HAW HAMBURG

Bachelorthesis

Thorbjörn Mehm

Schaltungsentwurf und Mikrocontrollersteuerung für ein Tunnel-Magnetoresistives Sensor-Array

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Thorbjörn Mehm

Schaltungsentwurf und Mikrocontrollersteuerung für ein Tunnel-Magnetoresistives Sensor-Array

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Elektrotechnik und Informationstechnik
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. Pawel Buczek
Abgegeben am: 07. Juni 2019

Thorbjörn Mehm

Thema der Bachelorarbeit

Schaltungsentwurf und Mikrocontrollersteuerung für ein Tunnel-Magnetoresistives Sensor-Array

Stichworte

Magnetoresistive-Sensoren, Automobil-Sensoren, Tunnel-Magnetoresistiver Effekt, Vortex, Magnetfeld, Sensor-Array, Vollbrücken, Ansteuerung, Sinus, Cosinus, Messplatz, analoger Multiplexer, Platinenentwurf, Gebermagnet

Kurzzusammenfassung

In der vorliegenden Bachelorarbeit wird ein Sensor-Array als Funktionsmuster basierend auf dem Tunnel-Magnetoresistiven Effekt erstellt und gefertigt. Dazu wird eine Platine bestehend aus 8x8 Sensoren aufgebaut. Die Steuerung und das Auslesen erfolgt dabei mit einem Mikrocontroller. Das Sensor-Array soll Daten für den Entwurf und die Erprobung neuer Auswertealgorithmen liefern. Für den Funktionsnachweis werden abschließende Messungen durchgeführt.

Thorbjörn Mehm

Title of the bachelor thesis

Circuit design and microcontroller programming for a tunnel-magnetoresistive sensor-array

Keywords

magnetoresistive sensors, automotive sensors, tunnel-magnetoresistive effect, vortex, magnetic field, sensor-array, bridge circuit, controlling, sine, cosine, measuring station, circuit design, magnet

Abstract

In this bachelor thesis a functional model of a sensor-array based on tunnel-magnetoresistive sensors has to be designed. Therefor an electric circuit board with 8x8 sensors will be constructed. The sensor-array will be controlled by a microcontroller. It should deliver data sets for the testing of new evaluation algorithms. Some measuarements will occur for proving the functionality.

Vorwort

Ich möchte mich zunächst für die Möglichkeit bedanken, im Projekt ISAR meine Bachelorarbeit schreiben zu können. Besonderen Dank möchte ich dafür meinem betreuenden Erstprüfer Herrn Prof. Dr-Ing. Karl-Ragmar Riemschneider aussprechen. Für die Übernahme der Zweitprüfung möchte ich mich herzlich bei Prof. Dr. Pawel Buczek bedanken.

Zudem möchte ich noch Herrn M.Sc. Thorben Schüthe danken. Er stand mir während der gesamten Bearbeitungszeit mit hilfreichen Tipps, Anregungen und fachlichem Rat zur Seite. Ebenfalls möchte ich mich bei allen Angehörigen der Arbeitsgruppen ISAR und BATSEN für das angenehme Forschungsklima bedanken.

Großen Dank möchte ich außerdem Herrn Dipl.-Ing. Günter Müller für seinen fachlichen Rat, das Korrekturlesen und die Kommunikation mit dem Leiterplattenbestücker bedanken.

Mein Dank gilt ebenfalls meinen Freunden und Kommilitonen, welche mich während des Studiums und darüber hinaus begleitet haben.

Weiterhin möchte ich mich bei meiner Familie bedanken. Insbesondere sind dort meine Eltern, mein Bruder und meine Großeltern zu nennen. Diese haben immer an mich geglaubt und mich während des gesamten Studiums unterstützt. Sofern ich Hilfe gebraucht habe, waren sie immer zur Stelle.

Zum Schluss möchte ich mich natürlich bei meiner Freundin bedanken. Ich danke ihr für ihre Geduld und ihr Verständnis während der Bearbeitungszeit. Beim Korrekturlesen war sie mir ebenfalls eine große Hilfe.

Vielen Dank!

Inhaltsverzeichnis

Anhang

1 Einleitung

In der elektronischen Messtechnik spielen Sensoren eine grundlegende Rolle. Sie haben die Aufgabe, physikalische Effekte wie z. B. Temperatur, Licht, Druck oder auch Magnetfelder zu erfassen und in elektrisch messbare Größen umzusetzen [2]. In der Automobilindustrie werden zunehmend Magnetfeldsensoren für verschiedenste Anwendungen genutzt. Die HAW Hamburg arbeitet in diesem Bereich mit ihren Partnern am Forschungsprojekt ISAR (Signalverarbeitung für Integrated Sensor-Arrays). Gefördert wird das Projekt durch das Bundesministerium für Bildung und Forschung und die Firma NXP Semiconductors. Das Hauptziel ist eine Signalverarbeitung für Sensor-Arrays, basierend auf dem TMR-Effekt (Tunnel-Magnetoresistiver Effekt), für den Einsatz in der Automobilindustrie. Das Ziel der Bachelorarbeit ist es, ein Sensor-Array mit TMR-Sensoren aufzubauen.

Inhaltlich sollen in der Bachelorarbeit zunächst die Grundlagen magnetoresistiver Sensoren behandelt werden. Dabei wird insbesondere auf den TMR-Effekt eingegangen. Anschließend werden verschiedene Ansätze für die Beschaltung des Sensor-Arrays erläutert. Sofern die Beschaltung festgelegt ist, folgt der praktische Entwurf der Leiterplatte. Nach Fertigstellung des Sensor-Arrays schließen sich die Inbetriebnahme und der Funktionstest an. Für das Auslesen aller Sensorsignale wird eine Software programmiert und beschrieben. Anschließend werden eine Visualisierung der Messwerte und die Aufnahme und Auswertung von Messdaten vorgenommen. Abschließend werden eine Zusammenfassung und Reflexion der Arbeit sowie ein Ausblick gegeben.

1.1 Magnetische Sensoren im Automobil

In modernen Automobilen werden zunehmend Sensoren eingesetzt. Steuer- und Regelfunktionen, welche in früheren Fahrzeugen noch auf mechanischem Wege realisiert worden sind, werden zunehmend durch eine elektronische Regelung verdrängt [5]. Für elektronische Regelungen sind Sensoren von fundamentaler Bedeutung, da diese physikalische Größen, welche es zu regeln gilt, in elektrisch messbare Größen umformen. Ohne Sensoren wäre somit nur eine Steuerung möglich. Fanden sich die ersten Anwendungen von Sensoren in Automobilen zunächst insbesondere in der Motorregelung, so werden heute eine Vielzahl an Sensoren in den verschiedensten Anwendungsbereichen genutzt. Da Automobile eine hohe Anzahl an drehenden Teilen beinhalten, besteht ein großes Interesse an Sensoren, welche der Erfassung von Rotationen dienen. Mit Hilfe von magnetischen Sensoren lassen sich Drehzahlen und Winkelinformationen berührungslos

erfassen. Durch Verschleiß hervorgerufene Kontaktprobleme können somit nicht auftreten. Einsatzmöglichkeiten von magnetischen Sensoren sind z. B. :

- Drehzahlmessung
- Geschwindigkeitsmessung
- Winkelstellung der Drosselklappe
- Servolenkung (Drehmomentsensor)
- Antiblockiersystem (ABS)

Diese Beispiele nutzen die physikalischen Größen der Drehzahl und des Winkels. Weitere Anwendungsmöglichkeiten von magnetoresitiven Sensoren liegen in der Strom- und Felderfassung.

1.2 Ziel dieser Arbeit

Der Beitrag zum Gesamtprojekt besteht darin, ein Sensor-Arrays als Funktionsmodell zu erstellen. Die räumliche Lage eines Permanentmagneten stellt die Nutzinformation dar. Durch die Arraystruktur wird eine Fehlerkorrektur möglich. Die verwendeten TMR-Sensoren vereinen zwei Vollbrücken in einem Gehäuse, wobei alle Brückensignale als Pins herausgeführt werden. Jeweils eine Brücke generiert das differentielle Sinussignal und die andere das Cosinussignal. Insgesamt sollen 64 Sensoren in einer 8x8-Anordnung verwendet werden. Dabei ist darauf zu achten, eine möglichst kompakte Anordnung zu designen. Auf eine Vorarbeit, welche mit anderen TMR-Sensoren mit jeweils zwei Halbbrücken in einem Gehäuse arbeitet, kann zurückgegriffen werden. Das Funktionsmodell soll dabei als Beispiel der Beschaltung und des Auslesens der Sensordaten dienen. Ebenfalls soll es Daten liefern, um die bereits erstellten Auswertealgorithmen testen und neue entwerfen zu können.

2 Grundlagen

Dieses Kapitel beinhaltet die Grundlagen zu den magnetoresistiven Sensoren. Dabei wird ein Überblick über die verschiedenen Technologien zur Erfassung von Magnetfeldern gegeben. Auf den TMR-Effekt soll hierbei genauer eingegangen werden, da das Sensorarray, welches im Rahmen dieser Bachelorarbeit entworfen wird, aus diesen Sensoren besteht. Zuletzt wird im Detail auf die Eigenschaften des verwendeten Sensors des Herstellers TDK, Typ TAS2141-AAAB, eingegangen.

2.1 Magnetische Sensoren

Im Jahre 1857 wurde der magnetoresistive Effekt seitens des britischen Physikers William Thomson entdeckt. Er erkannte, dass sich der elektrische Widerstand eines stromdurchflossenen Leiters durch äußere Magnetfelder verändert. Dieser Effekt wurde als „anisotroper magnetoresistiver Effekt“, kurz AMR-Effekt, bezeichnet und weist eine Widerstandsänderung von nur wenigen Prozent auf. Sensortechnisch wurde dieser Effekt erst vor etwa 30 Jahren, durch Voranschreiten der Dünnschichttechnik, kommerziell nutzbar. Als weit verbreitete Anwendung wurde er z. B. in Schreib-/Leseköpfen von Festplatten genutzt. Als Weiterentwicklung wurde Ende der 1980er Jahre der GMR-Effekt (Giant magnetoresistive) entdeckt. Dabei wurden Widerstandsänderungen von über 50% gemessen. Der Tunnel-magnetoresistive Effekt ist eine weitere Technologie zum Erfassen von Magnetfeldern. Bekannt ist er seit den 1970er Jahren, industriell nutzbar ist er jedoch erst seit wenigen Jahren durch die Weiterentwicklung der Prozesstechnik [10].

2.1.1 AMR-Effekt

Der AMR-Effekt ist in ferromagnetischen Materialien beobachtbar. Der Winkel zwischen Strom und Magnetfeld beeinflusst den spezifischen Widerstand in diesen Materialien (vgl. Abbildung 2.1). Stehen Strom und Magnetfeld senkrecht zueinander, wird der spezifische Widerstand minimal und bei paralleler bzw. antiparalleler Ausrichtung des Magnetfelds zur Stromflussrichtung wird der spezifische Widerstand maximal. Somit kann keine gesamte Umdrehung eindeutig erfasst werden, da sich die Widerstandsänderung ab 180° wiederholt. Die Widerstandsänderung wird durch Gleichung 2.1 beschrieben [10]. Sie besagt, dass der Gesamtwiderstand des AMR-Sensors vom Winkel um einen mittleren Widerstandswert abhängt. Im Cosinusterm fällt ebenfalls die Periodizität von 180° auf.

$$R(\alpha) = R_m + \frac{\Delta R}{2} \cdot cos(2\alpha) \tag{2.1}$$

mit

R_m = mittlerer Widerstand

ΔR = Widerstandsänderung

α = Winkel

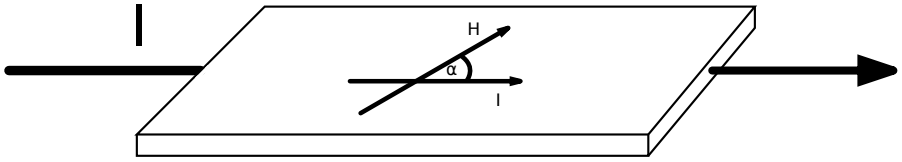


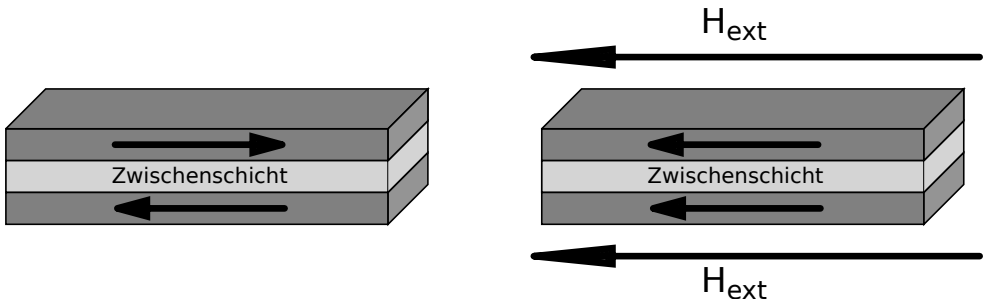
Abbildung 2.1: Prinzip des AMR-Effekts. Winkel zwischen Stromrichtung und Magnetfeld bewirkt Widerstandsänderung.

2.1.2 GMR-Effekt

Beim GMR-Effekt ist der Widerstand vom Winkel zwischen magnetisierten Dünnschichten abhängig. Im einfachsten Fall sind dies zwei Dünnschichten, welche durch eine dünne, unmagnetische Zwischenschicht getrennt werden. In einer solchen Anordnung richten sich die Magnetfelder der magnetischen Schichten automatisch antiparallel aus (vgl. Abbildung 2.2(a)). Dieses Phänomen wird „antiferromagnetische Kopplung“genannt. Durch den Einfluss äußerer Magnetfelder drehen sich die Magnetfelder in den Schichten. Dadurch verändert sich der Winkel zwischen beiden Schichten und damit auch der Widerstand. Sind die Magnetisierungen antiparallel, so ist der Gesamtwiderstand maximal (vgl. Abbildung 2.2(b)). Wenn die Magnetisierungen in die gleiche Richtung zeigen, wird der Gesamtwiderstand minimal. Ohne externes Magnetfeld hat der GMR-Sensor also seinen größten Widerstand. Durch ein externes Magnetfeld kann also eine Widerstandsabnahme erzielt werden [10].
Die Widerstandsänderung ist abhängig von der Ausrichtung der magnetischen Spins. Haben die Elektronen der benachbarten magnetischen Lagen die gleiche Spin-Ausrichtung, so können die Elektronen die dünne Zwischenschicht leicht passieren. Bei entgegengesetzter Spin-Richtung werden die Elektronen gestreut und der Widerstand wird größer. Die Periodizität von GMR-Sensoren liegt bei 360°.

2.1.3 TMR-Effekt

Der Tunnel-magnetoresistive Effekt (TMR) beruht auf dem quantenmechanischen Tunneleffekt und ist somit mit der klassischen Physik nicht erklärbar. Der quantenmecha-



(a) Hoher Widerstand durch antiparallele Ausrichtung der magnetischen Schichten.

(b) Niedriger Widerstand durch parallele Ausrichtung der magnetischen Schichten, verursacht durch externes Magnetfeld

Abbildung 2.2: GMR-Sensorprinzip mit und ohne externem Magnetfeld.

nische Tunneleffekt besagt, dass Elektronen durch eine extrem dünne Isolationsschicht durchdringen bzw. tunneln können. Diese Isolationsschicht wird auch Tunnelbarriere genannt. Im einfachsten Fall besteht ein TMR-Sensor aus zwei ferromagnetischen Dünnschichten, welche durch die Tunnelbarriere getrennt sind. Der Tunnelstrom hängt von der Tunnelbarriere selbst und von der magnetischen Orientierung der magnetisierten Dünnschichten ab [10]. Der Grundaufbau eines TMR-Elements ist dem des GMR-Elements ähnlich. Die Zwischenschicht des GMR-Sensors wird jedoch bei dem TMR-Sensor zur Tunnelbarriere, und die Richtung des Stromflusses ändert sich, da der Stromfluss nicht mehr in der Zwischenschicht stattfindet, sondern durch die Tunnelbarriere. In der Übersicht von AMR, GMR und TMR (Abbildung 2.3) wird dies deutlich.

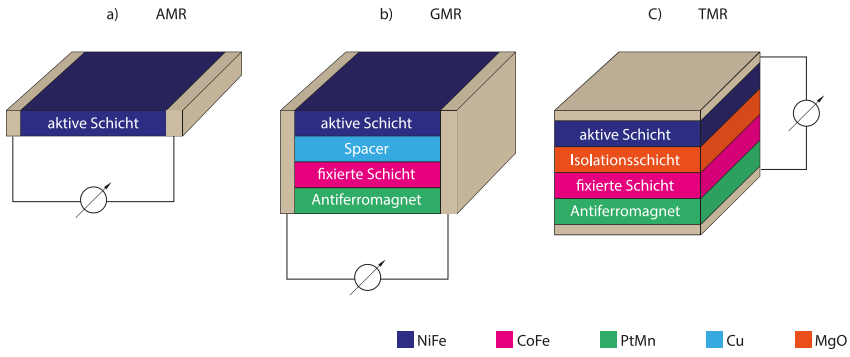


Abbildung 2.3: Prinzip von AMR-, GMR- und TMR-Sensoren. Beim AMR- und GMR-Sensor erfolgt der Stromfluss in der Schicht (horizontal). Beim TMR-Sensor erfolgt der Stromfluss durch die Tunnelbarriere (vetikal) [8].

Ausführungen von TMR-Sensoren

TMR-Sensoren lassen sich in zwei Ausführungen, linear und vortex, unterschieden. Der Unterschied liegt in der aktiven magnetischen Schicht. Bei linearen TMR-Sensoren ist die aktive Schicht linear angeordnet. Beim Vortex-TMR ist die aktive Schicht wirbelförmig. Äußere Magnetfelder wirken auf diese Schicht ein. Abbildung 2.4 illustriert beide Ausführungen. Beim linearen TMR-Sensor sorgt das äußere Magnetfeld dafür, dass sich das Magnetfeld in der aktiven Schicht entsprechend mitdreht. Der Widerstandswert verhält sich linear bis zur Sättigung der aktiven Schicht. Beim Vortex-TMR-Sensor wird zunächst der Mittelpunkt verschoben, die Widerstandsänderung verhält sich in diesem Bereich relativ linear. Wird das äußere Magnetfeld stärker, so ordnet sich das Magnetfeld der aktiven Schicht auch linear an. Alle inneren Dipole zeigen dann in die gleiche Richtung und der Mittelpunkt des Wirbels verschwindet [15]. In Abbildung 2.5 wird dieses Verhalten gezeigt.

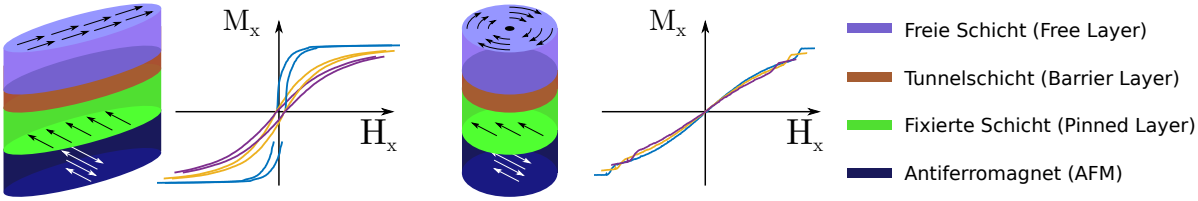


Abbildung 2.4: TMR-Sensoren links in linearer Ausführung, rechts die wirbelförmige Vortex-Variante [7].

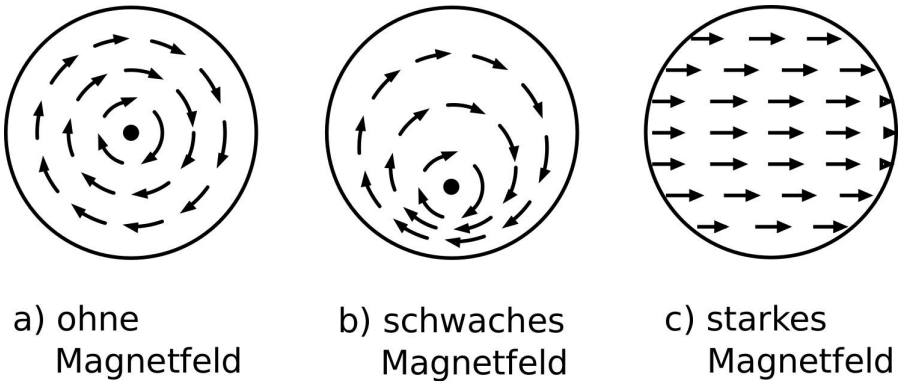


Abbildung 2.5: Verlauf des magnetischen Wirbels. Links: ohne externes Magnetfeld. Mitte: schwaches externes Magnetfeld. Rechts: starkes externes Magnetfeld

2.1.4 Anwendungen

Die typische Anwendung von Winkelsensoren ist die Lage- bzw. Winkelerfassung. Eine weitere Anwendung besteht in der Rotationsmessung. In beiden Fällen wird ein Gebermagnet benötigt. Abbildung 2.6 zeigt die prinzipielle Anordnung.

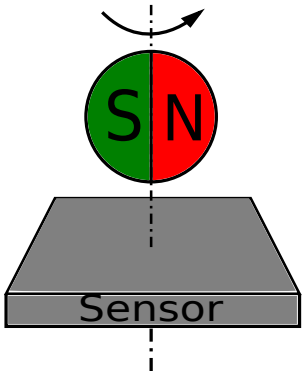


Abbildung 2.6: Prinzipielle Anordnung von Gebermagnet und Sensor. Die Rotation bzw. der Winkel wird erfasst.

2.2 Sensor-Array

Das Sensor-Array soll mit TMR-Sensoren aus der TAS-Serie des Herstellers TDK bestückt werden, da dieser bessere Eigenschaften als der zuvor verwendete Sensor von NVE, Typ AAT 001-10E [3], verspricht. Die genaue Typenbezeichnung lautet TAS2141-AAAB [4] und wird im kompakten achtpoligen TSSOP-Gehäuse geliefert. Der Sensor beinhaltet zwei getrennt voneinander nutzbare Vollbrücken mit differentiellen Sensorausgängen $\pm U_{sin}$ und $\pm U_{cos}$, welche eine Offsetkompensation ermöglichen. Durch seine hohe differentielle Ausgangsspannung von typischerweise 300 mV/V_{CC} kommt er ohne zusätzliche, aktive Verstärkerschaltungen aus. Technologiebedingt lässt sich, im Vergleich zum AMR-Sensor, eine komplette Drehung um 360°, bei maximalem Winkelfehler von 0,6° erfassen. Durch einen Brückenwiderstand im Bereich von $4 - 6\,\mathrm{k\Omega}$ stellt der Sensor eine niederohmige Quellimpedanz für das nachfolgende Messsystem zur Verfügung. Eine Übersicht der wichtigsten Parameter liefert Tabelle 2.1 [4].

Tabelle 2.1: Wichtige Parameter TDK TAS-2141-AAAB

Symbol	Parameter	Bedingungen	min.	typ.	max.	Einheit
V_{CC}	Versorgungsspannung		3	5	5,5	V
H_{ex}	externes Magnetfeld		20		80	30 mT
R_{bridge}	Brückenwiderstand	$T = 25\,°C, H_{ex} = 30\,mT$	4	5	6	kΩ
V_{out}	diff. Ausgangsspannung	$T = 25\,°C, H_{ex} = 30\,mT$	0,24	0,30	0,36	V_{PP}/V_{CC}

3 Hardwareentwurf

Dieses Kapitel ist in zwei Abschnitte gegliedert. Zunächst erfolgt der theoretische Teil, in welchem die verschiedenen Schaltungskonzepte vorgestellt werden. Am Ende folgt eine Bewertung der Vor- und Nachteile sowie eine Begründung der Entscheidung für die gewählte Variante. Anschließend geht es um die Aspekte der Umsetzung. Dabei wird insbesondere auf den Schaltungs- und Layoutentwurf mittels des EDA-Programms (Electronic Design Automation) Eagle eingegangen.

3.1 Anforderungen und Schaltungskonzepte

Das Sensor-Array soll aus 8x8 Sensoren bestehen. Die Anordnung soll möglichst kompakt sein und die Anschlussführung zwischen Sensor- und Controllerplatine ist zu optimieren. Eine kompakte Anordnung ist erforderlich, um schwache Magnetfelder gut erfassen zu können. Aus der 8x8-Anordnung ergibt sich, dass 64 Sensoren zu erfassen sind und dabei jeder Sensor vier Sensorsignale ausgibt. Insgesamt sind 256 Brückensignale zu erfassen. Bei dem verwendeten Mikrocontroller-Board handelt es sich um das EK-TM4C1294XL Connected LaunchPad von Texas Instruments [11], welches unter anderem 20 analoge Eingänge mit jeweils 12 Bit Auflösung bereitstellt. Im Folgenden soll eine Übersicht denkbarer Lösungsansätze gegeben werden. Dabei wird die Struktur welche in vorherigen Arbeiten gewählt wurde vorgestellt. Die notwendigen Änderungen werden herausgearbeitet, die für den verwendeten Sensor erfolgen müssten. Ebenfalls wird eine Variante mit Multiplexern und externen Analog-Digital-Umsetzern aufgezeigt. Abschließend wird eine Übersicht der Varianten mit ihren jeweiligen Vor- und Nachteilen aufgezeigt und bewertet.

3.1.1 Zeitmultiplexverfahren der Versorgungsspannung

Bei dem Zeitmultiplexverfahren der Versorgungsspannung wird das Sensor-Array in einer Matrixstruktur angeordnet, bei welcher die Sensorausgänge zeilenweise parallel geschaltet sind. Jede Spalte bekommt ihre eigenen Versorgungsleitungen. Diese müssen als Tri-State-Schaltung ausgeführt werden, um die nicht genutzten Versorgungspins hochohmig schalten zu können. Nun wird diejenige Spalte, welche ausgelesen werden soll, niederohmig eingeschaltet und alle anderen Versorgungsleitungen hochohmig. Die Sensoren der zugehörigen Spalte können nun ausgelesen werden, da sich die zeilenweise parallel geschalteten Sensoren nicht mehr beeinflussen. Nach dem Auslesen wird die nächste Spalte

aktiviert und die anderen Versorgungspins hochohmig geschaltet. Dieser Vorgang wird zyklisch wiederholt, bis das komplette Array erfasst worden ist. In Abbildung 3.1 wird das Einlesen und das Zeitmultiplex-Verfahren anhand eines 3x3-Sensor-Arrays veranschaulicht.
In den Vorarbeiten [1] wurde für die Spannungsversorgung auf GPIO-Pins des Mikrocontrollers zurückgegriffen. Als kritisch ist hier die Spannungsversorgung der Sensoren zu nennen, da HIGH- und LOW-Pegel keiner exakten Spannung entsprechen, sondern hierfür Spannungsbereiche definiert sind. Für eine genaue Messung ist diese Variante daher weniger geeignet. Mit dem in dieser Arbeit verwendeten Sensor würde dies bedeuteten, dass für alle acht Spalten, jeweils V_{Cos}, V_{Sin} und G_{Cos}, G_{Sin}, ein digitaler GPIO aufgewendet werden muss. Da die Vollbrücken voneinander unabhängig sind, lässt sich der Aufwand reduzieren, wenn die Versorgungspins für V_{Sin} und V_{Cos} sowie G_{Sin} und G_{Cos} parallel geschaltet werden. Insgesamt wären dann 16 GPIO-Pins nötig. Sollen alle gleichartigen Sensorausgänge zeilenweise parallel geschaltet werden, so sind pro Zeile vier ADC-Eingänge nötig, für alle Zeilen folglich 32. Auf diese Weise könnte jeweils der Sinus- und Cosinus-Wert parallel erfasst werden. Das Mikrocontroller-Board stellt jedoch nur 20 ADC-Eingänge bereit. Um dieses Problem zu umgehen, werden im Folgenden zwei Lösungsansätze vorgestellt.

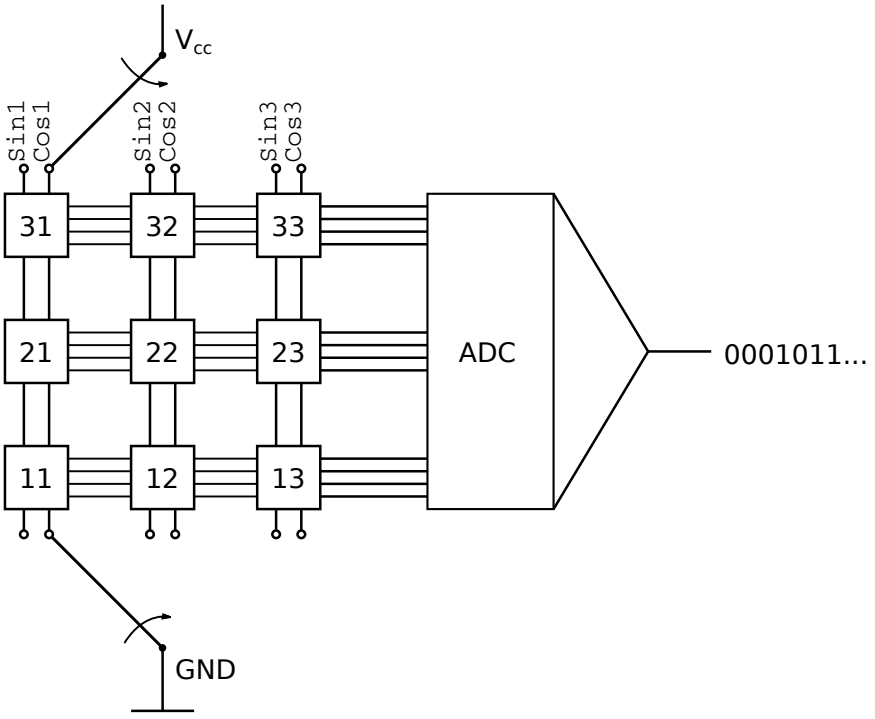


Abbildung 3.1: Zeitmultiplex der Versorgungsspannungen an einem 3x3-Beispiel. Im Bild sind die Cosinus-Brücken der ersten Spalte aktiv.

Analoge Differenzsignalbildung

Jeder Sensor vereint zwei voneinander unabhängige Vollbrücken, definiert als Sinus- und Cosinus-Messbrücke. Somit stehen pro Sensor die Sensorsignale $+Sin, -Sin$ und $+Cos, -Cos$ zur Verfügung. Das Differenzsignal aus diesen beiden Brückenspannungen kann nun digital auf dem Mikrocontroller berechnet werden oder auf analogem Wege in elektronischen Schaltungen. Hierzu wird im einfachsten Fall ein Subtrahierer mittels Operationsverstärker (OPV) aufgebaut [9, S.753]. Als problematisch ist hier der OPV anzusehen, da dieser parasitäre Effekte wie z. B. Offsetfehler mit sich bringt. Durch die Eingangswiderstände der Subtrahierschaltungen kann das Messsignal geschwächt und somit verfälscht werden. In Messsystemen müssen daher sehr genaue OPV gewählt werden. Ein weiteres Problem ist das Schaltungslayout, da selbst unter Verwendung von Quad-OPV (vier OPV in einem Gehäuse) acht Stück eingesetzt werden müssen. Jeder OPV muss außerdem durch äußere Beschaltung als Subtrahierer konfiguriert werden. Durch die analoge Differenzsignalbildung wird die benötigte Anzahl an ADC-Eingängen auf 16 halbiert. Ein weiterer Vorteil ist die Einsparung der Zykluszahl um die Hälfte, da zwei Werte in einem Zyklus verarbeitet werden. Abbildung 3.2 zeigt diesen Sachverhalt am Beispiel einer einzelnen Zeile.

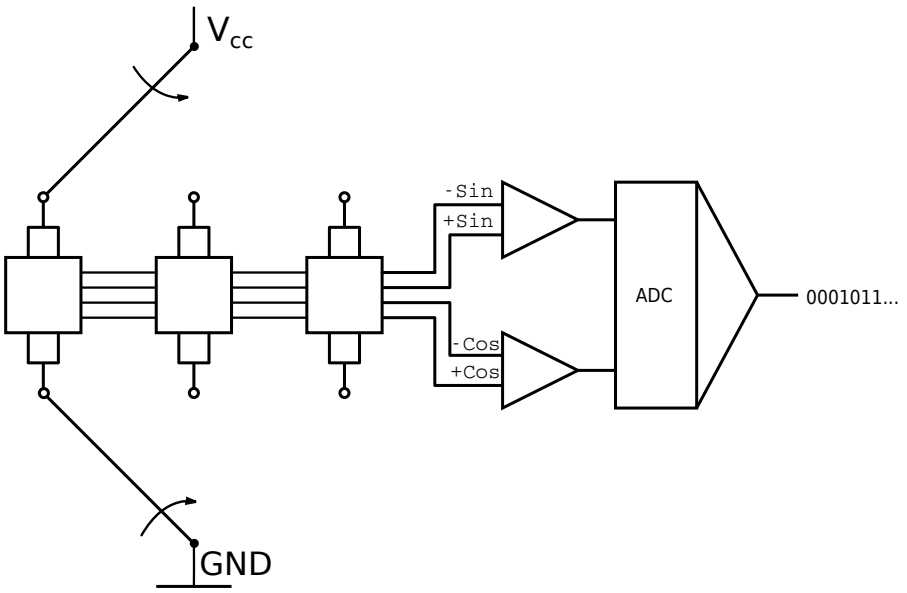


Abbildung 3.2: Zeitmultiplexverfahren mit analogen Subtrahierern. Versorgungspins eines Sensors können zusammengefasst werden. Durch den Einsatz von zwei OPV pro Zeile halbiert sich die Anzahl der ADC-Eingänge

Zusammenfassen von Sensorausgängen

Eine schaltungstechnisch sehr einfache Lösung ist das Zusammenfassen von Sensorausgängen. Dazu werden pro Zeile, welche normalerweise vier Ausgänge besitzt, einfach positive und negative Sensorausgänge parallelgeschaltet. Dadurch sind in jeder Zeile alle negativen und positiven Signale, egal ob Sinus oder Cosinus, zusammengefasst. Damit werden pro Zeile nur noch zwei Ausgänge bzw. ADC-Eingänge benötigt. Somit sinkt die Anzahl an benötigten ADC-Eingängen auf 16. Hierbei muss beachtet werden, dass die Vollbrücken innerhalb eines Sensors durch äußeres Parallelschalten nicht mehr unabhängig voneinander sind. Nun dürfen die Versorgungspins pro einzelnem Sensor nicht mehr parallel geschaltet werden. Der Aufwand an Pins ändert sich dadurch auf 32 GPIO-Pins für die Versorgungsleitungen und auf 16 ADC-Eingänge. Da bei dieser Variante Sinus- und Cosinus-Werte nicht mehr parallel erfasst werden können, verdoppelt sich die benötigte Zeit, der Messwert-Aufnahme. Jede Spalte wird in zwei Zyklen erfasst, einen für die Sinus- und einen zweiten für die Cosinus-Werte. In Abbildung 3.3 wird dies verdeutlicht.

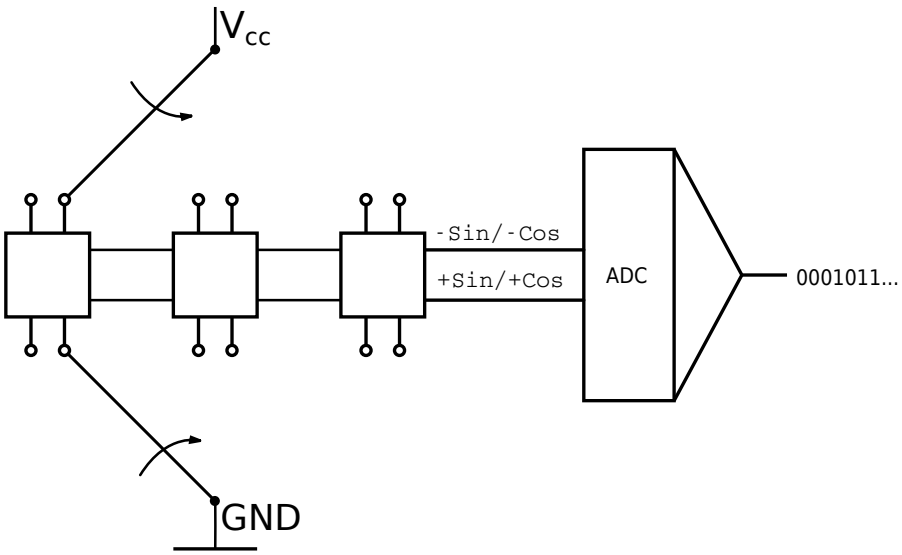


Abbildung 3.3: Zeitmultiplexverfahren durch Zusammenfassen von Sensorausgängen. Die Versorgungspins eines Sensors müssen wieder aufgetrennt werden. Die Zyklenzahl verdoppelt sich.

3.1.2 Externe Analog-Digital-Umsetzer

Eine weitere Variante der Verschaltung ist die Nutzung von externen Analog-Digital-Umsetzern, engl. analog-to-digital converter (ADC). Jeder ADC würde parallel arbeiten und die erfassten Messwerte, z. B. über SPI oder I2C, an den Mikrocontroller übertragen. Durch die entsprechende Wahl eines geeigneten ADC lassen sich hohe Auflösungen

erreichen. Eine hohe Auflösung setzt allerdings genaue, rauscharme Spannungsreferenzen voraus. Ebenso muss die Spannungsversorgung der Sensoren sehr genau sein, damit keine Bits im Rauschen verschwinden. Insgesamt steigt der Schaltungsaufwand durch die zusätzlichen ADC selbst und ebenfalls durch die nötige Spannungsreferenz. Außerdem muss eine genaue Spannungsversorgung der Sensoren gewährleistet sein.

3.1.3 Analogmultiplexer

Um die insgesamt 256 Sensorausgänge (64 Sensoren mit jeweils vier Brückensignalen) des Arrays zu erfassen, können die Signale nacheinander mittels Analogmultiplexer auf die 20 vorhandenen Analog-Pins geleitet werden. Analoge Multiplexer stellen eine elektrisch leitende Verbindung zwischen Ein- und Ausgang her. Somit können analoge Spannungen im Multiplexverfahren geschaltet werden. Um dies zu ermöglichen, werden elektronische Schalter bzw. CMOS-Schalter, sogenannte Transmission-Gates [14], genutzt. Diese bestehen aus parallelgeschalteten p- und n-Kanal-MOSFET und leiten Ströme bidirektional. Somit kann ein analoger Multiplexer auch als Demultiplexer fungieren [9, S. 965-969]. Ein wichtiges Merkmal, welches es zu berücksichtigen gilt, ist der Widerstand zwischen Ein- und Ausgang des Transmission-Gates. Er setzt sich aus Überlagerung der beiden R_{DS}-Kennlinien der MOSFET zusammen. In Datenblättern wird häufig der maximale R_{DS} angegeben. Analoge Multiplexer werden in verschiedenen Ausführungen angeboten. Häufig werden analoge Multiplexer mit acht Kanälen angeboten. Es gibt jedoch auch welche mit 16 Kanälen. Diese bieten sich für die hier vorgesehene Nutzung an. Ein verbreiteter Typ, welcher von verschiedenen Herstellern in CMOS-Technologie angeboten wird, ist der 74HC4067. Abbildung 3.4 zeigt Beispielhaft wie 16 Sensorsignale mit Hilfe eines analogen Multiplexers auf einen ADC-Eingang geschaltet werden können.

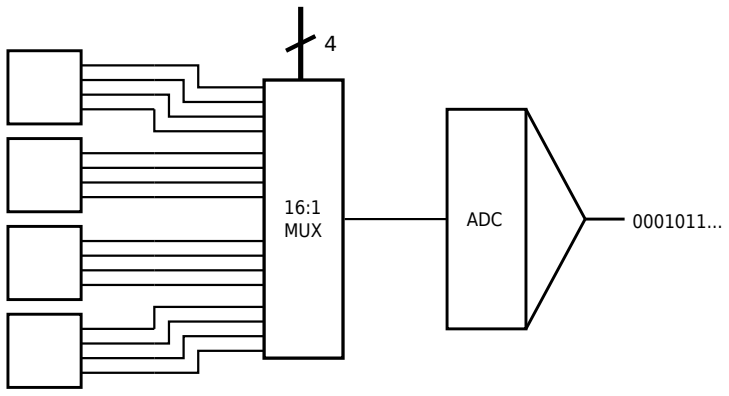


Abbildung 3.4: Mit einem 16:1 Analogmultiplexer lassen sich vier Sensoren mit einem ADC-Eingang erfassen. Die Spannungsversorgung wird nicht geschaltet und ist fest verbunden. Die Wahl des Kanals erfolgt über vier Adresspins.

3.1.4 Vergleich und Bewertung

Im Folgenden werden die verschiedenen Vor- und Nachteile aufgezeigt. Tabelle 3.1 gibt eine kompakte Auskunft der verschiedenen Varianten. Anschließend wird eine Stellungnahme für die gewählte Variante gegeben.

Tabelle 3.1: Vergleich der Varianten

Variante	Vorteile	Nachteile
Zeitmultiplex-verfahren	- einfache Schaltung	- Versorgungsspannung über GPIO kritisch
ext. ADC	- beliebig parallelisierbar - hohe Auflösung möglich	- hoher Schaltungsaufwand - genaue Spannungsversorgung
Analog-multiplexer	- Verfügbarkeit - Schaltungsaufwand rel. gering	- Widerstand zw. Ein- und Ausgang - Transistorrauschen

Da das Zeitmultiplexverfahren die Sensoren mit Spannungen aus den GPIO-Pins versorgt, ist dieses Verfahren zum genauen Messen der Magnetfelder weniger geeignet. Durch externe ADC lassen sich zwar hohe Auflösungen erreichen, jedoch wird auch eine Spannungsreferenz nötig. Die Güte der Spannungsreferenz bestimmt letztendlich auch die Genauigkeit der ADC. Das Erfassen der ADC-Werte lässt sich durch Wahl von entsprechend vielen ADC parallelisieren, jedoch werden die Daten seriell an den Mikrocontroller übertragen. Insgesamt ergibt sich bei dieser Variante ein hoher Schaltungsaufwand.

In dieser Bachelorarbeit wird die Variante mit Analogmultiplexern umgesetzt. Der Schaltungsaufwand ist moderat und stellt einen Kompromiss zwischen dem Zeitmultiplexverfahren und der Variante mit externen ADC dar. Die Sensoren werden von einer externen Spannungsquelle gespeist, wodurch das Problem mit den Logikpegeln, welches beim Zeitmultiplexverfahren besteht, nicht auftritt. Ein Nachteil ist der Widerstand zwischen Ein- und Ausgangssignal des Multiplexers, da jedoch keine hohen Ströme fließen und prinzipiell nur Spannungen gemessen werden, spielt dieser eine untergeordnete Rolle. Das relativ geringe Transistorrauschen wird als unproblematisch eingeschätzt.

3.2 Platinenentwurf

In diesem Abschnitt geht es um den praktischen Hardwareentwurf mittels des EDA-Programms Eagle. Dabei wird zunächst auf das Erstellen eigener Bauteile in Eagle eingegangen. Anschließend wird der eigentliche Entwurf der Platine angesprochen. Abschließend wird ein kurzer Bezug zur Fertigung und Bestückung der Leiterplatte genommen. Als Multiplexer wird auf einen 74HC4067 zurückgegriffen.

3.2.1 Erstellen eigener Bibliotheken

Bevor mit dem Schaltungsentwurf im Layoutprogramm Eagle begonnen werden kann, muss ein Bauteil für den gewählten TMR-Sensor erstellt werden. Dabei wird zunächst ein Symbol für den Schaltplan angelegt und im zweiten Schritt ein Package erstellt, in diesem Fall TSSOP8. Abschließend wird aus diesen beiden Teilen ein fertiges Device, wobei die Pins des Symbols den Footprints des Packages zugeordnet werden. Alle anderen benötigten Bauteile finden sich in den mitgelieferten Bibliotheken von Eagle, sodass auf diese zurückgegriffen werden kann. Eine Ausnahme stellt der Multiplexer dar. Dieser ist als Bauteil an sich vorhanden, jedoch nicht im verwendeten Package. Also wird hierfür ebenfalls ein Device erstellt.

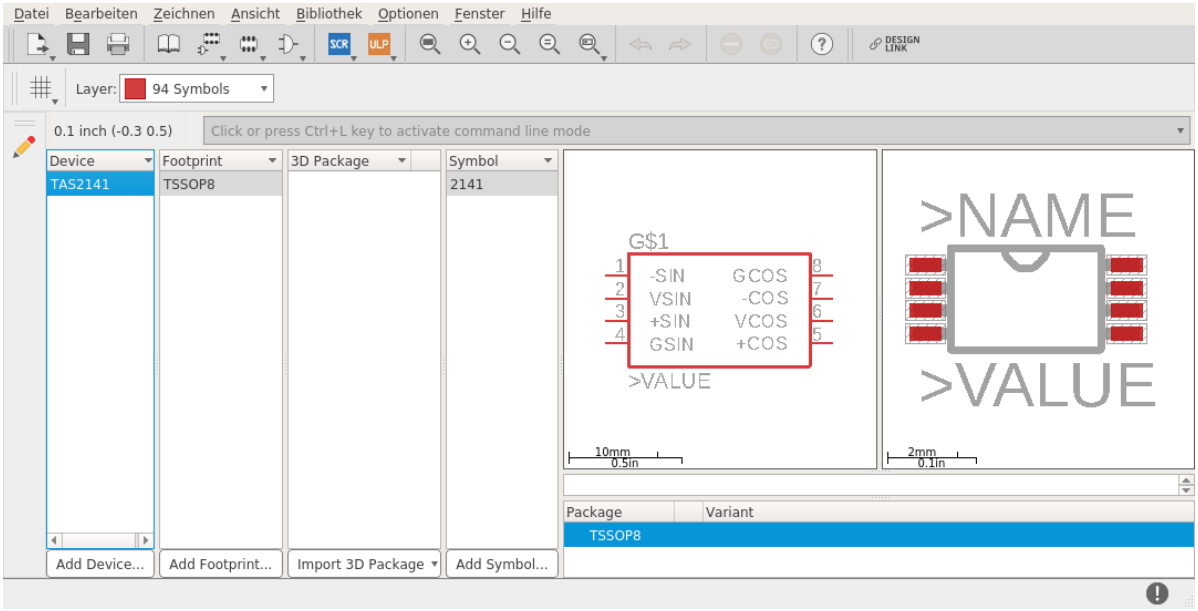
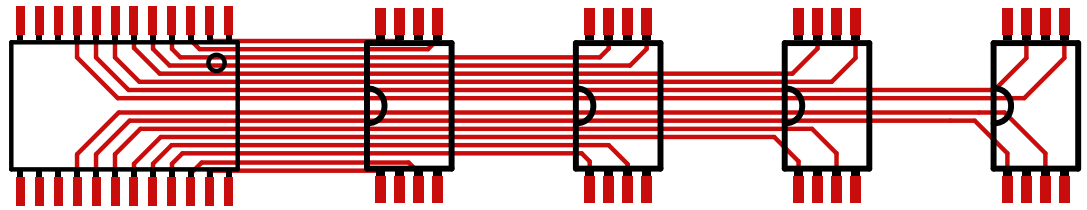


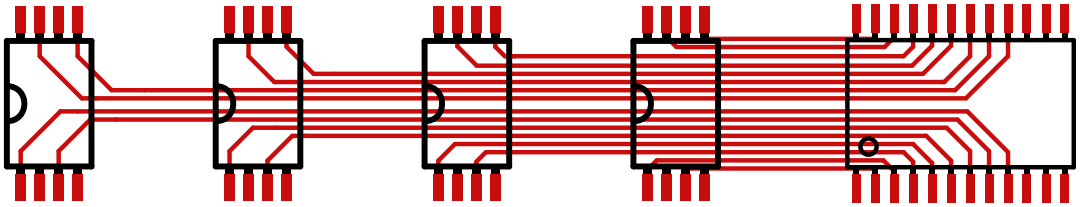
Abbildung 3.5: Fenster des Inhaltsverzeichnisses der erstellten Bibliothek in Eagle. Zu sehen ist der Devicename sowie das Footprint und Symbol.

3.2.2 Schaltplan- und Layoutentwurf

Nachdem die neuen Devices erfolgreich erstellt worden sind, wird der Schaltplan für das Sensor-Array entworfen. Um bei der Erstellung möglichst effizient vorzugehen, werden Module zusammengefasst, welche danach einfach zu kopieren sind. Eine Vorüberlegung hierbei ist, dass insgesamt 64 Sensoren von 16 Multiplexern erfasst werden müssen. Daraus folgt, dass ein Multiplexer vier Sensoren zusammenfasst. Begonnen wurde damit, die Bauteile auf dem Schaltplan zu platzieren. Um der Forderung nach einer möglichst kompakten Anordnung nachzugehen, wurde nun in den Layout-Editor gewechselt und erst



(a) Baugruppe links.



(b) Baugruppe rechts.

Abbildung 3.6: Baugruppen der linken und rechten Hälfte des Sensor-Arrays. Zusammengesetzt ergeben beide Baugruppen eine Zeile des Sensor-Arrays. Alle Verbindungen lassen sich in einer Lage unter den Footprints verdrahten.

einmal nur die Sensoren in verschiedenen Anordnungen platziert, um einen Überblick über die Platzverhältnisse zu erlangen. Um die Verdrahtung der Leiterbahnen vorzunehmen, wird wieder in den Schaltplaneditor gewechselt und die schaltplanmäßige Verdrahtung vorgenommen. Allgemein empfiehlt es sich, zwischen den Design-Schritten immer wieder einen ERC-Check (Electrical-Rule-Check) durchzuführen. Dabei untersucht das Programm den Schaltplan auf Fehler und zeigt diese an.

Schließlich ergab sich eine Baugruppe in einer 1x4-Anordnung der Sensoren mit zugehörigem Multiplexer (vgl. Abbildung 3.6(a)). Die Leiterbahnen zwischen Sensor und Multiplexer verlaufen unter den Sensorfootprints. Als Leiterbahnbreite muss dann auf eine Breite von 5 mil zurückgegriffen werden, um insgesamt 16 Leiterbahnen unter dem Sensor führen zu können. Wie im Schaltplaneditor sollte auch hier immer wieder auf Fehler überprüft werden, wobei das Äquivalent zum ERC-Check hier DRC-Check (Design-Rule-Check) heißt. Hilfreich ist dieser insbesondere um zu überprüfen, ob alle benötigten Mindestabstände eingehalten wurden und ob noch unverbundene Leitungen, sog. Airwires, vorhanden sind.

Zusätzlich wird noch eine zweite 1x4-Baugruppe erstellt, welche sich nur durch einen um 180° gedrehten Multiplexer unterscheidet (vgl. Abbildung 3.6(b)). Dadurch wird erreicht, dass alle Eingänge des Multiplexers gut zu verdrahten sind, gleichzeitig jedoch auch die Adresseingänge von der Seite noch gut erreicht werden können. Für die spätere Software bedeutet dies, dass das höchstwertigste Adressbit invertiert werden muss, um auf der linken sowie rechten Arrayhälfte die gleiche Reihenfolge der Sensorausgänge zu erhalten. Dies ergibt sich aus dem Pinout des Multiplexers.

Da beide Hälften erstellt sind, können diese in den Schaltplan und in das Layout kopiert werden. Beim Kopieren sollten unbedingt die Netznamen manuell überprüft und gegebenenfalls korrigiert werden. Ein besonderes Augenmerk ist dabei auf die Spannungsversorgung und das Ground-Netz zu richten. Das Grundgerüst des Arrays ist somit erstellt.

Anschlussführung zwischen Platine und Sensor-Array

Für die Verbindung zwischen dem Array und dem Mikrocontroller-Board werden nun im Datenblatt [11] geeignete Analog-, GPIO- und GND-Pins gewählt. Optional kann der ADC mit einer externen Spannungsreferenz arbeiten. Um die Möglichkeit nutzen zu können, wird ebenfalls der V_{ref}-Pin für den ADC verdrahtet. Danach wird eine Pinleiste im Schaltplan platziert und mit den entsprechenden Bauteilen verbunden. Nun wird wieder in das Board gewechselt und die Pinleiste platziert und verdrahtet. In Abbildung 3.7 ist die schematische Pinbelegung dargestellt. Abbildung 3.8 zeigt den entsprechenden Ausschnitt des Platinenlayouts.

Um auf mögliche Störungen im Betrieb Rücksicht nehmen zu können, sind auf dem Signalweg zwischen Multiplexer-Ausgängen und ADC-Eingängen einfache RC-Tiefpassfilter vorgesehen. Im Layout werden diese sinnvollerweise nahe der Pinleiste, also dicht an den ADC-Eingängen des Mikrocontroller-Boards platziert. Dem V_{ref}-Pin ist ebenfalls ein Tiefpassfilter vorgeschaltet. Die Tiefpassfilter sind jedoch nur als Bestückung vorgesehen. Die Kondensatoren werden weggelassen und als Widerstände dienen Null-Ohm-Widerstände. Die Verdrahtung der Adresspins erfolgt in der unteren Platinenlage bis unter die Sensorfootprints. Mit Vias wird dann in die obere Lage gewechselt und die Adresspads verbunden.

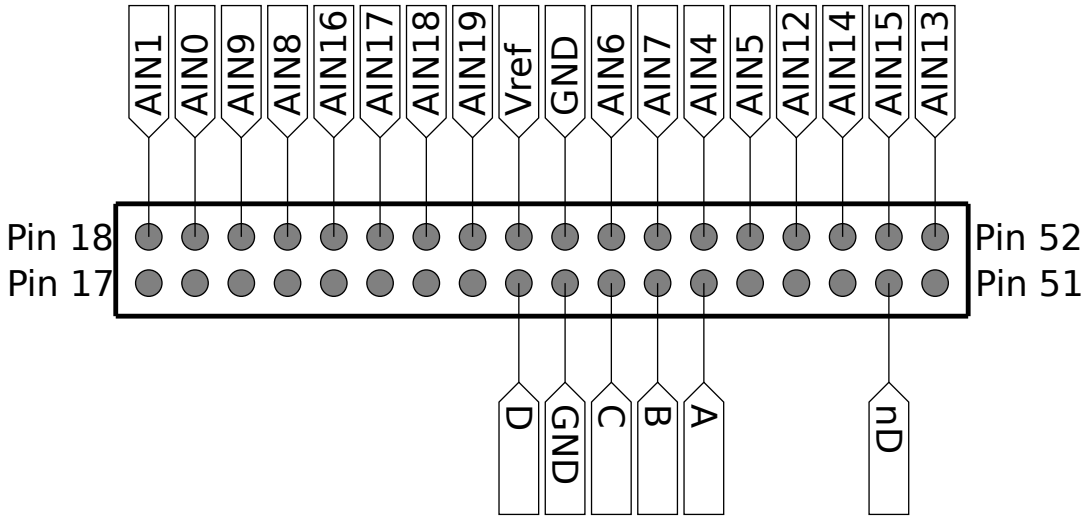


Abbildung 3.7: Pinzuordnung zwischen Sensor-Array und Mikroconroller-Board.

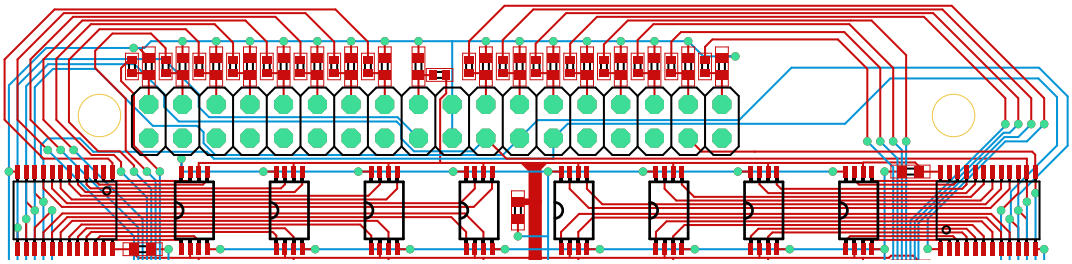


Abbildung 3.8: Anschlussführung zwischen Sensor-Array und Mikrocontroller-Board. Oberhalb der Pins finden sich die Tiefpassfilter. Mittig wird die Spannungsversorgung an den V_{ref}-Pin hochgeführt, das Ground-Netz wird ebenfalls verbunden.

Spannungsversorgung

Die Speisung der Sensoren erfolgt durch eine externe Spannungsversorgung. Die Spannung kann über zwei Möglichkeiten angelegt werden. Zum einen sind Schraubklemmen und zum anderen eine Hohlbuchse vorgesehen. Beide werden auf der Unterseite der Platine platziert, damit auf der Oberseite keine hohen Bauteile vorhanden sind. Diese würden sich beim späteren Anfahren der Sensoren im Bereich dieser Bauteile gegebenenfalls störend bemerkbar machen. Es wird von einer stabilen, genauen Spannungsversorgung ausgegangen. Trotzdem werden Maßnahmen zur Stabilität der Spannungsversorgung und zur Störunterdrückung getroffen. Für die Spannungsstabilisierung wurde nahe der Einspeisung ein Elektrolytkondensator der Größe 1000 µF parallelgeschaltet. Für die Unterdrückung hochfrequenter Störungen wird diesem noch ein 100 nF Keramikkondensator parallelgeschaltet. Der Hauptstrang der positiven Spannungsversorgung wird

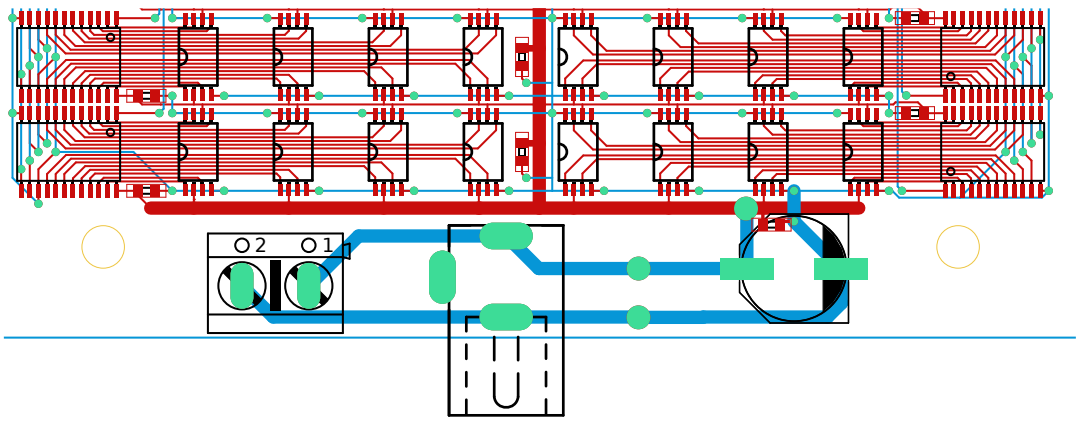


Abbildung 3.9: Spannungseinspeisung über Hohlbuchse und Schraubklemmen. Elektrolytkondensator als Puffer, parallel dazu ein Keramikkondensator. Die Leiterbahnbreite wurde vergrößert.

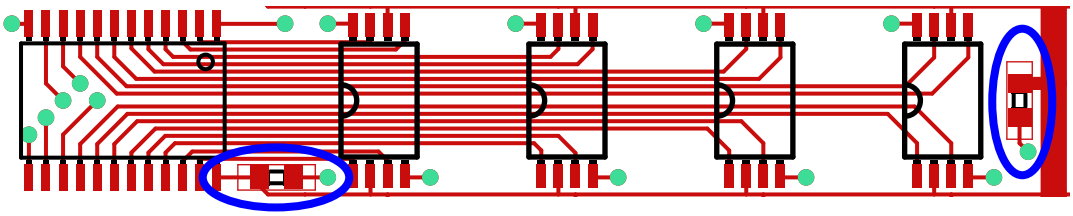


Abbildung 3.10: Keramikkondensatoren (blau eingekreist) links am Versorgungspin des Multiplexers und rechts am Hauptstrang der Spannungseinspeisung.

mittig von unten nach oben im Sensor-Array geführt. Von diesem Hauptstrang aus werden die Sensoren und Multiplexer in beide Richtungen verdrahtet. Abbildung 3.9 zeigt die Spannungseinspeisung. In jeder Zeile des Sensor-Arrays wird mittig noch ein weiterer 100 nF Keramikkondensator platziert, ebenso an jedem Spannungsversorgungspin der Multiplexer (vgl. Abbildung 3.10). Diese sollen ebenfalls der Störunterdrückung dienen [6]. Als besonders störanfällig werden die Multiplexer eingeschätzt, da bei der Wahl der Kanäle durch die GPIOs steile Flanken entstehen, welche auf den Rest der Schaltung übersprechen könnten. Nachdem alles verdrahtet worden ist, wird auf der unteren Seite der Platine ein Polygon über die gesamte Fläche gelegt und damit eine GND-Fläche erstellt. Dieses dient ebenfalls der Vermeidung von Störungen. Abbildung 3.11 zeigt das Fertiggestellte Layout.

3.2.3 Fertigung der Platine

Die Fertigung der Platine erfolgt extern beim Leiterplattenhersteller Beta LAYOUT GmbH. Dazu werden die erstellten Eagle-Dateien und die Gerber-Dateien übergeben. Während des Fertigungsprozesses werden fortlaufend Bilder der Platine zur Verfügung gestellt. Diese sind im Anhang zu finden und geben einen Überblick über den Verlauf der Fertigung. Nach Fertigstellung der Platine wird diese, zusammen mit den Bauteilen dem Bestücker übergeben. Dieser lötet die Bauteile auf und fertigt somit die komplette Platine. Abbildung 3.12 zeigt die fertige Platine.

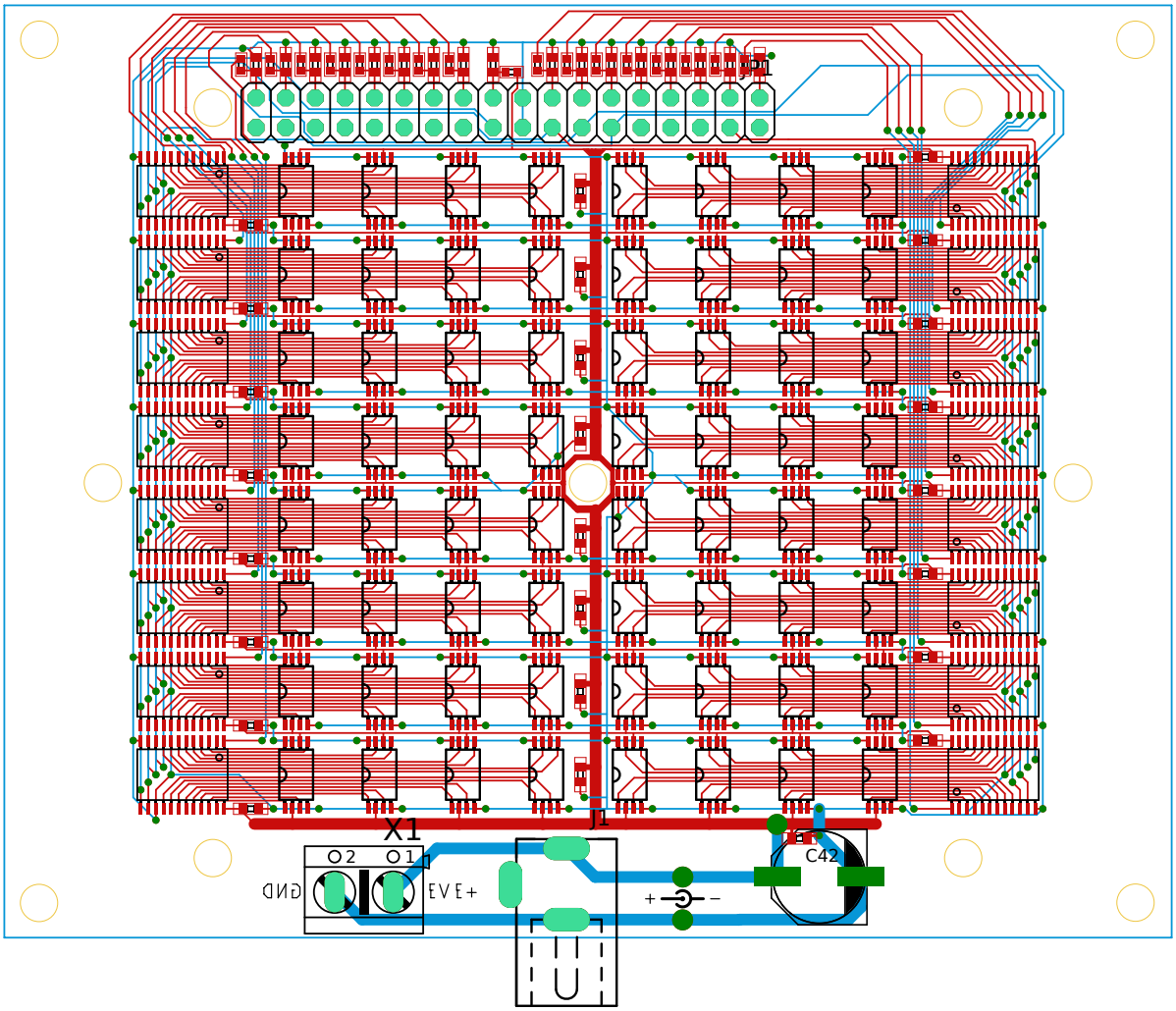


Abbildung 3.11: Finale Version des Leiterplattenlayouts. Massefläche der unteren Lage wird aus Gründen der Übersicht nicht dargestellt.

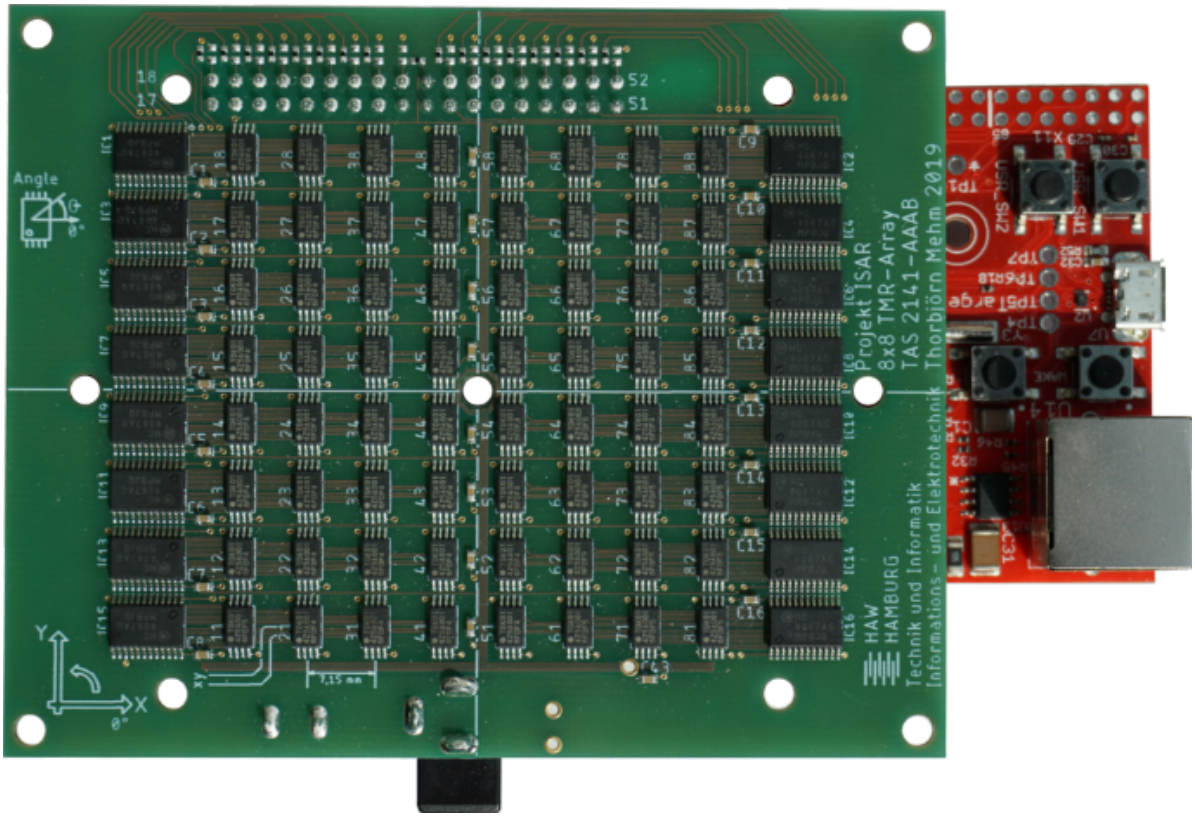


Abbildung 3.12: Fertig bestückte Platine bereit für Inbetriebnahme und Funktionstest.

4 Inbetriebnahme und Funktionstest

Nachdem die Platine fertig bestückt worden ist und die nötigen Erweiterungen des Mikrocontroller-Boards abgeschlossen sind, folgen in diesem Kapitel die Inbetriebnahme und der Funktionstest. Dabei werden das Vorgehen sowie mögliche Fehlerfälle angesprochen. Außerdem wird eine Software erstellt, welche das Auslesen des Arrays übernimmt. Des Weiteren werden die erfassten Werte von der Software über UART ausgegeben.

4.1 Überprüfung und Inbetriebnahme

Zunächst wird das Sensor-Array auf optische Auffälligkeiten überprüft. Für das Überprüfen der ICs steht ein Mikroskop zur Verfügung mit welchem es möglich ist, die Lötverbindung zwischen den feinen Pins der ICs und deren Footprints zu überprüfen. Auffällige Lötstellen sind mit Hilfe eines Vielfachmessinstruments auf Durchgang zu testen und gegebenenfalls nachzubessern. Beide Platinen wurden extern gefertigt und weisen ein sauberes Lötbild auf. Nun kann zur elektrischen Inbetriebnahme übergegangen werden. Abbildung 4.1 zeigt die komplette Platine mit einer ausgewählten Detailaufnahme einiger Lötstellen.

Beide Platinen werden mit einer Spannung von $U_B = 3,3\ V$ versorgt. Dazu wird ein Labornetzteil mit einstellbarer Spannung und Strombegrenzung verwendet. Laut Datenblatt [4] hat jeder Sensor einen minimalen Brückenwiderstand von $R_{min} = 4\ k\Omega$. Da

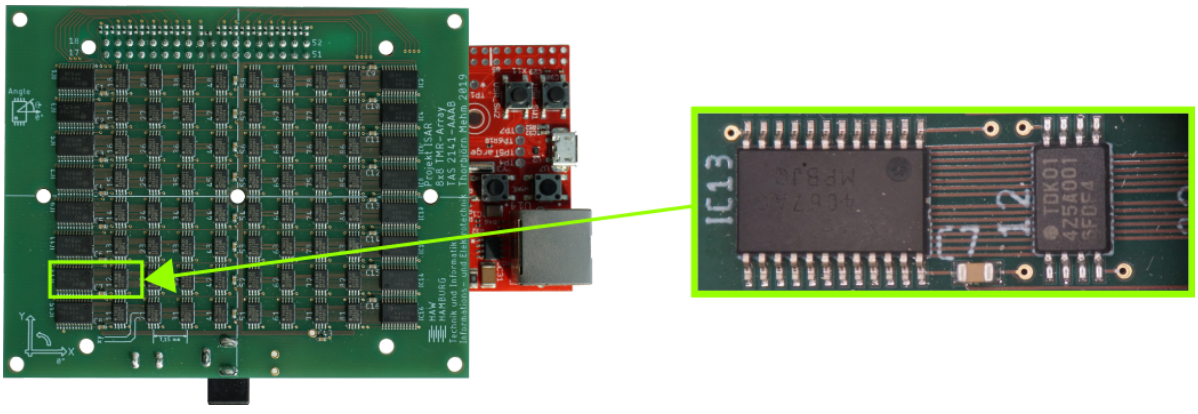


Abbildung 4.1: Vergrößerung eines Ausschnitts der Platine. Durch die automatisierte Bestückung weisen die Lötstellen ein sauberes Lötbild auf.

jeder Sensor zwei Vollbrücken beinhaltet, errechnet sich die Gesamtzahl der Vollbrücken zu $n = 128$. Der Gesamtwiderstand ergibt sich damit nach folgender Formel:

$$R_{ges} = \frac{R_{min}}{n} = \frac{4\ k\Omega}{128} = 31,25\ \Omega \tag{4.1}$$

Daraus lässt sich mit Hilfe des ohmschen Gesetzes der maximale Strom errechnen:

$$I_{ges} = \frac{U_B}{R_{ges}} = \frac{3,3\ V}{31,25\ \Omega} = 105,6\ mA \tag{4.2}$$

Anhand der maximalen Stromaufnahme der Sensoren kann nun die Strombegrenzung auf etwa diesen Wert eingestellt werden. Die Stromaufnahme der Multiplexer wird, da diese in CMOS-Technologie gefertigt sind und somit eine sehr geringe Stromaufnahme aufweisen, vernachlässigt. Mit Hilfe der Formeln lässt sich durch Einsetzten des maximalen Brückenwiderstandes die minimal zu erwartende Stromaufnahme errechnen. Bei einer gemessenen Spannung von $3,30\ V$ wird die Stromaufnahme beider Platinen gemessen. Die Ergebnisse sind in Tabelle 4.1 zusammengestellt. Die gemessenen Werte liegen mit $90,4\ mA$ und $87,6\ mA$ innerhalb der errechneten Grenzen.

Tabelle 4.1: Berechnete und gemessene Stromaufnahme der Sensoren des Sensor-Arrays. Die Werte für R_{min} und R_{max} sind dem Datenblatt des Sensors entnommen.

	Gesamtwiderstand [Ω]	Gesamtstrom [mA]
$R_{min} = 4\ k\Omega$	31,25	105,6
$R_{max} = 6\ k\Omega$	46,88	70,4
Platine I	36,50	90,4
Platine II	37,67	87,6

4.2 Software auf dem Mikrocontroller

Um das gesamte System testen zu können, wird eine entsprechende Software erstellt. Insgesamt gibt es 16 verschiedene Zustände, welche die Aufgabe haben, die einzelnen Kanäle der Multiplexer zu wählen. In jedem Zustand werden die ADC-Eingänge ausgelesen und in ein Speicherarray an die zugehörige Stelle geschrieben. Am Ende der 16 Zyklen werden die Speicherarrays über UART ausgegeben.

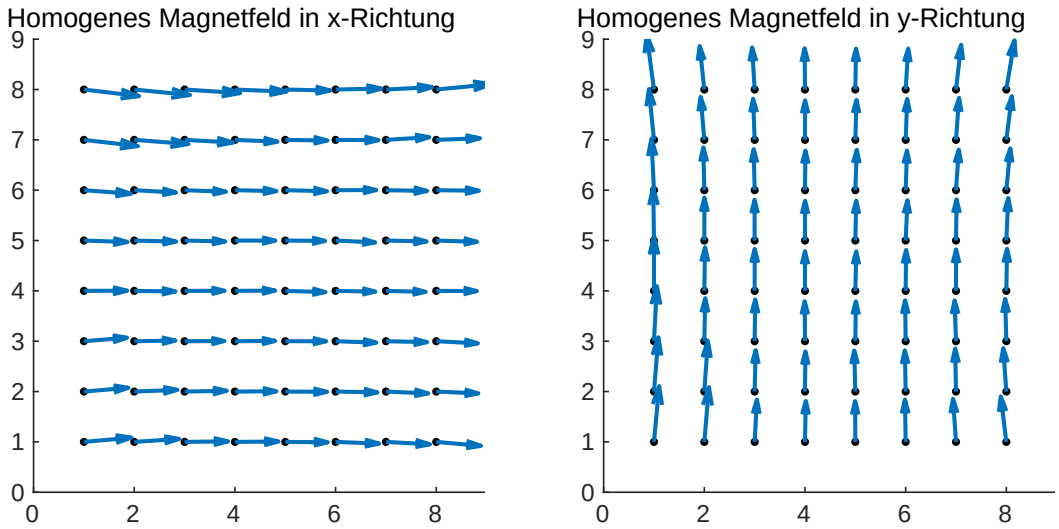
Da die ADC über eine Auflösung von 12 Bit verfügen, gibt es über den gesamten Erfassungsbereich 4096 verschiedene Stufen. Sofern das Sensor-Array nicht mit einem externen Magnetfeld beaufschlagt wird, sollten die Sensorbrücken bis auf Toleranzen abgeglichen sein. Somit sollte sich jeder ADC-Wert um 2048, entsprechend $3,3\ V / 2 = 1,65\ V$ bewegen. Wenn entsprechende Werte erzielt werden, wird fortgefahren. Wenn die Werte

fehlerhaft unplausibel scheinen, werden die Analog-Eingänge mit dem Oszilloskop beobachtet. Sofern auch hier stark abweichende Werte von $1,65\ V$ zu beobachten sind, sollte im Abschnitt Fehlerfälle fortgefahren werden. Sind jedoch die analog gemessenen Spannungen plausibel, so muss der Fehler in der Software liegen. Hierbei liegt ein besonderes Augenmerk auf der Zuordnung der Pins.

4.3 Visualisierung der Messwerte

Da die erfassten Werte der Controller-Software schwer überschaubar sind, erfolgt eine Visualisierung der Messwerte. Dazu wird ein Matlab-Skript erstellt, welches über UART die Werte entgegennimmt und nach erfolgreichem Empfang darstellt. Die Darstellung erfolgt als Quiver-Plot. Dabei handelt es sich um eine Vektordarstellung des Momentanwertes eines jeden Sensors. Durch die anschauliche Darstellung können eventuelle Fehler in der Zuordnung bzw. Ausrichtung der Sensoren leicht erkannt werden. Mittels verschiedener Magnete kann die Funktion erprobt werden. Im ersten Schritt kann beispielsweise ein homogenes Magnetfeld mittels Halbach-Ring angelegt werden, um zu prüfen, ob die Messwerte ebenfalls homogen sind. Dies zeigt Abbildung 4.2.

In einem zweiten Schritt kann unter Zuhilfenahme eines kleinen Magneten die Zuordnung der Sensoren zu den visualisierten Messpunkten getestet werden, indem die einzelnen Sensoren mit dem Magneten beaufschlagt und der Quiver-Plot beobachtet wird. Abbildung 4.3 zeigt beispielhaft die Positionierung im Zentrum der Sensoren (55)-(56) und (65)-(66). Auf diese Weise kann die Zuordnung aller Sensoren mit einem kleinen Stabmagneten überprüft und verifiziert werden.



(a) Magnetfeld in x-Richtung.

(b) Magnetfeld in y-Richtung.

Abbildung 4.2: Visualisierung eines relativ homogenen Magnetfeldes mit Hilfe eines Halbach-Rings. Magnetfeld ist insbesondere in den Ecken nicht mehr homogen.

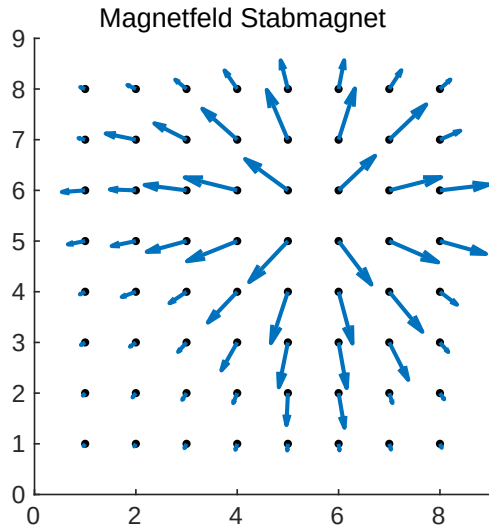


Abbildung 4.3: Inhomogenes Magnetfeld durch Stabmagnet. Nordpol zeigt auf das Array.

5 Controller-Software

Dieses Kapitel beschreibt die Software, welche auf dem Mikrocontroller umgesetzt wird. Die Aufgabe besteht darin, das komplette Sensor-Array auszulesen und die aufgenommenen Daten zur Weiterverarbeitung seriell über UART auszugeben. Für die Programmierung wird auf die Funktionen der „TivaWare Peripheral Driver Library“ von Texas Instruments zurückgegriffen [13].

5.1 Konfigurationen

Zunächst werden die Konfigurationen der verschiedenen Pins erstellt. Benötigt wird zum einen die Kommunikation über UART, hierbei wird eine Baudrate von 115200 gewählt. Zusätzlich werden die GPIO-Pins, welche als Ausgang dienen, für die Wahl der Adressen der Multiplexer konfiguriert. Anschließend werden noch die Einstellungen des ADC vorgenommen. Tabelle 5.1 gibt Aufschluss über die benötigten Pins.

Tabelle 5.1: Konfiguration der verwendeten Pins des Mikrocontrollers.

Modul	Pins	Konfiguration	Kommentar
UART	PA0	Rx	Empfangen
	PA1	Tx	Senden
GPIO	PM0...3	Ausgang	Adresse Multiplexer
	PL3	Ausgang	PM3 invertiert
ADC	AIN1...0	ADC-Eingang	
	AIN4...9	ADC-Eingang	
	AIN12...19	ADC-Eingang	

5.1.1 Konfiguration des Analog-Digital-Converters

Der verwendete Mikrocontroller stellt zwei ADC-Module zur Verfügung, welche sich die insgesamt 20 ADC-Eingangskanäle teilen. Es sind vier *sample sequencer* vorhanden, die es ermöglichen, mehrere Eingangskanäle pro einer gestarteten Umsetzung zu erfassen. Getriggert werden die ADC intern durch die Software. Weitere Parameter und Konfigurationsmöglichkeiten sind dem Datenblatt zu entnehmen [12]. Folgende Tabelle zeigt die Zuweisung der ADC-Eingänge zu den gewählten ADC-Modulen und ihren *sample sequencern*. Zusätzlich werden für die bessere Lesbarkeit des Quelltextes Makros der Analog-Pins definiert.

Tabelle 5.2: Zuordnung der ADC-Kanäle

ADC-Modul	sample sequencer	step	Kanal	Makro
ADC0	sample sequencer 0	0	CH1	ROW_1_L
		1	CH0	ROW_2_L
		2	CH9	ROW_3_L
		3	CH8	ROW_4_L
		4	CH16	ROW_5_L
		5	CH17	ROW_6_L
		6	CH18	ROW_7_L
		7	CH19	ROW_8_L
ADC1	sample sequencer 1	0	CH13	ROW_1_R
		1	CH15	ROW_2_R
		2	CH14	ROW_3_R
		3	CH12	ROW_4_R
	sample sequencer 2	0	CH5	ROW_5_R
		1	CH4	ROW_6_R
		2	CH7	ROW_7_R
		3	CH6	ROW_8_R

Die Namen der Makros ergeben sich aus der Arraystruktur. Das Array ist zeilenweise sortiert. Jedem Multiplexer sind die vier Sensoren links bzw. rechts davon zugeordnet (vgl. Abbildung 5.1). Jeder Analogeingang wird von einem Multiplexer gespeist. Dadurch ergibt sich folgendes Namensschema:

$$ROW\_\{Zeilennummer\}\_\{L(inks)/R(echts)\}$$

5.2 Erfassung der ADC-Werte

Durch den Hardwareentwurf ist die Platine in eine linke und eine rechte Hälfte strukturiert. Jedem Multiplexer ist ein ADC-Kanal zugeordnet und jeder Multiplexer erfasst vier Sensoren mit seinen jeweiligen Signalen $+Sin, -Sin$ und $+Cos, -Cos$. Um das gesamte Array komplett auszulesen, sind 16 Zyklen notwendig. In jedem Zyklus wird zunächst die binär kodierte Adresse an alle Multiplexer angelegt. Für die linke Hälfte wird die Zyklusnummer direkt als Adresse übernommen. Durch Inversion des höchstwertigen Bits für die Multiplexer der rechten Hälfte ergibt sich die äquivalente Reihenfolge der Sensorsignale. Dies ist die Folge der Hardwareentwicklung, wobei die Multiplexer der rechten Seite um 180° gedreht worden sind. Durch die Wahl der Multiplexer-Adressen ist somit genau bekannt, welcher zugehörige Sensor und welches Signal davon gewählt ist. Tabelle 5.3 gibt Auskunft über die Sensorsignale bei entsprechender Adresswahl.

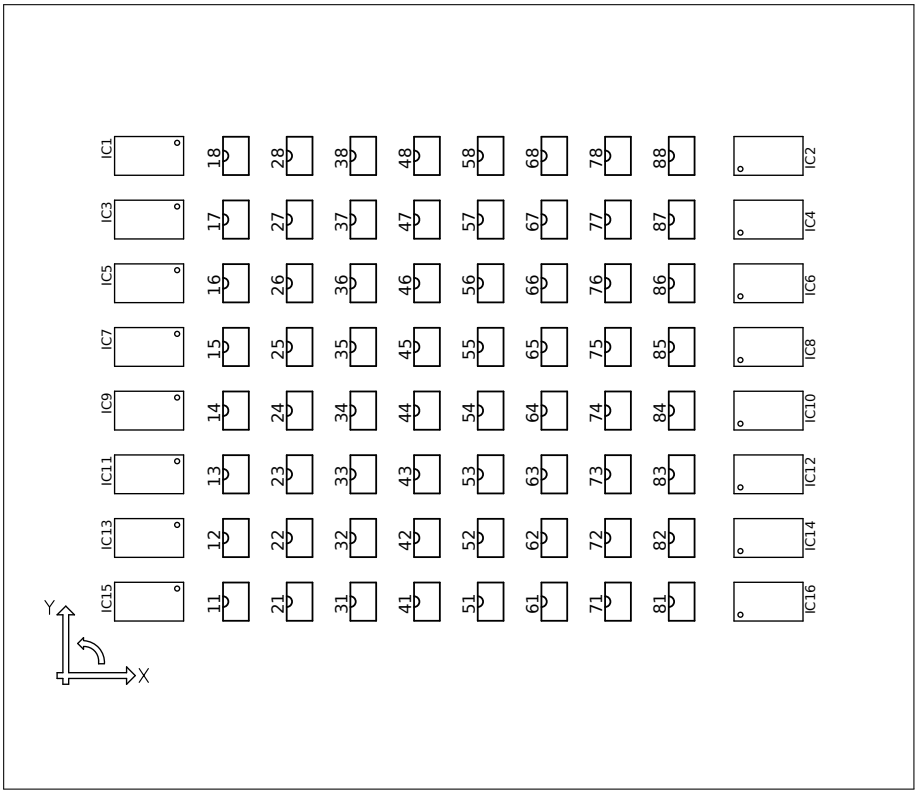


Abbildung 5.1: Platzierung der Sensoren und Multiplexer auf dem Board. Sensornamen folgen der Platzierung im ersten Quadranten eines kartesischen Koordinatensystems

Um das gesamte Array auszulesen, laufen nacheinander zwei Zählschleifen mit jeweils acht Durchläufen. Die erste liest alle Cosinus-Werte aus, während die zweite alle Sinus-Werte liest. Aus dem Laufindex wird direkt die Adresse für die Multiplexer gebildet und das MSB wird für die Multiplexer der rechten Hälfte invertiert. In jedem Schleifendurchlauf werden alle ADC-Kanäle gelesen und in globale Speicherarrays gesichert. Für die richtige Positionierung der aktuellen Werte wird wieder der Laufindex hinzugezogen.

Hinweise zu den Variablen

Die vom ADC erfassten Werte werden in *uint32_t*-Variablen gespeichert. Da der ADC eine Auflösung von 12 Bit aufweist, ist eine Breite von 32 Bit nicht notwendig. Da es sich außerdem um *unsigned*-Variablen handelt und aus den ausgelesenen Werten noch differentielle Werte gebildet werden, kann es zu Unterlauf-Problemen kommen. Deshalb erfolgt eine Typumwandlung in globale *int16_t*-Arrays. Durch die Verwendung von *int16_t*-Arrays ergeben sich ebenfalls Vorteile für die spätere Übertragung per UART. Eine Arrayzelle kann somit auf zwei *char*-Zeichen aufgeteilt werden. Dies führt zu einer schnelleren Übertragung über die UART durch weniger Redundanz.

Tabelle 5.3: Zuordnung Adresse und Sensorsignal

	links		rechts	
Zyklus	Adresse	Signal	Adresse	Signal
1	0x0	$+Cos4y$	0x8	$+Cos8y$
2	0x1	$-Cos4y$	0x9	$-Cos8y$
3	0x2	$+Cos3y$	0xA	$+Cos7y$
4	0x3	$-Cos3y$	0xB	$-Cos7y$
5	0x4	$+Cos2y$	0xC	$+Cos6y$
6	0x5	$-Cos2y$	0xD	$-Cos6y$
7	0x6	$+Cos1y$	0xE	$+Cos5y$
8	0x7	$-Cos1y$	0xF	$-Cos5y$
9	0x8	$-Sin1y$	0x0	$-Sin5y$
10	0x9	$+Sin1y$	0x1	$+Sin5y$
11	0xA	$-Sin2y$	0x2	$-Sin6y$
12	0xB	$+Sin2y$	0x3	$+Sin6y$
13	0xC	$-Sin3y$	0x4	$-Sin7y$
14	0xD	$+Sin3y$	0x5	$+Sin7y$
15	0xE	$-Sin4y$	0x6	$-Sin8y$
16	0xF	$+Sin4y$	0x7	$+Sin8y$

5.3 Berechnungen

Nachdem alle Rohdaten eingelesen sind folgen nun Berechnungen zur Differenzsignalbildung sowie die des Offsets und allgemein die Separation der Signale $+Sin, -Sin, +Cos$ und $-Cos$ in eigene Speicherarrays. Die Ergebnisse hieraus werden alle in *int16_t*-Arrays der Größe 8x8 gespeichert.

5.4 Kommunikation

Die Kommunikation erfolgt, wie bereits erwähnt, über UART. Das Programm wartet immer auf genau ein Zeichen und reagiert entsprechend darauf. Tabelle 5.4 gibt Auskunft über die implementierten Befehle. Um die Kommunikation entsprechend schlank zu gestalten, erfolgt sie über einzelne Zeichen. Dafür werden die Funktionen *UARTCharPut* und *UARTCharGet* aus der Tivaware Bibliothek von Texas Instruments herangezogen. Zunächst wird mit der Funktion *UARTCharGet* auf Zeichenempfang gewartet. Dabei wird nicht nur zum Zeitpunkt des Aufrufs der gegebenenfalls leere Buffer ausgelesen, sondern so lange gewartet, bis ein Zeichen eingegangen ist. Im Prinzip gibt es beim Empfang lediglich zwei Kategorien, zum einen das Lesen bzw. Aktualisieren der Daten und zum anderen das Senden von Daten.

Tabelle 5.4: Zulässige Befehle

ASCII-Zeichen	Name	Erläuterung
’0’	Read Array	Aktualisiert die Daten
’1’	Send diff. Sin.	Sendet differentielle Sinus-Werte
’2’	Send diff. Cos.	Sendet differentielle Cosinus-Werte
’3’	Send neg. Sin.	Sendet negative Sinus-Werte
’4’	Send pos. Sin.	Sendet positive Sinus-Werte
’5’	Send neg. Cos.	Sendet negative Cosinus-Werte
’6’	Send pos. Cos.	Sendet positive Cosinus-Werte
’7’	Send Sin. Offsets	Sendet Offsets der Sinus-Werte
’8’	Send Cos. Offsets	Sendet Offsets der Cosinus-Werte

Sollen Daten gesendet werden, so wird die Funktion *UARTCharPut* verwendet. Die Funktion füllt den Sendebuffer so lange mit Daten, bis dieser voll ist und wartet, bis wieder Platz verfügbar ist. Das Senden der Daten erfolgt mit Hilfe eines Pointers. Je nach angeforderten Daten wird der Pointer, umgewandelt als *uint8_t**-Zeiger, auf den Anfang des jeweiligen Speicherarrays gesetzt (vgl. Abbildung 5.2). Alle Speicherarrays haben eine Größe von 8x8 und besitzen 64 Elemente. Jedes Element hat eine Breite von 16 Bit. In einer for-Schleife werden mit Hilfe des Pointers die Zeichen an die Funktion *UARTCharPut* übergeben und nach jedem Schritt wird der *uint8_t*-Pointer um eins inkrementiert, was zur Folge hat, dass der Pointer auf die nächsten acht Bit im Speicher zeigt. Jedes Arrayelement wird somit durch zwei gesendete Zeichen repräsentiert. Daraus folgt, dass die for-Schleife für ein 64 Elemente umfassendes Array 128 mal durchlaufen wird.

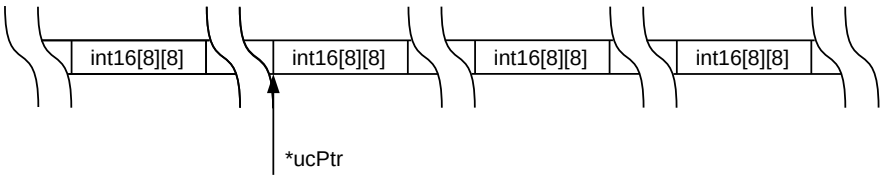


Abbildung 5.2: Lineare Anordnung im Speicher. In diesem Beispiel zeigt der Pointer auf das zweite Speicherarray.

Das Umrechnen der ADC-Werte in Spannungen erfolgt nicht auf dem Mikrocontroller und muss von der Auswerte-Software durchgeführt werden. Somit ist sichergestellt, dass keine Bits durch Rundung verloren gehen oder die berechneten Spannungen durch ein Zahlenformat in Festkommadarstellung in ihrer Nachkommastelle abgeschnitten werden und dadurch an Genauigkeit verlieren.

5.5 Optimierung der Verzögerungszeiten

Nach jedem umschalten der Multiplexer muss eine Verzögerungszeit eingehalten werden bis die Analog-Digital-Umsetzung gestartet werden kann. Dies resultiert aus den Einschwingzeiten der Multiplexer. Die Verzögerungszeit wird mit der Funktion *SysCtlDelay()* eingestellt. Dazu sollen einige Messungen folgen. Zunächst wird deshalb an den ADC-Eingängen gemessen. Zusätzlich wird das niederwertigste Adressbit A erfasst, da dieses die niedrigste Periodendauer hat und somit einen Zyklus repräsentiert. Abbildung 5.3 zeigt die Messskizze. Gemessen wird an einem ausgewählten Multiplexer der Arrayplatine.

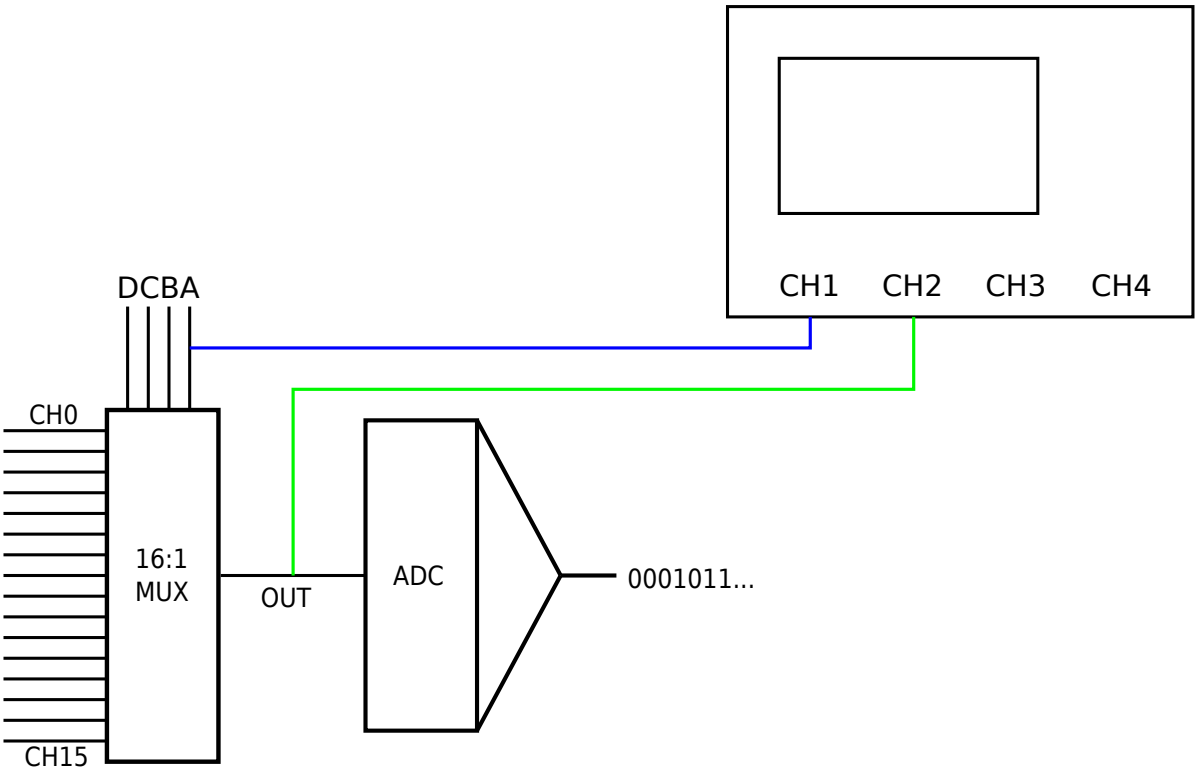


Abbildung 5.3: Messaufbau für die Bestimmung der Einschwingzeit

Einschwingzeit

Zunächst soll das Einschwingverhalten dargestellt werden. Dazu wird lediglich das Ausgangssignal des Multiplexers bzw. das Eingangssignal des ADC dargestellt. Als Verzögerungszeit wird zunächst eine relativ hohe angenommen und eingestellt. Dabei soll ein an- und absteigendes Signal gezeigt werden. Abbildung 5.4 zeigt, dass die Signale nach etwa 600 ns vollständig eingeschwungen sind.

Übersprechen

Da beim Schalten der Adressen steile Signalflanken auftreten, ist ein Übersprechen auf das Nutzsignal nicht auszuschließen. Trotz der beim Leiterplattendesign gesetzten Abblockkondensatoren nahe dem Versorgungspin des Multiplexers soll nun das Übersprechen genauer untersucht werden. Dazu wird das Schalt- und Nutzsignal gleichzeitig dargestellt und der entsprechende Bereich vergrößert (vgl. Abbildung 5.5). In Abbildung 5.5 ist zu sehen, wie die Schaltflanke das Nutzsignal etwa 50 ns beeinflusst. So ist im insgesamt fallenden Nutzsignal zunächst ein kurzer Anstieg durch die steigende Schaltflanke zu erkennen. Da das Signal jedoch insgesamt etwa 600 ns zum Einschwingen benötigt stellt das Übersprechen kein Problem dar.

Anpassung der Verzögerungszeit

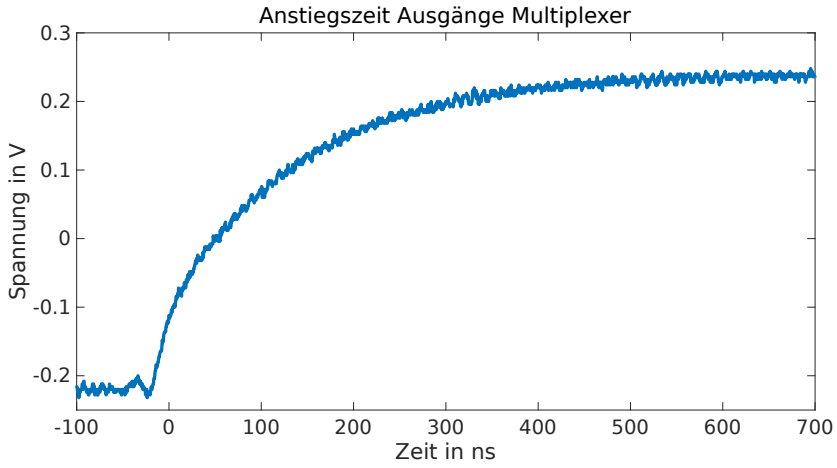
Die Funktion *SysCtlDelay()* aus der TivaWare Bibliothek von Texas Instruments stellt eine Verzögerung zur Verfügung. Es handelt sich um einen in Assembler geschriebenen Code von drei Instruktionen. In den Klammern wird angegeben, wie oft diese drei Instruktionen aufgerufen werden sollen. Mit der Taktfrequenz f_{Clock} des Mikrocontrollers und der gewünschten Verzögerungszeit t_{delay} lässt sich daraus die Anzahl wie folgt (Formel 5.1) bestimmen:

$$n = \frac{f_{Clock}}{3} \cdot t_{delay} \tag{5.1}$$

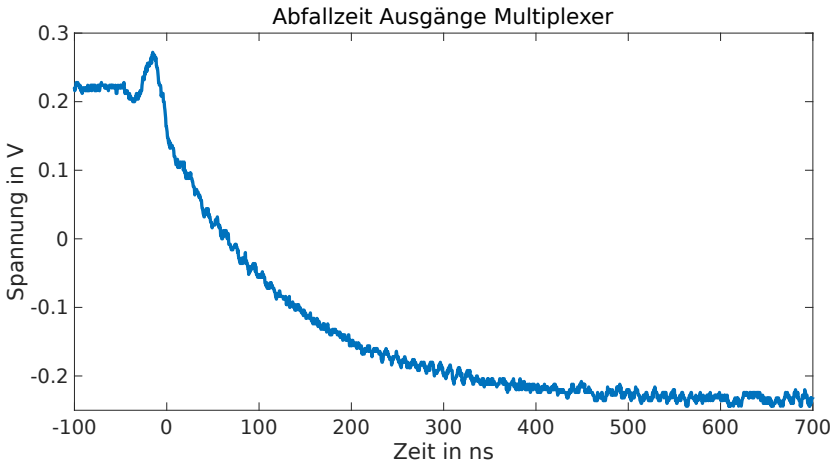
Um beispielsweise eine Verzögerung von $t_{delay} = 1\,\mu s$, bei einer Taktrate von $f_{Clock} = 120\,MHz$ zu erhalten werden n Schleifendurchläufe benötigt:

$$n = \frac{120\,MHz}{3} \cdot 1\,\mu s = 40$$

Um die Laufzeit zu optimieren wird die Verzögerungszeit verringert. Abbildung 5.6 zeigt die Anpassung der Verzögerungszeit auf 1 µs.



(a) Anstiegszeit



(b) Abfallzeit

Abbildung 5.4: Einschwingzeiten der Multiplexer für an- und absteigende Nutzsignale (AC-Kopplung). Beide Signale sind nach etwa 600 ns eingeschwungen.

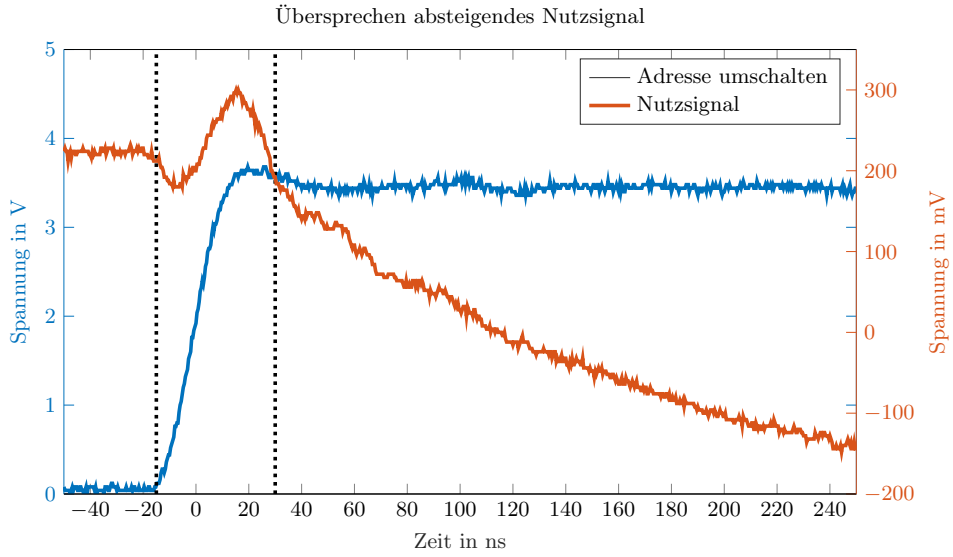


Abbildung 5.5: Übersprechen der Schaltflanke auf Multiplexersignal. Dauer etwa 50 ns.

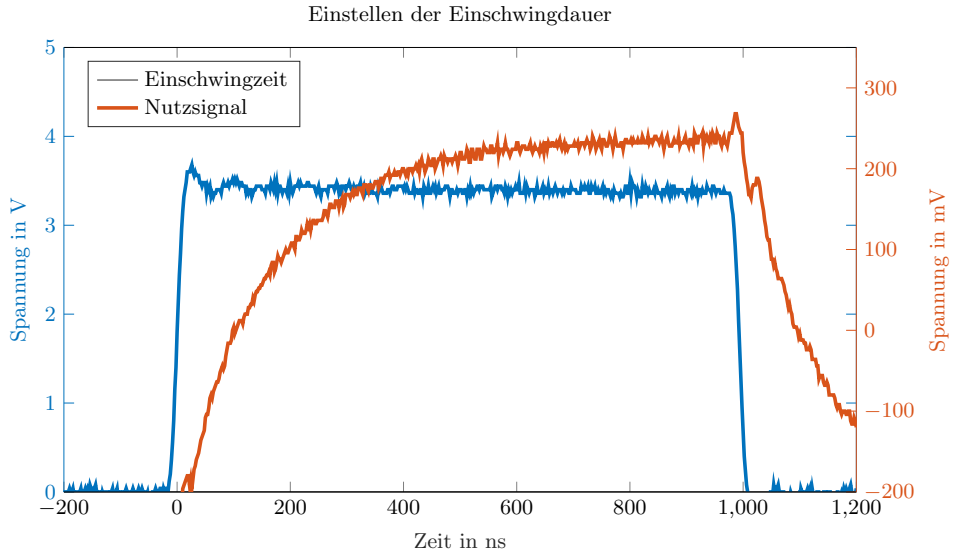


Abbildung 5.6: Einstellen der Verzögerungszeit auf 1 µs. Dargestellt werden die Verzögerungszeit und das Nutzsignal.

6 Auswerte-Software und Messungen

Um die Daten aus dem Mikrocontroller verarbeiten und darstellen zu können, wird in diesem Kapitel die Auswerte-Software, welche aus verschiedenen Matlab-Skripten besteht, beschrieben.
Zunächst folgt die Auswertesoftware für die Kommunikation zwischen Mikrocontroller-Board und Computer. Anschließend werden einige ausgewählte Datensätze visualisiert und ausgewertet.

6.1 Kommunikation zum Mikrocontroller

Um auf die serielle Schnittstelle mittels Matlab zugreifen zu können, muss zunächst ein Objekt der seriellen Schnittstelle erstellt werden. Dies geschieht mit der Funktion *serial()*. Das Objekt wird mit *fopen()* geöffnet und ist anschließend bereit für die Kommunikation. Die Funktion *fwrite()* wird dazu genutzt, um Befehle an den Mikrocontroller zu übermitteln. Im Kapitel 5 findet sich eine Übersicht über alle implementierten Befehle (Tabelle 5.4). Sofern Daten angefordert werden, ist immer bekannt, dass 64 *int16*-Werte zu empfangen sind, gesendet werden allerdings 128 *char*-Zeichen. Mit der Funktion *fread()* lassen sich die aufgeteilten Werte zusammenführen. Dazu wird angegeben, wie viele Werte in welchem Format gespeichert werden.
Da jeweils ein einzeiliger Vektor gesendet bzw. empfangen wird, muss dieser in die richtige Struktur überführt werden. Mit Hilfe der Funktion *reshape()* lassen sich die Werte wieder als 8x8-Array anordnen. Hierbei ist zu beachten, dass die Werte spaltenweise neu angeordnet werden. Deshalb ist die Ergebnismatrix zu transponieren, um die ursprüngliche Anordnung zu erhalten.

6.2 Messplatz

Im Projekt ISAR steht ein Robotermessplatz zur Verfügung. Mit diesem können Positionen präzise angefahren werden. Die Positionierung ist in x-, y- und z-Richtung möglich. Zusätzlich lassen sich eine Verkippung der Ebene erreichen und eine Umdrehung um die z-Achse realisieren. An der z-Achse lassen sich verschiedene Gebermagneten befestigen und somit eine Rotation um 360° erreichen. Die Software zum Steuern des Messplatzes wird aus Vorarbeiten übernommen.
Die Befestigungsplatte für das erstellte Sensor-Array wird neu gefertigt. Dies ist notwendig, um alle Sensoren anfahren zu können. Durch die Maße des neuen Sensor-Arrays

ist diese Eigenschaft mit der vorherigen Befestigungsplatte nicht mehr gegeben. Die Befestigung der Grundplatte auf dem Sockel geschieht durch M8-Schrauben anhand der vorhandenen Bohrungen. Abbildung 6.1 zeigt den Messaufbau.

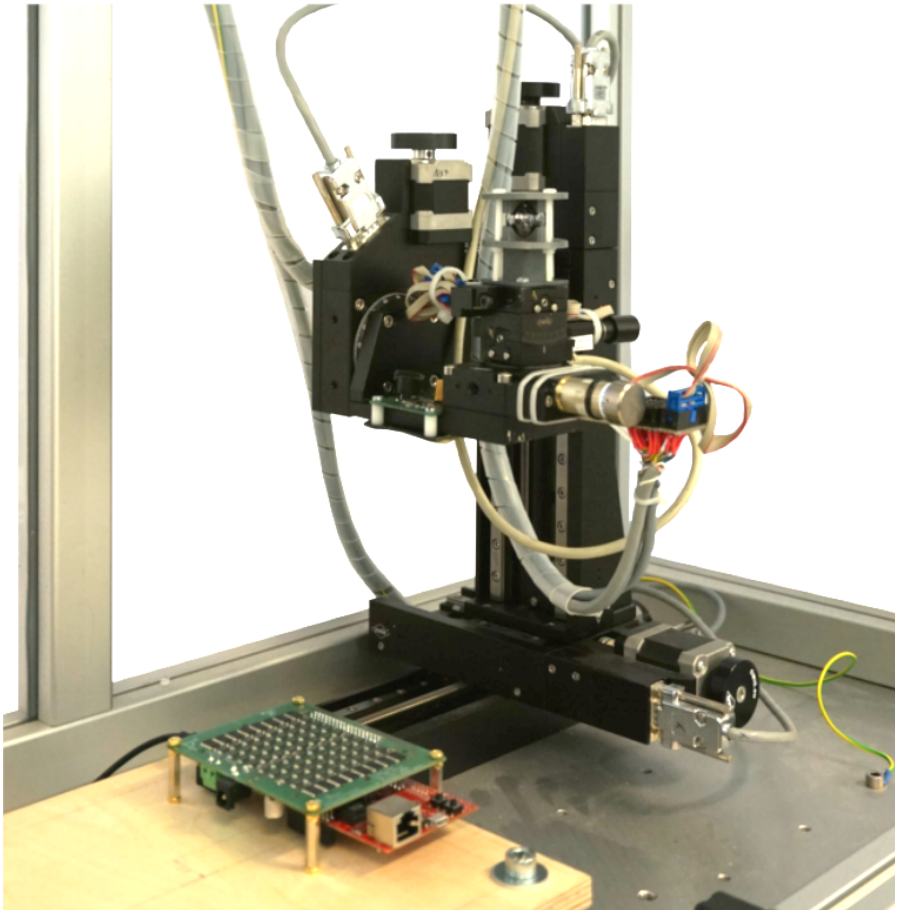


Abbildung 6.1: Das Sensor-Array ist auf dem Sockel des Messplatzes montiert und bereit für die Messaufnahme.

6.3 Messungen

Mit Hilfe des Robotermessplatzes wurde zunächst eine Messreihe aufgenommen, bei welcher der Gebermagnet um das Zentrum herum in neun verschiedenen Positionen eine 360° Messung in 2° Schritten durchführt. Für jede Position werden sämtliche Daten von allen Sensoren abgespeichert. Das Zentrum des Gebermagneten befindet sich etwa 3 cm von der Arrayoberfläche entfernt. Abbildung 6.2 gibt eine Übersicht der angefahrenen Positionen.

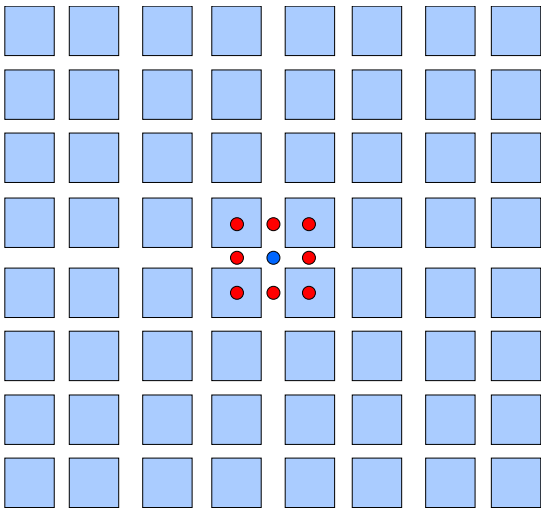


Abbildung 6.2: Visualisierung der angefahenen Messpunkte in der x-y-Ebene. Das Zentrum des Gebermagneten befindet sich ca. 3 cm über dem Array.

Im Folgenden sollen nun Auswertungen erfolgen, bei denen der Gebermagnet über Sensor (4,4) positioniert ist. Abbildung 6.3 zeigt den Betrag der Differenzspannung für das gesamte Array bei einem Winkel von 0°. Für den Sensor (4,4), im Bild als D4 bezeichnet, wird jeweils die positive, negative und die Differenz der Sinus- und Cosinus-Spannung dargestellt. Der Vergleich in Abbildung 6.4 zeigt entgegengesetztes Verhalten der Messpunkte A1 und H1. Zuletzt soll noch Bezug auf das Verhalten von TMR-Vortex Sensoren genommen werden. Diese lassen sich in drei Bereiche(Linar, Übergang und Sättigung) aufteilen. Abbildung 6.5 zeigt den Betrag der Sensorwerte bei 0° sowie eine Kreisdarstellung, in welcher sich die drei Bereiche erkennen lassen.

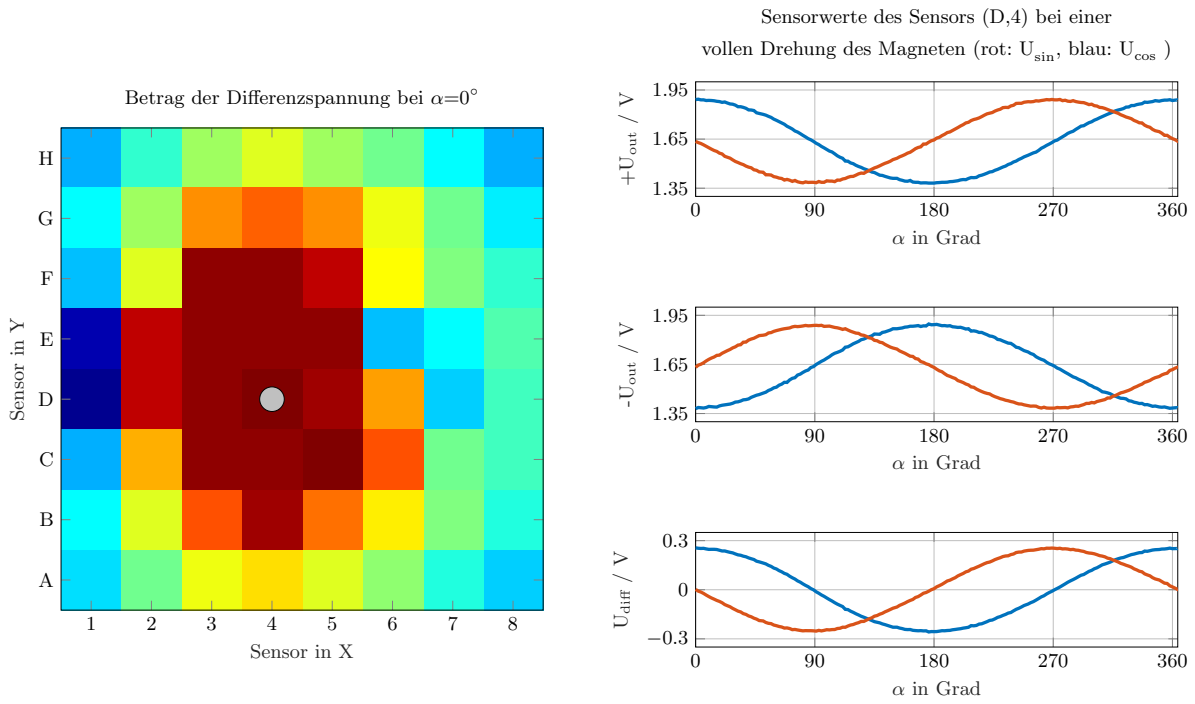
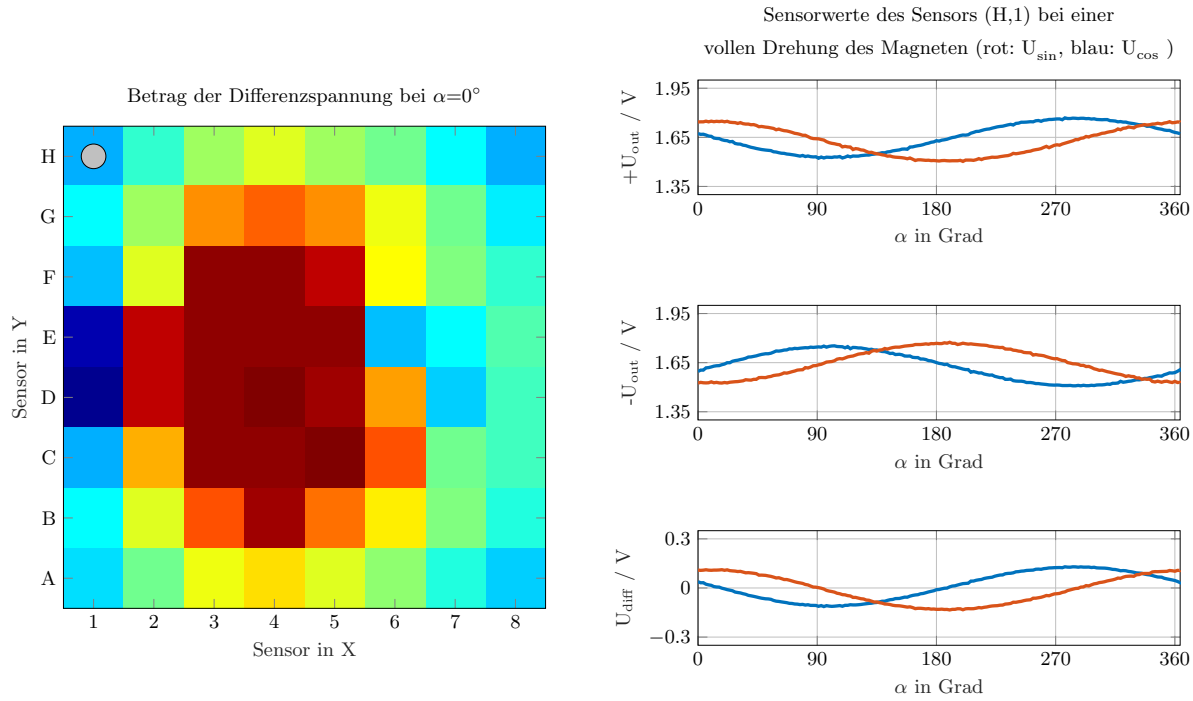
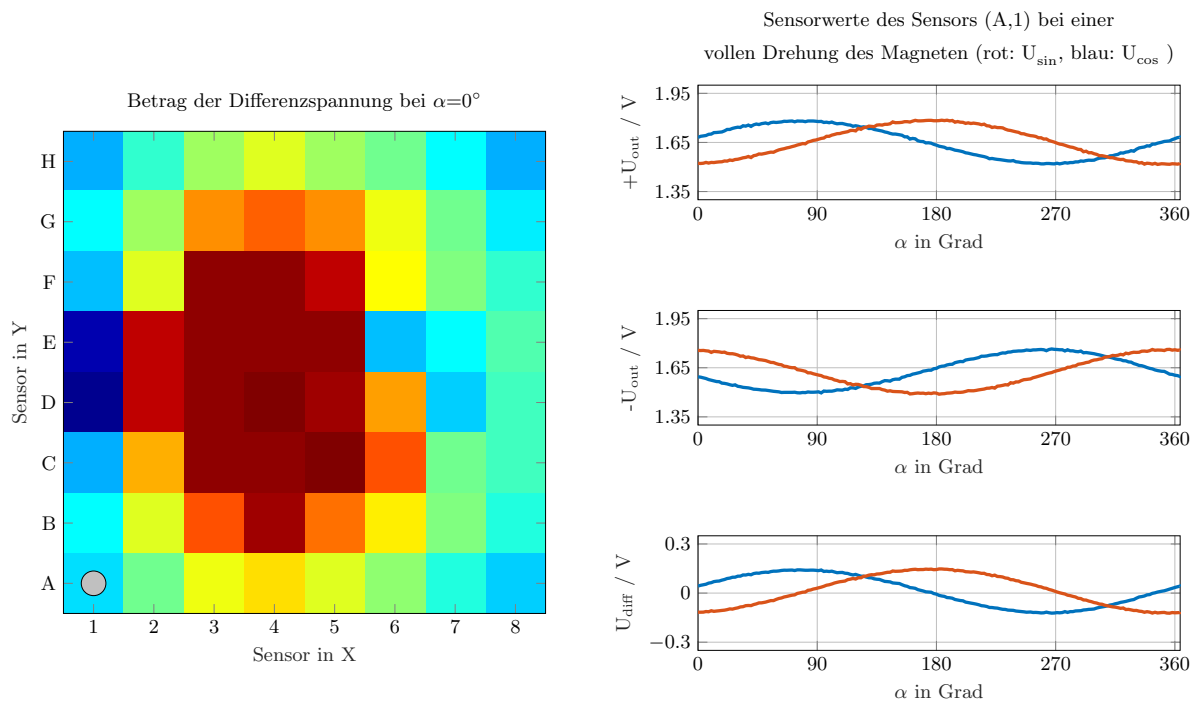


Abbildung 6.3: Gebermagnet über Messpunkt D4. Darstellung der Differenzspannung bei einem Winkel von 0° (links); Sensorwerte D4 (rechts).



(a) Darstellung des Messpunktes H1.



(b) Darstellung des Messpunktes A1.

Abbildung 6.4: Um 180° verschobenes Verhalten der entgegengesetzten Sensoren.

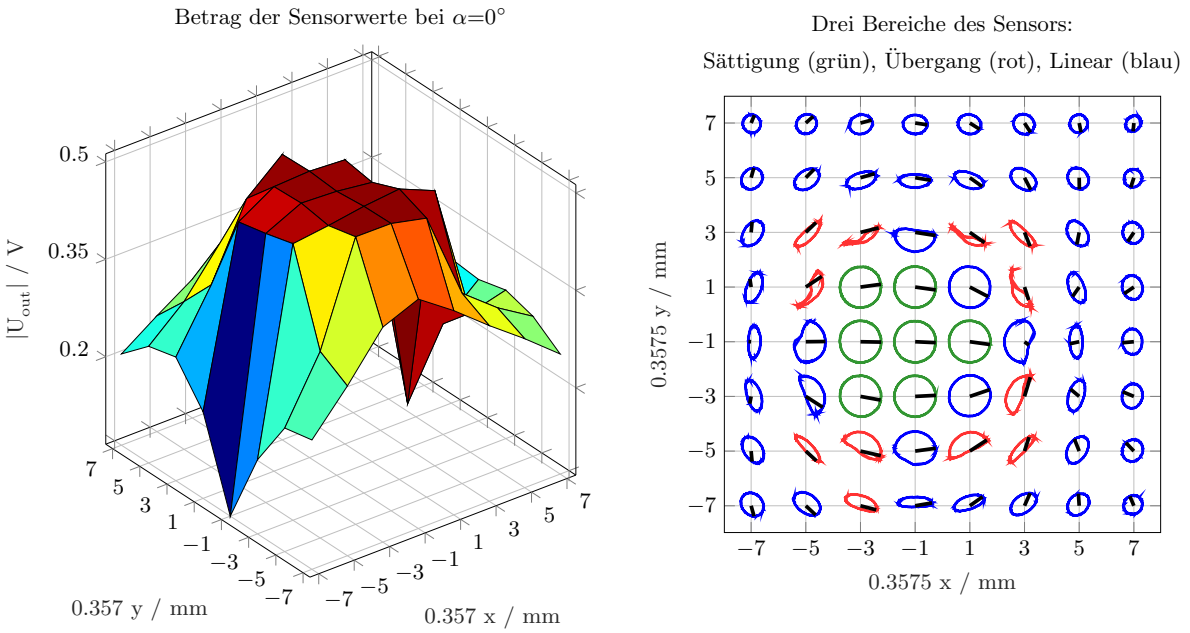


Abbildung 6.5: Gebermagnet über Messpunkt D4. Darstellung der Differenzspannung bei einem Winkel von 0° (links); Kreisdarstellung aller Sensorwerte D4 (rechts). Mittige Sensoren befinden sich im Sättigungsbereich (grün), alle äußeren Sensoren befinden sich im linearen Bereich (blau), der Übergangsbereich wird rot dargestellt.

7 Schlussfolgerungen

Dieses Kapitel stellt einen Überblick der Ergebnisse dieser Arbeit zusammen. Dabei werden die erreichten Ziele dargestellt und anschließend beschrieben, wie auf diese Arbeit aufgebaut werden kann. Dabei werden einige Anregungen für die Fortführung und Verbesserung gegeben.

7.1 Zusammenfassung

Das Kernziel dieser Arbeit war der Entwurf eines funktionstüchtigen 8x8 Sensor-Arrays auf Basis von TMR-Sensoren des Herstellers TDK. Dabei konnte auf Vorarbeiten, in denen ebenfalls Sensor-Arrays erstellt worden sind, zurückgegriffen werden. Die verwendeten Sensoren bringen einige Vorteile mit sich, da hier in einem Sensorgehäuse zwei Vollbrücken vereint sind. Dabei war es erforderlich ein neues Konzept für die Beschaltung und Software zu entwerden.

Anfangs wurden im Grundlagenteil die Prinzipien der magnetoresistiven Effekte erläutert. Im Speziellen wurde dabei auf den TMR-Effekt, mit seinen Varianten linear und vortex, eingegangen. Abschließend wurde der TMR-Sensor, welcher die Grundlage des Sensor-Arrays darstellt, betrachtet.

Nach den Grundlagen erfolgte der Hardwareentwurf des Sensor-Arrays. Dazu wurden zunächst verschiedene Schaltungskonzepte herausgearbeitet und dargestellt. Nach einer abschließenden Bewertung der Schaltungsvarianten, bei welcher die Variante mit analogen Multiplexern gewählt wurde, erfolgte der Schaltungs- und Layoutentwurf. Nach erfolgreichem Erstellen des Layouts wurde die Fertigung und die anschließende Bestückung der Platine in Auftrag gegeben.

Anschließend folgte der Funktionstest und die Inbetriebnahme der Hardware. Neben der optischen Überprüfung fand eine Abschätzung der Stromaufnahme mit anschließender Messung statt. Im weiteren Vorgehen wurde eine Software für den Mikrocontroller erstellt. Mit einer Visualisierung der Messdaten in Matlab konnte die Funktionalität der Hard- und Software erfolgreich überprüft werden.

Danach wurde die Software auf dem Mikrocontoller bearbeitet. Dazu wurde die bereits erstellte Software wiederverwendet und erweitert. Dabei wurde zunächst auf die Konfigurationen der Module und Pins eingegangen. Ebenfalls wurde das Erfassen der ADC-Werte mit den anschließenden Berechnungen genauer betrachtet. Die Kommunikation über UART wurde behandelt und abschließend wurden die Verzögerungszeiten optimiert.

Zuletzt sind die Auswertesoftware und eine abschließende Messung dargestellt. Dazu wurde die Kommunikation zum Mikrocontroller näher betrachtet. Für die Messungen wurde auf den im ISAR-Projekt vorhandenen Robotermessplatz mit den entsprechenden Skripten zur Steuerung zurückgegriffen. In der Messauswertung ist das Verhalten ausgewählter Sensoren bei einer vollen Drehung um 360° dargestellt. Ebenfalls wurden die drei Bereiche der TMR-Vortex Sensoren dargestellt.

In dieser Arbeit wurde ein 8x8 Sensor-Array auf Basis von TMR-Sensoren erfolgreich erstellt. Es wurde ein neues Konzept für die Schaltung und Software erarbeitet und erfolgreich umgesetzt. Das Sensor-Array kann als ein Beispiel für die Beschaltung der 64 Sensoren gesehen werden. Die Generierung neuer Datensätze zum Testen und Entwickeln neuer Auswertealgorithmen im ISAR-Projekt wird mit dem Sensor-Array möglich.

7.2 Herausforderungen

Da bei dieser Arbeit auf Sensoren mit jeweils vier Sensorausgängen zurückgegriffen wurde musste die Beschaltung aus den Vorarbeiten überdacht werden. In der vorherigen Arbeit wurden Sensoren mit zwei Sensorausgängen genutzt. Würde man die Beschaltung übernehmen, so bräuchte man die doppelte Anzahl an analogen Eingängen (32). Somit wurden mehrere Konzepte vorgestellt. Durchgesetzt hat sich letztendlich das Konzept mit analogen Multiplexern. Ein Vorteil liegt darin, dass nur eine Spannungsversorgung für alle Sensoren benötigt wird.

Beim Verdrahten der Platine ergaben sich zunächst Schwierigkeiten durch die hohe Anzahl an Verbindungen zwischen Sensoren und Multiplexer. Letztlich wurde eine günstige Anordnung der Sensoren und Multiplexer gefunden, welche das Verdrahten ermöglichte. Zunächst wurde eine Baugruppe für die linke Arrayhälfte erstellt, wobei die Pins zur Wahl der Addresse gut zu verdrahten waren. Für die rechte Arrayhälfte wurde der Multiplexer zunächst genau so wie für die linke Arrayhälfte ausgerichtet. Dabei waren jedoch die Leitungen zwischen Sensoren und Multiplexer störend, da diese die Adresspins der rechten Arrayhälfte kreuzten. Durch Drehung der Multiplexer um 180° ließ sich dieses Problem beheben. Dabei musste beachtet werden, dass sich durch die Drehung die Positionen der entsprechenden Pins des Multiplexers ändern. Damit ist die Reihenfolge der Kanäle und die der Sensorsignale auf der linken und rechten Arrayhälfte nicht mehr identisch. Durch Invertierung des höchstwertigen Bits ließ sich dieses Problem beheben.

Bedingt durch die neue Beschaltung musste eine grundlegend neue Software erstellt werden. Um beim Programmieren die Übersicht zu behalten, wurden Makros der analogen Eingänge erstellt. Dies half bei der Zuweisung der korrekten Reihe und Arrayhälfte an die richtigen Positionen der Speicherarrays. Für die Übertragung der Messwerte über UART wurde mit Zeigern gearbeitet. Dabei traten zunächst Fehler auf, welche durch eine Überschreitung der Grenzen der Speicherarrays verursacht wurde und zu fehlerhaften

Werten führte. Beim Einlesen der Daten in Matlab ergab sich aufgrund der Funktion *reshape()* eine andere Anordnung. Durch Transponieren der Empfangsmatrix ließ sich die ursprüngliche Anordnung wieder erreichen.

Bei den Messungen mittels Robotermessplatz ergaben sich erhebliche Schwierigkeiten. Zunächst fiel auf, dass die vorhandene Grundplatte zur Fixierung des Arrays für das in dieser Arbeit entworfene Array nicht geeignet ist. Da sich die Grundfläche des Sensor-Arrays vergrößert hat, konnten nicht mehr alle Sensoren angefahren werden. Aus diesem Grund wurde eine neue Grundplatte angefertigt. Bei den Messungen traten immer wieder Fehler auf, welche zum Abbruch der Messungen führten. Als fehleranfällig fiel dabei insbesondere die Drehung um die z-Achse auf, bei welcher der Schrittmotor blockierte und eine Neujustierung des Messplatzes erforderlich wurde.

Fazit

Zusammenfassend lässt sich sagen, dass im Rahmen dieser Bachelorarbeit ein funktionstüchtiges TMR-Sensor-Array erstellt worden ist. Dabei wurde ein neues Konzept für die Beschaltung entwickelt und umgesetzt. Eine neue Software wurde für das Sensor-Array erstellt und ermöglicht das Auslesen des Sensor-Arrays. In den Messungen konnten die drei Bereiche des TMR-Vortex-Sensors nachgewiesen werden. Das Sensor-Array kann im Projekt ISAR zur Detektion von Magnetfeldern eingesetzt werden und dient als Funktionsmodell.

7.3 Ausblick

Um mit dem Sensor-Array aussagekräftige Daten zu erfassen und zu veranschaulichen, sollten weitere Messungen erfolgen. Interessant wären beispielsweise Vergleichsmessungen zu den Vorgängerarbeiten, um auf die Eigenschaften des verwendeten Sensors Bezug nehmen zu können. Ebenfalls sollten Messungen des Arrays mit Störfeldaufschlag erfolgen. Im Rahmen weiterer Messungen könnten Hysterese-Effekte nachgewiesen werden. Ebenfalls interessant wäre gegebenenfalls eine Verkippung des Gebermagneten.

Der Robotermessplatz sollte überarbeitet werden. Die Software zum Steuern ist sehr umfangreich und bedarf einiger Einarbeitungszeit. Insbesondere sollte an der Drehung um die z-Achse gearbeitet werden, damit eine Messung verlässlich möglich wird.

Die Software stellt alle benötigten Funktionen zum Auslesen des Sensor-Arrays zur Verfügung, könnte jedoch auf Interrupts umprogrammiert werden. Weitere Funktionen wie eine Interpolation könnten noch hinzugefügt werden.

Literatur

[1] Abraham Begic. *Tunnel-Magnetoresistives Sensor-Array - Controllersteuerung, Platinen-Layout und Prüfstands-Erprobung*. 2018.

[2] Herbert Bernstein. *Messelektronik und Sensoren*. Springer Vieweg, 2014.

[3] NVE Corporation. *AAT00x Ultralow Power TMR Angle Sensors*. 2017.

[4] TDK Corporation. *Catalog TMR Angle Sensor – TAS series*. 2017.

[5] Konrad Reif. *Sensoren im Kraftfahrzeug*. Vieweg + Teubner Verlag, 2010.

[6] Sam Sattel. *What Are Decoupling Capacitors?* URL: https://www.autodesk.com.../products/eagle/blog/what-are-decoupling-capacitors/. (Besucht am 10. 05. 2019).

[7] Thorben Schüthe. *Abbildung TMR-Sensor Vergleich: linear und vortex*. 2019.

[8] Rolf Slatter. *Tunnelmagnetoresistive Sensoren für die Antriebstechnik*. 2017.

[9] Ulrich Tietze und Christoph Schenk. *Halbleiter-Schaltungstechnik*. Springer Verlag, 2002.

[10] Thomas Tille. *Automobil-Sensorik*. Springer Vieweg, 2016.

[11] *Tiva™ C Series TM4C1294 Connected LaunchPad Evaluation Kit EK-TM4C1294XL User's Guide*. 2014.

[12] *Tiva TM TM4C1294NCPDT Microcontroller DATA SHEET*. Texas Instruments. 2015.

[13] *TivaWareTM Peripheral Driver Library USER'S GUIDE*. Texas Instruments. 2013.

[14] Analog Devices MT-088 Tutorial. *Analog Switches and Multiplexers Basics*. 2009. (Besucht am 07. 05. 2019).

[15] Tobias Wurft. *Investigation of the Magnetic Vortex State for Spin-Valve Sensors*. 2018.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang

A Schaltung

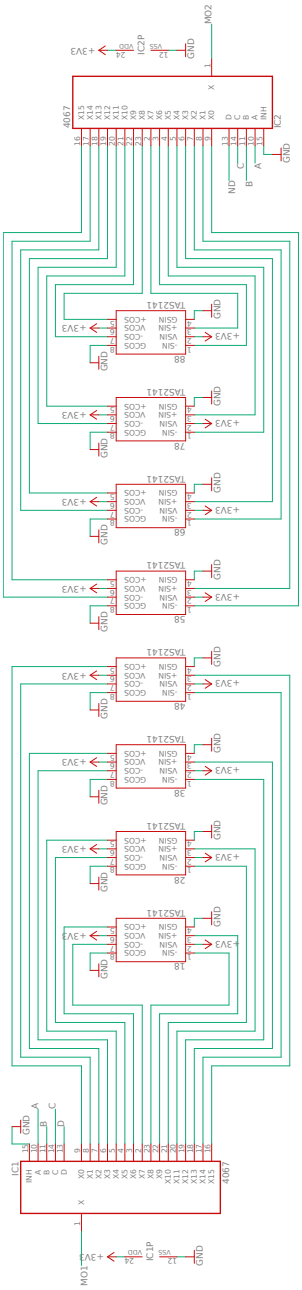


Abbildung A.1: Schaltplan einer einzelnen Reihe des Sensor-Arrays

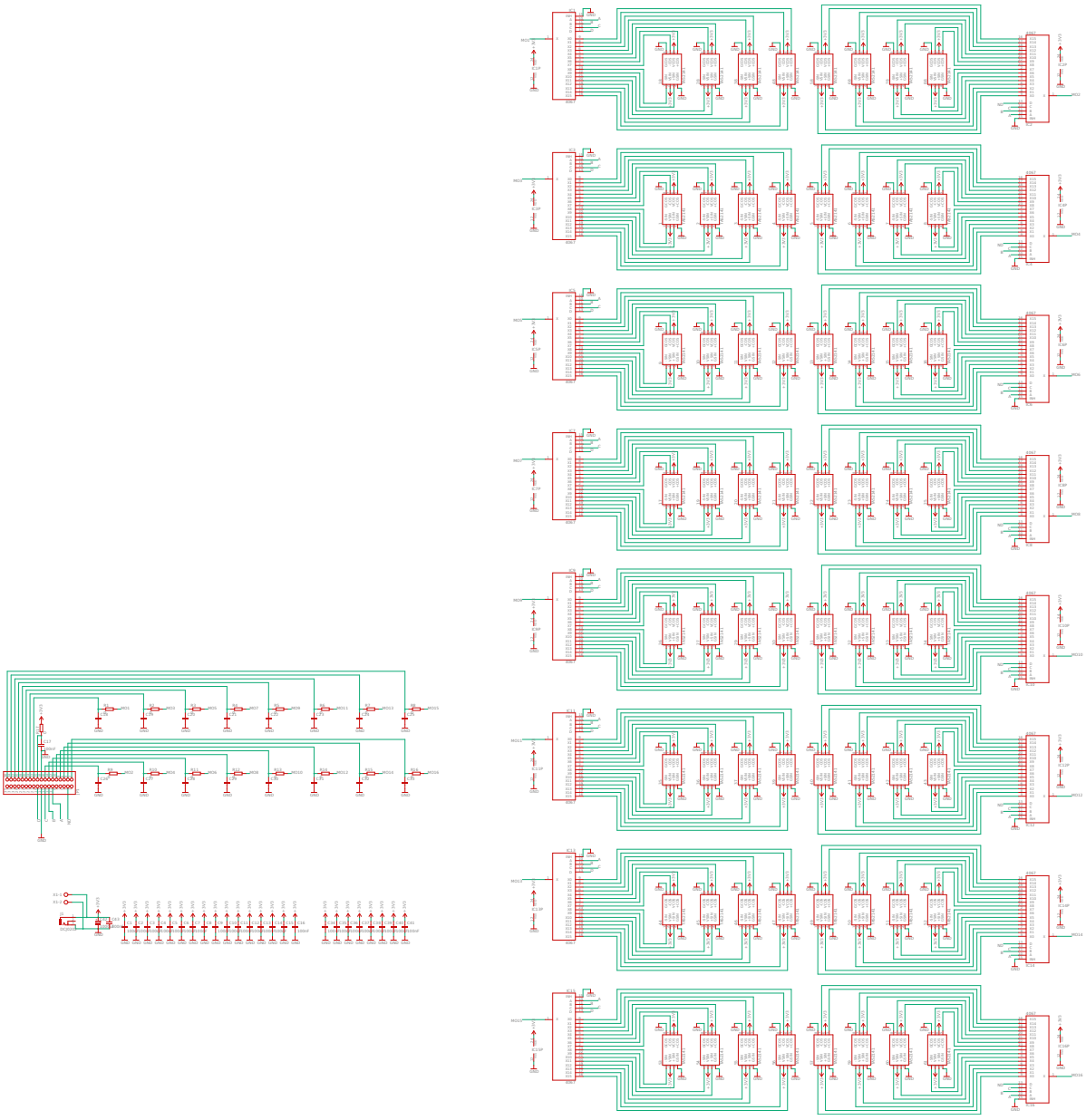


Abbildung A.2: Komplettansicht der Schaltung

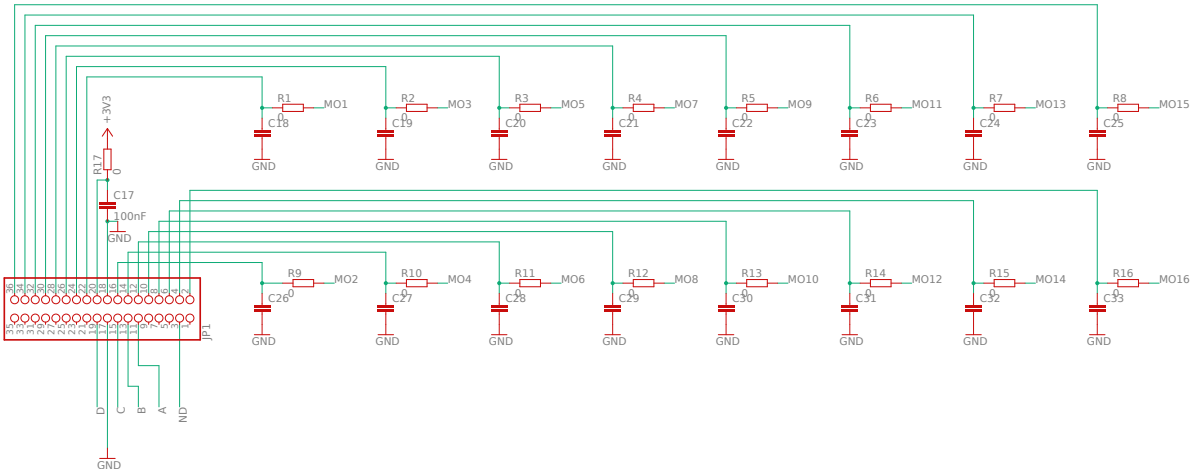


Abbildung A.3: Anschlussführung mit RC-Tiefpässen

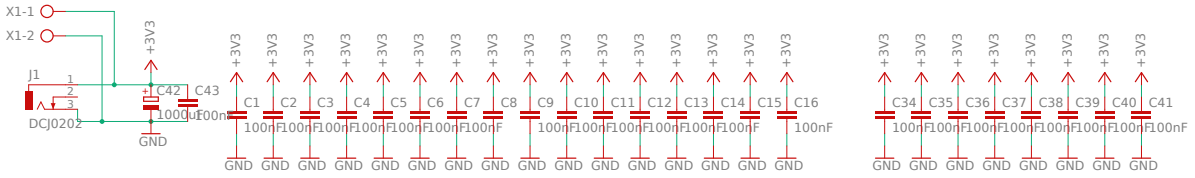


Abbildung A.4: Spannungseinspeisung und Abblockkondensatoren

B Platinenlayout

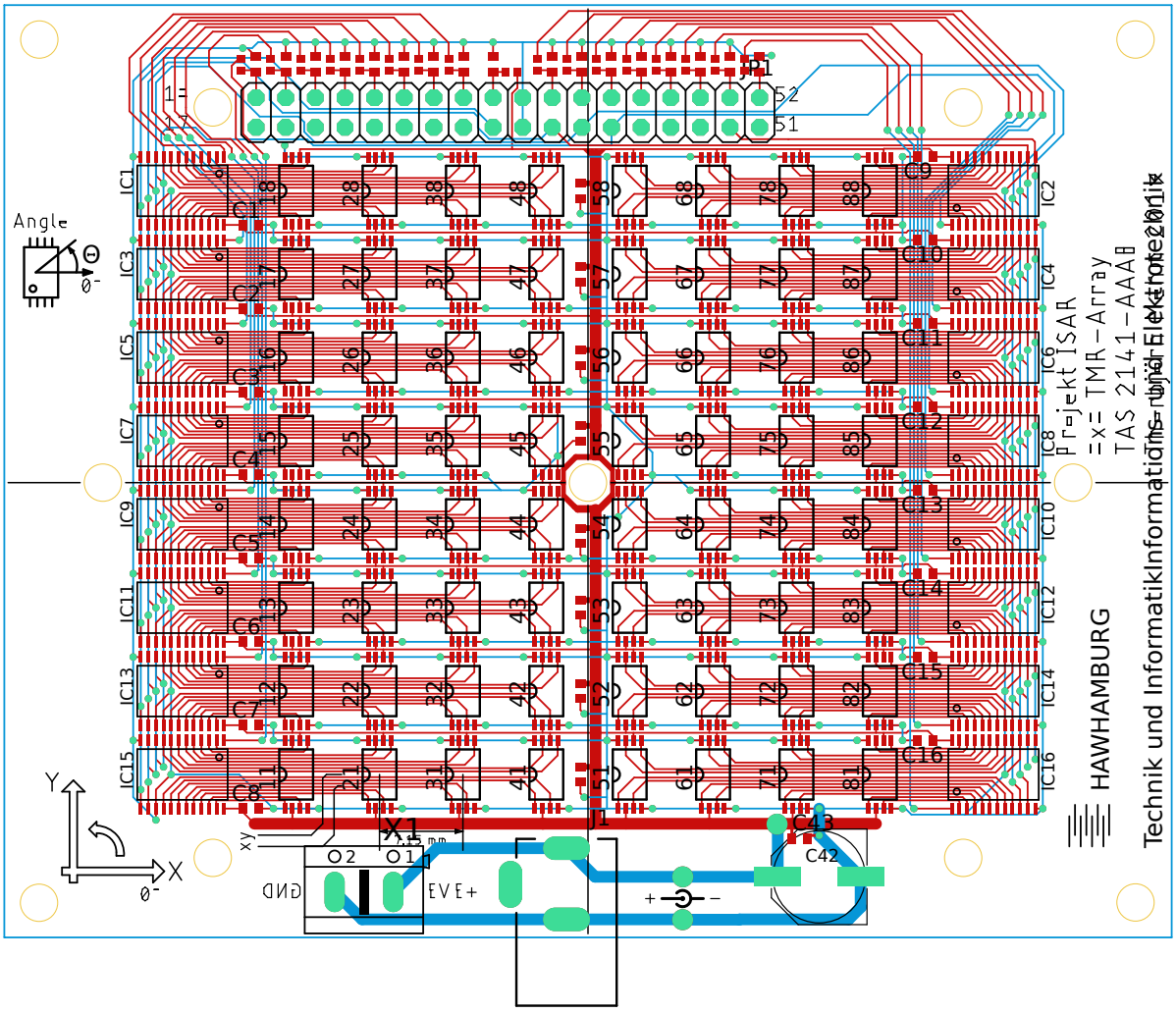


Abbildung B.1: Komplettansicht des Platinenlayouts

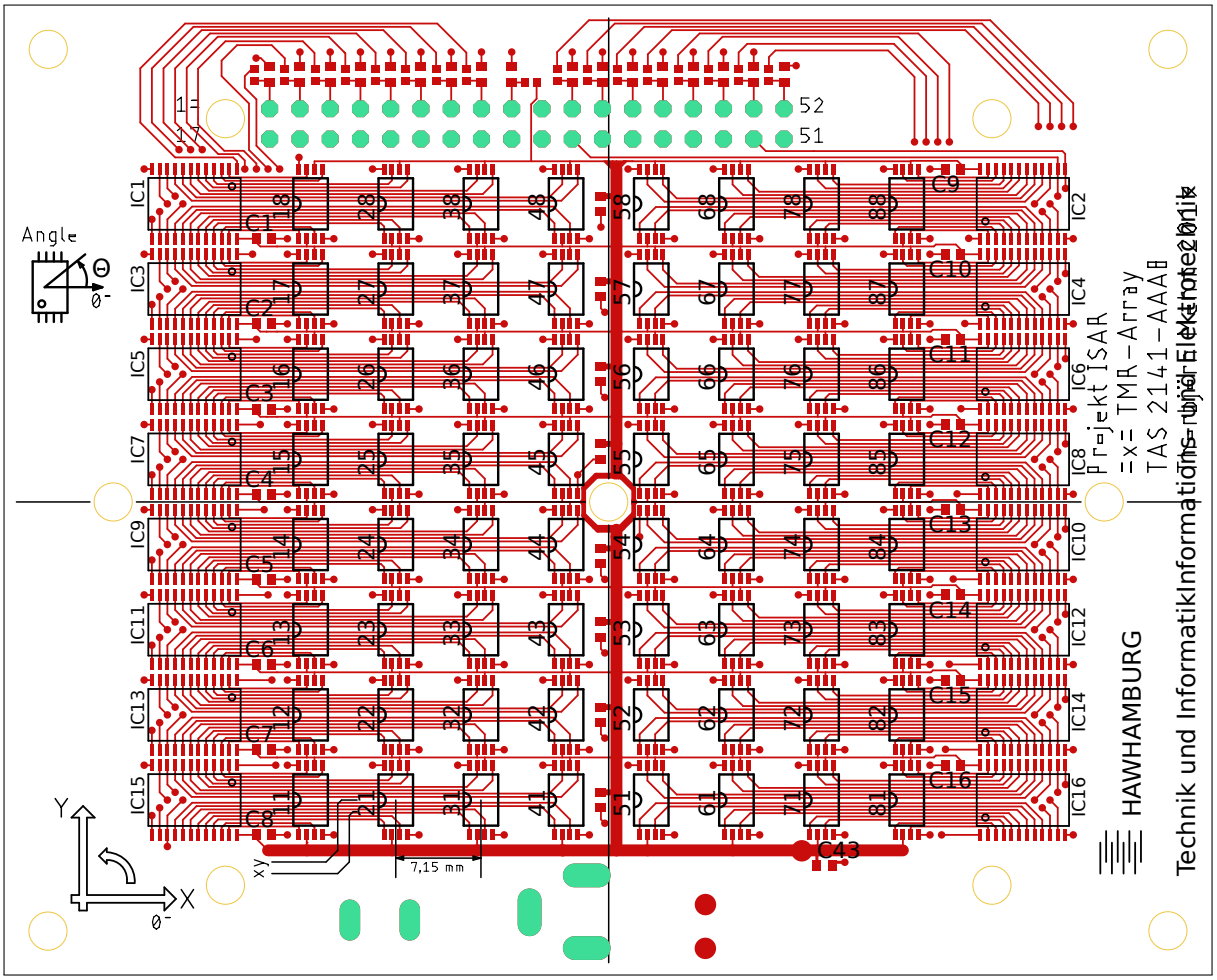


Abbildung B.2: Toplayer des Platinenlayouts

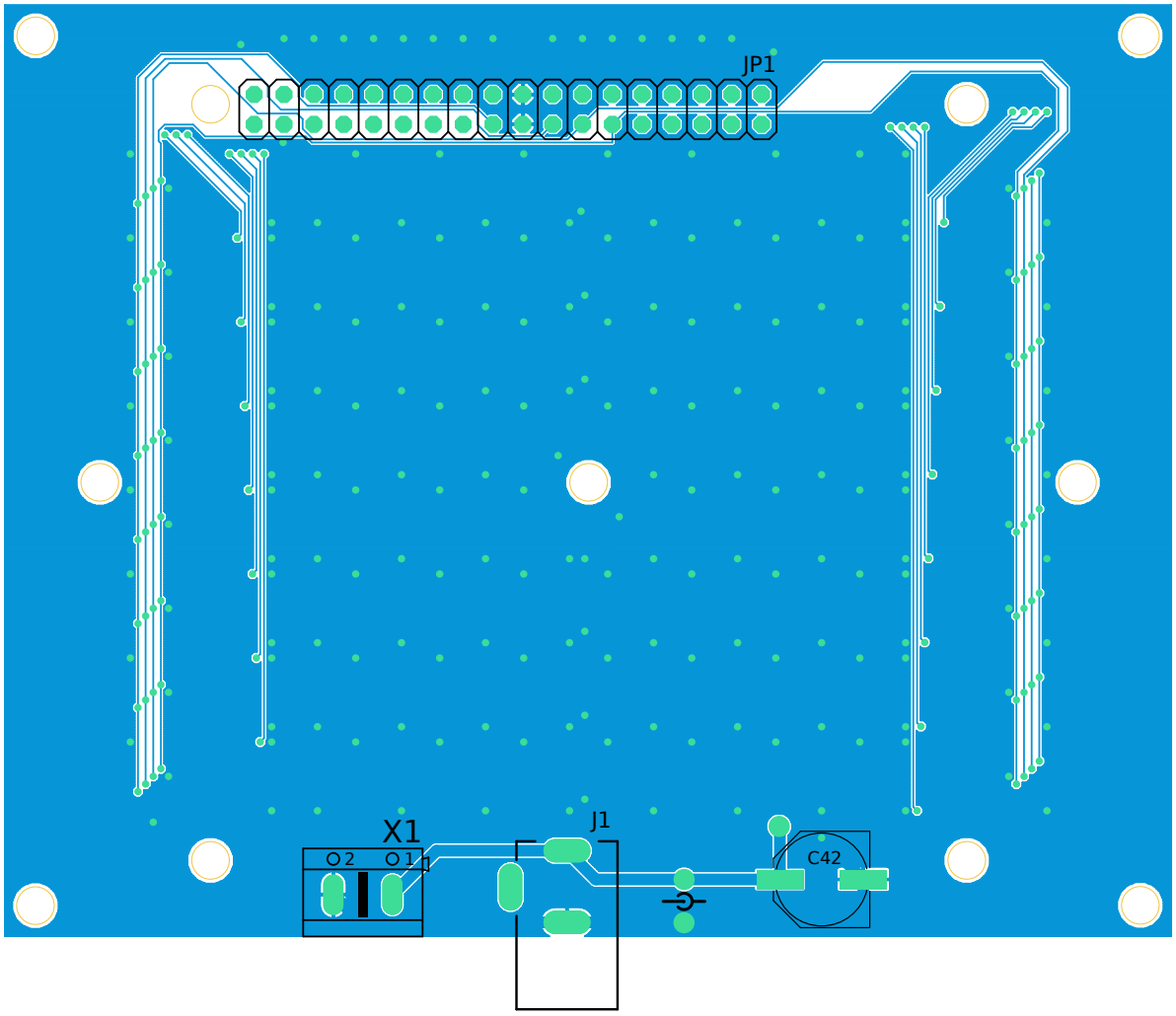


Abbildung B.3: Bottomlayer des Platinenlayouts

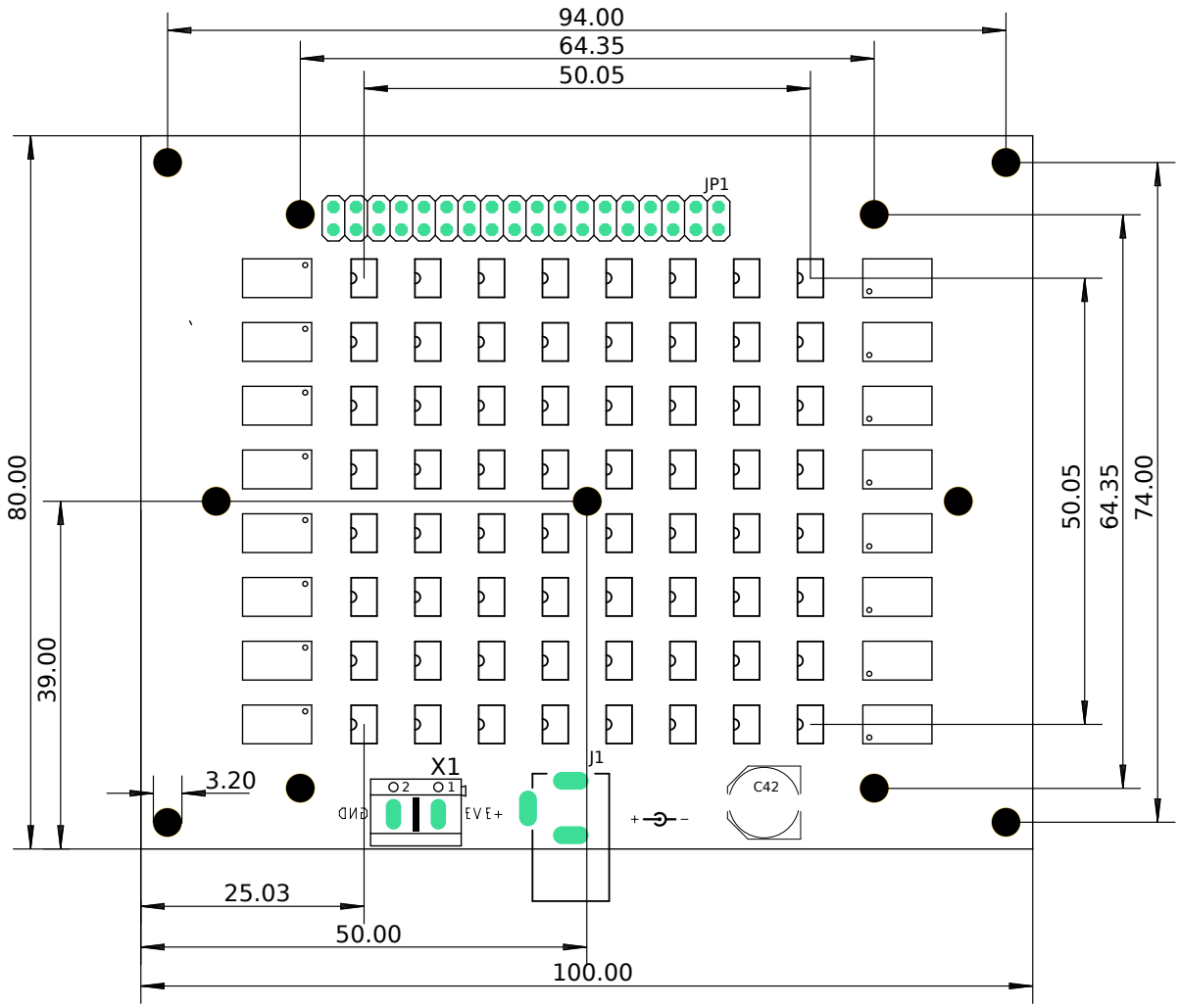


Abbildung B.4: Maße der Platine

C C-Quellcodes

Quellcode C.1: include.h

```
/*
 * include.h
 *
 *  Created on: 02.05.2019
 *      Author: tmehm
 */

#ifndef INCLUDE_H_
#define INCLUDE_H_

#include <stdint.h>
#include <stdlib.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/adc.h"
//#include "driverlib/adc.c"
#include "driverlib/gpio.h"
#include "drivers/pinout.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"


#define ROW_1_L ADC_CTL_CH1
#define ROW_2_L ADC_CTL_CH0
#define ROW_3_L ADC_CTL_CH9
#define ROW_4_L ADC_CTL_CH8
#define ROW_5_L ADC_CTL_CH16
#define ROW_6_L ADC_CTL_CH17
#define ROW_7_L ADC_CTL_CH18
#define ROW_8_L ADC_CTL_CH19

#define ROW_8_R ADC_CTL_CH6
#define ROW_7_R ADC_CTL_CH7
#define ROW_6_R ADC_CTL_CH4
#define ROW_5_R ADC_CTL_CH5
#define ROW_4_R ADC_CTL_CH12
#define ROW_3_R ADC_CTL_CH14
#define ROW_2_R ADC_CTL_CH15
#define ROW_1_R ADC_CTL_CH13

#define GPIO_PIN_3_DOWNTO_0 0x0000000F

//*****************************************************************************
//
// System clock rate in Hz.
//
```

```
//*****************************************************************************
uint32_t g_ui32SysClock;
uint32_t i;
uint32_t ADCValues_SS0[8];
uint32_t ADCValues_SS1[4];
uint32_t ADCValues_SS2[4];

// All ADC-Values as int16
int16_t SinResults[8][16];
int16_t CosResults[8][16];

// Signal matrices
int16_t negSinResults[8][8];
int16_t posSinResults[8][8];
int16_t negCosResults[8][8];
int16_t posCosResults[8][8];

int16_t DiffCosResults[8][8];
int16_t DiffSinResults[8][8];

int16_t SinOffset[8][8];
int16_t CosOffset[8][8];

int16_t receive;

// Base address pointer to Value Arrays
uint8_t *ucPtr;


#endif /* INCLUDE_H_ */
```

Quellcode C.2: main.c

```
//*****************************************************************************
//
//  Main program
//
//*****************************************************************************

#include "include.h"
#include "prototypes.h"
//*****************************************************************************
//
// The error routine that is called if the driver library encounters an error.
//
//*****************************************************************************

#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif


//*****************************************************************************
//
// Main program: configurations, read the array the first time, compute and
// wait for UART commands.
//
//*****************************************************************************
int
main(void)
{

    //
    // Run from the PLL at 120 MHz.
    //
    g_ui32SysClock = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
            SYSCTL_OSC_MAIN | SYSCTL_USE_PLL |
            SYSCTL_CFG_VCO_480), 120000000);


    //
    // Initialize the UART, GPIO and ADC.
    //
    ConfigureUART();
    ConfigureGPIO();
    ConfigureADC();

    //
    // Main-Loop.
    //
    while(1)
    {

        ReadArray();
        Computations();
        UARTHandler();
    }
}
```

Quellcode C.3: configurations.c

```
/*
 * configurations.c
 *
 *  Created on: 03.05.2019
 *      Author: tmehm
 */

#include "include.h"
#include "prototypes.h"

//*****************************************************************************
//
// Configure the ADC and its pins.  This must be called before GPIOPinWrite().
//
//*****************************************************************************
void
ConfigureADC(void){

    //
    // Enable the ADC0 and ADC1 module.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC1);

    //
    // Wait for the ADC0 module to be ready.
    //
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_ADC0))
    {
    }
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_ADC1))
    {
    }

    // ADC0, sample sequencer 0, trigger processor, priority 0
    // ADC1, sample sequencer 1, trigger processor, priority 1
    // ADC2, sample sequencer 2, trigger processor, priority 2
    ADCSequenceConfigure(ADC0_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceConfigure(ADC1_BASE, 1, ADC_TRIGGER_PROCESSOR, 1);
    ADCSequenceConfigure(ADC1_BASE, 2, ADC_TRIGGER_PROCESSOR, 2);

    // Sample sequencer 0, left side of array
    ADCSequenceStepConfigure(ADC0_BASE, 0, 0, ROW_1_L); //sequence 0, step 0
    ADCSequenceStepConfigure(ADC0_BASE, 0, 1, ROW_2_L); //sequence 0, step 1
    ADCSequenceStepConfigure(ADC0_BASE, 0, 2, ROW_3_L); //sequence 0, step 2
    ADCSequenceStepConfigure(ADC0_BASE, 0, 3, ROW_4_L); //sequence 0, step 3
    ADCSequenceStepConfigure(ADC0_BASE, 0, 4, ROW_5_L); //sequence 0, step 4
    ADCSequenceStepConfigure(ADC0_BASE, 0, 5, ROW_6_L); //sequence 0, step 5
    ADCSequenceStepConfigure(ADC0_BASE, 0, 6, ROW_7_L); //sequence 0, step 6
    ADCSequenceStepConfigure(ADC0_BASE, 0, 7, ROW_8_L); //sequence 0, step 7

    // Sample sequencer 1, upper right side of the array
    ADCSequenceStepConfigure(ADC1_BASE, 1, 0, ROW_1_R); //sequence 1, step 0
    ADCSequenceStepConfigure(ADC1_BASE, 1, 1, ROW_2_R); //sequence 1, step 1
    ADCSequenceStepConfigure(ADC1_BASE, 1, 2, ROW_3_R); //sequence 1, step 2
    ADCSequenceStepConfigure(ADC1_BASE, 1, 3, ROW_4_R); //sequence 1, step 3

    // Sample sequencer 1, lower right side of the array
    ADCSequenceStepConfigure(ADC1_BASE, 2, 0, ROW_5_R); //sequence 2, step 0
    ADCSequenceStepConfigure(ADC1_BASE, 2, 1, ROW_6_R); //sequence 2, step 1
    ADCSequenceStepConfigure(ADC1_BASE, 2, 2, ROW_7_R); //sequence 2, step 2
```

```
    ADCSequenceStepConfigure(ADC1_BASE, 2, 3, ROW_8_R | //sequence 2, step 3...
                             ADC_CTL_IE | ADC_CTL_END); //incl. interrupt

    // Enable ADC's
    ADCSequenceEnable(ADC0_BASE, 0);     // ADC0 for sample sequencer 0
    ADCSequenceEnable(ADC1_BASE, 1);     // ADC1 for sample sequencer 1
    ADCSequenceEnable(ADC1_BASE, 2);     // ADC1 for sample sequencer 2

}


//*****************************************************************************
//
// Configure the UART and its pins.  This must be called before UARTprintf().
//
//*****************************************************************************
void
ConfigureUART(void)
{
    //
    // Enable the GPIO Peripheral used by the UART.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //
    // Enable UART0
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    //
    // Configure GPIO Pins for UART mode.
    //
    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //
    // Initialize the UART for console I/O.
    //
    UARTStdioConfig(0, 115200, g_ui32SysClock);
}


//*****************************************************************************
//
// Configure GPIO PM3=D; PM2=C; PM1=B; PM0=A; PL3=nD
//
//*****************************************************************************
void
ConfigureGPIO(void)
{
    //
    // Enable the GPIO M,L peripheral
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL);
    //
    // Wait for the GPIO M,L module to be ready.
    //
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOM))
    {
    }
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOL))
```

```
125         {
            }

            //
            // Set pins as output, SW controlled.
130         //
            GPIOPinTypeGPIOOutput(GPIO_PORTM_BASE, GPIO_PIN_0 | GPIO_PIN_1 |
                                                   GPIO_PIN_2 | GPIO_PIN_3);
            GPIOPinTypeGPIOOutput(GPIO_PORTL_BASE, GPIO_PIN_3);
    }
```

Quellcode C.4: functions.c

```
/*
 * functions.c
 *
 *  Created on: 02.05.2019
 *      Author: tmehm
 */


#include "include.h"
#include "prototypes.h"

//*****************************************************************************
//
// UART handler: reads the UART and sends the answer.
//
//*****************************************************************************
void
UARTHandler(void){

    receive = UARTCharGet(UART0_BASE); // waits until character is received

    while(receive != 48){   // 48 == '0' : leave while-loop (and re-read the array)
        DetermineBaseAddresses(receive);
        TransmitAsChar();
        receive = UARTCharGet(UART0_BASE);
    }
}

//*****************************************************************************
//
// Determine base addresses to value arrays for the TransmitAsChar() function.
//
//
//*****************************************************************************
void
DetermineBaseAddresses(int type){
    switch (type){
    case 49:{
        ucPtr = (uint8_t*) &DiffSinResults[0][0];
        break;
    }
    case 50:{
        ucPtr = (uint8_t*) &DiffCosResults[0][0];
        break;
    }
    case 51:{
        ucPtr = (uint8_t*) &negSinResults[0][0];
        break;
    }
    case 52:{
        ucPtr = (uint8_t*) &posSinResults[0][0];
        break;
    }
    case 53:{
        ucPtr = (uint8_t*) &negCosResults[0][0];
        break;
    }
    case 54:{
        ucPtr = (uint8_t*) &posCosResults[0][0];
        break;
    }
```

```
        case 55:{
            ucPtr = (uint8_t*) &SinOffset[0][0];
            break;
        }
        case 56:{
            ucPtr = (uint8_t*) &CosOffset[0][0];
            break;
        }

        }
}


//*****************************************************************************
//
// Transmit 64 x int16  as 128 x char values
//
//*****************************************************************************
void
TransmitAsChar(void){
    int m = 0;
    for(m = 0; m <= 127; m++){
        UARTCharPut(UART0_BASE, *(ucPtr++));
    }
}

//*****************************************************************************
//
// Set-up time
//
//*****************************************************************************
void
SetUpTime(void){
    SysCtlDelay(g_ui32SysClock / 100 / 1000000);     // (120 MHz / 3) * 1 us = 40
}


//*****************************************************************************
//
// Read whole Array
//
//*****************************************************************************
void
ReadArray(void){

    for (i = 0; i <= 7; i++)            // read Cos-Values ( = first 8 cycles )
    {
        SetAddress();
        SetUpTime();                                    // set-up time
        GetADCValues();                                 // read current values
        // left half
        CosResults[0][7-i] = (int16_t) ADCValues_SS0[0];// assign Values to result
        CosResults[1][7-i] = (int16_t) ADCValues_SS0[1];// array ordered backwards!
        CosResults[2][7-i] = (int16_t) ADCValues_SS0[2];
        CosResults[3][7-i] = (int16_t) ADCValues_SS0[3];
        CosResults[4][7-i] = (int16_t) ADCValues_SS0[4];
        CosResults[5][7-i] = (int16_t) ADCValues_SS0[5];
        CosResults[6][7-i] = (int16_t) ADCValues_SS0[6];
        CosResults[7][7-i] = (int16_t) ADCValues_SS0[7];
        // right half
        CosResults[0][15-i] = (int16_t) ADCValues_SS1[0];
        CosResults[1][15-i] = (int16_t) ADCValues_SS1[1];
        CosResults[2][15-i] = (int16_t) ADCValues_SS1[2];
```

```
125             CosResults[3][15-i] = (int16_t) ADCValues_SS1[3];
                CosResults[4][15-i] = (int16_t) ADCValues_SS2[0];
                CosResults[5][15-i] = (int16_t) ADCValues_SS2[1];
                CosResults[6][15-i] = (int16_t) ADCValues_SS2[2];
                CosResults[7][15-i] = (int16_t) ADCValues_SS2[3];
130
        }

        for (i = 8; i <= 15; i++)          // read Sin-Values ( = second 8 cycles )
        {
135             SetAddress();
                SetUpTime();                                    // set-up time
                GetADCValues();                                 // read current values

                // left half
140             SinResults[0][i-8] = (int16_t) ADCValues_SS0[0];// assign Values to result
                SinResults[1][i-8] = (int16_t) ADCValues_SS0[1];// array ordered forwards!
                SinResults[2][i-8] = (int16_t) ADCValues_SS0[2];
                SinResults[3][i-8] = (int16_t) ADCValues_SS0[3];
                SinResults[4][i-8] = (int16_t) ADCValues_SS0[4];
145             SinResults[5][i-8] = (int16_t) ADCValues_SS0[5];
                SinResults[6][i-8] = (int16_t) ADCValues_SS0[6];
                SinResults[7][i-8] = (int16_t) ADCValues_SS0[7];
                // right half
                SinResults[0][i] = (int16_t) ADCValues_SS1[0];
150             SinResults[1][i] = (int16_t) ADCValues_SS1[1];
                SinResults[2][i] = (int16_t) ADCValues_SS1[2];
                SinResults[3][i] = (int16_t) ADCValues_SS1[3];
                SinResults[4][i] = (int16_t) ADCValues_SS2[0];
                SinResults[5][i] = (int16_t) ADCValues_SS2[1];
155             SinResults[6][i] = (int16_t) ADCValues_SS2[2];
                SinResults[7][i] = (int16_t) ADCValues_SS2[3];
        }
    }

160


    //*****************************************************************************
    //
    // Get ADC Values
165 //
    //*****************************************************************************
    void
    GetADCValues(void){
        //
170     // Trigger the sample sequence.
        //
        ADCProcessorTrigger(ADC0_BASE, 0);
        ADCProcessorTrigger(ADC1_BASE, 1);
        ADCProcessorTrigger(ADC1_BASE, 2);
175     //
        // Wait until the sample sequence has completed.
        //
        /*while(!ADCIntStatus(ADC0_BASE, 0, false))
        {
180     }

        while(!ADCIntStatus(ADC1_BASE, 1, false))
        {
        }
185      */
        while(!ADCIntStatus(ADC1_BASE, 2, false))
        {
```

```
    }

    // Quit interrupt
    ADCIntClear(ADC1_BASE, 2);


    //
    // Read the value from the ADC.
    //
    ADCSequenceDataGet(ADC0_BASE, 0, &ADCValues_SS0[0]);
    ADCSequenceDataGet(ADC1_BASE, 1, &ADCValues_SS1[0]);
    ADCSequenceDataGet(ADC1_BASE, 2, &ADCValues_SS2[0]);

}


//*****************************************************************************
//
// Write address to corresponding GPIO Pins
//
//*****************************************************************************
void
SetAddress(void){
    //
    // write address to the GPIO Pins for MU
    //
    GPIOPinWrite(GPIO_PORTM_BASE, GPIO_PIN_3_DOWNTO_0, i);  // address to Port M
    GPIOPinWrite(GPIO_PORTL_BASE, GPIO_PIN_3, (i ^ GPIO_PIN_3));//invert bit 'D'
}


//*****************************************************************************
//
// Compute differential signal
//
//*****************************************************************************
void
Computations(void){
    int m = 0;
    int n = 0;
    for (m = 0; m <= 7; m++)
    {
        for (n = 0; n <= 7; n++)
        {
            // shift left 1 : multiplication by 2
            // differential: 0-1; 2-3; ... ; 14-15
            negCosResults[m][n] = CosResults[m][(n<<1)];    // 0, 2, 4, ... , 14
            posCosResults[m][n] = CosResults[m][(n<<1)+1];  // 1, 3, 5, ... , 15
            DiffCosResults[m][n] = negCosResults[m][n] - posCosResults[m][n];
            CosOffset[m][n] = (negCosResults[m][n] + posCosResults[m][n]) >> 1;

            negSinResults[m][n] = SinResults[m][(n<<1)];
            posSinResults[m][n] = SinResults[m][(n<<1)+1];
            DiffSinResults[m][n] = negSinResults[m][n] - posSinResults[m][n];
            SinOffset[m][n] = (negSinResults[m][n] + posSinResults[m][n]) >> 1;
        }
    }
}
```

D Matlab-Quellcode

Quellcode D.1: liveplot.m

```
% Visualisierung der Momentanwerte
% 8x8 TMR-Array TDK
% T. Mehm

close all
clear all
clc

% initialisierung der differentiellen Arrays
sind= zeros(8,8);
cosd= zeros(8,8);

% erstelle serielles Objekt und Oeffne es
s = serial('/dev/ttyACM0', 'BaudRate', 115200);
fopen(s)

% Endlosschleife fuer die Darstellung der Momentanwerte
while(1)

    fwrite(s, '0')              % Messwerte aktualisieren

    fwrite(s, '1')              % Differentielle Sin-Werte anfordern
    sind = fread(s,64, 'int16'); % speichere 64 int16-Werte

    fwrite(s, '2')              % Differentielle Cos-Werte anfordern
    cosd = fread(s,64, 'int16'); % abspeichern

    sind = reshape(sind,[8,8])'; % Vektor in 8x8-Matrix umwandeln
    cosd = reshape(cosd,[8,8])'; % und transponieren

    quiver(cosd, sind),axis square % Darstellen der Vektoren

    xlim([0 9])
    ylim([0 9])
    drawnow

end
fclose(s) % Schliessen
```

Selbstständigkeitserklärung

Hiermit versichere ich, Thorbjörn Mehm, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5)APSO-TI-BM ohne fremde Hilfe selbständig 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, 7. Juni 2019