Bachelorthesis

Martin Stahl

Controllersystem zur Verstärkungsregelung und Offsetkompensation für ABS-Sensoren mit Diagnosefunktion

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Martin Stahl

Controllersystem zur Verstärkungsregelung und Offsetkompensation für ABS-Sensoren mit Diagnosefunktion

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter: Prof. Dr.-Ing. Hans Peter Kölzer

Abgegeben am 26. Februar 2010

Martin Stahl

Thema der Bachelorthesis

Controllersystem zur Verstärkungsregelung und Offsetkompensation für ABS-Sensoren mit Diagnosefunktion

Stichworte

ABS-Sensoren, AMR-Effekt, Mikrocontroller, Offsetkompensation, Verstärkungsregelung

Kurzzusammenfassung

In dieser Arbeit wird die Entwicklung eines Controllersystems zur Verstärkungsregelung und Offsetkompensation von ABS-Sensor-Signalen beschrieben. Es wurde eine Hardware auf Basis des TI-MSP430F1611 Mikrocontrollers entwickelt, welche ein bestehendes Controllersystem um die Funktionen Verstärkungsregelung sowie Offsetkompensation erweitert. Des Weiteren wurden zwei Radmessplätze automatisiert, einer mit aktivem, der andere mit passivem Encoderrad, die per Matlab-Skript gesteuert werden können.

Martin Stahl

Title of the paper

Controllersystem for gaincontrol and offsetcompensation of an anti-lock brake sensor with health monitoring

Keywords

ABS-sensors, AMR-effect, microcontroller, offsetcompensation, gaincontrol

Abstract

This paper is on the development of a system, that compensates the offset and controls the gain of the sensor signal of an anti-lock braking system. The designed hardware is based on a TI-MSP430F1611 microcontroller and it extends an existing hardware platform by the previously named functions. In addition two measurement setups, one with an active, the other with a passive encoderwheel were automated and controlled by a matlab-script.

Danksagung

Bedanken möchte ich mich an dieser Stelle bei Herrn Prof. Dr.-Ing. Karl-Ragmar Riemschneider, betreuender Prüfer, für seinen stets engagierten Einsatz im Rahmen dieser Arbeit. Ebenfalls bedanke ich mich bei Herrn Prof. Dr.-Ing. Hans Peter Kölzer für die Übernahme der Zweitprüfung.

Besonders bedanken möchte ich mich auch bei Herrn Dipl.-Ing. Martin Krey, wissenschaftlicher Mitarbeiter im Rahmen des Forschungsprojektes „ESZ-ABS“, für seinen fachlichen Rat und seine tatkräftige Unterstützung während meiner Arbeit an dem Projekt. Herrn Heiko Poppinga danke ich für die Unterstützung bei der Durchführung von Messungen.

Abschließend danke ich meinen Eltern Irmgard und Thomas Stahl für die erhaltene Unterstützung, die sehr zur Absolvierung meines Studiums beigetragen hat.

Inhaltsverzeichnis

1 Einführung

1.1 Einleitung

ABS-Sensoren werden heutzutage millionenfach in Kraftfahrzeugen eingebaut und stellen einen großen Gewinn für die Sicherheit im Straßenverkehr dar. Das Anti-Blockier-System (ABS) besteht im wesentlichen aus einem Sensor an jedem Rad und einem zentralen Steuergerät. Die Sensoren messen die Drehgeschwindigkeit der Räder und übermitteln diese an das Steuergerät. Für den Fall, dass bei einer Gefahrenbremsung ein oder mehrere Räder blockieren, kann das Steuergerät in den Bremsvorgang eingreifen, um das Blockieren der Räder zu vermeiden. Somit bleibt das Fahrzeug bei einer Vollbremsung kontrollierbar. Die Sensoren sind aufgrund ihrer Montage in der Nähe der Bremsanlage des Rades enormen mechanischen Belastungen wie Vibration, Verschmutzung sowie großen Temperaturschwankungen ausgesetzt.

ABS-Sensoren, die auf dem anisotropen magnetoresistiven Effekt (AMR-Effekt) basieren, sind für diese Anwendung aufgrund ihrer berührungslosen Drehzahlerfassung gut geeignet. An jeder Radnabe im Fahrzeug ist ein Encoderrad montiert, am Radträger ein Sensor (siehe Skizze in Abbildung 1.1).

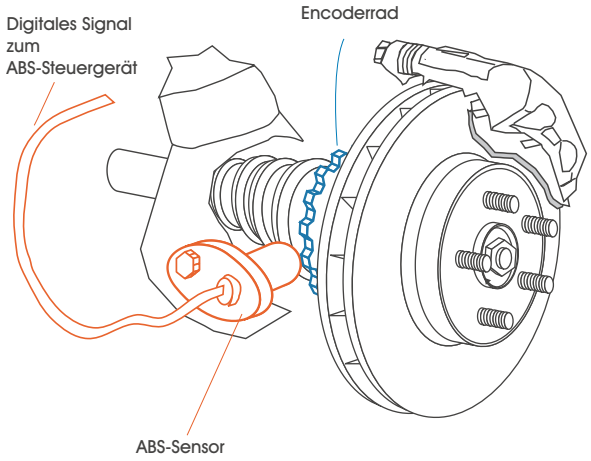


Abbildung 1.1: *Skizze des Sensoreinbaus am Fahrzeugrad, Quelle: [8]*

Das Encoderrad beeinflusst ein magnetisches Feld, welches durch einen Stützmagneten am Sensor erzeugt wird. Dieses modulierte Magnetfeld wird vom AMR-Sensor detektiert und dient zur Erkennung der Zähne des sich drehenden Encoderrades. Abbildung 1.2 zeigt die Beeinflussung der magnetischen Feldlinien durch die Zähne des Encoders, welcher aus ferromagnetischem Material besteht.

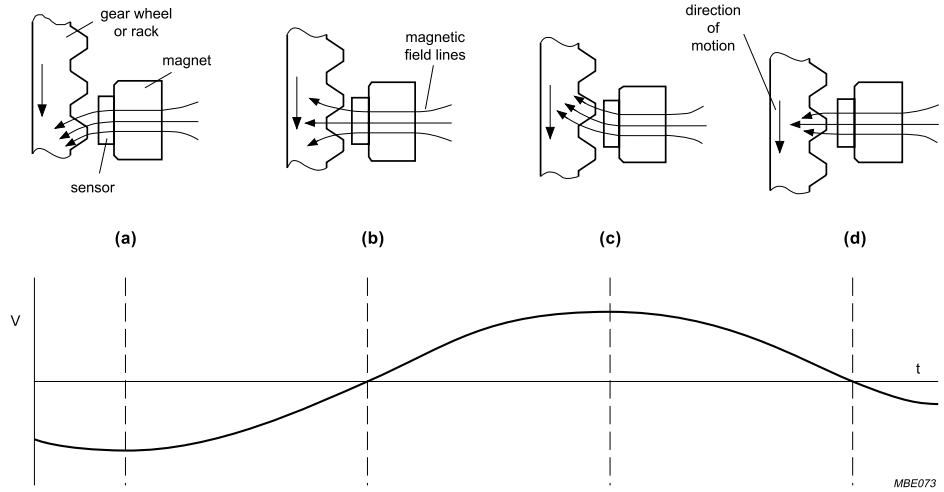


Abbildung 1.2: *Entstehung des sinusförmigen Sensorsignals, entnommen aus [14]*

Als magnetoresistiv werden Materialien bezeichnet, die durch das Anlegen von externen Magnetfeldern ihren elektrischen Widerstand verändern. Für ABS-Sensoren wird speziell der anisotrope magnetoresistive Effekt ausgenutzt, dieser beruht auf raumrichtungsabhängiger (anisotroper) Streuung in ferromagnetischen Materialien. Eine ausführliche Erläuterung des AMR-Effektes befindet sich unter [4].

Zur praktischen Nutzung dieses Effektes werden vier magnetoresistive Widerstände in einer Wheatstoneschen Messbrücke verschaltet. Durch Drehung des Encoderrades verändern sich die Widerstandswerte, was eine Verstimmung der Brückenschaltung zur Folge hat. Wird die Brücke mit einer Gleichspannung versorgt, ensteht durch Drehbewegung eine sinusförmige Brückendiagonalspannung. Die Nulldurchgänge des Sinussignals repräsentieren dabei die Zähne des Encoderrades.

Der AMR-Effekt weißt allerdings eine starke Arbeitspunkt-Abhängigkeit auf. Außerhalb des idealen Arbeitspunktes, der sich durch äußere Einflüsse, wie beispielsweise einer Veränderung der Einbaulage des Sensors, verschieben kann, treten unter Umständen starke Signalverzerrungen auf, die zu einer Fehlfunktion des Sensors führen könnten. Diese Bachelorthesis ist im Rahmen des Forschungsprojektes „Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren“ (ESZ-ABS) enstanden. Ziel des Forschungsvorhabens ist, eine Dia-

gnosefunktion in den Sensor zu integrieren, um solche Arbeitspunktverschiebungen festzustellen und an das ABS-Steuergerät zu melden.

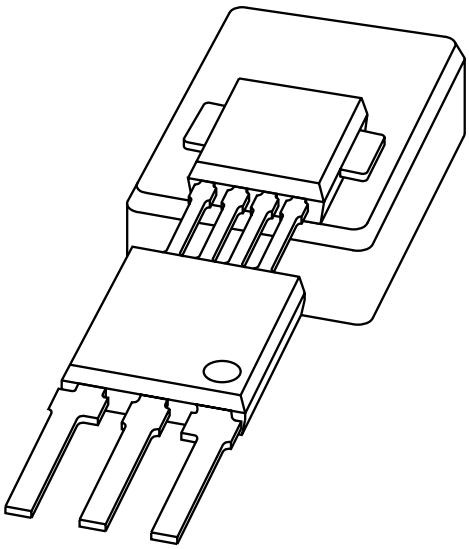


Abbildung 1.3: *Zeichnung des KMI22 ABS-Sensors von NXP, entnommen aus [15]*

Abbildung 1.3 zeigt eine Zeichnung des ABS-Sensors KMI22 der Firma NXP Semiconductors. Im oberen Teil befindet sich der sogenannte Sensorkopf, welcher die magnetoresistiven Widerstände enthält. Der Kopf ist auf einen Permanentmagneten aufgeklebt, der zur Erzeugung eines magnetischen Stützfeldes dient. Im unteren Sensorabschnitt befindet sich ein signalverarbeitender Teil. Die Hauptaufgabe dieser integrierten Schaltung (IC) liegt in der Erkennung der Nulldurchgänge des sinusförmigen Sensorsignals und der Ausgabe von digitalen Pulsen an das ABS-Steuergerät mit jedem Nulldurchgang. Zusätzlich zu den sogenannten „Speed-Pulsen“ wird ein digitales Protokoll ausgegeben, in dem beispielsweise der Sensorzustand, die Drehrichtung oder Information über den Luftspalt übertragen werden. Um eine Funktion über den spezifizierten Luftspalt zwischen Sensor und Encoderrad von bis zu 4,5mm garantieren zu können, ist eine schaltbare Verstärkung sowie Kompensation des Offsets des Sensorsignals erforderlich. Diese Funktionen befinden sich ebenfalls im Signalverarbeitungs-IC. Neben den bisher erwähnten passiven Encoderrädern gibt es auch aktive Encoder. Diese sind magnetisiert und erzeugen dadurch selbst ein Magnetfeld, welches den Sensor beeinflusst. In Abbildung 1.4 sind Fotos von Radnaben mit passivem und aktivem Encoder dargestellt.

Abbildung 1.4: *Links: Radnabe mit passivem Encoder vom VW Golf IV*
Rechts: Radnabe mit aktiven Encoderring vom VW Golf V

1.2 Bestehende Demonstrator-Plattform

Diese Arbeit stellt eine Weiterführung der Diplomarbeit *Entwicklung eines Controllersystems zur Zustandserkennung von ABS-Sensoren* von Niels Jegenhorst [10] dar. Damit diese Arbeit für sich allein verstanden werden kann, soll zunächst kurz auf das in [10] entwickelte Controllersystem, im Folgenden Demonstrator-Plattform genannt, eingegangen werden.

Die Demonstrator-Plattform erfüllt zwei Funktionen. Zum einen sollen die grundlegenden Merkmale eines KMI22 ABS-Sensors von NXP nachgebildet werden. Dazu gehören:

- Digitales Strom-Ausgangssignal
- Funktion bei verschiedenen Einbaulagen bis zu einem Luftspalt von 4,5mm
- Drehrichtungserkennnung
- Digitales Ausgabeprotokoll
- Stillstandserkennung
- Verstärkungsregelung
- Digitale Offset-Kompensation

Zum anderen sollen Sensordiagnosefunktionen für eine Zustandserkennung des Sensorsystems implementiert werden. In der vorangegangen Diplomarbeit wurde hierfür ein Verfahren zur Klirrfaktoranalyse des Sensorsignals entwickelt. Die Demonstrator-Plattform dient also als Evaluations-Hardware zur Erforschung von Diagnoseverfahren für die Sensor-Zustandserkennung.

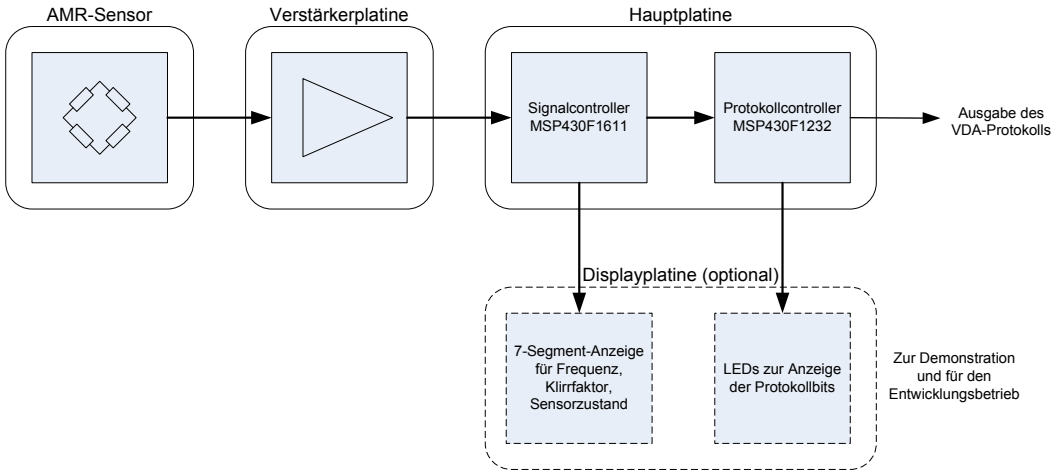


Abbildung 1.5: *Blockschaltbild der bestehenden Demonstrator-Plattform*

In Abbildung 1.5 ist ein Blockschaltbild der Demonstrator-Plattform dargestellt. Das System besteht im Wesentlichen aus den folgenden drei Komponenten:

1. Hauptplatine – bestehend aus Signalcontroller und Protokollcontroller
2. Verstärkerplatine – zur Verstärkung des Brückendifferenzsignals sowie der Halbbrückensignale
3. Displayplatine – beinhaltet 7-Segment-Anzeigen zur optischen Ausgabe von Zahnfrequenz, Klirrfaktor, Sensorzustand sowie LEDs zur Anzeige der Protokollbits des digitalen Ausgangeprotokolls

Bei dem im Blockschaltbild enthaltenen AMR-Sensor handelt es sich ledigleich um den sogenannten Sensorkopf, welcher die AMR-Widerstandsbrücke enthält. Der signalverarbeitende Teil des Sensorsystems wurde vom Kopf abgetrennt, da dieser durch die Demonstratorplattform ersetzt wird. In Abbildung 1.6 werden das Brückendifferenzsignal U_{Diff}, sowie die beiden Halbbrückensignale U_{HB1} und U_{HB2} definiert.

Im Folgenden wird kurz auf die drei Komponenten eingegangen, da dies wichtig für das Verständnis dieser Arbeit ist. Details zur Entwicklung sind in [10] nachzulesen.

1.2.1 Hauptplatine

Die Hauptplatine stellt die Basis des Systems dar. Auf ihr befinden sich Signal- und Protokollcontroller inklusive notwendiger Peripherie wie beispielsweise Spannungsversorgung, JTAG-Schnittstelle, UART-Schnittstelle, Reset-Schaltung. Des

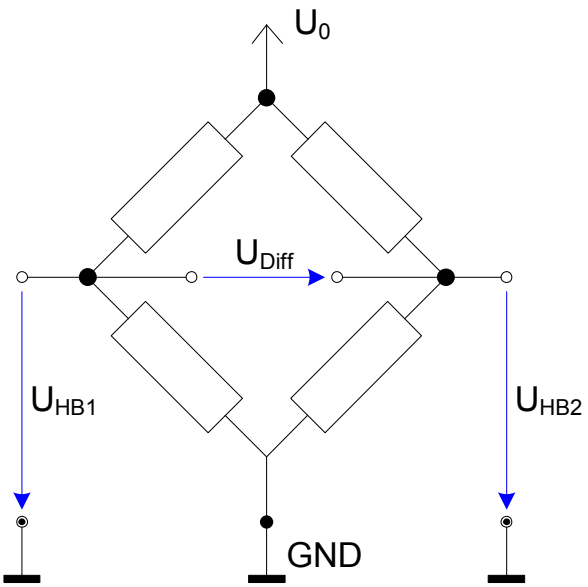


Abbildung 1.6: *Definition von Brückendifferenzsignal und Halbbrückensignalen*

Weiteren gibt es Jumper, die mit Portpins des Signalcontrollers verbunden sind, Anschlüsse für die optionale Displayplatine sowie Stiftleisten zum Aufstecken der Verstärkerplatine. Mit dem Protokollcontroller verbunden sind schaltbare Stromquellen, die für die Ausgabe des digitalen Stromprotokolls benötigt werden.

In der Diplomarbeit wurden alle Sensorgrundfunktionen sowie ein besonders minimalistisches Verfahren zur Klirrfaktorbestimmung für die Sensordiagnose erfolgreich implementiert. Die zugehörige Software musste für den Betrieb auf einem 16-Bit Mikrocontroller *MSP430F1611* mit sehr begrenzten Rescourcen stark optimiert und im Timing genau analysiert werden. Im laufenden Betrieb ist der Signalcontoller mit der Bestimmung des Klirrfaktors bereits vollständig ausgelastet, weshalb die Ausgabe des digitalen Stromprotokolls auf einen zusätzlichen Protokollcontroller ausgelagert wurde. Die Einstellung des Verstärkungsfaktors erfolgt manuell über Jumper und muss somit je nach Entfernung zwischen Sensor und Encoderrad angepasst werden. Die digitale Kompensation des Signal-Offsets erfolgt einmalig beim „PowerOn“ des Systems. Im aktiven Zustand bietet das System aufgrund der Auslastung durch die Diagnosefunktion keine Möglichkeit mehr die Verstärkung zu regeln oder das Offset zu kompensieren. Für die Aufnahme einer Messreihe muss das System somit oft vom Bediener neugestartet werden.

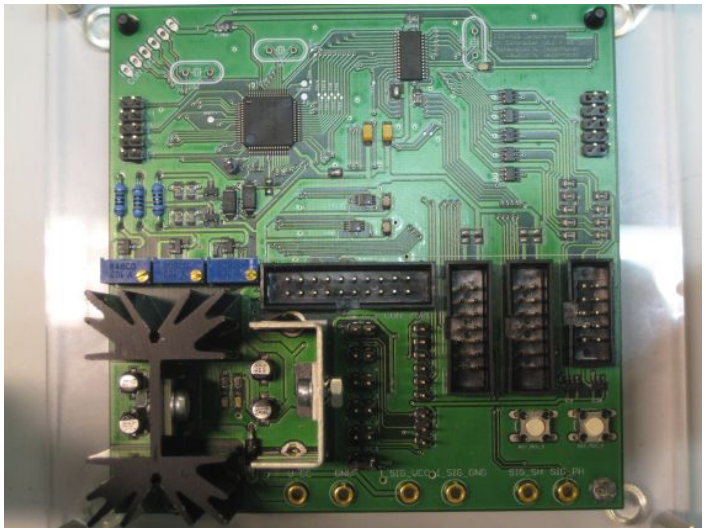


Abbildung 1.7: *Foto der Hauptplatine*

1.2.2 Verstärkerplatine

Die Verstärkerplatine wird auf die Hauptplatine aufgesteckt und dient zur Verstärkung der Brückendifferenzspannung U_{Diff} sowie der Halbbrückensignale U_{HB_1} und U_{HB_2}. Für die Bestimmung des Klirrfaktors muss das Differenzsignal U_{Diff} abgetastet werden. Es wird deshalb so verstärkt, dass es es stets im Aussteuerbereich des Analog-Digital-Converters $(0 \ldots 2{,}5V)$ liegt.

Zunächst wird das Differenzsignal mit einem Instrumentenverstärker[1] um den Faktor $v_1 = 25$ verstärkt. Im Anschluss folgt ein zweiter Instrumentenverstärker. Dieser ist über 4 digitale Leitungen (drei Datenleitungen, eine Chip Select Leitung) in der Verstärkung einstellbar. Folgende Verstärkungsfaktoren sind hierbei möglich:

$$v_2 = 2^x mit x = 0 \ldots 7 \tag{1.1}$$

Die resultierende minimale Verstärkung ergibt sich also zu:

$$v_{min} = v_1 \cdot 2^0 = 25 \cdot 1 = 25 \tag{1.2}$$

Der größtmögliche Verstärkungsfaktor lautet:

[1] präzise Operationsverstärker-Schaltung mit sehr hohem Eingangswiderstand, hier als integrierte Schaltung

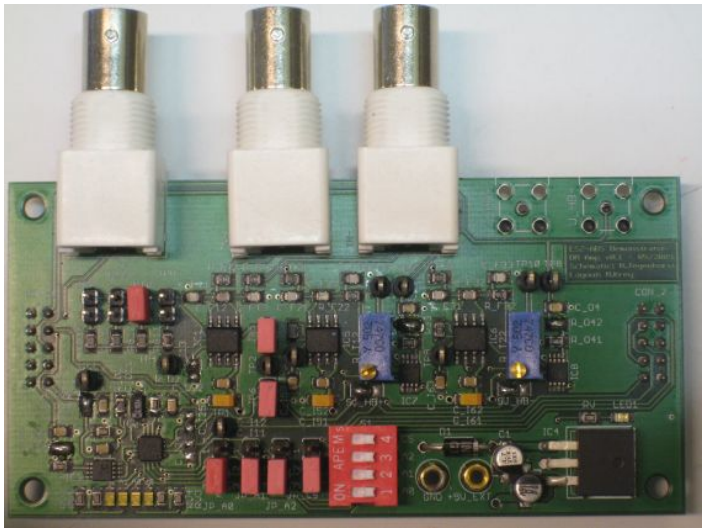


Abbildung 1.8: *Foto der Verstärkerplatine*

$$v_{max} = v_1 \cdot 2^7 = 128 \cdot 1 = 3200 \tag{1.3}$$

Die Einstellung des Verstärkungsfaktors erfolgt bei dem bisherigen System wie bereits erwähnt manuell über Jumper. Die Halbbrückensignale U_{HB_1} und U_{HB_2} werden stets um den festen Faktor $v_{HB} = 25$ verstärkt. Diese Verstärkungsfaktoren wurden aufgrund von vorhandenen Erfahrungen und Messungen an einem passivem Encoderrad gewählt und sind in [10] näher erläutert.

1.2.3 Displayplatine

Die Displayplatine kann an die Hauptplatine angeschlossen werden. Sie ist für die Funktion des Systems nicht notwendig, ermöglicht es aber, einige Parameter des Systems direkt abzulesen. Auf den oberen vier 7-Segment-Anzeigen wird die aktuelle Zahnfrequenz des Encoderrades angezeigt. Auf den zwei Stellen unten rechts erfolgt die Ausgabe des Klirrfaktors, welcher vom Signalcontroller ermittelt wurde. Unten links wird der aktuelle Zustand des Sensorsystems angezeigt (auf dem Signalcontroller sind verschachtelte Zustandsautomaten implementiert, auf die im weiteren Verlauf dieser Arbeit noch eingegangen wird).

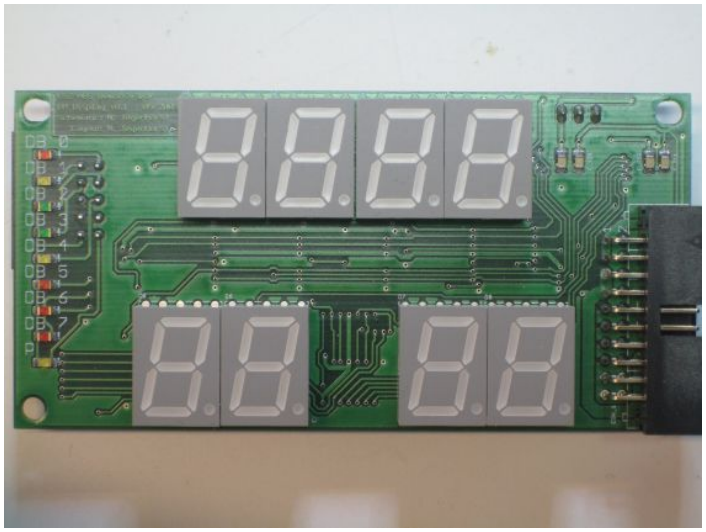


Abbildung 1.9: *Foto der Displayplatine*

1.3 Aufgabenstellung

Diese Arbeit soll die bestehende Demonstratorplattform um ein zusätzliches Modul erweitern. Mit Hilfe dieses zu entwickelnden Controllersystems sollen Offsetkompensation und Verstärkungsregelung in Form eines nebenläufigen Prozesses, parallel zur Diagnoseberechnung auf dem Signalcontroller, ermöglicht werden. Dabei ist zu beachten, dass ABS-Sensoren in einem relativ breitbandigen Frequenzbereich von $1Hz \ldots 2{,}5kHz$ spezifiziert sind. Eine Frequenz unterhalb von 1Hz muss als Stillstand des Rades erkannt werden, dieses ist relevant für die Funktion des ABS-Systems. Des Weiteren wird auf die Automatisierung zweier Radmessplätze eingegangen, mit dessen Hilfe vollautomatische Messreihen durchgeführt werden können. Dabei handelt es sich um einen Messplatz mit aktivem und einen mit passivem Encoderrad.

2 Analyse

2.1 Verstärkungsregelung

Der AMR-Effekt und damit auch die AMR-Brückendifferenzspannung ist stark von der Einbaulage des Sensors zum Encoderrad abhängig. In Abbildung 2.1 ist die entsprechende Kennlinie, entnommen aus dem KMI22 Datenblatt, dargestellt. Bei den drei Bits, welche auf einem Intervall dargestellt sind, handelt es sich um die sogenannten *air gap measurement bits*. Diese Bits werden aus dem eingestellten Verstärkungsfaktor abgeleitet und sollen so Informationen über den Luftspalt enthalten. Für die Übertragung an das ABS-Steuergerät werden diese Bits in das digitale Stromprotokoll der Sensoren codiert.

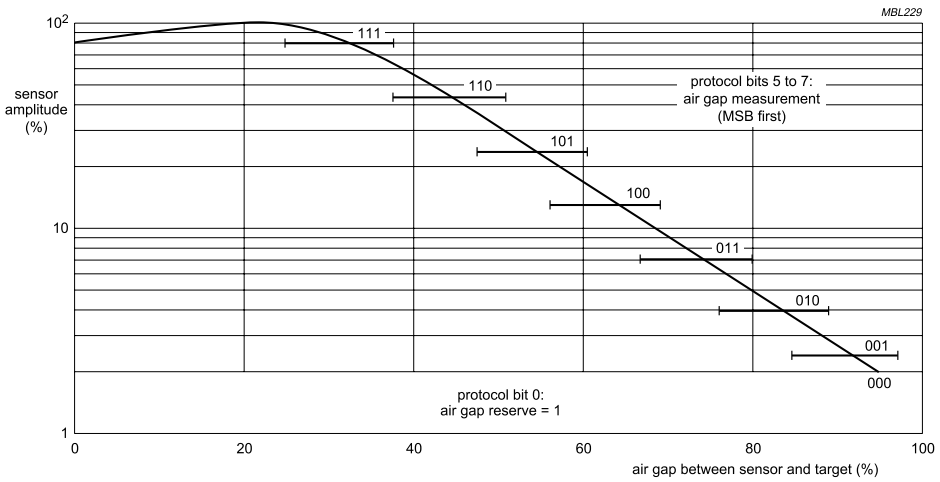


Abbildung 2.1: *Amplitude in Abhängigkeit des Luftspaltes, aus KMI22 Datenblatt [15]*

Da die im Datenblatt angegebene Kennlinie keine konkreten Werte für Spannung und Luftspalt enthält und zudem eher eine schematische Darstellung ist, wurde die Abhängigkeit zunächst einmal ausgemessen. Durchgeführt wurde die Messung auf einem Radmessplatz mit passivem Encoderrad, der per Matlab-Script gesteuert werden kann und der bestehenden Demonstrator-Plattform (Kapitel 1.2). Die Messung wurde in einem Bereich von $0 \ldots 6mm$ mit einer Schrittweite von $0{,}05mm$ duchgeführt. Das Ergebnis ist in Abbildung 2.2 dargestellt.

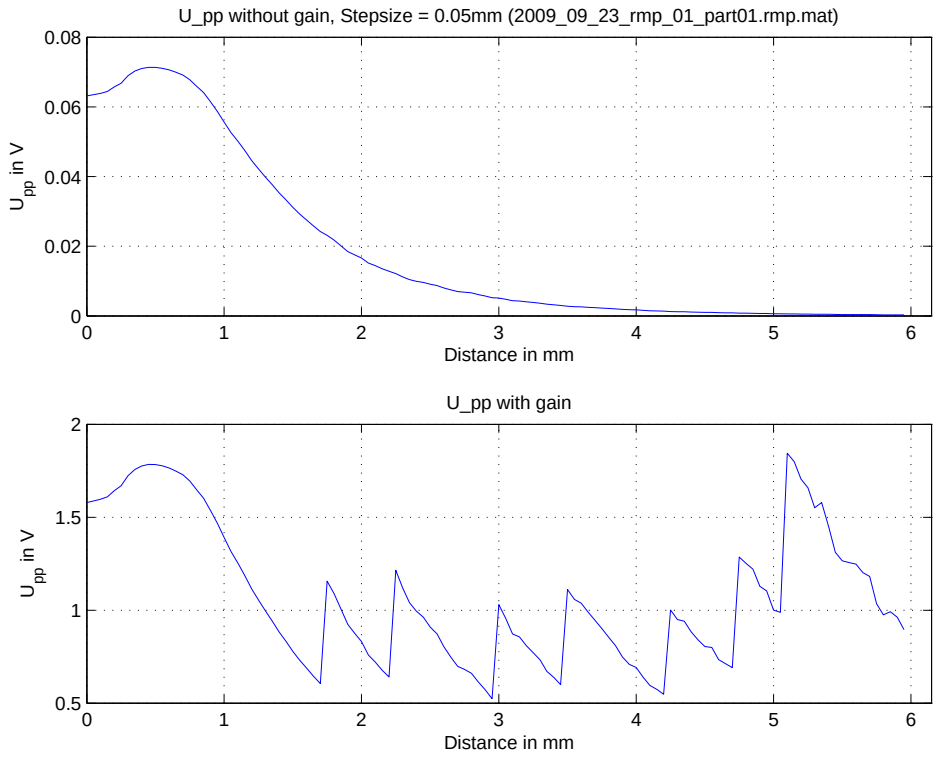


Abbildung 2.2: *Oben: Brückendifferenzspannung in Abhängigkeit der Einbaudistanz unverstärkt Unten: Verstärktes Sensorsignal in Abhängigkeit der Einbaudistanz, die Sprüunge enstehen durch manuelles Umschalten des Verstärkungsfaktors*

Die obere Kurve stellt die Differenzspannung U_{Diff} der AMR-Brücke dar. Der Sensor wurde mit einer Spannung von $U_0 = 3V$ versorgt. Es wird deutlich, dass die Brückenspannung über den Luftspalt nicht, wie im Datenblatt gezeichnet, linear, sondern eher exponentiell abnimmt. Die Amplitude liegt im Maximum bei $U_{peak} \approx 70mV$ und geht im Minimun gegen 0. Die untere Kurve berücksichtigt in jedem Messpunkt den eingestellten Verstärkungsfaktor und ist somit die Spannung wie sie am Analog-Digital-Converter des Mikrocontrollers anliegt. Das Maximum liegt bei $U_{peak} \approx 70mV \cdot 25 = 1{,}75V$. Jeder Sprung in der Kurve enststeht aufgrund einer Verdoppelung der Verstärkung. Dieser Verstärkungsfaktor muss bei der vorhandenen Demonstrator-Plattform allerdings manuell über Jumper eingestellt werden.

2.1.1 Notwendigkeit der Regelung

Es gibt zwei wichtige Gründe, weshalb eine Regelung des Verstärkungsfaktors erforderlich ist.

1. Für die Erkennung der Nulldurchgänge des Sensorsignals wird ein Komparator mit Hysterese (ähnlich Schmitt-Trigger) verwendet, welcher als „Smart-Comparator“ bezeichnet wird. Damit die Schaltschwellen des Komparators erreicht werden, muss das Signal eine entsprechend große Mindest-Amplitude haben.
2. Für die Diagnosefunktion des Sensorsystem muss das Signal abgetastet werden. Der Analog-Digital-Converter (ADC) hat einen Aussteuerbereich von $U_{ADC} = 0 \ldots 2{,}5V$. Das Signal muss daher so verstärkt werden, dass der Aussteuerbereich möglichst gut genutzt wird, ohne diesen jedoch zu übersteuern.

In der unter Abbildung 2.2 durchgeführten Messung wurde die Verstärkung so eingestellt, dass sich die Spannung am ADC im Bereich von $25 \ldots 75\%$ des Auststeuerbereiches befindet, dies entspricht einer Spannung im Bereich von $0{,}625 \ldots 1{,}875V$. Damit ist gewährleistet, dass die Nulldurchgänge des Signals sicher erkannt werden. Des weiteren führen Störungen oder kleine „Ausreißer“ im Signal nicht sofort zu einer Übersteuerung des ADC.

Im Folgenden werden verschiedene Möglichkeiten für eine Regelung des Verstärkungsfaktors diskutiert.

2.1.2 Erweiterung des Smart-Comparators

Zunächst soll kurz das Prinzip des sogenannten „Smart-Comparators“ erläutert werden, wie er in [10] implementiert wurde. Es handelt sich dabei um einen Komparator mit einstellbarer Hysterese. Dieser dient zur Erkennung der Signal-Nulldurchgänge und wird eingesetzt, um ein mehrfaches Schalten (Prellen) des Komparators bei langsamen Signalen zu verhindern. Die analogen Spannungen für die Hystereseschwellen werden vom internen Digital-Analog-Converter (DAC) des Mikrocontrollers generiert und auf den einen Eingang des ebenfalls internen Komparators gegeben (siehe Abbildung 2.3). Dadurch kann die Schaltschwelle des Komparators per Software eingestellt und somit eine Hysterese realisiert werden. Der Komparator erzeugt beim Erreichen der angelegten Schwellspannung eine Interruptanforderung. In der zugehörigen Interrupt Service Routine (ISR) wird mittels DAC eine neue Spannung für die nächste Schwelle vorgegeben. Abbildung 2.4 zeigt die Arbeitsweise des „Smart-Comparators“. Die Hystereseschwellen $\pm HYST$ werden in der Grafik als U_{s1} und U_{s2} bezeihnet.

Da also bereits ein „Smart-Comparator“ zur Nulldurchgangserkennung benötigt wird, liegt es auf der Hand, diesen auch für die Erkennung von Grenzwerten für die Verstärkungsregelung zu verwenden. Wie in Abbildung 2.5 zu erkennen ist, werden dafür zusätzlich zu den beschriebenen Hystereseschwellen $+HYST$

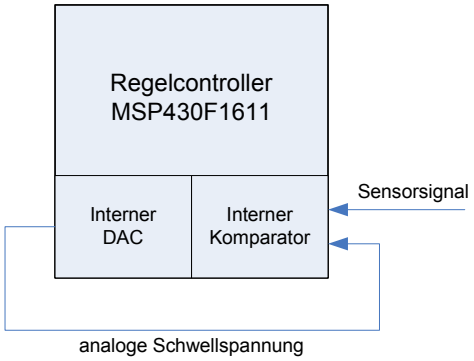


Abbildung 2.3: *Schematische Darstellung: Realisierung des „Smart-Comparators“*

und $-HYST$ vier weitere Schwellen $\pm GAIN_{Max}$ sowie $\pm GAIN_{Min}$ definiert. In der Zeichnung ist ein korrekt verstärktes Signal dargestellt. Die unteren Schwellwerte $\pm GAIN_{Min}$ werden passiert, die oberen $\pm GAIN_{Max}$ hingegen nicht. Genau hier ergibt sich bereits das erste Problem. Der Komparator schaltet erst, wenn die vorgegebene Schwellspannung erreicht wird. Da dies bei einem korrekt verstärktem Signal für $\pm GAIN_{Max}$ aber nie der Fall sein wird, gibt es keine Interrupt-Anforderung und damit auch kein Umschalten auf die nächste Schwelle.

Um dieses Problem zu umgehen, müsste die Periodenlänge des Signals vorausberechnet werden. Somit könnte eine Zeit vom erfolgten Nulldurchgang bis zum vermuteten Maximum „geschätzt“ werden. Nach Ablauf dieser Zeit würde der Komparator kurz auf die Schwelle $\pm GAIN_{Max}$ gesetzt. Sollte diese Schwelle passiert werden, wäre das ein Indikator für eine zu hohe Verstärkung. Falls der Komparator innerhalb eines kurzen Timeouts nicht schaltet, muss die Schwelle unbedingt wieder zurückgesetzt werden, damit der nächste Nulldurchgang erkannt werden kann.

Die Zuverlässigkeit dieses Verfahrens ist allerdings schwierig vorherzusagen. Das Sensorsignal ist nur im Idealfall sinusförmig. Bei falschen Einbaulagen oder anderen Problemen mit dem Sensor weist das Signal unter Umständen starke Verzerrungen, wie beispielsweise Einbrüche des Maximums auf. Der Komparator würde unter Umständen wärend des Zeitfensters der Schwelle $\pm GAIN_{Max}$ nicht schalten, obwohl die Verstärkung zu groß ist. Es ist also sehr kritisch, eine Aussage über die korrekte Funktion dieses Verfahrens unter allen erdenklichen Umständen zu treffen. Eine korrekte Erkennung der Nulldurchgänge muss auf jeden Fall gewährleistet sein. Dies ist für die Funktion des ABS-Systems absolut erforderlich.

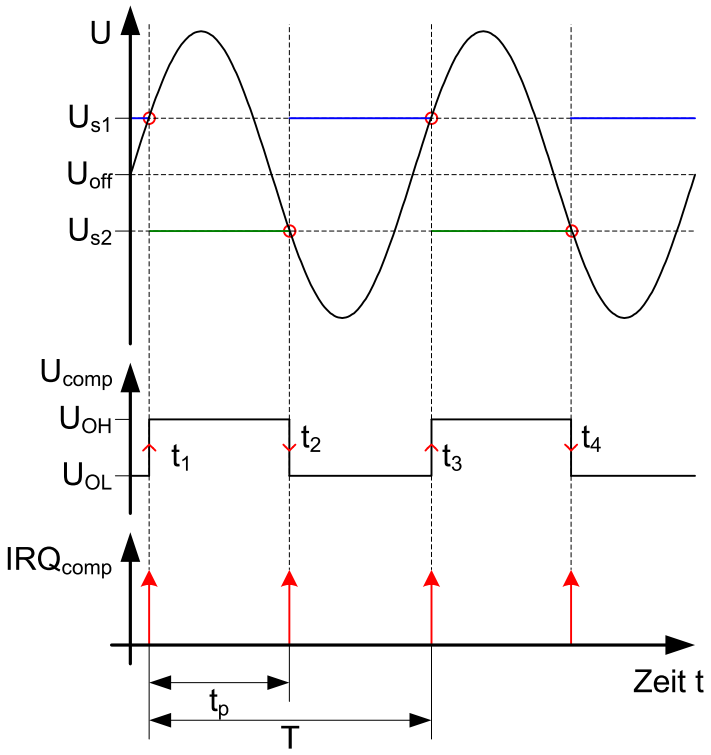


Abbildung 2.4: *Erkennung der Nulldurchgänge mittels „Smart-Comparator“, Quelle: [10]*

2.1.3 Nutzung eines zusätzlichen Smart-Comparators

Da ein „Smart-Comparators“ sowohl für die Nulldurchgangserkennung als auch für die Verstärkungsregelung, wie beschrieben, kritisch ist, könnte man stattdessen zwei Komparatoren verwenden. Der eine würde weiterhin die beschriebene Nulldurchgangserkennung durchführen, der zusätzliche Komparator wäre für die Erkennung der Gain-Schwellen zuständig. Durch diese Trennung der Aufgaben wäre eine Erkennung der Nulldurchgänge unter allen Umständen gewährleistet. Die Realisierung ist zwar prinzipiell möglich, es gibt aber hardware-spezifische Faktoren, die gegen solch eine Lösung sprechen.

1. Der verwendete Mikrocontroller *MSP430F1611* hat nur einen internen Komparator und nur zwei interne DACs, von denen einer für die digitale Offsetkompensation benötigt wird. Auf dem zu entwickelnden Controllersystem müsste also ein zusätzlicher externer Komparator sowie ein Digital-Analog-Converter vorgesehen werden.
2. In Hinblick auf einen späteren Chipentwurf sollte möglichst auf einen zweiten Komparator verzichtet werden, weil dieser viel Fläche auf dem Chip in Anspruch nimmt.

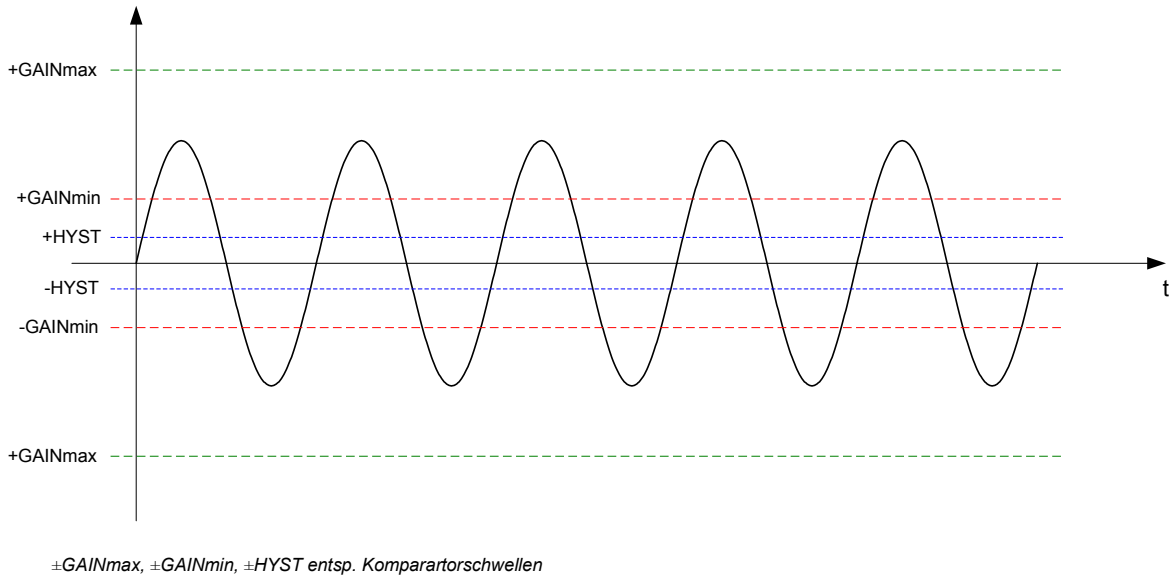


Abbildung 2.5: *Darstellung der Komparatorschwellen für Nulldurchgangserkennung und Verstärkungsregelung*

2.1.4 Abtastung des Signals mit freilaufendem ADC

Eine weitere Möglichkeit für die Realisierung einer Verstärkungsregelung ist die Abtastung des Sensorsignals U_{Diff}. Anhand der Abtastwerte kann dann festgestellt werden, ob das Signal zu schwach, zu stark oder korrekt verstärkt ist. Um das Abtasttheorem zu erfüllen, muss die Abtastfrequenz f_{Sample} mindestens der doppelten maximalen Signalfrequenz f_{Max} entsprechen. Es gilt also:

$$f_{Sample} > 2 \cdot f_{Max} \tag{2.1}$$
$$f_{Sample} > 2 \cdot 2{,}5kHz = 5KHz \tag{2.2}$$

Bei einer Abtastfrequenz von $5kHz$ und einer Signalfrequenz von $2{,}5kHz$ würden lediglich zwei Abtastwerte pro Periode genommen. Dies reicht laut Abtasttheorem zwar aus, um das unverzerrte Signal korrekt zu rekonstruieren, für eine Verstärkungsregelung hingegen sind zwei Werte pro Periode zu wenig. Die vier Schwellen $\pm GAIN_{Max}$ und $\pm GAIN_{Min}$ würden nicht sicher erkannt werden.

Zur Veranschaulichung des Problems dient Abbildung 2.6. Das dargestellte Signal überschreitet die oberen Grenzwerte, ist also zu stark verstärkt. Da die beiden Abtastwerte nicht im Maximum beziehungsweise Minimum des Sensorsignals genommen werden, kann die falsche Verstärkung nicht festgestellt werden. Für die-

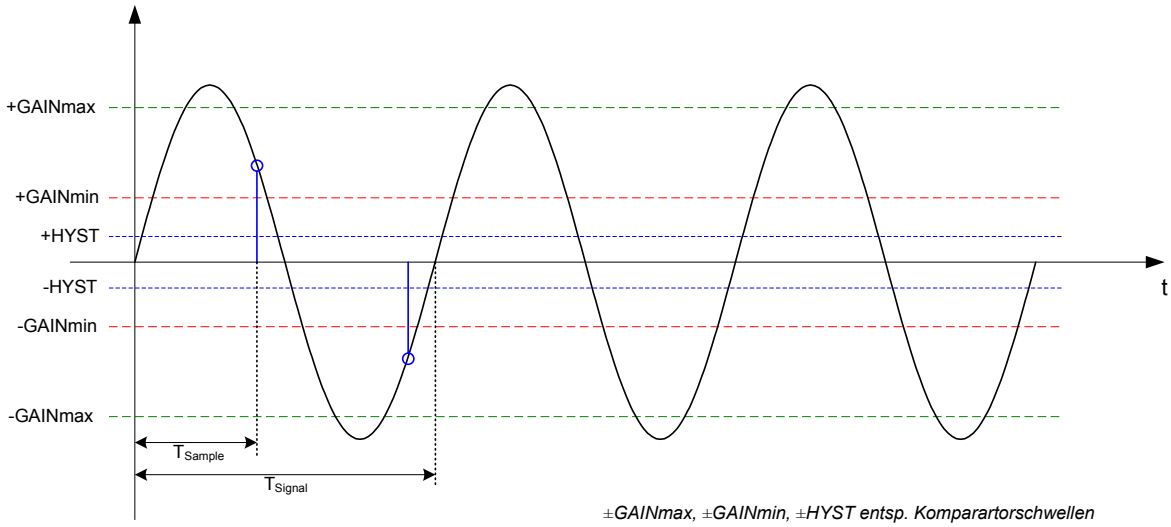


Abbildung 2.6: *Verstärkungsregelung bei zu geringer Abtastfrequenz*

ses Verfahren muss also eine Abtastfrequenz f_{Sample} gewählt werden, die deutlich größer als $5kHz$ ist.

Die Regelung der Verstärkung mittels Abtastung hat aber einen großen Vorteil: Die Implementierung in einem Controllersystem auf Basis des *MSP430F1611* lässt sich einfacher realisieren als die Verfahren mit Komparatoren. Die Offsetkompensation und Verstärkungsregelung wäre somit auf einen separaten Regelcontroller ausgelagert, so wie es für diese Arbeit gefordert ist. Für einen späteren Chipentwurf erscheint ein ADC für die Verstärkungsregelung zunächst aufwendig. Da aber bereots ein ADC für die Sensordiagnosefunktionen benötigt wird, ließe sich dieser auch für die Regelung mit verwenden, es wäre nur ein Multiplexer notwendig.

2.1.5 Phasenbezogene Abtastung des Signals

Eine Alternative zur kontinuierlichen Abtastung stellt die Abtastung im Maximum, beziehungsweise Minimum des Signals dar. Im Falle eines rein sinusförmigen Signals liegen diese Punkte bei $\frac{\pi}{2}$ und $\frac{3\pi}{2}$ (siehe Abbildung 2.7).

Ein Vorteil dieses Verfahrens ist, dass im Gegensatz zur freilaufenden Abtastung eine deutlich geringere Abtastfrequenz benötigt wird. Allerdings unterliegt das Sensorsignal unter Umständen deutlichen Verzerrungen, die sich auch in Form von Einbrüchen der Maxima äußern können. In diesem Falle wäre die Verstärkungsregelung nicht sehr zuverlässig.

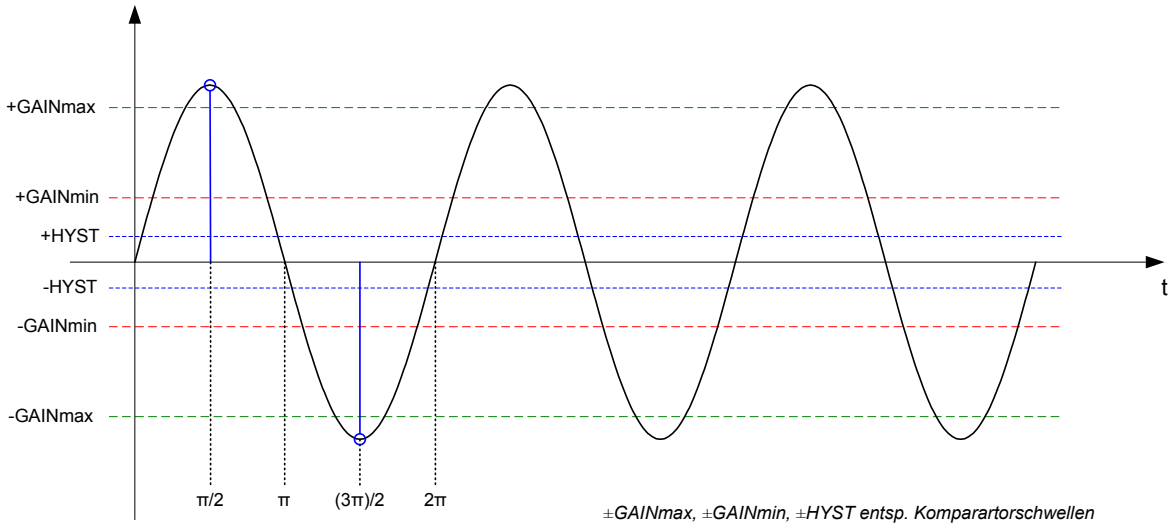


Abbildung 2.7: *Verstärkungsregelung mittels phasenbezogener Abtastung des Sensorsignals*

2.2 Offsetkompensation

2.2.1 Motivation

Ein Drehzahlsensor auf AMR-Basis besteht aus vier magnetoresistiven Widerständen, die zu einer Wheatstone-Brücke verschaltet sind. Idealerweise haben diese vier Widerstände identische elektrische Eigenschaften. Ein großer Vorteil der Brückenschaltung ist ihre hohe Empfindlichkeit, somit können die AMR-Effekt bedingt sehr kleinen Widerstandsänderungen detektiert werden. Außerdem ist eine Brückenschaltung relativ temperaturunempfindlich, da sich Temperaturänderungen auf alle Widerstände weitgehend gleichermaßen auswirken.

Dennoch treten im Betrieb des Sensors zwei grundlegende Arten von Offsets auf:

1. elektrische Offsets
2. magnetische Offsets

In Abbildung 2.8 ist die typische Kennlinie $U_{sig} = f(H_y)$ einer AMR-Widerstandsbrücke dargestellt. Bei der grünen Kurve handelt es sich um das anregende Signal, welches durch die Drehung des Encoderrades vor dem Sensor hervorgerufen wird. Das blaue Signal stellt die elektrische Ausgangsspannung U_{Diff} der AMR-Brücke dar.

Bei dem Kreuz in der Mitte handelt es sich um den idealen Arbeitspunkt des Sensors. Dieser weist einen magnetischen Offset von $H_{y_{Off}} = 0\frac{kA}{m}$ (Abszisse) und einen

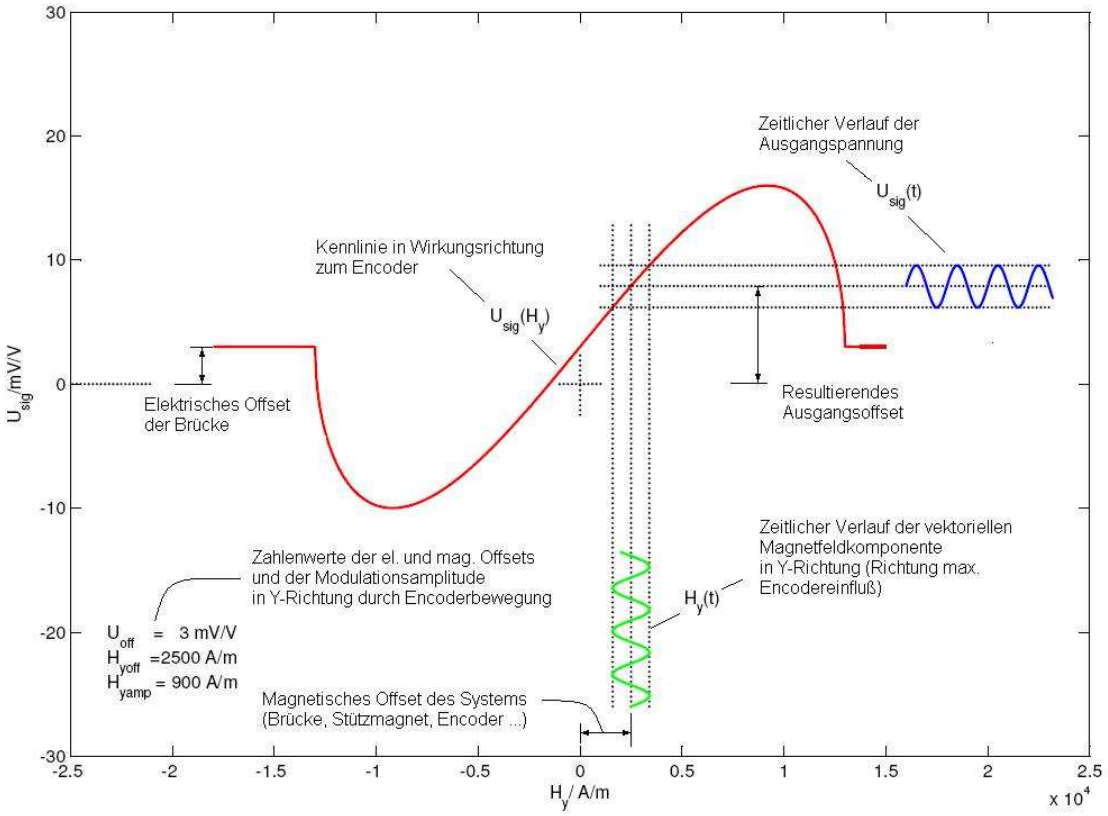


Abbildung 2.8: *Kennlinie der AMR-Brücke mit Offsets, Quelle: [16]*

elekrischen von $U_{Off} = 0mV$ (Ordinate) auf. Um diesen Punkt herum verläuft die Sensor-Kennlinie quasi linear, außerhalb wird sie deutlich nicht-linear.

2.2.2 Elektrisches Offset

Die magnetoresistiven Widerstände weisen nach dem Herstellungsprozess voneinander abweichende Werte auf, was eine Verstimmung der Brückenschaltung zur Folge hat. Um diese Verstimmung zu minimieren, ist jede Halbbrücke mit einer sogenannten Trimmstruktur versehen. Dabei handelt es sich um bis zu acht parallel geschaltete Widerstände, dessen Werte in 2er-Potenzen abgestuft sind. Jede Trimmstruktur ist in Reihe zu R_3 beziehungsweise R_4 geschaltet. Ein Ersatzschaltbild der Schaltung befindet sich in Abbildung 2.9. Mit Hilfe eines Lasers können einzelne Widerstände innerhalb einer Trimmstruktur abgetrennt werden, dies geschieht durch Zerstörung der elektrischen Anbindung. Durch das Lasertrimm-Verfahren werden also die Widerstände R_3 und R_4 beeinflusst, die Brückenschaltung wird abgeglichen. Offsetfrei – und damit abgeglichen – ist die Brücke bei einer Spannung

von $U_{Diff} = 0V$. Dies ist der Fall wenn die Gleichung

$$\frac{R_3}{R_1} = \frac{R_4}{R_2} \tag{2.3}$$

erfüllt ist.

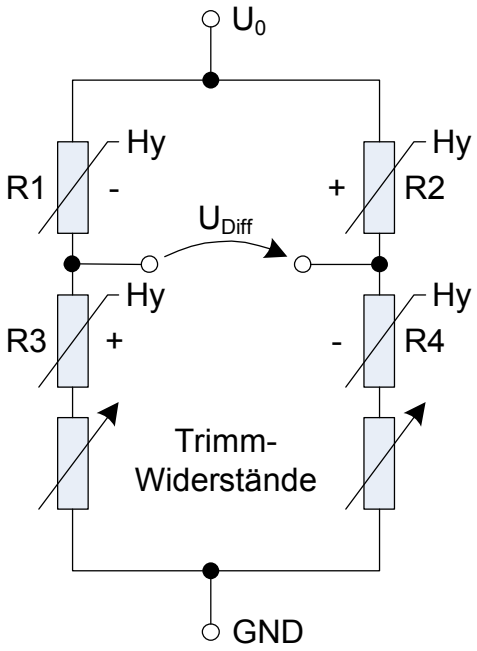


Abbildung 2.9: *Ersatzschaltbild der AMR-Brücke, R1…4: MR-Widerstände*

Auch die Lasertrimmung ist leider nicht beliebig genau. Durch die Verwendung von Trimmwiderständen, die in 2er-Potenzen gestuft sind, wird typischerweise ein Abgleich der Brücke auf 1% Genauigkeit erreicht. Hinzu kommt die Temperaturabhängigkeit des AMR-Effektes, die erwähnte Trimmung erfolgt aber bei einer festen Temperatur. Ein späterer Betrieb des Sensors unter anderen Temperaturbedingungen hat abweichende elektrische Eigenschaften der Brückenwiderstände und damit auch eine Verstimmung der Brücke zur Folge.

Die Schaltung wird im Betrieb mit einer Gleichspannung versorgt, somit entsteht bei ungleichen Brückenwiderständen auch zwangsläufig ein Gleichspannungs-Offset. Im Folgenden wurde eine Beispielrechnung durchgeführt, um zu verdeutlichen, wie sich bereits eine Widerstandsabweichung von 1% auswirkt.

$$U_{Diff} = U_3 - U_4 \tag{2.4}$$

$$\text{mit } U_3 = U_0 \frac{R_3}{R_1 + R_3} \tag{2.5}$$

$$\text{und } U_4 = U_0 \frac{R_4}{R_2 + R_4} \tag{2.6}$$

Für R_1, R_2, R_3 wird ein Wert von $R_1 = R_2 = R_3 = 1k\Omega$, für $R_4 = 1{,}01k\Omega$ angenommen. Die Versorgungsspannung betrage $U_0 = 3V$. Somit ergibt sich die Brücken-Diagonalspannung (unter Vernachlässigung der Trimmwiderstände) zu

$$U_{Diff} = U_0 \left(\frac{R_3}{R_1 + R_3} - \frac{R_4}{R_2 + R_4} \right) \tag{2.7}$$

$$= 3V \left(\frac{1k\Omega}{2k\Omega} - \frac{1{,}01k\Omega}{2{,}01k\Omega} \right) \tag{2.8}$$

$$= 3V \cdot (-2{,}49 \cdot 10^{-3}) \tag{2.9}$$

$$U_{Diff} = -7{,}46mV \tag{2.10}$$

Wie in Kapitel 1.2 beschrieben, wird die Spannung U_{Diff} vor der Verarbeitung zunächst verstärkt. Dieser Verstärkungsfaktor kann einen maximalen Wert von

$$v_{max} = v_{Amp_1} \cdot v_{Amp_2} = 25 \cdot 128 = 3200 \tag{2.11}$$

annehmen. Es ergibt sich somit eine theoretische Spannung am Verstärkerausgang von

$$\left| U_{Off} \right| = 3200 \cdot |-7{,}46mV| = 23{,}87V \tag{2.12}$$

In der praktischen Anwendung hängt das verstärkte Signal also aufgrund der Verstimmung der Brücke in der Begrenzung (Versorgungsspannung des Verstärkers). Eine Kompensation des elektrischen Offsets ist also unbedingt erforderlich.

2.2.3 Magnetisches Offset

Die im vorherigen Abschnitt beschriebene Lasertrimmung findet während der Produktion auf Wafer-Ebene statt und dient zum elektrischen Abgleich der Brücke. Am Ende des Herstellungsprozesses wird der Sensor auf einen Stützmagneten geklebt.

Dieser Stützmagnet wird gezielt auf eine bestimmte Magnetisierungsrichtung eingestellt, um ein symmetrisches Magnetfeld vor dem Sensor zu erzeugen. Das Einstellen der Magnetisierungsrichtung geschieht dabei durch das Anlegen von sehr starken, externen Magnetfeldern. Wie auch bei der Lasertrimmung wird einige Male im Wechsel ein externes Feld angelegt und anschließend die Magnetisierung gemessen. Da dieses Verfahren nicht beliebig genau ist, ensteht zwangsläufig ein nicht hundertprozentig symmetrisches Magnetfeld am Sensorkopf. Das Resultat ist ein magnetisches Offset.

Eine weitere Form von magnetischen Offsets tritt durch Wirbelströme im Encoderrad auf. Diese entstehen hauptsächlich bei hohen Zahnfrequenzen (die Sensoren sind bis $f_{Zahn} = 2{,}5kHz$ spezifiziert). Mit steigender Raddrehzahl drifted aufgrund der zunehmenden Wirbelströme auch das Signaloffset.

2.2.4 Kompensation

Sowohl elektrische, als auch magnetische Offsets machen sich als Gleichanteil im Sensorsignal bemerkbar. Anstatt den Gleichspannungsanteil nachträglich zu kompensieren, wäre es wesentlich einfacher, diesen vorher zu eliminieren. Eine Möglichkeit hierfür wäre ein analoges Hochpassfilter, beispielsweise in Form eines Koppelkondensators. Dieses Prinzip ist in der analogen Schaltungstechnik gängige Praxis und wird unter anderem bei Wechselspannungsverstärkern angewandt.

Für diese Anwendung wird allerdings eine untere Grenzfrequenz des Systems von $f_{min} = 1Hz$ gefordert. Der Sensor muss so tiefe Frequenzen – beziehungsweise langsame Raddrehzahlen – detektieren können, damit im Auto Funktionen wie beispielsweise eine Traktionskontrolle oder Berganfahrhilfe realisiert werden können. Das analoge Filter müsste also eine sehr hohe Flankensteilheit aufweisen, was einen großen schaltungstechnischen Aufwand bedeutet. Weiterhin ist mindestens ein Kondensator (alternativ eine Induktivität) erforderlich. Das Sensorsystem soll aber später in Form einer integrierten Schaltung hergestellt werden können, deshalb sind diese Lösungen nicht vertretbar.

2.2.4.1 Digitale Offsetkompensation

In Bezug auf einen späteren Chipentwurf sind möglichst digitale Lösungen für die Kompensation des Offsets zu bevorzugen. Ein gängiges Verfahren stellt dabei die Offsetkompensation anhand des Tastverhältnisses dar. Dieses Verfahren wird in kommerziellen ABS-Sensoren erfolgreich eingesetzt und wurde daher auch in der vorhergehenden Arbeit [10] implementiert. In Abbildung 2.10 ist die Funktionsweise der digitalen Offsetkompensation dargstellt.

Mittels „Smart-Comparator“ werden die Nulldurchgänge des Differenzsignals in digitale Pulse umgesetzt. Ein Übergang von der unteren in die obere Halbwelle hat eine steigende Flanke zur Folge. Eine fallende Flanke entsteht beim Übergang zurück in die untere Halbwelle. Am Anfang einer Signalperiode, also bei steigender Flanke des digitalen Signals, wird ein Timer im Mikrocontroller gestartet. Beim Eintritt einer fallenden Flanke wird der Zählerstand gepeichert. Bei der nun nächsten steigenden Flanke wird der Stand des Zähler erneut gesichert und der Timer anschließend neu gestartet. Die gespeicherten Zählerstanände repräsentieren dabei die Pulsdauer und Periodendauer des Signals, woraus sich das Tastverhältnis in Prozent berrechnen lässt:

$$\delta = \frac{Pulsdauer}{Periodendauer} \cdot 100\% \tag{2.13}$$

Wie im Bild 2.10 zu erkennen, hängen Tastverhältnis des digitalen Signals und Offset des Sensorsignals unmittelbar zusammen. Ein Tastverhälnis > 50% bedeutet ein positives Offset, bei 50% ist das Signal offsetfrei. Mittels Digital-Analog-Converter (DAC) im Mikrocontroller kann eine Offsetkompensationsspannung an die Instrumentenverstärker auf der Verstärkerplatine gelegt werden. Die Kompensation des Offsets erfolgt also durch Bestimmung des Tastverhältnisses und anschließender Anpassung der Kompensationsspannung. Somit lässt sich das Tastverhältnis sukzessiv auf 50% einstellen und dadurch das Offset kompensieren. Weitere Details zur Implementation befinden sich in [10].

Ein großer Vorteil dieses Verfahrens ist die Unempfindlichkeit gegen übersteuerte Signale. Für eine korrekte Funktionen muss ledigleich die Nulldurchgangserkennung gewährleistet sein.

2.2.4.2 Offsetkompensation über Mittelwertbildung

Eine Alternative zur Offsetkompensation anhand des Tastverhältnisses wäre die Abtastung und anschließende Mittelwertbildung des Sensorsignals. Dieser Mittelwert kann direkt in die erforderliche Kompensationspannung umgerechnet und per DAC ausgegeben werden. Dieses Verfahren arbeitet allerdings nur dann zuverlässig, wenn sich das Signal im Aussteuerbereich des Analag-Digital-Umsetzters (ADC) befindet. Beim PowerOn des Systems wird allerdings der maximale Verstärkungsfaktor eingestellt und dann schrittweise halbiert (siehe Abschnitt 2.4 auf Seite 32), wodurch der ADC stark übersteuert wird.

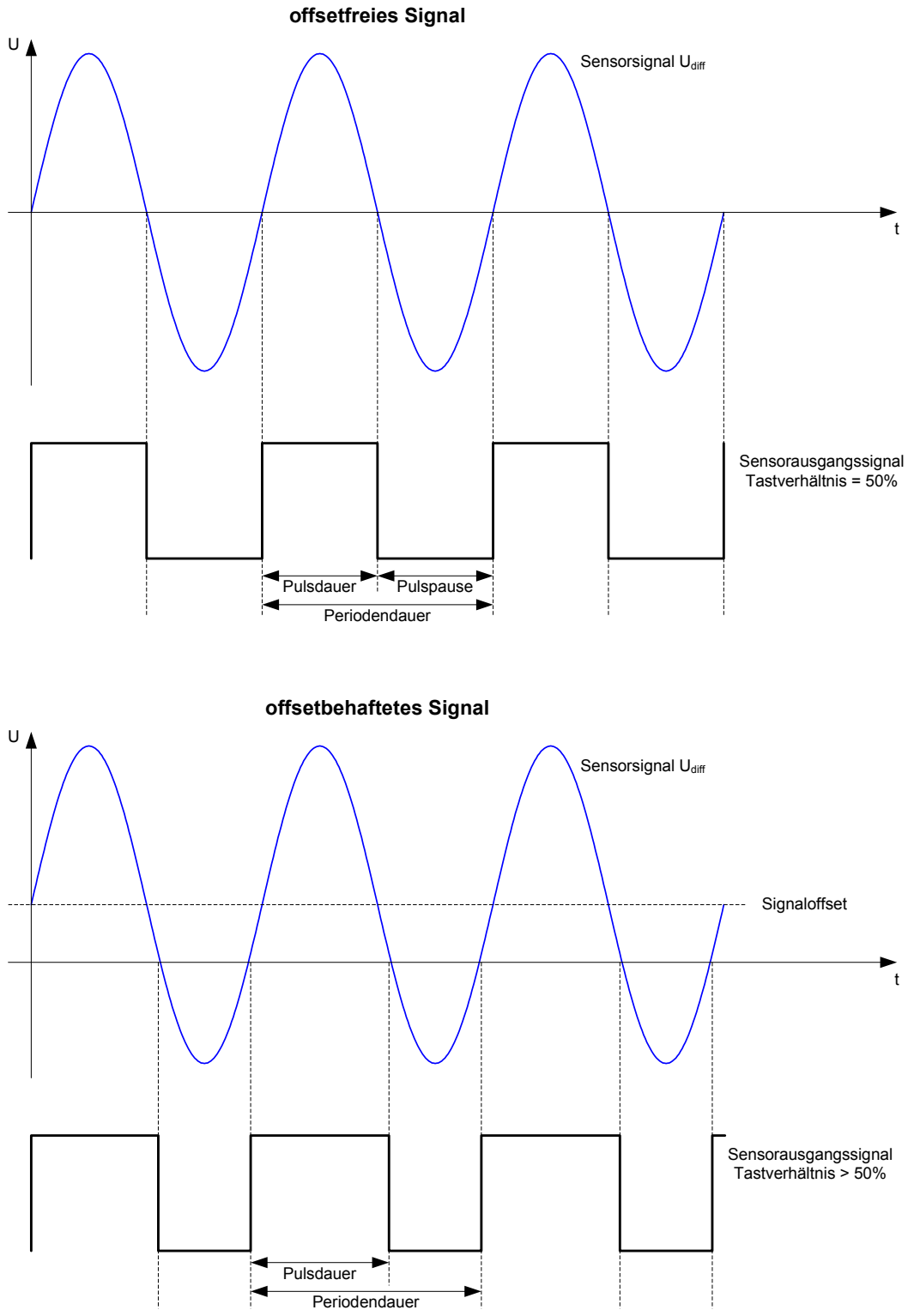


Abbildung 2.10: *Funktionsweise der digitalen Offsetkomensation*

2.3 Fazit der Konzeptentscheidung

Nach Diskussion der verschiedenen Verfahren zur Realsisierung von Offsetkompensation und Verstärkungsregelung wird nun auf die Entscheidungsfindung für eines der Verfahren eingegangen.

In Abbildung 2.11 sind noch einmal die beschriebenen Möglichkeiten zur Erfüllung der Aufgabenstellung dargestellt. Am geeignetsten für die Offsetkompensation erscheint die Regelung des Tastverhältnisses auf 50%. Dieses Verfahren wird in den erhältlichen ABS-Sensoren angewandt und wurde auch in [10] erfolgreich implementiert. Hinzu kommt, dass dieses Verfahren unempfindlich gegen eine Übersteuerung des Sensorsignals ist. Es funktioniert immer für den Fall, dass die Nulldurchgänge des Signals korrekt erkannt werden.

Für die Erfassung der Gain-Schwellen $\pm GAIN_{Max}$ und $\pm GAIN_{Min}$ wird die Abtastung mit freilaufendem ADC gewählt. Dadurch arbeitet die Verstärkungsregelung zum einen unabhängig von der Nulldurchgangserkennung, zum anderen ist das Verfahren robust gegen eventuelle Signalverzerrungen. Eine Rechnung für die erforderliche Abtastfrequenz wird in Abschnitt 2.5 durchgeführt.

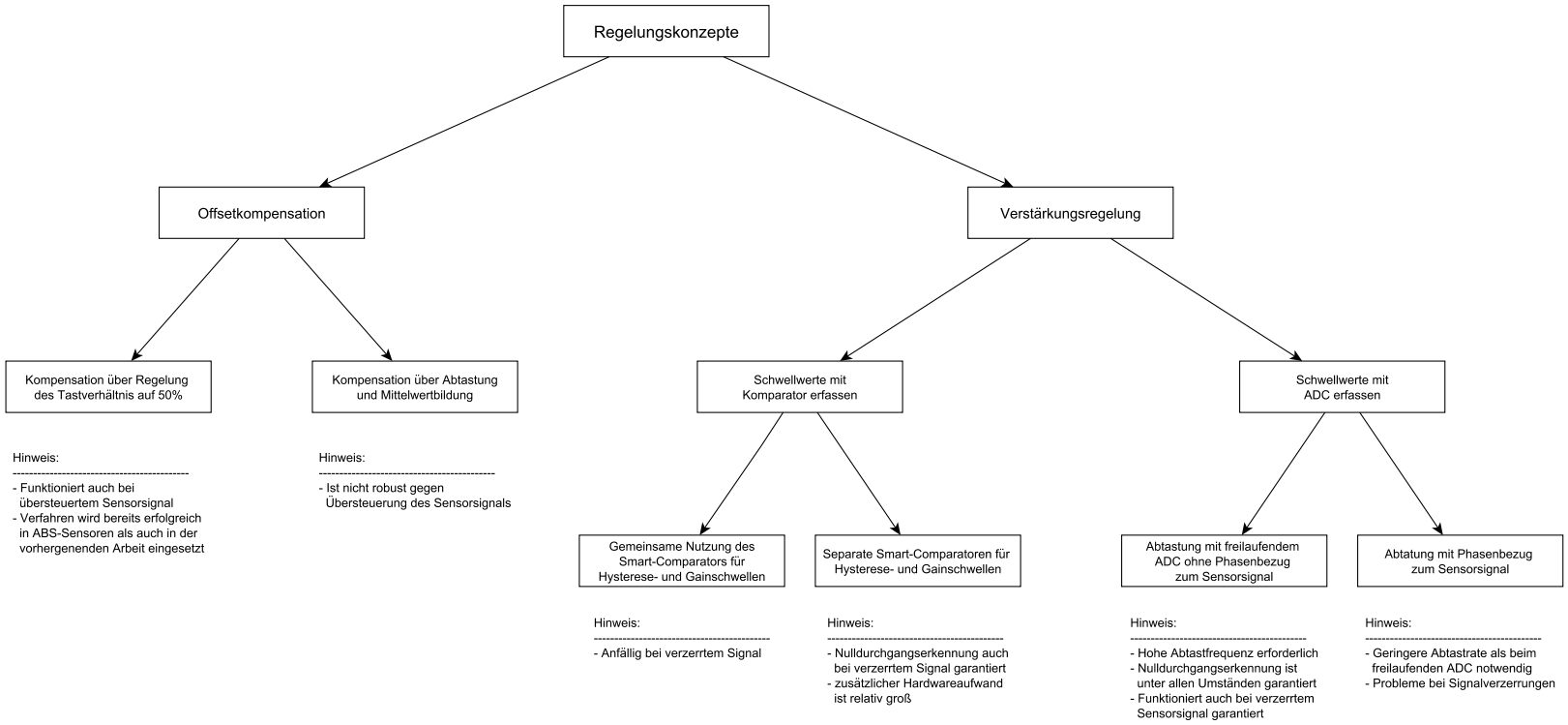


Abbildung 2.11: *Möglichkeiten zur Realisierung von Offsetkompensation und Verstärkungsregelung*

2.4 Matlab-Modell des Regelverfahrens

Um die ausgewählten Verfahren zu evaluieren, wurde zunächst ein Modell in Matlab erstellt. Der Ablauf des Matlab-Programms wurde in Hinblick auf den späteren Softwareentwurf für die Regelcontroller bereits vollständig automatengesteuert ausgelegt. Die Namensgebung der Variablen und der Zustände ist an die Software des Signalcontrollers angelehnt und wird in dieser Form auch Verwendung in der zu entwerfenden Controller-Software für die Regelung finden. Weiterhin wurden gemessene Daten für das Sensorsignal U_{Diff} verwendet, diese werden zu Beginn des Skriptes eingelesen.

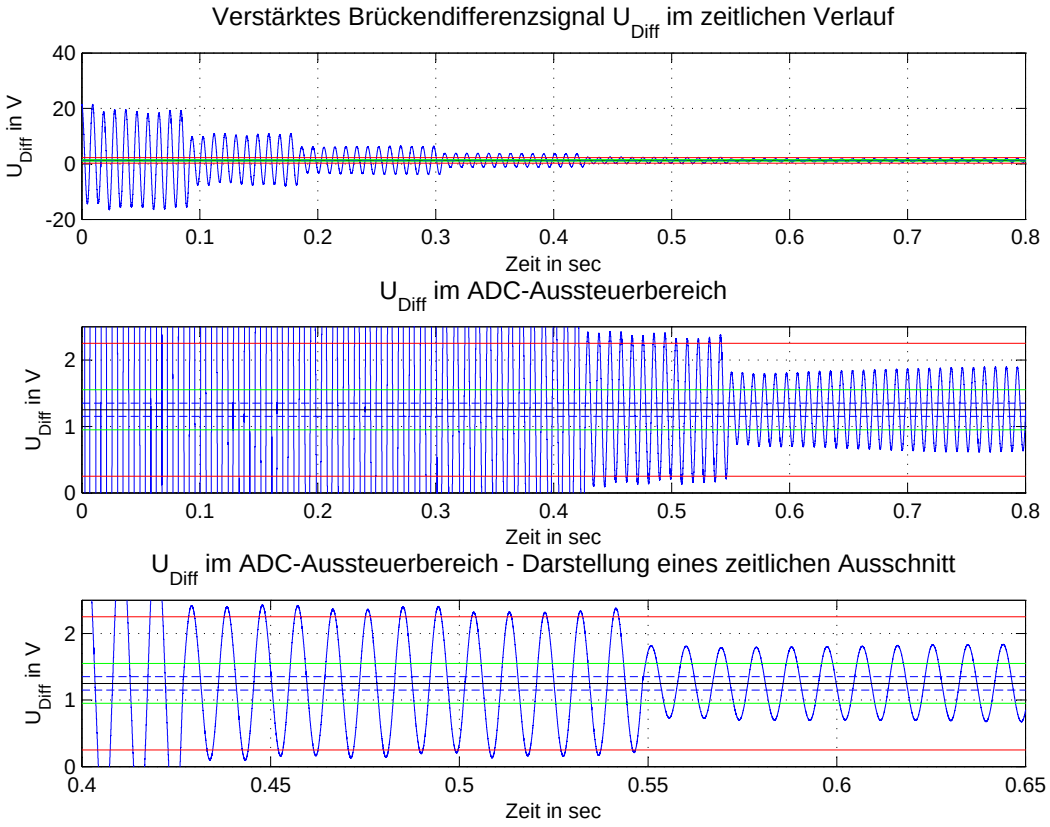


Abbildung 2.12: *Modellierung des Regelverfahrens in Matlab*

In Abbildung 2.12 ist das Ergebnis der Simulation dargestellt. Das Sensorsignal wird zu Beginn maximal verstärkt. Daraufhin wird das Offset kompensiert und die Überschreitung der Verstärkungs-Schwellwerte überprüft. Bei zu großer Verstärkung wird der Verstärkungsfaktor halbiert. Die roten Linien stehen für die Schwellen $\pm GAIN_{Max}$, die grünen für $\pm GAIN_{Min}$ und die gestrichelten blauen für die Hystereschwellen $\pm HYST$ des „Smart-Comparators“. Dieses Wechselspiel aus Offsetkompensation und Verstärkungsumschaltung wird solange durchgeführt, bis das

Signal innerhalb des zulässigen Bereiches liegt. Die obere Grafik zeigt das theoretische Signal am Ausgang der Verstärkung. In der Mitte wurde die Darstellung auf den Aussteuerbereich des verwendeten Analog-Digital-Converters $(0\ldots2{,}5V)$ beschränkt. Der untere Teil zeigt den zeitlichen Auschnitt, in dem das Signal von der Übersteuerung in den gültigen Bereich gebracht wird.

Der Quellcode dieses Matlab-Modells befindet sich in Anhang B.2.

2.5 Notwendige Abtastfrequenz zur Verstärkungsregelung

Die Verstärkungsregelung soll, wie in 2.3 beschrieben, mittels Abtastung des Signals realisiert werden. Damit eine Überschreitung der oberen Schwellwerte $\pm GAIN_{Max}$ sicher erkannt werden kann, ist eine Mindest-Abtastfrequenz erforderlich, die an dieser Stelle berechnet wird. Dazu wird ein Toleranzbereich für die zulässige Überschreitung des Grenzwertes eingeführt. Innerhalb dieses Bereiches muss mindestens ein Abtastwert genommen werden um die Überschreitung feststellen zu können.

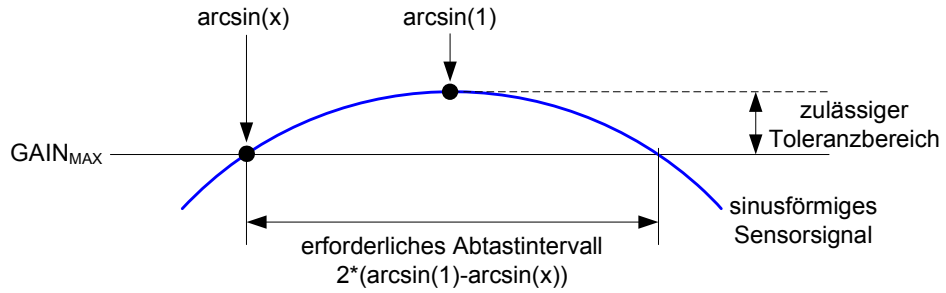


Abbildung 2.13: *Ausschnitt des Sensorsignals im Maximum*

Für eine maximale Signalfrequenz von $f_{Signal} = 2{,}5kHz$ berechnet sich die mindestens notwendige Abtastfrequenz zu:

$$f_{Sample} > \frac{2\pi \cdot f_{Sample}}{2 \cdot (arcsin(1) - arcsin(x))} = \frac{2\pi \cdot 2{,}5kHz}{2 \cdot (\frac{\pi}{2} - arcsin(x))} \tag{2.14}$$

Diese Rechnung wurde exemplarisch für drei mögliche Toleranzen durchgeführt, die Ergebnisse befinden sich in Tabelle 2.1. Der Parameter x steht dabei für die Lage der Schwelle $GAIN_{Max}$ zum Maximum des Signals. Ein Wert von $x = 0{,}95$ bedeutet beispielsweise, dass der Schwellwert bei 95% des Signalmaximums liegt.

Der ADC des verwendeten Mikrocontrollers MSP430F1611 (auf die Auswahl dieses Controllers wird in 3.3.2 eingegangen) kann im *Multiple Sample and Convert*-Modus

x	f_{Sample}
0,90	$17{,}4kHz$
0,95	$24{,}7kHz$
0,98	$39{,}2kHz$

Tabelle 2.1: *Exemplarische Berechnung der notwendigen Abtastfrequenz*

betrieben werden. Dadurch geschieht die Abtastung und Umwandlung mit maximal möglicher Geschwindigkeit ohne einer notwendigen Triggerung in Software (freilaufender Betrieb). Um diesen Betrieb des ADCs zu aktivieren muss das `MSC`-Bit im `ADC12CTL0`-Register des Controllers gesetzt werden. Die Abtastfrequenz im MSC-Modus wurde in einem Vorversuch durch das Toggeln eines Portpins in der zugehörigen Interrupt-Service-Routine gemessen, sie beträgt $f_{Sample} = 77{,}1kHz$ (unter Verwendung des internen Oszillators mit einer Taktfrequenz von $f_{Clock} = 4{,}9MHz$). Die Abtastgeschwindigkeit des Controllers reicht also aus, um die Überschreitung der Schwellwerte $\pm GAIN_{Max}$ auch bei einer maximaler Signalfrequenz von 2,5kHz sicher zu erkennen. Zur Verdeutlichung ist in Abbildung 2.14 ein Sinussignal mit einer Frequenz von 2,5kHz (normiert auf 2π) sowie den Abtastwerten (mit $f_{Sample} = 77{,}1kHz$) dargestellt. Die eingezeichnete Schwelle liegt bei 95% des Signal-Spitzenwertes, die Überschreitung wird in diesem Falle durch insgesamt 3 Samples festgestellt.

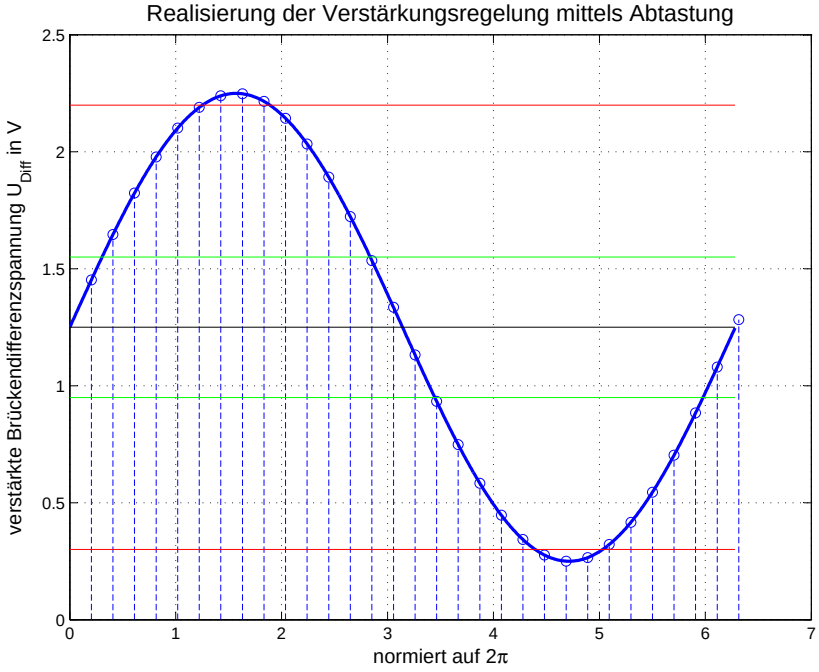


Abbildung 2.14: *Umsetzung der Verstärkungsregelung mittels Abtastung des Sensorsignals*

3 Umsetzung

3.1 Konzept des Aufbaus

Die Implementation von Offsetkompensation und Verstärkungsregelung soll auf einem eigenen Mikrocontroller erfolgen. Dadurch wird die Möglichkeit geschaffen, dass die zwei Prozesse „Sensordiagnose“ und „Regelung des Signals“ parallel ablaufen. Für diesen parallel arbeitenden Regelcontroller muss eine Hardware entwickelt werden, die sich in das bestehende System integrieren lässt.

In Abbildung 3.1 ist auf der linken Seite die bisherige Plattform, bestehend aus Hauptplatine, Verstärkerplatine sowie optionaler Displayplatine dargestellt. Eine geeignete Lösung für die Integration des Regelcontrollers in das System ist eine weitere Platine, die sogenannte Regelplatine (siehe Abbildung 3.1, rechte Seite).

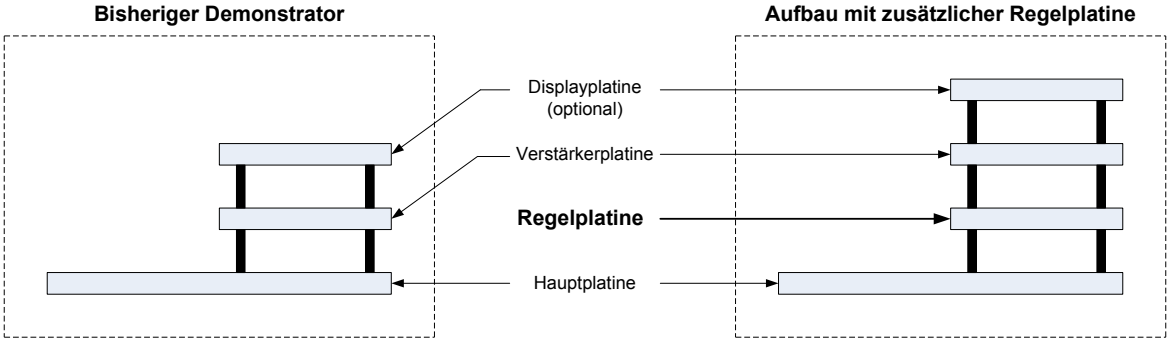


Abbildung 3.1: *Schematischer Aufbau Demonstrator mit Regelplatine*

Im Verlaufe dieses Kapitels wird auf Entwurf und Realisierung der Regelplatine sowie auf die zugehörige Software für den Regelcontroller eingegangen.

3.2 Systemarchitektur

Das Blockschaltbild in Abbildung 3.2 zeigt den Aufbau der Demonstrator-Plattform mit zusätzlicher Regelplatine. Die verstärkten Sensorsignale U_{Diff},

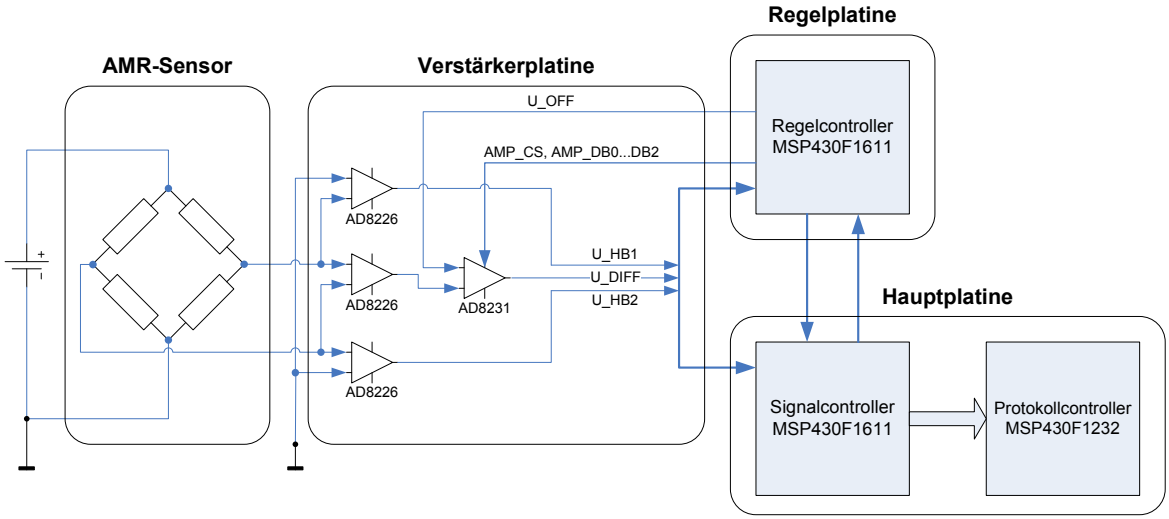


Abbildung 3.2: *Blockschaltbild der Systemarchitektur*

U_{Hb1}, U_{Hb2} gelangen sowohl an den Regelcontroller, wie auch weiterhin an den Signalcontroller. Der Regelcontroller steuert über einen internen Digital-Analog-Converter (DAC) die Offset-Spannung des Instrumentenverstärkers AD8231. Des Weiteren wird auch die Verstärkung des AD8231 mit vier digitalen Datenleitungen eingestellt. Der Signalcontroller auf der Hauptplatine erhält ebenfalls die drei Sensorsignale U_{Diff}, U_{Hb1} und U_{Hb2}, welche er für die Sensordiagnosefunktion benötigt. Im Gegensatz zum bestehenden Demonstratorsystem sind diese Signale durch den Regelcontroller aber bereits offsetfrei und so verstärkt, dass eine möglichst gute Nutzung der Dynamik der Analog-Digital-Umsetzung gewährleistet ist.

Durch die Integration des Regelcontrollers in das System wird eine Kommunikation zwischen diesem und dem Signalcontroller unbedingt erforderlich. Eine Diagnose des Sensorsignals während einer aktiven Regelung muss entweder vermieden oder als nicht gültig markiert werden. Außerdem muss der Signalcontroller den aktuell eingestellten Verstärkungsfaktor kennen. Für die Übertragung der acht möglichen Verstärkungsfaktoren (siehe Abschnitt 1.2.2) werden drei Leitungen benötigt. Eine weitere Leitung dient zur Signalisierung eines Regeleingriffes an den Signalcontroller. Insgesamt sind für die Kommunikation zwischen Regel- und Signalcontroller also vier Leitungen erforderlich.

3.3 Hardwareentwurf

3.3.1 Signalfluss zwischen den Platinen

Das zu entwickelnde Controllersystem soll, wie in Abschnitt 3.2 beschrieben, in den bisherigen Aufbau integriert werden. Bisher wurden Offset und Verstärkung vom Signalcontroller gesteuert, was nun auf den Regelcontroller verlagert werden soll. Die Sensorsignale müssen aber an beiden Controllern zur Verfügung stehen. Des Weiteren müssen Kommunikationsleitungen zwischen den Controllern vorgesehen werden. Zum besseren Verständnis ist in Abbildung 3.3 der bisherige Aufbau mit den Steckverbindern zwischen Haupt- und Verstärkerplatine und den Signalen, die über diese Verbindungen geführt werden, dargestellt.

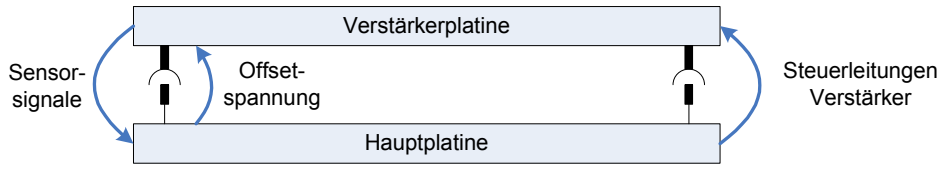


Abbildung 3.3: *Signalfluss zwischen Haupt- und Verstärkerplatine; bestehender Demonstrator*

Abbildung 3.4 zeigt den Aufbau mit integrierter Regelplatine. Die ehemaligen Verstärker-Steuerleitungen werden zur Kommunikation zwischen Regel- und Signalcontroller verwendet.

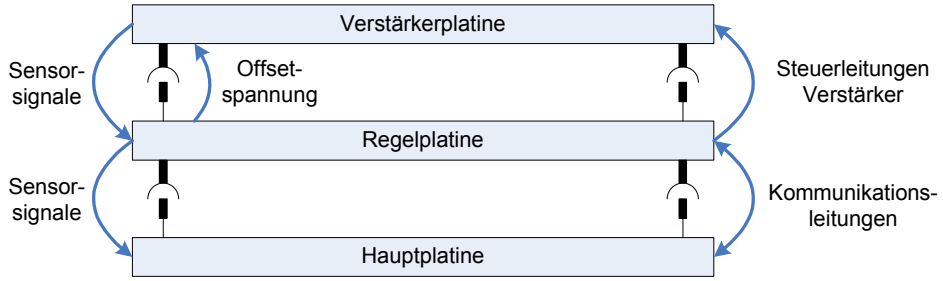


Abbildung 3.4: *Signalfluss zwischen Haupt-, Regel- und Verstärkerplatine; erweiterter Demonstrator*

In Abbildung 3.5 sind alle Signale dargestellt, welche über die Steckverbindungen zwischen den Platinen geführt werden. Die vier digitalen Steuerleitungen $A0, A1, A2, CS$ für den Instrumentenverstärker AD8231 lassen sich mittels Jumper entweder auf den Regel- oder auf den Signalcontroller schalten. Im Normalfall sollten die Jumper wie abgebildet geschlossen werden. Dennoch besteht so die Möglichkeit, den Verstärker wie zuvor vom Signalcontroller aus zu steuern. Über vier weitere Jumper lassen sich die ehemaligen Steuerleitungen mit dem Regelcontroller verbinden und so für die Kommunikation zwischen den Controllern nutzen.

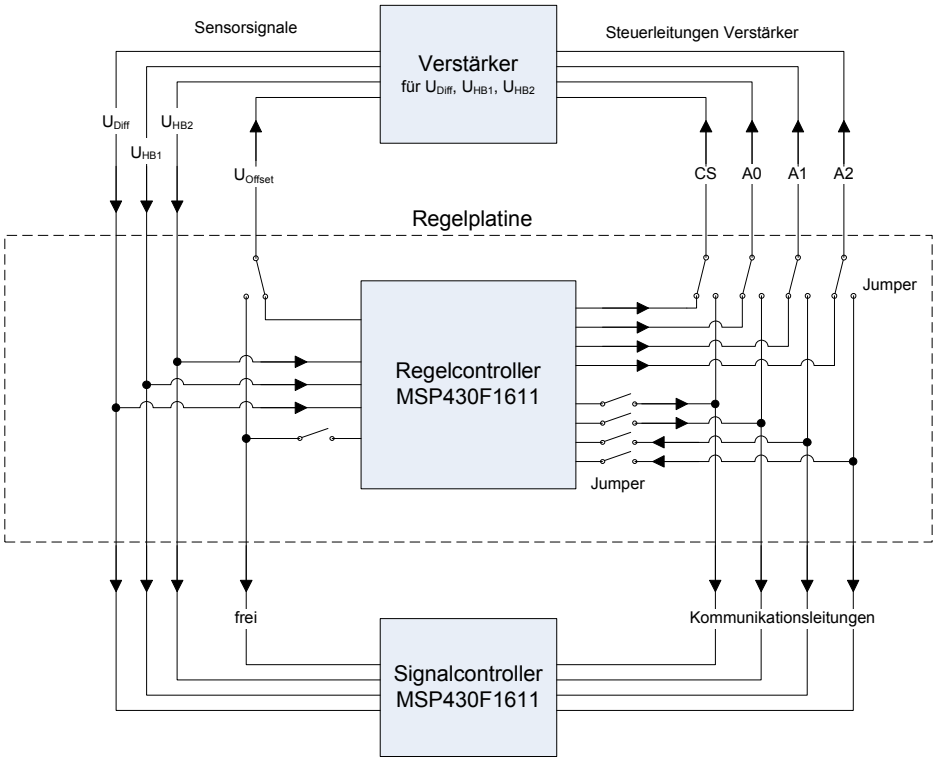


Abbildung 3.5: *Blockschaltbild Signalfluss*

Die verstärkten Sensorsignale U_{Diff}, U_{Hb1}, U_{Hb2} werden auf der Regelplatine geteilt und an beide Controller geführt. Für das Differenzsignal U_{Diff} wird eine optionale Pufferstufe vorgesehen, für den Fall, dass die Belastung durch zwei Sample-And-Hold Stufen der beiden Mikrocontroller zu groß ist und somit die Messwerte verfälscht werden. Als Quelle für die Offsetspannung U_{Off} kann wieder per Jumper der Regel- oder Signalcontroller gewählt werden. Nachfolgend wird auf den Entwurf der Regelplatine eingegangen.

3.3.2 Hardwareanforderungen

Für die Erfüllung der Aufgabenstellung wird ein Mikrocontroller mit mindestens folgender Austattung benötigt.

- 1x ADC mit mindestens 3 Kanälen für die Signale U_{Diff}, U_{Hb1}, U_{Hb2}
- 2x DAC zur Ausgabe von Offsetkompensations-Spannung und „Smart-Comparator“-Schwellwert
- 1x analoger Komparator zur Realisierung des „Smart-Comparators“

- 1x UART-Schnittstelle für Debug-Zwecke
- 2x Timer
- 30 I/O-Pins für Displayplatine, Verstärkersteuerung, Kommunikationsleitungen, LEDs, Jumper

Ausgewählt wurde der *MSP430F1611* von *Texas Instruments*. Dieser stellt die benötigte Ausstattung zur Verfügung, was andere Contoller allerdings auch leisten könnten. Für den *MSP430F1611* gibt es zwei weitere Gründe, die ihn für diese Aufgabenstellung als geeignet erscheinen lassen. Zum einen wurden im bisherigen Verlauf des Forschungsprojektes „ESZ-ABS“ bereits Arbeiten auf Basis der MSP430-Familie durchgeführt, so dass Entwicklungsumgebung, Programmier-Hardware sowie Know-How zur Verfügung stehen. Zum anderen ist der Signalcontroller auf der Hauptplatine ebenfalls ein *MSP430F1611*. Somit können eventuell Teile der Firmware auf den Regelcontroller übernommen werden, was den Entwicklungsaufwand verringert.

Des Weiteren sollte die Integration der Regelplatine in die bestehende Demonstrator-Plattform möglichst flexibel ausgelegt werden. Dazu zählen beispielsweise die in Abschnitt 3.3.1 beschriebenen Möglichkeiten, den Signalfluss zwischen den einzelnen Platinen zu steuern. Dadurch werden Experimente mit unterschiedlicher Aufgabenverteilung zwischen den Mikrocontrollern ermöglicht.

3.3.3 Schaltungsentwurf

Die entworfene Regelplatine hat die gleichen Abmessungen wie die Verstärkerplatine. Damit sie zwischen Haupt- und Verstärkerplatine gesteckt werden kann, wurden sowohl auf Löt- und Bestückungsseite SMD[1]-Steckverbinder vorgesehen. An dieser Stelle soll kurz auf die Baugruppen der Regelplatine eingegangen werden. Eine Übersicht ist ein Abbildung 3.6 dargestellt.

In der Mitte befindet sich der Regelcontroller *MSP430F1611*. Für die Anbindungen der einzelnen Baugruppen wurde eine Pinbelegungs-Liste erstellt, diese befindet sich in Anhang A.1. Die Versorgungsspannung wird aus den Steckverbindern zur Hauptplatine bezogen. Für die Analog-Digital-Umsetzung wurde ein extra Spannungsregler *LP2985* von National Semiconductor vorgesehen, um die durch steile Schaltflanken verursachten Störungen auf der digitalen Versorgungsspannung vom empfindlichen Analogteil fernzuhalten. Dazu wurde im linken unteren Bereich der Platine ein Areal mit seperater analoger Masse vorgesehen. In diesem

[1] Surface-Mount-Device, Bauteile für die Montage auf der Platinenoberfläche

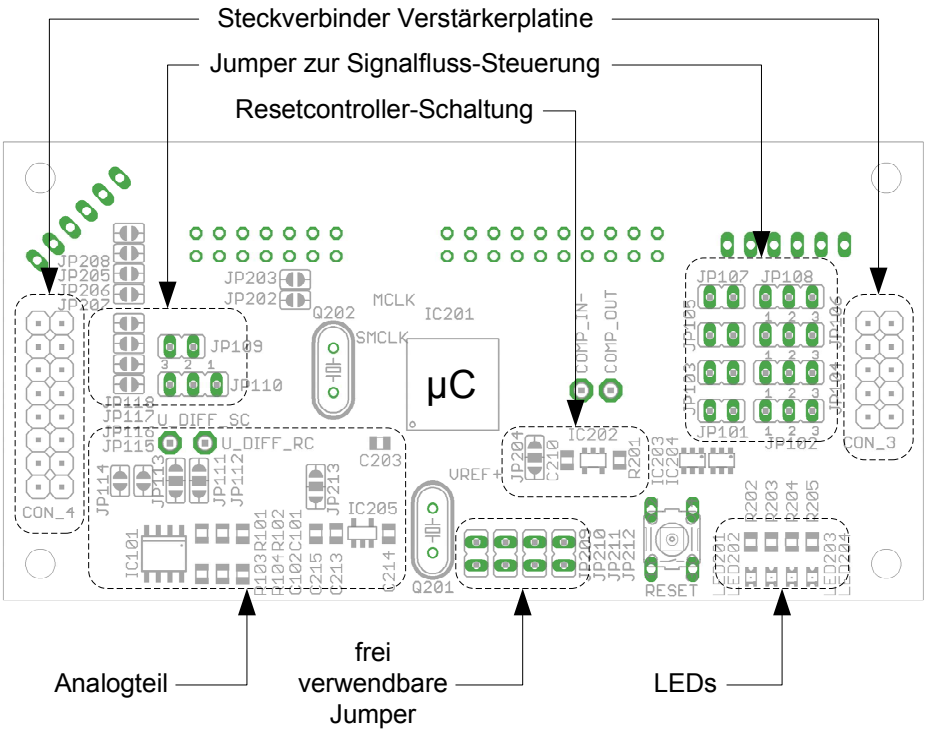


Abbildung 3.6: *Baugruppen der Regelplatine, Bestückungsseite*

befinden sich neben dem erwähnten Spannungsregler auch die analogen Signalleitungen für Brückendifferenzspannung, Halbbrückensignale und Offsetspannung. Diese Signale können über Jumper auf Regel- oder Signalcontroller geroutet werden. Für die Brückendifferenzspannung wurde eine optionale Pufferstufe vorgesehen. Diese besteht aus dem IC *AD8606* von Analog Devices. Es handelt sich dabei um zwei Opeartionsverstärker in einem Gehäuse, die beide als Impedanzwandler verschaltet wurden. Der eine Ausgang kann auf den Regelcontroller, der andere auf den Signalcontroller geroutet werden.

Weiterhin vefügt der Mikrocontroller über eine JTAG-Schnittstelle, eine UART-Schnittstelle und eine Schnittstelle zum Anschluss der Displayplatine. Diese Steckverbinder wurden so platziert, dass der Anschluss von Kabeln auch im vollständig aufgebauten Demonstrator-System möglich ist. Für einen einen sicheren Anlauf des Controllers beim „PowerOn“ oder bei externem Reset wurde ein Reset-Controller IC vom Typ *STM6717* von ST Microelectronics vorgesehen. Weiterhin befinden sich am unteren Rand der Platine vier frei verwendbare Jumper, welche mit Portpins des Mikrocontrollers verbunden sind. Rechts unten befinden sich vier LEDs, die über Buffer Gates *SN74LVC2G34* von Texas Instruments angesteuert werden, da der Controller selbst den notwendigen Strom nicht treiben kann.

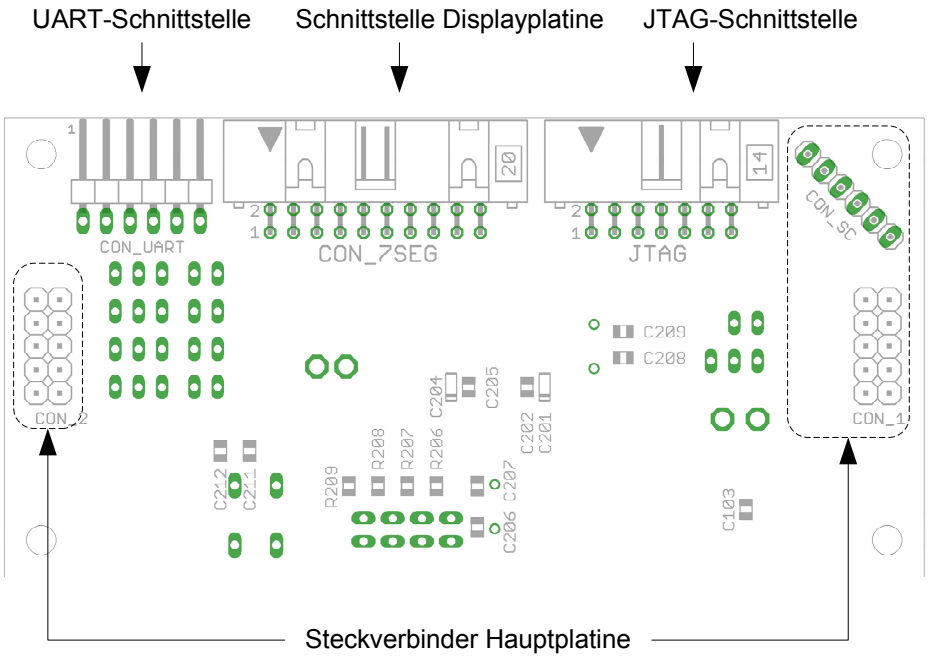


Abbildung 3.7: *Baugruppen der Regelplatine, Lötseite*

In den Anhängen A.2 . . . A.5 befinden sich Schaltplan, Layout, Bestückungsübersicht und Stückliste der Regelplatine.

3.4 Softwareentwurf

3.4.1 Grundfunktionen

3.4.1.1 Nulldurchgangserkennung

Die grundlegende Funktion eines ABS-Sensors ist die Erkennung der Nulldurchgänge des Sensorsignals U_{Diff}. Diese Erkennung erfolgt wie beschrieben durch die Nutzung eines „Smart-Comparators“. Auf dem Mikrocontroller löst jeder Nulldurchgang von U_{Diff} einen Schaltvorgang des Komparators und damit eine Interruptanforderung aus, wodurch die zugehörigen Interrupt-Service-Routine (ISR) `interrupt(COMPARATORA_VECTOR) isr_comp_a(void)` ausgeführt wird. Die weiteren Funktionen, die nun ausgeführt werden sind abhängig von den Zuständen zweier Automaten, die im Folgenden erläutert werden.

3.4.1.2 Stillstanderkennung

Eine ebenfalls grundlegende Funktion für einen ABS-Sensor ist die Erkennung eines stillstehenden Encoderrades. Dazu wird der Timer A des Mikrocontrollers bei jedem zweiten Nulldurchgang des Signals U_{Diff} (Anfang einer Periode) neu gestartet. Solange das Rad in Bewegung ist, erfolgt also stehts ein Neustart des Timers. Bei einem Stillstand wird der Timer nicht mehr neugestartet und läuft über. Dieser Überlauf fordert einen Interrupt an. In der Interrupt-Service-Routine des Timers werden die nachfolgend erläuterten Automaten zurückgesetzt, damit das System bei drehendem Encoderrad wieder anlaufen kann.

3.4.2 Zustandsautomaten

3.4.2.1 Übergeordneter Zustandsautomat

Für die Erfüllung der Aufgabenstellung eignet sich ein automatengesteuertes Softwarekonzept. In Anlehnung an das Datenblatt [15] gibt es einen übergeordneten Zustandsautomaten, der die folgenden Zustände durchläuft:

1. Initial_0Hz
2. Initial_1Hz
3. Active

Beim Systemstart („PowerOn“) gelangt der Automat auf dem Regelcontroller stets in den Zustand *Initial_0Hz*. In diesem Zustand wird auf eine Drehung des Encoderrades gewartet. Sobald die Drehung erkannt wurde, wechselt der Automat in den Zustand *Initial_1Hz*. Dort wird bei drehendem Encoderrad sowohl das Offset des Sensorsignals kompensiert, als auch die Verstärkung geschaltet. Dies geschieht immer im Wechselspiel. Es wird zunächst mit maximaler Verstärkung gestartet, damit unter allen Umständen die Nulldurchgänge des Signals erkannt werden. Dann wird das Tastverhältnis auf $\delta = 50\%$ geregelt, wodurch das Offset kompensiert wird, und anschließend die Verstärkung überprüft. Sollte diese zu groß sein, wird der Verstärkungsfaktor halbiert. Anschließend wird erneut das Offset kompensiert und wieder die Verstärkung überprüft. Sobald dieser Vorgang abgeschlossen ist, liegt ein eingeregeltes Sensorsignal vor und es wird in den Zustand *Active* gewechselt. Nun kann der Signalcontroller mit der Sensordiagnose beginnen, während der Regelcontroller weiterhin das Signal überprüft und bei Bedarf nachregelt. In Tabelle 3.1 sind die drei Sensorzustände mit ihren Eigenschaften zusammengefasst.

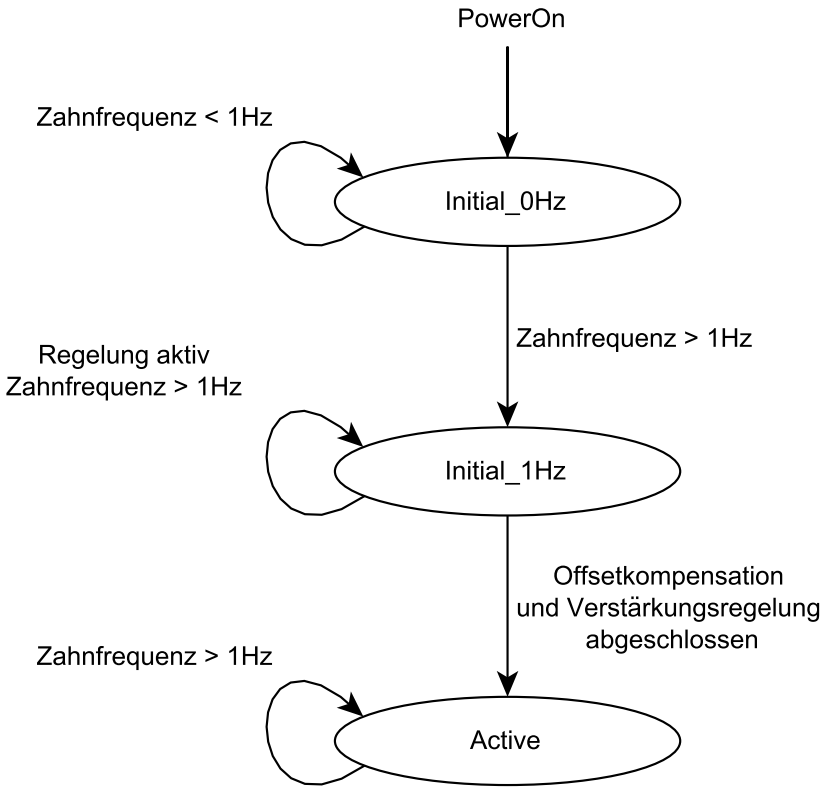


Abbildung 3.8: *Zustandsdiagramm des übergeordneten Automaten*

Zustand	Funktion	Regelung	Diagnose
Initial_0Hz	Warten auf Drehung des Encoderrades	keine Regelung	keine Diagnose
Initial_1Hz	Initiale Regelung des Signals	schnelle Regelung zur schnellen Sicherung der Sensorgrundfunktionen	Diagnose wartet auf eingeregeltes Signal
Active	Regulärer Betrieb des Sensors	langsame Regelung zum Ausgleich von Offset- und Amplituden- Drifts	Diagnose arbeitet

Tabelle 3.1: *Zustandseigenschaften des übergeordneten Automaten*

3.4.2.2 Untergeordneter Zustandsautomat

Zur Ablaufsteuerung der Offsetkompensation und Verstärkungsregelung, sowie zur Erkennung der Nulldurchgänge wird ein zweiter untergeordneter Zustandautomat eingeführt. Dieser kann, je nachdem in welchem Zustand sich der übergeordnete Automat befindet, die in Tabelle 3.2 aufgeführten Zustände durchlaufen:

Im übergeordneten Zustand *Initial_0Hz*, in dem auf eine Drehung des Encoder-

Zustand	Funktion des Zustandes
Serach_S1	Warten auf +*HYST*, Zählerstand sichern, Timer A neustarten
Search_S2	Warten auf -*HYST* (Periodenmitte), Zählerstand sichern
Search_S1_Calc	Tastverhältnis bestimmen und Offsetkorrektur durchführen
Search_S2_Pre	Warten auf -*HYST*, um aktuelle Periode zu beenden
Search_S1_Check_Gain	Abtastung der oberen Halbwelle
Search_S2_Check_Gain	Untere Halbwelle abtasten, ggf. Verstärkungsfaktor einstellen

Tabelle 3.2: *Zustände im untergeordneten Automaten*

rades gewartet wird, kann der untergeordnete Automat lediglich die Zustände *Search_S1* und *Search_S2* annehmen (siehe Abbildung 3.9). Im Zustand *Search_S1* wird darauf gewartet, dass das Sensorsignal U_{Diff} die obere Hystereseschwelle $+HYST$ passiert (Periodenanfang). Dadurch schaltet der Komparator im Regelcontroller, was eine Interruptanforderung zur Folge hat. Im zugehörigen Interrupthandler wird nun die Schaltschwelle des Komparators für die Erkennung des nächsten Nulldurchgangs auf $-HYST$ umgeschaltet und der Folgezustand auf *Search_S2* gesetzt. Außerdem wird der Timer A für die Stillstandserkennung gestartet. Der nächste Nulldurchgang stellt die Mitte der Signalperiode dar. Im Interrupthandler des Komparators wird die Schaltschwelle für die Erkennung des nächsten Periodenanfanges wieder zuück auf $+HYST$ gesetzt. Es handelt sich dabei um den sogenannten „Smart-Comparator“ wie in Abschnitt 2.3 erläutert. Da sich das Encoderrad jetzt in Bewegung befindet, wird der Folgezustand des übergeordneten Automaten auf *Initial_1Hz*, der Folgezustand des untergeordneten Automaten wieder zurück auf *Search_S1* gesetzt.

Im übergeordneten Zustand *Initial_1Hz* ist der Ablauf komplexer. Zunächst werden wie zuvor die Zustände *Search_S1* und *Search_S2* durchlaufen. In *Search_S2* (Periodenmitte) wird nun zusätzlich der Zählerstand des Timer A in der Variablen `g_sig.tp` gespeichert, der Folgezustand ist *Search_S1_Calc*. Mit dem nächsten Nulldurchgang wird vor dem Zurücksetzten des Timer A der Zählerstand in der Variablen `g_sig.tt` gespeichert. Anhand dieser beiden Zählerstände kann das Tastverhältnis nach der Formel

$$\delta = \frac{g\_sig.tp}{g\_sig.tt} \tag{3.1}$$

berechnet werden. Nun wird die Funktion `calc_offset()` aufgerufen, in der gegebenenfalls ein Offset-Korrekturfaktor bestimmt wird. Dieser Korrekturfaktor wird auf 12-Bit Registerbreite des DACs skaliert und auf diesem ausgegeben. Dadurch wird eine neue Offsetspannung am einstellbaren Instrumentenverstärker der Verstärkerplatine angelegt. Anschließend wird der Folgezustand des Automaten auf *Search_S2_Pre* gesetzt, von dort aus gelangt der Automat wieder zu *Search_S1*.

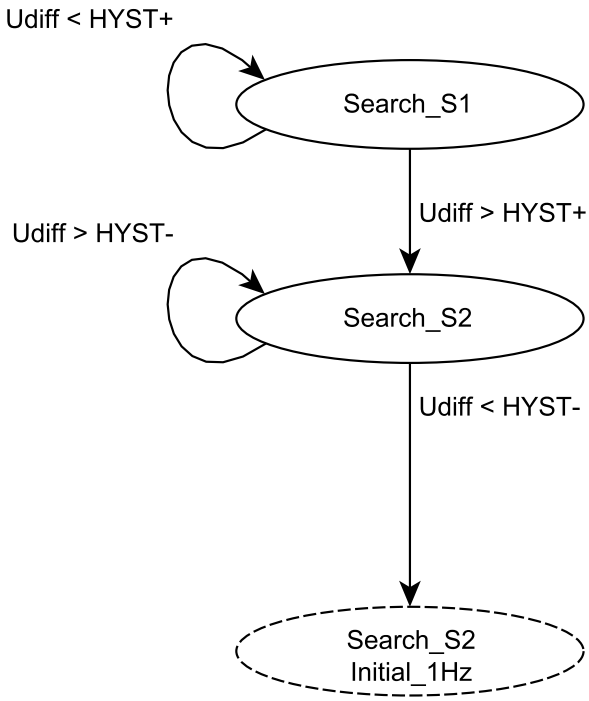


Abbildung 3.9: *Zustandsdiagramm des untergeordneten Automaten im übergeordneten Zustand Initial_0Hz*

Dieser Vorgang wiederholt sich insgesamt 6 Mal, wodurch eine sukzessive Annäherung an ein optimales Tastverhältnis von $\delta = 50\%$ erfolgt.

Nach Abschluss der Offsetkompensation wird der Analog-Digital-Converter (ADC) im Mikrocontroller gestartet und der Zustand *Search_S2_Check_Gain* wird erreicht. Der Folgezustand ist nun *Search_S1_Check_Gain*. Seit dem Start des ADCs sind zwei Nulldurchgänge, also eine Periode des Sensorsignals erfolgt. Während dieser Zeit hat der ADC das Signal abgetastet und jeder Abtastwert wurde auf die Schwellen $\pm GAIN_{Min}$ sowie $\pm GAIN_{Max}$ überprüft. Für jede Schwelle ist eine Zählvariable `g_sig.g1passed` ... `g_sig.g4passed` vorgesehen, die beim Überschreiten inkrementiert wird. Diese Überprüfung erfolgt in der Interrupt-Service-Routine des ADC, die Abtastwerte selbst werden nicht gespeichert. Mittels der Funktion `check_gain()` wird die Überschreitung der Verstärkungsschwellen ausgewertet und daraufhin der Folgezustand festgelegt. Falls die Schwellen $\pm GAIN_{Max}$ überschritten wurden, ist das Signal zu stark verstärkt. Der Verstärkungsfaktor wird daraufhin halbiert und der Folgezustand auf *Search_S2_Pre* festgelegt. Somit wird die Offestkompensation und anschließend die Überprüfung der Verstärkung erneut durchlaufen und zwar solange bis die Grenzwerte $\pm GAIN_{Max}$ nicht mehr überschritten wurden. Das Sensorsignal gilt dann also korrekt eingeregelt und der übergeordnete Automat wechselt in den Folgezustand *Active*.

Der Ablauf im aktiven Zustand ist ähnlich dem gerade beschriebenen. In Abildung 3.10 sind die Automatengraphen für die übergeordneten Zustände *Initial_1Hz* (linke Seite) und *Active* (rechte Seite) zum Vergleich nebeneinander gestellt. Der wesentliche Unterschied ist der, dass im aktiven Zustand zunächst eine Überwachung des Sensorsignals erfolgt. Es wird kontinuierlich das Tastverhältnis berechnet und anschließend die Verstärkung überprüft. Solange der Tastgrad bei $\delta = 50\%$ und die Amplitude zwischen minimaler und maximaler Verstärkungsschwelle liegt, findet kein weiterer Eingriff des Regelcontrollers in das Sensorsignal statt. Erst bei Verletzung einer oder beider Bedingungen, beispielsweise durch Veränderung des Luftspaltes, oder die Offsetverschiebung aufgrund von Temperaturschwankungen oder Wirbelströmen bei hohen Drehzahlen, greift die Regelung ein und korrigiert die Abweichung. Dies wird dem Signalcontroller über eine Kommunikationsleitung signalisiert. Im regulären Betrieb des Sensors wird der Zustand *Active* nicht mehr verlassen, mit Ausnahme eines stillstehenden Encoderrades.

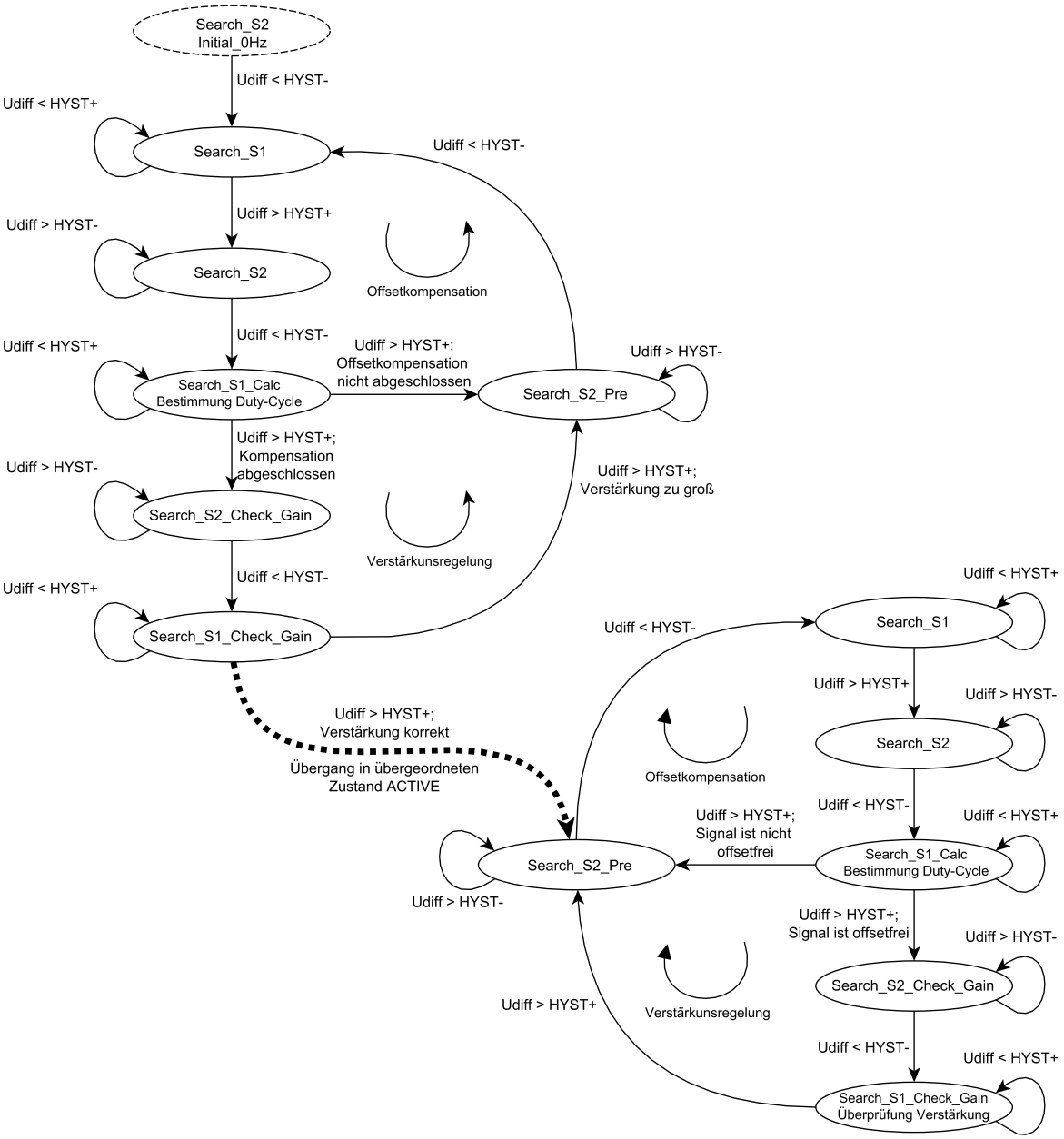


Abbildung 3.10: *Zustandsdiagramm des untergeordneten Automaten im übergeordneten Zustand Initial_1Hz (linker Teil) und Active (rechter Teil)*

3.4.3 Prioritäten

In diesem Abschnitt wird auf die Vergabe von Prioritäten für einzelne Aufgaben des Demonstratorsystems eingegangen. Wie in Abbildung 3.11 ersichtlich, ist die Erkennung der Nulldurchgänge – und damit die Ausgabe der sogenannten „Speed-Pulse“ an das ABS-Steuergerät – von größter Wichtigkeit. Es muss unter allen Umständen gewährleistet sein, dass ein Nulldurchgang des Sensorsignals U_{Diff} erkannt wird. An zweiter Stelle steht die Regelung. Dazu zählt die Kompensation des Offsets, sowie die korrekte Verstärkung des Signals. Dies ist für die Sensordiagnosefunktion von großer Bedeutung, ein übersteuertes Signal würde beispielsweise zu einer falschen Klirrfaktorberechnung führen. Die Regelung muss daher eine höhere Priorität als die Diagnose besitzen und in der Lage sein, letztere zu unterbrechen, sobald die Regelung eingreift. An unterster Stelle der Prioritätenliste stehen die Debugfunktionen. Dazu zählt die Ausgabe von Daten auf der Displayplatine und der seriellen Schnittstelle.

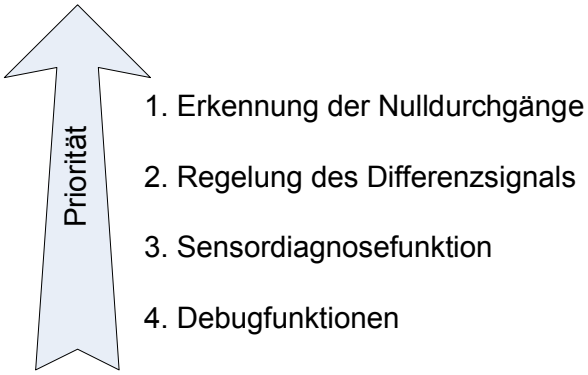


Abbildung 3.11: *Vergabe der Prioritäten*

3.4.4 Kommunikation zwischen Regel- und Signalcontroller

Wie zuvor beschrieben, muss dem Regelcontroller eine Möglichkeit gegeben werden, den Signalcontroller entweder zu unterbrechen oder die aktuelle Berechnung als nicht gültig zu markieren. Dazu wird eine Leitung (im Folgenden COMM_5 genannt) zwischen den beiden Controllern verwendet. Im Zustand *Initial_1Hz* wartet der Signalcontroller zunächst mit der Diagnose, bis das Sensorsignal korrekt eingeregelt wurde. Während des Regelvorganges zieht der Regelcontroller die Leitung auf „High-Pegel“. Sobald auf der Leitung COMM_5 „Low-Pegel“ anliegt, ist die initiale Regelung abgeschlossen und beide Controller wechseln in den übergeordneten Zustand *Active*.

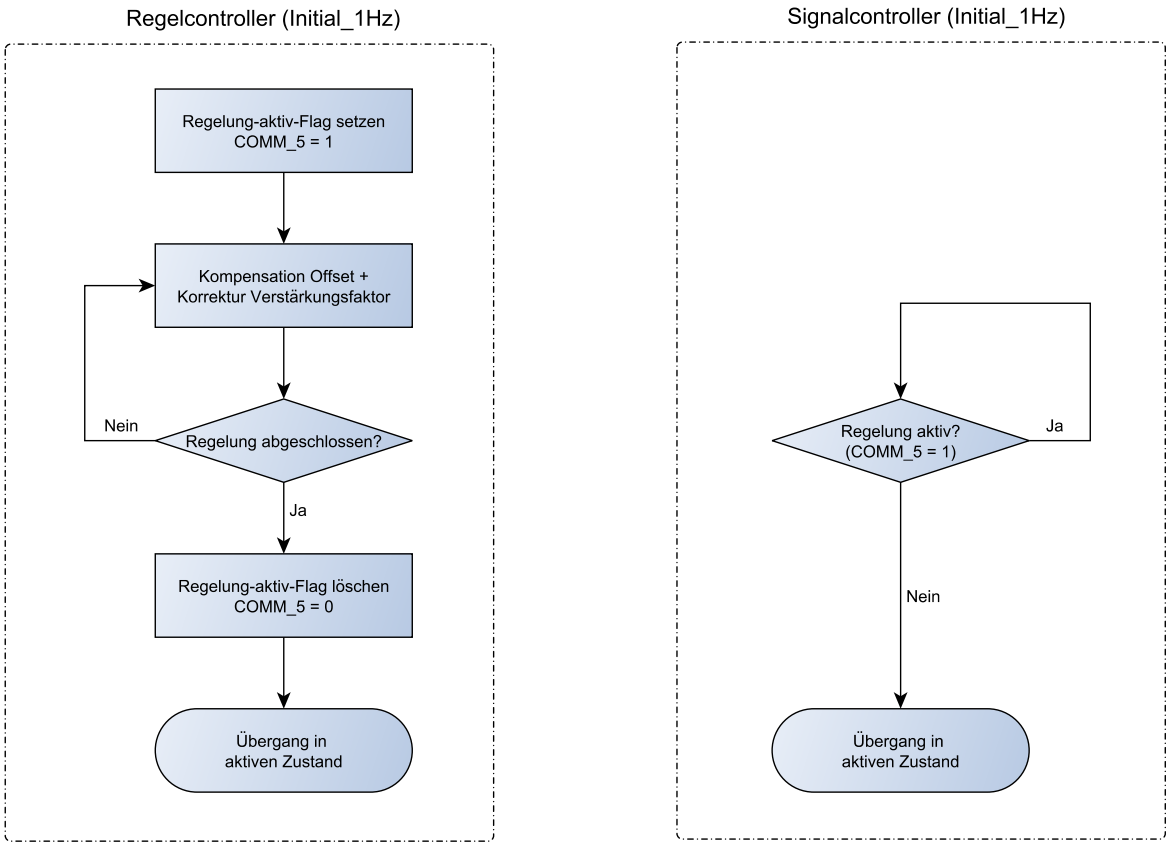


Abbildung 3.12: *Flussdiagramm Kommunikation zwischen Regel- und Signalcontroller im Zustand Initial_1Hz*

Im aktiven Zustand laufen nun die beiden Prozesse „Überwachung und gegebenenfalls Regelung des Sensorsignals“ sowie „Durchführung der Diagnoseberechnungen“ parallel ab. Da die Regelung gegenüber der Diagnoseberechnung eine höhere Priorität aufweist, wird die Leitung COMM_5 so gewählt, dass diese mit einem Pin des interrupt-fähigen Port_1 des Signalcontrollers verbunden ist. Somit kann auf diesem eine Interruptanforderung vom Regelcontroller erzeugt werden. In der zugehörigen Interrupt-Service-Routine wird das Flag `g_calc.not_valid` gesetzt. Anhand dieses Flags kann der Signalcontroller feststellen, dass innerhalb der letzten Berechnung ein Regeleingriff stattfand und das Ergebnis somit verwerfen oder als nicht gültig markieren. Am Ende einer Berechnung wird stets der Zustand der Leitung COMM_5 abgefragt. Falls diese wieder auf „Low-Pegel“ liegt (die Regelung also inaktiv ist), wird das Flag zurückgesetzt. Zum besseren Verständnis dieses Ablaufs ist in Abbildung 3.13 ein Flussdiagramm dargestellt.

Im Falle eines offsetfreien und korrekt verstärkten Signals wird der Signalcontroller nicht unterbrochen und gibt somit gültige Ergebnisse der Diagnoseberechnung aus.

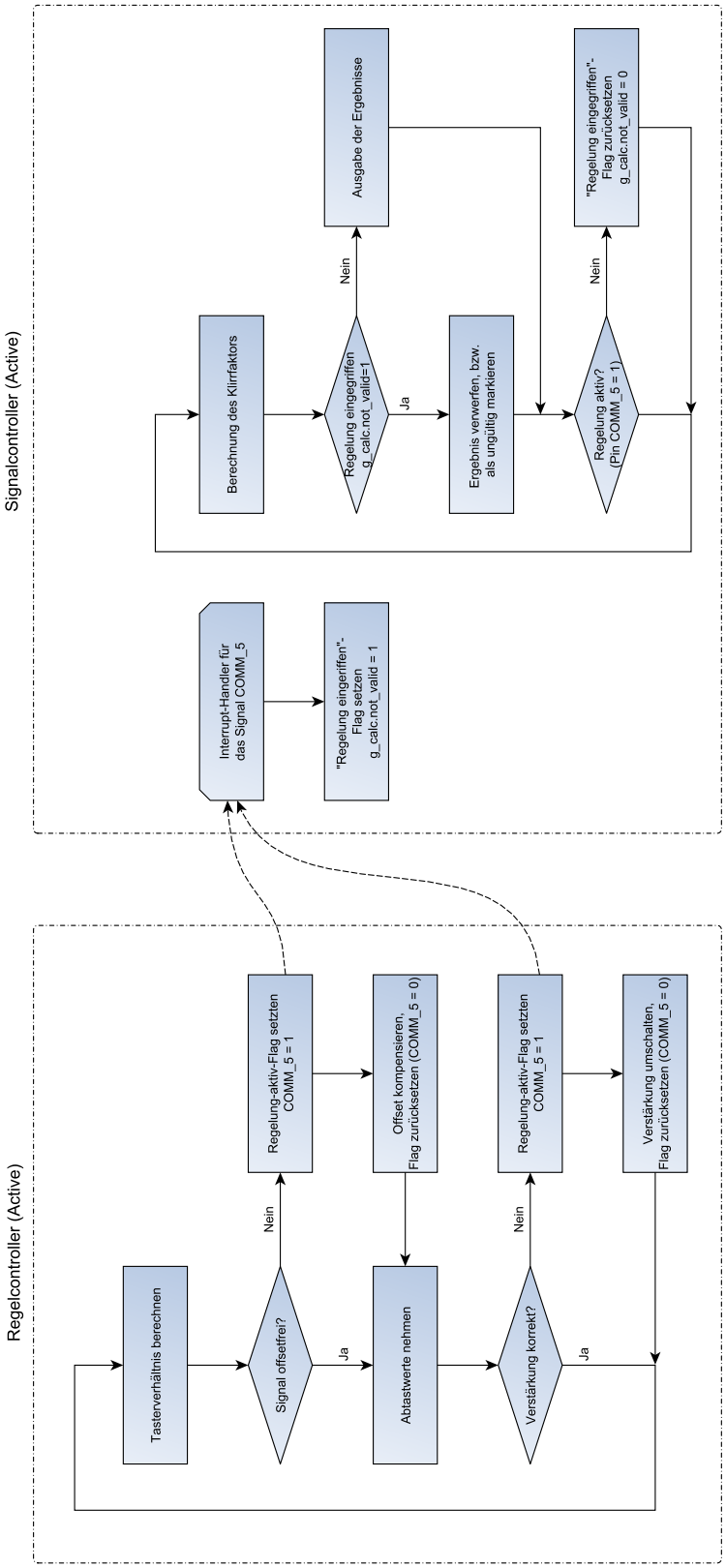


Abbildung 3.13: *Flussdiagramm Kommunikation zwischen Regel- und Signalcontroller im aktiven Zustand*

3.4.5 Grundlegende Softwarestruktur

Die Struktur der Software wird so angelegt, dass alle Aufgaben von hoher Priorität durch Interrupt-Anforderungen möglichst schnell abgearbeitet werden können. Beim „PowerOn“ des Systems werden zunächst alle notwendigen Initialisierungen der Peripherie sowie benötigter Variablen durchgeführt. Daraufhin gelangt das Programm in die für Mikrocontroller obligatorische Endlosschleife. In dieser Schleife werden lediglich Statusinformationen auf der Displayplatine ausgegeben, eine Aufgabe von unterster Priorität. Der weitere Ablauf des Programms erfolgt interruptgesteuert. Bei dem verwendetem Mikrocontroller *MSP430F1611* (siehe Abschnitt 3.3) kann allerdings kein Einfluss auf die Prioritäten einzelner Interruptquellen genommen werden. Diese sind durch die Position in der Interrupt-Vektor-Tabelle vom Hersteller festgelegt. Die zur Erfüllung der Aufgabenstellung benötigten Interruptquellen sind mit zugehörigen Prioritäten in Tabelle 3.3 aufgelistet.

Interruptquelle	**Priorität**
Comparator_A	11
ADC_12	7
Timer_A	5
USART_1 transmit	2

Tabelle 3.3: *Prioritäten der verwendeten Interruptquellen*

Die Erkennung der Nulldurchgänge besitzt höchste Priorität. Der Comparator_A eignet sich für diese Aufgabe sehr gut, da keine verwendete Interruptquelle in der Tabelle über dem Komparator steht. Zweithöchste Priorität besitzt die Regelung des Sensorsignals. Diese wird zum einen über das Tastverhältnis (Offsetkompensation), zum anderen mittels Abtastung (Verstärkungsregelung) realisiert. An dritter Stelle steht die Sensordiagnose, die vom Signalcontroller auf der Hauptplatine durchgeführt wird. Der Regelcontroller muss hierbei lediglich einen Portpin im Falle einer aktiven Regelung setzen.

Damit Änderungen an der Software leicht durchführbar sind, sollte diese möglichst modular in Form von separaten Quelldateien für zusammengehörige Funktionen aufgebaut werden. Alle Konstanten werden in eine Headerdatei in Form von Präprozessormakros ausgelagert, damit diese nur an einer Stelle im Programm angepasst werden müssen. Da diese Arbeit eine Weiterentwicklung darstellt, werden bestehende Namensgebungen für Funktionen und Variablen, sowie der Aufbau der Software möglichst beibehalten, damit das erweiterte System im Ganzen verstanden werden kann.

3.4.6 Anpassung der Signalcontroller-Software

Die Regelplatine stellt eine Erweiterung des bestehenden Demonstratorsystems dar. Die initiale Offsetkompensation sowie die Einstellung des Verstärkungsfaktors wurden bisher vom Signalcontroller durchgeührt. Der Regelcontroller soll diese Funktionen nun übernehmen sowie eine zusätzliche Signalüberprüfung und gegebenenfalls Regelung während der laufenden Diagnose durchführen. Für die Integration des Regelcontrollers in das Gesamtsystem müssen also auch Änderungen an der vorhandenen Software des Signalcontrollers durchgeführt werden.

Die folgenden Funktionen werden auf dem Signalcontroller nicht mehr benötigt und daher entfernt:

1. `void set_gain(void)` - Steuerung des digital einstellbaren Instrumentenverstärkers
2. `void calc_offset(void)` - Berechnung des Offset-Korrekturfaktors
3. Berechnung des Duty-Cycles im Zustand *Search_S1_Check_Gain*

Durch das Entfernen der drei genannten Funktionen müssen einige Anpassungen vorgenommen werden:

1. Im Zustand *Initial_1Hz* bleibt der Signalcontroller nun solange, bis der Regelcontroller die initiale Regelung durchgeführt hat. Die aktive Regelung wird durch einen Highpegel auf der Kommunikationsleitung COMM_5 signalisiert. Der Übergang in den Sensorzustand *Active* erfolgt durch einen Lowpegel an COMM_5.
2. In der Funktion `void get_gain(void)` wird der eingestellte Verstärkungsfaktor nicht mehr von den Jumpern auf der Hauptplatine, sondern von den Kommunikationsleitungen zum Regelcontroller gelesen. Die drei dazu verwendeten Datenleitungen liegen an den Portpins P1.0, P1.1, P1.2 des Signalcontrollers. Der eingestellte Verstärkunsgfaktor wird für die Übertragung der Protokollbits an den Protokollcontroller benötigt.

Die Leitung COMM_5 liegt am interruptfähigen Portpin P1.3. Jeder Regelvorgang des Regelcontrollers im aktiven Sensorzustand wird durch eine steigende Flanke auf dieser Leitung angezeigt, was eine Interruptanforderung auf dem Signalcontroller auslöst. Der folgende Quellcode wird in die Datei `isr_portx.c` eingefügt.

```
interrupt ( PORT1_VECTOR ) isr_port1 ( void ) {

if(P1IFG & PIN3) {
```

```
/* actual thd-calculation is not valid */
g_calc.not_valid = TRUE;

/* THD not vailid, for automated measurements with Matlab */
g_calc.hd = 99;

P1IFG &= ~PIN3; //reset interrupt-flag
}
...
}
```

Die im Code enthaltene Variable `g_calc.not_valid` wird neu eingeführt. Diese Variable wird gesetzt, sobald ein Regeleingriff stattgefunden hat. Weiterhin wird die Variable `g_calc.hd` auf 99 gesetzt. Dadurch wird ein aktuell ungültiges Ergebnis signalisiert, was zum einen für die Demonstration als auch zum anderen für die Durchführung von späteren Messreihen erforderlich ist. Die Ausgabe eines korrekten Ergebnisses der Klirrfaktorberechnung erfolgt erst wieder, wenn während der Berechnung kein „Regelinterrupt“ aufgetreten ist. Dazu wurden Anpassungen in der Funktion `void calc_hd(void)` vorgenommen.

Alle durchgeführten Änderungen an der Signalcontroller-Software wurden durch Kommentare an den jeweiligen Stellen deutlich gekennzeichnet. Die geänderte Software befindet sich auf der CD-ROM, welche im Anhang dieser Arbeit zu finden ist, im Verzeichnis „software_signalcontroller“.

3.5 Inbetriebnahme der Hardware

Ein Foto der fertig bestückten Regelplatine befindet sich in Abbildung 3.14. Für die Inbetriebnahme müssen im Anschluss an die Bestückung einige Jumper geschlossen werden, um den Signalfluss zwischen Verstärker-, Regel- und Hauptplatine herzustellen.

In Tabelle 3.4 sind alle Jumper aufgeführt, die für den Betrieb der Regelplatine relevant sind. Die angegebende Konfiguration ist so gewählt, dass Offsetkompensation und Verstärkungsregelung vom Regelcontroller durchgeführt werden. Messungen bei der Inbetriebnahme haben ergeben, dass die vorgesehene Pufferstufe nicht erforderlich ist, das verstärkte Brückendifferenzsignal wird daher auf beide Controller gesplittet. Zur Kommunikation zwischen den Controllern werden die vier ehemaligen Verstärkersteuerleitungen des Signalcontrollers verwendet.

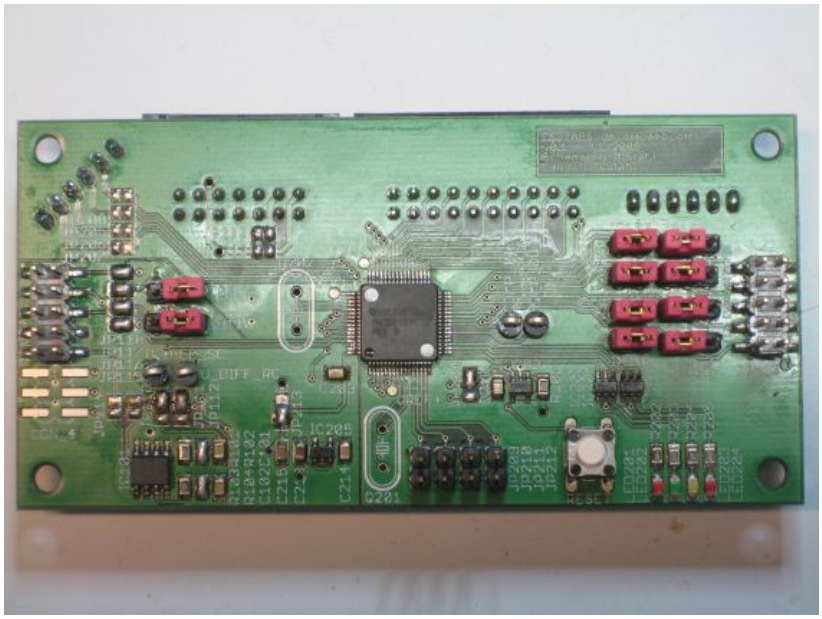


Abbildung 3.14: *Foto der bestückten Regelplatine*

Jumper	**Signal**	**Verwendung**	**Einstellung**
JP101, JP103, JP105, JP107	COMM_5…8	Kommunikation mit Signalcontroller	geschlossen
JP102, JP104, JP106, JP108	CS, A0…2	Verstärkersteuerleitungen	Pin 1&2 geschlossen
JP109	COMM_4	freie Leitung	offen
JP110	U_OFF	Offsetkompensations-spannung U_{Offset}	Pin 1&2 geschlossen
JP111, JP112	U_DIFF	U_{Diff} direkt an Regel- und Signalcontroller	Pin 3&2 geschlossen
JP113, JP114	U_DIFF_AMP	U_{Diff} nicht an Verstärkerstufe	offen
JP115, JP116, JP117, JP118	U_HB1 U_HB2	Halbbrückensignale an Regel- und Signalcontroller	geschlossen
JP202, 203	+3V	JTAG Versorgungsspannung	offen
JP204	JTAG_RST	Reset des Controllers per JTAG möglich	Pin 1&2 geschlossen
JP205, JP206 JP207, JP208	COMM_1…3 SW_3	freie Leitungen zum Signalcontroller	offen
JP213	+3V_ANALOG	Versorgung ADC	Pin 2&3 geschlossen

Tabelle 3.4: *Einstellungen der Jumper*

Abbildung 3.15 zeigt ein Foto mit ins Geamtsystem integrierter Regelplatine.

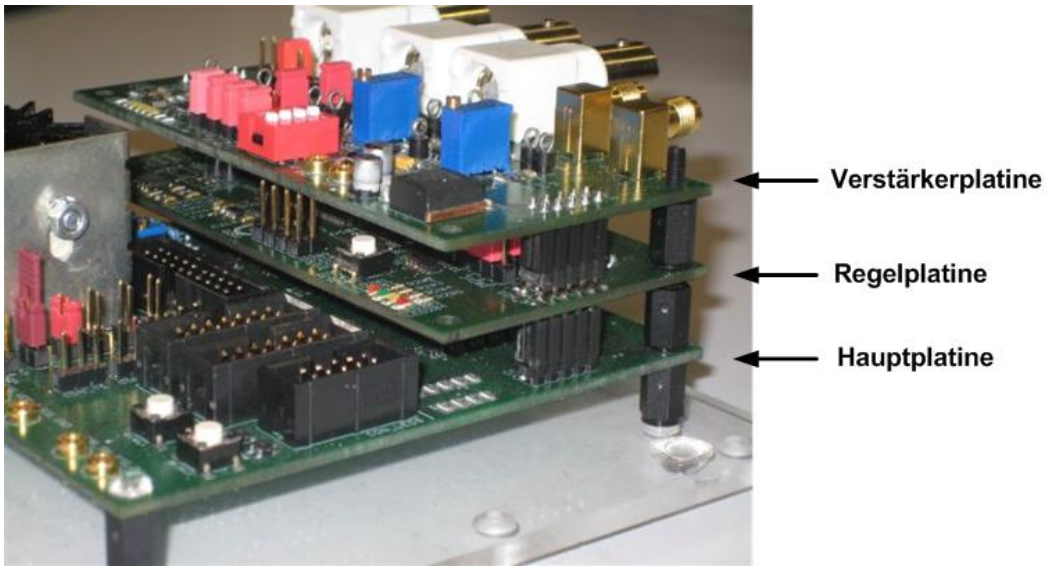


Abbildung 3.15: *Foto der Demonstrator-Plattform mit integrierter Regelplatine*

3.6 Implementation der Offsetkompensation

Die Implementation der digitalen Offsetkompensation erfolgt wie in Kapitel 2.2 beschrieben über die Einstellung des Tastverhältnisses auf $\delta = 50\%$. Die initiale Kompensation des Offsets beim „PowerOn“ des Sensors stellt kein weiteres Problem dar, die Kompensation während des aktiven Sensorbetriebes hingegen ist anspruchsvoller. Das eigentliche Signal ist bei großem Abstand zwischen Sensor und Encoder (ab etwa 5mm) bereits so schwach, dass Störungen einen starken Einfluss haben und unter anderem das Signaloffset erheblich beeinflussen. Um dieses Problem vorab einschätzen zu können wurde eine Messung ohne Offsetkompensation im aktiven Sensorzustand durchgeführt. Diese Messung erfolgte auf einem Radmessplatz mit passivem Encoderrad, welcher bereits in der vorhergehenden Diplomarbeit [10] zur Anwendung kam. Das Ergebnis befindet sich in Abbildung 3.16. Der Signaloffset schwankt hier in einem Bereich von ungefähr 500mV, was allerdings auf den sehr unrunden Lauf des Encoderrades zurückzuführen ist.

Eine kontinuierlich arbeitende Regelung würde in diesem Falle versuchen nach jeder Periode den Offset zu kompensieren, was bei so großen Schwankungen zu einem permanenten Eingriff in das Signal und damit zu einer Unterbrechung der Sensordiagnose führt. Der Regelcontroller blockiert in diesem Falle vollständig die Diagnose auf dem Signalcontroller durch Interruptanforderungen nach jeder

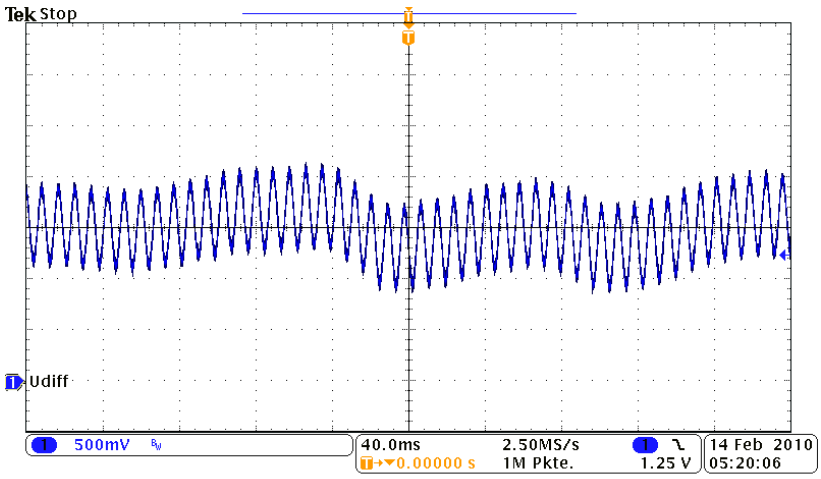


Abbildung 3.16: *Schwankender Offset bei einem Luftspalt von 5,5mm, ungeregelt*

Periode. Weiterhin kann es in der Praxis zu partiellen Beschädigungen an den Encoderrädern kommen, beispielsweise in Form eines Zahnes, der sich von den übrigen Zähnen in seiner Beschaffenheit unterscheidet. Die Regelung würde mit jeder Radumdrehung versuchen, diesen einen Zahn „auszuregeln“.

Um diese Probleme zu umgehen, wurde eine Art Mittelwertbildung für die Bestimmung des Tastverhältnisses implementiert. Dazu werden die ermittelten Werte für das Tastverhältnis zunächst nur aufsummiert. Nachdem die Summe für eine bestimmte Anzahl von Werten vorliegt, wird durch die Anzahl der Summanden dividiert. Die Offsetkompensation wird anschließend anhand des gemittelten Tastverhältnisses durchgeführt. Die Anzahl der Summanden für die Mittelwertbildung ist in Zweierpotenzen mittels des Präprozessormakros `PERIODES_FOR_MEAN_DUTY` in der Headerdatei „main.h“ einstellbar. Durch die sehr großen Offsetschwankungen ab 5mm Luftspalt wird eine Mittelwertbildung über viele Perioden erforderlich, damit der Signalcontroller in der Lage ist eine angefangene Berechnung auch abzuschließen. Dadurch wird die gesamte Regelung, inklusive der Verstärkungsumschaltung, aber sehr träge, was für Demonstrationszwecke nicht gut geeignet ist. Deshalb wurde eine weitere Option in der Software vorgesehen, nämlich die Geschwindigkeit der Regelung abhängig vom eingestellten Verstärkungsfaktor. Dadurch arbeitet die Regelung bei großem Luftspalt und stark schwankenden Signalen weiterhin langsam, bei einer guten Signalqualität und niedrigem Verstärkungsfaktor entsprechend schneller. Diese Funktion ist ebenfalls in der Headerdatei aktivierbar. Des Weiteren lässt sich die Regelung im aktiven Zustand des Sensors für eventuelle Untersuchungen auch vollständig deaktivieren.

3.7 Implementation der Verstärkungsregelung

Die Regelung des Verstärkungsfaktors wurde folgendermaßen implementiert. Der Analog-Digital-Converter fordert nach jedem Abtastwert einen Interrupt an. Im zugehörigen Interrupthandler wird zunächst überprüft, ob der Wert in der oberen oder unteren Halbwelle des Sensorsignals genommen wurde. Befindet sich der untergeordenete Automat im Zustand *Search_S2_Check_Gain*, handelt es sich um die obere, im Zustand *Search_S1_Check_Gain* um die untere Halbwelle. Daraufhin wird überprüft, ob der Abtastwert die eingestellten Schwellwerte überschritten hat. Für jede festgestellte Überschreitung einer Schwelle wird eine zugehörige Zählvariable erhöht. Diese Variablen dienen im weiteren Programmablauf dazu, das Signal auf korrekte Verstärkung zu prüfen. Die Definition der Schwellwerte erfolgt über die Präprozessormakros `LEVEL_GMIN` und `LEVEL_GMAX`.

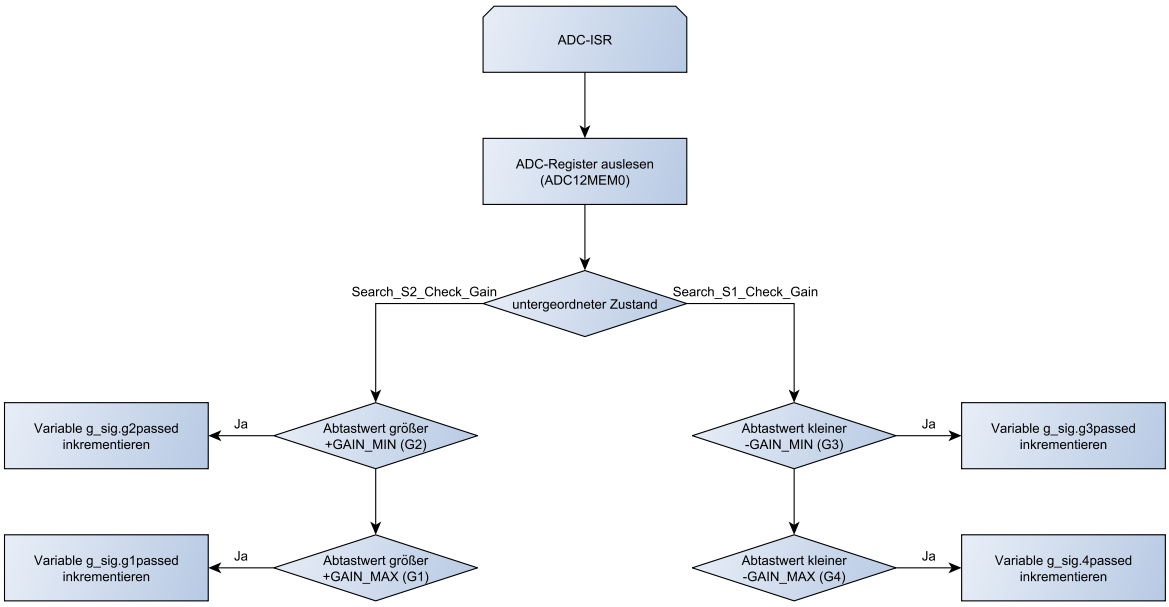


Abbildung 3.17: *Flussdiagramm der ADC-Interrupt-Service-Routine*

Sobald eine Signalperiode abgetastet wurde, wird die Funktion `check_gain()` aufgerufen. In dieser wird zunächst unterschieden, ob sich der Sensor im Zustand *Initial_1Hz* oder *Active* befindet. Anschließend werden die Variablen `g_sig.g1passed...g_sig.g4passed` abgefragt. Bei korrekt verstärktem Signal müssen `g_sig.g2passed` und `g_sig.g3passed` größer Null sein (Überschreitung der unteren Schwellwerte). Die zwei übrigen Variablen hingegen müssen Null sein, da die Verstärkung ansonsten zu groß wäre. Sollte eine Verletzung dieser Bedingungen vorliegen, wird der Verstärkungsfaktor umgeschaltet. Dazu wird die Funktion `set_gain()` aufgerufen. Anschließend wird das Signal `COMM_5` auf

logisch Eins gesetzt, damit auf dem Signalcontroller eine Interrupt-Anforderung zur Signalisierung des Regeleingriffs erfolgt. Bei korrekt verstärktem Signal wird `COMM_5` zurück auf Null gesetzt und damit die Diagnose auf dem Signalcontroller wieder freigegeben. In Abbildung 3.18 ist ein Flussdiagramm des beschriebenen Ablaufes dargestellt.

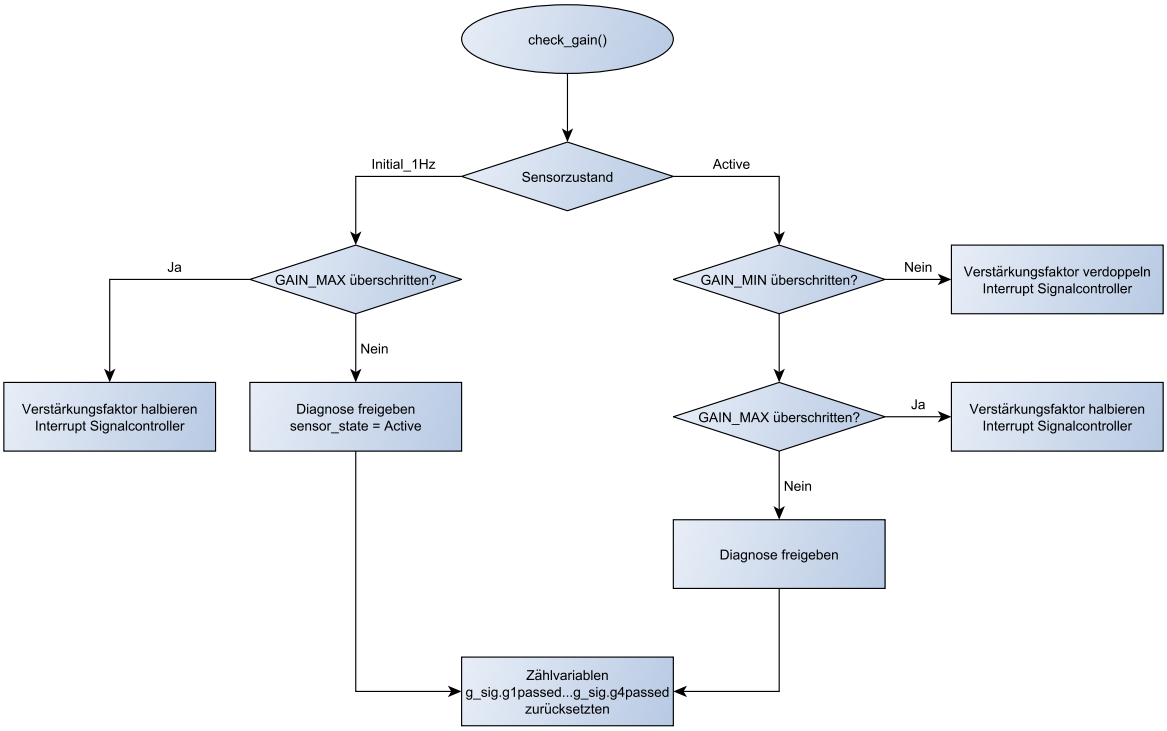


Abbildung 3.18: *Flussdiagramm der Funktion check_gain()*

4 Messungen

4.1 Funktionsnachweise

Nach erfolgreicher Inbetriebnahme der Regelplatine wurden einige Messungen zum Nachweis der Funktion durchgeführt. Da für die ideale Lage der Hysterese- und Gain-Schwellen noch keine Erfahrungswerte vorliegen wurden zunächst folgende Werte gewählt:

- $\pm HYST = 0,1V$
- $\pm GAIN_{Min} = 0,31V$
- $\pm GAIN_{Max} = 0,94V$

Bei den angegebenen Spannungswerten handelt es sich um den Betrag zur „Nulllinie“ des Sensorsignals. Diese liegt in der Mitte des Aussteuerbereiches von ADC und DAC bei $\frac{2,5V}{2} = 1,25V$. Die oberen Gain-Schwellen liegen also beispielsweise bei:

$$\left.\begin{aligned} +GAIN_{Max} = 1,25V + 0,94V &= 2,2V \\ -GAIN_{Max} = 1,25V - 0,94V &= 0,3V \end{aligned}\right\} \widehat{=} \frac{940mV}{1250mV} \cdot 100\% = 75,2\% \text{ des Aussteuerb.} \tag{4.1}$$

Da der interne DAC im Regelcontroller eine Auflösung von 12 Bit zur Verfügung stellt, wird eine entsprechende Skalierung vorgenommen:

$$\frac{\pm GAIN_{Max}}{\frac{U_{Ref}}{2^{12}}} = \frac{940mV}{2500mV} \cdot 4096 = 1540 \tag{4.2}$$

Dieser Wert kann in Form des Makros `LEVEL_GMAX` in der Datei `main.h` definiert werden, für die anderen Schwellen wird entsprechend verfahren.

Für die nachfolgenden Messungen wurde der Demonstrator (inklusive Regelplatine) mit Signalen aus einem Funktionsgenerator AFG3022B von Tektronix gespeist. Das Generatorsignal wurde dabei an den Verstärkereingang U_{HB1} angeschlossen, der zweite Verstärkereingang U_{HB2} wurde für die Messungen auf Massepotenzial gelegt. Beim verwendeten Oszilloskop handelt es sich um ein Tektronix MSO3034.

Der Messaufbau ist in Abbildung 4.1 skizziert. In den Messergebnissen ist oben stets das Eingangssignal (Generatorsignal), unten das geregelte Ausgangssignal des Demonstrators dargestellt. Die eingeblendeten Cursorlinien liegen bei 0V und 2,5V und repräsentieren somit den ADC-Aussteuerbereich.

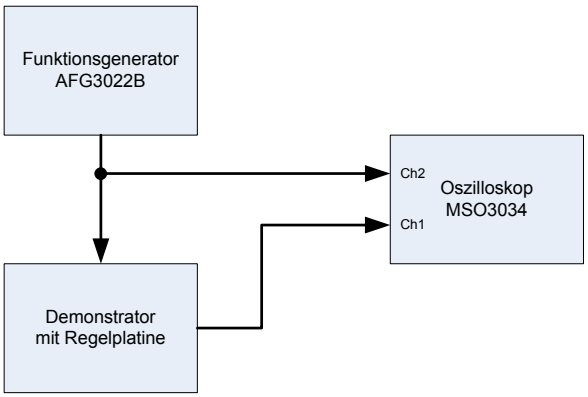


Abbildung 4.1: *Skizze des Messaufbaus*

4.1.1 Initiale Regelung

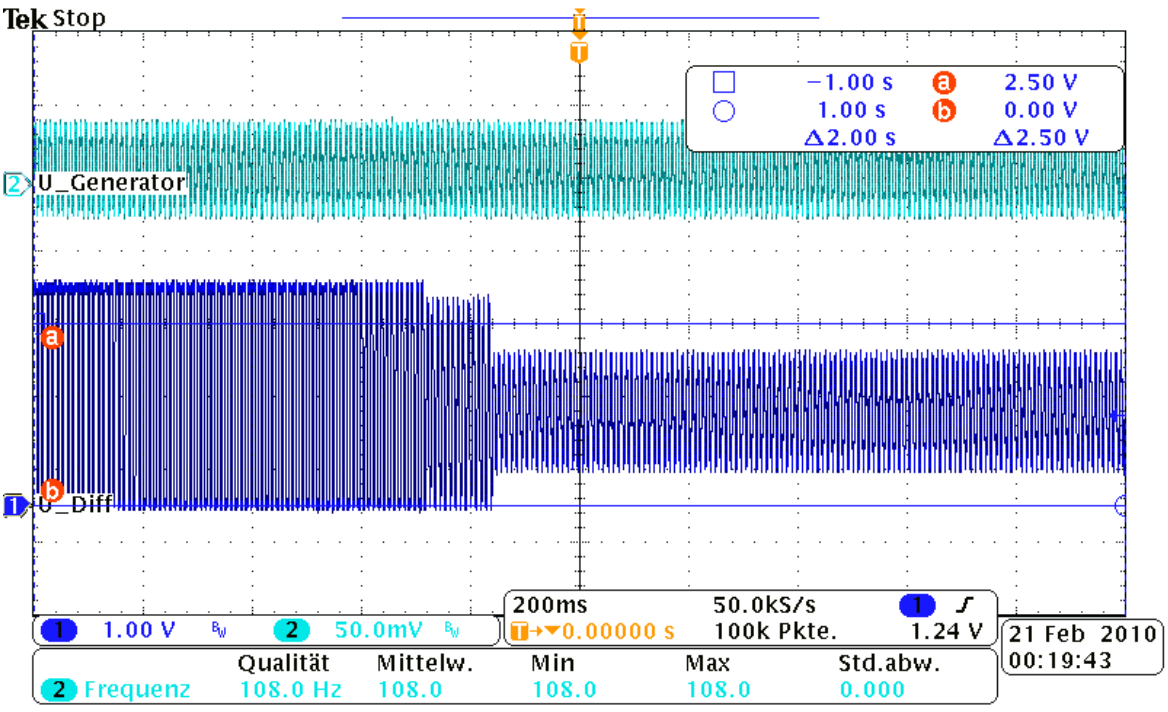


Abbildung 4.2: *Initiale Regelung beim PowerOn des Systems*

Abbildung 4.2 zeigt den Regelvorgang beim „PowerOn“ des Demonstrators. Das Eingangssignal wird zunächst maximal verstärkt, was zu einer völligen Übersteuerung führt. Das Signal wird jetzt abwechselnd offsetkompensiert und auf korrekte Verstärkung geprüft. Der Verstärkungsfaktor wird solange halbiert, bis die Signalamplitude im gültigen Bereich zwischen den Schwellen $\pm GAIN_{Max}$ liegt. Die initiale Regelung arbeitet wie gefordet.

Die Parameter des verwendeten Generatorsignals:

- Frequenz: $f = 107Hz$
- Amplitude $U_{pp} = 60mV$
- Offset $U_{Off} = 9mV$

4.1.2 Regelung im aktiven Sensorzustand

Die vier folgenden Messungen wurden durchgeführt, um die Funktion der Verstärkungs- und Offsetregelung im aktiven Sensorzustand zu demonstrieren. In Abbildung 4.3 wurde die Amplitude des Eingangssignals im Bereich von $65 \ldots 10mV_{pp}$ kontinuierlich verringert. Die Sprünge im unteren Signal zeigen den Einsatz der Verstärkungsregelung. Der Verstärkungsfaktor wurde bei Unterschreitung der Schwellen $\pm GAIN_{Min}$ verdoppelt.

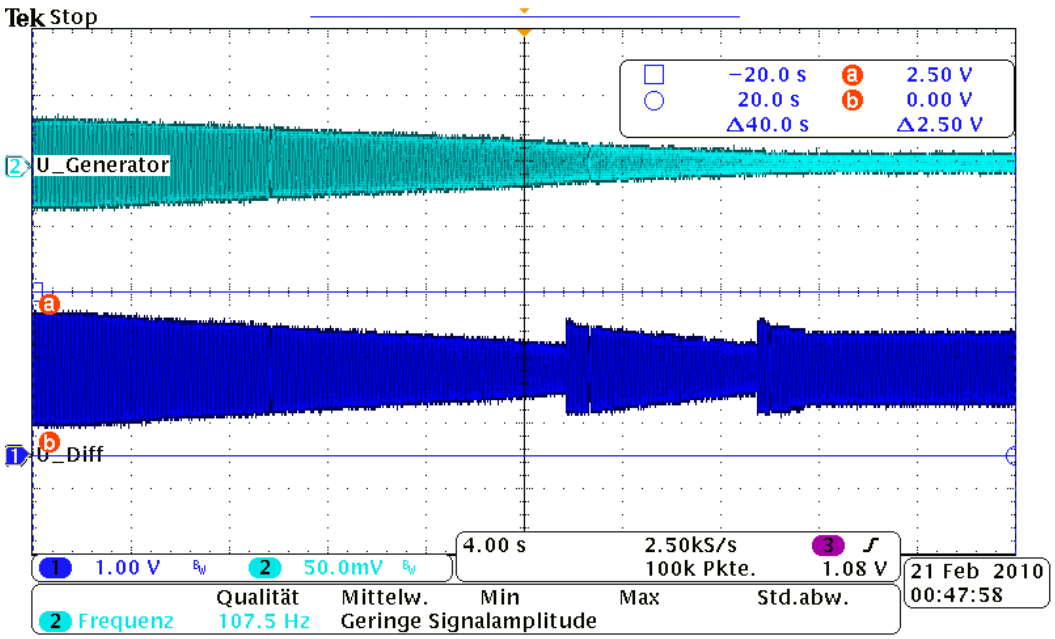


Abbildung 4.3: *Amplitudensweep des Eingangssignals von* $U_{pp} = 65 \ldots 10mV$ *mit* $f = 107Hz$

In Abbildung 4.4 wurde ein Sprung der Eingangsamplitude von 10mV auf 20mV bei einer Signalfrequenz von $f = 2{,}5kHz$ gemessen. Aufgrund des Sprunges überschreitet das verstärkte Signal die Schwellen $\pm GAIN_{Max}$, woraufhin der Verstärkungsfaktor vom Regelcontroller halbiert wird.

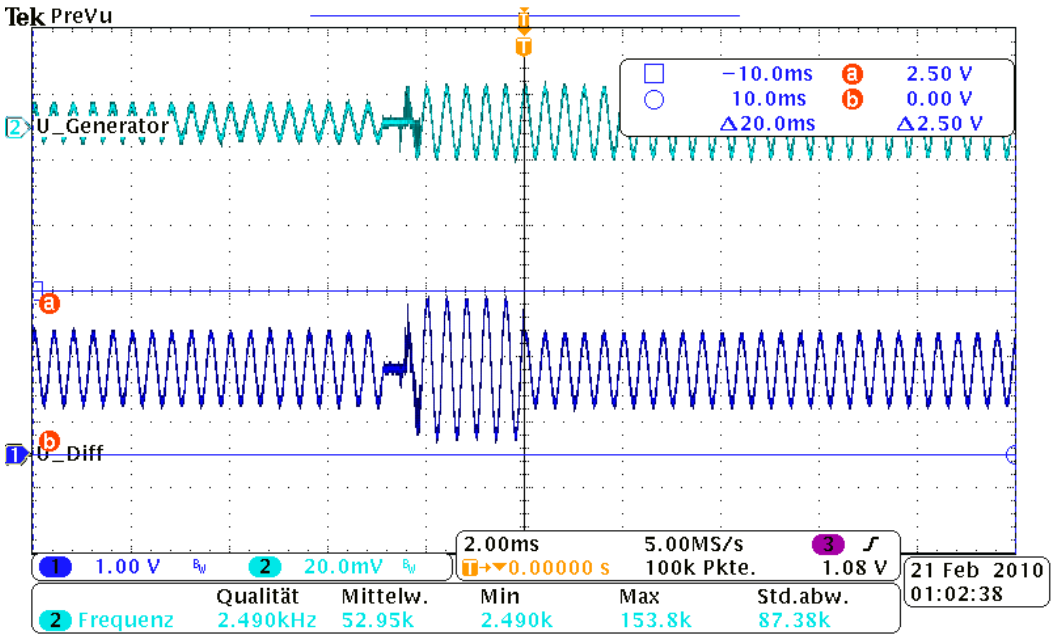


Abbildung 4.4: *Amplitudensprung des Eingangssignal von* $10mV_{pp}$ *auf* $20mV_{pp}$ *mit* $f = 2{,}5kHz$

Zur Demonstration der Offsetkompensation im aktiven Betrieb des Sensors wurde in Abbildung 4.5 ein Sweep des Eingangssignaloffsets im Bereich von $U_{Off} = -20 \cdots +20mV$ durchgeführt (oberes Signal). Dieses Offset wird durch den Regelcontroller permanent kompensiert, so dass das untere Signal stets offsetfrei in der Mitte des ADC-Aussteuerbereiches liegt.

In Abbildung 4.6 ist ein Sprung des Offsets von -4mV auf +4mV bei einer Signalfrequenz von $f = 2{,}5kHz$ aufgezeichnet. Zunächst ist das Offset von $U_{Off} = -4mV$ im unteren Signal kompensiert. Der Sprung der Offsetspannung am Eingang wirkt sich unmittelbar auf das Signal U_{Diff} aus. Der Regelcontroller registriert diese Offsetverschiebung und führt eine neue Kompensation durch.

Bei den durchgeführten Messungen handelt es sich lediglich um exemplarisch ausgewählte Funktionsnachweise. Ein vollständiger Nachweis, der alle möglichen Problemfälle abdeckt, ist im Rahmen dieser Arbeit nicht durchführbar. Dennoch konnte mit den erbrachten Messungen die Funktion der Regelplatine gezeigt werden.

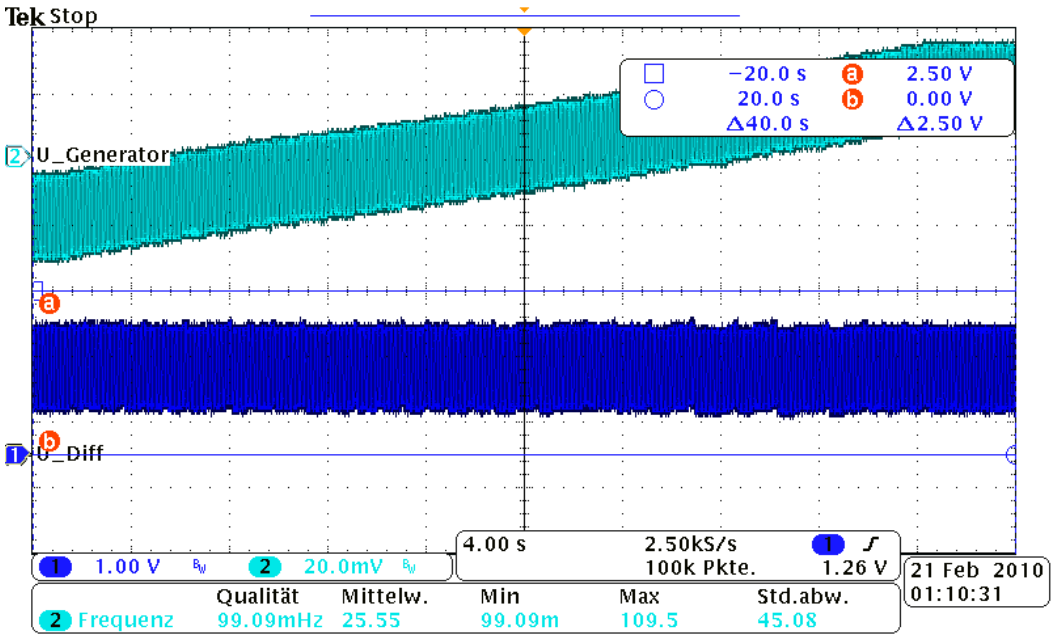


Abbildung 4.5: *Offsetsweep des Eingangssignals von* $U_{Off} = -20 \cdots +20mV$ *mit* $f = 107Hz$ *und* $U_{pp} = 25mV$

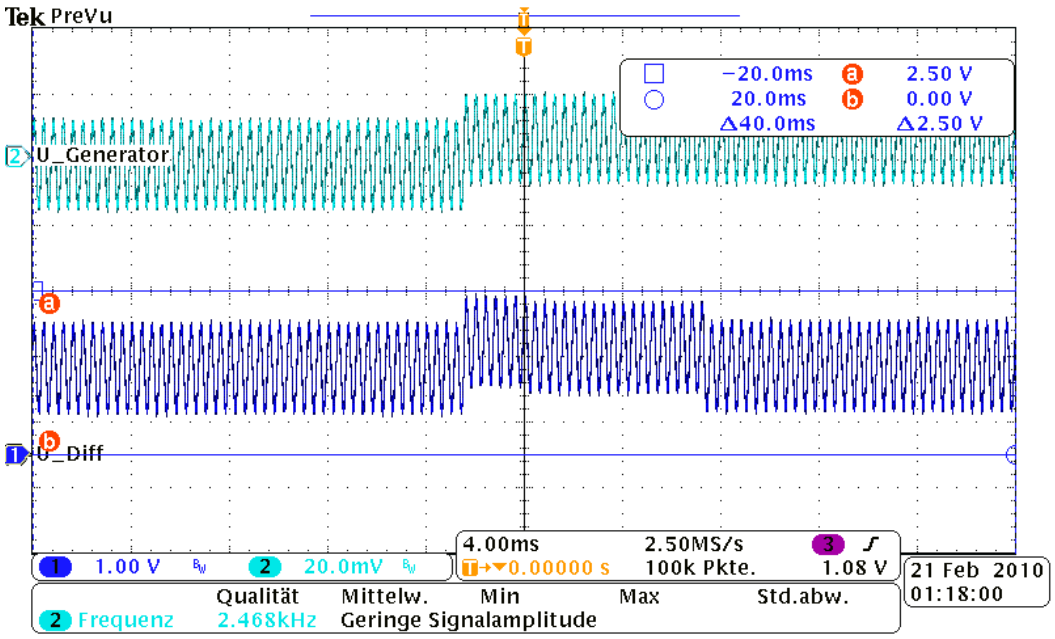


Abbildung 4.6: *Offsetsprung des Eingangssignals von -4mV auf +4mV bei* $U_{pp} = 25mV$ *und* $f = 2{,}5kHz$

4.2 Automatisierung von Radmessplätzen

Zusätzlich zur Erweiterung des Demonstrators um eine Regelplatine für die Offsetkompensation und Verstärkungsregelung, wurden im Rahmen dieser Arbeit zwei Radmessplätze automatisiert. Dabei handelt es sich um einen Messplatz mit passivem (siehe Abbildung 4.7), sowie um einen Messplatz mit aktivem Encoderrad (Abbildung 4.8). Mit Hilfe dieser Messplätze können vollautomatische Messreihen für verschiedene Einbaulagen des Sensors zum Encoderrad durchgeführt werden.

Dieser Abschnitt erhebt nicht den Anspruch, die Automatisierung der Messplätze in allen Details zu beschreiben. Es soll vielmehr ein kurzer Überblick über die erfolgten, praktischen Arbeiten, im Rahmen dieser Bachelorthesis, zur Automatisierung der Radmessplätze geschaffen werden. Die durchgeführten Arbeiten umfassen die Ausrüstung der Messplätz mit Schrittmotoren, die Ansteuerung von Motoren, Demonstrator und Oszilloskop mit Matlab, sowie die Programmierung von Matlab-Programmen zur vollautomatischen Aufnahme von Messreihen.

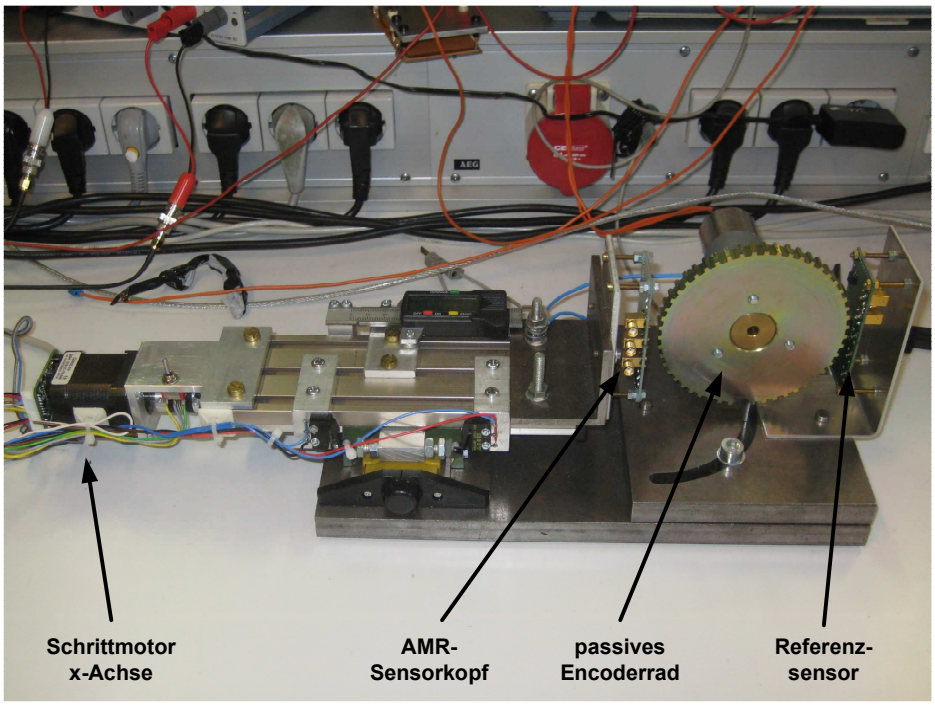


Abbildung 4.7: *Radmessplatz mit passivem Encoderrad und verfahrbarer x-Achse*

Der Messplatz mit passivem Encoder besteht im Wesentlichen aus den folgenden Komponenten:

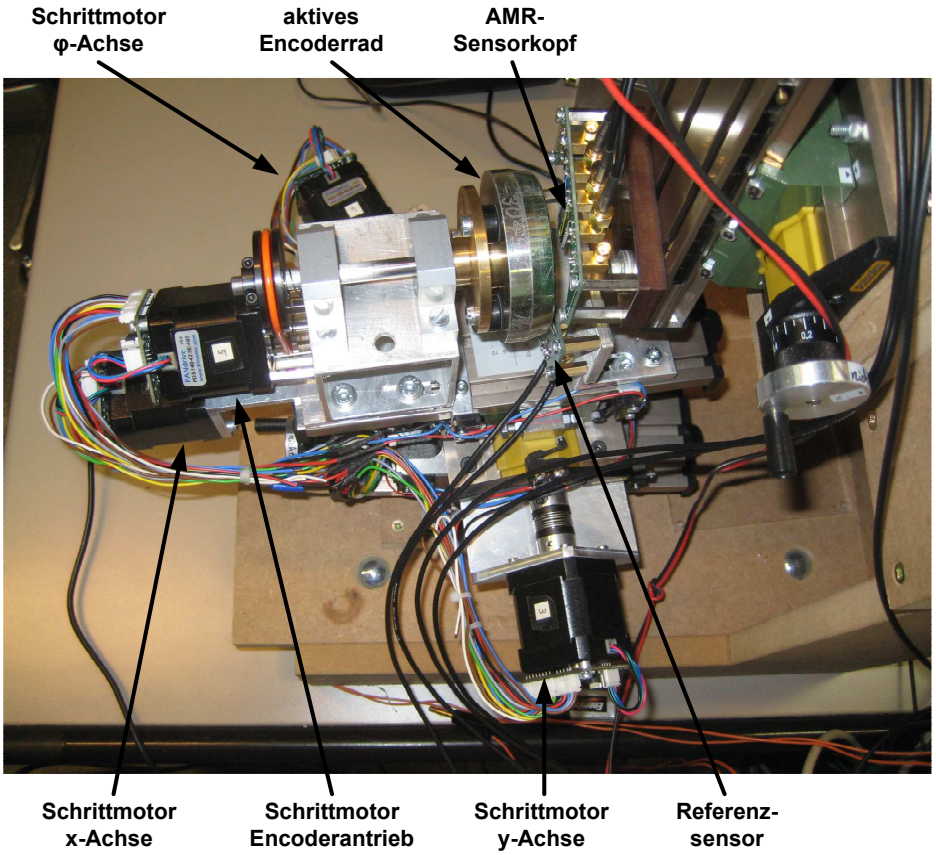


Abbildung 4.8: *Radmessplatz mit aktivem Encoderrad, drei Achsen verfahrbar*

- Feststehendes passives Encoderrad, welches durch einen Gleichstrom-Motor angetrieben wird
- AMR-Messbrücke ohne Signalverarbeitungs-IC, montiert auf einem x-y-Koordinatentisch
- Schrittmotor zum Verfahren der x-Achse des Koordinatentisches
- Referenzsensor zur genauen Messung der Raddrehzahl

Der Messplatz mit aktivem Encoderrad ist umfangreicher aufgebaut. Dieser lässt sich nicht nur in der x-Achse, sondern zusätzlich in der y-Achse verfahren, sowie um den Winkel φ drehen. Der Sensorkopf ist dabei fest montiert, das Encoderrad befindet sich auf dem verfahrbaren Teil. Der Aufbau zeichnet sich durch folgende Komponenten aus:

- Aktives Encoderrad, angetrieben durch einen Schrittmotor, montiert auf einem x-y-Koordinatentisch zur Positionierung des Encoders vor dem Sensor
- Schrittmotoren zum Verfahren der x- und y-Achse des Koordinatentisches.

- Drehtisch, drehbar durch angebrachten Schrittmotor, zum „Verkippen“ des Encoders vor dem Sensor
- AMR-Messbrücke ohne Signalverarbeitungs-IC, montiert auf einem Koordinatentisch zur initialen Ausrichtung des Sensors vorm Encoderrad
- Referenzsensor zur genauen Messung der Raddrehzahl

Bei den verwendeten Schrittmotoren *PD3-140-42-SE-485* der Firma *Trinamic Motion Control* handelt es sich um Schrittmotoren inklusive Ansteuerelektronik. Diese Motoren werden über eine RS485[1]-Schnittstelle miteinander zu einem Bus verschaltet und an einen Computer angeschlossen. Dazu dient das Trinamic *USB-2-485* Interface. Angesteuert werden die Motoren über eine Assembler ähnliche Sprache, die *Trinamic Motion Control Language* (TMCL). Um die Motoren auch aus Matlab-Skripten heraus verfahren zu können, existiert im Projekt bereits die Treiberdatei `tmcl.mdd` für die Matlab Instrument Control Toolbox. Diese Toolbox dient zur Ansteuerung von externer Hardware unter Matlab.

Wie im Blockschaltbild in Abbildung 4.9 zu erkennen ist, wird außerdem ein digitales Speicheroszilloskop sowie das Demonstratorsystem aus Matlab heraus gesteuert. Das Oszilloskop ist dabei per Ethernet an den Computer angebunden und wird ebenfalls per Treiberdatei über die Instrument Control Toolbox gesteuert, beziehungsweise ausgelesen. Die Daten des Oszilloskopes dienen in der späteren Auswertung zum Vergleich mit den Messdaten des Demonstrators. Dieser übeträgt die gemessenen Daten über eine serielle Schnittstelle (per Adapterkabel auf USB umgesetzt) an das Matlab-Programm. Das Blockschaltbild repräsentiert den Messplatz mit aktivem Encoderrad.

Der Ablauf einer Messreihe ist in Form eines Flussdiagrammes in Abbildung 4.10 dargestellt. Für die anschließende Auswertung der aufgezeichneten Daten wurden entsprechende Auswert-Skripte entwickelt. In Anhang C befinden sich weitere Informationen über die Programme zur Messplatz-Automatisierung.

[1] Schnittstellen-Standard für differentielle, serielle Datenübertragung

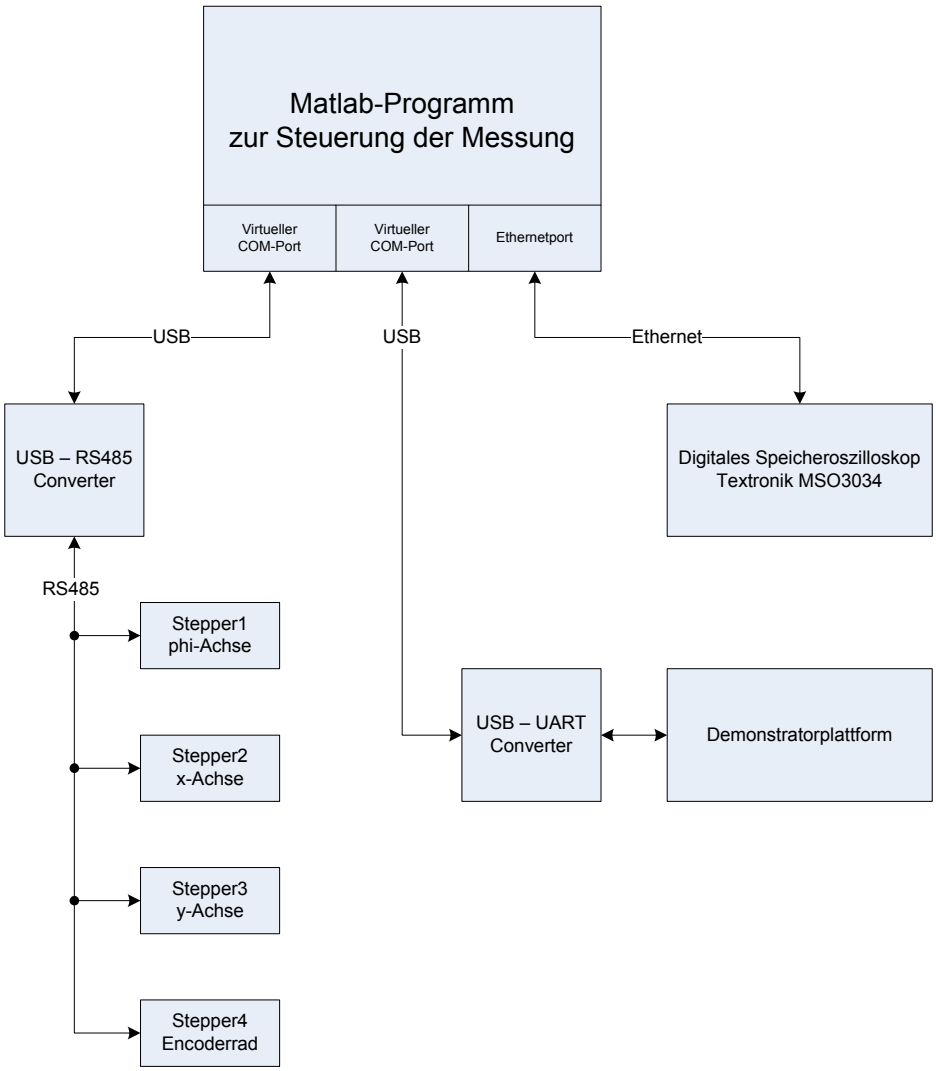


Abbildung 4.9: *Blockschaltbild des automatisierten Radmessplatzes mit aktivem Encoder*

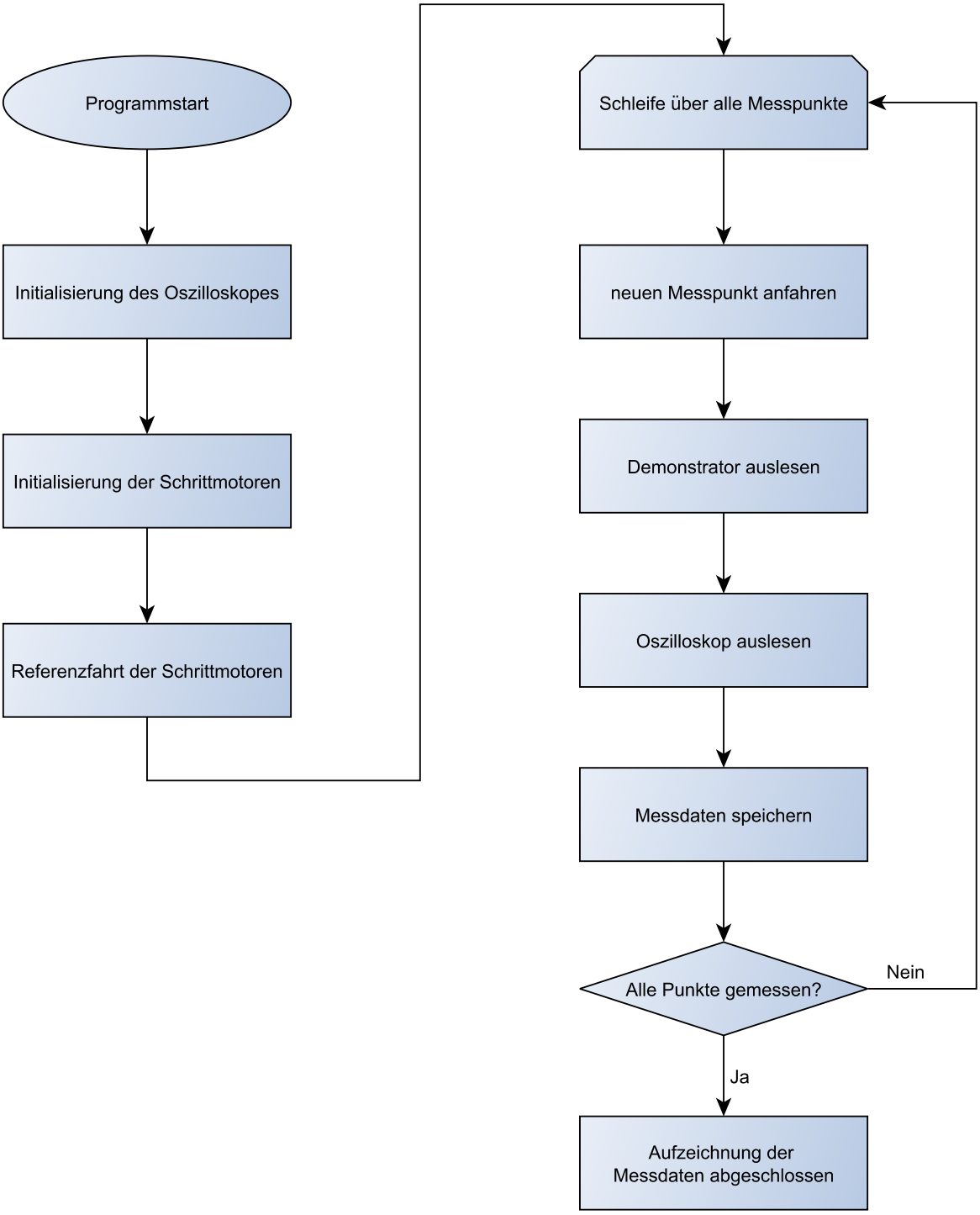


Abbildung 4.10: *Flussdiagramm zur vollautomatischen Aufzeichnung von Messdaten an den Radmessplätzen*

4.3 Ergebnisse

4.3.1 Messplatz mit passivem Encoderrad

Am Radmessplatz mit passivem Encoderrad wurde eine Vergleichsmessung zwischen „Demonstrator ohne Regelplatine“ und „Demonstrator mit Regelplatine“ durchgeführt. Die Ergebnisse sind in Abbildung 4.11 und 4.12 dargestellt.

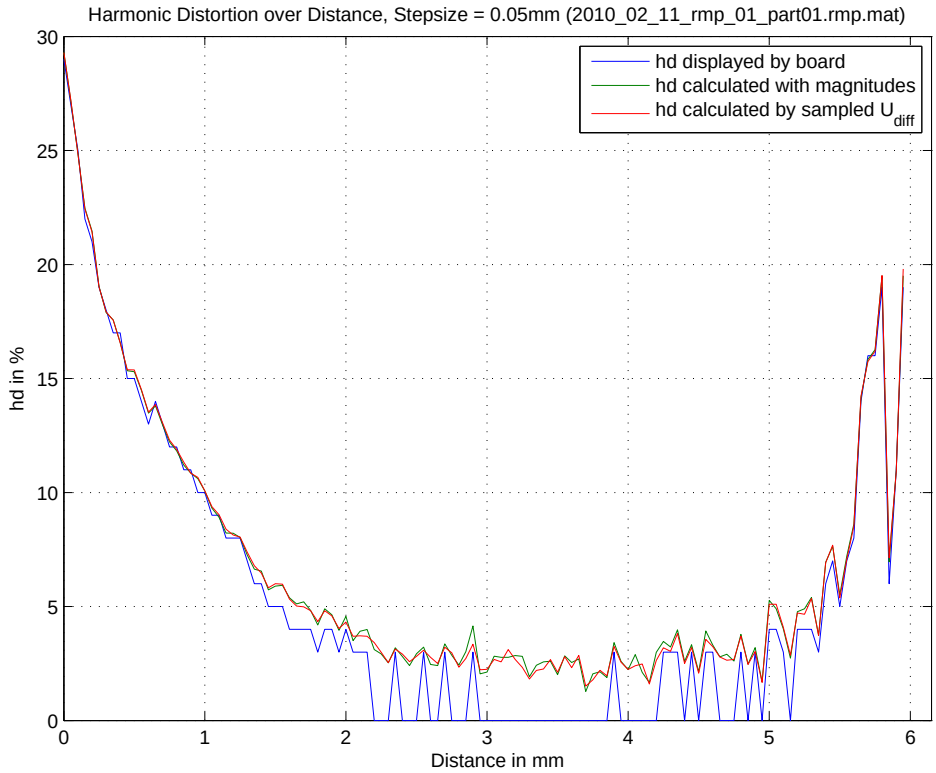


Abbildung 4.11: *Klirrfaktor in Abhängigkeit des Luftspaltes, gemessen mit Demonstrator ohne Regelplatine, manuelles Umschalten des Verstärkungsfaktors*

Aus den Grafiken geht hervor, dass es keine nennenswerten Abweichungen des Klirrfaktor-Verlaufes mit und ohne Einsatz der Regelplatine gibt, wodurch die korrekte Funktion der Regelplatine bestätigt wird. In der vorhergehenden Diplomarbeit [10] wurden bereits mehrere Messreihen an diesem Radmessplatz durchgeführt. Die Messreihen lassen sich mit der, um die Regelplatine erweiterten, Demonstrator-Plattform reproduzieren. Ein manuelles Umschalten des Verstärkungsfaktors ist nicht mehr erforderlich.

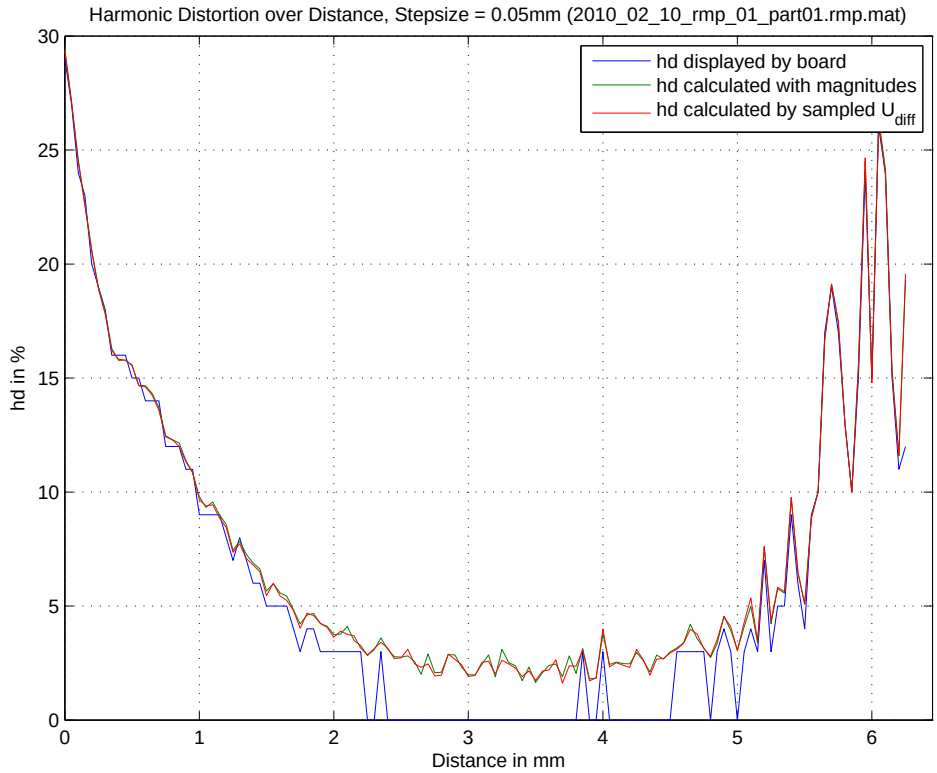


Abbildung 4.12: *Klirrfaktor in Abhängigkeit des Luftspaltes, gemessen mit Demonstrator inklusive Regelplatine, automatisches Umschalten des Verstärkungsfaktors*

4.3.2 Messplatz mit aktivem Encoderrad

Im Gegensatz zu den in [10] durchgeführten Messungen am Radmessplatz mit passivem Encoder, kann das aktive Encoderrad in mehreren Achsen verfahren werden. Dadurch kann der zuvor gemessene Klirrfaktor in Abhängigkeit des Luftspaltes nun auch für verschiedenen Verkippungswinkel gemessen werden. Die Regelplatine ermöglich dabei die vollautomatische Messung solcher Kennfelder. Im Rahmen dieser Arbeit waren aus Aufwandsgründen lediglich erste Messungen zur Demonstration möglich. Die grundsätzliche Funktion konnte dabei gezeigt werden.

Abbildung 4.13 zeigt ein exemplarisches Ergebnis eines gemessenen Kennfeldes. Der „Radius“ des Kreissektors entspricht dabei der Entfernung zwischen Sensor und Encoder (Luftspalt). Auf der Abszisse sind die Verkippungswinkel φ aufgetragen. Jeder gemessene Punkt entspricht einem Viereck im Kreissektor, der Klirrfaktor in diesem Punkt wird durch die Farbe des Viereckes repräsentiert. Der Verlauf des Klirrfaktors über die Entfernung für einen festen Winkel entspricht den bisher gemessenen Verläufen am Messplatz mit passivem Encoder.

In Abbildung 4.14 wurde die Messung erneut durchgeführt, allerdings mit dem

Unterschied, dass die x-Achse für jeden zu messenden Winkel stets von 3*mm* absteigend auf 0*mm* verfahren wurde. Bei der Messung zuvor wurde die x-Achse von 0*mm* aufsteigend auf 3*mm* verfahren. Der wesentliche Unterschied in den Ergebnissen liegt im Bereich zwischen $0{,}3 \ldots 1{,}0mm$ Distanz zwischen Sensor und Encoder. Bei der ersten Messung treten an dieser Stelle große Sprünge des Klirrfaktors auf, bei der zweiten Messung ist das nicht der Fall. Dieses Verhalten ist auf den mechanischen Aufbau des Messplatzes zurückzuführen. Die für solche Messungen erforderliche mechanische Präzision kann mit diesem Aufbau nicht erreicht werden. Der Entwurf eines präzisen Radmessplatzes wird für weitere Untersuchungen undedingt notwendig, und ist bereits Gegenstand einer entstehenden Diplomarbeit [17]. Die Ergebnisse weiterer Messungen befinden sich in Anhang C.2.

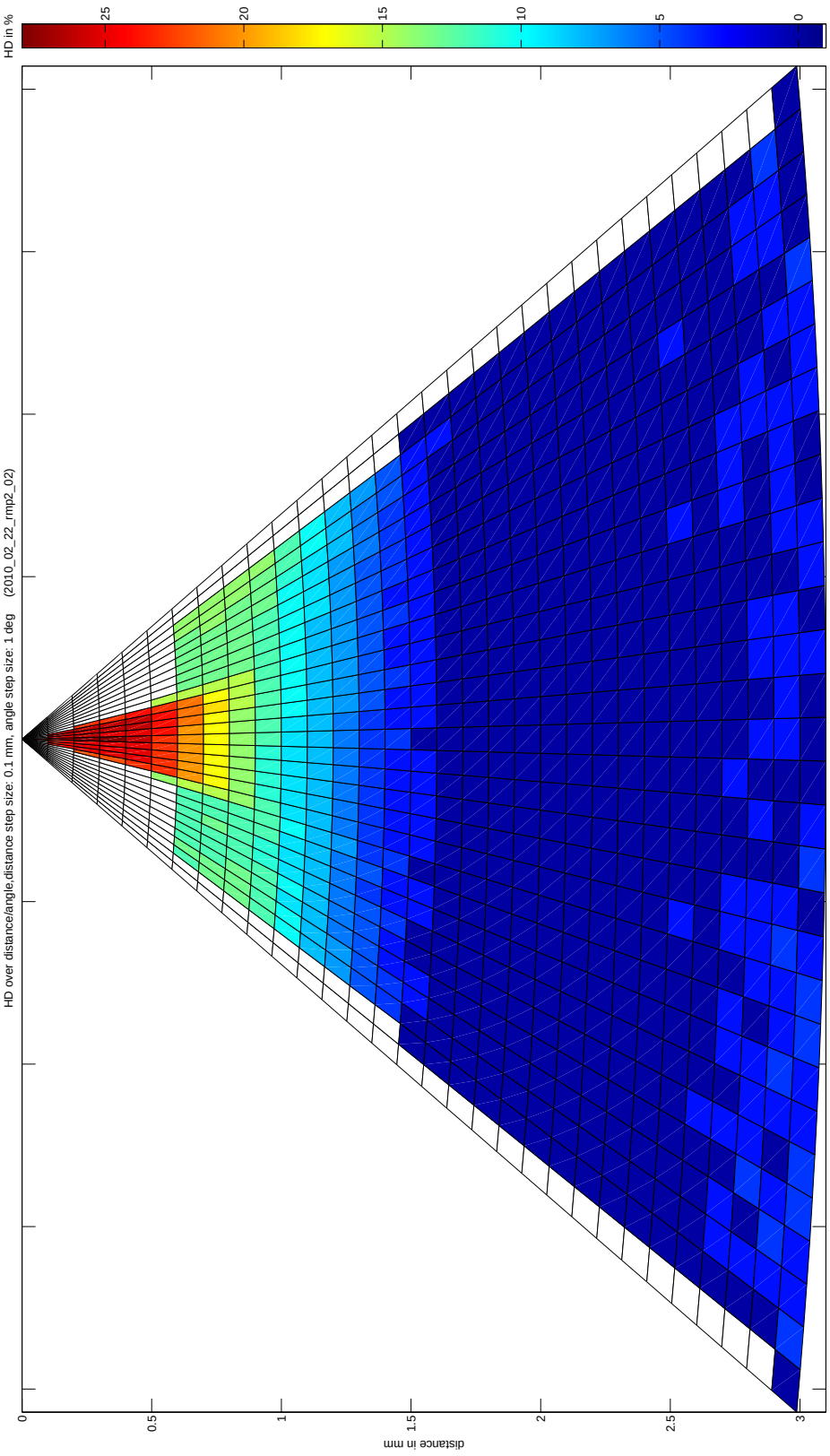


Abbildung 4.13: *Aktiver Encoder in x-Achse verfahren sowie in φ-Achse verkippt. Der Klirrfaktor ist in die Farbe codiert, in x-Richtung wurde von 0mm aufsteigend gemessen*

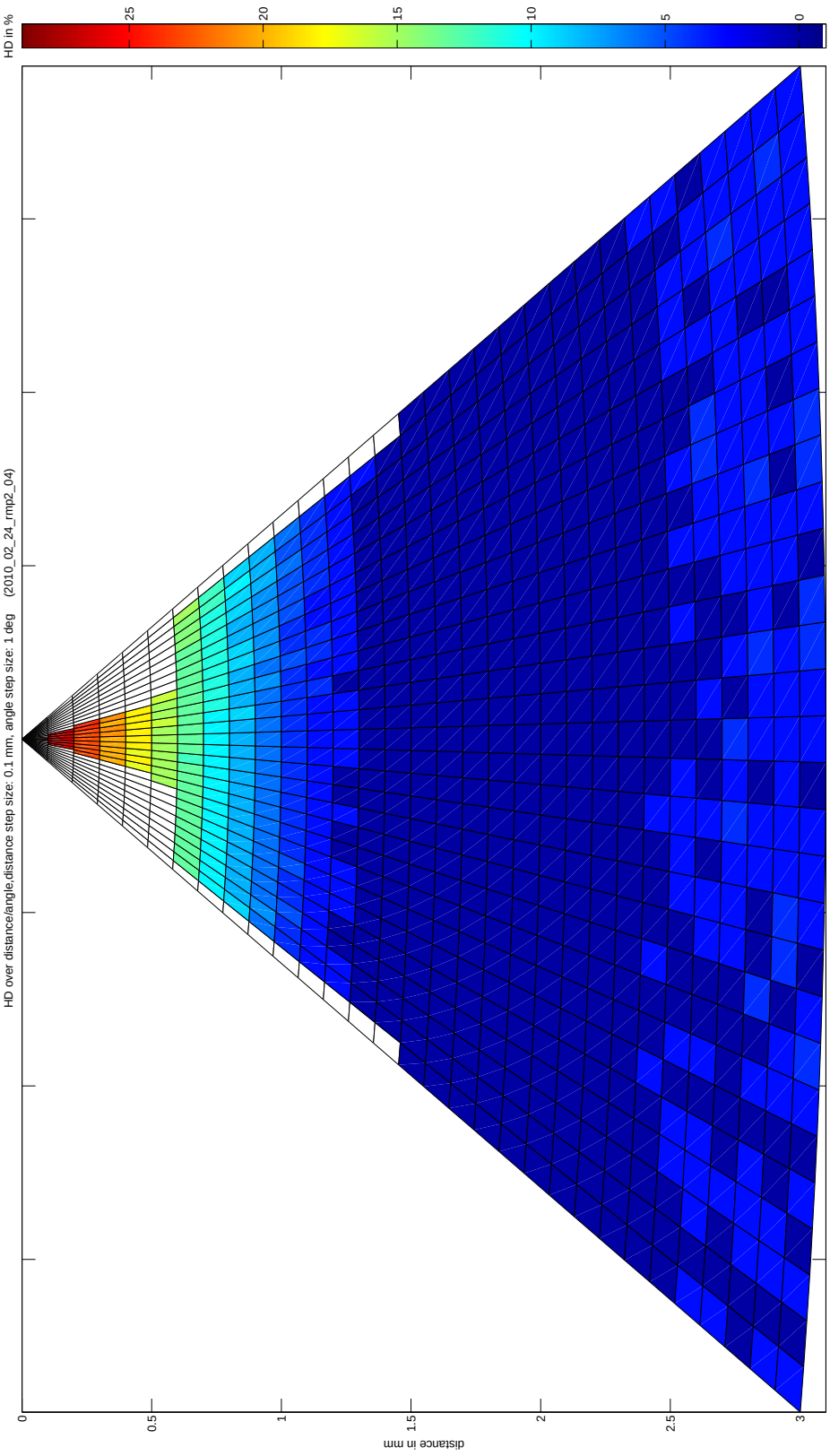


Abbildung 4.14: *Aktiver Encoder in x-Achse verfahren sowie in φ-Achse verkippt. Der Klirrfaktor ist in die Farbe codiert, in x-Richtung wurde von 3mm absteigend gemessen*

5 Schluss

5.1 Zusammenfassung

Im Rahmen dieser Arbeit wurden zwei wesentliche Dinge erreicht. Zum einen wurden die im Projekt vorhandenen Radmessplätze automatisiert, um vollautomatische Messreihen durchführen zu können. Dazu wurden Verfahr- und Verkippachsen mit Schrittmotoren ausgestattet. Zur Durchführung und anschließender Auswertung von Messreihen wurden Matlab-Programme entwickelt. Diese Programme zeichnen sowohl Messdaten des Demonstrator-Systems, als auch von einem digitalen Speicheroszilloskop auf. Die Ergebnisse lassen allerdings aufgrund mangelnder mechanischer Präzision der Messplätze noch keine quantitativen Aussagen zu. Dennoch wurden wertvolle Erfahrungen gesammelt, die als Grundlage für spätere Messungen an einem präzisen Messplatz, welcher in einer parallelen Diplomarbeit [17] entwickelt wird, dienen.

Zum Anderen wurde mit der Regelplatine ein Controllersystem entwickelt, welches sich sehr gut in die bestehende Demonstrator-Plattform integrieren lässt und diese um die Funktionen Offsetkompensation und Verstärkungsregelung erweitert. Der Entwurf der Regelplatine ist dabei in hohem Maße flexibel. Damit wird erstens eine vollständige Integration in die vorhandene Hardware ermöglicht, zweitens entsteht die Möglichkeit, den Signalfluss zwischen Haupt-, Regel-, und Verstärkerplatine zu beeinflussen. Dadurch können in anschließenden Arbeiten Experimente mit unterschiedlichen Konfigurationen durchgeführt werden, außerdem besteht die Möglichkeit für zusätzliche Systemerweiterungen.

Für den Entwurf der Software wurde zunächst ein geeignetes Verfahren zur Realisierung von Offsetkompensation und Verstärkungsregelung gefunden und in Matlab modelliert. Die Implementation des Verfahrens auf dem Regelcontroller erfolgte unter Berücksichtigung unterschiedlicher Prioritäten von Regelung und Diagnose. Die grundlegende Funktion eines ABS-Sensors, nämlich die Ausgabe von digitalen Pulsen mit den Nulldurchgängen des Signals, ist dabei zu jeder Zeit gewährleistet. Mit der Regelplatine wurden Offsetkompensation und Verstärkungsregelung als parallel arbeitender Prozess zur Sensordiagnose auf den Regelcontroller ausgelagert. Dem Regelcontroller wurde die Möglichkeit gegeben den Signalcontroller durch eine Interruptanforderung zu unterbrechen, um einen Eingriff der Regelung

während einer laufenden Diagnoseberechnung zu signalisieren. Für die Integration der Regelplatine in das Gesamtsystem wurden außerdem Anpassungen an der bestehenden Software des Signalcontrollers durchgeführt.

Durch die Erweiterung der Demonstrator-Plattform um die Regelplatine wurde die Möglichkeit für die Durchführung von vollautomatischen Messreihen, ohne manuelles Umschalten des Verstärkungsfaktors, geschaffen. Außerdem können nun Messungen mit hohen Raddrehzahlen bis $2{,}5kHz$ Zahnfrequenz, und dem damit verbundenen Offsetdrift, durchgeführt werden.

5.2 Ausblick

Eine völlig andere Vorgehensweise zur Kompensation des Signaloffsets wäre eine Alternative zur Gleichspannungsversorgung der AMR-Brücke. Die Brücke ließe sich beispielsweise auch mit einer, im Vergleich zum Sensorsignal, hochfrequenten Wechselspannung versorgen. Auf dieses Trägersignal würde das sinusförmige Sensorsignal dann aufmoduliert und im Anschluss an die Verstärkung durch synchrone Demodulation zurückgewonnen.

Anstelle des schaltbaren Instrumentenverstärkers ließen sich alternativ mehrere, parallele Pfade mit unterschiedlichen, aber konstanten Verstärkungsfaktoren aufbauen. Dann müsste lediglich auf den Zweig mit dem optimal verstärkten Signal geschaltet werden. Solch eine Vorgehensweise wäre wesentlich näher an einem Chipentwurf, der im Rahmen dieses Forschungsprojektes geplant ist.

Für die Umsetzung solcher Ideen wären umfangreiche Änderungen an der Demonstrator-Plattform erforderlich, aber sie bieten gleichzeitig großes Potenzial für weitere Untersuchungen.

5.3 Fazit

Mit dieser Arbeit konnte ein Beitrag zur Entwicklung einer Selbstdiagnosefunktion für ABS-Sensoren im Rahmen des Forschungsprojektes „Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren“ (ESZ-ABS) geleistet werden. Das erarbeitete Konzept ermöglicht dabei die erstmalige Sensordiagnose über den spezifizierten Arbeitsbereich des Sensors. Diese Arbeit rundet die bisherigen, erfolgreichen Arbeiten an dem Projekt in einem wichtigen Aspekt ab und ermöglicht somit die Durchführung von umfangreichen Messungen.

Literaturverzeichnis

[1] ANALOG DEVICES: *AD8231 - Zero Drift, Digitally Programmable Instrumentation Amplifier*. Version: A, Oktober 2007. http://www.analog.com/static/imported-files/data_sheets/AD8231.pdf, Abruf: 14.08.2009

[2] ANALOG DEVICES: *AD8605/AD8606/AD8608 - Precision, Low Noise, CMOS, Rail-to-Rail, Input/Output Operational Amplifier*. Version: I, Oktober 2008. http://www.analog.com/static/imported-files/data_sheets/AD8605_8606_8608.pdf, Abruf: 14.08.2009

[3] ANALOG DEVICES: *AD8226 - Wide Supply Range, Rail-to-Rail Output Instrumentation Amplifier*. Version: A, Juni 2009. http://www.analog.com/static/imported-files/data_sheets/AD8226.pdf, Abruf: 14.08.2009

[4] BOLL, R. ; OVERSHOTT, K. J.: Magnetic Sensors. In: GÖPEL, W. (Hrsg.) ; HESSE, J. (Hrsg.) ; ZEMEL, J. N. (Hrsg.): *Sensors a Comprehensive Survey* Bd. 5. Weinheim : VCH, 1989. – ISBN 3–527–26771–9. – insb. Kap. 9 von U. Dibbern

[5] BUTZMANN, STEFAN UND 23 MITAUTOREN: *Sensorik im Kraftfahrzeug*. Renningen : expert verlag, 2006. – ISBN 3–8169–2630–4

[6] DAIMLER AG: *Requirement Specifications for Standardized Interface for Wheel Speed Sensors with Additional Information „AK-Protokoll“*. 4.0. Februar 2008

[7] DRESCHHOFF, Jan-Heiner: *In Anfertigung: VHDL-Implemenation des Signalcontrollers*. Hamburg, Hochschule für Angewandte Wissenschaften, Diplomarbeit in Anfertigung

[8] HAW HAMBURG: *Homepage des Forschungsprojektes ESZ-ABS*. http://ieweb.etech.haw-hamburg.de/yamlt3/ESZ-ABS.313.0.html, Abruf: 19.02.2010

[9] HONEYWELL: *AN212 - Handling Sensor Bridge Offset*. Version: February 2005. http://www.ssec.honeywell.com/magnetic/datasheets/an212.pdf

[10] JEGENHORST, Niels: *Entwicklung eines Controllersystems zur Zustandserkennung von ABS-Sensoren*. Hamburg, Hochschule für Angewandte Wissenschaften, Diplomarbeit, Oktober 2009

[11] LINDNER, H. ; BRAUER, H. ; LEHMANN, C.: *Taschenbuch der Elektrotechnik und Elektronik*. 8. Aufl. Leipzig : Fachbuchverlag, 2005. – ISBN 978–3446210561

[12] MAHTOUF, Abdelkhalek: *Messungen und Signalanalyse an einem magnetischen Sensor*. Hamburg, Hochschule für Angewandte Wissenschaften, Diplomarbeit, Dezember 2008

[13] MIXED SIGNAL GROUP, SILICON & SOFTWARE SYSTEMS: *An Automotive Multi-Chip Module for Rotational Speed and Direction Measurement*. London : IEEE, 2000

[14] PHILIPS: *General Rotational speed measurement*. 1998

[15] PHILIPS: *KMI22/1 Rotational speed sensor for extended air gap application and direction detection*. 2000

[16] RIEMSCHNEIDER, Karl-Ragmar: *Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren ESZ-ABS*. Hamburg, Hochschule für Angewandte Wissenschaften, Forschungsantrag, Oktober 2007

[17] SCHOERMER, Christian: *In Anfertigung: Entwicklung eines Präzisionsradmessplatzes*. Hamburg, Hochschule für Angewandte Wissenschaften, Diplomarbeit in Anfertigung

[18] TEXAS INSTRUMENTS: *MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller (SLAS368F)*. Version: Mai 2009, Oktober 2002. http://focus.ti.com/lit/ds/symlink/msp430f1611.pdf, Abruf: 24.07.2009

[19] TEXAS INSTRUMENTS: *MSP430x1xx Family User's Guide (SLAU049F)*. Version: 2006. http://focus.ti.com/lit/ug/slau049f/slau049f.pdf, Abruf: 24.07.2009

[20] TRINAMIC MOTION CONTROL: *USB-2-485 Interface Converter*. Version: 1.00, April 2006. http://www.trinamic.com/tmc/media/Downloads/interfaces/USB-2-485_Manual.pdf, Abruf: 05.10.2009

[21] TRINAMIC MOTION CONTROL: *PDx-140-42-SE Manual*. Version: 1.02, October 2008. http://www.trinamic.com/tmc/media/Downloads/PANdrives/PD-140-42/PD-140-42-SE_manual.pdf, Abruf: 05.10.2009

[22] TRINAMIC MOTION CONTROL: *PDx-140-42-SE TMCL Firmware Manual*. Version: 1.00, April 2009. http://www.trinamic.com/tmc/media/Downloads/PANdrives/PD-140-42/PD-140_TMCL_firmware_manual.pdf, Abruf: 05.10.2009

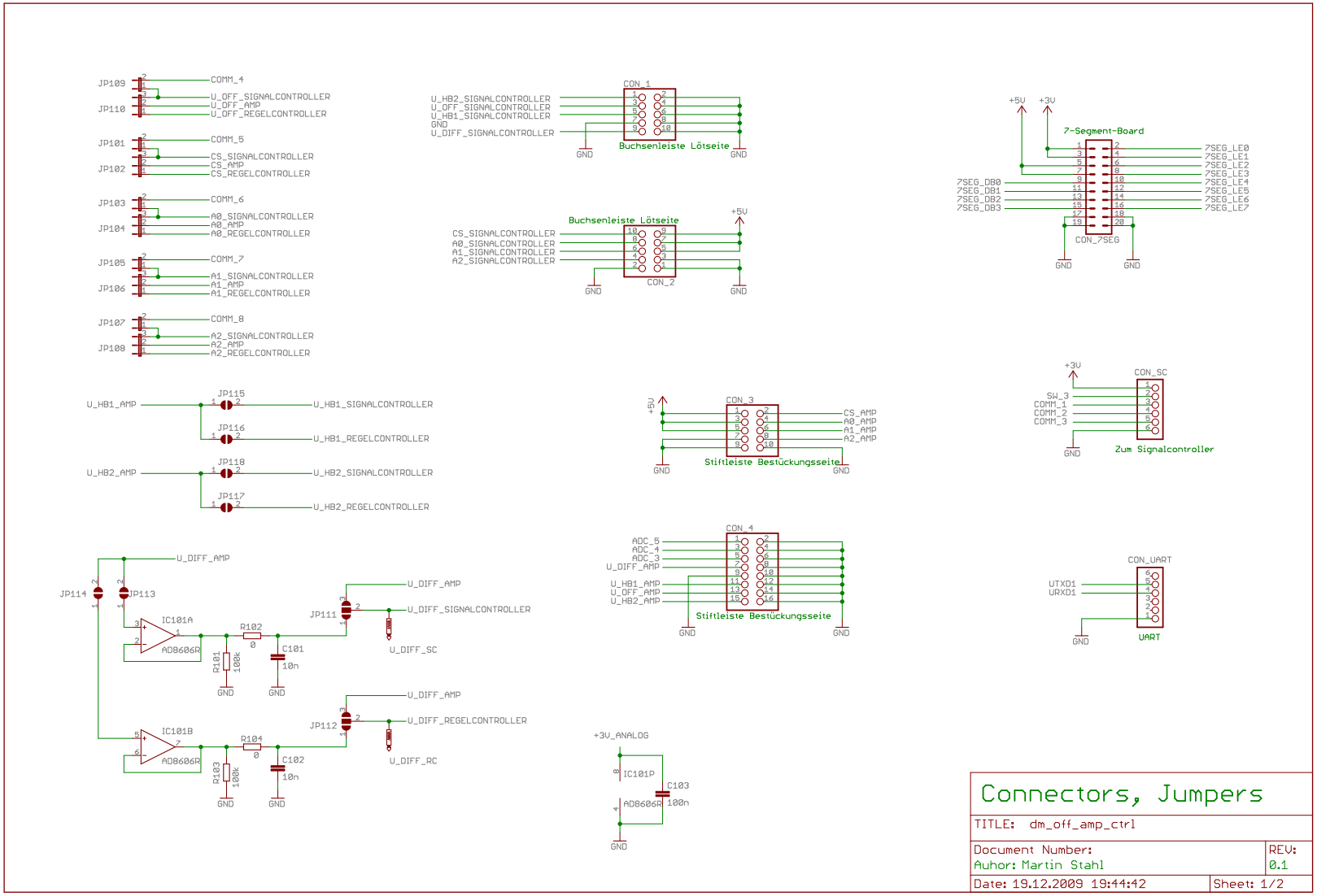
A Schaltpläne

A.1 Pinbelegungen

A.2 Schaltplan

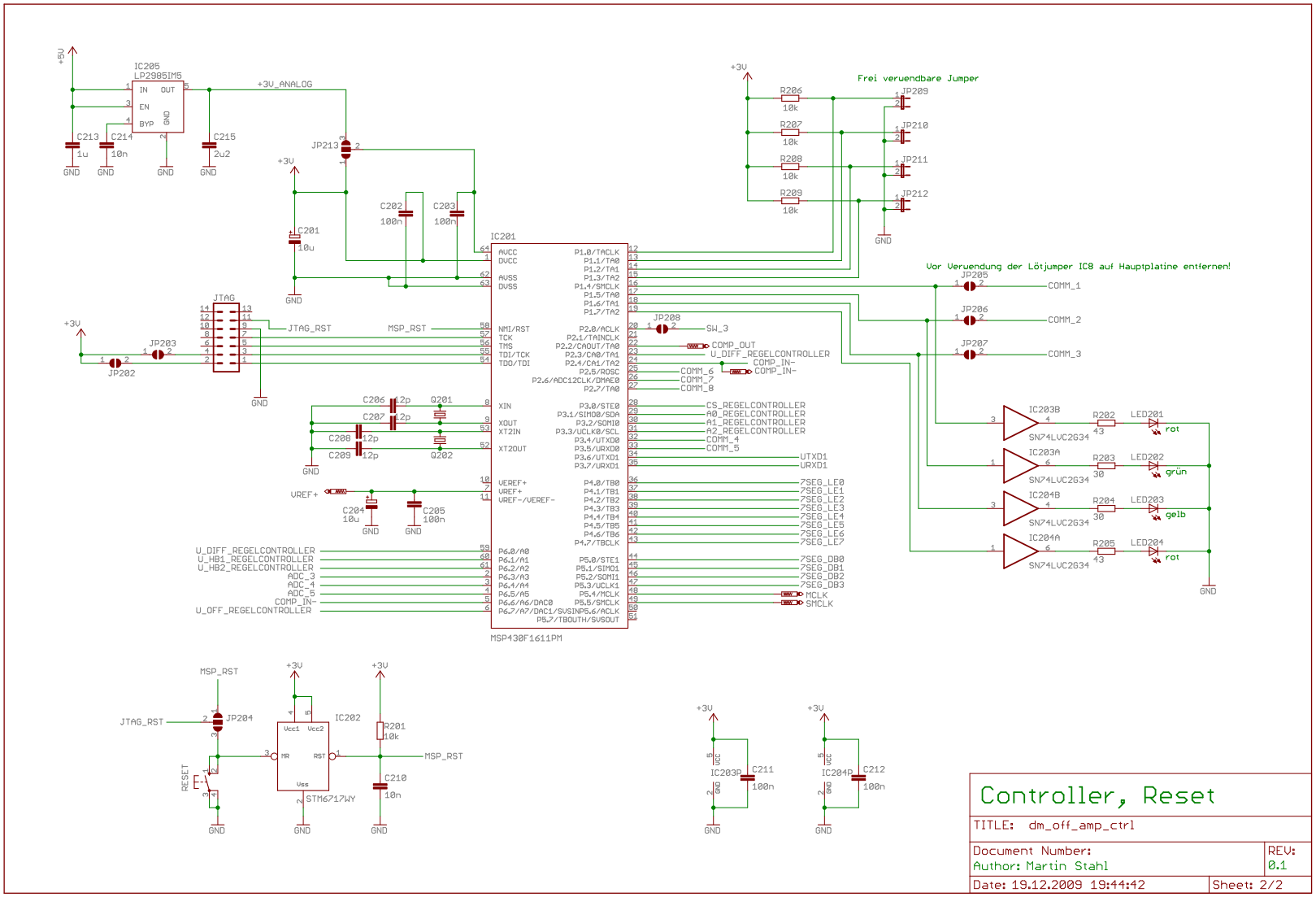
Port	Funktion	Datenrichtung	Signalname
1	P1.0	input	FREE_JUMPER_1
	P1.1	input	FREE_JUMPER_2
	P1.2	input	FREE_JUMPER_3
	P1.3	input	FREE_JUMPER_4
	P1.4	output	LED_1
	P1.5	output	LED_2
	P1.6	output	LED_3
	P1.7	output	LED_4
2	P2.0	input	SW_3
	P2.1	-	not used
	CAOUT	output	COMP_OUT
	CA0	input	U_DIFF_REGELCONTROLLER
	CA1	output	COMP_IN
	P2.5	output	COMM_6
	P2.6	output	COMM_7
	P2.7	output	COMM_8
3	P3.0	output	CS_REGELCONTROLLER
	P3.1	output	A0_REGELCONTROLLER
	P3.2	output	A1_REGELCONTROLLER
	P3.3	output	A2_REGELCONTROLLER
	P3.4	output	COMM_4
	P3.5	output	COMM_5
	UTXD1	output	UTXD1
	URXD1	output	URXD1
4	P4.0	output	7SEG_LE0
	P4.1	output	7SEG_LE1
	P4.2	output	7SEG_LE2
	P4.3	output	7SEG_LE3
	P4.4	output	7SEG_LE4
	P4.5	output	7SEG_LE5
	P4.6	output	7SEG_LE6
	P4.7	output	7SEG_LE7
5	P5.0	output	7SEG_DB0
	P5.1	output	7SEG_DB1
	P5.2	output	7SEG_DB2
	P5.3	output	7SEG_DB3
	MCLK	output	MCLK
	SMCLK	output	SMCLK
	P5.6	-	not used
	P5.7	-	not used
6	A0	input	U_DIFF_REGELCONTROLLER
	A1	input	U_HB1_REGELCONTROLLER
	A2	input	U_HB2_REGELCONTROLLER
	A3	input	ADC_3
	A4	input	ADC_4
	A5	input	ADC_5
	DAC0	output	COMP_IN-
	DAC1	output	U_OFF_REGELCONTROLLER

Tabelle A.1: *Pinbelegung des Regelcontrollers*



Connectors, Jumpers

TITLE: dm_off_amp_ctrl	
Document Number: Auhor: Martin Stahl	REV: 0.1
Date: 19.12.2009 19:44:42	Sheet: 1/2


Frei verwendbare Jumper
Vor Verwendung der Lötjumper IC8 auf Hauptplatine entfernen!
IC201
MSP430F1611PM
IC205
LP2985IM5
+3V_ANALOG
JTAG
JTAG_RST
MSP_RST
IC202
STM6717WY
RESET
IC203B
IC203A
IC204B
IC204A
SN74LVC2G34
COMP_OUT
U_DIFF_REGELCONTROLLER
COMP_IN-
CS_REGELCONTROLLER
A0_REGELCONTROLLER
A1_REGELCONTROLLER
A2_REGELCONTROLLER
UTXD1
URXD1
7SEG_LE0
7SEG_LE1
7SEG_LE2
7SEG_LE3
7SEG_LE4
7SEG_LE5
7SEG_LE6
7SEG_LE7
7SEG_DB0
7SEG_DB1
7SEG_DB2
7SEG_DB3
MCLK
SMCLK
COMM_1
COMM_2
COMM_3
SW_3
U_HB1_REGELCONTROLLER
U_HB2_REGELCONTROLLER
U_OFF_REGELCONTROLLER
ADC_3
ADC_4
ADC_5
VREF+
rot
grün
gelb
Controller, Reset
TITLE: dm_off_amp_ctrl
Document Number:
Author: Martin Stahl
REV: 0.1
Date: 19.12.2009 19:44:42
Sheet: 2/2

A.3 Layout

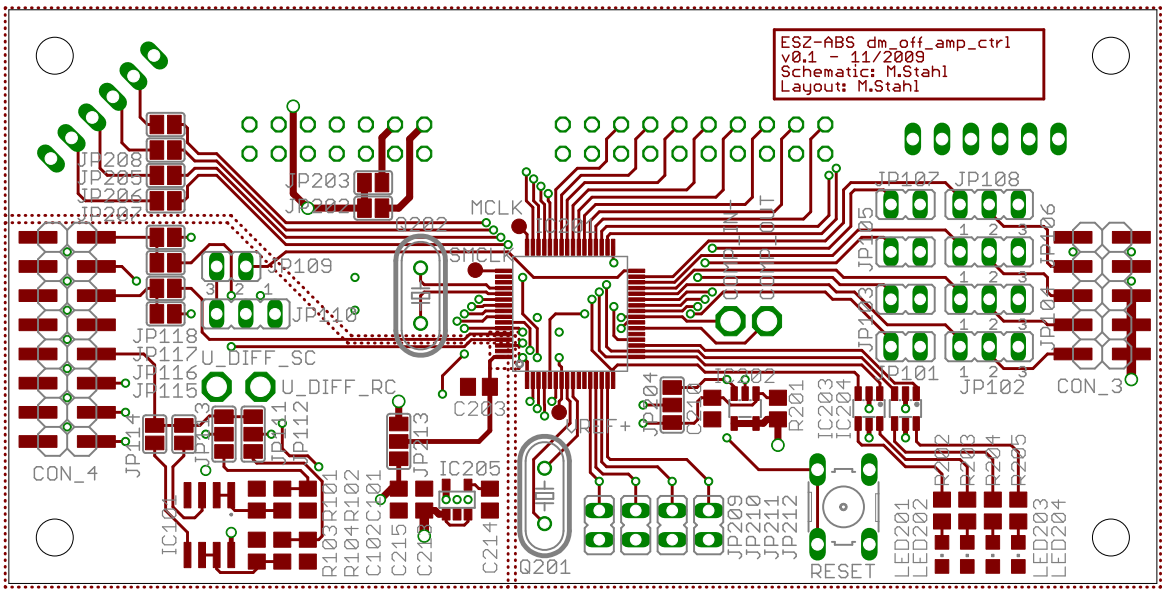


Abbildung A.1: *Layout Oberseite, Masseflächen sind ausgeblendet*

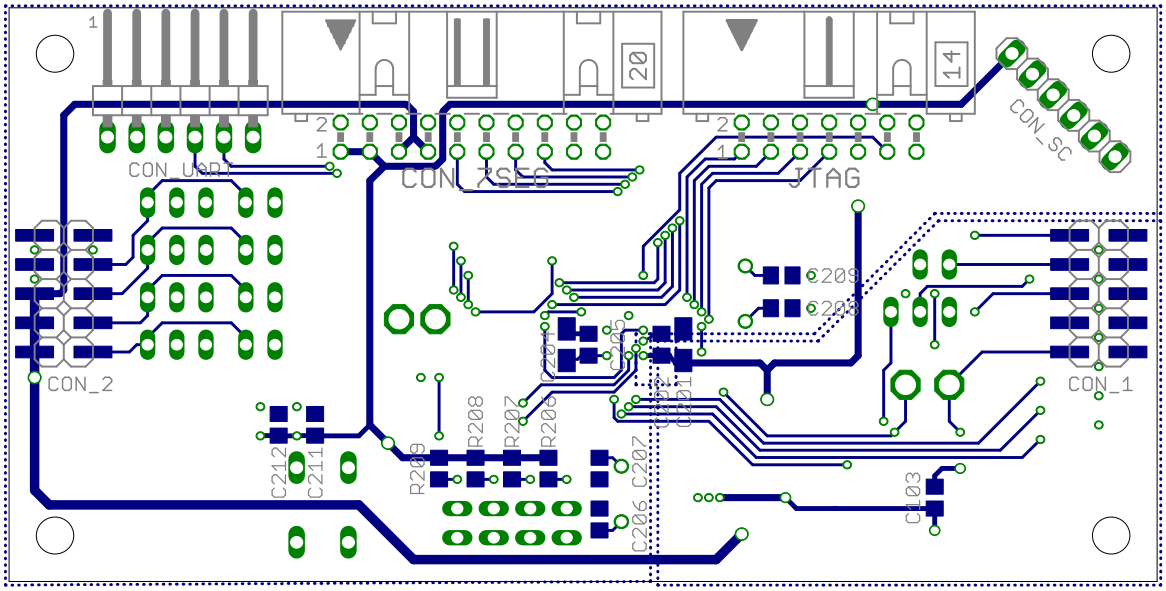


Abbildung A.2: *Layout Unterseite, Masseflächen sind ausgeblendet*

A.4 Bestückung

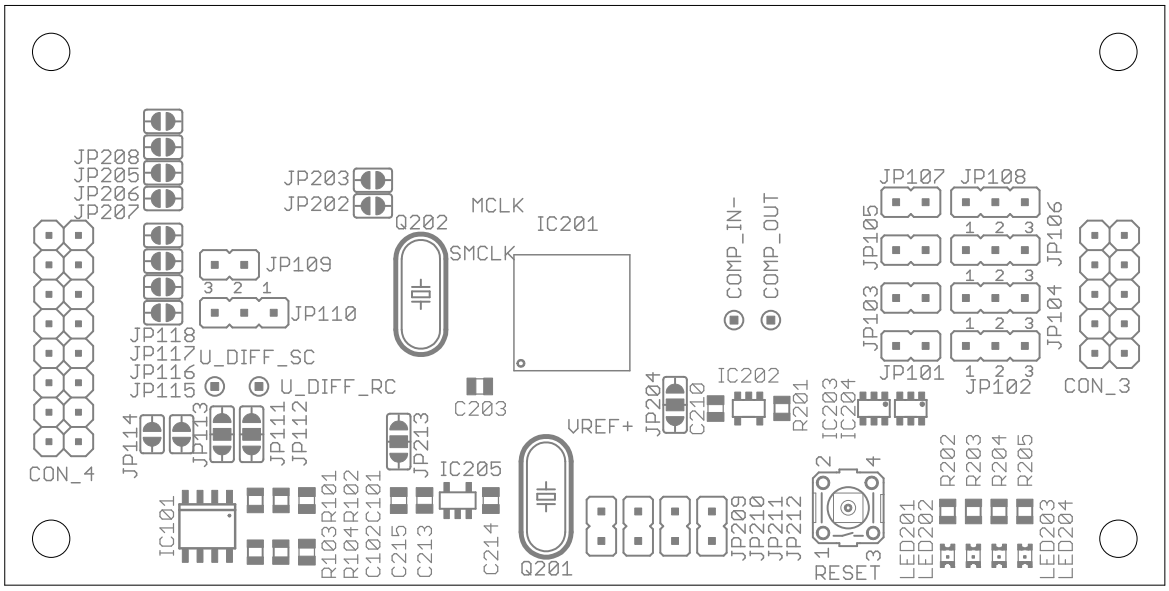


Abbildung A.3: *Bestückung Oberseite*

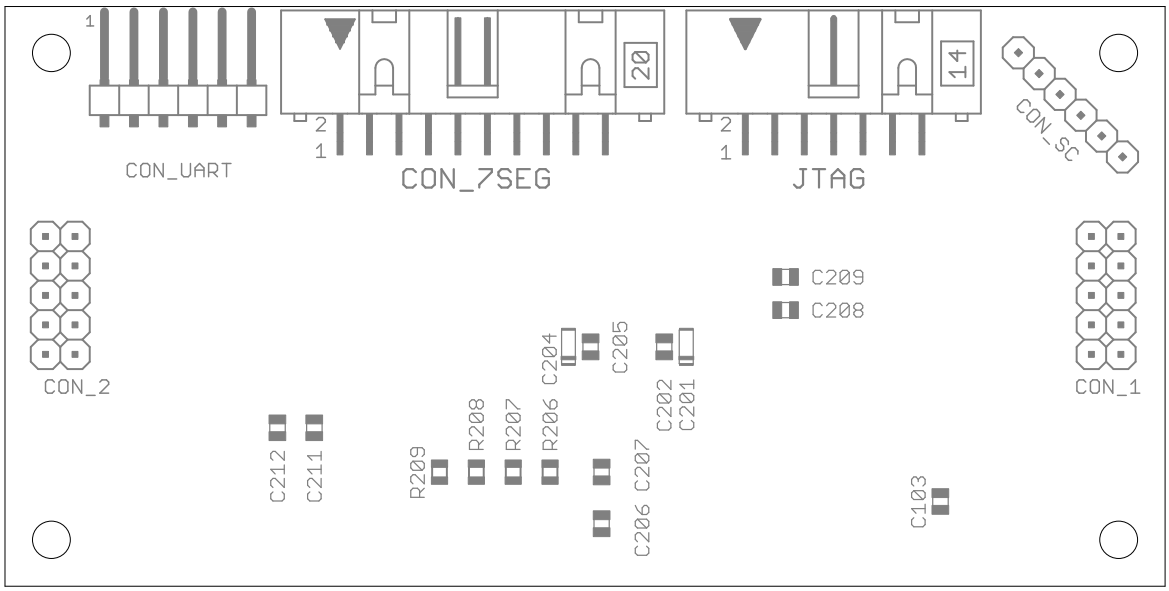


Abbildung A.4: *Bestückung Unterseite*

A.5 Stückliste

Menge	Wert	Device	Bauteile
1	Taster	10-XX	RESET
2	nicht Bestückt	CRYSTALHC49S	Q201, Q202
9	Lötjumper	JP1E	JP101, JP103, JP105, JP107, JP109, JP209, JP210, JP211, JP212
5	Lötjumper	JP2E	JP102, JP104, JP106, JP108, JP110
4	rot, grün, gelb, rot	LEDCHIPLED_0805	LED201, LED202, LED203, LED204
1	gewinkelt	ML14L	JTAG
1	gewinkelt	ML20L	CON_7SEG
1	gerade	PINHD-1X6_2.54	CON_SC
1	gewinkelt	PINHD-1X6_2.54-90°	CON_UART
3	gerade	PINHD-2X5_2.54-SMD	CON_1, CON_2, CON_3
1	gerade	PINHD-2X8_2.54-SMD	CON_4
12	Lötjumper	SJ	JP113, JP114, JP115, JP116, JP117, JP118, JP202, JP203, JP205, JP206, JP207, JP208
4	Lötjumper	SJ2W	JP111, JP112, JP204, JP213
2	0	R-EU_R0805	R102, R104
1	1u	C-EUC0805	C213
1	2u2	C-EUC0805	C215
5	10k	R-EU_R0805	R201, R206, R207, R208, R209
4	10n	C-EUC0805	C101, C102, C210, C214
2	10u	CPOL-EUA/3216-18R	C201, C204
4	12p	C-EUC0805	C206, C207, C208, C209
2	30	R-EU_R0805	R203, R204
2	43	R-EU_R0805	R202, R205
2	100k	R-EU_R0805	R101, R103
6	100n	C-EUC0805	C103, C202, C203, C205, C211, C212
1	AD8606R	AD8606R	IC101
1	LP2985IM5	LP2985IM5	IC205
1	MSP430F1611PM	MSP430F1611PM	IC201
3	Testpad	PTR1B1,27	MCLK, SMCLK, VREF+
4	Testpin	PTR1PAD1-17	COMP_IN-, COMP_OUT, U_DIFF_RC, U_DIFF_SC
2	SN74LVC2G34	SN74LVC2G34	IC203, IC204
1	STM6717WY	STM6717WY	IC202

Abbildung A.5: *Stückliste Regelplatine*

B Quellcode

B.1 Firmware Regelcontroller

Die folgende Tabelle gibt eine Übersicht, welche Funktionen in den einzelnen Quelldatein enthalten sind.

Datei	*Funktion*	*Seite*
main.c	main()	87
main.h	–	88
global.h	–	90
init.c	init()	91
calc_other.c	check_gain(), calc_offset()	94
cleanup.c	cleanup(),	96
isr_adc12.c	isr_adc12()	97
isr_comp_a.c	isr_comp_a()	98
isr_timer_a1.c	isr_timer_a1()	102
send.c	set_compensation_state(), send_usart(),	103
set_7seg.c	set_7seg()	104
set_gain.c	set_gain(),	106

Tabelle B.1: *Zuordnung von C-Dateien und enthaltenen Funktionen*

```
/*****************************************************************************************/
/*                                                                                       */
/*  Bachelorthesis Martin Stahl                                                          */
/*                                                                                       */
/*  Titel:          Controllersystem zur Verstärkungsregelung und Offsetkompensation     */
/*  Autor:          Martin Stahl                                                         */
/*  Datum:          16.12.2009                                                           */
/*  Hardware:       dm_off_amp_ctrl v0.1                                                 */
/*  Controller:     MSP430f1611 => "Regelcontroller"                                     */
/*  Toolchain:      MSPGCC v.20060502, Eclipse ganymede-SR2, USBExpress v1.021           */
/*                                                                                       */
/*  Datei:          main.c                                                               */
/*  Beschreibung:   Aufruf der Initialisierung des Systems                               */
/*                  In der Endlosschleife erfolgt lediglich die Ausgabe von Daten        */
/*                  auf der Display-Platine => Niedrigste Priorität                      */
/*                  Alle übrigen Funktionen werden interruptgesteuert ausgeführt         */
/*                                                                                       */
/*  Funktion:       ok                                                                   */
/*                                                                                       */
/*****************************************************************************************/


#define EXTERN

#include "main.h"
#include "global.h"

/*************************************************************/
/*                        Hauptschleife                      */
/*************************************************************/
int main(void)
{

        init(); // Initialisierung des Systems

        eint(); // Interrupts zulassen


        while(TRUE) {
            switch (sensor_state) {
                case Initial_0Hz:       //Sensor State: Initial_0Hz
                    #ifdef __USART_HD
                        send_usart(0xff);   // Nachricht Aufnahme beenden
                    #endif
                    PORT_LED &= ~(LED2 | LED1);
                    PORT_LED |= LED3; //gelb
                    set_7seg(1, 1, 0x10);
                    break;
                case Initial_1Hz:       // Sensor State: Initial_1Hz
                    PORT_LED |= LED1; //rot
                    set_7seg(2, 2, 0x20);
                    switch(state_periode) {
                    case Search_S1:
                        set_7seg(2, 2, 0x21);
                        break;
                    case Search_S2:
                        set_7seg(2, 2, 0x22);
                        break;
                    case Search_S1_Calc:
                        set_7seg(2, 2, 0x23);
                        break;
                    case Search_S2_Pre:
                        set_7seg(2, 2, 0x24);
                        break;
                    case Search_S2_Check_Gain:
                        set_7seg(2, 2, 0x25);
                        break;
                    case Search_S1_Check_Gain:
                        set_7seg(2, 2, 0x26);
                        break;
                    }
                    break;
                case Active:        // Sensor State: Active
                    set_7seg(3, 3, 0x30);
                    break;
                default: break;
            }//Ende switch(sensor_state)
        }// Ende while(TRUE)
} // Ende main()
```

```
/*****************************************************************************************/
/*                                                                                       */
/*  Bachelorthesis Martin Stahl                                                          */
/*                                                                                       */
/*  Titel:          Controllersystem zur Verstärkungsregelung und Offsetkompensation     */
/*  Autor:          Martin Stahl                                                         */
/*  Datum:          16.12.2009                                                           */
/*  Hardware:       dm_off_amp_ctrl v0.1                                                 */
/*  Controller:     MSP430f1611 => "Regelcontroller"                                     */
/*  Toolchain:      MSPGCC v.20060502, Eclipse ganymede-SR2, USBExpress v1.021           */
/*                                                                                       */
/*  Datei:          main.h                                                               */
/*  Beschreibung:   Header-File - enthält alle Definitionen und Prototypen               */
/*                                                                                       */
/*  Funktion:       ok                                                                   */
/*                                                                                       */
/*****************************************************************************************/

#ifndef MAIN_H_
#define MAIN_H_

    #include <msp430x16x.h>
    #include <signal.h>


/******* Allgemeine Definitionen ******/
    #define FALSE   0
    #define TRUE    !FALSE


/******* Spezielle Definitionen *******/

    #define __OFFSET_CAL            // definieren wenn Offset Kompensation
    #define __INIT_FALL_BACK        // Rücksprung von Active --> Init_0Hz

/******* Makros für Offsetkompensation  *******/

                    // Regelung im Active-Mode wird zugelassen
    #define __ACTIVE_COMPENSATION
                // 2^x Perioden fuer die Mittelwertbildung des Duty-Cycles
    #define PERIODES_FOR_MEAN_DUTY  7
                //Toleranzfenster für die Duty-Cycle-Korrektur
    #define DUTY_TOLERANCE  4
                //Periodenanzahl abhängig vom Verstärkungsfaktor
    #define __MEAN_DUTY_DEPEND_ON_GAIN
        // Anzahl an durchzufuehrenden Offset-Kompensationen im Zustand Initial_1Hz
    #define COMPENSATIONS   6

    #define GAIN_START      7   // Gain in Exponent von 2
                //Schwellwert-Differenz zum Offset => Mindestverstärkrung (300mV)
    #define LEVEL_GMIN      492
                //Schwellwert-Differenz zum Offset => maximale Verstärkung (940mV)
    #define LEVEL_GMAX      1542//+200
                //Anzahl der Samples, die die Schwellwerte überschreiten müssen
    #define MIN_SAMPLES_PASSED 2

/********* Timer-Definitionen *********/

    #define TIMER_A_PER     13488       // Periodendauer Timer A in ns
                                        //  = T_SMCLK * 8 in ns
    #define TCCR_TEST       30000       // --> T_SMCLK = 1686ns (div1)

    #define SAMPLETIME      800         // 4,9MHz/800=6,125kHz
    #define TIMER_B_PER_F   169247      // Periodendauer Timer B in ns * 10
    #define TIMER_B_PER     1693        // Periodendauer Timer B in ns
                                        // Festlegung des Prescalers Timer B7:
/********** DAC-Definitionen **********/

    #define OFFSET_START    2048        // Offset, Anfangswert (1.25V)
    #define LEVEL_SX        164         // Schwellwert-Differenz zum Offset(0.1V)

    #define MIN_TIME_S1     4           // Minimale Zeit am Punkt S1
    #define MIN_TIME_S2     2           // Minimale Zeit am Punkt S2

/********** Pin-Definitionen **********/
    #define PIN0    0x01
    #define PIN1    0x02
    #define PIN2    0x04
    #define PIN3    0x08
    #define PIN4    0x10
    #define PIN5    0x20
    #define PIN6    0x40
    #define PIN7    0x80

    #define PORT_LED    P1OUT
    #define LED1    PIN4
    #define LED2    PIN5
    #define LED3    PIN6
    #define LED4    PIN7
```

```
    #define COMM_4  PIN4
    #define COMM_5  PIN5
    #define COMM_6  PIN5
    #define COMM_7  PIN6
    #define COMM_8  PIN7

/********* 7-Segment-Anzeigen *********/

    #define PORT_7SEG_LE    P4OUT
    #define PORT_7SEG_DB    P5OUT
    #define PINS_7SEG_DB    0x0F


/********** Verstaerker-Pins **********/

    #define PORT_AMP    P3OUT
    #define AMP_CS      PIN0
    #define AMP_A0      PIN1
    #define AMP_A1      PIN2
    #define AMP_A2      PIN3


/******** Zustandsdefinitionen ********/

    typedef enum {  // Systemzustände Sensor
        Initial_0Hz,
        Initial_1Hz,
        Active
    } states;

    typedef enum {  // Systemzustände Periodenerkennung
        Search_S1,
        Search_S1_Calc,
        Search_S2,
        Search_S2_Pre,
        Search_S1_Check_Gain,
        Search_S2_Check_Gain
    } states_per;

/******* Funktions-Prototypen ********/

    void init(void);
    void set_7seg(uint16_t frequenz, uint8_t thd, uint8_t status);
    uint8_t send_usart(uint8_t value);
    void set_gain(void);
    void cleanup(void);
    void calc_offset(void);
    void check_gain(void);
    void set_compensation_state(uint8_t state);

#endif
```

```
/*****************************************************************************************/
/*                                                                                       */
/*  Bachelorthesis Martin Stahl                                                          */
/*                                                                                       */
/*  Titel:          Controllersystem zur Verstärkungsregelung und Offsetkompensation     */
/*  Autor:          Martin Stahl                                                         */
/*  Datum:          22.12.2009                                                           */
/*  Hardware:       dm_off_amp_ctrl v0.1                                                 */
/*  Controller:     MSP430f1611 => "Regelcontroller"                                     */
/*  Toolchain:      MSPGCC v.20060502, Eclipse ganymede-SR2, USBExpress v1.021           */
/*                                                                                       */
/*  Datei:          global.h                                                             */
/*  Beschreibung:   Header-File - enthält globale Variablen                              */
/*                  Namensgebung basiert auf Diplomarbeit NJ                             */
/*                                                                                       */
/*  Funktion:       ok                                                                   */
/*                                                                                       */
/*****************************************************************************************/

#ifndef GLOBAL_H_
#define GLOBAL_H_

    #ifndef EXTERN
    #define EXTERN extern
    #endif

    struct {
        uint32_t duty;              // Tastverhältnis
        uint32_t offset;            // Ausgangswert für DAC12_1
        int16_t  offset_faktor;     // Faktor für Offset-Änderung
        uint8_t  compensations;     // Kompensationen der Offset Kompensation
        uint8_t  gain;              // Verstärung des 2.TIA
        uint16_t g1_passed;         // maximale Verstärkung obere Halbwelle
        uint16_t g2_passed;         // minimale Verstärkung obere Halbwelle
        uint16_t g3_passed;         // minimale Verstärkung untere Halbwelle
        uint16_t g4_passed;         // maximale Verstärkung untere Halbwelle
        uint32_t duty_sum;          // Summe von x Duty-Cycles zut Mittelwertbildung
        uint16_t  duty_count;       // Zähler für aufsummierte Duty_Cycles
    } EXTERN volatile g_sig;

    struct {                        // ISR Comperator:
        uint16_t tp;                // halbe Periode
        uint16_t tt;                // Periodendauer
        uint16_t frequenz;          // Frequenz in Hz
        uint16_t sample_inst;       // Zeitpunkt zum Samplen
    } EXTERN volatile g_time;

    struct {                        // DAC:
        uint16_t u_diff;            // Differenz Signal
        int16_t  u_diff_b;          // Differenz Signal ohne Offset
        uint16_t u_hb1;             // Halbbrücken Signal 1
        uint16_t u_hb2;             // Halbbrücken Signal 2
        uint16_t u_sum1;            // Summe HB1 + HB2
    } EXTERN volatile g_adc;

/************** Zustaende des Systems  ***************/
    EXTERN volatile states      sensor_state;
    EXTERN volatile states_per  state_periode;

#endif
```

```
/*******************************************************************************************/
/*                                                                                         */
/*  Bachelorthesis Martin Stahl                                                            */
/*                                                                                         */
/*  Titel:          Controllersystem zur Verstärkungsregelung und Offsetkompensation       */
/*  Autor:          Martin Stahl                                                           */
/*  Datum:          16.12.2009                                                             */
/*  Hardware:       dm_off_amp_ctrl v0.1                                                   */
/*  Controller:     MSP430f1611 => "Regelcontroller"                                       */
/*  Toolchain:      MSPGCC v.20060502, Eclipse ganymede-SR2, USBExpress v1.021             */
/*                                                                                         */
/*  Datei:          init.c                                                                 */
/*  Beschreibung:   Initialisierung der Controller-Register und -Peripherie                */
/*                  Verwendet sind interner DCO, USART1, TimerA, TimerB, ComparartorA,     */
/*                  ADC12, DAC12_0, DAC12_1, div. I/O-Pins                                 */
/*                                                                                         */
/*  Funktion:       ok                                                                     */
/*                                                                                         */
/*******************************************************************************************/

#include "main.h"
#include "global.h"


/*************************************************************/
/*              Initialisierungsroutine                      */
/*************************************************************/

void init(void)
{

/************ Initialisierung interner Oszillator DCO = 4.9 MHz **************/

    // DCO  = 7, Mod ignored
    DCOCTL   = DCO2 | DCO1 | DCO0;
    // RSEL = 7, XT2 Oscillator off, ACLK Div = 1
    BCSCTL1  = RSEL2 | RSEL1 | RSEL0 | XT2OFF;
    // MCLK von DCOCLK, SMCLK von DCOCLK, SMCLK Div = 8
    //BCSCTL2 |= (DIVS0 | DIVS1);



/********************** Initialisierung USART1 ***********************/
    //zum debuggen

        // USART1 Control Reg.: Parität = even, StopBits = 2, CharLength = 8, UART Mode
    U1CTL  = PENA | PEV | SPB | CHAR;
        // USART1 Transmit Control Reg.:    Clock = SMCLK, Transmit Empty Flag = 1
    U1TCTL = SSEL_SMCLK | TXEPT;
        // USART1 Receive Control Reg.:
    U1RCTL = 0x00;
        // USART1 Baud Rate Control Reg.0:  (U1BR1+U1BR0) = U1BR = SMCLK / BaudRate
        //                                  BaudRate = 115200 --> N = 5.1485, UBR = 5
    U1BR0  = 0x05;
        // USART1 Baud Rate Control Reg.1:
    U1BR1  = 0x00;
        // USART1 Modulation Control Reg.:  1/n * UMOD = 1/12 * 2 = 0.166 --> N = 5.166
    U1MCTL = 0x90;
        // Module Enable Reg.2:     Transmitter disable, Receiver disable
    ME2   &= ~(UTXE1 | URXE1);
        // Interrupt Flag Reg.2:    Flags löschen
    IFG2  &= ~(UTXIFG1 | URXIFG1);
        // Interrupt Enable Reg.2:  Transmit IRQ disable
    IE2   &= ~UTXIE1;


/********************** Initialisierung Timer A ***********************/
    // Zur Stillstandserkennung und Bestimmung des Tastverhältnisses

    // Timer Control Reg.: SMCLK, prescaler=8, clear TAR, IRQ en
    TACTL   = TACLR;
    TACTL   = TASSEL1 | ID1 | ID0 | TAIE;
    TAR     = 0x0000;

    // Timer Capture/Compare Control Reg.0:
    TACCTL0 = 0x0000;                       // IRQ aus
    TACCR0  = 0x0000;
    TACCTL1 = 0x0000;
    TACCR1  = 0x0000;
    TACCTL2 = 0x0000;
    TACCR2  = 0x0000;


/********************** Initialisierung Timer B ***********************/
    //Triggert den ADC

    // Timer Control Reg.: SMLK, prescaler=1, clear TAR, IRQ en
    TBCTL   = TBCLR;
    TBCTL   = TBSSEL1;
//  TBCCTL0 |= CCIE;
```

```
    TBR     = 0x0000;

    // Timer Capture/Compare Control Reg.0:
    TBCCTL0 |= OUTMOD_3;
    TBCCR0  = SAMPLETIME-1;             // 0xFFFF - 1
    TBCCTL1 = 0x0000;                   // Output Mode 1 wird erst später gesetzt
    TBCCR1  = 0x0000;
    TBCCTL2 = 0x0000;
    TBCCR2  = 0x0000;
    TBCCTL3 = 0x0000;
    TBCCR3  = 0x0000;
    TBCCTL4 = 0x0000;
    TBCCR4  = 0x0000;
    TBCCTL5 = 0x0000;
    TBCCR5  = 0x0000;
    TBCCTL6 = 0x0000;
    TBCCR6  = 0x0000;


/********************* Initialisierung Comparator ***********************/

    // Comperator Control Reg.1: no exchange, Internal reference off, Comperator off,
    //                           IRQ rising edge, IRQ disable, Flag delete,
    //                           CA0 pos. - CA1 neg. Input
    CACTL1 = 0x00;
    // Comperator Control Reg.2: P2CA1 -> CA1, P2CA0 -> CA0, output filtered
    CACTL2 = P2CA1 | P2CA0 | CAF;
    // Comperator Port Disable Reg.: input buffer P2.2, P2.3, P2.4 disable
    CAPD   = CAPD2 | CAPD3 | CAPD4;


/************************ Initialisierung ADC ***************************/

    // ADC12 Control Reg.0: Sample & Hold Time = 32 , Reference = 2.5V, Reference on,
    //                      ADC12 off, IRQ MEMx overflow,
    //                      IRQ conversion time overflow,
    ADC12CTL0 = SHT0_DIV8 | REF2_5V | REFON | ADC12OVIE | ADC12TOVIE | ADC12ON;
    // ADC12 Control Reg.1: MCTL0 = first sample, Clock = MCLK, Sequence of channels
    //                      Trigger by TBCCR0
    ADC12CTL1 = CSTARTADD_0 | ADC12SSEL_MCLK | SHS_TBCCR0 | ADC12DIV_0 |
                SHP | CONSEQ_REPEAT_SINGLE;
    // ADC12 Conversion Memory Control Reg.0: Reference = Vref+ / AVss, Channel = A0
    ADC12MCTL0 = SREF_VREF_AVSS | INCH_0;
    // ADC12 Conversion Memory Control Reg.1: Reference = Vref+ / AVss, Channel = A1
    //ADC12MCTL1 = SREF_VREF_AVSS | INCH_1;
    // ADC12 Conversion Memory Control Reg.1: Reference = Vref+ / AVss, Channel = A2
    //                                        End of Sequence
    //ADC12MCTL2 = SREF_VREF_AVSS | INCH_2 | EOS;
    // ADC12 Interrupt Enable Reg: MCTL 2 enable IRQ
    ADC12IE = 0x0001;


/************************ Initialisierung DAC ***************************/

    // DAC12_0 Control Reg.: Internal reference, 12-Bit, load when written,
    //                       Fullscale = 1*ref, Amp = high speed, binary format,
    //                       no IRQ, DAC12 off, not grouped, calibration on
    DAC12_0CTL = DAC12SREF_0 | DAC12LSEL_0 | DAC12CALON | DAC12IR | DAC12AMP_7;
    DAC12_0DAT = OFFSET_START + LEVEL_SX;   // zu Begin Level S1 ausgeben
    // DAC12_1 Control Reg.: Internal reference, 12-Bit, load when written,
    //                       Fullscale = 1*ref, Amp = high speed, binary format,
    //                       no IRQ, DAC12 off, not grouped, calibration on
    DAC12_1CTL = DAC12SREF_0 | DAC12LSEL_0 | DAC12CALON | DAC12IR | DAC12AMP_7;
    DAC12_1DAT = OFFSET_START;              // zu Begin 1.25V ausgeben


/********************* Initialisierung der I/O - Ports **********************/

    P1OUT  = 0x00;                              // Ausgangregister Port1 löschen
    P1DIR |= LED1 | LED2 | LED3 | LED4 | PIN3|PIN0;         // LED-Pins als Ausgang
    P1DIR &= ~( /*PIN0 |*/ PIN1 | PIN2);// | PIN3 );    // JP209...212 als Eingang
    P1SEL &= ~( LED1 | LED2 | LED3 | LED4 );    // IO-Mode für LED-Pins
    P1SEL &= ~( PIN0 | PIN1 | PIN2 | PIN3 );    // IO-Mode für JP209...212


    P2OUT  = 0x00;                      // Port2 Ausgangsregister LOW
    P2DIR  |= PIN2 | COMM_6 | COMM_7 | COMM_8;      //Ausgaenge an Signalcontroller
    P2SEL &= ~( PIN0 | PIN1 | PIN5 | PIN6 | PIN7 ); // IO-Mode
    P2SEL  |= PIN2 | PIN3 | PIN4;                   // Comparator Pins

    P3OUT  = 0x00;                      // Port3 Ausgangsregister LOW
    P3DIR |= AMP_CS | AMP_A0 | AMP_A1 | AMP_A2; // Verstärkungs-Einstellung
    P3DIR |= COMM_5; //Ausgang U_Diff not valid => Regelung aktiv wenn high
    P3SEL &= ~( PIN0 | PIN1 | PIN2 | PIN3 | PIN4 | PIN5 );  // IO-Mode
    P3SEL |= PIN6 | PIN7;                                   // UART Rx & Tx


    P4OUT  = 0x00;
    P4DIR |= 0xFF;  // 7Seg_LE0...7 als Ausgang
```

```
    P4SEL &= ~0xFF; // IO-Mode für 7Seg_LE0...7


    P5OUT  = 0x00;
    P5DIR |= PIN0 | PIN1 | PIN2 | PIN3;         // 7Seg_DB0...3 als Ausgang
    P5SEL |= PIN4 | PIN5;
    P5SEL &= ~( PIN0 | PIN1 | PIN2 | PIN3 );    // IO-Mode für 7Seg_DB0...3


    P6OUT  = 0x00;
    P6DIR  |= PIN6 | PIN7;
    P6SEL  |= PIN0 | PIN1 | PIN2 | PIN6 | PIN7; // ADC- und DAC-Pins


/********************* Initialisierung des Systems ***********************/

    // globale Variable zuruecksetzen
    cleanup();

    // 7 Segment Anzeige
    set_7seg(0x0000, 0x00, 0x00);

    // Verstärker
    g_sig.gain = GAIN_START; //maximale Verstärkung beim PowerOn
    set_gain();
    P3OUT |= COMM_5; //Regelung aktiv, U_Diff ungueltig


    sensor_state    = Initial_0Hz;                 // Power on --> Initial_0Hz
    state_periode   = Search_S1;

    while(TRUE) {   // warten bis DAC-Kalibration beendet
        if( ((DAC12_0CTL & DAC12CALON) != DAC12CALON) &&
            ((DAC12_1CTL & DAC12CALON) != DAC12CALON)   ) break;
    }

                //Timer_B triggert den ADC
    TBCTL |= TBCLR;                 // Timer B zurücksetzen
    TBCTL |= MC_1;                  // Timer B Start compare mode
    ADC12CTL0 &= ~ENC;          /* disable ADC */

    CACTL1 |= CAON | CAIE;                         // Comperator on, IRQ enable

}//Ende init()
```

```
/*****************************************************************************************/
/*                                                                                       */
/*  Bachelorthesis Martin Stahl                                                          */
/*                                                                                       */
/*  Titel:          Controllersystem zur Verstärkungsregelung und Offsetkompensation     */
/*  Autor:          Martin Stahl                                                         */
/*  Datum:          05.01.2010                                                           */
/*  Hardware:       dm_off_amp_ctrl v0.1                                                 */
/*  Controller:     MSP430f1611 => "Regelcontroller"                                     */
/*  Toolchain:      MSPGCC v.20060502, Eclipse ganymede-SR2, USBExpress v1.021           */
/*                                                                                       */
/*  Datei:          calc_other.c                                                         */
/*  Beschreibung:   check_gain() zur Ueberpruefung auf Ueberschr. der Gain-Schwellen     */
/*                  Berechnung des Offsetkorrekturfaktors basiert auf Diplomarbeit NJ    */
/*                                                                                       */
/*  Funktion:       ok                                                                   */
/*                                                                                       */
/*****************************************************************************************/

#include "main.h"
#include "global.h"


/*************************************************************/
/*               Überprüfung der Verstärkung                 */
/*************************************************************/

void check_gain(void) {

    switch(sensor_state) {

        case Initial_1Hz:
            // G1 und G4 wurden überschritten => Verstärkung ist zu groß
            if( g_sig.g1_passed > MIN_SAMPLES_PASSED || g_sig.g4_passed > MIN_SAMPLES_PASSED) {

                if(g_sig.gain>0) {
                    set_compensation_state(TRUE); //Regelung aktiv
                    g_sig.gain--;
                    set_gain();
                }


                state_periode = Search_S2_Pre;
            }

            else {
                set_compensation_state(FALSE); //Diagnose freigeben
                state_periode = Search_S2_Pre;
                sensor_state  = Active;     // next Sensor State
            }
            break;

        case Active:
            // G2 und G3 wurden nicht erreicht => Verstärkung ist zu klein
            if( !g_sig.g2_passed  || !g_sig.g3_passed ) {

                if(g_sig.gain<7) {
                    set_compensation_state(TRUE);
                    g_sig.gain++;
                    set_gain();
                }
                state_periode = Search_S2_Pre;
            }
            // G1 und G4 wurden überschritten => Verstärkung ist zu groß
            else if( g_sig.g1_passed > MIN_SAMPLES_PASSED || g_sig.g4_passed > MIN_SAMPLES_PASSED) {

                if(g_sig.gain>0) {
                    set_compensation_state(TRUE);
                    g_sig.gain--;
                    set_gain();
                }
                state_periode = Search_S2_Pre;
            }

            else{
                set_compensation_state(FALSE);
                state_periode = Search_S2_Pre;
            }
            break;

        default:
            break;

    }
    //Schwellenzähler zurücksetzten
    g_sig.g1_passed = 0x0000;
    g_sig.g2_passed = 0x0000;
    g_sig.g3_passed = 0x0000;
    g_sig.g4_passed = 0x0000;
```

```
}

/*************************************************************/
/*              Berechnung des Offsets                       */
/*************************************************************/

void calc_offset(void) {

        // Berechnung des Offset's anhand des Duty-Cycle

                                        // Offset - Korrekturtabelle
    if(g_sig.duty >= 122)       g_sig.offset_faktor = 32;
    else if(g_sig.duty >= 110)  g_sig.offset_faktor = 29;
    else if(g_sig.duty >= 101)  g_sig.offset_faktor = 25;
    else if(g_sig.duty >=  97)  g_sig.offset_faktor = 23;
    else if(g_sig.duty >=  94)  g_sig.offset_faktor = 21;
    else if(g_sig.duty >=  90)  g_sig.offset_faktor = 19;
    else if(g_sig.duty >=  87)  g_sig.offset_faktor = 17;
    else if(g_sig.duty >=  84)  g_sig.offset_faktor = 15;
    else if(g_sig.duty >=  81)  g_sig.offset_faktor = 13;
    else if(g_sig.duty >=  79)  g_sig.offset_faktor = 11;
    else if(g_sig.duty >=  76)  g_sig.offset_faktor = 9;
    else if(g_sig.duty >=  73)  g_sig.offset_faktor = 7;
    else if(g_sig.duty >=  70)  g_sig.offset_faktor = 5;
    else if(g_sig.duty ==  69)  g_sig.offset_faktor = 4;
    else if(g_sig.duty >=  66)  g_sig.offset_faktor = 2;
    else if(g_sig.duty ==  65)  g_sig.offset_faktor = 1;
    else if(g_sig.duty ==  64)  g_sig.offset_faktor = 0;
    else if(g_sig.duty ==  63)  g_sig.offset_faktor = -1;
    else if(g_sig.duty >=  60)  g_sig.offset_faktor = -3;
    else if(g_sig.duty >=  57)  g_sig.offset_faktor = -5;
    else if(g_sig.duty >=  55)  g_sig.offset_faktor = -7;
    else if(g_sig.duty >=  52)  g_sig.offset_faktor = -9;
    else if(g_sig.duty >=  49)  g_sig.offset_faktor = -11;
    else if(g_sig.duty >=  46)  g_sig.offset_faktor = -13;
    else if(g_sig.duty >=  43)  g_sig.offset_faktor = -15;
    else if(g_sig.duty >=  40)  g_sig.offset_faktor = -17;
    else if(g_sig.duty >=  37)  g_sig.offset_faktor = -19;
    else if(g_sig.duty >=  34)  g_sig.offset_faktor = -21;
    else if(g_sig.duty >=  30)  g_sig.offset_faktor = -23;
    else if(g_sig.duty >=  26)  g_sig.offset_faktor = -25;
    else if(g_sig.duty >=  22)  g_sig.offset_faktor = -27;
    else if(g_sig.duty >=  16)  g_sig.offset_faktor = -29;
    else if(g_sig.duty >=   7)  g_sig.offset_faktor = -31;
    else if(g_sig.duty <=   6)  g_sig.offset_faktor = -32;

    g_sig.offset_faktor <<= 7;              // * 128
                                            // 64 * 128 + (Faktor / Verstärkung)
    g_sig.offset_faktor = 8192 + (g_sig.offset_faktor >> g_sig.gain);
                                            // maximal: 32*128 + 64*128 = 15 Bit + VZ
    g_sig.offset *= (int32_t) g_sig.offset_faktor;
    g_sig.offset >>= 13;                    // / 64*128
}
```

```
/*****************************************************************************************/
/*                                                                                       */
/*  Bachelorthesis Martin Stahl                                                          */
/*                                                                                       */
/*  Titel:          Controllersystem zur Verstärkungsregelung und Offsetkompensation     */
/*  Autor:          Martin Stahl                                                         */
/*  Datum:          22.12.2009                                                           */
/*  Hardware:       dm_off_amp_ctrl v0.1                                                 */
/*  Controller:     MSP430f1611 => "Regelcontroller"                                     */
/*  Toolchain:      MSPGCC v.20060502, Eclipse ganymede-SR2, USBExpress v1.021           */
/*                                                                                       */
/*  Datei:          cleanup.c                                                            */
/*  Beschreibung:   Funktion zum zuruecksetzten aller globalen Variablen                 */
/*                  Basiert auf Diplomarbeit NJ                                          */
/*                                                                                       */
/*  Funktion:       ok                                                                   */
/*                                                                                       */
/*****************************************************************************************/

#include "main.h"
#include "global.h"


/*************************************************************/
/*              Zuruecksetzten der globalen Variablen        */
/*************************************************************/

void cleanup(void)
{
    PORT_LED &= ~(LED1 | LED2 | LED3 | LED4); //LEDs ausschalten
    P3OUT |= COMM_5; //Regelung aktiv => U_Diff not valid

    g_sig.offset        = OFFSET_START;
    g_sig.compensations = 0x00;
    g_sig.duty          = 0x00000000;
    g_sig.gain          = GAIN_START;
    g_sig.offset_faktor = 0x0000;
    g_sig.g1_passed     = 0x00;
    g_sig.g2_passed     = 0x0000;
    g_sig.g3_passed     = 0x0000;
    g_sig.g4_passed     = 0x0000;
    g_sig.duty_sum      = 0x00000000;
    g_sig.duty_count    = 0x0000;

    set_gain();

    g_adc.u_diff    = 0x0000;
    g_adc.u_hb1     = 0x0000;
    g_adc.u_hb2     = 0x0000;
    g_adc.u_diff_b  = 0x0000;
    g_adc.u_sum1    = 0x0000;

    g_time.tp       = 0x0000;
    g_time.tt       = 0x0000;

}
```

```
/*****************************************************************************************/
/*                                                                                       */
/*  Bachelorthesis Martin Stahl                                                          */
/*                                                                                       */
/*  Titel:          Controllersystem zur Verstärkungsregelung und Offsetkompensation     */
/*  Autor:          Martin Stahl                                                         */
/*  Datum:          05.01.2010                                                           */
/*  Hardware:       dm_off_amp_ctrl v0.1                                                 */
/*  Controller:     MSP430f1611 => "Regelcontroller"                                     */
/*  Toolchain:      MSPGCC v.20060502, Eclipse ganymede-SR2, USBExpress v1.021           */
/*                                                                                       */
/*  Datei:          isr_adc12.c                                                          */
/*  Beschreibung:   Interrupt Service Routine für ADC12                                  */
/*                  Ueberprüft aktuellen Sample auf Ueberschreitung der Scwhellen        */
/*                                                                                       */
/*  Funktion:       ok                                                                   */
/*                                                                                       */
/*****************************************************************************************/

#include "main.h"
#include "global.h"


/*************************************************************/
/*         Interrupt Service Routine ADC_12                  */
/*************************************************************/
interrupt ( ADC12_VECTOR ) isr_adc12 ( void )
{

    if(ADC12IFG & 0x0001)
            g_adc.u_diff = ADC12MEM0;              // Channel 0 auslesen

    switch(state_periode) {

        /********** obere Halbwelle des Differenzsignals ***********/
        case Search_S2_Check_Gain:

            if(g_adc.u_diff > OFFSET_START+LEVEL_GMIN) //Schwelle G2 überschritten
                g_sig.g2_passed++;

            if(g_adc.u_diff > OFFSET_START+LEVEL_GMAX) //Schwelle G1 überschritten
                g_sig.g1_passed++;

            break;

        /********** untere Halbwelle des Differenzsignals **********/
        case Search_S1_Check_Gain:

            if(g_adc.u_diff < OFFSET_START-LEVEL_GMIN) //Schwelle G3 überschritten
                g_sig.g3_passed++;

            if(g_adc.u_diff < OFFSET_START-LEVEL_GMAX) //Schwelle G4 überschritten
                g_sig.g4_passed++;

            break;

        default:
            break;
    } //Ende switch(state_periode)
    //ADCIFG wird beim auslesen des ADC12MEM0 Registers gelöscht
}
```

```
/*****************************************************************************************/
/*                                                                                       */
/*  Bachelorthesis Martin Stahl                                                          */
/*                                                                                       */
/*  Titel:          Controllersystem zur Verstärkungsregelung und Offsetkompensation     */
/*  Autor:          Martin Stahl                                                         */
/*  Datum:          05.01.2010                                                           */
/*  Hardware:       dm_off_amp_ctrl v0.1                                                 */
/*  Controller:     MSP430f1611 => "Regelcontroller"                                     */
/*  Toolchain:      MSPGCC v.20060502, Eclipse ganymede-SR2, USBExpress v1.021           */
/*                                                                                       */
/*  Datei:          isr_comp_a.c                                                         */
/*  Beschreibung:   Interrupt Service Routine für Comparator A                           */
/*                  Beinhaltet die Steuerung der geschachtelten Zustandsautomaten        */
/*                  Namensgebungen basieren auf Diplomarbeit NJ                          */
/*                                                                                       */
/*  Funktion:       ok                                                                   */
/*                                                                                       */
/*****************************************************************************************/

#include "main.h"
#include "global.h"


/*************************************************************/
/*         Interrupt Service Routine Comparator A            */
/*************************************************************/

interrupt ( COMPARATORA_VECTOR ) isr_comp_a ( void )
{
    switch (sensor_state) {

/********************** Systemstart ************************/
        case Initial_0Hz: {
            switch (state_periode) {

/******************** Periodenanfang **********************/
                case Search_S1:
                                            // Timer ist aus || Entprellzeit erreicht?
                        if((TAR == 0x0000) || (TAR > MIN_TIME_S1)) {
                            TACTL |= TACLR;                 // Timer zurücksetzen
                            TACTL |= (ID_3 | MC_2);         // Timer start cont. mode
                            DAC12_0DAT = OFFSET_START - LEVEL_SX;   // Level S2
                            CACTL1 |= CAIES;                // IRQ fallende Flanke
                            state_periode = Search_S2;
                        }
                        break;

/******************** Periodemitte ***********************/
                case Search_S2:
                        if(TAR > MIN_TIME_S2) {             // Entprellzeit erreicht
                            DAC12_0DAT = OFFSET_START + LEVEL_SX;   // Level S1
                            CACTL1 &= ~CAIES;               // IRQ steigende Flanke
                            state_periode = Search_S1;
                            sensor_state  = Initial_1Hz;
                            //sensor_state = Active;
                        }
                        break;
                default:
                        state_periode = Search_S1;
            }
            break;
        }

/********************** Regelung beim PowerOn ************************/
        case Initial_1Hz: {
            switch (state_periode) {

                case Search_S1:      //_ Anfang einer Periode zum Messen des Duty __
                        if(TAR > MIN_TIME_S1) {             // Entprellzeit erreicht?
                            TACTL |= TACLR;                 // Timer zurücksetzen
                            TACTL |= (ID_3 | MC_2);         // Timer start cont. mode
                            DAC12_0DAT = OFFSET_START - LEVEL_SX;   // Level S2
                            CACTL1 |= CAIES;                // IRQ fallende Flanke
                            #ifdef __LOG_TIME
                                g_log.new_time = TRUE;      // für log_time()
                            #endif
                            state_periode = Search_S2;
                        }
                        break;

                case Search_S2:      //_ Mitte einer Periode zum Messen des Duty ___
                        if(TAR > MIN_TIME_S2) {             // Entprellzeit erreicht
                            g_time.tp = TAR;                // Zählerstand sichern
                            DAC12_0DAT = OFFSET_START + LEVEL_SX;   // Level S1
                            CACTL1 &= ~CAIES;               // IRQ steigende Flanke
                            state_periode = Search_S1_Calc;
                            //state_periode = Search_S2;
                            //sensor_state  = Active;       // next Sensor State
                        }
```

```
                        break;

/******************** Periodenende => Duty-Cycle berechnen **********************/
                case Search_S1_Calc:

                    if(TAR > MIN_TIME_S1) {                 // Entprellzeit erreicht?
                            g_time.tt = TAR;                    // Zählerstand sichern
                            TACTL |= TACLR;                     // Timer zurücksetzen
                            TACTL |= (ID_3 | MC_2);             // Timer start cont. mode
                            DAC12_0DAT = OFFSET_START - LEVEL_SX;   // Level S2
                            CACTL1 |= CAIES;                    // IRQ fallende Flanke

                                                        // tp/T * 128 = duty-cycle in %
                            g_sig.duty = ((uint32_t)g_time.tp << 7)  / g_time.tt;

                            #ifdef __OFFSET_CAL
                                calc_offset();
                                DAC12_1DAT = g_sig.offset;  // DAC12_1 neuer Wert
                            #endif

                            g_sig.compensations++;

                            if(g_sig.compensations == COMPENSATIONS) {
                                g_sig.compensations = 0;
                                state_periode = Search_S2_Check_Gain;
                                ADC12CTL0 |= ENC;           /* enable ADC */
                            }
                            else {  // eine Periode überspringen für neue Kompensation
                                state_periode = Search_S2_Pre;
                            }
                        }
                        break;
                case Search_S2_Pre:  //_ Mitte einer Periode, vor Search_S1

                        if(TAR > MIN_TIME_S2) {                 // Entprellzeit erreicht
                            g_time.tp = TAR;
                            DAC12_0DAT = OFFSET_START + LEVEL_SX;   // Level S1
                            CACTL1 &= ~CAIES;                   // IRQ steigende Flanke
                            state_periode = Search_S1;
                        }
                        break;

                case Search_S2_Check_Gain:
                    if(TAR > MIN_TIME_S2) {                 // Entprellzeit erreicht

                        g_time.tp = TAR;                    // Zählerstand Timer A sichern
                        DAC12_0DAT = OFFSET_START + LEVEL_SX;   // Level S1
                        CACTL1 &= ~CAIES;                   // IRQ steigende Flanke
                        state_periode = Search_S1_Check_Gain;
                        //sensor_state  = Active;       // next Sensor State

                    }
                    break;

                case Search_S1_Check_Gain:
                    if(TAR > MIN_TIME_S1) {                 // Entprellzeit erreicht?

                        ADC12CTL0 &= ~ENC;          //disable ADC

                        TACTL |= TACLR;                     // Timer zurücksetzen
                        TACTL |= (ID_3 | MC_2);             // Timer start cont. mode
                        DAC12_0DAT = OFFSET_START - LEVEL_SX;   // Level S2
                        CACTL1 |= CAIES;                    // IRQ fallende Flanke

                        check_gain(); //Verstärkung überprüfen und Folgezustand festlegen
                    }
                    break;
            }
            break;
        }

/********************** Normaler Betrieb "ACTIVE" ************************/
        case Active:

            switch (state_periode) {

/******************** Periodenanfang **********************/
                case Search_S1:
                    g_time.tt = TAR;                // Zählerstand sichern
                    TACTL |= TACLR;                 // Timer A zurücksetzen
                    TACTL |= (ID_3 | MC_2);         // Timer A Start Continuis - Mode
                    DAC12_0DAT = OFFSET_START - LEVEL_SX;   // Level S2
                    CACTL1 |= CAIES;                // IRQ Comp. bei fallender Flanke
                    state_periode = Search_S2;
                    ADC12CTL0 |= ENC;           /* enable ADC */
                    break;

/******************** Periodenmitte **********************/
                case Search_S2:
                    g_time.tp = TAR;                    // Zählerstand Timer A sichern
```

```
                    DAC12_0DAT = OFFSET_START + LEVEL_SX;   // Level S1
                    CACTL1 &= ~CAIES;                   // IRQ Comp. bei steigender Flanke

                    #ifdef __ACTIVE_COMPENSATION
                        state_periode = Search_S1_Calc;
                    #else
                        state_periode = Search_S1;
                    #endif

                    break;


/******************** Periodenende => Duty-Cycle berechnen **********************/
                case Search_S1_Calc:

                    if(TAR > MIN_TIME_S1) {                 // Entprellzeit erreicht?
                        g_time.tt = TAR;                    // Zählerstand sichern
                        TACTL |= TACLR;                     // Timer zurücksetzen
                        TACTL |= (ID_3 | MC_2);             // Timer start cont. mode
                        DAC12_0DAT = OFFSET_START - LEVEL_SX;   // Level S2
                        CACTL1 |= CAIES;                    // IRQ fallende Flanke

                        // aktuellen Duty-Cycle ausrechnen und aufsummieren
                    #ifdef __MEAN_DUTY_DEPEND_ON_GAIN
                        if(g_sig.duty_count < (1<<(g_sig.gain+2))) {
                    #else
                        if(g_sig.duty_count < (1<<PERIODES_FOR_MEAN_DUTY)) {
                    #endif
                            g_sig.duty = ((uint32_t)g_time.tp << 7)  / g_time.tt; //Duty rechnen
                            g_sig.duty_sum += g_sig.duty;
                            g_sig.duty_count++;
                            state_periode = Search_S2_Pre;
                        }

                        // Bildung eines Mittelwertes für den Duty-Cycle über x Perioden
                        else{
                            //Division durch Anzahl der aufsummierten Duty-Cycles
                        #ifdef __MEAN_DUTY_DEPEND_ON_GAIN
                            g_sig.duty = g_sig.duty_sum >> (g_sig.gain+2);
                        #else
                            g_sig.duty = g_sig.duty_sum >> PERIODES_FOR_MEAN_DUTY;
                        #endif
                            g_sig.duty_count = 0;
                            g_sig.duty_sum = 0;

                                //Signal ist offsetfrei
                            if(g_sig.duty > 64-DUTY_TOLERANCE && g_sig.duty < 64+DUTY_TOLERANCE) {
                                state_periode = Search_S2_Check_Gain;
                                ADC12CTL0 |= ENC;           /* enable ADC */
                            }

                            else {
                                set_compensation_state(TRUE); //Regelung aktiv

                                #ifdef __OFFSET_CAL
                                    calc_offset();
                                    DAC12_1DAT = g_sig.offset;  // DAC12_1 neuer Wert
                                #endif

                                set_compensation_state(FALSE); //Regelung abgeschlossen
                                state_periode = Search_S2_Pre;
                            }
                        }
                    }

                    break;

/******************** Periodenmitte **********************/
                case Search_S2_Pre:

                    if(TAR > MIN_TIME_S2) {                 // Entprellzeit erreicht
                        g_time.tp = TAR;
                        DAC12_0DAT = OFFSET_START + LEVEL_SX;   // Level S1
                        CACTL1 &= ~CAIES;                   // IRQ steigende Flanke
                        state_periode = Search_S1;
                    }
                    break;


                case Search_S2_Check_Gain:
                    if(TAR > MIN_TIME_S2) {                 // Entprellzeit erreicht

                        g_time.tp = TAR;                    // Zählerstand Timer A sichern
                        DAC12_0DAT = OFFSET_START + LEVEL_SX;   // Level S1
                        CACTL1 &= ~CAIES;                   // IRQ steigende Flanke
                        state_periode = Search_S1_Check_Gain;
                    }
                    break;
```

```
                case Search_S1_Check_Gain:
                    if(TAR > MIN_TIME_S1) {                 // Entprellzeit erreicht?

                        ADC12CTL0 &= ~ENC;          //disable ADC

                        TACTL |= TACLR;                     // Timer zurücksetzen
                        TACTL |= (ID_3 | MC_2);             // Timer start cont. mode
                        DAC12_0DAT = OFFSET_START - LEVEL_SX;   // Level S2
                        CACTL1 |= CAIES;                    // IRQ fallende Flanke

                        check_gain(); //Verstärkung überprüfen und Folgezustand festlegen
                    }
                    break;


            default:
                state_periode = Search_S1;
                sensor_state  = Initial_0Hz;
                break;
        }

   } // Ende switch(sensor_state)

   CACTL1 &= ~CAIFG;   // IRQ Flag löscht automatisch, aber da bei niedrigen Frequenzen
                       // der Comparator prellt muss das widerholt gelöscht werden
}
```

```
/*****************************************************************************************/
/*                                                                                       */
/*  Bachelorthesis Martin Stahl                                                          */
/*                                                                                       */
/*  Titel:          Controllersystem zur Verstärkungsregelung und Offsetkompensation     */
/*  Autor:          Martin Stahl                                                         */
/*  Datum:          05.01.2010                                                           */
/*  Hardware:       dm_off_amp_ctrl v0.1                                                 */
/*  Controller:     MSP430f1611 => "Regelcontroller"                                     */
/*  Toolchain:      MSPGCC v.20060502, Eclipse ganymede-SR2, USBExpress v1.021           */
/*                                                                                       */
/*  Datei:          isr_timer_a1.c                                                       */
/*  Beschreibung:   Interrupt Service Routine für Timer A                                */
/*                  Zur Stillstandserkennung                                             */
/*                  Basiert auf Diplomarbeit NJ                                          */
/*                                                                                       */
/*  Funktion:       ok                                                                   */
/*                                                                                       */
/*****************************************************************************************/

#include "main.h"
#include "global.h"


/*************************************************************/
/*         Interrupt Service Routine Timer A                 */
/*************************************************************/

interrupt ( TIMERA1_VECTOR ) isr_timer_a1 ( void )
{
        // Routine wird aufgerufen, wenn Überlauf des Zeitgeber Timer_A auftritt.
        // Idealerweise, wenn die Zahnfrequnez unter 1Hz fällt. Real aufgrund der Länge
        // des Zeitgebers, sowie dessen Taktes bereits nach 838ms (1,2Hz).
        // Somit weicht dieser Punkt von der eigentlichen Spezifikation ab.

    if((TACTL & TAIFG) == TAIFG) {          // Timer Overflow
        TACTL |= TACLR;                     // Timer zurücksetzen
        TACTL |= (ID_3 | MC_2);             // Timer start cont. mode

                                        // Wenn Sensor vom Zustand Aktiv in den
                                        // Zustand Init0_Hz zurückfallen soll
        #ifdef __INIT_FALL_BACK
            cleanup();
            DAC12_0DAT = OFFSET_START + LEVEL_SX;   // Schwellwert Comp. Level S1
            DAC12_1DAT = OFFSET_START;              // Anfangswert Offsetcompensation
            CACTL1    &= ~CAIES;                    // IRQ Comparator steigende Flanke

            g_sig.compensations = 0x00;             // Anzahl der Offsetcomp = 0
            g_sig.offset        = OFFSET_START;     // Anfangswert Offsetcompensation

            state_periode = Search_S1;              // Automaten zurücksetzen
            sensor_state  = Initial_0Hz;


        #else                           // Wenn Sensor nicht zurückfallen soll,
                                        // z.B. bei Messreihenaufnahme
            if(sensor_state == Active) {    // nur State_Calc zurücksetzen wenn
                                            // im Zustand Active
                state_calc = S1_Prepare_Measure_Periode;
            }
            else {                          // nur zurückfallen, wenn nicht im
                                            // Zustand Active
                DAC12_0DAT = OFFSET_START + LEVEL_SX;   // Schwellwert Comp. Level S1
                DAC12_1DAT = OFFSET_START;              // Anfangswert Offsetcompensation
                CACTL1    &= ~CAIES;                    // IRQ Comparator steigende Flanke

                g_sig.compensations = 0x00;             // Anzahl der Offsetcomp = 0
                g_sig.offset        = OFFSET_START;     // Anfangswert Offsetcompensation

                state_periode = Search_S1;              // Automaten zurücksetzen
                sensor_state  = Initial_0Hz;
            }
        #endif

        g_time.frequenz           = 0x0000; // Frequenzanzeige = 0

        TACTL &= ~TAIFG;                    // Flag loeschen
    }
}
```

```
/*****************************************************************************************/
/*                                                                                       */
/*  Bachelorthesis Martin Stahl                                                          */
/*                                                                                       */
/*  Titel:          Controllersystem zur Verstärkungsregelung und Offsetkompensation     */
/*  Autor:          Martin Stahl                                                         */
/*  Datum:          17.12.2009                                                           */
/*  Hardware:       dm_off_amp_ctrl v0.1                                                 */
/*  Controller:     MSP430f1611 => "Regelcontroller"                                     */
/*  Toolchain:      MSPGCC v.20060502, Eclipse ganymede-SR2, USBExpress v1.021           */
/*                                                                                       */
/*  Datei:          send.c                                                               */
/*  Beschreibung:   Funktionen zur Datenausgabe auf der seriellen Schnittstelle          */
/*                  und zur Kommunikation mit dem Signalcontroller                       */
/*                                                                                       */
/*  Funktion:       ok                                                                   */
/*                                                                                       */
/*****************************************************************************************/

#include "main.h"
#include "global.h"

/*************************************************************/
/*            Zur Kommunikation mit Signalcontroller         */
/*************************************************************/
void set_compensation_state(uint8_t state) {

    //Regelung aktiv => U_Diff not valid
    if(state) {
        P3OUT |= COMM_5;
        PORT_LED |= PIN3;
        PORT_LED &= ~(LED2 | LED3);
        PORT_LED |= LED1; //rote LED ein

    }

    //Sensordiagnose freigeben
    else {
        P3OUT &= ~COMM_5;
        PORT_LED &= ~PIN3;
        PORT_LED &= ~(LED1 | LED3);
        PORT_LED |= LED2; //grüne LED ein
    }
}


/*************************************************************/
/*              Ausgabe auf serieller Schnittstelle          */
/*************************************************************/
uint8_t send_usart(uint8_t value) {

        // Ausgabe einer 8-Bit Variablen mit dem USART1.

    if((U1TCTL & TXEPT) == TXEPT) {     // wenn Buffer & Register leer --> Daten senden
        ME2  &= ~UTXE1;                 // USART1 TX disable
        U1TXBUF = value;                // Daten in Transmitter Buffer (max 8Bit)
        ME2  |= UTXE1;                  // USART1 TX enable
        return TRUE;
    }
    else
        return FALSE;
}
```

```
/*****************************************************************************************/
/*                                                                                       */
/*  Bachelorthesis Martin Stahl                                                          */
/*                                                                                       */
/*  Titel:          Controllersystem zur Verstärkungsregelung und Offsetkompensation     */
/*  Autor:          Martin Stahl                                                         */
/*  Datum:          16.12.2009                                                           */
/*  Hardware:       dm_off_amp_ctrl v0.1                                                 */
/*  Controller:     MSP430f1611 => "Regelcontroller"                                     */
/*  Toolchain:      MSPGCC v.20060502, Eclipse ganymede-SR2, USBExpress v1.021           */
/*                                                                                       */
/*  Datei:          set_7seg.c                                                           */
/*  Beschreibung:   Ausgabe von Daten auf dem Display-Board                              */
/*                  Basiert auf Diplomarbeit NJ => angepasst an Regelplatine             */
/*                                                                                       */
/*  Funktion:       ok                                                                   */
/*                                                                                       */
/*****************************************************************************************/

#include "main.h"
#include "global.h"

/*************************************************************/
/*              Ausgabe auf 7-Segment Anzeigen               */
/*************************************************************/

void set_7seg(uint16_t frequenz, uint8_t thd, uint8_t status)
{
    uint8_t data[3];

    PORT_7SEG_LE |= 0xFF;       // Latches 0...7 inaktiv
    PORT_7SEG_DB &= ~0x0F;      // Databits clear

/******************** Ausgabe Frequenz ***********************/
    if(frequenz > 9999) frequenz = 9999;

    // 4.Stelle Frequenz
    data[2] = frequenz / 1000;
    if(data[2] == 0x00) data[2] = 0x0F;         // führende 0 --> diese Stelle aus
    PORT_7SEG_DB |= (PINS_7SEG_DB & data[2]);   // Daten zuweisen (7Seg_DB0...3)
    PORT_7SEG_LE &= ~PIN0;                      // Latch 0 aktiv (7Seg_Le0)
    PORT_7SEG_LE |= PIN0;                       // Latch 0 inaktiv
    PORT_7SEG_DB &= ~0x0F;                      // Databits clear

    // 3.Stelle Frequenz
    data[1] = (frequenz % 1000) / 100;
    if((data[1] == 0) && (data[2] == 0x0F)) data[1] = 0x0F;
    PORT_7SEG_DB |= (PINS_7SEG_DB & data[1]);
    PORT_7SEG_LE &= ~PIN1;                      // Latch 1 aktiv
    PORT_7SEG_LE |= PIN1;
    PORT_7SEG_DB &= ~0x0F;

    // 2.Stelle Frequenz
    data[0] = (frequenz % 100) / 10;
    if((data[0] == 0) && (data[1] == 0x0F) && (data[2] == 0x0F)) data[0] = 0x0F;
    PORT_7SEG_DB |= (PINS_7SEG_DB & data[0]);
    PORT_7SEG_LE &= ~PIN2;                      // Latch 2 aktiv
    PORT_7SEG_LE |= PIN2;
    PORT_7SEG_DB &= ~0x0F;

    // 1.Stelle Frequenz:
    data[0] = frequenz % 10;
    PORT_7SEG_DB |= (PINS_7SEG_DB & data[0]);
    PORT_7SEG_LE &= ~PIN3;                      // Latch 3 aktiv
    PORT_7SEG_LE |= PIN3;
    PORT_7SEG_DB &= ~0x0F;


/******************** Ausgabe Zustand ************************/

    // 2.Stelle Zustand
    data[0] = (status & 0xF0) >> 4;
    PORT_7SEG_DB |= (PINS_7SEG_DB & data[0]);
    PORT_7SEG_LE &= ~PIN4;                      // Latch 4 aktiv
    PORT_7SEG_LE |= PIN4;
    PORT_7SEG_DB &= ~0x0F;

    // 1.Stelle Status
    data[0] = status & 0x0F;
    PORT_7SEG_DB |= (PINS_7SEG_DB & data[0]);
    PORT_7SEG_LE &= ~PIN5;                      // Latch 5 aktiv
    PORT_7SEG_LE |= PIN5;
    PORT_7SEG_DB &= ~0x0F;


/********************** Ausgabe THD **************************/

    // 2.Stelle THD
    data[0] = thd / 10;
    if(data[0] == 0) data[0] = 0x0F;
```

```
    PORT_7SEG_DB |= (PINS_7SEG_DB & data[0]);
    PORT_7SEG_LE &= ~PIN6;                      // Latch 6 aktiv
    PORT_7SEG_LE |= PIN6;
    PORT_7SEG_DB &= ~0x0F;

    // 1.Stelle THD
    data[0] = thd % 10;
    PORT_7SEG_DB |= (PINS_7SEG_DB & data[0]);
    PORT_7SEG_LE &= ~PIN7;                      // Latch 7 aktiv
    PORT_7SEG_LE |= PIN7;
    PORT_7SEG_DB &= ~0x0F;

}
/*****************************************************************************************/
```

```
/*****************************************************************************************/
/*                                                                                       */
/*  Bachelorthesis Martin Stahl                                                          */
/*                                                                                       */
/*  Titel:          Controllersystem zur Verstärkungsregelung und Offsetkompensation     */
/*  Autor:          Martin Stahl                                                         */
/*  Datum:          22.12.2009                                                           */
/*  Hardware:       dm_off_amp_ctrl v0.1                                                 */
/*  Controller:     MSP430f1611 => "Regelcontroller"                                     */
/*  Toolchain:      MSPGCC v.20060502, Eclipse ganymede-SR2, USBExpress v1.021           */
/*                                                                                       */
/*  Datei:          set_gain.c                                                           */
/*  Beschreibung:   Stellt Verstaerkung des AD8231 auf DM Amp v0.1 Hardware ein          */
/*                  Gibt Verstaerkung zusaetzlich auf COMM_6...8 aus                     */
/*                  (Kommunikationsleitungen zum Signalcontroller)                       */
/*                  Einzustellende Verstärkung ist in g_sig.gain gespeichert             */
/*                                                                                       */
/*  Funktion:       ok                                                                   */
/*                                                                                       */
/*****************************************************************************************/

#include "main.h"
#include "global.h"

/*************************************************************/
/*                  Einstellung der Verstärkung              */
/*************************************************************/
void set_gain() {


    if(g_sig.gain == 0) {                           // Gain = 1
        PORT_AMP &= ~0x07;                          // Datenbits zurücksetzen
        PORT_AMP &= ~AMP_CS;                        // CS an AD8231
        P2OUT &= ~(COMM_6 | COMM_7 | COMM_8);       // Kommunikationsleitungen zurücksetzten
    }
    else if(g_sig.gain == 1) {                      // Gain = 2
        PORT_AMP &= ~0x07;                          // Datenbits zurücksetzen
        PORT_AMP |= AMP_A0;                         // Datenbits setzen
        PORT_AMP &= ~AMP_CS;                        // CS an AD8231
        P2OUT &= ~(COMM_6 | COMM_7 | COMM_8);
        P2OUT |= COMM_6;
    }
    else if(g_sig.gain == 2) {                      // Gain = 4
        PORT_AMP &= ~0x07;
        PORT_AMP |= AMP_A1;
        PORT_AMP &= ~AMP_CS;
        P2OUT &= ~(COMM_6 | COMM_7 | COMM_8);
        P2OUT |= COMM_7;
    }
    else if(g_sig.gain == 3) {                      // Gain = 8
        PORT_AMP &= ~AMP_CS;
        PORT_AMP |= AMP_A0 | AMP_A1;
        PORT_AMP &= ~PIN3;
        P2OUT &= ~(COMM_6 | COMM_7 | COMM_8);
        P2OUT |= COMM_6 | COMM_7;
    }
    else if(g_sig.gain == 4) {                      // Gain = 16
        PORT_AMP &= ~0x07;
        PORT_AMP |= AMP_A2;
        PORT_AMP &= ~AMP_CS;
        P2OUT &= ~(COMM_6 | COMM_7 | COMM_8);
        P2OUT |= COMM_8;
    }
    else if(g_sig.gain == 5) {                      // Gain = 32
        PORT_AMP &= ~0x07;
        PORT_AMP |= AMP_A0 | AMP_A2;
        PORT_AMP &= ~AMP_CS;
        P2OUT &= ~(COMM_6 | COMM_7 | COMM_8);
        P2OUT |= COMM_6 | COMM_8;
    }
    else if(g_sig.gain == 6) {                      // Gain = 64
        PORT_AMP &= ~0x07;
        PORT_AMP |= AMP_A1 | AMP_A2;
        PORT_AMP &= ~AMP_CS;
        P2OUT &= ~(COMM_6 | COMM_7 | COMM_8);
        P2OUT |= COMM_7 | COMM_8;
    }
    else if(g_sig.gain == 7) {                      // Gain = 128
        PORT_AMP &= ~0x07;
        PORT_AMP |= AMP_A0 | AMP_A1 | AMP_A2;
        PORT_AMP &= ~AMP_CS;
        P2OUT &= ~(COMM_6 | COMM_7 | COMM_8);
        P2OUT |= COMM_6 | COMM_7 | COMM_8;
    }
    PORT_AMP |= AMP_CS;                             // not CS an AD8231
}
```

B.2 Matlab

B.2.1 Modellierung des Regelverfahrens

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%+
%+  Simulation des Regelverfahrens zur Ablauf-Veranschaulichung
%+  Funktionsweise angelehnt an Diplomarbeit NJ => Automatengesteuert
%+
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%+
%+ Autor:   Martin Stahl
%+ Datum:   09.11.2009
%+ Name:    simulation_gain_control_real.m
%+
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

clear all
close all

%% initialisations

GAIN_START      = 7;    % Gain als Exponent von 2
COMPENSATIONS   = 6;    % Anzahl an Offset-Kompenationen

% DAC_12
OFFSET_START   = 1.25; % Nulllinie bei 1,25V (Uref/2)
LEVEL_HYST      = 0.1; % Differenz der Hystereschwellen zum Offset (0,1V)
LEVEL_GAIN_MAX  = 1.0; % Maximal Verstärkung (10*LEVEL_HYST)
LEVEL_GAIN_MIN  = 0.3; % Minimale Verstärkung

% Messwerte einlesen
load 2009_09_30_rmp_01_part03.rmp.mat %107Hz Zahnfrequenz

u_diff  = measure_scope(6).u_diff_y;% gemessenes Bruecken-Differenzsignal
gain_demo = measure_demo(6).gain;
delta_t = 1/parameters.samplerate;   % Abtastintervall Oszilloskop
time    = 0 : delta_t : 4-delta_t;   % Zeit t bis 4sec
k       = 1; % Laufindex Differenzsignal
timer   = 1; % Timer fuer Duty-Cycle und Stillstandserkennung
noise   = rand(1,length(time))*2.5e-4; % Rauschen

% Globale Variablen
g_sig  = struct( 'duty',           0, ...
                 'offset',         0, ...
                 'offset_dac',     2048, ...
                 'offset_faktor',  0, ...
                 'compensations',  0, ...
                 'gain',           2^GAIN_START,  ...
                 'g1_passed',      0, ...
                 'g2_passed',      0, ...
                 'g3_passed',      0, ...
                 'g4_passed',      0, ...
                 'gain_ok',        0  ...
               );

g_time = struct( 'tp',             0, ...
                 'tt',             0, ...
                 'tt_estimated',   0, ...
                 'tt_at_sample',   0, ...
                 'tt_avg',         0, ...
                 'id_tb',          0, ...
                 'frequenz',       0, ...
```

```
                'sample_inst',     0, ...
                'tt_at_sample_valid', 0 ...
              );


sensor_state    = 'Initial_0Hz';
state_periode   = 'Search_S1';
state_calc      = 'S1_Prepare_Measure_Periode';

Signal_Level_Temp = 0;

% Rechtecksignal der Nulldurchgangserkennung
Signal_Level    = zeros(1,length(u_diff));
% Nulldurchgangspuls
Signal_Inc      = zeros(1,length(u_diff));


% Signaloffset addieren, da Messdaten offsetfrei sind
u_diff = u_diff + 0.05; %+ noise;

disp('======================== START ================================');

%% statemachines

while(k<length(u_diff))
    % Differenzsignal
    u_diff(k) = (u_diff(k) - OFFSET_START + g_sig.offset) / ...
        gain_demo*g_sig.gain + OFFSET_START;

    switch sensor_state

        % ================ POWER ON => auf Bewegung warten ============
        % ================ Initial_0Hz ================================
        case 'Initial_0Hz'

            switch(state_periode)
                case 'Search_S1'
                        % HYST+ ueberschritten
                    if( u_diff(k) > (OFFSET_START + LEVEL_HYST))
                        timer = 0; % Timer neustarten
                        Signal_Level_Temp  = 1;
                        Signal_Inc(k)      = 1;
                        state_periode = 'Search_S2';
                    end

                case 'Search_S2'
                        % HYST- unterschritten
                    if( u_diff(k) < (OFFSET_START - LEVEL_HYST))
                        timer = 0; % Timer neustarten
                        Signal_Level_Temp = 0;
                        Signal_Inc(k)     = 1;
                        state_periode = 'Search_S1';
                        sensor_state = 'Initial_1Hz';
                    end
                otherwise
                    state_periode = 'Search_S1';
            end % Ende switch(state_periode)
```

```
% =============== OFFSET KOMPENSATION =======================
% =============== Initial_1Hz ================================
case 'Initial_1Hz'
    switch(state_periode)
        case 'Search_S1'
                 % HYST+ ueberschritten
            if( u_diff(k) > (OFFSET_START + LEVEL_HYST))
                timer = 0; % Timer neustarten
                Signal_Level_Temp  = 1;
                Signal_Inc(k)      = 1;
                state_periode = 'Search_S2';
            end
        case 'Search_S2'
                % HYST- unterschritten
            if( u_diff(k) < (OFFSET_START - LEVEL_HYST))
                g_time.tp = timer; % Zaehlerstand sichern
                Signal_Level_Temp = 0;
                Signal_Inc(k)     = 1;
                state_periode = 'Search_S1_Calc';
            end
        case 'Search_S1_Calc'

            % HYST+ ueberschritten
        if( u_diff(k) > (OFFSET_START + LEVEL_HYST))
            g_time.tt = timer; % Zaehlerstand sichern
            timer = 0; % Timer neustarten
            Signal_Level_Temp  = 1;
            Signal_Inc(k)      = 1;

            % duty-cycle auf 0...128 skaliert
            g_sig.duty = g_time.tp*128/g_time.tt;

            % Offset Korrekturtabelle
            if(g_sig.duty >= 122);      g_sig.offset_faktor = 32;
            elseif(g_sig.duty >= 110);  g_sig.offset_faktor = 29;
            elseif(g_sig.duty >= 101);  g_sig.offset_faktor = 25;
            elseif(g_sig.duty >=  97);  g_sig.offset_faktor = 23;
            elseif(g_sig.duty >=  94);  g_sig.offset_faktor = 21;
            elseif(g_sig.duty >=  90);  g_sig.offset_faktor = 19;
            elseif(g_sig.duty >=  87);  g_sig.offset_faktor = 17;
            elseif(g_sig.duty >=  84);  g_sig.offset_faktor = 15;
            elseif(g_sig.duty >=  81);  g_sig.offset_faktor = 13;
            elseif(g_sig.duty >=  79);  g_sig.offset_faktor = 11;
            elseif(g_sig.duty >=  76);  g_sig.offset_faktor = 9;
            elseif(g_sig.duty >=  73);  g_sig.offset_faktor = 7;
            elseif(g_sig.duty >=  70);  g_sig.offset_faktor = 5;
            elseif(g_sig.duty >=  69);  g_sig.offset_faktor = 4;
            elseif(g_sig.duty >=  66);  g_sig.offset_faktor = 2;
            elseif(g_sig.duty >=  65);  g_sig.offset_faktor = 1;
            elseif(g_sig.duty >=  64);  g_sig.offset_faktor = 0;
            elseif(g_sig.duty >=  63);  g_sig.offset_faktor = -1;
            elseif(g_sig.duty >=  60);  g_sig.offset_faktor = -3;
            elseif(g_sig.duty >=  57);  g_sig.offset_faktor = -5;
            elseif(g_sig.duty >=  55);  g_sig.offset_faktor = -7;
            elseif(g_sig.duty >=  52);  g_sig.offset_faktor = -9;
            elseif(g_sig.duty >=  49);  g_sig.offset_faktor = -11;
            elseif(g_sig.duty >=  46);  g_sig.offset_faktor = -13;
            elseif(g_sig.duty >=  43);  g_sig.offset_faktor = -15;
            elseif(g_sig.duty >=  40);  g_sig.offset_faktor = -17;
```

```
    elseif(g_sig.duty >=  37);  g_sig.offset_faktor = -19;
    elseif(g_sig.duty >=  34);  g_sig.offset_faktor = -21;
    elseif(g_sig.duty >=  30);  g_sig.offset_faktor = -23;
    elseif(g_sig.duty >=  26);  g_sig.offset_faktor = -25;
    elseif(g_sig.duty >=  22);  g_sig.offset_faktor = -27;
    elseif(g_sig.duty >=  16);  g_sig.offset_faktor = -29;
    elseif(g_sig.duty >=   7);  g_sig.offset_faktor = -31;
    elseif(g_sig.duty <    7);  g_sig.offset_faktor = -32;
    end

    % Skalierung und Berechnung der Offset-Korrektur
    g_sig.offset_faktor = g_sig.offset_faktor*128;
    g_sig.offset_faktor = 8192 + (g_sig.offset_faktor);
    g_sig.offset_dac = g_sig.offset_dac * ...
                       g_sig.offset_faktor;
    g_sig.offset_dac = g_sig.offset_dac / 8192;
    g_sig.offset = 1.25 - g_sig.offset_dac * 2.5/4096;

    g_sig.compensations = g_sig.compensations+1;

    if(g_sig.compensations == COMPENSATIONS)
         g_sig.compensations = 0;
         state_periode = 'Search_S2_Check_Gain';
       %  sensor_state = 'Active';
    else
            state_periode = 'Search_S2_Pre';
    end

end % Ende if HYST+ ueberschritten

case 'Search_S2_Pre'
        % HYST- unterschritten
    if( u_diff(k) < (OFFSET_START - LEVEL_HYST))
        Signal_Level_Temp = 0;
        Signal_Inc(k)     = 1;
        state_periode = 'Search_S1';
    end
case 'Search_S2_Check_Gain'
    % HYST- unterschritten
    if( u_diff(k) < (OFFSET_START - LEVEL_HYST))
        timer = 0; % Timer neustarten
        Signal_Level_Temp = 0;
        Signal_Inc(k)     = 1;
        state_periode = 'Search_S1_Check_Gain';
    else
                %G2 ueberschritten
        if( u_diff(k) > OFFSET_START+LEVEL_GAIN_MIN )
            g_sig.g2_passed = 1;
        end
                %G1 ueberschritten
        if( u_diff(k) > OFFSET_START+LEVEL_GAIN_MAX )
            g_sig.g1_passed = 1;
        end
    end
case 'Search_S1_Check_Gain'

        % HYST- ueberschritten
     if( u_diff(k) > (OFFSET_START + LEVEL_HYST))
        timer = 0; % Timer neustarten
```

```
                        Signal_Level_Temp = 0;
                        Signal_Inc(k)     = 1;

                        if( g_sig.g2_passed && g_sig.g3_passed )
                            if( ~(g_sig.g1_passed || g_sig.g4_passed) )
                                g_sig.gain_ok = 1;
                                sensor_state = 'Active';
                            else    %Verstaerkung halbieren
                                g_sig.gain = g_sig.gain/2;
                                g_sig.g1_passed = 0;
                                g_sig.g2_passed = 0;
                                g_sig.g3_passed = 0;
                                g_sig.g4_passed = 0;
                                g_sig.gain_ok   = 0;
                                state_periode = 'Search_S2_Pre';
                            end
                        else    %Verstaerkung halbieren
                             g_sig.gain = g_sig.gain/2;
                                g_sig.g1_passed = 0;
                                g_sig.g2_passed = 0;
                                g_sig.g3_passed = 0;
                                g_sig.g4_passed = 0;
                                g_sig.gain_ok   = 0;
                                state_periode = 'Search_S2_Pre';
                        end
                    else
                                %G3 ueberschritten
                         if( u_diff(k) < OFFSET_START-LEVEL_GAIN_MIN )
                             g_sig.g3_passed = 1;
                         end
                                %G4 ueberschritten
                         if( u_diff(k) < OFFSET_START-LEVEL_GAIN_MAX )
                             g_sig.g4_passed = 1;

                         end
                    end
            end % Ende switch(state_periode)

    % =============== NORMALER BETRIEB ==========================
    % =============== Active ====================================
        case 'Active'
          % disp('active...');
           %g_sig
          % break;

    end % Ende switch

    k = k+1;
    timer = timer+1;
    Signal_Level(k) = Signal_Level_Temp;
    if(k>220000)
        break;
    end

end % Ende while
disp('======================= ENDE ================================');

%% plots
```

```
% ========= SENSORSIGNAL ==============
subplot(3,1,1);
plot(time,u_diff);
title(['Verstärktes Brückendifferenzsignal U_{Diff}', ...
       'im zeitlichen Verlauf'],'FontSize',12);
xlabel('Zeit in sec','FontSize',10);
ylabel('U_{Diff} in V','FontSize',10);
xlim([0 0.8]);
%ylim([0 2.5]);
grid;

hold all

% Verstaerkungs- und Hystereseschwellen
line([time(1) time(end)],[OFFSET_START OFFSET_START], 'color', 'k' );
line([time(1) time(end)],[OFFSET_START+LEVEL_HYST OFFSET_START+ ...
                          LEVEL_HYST], 'color', 'b', 'linestyle', '--' );
line([time(1) time(end)],[OFFSET_START-LEVEL_HYST OFFSET_START- ...
                          LEVEL_HYST], 'color', 'b', 'linestyle', '--' );
line([time(1) time(end)],[OFFSET_START+LEVEL_GAIN_MAX OFFSET_START+ ...
                      LEVEL_GAIN_MAX], 'color', 'r', 'linestyle', '-' );
line([time(1) time(end)],[OFFSET_START+LEVEL_GAIN_MIN OFFSET_START+ ...
                      LEVEL_GAIN_MIN], 'color', 'g', 'linestyle', '-' );
line([time(1) time(end)],[OFFSET_START-LEVEL_GAIN_MIN OFFSET_START- ...
                      LEVEL_GAIN_MIN], 'color', 'g', 'linestyle', '-' );
line([time(1) time(end)],[OFFSET_START-LEVEL_GAIN_MAX OFFSET_START- ...
                      LEVEL_GAIN_MAX], 'color', 'r', 'linestyle', '-' );

% plot Eigenschaften

hold off

% =========== Signal im ADC-Aussteuerbereich =====================

subplot(3,1,2);
plot(time,u_diff);
title('U_{Diff} im ADC-Aussteuerbereich','FontSize',12);
xlabel('Zeit in sec','FontSize',10);
ylabel('U_{Diff} in V','FontSize',10);
xlim([0 0.8]);
ylim([0 2.5]);
grid;

hold all

% Verstaerkungs- und Hystereseschwellen
line([time(1) time(end)],[OFFSET_START OFFSET_START], 'color', 'k' );
line([time(1) time(end)],[OFFSET_START+LEVEL_HYST OFFSET_START+ ...
                          LEVEL_HYST], 'color', 'b', 'linestyle', '--' );
line([time(1) time(end)],[OFFSET_START-LEVEL_HYST OFFSET_START- ...
                          LEVEL_HYST], 'color', 'b', 'linestyle', '--' );
line([time(1) time(end)],[OFFSET_START+LEVEL_GAIN_MAX OFFSET_START+ ...
                      LEVEL_GAIN_MAX], 'color', 'r', 'linestyle', '-' );
line([time(1) time(end)],[OFFSET_START+LEVEL_GAIN_MIN OFFSET_START+ ...
                      LEVEL_GAIN_MIN], 'color', 'g', 'linestyle', '-' );
line([time(1) time(end)],[OFFSET_START-LEVEL_GAIN_MIN OFFSET_START- ...
                      LEVEL_GAIN_MIN], 'color', 'g', 'linestyle', '-' );
line([time(1) time(end)],[OFFSET_START-LEVEL_GAIN_MAX OFFSET_START- ...
                      LEVEL_GAIN_MAX], 'color', 'r', 'linestyle', '-' );
```

```
% plot Eigenschaften

hold off

% ============= SIGNALAUSSCHNITT ========================

subplot(3,1,3);
plot(time(100000:200000),u_diff(100000:200000));
title(['U_{Diff} im ADC-Aussteuerbereich - Darstellung eines', ...
       'zeitlichen Ausschnitt'],'FontSize',12);
xlabel('Zeit in sec','FontSize',10);
ylabel('U_{Diff} in V','FontSize',10);
xlim([0.4 0.65]);
ylim([0 2.5]);
grid;

hold all

% Verstaerkungs- und Hystereseschwellen
line([time(1) time(end)],[OFFSET_START OFFSET_START], 'color', 'k' );
line([time(1) time(end)],[OFFSET_START+LEVEL_HYST OFFSET_START+ ...
                          LEVEL_HYST], 'color', 'b', 'linestyle', '--' );
line([time(1) time(end)],[OFFSET_START-LEVEL_HYST OFFSET_START- ...
                          LEVEL_HYST], 'color', 'b', 'linestyle', '--' );
line([time(1) time(end)],[OFFSET_START+LEVEL_GAIN_MAX OFFSET_START+ ...
                      LEVEL_GAIN_MAX], 'color', 'r', 'linestyle', '-' );
line([time(1) time(end)],[OFFSET_START+LEVEL_GAIN_MIN OFFSET_START+ ...
                      LEVEL_GAIN_MIN], 'color', 'g', 'linestyle', '-' );
line([time(1) time(end)],[OFFSET_START-LEVEL_GAIN_MIN OFFSET_START- ...
                      LEVEL_GAIN_MIN], 'color', 'g', 'linestyle', '-' );
line([time(1) time(end)],[OFFSET_START-LEVEL_GAIN_MAX OFFSET_START- ...
                      LEVEL_GAIN_MAX], 'color', 'r', 'linestyle', '-' );

hold off
```

B.2.2 Berechnung der Abtastfrequenz

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%+
%+  Berechnung um nachzuweisen, dass eine Samplerate von 77kHz
%+  ausreichend für die Verstärkungsregelung ist.
%+
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%+
%+ Autor:   Martin Stahl
%+ Datum:   14.02.2010
%+ Name:    samplerate_for_gain_control.m
%+
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

clear all
close all

% Variablen
OFFSET_START    = 1.25; % Nulllinie bei 1,25V (Uref/2)
LEVEL_GAIN_MAX  = 0.95;  % Maximal Verstärkung (10*LEVEL_HYST)
LEVEL_GAIN_MIN  = 0.3;  % Minimale Verstärkung

f_sample = 77100;
f_signal = 2500;
u0       = 1;
t        = 0:0.01:2*pi;

u_diff  = OFFSET_START + u0*sin(t);
samples = zeros(1,ceil(f_sample/f_signal));

for i=1:length(samples)
    samples(i) = i*2*pi/(f_sample/f_signal);
end

plot(t,u_diff,'LineWidth',2);
hold all
stem(samples, OFFSET_START+u0*sin(samples),'--','color','b');
% Verstaerkungs- und Hystereseschwellen
line( [t(1) t(end)], [OFFSET_START OFFSET_START], 'color', 'k' );
line( [t(1) t(end)], [OFFSET_START+LEVEL_GAIN_MAX OFFSET_START+ ...
                      LEVEL_GAIN_MAX], 'color', 'r', 'linestyle', '-' );
line( [t(1) t(end)], [OFFSET_START+LEVEL_GAIN_MIN OFFSET_START+ ...
                      LEVEL_GAIN_MIN], 'color', 'g', 'linestyle', '-' );
line( [t(1) t(end)], [OFFSET_START-LEVEL_GAIN_MIN OFFSET_START- ...
                      LEVEL_GAIN_MIN], 'color', 'g', 'linestyle', '-' );
line( [t(1) t(end)], [OFFSET_START-LEVEL_GAIN_MAX OFFSET_START- ...
                      LEVEL_GAIN_MAX], 'color', 'r', 'linestyle', '-' );

hold off
grid on;
title(['Realisierung der Verstärkungsregelung' ...
       'mittels Abtastung'],'FontSize',14);
xlabel('normiert auf 2\pi','FontSize',12);
ylabel('verstärkte Brückendifferenzspannung U_{Diff} in V','FontSize',12);
```

C Automatisierung der Radmessplätze

C.1 Dateien zur Automatisierung

Zur vollautomatischen Aufzeichnung und Auswertung von Messreihen an den Radmessplätzen werden nachfolgend aufgeführte Dateien benötigt:

- `rmp_stepper_scope_record.m` - Automatische Aufzeichnung einer Messreihe am Messplatz mit passivem Encoderrad. Die gemessenen Daten von Oszilloskop und Demonstrator werden zusammen mit den Parameteren der Messreihe in Matlab-Strukturen gespeichert.
- `rmp_stepper_scope_record_analyse.m` - Auswertung einer zuvor aufgezeichneten Messreihe. Folgende Grafiken werden erstellt:
  - Klirrfaktor in Abhängigkeit des Luftspaltes
  - Beträge der Harmonischen in Abhängigkeit des Luftspaltes
  - Phasen der Harmonischen in Abhängigkeit des Luftspaltes
  - Amplitude des Differenzsignals in Abhängigkeit des Luftspaltes, mit und ohne Berücksichtigung des Verstärkungsfaktors
- `rmp_stepper_scope_record_analyze_tekdata.m` - Umfangreiche Auswertung einer aufgenommenen Messreihe. Anhand der vom Oszilloskop gesampelten Daten wird mit Matlab eine FFT gerechnet und mit dem berechneten Klirrfaktor des Demonstrators verglichen
- `rmp_active_enc_stepper_demo_scope_record.m` - Automatische Aufzeichnung einer Messreihe am Messplatz mit aktivem Encoderrad
- `rmp_active_enc_stepper_demo_scope_record_analyse.m` - Auswertung von zuvor gemessenen Daten
- `get_measurement.m` - Funktion zur Kommunikation mit dem Demonstrator
- `tmcl.mdd` - Treiberdatei zur Ansteuerung der Schrittmotoren

- `dpo4054.mdd` - Treiberdatei zur Ansteuerung eines Tektronix DPO4054 Oszilloskop, funktioniert aber auch für Oszilloskope des Herstellers Tektronix

Die Datein `rmp_stepper_scope_record.m` und `rmp_stepper_scope_record_analyse.m` sind exemplarisch in Anhang C.3 abgedruckt, alle weiteren Dateien befinden sich auf der beigefügten CD-ROM im Verzeichnis „automatisierung_radmessplaetze“.

C.2 Messungen

In den Abbildungen C.1 und C.2 wurde anstelle des Klirrfaktors die Spannung U_{pp} des Differenzsignals in die Farbe codiert. Die erste Messung wurde von $0mm$ aufsteigend, die zweite von $3mm$ absteigend gemessen. Es ergibt sich der charakterische Verlauf mit einer zunächst ansteigenden Amplitude von circa $0 \ldots 1mm$, einem Maximum und daraufhin wieder fallenden Amplitude (siehe auch Abbildung 2.2 auf Seite 17). Wie auch bei der Messung des Klirrfaktors auf Seite 72 entstehen Sprünge der Messwerte im Bereich um $0{,}5mm$, die ebenfalls auf mangelnde mechanische Präzision zurückzuführen sind.

Abbildungen C.3 und C.4 stellen den eingestellten Verstärkungsfaktor farbig dar. Es fällt auf, dass die Verdoppelung des Verstärkungsfaktors bei aufsteigend verfahrener x-Achse später erfolgt als bei absteigender Messung.

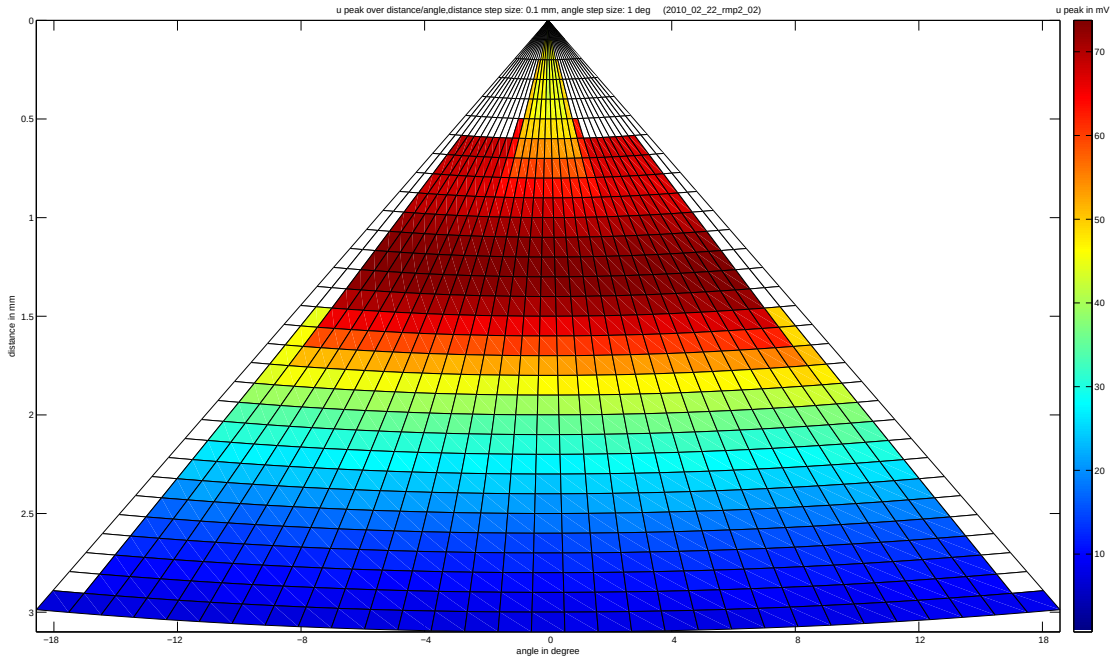


Abbildung C.1: *Aktiver Encoder in x-Achse verfahren sowie in φ-Achse verkippt. U_{pp} ist in die Farbe codiert, in x-Richtung wurde von 0mm aufsteigend gemessen*

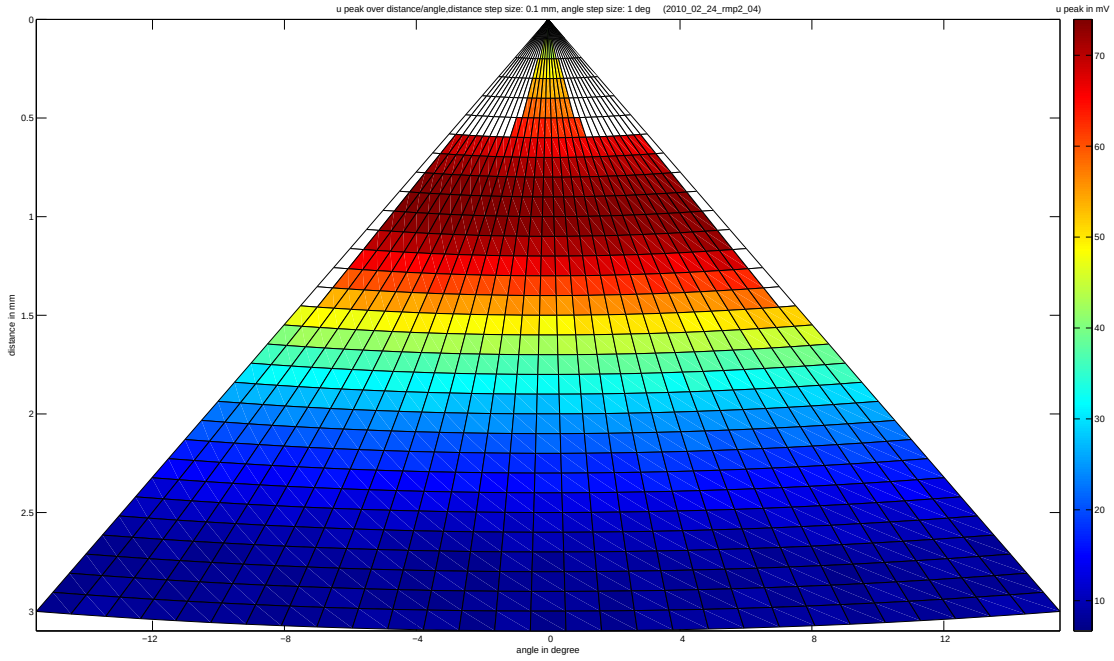


Abbildung C.2: *Aktiver Encoder in x-Achse verfahren sowie in φ-Achse verkippt. U_{pp} ist in die Farbe codiert, in x-Richtung wurde von 3mm absteigend gemessen*

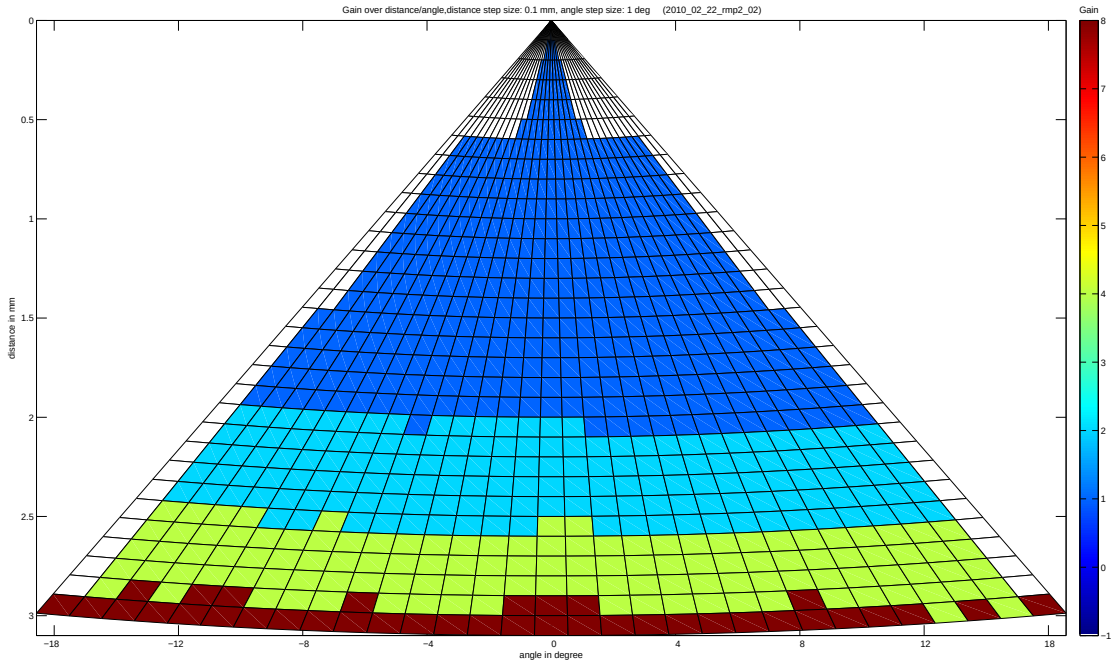


Abbildung C.3: *Aktiver Encoder in x-Achse verfahren sowie in φ-Achse verkippt. Der Verstärkungsfaktor ist in die Farbe codiert, in x-Richtung wurde von 0mm aufsteigend gemessen*

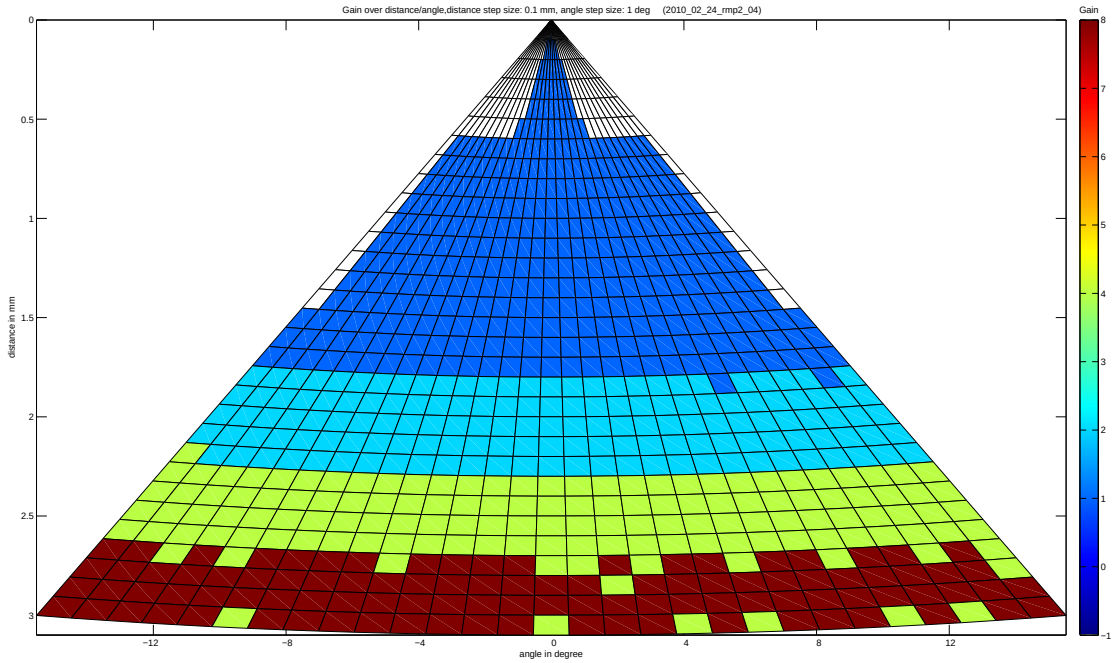


Abbildung C.4: *Aktiver Encoder in x-Achse verfahren sowie in φ-Achse verkippt. Der Verstärkungsfaktor ist in die Farbe codiert, in x-Richtung wurde von 3mm absteigend gemessen*

C.3 Matlab-Quellcode

```
%-------------------------------------------------------------------------
% Programm zur automatisierten Messung mit Radmessplatz und Scope
%
% Version 0.1
% Datei:        rmp_stepper_scope_record.m
%-------------------------------------------------------------------------
%
% erstellt von: Martin Stahl
% erstellt am:  08.09.2009
%
% Änderungen:   09.01.2010
%               Anpassung an Demonstartor 2.Generation mit Regelplatine
%               Verstärkung wird automatisch eingestellt und übertragen
%               99% THD steht für ungültiges Ergebnis (Regelung war aktiv)
%-------------------------------------------------------------------------
%
% Beschreibung: Radmessplatz wird per Stepper automatisch verfahren,
%               anschließend werden per Oszilloskop Messwerte aufgezeichnet
%             und über LAN an den PC übertragen. Zusätzlich werden an jeder
%               Messposition Daten mit dem Demo-Board aufgezeichnet.
%
%
%-------------------------------------------------------------------------

clear all

%% set parameters

stepsize  = 0.05;
%e.g. 0:0.1:1.5 - from 0 to 1.5mm in 0.1mm steps
distances = [0:stepsize:7.05];

% [distance, necessary gain; .....]
amp_setting = [0.0 1; 1.75 2; 2.25 4; 3.0 8; ...
    3.5 16; 4.25 32; 4.75 64; 5.1 128];

scope_channels = [1, 4];    % Channel 1: U_diff
                            % Channel 4: reference signal

rec_length = 1e6;           % 1M Points Recordlength
rec_time = 4;               % 4s => 400ms/div
k=1;
count=0;
for i=1:length(amp_setting)
   if( mod( amp_setting(i,1) , stepsize) == 0)
       count = count + 1;
   end
end

if(length(amp_setting) ~= count)
    error('Check stepsize => amp_setting does not match with stepsize!');
end


%% --- Instrument verbinden ------------------------------------------------
disp('Instrument verbinden...');
% Create a VISA-USB object.
interfaceObj_scope = instrfind('tek', 'TCPIP::192.168.0.9::INSTR');
```

```
% Create the VISA-USB object if it does not exist
% otherwise use the object that was found.
if isempty(interfaceObj_scope)
    interfaceObj_scope = visa('tek', 'TCPIP::192.168.0.9::INSTR');
else
    fclose(interfaceObj_scope);
    interfaceObj_scope = interfaceObj_scope(1);
end

% Create a device object.
deviceObj_scope = icdevice('dpo4054.mdd', interfaceObj_scope);

% Connect device object to hardware.
connect(deviceObj_scope);
disp('Scope connected');

%% set scope parameters

set(deviceObj_scope.Channel(scope_channels), 'state', 'on');
set(deviceObj_scope.Channel(2), 'state', 'off');
set(deviceObj_scope.Channel(3), 'state', 'off');
% vertical gain U_diff
set(deviceObj_scope.Channel(scope_channels(1)), 'scale', 0.5);
% vertical gain ref_signal
set(deviceObj_scope.Channel(scope_channels(2)), 'scale', 1.0);
% vertical position U_diff
set(deviceObj_scope.Channel(scope_channels(1)), 'position', -6.0);
% vertical position ref_signal
set(deviceObj_scope.Channel(scope_channels(2)), 'position', +2.0);
% 20MHz Bandwidth
set(deviceObj_scope.Channel(scope_channels), 'bandwidthLimit', 'twenty');
% DC-Coupling
set(deviceObj_scope.Channel(scope_channels), 'coupling', 'dc');

set(deviceObj_scope.Acquisition, 'timebase', rec_time/10)% 400ms/div
set(deviceObj_scope.Acquisition, 'mode', 'highres') % Mode: High Resultion
set(deviceObj_scope.Acquisition, 'control', 'single')   % singleshot-mode

set(deviceObj_scope.Trigger, 'Source', 'channel1');   % triggersource ch1
set(deviceObj_scope.Trigger, 'Coupling', 'dc');    % trigger coupling: dc
set(deviceObj_scope.Trigger, 'level', 1.25);     % trigger-level at 1.25V
set(deviceObj_scope.Trigger, 'TriggerType', 'edge'); % trigger type: edge
set(deviceObj_scope.Trigger, 'slope', 'rising'); % trigger at rising edge

set(deviceObj_scope.Waveform, 'MaxNumberPoint', rec_length);
set(deviceObj_scope.Waveform, 'FirstPoint', 1);
set(deviceObj_scope.Waveform, 'EndingPoint', rec_length);

disp('Scope adjusted');

%% connect an initalize stepper

% Create the Interface object if it does not exist
% otherwise use the object that was found.
disp('Stepper verbinden...');

interfaceObj_stepper = instrfind('Name', 'Serial-COM1');

if isempty(interfaceObj_stepper)
```

```
    interfaceObj_stepper = serial('COM1');
else
    fclose(interfaceObj_stepper);
    interfaceObj_stepper = interfaceObj_stepper(1);
end

%interfaceObj = serial('COM1');
deviceObj_stepper = icdevice('tmcl.mdd', interfaceObj_stepper);
connect(deviceObj_stepper);

%%  set stepper parameters

set(deviceObj_stepper.Axis,'maximum_acceleration', 10);  % set acceleration
set(deviceObj_stepper.Axis,'maximum_positioning_speed', 500);%set max speed
set(deviceObj_stepper.Axis,'right_limit_switch_disable',0);%enable switches
set(deviceObj_stepper.Axis,'left_limit_switch_disable', 0);%enable switches
% disable soft stop for ref switch
set(deviceObj_stepper.Axis,'soft_stop_flag', 0);

disp('Stepper adjusted, start reference search...');

% reference search
% first right switch then left switch
set(deviceObj_stepper.Axis,'referencing_mode', 2);
% switch search speed
set(deviceObj_stepper.Axis,'referencing_search_speed', 300);
% speed after finding switch
set(deviceObj_stepper.Axis,'referencing_switch_speed', 50);
invoke(deviceObj_stepper,'rfs','start');

while(1)
    if( invoke(deviceObj_stepper,'rfs','status') == 0 )
        break;
    end
    pause(1);
end

disp('finished reference search');
disp('please reset THD demonstrator and check Frequency');
disp('press any key to continue...');
pause();


%% Get Data from Demo-Board and Scope
date_vec=datevec(date);

for k=1:99
    DirName = sprintf('%04d_%02d_%02d_%s_%02d', date_vec(1),...
        date_vec(2), date_vec(3), 'rmp', k);
    if( exist(DirName,'dir')==0 )
        break;
    end
end

mkdir(DirName); % make new directory for datafiles
DirName_complete = strcat(pwd,'\',DirName);

measure_demo = struct(   'distance',    {},...
                         'u_diff',      {},...
```

```
                        'u_br1',       {},...
                        'u_br2',       {},...
                        'u_pp',        {},...
                        'gain',        {},...
                        'hd_lut',      {},...
                        'hd_abs',      {},...
                        'mag',         {},...
                        'mag_v',       {},...
                        'mag_db',      {},...
                        'comp',        {},...
                        'angle',       {} );

measure_scope = struct('distance', {}, 'u_diff_y', {}, 'u_diff_yunit', ...
                       {}, 'ref_y', {}, 'ref_yunit', {});

not_get_measurement = false;
part = 1;
for i=1:length(distances)   % ------ Messpunkte durchlaufen ---------------


    %move to new position - 12800 steps = 1mm
    invoke(deviceObj_stepper,'mvp','absolute', distances(i)*12800);
    pause(15);              %wait for valid data

    if(~not_get_measurement)
     %   while(true)
            while(k<10)
                try
                    l=1;
                    while(l<100)
                  fprintf('%02d. try to read data from demonstrator\n',l);
                        var = get_measurement();
                        if(var.hd_lut > 98)
                            l = l+1;
                        else
                            l=101;
                        end

                        if(l==100)
                            disp('KEINE DATEM MEHR ERHALTEN');
                            break;
                        end
                    end

                    k=10;
                catch
                    k=k+1;
                    beep; pause(1); beep;
                    if(k==10)   % User Eingabe wenn keine Daten erhalten
                        fprintf('Stoppped at %04.2fmm...',distances(i));
                        while(true)
                            not_get_measurement = true;

                            break;

                        end
                    end
                end
            end
```

```
            k=1;                % Eingestellt Verstärkung überprüfen

         fprintf('  THD bei %04.4fmm: %d%% \n',distances(i), var.hd_lut);

         %capture data from Demo-Board
         measure_demo(end+1) = struct('distance',    distances(i),...
                                      'u_diff',      var.u_diff,...
                                      'u_br1',       var.u_br1,...
                                      'u_br2',       var.u_br2, ...
                                      'u_pp',        var.u_pp,...
                                      'gain',        var.gain,...
                                      'hd_lut',      var.hd_lut,...
                                      'hd_abs',      var.hd_abs,...
                                      'mag',         var.mag,...
                                      'mag_v',       var.mag_v,...
                                      'mag_db',      var.mag_db, ...
                                      'comp',        var.comp,...
                                      'angle',       var.angle );
%     else
%         fprintf('  Distanz: %04.4fmm \n',distances(i));
%     end

    disp('Scope start singleshot');
    %capture data from Scope
    measure_scope(end+1).distance = distances(i);
    set(deviceObj_scope.Acquisition, 'state', 'run') % start singleshot
    % wait for recording
    pause(11* get(deviceObj_scope.Acquisition, 'timebase'));

    %signal U_diff, channel 1
    [measure_scope(end).u_diff_y, temp1,...
     measure_scope(end).u_diff_yunit, temp2] = ...
     invoke(deviceObj_scope.Waveform, 'readwaveform', 'channel1');

    %signal reference-sensor channel 4
    [measure_scope(end).ref_y, temp1,...
     measure_scope(end).ref_yunit, temp2] = ...
     invoke(deviceObj_scope.Waveform, 'readwaveform', 'channel4');
    disp('Scope read completed');

    if( mod(i,20) == 0)
      %%
       parameters = struct('startpoint', distances(1), 'endpoint', ...
             distances(end),'stepsize', (distances(2)-distances(1)),  ...
             'samplerate', rec_length/rec_time,'amp_setting', amp_setting);

       for k=1:99
         FileName_Out=sprintf('%04d_%02d_%02d_%s_%02d_part%02d.rmp.mat',...
             date_vec(1), date_vec(2), date_vec(3), 'rmp', k, part);
            if( exist(FileName_Out,'file')==0 )
               break;
            end
       end
       part = part + 1;
       disp(['Saving file to: ' FileName_Out]);
       save(strcat(DirName_complete,'\',FileName_Out),...
           'parameters', 'measure_demo', 'measure_scope');

       clear measure_demo;
```

```
        clear measure_scope;
        clear parameters;
        %%
        measure_demo = struct('distance',    {},...
                              'u_diff',      {},...
                              'u_br1',       {},...
                              'u_br2',       {},...
                              'u_pp',        {},...
                              'gain',        {},...
                              'hd_lut',      {},...
                              'hd_abs',      {},...
                              'mag',         {},...
                              'mag_v',       {},...
                              'mag_db',      {},...
                              'comp',        {},...
                              'angle',       {} );
        measure_scope = struct('distance', {}, 'u_diff_y', {}, ...
                  'u_diff_yunit', {},'ref_y', {}, 'ref_yunit', {});
    end

    else % not_get_measurement = TRUE
        fprintf('  Distanz: %04.4fmm \n',distances(i));
    end


end

%% saving file at last
beep;beep;
parameters = struct('startpoint', distances(1), 'endpoint', ...
     distances(end),'stepsize', (distances(2)-distances(1)),  ...
        'samplerate', rec_length/rec_time,'amp_setting', amp_setting);

        for k=1:99
          FileName_Out=sprintf('%04d_%02d_%02d_%s_%02d_part%02d.rmp.mat',...
           date_vec(1), date_vec(2), date_vec(3), 'rmp', k, part);
             if( exist(FileName_Out,'file')==0 )
                 break;
             end
        end
        part = part + 1;
        disp(['Saving file to: ' FileName_Out]);
        save(strcat(DirName_complete,'\',FileName_Out), ...
            'parameters', 'measure_demo', 'measure_scope');

        clear measure_demo;
        clear measure_scope;
        clear parameters;


disp('.. Ende');

%% close
disconnect(deviceObj_scope);
disconnect(deviceObj_stepper);
delete([deviceObj_scope interfaceObj_scope]);
delete([deviceObj_stepper interfaceObj_stepper]);
```

```
%-------------------------------------------------------------------------
% Programm zur Auswertung der aufgezeichneten Demoboard-Daten
%
% Version 1.0
% Datei:        rmp_stepper_scope_record_analyze.m
%-------------------------------------------------------------------------
%
% erstellt von: Martin Stahl
% erstellt am:  10.09.2009
%
% geändert von: Niels Jegenhorst
%
% Änderungen:   Einlesen der Daten, Darstellung, Speicherung der Plots,
%               Speicherung der Verläufe in .mat File
%-------------------------------------------------------------------------
%
% Beschreibung: Aufgrund der Datenmenge werden mehrere mat-files eingelesen
%
% Folgende Plots werden erstellt:  - THD over airgap
%                                  - Magnitudes over airgap
%                                  - Phases over airgap
%                                  - Upeak over airgap gained / not gained
%
%-------------------------------------------------------------------------


clear all
close all

EXPORT_PDF_FIGURES  = true;
CALC_HARM_DIST_UOFF = true;

disp('-- Start rmp_stepper_scope_record_analyze.m --');

    % Verzeichnis zum Einlesen auswählen
DirName = uigetdir(pwd,'Choose directory with data-files');
cd(DirName);
list_of_files   = ls('*.rmp.mat');
number_of_files = size(list_of_files, 1);


%% --- Einlesen der Messdaten ------------------------------------------
k = 1;
for i=1:number_of_files                % Dateien durchlaufen

    disp(['loading ' list_of_files(i,:)]);
    load(list_of_files(i,:));          % jeweilige Datei laden

    if(i == 1)
        amp_setting = parameters.amp_setting;   % Umschaltpunkte
        stepsize    = parameters.stepsize;      % Schrittweite in mm
        startpoint  = parameters.startpoint;    % Anfangspunkt in mm
        endpoint    = parameters.endpoint;      % Endpunkt in mm

        distance = startpoint : stepsize : endpoint;
        distance = distance';
    end
                                       % einzelne Strukturen auslesen
    for j=1:length(measure_demo)
```

```
        distance(k)      = measure_demo(j).distance;
        harm_dist_lut(k) = measure_demo(j).hd_lut;
        harm_dist_abs(k) = measure_demo(j).hd_abs;
        magnitudes(k,:)  = measure_demo(j).mag(:,1);
        magnitudes_v(k,:)= measure_demo(j).mag_v(:,1);  % in Volt
        complex(k,:)     = measure_demo(j).comp(:,1);
        phases(k,:)      = measure_demo(j).angle(:,1);
        u_peak(k)        = measure_demo(j).u_pp;        % in Volt
        u_diff(k,:)      = measure_demo(j).u_diff;      % in Volt
        k=k+1;
    end
    clear measure_scope                        % cleanup workspace
    clear measure_demo
end

%% --- Nachverarbeitung der Daten ----------------------------------------
                                    % Normierung der Harmonischen
magnitudes_normed = magnitudes / max(magnitudes(:,1));

if(CALC_HARM_DIST_UOFF)
    % ___ Berechnung des Klirrfakors anhand von U_diff ____________________
MAX_HARM = 5;                       % Anzahl der Harmonischen
spp = 64.0;                         % Samples pro Periode

harm_dist_udiff = zeros(k-1, 1);
omega_h0  = 1.0/spp;
omega_inc = -omega_h0;

for i=1 : k-1              % ___ Messpunkte durchlaufen _______________
    re    = zeros(1, MAX_HARM);
    im    = zeros(1, MAX_HARM);
    h_abs = zeros(1, MAX_HARM);

    for n = 1 :1: spp               % ___ samples durchlaufen __________
        omega_inc = omega_inc + omega_h0;       % == omega_inc * n
        omega_hx  = 0.0;
        for harm = 1:MAX_HARM       % ___ harmonische durchlaufen _____
            omega_hx = omega_hx + omega_inc;    % == omega_hx * harm
            re(harm) = re(harm) + u_diff(i, n) * cos(omega_hx * 2.0*pi);
            im(harm) = im(harm) - u_diff(i, n) * sin(omega_hx * 2.0*pi);
        end
    end
    for harm = 1:MAX_HARM           % Betragsquadrat berechnen
        h_abs(harm)  = re(harm)^2 + im(harm)^2;
    end                             % Klirrfaktor berechnen
    harm_dist_udiff(i) = 100.0 * sqrt( sum(h_abs(2:MAX_HARM)) / ...
                                       sum(h_abs(1:MAX_HARM))  );
end
end

%% --- plot harmonic distortion over distance -----------------------------

filename = list_of_files(1,:);

h=figure('Name','Harmonic Distortion over Distance');
if(CALC_HARM_DIST_UOFF)
    plot(distance(1:k-1), harm_dist_lut,...
         distance(1:k-1), harm_dist_abs,...
         distance(1:k-1), harm_dist_udiff);
```

```
else
    plot(distance(1:k-1), harm_dist_lut,...
         distance(1:k-1), harm_dist_abs);
end
grid on;
xlim([0 distance(k-1)+0.2]);
ylabel('hd in %');
xlabel('Distance in mm');
if(CALC_HARM_DIST_UOFF)
    legend('hd displayed by board', ...
           'hd calculated with magnitudes',...
           'hd calculated by sampled U_{diff}');
else
    legend('hd displayed by board', ...
           'hd calculated with magnitudes');
end
h=title(['Harmonic Distortion over Distance, Stepsize = ',...
         num2str(stepsize) 'mm (' filename ')']);
set(h,'interpreter','none');
if(EXPORT_PDF_FIGURES)
    saveas(h, [filename '__dm_hd.pdf']);
end

%% --- plot magnitudes over distance ----------------------------------------

h=figure('Name','Magnitudes over Distance');
plot(distance(1:k-1), magnitudes_v);
grid on;
xlim([0 distance(k-1)+0.2]);
ylabel('Magnitudes');
xlabel('Distance in mm');
legend('1^s^t harm.', '2^n^d harm.', '3^r^d harm.',...
       '4^t^h harm.', '5^t^h harm.');
h=title(['Magnitudes over Distance, Stepsize = ',...
         num2str(stepsize) 'mm (' filename ')']);
set(h,'interpreter','none');
if(EXPORT_PDF_FIGURES)
    saveas(h, [filename '__dm_magnitudes.pdf']);
end

%% --- plot phases over distance ---------------------------------------------

h=figure('Name','Phases over Distance');
plot(distance(1:k-1),phases(:,1:3));
grid on;
xlim([0 distance(k-1)+0.2]);
ylabel('Phases in rad');
xlabel('Distance in mm');
legend('1^s^t harmonic', '2^n^d harmonic', '3^r^d harmonic')%...
   %    '4^t^h harmonic', '5^t^h harmonic');
h_t=title(['Phases over Distance, Stepsize = ',...
         num2str(stepsize) 'mm (' filename ')']);
set(h_t,'interpreter','none');
if(EXPORT_PDF_FIGURES)
    saveas(h, [filename '__dm_phases.pdf']);
end

%% --- plot Upeak over distance ----------------------------------------------
```

```
gain=25.0;
for i=1 : k-1
    sw=find( distance(i)==amp_setting(:,1) );

    if(~isempty(sw))       % is_not_empty
        gain = 25.0 * amp_setting(sw,2);
    end
    u_peak_ug(i) = u_peak(i) / gain;
end

h=figure('Name','U_pp over Distance');
subplot(2,1,1);
    plot(distance(1:k-1), u_peak_ug);
    grid on;
    xlim([0 distance(k-1)+0.2]);
    ylabel('U_{pp} in V');
    xlabel('Distance in mm');
    h_t=title(['U_pp without gain, Stepsize = ',...
               num2str(stepsize) 'mm (' filename ')']);
    set(h_t,'interpreter','none');

subplot(2,1,2);
    plot(distance(1:k-1), u_peak);
    grid on;
    xlim([0 distance(k-1)+0.2]);
    ylabel('U_{pp} in V');
    xlabel('Distance in mm');
    h_t=title('U_pp with gain');
    set(h_t,'interpreter','none');
if(EXPORT_PDF_FIGURES)
    saveas(h, [filename '__dm_upeak.pdf']);
end

%% --- Save Data ------------------------------------------------------------

save([filename '__dm_analyzed.mat'], 'harm_dist_lut', 'harm_dist_abs',...
                                     'harm_dist_udiff',...
                                    'magnitudes_v', 'magnitudes_normed',...
                                     'stepsize');

disp('-- End rmp_stepper_scope_record_analyze.m --');
```

Abkürzungsverzeichnis

ABS Antiblockiersystem

ADC Analog Digital Converter

AMR-Effekt anisotroper magnetoresistiver Effekt

DAC Digital Analog Converter

ISR Interrupt-Service-Routine

JTAG Joint Test Action Group

k Klirrfaktor

MR magnetoresistiv

PowerOn Start des Controllersystems

TMCL Trinamic Motion Control Language

UART Universal Asynchronous Receiver Transmitter

Tabellenverzeichnis

Abbildungsverzeichnis

Index

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 26. Februar 2010	
Ort, Datum	Unterschrift