode current	8		120		µA	
Interface Levels						
Data Output Logic 1			3.1		Volts	10k load to +Vcc supply
Data Output Logic 0			0.1		Volts	10k load to +Vcc supply
Logic Output Current				25	MA	
Data Input Logic 1		2.0		3.6	Volts	
Data Input Logic 0				0.2	Volts	
Input Pull-ups			100		KΩ	
Antenna Impedance	1		50		Ohms	
RF Frequency		433 868 902	434 869.85 915	434 870 928	MHz MHz MHz	See ER Configuration Command set
Receiver						
Receive Sensitivity		-99	-102	-105	DBm	BER = 10^{-3}
LO leakage			< -60		DBm	Meets EN 300 220-3
Data Rate		2.4	19.2	38.4	Kbps	
RSSI Output	3	0		1.2	Volt	See Figure 10
Logic Timing	**Pin**	**Min**	**Typical**	**Max**	**Units**	**Notes**
Initial Power Up Time			14	75	mS	1
Mechanical						
Size		38 x 14 x 4		Mm		
Pin Pitch			2.54		mm	Standard 0.1 Inches
Weight			3.5		gms	

Notes

1. When power is first applied to the module the processor retrieves 'calibration' data for the RF section that compensates for temperature and power supply voltage variations. The receiver will then be ready to receive. It would normally be left in this powered state ready to receive data.

Performance Data: ERx00TRS transceiver Supply +5.0 Volt ± 5%, Temperature 20° C

DC Parameters	Pin	Min	Typical	Max	Units	Notes
Supply Voltage (Vcc)	8	2.5	5.0	5.5	Volts	
Transmit supply current	8		25		mA	
Receive supply current	8		21		mA	
Sleep Mode current	8		120		µA	
Interface Levels						
Data Output Logic 1			3.1		Volts	10k load to +Vcc supply
Data Output Logic 0			0.1		Volts	10k load to +Vcc supply
Logic Output Current				25	mA	
Data Input Logic 1		2.0			Volts	
Data Input Logic 0				0.2	Volts	
Input Pull-ups			100		KΩ	1
RF Parameters	**Pin**	**Min**	**Typical**	**Max**	**Units**	**Notes**
Antenna Impedance	1		50		Ohms	
RF Frequency		433 868 902	434 869.85 915	434 870 928	MHz MHz MHz	See ER Configuration Command set
Transmitter						
RF Power Output	1	-5 -5	+10 +5	+10 +5	dBm (434MHz) dBm (869MHz)	50Ω load Depends on Frequency
Frequency accuracy			±10		ppm	Overall
FM deviation			64		kHz	
Harmonics			-25		dBc	
Over Air Data rate		2400	19200	38400	bps	Manchester Encoded
Receiver						
Receive Sensitivity		-99	-102	-105	dBm	BER = 10^{-3}
LO leakage			< -60		dBm	Meets EN 300 220-3
Serial Data Rate		0.3	19.2	38.4	Kbps	Host interface. 6
RSSI Output	3	0		1.2	Volt	See Figure 10
Logic Timing	**Pin**	**Min**	**Typical**	**Max**	**Units**	**Notes**
Initial Power Up Time			13	75	mS	2,3
Mechanical						
Size		38 x 14 x 4		Mm		
Pin Pitch			2.54		mm	Standard 0.1 Inches
Weight			3.5		gms	

Notes

1. The 'Host Ready Input' and the 'Serial Data Input' have 'weak' internal pull-ups enabled. These inputs should not however be left 'floating' but should be tied to either Vcc or Ground 0 Volts.
2. When power is first applied to the module the processor retrieves 'calibration' data for the RF section that compensates for temperature and power supply voltage variations. The transceiver will then be ready to receive (default) or transmit. It would normally be left in this powered state ready to receive data.
3. During power up the Busy Output line goes high.

Easy-Radio Configuration Command Set

The programming software sends 'Text Commands' to the modules and this action can be performed by terminal software or the host's Microcontroller using the following list of commands:
Note that shaded items are new **02** commands only.

Command	Function	ER400	ER900	Notes
RS232 Communication Settings				
ER_CMD#U0	Custom BAUD rate	300	300	Programmable via ER Windows Software
ER_CMD#U1	UART Data Rate	2400	2400	
ER_CMD#U2		4800	4800	
ER_CMD#U3		9600	9600	
ER_CMD#U4		19200	19200	
ER_CMD#U5		38400	38400	
ER_CMD#U?	Get UART Value			The module replies echos with the UART value. Eg: ER_CMD#U2 No ACK is required.
ER_CMD#H1	Handshaking ON	OFF	OFF	Only effects RTS Pin.
ER_CMD#H2	Handshaking OFF			
ER_CMD#A70	PARITY DISABLE	DISABLED BY DEFAULT When enabled data = 1 Start, 8 Data, 1 Parity, 1 Stop		
ER_CMD#A71	EVEN PARITY			
ER_CMD#A72	ODD PARITY			
ER_CMD#I7	FAST ACK Enable	OFF	OFF	(Upper case i) See notes on "FAST ACK" below.
ER_CMD#I8	FAST ACK Disable			
RF POWER Settings				
		ER400Series	**ER900 Series**	
ER_CMD#P0	RF Power Output Sets output power on a channel. Warning! This level will be set to the default setting when the frequency is changed or reset via a Channel command.	1mW	0.0625mW (TS)	0.0625mW (TRS)
ER_CMD#P1		2mW	0.125mW (TS)	0.125mW (TRS)
ER_CMD#P2		3mW	0.25mW (TS)	0.25mW (TRS)
ER_CMD#P3		4mW	0.5mW (TS)	0.5mW (TRS)
ER_CMD#P4		5mW	1.2mW (TS)	1mW (TRS)
ER_CMD#P5		6mW	1.5mW (TS)	1.2mW (TRS)
ER_CMD#P6		7mW	2mW (TS)	1.5mW (TRS)
ER_CMD#P7		8mW	3.1mW (TS)	2mW (TRS)
ER_CMD#P8		9mW	4mW (TS)	2.5mW (TRS)
ER_CMD#P9		10mW	5mW (TS)	3.1mW (TRS)
ER_CMD#P?	Get Power Value			The module replies with the power value. eg: ER_CMD#P9 No ACK is required.
ER_CMD#p0	Set **Default** RF Power Output. This allows the host to set each channel to a different default	1mW	0.0625mW	NOTE lower case 'p'
ER_CMD#p1		2mW	0.125mW	
ER_CMD#p2		3mW	0.25mW	
ER_CMD#p3		4mW	0.5mW	
ER_CMD#p4		5mW	0mW	

ER_CMD#p5	power setting. (Ideal for automatic selection between bands like 869 & 914 etc.)	6mW	1mW	
ER_CMD#p6		7mW	2mW	
ER_CMD#p7		8mW	3mW	
ER_CMD#p8		9mW	4mW	
ER_CMD#p9		10mW	5mW	

RF Channel Settings

ER_CMD#C0	Channel 0	433.23 MHz	869.9MHz	All channels can now be chosen in software. Custom frequencies can only be set using software available from LPRS.
ER_CMD#C1	Channel 1	433.30 MHz	914.65MHz	
ER_CMD#C2	Channel 2	433.45 MHz	Not Set	
ER_CMD#C3	Channel 3	433.55 MHz	Not Set	
ER_CMD#C4	Channel 4	433.68 MHz	869.85MHz(i)	
ER_CMD#C5	Channel 5	433.83 MHz	Not Set	
ER_CMD#C6	Channel 6	433.88 MHz	Not Set	
ER_CMD#C7	Channel 7	434.00 MHz	Not Set	
ER_CMD#C8	Channel 8	434.15 MHz	Not Set	
ER_CMD#C9	Channel 9	434.35 MHz	Not Set	
ER_CMD#C?	Get Channel Value			The module replies echos with the power value. Eg: ER_CMD#C9 No ACK is required.
ER_CMD#C**:**	TS ONLY Set Frequency Bank 1			**:** = colon
ER_CMD#C**;**	TS ONLY Set Frequency Bank 2			**;** = semi-colon

MISCELLANEOUS COMMANDS

ER_CMD#R1	Reset to Default Settings	U4, P9, C7	U4, P9, C0	Factory Default
ER_CMD#A00	DCS OFF (default)	Recommended for new designs where back compatibility to older devices is not required		
ER_CMD#A01	DCS ON			
ER_CMD#A10	Encryption OFF (default)	Encryption algorithm is created and owned solely by LPRS. It uses a 16-bit seed that can be set by the developer.		
ER_CMD#A11	Encryption ON			
ER_CMD#A20	CRC16 OFF (default)	The CRC16 routines are more efficient and secure than the old CRC8. For new applications it is recommended.		
ER_CMD#A21	CRC16 ON			
ER_CMD#A30	Repeater Mode OFF (default)	When enabled, this mode will simply echo EVERY message is hears. Therefore no more than 1 repeater should be used UNLESS specific rules are followed. An application note will be available shortly to explain this further.		
ER_CMD#A31	Repeater Mode ON			
ER_CMD#Fds	FREQUENCY SWAP 'd' = destination 's' = source	This feature allows you to transmit on a different channel frequency to the receive frequency. e.g. ER_CMD#F07 After reselecting channel 0, the module will listen on CH0 but transmit on CH7. To reset the module to its default, send the same source channel number to the destination: e.g. ER_CMD#F00 **Warning DCS should be disabled to use this feature.**		

TEST MODES				
ER_CMD#T0	Upper FSK Carrier			Test Mode 0
ER_CMD#T1	Modulated Carrier			Test Mode 1
ER_CMD#T2	Lower FSK Carrier			Test Mode 2
ER_CMD#T3	Get Firmware Revision			Returns Firmware String: eg ER400TRS-02V2.01.5
ER_CMD#T4	RAW Data Test			Out of CTS pin

i) Recommended setting for EN300-220 compliance This setting is effective on ALL ER900TRS/RS/TS despatched on or after 5th September 2005.

To successfully send a command do the following:

1. Send Command from host: e.g. ER_CMD#U5 (Set UART BAUD to 38400)
2. In the case of a TRS/RS:
   - Wait for echo of command from module. e.g. ER_CMD#U5

   In the case of a TS:
   - Wait 20mS
3. Send the ASCII string from the host: ACK

The commands should be sent exactly as shown (case sensitive) with no spaces between characters. The ACK command is sent as three ASCII characters, ACK in sequence. 'A''C''K' .
Note that the TS (transmitter) devices send data 'over air' as they are not equipped with a serial data out or handshake pins. This takes approximately 20mS and time should be taken in to account before sending the 'ACK' sequence

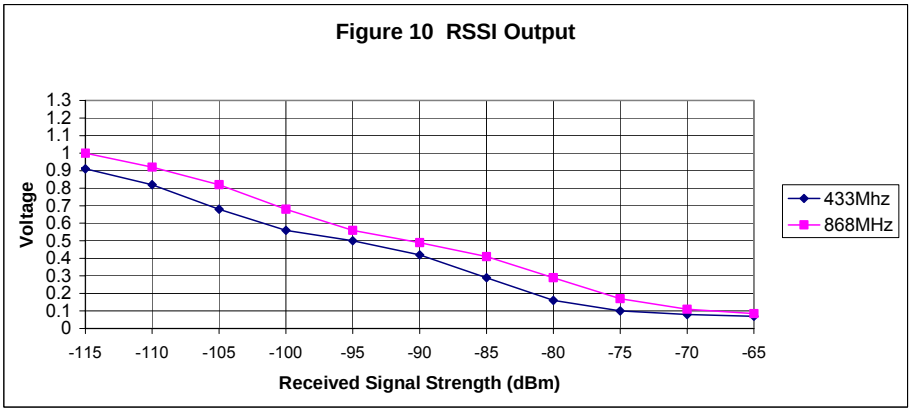
"FAST ACK"

In this mode the procedure to update settings is made much faster. In response to an ER_CMD#x string the TRS/RS module will reply with a single HEX 6 (0x06) which is the ASCII ACK value. The host will then issue the same single byte 0x06 in replacement of the Txt version of "ACK".

Notes:

The Receiver/Transceiver has a built in RSSI (Received Signal Strength Indicator) that provides an analogue output voltage that is inversely proportional to the RF energy present within the pass band of the receiver. It ranges from 0 Volt (maximum signal, –65dBm) to 1 Volt (minimum signal, -115bBm) and has a slope of approximately 50dB/Volt. This analogue output signal should only be connected to a high impedance load (>100kΩs) and can be used to provide a measure of the signal strength and any interfering signals (noise) within band during the installation and operation of systems.

Figure 10 RSSI Output



MAX232 Application

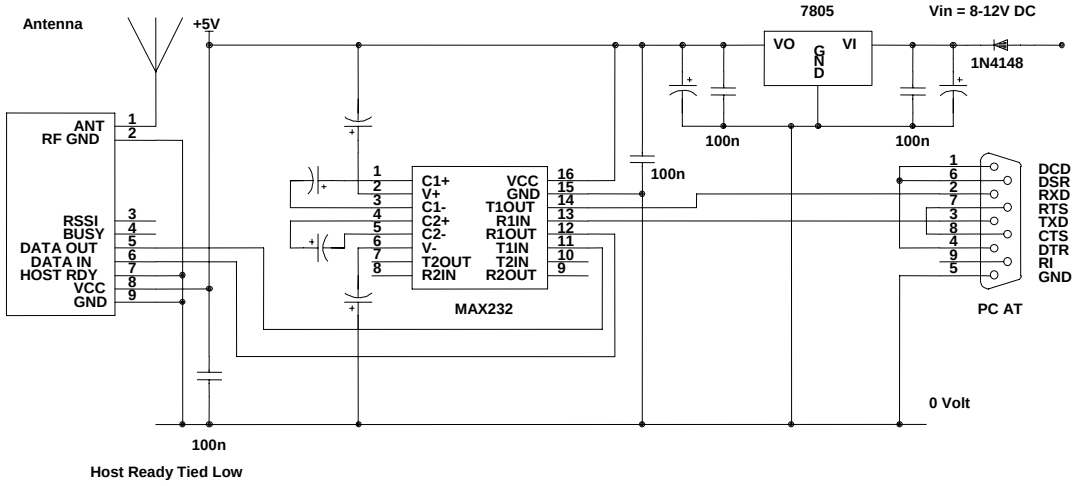


Figure 11 MAX232 Application Circuit

The ERx00TS, ERx00RS & ERx00TRS use crystal controlled synthesisers to accurately define transmit and receive frequencies incorporating RS232 protocols, and so should not be used in connection with Non-Easy Radio RF modules unless the firmware allows.

The Ground (0 Volt) pins of the receiver should be connected to a substantial ground plane (large area of PCB copper) connected to 0 Volt. It is suggested that a double sided PCB be used with one layer being the ground plane.

The supply used to power the receiver should be 'clean' and free from ripple and noise (<20mV p-p total). It is suggested that 100nF ceramic capacitors be used to de-couple the supply close to the power pins of the receiver. The use of 'switch mode' power supplies should be avoided as they can generate both conducted and radiated high frequency noise that can be very difficult to eliminate. This noise may considerably reduce the performance of any radio device that is connected or adjacent to the supply.

The receiver can be used with the various common types of antenna that match the 50Ω RF Input/Output such as a monopole (whip), helical or PCB/Wire loop antennas.

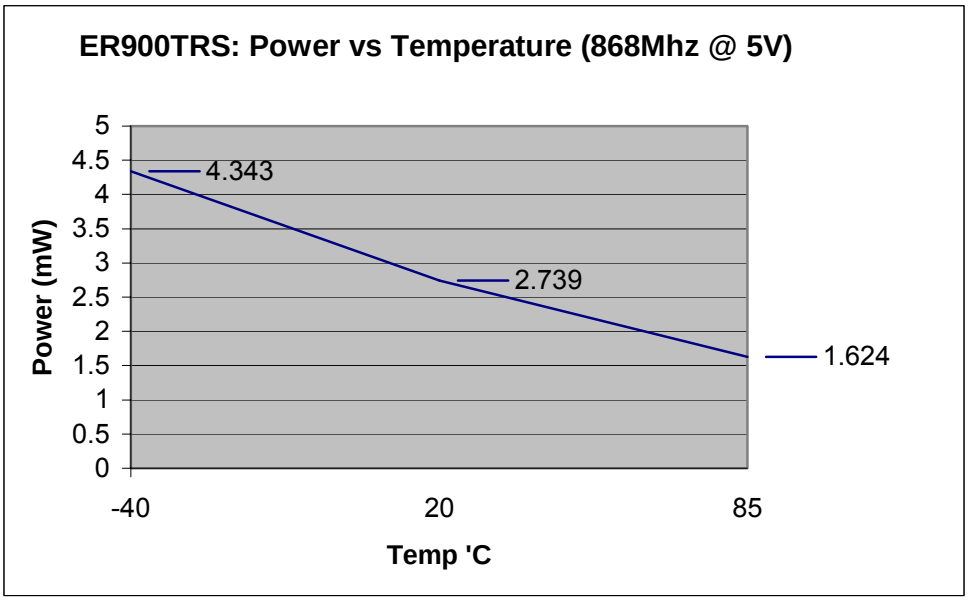
Monopole antennas are resonant with a length corresponding to one quarter of the electrical wavelength ($\lambda/4$). They are very easy to implement and can simply be a 'piece of wire' or PCB track which at 434MHz should be 16.4cms in length. This should be straight, in 'free space' (kept well away from all other circuitry) and should be connected directly to the Antenna pin of the receiver. If the antenna is remote it should be connected via a 50Ω coaxial feeder cable or transmission line. A 50Ω transmission line can be constructed on FR4 board material by using a 3mm wide PCB track over a ground plane. This should be kept as short as possible.

Helical antennas are also resonant and generally chosen for their more compact dimensions. They are more difficult to optimise than monopole antennas and are critical with regard to surrounding objects that can easily 'de-tune' them. They operate most efficiently when there is a substantial ground plane for them to radiate against.

Wire or PCB Loop antennas are the most compact antennas but are less effective than the other types. They are also more difficult to design and must be carefully 'tuned' for best performance.

The Internet can provide much useful information on the design of Short Range Device (SRD) Antennas.

Technical Characteristics



easyRadio errata notes:

(LPRS ERRATA-001.pdf)
This Errata sheet applies to all ER400TRS-02 & ER400RS-02 modules shipped prior to 4th May 2005.

ISSUE:
Frequency Channels 1, 4 & 9 may have poor or no communication.

DESCRIPTION:
The receiving frequency channels 1, 4 & 9 which can be selected using the easyRadio command structure, ("ER_CMD#Cx" where x is the channel number) as set by the factory may be sufficiently off frequency to cause poor communication and in worst cases none at all on these channels.
All other channels are unaffected.
This has been caused during production where incorrect values have been set in the EEPROM locations that control each channel frequency.
This is NOT a software bug, but rather a setting used by the firmware. A couple of workarounds can be suggested if this poses an issue, when frequencies other than the defaults are used.

Work Around:
i) Do not use channels 1,4, or 9.
ii) Update module settings.

The latest easyRadio Evaluation software (Version 2 and above) incorporates the necessary settings to update the module and is available for download via http://www.lprs.co.uk
As this is NOT a firmware update, the firmware version will remain unaffected.

(LPRS ERRATA-002.pdf)
This Errata sheet applies to all ER400TRS-02 & ER400RS-02 modules with firmware up to V2.01.5

ISSUE:
RTS (Host Ready) input, does not allow restart of data flow when set low after being held high.

DESCRIPTION:
When Handshaking it enabled, the data received over air will only be delivered if the RTS pin is held low continually.
Any break in the sequence (ie. RTS being set) will prevent remaining data being streamed.
The module may also reset, causing a recalibration to take place.
No settings will be lost and reception will carry on transparent to the host.

Work Around:
1. Do not enable the handshaking mode, then:
2. Use the CTS to prepare the system for data and dedicate time to receive the data.

This issue will be resolved in the next firmware revision (V2.01.6 & above)

(LPRS ERRATA-003.pdf)
This Errata sheet applies to all ER4/900TRS-02 modules up to and including firmware V2.01.5

ISSUE:
Radio appears to stop responding after an undefined period.

DESCRIPTION:
When a host device is dependant on receiving a signal from another remote unit before transmitting and usually when a lot of data is being exchanged (ie multiple transmissions & receptions per second over long periods), occasionally the PLL may not lock in frequency when returning to RX mode after a transmission. In our tests this may only happen after thousands of exchanges, typically above 5000, although this number is not definitive.

Work Around:
1) Test for a lock condition:
Send the command:
ER_CMD#L30D?
After the final "ACK" the module will return a two-byte ASCII number. E.g. 83.
The '3' is the important number. Any other number will show the PLL is not locked and needs to be reset.
Either:

i. Send a 'Dummy' transmission. This will force the RX to reset again after the transmission.
ii. Send a channel update (eg. ER_CMD#C7 etc)
iii. Reset Module through a power on reset

2) If you do not wish to test for a lock condition, you may be able to detect (through timeouts in your own software etc) that a problem has occurred, and perform one of the resolutions as described in 1).

This issue will be resolved in the next firmware revision (V2.01.6 & above)

Product Order Codes

Name	Description	Order Code
Easy-Radio 400 Transmitter	UK/European Transmitter Module on 433 MHz	ER400TS-02
Easy-Radio 400 Receiver	UK/European Receiver Module on 433 MHz	ER400RS-02
Easy-Radio 400 Transceiver	UK/European Transceiver Module on 433 MHz	ER400TRS-02
Easy-Radio 900 Transmitter	Europe/US Transmitter Module 869/915MHZ	ER900TS-02
Easy-Radio 900 Receiver	Europe/US Receiver Module 869/915MHZ	ER900RS-02
Easy-Radio 900 Transceiver	Europe/US Transceiver Module 869/915MHZ	ER900TRS-02

Please contact the sales office for availability and other variants of the standard product. The software interface can be customised to specific requirements for high volume applications.

Easy-Radio Module Firmware Version

Version	Date	Revision	Known Issues
2.01.7	Sept 2005	**Fixes:** Busy pin sets when waking from sleep	None
2.01.6	Sept 2005	**Fixes:** (See errata for published issues) RTS issue Frequency lock issue. Frequency settings updated. **Features added:** Encryption, 16-Bit CRC, Repeater, RS232 Parity, Sleep mode	Busy will not set while waking from sleep. Workaround: wait 14mS before sending data after waking from sleep.
2.01.5	March 2005	Fixed 2.01.4 Issue Added Default Power settings using lowercase 'p' (See Command Table)	See errata sheets 001-003
2.01.4	Feb 2005	Fixed RSSI. (Previous Releases did not work correctly)	ER_CMD#R1 Reset to defaults sets UART/POWER/CHANNEL all to 0
2.01.3	Jan 2005	Initial Release	

Document History

Issue	Date	Revision
2.3	September 2005	Corrected Voltage and other minor Specifications. Added Temperature Characteristic table, Contents, Command list items, errata.
2.2	March 2005	Removed Provisional status. Updated commands & feature lists.
2.1	Feb 2005	First Provisional Datasheet for '02' series modules

Copyright

Certificates of Conformance to EN300-220 Part3

The following pages contain the certificates of conformance to the regulations as set out in the ETSI documentation for EN300-220, and the essential requirements, as laid out within part 3.

You may add these certificates to your own technical construction file under the provisions provided within article 3 of the Directive 1999/5/EC.

However, during design, care must be taken so that there is no amplification made to the module, either by use of an antenna with gain or an external RF amplifier. Any such changes or modifications will make these certificates invalid, and further testing will remain the responsibility of the final product.

Low Power Radio Solutions Ltd
Two Rivers Industrial Estate
Station Lane
Witney
Oxon. OX8 6BH
UK
✉ info@lprs.co.uk

Certificate of Conformance

Date: 8th July 2005

Module Family: easy-Radio

Type: ER400TS-02 and ER900TS-02

Frequency: 433.23 – 434.35MHz (ER400TS-02)
869.7MHz – 870MHz (ER900TS-02)

Description: Low Power Radio Transmitter

Intended Purpose: RF Short Range Devices

Equipment Class: Class 3 Radio Communications Equipment

We hereby declare the above listed product complies with the essential requirements of article 3 and other relevant provisions of the Directive 1999/5/EC when used for its intended purpose, as self-certified and tested internally by LPRS Ltd under the provisions provided within article 3 of the Directive 1999/5/EC.

Measure for the efficient use of the radio frequency spectrum pursuant to Article 3.2

Standards applied: ETS 300 220-3 09/2000

This certificate does not cover safety, fitness for any purpose or any requirements other than those stated

Signed for and on behalf of
Low Power Radio Solutions Ltd

……..

Low Power Radio Solutions Ltd
Two Rivers Industrial Estate
Station Lane
Witney
Oxon. OX8 6BH
UK
✉ info@lprs.co.uk

Certificate of Conformance

Date: 8th July 2005

Module Family: easy-Radio

Type: ER400RS-02 and ER900RS-02

Frequency: 433.23 – 434.35MHz (ER400RS-02)
869.7MHz – 870MHz (ER900RS-02)

Description: Low Power Radio Receiver

Intended Purpose: RF Short Range Devices

Equipment Class: Class 3 Radio Communications Equipment

We hereby declare the above listed product complies with the essential requirements of article 3 and other relevant provisions of the Directive 1999/5/EC when used for its intended purpose, as self-certified and tested internally by LPRS Ltd under the provisions provided within article 3 of the Directive 1999/5/EC.

Measure for the efficient use of the radio frequency spectrum pursuant to Article 3.2

Standards applied: ETS 300 220-3 09/2000

This certificate does not cover safety, fitness for any purpose or any requirements other than those stated

Signed for and on behalf of
Low Power Radio Solutions Ltd

……..

Low Power Radio Solutions Ltd
Two Rivers Industrial Estate
Station Lane
Witney
Oxon. OX8 6BH
UK
✉ info@lprs.co.uk

Certificate of Conformance

Date:	6th July 2005
Module Family:	easy-Radio
Type:	ER400TRS-02 and ER900TRS-02
Frequency:	433.23 – 434.35MHz (ER400TRS-02) 869.7MHz – 870MHz (ER900TRS-02)
Description:	Low Power Radio Transceiver
Intended Purpose:	RF Short Range Devices
Equipment Class:	Class 3 Radio Communications Equipment

We hereby declare the above listed product complies with the essential requirements of article 3 and other relevant provisions of the Directive 1999/5/EC when used for its intended purpose, as self-certified and tested internally by LPRS Ltd under the provisions provided within article 3 of the Directive 1999/5/EC.

Measure for the efficient use of the radio frequency spectrum pursuant to Article 3.2

Standards applied: ETS 300 220-3 09/2000

This certificate does not cover safety, fitness for any purpose or any requirements other than those stated

Signed for and on behalf of
Low Power Radio Solutions Ltd

……..

Disclaimer

Low Power Radio Solutions Ltd has an on going policy to improve the performance and reliability of their products; we therefore reserve the right to make changes without notice. The information contained in this data sheet is believed to be accurate however we do not assume any responsibility for errors or any liability arising from the application or use of any product or circuit described herein. This data sheet neither states nor implies warranty of any kind, including fitness for any particular application.

EasyRadio modules are a component part of an end system product and should be treated as such. Testing to fitness is the sole responsibility of the manufacturer of the device into which easyRadio products are fitted, as is also the deployment into the field.
Any liability from defect or malfunction is limited to the replacement of product ONLY, and does not include labour or other incurred corrective expenses.

Using or continuing to use these devices hereby binds the user to these terms.

CONTACT INFORMATION

For further information or technical assistance please contact:

Low Power Radio Solutions Ltd
Two Rivers Industrial Estate
Station Lane
Witney
Oxon
OX28 4BH
England

Tel: +44 (0)1993 709418
Fax: +44 (0)1993 708575
Web: http://www.lprs.co.uk
Email: info@lprs.co.uk

Web: http://www.easy-radio.co.uk

The above address is a dedicated web site for Easy-Radio

19-0123; Rev. 4; 8/96

EVALUATION KIT AVAILABLE

Low-Power, 8-Channel, Serial 12-Bit ADCs

General Description

The MAX186/MAX188 are 12-bit data-acquisition systems that combine an 8-channel multiplexer, high-bandwidth track/hold, and serial interface together with high conversion speed and ultra-low power consumption. The devices operate with a single +5V supply or dual ±5V supplies. The analog inputs are software configurable for unipolar/bipolar and single-ended/differential operation.

The 4-wire serial interface directly connects to SPI™, QSPI™ and Microwire™ devices without external logic. A serial strobe output allows direct connection to TMS320 family digital signal processors. The MAX186/MAX188 use either the internal clock or an external serial-interface clock to perform successive-approximation A/D conversions. The serial interface can operate beyond 4MHz when the internal clock is used.

The MAX186 has an internal 4.096V reference while the MAX188 requires an external reference. Both parts have a reference-buffer amplifier that simplifies gain trim .

The MAX186/MAX188 provide a hard-wired $\overline{SHDN}$ pin and two software-selectable power-down modes. Accessing the serial interface automatically powers up the devices, and the quick turn-on time allows the MAX186/MAX188 to be shut down between every conversion. Using this technique of powering down between conversions, supply current can be cut to under 10µA at reduced sampling rates.

The MAX186/MAX188 are available in 20-pin DIP and SO packages, and in a shrink small-outline package (SSOP), that occupies 30% less area than an 8-pin DIP. For applications that call for a parallel interface, see the MAX180/MAX181 data sheet. For anti-aliasing filters, consult the MAX274/MAX275 data sheet.

Applications

Portable Data Logging

Data-Acquisition

High-Accuracy Process Control

Automatic Testing

Robotics

Battery-Powered Instruments

Medical Instruments

SPI and QSPI are registered trademarks of Motorola.
Microwire is a registered trademark of National Semiconductor.

Features

- ♦ **8-Channel Single-Ended or 4-Channel Differential Inputs**
- ♦ **Single +5V or ±5V Operation**
- ♦ **Low Power: 1.5mA (operating mode)**
  2µA (power-down mode)
- ♦ **Internal Track/Hold, 133kHz Sampling Rate**
- ♦ **Internal 4.096V Reference (MAX186)**
- ♦ **SPI-, QSPI-, Microwire-, TMS320-Compatible 4-Wire Serial Interface**
- ♦ **Software-Configurable Unipolar or Bipolar Inputs**
- ♦ **20-Pin DIP, SO, SSOP Packages**
- ♦ **Evaluation Kit Available**

Ordering Information

PART†	TEMP. RANGE	PIN-PACKAGE
MAX186_CPP	0°C to +70°C	20 Plastic DIP
MAX186_CWP	0°C to +70°C	20 SO
MAX186_CAP	0°C to +70°C	20 SSOP
MAX186DC/D	0°C to +70°C	Dice*
MAX186_EPP	-40°C to +85°C	20 Plastic DIP
MAX186_EWP	-40°C to +85°C	20 SO
MAX186_EAP	-40°C to +85°C	20 SSOP
MAX186_MJP	-55°C to +125°C	20 CERDIP**

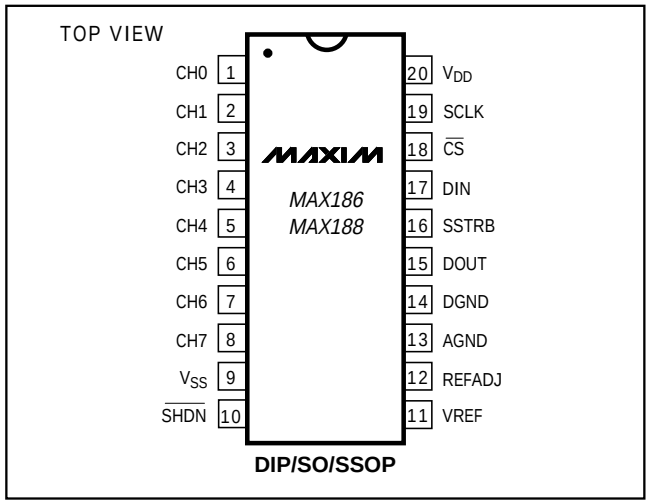
Ordering Information continued on last page.

† NOTE: Parts are offered in grades A, B, C and D (grades defined in Electrical Characteristics). When ordering, please specify grade. Contact factory for availability of A-grade in SSOP package.

** Dice are specified at +25°C, DC parameters only.*

*** Contact factory for availability and processing to MIL-STD-883.*

Pin Configuration



Low-Power, 8-Channel, Serial 12-Bit ADCs

ABSOLUTE MAXIMUM RATINGS

V_{DD} to AGND .....-0.3V to +6V
V_{SS} to AGND .....+0.3V to -6V
V_{DD} to V_{SS} .....-0.3V to +12V
AGND to DGND .....-0.3V to +0.3V
CH0–CH7 to AGND, DGND .....(V_{SS} - 0.3V) to (V_{DD} + 0.3V)
CH0–CH7 Total Input Current .....±20mA
VREF to AGND .....-0.3V to (V_{DD} + 0.3V)
REFADJ to AGND .....-0.3V to (V_{DD} + 0.3V)
Digital Inputs to DGND .....-0.3V to (V_{DD} + 0.3V)
Digital Outputs to DGND .....-0.3V to (V_{DD} + 0.3V)
Digital Output Sink Current 25mA

Continuous Power Dissipation (T_A = +70°C)
Plastic DIP (derate 11.11mW/°C above +70°C) 889mW
SO (derate 10.00mW/°C above +70°C) 800mW
SSOP (derate 8.00mW/°C above +70°C) 640mW
CERDIP (derate 11.11mW/°C above +70°C) 889mW
Operating Temperature Ranges:
MAX186_C/MAX188_C .....0°C to +70°C
MAX186_E/MAX188_E .....-40°C to +85°C
MAX186_M/MAX188_M .....-55°C to +125°C
Storage Temperature Range .....-60°C to +150°C
Lead Temperature (soldering, 10sec) .....+300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS

(V_{DD} = 5V ±5%; V_{SS} = 0V or -5V; f_{CLK} = 2.0MHz, external clock (50% duty cycle); 15 clocks/conversion cycle (133ksps); MAX186—4.7µF capacitor at VREF pin; MAX188—external reference, VREF = 4.096V applied to VREF pin; T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
DC ACCURACY (Note 1)							
Resolution				12			Bits
Relative Accuracy (Note 2)		MAX186A/MAX188A				±0.5	LSB
		MAX186B/MAX188B				±0.5	
		MAX186C				±1.0	
		MAX188C				±0.75	
		MAX186D/MAX188D				±1.0	
Differential Nonlinearity	DNL	No missing codes over temperature				±1	LSB
Offset Error		MAX186A/MAX188A				±2.0	LSB
		MAX186B/MAX188B				±3.0	
		MAX186C/MAX188C				±3.0	
		MAX186D/MAX188D				±3.0	
Gain Error (Note 3)		MAX186 (all grades)				±3.0	LSB
		External reference 4.096V (MAX188)	MAX188A			±1.5	
			MAX188B			±2.0	
			MAX188C			±2.0	
			MAX188D			±3.0	
Gain Temperature Coefficient		External reference, 4.096V			±0.8		ppm/°C
Channel-to-Channel Offset Matching					±0.1		LSB
DYNAMIC SPECIFICATIONS (10kHz sine wave input, 4.096V_{P-P}, 133ksps, 2.0MHz external clock, bipolar input mode)							
Signal-to-Noise + Distortion Ratio	SINAD			70			dB
Total Harmonic Distortion (up to the 5th harmonic)	THD					-80	dB
Spurious-Free Dynamic Range	SFDR			80			dB
Channel-to-Channel Crosstalk		65kHz, V_{IN} = 4.096V_{P-P} (Note 4)			-85		dB

ELECTRICAL CHARACTERISTICS (continued)

(V_{DD} = 5V ±5%; V_{SS} = 0V or -5V; f_{CLK} = 2.0MHz, external clock (50% duty cycle); 15 clocks/conversion cycle (133ksps); MAX186—4.7µF capacitor at VREF pin; MAX188—external reference, VREF = 4.096V applied to VREF pin; T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
Small-Signal Bandwidth		-3dB rolloff			4.5		MHz
Full-Power Bandwidth					800		kHz
CONVERSION RATE							
Conversion Time (Note 5)	t_{CONV}	Internal clock		5.5		10	µs
		External clock, 2MHz, 12 clocks/conversion		6			
Track/Hold Acquisition Time	t_{AZ}					1.5	µs
Aperture Delay					10		ns
Aperture Jitter					<50		ps
Internal Clock Frequency					1.7		MHz
External Clock Frequency Range		External compensation, 4.7µF		0.1		2.0	MHz
		Internal compensation (Note 6)		0.1		0.4	
		Used for data transfer only			10		
ANALOG INPUT							
Input Voltage Range, Single-Ended and Differential (Note 9)		Unipolar, V_{SS} = 0V				0 to VREF	V
		Bipolar, V_{SS} = -5V				±VREF/2	
Multiplexer Leakage Current		On/off leakage current, V_{IN} = ±5V			±0.01	±1	µA
Input Capacitance		(Note 6)			16		pF
INTERNAL REFERENCE (MAX186 only, reference buffer enabled)							
VREF Output Voltage		T_A = +25°C		4.076	4.096	4.116	V
VREF Short-Circuit Current						30	mA
VREF Tempco		MAX186A, MAX186B, MAX186C	MAX186_C		±30	±50	ppm/°C
			MAX186_E		±30	±60	
			MAX186_M		±30	±80	
		MAX186D			±30		
Load Regulation (Note 7)		0mA to 0.5mA output load			2.5		mV
Capacitive Bypass at VREF		Internal compensation		0			µF
		External compensation		4.7			
Capacitive Bypass at REFADJ		Internal compensation		0.01			µF
		External compensation		0.01			
REFADJ Adjustment Range					±1.5		%
EXTERNAL REFERENCE AT VREF (Buffer disabled, VREF = 4.096V)							
Input Voltage Range				2.50		V_{DD} + 50mV	V
Input Current					200	350	µA
Input Resistance				12	20		kΩ
Shutdown VREF Input Current					1.5	10	µA
Buffer Disable Threshold REFADJ				V_{DD} - 50mV			V

ELECTRICAL CHARACTERISTICS (continued)

(V_{DD} = 5V ±5%; V_{SS} = 0V or -5V; f_{CLK} = 2.0MHz, external clock (50% duty cycle); 15 clocks/conversion cycle (133ksps); MAX186—4.7µF capacitor at VREF pin; MAX188—external reference, VREF = 4.096V applied to VREF pin; T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
EXTERNAL REFERENCE AT REFADJ						
Capacitive Bypass at VREF		Internal compensation mode	0			µF
		External compensation mode	4.7			
Reference-Buffer Gain		MAX186		1.678		V/V
		MAX188		1.638		
REFADJ Input Current		MAX186			±50	µA
		MAX188			±5	
DIGITAL INPUTS (DIN, SCLK, $\overline{CS}$, $\overline{SHDN}$)						
DIN, SCLK, $\overline{CS}$ Input High Voltage	V_{INH}		2.4			V
DIN, SCLK, $\overline{CS}$ Input Low Voltage	V_{INL}				0.8	V
DIN, SCLK, $\overline{CS}$ Input Hysteresis	V_{HYST}			0.15		V
DIN, SCLK, $\overline{CS}$ Input Leakage	I_{IN}	V_{IN} = 0V or V_{DD}			±1	µA
DIN, SCLK, $\overline{CS}$ Input Capacitance	C_{IN}	(Note 6)			15	pF
$\overline{SHDN}$ Input High Voltage	V_{INH}		V_{DD} - 0.5			V
$\overline{SHDN}$ Input Low Voltage	V_{INL}				0.5	V
$\overline{SHDN}$ Input Current, High	I_{INH}	$\overline{SHDN}$ = V_{DD}			4.0	µA
$\overline{SHDN}$ Input Current, Low	I_{INL}	$\overline{SHDN}$ = 0V	-4.0			µA
$\overline{SHDN}$ Input Mid Voltage	V_{IM}		1.5		V_{DD} -1.5	V
$\overline{SHDN}$ Voltage, Floating	V_{FLT}	$\overline{SHDN}$ = open		2.75		V
$\overline{SHDN}$ Max Allowed Leakage, Mid Input		$\overline{SHDN}$ = open	-100		100	nA
DIGITAL OUTPUTS (DOUT, SSTRB)						
Output Voltage Low	V_{OL}	I_{SINK} = 5mA			0.4	V
		I_{SINK} = 16mA		0.3		
Output Voltage High	V_{OH}	I_{SOURCE} = 1mA	4			V
Three-State Leakage Current	I_L	$\overline{CS}$ = 5V			±10	µA
Three-State Output Capacitance	C_{OUT}	$\overline{CS}$ = 5V (Note 6)			15	pF
POWER REQUIREMENTS						
Positive Supply Voltage	V_{DD}			5 ±5%		V
Negative Supply Voltage	V_{SS}			0 or -5 ±5%		V
Positive Supply Current	I_{DD}	Operating mode		1.5	2.5	mA
		Fast power-down		30	70	µA
		Full power-down		2	10	
Negative Supply Current	I_{SS}	Operating mode and fast power-down			50	µA
		Full power-down			10	

ELECTRICAL CHARACTERISTICS (continued)

(V_{DD} = 5V ±5%; V_{SS} = 0V or -5V; f_{CLK} = 2.0MHz, external clock (50% duty cycle); 15 clocks/conversion cycle (133ksps); MAX186—4.7µF capacitor at VREF pin; MAX188—external reference, VREF = 4.096V applied to VREF pin; T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Positive Supply Rejection (Note 8)	PSR	V_{DD} = 5V ±5%; external reference, 4.096V; full-scale input		±0.06	±0.5	mV
Negative Supply Rejection (Note 8)	PSR	V_{SS} = -5V ±5%; external reference, 4.096V; full-scale input		±0.01	±0.5	mV

Note 1: Tested at V_{DD} = 5.0V; V_{SS} = 0V; unipolar input mode.
Note 2: Relative accuracy is the deviation of the analog value at any code from its theoretical value after the full-scale range has been calibrated.
Note 3: MAX186 – internal reference, offset nulled; MAX188 – external reference (VREF = +4.096V), offset nulled.
Note 4: Ground on-channel; sine wave applied to all off channels.
Note 5: Conversion time defined as the number of clock cycles times the clock period; clock has 50% duty cycle.
Note 6: Guaranteed by design. Not subject to production testing.
Note 7: External load should not change during conversion for specified accuracy.
Note 8: Measured at V_{SUPPLY} +5% and V_{SUPPLY} -5% only.
Note 9: The common-mode range for the analog inputs is from V_{SS} to V_{DD}.

TIMING CHARACTERISTICS

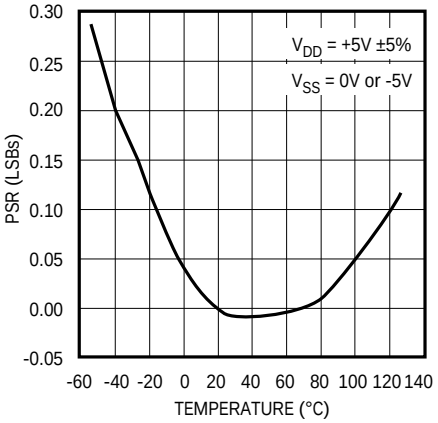
(V_{DD} = 5V ±5%; V_{SS} =0V or -5V, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
Acquisition Time	t_{AZ}			1.5			µs
DIN to SCLK Setup	t_{DS}			100			ns
DIN to SCLK Hold	t_{DH}					0	ns
SCLK Fall to Output Data Valid	t_{DO}	C_{LOAD} = 100pF	MAX18_ _C/E	20		150	ns
			MAX18_ _M	20		200	ns
$\overline{CS}$ Fall to Output Enable	t_{DV}	C_{LOAD} = 100pF				100	ns
$\overline{CS}$ Rise to Output Disable	t_{TR}	C_{LOAD} = 100pF				100	ns
$\overline{CS}$ to SCLK Rise Setup	t_{CSS}			100			ns
$\overline{CS}$ to SCLK Rise Hold	t_{CSH}			0			ns
SCLK Pulse Width High	t_{CH}			200			ns
SCLK Pulse Width Low	t_{CL}			200			ns
SCLK Fall to SSTRB	t_{SSTRB}	C_{LOAD} = 100pF				200	ns
$\overline{CS}$ Fall to SSTRB Output Enable (Note 6)	t_{SDV}	External clock mode only, C_{LOAD} = 100pF				200	ns
$\overline{CS}$ Rise to SSTRB Output Disable (Note 6)	t_{STR}	External clock mode only, C_{LOAD} = 100pF				200	ns
SSTRB Rise to SCLK Rise (Note 6)	t_{SCK}	Internal clock mode only		0			ns

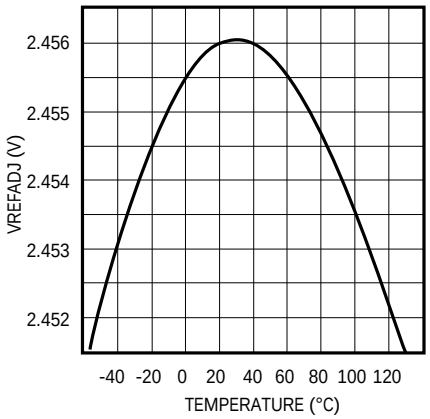
Low-Power, 8-Channel, Serial 12-Bit ADCs

Typical Operating Characteristics

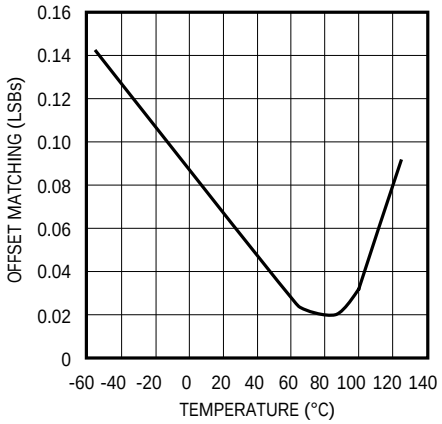
POWER-SUPPLY REJECTION vs. TEMPERATURE



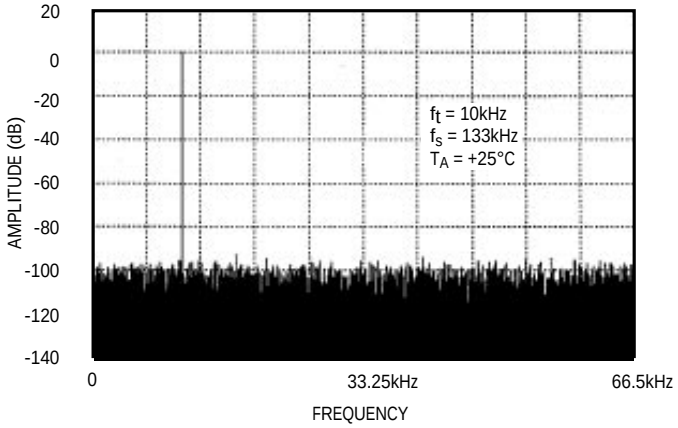
INTERNAL REFERENCE VOLTAGE vs. TEMPERATURE



CHANNEL-TO-CHANNEL OFFSET MATCHING vs. TEMPERATURE



MAX186/MAX188 FFT PLOT – 133kHz



Pin Description

PIN	NAME	FUNCTION
1-8	CH0-CH7	Sampling Analog Inputs
9	V_{SS}	Negative Supply Voltage. Tie to -5V ±5% or AGND
10	$\overline{SHDN}$	Three-Level Shutdown Input. Pulling $\overline{SHDN}$ low shuts the MAX186/MAX188 down to 10µA (max) supply current, otherwise the MAX186/MAX188 are fully operational. Pulling $\overline{SHDN}$ high puts the reference-buffer amplifier in internal compensation mode. Letting $\overline{SHDN}$ float puts the reference-buffer amplifier in external compensation mode.
11	VREF	Reference Voltage for analog-to-digital conversion. Also, Output of the Reference Buffer Amplifier (4.096V in the MAX186, 1.638 x REFADJ in the MAX188). Add a 4.7µF capacitor to ground when using external compensation mode. Also functions as an input when used with a precision external reference.

Pin Description (continued)

PIN	NAME	FUNCTION
12	REFADJ	Input to the Reference-Buffer Amplifier. To disable the reference-buffer amplifier, tie REFADJ to V_{DD}.
13	AGND	Analog Ground. Also IN- Input for single-ended conversions.
14	DGND	Digital Ground
15	DOUT	Serial Data Output. Data is clocked out at the falling edge of SCLK. High impedance when $\overline{CS}$ is high.
16	SSTRB	Serial Strobe Output. In internal clock mode, SSTRB goes low when the MAX186/MAX188 begin the A/D conversion and goes high when the conversion is done. In external clock mode, SSTRB pulses high for one clock period before the MSB decision. High impedance when $\overline{CS}$ is high (external mode).
17	DIN	Serial Data Input. Data is clocked in at the rising edge of SCLK.
18	$\overline{CS}$	Active-Low Chip Select. Data will not be clocked into DIN unless $\overline{CS}$ is low. When $\overline{CS}$ is high, DOUT is high impedance.
19	SCLK	Serial Clock Input. Clocks data in and out of serial interface. In external clock mode, SCLK also sets the conversion speed. (Duty cycle must be 40% to 60% in external clock mode.)
20	V_{DD}	Positive Supply Voltage, +5V ±5%

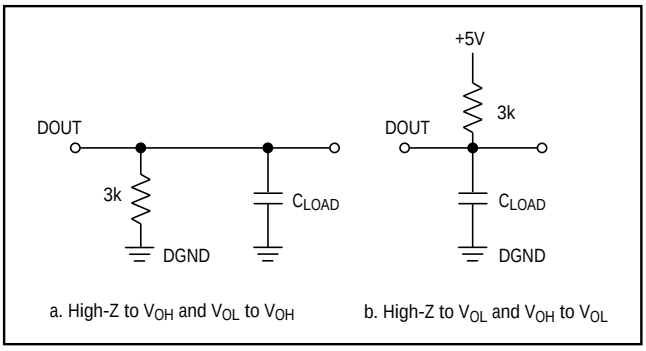


Figure 1. Load Circuits for Enable Time

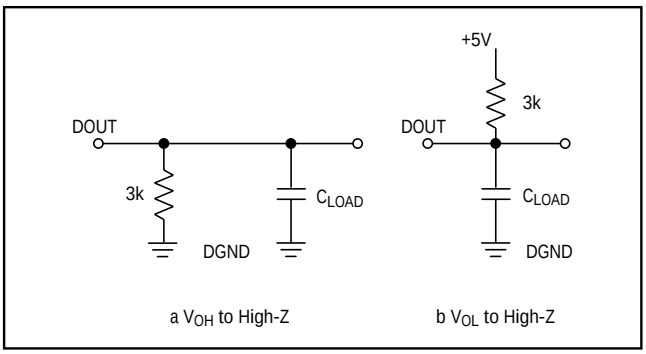


Figure 2. Load Circuits for Disabled Time

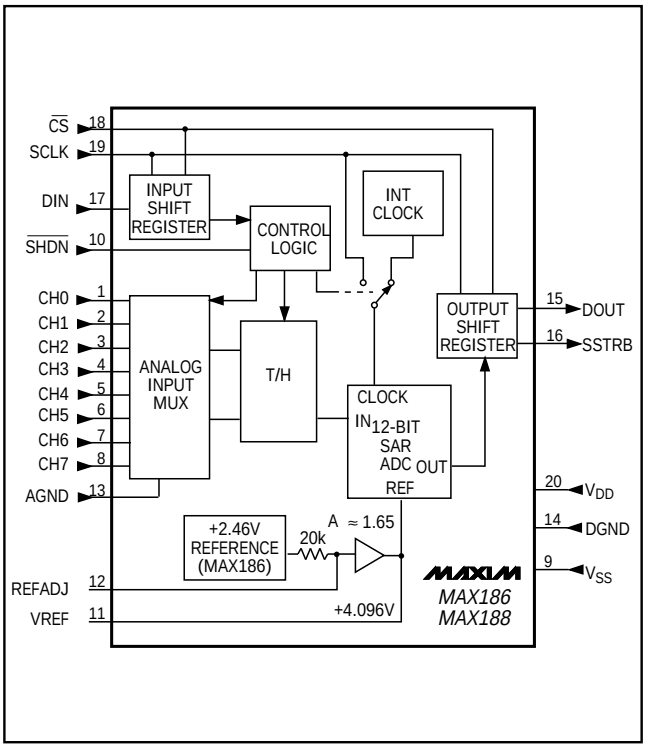


Figure 3. Block Diagram

Low-Power, 8-Channel, Serial 12-Bit ADCs

Detailed Description

The MAX186/MAX188 use a successive-approximation conversion technique and input track/hold (T/H) circuitry to convert an analog signal to a 12-bit digital output. A flexible serial interface provides easy interface to microprocessors. No external hold capacitors are required. Figure 3 shows the block diagram for the MAX186/MAX188.

Pseudo-Differential Input

The sampling architecture of the ADC's analog comparator is illustrated in the Equivalent Input Circuit (Figure 4). In single-ended mode, IN+ is internally switched to CH0-CH7 and IN- is switched to AGND. In differential mode, IN+ and IN- are selected from pairs of CH0/CH1, CH2/CH3, CH4/CH5 and CH6/CH7. Configure the channels with Table 3 and Table 4.

In differential mode, IN- and IN+ are internally switched to either one of the analog inputs. This configuration is pseudo-differential to the effect that only the signal at IN+ is sampled. The return side (IN-) must remain stable within ±0.5LSB (±0.1LSB for best results) with respect to AGND during a conversion. Accomplish this by connecting a 0.1µF capacitor from AIN- (the selected analog input, respectively) to AGND.

During the acquisition interval, the channel selected as the positive input (IN+) charges capacitor C_{HOLD}. The acquisition interval spans three SCLK cycles and ends on the falling SCLK edge after the last bit of the input control word has been entered. At the end of the acquisition interval, the T/H switch opens, retaining charge on C_{HOLD} as a sample of the signal at IN+.

The conversion interval begins with the input multiplexer switching C_{HOLD} from the positive input (IN+) to the negative input (IN-). In single-ended mode, IN- is simply AGND. This unbalances node ZERO at the input of the comparator. The capacitive DAC adjusts during the remainder of the conversion cycle to restore node ZERO to 0V within the limits of 12-bit resolution. This action is equivalent to transferring a charge of 16pF x $[(V_{IN}+) - (V_{IN}-)]$ from C_{HOLD} to the binary-weighted capacitive DAC, which in turn forms a digital representation of the analog input signal.

Track/Hold

The T/H enters its tracking mode on the falling clock edge after the fifth bit of the 8-bit control word has been shifted in. The T/H enters its hold mode on the falling clock edge after the eighth bit of the control word has been shifted in. If the converter is set up for single-ended inputs, IN- is connected to AGND, and the converter samples the "+" input. If the converter is set up for differential inputs, IN- connects to the "-" input, and the difference of |IN+ - IN-| is sampled. At the end of the conversion, the positive input connects back to IN+, and C_{HOLD} charges to the input signal.

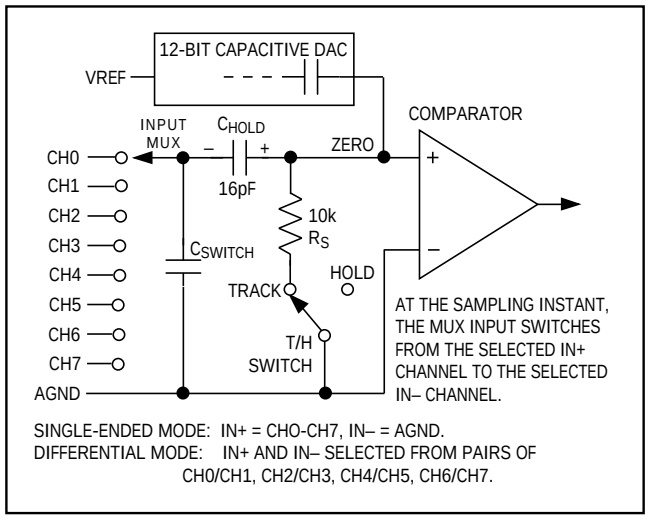


Figure 4. Equivalent Input Circuit

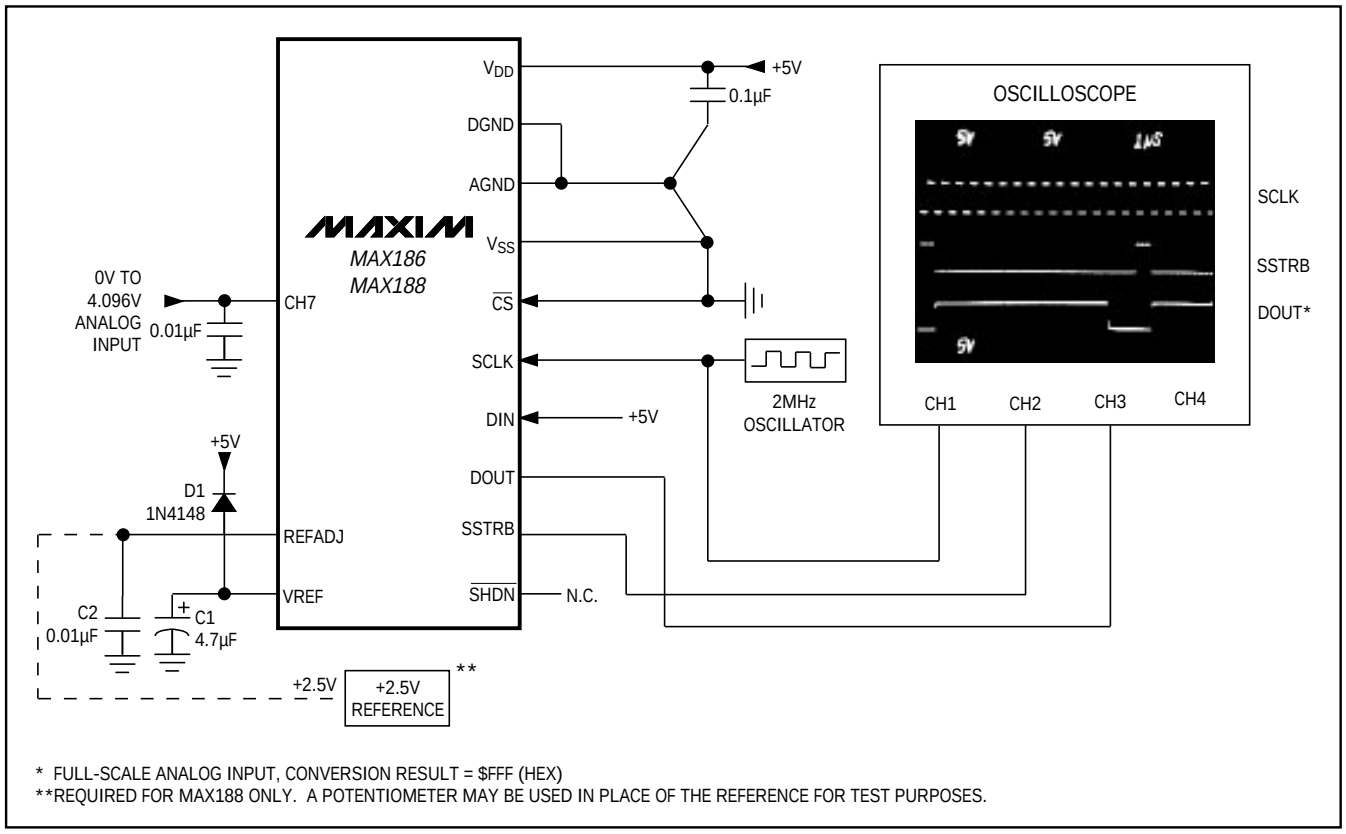
The time required for the T/H to acquire an input signal is a function of how quickly its input capacitance is charged. If the input signal's source impedance is high, the acquisition time lengthens and more time must be allowed between conversions. Acquisition time is calculated by:

$$t_{AZ} = 9 \times (R_S + R_{IN}) \times 16pF,$$

where R_{IN} = 5kΩ, R_S = the source impedance of the input signal, and t_{AZ} is never less than 1.5µs. Note that source impedances below 5kΩ do not significantly affect the AC performance of the ADC. Higher source impedances can be used if an input capacitor is connected to the analog inputs, as shown in Figure 5. Note that the input capacitor forms an RC filter with the input source impedance, limiting the ADC's signal bandwidth.

Input Bandwidth

The ADC's input tracking circuitry has a 4.5MHz small-signal bandwidth, so it is possible to digitize high-speed transient events and measure periodic signals with bandwidths exceeding the ADC's sampling rate by using undersampling techniques. To avoid high-frequency signals being aliased into the frequency band of interest, anti-alias filtering is recommended.



* FULL-SCALE ANALOG INPUT, CONVERSION RESULT = $FFF (HEX)
**REQUIRED FOR MAX188 ONLY. A POTENTIOMETER MAY BE USED IN PLACE OF THE REFERENCE FOR TEST PURPOSES.

Figure 5. Quick-Look Circuit

Analog Input Range and Input Protection

Internal protection diodes, which clamp the analog input to V_{DD} and V_{SS}, allow the channel input pins to swing from V_{SS} - 0.3V to V_{DD} + 0.3V without damage. However, for accurate conversions near full scale, the inputs must not exceed V_{DD} by more than 50mV, or be lower than V_{SS} by 50mV.

If the analog input exceeds 50mV beyond the supplies, do not forward bias the protection diodes of off-channels over two milliamperes, as excessive current will degrade the conversion accuracy of the on-channel.

The full-scale input voltage depends on the voltage at VREF. See Tables 1a and 1b.

Quick Look

To evaluate the analog performance of the MAX186/MAX188 quickly, use the circuit of Figure 5. The MAX186/MAX188 require a control byte to be written to DIN before each conversion. Tying DIN to +5V feeds in control bytes of $FF (HEX), which trigger

Table 1a. Unipolar Full Scale and Zero Scale

Reference	Zero Scale	Full Scale
Internal Reference (MAX186 only)	0V	+4.096V
External Reference at REFADJ	0V	V_{REFADJ} x A*
at VREF	0V	VREF

** A = 1.678 for the MAX186, 1.638 for the MAX188*

Table 1b. Bipolar Full Scale, Zero Scale, and Negative Full Scale

Reference	Negative Full Scale	Zero Scale	Full Scale
Internal Reference (MAX186 only)	-4.096V/2	0V	+4.096V/2
External Reference at REFADJ	-1/2V_{REFADJ} x A*	0V	+1/2V_{REFADJ} x A*
at VREF	-1/2 VREF	0V	+1/2 VREF

** A = 1.678 for the MAX186, 1.638 for the MAX188*

single-ended unipolar conversions on CH7 in external clock mode without powering down between conversions. In external clock mode, the SSTRB output pulses high for one clock period before the most significant bit of the 12-bit conversion result comes out of DOUT. Varying the analog input to CH7 should alter the sequence of bits from DOUT. A total of 15 clock cycles is required per conversion. All transitions of the SSTRB and DOUT outputs occur on the falling edge of SCLK.

How to Start a Conversion

A conversion is started on the MAX186/MAX188 by clocking a control byte into DIN. Each rising edge on SCLK, with $\overline{CS}$ low, clocks a bit from DIN into the MAX186/MAX188's internal shift register. After $\overline{CS}$ falls, the first arriving logic "1" bit defines the MSB of the control byte. Until this first "start" bit arrives, any number of logic "0" bits can be clocked into DIN with no effect. Table 2 shows the control-byte format.

The MAX186/MAX188 are fully compatible with Microwire and SPI devices. For SPI, select the correct clock polarity and sampling edge in the SPI control registers: set CPOL = 0 and CPHA = 0. Microwire and SPI both transmit a byte and receive a byte at the same time. Using the *Typical Operating Circuit*, the simplest software interface requires only three 8-bit transfers to perform a conversion (one 8-bit transfer to configure the ADC, and two more 8-bit transfers to clock out the 12-bit conversion result).

Example: Simple Software Interface

Make sure the CPU's serial interface runs in master mode so the CPU generates the serial clock. Choose a clock frequency from 100kHz to 2MHz.

1) Set up the control byte for external clock mode, call it TB1. TB1 should be of the format: 1XXXXX11 Binary, where the Xs denote the particular channel and conversion-mode selected.

Table 2. Control-Byte Format

Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)
START	SEL2	SEL1	SEL0	UNI/$\overline{BIP}$	SGL/$\overline{DIF}$	PD1	PD0

Bit	Name	Description
7(MSB)	START	The first logic "**1**" bit after $\overline{CS}$ goes low defines the beginning of the control byte.
6 5 4	SEL2 SEL1 SEL0	These three bits select which of the eight channels are used for the conversion. See Tables 3 and 4.
3	UNI/$\overline{BIP}$	**1** = unipolar, **0** = bipolar. Selects unipolar or bipolar conversion mode. In unipolar mode, an analog input signal from 0V to VREF can be converted; in bipolar mode, the signal can range from -VREF/2 to +VREF/2.
2	SGL/$\overline{DIF}$	**1** = single ended, **0** = differential. Selects single-ended or differential conversions. In single-ended mode, input signal voltages are referred to AGND. In differential mode, the voltage difference between two channels is measured. See Tables 3 and 4.
1 0(LSB)	PD1 PD0	Selects clock and power-down modes. PD1 PD0 Mode **0** **0** Full power-down (I_Q = 2µA) **0** **1** Fast power-down (I_Q = 30µA) **1** **0** Internal clock mode **1** **1** External clock mode

Table 3. Channel Selection in Single-Ended Mode (SGL/$\overline{DIFF}$ = 1)

SEL2	SEL1	SEL0	CH0	CH1	CH2	CH3	CH4	CH5	CH6	CH7	AGND
0	0	0	+								–
1	0	0		+							–
0	0	1			+						–
1	0	1				+					–
0	1	0					+				–
1	1	0						+			–
0	1	1							+		–
1	1	1								+	–

Table 4. Channel Selection in Differential Mode (SGL/$\overline{DIFF}$ = 0)

SEL2	SEL1	SEL0	CH0	CH1	CH2	CH3	CH4	CH5	CH6	CH7
0	0	0	+	–						
0	0	1			+	–				
0	1	0					+	–		
0	1	1							+	–
1	0	0	–	+						
1	0	1			–	+				
1	1	0					–	+		
1	1	1							–	+

2) Use a general-purpose I/O line on the CPU to pull $\overline{CS}$ on the MAX186/MAX188 low.
3) Transmit TB1 and simultaneously receive a byte and call it RB1. Ignore RB1.
4) Transmit a byte of all zeros ($00 HEX) and simultaneously receive byte RB2.
5) Transmit a byte of all zeros ($00 HEX) and simultaneously receive byte RB3.
6) Pull $\overline{CS}$ on the MAX186/MAX188 high.

Figure 6 shows the timing for this sequence. Bytes RB2 and RB3 will contain the result of the conversion padded with one leading zero and three trailing zeros. The total conversion time is a function of the serial clock frequency and the amount of dead time between 8-bit transfers. Make sure that the total conversion time does not exceed 120µs, to avoid excessive T/H droop.

Digital Output

In unipolar input mode, the output is straight binary (see Figure 15). For bipolar inputs, the output is twos-complement (see Figure 16). Data is clocked out at the falling edge of SCLK in MSB-first format.

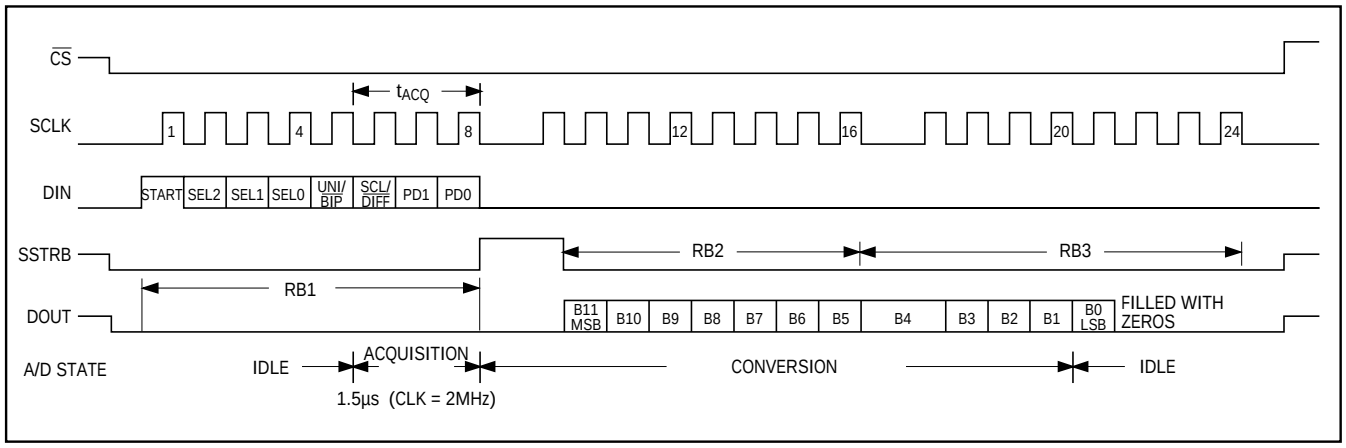


Figure 6. 24-Bit External Clock Mode Conversion Timing (SPI, QSPI and Microwire Compatible)

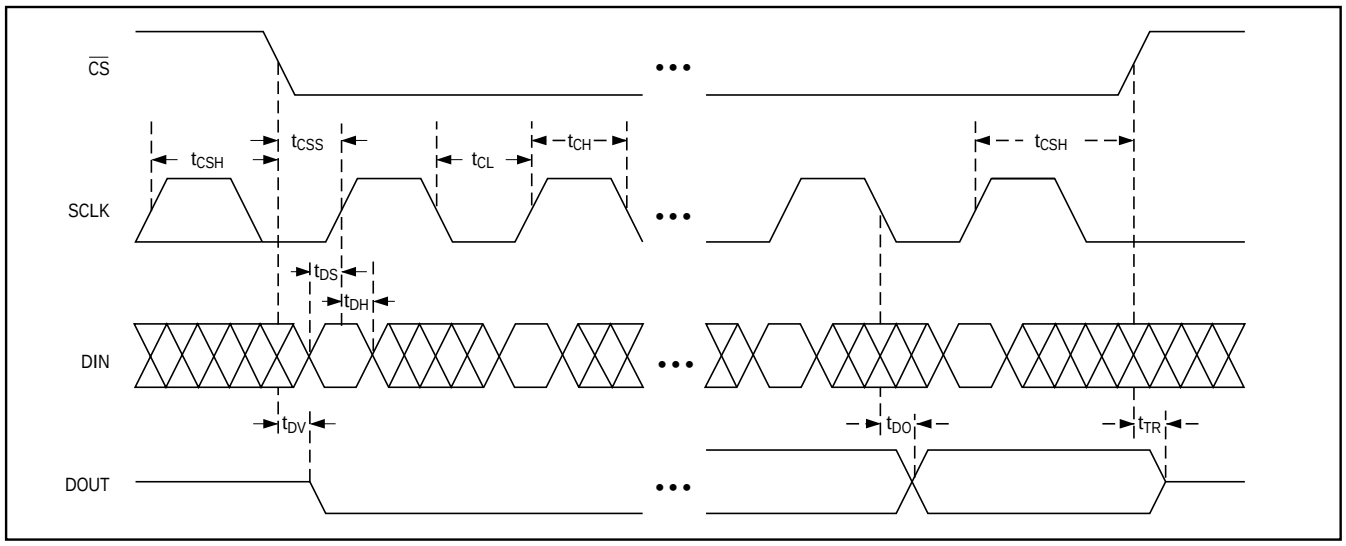


Figure 7. Detailed Serial-Interface Timing

Internal and External Clock Modes

The MAX186/MAX188 may use either an external serial clock or the internal clock to perform the successive-approximation conversion. In both clock modes, the external clock shifts data in and out of the MAX186/MAX188. The T/H acquires the input signal as the last three bits of the control byte are clocked into DIN. Bits PD1 and PD0 of the control byte program the clock mode. Figures 7 through 10 show the timing characteristics common to both modes.

External Clock

In external clock mode, the external clock not only shifts data in and out, it also drives the analog-to-digital conversion steps. SSTRB pulses high for one clock period after the last bit of the control byte. Successive-approximation bit decisions are made and appear at DOUT on each of the next 12 SCLK falling edges (see Figure 6). SSTRB and DOUT go into a high-impedance state when $\overline{CS}$ goes high; after the next $\overline{CS}$ falling edge, SSTRB will output a logic low. Figure 8 shows the SSTRB timing in external clock mode.

The conversion must complete in some minimum time, or else droop on the sample-and-hold capacitors may degrade conversion results. Use internal clock mode if the clock period exceeds 10µs, or if serial-clock interruptions could cause the conversion interval to exceed 120µs.

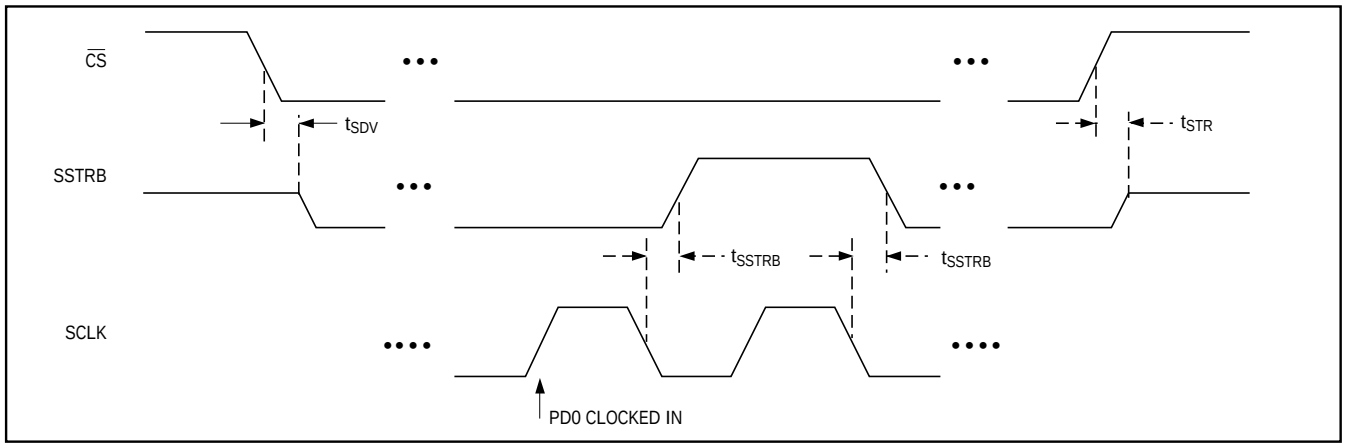


Figure 8. External Clock Mode SSTRB Detailed Timing

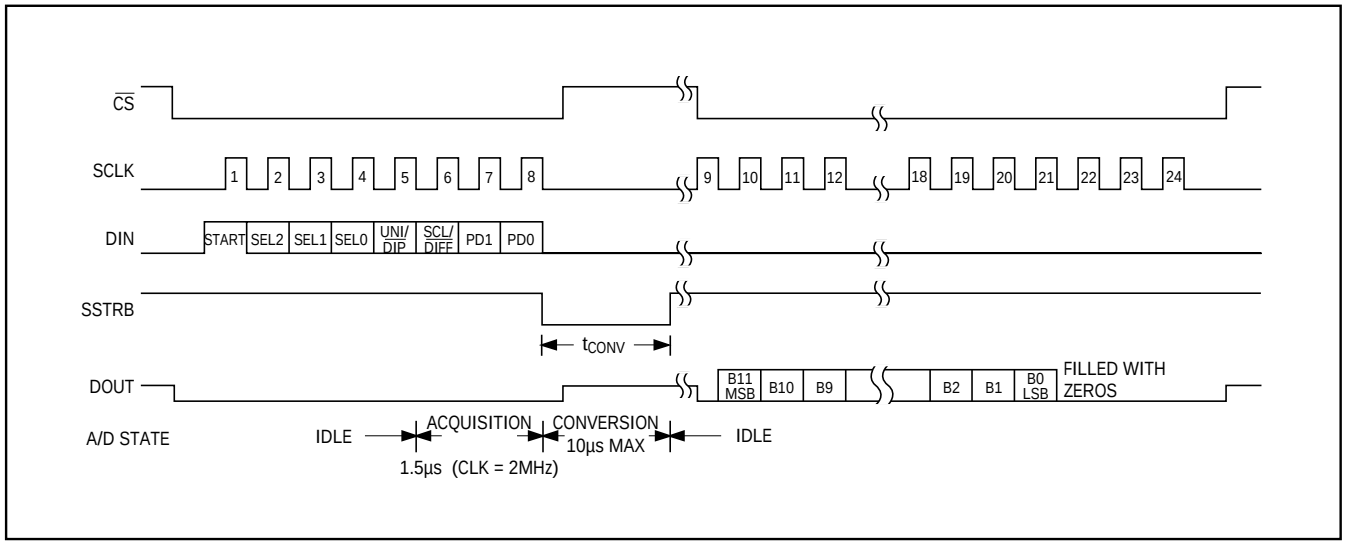


Figure 9. Internal Clock Mode Timing

Internal Clock

In internal clock mode, the MAX186/MAX188 generate their own conversion clock internally. This frees the microprocessor from the burden of running the SAR conversion clock, and allows the conversion results to be read back at the processor's convenience, at any clock rate from zero to typically 10MHz. SSTRB goes low at the start of the conversion and then goes high when the conversion is complete. SSTRB will be low for a maximum of 10µs, during which time SCLK should remain low for best noise performance. An internal register stores data when the conversion is in progress. SCLK clocks the data out at this register at any time after the conversion is complete. After SSTRB goes high, the next falling clock edge will produce the MSB of the conversion at DOUT, followed by the remaining bits in MSB-first format (see Figure 9). $\overline{CS}$ does not need to be held low once a conversion is started. Pulling $\overline{CS}$ high prevents data from being clocked into the MAX186/MAX188 and three-states DOUT, but it does not adversely effect an internal clock-mode conversion already in progress. When internal clock mode is selected, SSTRB does not go into a high-impedance state when $\overline{CS}$ goes high.

Figure 10 shows the SSTRB timing in internal clock mode. In internal clock mode, data can be shifted in and out of the MAX186/MAX188 at clock rates exceeding 4.0MHz, provided that the minimum acquisition time, t_{AZ}, is kept above 1.5µs.

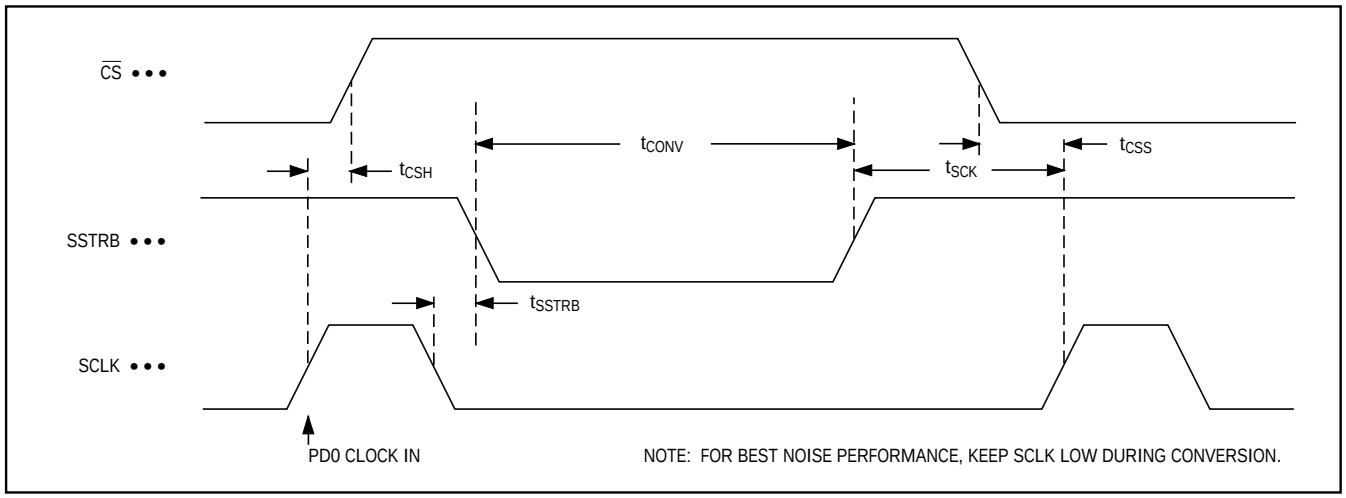


Figure 10. Internal Clock Mode SSTRB Detailed Timing

Data Framing

The falling edge of $\overline{CS}$ does **not** start a conversion on the MAX186/MAX188. The first logic high clocked into DIN is interpreted as a start bit and defines the first bit of the control byte. A conversion starts on the falling edge of SCLK, after the eighth bit of the control byte (the PD0 bit) is clocked into DIN. The start bit is defined as:

The first high bit clocked into DIN with $\overline{CS}$ low anytime the converter is idle, e.g. after V_{CC} is applied.

OR

The first high bit clocked into DIN after bit 5 of a conversion in progress is clocked onto the DOUT pin.

If a falling edge on $\overline{CS}$ forces a start bit before bit 5 (B5) becomes available, then the current conversion will be terminated and a new one started. Thus, the fastest the MAX186/MAX188 can run is 15 clocks per conversion. Figure 11a shows the serial-interface timing necessary to perform a conversion every 15 SCLK cycles in external clock mode. If $\overline{CS}$ is low and SCLK is continuous, guarantee a start bit by first clocking in 16 zeros.

Most microcontrollers require that conversions occur in multiples of 8 SCLK clocks; 16 clocks per conversion will typically be the fastest that a microcontroller can drive the MAX186/MAX188. Figure 11b shows the serial-interface timing necessary to perform a conversion every 16 SCLK cycles in external clock mode.

Applications Information

Power-On Reset

When power is first applied and if $\overline{SHDN}$ is not pulled low, internal power-on reset circuitry will activate the MAX186/MAX188 in internal clock mode, ready to convert with SSTRB = high. After the power supplies have been stabilized, the internal reset time is 100µs and no conversions should be performed during this phase. SSTRB is high on power-up and, if $\overline{CS}$ is low, the first logical 1 on DIN will be interpreted as a start bit. Until a conversion takes place, DOUT will shift out zeros.

Reference-Buffer Compensation

In addition to its shutdown function, the $\overline{SHDN}$ pin also selects internal or external compensation. The compensation affects both power-up time and maximum conversion speed. Compensated or not, the minimum clock rate is 100kHz due to droop on the sample-and-hold.

To select external compensation, float $\overline{SHDN}$. See the *Typical Operating Circuit*, which uses a 4.7µF capacitor at VREF. A value of 4.7µF or greater ensures stability and allows operation of the converter at the full clock speed of 2MHz. External compensation increases power-up time (see the *Choosing Power-Down Mode* section, and Table 5).

Internal compensation requires no external capacitor at VREF, and is selected by pulling $\overline{SHDN}$ high. Internal compensation allows for shortest power-up times, but is only available using an external clock and reduces the maximum clock rate to 400kHz.

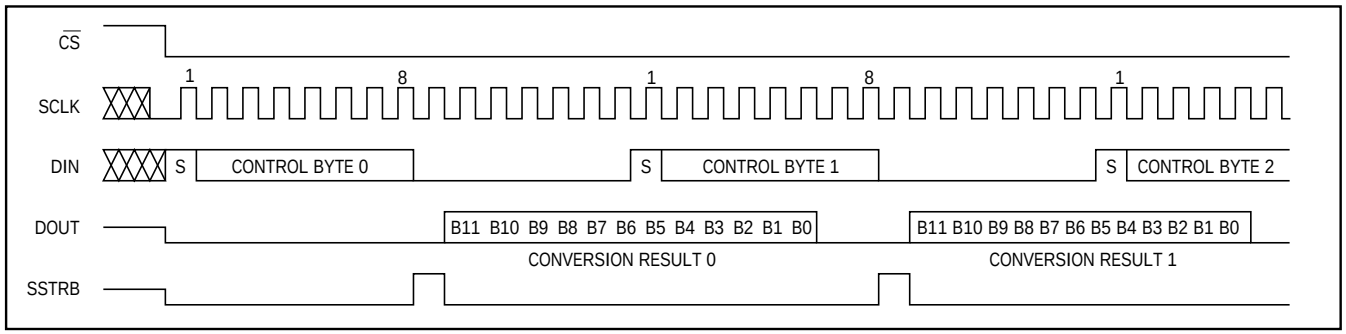


Figure 11a. External Clock Mode, 15 Clocks/Conversion Timing

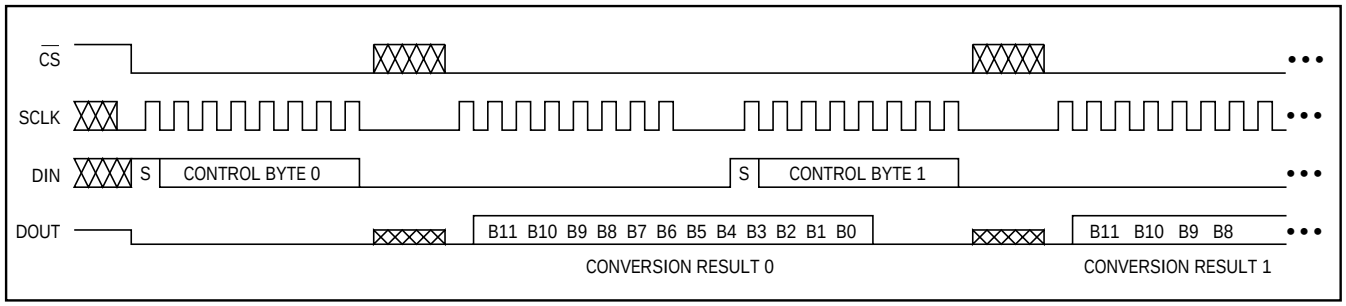


Figure 11b. External Clock Mode, 16 Clocks/Conversion Timing

Power-Down

Choosing Power-Down Mode

You can save power by placing the converter in a low-current shutdown state between conversions. Select full power-down or fast power-down mode via bits 7 and 8 of the DIN control byte with $\overline{SHDN}$ high or floating (see Tables 2 and 6). Pull $\overline{SHDN}$ low at any time to shut down the converter completely. $\overline{SHDN}$ overrides bits 7 and 8 of DIN word (see Table 7).

Full power-down mode turns off all chip functions that draw quiescent current, reducing I_{DD} and I_{SS} typically to 2µA.

Fast power-down mode turns off all circuitry except the bandgap reference. With the fast power-down mode, the supply current is 30µA. Power-up time can be shortened to 5µs in internal compensation mode.

In both software shutdown modes, the serial interface remains operational, however, the ADC will not convert. Table 5 illustrates how the choice of reference-buffer compensation and power-down mode affects both power-up delay and maximum sample rate.

In external compensation mode, the power-up time is 20ms with a 4.7µF compensation capacitor (200ms with a 33µF capacitor) when the capacitor is fully discharged. In fast power-down, you can eliminate start-up time by using low-leakage capacitors that will not discharge more than 1/2LSB while shut down. In shutdown, the capacitor has to supply the current into the reference (1.5µA typ) and the transient currents at power-up.

Figures 12a and 12b illustrate the various power-down sequences in both external and internal clock modes.

Software Power-Down

Software power-down is activated using bits PD1 and PD0 of the control byte. As shown in Table 6, PD1 and PD0 also specify the clock mode. When software shutdown is asserted, the ADC will continue to operate in the last specified clock mode until the conversion is complete. Then the ADC powers down into a low quiescent-current state. In internal clock mode, the interface remains active and conversion results may be clocked out while the MAX186/MAX188 have already entered a software power-down.

The first logical 1 on DIN will be interpreted as a start bit, and powers up the MAX186/MAX188. Following the start bit, the data input word or control byte also determines clock and power-down modes. For example, if the DIN word contains PD1 = 1, then the chip will remain powered up. If PD1 = 0, a power-down will resume after one conversion.

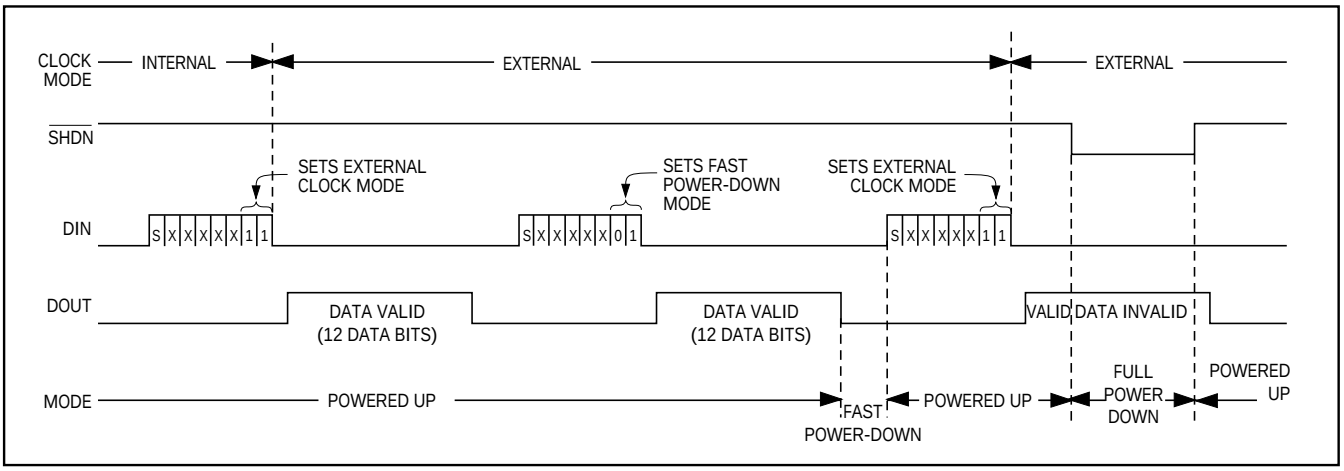


Figure 12a. Timing Diagram Power-Down Modes, External Clock

Table 5. Typical Power-Up Delay Times

Reference Buffer	Reference-Buffer Compensation Mode	VREF Capacitor (µF)	Power-Down Mode	Power-Up Delay (sec)	Maximum Sampling Rate (ksps)
Enabled	Internal		Fast	5µ	26
Enabled	Internal		Full	300µ	26
Enabled	External	4.7	Fast	See Figure 14c	133
Enabled	External	4.7	Full	See Figure 14c	133
Disabled			Fast	2µ	133
Disabled			Full	2µ	133

Table 6. Software Shutdown and Clock Mode

PD1	PD0	Device Mode
1	1	External Clock Mode
1	0	Internal Clock Mode
0	1	Fast Power-Down Mode
0	0	Full Power-Down Mode

Table 7. Hard-Wired Shutdown and Compensation Mode

$\overline{SHDN}$ State	Device Mode	Reference-Buffer Compensation
1	Enabled	Internal Compensation
Floating	Enabled	External Compensation
0	Full Power-Down	N/A

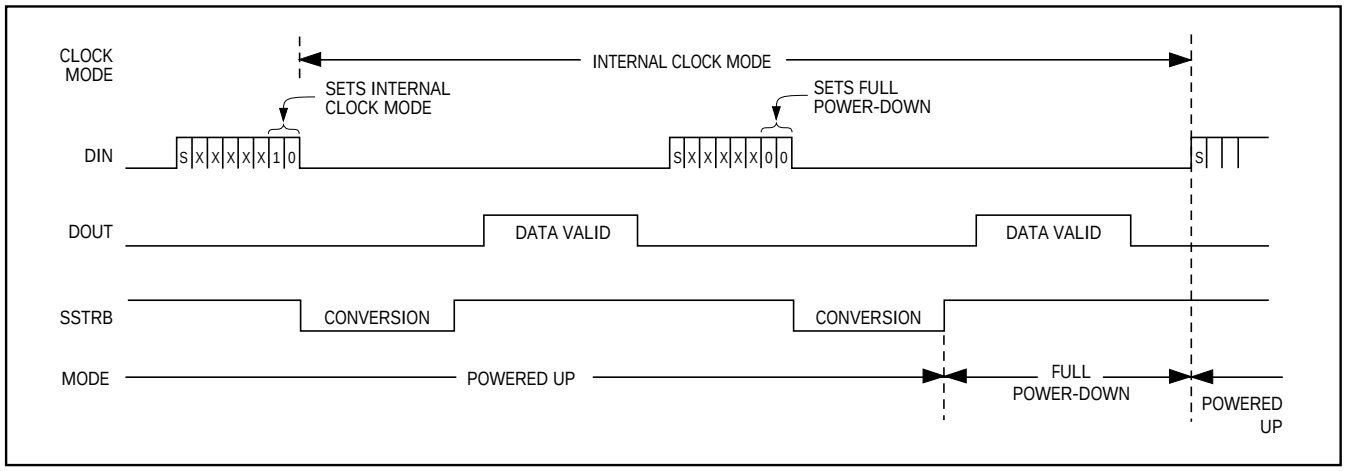


Figure 12b. Timing Diagram Power-Down Modes, Internal Clock

Hardware Power-Down

The $\overline{SHDN}$ pin places the converter into the full power-down mode. Unlike with the software shut-down modes, conversion is not completed. It stops coincidentally with $\overline{SHDN}$ being brought low. There is no power-up delay if an external reference is used and is not shut down. The $\overline{SHDN}$ pin also selects internal or external reference compensation (see Table 7).

Power-Down Sequencing

The MAX186/MAX188 auto power-down modes can save considerable power when operating at less than maximum sample rates. The following discussion illustrates the various power-down sequences.

Lowest Power at up to 500 Conversions/Channel/Second

The following examples illustrate two different power-down sequences. Other combinations of clock rates, compensation modes, and power-down modes may give lowest power consumption in other applications.

Figure 14a depicts the MAX186 power consumption for one or eight channel conversions utilizing full power-down mode and internal reference compensation. A 0.01µF bypass capacitor at REFADJ forms an RC filter with the internal 20kΩ reference resistor with a 0.2ms time constant. To achieve full 12-bit accuracy, 10 time constants or 2ms are required after power-up. Waiting 2ms in FASTPD mode instead of full power-up will reduce the power consumption by a factor of 10 or more. This is achieved by using the sequence shown in Figure 13.

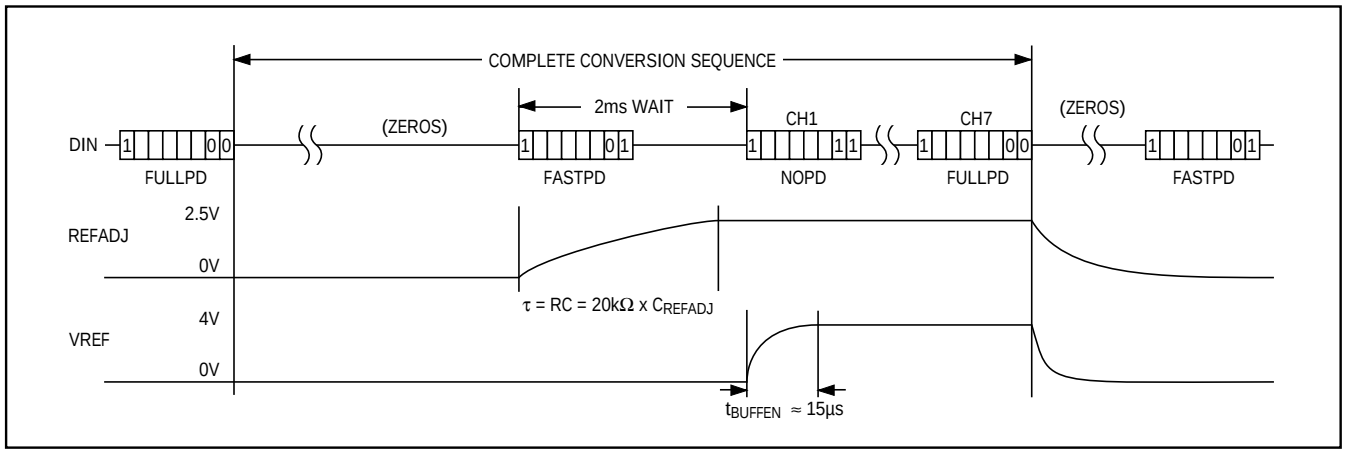


Figure 13. MAX186 FULLPD/FASTPD Power-Up Sequence

Low-Power, 8-Channel, Serial 12-Bit ADCs

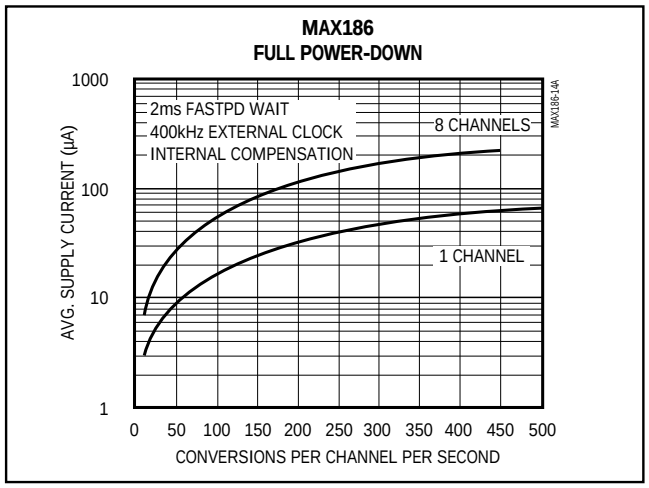


Figure 14a. MAX186 Supply Current vs. Sample Rate/Second, FULLPD, 400kHz Clock

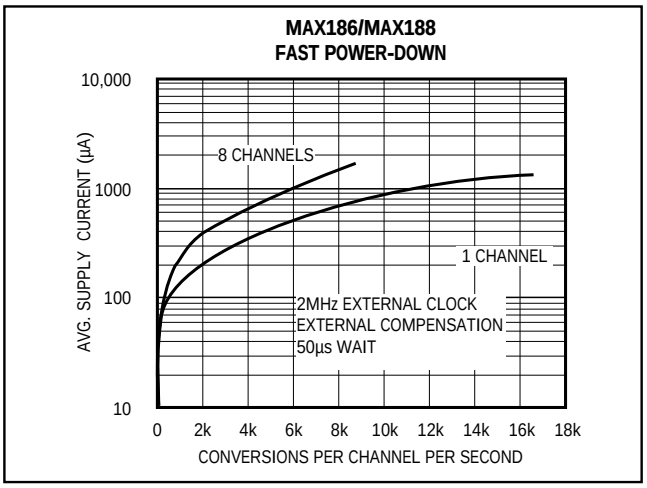


Figure 14b. MAX186/MAX188 Supply Current vs. Sample Rate/Second, FASTPD, 2MHz Clock

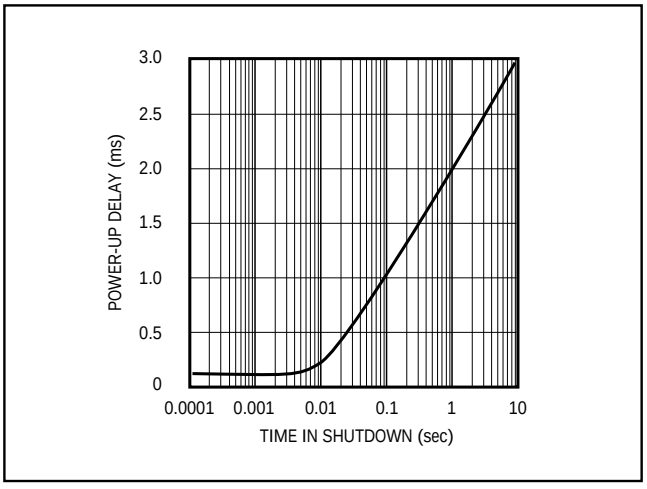


Figure 14c. Typical Power-Up Delay vs. Time in Shutdown

Lowest Power at Higher Throughputs

Figure 14b shows the power consumption with external-reference compensation in fast power-down, with one and eight channels converted. The external 4.7µF compensation requires a 50µs wait after power-up, accomplished by 75 idle clocks after a dummy conversion. This circuit combines fast multi-channel conversion with lowest power consumption possible. Full power-down mode may provide increased power savings in applications where the MAX186/MAX188 are inactive for long periods of time, but where intermittent bursts of high-speed conversions are required.

External and Internal References

The MAX186 can be used with an internal or external reference, whereas an external reference is required for the MAX188. Diode D1 shown in the *Typical Operating Circuit* ensures correct start-up. Any standard signal diode can be used. For both parts, an external reference can either be connected directly at the VREF terminal or at the REFADJ pin.

An internal buffer is designed to provide 4.096V at VREF for both the MAX186 and MAX188. The MAX186's internally trimmed 2.46V reference is buffered with a gain of 1.678. The MAX188's buffer is trimmed with a buffer gain of 1.638 to scale an external 2.5V reference at REFADJ to 4.096V at VREF.

MAX186 Internal Reference

The full-scale range of the MAX186 with internal reference is 4.096V with unipolar inputs, and ±2.048V with bipolar inputs. The internal reference voltage is adjustable to ±1.5% with the Reference-Adjust Circuit of Figure 17.

External Reference

With both the MAX186 and MAX188, an external reference can be placed at either the input (REFADJ) or the output (VREF) of the internal buffer amplifier. The REFADJ input impedance is typically 20kΩ for the MAX186 and higher than 100kΩ for the MAX188, where the internal reference is omitted. At VREF, the input impedance is a minimum of 12kΩ for DC currents. During conversion, an external reference at VREF must be able to deliver up to 350µA DC load current and have an output impedance of 10Ω or less. If the reference has higher output impedance or is noisy, bypass it close to the VREF pin with a 4.7µF capacitor.

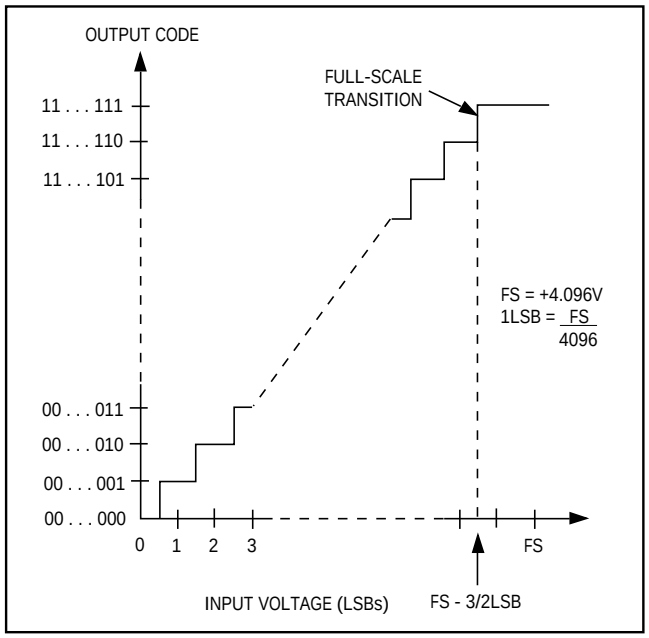


Figure 15. MAX186/MAX188 Unipolar Transfer Function, 4.096V = Full Scale

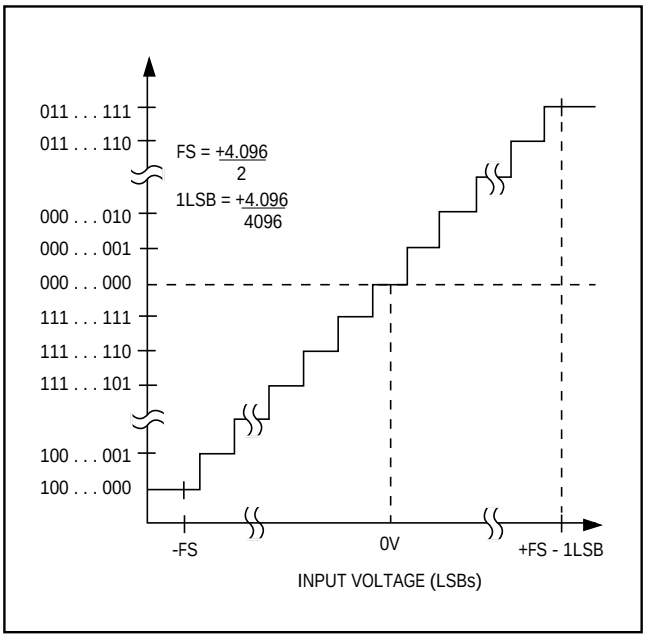


Figure 16. MAX186/MAX188 Bipolar Transfer Function, ±4.096V/2 = Full Scale

Using the buffered REFADJ input avoids external buffering of the reference. To use the direct VREF input, disable the internal buffer by tying REFADJ to V_{DD}.

Transfer Function and Gain Adjust

Figure 15 depicts the nominal, unipolar input/output (I/O) transfer function, and Figure 16 shows the bipolar input/output transfer function. Code transitions occur halfway between successive integer LSB values. Output coding is binary with 1 LSB = 1.00mV (4.096V/4096) for unipolar operation and 1 LSB = 1.00mV ((4.096V/2 - -4.096V/2)/4096) for bipolar operation.

Figure 17, the MAX186 Reference-Adjust Circuit, shows how to adjust the ADC gain in applications that use the internal reference. The circuit provides ±1.5% (±65LSBs) of gain adjustment range.

Layout, Grounding, Bypassing

For best performance, use printed circuit boards. Wire-wrap boards are not recommended. Board layout should ensure that digital and analog signal lines are separated from each other. Do not run analog and digital (especially clock) lines parallel to one another, or digital lines underneath the ADC package.

Figure 18 shows the recommended system ground connections. A single-point analog ground ("star" ground point) should be established at AGND, separate from the logic ground. All other analog grounds

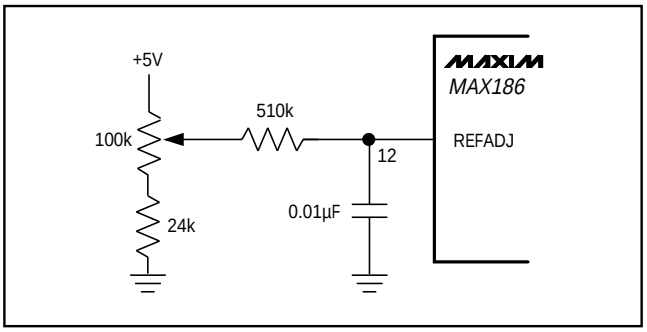


Figure 17. MAX186 Reference-Adjust Circuit

and DGND should be connected to this ground. No other digital system ground should be connected to this single-point analog ground. The ground return to the power supply for this ground should be low impedance and as short as possible for noise-free operation.

High-frequency noise in the V_{DD} power supply may affect the high-speed comparator in the ADC. Bypass these supplies to the single-point analog ground with 0.1µF and 4.7µF bypass capacitors close to the MAX186/MAX188. Minimize capacitor lead lengths for best supply-noise rejection. If the +5V power supply is very noisy, a 10Ω resistor can be connected as a low-pass filter, as shown in Figure 18.

Low-Power, 8-Channel, Serial 12-Bit ADCs

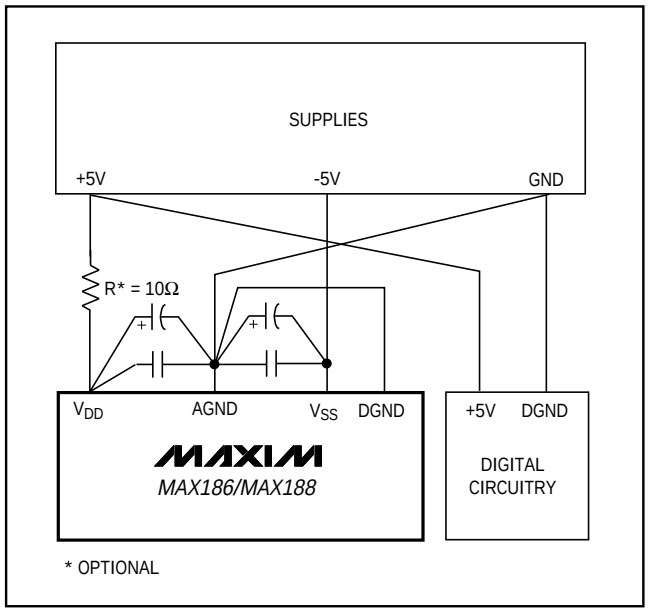


Figure 18. Power-Supply Grounding Connection

High-Speed Digital Interfacing with QSPI

The MAX186/MAX188 can interface with QSPI at high throughput rates using the circuit in Figure 19. This QSPI circuit can be programmed to do a conversion on each of the eight channels. The result is stored in memory without taxing the CPU since QSPI incorporates its own micro-sequencer. Figure 19 depicts the MAX186, but the same circuit could be used with the MAX188 by adding an external reference to VREF and connecting REFADJ to V_{DD}.

Figure 20 details the code that sets up QSPI for autonomous operation. In external clock mode, the MAX186/MAX188 perform a single-ended, unipolar conversion on each of their eight analog input channels. Figure 21, QSPI Assembly-Code Timing, shows the timing associated with the assembly code of Figure 20. The first byte clocked into the MAX186/MAX188 is the control byte, which triggers the first conversion on CH0. The last two bytes clocked into the MAX186/MAX188 are all zero and clock out the results of the CH7 conversion.

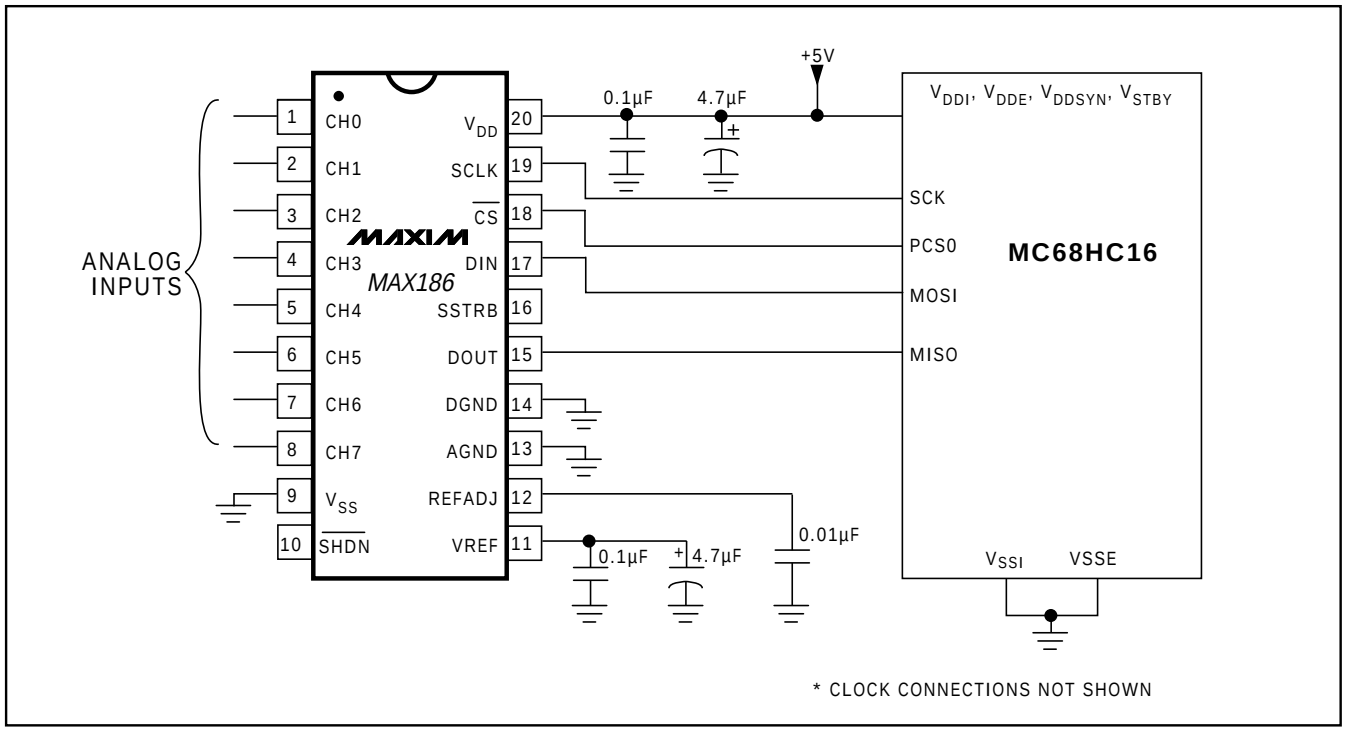


Figure 19. MAX186 QSPI Connection

```
*Title : MAX186.ASM
* Description :
*             This is a shell program for using a stand-alone 68HC16 without any external memory. The internal 1K RAM
*             is put into bank $0F to maintain 68HC11 code compatibility. This program was written with software
*             provided in the Motorola 68HC16 Evaluation Kit.
*
*  Roger J.A. Chen, Applications Engineer
*  MAXIM Integrated Products
*  November 20, 1992
*
*****************************************************************************************************************************************************************
      INCLUDE      'EQUATES.ASM'   ;Equates for common reg addrs
      INCLUDE      'ORG00000.ASM'  ;initialize reset vector
      INCLUDE      'ORG00008.ASM'  ;initialize interrupt vectors
      ORG    $0200                 ;start program after interrupt vectors
      INCLUDE  'INITSYS.ASM'       ;set EK=F,XK=0,YK=0,ZK=0
                                   ;set sys clock at 16.78 MHz, COP off
      INCLUDE  'INITRAM.ASM'       ;turn on internal SRAM at $10000
                                   ;set stack (SK=1, SP=03FE)
MAIN:
      JSR    INITQSPI
MAINLOOP:
      JSR    READ186
WAIT:
      LDAA   SPSR
      ANDA   #$80
      BEQ    WAIT                  ;wait for QSPI to finish
      BRA MAINLOOP
ENDPROGRAM:

INITQSPI:

;This routine sets up the QSPI microsequencer to operate on its own.
;The sequencer will read all eight channels of a MAX186/MAX188 each time
;it is triggered. The A/D converter results will be left in the
;receive data RAM. Each 16 bit receive data RAM location will
;have a leading zero, 12 bits of conversion result and three zeros.
;
;Receive RAM Bits 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
;A/D Result           0   MSB                                LSB 0   0   0
*****  Initialize the QSPI Registers   ******
      PSHA
      PSHB
      LDAA   #%01111000
      STAA   QPDR              ;idle state for PCS0-3 = high
      LDAA   #%01111011
      STAA   QPAR              ;assign port D to be QSPI
      LDAA   #%01111110
      STAA   QDDR              ;only MISO is an input
      LDD    #$8008
      STD    SPCR0             ;master mode,16 bits/transfer,
                               ;CPOL=CPHA=0,1MHz Ser Clock
      LDD    #$0000
      STD    SPCR1             ;set delay between PCS0 and SCK,
```

Figure 20. MAX186/MAX188 Assembly-Code Listing

```
                                    ;set delay between transfers
      LDD     #$0800
      STD     SPCR2                 ;set ENDQP to $8 for 9 transfers
*****  Initialize QSPI Command RAM  *****

      LDAA    #$80          ;CONT=1,BITSE=0,DT=0,DSCK=0,PCS0=ACTIVE
      STAA    $FD40          ;store first byte in COMMAND RAM
      LDAA    #$C0          ;CONT=1,BITSE=1,DT=0,DSCK=0,PCS0=ACTIVE
      STAA    $FD41
      STAA    $FD42
      STAA    $FD43
      STAA    $FD44
      STAA    $FD45
      STAA    $FD46
      STAA    $FD47
      LDAA    #$40          ;CONT=0,BITSE=1,DT=0,DSCK=0,PCS0=ACTIVE
      STAA    $FD48
*****  Initialize QSPI Transmit RAM *****

      LDD     #$008F
                                    STD     $FD20
      LDD     #$00CF
                                    STD     $FD22
      LDD     #$009F
                                    STD     $FD24
      LDD     #$00DF
                                    STD     $FD26
      LDD     #$00AF
                                    STD     $FD28
      LDD     #$00EF
                                    STD     $FD2A
      LDD     #$00BF
                                    STD     $FD2C
      LDD     #$00FF
                                    STD     $FD2E
      LDD     #$0000
                                    STD     $FD30
      PULB
      PULA
      RTS

READ186:
;This routine triggers the QSPI microsequencer to autonomously
;trigger conversions on all 8 channels of the MAX186. Each
;conversion result is stored in the receive data RAM.
      PSHA
      LDAA    #$80
      ORAA    SPCR1
      STAA    SPCR1          ;just set SPE
      PULA
      RTS

*****  Interrupts/Exceptions  *****

BDM: BGND                ;exception vectors point here
```

Figure 20. MAX186/MAX188 Assembly-Code Listing (continued)

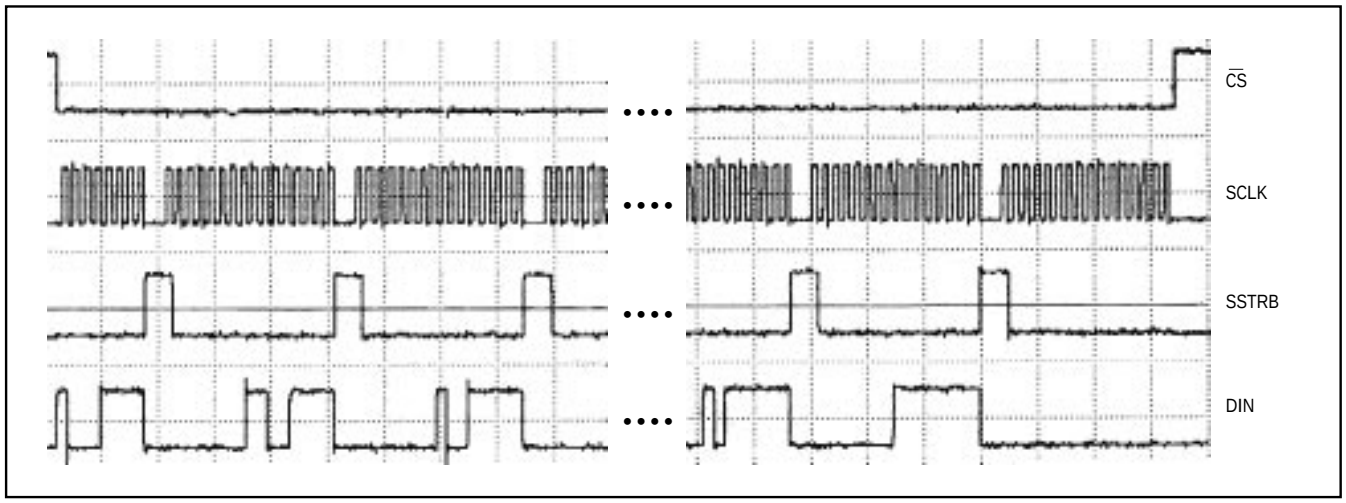


Figure 21. QSPI Assembly-Code Timing

TMS320C3x to MAX186 Interface

Figure 22 shows an application circuit to interface the MAX186/MAX188 to the TMS320 in external clock mode. The timing diagram for this interface circuit is shown in Figure 23.

Use the following steps to initiate a conversion in the MAX186/MAX188 and to read the results:

1) The TMS320 should be configured with CLKX (transmit clock) as an active-high output clock and CLKR (TMS320 receive clock) as an active-high input clock. CLKX and CLKR of the TMS320 are tied together with the SCLK input of the MAX186/MAX188.
2) The MAX186/MAX188 $\overline{CS}$ is driven low by the XF_ I/O port of the TMS320 to enable data to be clocked into DIN of the MAX186/MAX188.
3) An 8-bit word (1XXXXX11) should be written to the MAX186/MAX188 to initiate a conversion and place the device into external clock mode. Refer to Table 2 to select the proper XXXXX bit values for your specific application.
4) The SSTRB output of the MAX186/MAX188 is monitored via the FSR input of the TMS320. A falling edge on the SSTRB output indicates that the conversion is in progress and data is ready to be received from the MAX186/MAX188.

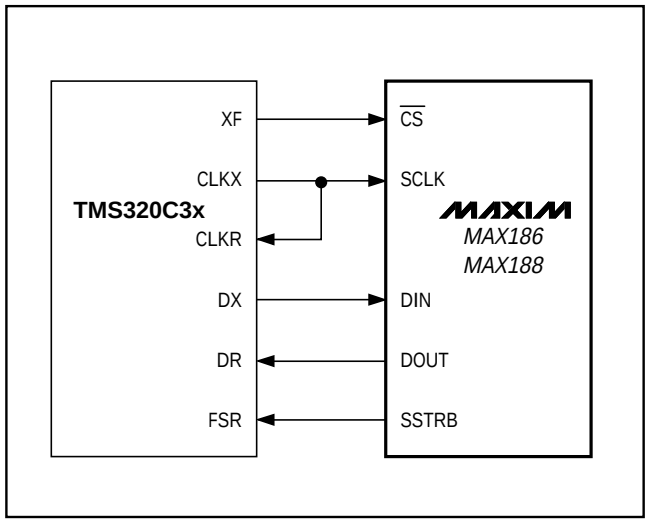


Figure 22. MAX186/MAX188 to TMS320 Serial Interface

5) The TMS320 reads in one data bit on each of the next 16 rising edges of SCLK. These data bits represent the 12-bit conversion result followed by four trailing bits, which should be ignored.
6) Pull $\overline{CS}$ high to disable the MAX186/MAX188 until the next conversion is initiated.

Low-Power, 8-Channel, Serial 12-Bit ADCs

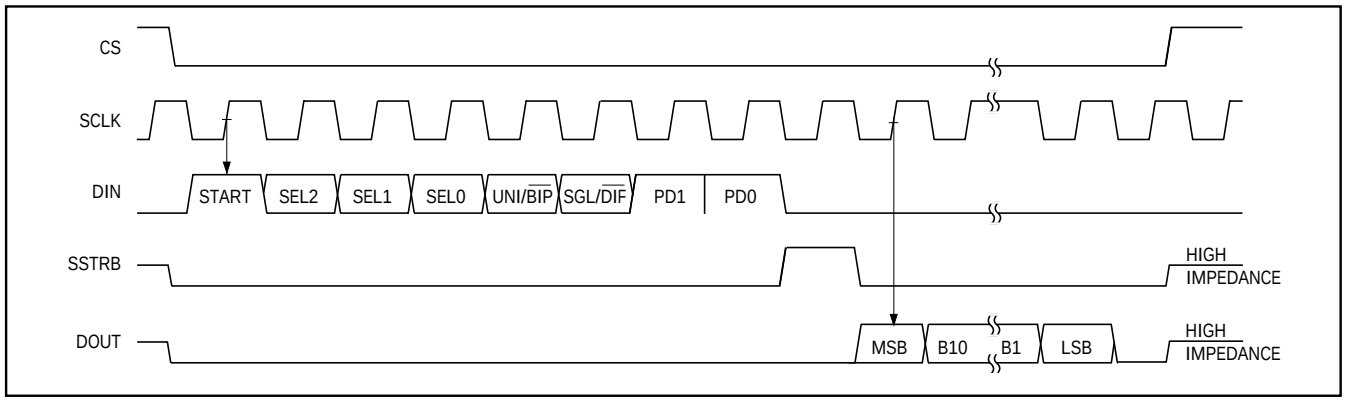
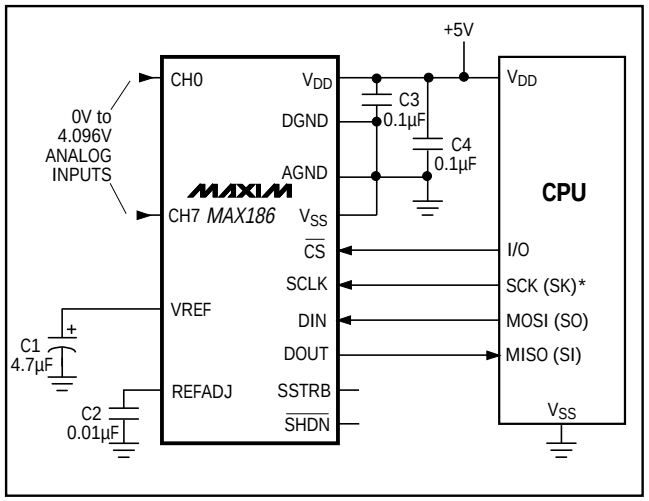
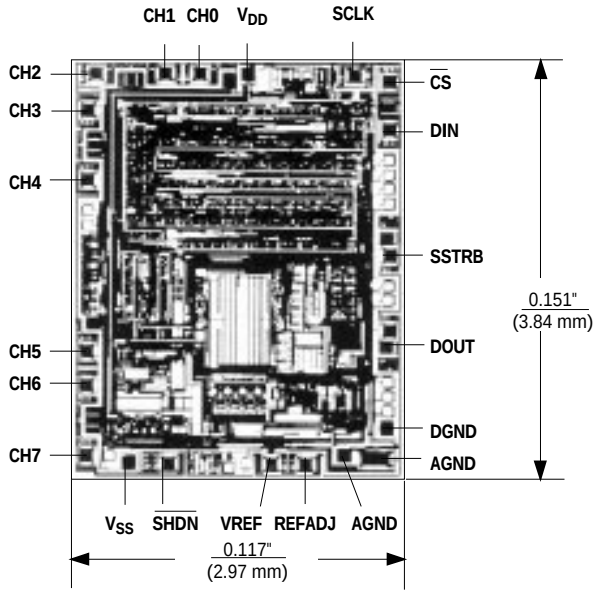


Figure 23. TMS320 Serial Interface Timing Diagram

Typical Operating Circuit



Chip Topography



MAX186/MAX188

TRANSISTOR COUNT: 2278;
SUBSTRATE CONNECTED TO V_{DD}

Ordering Information (continued)

PART†	TEMP. RANGE	PIN-PACKAGE
MAX188_CPP	0°C to +70°C	20 Plastic DIP
MAX188_CWP	0°C to +70°C	20 SO
MAX188_CAP	0°C to +70°C	20 SSOP
MAX188DC/D	0°C to +70°C	Dice*
MAX188_EPP	-40°C to +85°C	Plastic DIP
MAX188_EWP	-40°C to +85°C	20 SO
MAX188_EAP	-40°C to +85°C	20 SSOP
MAX188_MJP	-55°C to +125°C	20 CERDIP**

PART	TEMP. RANGE	BOARD TYPE
MAX186EVKIT-DIP	0°C to +70°C	Through-Hole

† NOTE: Parts are offered in grades A, B, C and D (grades defined in Electrical Characteristics). When ordering, please specify grade.
** Dice are specified at +25°C, DC parameters only.*
*** Contact factory for availability and processing to MIL-STD-883.*

Integrated Silicon Pressure Sensor Manifold Absolute Pressure Sensor On-Chip Signal Conditioned, Temperature Compensated and Calibrated

MPX4250A
MPXA4250A
SERIES

INTEGRATED
PRESSURE SENSOR
20 TO 250 kPA (2.9 TO 36.3 psi)
0.2 TO 4.9 V OUTPUT

The MPX4250A/MPXA4250A series Manifold Absolute Pressure (MAP) sensor for engine control is designed to sense absolute air pressure within the intake manifold. This measurement can be used to compute the amount of fuel required for each cylinder.

The MPX4250A/MPXA4250A series piezoresistive transducer is a state-of-the-art monolithic silicon pressure sensor designed for a wide range of applications, particularly those employing a microcontroller or microprocessor with A/D inputs. This transducer combines advanced micromachining techniques, thin-film metallization and bipolar processing to provide an accurate, high-level analog output signal that is proportional to the applied pressure. The small form factor and high reliability of on-chip integration make the Freescale sensor a logical and economical choice for the automotive system engineer.

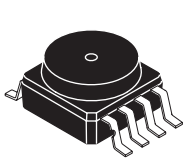
Features

- 1.5% Maximum Error Over 0° to 85°C
- Specifically Designed for Intake Manifold Absolute Pressure Sensing in Engine Control Systems
- Patented Silicon Shear Stress Strain Gauge
- Temperature Compensated Over -40° to +125°C
- Offers Reduction in Weight and Volume Compared to Existing Hybrid Modules
- Durable Epoxy Unibody Element or Thermoplastic Small Outline, Surface Mount Package
- Ideal for Non-Automotive Applications

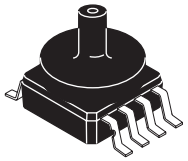
Typical Applications

- Turbo Boost Engine Control
- Ideally Suited for Microprocessor or Microcontroller-Based Systems

SMALL OUTLINE PACKAGES



MPXA4250A6U/6T1
CASE 482-01

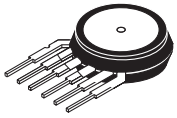


MPXA4250AC6U/C6T1
CASE 482A-01

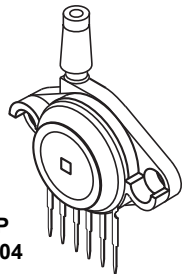
SMALL OUTLINE PACKAGE PIN NUMBERS			
1	N/C(1), (2)	5(2)	N/C
2	V_S	6(2)	N/C
3	GND	7(2)	N/C
4	V_{OUT}	8	N/C

1. Pin 1 in noted by the notch in the lead.
2. Pins 1, 5, 6, and 7 are internal device connections. Do not connect to external circuitry or ground.

UNIBODY PACKAGES



MPX4250A
CASE 867-08



MPX4250AP
CASE 867B-04

UNIBODY PACKAGE PIN NUMBERS			
1	V_{OUT}(1)	4	N/C(2)
2	GND	5	N/C(2)
3	V_S	6	N/C(2)

1. Pin 1 in noted by the notch in the lead.
2. Pins 4, 5, and 6 are internal device connections. Do not connect to external circuitry or ground.

ORDERING INFORMATION					
Device Type	**Options**	**Case No.**	**MPX Series Order Number**	**Packing Options**	**Device Marking**
SMALL OUTLINE PACKAGE(1) (MPXA4250A SERIES)					
Basic Elements	Absolute, Element Only	482	MPXA4250A6U	Rails	MPXA4250A
		482	MPXA4250A6T1	Tape & Reel	MPXA4250A
Ported Elements	Absolute, Axial Port	482A	MPXA4250AC6U	Rails	MPXA4250A
		482A	MPXA4250AC6T1	Tape & Reel	MPXA4250A
UNIBODY PACKAGE(2) (MPX4250A SERIES)					
Basic Element	Absolute, Element Only	867	MPX4250A	—	MPX4250A
Ported Elements	Absolute, Ported	867B	MPX4250AP	—	MPX4250AP

1. The MPXA4250A series pressure sensors are available in the basic element package or with pressure port fitting. Two packing options are offered for each type.
2. The MPX4250A series pressure sensors are available in the basic element package or with pressure port fittings providing mounting ease and barbed hose connections.

freescale semiconductor

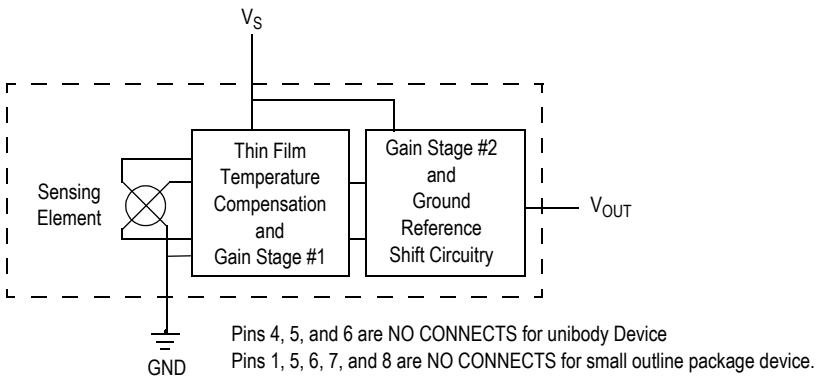


Figure 1. Fully Integrated Pressure Sensor Schematic

Table 1. Maximum Ratings(1)

Rating	Symbol	Value	Unit
Maximum Pressure(2) (P1 > P2)	P_{MAX}	1000	kPa
Storage Temperature	T_{STG}	-40 to +125	°C
Operating Temperature	T_A	-40 to +125	°C

1. TC = 25°C unless otherwise noted.
2. Exposure beyond the specified limits may cause permanent damage or degradation to the device.

Table 2. Operating Characteristics (V_S = 5.1 V_{DC}, T_A = 25°C unless otherwise noted, P1 > P2, Decoupling circuit shown in Figure 3 required to meet electrical specifications.)

Characteristic		Symbol	Min	Typ	Max	Units
Differential Pressure Range(1)		P_{OP}	20	—	250	kPa
Supply Voltage(2)		V_S	4.85	5.1	5.35	V_{DC}
Supply Current		I_O	—	7.0	10	mAdc
Minimum Pressure Offset(3) @ V_S = 5.1 Volts	(0 to 85°C)	V_{OFF}	0.133	0.204	0.264	V_{DC}
Full Scale Output(4) @ V_S = 5.1 Volts	(0 to 85°C)	V_{FSO}	4.826	4.896	4.966	V_{DC}
Full Scale Span(5) @ V_S = 5.1 Volts	(0 to 85°C)	V_{FSS}	—	4.692	—	V_{DC}
Accuracy(6)	(0 to 85°C)	—	—	—	±1.5	$\%V_{FSS}$
Sensitivity		$\Delta V/\Delta P$	—	20	—	mV/kPa
Response Time(7)		t_R	—	1.0	—	msec
Output Source Current at Full Scale Output		I_O+	—	0.1	—	mAdc
Warm-Up Time(8)		—	—	20	—	msec
Offset Stability(9)		—	—	±0.5	—	$\%V_{FSS}$

1. 1.0 kPa (kiloPascal) equals 0.145 psi.
2. Device is ratiometric within this specified excitation range.
3. Offset (V_{OFF}) is defined as the output voltage at the minimum rated pressure.
4. Full Scale Output (V_{FSO}) is defined as the output voltage at the maximum or full rated pressure.
5. Full Scale Span (V_{FSS}) is defined as the algebraic difference between the output voltage at full rated pressure and the output voltage at the minimum rated pressure.
6. Accuracy (error budget) consists of the following:
   - Linearity: Output deviation at any temperature from a straight line relationship with pressure over the specified pressure range.
   - Temperature Hysteresis: Output deviation at any temperature within the operating temperature range, after the temperature is cycled to and from the minimum or maximum operating temperature points, with zero differential pressure applied.
   - Pressure Hysteresis: Output deviation at any pressure within the specified range, when this pressure is cycled to and from the minimum or maximum rated pressure, at 25°C.
   - TcSpan: Output deviation over the temperature range of 0° to 85°C, relative to 25°C.
   - TcOffset: Output deviation with minimum rated pressure applied, over the temperature range of 0° to 85°C, relative to 25°C.
   - Variation from Nominal: The variation from nominal values, for Offset or Full Scale Span, as a percent of VFSS, at 25°C.
7. Response Time is defined as the time form the incremental change in the output to go from 10% to 90% of its final value when subjected to a specified step change in pressure.
8. Warm-up Time is defined as the time required for the product to meet the specified output voltage after the pressure is stabilized.
9. Offset stability is the product's output deviation when subjected to 1000 hours of Pulsed Pressure, Temperature Cycling with Bias Test.

Table 3. Mechanical Characteristics

Characteristics	Typ	Unit
Weight, Basic Element (Case 867)	4.0	Grams
Weight, Small Outline Package (Case 482)	1.5	Grams

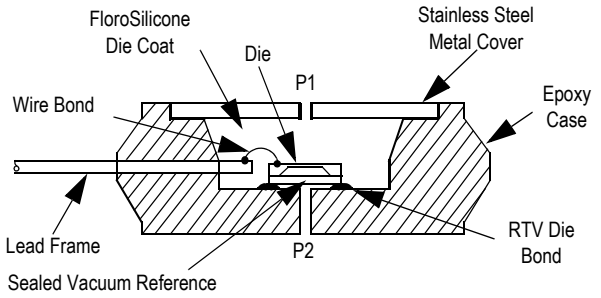


Figure 2. Cross Sectional Diagram (Not to Scale)

Figure 2 illustrates the absolute pressure sensing chip in the basic chip carrier (Case 867). A fluorosilicone gel isolates the die surface and wire bonds from the environment, while allowing the pressure signal to be transmitted to the sensor diaphragm.

The MPX4250A/MPXA4250A series pressure sensor operating characteristics and internal reliability and qualification tests are based on use of dry air as the pressure media. Media, other than dry air, may have adverse effects on sensor performance and long-term reliability.

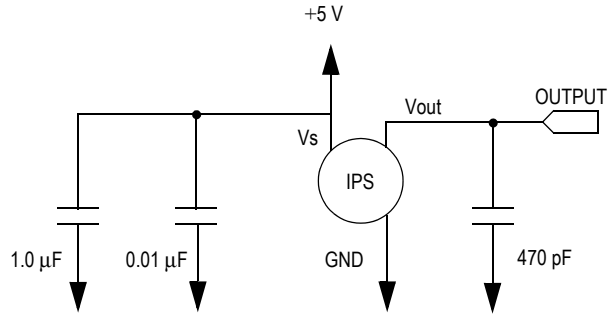


Figure 3. Recommended Power Supply Decoupling and Output Filtering

(For additional output filtering, please refer to Application Note AN1646.

Contact the factory for information regarding media compatibility in your application.

Figure 3 shows the recommended decoupling circuit for interfacing the output of the integrated sensor to the A/D input of a microprocessor or microcontroller.

Figure 4 shows the sensor output signal relative to pressure input. Typical, minimum, and maximum output curves are shown for operation over temperature range of 0° to 85°C using the decoupling circuit shown in Figure 3. The output will saturate outside of the specified pressure range.

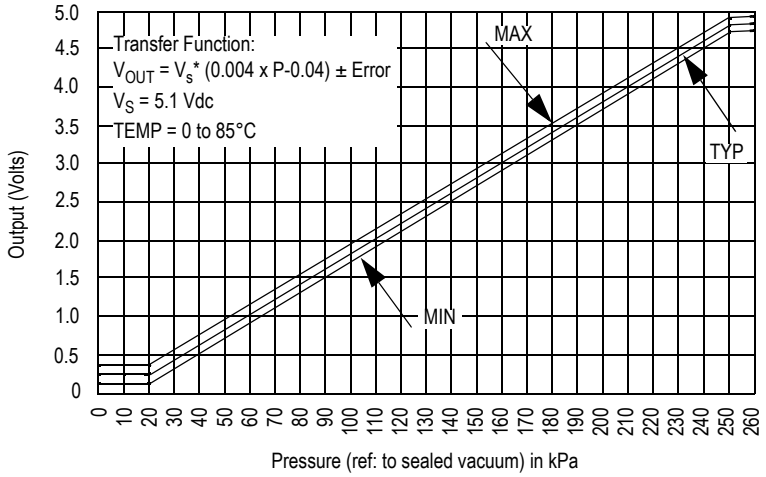
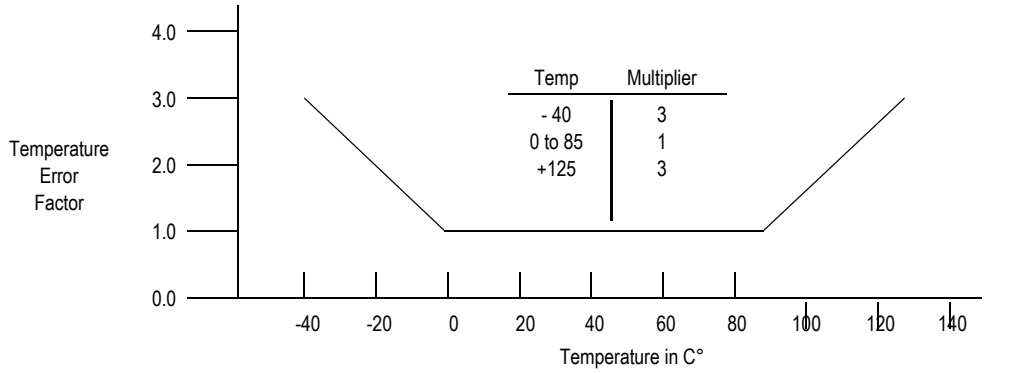


Figure 4. Output vs. Absolute Pressure

Transfer Function

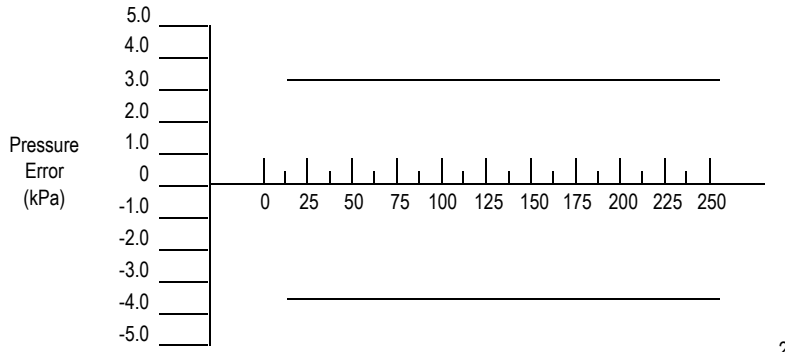
Nominal Transfer Value: $V_{OUT} = V_S (P \times 0.004 - 0.04)$
$\pm$ (Pressure Error $\times$ Temp. Factor $\times$ 0.004 $\times$ V_S)
$V_S = 5.1\ V \pm 0.25\ V_{DC}$

Temperature Error Band



NOTE: The Temperature Multiplier is a linear response from 0× to -40°C and from 85° to 125°C.

Pressure Error Band



Pressure	Error (Max)
20 to 250 kPa	±3.45 (kPa)

INFORMATION FOR USING THE SMALL OUTLINE PACKAGE (CASE 482)

MINIMUM RECOMMENDED FOOTPRINT FOR SURFACE MOUNTED APPLICATIONS

Surface mount board layout is a critical portion of the total design. The footprint for the surface mount packages must be the correct size to ensure proper solder connection interface between the board and the package. With the correct Footprint, the packages will self align when subjected to a solder reflow process. It is always recommended to design boards with a solder mask layer to avoid bridging and shorting between solder pads.

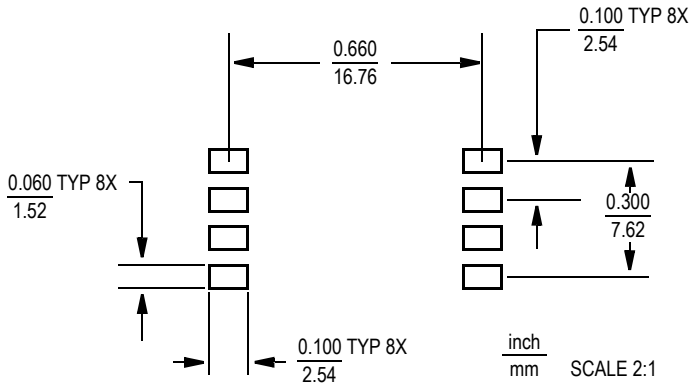
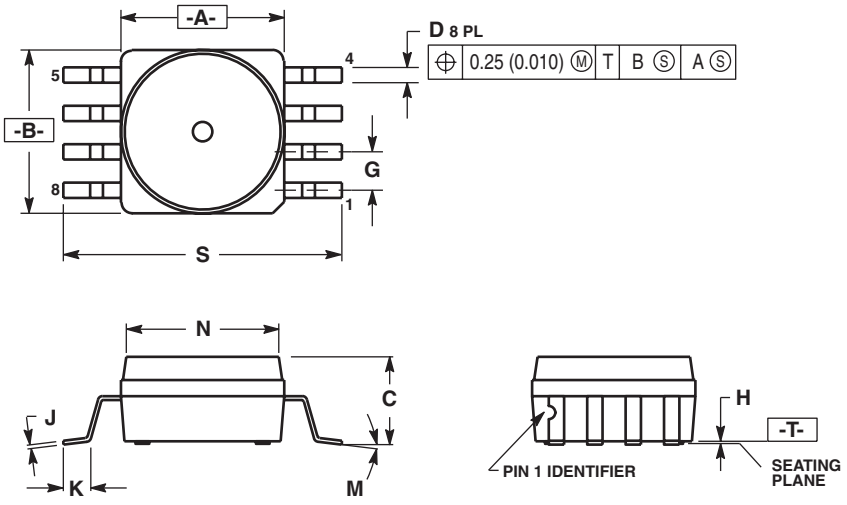


Figure 5. SOP Footprint (Case 482)

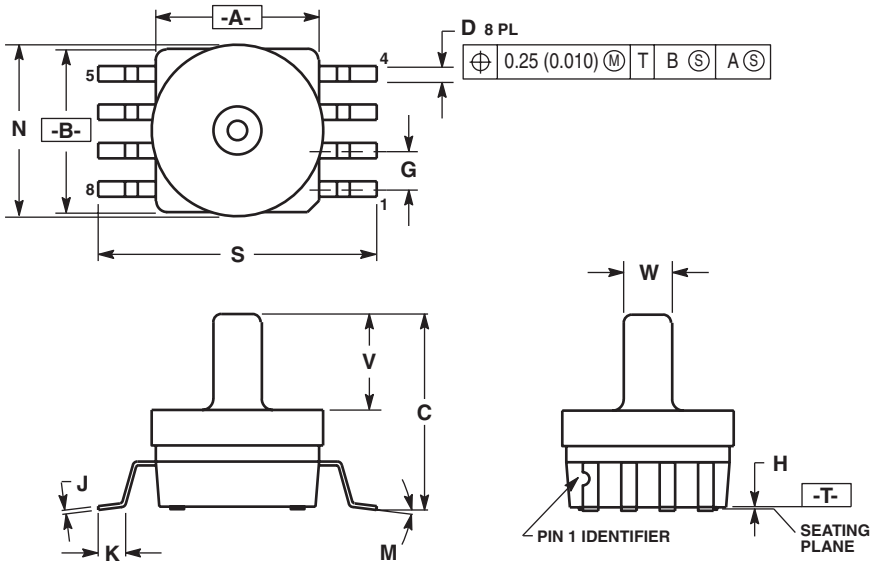
PACKAGE DIMENSIONS



NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.
3. DIMENSION A AND B DO NOT INCLUDE MOLD PROTRUSION.
4. MAXIMUM MOLD PROTRUSION 0.15 (0.006).
5. ALL VERTICAL SURFACES 5° TYPICAL DRAFT.

	INCHES		MILLIMETERS	
DIM	MIN	MAX	MIN	MAX
A	0.415	0.425	10.54	10.79
B	0.415	0.425	10.54	10.79
C	0.212	0.230	5.38	5.84
D	0.038	0.042	0.96	1.07
G	0.100 BSC		2.54 BSC	
H	0.002	0.010	0.05	0.25
J	0.009	0.011	0.23	0.28
K	0.061	0.071	1.55	1.80
M	0°	7°	0°	7°
N	0.405	0.415	10.29	10.54
S	0.709	0.725	18.01	18.41

CASE 482-01
ISSUE O
SMALL OUTLINE PACKAGE

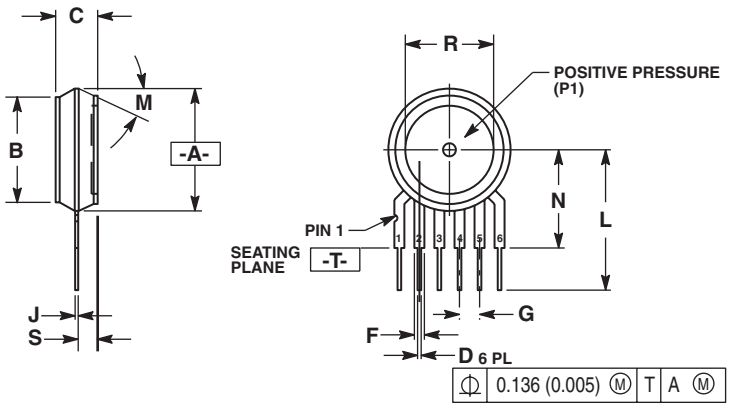


NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.
3. DIMENSION A AND B DO NOT INCLUDE MOLD PROTRUSION.
4. MAXIMUM MOLD PROTRUSION 0.15 (0.006).
5. ALL VERTICAL SURFACES 5° TYPICAL DRAFT.

	INCHES		MILLIMETERS	
DIM	MIN	MAX	MIN	MAX
A	0.415	0.425	10.54	10.79
B	0.415	0.425	10.54	10.79
C	0.500	0.520	12.70	13.21
D	0.038	0.042	0.96	1.07
G	0.100 BSC		2.54 BSC	
H	0.002	0.010	0.05	0.25
J	0.009	0.011	0.23	0.28
K	0.061	0.071	1.55	1.80
M	0°	7°	0°	7°
N	0.444	0.448	11.28	11.38
S	0.709	0.725	18.01	18.41
V	0.245	0.255	6.22	6.48
W	0.115	0.125	2.92	3.17

CASE 482A-01
ISSUE A
SMALL OUTLINE PACKAGE

PACKAGE DIMENSIONS



NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.
3. DIMENSION -A- IS INCLUSIVE OF THE MOLD STOP RING. MOLD STOP RING NOT TO EXCEED 16.00 (0.630).

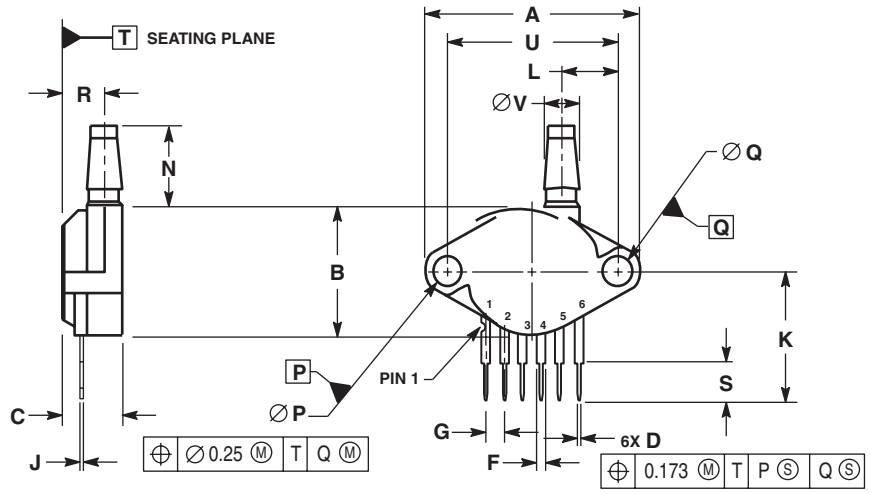
	INCHES		MILLIMETERS	
DIM	MIN	MAX	MIN	MAX
A	0.595	0.630	15.11	16.00
B	0.514	0.534	13.06	13.56
C	0.200	0.220	5.08	5.59
D	0.027	0.033	0.68	0.84
F	0.048	0.064	1.22	1.63
G	0.100 BSC		2.54 BSC	
J	0.014	0.016	0.36	0.40
L	0.695	0.725	17.65	18.42
M	30° NOM		30° NOM	
N	0.475	0.495	12.07	12.57
R	0.430	0.450	10.92	11.43
S	0.090	0.105	2.29	2.66

STYLE 1:
PIN 1. VOUT
2. GROUND
3. VCC
4. V1
5. V2
6. VEX

STYLE 2:
PIN 1. OPEN
2. GROUND
3. -VOUT
4. VSUPPLY
5. +VOUT
6. OPEN

STYLE 3:
PIN 1. OPEN
2. GROUND
3. +VOUT
4. +VSUPPLY
5. -VOUT
6. OPEN

CASE 867-08
ISSUE N
UNIBODY PACKAGE



NOTES:
1. DIMENSIONS ARE IN MILLIMETERS.
2. DIMENSIONING AND TOLERANCING PER ASME Y14.5M, 1994.

	MILLIMETERS	
DIM	MIN	MAX
A	29.08	29.85
B	17.4	18.16
C	7.75	8.26
D	0.68	0.84
F	1.22	1.63
G	2.54 BSC	
J	0.36	0.41
K	17.65	18.42
L	7.37	7.62
N	10.67	11.18
P	3.89	4.04
Q	3.89	4.04
R	5.84	6.35
S	5.59	6.1
U	23.11 BSC	
V	4.62	4.93

STYLE 1:
PIN 1. V_{OUT}
2. GROUND
3. V_{CC}
4. V1
5. V2
6. V_{EX}

CASE 867B-04
ISSUE F
UNIBODY PACKAGE

NOTES

NOTES

NOTES

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

MPX4250A
Rev. 5
05/2005

Advanced Information
Preliminary

OV6620/OV6120

OV6620 SINGLE-CHIP CMOS CIF COLOR DIGITAL CAMERA
OV6120 SINGLE-CHIP CMOS CIF B&W DIGITAL CAMERA

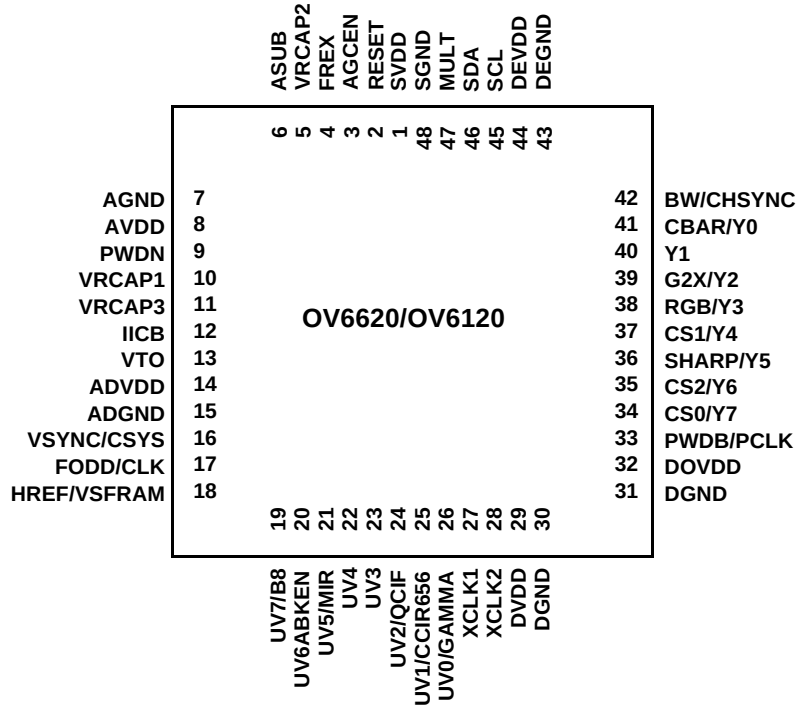
Features

- **101,376 pixels, 1/4" lens, CIF/QCIF format**
- **Progressive scan read out**
- **Data format - YCrCb 4:2:2, GRB 4:2:2, RGB Raw Data**
- **8/16 bit video data: CCIR601, CCIR656, ZV port**
- **Wide dynamic range, anti-blooming, zero smearing**
- **Electronic exposure / Gain / white balance control**
- **Image enhancement - brightness, contrast, gamma, saturation, sharpness, window, etc.**
- **Internal/external synchronization**
- **Frame exposure/line exposure option**
- **5-Volt operation, low power dissipation**
  - < 80 mW active power
  - < 10 µA in power-save mode
- **Gamma correction (0.45/0.55/1.00)**
- **I^2C programmable (400 kb/s):**
  - color saturation, brightness, contrast, white balance, exposure time, gain

General Description

The OV6620 (color) and OV6120 (black and white) CMOS Image sensors are single-chip video/imaging camera devices designed to provide a high level of functionality in a single, small-footprint package. Both devices incorporate a 352 x 288 image array capable of operating up to 60 frames per second image capture. Proprietary sensor technology utilizes advanced algorithms to cancel Fixed Pattern Noise (FPN), eliminate smearing, and drastically reduce blooming. All needed camera functions including exposure control, gamma, gain, white balance, color matrix, windowing, and more, are programmable through an I^2C interface. Both devices can be programmed to provide image output in either 8-bit or 16-bit digital formats.

Applications include: Video Conferencing, Video Phone, Video Mail, Still Image, and PC Multimedia.



OV6620/OV6120 PIN ASSIGNMENTS

Array Elements (CIF) (QCIF)	356 x 292 (176 x 144)
Pixel Size	9.0 x 8.2 µm
Image Area	3.1 x 2.5 mm
Max Frames/Sec	Up to 60 FPS
Electronic Exposure	Up to 500 : 1 (for selected FPS)
Scan Mode	progressive
Gamma Correction	0.45/.55/1.0
Min. Illumination (3000K)	OV6620 - < 3 lux @ f1.2 OV6120 - < 0.5 lux @ f1.2
S/N Ratio (Digital Camera Out)	> 48 dB (AGC = Off, Gamma = 1)
FPN	< 0.03% V_{P-P}
Dark Current	< 0.2 nA/cm^2
Dynamic Range	> 72 dB
Power Supply	5VDC, ±5% (Anal.) 5VDC or 3.3VDC (DIO)
Power Requirements	< 80mW Active < 30µW Standby
Package	48 pin LCC

OmniVision Technologies, Inc. 930 Thompson Place Sunnyvale, CA 94086 U.S.A.
Tel: (408) 733-3030 Fax: (408) 733-3061
e-mail: info@ovt.com
Website: http://www.ovt.com

Version 1.11, 14 May 1999

Table 1. Pin Description

Pin No.	Name	Pin Type	Function/Description
01	SVDD	V_{in}	Array power (+5VDC)
02	RESET	Function (Default = 0)	Chip Reset, active high
03	AGCEN	Function (Default = 0)	Automatic Gain Control (AGC) selection "0" - Disable AGC "1" - Enable AGC NOTE: This function is disabled when OV6620/OV6120 sensor is configured in I^2C mode.
04	FREX	Function (Default = 0)	Frame Exposure Control "0" - Disable Frame Exposure Control "1" - Enable Frame Exposure Control
05	VRCAP2	V_{ref} (2.5V)	Array reference. Connect to ground through 0.1 uF capacitor.
06	ASUB	V_{in}	Analog substrate voltage
07	AGND	V_{in}	Analog ground
08	AVDD	V_{in}	Analog power supply (+5VDC)
09	PWDN	Function (Default = 0)	Power down mode selection "0" - normal mode "1" - power down mode
10	VrCAP1	N/C	Internal voltage reference. Connect to ground through 0.1 µF capacitor.
11	VrCAP3		Internal voltage reference. Connect to ground through 1 µF capacitor.
12	IICB	Function (Default = 0)	I^2C enable selection "0" - Enable I^2C "1" - Enable autocontrol mode
13	VTO	O	Luminance Composite Signal Output
14	ADVDD	V_{in}	Analog power supply (+5VDC)
15	ADGND	V_{in}	Analog signal ground
16	VSYNC/CSYS	I/O	Vertical sync output. At power up, read as CSYS.
17	FODD/CLK	I/O	Field ID FODD output or main clock output
18	HREF/VSFRAM	I/O	HREF output. At power up, read as VSFRAM
19	UV7/B8	I/O	Bit 7 of U video component output. At power up, sampled as B8.
20	UV6/ABKEN	I/O	Bit 6 of U video component output. At power up, sampled as ABKEN.
21	UV5/MIR	I/O	Bit 5 of U video component output. At power up, sampled as MIR.
22	UV4	I/O	Bit 4of U video component output.
23	UV3	I/O	Bit 3 of U video component output.

Table 1. Pin Description

Pin No.	Name	Pin Type	Function/Description
24	UV2/QCIF	I/O	Bit 2 of U video component output. At power up, sampled as QCIF.
25	UV1/CC656	I/O	Bit 1 of U video component output. At power up, sampled as CC656.
26	UV0/GAMMA	I/O	Bit 0 of U video component output. At power up, sampled as GAMMA.
27	XCLK1	I	Crystal clock input
28	XCLK2	O	Crystal clock output
29	DVDD	V_{in}	Digital power supply (+5VDC)
30	DGND	V_{in}	Digital ground
31	DOGND	V_{in}	Digital interface output buffer ground
32	DOVDD	V_{in}	Digital interface output buffer power supply (+5VDC)
33	PCLK/PWDB	I/O	PCLK output. At power up sampled as PWDB.
34	Y7/CS0	I/O	Bit 7 of Y video component output. At power up, sampled as CS0.
35	Y6/CS2	I/O	Bit 6 of Y video component output. At power up, sampled as CS2.
36	Y5/SHARP	I/O	Bit 5 of Y video component. At power up, sampled as SHARP.
37	Y4/CS1	I/O	Bit 4 of Y video component. At power up, sampled as CS1
38	Y3/RGB	I/O	Bit 3 of Y video component output. At power up, sampled as RGB.
39	Y2/G2X	I/O	Bit 2 of Y video component output. At power up, sampled as G2X.
40	Y1	I/O	Bit 1 of Y video component output.
41	Y0/CBAR	I/O	Bit 0 of Y video component output. At power up, sampled as CBAR.
42	CHSYNC/BW	I/O	CHSYNC output. At power up, sampled as BW.
43	DEGND	V_{in}	Decoder ground.
44	DEVDD	V_{in}	Decoder power supply (+5VDC)
45	SCL	I	I^2C serial interface clock input
46	SDA	I/O	I^2C serial interface data input and output.
47	MULT	Function (Default = 0)	I^2C slave selection "0" - Select single slave ID "1" - Enable multiple (8) slaves
48	SGND	V_{in}	Array ground

1. Functional Description

*(**Note:** References to color features do not apply to the OV6120 B&W Digital Image Sensor.)*

1.1 Overview

Referring to Figure 1, OV6620/OV6120 CMOS Image Sensor Block Diagram below, the OV6620 sensor includes a 356 x 292 resolution image array, an analog signal processor, dual 8-bit Analog-to-Digital converters, analog video multiplexer, digital data formatter and video port, I^2C interface and registers, digital controls including timing block, exposure, and black and white balance.

The OV6620/OV6120 sensor is a 1/4-inch CMOS imaging device. The sensor contains approximately 101,376 pixels. Its design is based on a field integration read-out system with line-by-line transfer and an electronic shutter with a synchronous pixel read out scheme. The color filter of the sensor consists of a primary color RG/GB array arranged in line-alternating fashion.

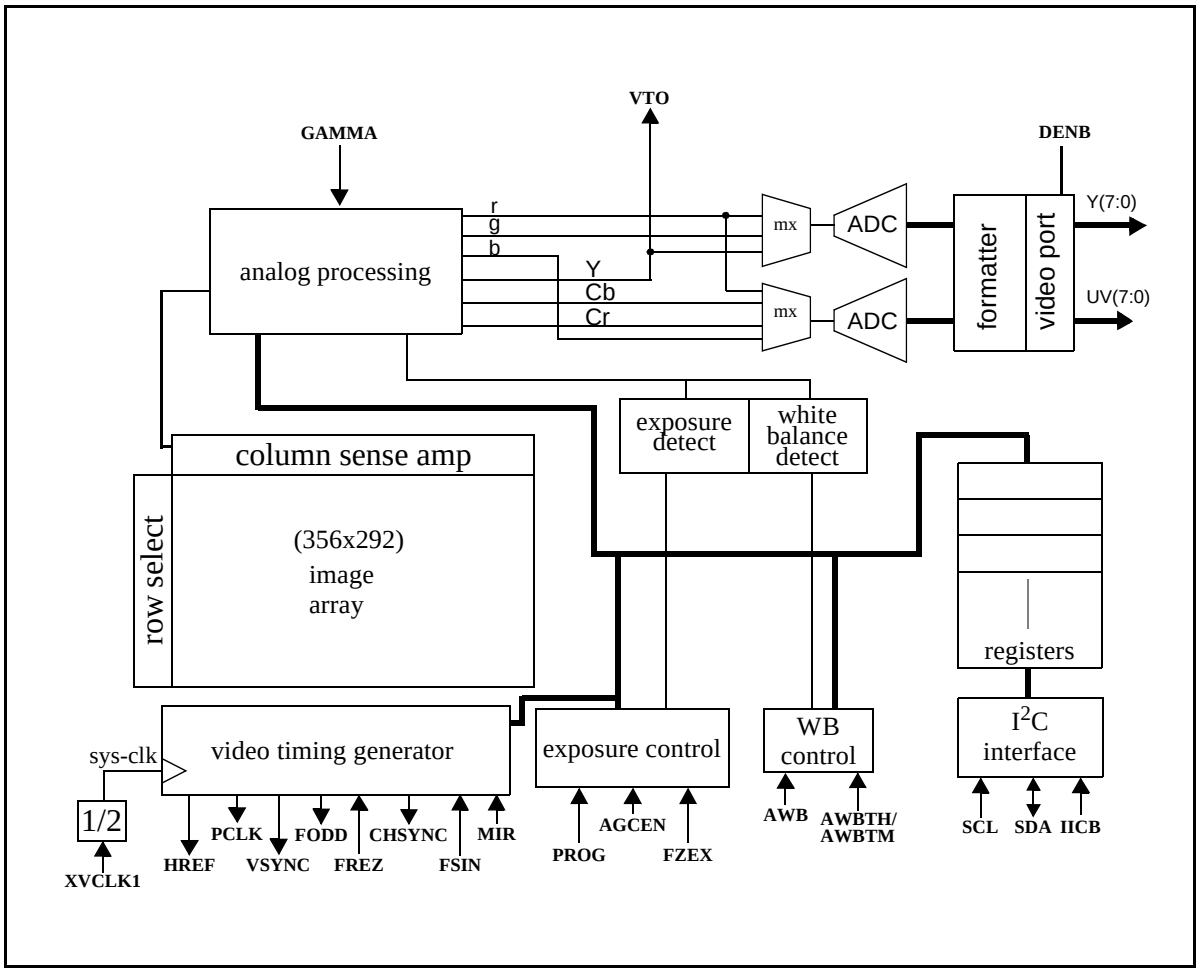


Figure 1. OV6620/OV6120 CMOS Image Sensor Block Diagram

1.2 Analog Processor Circuits

1.2.1 Overview

The image is captured by the 356 x 592 pixel image array and routed to the analog processing section where the majority of signal processing occurs. This block contains the circuitry which performs color separation, matrixing, Automatic Gain Control (AGC), gamma correction, color correction, color balance, black level calibration, “knee” smoothing, aperture correction, and controls for picture luminance, chrominance, and anti-alias filtering. The analog video signals are based on the following formula:

Y = 0.59G + 0.31R + 0.11B
U = R-Y
V = B-Y

where R,G,B are the equivalent color components in each pixel.

A YCrCb format is also supported, based on the formula below:

Y = 0.59G + 0.31R + 0.11B
Cr = 0.713 x (R - Y)
Cb = 0.564 x (B - Y)

The YCrCb/RGB Raw Data signal from the analog processing section is fed to two on-chip 8-bit Analog-to-Digital (A-to-D) converters: one for the Y/RG channel and one shared by the CrCb/BG channels. The A-to-D converted data stream is further conditioned in the digital formatter. The processed signal is delivered to the digital video port through the video multiplexer which routes the user-selected 16-, 8-, or 4-bit video data the correct output pins.

The on-chip 8-bit A-to-D converters operate at up to 9 MHz, fully synchronous to the pixel rate. Actual conversion rate is set as a function of the frame rate. A-to-D black-level calibration circuitry ensures the following:

– the black level of Y/RGB is normalized to a value of 16

– the peak white level is limited to 240

– CrCb black level is 128

– Peak/Bottom is 240/16

– RGB raw data output range is 16/240

(Note: Values 0 and 255 are reserved for sync flag)

1.2.2 Image Processing

The algorithm used for the electronic exposure control is based on the brightness of the full image. The exposure is optimized for a “normal” scene which assumes the subject is well lit relative to the background. In situations where the image is not well lit, the Automatic Exposure Control (AEC) White/Black ratio may be adjusted to suit the needs of the application.

Additional on-chip functions include Automatic Gain Control (AGC) which provides a gain boost of up to 24dB. White balance control enables setting of proper color temperature and can be programmed for automatic or manual operation. Separate saturation, brightness, contrast, and sharpness adjustments allow for further fine tuning of the picture quality and characteristics. The OV6620 image sensor also provides control over the White Balance ratio for increasing/decreasing the image field Red/Blue component ratio. The sensor provides a default setting which may be sufficient for many applications.

1.2.3 Windowing

The windowing feature of the OV6620/OV6120 image sensors allows user-definable window sizing as required by the application. Window size setting (in pixels) ranges from 2 x 2 to 356 x 292, and can be positioned anywhere inside the 356 x 292 boundary. Note that modifying window size and/or position does not change frame or data rate. The OV6620/OV6120 imager alters the assertion of the HREF signal to be consistent with the programmed horizontal and vertical region. The default output window is 352 x 288.

1.2.4 Zoom Video Port (ZV)

The OV6620/OV6120 image sensor includes a Zoom Video (ZV) function that supports standard ZV Port interface timing. Signals available include VSYNC, CHSYNC, PCLK and 16-bit data bus: Y[7:0] and UV[7:0]. The rising edge of PCLK clocks data into the ZV port. See Figure 2, Zoom Video Port Timing below.

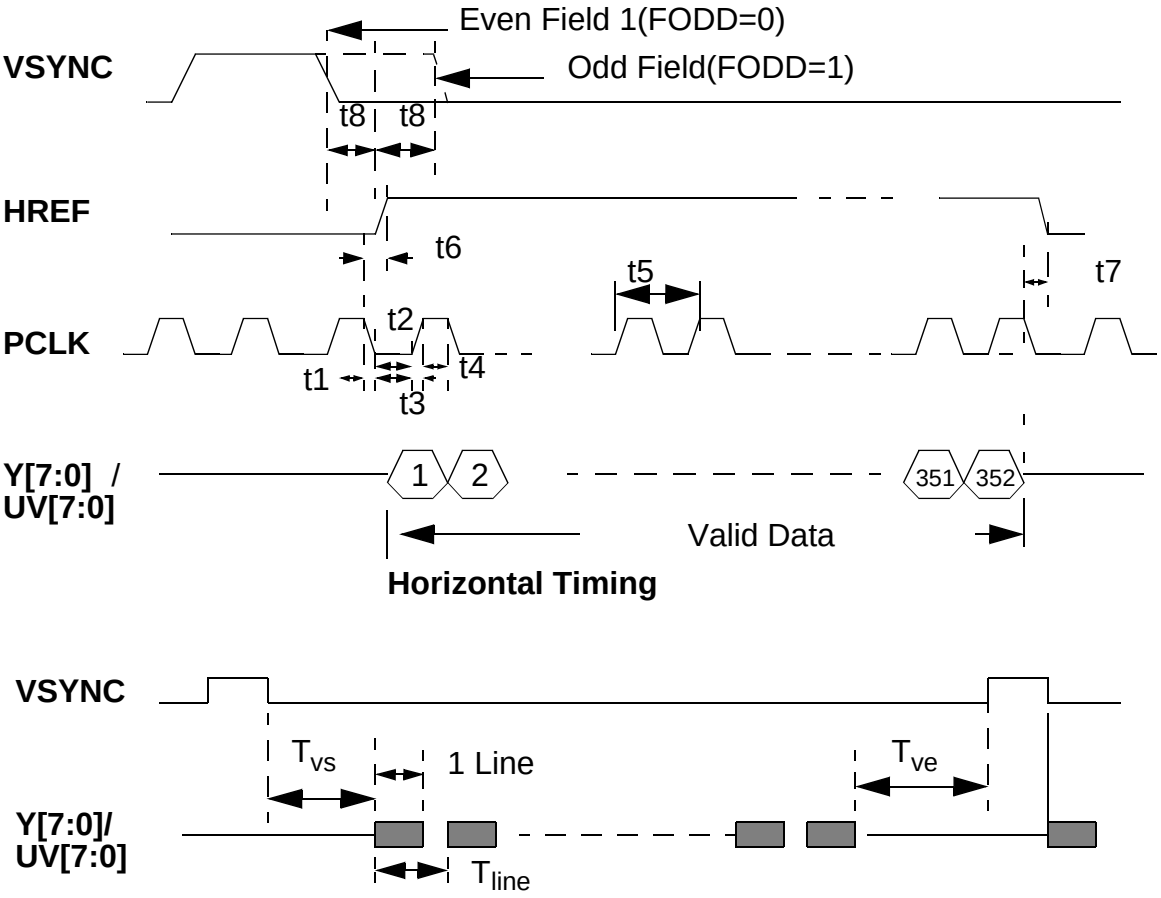


Figure 2. Zoom Video Port Timing

Notes:

1. Zoom Video Port format output signal includes:
   VSYNC: Vertical sync pulse.
   HREF: Horizontal valid data output window.
   PCLK: Pixel clock used to clock valid data and CHSYNC into Zoom V Port. Default frequency is 8.86MHz when use 17.73MHz as system clock. Rising edge of PCLK is used to clock the 16 Bit data.
   Y[7:0]: 8 Bit luminance data bus.
   UV[7:0]: 8 Bit chrominance data bus.
2. All timing parameters are provided in Table 13. Zoom Video Port AC Parameters

1.2.5 QCIF Format

A QCIF mode is available for applications where high resolution image capture is not required. When programmed in this mode, the pixel rate is reduced by one-half. Default resolution is 176 x 144 pixels and can be user-programmed for other resolutions. Refer to Table 7. QCIF Digital Output Format (YUV, beginning of line) and Table 8. QCIF Digital Output Format (RGB Raw Data, Beginning of Line) for further information.

1.2.6 Video Output

The video output port of the OV6620/OV6120 image sensors provides a number of output format/standard options to suit many different application requirements. Table 2. Digital Output Formats, below, indicates the output formats available. These formats are user programmable through the I^2C interface (See Section 3.1 I^2C Bus Protocol Format).

The OV6620/OV6120 imager supports both CCIR601 and CCIR656 output formats in the following configurations (See Table 3. 4:2:2 16-bit Format for further details):

– 16-bit, 4:2:2 format

(This mode complies with the 60/50 Hz CCIR601 timing standard. See Table 3. 4:2:2 16-bit Format below)

– 8-bit data mode

(In this mode, video information is output in Cb Y Cr Y order using the Y port only and running at twice the pixel rate during which the UV port is inactive. See Table 4. 4:2:2 8-bit Format below)

– 4-bit nibble mode

(In the nibble mode, video output data appears at bits Y4-Y7. The clock rate for the output runs at twice the normal output speed when in B/W mode, and 4 times the normal output speed in when in color mode.)

– 704 x 288 format

(When programmed for this mode, the OV6620/OV6120 pixel clock is doubled and the video output sequence is Y0Y0Y1Y1 ... and U0U0V0V0 ... See Figure 3, Pixel Data Bus (YUV Output), below.)

The OV6620/OV6120 imaging devices provide VSYNC, HREF, PCLK, FODD, CHSYNC as standard output video timing signals**.**

The OV6620/OV6120 image sensor can also be programmed to provide video output in RGB Raw Data 16-bit/8-bit/4-bit format. The output sequence is matched to the OV6620 Color Filter Pattern (See Section Figure 4. Pixel Data Bus (RGB Output), below):

– Y channel output sequence is G R G R

– UV channel output sequence is B G B G

For 8-bit RGB Raw Data video output appears on the Y channel (with an output sequence of B G R G) and the UV channel is disabled.

In RGB Raw Data CCIR656 modes, the OV6620/OV6120 imager asserts SAV (Start of Active Video) and EAV (End of Active Video) to indicate the beginning and the ending of HREF window. As a result, SAV and EAV change with the active pixel window. The 8-bit RGB raw data is also accessible without SAV and EAV information.

The OV6620/OV6120 imagers offer flexibility in YUV output format. The devices may be programmed as standard YUV 4:2:2. These devices may be configured to “swap” the U V sequence. When swapped, the UV channel output format for 16-bit configurations becomes:

– V U V U...etc.

and for 8-bit configurations becomes:

– V Y U Y ...etc.

A third format is available for the 8-bit configurations and OV6620/OV6120 enables the Y/UV sequence swap:

– Y U Y V ...etc.

The OV6620 color single-IC camera can be configured for use as a black and white imaging device. In this mode, vertical resolution is greater than in color. Video data output is provided at the Y port (pins 34:41) and the UV port is tri-stated. The data (Y/RGB) output rate is equivalent to operating in 16-bit mode.

The Y/UV or RGB output byte MSB and LSB can be reverse-ordered on the OV6620/OV6120 device. The Y7 - Y0 default sequence sets Y7 as MSB and Y0 as LSB. Programming a reverse order configuration sets Y7 as LSB and Y0 is MSB, with bits Y2-Y6 reversed-ordered appropriately.

Table 2. Digital Output Formats

Resolution	Pixel Clock	352 x 288	704 x 288	176 x 144
YUV 4:2:2	16-bit	Y	Y	Y
	8-bit	Y	Y	Y
	CCIR656	Y	Y	Y
	Nibble	Y	Y	Y
RGB	16-bit	Y	Y	Y
	8-bit	Y	Y	Y
	CCIR656[1]	Y	Y	Y
	Nibble	Y	Y	Y
Y/UV swap[2]	16-bit			
	8-bit	Y	Y	Y
U/V swap	YUV[3]	Y	Y	Y
	RGB[4]	Y	Y	Y
YG	16-bit	Y	Y	Y
	8-bit			
One Line	16-bit	Y		
	8-bit			
MSB/LSB swap[5]		Y	Y	Y

Notes:

("Y" indicates mode/combination is supported by OV6620/OV6120.)

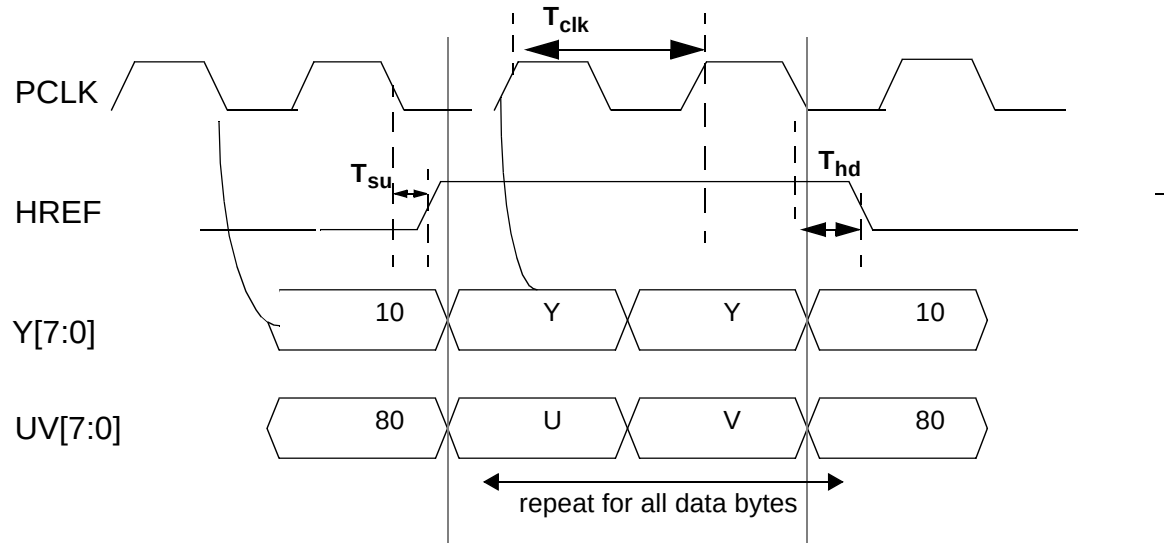
1. When in RGB CCIR656 format, output is 8 bits. SAV and EAV are inserted at the beginning and ending of HREF, which synchronize the acquisition of Vsync and Hsync. In this format, an 8-bit data bus configuration (without VSYNC and CHSYNC) may be used.
2. Y/UV swap is valid only in 8-bit mode. Y channel output sequence is Y U Y V...
3. In YUV format, U/V swap means UV channel output sequence swap. V U V U... for 16 bit; V Y U Y ... for 8-bit.
4. In RGB format, U/V swap means neighbor row B R output sequence swap. Refer to RGB raw data output format for further details

Table 3. 4:2:2 16-bit Format

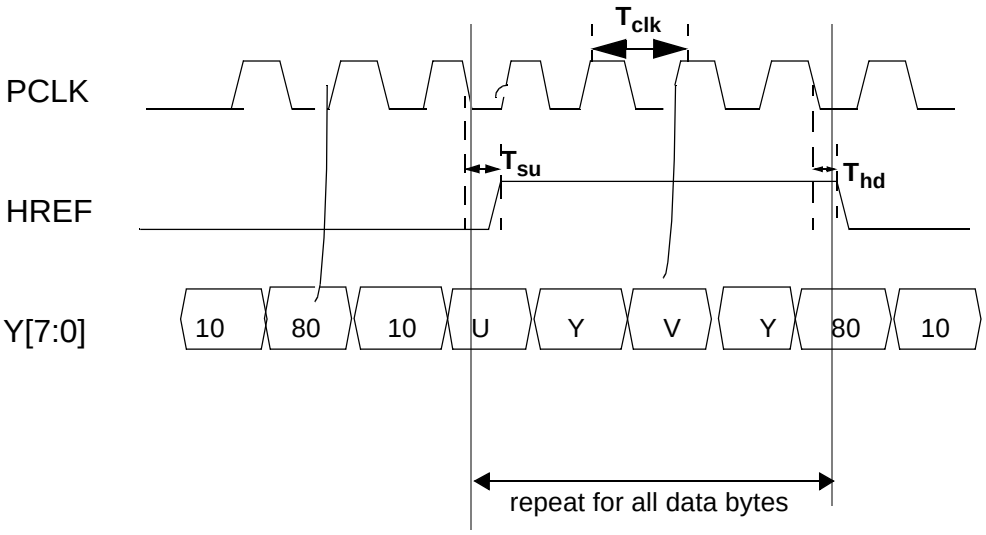
Data Bus	Pixel Byte Sequence					
Y7	Y7	Y7	Y7	Y7	Y7	Y7
Y6	Y6	Y6	Y6	Y6	Y6	Y6
Y5	Y5	Y5	Y5	Y5	Y5	Y5
Y4	Y4	Y4	Y4	Y4	Y4	Y4
Y3	Y3	Y3	Y3	Y3	Y3	Y3
Y2	Y2	Y2	Y2	Y2	Y2	Y2
Y1	Y1	Y1	Y1	Y1	Y1	Y1
Y0	Y0	Y0	Y0	Y0	Y0	Y0
UV7	U7	V7	U7	V7	U7	V7
UV6	U6	V6	U6	V6	U6	V6
UV5	U5	V5	U5	V5	U5	V5
UV4	U4	V4	U4	V4	U4	V4
UV3	U3	V3	U3	V3	U3	V3
UV2	U2	V2	U2	V2	U2	V2
UV1	U1	V1	U1	V1	U1	V1
UV0	U0	V0	U0	V0	U0	V0
Y FRAME	0	1	2	3	4	5
UV FRAME	0		2		4	

Table 4. 4:2:2 8-bit Format

Data Bus	Pixel Byte Sequence							
Y7	U7	Y7	V7	Y7	U7	Y7	V7	Y7
Y6	U6	Y6	V6	Y6	U6	Y6	V6	Y6
Y5	U5	Y5	V5	Y5	U5	Y5	V5	Y5
Y4	U4	Y4	V4	Y4	U4	Y4	V4	Y4
Y3	U3	Y3	V3	Y3	U3	Y3	V3	Y3
Y2	U2	Y2	V2	Y2	U2	Y2	V2	Y2
Y1	U1	Y1	V1	Y1	U1	Y1	V1	Y1
Y0	U0	Y0	V0	Y0	U0	Y0	V0	Y0
Y FRAME	0		1		2		3	
UV FRAME	0 1				2 3			



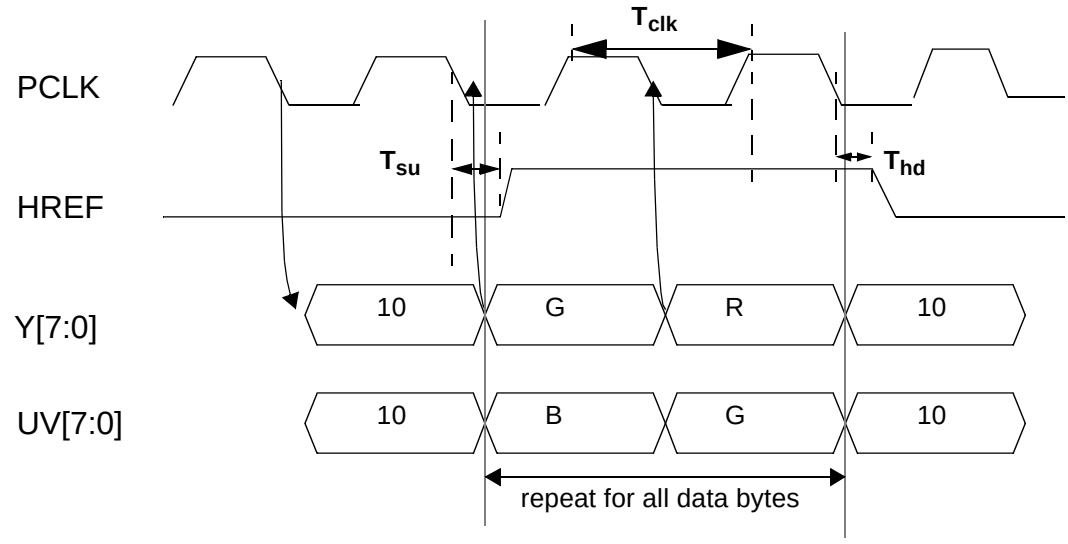
Pixel Data 16-bit Timing
(PCLK rising edge latches data bus)



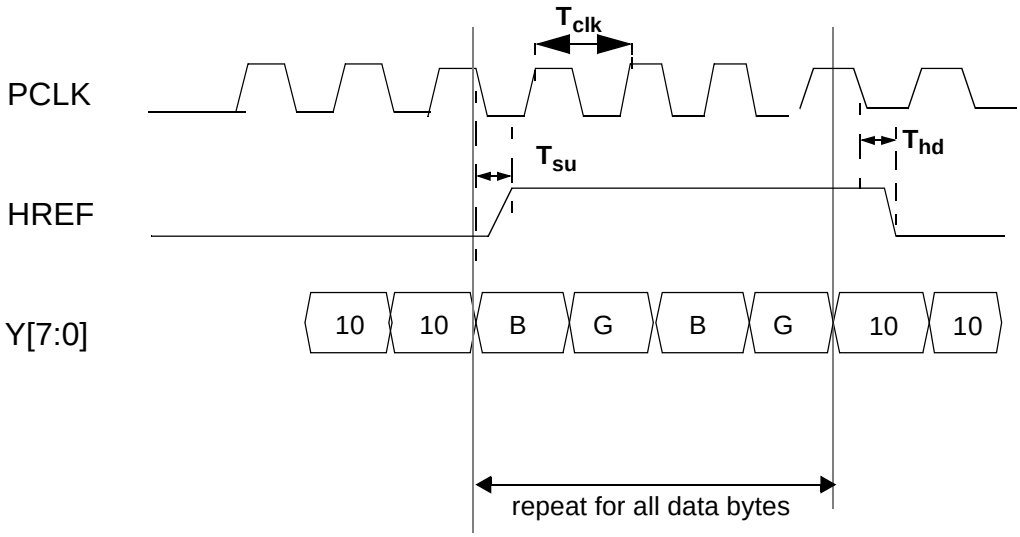
Pixel Data 8-bit Timing
(PCLK rising edge latches data bus)

Note: T_{clk} is pixel clock period. When the OV6620 system clock is 17.73 MHz, T_{clk} = 112 ns for 16-bit output; T_{clk} = 56 ns for 8-bit output. T_{su} is HREF set-up time, maximum is 15 ns; T_{hd} is HREF hold time, maximum is 15 ns.

Figure 3. Pixel Data Bus (YUV Output)



Pixel Data 16-bit Timing
PCLK rising edge latches data bus



Pixel Data 8-bit Timing
PCLK rising edge latches data bus

Note: T_{clk} is pixel clock period. When the OV6620 system clock is 17.73MHz, T_{clk} = 112ns for 16-bit output; T_{clk} = 56ns for 8-bit output. T_{su} is HREF set-up time, maximum is 15 ns; T_{hd} is HREF hold time, maximum is 15 ns.

Figure 4. Pixel Data Bus (RGB Output)

MSB/LSB swap means: Default Y/UV channel output port relationship is:

Table 5. Default Output Sequence

	MSB							LSB
Output Port	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
Internal Output data	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

If the device is programmed for data swap, the sequence is changed to:

Table 6. Swapped MSB/LSB Output Sequence

	MSB							LSB
Output Port	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
Internal Output data	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7

Table 7. QCIF Digital Output Format (YUV, beginning of line)

Pixel #	1	2	3	4	5	6	7	8
Y	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
UV	U0,V0	U1,V1	U2,V2	U3,V3	U4,V4	U5,V5	U6,V6	U7,V7

Y channel output Y2 Y3 Y6 Y7 Y10 Y11 ...
- UV channel output U2 V3 U6 V7 U10 V11 ...
- Every line output data number is half(176 pixels) and only one half line data (every other line, total 144 line) in one frame will be output.

Table 8. QCIF Digital Output Format (RGB Raw Data, Beginning of Line)

Pixel #	1	2	3	4	5	6	7	8
Line 1	B0	G1	B2	G3	B4	G5	B6	G7
Line 2	G0	R1	G2	R3	G4	R5	G6	R7

1. Default RGB two line output mode:
   - Y channel output G0 R1 G4 R5 G8 R9 ...
   - UV channel output B0 G1 B4 G5 B8 G9 ...
   - Every line output half data(176 pixels) and all line(144 line) data in one frame will be output.

2. YG two line output mode:
   - Y channel output G0 R1 G4 R5 G8 R9 ...
   - UV channel output B0 G1 B4 G5 B8 G9 ...
   - Every line outputs half data (176 pixels) and all line (144 line) data in one frame will be output.

3. QCIF Resolution Digital Output Format
   - Y channel output Y2 Y3 Y6 Y7 Y10 Y11 ...
   - UV channel output U2 V3 U6 V7 U10 V11 ...
   - Every line output data number is half (176 pixels) and only one half line data (every other line, total 144 line) in one frame will be output.

Table 9. RGB Raw Data Format

R\C	1	2	3	4	.	353	354	355	356
1	B_{11}	G_{12}	B_{13}	G_{14}		B	G	B	G
2	G_{21}	R_{22}	G_{23}	R_{24}		G	R	G	R
3	B_{31}	G_{32}	B_{33}	G_{34}		B	G	B	G
4	G_{41}	R_{42}	G_{43}	R_{44}		G	R	G	R
5	B_{51}	G_{52}	B_{53}	G_{54}		B	G	B	G
.									
289	B	G	B	G		B	G	B	G
290	G	R	G	R		G	R	G	R
291	B	G	B	G		B	G	B	G
292	G	R	G	R		G	R	G	R

Notes:

A. Y port output data sequence: G R G R G R ... or G G G G ... ; UV port output data sequence: B G B G B G ... or B R B R ... ;
Array Color Filter Patter is Bayer-Pattern

B. Output Modes

16-bit Format (HREF total 292)

Default mode:

– 1st HREF Y channel output unstable data, UV output B_{11} G_{12} B_{13} G_{14} ....
– 2nd HREF Y channel output G_{21} R_{22} G_{23} R_{24} ..., UV output B_{11} G_{12} B_{13} G_{14} ...
– 3rd HREF Y channel output G_{21} R_{22} G_{23} R_{24} ..., UV output B_{31} G_{23} B_{33} G_{34} ....
– Every line of data is output twice.

YG mode:

– 1st HREF Y and UV output unstable data.
– 2nd HREF Y channel output G_{21} G_{12} G_{23} G_{14} ..., UV output B_{11} R_{22} B_{13} R_{24} ...
– 3rd HREF Y is G_{21} G_{32} G_{23} G_{34} ..., UV channel is B_{31} R_{22} B_{33} R_{24} ...
– Every line data output twice.

One line mode:

– 1st HREF Y channel output B_{11} G_{12} B_{13} G_{14} ...,
– 2nd HREF Y channel output G_{21} R_{22} G_{23} R_{24} ...,
– UV channel tri-state.

8-bit Format (HREF total 292)

– 1st HREF Y channel output unstable data.
– 2nd HREF Y channel output B11 G21 R22 G12 ...
– 3rd HREF Y channel output B31 G21 R22 G32 ..., etc.
– PCLK timing is double and PCLK rising edge latch data bus. UV channel tri-state. Every line data output twice.

4-bit Nibble Mode Output Format

– Uses higher 4 bits of Y port (Y[7:4]) as output port.
– Supports YCrCb/RGB data, CCIR601/CCIR656 timing, Color/B&W.
– Output sequence: High order 4 bits followed by lower order 4 bits
Y0h Y0l Y1h Y1l ...
U0h U0l V0h V0l ...

For B/W or one-line RGB raw data, the output data clock speed is doubled. For color YUV, output clock is four times that of the 16-bit output data. In color mode, sensor must be set to 8-bit mode, and the nibble timing, clock divided by 2.

– Output sequence: U0h U0l Y0h Y0l V0h V0l Y1h Y1l ...

1.2.7 Slave Mode Operation

The OV6620/OV6120 sensors can be programmable to operate in slave mode configuration (COMI[6] = 1, default is master mode). HSYNC and VSYNC output signals are provided.

When used as a slave device, the external master must provide the OV6620/OV6120 imager with the following:

1. System clock CLK to XCLK1 pin;
2. Horizontal sync, Hsync, to CHSYNC pin, positive assertion;
3. Vertical frame sync, Vsync, to VSYNC pin, positive assertion

When in slave mode, the OV6620/OV6120 tri-states CHSYNC (pin 42) and VSYNC (pin 16) output pins, which may then be used as input pins. To synchronize multiple devices, the OV6620/OV6120 image sensors use external system clock, CLK, to synchronize external horizontal sync, HSYNC, which is then used to synchronize external vertical frame sync, Vsync. See Figure 5, Slave Mode External Sync Timing for timing considerations.

1.2.8 Frame Exposure Mode

The OV6620/OV6120 sensors support frame exposure mode when programmed for Progressive Scan. FREX (pin 4) is asserted by an external master device to set exposure time. When FREX = 1, the OV6620/OV6120 pixel array will be quickly precharged. Based on the external master's assertion of FREX, the OV6620/OV6120 devices capture the image. When the master de-asserts FREX (FREX = 0), the video output data stream is delivered to the OV6620/OV6120 output port in a line-by-line manner.

It should be noted that FREX must active long enough to ensure the complete image array has been precharged.

When data is being output from the OV6620/OV6120 image sensor, care must be taken so as not to expose the image array to light. This may affect the integrity of the image data captured. A mechanical shutter synchronized with the frame exposure rate can be used to minimize this situation. Frame exposure mode timing is shown in Section Figure 6. Frame Exposure Timing below.

1.2.9 Reset

The OV6620/OV6120 image sensor includes a RESET pin (pin 2) which forces a complete hardware reset when pulled high (Vcc). When a hardware reset occurs, the OV6620/OV6120 sensor clears all registers or sets them to their default values. Reset may also be initiated through the I^2C interface.

1.2.10 Power Down Mode

Two methods are available for placing the OV6620/OV6120 devices into power-down mode: hardware power down and I^2C/software power down.

To initiate hardware power down the PWDN pin (pin 9) must be tied to high (+5VDC). When this occurs, the OV6620/OV6120 internal device clock is halted and all internal registers (except I^2C registers) are reset. In this mode, current draw is less than 10uA.

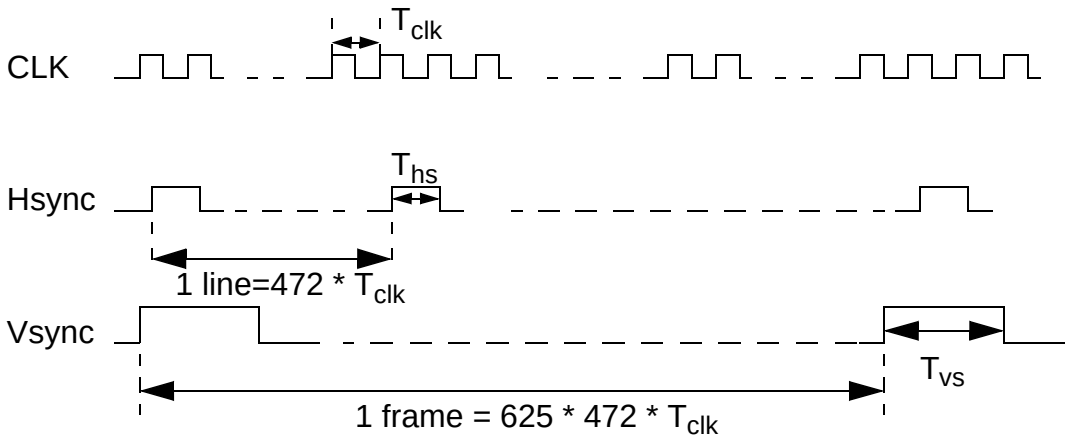
Executing a software power down through the I^2C interface suspends internal circuit activity, but does halt the device clock. In this mode, current requirements drop to less than 1mA.

1.2.11 Configuring the OV6620/OV6120 Image Sensors

Two methods are provided for configuring the OV6620/OV6120 IC for specific application requirements.

At power up, the OV6620/OV6120 sensor reads the status of certain pins to determine what, if any, power up default settings are requested. Once the reading of the external pins is completed, the device configures its internal registers according to the specified pins. Not all device functions are available for configuration through external pin.

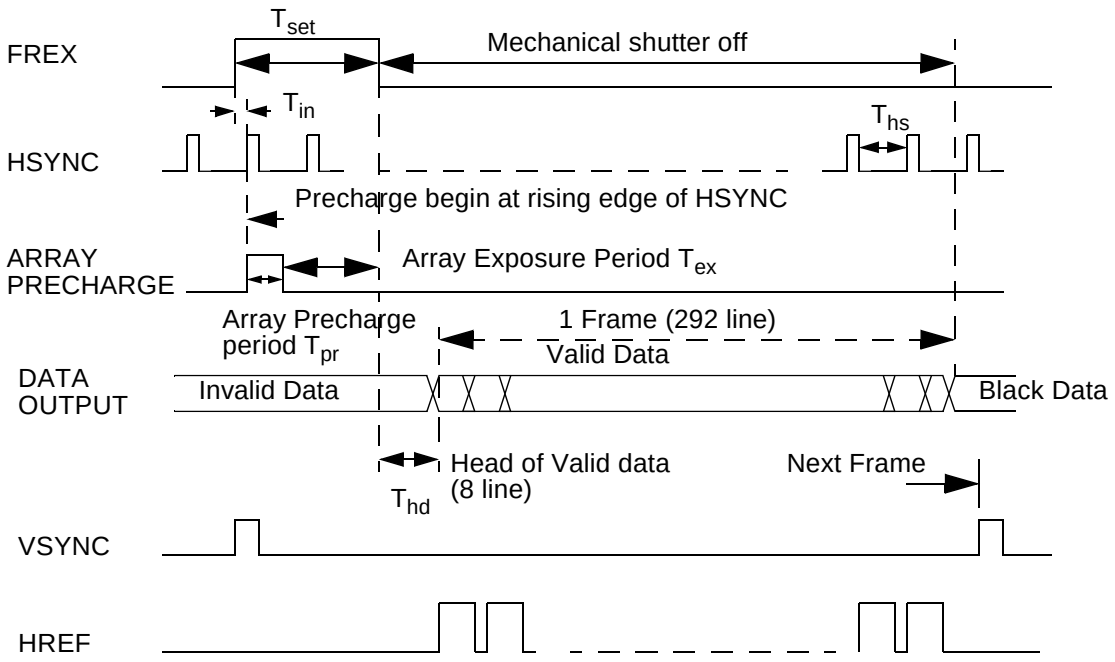
A more flexible and comprehensive method for configuring the OV6620/OV6120 IC is to use its on-chip I^2C register programming capability. The I^2C interface provides access to all of the device's programmable internal registers. See Section 3.1 I^2C Bus Protocol Format for further details about using the I^2C interface on the OV6620/OV6120 camera device.



Notes:

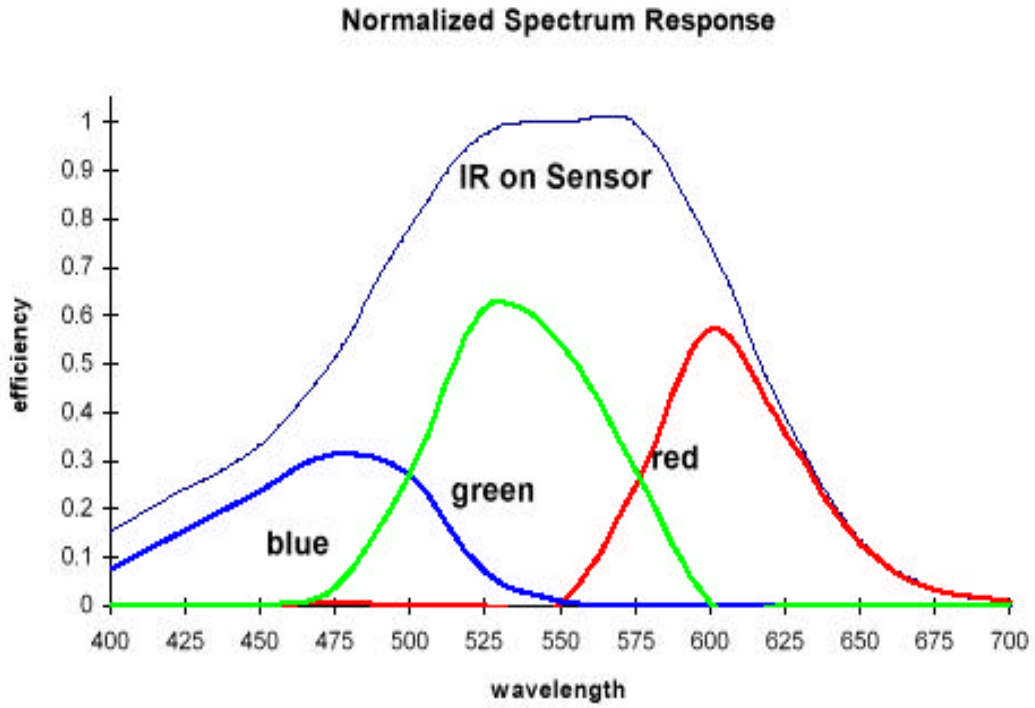
1. $T_{hs} > 6 * T_{clk}$ (2) $T_{hs} < T_{vs} < 472 * T_{clk}$
2. Hsync period is 472 * CLK
3. Vsync period is 625 * 472 * CLK
4. OV6620 will be stable after 1 frame. (2nd Vsync).

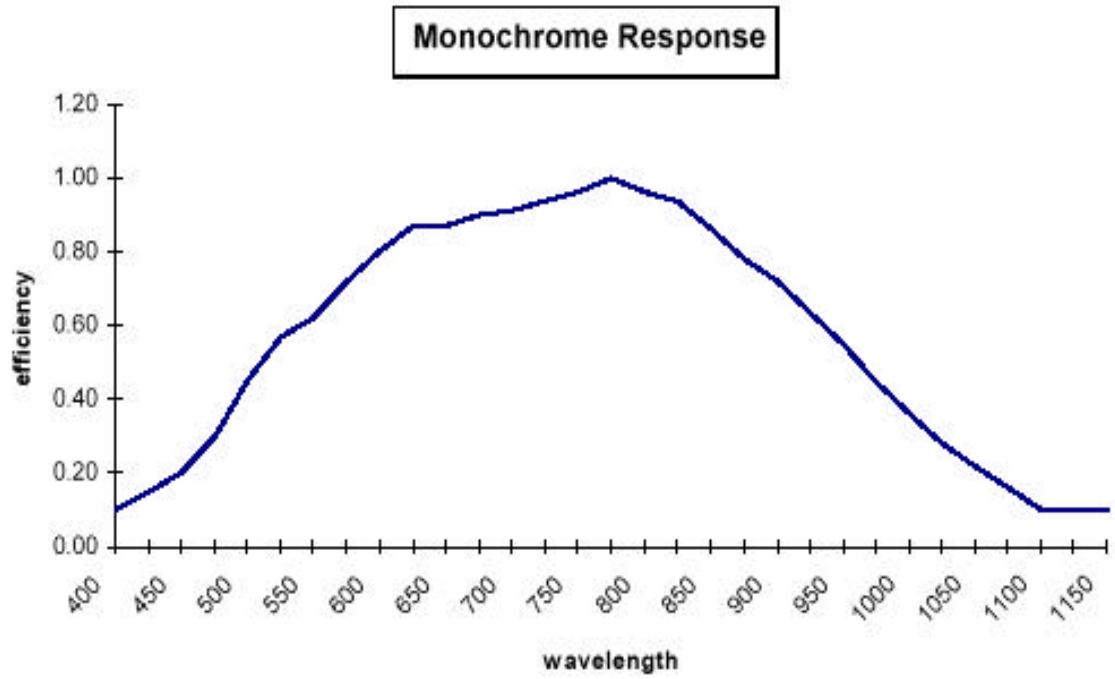
Figure 5. Slave Mode External Sync Timing



Note: T_{pr} = 292* 4 * T_{clk}, T_{clk} is internal pixel period. For default 17.73 MHz, T_{clk}=112 us. If CLK[5:0] set to divided number, T_{clk} will increase accordingly.
T_{ex} is array exposure time which is decided by external master device.
T_{in} is uncertain time due to using **HSYNC** rising edge synchronize **FREX**, $T_{in} < T_{hs}$
After **FREX**=0, there are 8 line data output before valid data output. $T_{hd} = 4 * T_{hs}$. Valid data is output when **HRE**
$T_{set} = T_{in} + T_{pr} + T_{ex}$. $T_{set} > T_{pr} + T_{in}$. Because T_{in} is uncertain, so exposure time setting resolution is T_{hs} (one line).

Figure 6. Frame Exposure Timing


Normalized Spectrum Response
efficiency
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
IR on Sensor
green
red
blue
400
425
450
475
500
525
550
575
600
625
650
675
700
wavelength


Monochrome Response
efficiency
1.20
1.00
0.80
0.60
0.40
0.20
0.00
400
450
500
550
600
650
700
750
800
850
900
950
1000
1050
1100
1150
wavelength

2. Electrical Characteristics

Table 10. DC Characteristics ($0^oC \leq TA \leq 85^oC$, Voltages referenced to GND)

Symbol	Descriptions	Max	Typ	Min	Units
Supply					
V_{DD1}	Supply voltage- internal analog (**DEVDD,ADVDD,AVDD,SVDD,AOVDD,DVDD**)	5.25	5.0	4.75	V
V_{DD2}	Supply voltage - internal digital &output digital (**DOVDD**)	5.5 3.6	5.0 3.3	4.5 3.0	V V
I_{DD1}	Supply Current (@ 50Hz frame rate & 5 volt digital I/O,25pf + 1TTL load on 16 bit data bus)	40	-	-	mA
I_{DD2}	Standby supply current	10	5	-	uA
Digital Inputs					
V_{IL}	input voltage LOW	0.8	-	-	V
V_{IH}	input voltage HIGH	-	-	2.0	V
Cin	input capacitor	10	-	-	pF
Digital Outputs - standard load 25pf, 1.2kΩ to 3.0volts					
V_{OH}	output voltage HIGH	-	-	2.4	V
V_{OL}	output voltage LOW	0.6	-	-	V
I^2C Inputs - 5k pull up + 100pf					
V_{IL}	SDA and SCL (V_{DD2}=5V)	1.5	0	-0.5	V
V_{IH}	SDA and SCL(V_{DD2}=5V)	V_{dd} + .5	5	3.0	V
V_{IL}	SDA and SCL (V_{DD2}=3V)	1	0	-0.5	V
V_{IH}	SDA and SCL(V_{DD2}=3V)	3.5	3	2.5	

Table 11. AC Characteristics (T_A=25^{o}C; Vdd=5V)

Symbol	Descriptions	Max	Typ	Min	Units
RGB/YCrCb output					
Iso	maximum sourcing current		15		mA
Vy	DC level at zero signal Y peak-peak 100% amplitude (without sync) sync amplitude		1.2 1 0.4		V V V
ADC parameters					
B	analog bandwidth				MHz
Φdiff					
DLE	DC differential linearity error		0.5		LSB
ILE	DC integral linearity error		1		LSB

Table 12. Timing Characteristics

Symbol	Descriptions	Max	Typ	Min	Units
Oscillator & Clock in					
f_{osc}	frequency (XCLK1,XCLK2)	30	17.734	10	MHz
t_r , t_f	clock input rise/fall time	5			ns
	clock input duty cycle	55	50	45	%
I^2C timing(400kbit/s)					
t_{BUF}	Bus free time between STOP & START	-	-	1.3	ms
$t_{HD:SAT}$	SCL change after START status	-	-	0.6	µs
t_{LOW}	SCL low period	-	-	1.3	µs
t_{HIGH}	SCL high period	-	-	0.6	µs
$t_{HD:DAT}$	Data hold time	-	-	0	µs
$t_{SU:DAT}$	Data set-up time	-	-	0.1	µs
$t_{SU:STP}$	Set-up time for STOP status	-	-	0.6	µs
Digital timing					
t_{pclk}	PCLK cycle time 16 bit operation 8 bit operation	-	-	 112 56	 ns ns

Table 12. Timing Characteristics

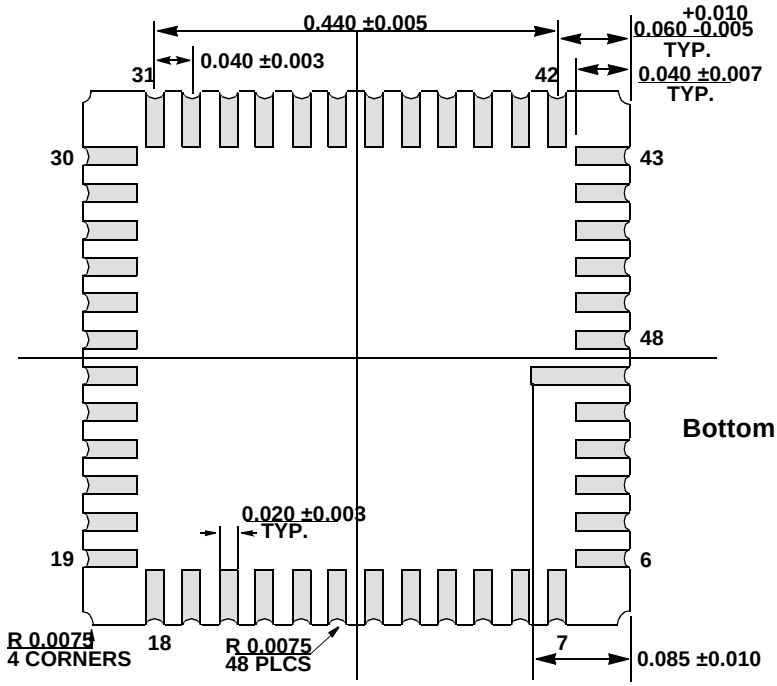
Symbol	Descriptions	Max	Typ	Min	Units
t_r, t_f	PCLK rise/fall time	15	-	-	ns
t_{pdd}	PCLK to data valid	15	-	-	ns
t_{phd}	PCLK to HREF delay	20	10	5	ns

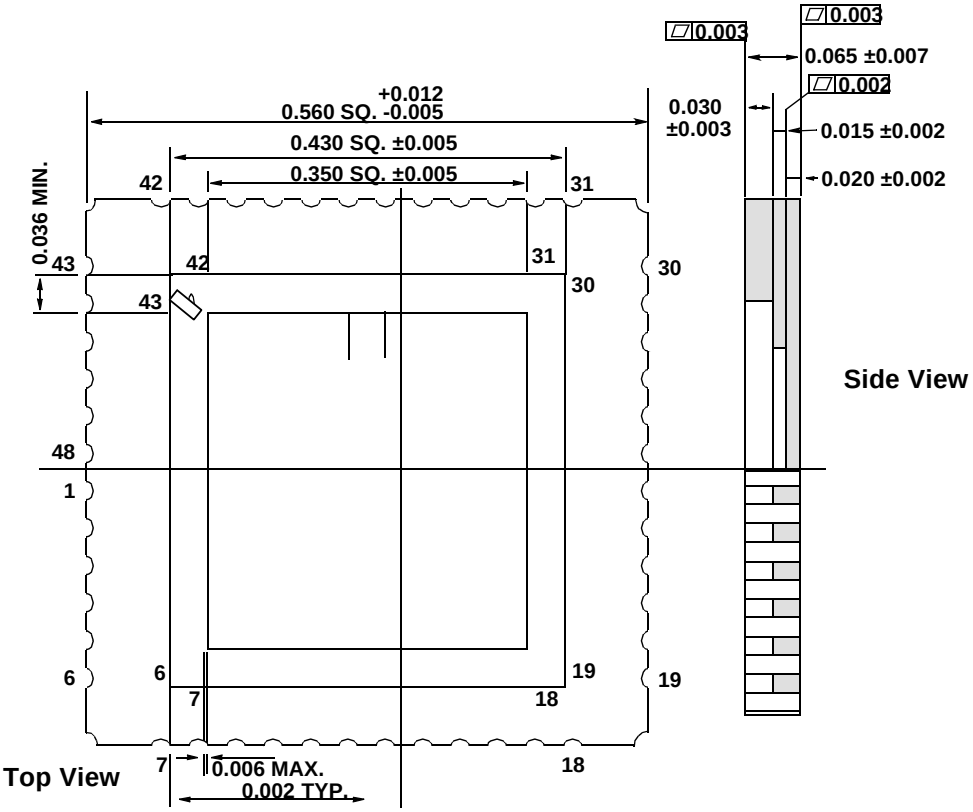
Table 13. Zoom Video Port AC Parameters

Symbol	Parameter	Min.	Max.
t1	PCLK fall timing	4 ns	8 ns
t2	PCLK low time	50 ns	
t3	PCLK rise time	4 ns	8 ns
t4	PCLK high time	50 ns	
t5	PCLK period	106 ns	
t6	Y/UV/HREF setup time	10 ns	
t7	Y/UV/HREF hold time	20 ns	
t8	VSYNC setup/hold time to HREF	1 us	

Notes:

1. In Interlaced Mode, there are Even/Odd field different (t8). When In Progressive Scan Mode, only frame timing same as Even field(t8).
2. After VSYNC falling edge, OV6620 will output black reference level, the line number is T_{vs}, which is the line number between the 1st HREF rising edge after VSYNC falling edge and 1st valid data CHSYNC rising edge. Then valid data, then black reference, line number is T_{ve}, which is the line number between last valid data CHSYNC rising edge and 1st CHSYNC rising edge after VSYNC rising edge. The black reference output line number is dependent on vertical window setting.
3. When in default setting, $T_{vs} = 14 * T_{line}$, which is changed with register VS[7:0]. VS[7:0] step equal to 1 line.
4. When in default setting, $T_{ve} = 4 * T_{line}$ for Odd Field, $T_{ve} = 3 * T_{line}$ for Even Field, which is changed with register VE[7:0]. VE[7:0] step equal to 1 line.


0.440 ±0.005
+0.010
0.060 -0.005
TYP.
0.040 ±0.003
0.040 ±0.007
TYP.
31
42
30
43
48
Bottom View
0.020 ±0.003
TYP.
19
6
R 0.0075
4 CORNERS
18
R 0.0075
48 PLCS
7
0.085 ±0.010


0.003
0.003
0.065 ±0.007
0.002
+0.012
0.560 SQ. -0.005
0.030
±0.003
0.015 ±0.002
0.430 SQ. ±0.005
0.350 SQ. ±0.005
0.020 ±0.002
42
31
0.036 MIN.
43
42
31
30
30
43
Side View
48
1
6
6
19
19
7
18
7
18
0.006 MAX.
0.002 TYP.
Top View

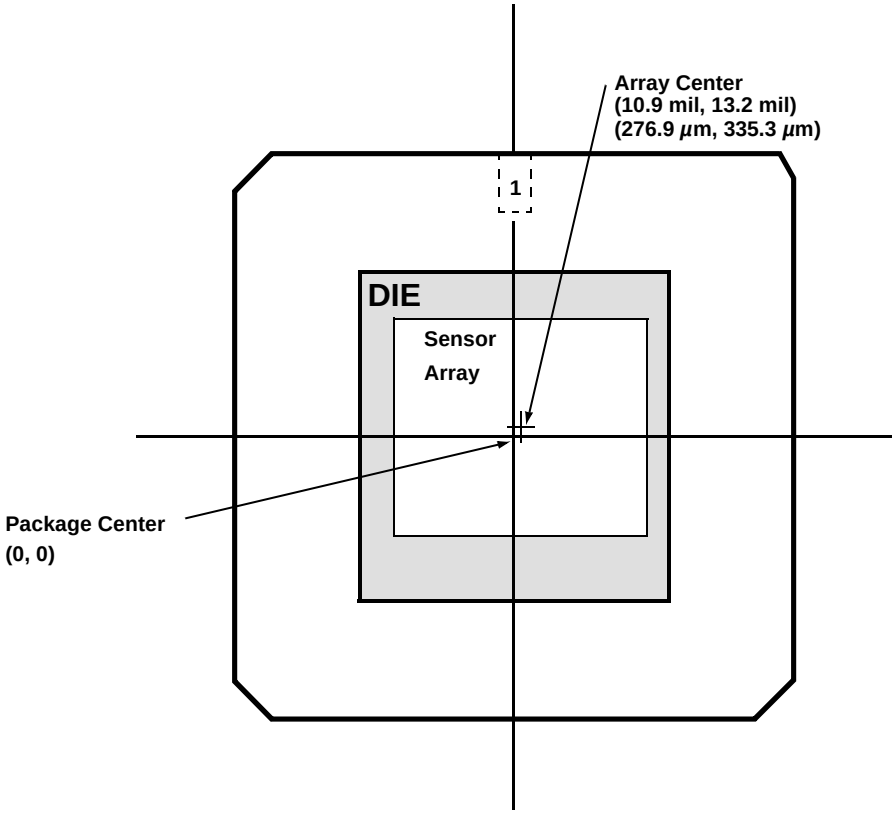


Figure 7. OV6620/OV6120 Package Outline

Table 14. Ordering Information

Part Number	Description	Package
OV6620	COLOR Image Sensor, CIF, Digital, I^2C Bus Control	48 pin LCC
OV6120	B/W Image Sensor, CIF, Digital, I^2C Bus Control	48 pin LCC

3. I²C Bus

Many of the functions and configuration registers in the OV6620/OV6120 image sensors are available through the I²C interface. The I²C port is enabled by asserting the I2CB line (pin 12) through a 10K ohm resistor to V_{DD}. When the I²C capability is enabled (I2CB = 1), the OV6620/OV6120 imager operates as a slave device that supports up to 400 kbps serial transfer rate using a 7-bit address/data transfer protocol .

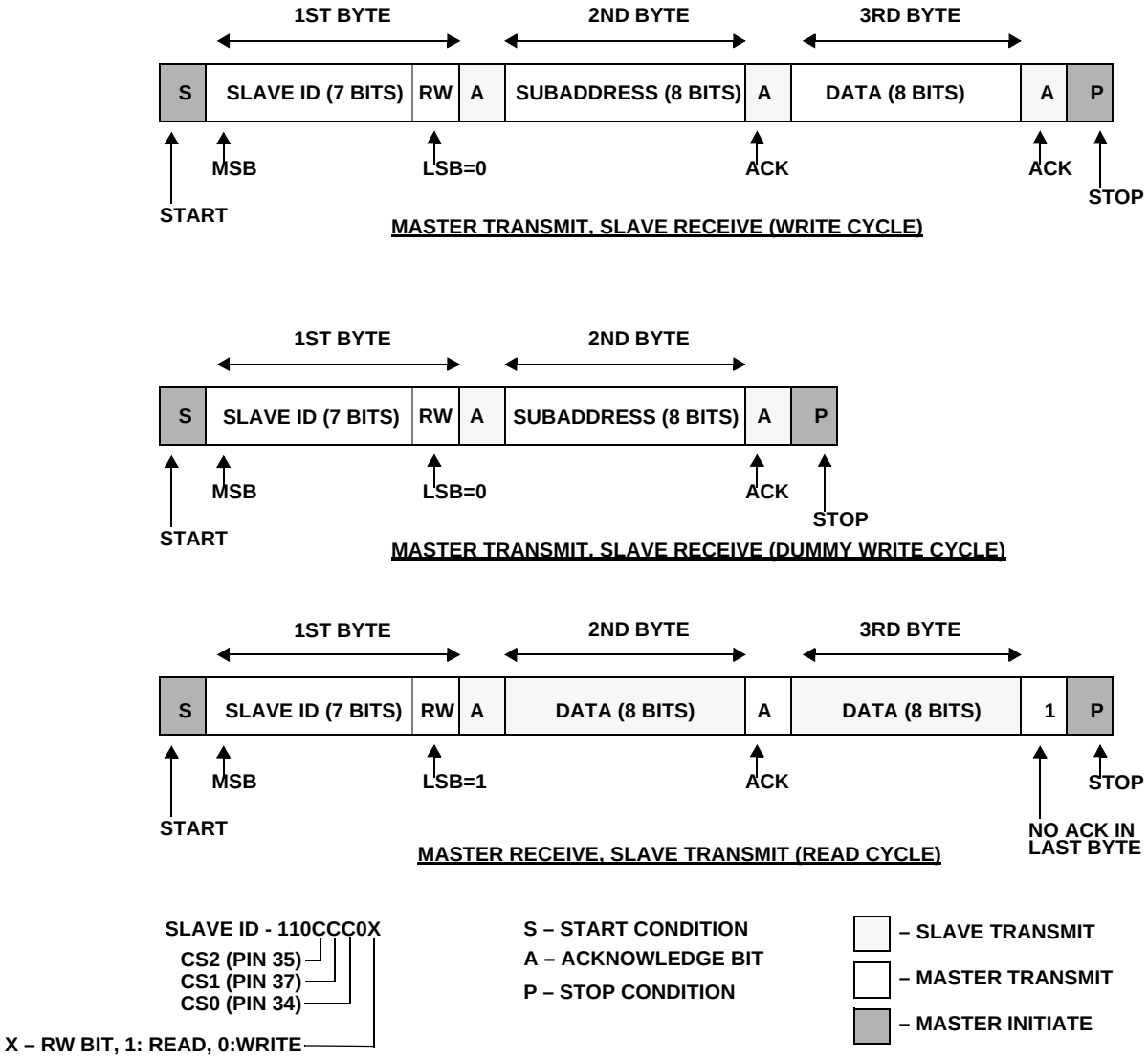


Figure 8. I²C Bus Protocol Format

3.1 I²C Bus Protocol Format

In I²C operation, the master must perform the following operations:

- **Generate the start/stop condition**
- **Provide the serial clock on SCL**
- **Place the 7-bit slave address, the RW bit, and the 8-bit subaddress on SDA**

The receiver must pull down SDA during the acknowledge bit time. During the write cycle, the OV6620/OV6120 device returns the acknowledgment and, during read cycle, the master returns the acknowledgment except when the read data is the last byte. If the read data is the last byte, the master does not perform an acknowledge, indicating to the slave that the read cycle can be terminated. Note that the restart feature is not supported here.

Within each byte, MSB is always transferred first. Read/write control bit is the LSB of the first byte.

Standard I²C communications require only two pins: SCL and SDA. SDA is configured as open drain for bi-directional purpose. A HIGH to LOW transition on the SDA while SCL is HIGH indicates a START condition. A LOW to HIGH transition on the SDA while SCL is HIGH indicates a STOP condition. Only a master can generate START/STOP conditions.

Except for these two special conditions, the protocol that SDA remain stable during the HIGH period of the clock, SCL. Each bit is allowed to change state only when SCL is LOW (See Figure 9. Bit Transfer on the I²C Bus and Figure 10. Data Transfer on the I²C Bus below).

The OV6620/OV6120 I²C supports multi-byte write and multi-byte read. The master must supply the subaddress. in the write cycle, but not in the read cycle.

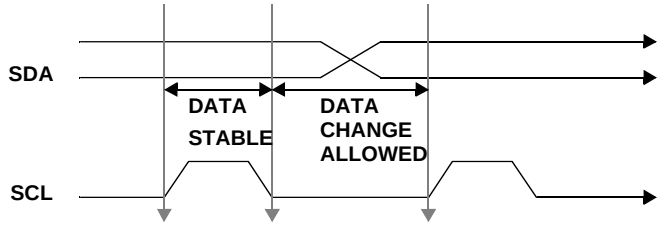


Figure 9. Bit Transfer on the I²C Bus

I

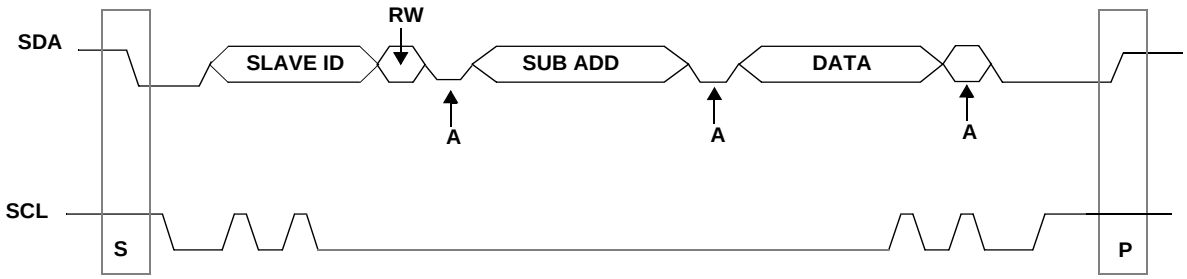


Figure 10. Data Transfer on the I²C Bus

Therefore, the OV6620/OV6120 sensor takes the read subaddress from the previous write cycle. In multi-byte write or multi-byte read cycles, the subaddress is automatically increment after the first data byte so that continuous locations can be accessed in one bus cycle. A multi-byte cycle overwrites its original subaddress; therefore, if a read cycle immediately follows a multi-byte cycle, you must insert a single byte write cycle that provides a new subaddress.

The OV6620/OV6120 imager can be programmed to one-of-eight slave ID addresses. Function pins CS[2:0] pins 35, 37, 34, respectively).

Table 15. Slave ID Addresses

CS[2:0]	000	001	010	011	100	101	110	111
WRITE ID (hex)	C0	C4	C8	CC	D0	D4	D8	DC
READ ID (hex)	C1	C5	C9	CD	D1	D5	D9	DD

The OV6620/OV6120 sensors support both single chip and multiple chip configurations. By asserting MULT (pin 47) high, the sensor can be programmed for up to 8 slave ID addresses. Asserting MULT low configures the OV6620/OV6120 imagers for single ID slave address with address C0 for writes and address C1 for reads. MULT is internally defaulted to a low condition.

In the write cycle, the second byte in I^2C bus is the subaddress for selecting the individual on-chip registers, and the third byte is the data associated with this register. Writing to unimplemented subaddress is ignored. In the read cycle, the second byte is the data associated with the previous stored subaddress. Reading of unimplemented subaddress returns unknown.

3.2 Register Set

The table below provides a list and description of available I^2C registers contained in the OV6620/OV6120 image sensor.

Table 16. I^2C Registers

Subaddress (hex)	Register	Default (hex)	Read/ Write	Descriptions
00	Gain[6:0]	00	RW	AGC Gain Control GC[7:6] - unimplemented bit, returns 'X' when read. GC[5:0] – Storage for the current AGC Gain setting. This register is updated automatically. If AGC is enabled, the internal control stores the optimal gain value in this register. IF AGC is not enabled, a "00" is stored in this register.
01	Blue[7:0]	80	RW	Blue Gain Control BLU[7] – "0" decrease gain, "1" increase gain. BLU[6:0] – blue channel gain balance value.
02	Red[7:0]	80	RW	Red Gain Control RED[7] – "0" decrease gain, "1" increase gain. RED[6:0] – red channel balance value.
03	Sat	80	RW	Saturation Control SAT[7:0] – saturation adjustment. "FFh"- highest, "00h"-lowest
04	Rsvd04	XX	-	reserved
05	Cnt	48	RW	Contrast Control CTR[7:0] – contrast adjustment. "FFh"-highest, "00h"-lowest
06	Brt	80	RW	Brightness Control BRT[7:0] – brightness adjustment. "FFh"-highest,"00h"-lowest
07	Sharpness	C6	RW	Sharpness Control SHP[7:4] – Threshold of sharpness. Range: 0~80mV, Step: 5 mV SHP[3:0] – Sharpness control. Range: 0 ~ 8x, Step: 0.5x
08	Rsvd08	XX	-	reserved
09	Rsvd09	XX	-	reserved
0A	Rsvd0A	XX	-	reserved
0B	Rsvd0B	XX	-	reserved

Subad-dress (hex)	Register	Default (hex)	Read/ Write	Descriptions
0C	AWB - Blue	20	R/W	White Balance Background: Blue Channel ABLU[7:6] - rsvd ABLU[5] - Sign bit. "0" - decrease background blue component "1" - increase background blue component ABLU[4:0] - White balance blue ratio adjustment
0D	AWB - Red	20	R/W	White Balance Background: Red Channel ARED[7:6] - rsvd ARED[5] - Sign bit. "0" - decrease background red component "1" - increase background red component ABLU[4:0] - White balance red ratio adjustment
0E	COMR	0D	RW	Common Control R COMR[7] - Analog signal 2x gain control bit. "1" - Additional 2x gain, "0" - normal COMR[6:0] - Reserved
0F	COMS	05	RW	Common Control S COMS[7:6] - Reserved COMS[5:4] - Black expanding level "00" - 1.2V, "01" - 1.26V, "10" - 1.3V, "11" - 1.4V COMS[3:2] - Set high threshold level "00" - 1.9V, "01" - 2.0V, "10" - 2.1V, "11" - 2.2V COMS[1:0] - Set low threshold level "00" - 1.3V, "01" - 1.45V, "10" - 1.5V, "11" - 1.6V
10	AEC	9A	R	Automatic Exposure Control AEC[7:0] - Set exposure time Interlaced: $T_{ex} = T_{line}$ x AEC[7:0] Progressive: $T_{ex} = T_{line}$ x AEC[7:0] x 2
11	CLKRC	00	R	Clock Rate Control CLKRC[7:5] – Sync output polarity selection "00" - HSYNC=Neg, CHSYNC=Neg, VSYNC=Pos "01" - HSYNC=Neg, CHSYNC=Neg, VSYNC=Neg "10" - HSYNC=Pos, CHSYNC=Neg, VSYNC=Pos "11" - HSYNC=Pos, CHSYNC=Pos, VSYNC=Pos CLKRC[5:0] – Clock prescaler CLK = (CLK_main / ((CLKRC[5:0] + 1) x 2)) / 2
12	COMA	24	RW	Common Control A COMA[7] - SRST, "1" initiates soft reset. Initiate soft reset. All registers are set to default values and chip is reset to known state and resumes normal operation. This bit is automatically cleared after reset. COMA[6] - MIRR, "1" selects mirror image COMA[5] - VSFR, "1" enables AGC, COMA[4] - Digital output format, "1" selects 8-bit: Y U Y V Y U Y V COMA[3] - Select video data output: "1" - select RGB, "0" - select YCrCb COMA[2] - Auto White Balance "1" - Enable AWB, "0" - Disable AWB COMA[1] - Color Bar Test Pattern: "1" - Enable color bar test pattern COMA[0] - reserved
13	COMB	01	RW	Common Control B COMB[7] - reserved COMB[6] - reserved COMB[5] - Select data format. "1" - Select 8-bit format, Y/CrCb and RGB is multiplexed to 8-bit Y bus, UV bus is tri-stated, "0" - Select 16-bit format COMB[4] - "1" - enable digital output in CCIR656 format COMB[3] - CHSYNC output: "1" - Horizontal sync, "0" - composite sync COMB[2] - "1" - Tristate Y and UV busses. "0" - enable both busses COMB[1] - "1" - Initiate single frame transfer COMB[0] - "1" - Enable auto adjust mode
14	COMC	00	RW	Common Control C COMC[7] - reserved COMC[6] - reserved COMC[5] - QCIF digital output format selection. 1 - 176x144; 0 - 352x288. COMC[4] - Field/Frame vertical sync output in VSYNC port selection: 1 - frame sync, only ODD field vertical sync; 0 - field vertical sync, effect in Interlaced mode COMC[3] - HREF polarity selection: 0 - HREF positive effective, 1 - HREF negative. COMC[2] - gamma selection: 1 - RGB Gamma on ; 0 - gamma is 1. COMC[1] - reserved COMC[0] - reserved

Subad-dress (hex)	Register	Default (hex)	Read/ Write	Descriptions
15	COMD	01	RW	Common Control D COMD[7] - reserved bit. COMD[6] - PCLK polarity selection. “0” OV6620 output data at PCLK falling edge and data bus will be stable at PCLK rising edge; “1” rising edge output data and stable at PCLK falling edge. When OV6620 work as CCIR656 format, COMB4=1, this bit is disable and should use PCLK rising edge latch data bus. COMD[5:1] - reserved bit. COMD[0] - U V digital output sequence exchange control. 1 - UV UV ... for 16-bit, U Y V Y ... for 8-bit; 0 - V U V U ... for 16Bit and V Y U Y ... for 8 Bit.
16	FSD	03	RW	Field Slot Division FSD[7:2] - Field interval selection. Odd Even mode defined by FD[1:0] 000000 - disable digital data output, only output black reference level. 000001 - divide to 2 slots, HREF is active one in every 2 field/frame 000010 - divide to 4 slots, HREF is active one in every 4 field/frame 000100 - divide to 8 slots, HREF is active one in every 8 field/frame 001000 - divide to 16 slots, HREF is active one in every16 field/frame 010000 - divide to 32 slots, HREF is active one in every 32 field/frame 100000 - divide to 64 slots, HREF is active one in every 64field/frame FSD[1:0]- field mode selection. Each frame consists of two fields: Odd & Even, these bits defines the assertion of HREF in relation to the two fields. 00 - OFF mode; HREF is not asserted in both fields, one exception is the single frame transfer operation (see the description for the register 13) 01 - ODD mode; HREF is asserted in odd field only. 10 - EVEN mode; HREF is asserted in even field only. 11 - FRAME mode; HREF is asserted in both odd field and even field. FD[7:2] useless.
17	HREFST	38	RW	Horizontal HREF Start HS[7:0] - selects the starting point of HREF window, each LSB represents two pixels for CIF resolution mode, one pixels for QCIF resolution mode, this value is set based on an internal column counter, the default value corresponds to 352 horizontal window. Maximum window size is 356. see window description below. HS[7:0] programmable range is [38]- [EB], and should less than HE[7:0]. HS[7:0] should be programmable to value larger than or equal to [38]. Value larger than [EC] is invalid. See window description below.
18	HREFEND	EA	RW	Horizontal HREF End HE[7:0] - selects the ending point of HREF window, each LSB represents two pixels for full resolution and one pixels for QCIF resolution, this value is set based on an internal column counter, the default value corresponds to the last available pixel. The HE[7:0] programmable range is [39] - [EC]. HE[7:0] should be larger than HS[7:0] and less than or equal to [EC]. Value larger than [EC] is invalid. See window description below.
19	VSTRT	03	RW	Vertical Line Start VS[7:0] - selects the starting row of vertical window, in full resolution mode, each LSB represents 1 scan line in one frame. see window description below. Min. is [03], max. is [93] and should less than VE[7:0].
1A	VEND	92	RW	Vertical Line End VE[7:0]- selects the ending row of vertical window, in full resolution mode, each LSB represents 1 scan line in one frame, see window description below. Min. is [04], max. is [94] and should larger than VS[7:0].
1B	PSHFT	00	RW	Pixel Shift PS[7:0] - to provide a way to fine tune the output timing of the pixel data relative to that of HREF, it physically shifts the video data output time late in unit of pixel clock as shown in the figure below. This function is different from changing the size of the window as is defined by HS[7:0] & HE[7:0] in register 17&18. Higher than default number shifts the pixel in delay(right) direction, the highest number is “FF”. so maximum shift number is: Late: 256 pixels.
1C	MIDH	7F	R	Manufacture ID Byte: High MIDH[7:0] - read only, always returns “7F” as manufacturer’s ID no.
1D	MIDL	A2	R	Manufacture ID Byte: Low MIDL[7:0] - read only, always returns “A2” as manufacturer’s ID no.
1E	Rsvrd1E	C4	R	reserved
1F	Rsvrd1F	04	R	reserved

Subad-dress (hex)	Register	Default (hex)	Read/ Write	Descriptions
20	COME	00	RW	Common Control E COME[7] - HREF pixel number selection. "1" - HREF include 704 PCLK, every data output twice. COME[6] - reserved. COME[5] - "1" First stage aperture correction enable. Correction strength will be decided by register [07]. "0" disable first stage aperture correction. COME[4] - "1" Second stage aperture correction enable. Correction strength and threshold value will be decided by COMF[7] ~ COMF[4]. COME[3] - AWB smart mode enable. 1 - Drop out pixel when compare pixel red, blue and green component level to change register [01] and [02], which luminance level is higher than presetting level and lower than presetting level, this two level is set by register [0F]. 0 - calculate all pixels to get AWB result. Valid only when COMB[0]=1 and COMA[2]=1 COME[2] - AWB stop when field/frame image average luminance level is lower than a presetting level enable. 1 - enable stop AWB when image luminance level is low. 0 - AWB is independent with field/frame luminance level. Valid only when COMB0=1 and COMA[2]=1. Average compare level is set by GAM[7:5]. COME[1] - AWB fast/slow mode selection. "1" - AWB is always fast mode, that is register [01] and [02] is changed every field/frame. "0" AWB is slow mode, [01] and [02] change every 16/64 field/frame decided by COMK[1]. When AWB enable, COMA[2]=1, AWB is working as fast mode at first 1024 field/frame, than as slow mode later. COME[0] - Digital output driver capability increase selection: "1" Double digital output driver current; "0" low output driver current status.
21	YOFF	80	RW	Y Channel Offset Adjustment YOFF[7] - Offset adjustment direction 0 - Add Y[6:0]; 1 -Substrate Y[6:0]. YOFF[6:0] -Y channel digital output offset adjustment. Range: +127mV ~ -127mV. If COMG[2]=0, this register will be updated by internal auto A/D BLC circuit, and write a value to this register with I^2C has no effect. If COMG[2]=1, Y channel offset adjustment will use the register stored value which can be changed by I^2C. If COMF[1]=0, this register has no adjustment effect to A/D output data. If output RGB raw data, this register will adjust R/G/B data.
22	UOFF	80	RW	U Channel Offset Adjustment UOFF[7]: - Offset adjustment direction: 0 - Add U[6:0]; 1 -Substrate U[6:0]. UOFF[6:0] - U channel digital output offset adjustment. Range: +128mV ~ -128mV. If COMG[2]=0, this register will be updated by internal auto A/D BLC circuit, and write a value to this register with I^2C has no effect. If COMG[2]=1, U channel offset adjustment will use the register stored value which can be changed by I^2C. If COMF[1]=1, this register has no effect to A/D output data. If output RGB raw data, this register will adjust R/G/B data.
23	REFC	04	RW	Reference Control REFC[7:6] - Select different crystal circuit power level (11 = minimum). REFC[5:4] - reserved REFC[3:0]: Reference Voltage range selection. 2.5V - 3.5V and step is 0.0625V.
24	AEW	33	RW	Automatic Exposure Control: Bright Pixel Ratio Adjustment AEW[7:0] - Used as calculate bright pixel ratio. OV6620 AEC algorithm is count whole field/frame bright pixel (its luminance level is higher than a fixed level) and black pixel (its luminance level is lower than a fixed level) number. When bright/black pixel ratio is same as the ratio defined by register [25] and [26], image stable. This register is used to define bright pixel ratio, default is 25%, each LSB represent step: 1.3% Change range is: [01] ~ [CA]; Increase AEW[7:0] will increase bright pixel ratio. For same light condition, the image brightness will increase if AEW[7:0] increase. **Note: AEW[7:0] must combine with register [26] AEB[7:0]. The relation must be as follows: AEW[7:0] + AEB[7:0] > [CA].**
25	AEB	97	RW	Automatic Exposure Control: Black Pixel Ration Adjustment AEB[7:0] - used as calculate black pixel ratio. OV6620 AEC algorithm is count whole field/frame bright pixel (its luminance level is higher than a fixed level) and black pixel (its luminance level is lower than a fixed level) number. When bright/black pixel ratio is same as the ratio defined by register [25] and [26], image stable. This register is used to define black pixel ratio, default is 75%, each LSB represent step: 1.3%; Change range is: [01] ~ [CA]; Increase AEB[7:0] will increase black pixel ratio. For same light condition, the image brightness will decrease if AEB[7:0] increase. **Note: AEB[7:0] must e combined with register [25] AEW[7:0]. The relation must be as follows: EW[7:0] + AEB[7:0] > [CA].**

Subad-dress (hex)	Register	Default (hex)	Read/ Write	Descriptions
26	COMF	B0	RW	Common Control F COMF[7:6] - Second aperture correction threshold selection. [00] - Difference of neighbor pixel luminance is larger than 8 mV, correction on. [01] - 16 mV. [10] - 32 mV. [11] - 64 mV. COMF[5:4] - Second aperture correction strength selection. [00] and [01] - Strength is 50% of difference of neighbor pixel luminance. [10] - 100%. [11] - 200%. COMF[3] - UV BLC swap. “1” swap; “0” no swap. COMF[2] - Digital data MSB/LSB swap. “1” LSB->Bit7, MSB->Bit0; “0” normal. COMF[1] - “1” A/D Black level calibration enable. “0” Disable A/D BLC. COMF[0] - “1” Output first 4 line black level before valid data output. HREF number will increase 4 relatively. “0” no black level output.
27	COMG	A0	RW	Common Control G COMG[7] - reserved COMG[6] - reserved. COMG[5] - Select CKOUT pin output V flag. 1 - CKOUT output V flag signal. CKOUT=1, means related UV channel output V component (or Red component), CKOUT=0 pointed to U component (or Blue component). 0 - CKOUT output buffered XCLK2 COMG[4] - reserved. COMG[3] - reserved COMG[2] - “1” A/D offset adjustment manually mode enable: 1 - A/D data will be add/substrate a value defined by register [21] and [22], which content is written by I^2C. 0 - A/D data will be added/substrate a value defined by register [21] and [22], which is updated by internal circuit. COMG[1] - Digital output full range selection. OV6620 output data value range is [10] - [F0], if COMG[1] -1, range change to [01] - [FE] with signal overshoot and undershoot level. COMG[0] - reserved.
28	COMH	01	RW	Common Control H COMH[7]: - “1” selects One-Line RGB raw data output format, “0” selects normal two-line RGB raw data output, effective only in Progressive Scan mode. COMH[6]: - “1” enable Black/White mode. When OV6620 working as BW camera, its vertical resolution will be higher than color mode. At this mode, can’t set OV6620 working at 8 bit output mode. OV6620 output data YUV/RGB from Y port. UV port will be tri-state. COMB[5] and COMB[4] will be set to “0”. “0” normal color mode. COMH[5]: - reserved. COMH[4]: - Freeze AEC/AGC value, effective only when COMB0=1. “1” - register [00] and [10] will not be updated and hold latest value. “0” - AEC/AGC normal working status. COMH[3]: - AGC disable. 1 - when COMB[0]=1 and COMA[5]=1, internal circuit will not update register [00], register [00] will kept latest updated value before COMH[3]=1. 0 - when COMB0=1 and COMA[5]=1, register [00] will be updated by internal algorithm. COMH[2]: - RGB raw data output YG format: 1 - Y channel G, UV channel B R; 0 - Y channel: G R G R..., UV channel B G B G.... COMH[1]: - Gain control bit. “1” Double PreAmp gain to 12dB. “0” PreAmp gain is 6dB. COMH[0]: - High gain mode. “1” - AGC maximum gain is 24dB. AGC step is 1/8. “0” AGE maximum gain is 18dB, AGC step is 1/16. Only effective when COMB[0]=1, COMA[5]=1 and COMH[3]=0.
29	COMI	00	RW	Common Control I COMI[7]: - AEC disable. “1” If COMB[0]=1, AEC stop and register [10] value will be held at last AEC value and not be updated by internal circuit. “0” - if COMB[0]=1, register [10] value will be updated by internal circuit COMI[6]: - Slave mode selection. “1” slave mode, use external Sync and Vsync; “0” master mode COMI[5]: - reserved COMI[4]: - reserved COMI[3]: - Central 1/4 image area rather whole image used to calculate AEC/AGC. “0” use whole image area to calculate AEC/AGC. COMI[2]: - reserved COMI[1:0] - Version flag. For Version A, value is [00], these two bits can only be read.
2A	FRARH	84	RW	Frame Rate Adjust High FRARH[7] - Frame Rate adjustment enable bit. “1” Enable. FRARH[6] - reserved FRARH[5] - Highest 1bit of frame rate adjust control byte. see explanation below. FRARH[4] - reserved FRARH[3] - Y channel brightness adjustment enable. When COMF[2]=1 active. FRARH[2] - reserved FRARH[1] - “1” When in Frame exposure mode, only One frame data output. FRARH[0] - reserved

Subad-dress (hex)	Register	Default (hex)	Read/ Write	Descriptions
2B	FRARL	5E	RW	Frame Rate Adjust Low FRARL[7:0] - Lowest 8 bit of frame rate adjust control byte. Frame rate adjustment resolution is 0.21%. Control byte is 10 bit. Every LSB equal decrease frame rate 0.21%. Range is 0.21% - 109%. IF frame rate adjustment enable, COME7 must set to “0”.
2C	Rsvd2C	88	RW	reserved
2D	COMJ	03	RW	Common Control J COMJ[7:5] - reserved COMJ[4] - Enable auto black expanding mode. COMJ[3] - “1” = White Balance update when AGC/AEC stable. “0” = White Balance register update independent with AEC/AGC. COMJ[2] - Band filter enable. After adjust frame rate to match indoor light frequency, this bit enable a different exposure algorithm to cut light band induced by fluorescent light. COMJ[1] - reserved COMJ[0] - A/D U and V BLC separate mode. “1” = U and V offset cancelled by different register. “0” = U V offset cancelled by one common register [2E].
2E	VCOFF	80	RW	V Channel Offset Adjustment VCOFF[7]: Offset adjustment direction: “0” = Add V[6:0]; “1” = Substrate V[6:0]. VCOFF[6:0] - V channel digital output offset adjustment. Range: +128mV ~ -128mV. If COMG[2]=0, this register will be updated by internal auto A/D BLC circuit, and write a value to this register with I^2C has no effect. If COMG[2] =1, V channel offset adjustment will use the register stored value which can be changed by I^2C. If COMF[1] =1, this register has no effect to A/D output data. If output RGB raw data, this register will adjust R/G/B data.
2F-32	Rsvd2F-Rsvd32	xx	-	Reserved
33	CPP	00	RW	Color Processing Parameter Control CPP[7:6] - reserved CPP[5] - Luminance gamma on/off. “1” - luminance gamma on; “0” - luminance gamma is 1. CPP[4:0] - reserved
34	BIAS	A2	RW	Bias Adjustment BIAS[7:6] - A/D reference level adjustment. [00] - 110% internal full signal range; [01] - 120%, [10] - 130%, [11] - 140%. BIAS[5:0] - reserved
35	Rsvd35	80	RW	reserved
36	Rsvd36	48	RW	reserved
37	Rsvd37	41	RW	reserved
38	COMK	81	RW	Common Control K COMK[7] - HREF edge latched by PCLK falling edge (When COMD[6] = 0). “0” HREF edge is 10 ns after PCLK rising edge. COMK[6] - Output port drive current additional 2x control bit. COMK[5] - reserved. COMK[4] - ZV port Vertical timing selection. “1” VSYNC output ZV port vertical sync signal. “0” = normal TV vertical sync signal. COMK[3] - Quick stable mode when camera mode change. After relative control bit set, the first VS will be the stable image with suitable AEC/AWB setting. “0” - slow mode, after mode change need more field/frame to get stable AEC/AWB setting image. COMK[2] - reserved COMK[1] - AWB stable time selection when in slow mode. “1” - 4 times less time needed to get stable AWB setting when in slow AWB mode. COMK[0] -reserved.
39	COML	00	RW	Common Control L COML[7] - reserved COML[6] - PCLK output timing selection. 1 -- PCLK valid only when HREF is high; 0 -- PCLK is free running. COML[5] - Vertical sync selection, 1 -- Same period between 1st HREF and VS falling edge in two field; 0 - Different timing period between 1st HREF and VS falling edge COML[4] - “1” select CHSYNC output from HREF port. “0” normal COML[3] - “1” select HREF output from CHSYNC port. “0” normal COML[2] - Tristate all control signal output (FODD, CHSYNC, HREF, PCLK) COML[1] - Highest 1 bit of horizontal sync starting position, combined with register [3A] COML[0] - Highest 1 bit of horizontal sync ending position, combined with register [3B]
3A	HSST	0F	RW	Horizontal Sync Start Position HSST[7:0] - lower 8 bit of horizontal sync starting position, combined with register bit of COML[1], total 9 bit control. range: [00] -- [FF]. HSEND[8:0] must less than HSST[8:0]
3B	HSEND	3C	RW	Horizontal Sync End Position HEND[7:0] - lower 8 bit of horizontal sync ending position, combined with register bit of COML[0], total 9 bit control. range: [00] -- [FF]. HSEND[8:0] must be larger than HSST[8:0]

Subad-dress (hex)	Register	Default (hex)	Read/ Write	Descriptions
3C	COMM	21	RW	Common Control M COMM[7:5] - Select minimum AEC number if Banding filter enable. [000] -- 1 field, [001] -- 1/2; [010] -- 1/4; [011] -- 1/8; [100] -- 1/16; [101]~[111] -- 1/32; COMM[4] - AEC/AGC change mode selection COMM[3] - AEC/AGC change mode selection COMM[2] - AEC/AGC change fastest mode COMM[1] - AEC/AGC change fast mode COMM[0] - AEC/AGC change slowest mode
3D	COMN	08	RW	Common Control N COMN[7] - Enable one frame drop when AEC change to keep data valid when Banding filter mode enable. COMN[6:4] - reserved COMN[3] - Enable 50 Hz PAL video timing, so VTO analog signal can be displayed on TV COMN[2:0] - reserved
3E	COMO	80	RW	Common Control O COMO[7] - Input main clock divided by 2 or 4 selection. 1 -- 2; 0 -- 4 COMO[6:5] - reserved COMO[4] - Select 4 bit nibble mode output COMO[3] - reserved COMO[2] - Enable Minimum exposure time is 4 line. Default is 1 line COMO[1] - reserved COMO[0] - reserved
3F	COMP	02	RW	Common Control P COMP[7] - reserved COMP[6] - Output main clock output from FODD port COMP[5] - reserved COMP[4] - Software whole chip power down enable, can be waked up by disable this bit COMP[3:2] - reserved COMP[1] - CCIR656 output control COMP[0] - Reset internal timing circuit without reset AEC/AGC/AWB value
40	Rsvd40 - Rsvd4C	XX	-	reserved
4D	YMXA	02	RW	YUV Matrix Control (Main) YMXA[7:5] - reserved YMXA[4:3] - YUV/YCrCB selection: [00] U = u, V = v [01] U = 0.938u, V = 0.838v [10] U = 0.563u, V = 0.714v [11] U = 0.5u, V = 0.877v YMXA[2:0] - Reserved
4E	Rsvd4E	XX	-	reserved
4F	YMXB	00	RW	YUV Matrix Control (Secondary) YMXB[7:6] - Y channel delay selection: 0 ~ 3 tp YMXB[5:4] - UV delay selection: 0 ~ 6 tp YMXB[3:2] - Select UV average mode. [00] & [10]: U0/V0 (no delay); [01] -- 3 point average; [11] -- 5 point average mode YMXB[1:0] - Color killer control: [00]:2.4v;[01]:2.6v;[10]:2.8v;[11]:3.0v
50	Rsvd50 - Rsvd53	XX	-	reserved