Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Andreas Roloff

Entwicklung einer Messdatenerfassungskarte für einen Höhenballon mit 32bit PIC Controller und Speicherung auf SD-Karte

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Andreas Roloff

Entwicklung einer Messdatenerfassungskarte für einen Höhenballon mit 32bit PIC Controller und Speicherung auf SD-Karte

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

Betreuender Prüfer : Prof. Dr. rer. nat. Jochen Schneider
Zweitgutachter : Prof. Dr.Ing. Karl-Ragmar Riemschneider

Abgegeben am 14. September 2011

Andreas Roloff

Thema der Bachelorarbeit

Entwicklung einer Messdatenerfassungskarte für einen Höhenballon mit 32bit PIC Controller und Speicherung auf SD-Karte

Stichworte

Datenerfassung, Startosphärenballon, CTA, Microcontroller, DMA-Controller, SD-Karte

Kurzzusammenfassung

Diese Arbeit beschreibt die Entwicklung einer Messdatenerfassungskarte für Höhenballons, die für die Turbulenzmessung in der Stratosphäre eingesetzt werden soll. Bei der Messkarte handelt es sich um ein Mikrocontroller-System basierend auf dem PIC32-Kern. Das System misst die analogen Signale von CTA/CCA-Sensoren und empfängt Daten von einem Initialsensor und von einem Magnetometer. Es misst noch weitere Umgebungsparameter und überwacht die eigene Versorgungsspannung und die Versorgung der CTAs. Die Messkarte speichert die gemessen Daten auf SD-Karte und kommuniziert mit einem Personal Computer.

Andreas Roloff

Title of the paper

Development of a data acquisition system for a stratospheric balloon using a 32bit PIC controller and storage on SD card

Keywords

data acquisition, startospheric balloon, CTA, microcontroller, DMA-Controller, SD-Card

Abstract

This thesis describes the development of a data acquisition system for turbulence measurements on stratospheric balloons. The new device is an embedded microcontroller system based on the PIC32 core. The system measures the analog signals of CTA / CCA sensors as well as the attitude of the gondola using an initial sensor and a magnetometer. Furthermore environmental parameters are recorded and the power supply of the system and the CTAs is monitored. The measured data are saved on a SD card. Communication with a personal computer is also implemented.

Inhaltsverzeichnis

Tabellenverzeichnis

Bildverzeichnis

1. Einleitung

Diese Bachelorarbeit wurde in Zusammenarbeit mit dem Leibniz-Institut für Atomsphärenphysik (IAP) in Kühlungsborn erstellt. Die Hauptaufgabe des IAP ist die Erforschung der mittleren Atmosphäre im Höhenbereich von 10 bis 100 km. Dies erfolgt unter Berücksichtigung der dynamischen Wechselwirkungen zwischen der unteren und der mittleren Atmosphäre.

Eines ihrer Forschungsprojekte ist die ballongetragene in-situ Messung von kleinskaligen Windturbulenzen in der Stratosphäre. Zum Verständnis der Energie- und der Impulsspurenstoffbilanz in der Atmosphäre sind Turbulenzen und Schwerewellen von großer Bedeutung. Die Stratosphäre ist ein kaum erforschtes Gebiet in Bezug auf die Turbulenzmessung. Zur Zeit gibt es kaum geeignete Messverfahren, um kleinskalige Turbulenzen in der Stratosphäre zu messen. Deshalb wurde beim IAP ein Messverfahren entwickelt, das die turbulenten Strukturen im horizontalen Windfeld vom Boden bis in ca. 35 km Höhe mit Hilfe von Höhenballons nachweisen soll.

Die Elektronik für das entwickelte Messverfahren bestand vor Beginn der Arbeit aus zwei separaten Systemen, einem zur Messung der Windgeschwindigkeit und einem weiteren zur Bestimmung der Lage der Gondel und der Umgebungsbedingungen. Das System, das die Windgeschwindigkeiten misst, nimmt diese Daten ohne eine Zeitinformation auf, und somit ist es schwer, diese Daten mit den Informationen des Systems für Umgebungsbedingungen zu verknüpfen. Es musste immer vor einem Ballonstart die Startzeit notiert werden oder beim Auswerten versucht werden, den Start oder das Platz des Ballons in den gemessenen Daten zu finden. Um ein Messsystem zu haben, das beide Informationen mit einer Zeitinformation aufnimmt, wurde ein neues System benötigt. Mit der Abteilung Optische Sondierung wurde deshalb eine Messdatenerfassungskarte entwickelt, die kleinskalige Windturbulenzen in der Stratosphäre mit Hilfe eines Höhenballons aufnehmen soll.

Im Rahmen der Neuentwicklung der Messdatenerfassungskarte nimmt der Autor zusammen mit einem Studenten-Team des IAP an dem BEXUS-Studentenprogramm[1] teil. Das BEXUS-Studentenprogramm wird von dem Deutschen Zentrum für Luft- und Raumfahrt gesponsert und ermöglicht Studenten, Experimente auf einem großen Stratosphärenballon fliegen zu lassen. Die zu entwickelnde Platine soll bei der BEXUS-Kampagne im September 2011 eingesetzt werden.

1 Weitere Informationen können unter www.rexusbexus.net gefunden werden.

2. Aufgabenstellung

Für das Messsystem, das die kleinskaligen Windturbulenzen in der Stratosphäre misst, soll eine mikrocontrollergesteuerte Elektronik zur Datenaufnahme und Zustandsüberwachung entwickelt werden. Die neue Elektronik soll die bisherige Datenaufnahme, die über zwei unabhängige Systeme erfolgte, ersetzen. Eines dieser Systeme war für die Aufnahme des Windsignals [Reimesch-Box] und das andere für die Lagemessung und Überwachung [Housekeeping] verantwortlich.

An die neue Elektronik werden folgende Anforderungen gestellt:

- Die als Spannungssignal vorliegenden Winddaten sind mit einer Abtastrate von 8 kHz aufzunehmen.
- Die Informationen über die Lage der Gondel sind mit 100 Hz zu messen.
- Jede Sekunde sollen der Luftdruck und die Temperatur ermittelt werden.
- Die Strom- und Spannungsversorgung ist zu überwachen.
- Die ermittelten Daten sind auf eine SD-Karte zu speichern und an ein Telemetriesystem zu übergeben.
- Zur Konfigurierung bzw. Ausgabe von Statusinformationen ist eine Verbindung zu einem Computer erforderlich.

Das Bild 2.1 auf Seite 11 zeigt die Anforderungen an die neue Elektronik und die vorgegebenen Sensoren in einem groben Blockschaltbild. Die neue Messdatenerfassungskarte soll bis zu 3 Windkanäle abtasten und die Spannungsversorgung für die Windsensoren (MiniCTA) schalten können.

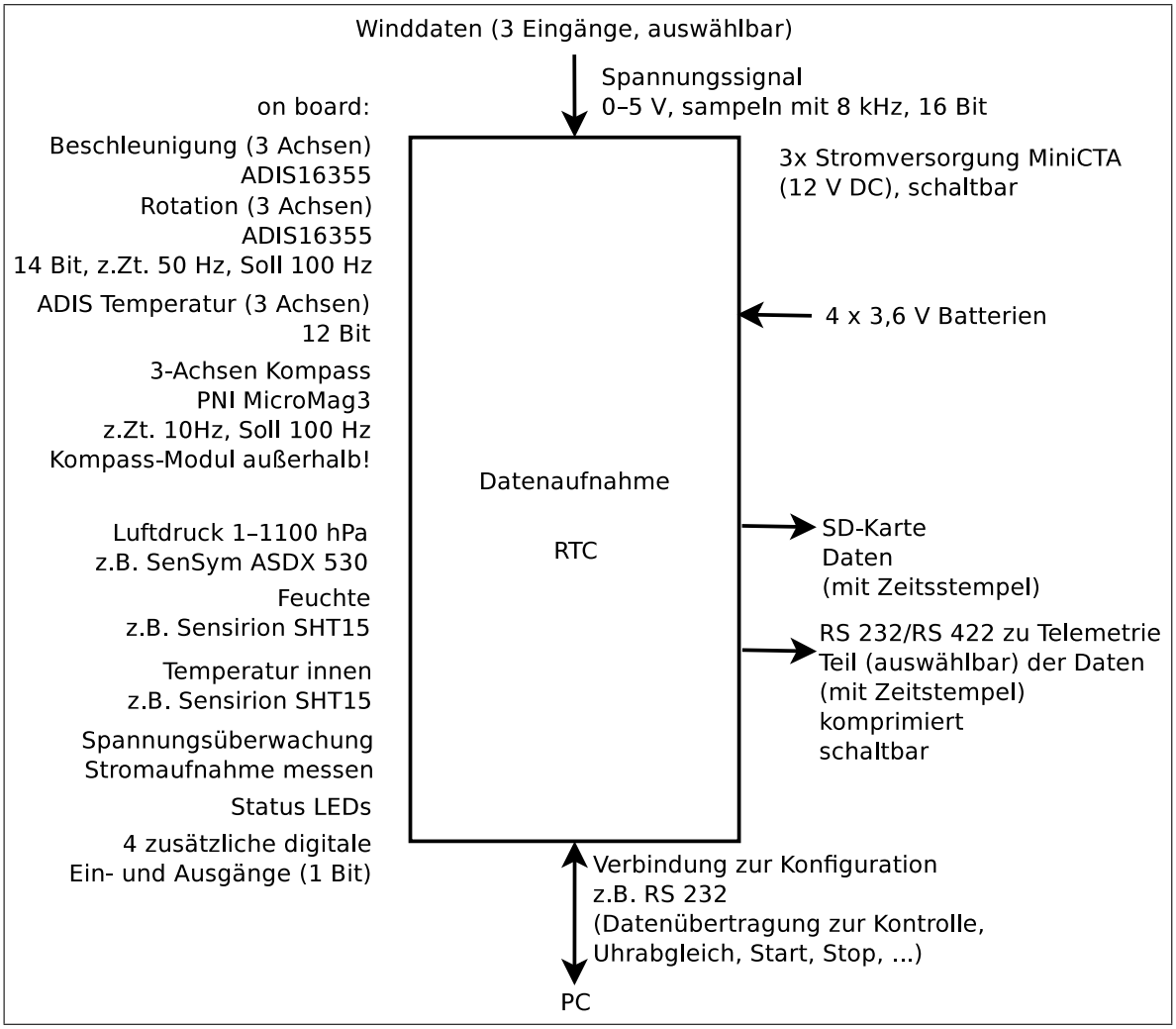


Bild 2.1.: Grobes Blockschaltbild

3. Umsetzung

Zur Erfüllung der Aufgabenstellung, der Entwicklung einer Messdatenerfassungskarte, musste das System komplett neu entwickelt werden. Schaltung und Platinenlayout der neuen Hardware wurden mit der Leiterplattendesign Software Eagle entworfen. Die Entscheidung fiel auf dieses Designprogramm, weil es das Entwerfen des Schaltplans und das Routen der Platine in einem Editor vereint und eine einfache Bedienoberfläche hat. Das Entwerfen von Schaltungen ist in Eagle recht einfach, da für die meisten elektrischen Bauteile die Schaltzeichen in Bibliotheken zur Verfügung stehen. Zusätzlich können schon mit der Freeware-Version von Eagle Schaltungen mit 2 Lagen der Größe einer halben Eurokarte (80 x 100) erzeugt werden. Diese Version reichte für den Anfang aus, aber für die Fertigstellung des endgültigen Layouts wurde doch eine höherwertige Version gebraucht. Der Schaltplan und das Platinlayout sind im Anhang A auf Seite 80 zu finden.

Da der Mikrocontroller von Microchip aus der PIC32MX-Controllerserie mit einer hohen Taktfrequenz von bis zu 80 MHz betrieben werden kann und über die nötigen Peripherie-Module, z. B. SPI, UART und einen Parallelport, verfügt, um die geforderten Sensoren anzusteuern, wurde dieser verwendet. Die detaillierte Beschreibung des Mikrocotrollers kann dem Abschnitt 3.2.1 „*Mikrocontroller*“ auf Seite 28 entnommen werden.

Zur Programmierung des verwendeten Mikrocontrollers wird die Entwicklungsumgebung MPLAB benutzt, die Microchip bereit stellt. Der Vorteil von MPLAB ist, dass sie auf die PIC-Controller von Microchip angepasst ist. Die Entwicklungsumgebung besitzt neben dem Editor, in dem die Software geschrieben wird, noch einen Projekt-Manager, mit dem Projekte angelegt werden können und die Umsetzung der Software auf mehrere Dateien erfolgen kann. Zum besseren Verständnis der Software wurde sie in mehrere Dateien aufgeteilt, in den jeweils ähnliche Bereiche zusammengefasst sind.

In der Entwurfs- und Layoutphase der neuen Hardware wurden die ersten Teile der Software mit Hilfe eines Starter-Kits, eines zusätzlichen I/O-Boards und eines SD-Karten-Adapter-Boards von Microchip programmiert. Das Bild 3.1 auf Seite 13 zeigt das Entwicklungsboard. Hierbei wurde darauf geachtet, dass der Controller des Starter-Kits die Anforderungen der Messkarte erfüllt. Ein weiterer Vorteil von MPLAB besteht darin, dass die Software der späteren Messkarte mittels eines Adapters, dem Pickit3, aus der Entwicklungsumgebung heraus in den Controller geladen werden kann. Es wird für das Brennen des Controllers keine weitere Software benötigt.

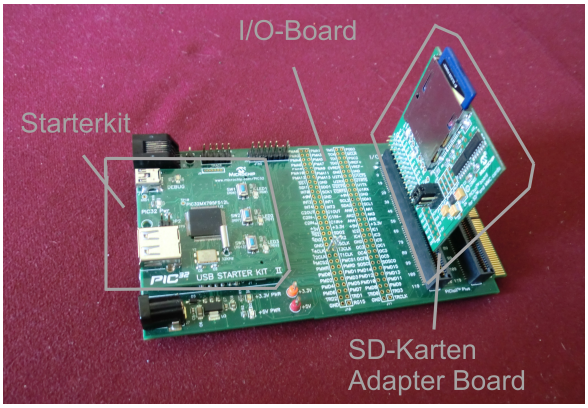


Bild 3.1.: Starterkit mit I/O-Board und SD-Karten-Adapter-Board

Für die Speicherung der gemessenen Daten wird ein Treiber, der von Microchip zur Verfügung gestellt wird, verwendet. Dabei wird auf eine spezielle Dateisystemstruktur verzichtet, und nur ein große Datei auf der SD-Karte erzeugt, um dann sektorenweise die aufgenommen Daten in die Datei zu schreiben. Diese Variante wurde gewählt, da die Speicherung über ein Dateisystem zu viel Zeit in Anspruch genommen hätte und es somit sonst zu Datenverlusten gekommen wäre. Beim sektorweisen Beschreiben muss nur darauf geachtet werden, dass nicht in die Sektorbereiche geschrieben wird, in denen auf der SD-Karte die Dateisystem-Informationen abgelegt sind.

In dem Abschnitt 3.1 „*Sensoren*“ ab Seite 13 werden die in der Aufgabenstellung aufgeführten Sensoren näher beschrieben. Die Hardwarekomponenten, die benötigt werden, um die Messkarte umzusetzen, sind in dem Abschnitt 3.2 „*Hardware*“ ab Seite 26 näher aufgeführt. Die Software-Module werden in dem Abschnitt 3.3 „*Software*“ ab Seite 49 beschrieben. Im Abschnitt 4 „*Test*“ ab Seite 73 sind die Tests, die durchgeführt wurden, zu finden.

3.1. Sensoren

In diesem Kapitel sollen die aus der Aufgabenstellung vorgegebenen Sensoren (siehe Bild 2.1 auf Seite 11) erläutert werden. Die Hitzdraht-Sensoren werden zur präzisen Messung der Windgeschwindigkeiten bzw. der Temperaturen zwecks Bestimmung der feinskaligen Windturbulenzen benutzt. Die Bestimmung der Lage der Gondel erfolgt durch ein Magnetometer und einen Inertialsensor, der sowohl die Rotation als auch die Beschleunigung misst. Die Lageinformationen der Gondel sind wichtig, um bei der späteren Auswertung der Messwerte den induzierten Wind, der durch die Bewegung der Gondel entsteht, zu kennen. Mit Hilfe des Drucksensors kann auf die Höhe des Ballons geschlossen werden.

Der Temperatur- und Feuchtigkeitssensor soll den Zustand in der Box messen, um eine Erklärung für eventuelle Fehler und Datenverluste zu haben, z. B. wenn die Elektronik zu heiß wird.

3.1.1. Hitzdraht-Sensoren

Die Hitzdraht-Sensoren, auch Hitzdrahtanemometer genannt, werden benutzt, um die Strömungsgeschwindigkeit von Fluiden zeitlich hochauflösend zu messen.

Bei den Hitzdrahtanemometern wird ein elektrisch beheizter sehr dünner Draht verwendet, der einen Durchmesser von 1 µm bis 10 µm aufweist. Die Länge des verwendeten Drahts sollte das 200-fache des Durchmessers betragen, um Randeinflüsse gering zu halten. Diese speziellen Drähte bestehen meist aus Nickel, Platin oder Wolfram. Je dünner diese Drähte sind, desto höhere Frequenzen können damit erfasst werden, allerdings haben sie dann auch eine höhere mechanische Empfindlichkeit. Der Draht wird zwischen zwei etwas dickeren Haltespitzen, den Prongs, gespannt und am oberen Ende festgeschweißt. Diese Prongs sind außerdem zur mechanischen Stabilisierung und zur elektrischen Isolation in einem Keramikkörper befestigt. Es gibt Eindraht-, Zweidraht- oder Dreidrahtsensoren, wie im Bild 3.2 auf Seite 14 dargestellt, und jeder Draht ist in der Lage, eine Komponente der Strömungsgeschwindigkeit aufzunehmen.[1]

Bild 3.2.: Hitzdraht-Sensoren [2]

Von diesen Sensoren gibt es zwei Arten:

- Konstant-Strom-Anemometer (Constant Current Anemometer, CCA)
- Konstant-Temperatur-Anemometer (Constant Temperature Anemometer, CTA)

Konstant-Strom-Anemometer (CCA Constant Current Anemometer)
Die CCA-Sensoren arbeiten wie ein Widerstandsthermometer. Mit ihnen wird die Temperatur gemessen. Damit dies möglich ist, wird der Sensor mit einem konstanten Strom betrieben. Die gemessene Spannung der Sensoren ist proportional zur Temperatur. Da der dünne

Draht (ca. 1 µm) eine geringe Wärmekapazität hat, nehmen diese Sensoren schnell die Temperatur der Umgebung an, was es ermöglicht, sie mit einer hohen Frequenz (mehrere kHZ) abzutasten. Die in diesem Projekt eingesetzten CCA-Sensoren werden von TSI Corporation hergestellt.

Konstant-Temperatur-Anemometer (CTA Constant Temperature Anemometer)
Zur Bestimmung der Strömungsgeschwindigkeit wird die abgegebene elektrische Leistung der CTA-Sensoren gemessen. Dies geschieht durch eine Widerstandsbrückenschaltung (Wheatstone-Brückenschaltung). Bei dieser wird der Widerstand des Drahts konstant gehalten und dadurch auch die Temperatur. Durch die Strömung wird der Draht des CTA-Sensors abgekühlt, was dazu führt, dass die Wheatstone-Widerstandsbrücke nicht mehr abgeglichen ist. Ein Regelverstärker hält die ganze Zeit die Brücke durch Kontrolle des Stroms im abgeglichenen Zustand. Die Spannungsdifferenz, die der Regelverstärker kompensieren muss, ist ein Maß für die Strömungsgeschwindigkeit. Die CTA-Messmethode ist die am häufigsten eingesetzte bei den Hitzdraht-Sensoren. In diesem Projekt werden CTA-Sensoren der Firma Dantec Dynamics benutzt. Sie bestehen aus einem platinummantelten Wolframdraht mit einer Länge von 1,25 mm und einer Dicke von 5 µm.

3.1.2. Rotations- und Beschleunigungssensor

Die Rotationssensoren, auch Drehratensensoren genannt, messen die Rotationsgeschwindigkeit eines Körpers. Um welchen Winkel sich ein Körper innerhalb einer Zeit gedreht hat, kann durch Integration bestimmt werden. Mit einem Beschleunigungssensor kann die Geschwindigkeitszunahme und -abnahme gemessen werden. Dies geschieht durch die Trägheitskraft, die auf eine Testmasse wirkt. [3, 4]

In diesem Projekt wird ein Inertialsensor, ein Rotations- und Beschleunigungssensor, aus der ADIS-Serie der Firma Analog Devices eingesetzt, das Blockschaltbild ist im Bild 3.3 auf Seite 16 dargestellt. Er beinhaltet drei orthogonal angeordnete Beschleunigungssensoren zum Messen der Beschleunigung in der X-, Y- und Z-Achse und drei orthogonale Rotationssensoren zur Bestimmung der Winkelgeschwindigkeit der drei Achsen. Das Ganze ist in einem Mikroelektronischen-Mechanischen-System-Bauteil (MEMS) untergebracht. Der ADIS-Sensor hat sich schon in früheren Ballonflügen bewährt. [5]

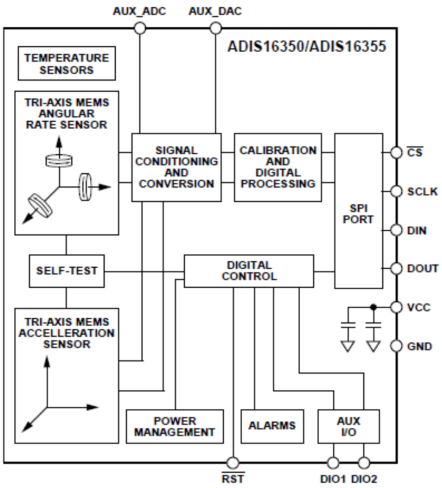


Bild 3.3.: Funktion-Blockschaltbild ADIS-Sensor[12, Seite 1]

Für die Anbindung des ADIS-Sensors an den Mikrocontroller ist eine 5 V Versorgungsspannung und eine SPI-Schnittstelle mit vier Leitungen nötig. Diese vier Leitungen sind: Chip Select ($\overline{CS}$), Serial Clock (SCLK), Data Input (DIN) und Data Output (DOUT). Zum Freigeben der SPI-Schnittstelle des ADIS-Sensors wird die Chip Select-Leitung benötigt, und sie muss einen Low-Pegel besitzen. Wenn die $\overline{CS}$-Leitung einen High-Pegel hat, ist die SPI-Schnittstelle des ADIS-Sensors deaktiviert. Dies hat zur Folge, dass die DOUT-Leitung auf einen High-Pegel gesetzt wird und die Signale auf den DIN- und SCLK-Leitungen keinen Einfluss auf den Sensor haben. Ein kompletter Datenrahmen (SPI-Event) benötigt 16 Takte und während der Übermittlung muss die SPI-Schnittstelle durchgehend freigegeben sein. In dieser Arbeit wird auf das Aktivieren der Schnittstelle für jeden einzelnen SPI-Event verzichtet, stattdessen wird die SPI-Schnittstelle für die gesamte Messung durchgehend freigegeben. Die SPI-Schnittstelle des Rotations- und Beschleunigungssensors arbeitet im Full Duplex-Betrieb, das heißt, der Sensor kann gleichzeitig in einem Datenrahmen 16 Bit empfangen und senden. Damit besteht die Möglichkeit, mit dem Senden des nächsten Lesebefehls gleichzeitig die Daten des vorherigen Lesebefehls zu empfangen.

Alle Ausgabe- und Benutzereinstellungsfunktionen werden mittels einer einfachen Registerstruktur erreicht. Jedes Register hat eine Größe von 16 Bit und die Daten in den Registern sind auf das obere (D8 bis D15) und das untere Byte (D0 bis D7) aufgeteilt. Jedes dieser Bytes hat seine eigene Adresse, die jeweils 6 Bit lang ist. Im Nachfolgenden wird beschrieben, wie ein Schreib- und Lesevorgang auf die Ausgabe- oder Steuerregister des Rotations- und Beschleunigungssensors zu erfolgen hat.

Ein typischer Datenrahmen zum Schreiben eines Befehls an die Steuerregister beginnt mit einer Eins, gefolgt von einer Null. Diese beiden Bits müssen zuerst über die DIN-Leitung zum Sensor übermittelt werden, dann die 6 Bit-Adresse des Registers, das man ansprechen

möchte, und zuletzt der 8 Bit-Datenbefehl. Beim Schreibvorgang wird immer nur ein Datenbyte übertragen. Um das gesamte 16 Bit-Register zu beschreiben, müssen zwei Schreibvorgänge erfolgen. Für einen gültigen Schreibvorgang sind 16 Takte erforderlich. Das Bild 3.4 auf Seite 17 zeigt, wie ein Datenrahmen für einen einfachen Schreibvorgang aussieht.

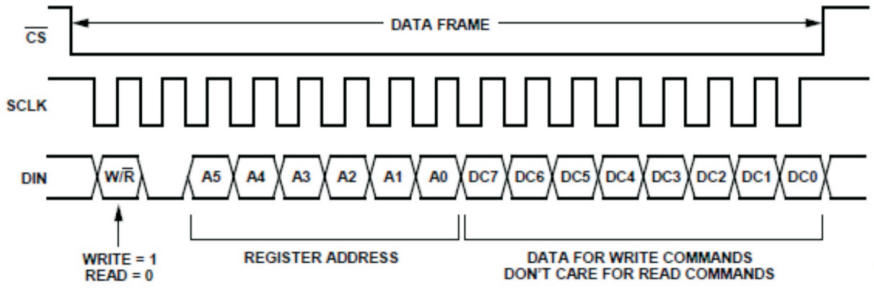


Bild 3.4.: Datenrahmen für einen Schreibvorgang des Steuerregisters des ADIS-Sensors [12, Seite 13]

Der Datenrahmen zum Auslesen des Inhalts aus den Registern ist ähnlich aufgebaut wie der des Schreibvorgangs im Bild 3.4. Der einzige Unterschied besteht darin, dass die ersten beiden Bits im Gegensatz zum Schreibvorgang eine Null haben. Auch hier folgt die 6 Bit-Adresse, mit der das jeweilige Register angesprochen wird. Wie anfangs schon erwähnt hat jedes Register eine obere und eine untere Adresse. Zum Auslesen des 16 Bit-Inhalts wird nur eine der beiden Adressen benötigt. Beim Lesevorgang haben die 8 Datenbits vom Schreibvorgang keine Bedeutung und können mit beliebigen Werten „Don't Care“ beschaltet werden. Mit dem nächsten gesendeten Datenrahmen wird der 16 Bit-Inhalt des Registers über die DOUT-Leitung an den Controller übertragen. Somit werden für einen einfachen Lesevorgang immer zwei separate Datenrahmen benötigt. Durch den Full Duplex-Betrieb kann der Kommunikationsaufwand mit dem Sensor minimiert werden, da nur ein zusätzlicher Datenrahmen gebraucht wird, um kontinuierlich Daten zu erhalten. Das Bild 3.5 auf Seite 18 zeigt einen Lesevorgang.

In der Tabelle 3.1 auf Seite 19 sind die möglichen Ausgaberegister des Rotations- und Beschleunigungssensors und die jeweiligen Adressen der Register und deren Funktionen aufgeführt. Außerdem gibt die Tabelle Auskunft über das Datenformat und die Größe der Daten in dem Ausgaberegister. Die 16 Bit-Ausgabedaten in den Registern besitzen die folgende Struktur: Das erste Bit ist das neue Datenflag (ND), es folgt das Error/Alarm-Flag (EA) und zuletzt kommen die restlichen 14 Datenbits. Wenn die Ausgangsdaten nur 12 Bit groß sind, werden die oberen beiden Bits (Bit 13 und Bit 12) nicht benutzt. Wie der Name schon sagt, wird mit dem ND-Flag angezeigt, dass noch neue Daten in dem Ausgaberegister vorhanden

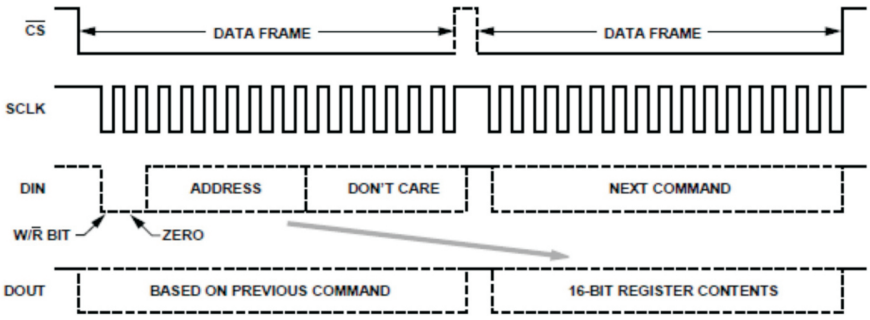


Bild 3.5.: Datenrahmen für einen Lesevorgang des Ausgaberegisters des ADIS-Sensors [12, Seite 13]

sind. Nach dem Auslesen des Ausgaberegisters wird dieses Flag von der Hardware des Sensors gelöscht und nach dem nächsten internen Abtasten erneut gesetzt. Das EA-Flag signalisiert, dass ein Fehler aufgetreten ist. Um herauszufinden, um was für einen Fehler es sich handelt, muss das Statusregister des Sensors ausgelesen werden. In dieser Arbeit werden alle Ausgaberegister außer dem zusätzlichen analogen Eingangsregister ausgelesen, und zum Ansprechen der Register wird immer die untere Adresse der Register genommen.

Name	Funktion/Messung	Adressen	Datenlänge	Datenformat
SUPPLY_OUT	Versorgungs-spannung	0x03, 0x02	12 Bit	Binär
XGYRO_OUT	X-Achse Rotations-ausgabe	0x05, 0x04	14 Bit	Zweierkompliment
YGYRO_OUT	Y-Achse Rotations-ausgabe	0x07, 0x06	14 Bit	Zweierkompliment
ZGYRO_OUT	Z-Achse Rotations-ausgabe	0x09, 0x08	14 Bit	Zweierkompliment
XACCL_OUT	X-Achse Beschleu-nigungsausgabe	0x0B, 0x0A	14 Bit	Zweierkompliment
YACCL_OUT	Y-Achse Beschleu-nigungsausgabe	0x0D, 0x0C	14 Bit	Zweierkompliment
ZACCL_OUT	Z-Achse Beschleu-nigungsausgabe	0x0F, 0x0E	14 Bit	Zweierkompliment
XTEMP_OUT	X-Achse Rotations-sensor Temperatur	0x11, 0x10	12 Bit	Zweierkompliment
YTEMP_OUT	Y-Achse Rotations-sensor Temperatur	0x13, 0x12	12 Bit	Zweierkompliment
ZTEMP_OUT	Z-Achse Rotations-sensor Temperatur	0x15, 0x14	12 Bit	Zweierkompliment
AUX_ADC	Zusätzlicher Analog Eingang	0x17,0x16	12 Bit	Binär

Tabelle 3.1.: Information zu den Ausgaberegistern [12, Seite 14]

3.1.3. Magnetometer

Mit dem Magnetometer können magnetische Flussdichten gemessen werden. Die magnetische Flussdichte wird in der Einheit Telsa (T) angegeben. Der Magnetometer wird als Kompass benutzt. [6]

Die Kommunikation mit dem Magnetometer (MicroMag3) erfolgt mittels einer der vorhandenen SPI-Schnittstellen des Mikrocontrollers. Die Bezeichnungen der folgenden Leitungen entsprechen denen bei MicroMag3. Dazu zählen einmal die Master Out Slave In- (MOSI), die Serial Clock (SCLK) und die Master In Slave Out- (MISO) Leitungen. Zusätzlich werden noch die Leitungen Slave Select Line (SSNOT), die Reset- (RESET) und die Data-Ready- (DRDY) Leitung benötigt.

Mittels der **MOSI**-Leitung werden die Daten von dem Master (PIC32MX) zu dem Slave (MicroMag3) gesendet. Die Daten werden mit dem höchstwertigen Bit zuerst übertragen. Der MicroMag3 akzeptiert nur Daten auf der MOSI-Leitung, wenn seine SPI-Schnittstelle freigegeben ist, und dies erfolgt durch einen Low-Pegel auf der SSNOT-Leitung. Die gesendeten Daten müssen 100 ns vor und 100 ns nach der steigenden Flanke des Taktes einen konstanten Wert haben, um als gültige Daten anerkannt zu werden. Neue Daten haben sich mit der fallenden Flanke zu ändern.

Die **SCLK**-Leitung wird zur Synchronisation der Daten in beide Richtungen benutzt, also auf der MOSI- und der MISO-Leitung. Der Takt wird von dem Master (PIC32MX) erzeugt und kann maximal 1 MHz betragen. Zur Übermittlung von einem Byte werden 8 Takte benötigt. Der Mikrocontroller erfasst die Daten bei der steigenden Flanke des Taktes.

Mit Hilfe der **MISO**-Leitung werden die Daten von dem MicroMag3 zum Mikrocontroller gesendet. Wie schon bei der MOSI-Leitung werden auch hier die Daten mit dem höchstwertigen Bit zuerst übertragen. Wenn die SPI-Schnittstelle des MicroMag3 nicht aktiviert ist (SSNOT-Leitung High-pegel), hat die MISO-Leitung einen High-Pegel.

Die **SSNOT**-Leitung hat die Aufgabe, den MicroMag3 als aktives Slave-Gerät auszuwählen. Die Slave Select Line muss während der Übertragung der Daten einen Low-Pegel haben, unabhängig davon, ob es sich um das Kommando-Byte oder den Messwert handelt. Zusätzlich gibt es noch die zwei Hardware-Handshaking-Leitungen. Diese Leitungen sind einmal die Reset- und zum anderen die DRDY-Leitung.

Die **Reset**-Leitung hat normalerweise immer einen Low-Pegel. Um einen gültigen Reset-Puls zu senden, muss diese Leitung von Low zu High und wieder zu Low wechseln.

Die **Data Ready**-Leitung hat die Aufgabe zu signalisieren, dass der Sensor mit seiner Messung fertig ist und die Daten nun abgeholt werden können. Nachdem ein Reset-Puls übertragen wurde, hat die DRDY-Leitung einen Low-Pegel und geht, wie schon erwähnt, nach Beendigung der Messung wieder auf High-Pegel. Um auch wirklich neue und gültige Daten

vom Sensor zu erhalten, sollte immer die DRDY-Leitung abgefragt werden. Wenn dies aus Mangel an I/O-Ports nicht möglich sein sollte, kann so lange gewartet werden, wie in der Tabelle 3.2 auf Seite 21 angegeben ist. Die Zeiten in der Tabelle 3.2 geben an, wie viel Zeit der MicroMag3 vom letzten Takt des übertragenden Kommandos bis zum Setzen auf High der Data Ready-Leitung bei der ausgewählten Periode benötigt.

Period Select	Maximum Delay
/32	500 µs
/64	1,0 ms
/128	2,0 ms
/256	4,0 ms
/512	7,5 ms
/1024	15 ms
/2048	35,5 ms
/4096	60 ms

Tabelle 3.2.: Delay-Zeiten für die Beendigung einer MicroMag3-Messung [16, Seite 5]

Der normale Ablauf einer Messung wird in den folgenden Schritten dargestellt.

1. Die SSNOT-Leitung muss während der gesamten Messung einen Low-Pegel haben.
2. Vor jeder Messung muss ein Reset-Puls erfolgen.
3. Übermittlung des Kommando-Bytes über die MOSI-Leitung. Nach den ersten 8 Takten erkennt der Sensor das Kommando.
4. Der MicroMag3 führt nun die Messung durch.
5. Nach Beendigung der Messung wird die DRDY-Leitung auf High-Pegel gesetzt. Anschließend wird mit den nächsten 16 Takten der gemessene Wert über die MISO-Leitung an den Master gesendet.

Wenn die Kommunikation nach einer Messung beendet werden soll, wird einfach die SSNOT-Leitung auf High-Pegel gesetzt. Sollen jedoch noch weitere Messungen durchgeführt werden, müssen die Schritte ab Punkt 2 wiederholt werden. Das Bild 3.6 auf Seite 22 zeigt das Timing-Diagramm zu dem gerade beschriebenen Ablauf. überarbeiten Die Syntax des schon mehrmals erwähnten Kommando-Bytes kann der Tabelle 3.3 auf Seite 22 entnommen werden.

Für diese Arbeit spielen nur die AS0- und AS1- sowie die PS0- bis PS2-Bits eine Rolle. Die anderen Bits, die noch in dem Kommando-Byte gesetzt werden können, haben im normalen Betrieb keine Bedeutung. Sie sind für den Debug-Modus des MicroMag3 bestimmt. Mit Hilfe der AS0- und AS1-Bits kann die Achse eingestellt werden, die als nächste gemessen

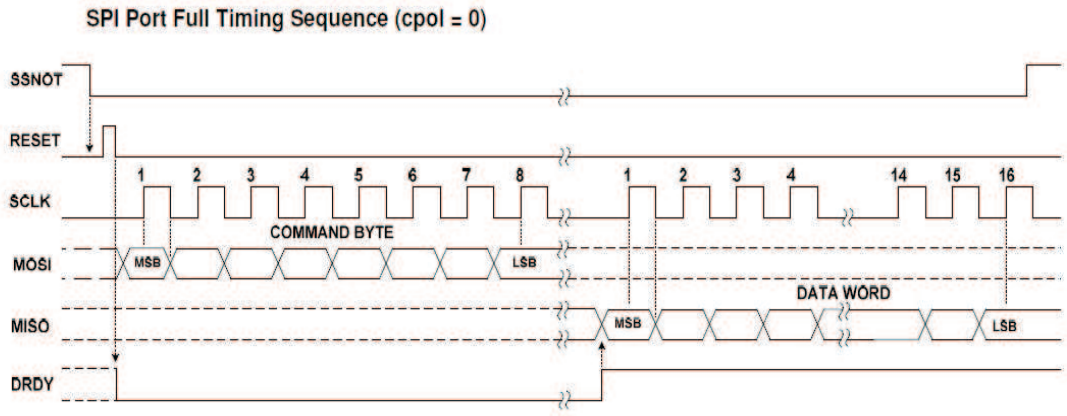


Bild 3.6.: Timing-Diagramm der SPI-Schnittstelle des MicroMag3 [16, Seite 7]

Position	7	6	5	4	3	2	1	0
Bit	DHST	PS2	PS1	PS0	ODIR	MOT	AS1	AS0
Default-Wert	0	0	0	0	0	0	0	0

Tabelle 3.3.: Kommando-Byte des MicroMag3 [16, Seite 9]

werden soll. Die Tabelle 3.4 auf Seite 22 zeigt, welche Bits wie zu setzen sind, um die entsprechende Achse auszuwählen. Der MicroMag3 beinhaltet den PNI-11096, einen Chip mit

Funktion	AS1	AS0
X-Achse	0	1
Y-Achse	1	0
Z-Achse	1	1

Tabelle 3.4.: MicroMag3-Achsenauswahl [16, Seite 9]

einer anwendungsspezifischen integrierten Schaltung (ASIC). Dieser PNI-11096 besitzt die notwendige Schaltung, um die drei magnetisch-induktiven-Sensoren von der Firma PNI-Corp zu benutzen. Der Sensor von der PNI-Corp ändert seine Induktivität, wenn sich das Magnetfeld, das sich parallel zum Sensor befindet, ändert. Zum Messen wird der Sensor in einem RL-Schwingkreis geschaltet. Hierbei wird der eine Anschluss an Masse geschaltet und der andere wird durch den Schwingkreis geladen und entladen. Der PNI-11096 misst dann mittels eines 16 Bit Zählers die Zeit, die es braucht, um eine gewisse Anzahl an Schwingungen durchzuführen. Nachdem das erfolgreich war, werden die Anschlüsse gewechselt und der PNI misst noch mal die Zeit. Abschließend wird die Differenz zwischen den beiden Zeiten ermittelt und daraus dann das aktuelle Magnetfeld bestimmt. Mit den Bits PS0 bis PS2 kann

eingestellt werden, wie lange der ASIC messen soll. Je länger die Messperiode ist, desto genauer kann der PNI-11096 das Magnetfeld bestimmen. Wenn die größtmögliche Messperiode eingestellt wurde, kann es bei einem starken Magnetfeld dazu kommen, dass der 16 Bit-vorzeichenbehaftete Zähler überläuft, deshalb sollte die maximale Messperiode auf 2048 gesetzt werden. Die Tabelle 3.5 auf Seite 23 zeigt, wie die PS0- bis PS2-Bits für den gewünschten Teiler gesetzt werden müssen. [15]

PS2	PS1	PS0	Teiler
0	0	0	/32
0	0	1	/64
0	1	0	/128
0	1	1	/256
1	0	0	/512
1	0	1	/1024
1	1	0	/2048
1	1	1	/4096

Tabelle 3.5.: Bit-Einstellung für die Messperiode [16, Seite 10]

Die Daten des Magnetometers sollen mit 100 Hz abgeholt werden, somit können von den acht möglichen Teilern nur drei überhaupt benutzt werden, da die anderen nicht rechtzeitig mit der Messung aller drei Achsen fertig werden. In dieser Arbeit wurde sich für den ersten Teiler entschieden, um möglichst schnell die Daten zu erhalten und damit wieder Zeit für andere Aufgaben zu haben.

3.1.4. Temperatur- und Feuchtigkeitssensor

Der Temperatur- und Feuchtigkeitssensor, der SHT, hat ein serielles Interface, das mit zwei Leitungen als Anbindung an den Mikrocontroller auskommt. Diese Leitungen sind die Taktleitung (SCK) und die Datenleitung (DATA). Die Taktleitung ist für die Synchronisation der Kommunikation mit dem Mikrocontroller und die DATA-Leitung für den Datentransfer zwischen dem Controller und dem Sensor zuständig. Da das Protokoll, mit dem der Sensor kommuniziert, und das I^2C-Protokoll sich zwar stark ähneln, aber leider nicht identisch sind, wurde ein Timer Interrupt benutzt, der die Signalverläufe auf den beiden Portleitungen so setzt, wie der Sensor sie für sein Protokoll benötigt. Das Protokoll sieht so aus, dass zuerst eine Startsequenz erwartet wird gefolgt von einem Befehlscode. Die Antwort des Sensors ist vom Befehlscode abhängig. Der Tabelle 3.6 auf Seite 24 können die Befehle, die der SHT-Sensor kennt, entnommen werden.

Befehl	Code
Reserviert	0000x
Temperaturmessung	00011
Feuchtigkeitsmessung	00101
Statusregister lesen	00111
Statusregister schreiben	00110
Reserviert	0101x – 1110x
Soft-Reset	11110

Tabelle 3.6.: Befehlscodes für den SHT-Sensor, das x ist ein Platzhalter für eine Eins oder Null [14, Seite 6]

Sollte der Soft Reset-Befehl gesendet werden, wird die Schnittstelle zurückgesetzt und die Werte im Statusregister erhalten die Default-Werte. Es muss dann 11 ms gewartet werden, bis der nächste Befehl an den Sensor gesendet werden kann. Bei dieser Arbeit kommen nur die Befehlscodes zur Messung der Feuchtigkeit und der Temperatur zum Einsatz.

Wie schnell der Sensor getaktet werden darf, hängt davon ab, mit welcher Versorgungsspannung der Sensor betrieben wird. Ist die Versorgungsspannung größer 4,5 V, dann kann der Sensor mit 5 MHz betrieben werden. Sollte die Spannung allerdings unter 4,5 V sein, kann er nur mit maximal 1 MHz getaktet werden. In dieser Arbeit wird der Sensor mit 3,3 V betrieben und das hat zur Folge, dass der Sensor nur eine maximale Taktfrequenz von 1 MHz haben kann.

Wie schon erwähnt beginnt jede Kommunikation mit dem Sensor mit der sogenannten „Startsequenz“. Diese Startsequenz ist so definiert, dass zuerst eine Absenkung der Datenleitung (DATA) erfolgen muss, während die Taktleitung High-Pegel hat. Daraufhin muss die Taktleitung von High zu Low und kurze Zeit später wieder von Low zu High wechseln. Währenddessen muss die Datenleitung die ganze Zeit auf Low-Pegel bleiben. Die Datenleitung darf erst wieder auf High gesetzt werden, wenn die Taktleitung auch wieder auf High steht. Das Bild 3.7 auf Seite 24 zeigt, wie die Startsequenz auszusehen hat.

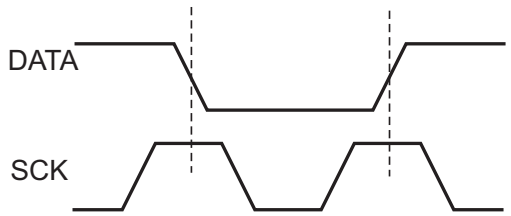


Bild 3.7.: Startsequenz [28, Seite 6]

Nachdem die Startsequenz an den Sensor übermittelt wurde, muss einer der Befehlscodes aus der Tabelle 3.6 von Seite 24 gesendet werden. Zusätzlich zu dem 5 Bit-Befehlscode werden davor noch drei Nullen als Adresse gesetzt, sodass nach der Startsequenz also ein 8 Bit großer Befehl übertragen wird. So lautet der Befehl zum Messen der Feuchtigkeit „00000101“ und der Befehl für die Temperaturmessung „00000011“. Zur Bestätigung, dass ein gültiger Befehlscode gesendet wurde, antwortet der SHT-Sensor mit einem Acknowledge. Das Acknowledge sieht so aus, dass der Sensor die Datenleitung auf Low zieht und anschließend wieder auf High setzt. In der Zeit, in der der SHT-Sensor eine Messung ausführt, befinden sich die Datenleitung auf High-Pegel und die Taktleitung auf Low-Pegel. Die Zeit, die der Sensor für die Messung benötigt, ist abhängig von der Auflösung für die Messung (8,12,14 Bit), die im Statusregister eingestellt wurde. Laut Datenblatt sind das bei einer 8 Bit-Auflösung maximal 20 ms, bei 12 Bit ca. 80 ms und bei der 14 Bit-Auflösung sogar bis zu 320 ms, bis der Sensor die Messung abgeschlossen hat. In dieser Arbeit werden die Default-Werte für die Auflösung benutzt, bei der Temperaturmessung sind dies die 14 Bit- und bei der Feuchtigkeitsmessung die 12 Bit-Auflösung. Durch Testmessungen konnte festgestellt werden, dass der SHT-Sensor für die Temperatur- und Feuchtigkeitsmessung nicht, wie laut Datenblatt zu vermuten war, 400 ms benötigt, sondern nur ca. 300 ms. Nach Beendigung der Messung setzt der Sensor die Datenleitung auf Low und signalisiert so dem Controller, dass der Messwert abgeholt werden kann. Anschließend liest der Mikrocontroller das höchstwertige Byte (MSB) Bit für Bit mit dem msb beginnend ein. Den Empfang des höchstwertigen Bytes muss nun der Mikrocontroller seinerseits mit einem Acknowledge (ACK) bestätigen, indem er die Datenleitung auf Low zieht. Danach kann er das niederwertige Byte (LSB) wieder mit dem msb beginnend Bit für Bit einlesen und dieses auch erneut bestätigen. Zum Schluss kann dann noch eine Prüfsumme empfangen werden. Die Kommunikation mit dem Sensor endet mit dem Bestätigungsbit (ACK) nach der Prüfsumme. Wenn keine Prüfsumme benötigt wird, kann die Kommunikation schon nach dem letzten Datenbit vom Mikrocontroller nach dem Bestätigungsbit beendet werden, indem er die Datenleitung dann auf High setzt. In dieser Arbeit wird auf die Prüfsumme verzichtet. Die ersten zwei Bits des MSB bei der Temperaturmessung sind Nullen und bei der Feuchtigkeitsmessung sind die ersten vier Bits Nullen.[28, 14]

3.1.5. Drucksensor

Ein Drucksensor ist ein Bauteil, das die physikalische Größe Druck (Kraft pro Fläche) in ein proportionales, elektrisches Signal umwandelt. Auf dem Markt gibt es viele verschiedene Arten von Drucksensoren, nachfolgend sind einige aufgeführt:

Relativdrucksensoren messen die Druckdifferenz zum gerade herrschenden Umgebungsdruck. Dieser Wert ändert sich mit der Temperatur und der Höhe zum Meeresspiegel.

Absolutdrucksensoren messen die Druckdifferenz zum absoluten Vakuum. Das Ergebnis der Absolutmessung ist unabhängig vom Wetter und der Höhe.

Differenzdrucksensoren messen die Druckdifferenz von zwei verschiedenen Absolutdrücken.

In dieser Arbeit wird ein Drucksensor der Firma Sensortechnics aus der ASDX-Serie eingesetzt. Er wird benötigt, um mit ihm die Höhe zu bestimmen. Es gibt sowohl einen Absolut-, einen Relativ- und einen Differenzdruckmessenden Sensor in der ASDX-Serie, die in den Bereichen von 1 PSI bis 100 PSI[1] messen können. Die Einheit PSI ist ein älteres Maß zur Angabe des Drucks, wird aber speziell noch im amerikanischen Raum angewendet. Der Umrechnungsfaktor von einem 1 PSI zu Pascal beträgt:

$$1\,\mathrm{PSI} = 6894{,}757\,293\,\mathrm{Pa}$$

Für die Umrechnung von Bar zu Pascal gilt der Faktor 1:100000:

$$1\,\mathrm{PSI} = \frac{6894{,}757\,293}{100\,000}\mathrm{bar}$$

Der Umrechnungsfaktor von 1 PSI zu Bar lautet somit 0,068757293. Damit liegt der Messbereich des hier verwendeten Sensors zwischen ca. 70 mbar bis zu ca. 6,9 bar.

Der ASDX-Sensor bietet ein ratiometrisches Ausgangssignal im Bereich von 0,5 V bis 4,5 V an. Das bedeutet, dass das Verhältnis vom Ausgangssignal zur Versorgungsspannung gleich bleibt. Diese Sensoren sind kalibriert und temperaturkompensiert von 0 °C bis 85 °C. Durch den Einsatz neuester ASIC-Technologie können diese Sensoren kompakter und in kleineren DIP-Gehäusen angeboten werden. In diesem Projekt wird der Sensortyp eingesetzt, der eine Absolutmessung durchführt. Er besitzt eine interne Vakuumreferenz und erzeugt eine zum Absolutdruck proportionale Ausgangsspannung. [27, 7]

3.2. Hardware

Die Messdatenerfassungskarte beruht, wie schon in Abschnitt 3 „*Umsetzung*“ erwähnt, auf einem PIC32-Mikrocontroller der Firma Microchip.

Die Signale der CTA/CCA-Sensoren sind analogen Ursprungs und müssen zur digitalen Weiterverarbeitung durch den Mikrocontroller auf TTL konforme Signalpegel gebracht werden. Die Digitalisierung der durch die CTA/CCA-Sensoren erzeugten Signale erfolgt durch einen

1 steht für Pound-force per square inch; das ist die Kraft, mit der ein Pfund auf eine Fläche von 1 x 1 Zoll drückt.

externen Analog-Digital-Umsetzer (ADU), der im Abschnitt 3.2.2 „*Externer ADU*“ auf Seite 38 näher beschrieben ist.

Die Versorgungsspannungen der Wheatstone-Brückenschaltung werden durch Photomos-Relais geschaltet, da sie sicherer sind als mechanische. Bei Erschütterungen könnten sich die Kontakte der mechanischen Relais lösen, was bei denen der Photomos-Relais nicht passieren kann.

Die Konfigurierung und die Statusinformationen werden über eine serielle Schnittstelle nach der RS-232 Spezifikation gesendet und empfangen. Damit der Mikrocontroller die Daten mit einem seiner Universal Asynchronous Receiver/Transmitter-Module (UART) verarbeiten kann, wird ein Treiber benötigt. Dieser Treiber sorgt dafür, dass der serielle Datenstrom der RS-232-Spezifikation angepasst wird.

Die Telemetrie-Funktionalität der Messkarte wird durch ein weiteres UART-Modul umgesetzt. Zum Anpassen der TTL-Pegel des Mikrocontrollers in die differentielle serielle Datenübertragung wird ein RS-422-Treiber benötigt, der die Signalanpassung durchführt.

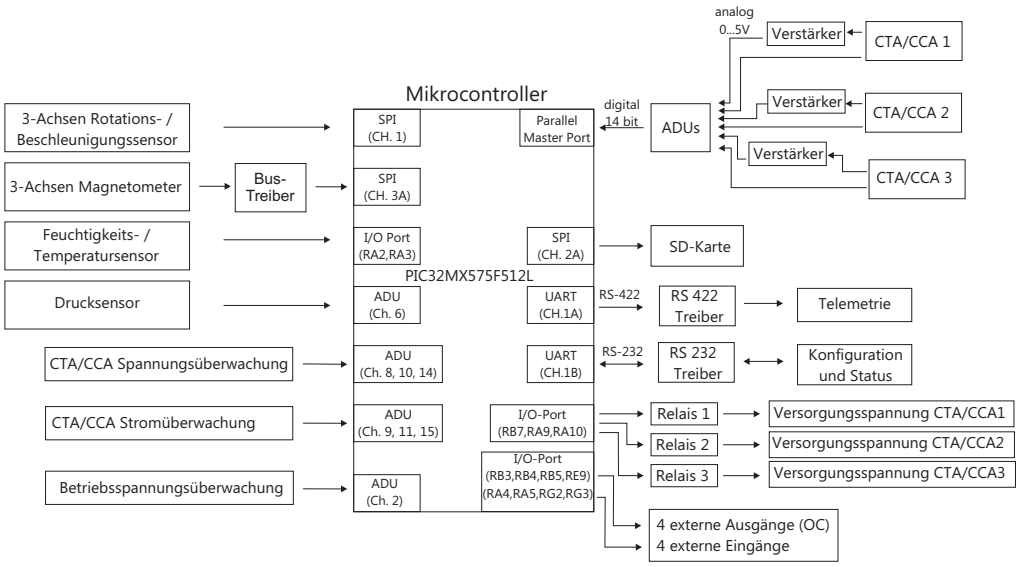


Bild 3.8.: Hardware Blockschaltbild

Das Blockschaltbild 3.8 auf Seite 27 veranschaulicht, wie die geforderte Aufgabe in der Hardware umgesetzt wird.

Die Anbindung der Sensoren erfolgt, wie es in deren Abschnitten erklärt wurde. Die Signale des Magnetometers werden noch über einen Bustreiber geführt, da dieser Sensor nicht direkt auf der Platine montiert wird, sondern außerhalb der Box. Damit der Sensor auch die

richtigen Signalpegel erkennt, werden diese durch den Bustreiber noch etwas verstärkt. Das Magnetometer soll das Erdmagnetfeld messen. Wenn es in der Box befestigt wird, könnten die Messungen durch die Elektronik oder die Batterien beeinflusst werden. Deshalb wird der Sensor außerhalb angebracht.

Die Spannungsversorgung der Messkarte soll durch Batterien erfolgen, die zusammen 14,4 V bereitstellen. Daraus werden die benötigten Betriebsspannungen durch DCDC-Wandler und einen Spannungsregler erzeugt.

3.2.1. Mikrocontroller [2]

In diesem Projekt wird ein Mikrocontroller der PIC32MX-Controllerfamilie eingesetzt. Diese Controller besitzen eine 32 Bit-CPU[3] und können mit einer maximalen Taktfrequenz von 80 MHz betrieben werden. Die Taktfrequenz für die Peripheriegeräte des Mikrocontrollers werden mittels eines Taktteilers aus dem CPU-Takt erzeugt. Die Architektur des PIC32MX-CPUs ist eine RISC[4]-Architektur. Diese Mikrocontroller operieren in einem Spannungsbereich von 2,3 V bis 3,6 V und haben einen Flash-Speicher von 32 bis 512 KByte. Zusätzlich können noch je nach Gerät 8 bis 128 KByte externer RAM-Speicher adressiert werden. Zur Peripherie gehören unter anderem I^2C-, SPI- und UART-Schnittstellen für die serielle Datenübertragung. Die Anzahl der Peripherie-Schnittstellen variiert in den einzelnen Geräten. Ebenfalls besitzen die PIC32-Mikrocontroller einen parallelen Port für eine 8 Bit oder 16 Bit breite Datenübertragung sowie 16 Adressleitungen. Zusätzlich verfügt dieser Mikrocontroller noch über ein *Real-Time Clock und Calendar (RTCC)-Modul* und fünf 16 Bit-Timer, wobei man zwei Timer zu einem 32 Bit-Timer zusammenschalten kann. Außerdem gibt es je fünf Capture-Eingänge, Compare-Ausgänge, die als Pulsweitenmodulation (PWM) benutzt werden können, und externe Interrupt-Pins. Ebenfalls vorhanden ist ein 10 Bit-ADU mit 16 Kanälen. Die PIC32-Mikrocontroller gibt es in einer 64-Pin- und in einer 100-Pin-Variante. Bei der 64-Pin-Variante gibt es nur eine 8 Bit parallele Datenübertragung. Außerdem sind viele Pins in der 64- als auch in der 100-Pin-Variante mehrfach belegt.

Das Bild 3.9 auf Seite 29 zeigt das Blockschaltbild des PIC32MX Mikrocontrollers.

Interrupts

Der PIC32MX hat bis zu 96 Interrupt-Quellen mit 64 Interrupt-Vektoren. Da mehr Interrupt-Quellen (IRQ) als Vektornummern vorhanden sind, müssen sich manche IRQs Vektornummern teilen. Von den 96 Interrupt-Quellen sind fünf externe Interrupt-Pins

2 [20, 19]
3 CPU steht für *central processing unit*; Hauptprozessor
4 *Reduced Instruction Set Computer* engl. für Rechner mit reduziertem Befehlssatz

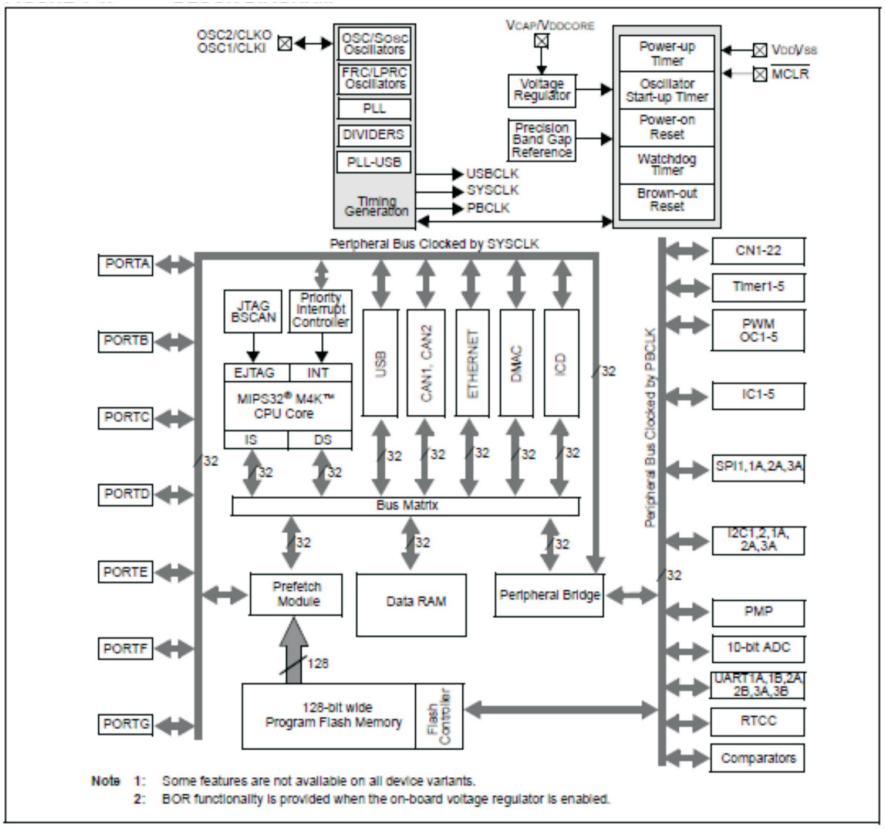


Bild 3.9.: Blockschaltbild des PIC32-Mikrocontrollers[20]

verfügbar. Bei der steigenden Flanke des Systemtaktes werden alle Interrupt-Anfragen zwischengespeichert und der Status in die jeweiligen *Interrupt Flag Status Register (IFSx)* geladen, wobei x die Registernummer angibt. Ein ausgelöster Interrupt wird durch eine „1“ in dem jeweiligen Statusregister (IFSx) dargestellt. Diese Interrupt-Anfrage wird nur dann weiter ausgeführt, wenn das entsprechende Enable Bit im *Interrupt Enable Control Register (IECx)* gesetzt wurde. Eine Auflistung der Interrupt Quellen und den dazugehörigen Status- und Enable-Registern kann dem Anhang A.1 auf Seite 80 entnommen werden.

Jeder Interrupt erhält eine von sieben Prioritäten und eine von vier Subprioritäten. Werden zeitgleich zwei Interrupts ausgelöst, die dieselben Prioritäten haben, dann wird über die Subpriorität entschieden, welcher zuerst ausgeführt wird. Sollten sowohl die Priorität als auch die Subpriorität dieselbe sein, so wird der Interrupt mit der geringeren Interrupt-Vektor-Adresse zuerst bearbeitet. Der Interrupt Controller des PIC32MX kann in einem von zwei Modi konfiguriert werden:

- Single Vektor Mode – Alle Interrupt-Anfragen haben nur eine Vektoradresse.

- Multi Vektor Mode – Jede Interrupt-Anfrage hat eine eigene Vektoradresse.

In dieser Arbeit wird der Multi Vektor Modus benutzt, weil mehrere Interrupts zur Erfüllung der Aufgabe benötigt werden. Jeder dieser Interrupts soll seine eigene Vektoradresse erhalten.

Das Bild 3.10 auf Seite 30 zeigt, wie ein Interrupt-Prozess beim PIC32MX aussieht.

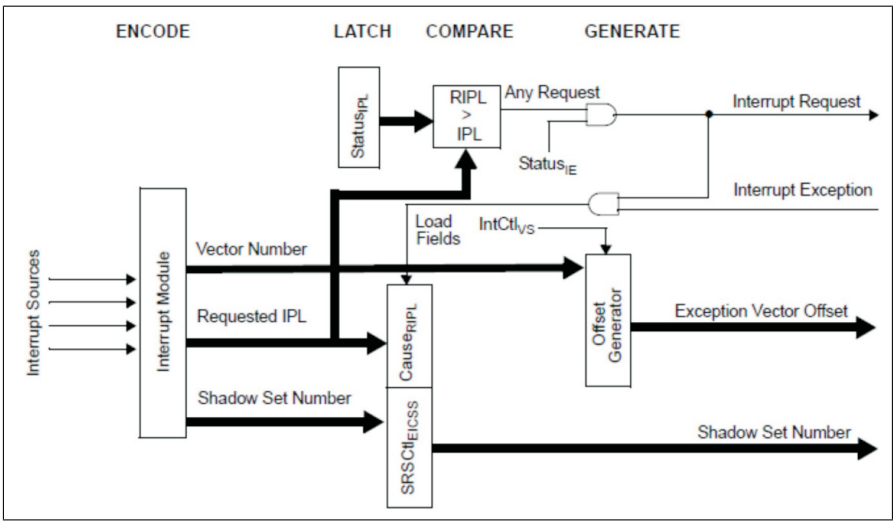


Bild 3.10.: Interrupt-Prozess beim PIC32MX [19]

Timer

Zur Verfügung stehen dem PIC32MX fünf Timer. Diese Timer können als 16 Bit synchron laufende Timer oder Zähler benutzt werden (Timer Variante B). Durch eine Verknüpfung von Timer2 und Timer3 bzw. Timer4 und Timer5 besteht die Möglichkeit, einen 32 Bit-Timer oder Zähler herzustellen. Beim gleichen Wert des Zählregisters TMRx (die Timer-Nummer wird durch das x dargestellt) und des dazugehörigen Periodenregisters PRx erzeugen der Timer oder der Zähler einen Interrupt. Beim nächsten Takt wird das Zählregister TMRx zurück auf Null gesetzt. Bei jeder positiven Flanke des Peripherietaktes wird der Wert im Zählregister erhöht. Die Geschwindigkeit des Timers kann durch das Vorschalten von einem Taktteiler reduziert werden. Dieser Teiler kann 1:1 oder bis zu 1:256 sein. Bei einer Taktteilung von 1:64 würde dies zur Folge haben, dass das Zählregister bei jeder 64ten positiven Flanke des Peripherietaktes erhöht wird.

In diesem Projekt kommen zwei dieser 16 Bit-Timer zum Einsatz. Der eine Timer (Timer3) wird benutzt, um den Takt für die Kommunikation mit dem

Temperatur- und Feuchtigkeitssensor, der im Abschnitt 3.1.4 „*Temperatur- und Feuchtigkeitssensor*“ auf Seite 23 vorgestellt wurde, zu erzeugen. Der andere Timer (Timer2) tastet das CTA/CCA-Signal mit 8 kHz ab. Bei beiden Timern wurde der Taktteiler auf 1:1 eingestellt.

UART

Dem PIC32MX stehen bis zu sechs UART-Schnittstellen unter der Voraussetzung, dass andere Schnittstellen wie z.B. I^2C oder SPI nicht mehr verwendet werden können, zur Verfügung. Hier können und werden auch nur zwei von den sechs UART-Modulen genutzt, weil die Pins der anderen Module für die SPI-Schnittstellen oder als I/O-Pins gebraucht werden. Die UART-Module arbeiten im Voll-Duplex-Modus. Die eine UART-Schnittstelle wird für die Kommunikation mit dem PC benötigt und die andere ist für das Senden der Daten über die Telemetrie vorgesehen.

Die UART-Module des hier verwendeten PIC32-Controllers unterstützen auch die Hardware-Option für die Flusskontrolle. Wenn die Flusskontrolle in Anspruch genommen wird, werden vier Leitungen statt nur der zwei Leitungen (TX und RX) benötigt. Die zusätzlichen Leitungen lauten $\overline{CTS}$ (Clear to send) und $\overline{RTS}$ (Request to send). In diesem Projekt kann die Hardware Flusskontrolle nicht genutzt werden, da die Schnittstelle, die noch nicht von der Funktion eines anderen Peripherie-Moduls in Anspruch genommen wird, in die zwei erforderlichen UART-Schnittstellen aufgeteilt werden muss.

Die UART-Module können eine 8 Bit- oder 9 Bit-Übertragung haben. Bei der 8 Bit-Übertragung besteht noch die Möglichkeit, zur Sicherheit ein Paritätsbit (gerade, ungerade oder keins) anzugeben. Zudem unterstützt der PIC32 auch das Setzen von einem oder zwei Stop-Bits. Die Standardkonfigurierung der UART-Module ist die 8N1, d. h. 8 Datenbits, keine (no) Parität und 1 Stop-Bit, und sie benutzen das NRZ[5] Format. Dieses Übertragungsformat ist am verbreitetsten bei der Kommunikation mit vielen Geräten.

Zur Einstellung der Baudraten besitzen die UART-Module einen 16 Bit Baud Rate Generator (BRG). In dem BRG-Register des jeweiligen Moduls wird die Periode des 16 Bit freilaufenden Timers eingestellt. Es ist möglich, Baudraten von 76 bps (Bits pro Sekunde) bis zu einer Baudrate von 20 Mbps (Megabits pro Sekunde) bei einem Systemtakt von 80 MHz einzustellen. Die Kommunikation mit dem PC erfolgt mit einer Baudrate von 9600 bps. Außerdem besitzen die Module separate Datenfifos für die Empfangs- und Sendeseite und können auch separate Interrupts für das Empfangen und Senden erzeugen.

5 NRZ steht für non-return-to-zero

Takterzeugung

Der PIC32MX-Mikrocontroller besitzt drei Haupttakte:

- der Systemtakt (SYSCLK) für die CPU und manche Peripheriegeräte
- der Peripherie-Bustakt (PBCLK) für die meisten Peripheriegeräte
- der USB-Takt für die USB-Peripheriegeräte

Die USB-Funktion, die der Mikrocontroller unterstützt, wird in diesem Projekt nicht genutzt.

Systemtakt (SYSCLK)
Der Systemtakt wird hauptsächlich von der CPU benutzt, aber gewisse Peripheriegeräte wie die DMA- und der Interrupt-Controller benutzen den Systemtakt auch. Der Systemtakt wird in diesem Projekt durch eine externe Taktquelle (Quarz), die an den Pins OSCI und OSCO angeschlossen ist, erzeugt. Diese Variante der Takterzeugung wird der „Primäroszillator“ (POSC) genannt.

Peripherie-Bustakt (PBCLK)
Der zweite Haupttakt bei PIC32MX ist der Peripherie-Bustakt (PBCLK) und wird durch einen Teiler aus dem Systemtakt (SYSCLK) erzeugt. Wie der Name schon sagt, werden mit dem Peripherie-Bustakt die meisten Peripherie-Module getaktet. Der Teiler wird durch Setzen der PBDIV-Bits im OSCON-Register eingestellt und kann den Wert 1, 2, 4 oder 8 haben. Der Peripherie-Bustakt kann nicht schneller sein als der Systemtakt, und wenn der Taktteiler 1 gewählt wurde, haben der SYSCLK und der PBCLK die gleich Frequenz. Der Taktteiler wurde hier auf eins eingestellt, damit die Peripherie-Module mit der höchstmöglichen Frequenz arbeiten können.

Für die Takterzeugung mit dem **Primäroszillator** gibt es sechs Betriebsmodi, die der Tabelle 3.7 auf Seite 33 entnommen werden können. Die drei oberen Modi in der Tabelle stellen die einfachen dar: High Speed (HS), External Resonator (XT) und External Clock (EC). In Verbindung mit der im Chip befindlichen Phase Locked Loop (PLL)[6] können die ersten Modi zu den letzten Modi der Tabelle geformt werden. Die PLL besitzt einen Eingangsteiler, einen Multiplikator und einen Ausgangsteiler, damit können dann höhere Frequenzen eingestellt werden. Die Bezeichnung dieser letzten drei Modi sind High Speed PLL (HSPLL), External Resonator PLL (XTPLL) und External Clock (ECPLL). Damit der POSC mit der PLL arbeitet, muss der Eingangsteiler so gewählt werden, dass die daraus resultierende Frequenz in dem Bereich zwischen 4 MHz und 5 MHz liegt. Wie schon erwähnt wird der Primäroszillator an die Pins OSCI und OSCO des Mikrocontrollers angeschlossen. In diesem Projekt soll der PIC32-Controller mit der maximal möglichen Taktfrequenz von 80 MHz arbeiten. Um das zu bewerkstelligen, wurde hier die Variante mit einem externen 8 MHz-Quarz und die Benutzung

6 Phasenregelschleife

Oszillator-Modus	Beschreibung
HS	10 MHz – 40 MHz Quarz, High Speed Quarz
XT	3,5 MHz – 10 MHz Resonator, Quarz oder Resonator
EC	Externer Takteingang
HSPLL	Quarz mit PLL
XTPLL	Quarzschwinger mit PLL
ECPLL	Externer Takteingang mit PLL

Tabelle 3.7.: Primäroszillator-Betriebsmodi [19]

der im Chip befindlichen PLL gewählt. Somit wird der Systemtakt mit dem XTPLL-Modus erzeugt. Um mit dem hier verwendeten Quarz in den gültigen Bereich der PLL zu kommen, muss der Eingangsteiler auf 2 eingestellt werden. Damit der SYSCLK auch am Ende mit 80 MHz arbeitet, muss der Multiplikator auf 20 und der Ausgangsteiler auf 1 gesetzt werden.

Zusätzlich zu dem Primäroszillator wird noch ein Sekundäroszillator benötigt. Der Sekundäroszillator ist für den Low-Power-Betrieb mit einem externen 32,768 kHz-Kristall ausgelegt. Der externe Kristall wird an die Pins SOSCO und SOSCI des Mikrocontrollers angeschlossen und dient als ein zweiter Takt für einen Low-Power-Betrieb. Der Sekundäroszillator wird für die Benutzung des RTCC-Moduls benötigt.

DMA-Controller

Der Direct Memory Access (DMA) Controller ist ein Bus Master-Modul, das Daten zwischen unterschiedlichen Peripherie-Modulen hin und her transportiert, ohne dass die CPU dadurch belastet wird. Dabei werden die Daten von einer Start- zu einer Zieladresse transportiert. Die Start- und die Zieladresse können eine Adresse aus dem RAM-Speicher oder eine Registeradresse aus jedem beliebigen Peripherie-Modul sein, wie z. B. SPI oder UART. In diesem Projekt werden vier der acht möglichen DMA-Kanäle verwendet. Der Kanal 4 wird benutzt, um ein Kommando, das asynchron zum normalen Betrieb über die UART-Schnittstelle 1B übertragen wird, zu empfangen und darauf zu reagieren. Bei diesem Kanal läuft der Transfer zwischen der Speicheradresse des Empfangsregisters der UART-Schnittstelle 1B und einer Adresse im RAM-Speicher ab. Mit dem Kanal 3 erfolgt der entgegengesetzte Transfer zum Kanal 4, wo die Daten von einer RAM-Adresse zu dem Senderegister der UART-Schnittstelle 1B transportiert werden. Und die DMA-Kanäle 1 und 2 werden für die Kommunikation mit dem Magnetometer über die SPI-Schnittstelle 3A benötigt.

In diesem Projekt werden der Basic Transfer-Modus und der Pattern Match Termination-Modus eingesetzt. Beim Basic Transfer-Modus funktioniert der Datentransfer folgenderma-

ßen: Dort werden, wie am Anfang des Abschnitts schon erwähnt, Daten von einer Start- zu einer Zieladresse überführt. Zusätzlich wird noch angegeben, wie viele Daten verschoben werden sollen und wie viele Bytes pro einzelnem Transfer übermittelt werden. Der DMA-Transfer kann durch das Setzen des CForce-Bits im *DMA Channel x Event Control Register (DCHxECON)* oder durch eine Interrupt-Quelle des Controllers gestartet werden. Das x steht für die Kanalnummer. Bei den vier verwendeten DMA-Kanälen wird der Transfer durch die Interrupt-Quellen der UART 1B- und der SPI 3A-Schnittstellen gestartet. Der erfolgreiche Datentransfer wird durch Auslösung eines Interrupts signalisiert. Hat ein erfolgreicher DMA-Transfer stattgefunden, so muss der Kanal erst freigegeben werden, bevor er wieder benutzt werden kann. Die Kommunikation mit dem Magnetometer ist in dem Basic Transfer-Modus umgesetzt worden.

Die beiden Kanäle 3 und 4, die einen Transfer mit den UART-Registern durchführen, sind in dem Pattern Match Termination-Modus konfiguriert. Beim Pattern-Modus kann der DMA-Transfer, zusätzlich zu der Variante des Basic-Modus, auch durch ein vordefiniertes Zeichen, das im *DMA Channel x Pattern Data Register (DCHxDAT)* gespeichert ist, abgebrochen werden. Es ist egal, wie lang die Nachricht ist, bei der ersten Übereinstimmung mit dem definierten Zeichen wird der Transfer abgebrochen. Ansonsten funktioniert der Datentransfer genauso wie beim Basic-Modus. Das Bild 3.11 auf Seite 34 stellt einen einfachen DMA-Transfer zwischen einer Start- und einer Zieladresse dar.

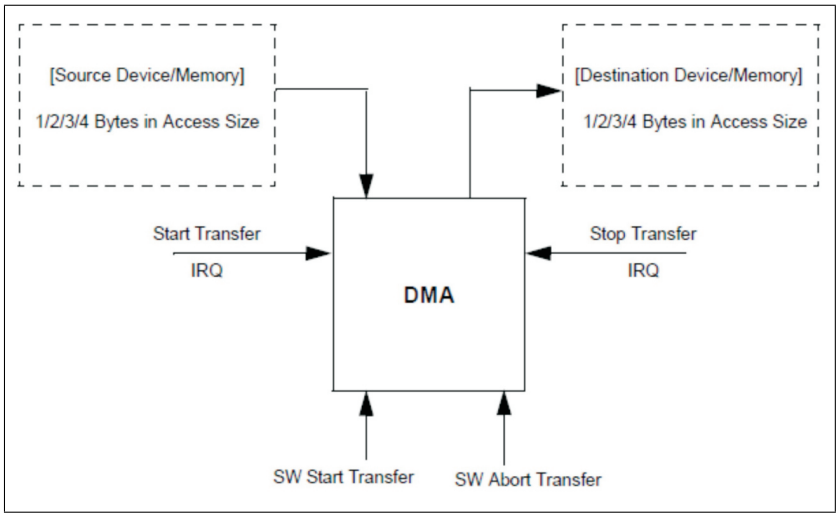


Bild 3.11.: Typischer DMA-Datentransfer zwischen Start- und Zieladresse [19]

I/O-Port

Das einfachste Peripherie-Modul des PIC32MX-Mikrocontrollers sind die Ein- und Ausgänge. Bei dem hier verwendeten PIC32MX-Controller wären, wenn keine anderen Peripheriegeräte benötigt würden, bis zu 83 Pins als Ein- oder Ausgänge vorhanden. Die sieben I/O-Ports sind mit den Buchstaben A bis G gekennzeichnet. Die meisten der I/O-Pins sind allerdings mit Funktionen von anderen Peripherie-Modulen doppelt belegt. Mit dem *ADC Port Configuration Register (AD1PCFG)* kann eingestellt werden, ob die Pins des Ports B im analogen oder im digitalen Modus arbeiten. Durch Setzen einer Null wird der entsprechende Pin in den analogen Modus gesetzt. Jeder I/O-Port besitzt die drei Register TRIS, LAT und PORT. In dem TRIS-Register wird die Richtung für die digitalen Pins eingestellt. Eine „1“ konfiguriert diesen Pin als einen Eingang und eine „0“ als einen Ausgang. Wenn ein Pin als Eingang definiert wird, ist das ein Schmitts Trigger. Wenn die Pins aber als Ausgang konfiguriert werden, sind das CMOS-Treiber oder können als Open Drain-Ausgänge durch Setzen der entsprechenden Bits in dem Open *Drain Configuration Register (ODC)* definiert werden. Mit der Open Drain-Funktionalität können Spannungen, die größer als die Versorgungsspannung sind, geschaltet werden. Diese Funktion wird in dieser Arbeit nicht benötigt, da die Sensoren und der externe ADU keine Signale größer 3,3 V brauchen. Mit dem LAT-Register können Signale vom Prozessor auf die Pins geschaltet werden. Über das PORT-Register wird mittels eines Lesezugriffs der aktuelle Zustand des Pins abgefragt. Zusätzlich zu dem gerade beschriebenen Register besitzen die Ports noch drei weitere Register, mit denen schneller eine Bit-Änderung erfolgen kann, das SET-, das CLR- und das INV-Register.

Parallel Master Port (PMP)

Der Parallel Master Port des PIC32MX ist ein 8 oder 16 Bit breiter I/O-Port, mit dem die Kommunikation mit einer Vielzahl von verschiedenen parallelen Geräten möglich ist. Für diese Arbeit wird ein 100 Pin-Typ eingesetzt, denn nur bei diesem funktioniert die 16 Bit breite Datenübertragung. Dem Modul stehen bis zu 16 Adress- und zwei Chip Select-Leitungen zur Verfügung. Hinzu kommen noch die zusätzlichen Steuerleitungen (z.B Read oder Write), die für die Kommunikation mit den parallelen Geräten benötigt werden. Der PMP wird in diesem Projekt für das Abholen der Daten von dem externen ADU, der im Abschnitt 3.2.2 „*Externer ADU*“ auf Seite 38 beschrieben ist, benötigt. Die 16 Adressleitungen werden nicht gebraucht und von anderen Peripherie-Modulen oder als normale I/O-Pins benutzt. Der Parallel Master Port setzt auch die Signale für einen Lese- oder Schreibzugriff. Dabei gibt es abhängig von der Einstellung des Master Modes zwei mögliche Optionen für die Leitungen:

1. individuelle Read- und Write-Leitung (Master Mode 2)
2. eine Read/Write-Leitung mit einer Enable-Leitung (Master Mode 1).

Das Modul ist in dem Master Mode 2 konfiguriert, um getrennte Write- und Read-Leitungen zu erhalten. Es wird nur die Read-Leitung verwendet, da der externe ADU nur ausgelesen werden soll und die Write-Leitung wird nicht benötigt. Die Polarität der Read-Leitung wurde, wie von dem externen ADU gefordert, in dem *Parallel Port Control Register (PMCON)* auf Low aktiv gesetzt. Die beiden Chip-Select-Leitungen teilen sich ihre Funktion mit den Adressleitungen A15 (PMCS2) und A14 (PMCS1). Der Parallel Master Port unterstützt auch die Konfigurierung als Slave, aber das wird in dieser Arbeit nicht benötigt und daher wird nicht weiter darauf eingegangen. Das PMP-Modul unterstützt nur, dass die Chip Select-Leitung 2 allein vorhanden sein darf, aber nicht die Chip Select-Leitung 1. Durch die mehrfache Belegung der Pins kann in diesem Projekt die Chip Select-Unterstützung nicht in Anspruch genommen werden.

Analog-Digital-Umsetzer (ADU)

Der PIC32MX besitzt einen 10 Bit Analog-Digital-Umsetzer (ADU), der eine Umsetzgeschwindigkeit von bis zu 1 Msps (mega sample per seconds) haben kann. Von den möglichen 16 analogen Eingängen werden hier acht gebraucht. Die Eingänge des ADUs sind mit dem „Sample und Hold“-Verstärker über zwei Multiplexer verbunden. Es gibt zwei Möglichkeiten, eine Referenzquelle für den ADU anzugeben: über zwei analoge Pins als externe oder über die analoge Versorgungsspannung und -masse als interne Referenzquelle. Da die analoge Versorgung immer an den Mikrocontroller angeschlossen werden muss, wurde hier einfachheitshalber auf die Variante mit der externen Referenzquelle verzichtet. Das ADU-Modul besitzt einen 16 Word großen Ergebnispuffer, in dem die digitalen Werte zwischengespeichert werden. Diese Werte werden in zwei 8 Word große Puffer aufgeteilt, die abwechselnd mit den gemessenen Daten beschrieben werden, sodass sich die Gefahr verringert, dass ungelesene Daten einfach verloren gehen.

Die Umsetzung des analogen Werts in einen digitalen erfolgt in zwei Vorgängen: die Erfassung des analogen Werts und die Umsetzung in den digitalen Wert. Während der Erfassung ist der zu messende Eingang mit dem „Sample und Hold“-Verstärker verbunden, damit nach einer ausreichenden Zeitspanne die erfasste Eingangsspannung gleich der Ausgangsspannung des Verstärkers ist. Dann wird der Eingangspin am „Sample and Hold“-Verstärker gelöst, um eine stabile Spannung für die Umsetzung zu erhalten. Abschließend wird die erfasste Eingangsspannung in einen digitalen Wert gewandelt.

Das ADU-Modul kann sowohl nur einzelne als auch im Scan-Modus mehrere Eingänge hintereinander in digitale Werte wandeln. Wie lange der Eingang jeweils abgetastet werden soll, kann entweder manuell durch die Software oder automatisch durch die Hardware erfolgen. In dem Auto Sample-Modus beginnt der ADU nach einer vollständigen Wandlung automatisch mit der nächsten. Die Auto Sample-Funktion wird durch das *ADC Sample Auto-Start-Bit*

(ASAM-Bit) im *ADC Control Register 1 (AD1CON1)* kontrolliert. Da die acht Eingänge, die in diesem Projekt benutzt werden, im Scan-Modus abgefragt werden, empfiehlt es sich, mit dem Auto Sample-Modus zu arbeiten. Der AD-Umsetzer braucht für die Umsetzung eines Bits genau einen ADU-Taktzyklus (TAD), das bedeutet, dass für das 10 Bit Ergebnis 10 Takte gebraucht werden würden. Allerdings ist der endgültige digitale Wert erst nach zwölf TAD fertig, da der ADU noch zwei zusätzliche Zyklen benötigt. Der ADU-Takt hat den Peripherie-Bustakt (PBCLK) als Quelle.

Im *ADC Control Register 2 (AD1CON2)* wird eingestellt, nach wie vielen Umsetzungen der Interrupt auslösen soll. Da in diesem Projekt acht Eingänge nacheinander gewandelt werden sollen, wurden die SMPI-Bits im AD1CON2 so gesetzt, dass der Interrupt nach der achten Wandlung auslöst. Zusätzlich wurde noch festgelegt, dass das ASAM-Bit automatisch gelöscht wird, wenn der Interrupt ausgelöst hat.

SPI

Die SPI-Module des PIC32MX sind Schnittstellen, die eine synchrone serielle Kommunikation mit den hier verwendeten Sensoren und dem externem Speicher ermöglichen. Der PIC32MX besitzt abhängig vom Typ, der gewählt wurde, 3 oder 4 SPI-Schnittstellen, allerdings nur unter der Voraussetzung, dass keine anderen seriellen Schnittstellen (UART oder I^2C) benötigt werden, denn manche SPI-Module teilen sich die Pins mit den Funktionen der anderen seriellen Schnittstellen. In diesem Projekt wird ein Mikrocontrollertyp eingesetzt, der bis zu vier SPI-Schnittstellen hat. Die SPI-Module können einmal im Master-Modus oder im Slave-Modus konfiguriert werden. In dieser Arbeit werden die SPI-Module im Master Modus konfiguriert, denn der Controller soll die führende Kraft bei der Kommunikation mit den Sensoren oder der SD-Karte sein. Die Übertragungsbreite der Daten zum Magnetometer ist auf 8 Bit eingestellt, wie der Sensor es erwartet (siehe Abschnitt 3.1.3 „*Magnetometer*“ auf Seite 20). Die SPI-Module unterstützen auch eine 16 Bit-Übertragung; diese Übertragungsbreite hat auch der ADIS-Sensor. Die Übertragungsbreite für die Kommunikation mit der SD-Karte ist auf 8 Bit eingestellt. Gleichzeitig besitzen die SPI-Module getrennte Empfangs- und Sendeschieberegister, damit die Möglichkeit des gleichzeitigen Sendens und Empfangens von Daten besteht. Um Daten zu empfangen, müssen vorher welche vom Master (PIC32) gesendet werden. Die Taktpulse für die Kommunikation mit den Slave-Geräten erfolgt durch den Master. Die Anzahl hängt von der Übertragungsbreite ab. Bei dem ADIS-Sensor erzeugt der Master 16 Taktpulse, bei der Kommunikation mit der SD-Karte und dem Magnetometer sind es 8 Taktpulse. Die Frequenz mit der die SPI-Module ihre Daten übertragen, wird aus dem Peripherie-Bustakt gewonnen und in dem *SPI Baud Rate Generator Register (SPIxBRG)* eingestellt. Der Wert für das SPIxBRG-Register wird nach der Formel 3.1 berechnet.

$$F_{SCK} = \frac{F_{PB}}{2 \cdot (SPIxBRG + 1)}$$

$$SPIxBRG = \frac{F_{PB}}{2 \cdot F_{SCK}} - 1 \tag{3.1}$$

Die SPI-Module können drei Interrupts erzeugen:

- Receive Data Available Interrupt – Es wird ausgelöst, wenn neue Daten im Empfangspuffer vom SPIBUF-Register sind.
- Transmit Buffer Empty Interrupt – Es wird ausgelöst, wenn Platz für neue Daten im Sendepuffer des SPIBUF-Registers ist.
- Receive Buffer Overflow Interrupt – Es wird ausgelöst, wenn neue Daten empfangen, aber die alten noch nicht aus dem Empfangspuffer gelesen wurden.

RTCC

Das Real-Time Clock und Calendar (RTCC) ist das Hardware-Modul des PIC32MX, mit dem die Uhrzeit- und Kalender-Funktion ausgeführt wird. Wie schon im Abschnitt „*Taktererzeugung*“ auf Seite 32 erwähnt, wird für die Benutzung des RTCC-Moduls ein zusätzlicher Kristall mit einer Frequenz von 32.768 kHz benötigt. Das RTCC-Modul stellt die Uhrzeit im 24-Stunden-Format und nicht im dem am/pm-Format dar. Die Zeitinformation wird mittels des BCD[7]-Formats in dem RTCC-Register (*RTC Time Value Register (RTCTIME)* und *RTC Date Value Register (RTCDATE)*) gespeichert. Es ist möglich, einen Zeitbereich von 00:00:00 des 1. Januars 2000 bis 23:59:59 des 31. Dezembers 2099 darzustellen. Zusätzlich unterstützt das RTCC-Modul die Schaltjahrerkennung.

In diesem Projekt kommt nur die Uhrzeit-Funktion des RTCC-Moduls zum Einsatz. Diese Funktion wird benötigt, um zu erfahren, wann die Daten aufgenommen wurden und ob beim Speichern vielleicht Datenpakete verloren gegangen sind.

3.2.2. Externer Analog-Digital Umsetzer [8]

Das Signal der CTA/CCA-Sensoren soll mit einer Auflösung von 14 Bit abgetastet werden. Der ADU des PIC32MX-Controllers erreicht nur eine Auflösung von 10 Bit, deshalb wurde ein externer ADU benötigt, der eine höhere Auflösung hat. Folgende Anforderungen musste der ADU erfüllen:

- Er sollte mindestens eine Auflösung von 14 Bit erreichen.

7 Binary Coded Decimal; engl. für binär kodierte Dezimalzahl
8 [13]

- Er sollte möglichst drei oder mehr Kanäle haben.
- Er sollte eine serielle (i^2C, SPI) oder eine parallele Schnittstelle haben.
- Er sollte Spannungen zwischen 0 V und 5 V wandeln können.
- Er sollte mindestens eine Abtastrate von 8 kHz haben.
- Er sollte in dem Temperaturbereich zwischen $-40\,^{\circ}C$ – $80\,^{\circ}C$ arbeiten können.

Einen geeigneten Analog-Digital-Umsetzer zu finden, war nicht einfach, da kaum ein ADU alle Bedingungen erfüllte. Die anfängliche Anforderung für den Datentransfer musste noch mal verfeinert werden, denn wegen der schon mehrmals erwähnten Doppelbelegung der Pins des Mikrocontrollers sind nur ADUs mit einer parallelen Schnittstelle oder eine Schnittstelle mit dem I^2C-Protokoll geeignet. Es wurde ein Analog-Digital-Umsetzer von Analog Devices eingesetzt, da er bis zu 6 Kanäle gleichzeitig wandeln kann. Außerdem hat der verwendete ADU für jeden einzelnen Kanal einen eigenen „Sample und Hold“-Verstärker und jeweils einen ADU, der nach dem Sukzessive Approximationsverfahren das analoge Signal in den entsprechenden Digitalwert wandelt. Dieser ADU hat zum Datenaustausch mit einem Mikrocontroller eine parallele oder eine serielle Schnittstelle. Benutzt wird die parallele Schnittstelle, weil der PIC32MX-Controller über einen parallelen Port verfügt. Gestartet werden die Messungen mit dem ADU über die CONVST-Leitungen des Umsetzers. Als Referenzquelle besitzt der ADU eine 2,5 V Referenzspannung (V_{REF}) im Chip und der Spannungsbereich kann per Software durch Beschalten des Range-Pins auf ± 4 mal die V_{REF} oder ± 2 mal die V_{REF} eingestellt werden. Wenn kein flexibler Spannungsbereich benötigt wird, kann der Range-Pin auch fest auf einen Bereich eingestellt werden. Das Blockschaltbild zu dem verwendeten Analog-Digital-Umsetzer ist auf dem Bild 3.12 auf Seite 40 dargestellt.

Die Beschaltung des Analog-Digital-Umsetzers wurde so umgesetzt, wie es im Datenblatt beschrieben ist. Jeder der acht analogen Versorgungspins AV_{CC} wird zur Entkopplung mit einem 10 µF Tantal- und 100 nF Keramikkondensator beschaltet. Dieselbe Beschaltung zur Entkopplung wird auch für den digitalen Versorgungspin DV_{CC} und die beiden Pins V_{DD} und V_{SS} benutzt. Diese beiden Pins werden benötigt, damit die Eingangssignale auch zwischen ± 10 V liegen dürfen. Zusätzlich werden noch weitere Entkopplungskondensatoren benötigt, z. B. für den V_{DRIVE}-Pin. Wird dieselbe Versorgungsspannung für die analoge und die digitale Versorgung benutzt, so soll eine Spule zwischen den beiden Versorgungen verhindern, dass sich Störungen durch zu hohe Ströme auf die restliche Schaltung auswirken. Damit die Daten über die parallele Schnittstelle übermittelt werden, muss der Pin „SER/$\overline{PAR}$“ auf Low-Pegel geschaltet werden. Zusätzlich kann eingestellt werden, ob die Daten Byte- oder Word-breite haben. Da der Mikrocontroller auch die Word-breite Datenübertragung unterstützt, wird diese verwendet, was durch Setzen eines Low-Pegels an den „$\overline{W}$/B“-Pin des ADU erreicht wird.

Wie am Anfang schon erwähnt, werden mit den CONVST-Leitungen die Messungen gestartet. Der ADU besitzt insgesamt drei CONVST-Leitungen und jede ist einem Paar der

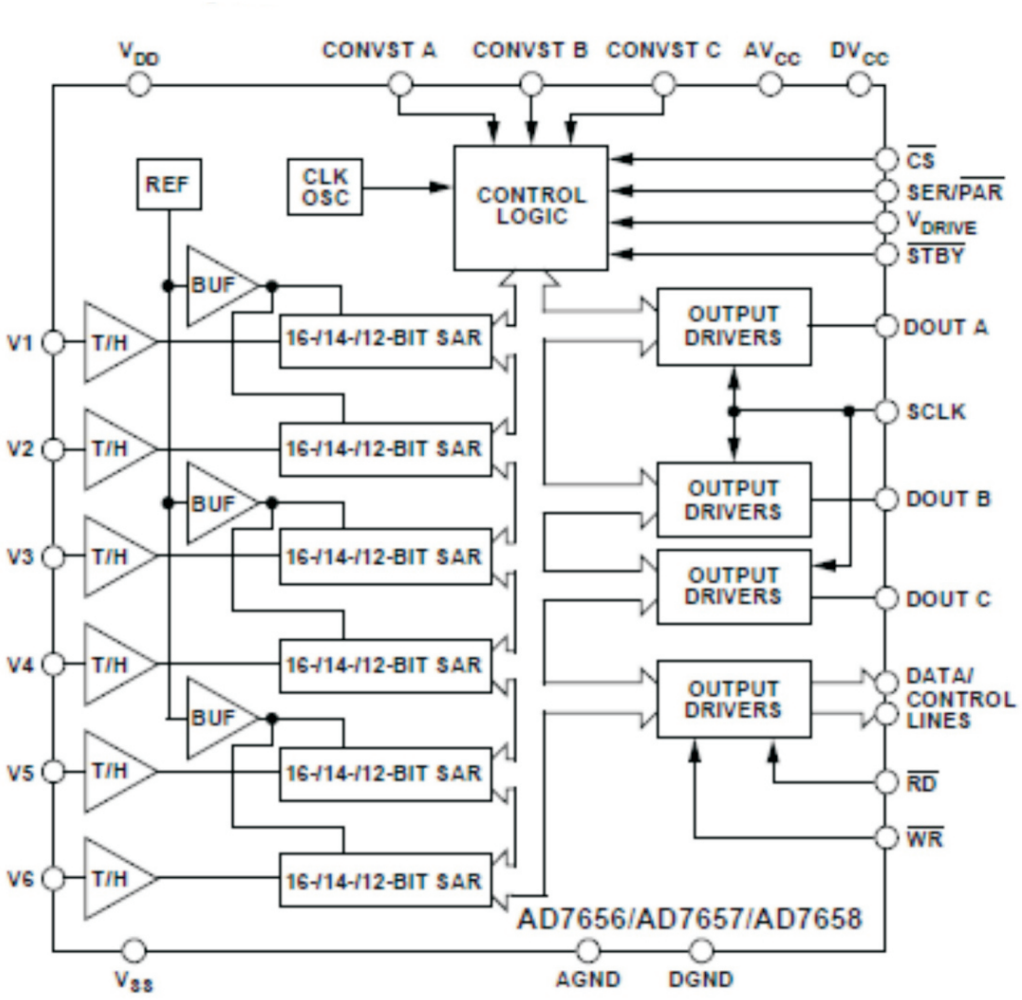


Bild 3.12.: Blockschaltbild des externen ADUs [13, Seite 1]

internen ADUs zugeordnet. Zum Starten muss ein Puls über die CONVST-Leitung übertragen werden. Dabei wird mit der CONVST-Leitung A das ADU-Paar 1 und 2, mit der CONVST-Leitung B das ADU-Paar 3 und 4 und mit der CONVST-Leitung C das ADU-Paar 5 und 6 gestartet. Zusätzlich unterstützt der ADU noch das simultane Wandeln durch gleichzeitiges Setzen der drei CONVST-Leitungen.

Nachdem ein gültiger CONVST-Puls übermittelt wurde, führt der ADU seine Messung durch, was ungefähr 3 µs dauert. Eine fertige Wandlung wird durch die fallende Flanke der Busy-Leitung signalisiert. Mit Hilfe der $\overline{RD}$- und der $\overline{CS}$-Leitung können die gewandelten Werte vom ADU ausgelesen werden. Wie viele Lesezugriffe nötig sind, um alle gewandelten Daten auszulesen, hängt davon ab, wie viele CONVST-Leitungen gleichzeitig gesetzt wurden. Das Bild 3.13 auf Seite 41 zeigt das Timing-Diagramm der parallelen Schnittstelle. In dem

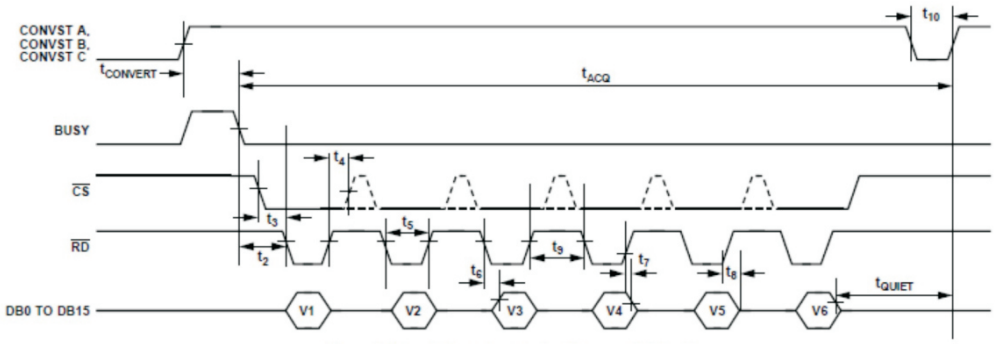


Bild 3.13.: Timing-Diagramm der parallelen Schnittstelle des externen ADUs [13, Seite 23]

Timing-Diagramm (Bild 3.13 auf Seite 41) wurden alle drei CONVST-Leitungen gleichzeitig gesetzt und somit waren sechs Lesezugriffe nötig, um alle gemessenen Werte auszulesen. Außerdem wird eine Vereinfachung des Lesezugriffs dargestellt, indem die $\overline{CS}$ zum Auslesen dauerhaft auf Low-Pegel gesetzt wurde; dadurch muss nur noch die $\overline{RD}$ gesetzten werden, um die Daten auszulesen. Die gewandelten Werte sind im Zweierkompliment angegeben.

Zusätzlich wird im Datenblatt des ADUs noch empfohlen, die Eingangssingle über den Verstärker AD8021 an den ADU-Eingängen zu schalten. Der ADU besitzt insgesamt sechs Kanäle und es sollen nur maximal drei CTA/CCA-Sensoren angeschlossen werden. Um nicht drei Kanäle ungenutzt zu lassen, werden die Sensorsignale direkt auf die ADU-Eingänge und über den empfohlen Verstärker geführt. Der Verstärker wurde so dimensioniert, dass er als ein nichtinvertierender Verstärker mit der Verstärkung von Eins arbeitet. Per Software kann dann konfiguriert werden, ob der ADU nun die CTA/CCA-Sensorsignale direkt messen oder die über den Verstärker geführten Signale messen soll. Die Beschaltung zu dem Verstärker kann dem Bild 3.14 auf Seite 42 entnommen werden.

Zusätzlich zu den Widerständen, mit denen der Verstärkungsfaktor eingestellt wird, benötigt

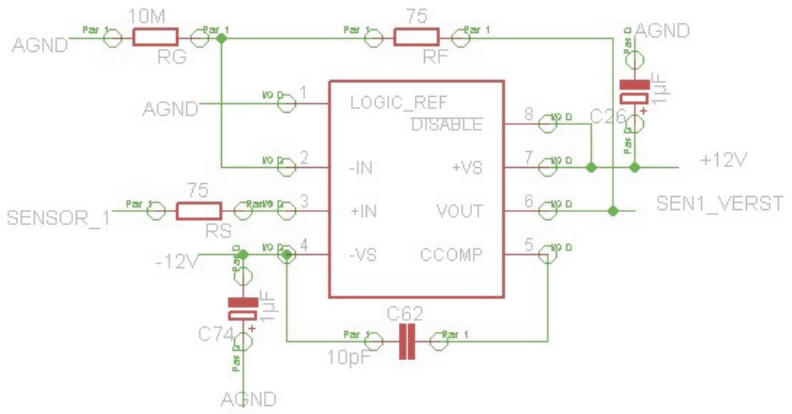


Bild 3.14.: Beschaltung des Verstärkers AD8021 [11, Seite 19,20]

der AD8021 nur noch zwei Entkopplungskondensatoren und einen Kompensationskondensator. Im Datenblatt vom AD8021 stehen die Werte für die Widerstände und den Kompensationskondensator, mit denen eine Verstärkung von Eins eingestellt wird. [11]

3.2.3. RS-232

Die in der Aufgabenstellung geforderte PC-Anbindung zum Konfigurieren der Messkarte wird mittels des RS232-Übertragungsprotokolls erreicht. Um das zu bewerkstelligen, wird der Treiberbaustein MAX232 von der Firma MAXIM eingesetzt. Er hat die Aufgabe, eine Pegelanpassung vorzunehmen. Die Signale aus der UART-Schnittstelle des Mikrocontrollers haben TTL-Pegel, aber laut der RS-232-Spezifikation wird bei einem High-Signal eine Spannung kleiner $-3\,\mathrm{V}$ und bei einem Low-Signal eine Spannung größer 3 V erwartet. Der Treiberbaustein MAX232 beinhaltet neben der Sender- und Empfängerkomponente auch zwei DCDC-Wandler, die für die Pegelanpassung benötigt werden. Er wird mit 5 V Spannung versorgt und erzeugt daraus eine Ausgangssignalspannung von $\pm 10\,\mathrm{V}$, was dem RS-232-Standard genügt.

Die zwei integrierten DCDC-Wandler benötigen als externe Beschaltung nur 4 Elektrolyt-Kondensatoren. Da bei diesen die Gefahr besteht, dass auf Grund der Druckenverhältnissen in der Stratosphäre explodieren können, werden stattdessen Tantal-Kondensatoren verwendet. Der Kondensator C1 zwischen den Pins 1 und 3 bewirkt eine Verdopplung des Spannungspegels von 5 V auf 10 V. Um die negative 10 V Spannung zu erzeugen, wird der Kondensator C2 zwischen Pin 4 und 5 am zweiten DCDC-Wandler benötigt. Die Kondensatoren C3 (V+) und C4 (V-) werden zur Stabilisierung der erzeugten Spannung benutzt. Neben dem Anpassen der Pegel besteht außerdem die Möglichkeit,

zwei RS-232-Kanäle über den MAX232 zu steuern. Die Beschaltung des Bauteils kann dem Bild 3.15 auf Seite 43 entnommen werden.[8, 21]

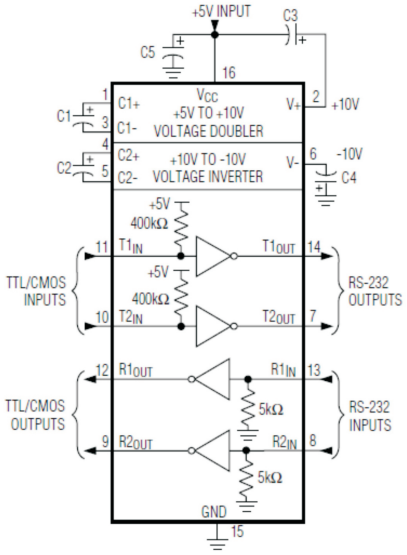


Bild 3.15.: Treiberbaustein RS-232 [21, Seite 17]

3.2.4. RS-422

Der RS-422 ist ein Schnittstellen-Standard für eine leitungsgebundene differentielle serielle Datenübertragung. Mit der RS-422 werden nur die elektrischen Eigenschaften der Schnittstelle spezifiziert. Die hier beschriebene Schnittstelle RS-422 ist für symetrische Übertragungen ausgelegt. Somit werden jeweils das positive und das invertierende Sendesignal vom Sender zum Empfänger in Form eines verdrillten Leistungspaares übertragen. Die Vorteile beim Verwenden dieses Standards sind die Minimierung der Gleichtaktstörungen und das Erzielen von höheren Datenraten als bei der Verwendung der unsymmetrischen Schnittstelle RS-232. Die Übertragung bei dieser Schnittstellenvariante ist unidirektional, d.h. die Übertragung erfolgt nur in eine Richtung über ein Leitungspaar. Zur Erfüllung der Aufgabenstellung, das Senden und Empfangen von Daten über die RS-422 Schnittstelle, werden vier Leitungen benötigt. Bei höheren Datenraten bis zu einigen MBit/s ist es erforderlich, die Leitungspaare dieser Schnittstelle RS-422 am Empfänger mit einem Abschlusswiderstand von 120 Ohm abzuschließen. In dem Bild 3.16 auf Seite 44 ist zu erkennen, dass an dem RS-422-Treiberbaustein (IC5) zwei Optokoppler (OK2, OK3) zur Erzeugung einer galvanischen Trennung auf den Datenleitungen angeschlossen sind. Der DCDC-Wandler (RO-0505S) wird benötigt, um die Stromversorgung für die RS-422-Schnittstelle ebenfalls galvanisch zu trennen. Durch die galvanische Trennung werden elektromagnetische Störungen

verhindert und die Gleichtaktstörungen von den Signaleingängen ferngehalten. citers422-data, galvanisch

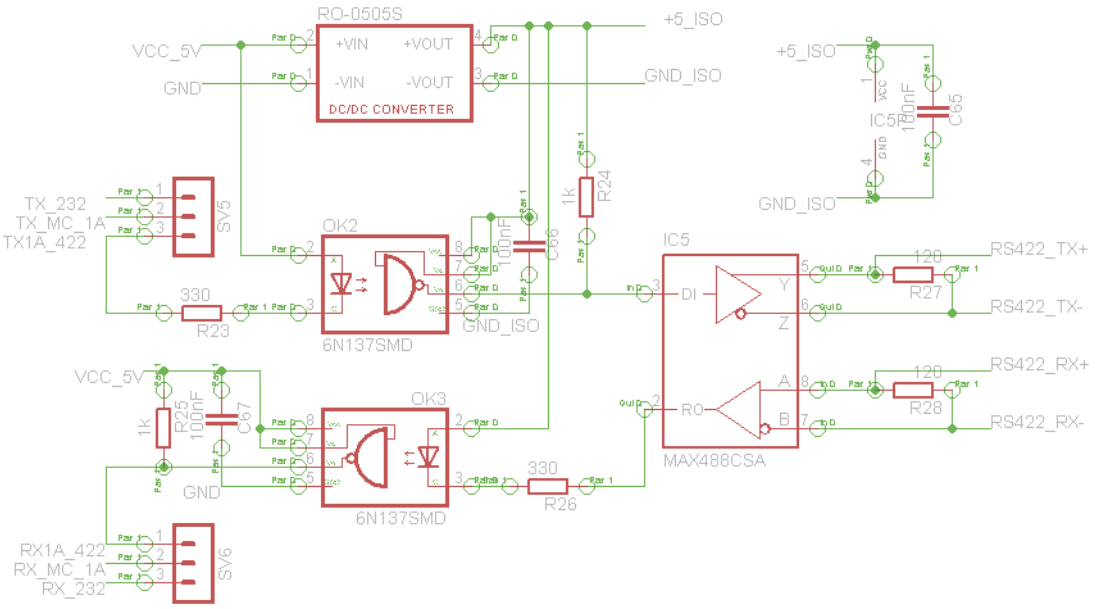


Bild 3.16.: RS422-Beschaltung [9]

3.2.5. Spannungsversorgung

Die Spannungsversorgung für die Platine erfolgt mit 4 x 3,6 V Batterien. Es werden Lithium-Thionylchlorid-Batterien von der Firma Saft eingesetzt. Dieser Batterietyp hat sich schon bei anderen Ballonflügen bewährt, da seine Spannungen ziemlich konstant bleiben und sie erst, wenn ihre Kapazität aufgebraucht ist, steil abfallen. Das Bild 3.17 auf Seite 45 zeigt die Entladekurven der Saft-Batterien. [26] Angeschlossen wird die Spannungsversorgung über den zweipoligen Molex Stecker x1. In diesem Projekt werden unterschiedliche Spannungsniveaus benötigt, dies sind die Niveaus und die Bauteile, mit der jeweils von ihnen benötigten Spannung:

3,3 V	Mikrocontroller, Temperatur- und Feuchtigkeitssensor
5 V	Rotations- und Beschleunigungssensor, externer ADU, Drucksensor
12 V	externer ADU, Vorverstärker
−12 V	externer ADU, Vorverstärker

Tabelle 3.8.: Spannungsniveaus bei der Messkarte

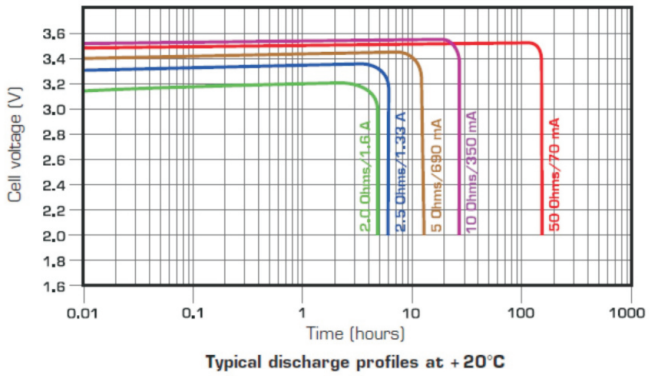
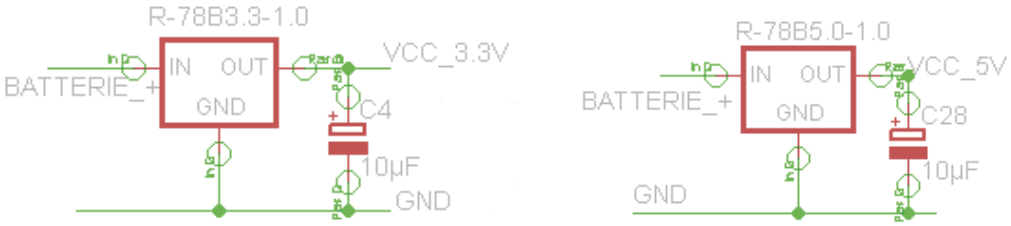


Bild 3.17.: Entladekurve der Saft-Batterien [26, Seite 2]

Um aus den 14,4 V Versorgungsspannung der Batterien die notwendigen Spannungsniveaus zu erzeugen, wurden Schaltregler, Low Drop Linear-Regler und DCDC-Wandler eingesetzt. Dabei musste darauf geachtet werden, dass die Eingangsspannungsbereiche der einzelnen Spannungsregler groß genug sind und dass sie einen hohen Wirkungsgrad haben. Bei dem Schaltregler und dem DCDC-Wandler war es wichtig, dass die Restwelligkeit, die von diesen Bauteilen erzeugt wird, so niedrig wie möglich ist. Bei dem Linear-Regler sollte ein solcher ausgesucht werden, der eine geringe Low-Drop-Spannung hat, damit nicht zu viel der Batteriespannung bereits am Bauteil abfällt.

Für die Erzeugung der Spannungen 5 V und 3,3 V wurden die Festspannungsschaltregler von der Firma Recom aus der R78B-Serie eingesetzt. Diese Schaltregler wurden ausgewählt, weil sie einen sehr hohen Wirkungsgrad von 89 % für den 3,3 V Regler und sogar 93 % für den 5 V Regler haben. Zusätzlich zur Stabilisierung der Ausgangsspannungen wurden 10 µF Tantal-Kondensatoren verwendet. [24, 25]



(a) Spannungsversorgung 3,3V (b) Spannungsversorgung 5 V

Bild 3.18.: Eagle-Schaltungen der Recom-Schaltregler

Die 12 V für die Betriebsspannungen des Vorverstärkers und des externen ADUs werden mit Hilfe eines Low Drop Linear-Reglers von STMicroelectronics aus der L4931-Serie er-

zeugt. Der Spannungsabfall dieses Reglers liegt zwischen ca. 0,4 V und 0,8 V. Auch wenn die Dropspannung im schlimmsten Fall 0,8 V beträgt, sinkt die Batteriespannung nicht unter 12 V. Ebenfalls zur Stabilisierung der Ausgangsspannung hat dieser Regler einen Tantal-Kondensator, er braucht nur einen 2,2 µF Kondensator. Zur Stabilisierung der Eingangsspannung hat dieser Regler außerdem noch einen 100 nF Keramik-Kondensator. [29]

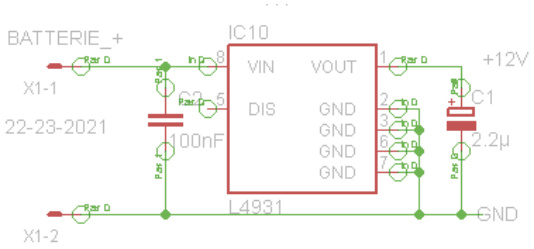


Bild 3.19.: Spannungsversorgung 12 V

Die negativen 12 V Betriebsspannungen für den Vorverstärker und den externen ADU werden mit Hilfe eines DCDC-Wandlers der TDR-3WI-Serie von Traco Power erzeugt. Zur Minimierung von EMV[9] wird die im Datenblatt empfohlene Eingangsschaltung verwendet. Dieser DCDC-Wandler hat einen Wirkungsgrad von 82 %. [23]

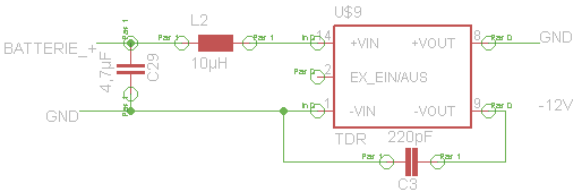


Bild 3.20.: Spannungsversorgung −12 V

3.2.6. SD-Karten-Interface

Bedingt durch die Aufgabe ist es erforderlich, die gemessenen Daten auf einer SD-Karte zu speichern. Eine SD-Karte ist ein Wechseldatenträger, der vielfältig zur Speicherung von Daten eingesetzt werden kann. Bei SD-Karten gibt es drei unterschiedliche Größen MiniSD, MicroSD und die Standard-SD. Für das Projekt wurde die Standard-Karte mit einer Größe von 32 x 24 mm verwendet. Es gibt zwei Möglichkeiten, die SD-Karte anzuschließen, entweder über den SD- oder über den SPI-Modus. Die entsprechenden Pinbelegungen bei den jeweiligen Modi sind in der Tabelle 3.9 auf Seite 47 dargestellt.

9 EMV steht für elektromagnetische Verträglichkeit

Pin	Name	Funktion	
		SD-Modus	SPI-Modus
1	**DAT3**/CS	Data Line 3	Chip Select/$\overline{SS}$
2	**CMD**/DI	Command Line	Master Out Slave In (MOSI)
3	VSS_1	Ground	Ground
4	VDD	Supply Voltage	Supply Voltage
5	CLK	Clock	Clock
6	VSS_2	Ground	Ground
7	**DAT0**/DO	Data Line 0	Master In Slave Out (MISO)
8	**DAT1**	Data Line 1	Unused
9	**DAT2**	Data Line 2	Unused

Tabelle 3.9.: Pinbelegung SD-Karte;die fett markierten Namen sind die für den SD-Modus [17, Seite 2]

Wie schon im Abschnitt „*SPI*“ auf Seite 37 kurz erwähnt wurde, kommuniziert der Mikrocontroller über eine SPI-Schnittstelle mit der SD-Karte. Somit wurden die Pins des SD-Karten-Sockels im SPI-Modus belegt. Zusätzlich besitzen die Leitungen noch Pullup-Widerstände, um definierte Pegel an den Eingangspins zu haben, wenn die SD-Karte unbenutzt ist. Mehr wird nicht benötigt, um eine SD-Karte in einer Mikrocontroller-gesteuerten Anwendung zu benutzen. Allerdings wurden zur Sicherheit in dieser Arbeit noch Pufferbausteine zwischen dem Mikrocontroller und dem SD-Karten-Sockel geschaltet. Diese Pufferbausteine erfüllen gleich zwei Aufgaben, einmal die Verstärkung der Signale auf den Leitungen und zum anderen einen Schutz für die restliche Schaltung vor Spannungsspitzen, die durch das Rausnehmen der Karte entstehen können. [18, 17],[10]

3.2.7. Spannungs- und Stromüberwachung

Die Überwachung der Versorgungsspannung der CTA/CCA-Sensoren und der Betriebsspannung erfolgt mittels des internen ADUs. Die obere Grenze des Spannungsbereichs des ADUs beträgt 3,6 V und die zu überwachenden Spannungen überschreiten diesen Wert. Deshalb können die Spannungen nicht direkt an den ADU geführt werden, sondern sie müssen erst mittels eines Spannungsteilers in den richtigen Spannungsbereich gewandelt werden. In dem Bild 3.21 auf Seite 48 ist der Spannungsteiler zu sehen, mit dem die Betriebs- und die drei CTA/CCA-Versorgungsspannungen in den richtigen Spannungsbereich gebracht werden.

Die Stromüberwachung für die CTA/CCA-Sensoren erfolgt mittels eines kleinen Strommesswiderstands, der dem R11 in dem Bild 3.22 auf Seite 48 entspricht. Die Spannung, die über diesen Messwiderstand abfällt, wird von einem Spannungsverstärker weiterverarbeitet, der

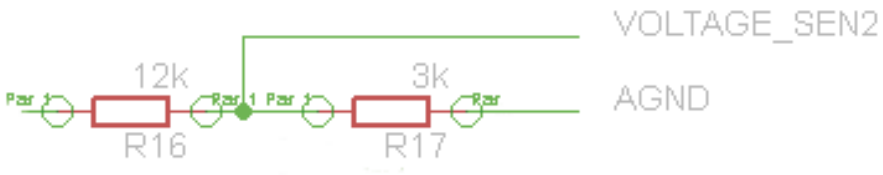


Bild 3.21.: Spannungsteiler zur Spannungsüberwachung

eine interne Verstärkung hat. In diesem Projekt wird ein Spannungsverstärker von der Marke Maxim mit der Bezeichnung MAX4080 eingesetzt. Es gibt diesen Verstärker in drei verschiedenen Varianten mit den Verstärkungen 5 V/V, 20 V/V und 60 V/V. Vom Verstärkungsfaktor hängt der maximale Spannungsbereich ab, der über den Messwiderstand abfallen darf, siehe Tabelle 3.10 auf Seite 48. Der maximale Strom, der von den CTA/CCA-Sensoren benötigt wird, beträgt ungefähr 1 A. Um die Verlustleistung über den Messwiderstand gering zu halten, wurde hier der Typ mit der 60 V/V-Verstärkung gewählt. [22] Die Spannung, die von dem

Verstärkung	maximaler Spannungsbereich
5V/V	1000 mV
20V/V	250 mV
60V/V	100 mV

Tabelle 3.10.: Verstärkung Strom [22, Seite 1,2]

Spannungsverstärker erzeugt wird, wandelt ebenfalls der interne ADU in einen digitalen Wert um.

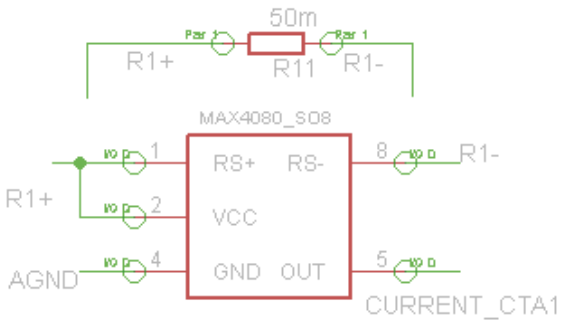


Bild 3.22.: Schaltung für die Stromüberwachung der CTA/CCA-Sensoren

3.3. Software

Die Software für die Messkarte besteht aus dem Hauptprogramm, den Interrupt Service Routinen (ISR) und weiteren Funktionen, die für die Verarbeitung der Daten zuständig sind. Damit die gemessenen Daten später auch ordentlich weiterverarbeitet und auf die SD-Karte gespeichert werden können, gibt es für jedes Messintervall eine eigene Datenstruktur. Die Messintervalle sind die 125 µs für die CTA/CCA-Sensordaten, die 10 ms für die Informationen der Lagesensoren (ADIS-Sensor und Magnetometer) und das Messintervall für die Überwachungsdaten, die jede Sekunde einmal aufgenommen werden sollen. Nachfolgend sind die Datenstrukturen mit ihren Komponenten aufgeführt. Die Datenstruktur für die CTA/CCA-Sensoren zeigt die Tabelle 3.11 auf Seite 49:

Name	Größe	reservierter Speicher
CTA/CCA-Sensor 1	16 Bit	2 Byte
CTA/CCA-Sensor 2	16 Bit	2 Byte
CTA/CCA-Sensor 3	16 Bit	2 Byte

Tabelle 3.11.: Datenstruktur für Messintervall 125 µs

Die zweite Datenstruktur beinhaltet die Informationen der Lagesensoren, die in der Tabelle 3.12 auf Seite 49 noch einmal genauer aufgeführt sind:

Name	Größe	reservierter Speicher
Rotation X-Achse	14 Bit	2 Byte
Rotation Y-Achse	14 Bit	2 Byte
Rotation Z-Achse	14 Bit	2 Byte
Beschleunigung X-Achse	14 Bit	2 Byte
Beschleunigung Y-Achse	14 Bit	2 Byte
Beschleunigung Z-Achse	14 Bit	2 Byte
Magnetfeld X-Achse	16 Bit	2 Byte
Magnetfeld Y-Achse	16 Bit	2 Byte
Magnetfeld Z-Achse	16 Bit	2 Byte
Zeitstempel	32 Bit	4 Byte
Flag zur Anzeige der Vollständigkeit der Lagedaten	2 Bit	1 Byte
Flag der ADIS-Variante	1 Bit	1 Byte

Tabelle 3.12.: Datenstruktur für Messintervall 10 ms

Das Flag „Lagedaten fertig“ zeigt an, dass die beiden Sensoren, der Initialsensor

(ADIS-Sensor) und das Magnetometer (MicroMag3), mit ihren Messungen für die Lagebestimmung fertig sind. Dafür werden von dem reservierten Byte nur die ersten zwei Bits benötigt, das erste Bit für den MicroMag3 und das zweite für den ADIS-Sensor. Mit dem Flag „ADIS-Variante“ wird unterschieden, ob nur die Lagedaten, also Beschleunigungs- und Rotationsdaten, oder auch die Überwachungsdaten gemessen werden sollen.

Die letzte Struktur, Tabelle 3.13 auf Seite 49 zeigt, die Überwachungsdaten.

Name	Größe	reservierter Speicher
Temperaturwert SHT	14 Bit	2 Byte
Feuchtigkeitswert SHT	12 Bit	2 Byte
Überwachung Batteriespannung	10 Bit	2 Byte
Drucksensorwert	10 Bit	2 Byte
Spannungsüberwachung CTA/CCA-Sensor 1	10 Bit	2 Byte
Spannungsüberwachung CTA/CCA-Sensor 2	10 Bit	2 Byte
Spannungsüberwachung CTA/CCA-Sensor 3	10 Bit	2 Byte
Stromüberwachung CTA/CCA-Sensor 1	10 Bit	2 Byte
Stromüberwachung CTA/CCA-Sensor 2	10 Bit	2 Byte
Stromüberwachung CTA/CCA-Sensor 3	10 Bit	2 Byte
Temperatur X-Achse vom ADIS	12 Bit	2 Byte
Temperatur Y-Achse vom ADIS	12 Bit	2 Byte
Temperatur Z-Achse vom ADIS	12 Bit	2 Byte
Versorgungsspannung vom ADIS	12 Bit	2 Byte
Sekundenzähler	16 Bit	2 Byte
Zeitstempel	32 Bit	4 Byte
Flag zur Anzeige der Vollständigkeit der Überwachungsdaten	2 Bit	1 Byte

Tabelle 3.13.: Datenstruktur für Messintervall 1 s

Um anzuzeigen, dass die Überwachungsdaten aufgenommen wurden, ist das Flag „Überwachungsdaten fertig“ verantwortlich. Das erste Bit signalisiert, dass der Temperatur- und Feuchtigkeitssensor mit der Messung fertig ist und das zweite Bit, dass der interne ADU seine Wandlung beendet hat und die Daten ausgelesen wurden.

Mit dem Ausdruck Zeitstempel in den Datenstrukturen für die Lage- und die Überwachungsdaten ist der Zähler Aufrufe des Timer 2 Interrupts gemeint. Dieser Interrupt wird in dem Abschnitt „Timer 2“ auf Seite 54 genauer beschrieben.

Mit Hilfe dieser Strukturen werden drei FIFOs erzeugt. Die Größen dieser FIFOs können der Tabelle 3.14 auf Seite 51 entnommen werden und geben an, wie viele Elemente die FIFOs haben. Jedes Element besteht aus der jeweiligen Datenstruktur.

Name	Größe
CTA/CCA-FIFO (Wind-FIFO)	7000
Lagedaten-FIFO	100
Überwachungsdaten-FIFO	20

Tabelle 3.14.: FIFO-Größen

3.3.1. Hauptprogramm

Das Hauptprogramm besteht aus mehreren funktionalen Einheiten: der Initialisierung, der Datenaufbereitung und Speicherung und der Statusübermittlung. In der Initialisierung werden die I/O-Ports und die Peripheriegeräte initialisiert, die in diesem Projekt benötigt werden. Außerdem wird auch der erste Sektor in der Datei auf der SD-Karte bestimmt. Die Ermittlung des ersten freien Sektors kann erst erfolgen, wenn eine SD-Karte vorhanden ist und sowohl die Checks als auch die Initialisierung des Dateisystems erfolgreich waren. Ebenfalls gehört eine Pause von 150 ms zur Initialisierung für den Start-up des Rotations- und Beschleunigungssensors (ADIS-Sensor). Zum Abschluss der Initialisierung werden die Interrupts freigegeben (auf die einzelnen Interrupts wird im Abschnitt 3.3.2 „*Interrupt*“ auf Seite 54 näher eingegangen). Mittels der Interrupts werden die angeschlossenen Sensoren ausgelesen. In dem Bild 3.23 auf Seite 52 ist das Struktogramm zur Initialisierung dargestellt.

Anschließend werden die gemessen Daten in einer Endlosschleife so lange auf die SD-Karte gespeichert, wie verfügbare Sektoren vorhanden sind. Dazu werden sekündlich die gemessenen Werte von den Sensoren mittels der UART-Schnittstelle (UART 1B) als Statusinformationen übermittelt. Damit die aufgenommenen Werte beim Auswerten richtig interpretiert werden, müssen die Daten, bevor sie auf die Karte geschrieben werden, erst in ein Datenformat gebracht werden. Dabei gibt es drei verschiedene Arten für das Datenformat: eins für die CTA/CCA-Sensordaten mit den Lagedaten, eins für die CTA/CCA-Daten und eins für die Überwachungsdaten. Diese Trennung wurde deshalb vorgenommen, weil genau einmal Lagedaten mit 80 CTA/CCA-Daten in einen Sektor passen. Sollte allerdings der Fall eintreten, dass nach 80 CTA/CCA-Werten noch kein Lagedatensatz vorhanden ist, wird dieser Sektor nur mit CTA/CCA-Daten beschrieben. Bei dem dritten Datenformat werden erstmal 12 Überwachungsdatenpakete gesammelt und dann auf die SD-Karte geschrieben. Die Tabelle 3.15 auf Seite 54 zeigt, wie so ein Datenpaket aussieht.

Durch das interne Flag „neue Daten“ wird dem Hauptprogramm signalisiert, dass neue Daten vorhanden sind. Daraufhin werden die neuen Daten in das jeweilige Datenformat geschrieben und dies wird solange wiederholt, bis das Sektorarray keinen Platz für neue Daten hat. Anschließend werden die Daten auf die SD-Karte geschrieben. Das Struktogramm in Bild 3.24 auf Seite 53 zeigt den Programmcode für die Aufbereitung und Speicherung der

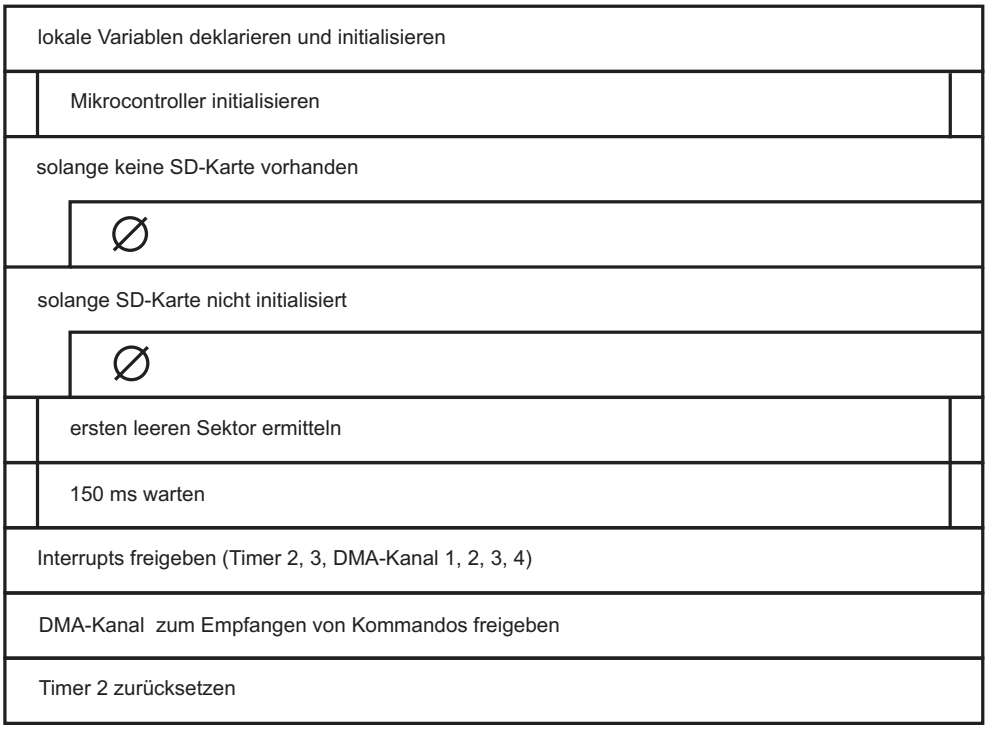


Bild 3.23.: Struktogramm Initialisierungsbereich

CTA/CCA- und der Lagedaten. Die Programmstruktur zur Aufbereitung und Speicherung der Überwachungsdaten ist die gleiche.

Damit die FIFOs für die Sensordaten nicht volllaufen, speziell die für die Wind- und Lagedaten, wird am Ende des Hauptprogramms, bevor die Endlosschleife von vorn beginnt, noch die Position der Schreibe- und der Lesezeiger verglichen. Sollte eine Differenz bei den Windzählern vorhanden sein, sind noch oder schon wieder Winddaten im Fifo vorhanden, die aufbereitet werden können. Das hat zur Folge, dass das Flag für „neue Daten“ und das Flag für „neue Winddaten“ nochmals gesetzt werden. Zusätzlich wird noch geprüft, ob eine Differenz zwischen dem Schreibe- und Lesezeiger der Lagedaten vorhanden ist. Da auf Grund der vorliegenden Implementierung der FIFOs die Schreibe- und Lesezähler wieder von vorn anfangen, wenn das Ende des FIFO-Puffers erreicht wurde, muss geprüft werden, welcher von beiden den größeren Wert hat. Anschließend wird geprüft, ob mehrere Lagedaten in dem Lagedaten-FIFO vorhanden sind. Sollte das der Fall sein, wird zusätzlich noch das Flag für „neue Lagedaten“ gesetzt. Das Bild 3.25 auf Seite 53 zeigt das Struktogramm zu dieser Sicherheitsroutine.

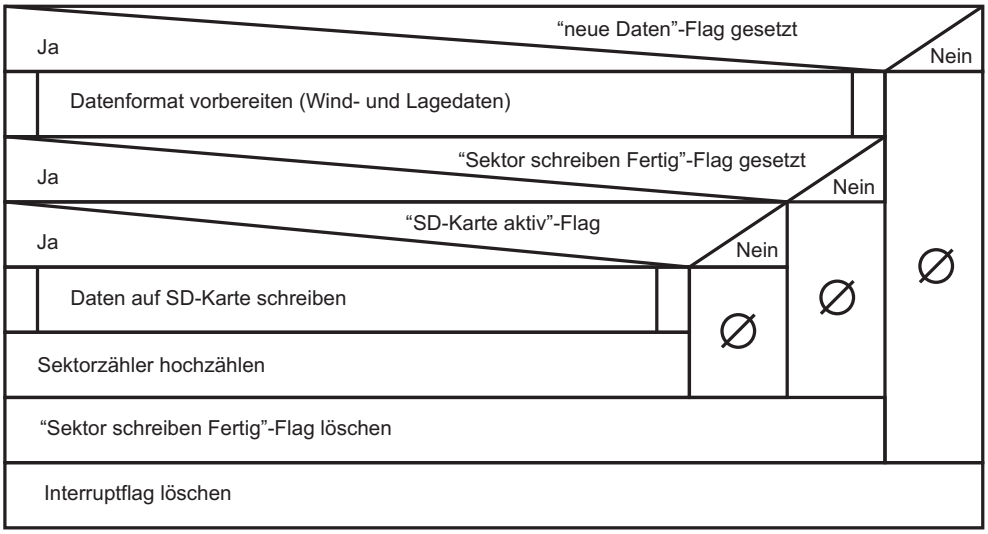


Bild 3.24.: Struktogramm für die Aufbereitung und Speicherung der CTA/CCA- und Lagedaten

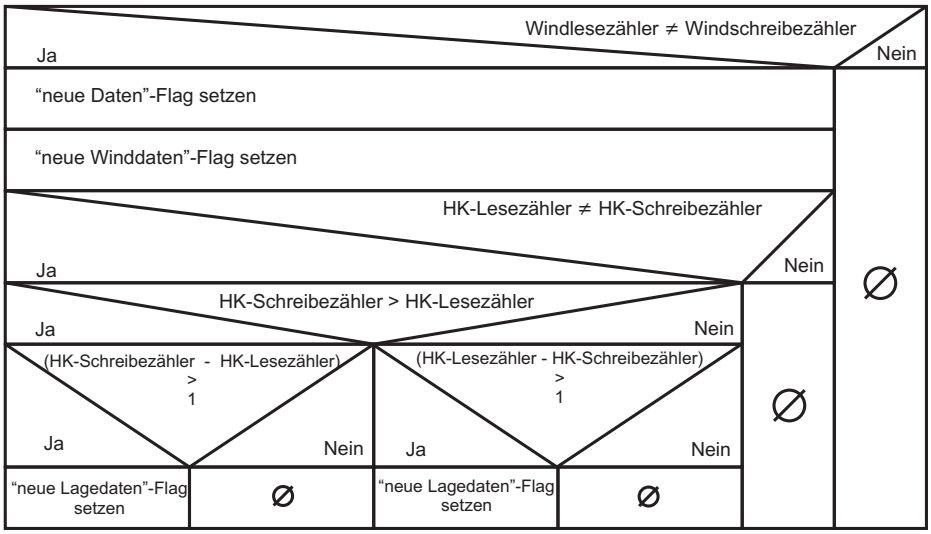


Bild 3.25.: Struktogramm FIFO-Check (HK steht für Housekeeping und damit sind die Lagedaten gemeint)

Name	Größe
ADIS Temperatur X-Achse	16 Bit
ADIS Temperatur Y-Achse	16 Bit
ADIS Temperatur Z-Achse	16 Bit
ADIS Versorgungsspannung	16 Bit
Batterie-Betriebspannung	16 Bit
Drucksensor-Signal	16 Bit
CTA/CCA-Batteriespannung 1	16 Bit
CTA/CCA-Batteriespannung 2	16 Bit
CTA/CCA-Batteriespannung 3	16 Bit
Stromverbrauch CTA/CCA 1	16 Bit
Stromverbrauch CTA/CCA 2	16 Bit
Stromverbrauch CTA/CCA 3	16 Bit
Temperaturwert SHT-Sensor	16 Bit
Feuchtigkeitswert SHT-Sensor	16 Bit
Relaisstatus	8 Bit
RTC-Time	24 Bit
Sekundenzähler	16 Bit
Zeitstempel	32 Bit

Tabelle 3.15.: Überwachungsdatenpaket

3.3.2. Interrupt

In dieser Arbeit werden insgesamt sieben Interrupts benötigt. Der Interrupt des Timers 2 ist der wichtigste der sieben Interrupts, da er die 8 kHz Abtastung erzeugt. Zusätzlich erledigt dieser Interrupt noch andere Aufgaben, diese sind im Bild 3.26 auf Seite 55 aufgeführt. Die Kommunikation mit dem Temperatur- und Feuchtigkeitssensor (SHT) erfolgt über den Timer 3 Interrupt. Die zu überwachenden Daten werden über den internen ADU-Interrupt ermittelt. Die letzten vier Interrupts erzeugt der DMA-Controller des PIC32. Jeder der einzelnen DMA-Transfers löst einen Interrupt aus. Die Interrupt-Routinen werden nachfolgend beschrieben.

Timer 2

Die Interrupt Service Routine des Timers 2 ist das Herzstück der Software für die neue Messkarte, denn mittels dieser ISR wird das Auslesen der Sensoren gesteuert und ausgeführt. Um dies zu erreichen, ist die Interrrupt Service Routine in mehrere Bereiche eingeteilt. Es gibt einmal den Bereich, der sich um das Auslesen des externen ADUs kümmert, zum

anderen die Bereiche zur Bestimmung des Startpunkts der Messung der Lagesensoren und der Überwachungssensoren. Ein weiterer Bereich ist die Bestimmung der Messintervalle, also wann die 10 ms und die 1 s abgelaufen sind. Dann gibt es noch die Bereiche, die die Routinen zum Auslesen der Lagesensoren beinhalten, also die des Magnetometers und des ADIS-Sensors. Ein Bereich legt fest, was passieren soll, wenn die Lagesensoren mit ihrer Messung fertig sind. Ebenfalls gibt es einen kleinen Bereich, der sich um das Auslesen des Temperatur- und Feuchtigkeitssensors (SHT) kümmert. Das Bild 3.26 auf Seite 55 zeigt die eben beschriebenen Bereiche als Übersicht in der Reihenfolgee, wie sie im Programmcode implementiert wurde. Im Nachfolgenden wird auf die eben aufgeführten Bereiche noch genauer eingegangen.

Routine zum Auslesen des Externen ADUs Der wichtigste Bereich ist der, der sich um das Auslesen des externen ADUs kümmert, denn die Timer 2 Interrupt Routine wird alle 125 µs aufgerufen, was 8 kHz, mit denen das CTA/CCA-Signal abgetastet werden soll, entspricht. Damit der ADU eine Wandlung durchführt, benötigt er ein Startsignal, das er durch die CONVST-Leitungen erhält. Jede CONVST-Leitung ist einem Paar der sechs vorhande-

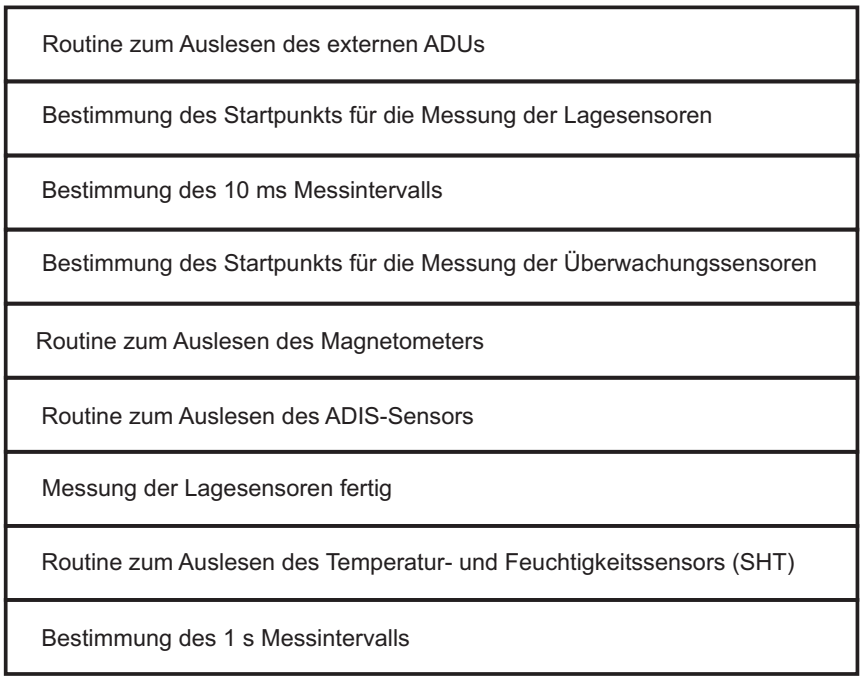


Bild 3.26.: Reihenfolge der Bereiche in der Timer 2 ISR

nen Kanäle zugeordnet, wie im Abschnitt 3.2.2 „*Externer ADU*“ auf Seite 38 beschrieben wurde. Es besteht die Möglichkeit, dass nur zwei Kanäle oder vier oder sogar alle 6 Kanäle gleichzeitig gewandelt werden. Bei der Variante, bei der vier Kanäle gewandelt werden, spielt es keine Rolle, ob die vier Kanäle hintereinander sind oder es eine Lücke gibt, es können also beispielsweise auch die Kanäle 1, 2, 5 und 6 gleichzeitig gemessen werden. In dieser Arbeit werden die Kanäle 1, 2, 3 und 4 oder 3, 4, 5 und 6 gleichzeitig gemessen, je nachdem, ob mit oder ohne den Vorverstärker gemessen werden soll. Dabei werden die Werte des 4 oder 3 Kanals als Dummy-Daten behandelt.

Nachdem das Startsignal an den ADU übermittelt wurde, reagiert er darauf mit dem High-Setzen der Busy-Leitung, was bedeutet, dass er gerade eine Wandlung durchführt und somit beschäftigt ist. Wenn der externe ADU mit der Messung der ausgewählten Kanäle fertig ist, was laut Datenblatt nach ca. 3 µs der Fall sein sollte, wird die Busy-Leitung auf Low gezogen. In der Routine wird nach dem Senden des Startsignals darauf gewartet, dass die Busy-Leitung wieder einen Low-Pegel hat, bevor mit dem Auslesen der Werte der einzelnen Kanäle begonnen wird.

Davor gibt es noch einen Check, ob das FIFO für die Winddaten schon voll ist, um zu verhindern, dass ungelesene Daten überschrieben werden. Um die Daten des externen ADUs auszulesen, müssen die $\overline{CS}$-Leitung und die $\overline{RD}$-Leitung auf Low gesetzt werden. Die Daten werden von dem Mikrocontroller mittels seines parallelen Ports abgeholt. Der Mikrocontroller unterstützt dabei das Setzen der $\overline{RD}$-Leitung. Um die richtigen Daten vom parallelen Port zu erhalten, muss immer ein Dummy-Lesevorgang vor dem Auslesen des ADUs erfolgen, da immer der Wert des vorherigen Lesevorgangs aus dem *Parallel Port Data Input-Register (PMDIN)* ausgelesen wird. Somit werden bei der Variante ohne den Vorverstärker beim ersten Auslesen des PMDIN-Registers dessen Default-Werte ausgelesen, ansonsten immer der Wert, der von dem vierten Kanal gemessen wurde. Bei der Variante mit dem Vorverstärker ist es immer der Wert des dritten Kanals und somit muss keine besondere Unterscheidung in der Software gemacht werden, unabhängig davon welche Variante gerade ausgeführt wird.

Damit ist die Routine zum Auslesen des externen ADUs beschrieben. Jede Sekunde soll der aktuelle Status von den CTA/CCA-Sensoren übermittelt werden. Um das zu bewerkstelligen, wird jeder 8000ste Wert, der gemessen wurde, in einer gesonderten Struktur gespeichert. Auf diese Struktur wird in dem Abschnitt 3.3.3 „*Status senden*“ auf Seite 69 noch genauer eingegangen. Zusätzlich wird noch geprüft, ob das gemessene Signal in dem richtigen Messbereich liegt. Das CTA/CCA-Signal muss in dem Bereich zwischen 0,8 V und 1,9 V liegen, ansonsten wäre der Sensor defekt.

Bestimmung des Startpunkts für die Messung der Lagesensoren Damit ist der erste Bereich der Timer 2 Interrupt Service Routine beendet und es wird überprüft, ob es schon

Zeit ist, die Messung für die Lagesensoren zu starten. Dies ist der zweite Bereich in der ISR. Die Überprüfung erfolgt durch zwei interne statische Zähler. Dabei wird das Magnetometer knapp 2,5 ms vor dem 10 ms Messintervall gestartet, damit der Sensor auch mit seiner Messung von allen drei Achsen fertig ist. Bei dem ADIS-Sensor muss noch unterschieden werden, ob nur die Lagedaten, also Beschleunigungs- und Rotationsdaten, oder auch noch die Überwachungsdaten gemessen werden sollen. Von der Wahl der Variante hängt der Startzeitpunkt der Messung ab. In der Zeit, in der der ADIS-Sensor seine Kalibrierung durchführt, werden für ca. 30 s keine Lagedaten aufgenommen.

Bestimmung der Messintervalle Die Bestimmung der Messintervalle erfolgt mittels eines globalen Zählers, der die Aufrufe dieses Interrupts zählt. Da dieser Interrupt alle 125 µs aufgerufen wird, kann mit Hilfe der Modulo-Funktion der Zähler durch 80 und durch 8000 geteilt werden. Wenn bei den Teilungen der Rest den Wert Null hat, sind genau die 10 ms (Teiler 80) oder die 1 s (Teiler 8000) vergangen. Im nachfolgenden wird beschrieben, was in den jeweiligen Bereichen passiert.

10 ms Messintervall und Bestimmung des Startpunkts der Messung der Überwachungssensoren In dem 10 ms Bereich werden die statischen Zähler für die Bestimmung des Startpunkts der Lagesensoren zurück auf Null gesetzt. Außerdem wird, wenn der ADIS-Sensor kalibriert wird, hier geprüft, ob die 30 s vergangen sind und nun wieder neue Lagedaten aufgenommen werden können. Zusätzlich gibt es hier noch den anderen schon erwähnten Bereich, der bestimmt, wann mit dem Starten der Überwachungssensoren begonnen werden soll. Dafür gibt es auch hier wieder einen statischen Zähler, der zählt, wie oft bereits 10 ms vergangen sind. Der Temperatur- und Feuchtigkeitssensor benötigt ca. 300 ms, bis er beide Messungen erledigt hat, genaueres zum SHT-Sensor ist im Abschnitt 3.1.4 „*Temperatur- und Feuchigkeitssensor*“ auf Seite 23 beschrieben. Ebenfalls wird in diesem Bereich, in dem die Überwachung gestartet wird, das ASAM gesetzt. Damit wird der interne ADU gestartet.

1 s Messintervall In dem zweiten Messintervall, dem 1 s Bereich, wird ein globaler Sekundenzähler erhöht und in das aktuelle Überwachungselement abgespeichert. Zusätzlich wird hier geprüft, ob alle Überwachungssensoren fertig gemessen haben und somit die Überwachungsdaten schon vorhanden sind. Sollte dies der Fall sein, wird das Flag „neue Überwachungsdaten“ gesetzt und im Hauptprogramm kann die Funktion zum Aufbereiten der Überwachungsdaten ins entsprechende Datenformat aufgerufen werden. Abschließend wird dann der Schreibezähler des Überwachungs-FIFOs um einen erhöht und das Statusflag “Überwachungsdaten vorhanden“ gesetzt.

Messung der Lagesensoren fertig In diesem Teil der Interrupt Routine wird geprüft, ob die Lagesensoren ihre Messungen beendet haben. Wenn das der Fall ist, wird der aktuelle Wert des globalen Zählers, der diese Interrupt-Aufrufe zählt, in das aktuelle FIFO-Element für die Lagedaten gespeichert. Anschließend wird der Schreibezeiger auf das nächste Element in dem Lagedaten-FIFO gesetzt. Zusätzlich gibt es noch eine Prüfung, ob die aktuell gemessenen Lagedaten auch als Statusdaten übermittelt werden sollen. Daraufhin wird, wenn die Daten übertragen werden sollen, das Statusflag „Lagedaten vorhanden“ gesetzt, das signalisiert, dass auch die Lagedaten in der Statusstruktur vorhanden sind. Abschließend wird in diesem Bereich das steuernde Flag „neue Lagedaten vorhanden“ gesetzt.

Die nachfolgenden Bereiche beschreiben, wie die Routinen zum Auslesen des Magnetometers und des ADIS-Sensors aussehen und was passiert, wenn das Startsignal zum Messen des Temperatur- und Feuchtigkeitssensors erfolgt.

Routine zum Auslesen des Temperatur- und Feuchtigkeitssensors (SHT) Der Bereich, der sich um den Temperatur- und Feuchtigkeitssensor in der ISR kümmert, ist in zwei Teile unterteilt, in den Start- und in den Ergebnisteil. Die Aufgabe des Startteils ist es, den Timer 3-Interrupt freizugeben und steuernde Signale zur Kommunikation zwischen diesen beiden Interrupt Service-Routinen zu setzen. Die genaue Beschreibung zu der Interrupt-Routine des dritten Timers kann dem Abschnitt 3.3.2 „*Timer 3*“ auf Seite 60 entnommen werden. Zusätzlich wird der Zähler des dritten Timers zurückgesetzt. Abschließend wird noch ein Flag „Timer 3 Interrupt freigegeben“ gesetzt, das anzeigt, dass der Timer-Interrupt schon freigegeben wurde.

In dem Ergebnisteil des Temperatur- und Feuchtigkeitssensorbereichs wird bei jedem Aufruf dieser Interrupt-Routine kontrolliert, ob der Sensor bereits mit seiner Messung fertig ist. Ist dieser Check erfolgreich, werden die gemessenen Temperatur- und Feuchtigkeitswerte aus der Datenstruktur des SHT-Sensors in das aktuelle Überwachungselement übertragen. Ebenfalls wird der Interrupt für den Timer 3 wieder deaktiviert und das Flag “Timer 3 Interrupt freigegeben“ gelöscht. Zusätzlich werden die gemessenen Temperatur- und Feuchtigkeitswerte in die Statusstruktur gespeichert. Abschließend wird das Flag „SHT-Daten vorhanden“ gesetzt, dass die Daten von dem SHT-Sensor vorhanden sind. Das Flag „Starte SHT-Messung“, das signalisiert, dass eine Messung des SHT-Sensors erfolgen soll, wird nach erfolgreicher Messung gelöscht.

Routine zum Auslesen des Magnetometers Die Umsetzung des Kommunikationsprotokolls zwischen dem Mikrocontroller und dem MicroMag3-Sensor erfolgt mittels dieses Bereichs der Interrupt Routine, zwei DMA-Kanälen (1 und 2) und einer der SPI-Schnittstellen (SPI 3A). Die Ausleseroutine ist in zwei Abschnitte unterteilt, in den

Bereich, in dem das Kommando-Byte zum Magnetometer gesendet wird, und in den Bereich, der kontrolliert, ob die Messung schon beendet wurde und die Daten abgeholt werden können. Zur Unterscheidung, welcher Abschnitt gerade abgearbeitet werden muss, gibt es auch hier interne Steuerflags, die signalisieren, dass z. B. die Kommunikation mit dem Sensor begonnen hat und somit die Slave Select Line den Low-Pegel hat, damit nicht bei jedem nachfolgenden Aufruf dieser Interrupt Routine der Pegel der Slave Select-Leitung geändert wird, sondern erst wenn alle Messungen beendet sind. Zusätzlich gib es noch das Steuerflag „Kommando-Byte senden“, das unterscheidet, ob gerade der Abschnitt „senden Kommando-Byte“ bearbeitet wird oder das Ergebnis von der gerade angeforderten Messung abgeholt werden kann. Im ersten Abschnitt wird, wie vom Protokoll des Sensors gefordert, nachdem die SSNOT-Leitung auf Low-Pegel gesetzt wurde, vor jedem Kommando-Byte ein Reset-Puls übermittelt. Daraufhin wird das Kommando-Byte an die Startadresse für die DMA-Übertragung gesetzt. Die Zieladresse der DMA-Übertragung ist das *SPI-Transmit-Buffer-Register*. Der Vorteil der Übertragung mittels des DMA-Controllers des Mikrocontrollers ist, dass die Übertragung nicht von dem Prozessorkern erfolgen muss. Damit kann die CPU in der Zeit schon wieder etwas anderes erledigen. Mehr zu dem, was die DMA-Kanäle 1 und 2 machen, kann dem Abschnitt „*DMA1 und DMA2*“ auf Seite 63 entnommen werden.

Im zweiten Abschnitt wird immer wieder die DRDY-Leitung kontrolliert, ob der Sensor seine Messung schon beendet hat. Wenn das der Fall ist, müssen zwei Dummy-Bytes über die SPI-Schnittstelle an den Sensor übermittelt werden, um das 16 Bit große Ergebnis abzuholen. Die Dummy-Bytes sind nötig, damit die geforderten 16 Takte zum Auslesen erzeugt werden. Die gesendeten Daten werden dann mit dem zweiten DMA-Kanal empfangen. Ist die Messung noch nicht beendet, wird erst beim nächsten Aufruf dieser Interrupt-Routine wieder geprüft, ob die Messung fertig ist.

Bei der gewählten Messdauer mit dem Teiler 32 (siehe Tabelle 3.2 auf Seite 21), vergehen ungefähr 4 bis 5 Interrupt-Aufrufe nach der Übermittlung des Kommando-Bytes, bis die DRDY-Leitung auf High-Pegel geht. Dies ist der Teil, den diese Interrupt Routine zum Auslesen des Magnetometers beiträgt; der Teil über DMA folgt in dem Abschnitt „*DMA1 und DMA2* auf Seite 63.

Routine zum Auslesen des ADIS-Sensors Der letzte Bereich in dieser Interrupt Service Routine beschäftigt sich mit dem Auslesen des ADIS-Sensors. Dieser Bereich wird erst abgearbeitet, wenn das interne Startflag „ADIS messen“ zum Messen des Rotations- und Beschleunigungssensors gesetzt wurde. Ansonsten wird bei ISR-Aufruf diese Ausleseroutine nicht beachtet. Die Kommunikation mit dem ADIS-Sensor erfolgt über die SPI-Schnittstelle (SPI 1). In der Ausleseroutine wird zuerst das Flag „ADIS-Variante“ geprüft, welche der zwei Varianten ausgeführt werden soll. d. h. ob von dem ADIS-Sensor

nur die Lagedaten oder die Lage- und die Überwachungsdaten ermittelt werden sollen. In Abhängigkeit davon, welche Variante bearbeitet wird, werden unterschiedliche Start- und Endadressen der Ausgaberegister des ADIS-Sensors eingestellt. Danach wird, wie im Abschnitt 3.1.2 „*Rotations- und Beschleunigungssensor*“ auf Seite 15 erwähnt, der SPI-Port durch die $\overline{CS}$-Leitung des Sensors freigegeben, damit die Kommunikation mit dem Sensor erfolgen kann. Zusätzlich wird durch ein weiteres internes Flag „ADIS Messung gestartet“ verhindert, dass die Start- und Endadressen sich nochmal verändern, während die Messung läuft, da dadurch dieser Teil der Ausleseroutine beim nächsten Aufruf dieser ISR nicht noch mal abgearbeitet wird, sondern erst, wenn eine neue Messung gestartet wird. Nachdem der SPI-Port des Sensors aktiviert ist, kann, wie in dem Bild 3.5 auf Seite 18 gezeigt, die Übermittlung der Adresse des Registers erfolgen. Damit wäre der erste Teil des Lesevorganges und das Übertragen der Adresse, so wie es in der Software umgesetzt wurde, beschrieben.

Der Rotations- und Beschleunigungssensor arbeitet im Full Duplex-Betrieb und die SPI-Schnittstellen des Mikrocontrollers können gleichzeitig Daten senden und empfangen. Dadurch muss nach dem Senden der ersten Adresse erstmal das Empfangsregister der SPI-Schnittstelle (SPI 1) des Mikrocontrollers ausgelesen und der Inhalt als ein Dummy-Wert abgespeichert werden. Zur Erkennung, welche Adresse gerade übermittelt wurde und welche Daten der Mikrocontroller gerade empfangen hat, gibt es eine Zustandsvariable, die dann überprüft wird. Anschließend wird kontrolliert, ob die übermittelte Adresse für die Lagedaten oder eine für die Überwachungsdaten war. Wenn nur die Lagedaten gemessen werden, sind insgesamt 7 Aufrufe für diese Daten nötig. Sollen allerdings auch die Überwachungsdaten gemessen werden, so sind es insgesamt 11 Aufrufe.

Timer 3

Die Aufgabe der Interrupt Service Routine (ISR) des Timers 3 ist es, den Takt zu erzeugen, mit dem der SHT-Sensor betrieben werden soll, und die passenden Pegel auf die Datenleitung zu schalten, damit das Kommunikationsprotokoll, das im Abschnitt 3.1.4 „*Temperatur- und Feuchtigkeitssensor*“ auf Seite 23 beschrieben wurde, erfüllt wird. Um das zu bewerkstelligen, ist die ISR in drei Bereiche aufgeteilt, einmal in den Bereich, in dem der Mikrocontroller Daten zum SHT-Sensor schickt, dann in den Bereich, in dem geprüft wird, ob der Sensor fertig gemessen hat, und in den Bereich, in dem der Controller den gemessenen Wert vom Sensor einliest. Durch interne Flags wird in der ISR unterschieden, welcher der Bereiche gerade abzuarbeiten ist. Bei jedem Aufruf der ISR wird eine Zustandsvariable erhöht, die dann in einer switch/case-Programmstruktur abgefragt wird, und daraufhin die Pegel auf den Leitungen zum passenden Zustand geschaltet. Die Zustandsvariable spielt nur in dem ersten und letzten Bereich eine Rolle und hat auch nur da Auswirkungen auf die Zustände der Portleitungen. Nachdem der Controller die Startsequenz und den Befehl, was gemessen werden

soll, übermittelt hat, muss die Richtung der Datenleitung auf Eingang schaltet werden, damit der Mikrocontroller das Acknowledge empfangen kann und mitbekommt, ob der Sensor den Befehl verstanden hat. Das Umschalten der Datenleitung erfolgt, wie im Abschnitt IO-Port auf Seite 35 beschrieben, durch Beschalten des entsprechenden Bits im TRIS-Register. Wenn der Befehl erkannt wurde, geht es in den zweiten Bereich der Interrupt Service Routine, und die Datenleitung wird von dem Sensor auf High gesetzt. Nachdem der Sensor durch Herunterziehen der Datenleitung auf Low signalisiert hat, dass er fertig gemessen hat, geht es in den dritten Bereich der ISR. In diesem Bereich wird der gemessene Wert eingelesen und in einer speziellen Datenstruktur zwischengespeichert. Nachfolgend ist der Aufbau dieser Datenstruktur dargestellt:

- Feuchtigkeitswert LSB (8 Bit)
- Feuchtigkeitswert MSB (8 Bit)
- Temperaturwert LSB (8 Bit)
- Temperaturwert MSB (8 Bit)

Wie im Abschnitt 3.1.4 „*Temperatur- und Feuchtigkeitssensor*“ auf Seite 23 beschrieben wird, wird erst das MSB und anschließend das LSB eingelesen, somit muss der dritte Bereich der Interrupt Service Routine zweimal für eine Messung durchlaufen werden. Nach dem Empfang der Bytes muss jedes Mal die Richtung der Datenleitung nochmals geändert werden, damit der Mikrocontroller den Empfang bestätigen kann. Nachdem der Timer-Interrupt freigegeben wurde, führt die ISR erstmal eine Feuchtigkeitsmessung durch und im Anschluss wird dann gleich eine Temperaturmessung gemacht. Nach dem zweiten und dem vierten Durchlauf des dritten Bereichs werden die internen Flags so gesetzt, dass das Kommunikationsprotokoll von vorne beginnt. Zusätzlich wird nach dem vierten Durchlauf noch das Flag “SHT fertig gemessen“ gesetzt, da der Sensor fertig ausgelesen wurde.

ADU

Der ADU-Interrupt ist so initialisiert, dass er nach jeder achten Wandlung des internen ADUs auslösen soll. Der ADU ist im Scan-Modus konfiguriert, siehe „*ADU*“ auf Seite 36. Dies hat zur Folge, dass er eine Wandlung nach der anderen ausführt, solange das ASAM-Bit gesetzt ist. Zusätzlich ist der ADU so konfiguriert, dass nach der Interrupt-Auslösung das ASAM-Bit durch die Hardware gelöscht und somit die kontinuierliche Wandlung abgebrochen wird. Um ausschließen zu können, dass ungelesene Daten aus Versehen überschrieben werden, wurde der 16 Word große ADU-Puffer, wie im Abschnitt „*ADU*“ auf Seite 36 beschrieben, in zwei acht Word große Puffer aufgeteilt.

Mittels dieses Interrupts werden die Überwachungsdaten, der Stromverbrauch und die Versorgungsspannung der CTA/CCA-Sensoren sowie die Versorgungsspannung der Batterie aufgenommen. Ebenfalls wird in diesem Scan-Modus der analoge Wert des Drucksensors digitalisiert und aufgenommen. In der Interrupt Service Routine (ISR) muss am Anfang erst ermittelt werden, in welchem der zwei aufgeteilten ADU-Puffer die aktuellen Werte abgespeichert sind, in dem Puffer-Block mit den Adressen 0_{hex} bis 7_{hex} oder in dem Block mit den Adressen 8_{hex} bis F_{hex}. Dies geschieht durch Abfrage des *Buffer-Fill-Status-Bit (BUFS Bit)*. Anschließend werden die Daten aus dem jeweiligen Puffer gelesen und in dem Überwachungs-FIFO gespeichert. Ebenfalls werden die gerade ausgelesenen Daten auch in die Struktur für die Statusinformationen gespeichert. Zuletzt wird dann in der ISR noch das interne Flag "´Überwachungsdaten fertig“ gesetzt, das signalisiert, dass die Überwachungsdaten ausgelesen wurden. Zur späteren Erkennung, zu welchem Zeitpunkt die Überwachungsdaten aufgenommen wurden, gibt es eine Prüfung, ob die gespeicherte Zeitinformation im Überwachungs-FIFO die aktuellste ist oder das Aufnehmen der Überwachungsdaten zu einem späteren Zeitpunkt erfolgte. Sollte dies der Fall sein, wird dieser Zeitpunkt als die neue Zeitinformation in das Überwachungs-FIFO gespeichert. Die letzte Anweisung in der ISR ist das Löschen des Interrupt-Flags. Das gerade beschriebene ist in dem Struktogramm 3.27 auf Seite 62 dargestellt.

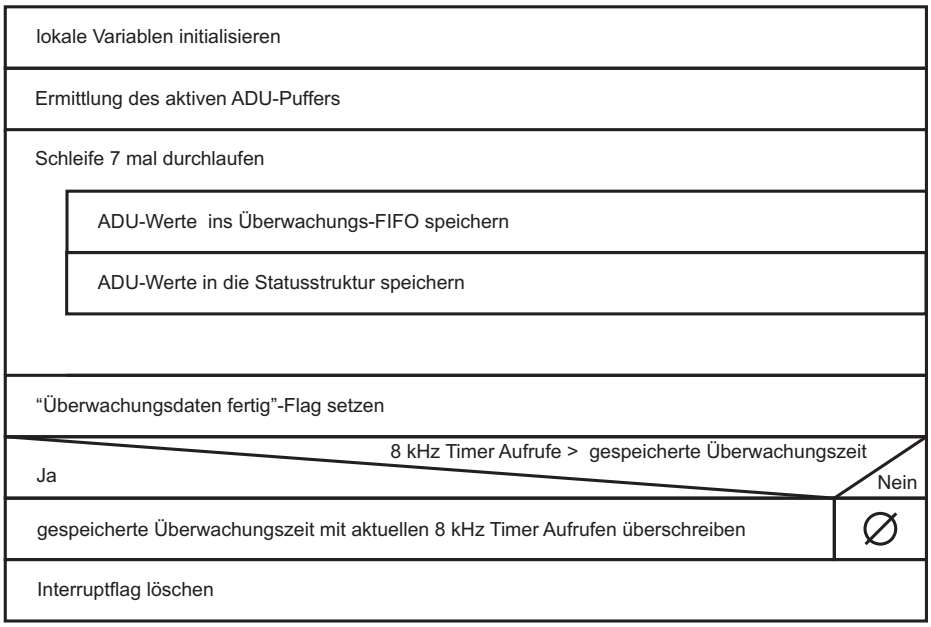


Bild 3.27.: Struktogramm der ISR des ADUs

DMA1 und DMA2

Die Beiden DMA-Kanäle 1 und 2 helfen, wie schon im Abschnitt „*DMA-Controller*“ auf Seite 33 erwähnt, bei der Umsetzung des Kommunikationsprotokolls des Magnetometers über die SPI-Schnittstelle mit. Die Interrupts von DMA 1 und DMA 2 signalisieren, dass der Datentransfer von der Start- zur Zieladresse jeweils erfolgreich war. Es werden zwei DMA-Kanäle benötigt, da jeweils nur eine Datenrichtung damit erfüllt werden kann. Allerdings müssen, um Daten über die SPI-Schnittstelle des PIC-Controllers, zu empfangen, vorher Daten über die SPI-Schnittstelle gesendet werden, siehe Abschnitt „*SPI*“ auf Seite 37. Der DMA-Kanal 1 wird benutzt, um Daten zum Magnetometer zu senden, und über den Kanal 2 werden dann die geantworteten Daten empfangen. Durch die Benutzung der DMA-Kanäle wird die CPU des Controllers entlastet und kann in der Zwischenzeit andere Aufgaben erledigen. Durch die Interrupt Service Routinen und die DMA-Kanäle wird eine parallele Abarbeitung erreicht.

In der Interrupt Routine des DMA-Kanals 1 wird zuerst das Flag abgefragt, welcher Event verantwortlich war, um diesen Interrupt zu aktivieren. Mit dem „Block done“-Flag wird signalisiert, dass ein erfolgreicher Datentransfer stattgefunden hat. Nur bei diesem Flag erfolgt eine Abarbeitung der Interrupt Routine. Wenn es das „Block Done“-Flag war, wird unterschieden ob das Kommando-Byte oder das erste Dummy-Byte gesendet wurde. Die Unterscheidung erfolgt durch das interne„Magneto TX Interrupt“-Flag. Wann welches der beiden Bytes übertragen wird, wurde im Abschnitt 3.3.2 „*Timer 2*“ auf Seite 54 in dem Bereich zum Auslesen des MicroMag3 beschrieben. Der DMA-Transfer zur Übermittlung des Kommando- und des ersten Dummy-Bytes wird im Timer 2 gestartet, allerdings wird der Transfer des zweiten Dummy-Bytes in der ISR des DMA-Kanals 1 gestartet. Abschließend werden das Eventflag und das Interruptflag gelöscht. Das Struktogramm zu der ISR des DMA-Kanals 1 ist im Bild 3.28 auf Seite 64 dargestellt.

Der Anfangsbereich der ISR des DMA-Kanals 2 ist genauso aufgebaut wie der der Interrupt Service Routine des ersten Kanals. Hier wird auch zuerst das Eventflag abgefragt und geprüft, ob es sich um das „Block Done“-Flag handelt. Danach wird durch das interne „Magneto RX Interrupt“-Flag unterschieden, welche Daten gerade empfangen wurden, ob es sich um die Antwort auf das Kommando- oder das erste Dummy-Byte handelt. Wenn es die Daten des Kommando-Bytes waren, wird das interne „Magneto RX Interrupt“-Flag gesetzt und die Abarbeitung der ISR ist nach dem Löschen des Event- und des Interruptflags beendet. Sollten es allerdings die gemessenen Werte des Sensors sein, wird zuerst noch geprüft, ob das MSB oder das LSB der jeweiligen Achsenmessung empfangen wurde. Die Unterscheidung, welches Byte gerade empfangen wurde, wird mittels einer Zählvariable „RX Count“ erreicht. Wenn der 16 Bit breite Messwert komplett empfangen wurde, wird dieser in das aktuelle Element des Lage-FIFOs gespeichert und zusätzlich noch in die Statusstruktur. Anschließend werden die steuernden Flags „Kommando-Byte senden“ gelöscht. Damit wird dem Timer 2-Interrupt signalisiert, dass die nächste Achse gemessen werden kann, und das andere Flag

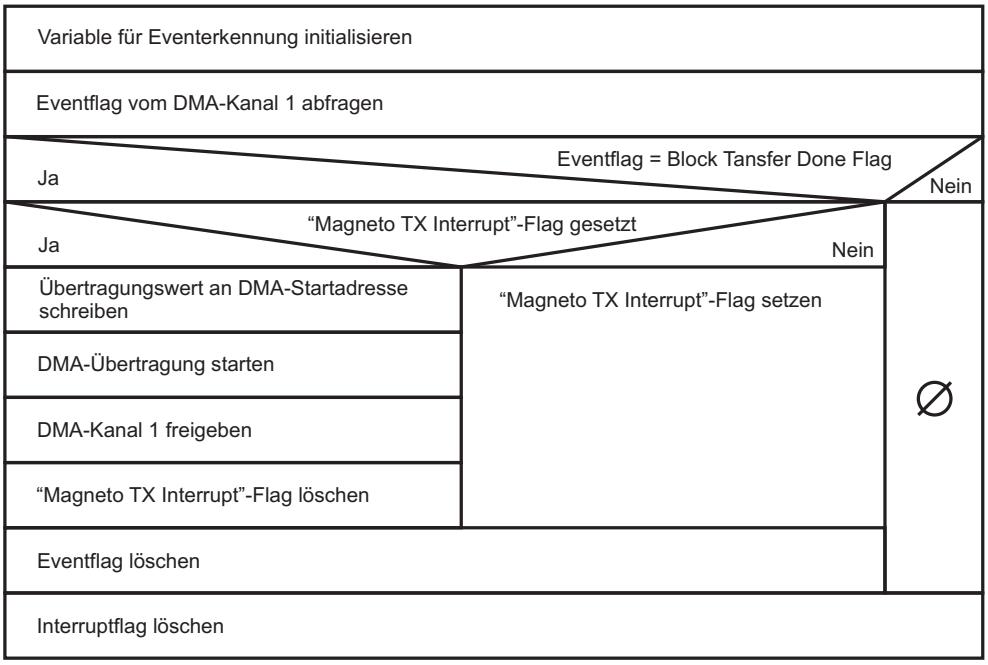


Bild 3.28.: Struktogramm der ISR des DMA-Kanals 1

„Magneto RX Interrupt“ wird benötigt, um dieser ISR zu zeigen, dass der nächste empfangene Wert der vom Kommando-Byte ist. Mittels einer weiteren Zählvariable „Achsencount“ wird überprüft, wie viele der drei zu messenden Achsen schon ausgelesen wurden. Wenn alle drei Achsen ausgelesen wurden, wird die SPI-Schnittstelle des Magnetometers durch Setzen der SSNOT-Leitung auf High deaktiviert, das Flag „Magnetometer messen“ wird gelöscht und das Flag „Magnetometer fertig“ gesetzt. Insgesamt wird diese ISR dreimal aufgerufen, um einen vollständigen Messwert zu erhalten. Bevor die Interrupt Service Routine verlassen wird, muss der DMA-Kanal wieder aktiviert werden, sonst würden die Daten nicht empfangen werden. In dem Bild 3.29 auf Seite 65 ist das Struktogramm zu dem Beschriebenen dargestellt.

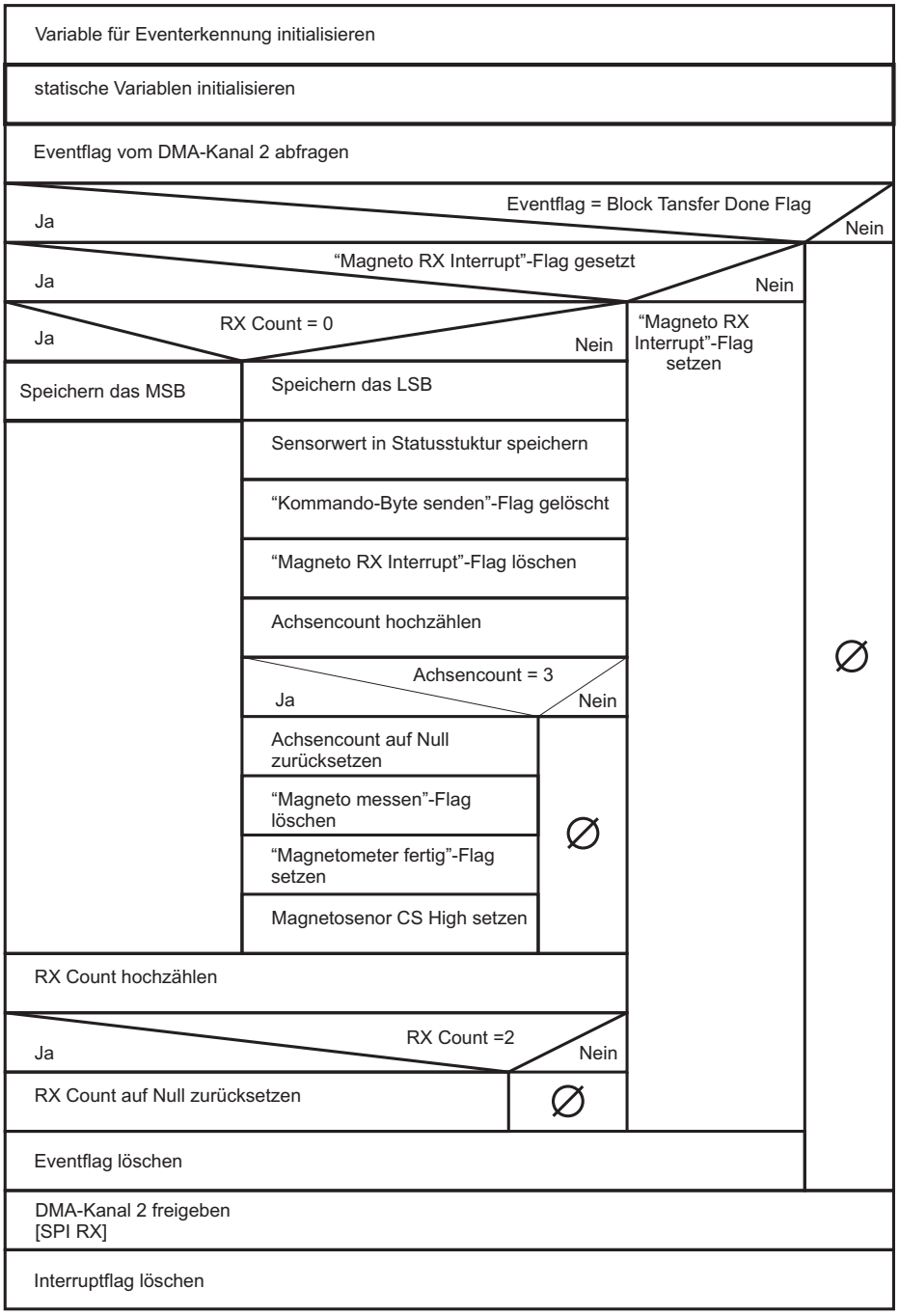


Bild 3.29.: Struktogramm der ISR des DMA-Kanals 2

DMA3

Die Interrupt Service Routine des dritten DMA-Kanals wird benutzt, um auf die asynchron eintreffenden Kommandos zu reagieren. Dafür wurde der DMA-Transfer, wie schon im Abschnitt „*DMA-Controller*“ auf Seite 33 kurz erwähnt, zwischen dem Empfangsregister der UART-Schnittstelle (UART 1B) und einer Speicheradresse eines Arrays in dem Pattern-Modus konfiguriert. Dieser Datentransfer wird durch den Empfang eines Bytes und dem damit verbundenen Interrupt (*receive buffer is not empty*) gestartet. Die Kommandos müssen im ASCII-Format übertragen werden und haben folgende Bestandteile: ein Startzeichen ($), das Kommando, ein Trennzeichen(@) und ein Endzeichen (*). Der Interrupt des DMA-Kanals 3 kann durch zwei Möglichkeiten ausgelöst werden: Die Erste Möglichkeit besteht, wenn das vordefinierte Zeichen, das im *DMA Channel x Pattern Data Register (DCHxDAT)* steht, empfangen wurde. In diesem Projekt ist es das Endzeichen des Kommandos, das dort definiert ist. Die zweite ist, wenn das Arrays komplett mit Daten gefüllt wurde. Im Nachfolgenden sind die Kommandos, die empfangen werden können, aufgeführt:

- CTA-Sensor n ein/aus ($n \in \{1, 2, 3\}$)
  `$CTAn@m*` $m \in \{0, 1\}$
- Datenspeicherung (SD card) ein/aus
  `$SDWRT@n*` $n \in \{0, 1\}$
- Telemetrie ein/aus
  `$TELE@n*` $n \in \{0, 1\}$
- Synchronisieren der Zeit (setze RTCC)
  `$SETTIME@hhmmss*`
- Kalibrierung des ADIS-Sensors starten
  `$ADISCAL*`

Diese Interrupt Service Routine ist am Anfang genauso aufgebaut, wie die zwei anderen ISR (DMA1 und DMA2). Der neue Teil beginnt erst nach der erfolgreichen Prüfung, ob das „Block-Done“-Flag gesetzt wurde. Dieses Flag wird auch gesetzt, wenn der DMA-Transfer durch den Empfang des Endzeichens abgebrochen wurde. Durch einzelne Stringvergleiche wird ermittelt, um welches Kommando es sich bei dem gerade Empfangenen handelt.

Der folgende Programmcode ist ein Auszug der ISR des dritten DMA-Kanals und soll die Umsetzung des Stringvergleichs verdeutlichen. Dabei wird das empfangene Kommando mit dem String „$CTA“ durch die Funktion strncmp ( ) verglichen. Bei Gleichheit gibt die Funktion eine Null zurück. Der dmaBuff ist das Array, das die empfangenen Daten, das Kommando, speichert. Die Funktion strncmp ( ) vergleicht so viele Zeichen wie als Parameter angegeben wurde, hier vier. Bei Gleichheit gibt die Funktion eine Null aus, deshalb lautet die Abfrage auf ungleich. Die anderen Vergleiche sind in der gleichen Art umgesetzt.

```
1   if(!strncmp(dmaBuff, "\$CTA", 4))
    {
3       commando=COMMAND_Switch_CTA_Power;
    }
```

Listing 3.1: Kommando-Erkennung mit Stringvergleich

Wenn das Kommando „CTA-Sensor ein/aus“ empfangen wurde, wird geprüft, bei welchem der drei CTA-Sensoren sich der Zustand ändern soll. Daraufhin wird der entsprechende Pin auf High-Pegel (ein) oder Low-Pegel (aus) gesetzt und das jeweilige Photomos-Relais damit beschaltet.

Mit Hilfe des Befehls „Datenspeicherung ein/aus“ wird ein internes Flag „SD-Karte aktiv“ gesetzt. Durch dieses Flag kann die Speicherung auf die SD-Karte erlaubt oder untersagt werden. Nach dem Programmstart ist das Flag „SD-Karte aktiv“ so gesetzt, dass Daten auf die SD-Karte gespeichert werden.

Das Kommando „Telemetrie ein/aus“ ist eine Mischung aus den Aktionen, die in den zwei vorangegangen Befehlen beschrieben wurden, die Beschaltung des Pins des Photomos-Relais und das Setzen des steuernden Flags „Telemetrie an“. Mit diesem Flag wird verhindert oder erlaubt, dass die Telemetriedaten aufbereitet und gesendet werden.

Durch das Kommando „Setze RTCC“ wird die Uhrzeit gestellt. Zu Beginn ist die Uhrzeit auf 00:00:00 Uhr gesetzt. Die übertragene Uhrzeit wird nach Stunden, Minuten und Sekunden getrennt und damit wird das *RTC Time Value Register (RTCTIME)* konfiguriert. Zusätzlich wird noch ein Flag „RTC gestellt“ gesetzt, das signalisiert, dass die Uhrzeit gesetzt wurde und nun als Statusinformation übermittelt werden kann.

Nach dem Empfang des Kommandos zur „Kalibrierung des ADIS-Sensors“ wird ein Schreibvorgang an den ADIS-Sensor gesendet. Außerdem wird noch der Startzeitpunkt aufgenommen und damit der Endzeitpunkt für die Kalibrierungszeit (ca. 30 s) berechnet. Ebenfalls wird noch ein internes Flag „ADIS Kalibrieren“ gesetzt, das verhindert, dass in der Kalibrierungszeit eine Lagemessung ausgeführt wird.

Als abschließende Aktion muss der DMA-Kanal wieder aktiviert werden, um das nächste Kommando zu empfangen. Die Beschreibung der Interrupt Service Routine des DMA-Kanals 3 ist in dem Struktogramm 3.30 auf Seite 68 dargestellt.

DMA4

Der vierte DMA-Kanal wird benutzt, um die Statusinformationen zu übermitteln. Dabei werden die zu übertragenden Statusdaten von der Speicheradresse des Statusarrays zum Senderegister der UART-Schnittstelle (UART 1B) transportiert. Wie die einzelnen Statusdaten in

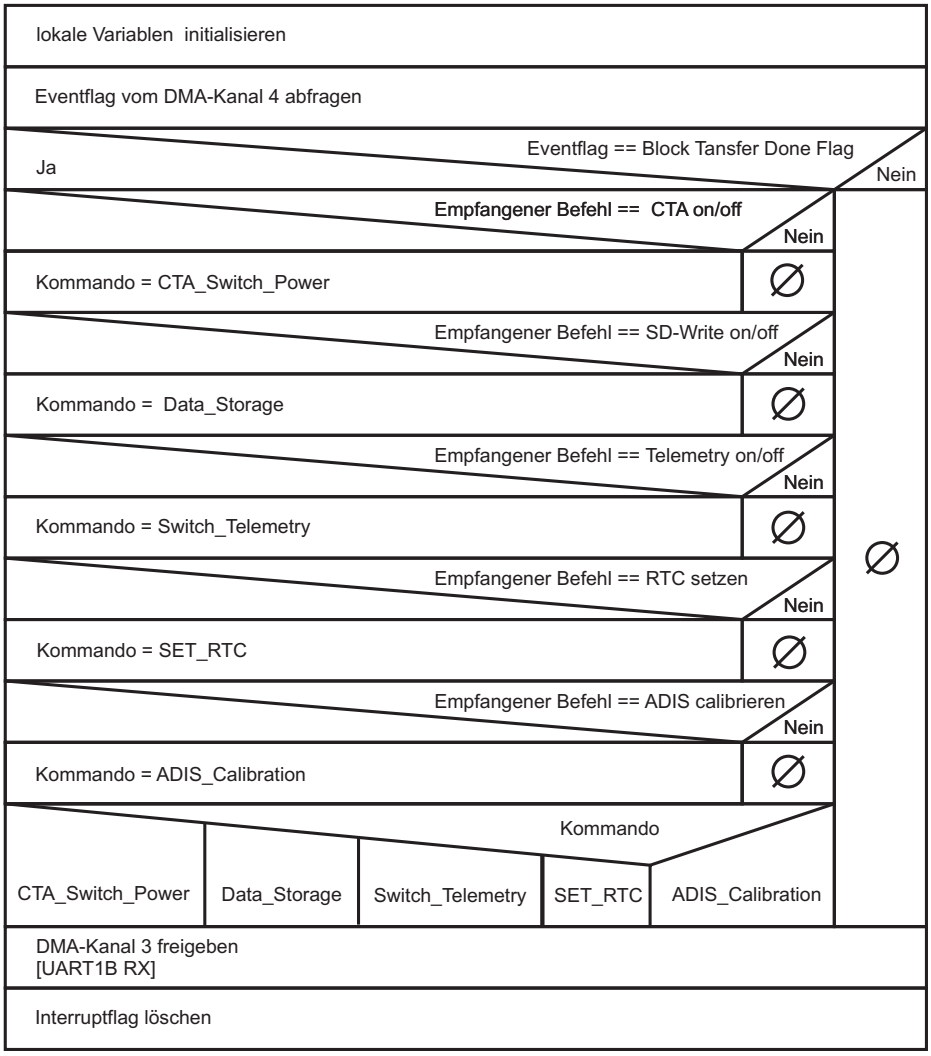


Bild 3.30.: Struktogramm der ISR des DMA-Kanals 3

dem Array zusammengefügt werden, wird in dem Abschnitt 3.3.3 „Status senden“ auf Seite 69 beschrieben. Dieser Kanal ist in dem Pattern-Modus konfiguriert und sein Transfer wird durch den TX-Interrupt der UART-Schnittstelle (UART 1B) gestartet. Der TX-Interrupt soll immer dann auslösen, wenn Platz für neue Daten im Senderegister ist. Zum Aktivieren der DMA 4 Interrupt Routine, gibt es zwei Möglichkeiten, wenn die komplette Anzahl an Daten (Arraygröße) übrtragen wurde und wenn das Zeichen übertragen wird, dass im *DMA Channel x Pattern Data Register (DCHxDAT)* definiert wurde (siehe Pattern-Modus im Abschnitt *DMA-Controller* auf Seite 33). Bei beiden Möglichkeiten wird die ISR des vierten DMA-Kanals durch den „Block Done“-Interrupt ausgelöst. Diese ISR hat die Aufgabe anzuzeigen, dass die Statusinformationen ordentlich übertragen wurden. Der Grundaufbau dieser Interrupt Routine ist dieselbe wie bei den anderen ISR der DMA-Kanäle. Es wird wieder das Eventflag abgefragt und mit dem „Block Done“-Flag vergleichen. Sollte der Vergleich erfolgreich sein, wird das Eventflag gelöscht, ansonsten wird nur das Interruptflag gelöscht. Damit ist der Controller mit der Bearbeitung dieser ISR fertig. Das Struktogramm zu dieser Interrupt Routine ist im Bild 3.31 dargestellt.

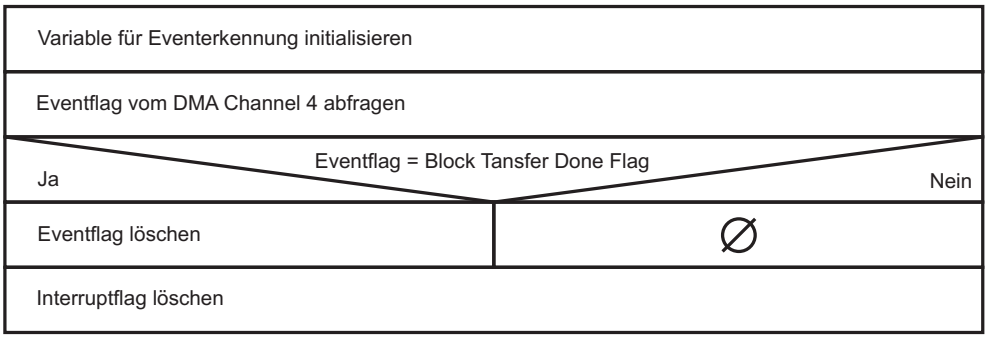


Bild 3.31.: Struktogramm der ISR des DMA-Kanals 4

3.3.3. Status senden

Die Statusinformationen werden wie schon erwähnt über die UART-Schnittstelle (UART 1B) übertragen. Die einzelnen Statusinformationen sollen jede Sekunde übermittelt werden. Dazu werden die jeweils gemessenen Werte in die schon angedeutete Statusinformationsstruktur gespeichert. Die Struktur ist in der Tabelle 3.16 auf Seite 70 dargestellt:

Nach einer Sekunde wird das Flag “Status senden“ in der ISR des Timers 2 gesetzt, woraufhin die Funktion „*status_senden( )*“ in der Endlosschleife des Hauptprogramms aufgerufen wird. Sollte das Flag „Status senden“ nicht gesetzt werden, wird diese Funktion nicht gestartet und abgearbeitet. Diese Funktion hat die Aufgabe, das Statusarray zusammenzustellen,

Name	Größe	reservierter Speicher
CTA/CCA-Sensor 1	16 Bit	2 Byte
CTA/CCA-Sensor 2	16 Bit	2 Byte
CTA/CCA-Sensor 3	16 Bit	2 Byte
Rotation X-Achse	14 Bit	2 Byte
Rotation Y-Achse	14 Bit	2 Byte
Rotation Z-Achse	14 Bit	2 Byte
Beschleunigung X-Achse	14 Bit	2 Byte
Beschleunigung Y-Achse	14 Bit	2 Byte
Beschleunigung Z-Achse	14 Bit	2 Byte
Magnetfeld X-Achse	16 Bit	2 Byte
Magnetfeld Y-Achse	16 Bit	2 Byte
Magnetfeld Z-Achse	16 Bit	2 Byte
Temperaturwert SHT	14 Bit	2 Byte
Feuchtigkeitswert SHT	12 Bit	2 Byte
Überwachung Batteriespannung	10 Bit	2 Byte
Drucksensorwert	10 Bit	2 Byte
Spannungsüberwachung CTA/CCA-Sensor 1	10 Bit	2 Byte
Spannungsüberwachung CTA/CCA-Sensor 2	10 Bit	2 Byte
Spannungsüberwachung CTA/CCA-Sensor 3	10 Bit	2 Byte
Stromüberwachung CTA/CCA-Sensor 1	10 Bit	2 Byte
Stromüberwachung CTA/CCA-Sensor 2	10 Bit	2 Byte
Stromüberwachung CTA/CCA-Sensor 3	10 Bit	2 Byte
Temperatur X-Achse vom ADIS	12 Bit	2 Byte
Temperatur Y-Achse vom ADIS	12 Bit	2 Byte
Temperatur Z-Achse vom ADIS	12 Bit	2 Byte
Versorgungsspannung vom ADIS	12 Bit	2 Byte
Status des CTA/CCA-Sensors 1	8 Bit	1 Byte
Status des CTA/CCA-Sensors 2	8 Bit	1 Byte
Status des CTA/CCA-Sensors 3	8 Bit	1 Byte

Tabelle 3.16.: Datenstruktur für die Statusinformationen

von dem der DMA-Transfer (Kanal 4) erfolgt, und diesen zu starten. Die Statusinformationen in dem Array haben das ASCII-Format. Jede Statusinformation hat ein Startzeichen ($), einen Namen, ein Trennzeichen (@), einen Datenstring und ein Endzeichen (*). Es werden die unbearbeiteten (Raw) Daten der Sensoren übertragen. Nachfolgend sind die einzelnen Statusinformationen aufgeführt. Dabei stehen die Buchstaben c, n, m, v, x, y und z für hexadezimale Ziffern:

- SD-Karte aktiv Status
  $SDWRT@n* $n \in \{0, 1\}$
- Betriebsspannung der Messdatenerfassungskarte
  $VCC@vvvv*
- Relais Status (CTA 1–3, Telemetrie)
  $REL@nnnn* $n \in \{0, 1\}$
- Uhrzeit
  $TIME@hhmmss*
- Momentaner Zustand der Messdatenerfassungskarte
  (Frei (IDLE), Datenerfassung(ACQ))
  $STAT@str* str = IDLE, ACQ
- Magnetometer-Daten x, y, z
  $MAG@xxxx,yyyy,zzzz*
- SHT15 Temperatur, SHT15 Feuchtigkeit, Druck
  $THP@xxxx xxxx xxxx*
- ADIS-Daten: Rotation x, y, z, Beschleunigung x, y, z, Temperatur x, y, z
  $ADIS@xxxx,yyyy,zzzz xxxx,yyyy,zzzz xxxx,yyyy,zzzz*
- CTA-Status (1–3): Versorgungsspannung (v), Stromverbrauch (c), Signalwert (x), Zustandsstatus (n) mit 0 = aus, 1 = OK, 2 = kaputt
  $CTA@vvvv,cccc,xxxx,n vvvv,cccc,xxxx,n vvvv,cccc,xxxx,n*

Die Funktion „status_senden( )“ fügt die einzelnen Statusinformationen in der oben aufgeführten Reihenfolge in dem Statusarray zusammen. Durch die Benutzung des ASCII-Formats können für das Zusammenstellen String-Funktionen benutzt werden. Der nachfolgende Programmcode zeigt, wie das Zusammenfügen umgesetzt wurde:

```
1   //SD-Card on
    sprintf(zwischenstatusbuffer,"$SDWRT@%c*",(SD_WRITE+0x30));
3   strcat(statusbuffer,zwischenstatusbuffer);

5   //versporgungsspannung
    sprintf(zwischenstatusbuffer,"$VCC@%x*",StatusInfos.Monitoring[0]);
7   strcat(statusbuffer,zwischenstatusbuffer);
```

Listing 3.2: Auszug aus der Funktion status_senden( )

Dabei wird zuerst immer die einzelne Statusinformation in dem „zwischenstatusbuffer“ mit „sprintf( )“ zusammengestellt und später mit dem Statusarray (statusbuffer) verkettet durch „strcat( )“. In dem Programmcode-Auszug werden die Statusinformationen „SD-Karte aktiv“ und die Information der Batterieversorgungsspannung zusammengestellt. Nach dem gleichen Schema sind auch die anderen Statusinformationen zusammengefügt. Das Array mit dem Namen Monitoring in der StatusInfos-Struktur speichert die Überwachungsdaten, die mit dem internen ADU ermittelt werden. Am Ende der Funktion „status_senden“ wird der DMA-Transfer gestartet.

4. Test

Zur Überprüfung der Funktionalität der Messkarte wurden einige Tests durchgeführt, die nachfolgend aufgeführt sind.

Senden und Empfangen über UART 1B

Zum Testen, ob das Senden der Statusinformation und das Empfangen der Kommandos über die UART-Schnittstelle (UART 1B) einwandfrei funktionieren, wurde die Messkarte mit einem Computer über ein Nullmodem-Kabel verbunden. Zum Anzeigen der Statusinformationen wurde ein Programm (Groundsupport) benutzt, das von einem IAP-Mitarbeiter für die Ballonstarts entwickelt wurde. Dieses Programm soll benutzt werden, um die Messkarte vor den Ballonstarts zu konfigurieren und zu überprüfen, ob alles einwandfrei funktioniert. Mit diesem Programm konnte somit auch getestet werden, ob der Controller auf Kommandos reagiert. Ein Screenshot des Programms ist im Bild 4.1 auf Seite 74 dargestellt. Dass in Bild 4.1 beim Magnetometer jeweils Null angezeigt wird, liegt daran, dass zu dem Zeitpunkt keines angeschlossen war. Auf der rechten Seite des Screenshots sind die übertragen Raw-Daten dargestellt, wie sie übermittelt wurden. Das Empfangen der Kommandos über den DMA-Kanal 3 und das Senden der Statusinformation mit dem DMA-Kanal 4 funktioniert einwandfrei. Das Hauptprogramm des Controllers arbeitet weiter, ohne dass die Messung der CTA-Sensoren beeinträchtigt wird.

Lagesensoren

Um zu überprüfen, ob die Lagesensoren plausible Werte liefern, wurden ihre Werte mit dem Groundsupport-Programm dargestellt. Durch Bewegen der Messkarte oder durch Drehen des Magnetometers wurde kontrolliert, ob sich die Messwerte der drei Sensoren, die für die Lagebestimmung verantwortlich sind, ändern. In Ruheposition müsste die Messkarte theoretisch für alle drei Achsen der Rotation Null anzeigen; de facto haben die Rotationssensoren des ADIS-Sensors aber eine gewisse Ungenauigkeit und ein Rauschen, wodurch von Null abweichende Werte angezeigt werden. Bei dem Beschleunigungssensor wird in Ruhe für die X- und die Y-Achse eine Beschleunigung von Null gemessen und für die Z-Achse eine minus Eins. Dies kommt daher, dass der ADIS-Sensor seine Beschleunigungswerte in Einheiten der Erdbeschleunigung angibt und auch die Beschleunigungssensoren haben eine gewisse Ungenauigkeit. Das Beschriebene ist im Bild 4.1 auf Seite 74 zu sehen. Durch das Drehen

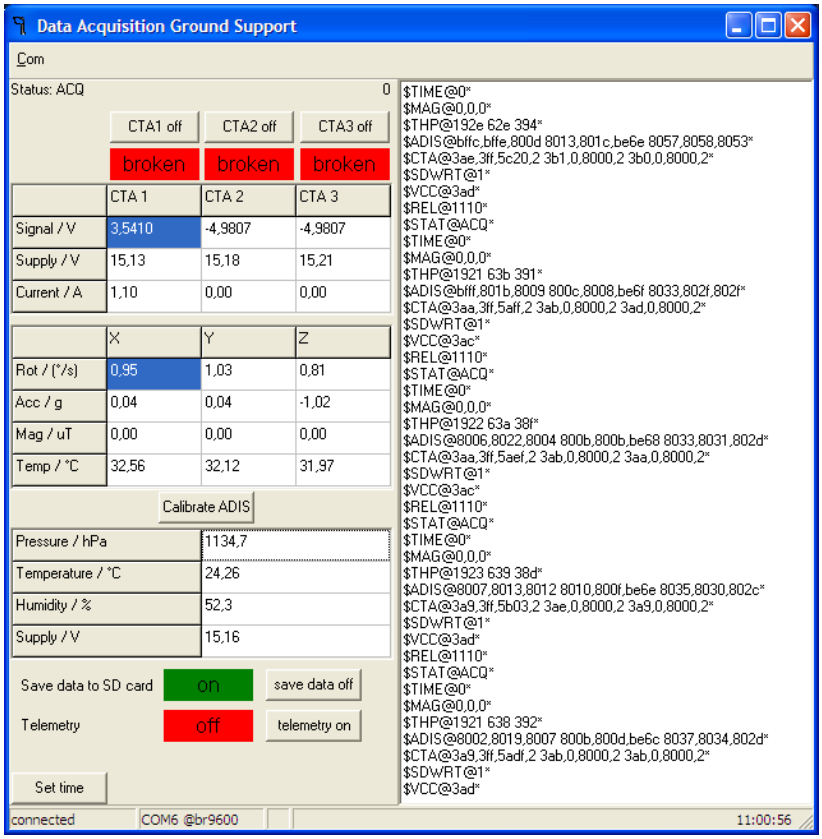


Bild 4.1.: Screenshot des Groundsupport-Programms

um 90 Grad in eine Richtung zeigt diese Achse wie gewünscht die Erdbeschleunigung an. Zusätzlich wurde noch eine Kontrollmessung mit dem alten Housekeeping-System durchgeführt, um zu vergleichen, ob bei der neuen Messkarte ähnliche Werte herauskommen. Ebenfalls konnte durch Drehen des Magnetometers gesehen werden, dass sich die X- und Y-Werte verändern, wie es bei einem Kompass zu erwarten ist.

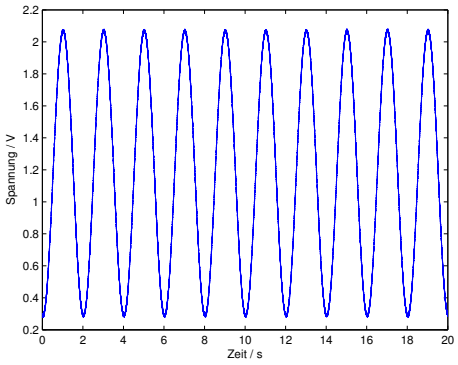
Temperatur- und Feuchtigkeitssensor

Der Temperatur- und Feuchtigkeitssensor zeigte die Zimmertemperatur von ca. 24 °C an, was vermuten ließ, dass der Sensor die Temperatur richtig misst. Zur Sicherheit wurde überprüft, ob sich die Temperatur- und Feuchtigkeitswerte ändern, wenn der Sensor erwärmt oder abgekühlt wurde, was der Fall war.

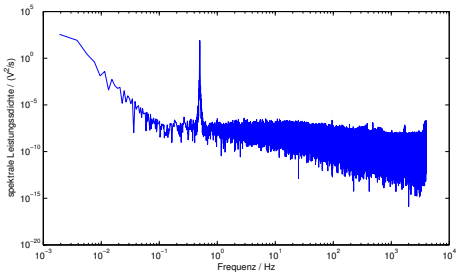
Abtastung und Speicherung

Bei einem Test wurde geprüft, ob das Spannungssignal mit 8 kHz abgetastet wird und ob die Speicherung ordentlich funktioniert. Dafür wurde das CTA-Spannungssignal eines Sen-

sors mit einem Funktionsgenerator simuliert, der ein Sinussignal mit 0,5 Hz erzeugte. Durch Auslesen der Daten der SD-Karte wurde festgestellt, dass die Speicherung und die Abtastung funktionieren. Das reproduzierte Signal und dessen spektrale Leitungsdichte, die aus den gespeicherten Daten erzeugte wurde, kann dem Bild 4.2 auf Seite 75 entnommen werden. Die spektrale Leistungsdichte ist über den gesamten aufgenommen Zeitbereich (300 s) dargestellt, das Spannungssignal nur für einen Teilausschnitt (20 s), damit die einzelnen Schwingungen besser erkennbar sind. Neben dem Test mit den 0,5 Hz wurden noch andere Frequenzen, wie 1 kHz, 2 kHz und 3 kHz, mit dem Funktionsgenerator aufgenommen, und durch eine Spektralanalyse der gemessenen Signale konnte festgestellt werden, dass die eingestellten Frequenzen richtig gemessen wurden.



(a) 0,5 Hz Sinussignal



(b) Spektrale Leistungsdichte des 0,5 Hz Sinussignals

Bild 4.2.: Tests mit dem Funktionsgenerator

CTA-Sensor

Die neue Messkarte wurde mit einem Sensor getestet, der über die zugehörige Wheatstone-Brücke an die Platine angeschlossen war. Das aufgezeichnete Signal ist in Bild 4.3 auf Seite 76 dargestellt. Es liegt im erwarteten Spannungsbereich. Während der Messung wurde mehrfach gegen den Draht gepustet, was im Bild als Spannungsspitzen zu sehen ist, welche die höhere Windgeschwindigkeit repräsentieren.

Genauigkeit des Externen ADUs

Durch Wandeln des Spannungssignals einer Saft-Batterie wurde die Genauigkeit des externen ADUs getestet. Die Batterie erzeugte ein Signal von 3,6 V. Nach Auswertung des gespeicherten Signals musste festgestellt werden, dass das digitale Signal ein Rauschen von ungefähr 40 mV aufwies. Das Bild 4.4 auf Seite 76 zeigt das gemessene Batteriesignal

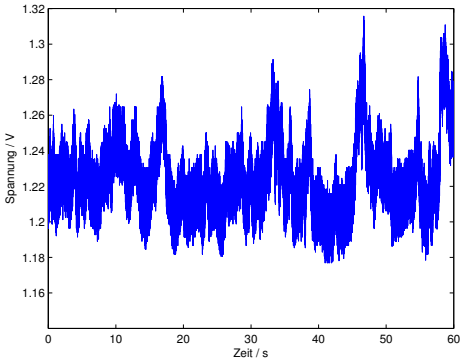
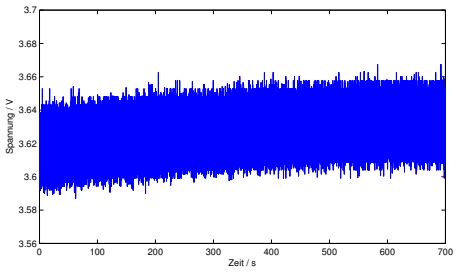
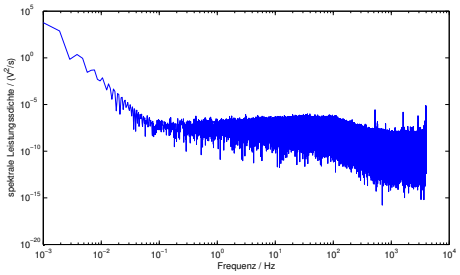


Bild 4.3.: Testmessung mit einem CTA-Sensor

mit dem Rauschen und die spektrale Leistungsdichte dieses Signals. An dem Spektrum kann gesehen werden, dass keine bestimmte Frequenz für das Rauschen verantwortlich ist, was das Finden der Ursache leider erschwert. Eine mögliche Ursache könnte sein, dass einer der verwendeten DCDC-Wandler mit seinen Schaltfrequenzen das Rauschen erzeugt. Um dies ggf. zu verbessern, wurde dem Entkopplungskondensator des DCDC-Wandlers, der die negagtiven Spannungen erzeugt, ein RL-Tiefpass zugeschaltet, was aber keine Minderung brachte. Das Rauschen tritt auch auf, wenn der Eingang auf Masse gelegt wird, und kann noch direkt am Eingangspin des externen ADUs gemessen werden. Mit und ohne Vorverstärker ist kein Unterschied darin feststellbar. Bis zur Abgabe der Arbeit konnte noch nicht bestimmt werden, wie das Rauschen entsteht.



(a) 3,6 V Digitalisiertes Batteriesignal



(b) Spektrale Leistungsdichte des 3,6 V Batteriesignals

Bild 4.4.: Tests mit einer Saft-Batterie

5. Schluss

Die Aufgabe dieser Arbeit war, eine neue Messdatenerfassungskarte für Höhenballons zu entwickeln. Diese soll die CTA-Sensoren, die die Windgeschwindigkeit messen, mit 8 kHz abtasten und auch Informationen über die Lage der Gondel mit 100 Hz aufnehmen. Ausserdem sollen Umgebungsbedingungen mit 1 Hz erfasst werden. Sämtliche gemessenen Werte sollen auf einer SD-Karte gespeichert und an ein Telemetriesystem übergeben werden. Für die Konfigurierung soll es eine Möglichkeit geben, mit einem PC zu kommunizieren.

Zur Erfüllung der Aufgabe wurde eine komplett neue Hardware entwickelt. Es wurde sich für den PIC-Controller entschieden, da er die nötigen Schnittstellen besitzt, um die Sensoren zur Bestimmung der Windgeschwindigkeit und der Lage anzuschließen. Zusätzlich konnte der Controller schnell genug getaktet werden, um die gestellten Aufgaben zu lösen. Durch seinen DMA-Controller konnte erreicht werden, dass gewisse Prozesse, z. B. das asynchrone Empfangen der Kommandos, erfolgen, ohne den Prozessor zu belasten. Die Frequenzen zum Abtasten der verschiedenen Sensoren hätten auch jeweils über einen eigenen Timer erfolgen können, aber um Synchronisationsfehler zu vermeiden, wurde die Umsetzung mit nur einem Timer gewählt.

Wie im Abschnitt 4 „*Test*“ auf Seite 73 beschrieben, gibt es zur Zeit noch ein Rauschen in dem CTA/CCA-Signal. Im Hinblick auf die weitere Vorgehensweise ist erstmal das Rauschen zu lokalisieren und zu beseitigen, da mit diesem Rauschen in dem Windsignal nur sehr starke Turbulenzen zu erkennen sind. Ein weiter Punkt ist die Implementierung für die drahtlose Datenübertragung zum Boden. In der Hardware wurde dafür die serielle Schnittstelle gemäß RS-232 und RS-422 bereitgestellt, über die die Daten später an ein Funkmodem übertragen werden. Abhängig vom eingesetzten Modem und der Zahl der bei der jeweiligen Messung verwendeten Sensoren müssen die Daten noch per Software komprimiert und aufbereitet werden. Dieser Bereich der Software ist nicht Teil der vorliegenden Arbeit.

Es könnte noch versucht werden, die Effizienz des Schreibens auf die SD-Karte zu erhöhen, indem z. B. ein eigenes Treiberprogramm geschrieben wird, das auch den DMA-Controller nutzt, um den Prozessorkern zu entlasten.

Sollte das IAP in Erwägung ziehen, weitere Sensoren an die Messdatenerfassungskarte anzuschließen, sollte darüber nachgedacht werden, die Software auf ein Real-Time Operating System (RTOS) umzustellen.

Literaturverzeichnis

[1] http://de.wikipedia.org/wiki/Thermische_Anemometrie.

[2] http://www.dantecdynamics.com/Default.aspx?ID=1057.

[3] http://de.wikipedia.org/wiki/Drehratensensor.

[4] http://de.wikipedia.org/wiki/Beschleunigungssensor.

[5] http://de.wikipedia.org/wiki/Inertialsensor.

[6] http://de.wikipedia.org/wiki/Magnetometer.

[7] http://de.wikipedia.org/wiki/Drucksensor.

[8] http://de.wikipedia.org/wiki/RS232.

[9] http://www.mikrocontroller.net/articles/Galvanische_Trennung.

[10] http://knol.google.com/k/sd-card-interfacing.

[11] Inc. Analog Devices. Ad8021 low noise, high speed amplifier for 16-bit systems data sheet (rev. f). www.analog.com, 2006.

[12] Inc. Analog Devices. Adis16350/adis16355 (rev. b) high precision tri-axis inertial sensor data sheet. www.analog.com, 2009.

[13] Inc. Analog Devices. Ad7656/ad7657/ad7658 (rev. c) 250 ksps, 6-channel, simultaneous sampling, bipolar 16-/14-/12- bit adc. www.analog.com, 2010.

[14] SENSIRION The Sensor Company. Datasheet_sht1x_v4.0_c1, 2008.

[15] PNI Corporation. Pni-11096 3 - axis magneto-inductive sensor driver and controller with spi serial interface, 2003.

[16] PNI Corporation. Micromag3 3-axis magnetic sensor module, 2005.

[17] F. Foust. Application note secure digital card interface for the msp430, 2004.

[18] Kingmax Digital Inc. Sd card specification, 2004.

[19] Microchip Technology Inc. Pic32 peripheral libraries for mplab c32 compiler. www.microchip.com, 2007.

[20] Microchip Technology Inc. Pic32mx5xx/6xx/7xx family data sheet. www.microchip.com, 2009.

[21] MAXIM. Max220max249 +5v-powered, multichannel rs-232 drivers/receivers. www.maxim-ic.com., 2004.

[22] MAXIM. 76v, high-side, current-sense amplifiers with voltage output. www.maxim-ic.com., 2010.

[23] Traco Power. Dc/dc-konverter tdr 3wi serie 3 watt. www.tracopower.com, 2011.

[24] Recom. Dc – dc wandler applikationen. www.recom-international.com, 2008.

[25] Recom. Innoline dc/dc-converter r-78b-1.0. www.recom-international.com, 2010.

[26] Saft. Primary lithium battery lsh 20, 2006.

[27] sensortechnics. Asdx series signal conditioned pressure transducers. http://www.sensortechnics.com/download/asdx-530.pdf, 2009.

[28] Buchgeher Stefan. Ansteuerung eines feuchte- temperatur- sensors (shtxx) (mit pic-mikrocontroller), 2004.

[29] STMicroelectronics. Very low drop voltage regulators with inhibit. www.st.com., 2008.

A. Anhang

A.1. Interrupt Quellen

TABLE 7-1: INTERRUPT IRQ, VECTOR, AND BIT LOCATION

Interrupt Source[1]	IRQ	Vector Number	Interrupt Bit Location			
			Flag	Enable	Priority	Sub-Priority
Highest Natural Order Priority						
CT – Core Timer Interrupt	0	0	IFS0<0>	IEC0<0>	IPC0<4:2>	IPC0<1:0>
CS0 – Core Software Interrupt 0	1	1	IFS0<1>	IEC0<1>	IPC0<12:10>	IPC0<9:8>
CS1 – Core Software Interrupt 1	2	2	IFS0<2>	IEC0<2>	IPC0<20:18>	IPC0<17:16>
INT0 – External Interrupt 0	3	3	IFS0<3>	IEC0<3>	IPC0<28:26>	IPC0<25:24>
T1 – Timer1	4	4	IFS0<4>	IEC0<4>	IPC1<4:2>	IPC1<1:0>
IC1 – Input Capture 1	5	5	IFS0<5>	IEC0<5>	IPC1<12:10>	IPC1<9:8>
OC1 – Output Compare 1	6	6	IFS0<6>	IEC0<6>	IPC1<20:18>	IPC1<17:16>
INT1 – External Interrupt 1	7	7	IFS0<7>	IEC0<7>	IPC1<28:26>	IPC1<25:24>
T2 – Timer2	8	8	IFS0<8>	IEC0<8>	IPC2<4:2>	IPC2<1:0>
IC2 – Input Capture 2	9	9	IFS0<9>	IEC0<9>	IPC2<12:10>	IPC2<9:8>
OC2 – Output Compare 2	10	10	IFS0<10>	IEC0<10>	IPC2<20:18>	IPC2<17:16>
INT2 – External Interrupt 2	11	11	IFS0<11>	IEC0<11>	IPC2<28:26>	IPC2<25:24>
T3 – Timer3	12	12	IFS0<12>	IEC0<12>	IPC3<4:2>	IPC3<1:0>
IC3 – Input Capture 3	13	13	IFS0<13>	IEC0<13>	IPC3<12:10>	IPC3<9:8>
OC3 – Output Compare 3	14	14	IFS0<14>	IEC0<14>	IPC3<20:18>	IPC3<17:16>
INT3 – External Interrupt 3	15	15	IFS0<15>	IEC0<15>	IPC3<28:26>	IPC3<25:24>
T4 – Timer4	16	16	IFS0<16>	IEC0<16>	IPC4<4:2>	IPC4<1:0>
IC4 – Input Capture 4	17	17	IFS0<17>	IEC0<17>	IPC4<12:10>	IPC4<9:8>
OC4 – Output Compare 4	18	18	IFS0<18>	IEC0<18>	IPC4<20:18>	IPC4<17:16>
INT4 – External Interrupt 4	19	19	IFS0<19>	IEC0<19>	IPC4<28:26>	IPC4<25:24>
T5 – Timer5	20	20	IFS0<20>	IEC0<20>	IPC5<4:2>	IPC5<1:0>
IC5 – Input Capture 5	21	21	IFS0<21>	IEC0<21>	IPC5<12:10>	IPC5<9:8>
OC5 – Output Compare 5	22	22	IFS0<22>	IEC0<22>	IPC5<20:18>	IPC5<17:16>
SPI1E – SPI1 Fault	23	23	IFS0<23>	IEC0<23>	IPC5<28:26>	IPC5<25:24>
SPI1RX – SPI1 Receive Done	24	23	IFS0<24>	IEC0<24>	IPC5<28:26>	IPC5<25:24>
SPI1TX – SPI1 Transfer Done	25	23	IFS0<25>	IEC0<25>	IPC5<28:26>	IPC5<25:24>
U1AE – UART1A Error SPI1AE – SPI1A Fault I2C1AB – I2C1A Bus Collision Event	26	24	IFS0<26>	IEC0<26>	IPC6<4:2>	IPC6<1:0>
U1ARX – UART1A Receiver SPI1ARX – SPI1A Receive Done I2C1AS – I2C1A Slave Event	27	24	IFS0<27>	IEC0<27>	IPC6<4:2>	IPC6<1:0>
U1ATX – UART1A Transmitter SPI1ATX – SPI1A Transfer Done I2C1AM – I2C1A Master Event	28	24	IFS0<28>	IEC0<28>	IPC6<4:2>	IPC6<1:0>
I2C1B – I2C1 Bus Collision Event	29	25	IFS0<29>	IEC0<29>	IPC6<12:10>	IPC6<9:8>
I2C1S – I2C1 Slave Event	30	25	IFS0<30>	IEC0<30>	IPC6<12:10>	IPC6<9:8>

Note 1: Not all interrupt sources are available on all devices. See TABLE 1: **"PIC32MX Features"** for the list of available peripherals.

TABLE 7-1: INTERRUPT IRQ, VECTOR, AND BIT LOCATION (CONTINUED)

Interrupt Source[1]	IRQ	Vector Number	Interrupt Bit Location			
			Flag	Enable	Priority	Sub-Priority
I2C1M – I2C1 Master Event	31	25	IFS0<31>	IEC0<31>	IPC6<12:10>	IPC6<9:8>
CN – Input Change Interrupt	32	26	IFS1<0>	IEC1<0>	IPC6<20:18>	IPC6<17:16>
AD1 – ADC1 Convert Done	33	27	IFS1<1>	IEC1<1>	IPC6<28:26>	IPC6<25:24>
PMP – Parallel Master Port	34	28	IFS1<2>	IEC1<2>	IPC7<4:2>	IPC7<1:0>
CMP1 – Comparator Interrupt	35	29	IFS1<3>	IEC1<3>	IPC7<12:10>	IPC7<9:8>
CMP2 – Comparator Interrupt	36	30	IFS1<4>	IEC1<4>	IPC7<20:18>	IPC7<17:16>
U2AE – UART2A Error SPI2AE – SPI2A Fault I2C2AB – I2C2A Bus Collision Event	37	31	IFS1<5>	IEC1<5>	IPC7<28:26>	IPC7<25:24>
U2ARX – UART2A Receiver SPI2ARX – SPI2A Receive Done I2C2AS – I2C2A Slave Event	38	31	IFS1<6>	IEC1<6>	IPC7<28:26>	IPC7<25:24>
U2ATX – UART2A Transmitter SPI2ATX – SPI2A Transfer Done IC2AM – I2C2A Master Event	39	31	IFS1<7>	IEC1<7>	IPC7<28:26>	IPC7<25:24>
U3AE – UART3A Error SPI3AE – SPI3A Fault I2C3AB – I2C3A Bus Collision Event	40	32	IFS1<8>	IEC1<8>	IPC8<4:2>	IPC8<1:0>
U3ARX – UART3A Receiver SPI3ARX – SPI3A Receive Done I2C3AS – I2C3A Slave Event	41	32	IFS1<9>	IEC1<9>	IPC8<4:2>	IPC8<1:0>
U3ATX – UART3A Transmitter SPI3ATX – SPI3A Transfer Done IC3AM – I2C3A Master Event	42	32	IFS1<10>	IEC1<10>	IPC8<4:2>	IPC8<1:0>
I2C2B – I2C2 Bus Collision Event	43	33	IFS1<11>	IEC1<11>	IPC8<12:10>	IPC8<9:8>
I2C2S – I2C2 Slave Event	44	33	IFS1<12>	IEC1<12>	IPC8<12:10>	IPC8<9:8>
I2C2M – I2C2 Master Event	45	33	IFS1<13>	IEC1<13>	IPC8<12:10>	IPC8<9:8>
FSCM – Fail-Safe Clock Monitor	46	34	IFS1<14>	IEC1<14>	IPC8<20:18>	IPC8<17:16>
RTCC – Real-Time Clock	47	35	IFS1<15>	IEC1<15>	IPC8<28:26>	IPC8<25:24>
DMA0 – DMA Channel 0	48	36	IFS1<16>	IEC1<16>	IPC9<4:2>	IPC9<1:0>
DMA1 – DMA Channel 1	49	37	IFS1<17>	IEC1<17>	IPC9<12:10>	IPC9<9:8>
DMA2 – DMA Channel 2	50	38	IFS1<18>	IEC1<18>	IPC9<20:18>	IPC9<17:16>
DMA3 – DMA Channel 3	51	39	IFS1<19>	IEC1<19>	IPC9<28:26>	IPC9<25:24>
DMA4 – DMA Channel 4	52	40	IFS1<20>	IEC1<20>	IPC10<4:2>	IPC10<1:0>
DMA5 – DMA Channel 5	53	41	IFS1<21>	IEC1<21>	IPC10<12:10>	IPC10<9:8>
DMA6 – DMA Channel 6	54	42	IFS1<22>	IEC1<22>	IPC10<20:18>	IPC10<17:16>
DMA7 – DMA Channel 7	55	43	IFS1<23>	IEC1<23>	IPC10<28:26>	IPC10<25:24>
FCE – Flash Control Event	56	44	IFS1<24>	IEC1<24>	IPC11<4:2>	IPC11<1:0>
USB – USB Interrupt	57	45	IFS1<25>	IEC1<25>	IPC11<12:10>	IPC11<9:8>
CAN1 – Control Area Network 1	58	46	IFS1<26>	IEC1<26>	IPC11<20:18>	IPC11<17:16>
CAN2 – Control Area Network 2	59	47	IFS1<27>	IEC1<27>	IPC11<28:26>	IPC11<25:24>
ETH – Ethernet Interrupt	60	48	IFS1<28>	IEC1<28>	IPC12<4:2>	IPC12<1:0>

Note 1: Not all interrupt sources are available on all devices. See TABLE 1: **"PIC32MX Features"** for the list of available peripherals.

TABLE 7-1: INTERRUPT IRQ, VECTOR, AND BIT LOCATION (CONTINUED)

Interrupt Source[1]	IRQ	Vector Number	Interrupt Bit Location			
			Flag	Enable	Priority	Sub-Priority
IC1E – Input Capture 1 Error	61	5	IFS1<29>	IEC1<29>	IPC1<12:10>	IPC1<9:8>
IC2E – Input Capture 2 Error	62	9	IFS1<30>	IEC1<30>	IPC2<12:10>	IPC2<9:8>
IC3E – Input Capture 3 Error	63	13	IFS1<31>	IEC1<31>	IPC3<12:10>	IPC3<9:8>
IC4E – Input Capture 4 Error	64	17	IFS2<0>	IEC2<0>	IPC4<12:10>	IPC4<9:8>
IC4E – Input Capture 5 Error	65	21	IFS2<1>	IEC2<1>	IPC5<12:10>	IPC5<9:8>
PMPE – Parallel Master Port Error	66	28	IFS2<2>	IEC2<2>	IPC7<4:2>	IPC7<1:0>
U1BE – UART1B Error	67	49	IFS2<3>	IEC2<3>	IPC12<12:10>	IPC12<9:8>
U1BRX – UART1B Receiver	68	49	IFS2<4>	IEC2<4>	IPC12<12:10>	IPC12<9:8>
U1BTX – UART1B Transmitter	69	49	IFS2<5>	IEC2<5>	IPC12<12:10>	IPC12<9:8>
U2BE – UART2B Error	70	50	IFS2<6>	IEC2<6>	IPC12<20:18>	IPC12<17:16>
U2BRX – UART2B Receiver	71	50	IFS2<7>	IEC2<7>	IPC12<20:18>	IPC12<17:16>
U2BTX – UART2B Transmitter	72	50	IFS2<8>	IEC2<8>	IPC12<20:18>	IPC12<17:16>
U3BE – UART3B Error	73	51	IFS2<9>	IEC2<9>	IPC12<28:26>	IPC12<25:24>
U3BRX – UART3B Receiver	74	51	IFS2<10>	IEC2<10>	IPC12<28:26>	IPC12<25:24>
U3BTX – UART3B Transmitter	75	51	IFS2<11>	IEC2<11>	IPC12<28:26>	IPC12<25:24>
(Reserved)	—	—	—	—	—	—
Lowest Natural Order Priority						

Note 1: Not all interrupt sources are available on all devices. See TABLE 1: **"PIC32MX Features"** for the list of available peripherals.

A.2. Schaltung und Layout

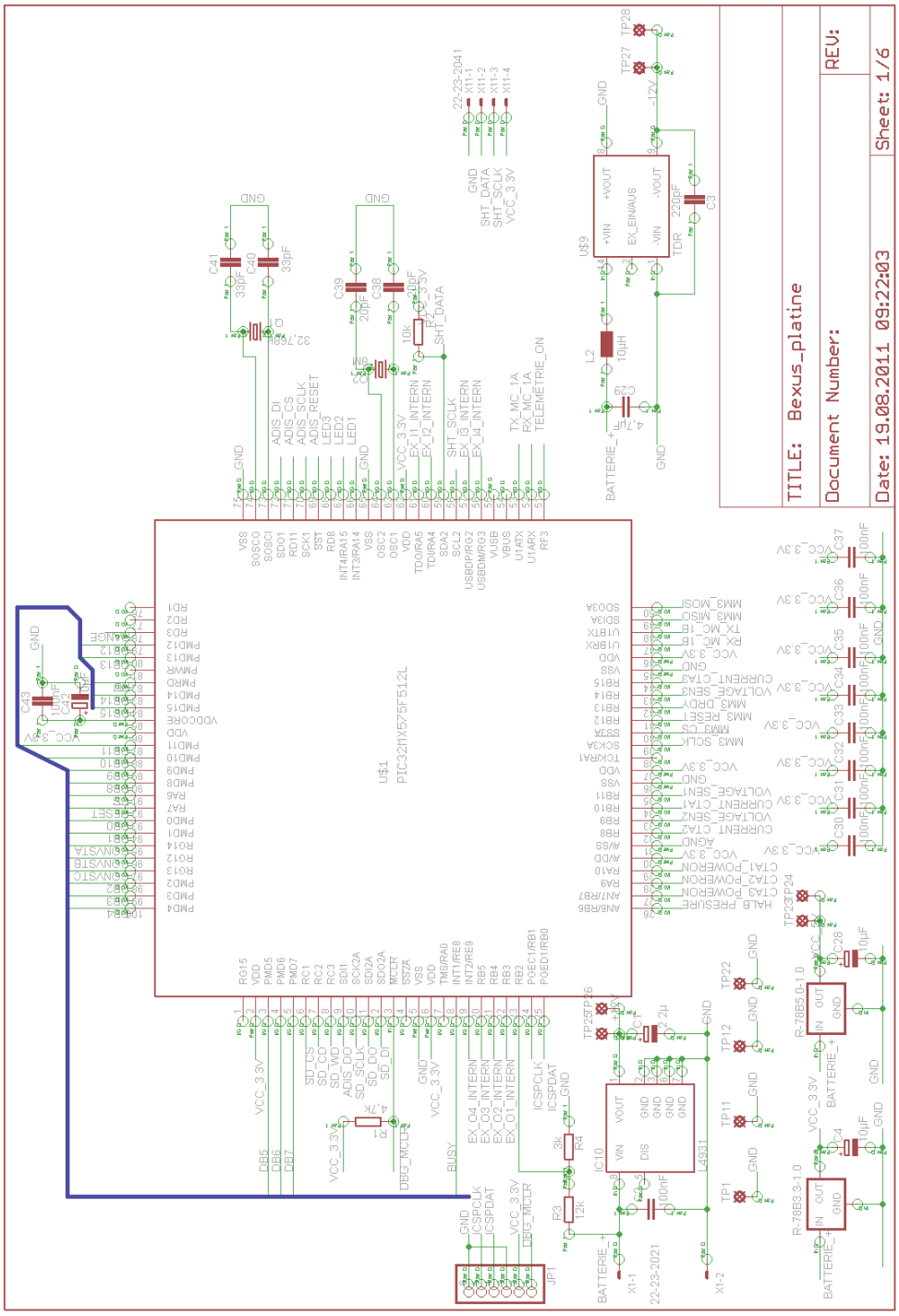


Bild A.1.: Schaltplan Messdatenerfassungskarte (Seite 1)

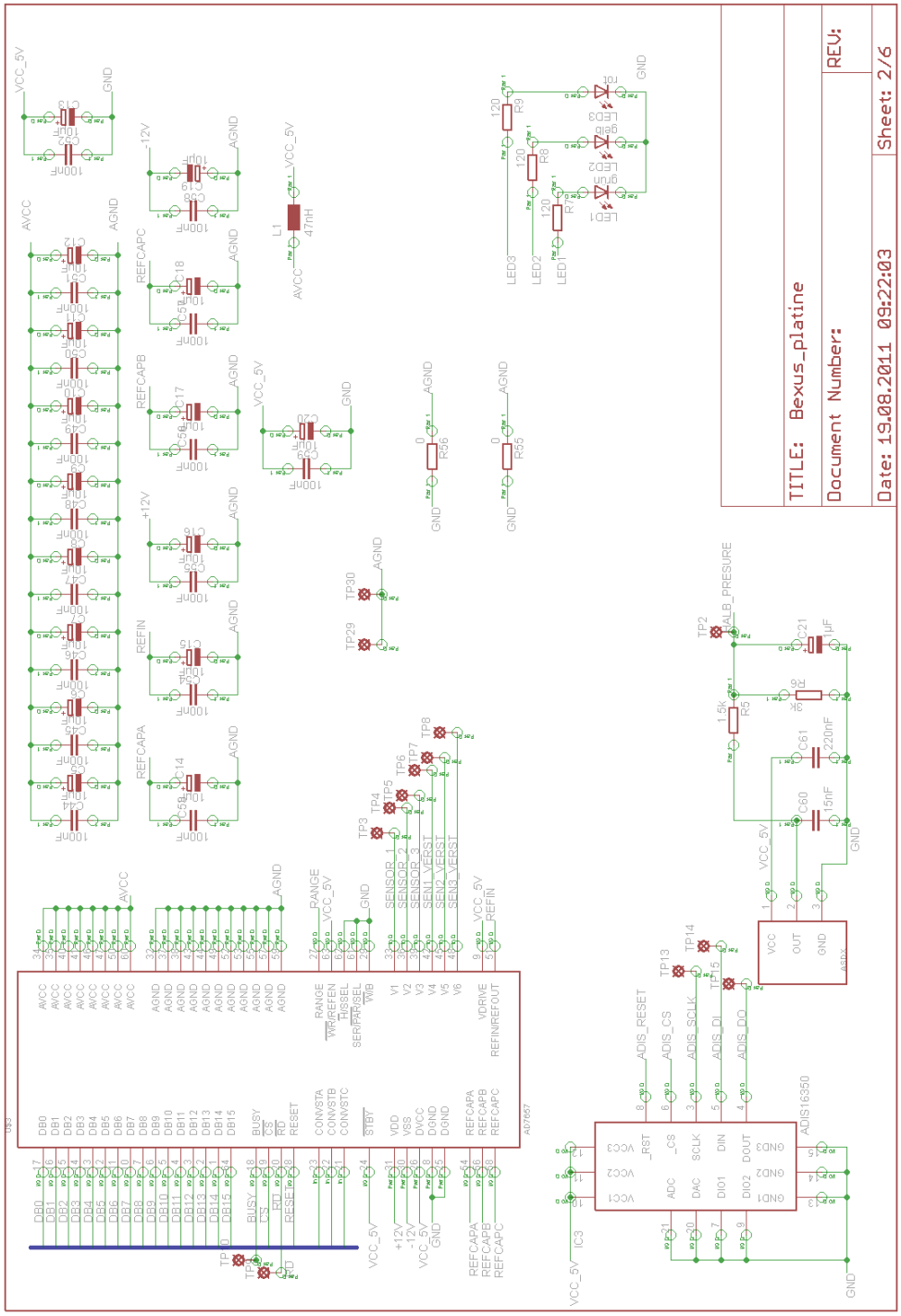


Bild A.2.: Schaltplan Messdatenerfassungskarte (Seite 2)

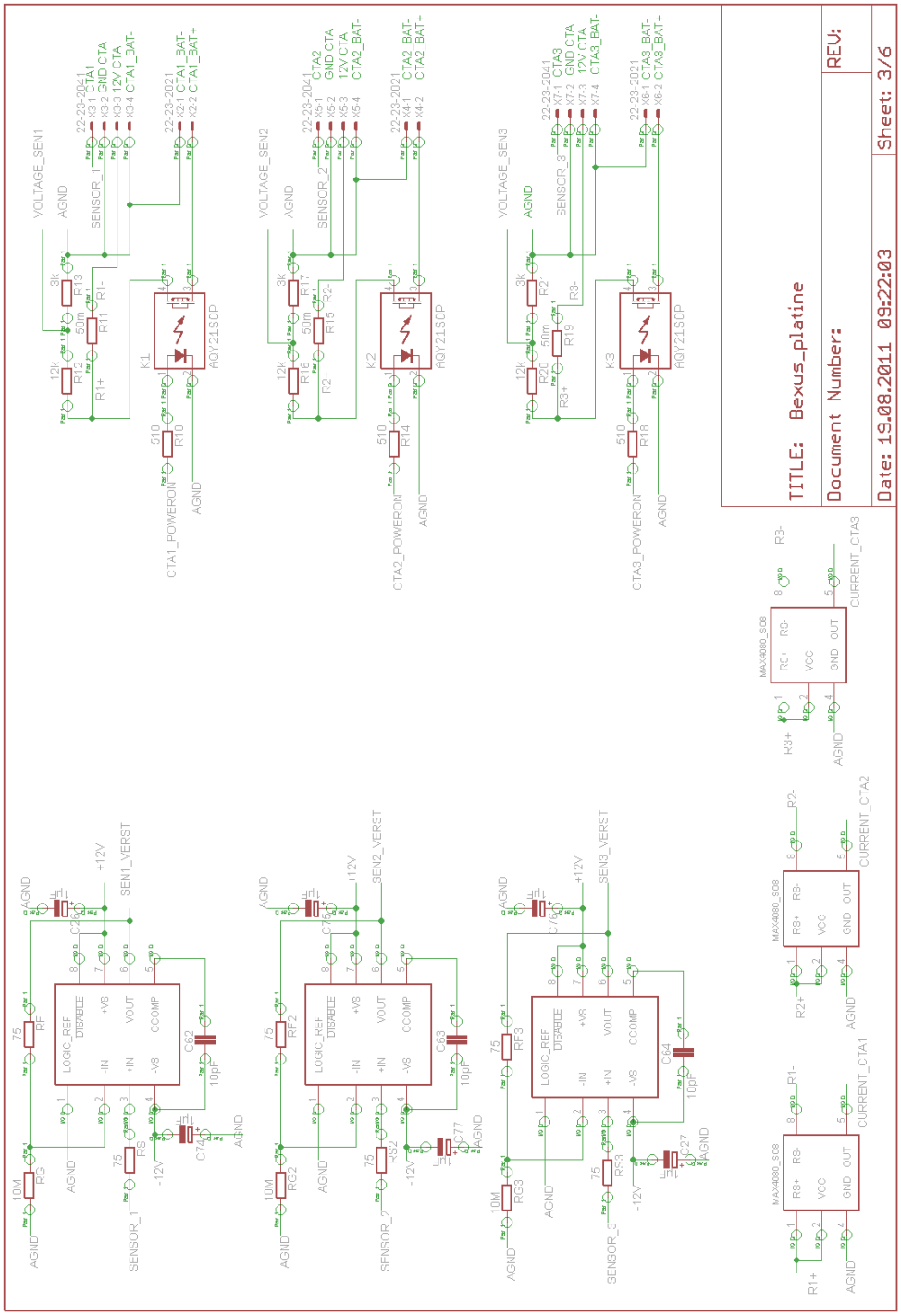


Bild A.3.: Schaltplan Messdatenerfassungskarte (Seite 3)

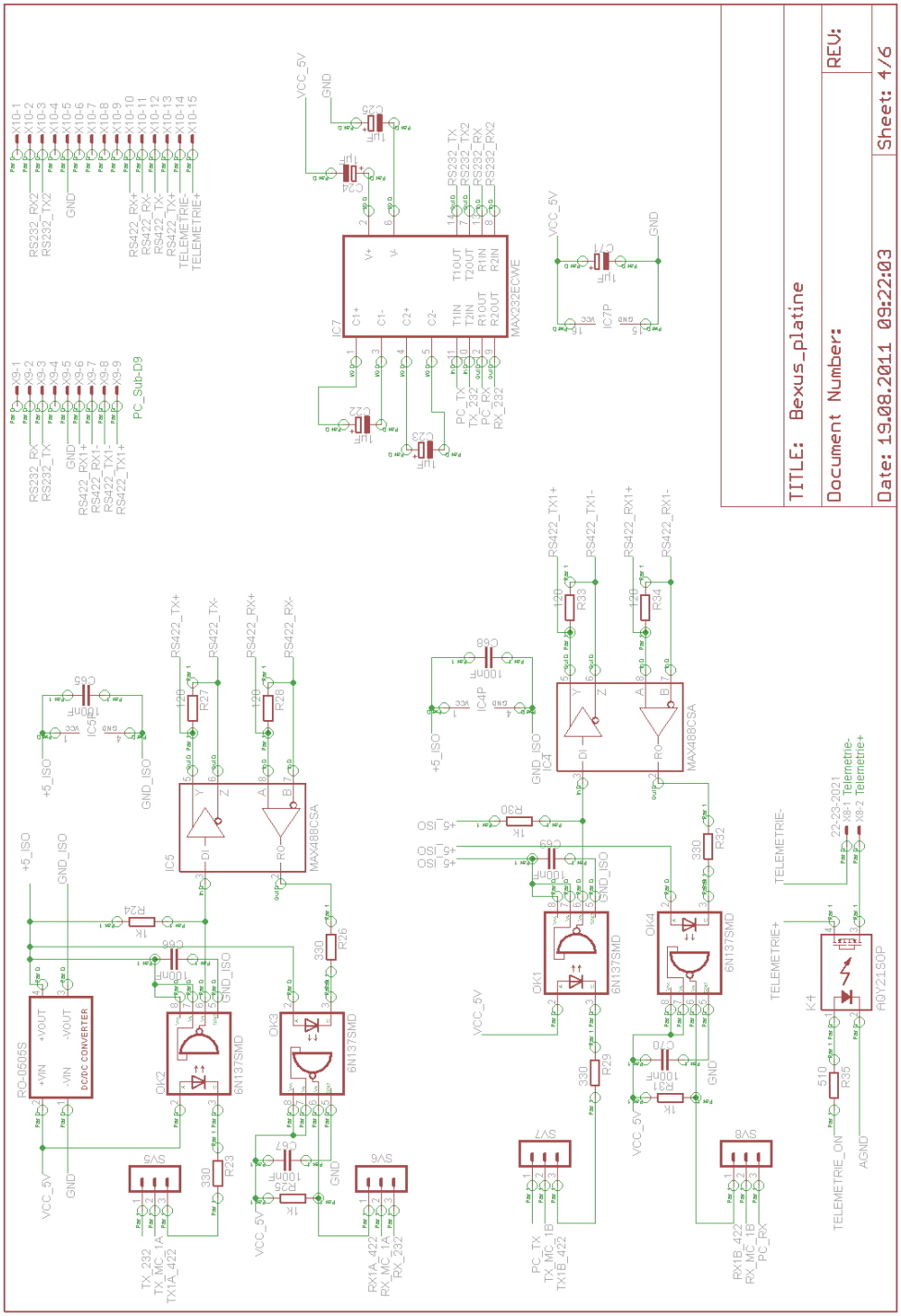


Bild A.4.: Schaltplan Messdatenerfassungskarte (Seite 4)

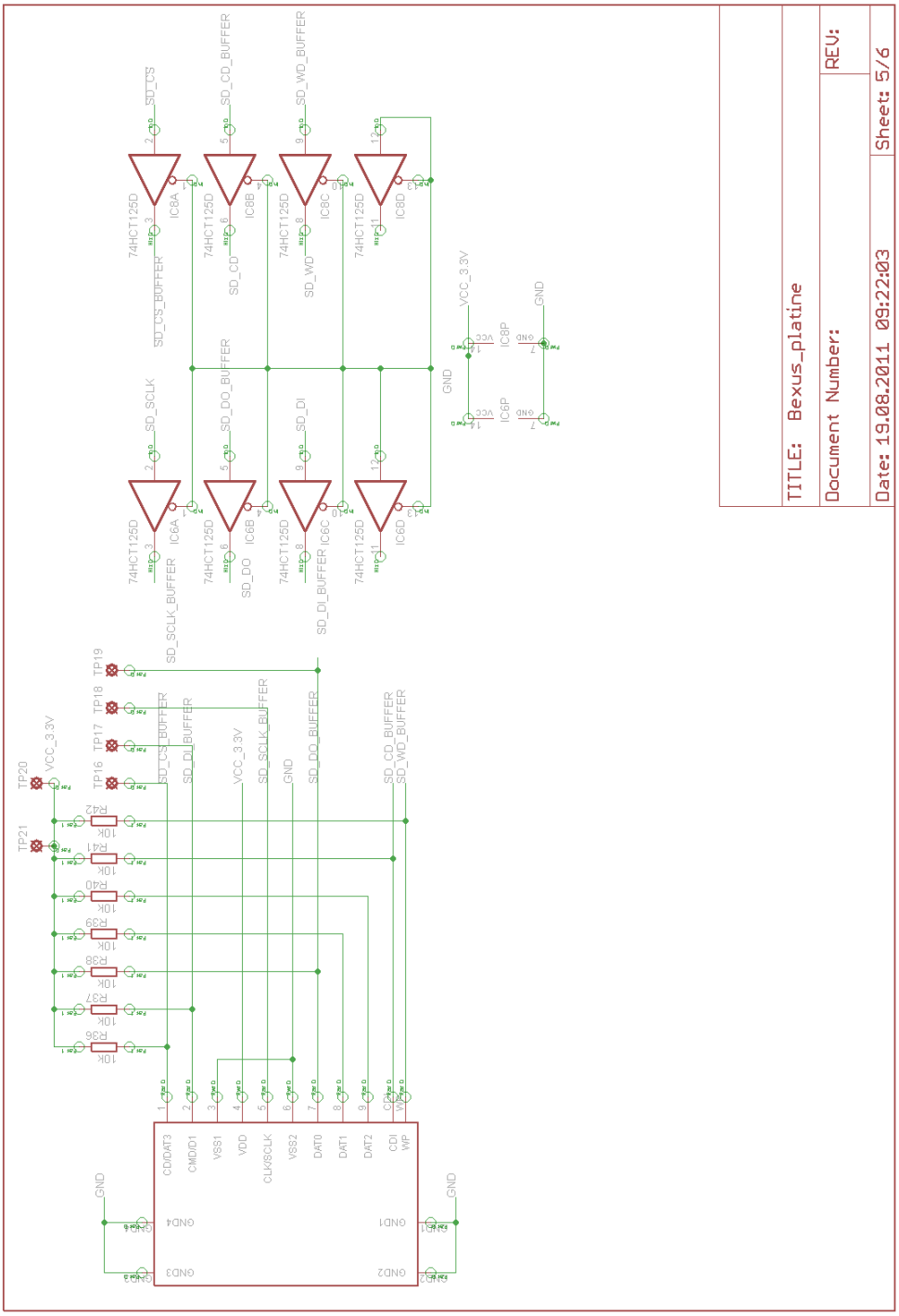


Bild A.5.: Schaltplan Messdatenerfassungskarte (Seite 5)

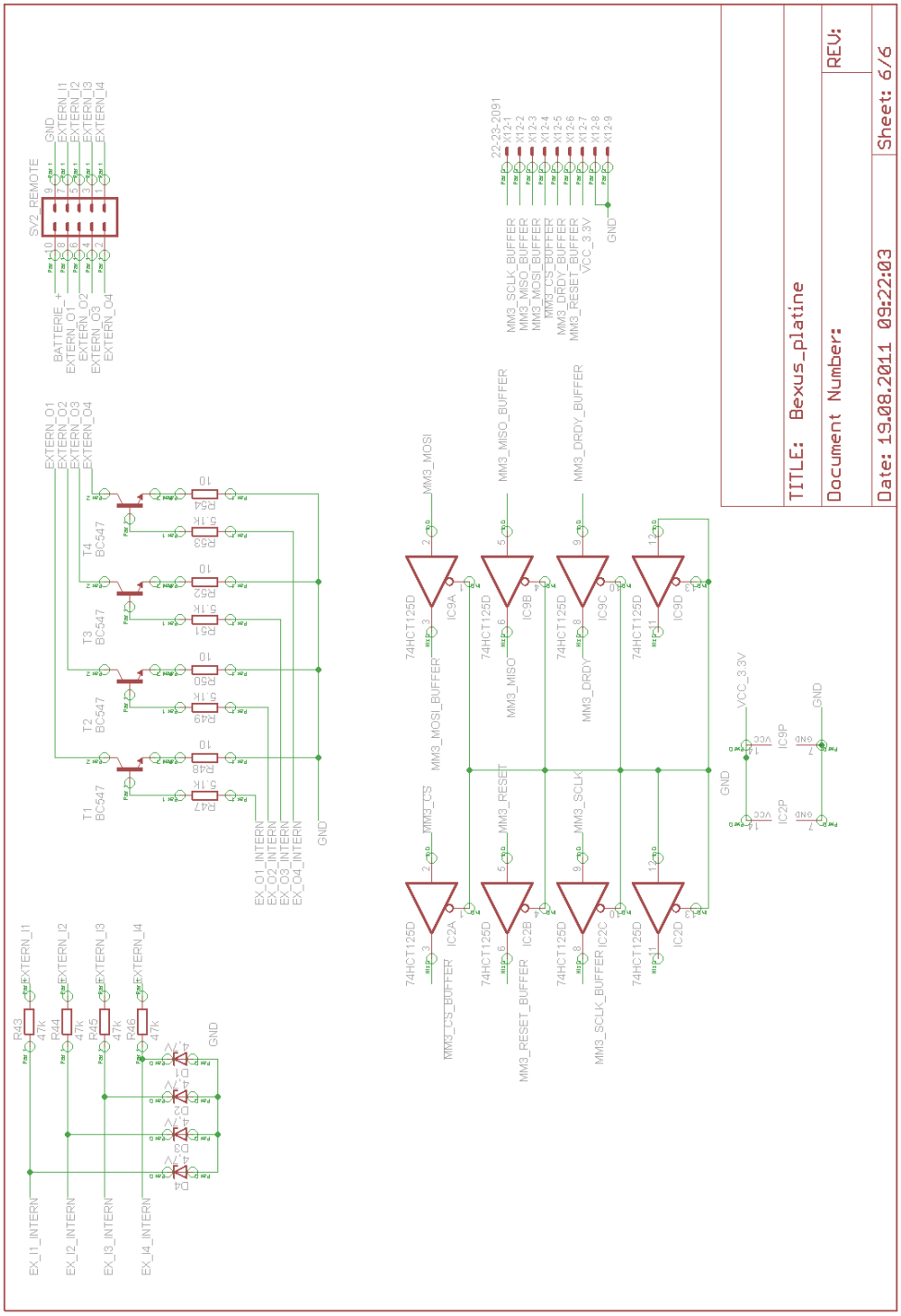


Bild A.6.: Schaltplan Messdatenerfassungskarte (Seite 6)

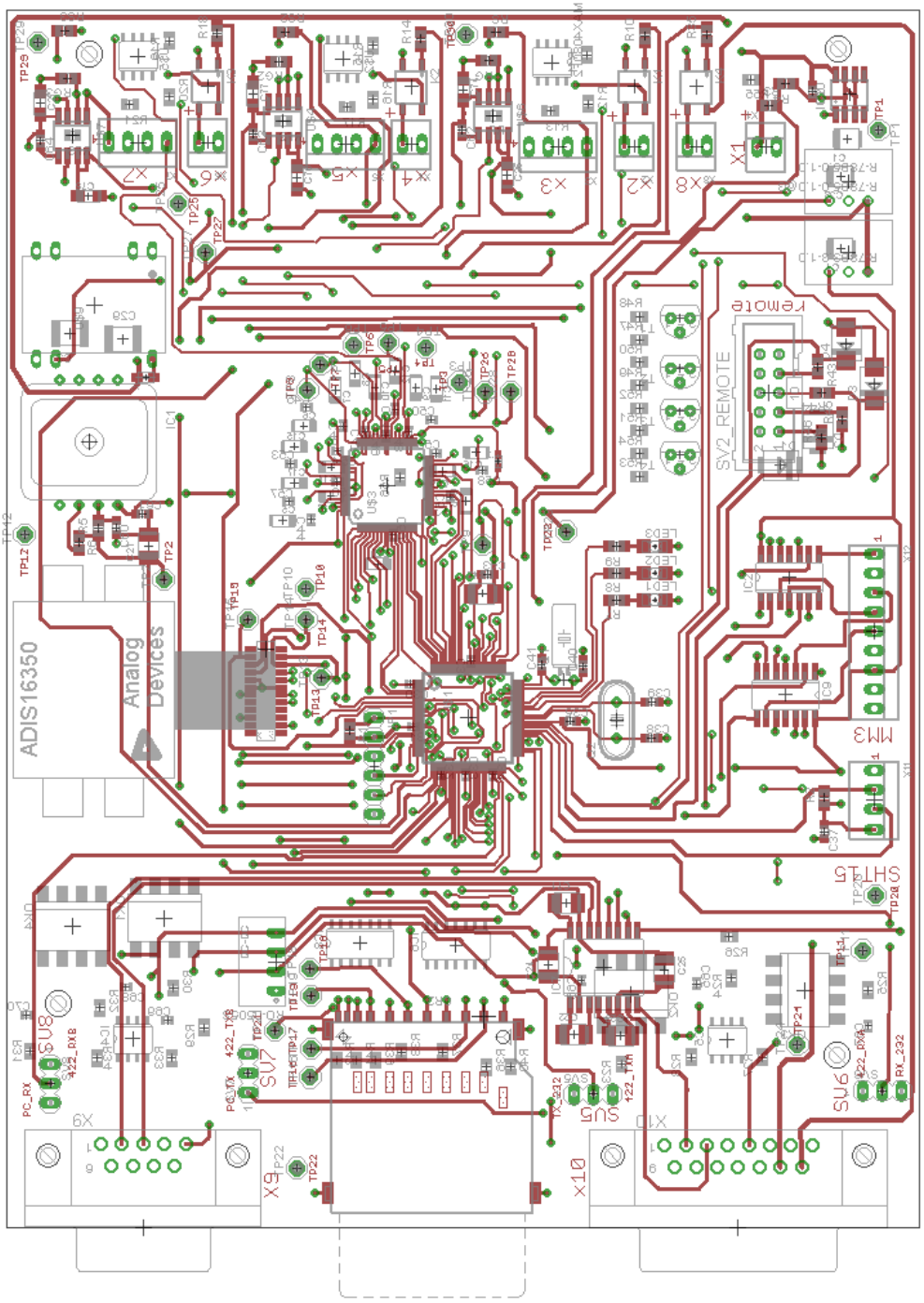


Bild A.7.: Layout der Messdatenerfassungskarte (oben)

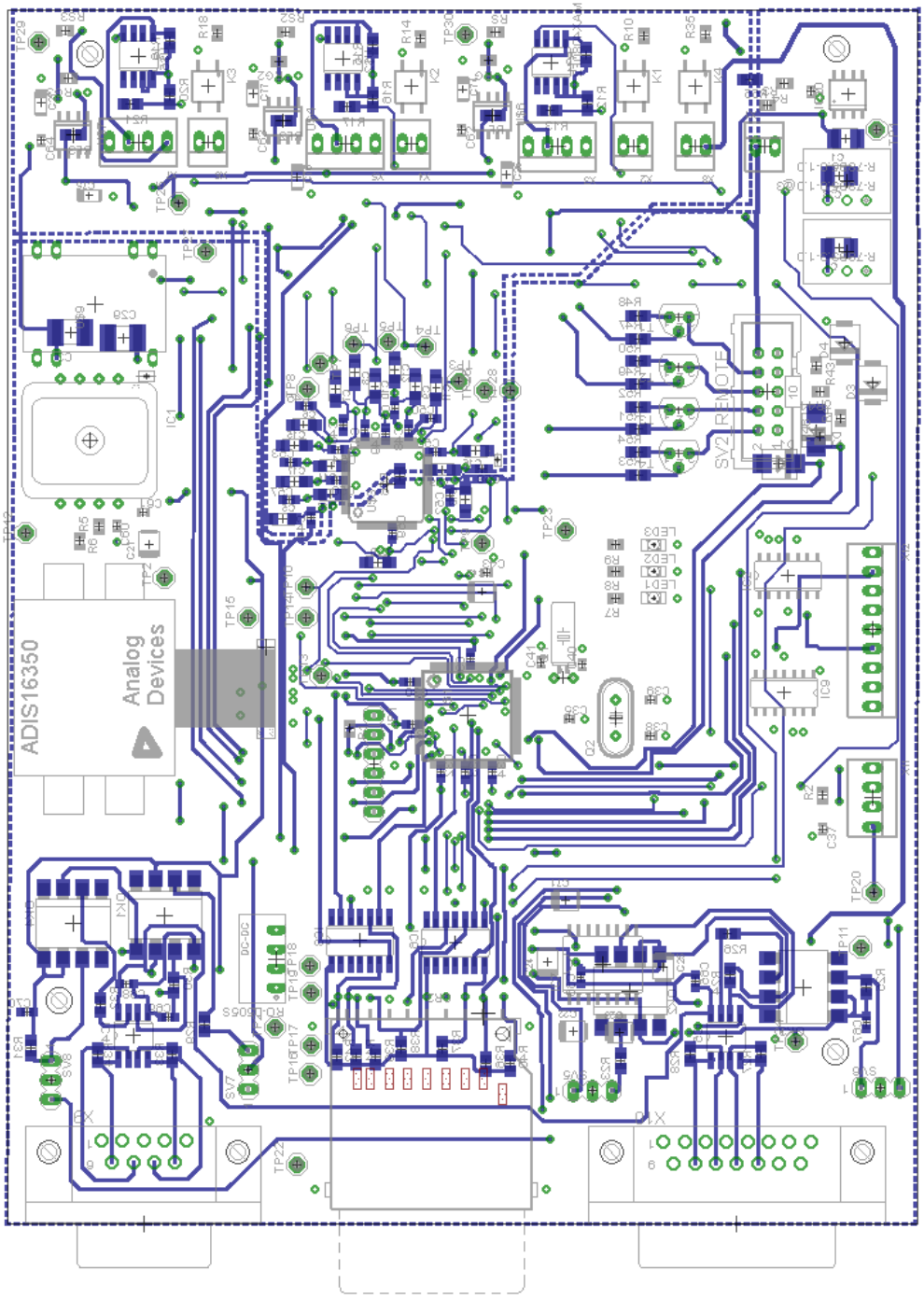


Bild A.8.: Layout der Messdatenerfassungskarte (unten)

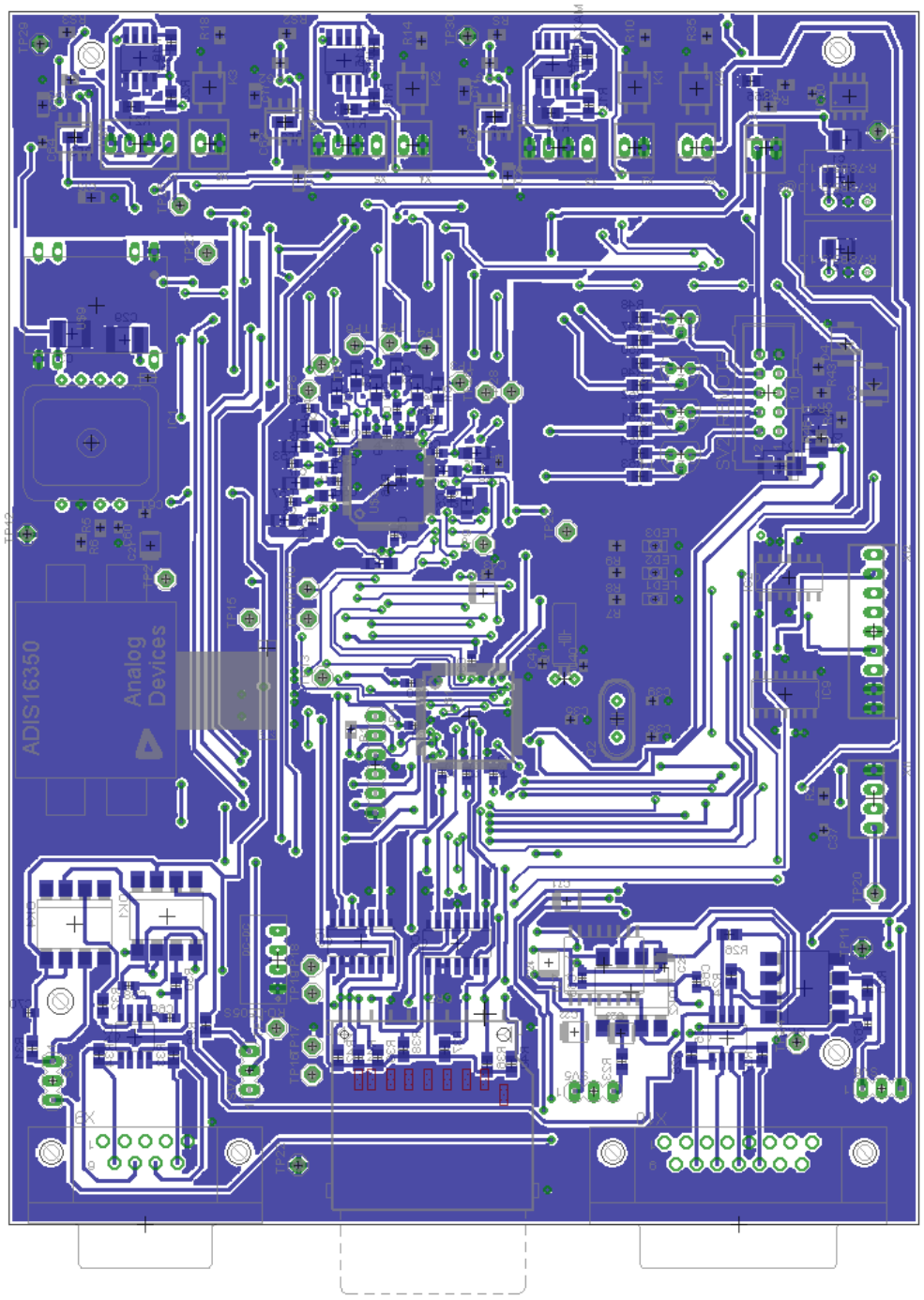


Bild A.9.: Layout der Messdatenerfassungskarte (unten mit Massefläche)

Bild A.10.: Messdatenerfassungskarte von oben

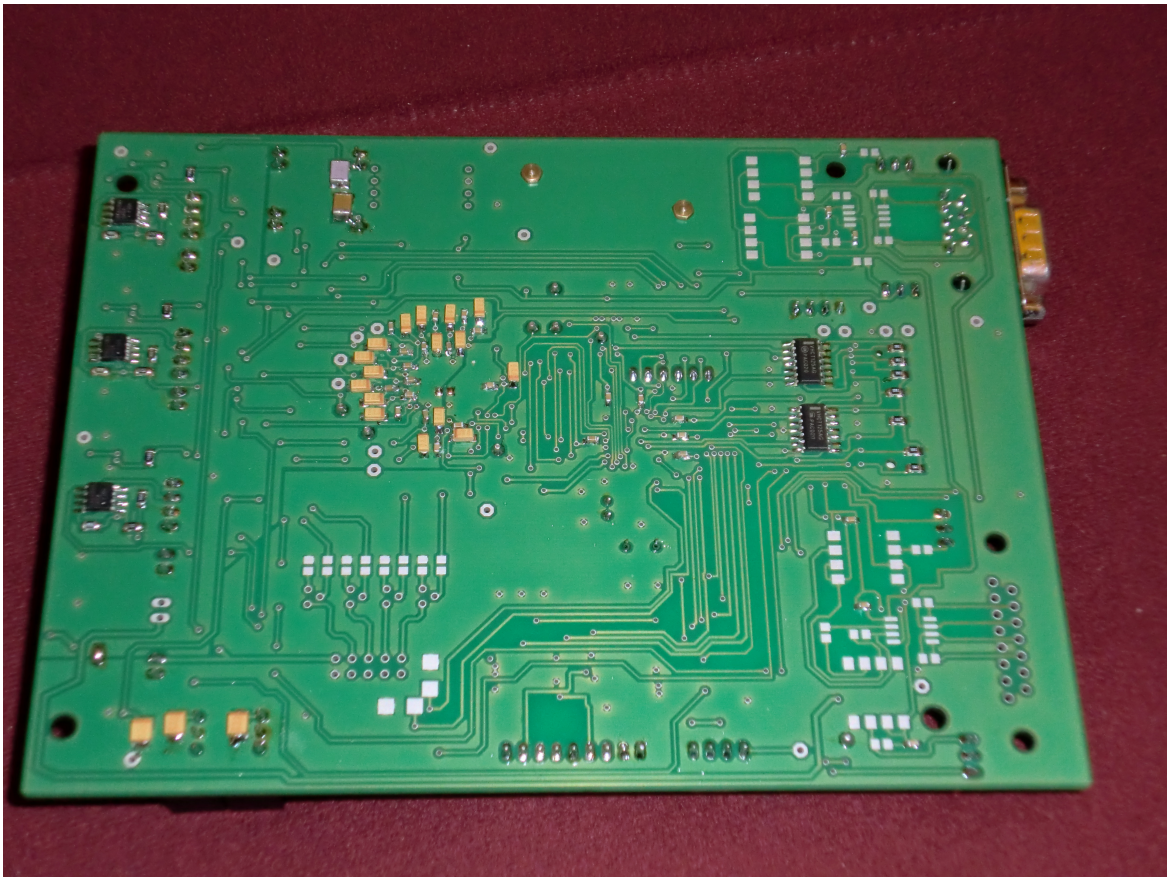


Bild A.11.: Messdatenerfassungskarte von unten

A.3. Software Quellcode

A.3.1. Include-Files

CD // Bexus / Header / BEXUS.h
CD // Bexus / Header / Compiler.h
CD // Bexus / Header / FScpnfig.h
CD // Bexus / Header / FScpnfig.h
CD // Bexus / Header / FSIO.h
CD // Bexus / Header / HardwareProfile.h
CD // Bexus / Header / SD-SPI.h

A.3.2. Source-Files

CD // Source Code / aufbereiten_funktion.c
CD // Source Code / bexus_main.c
CD // Source Code / FSIO.c
CD // Source Code / Funktion.c
CD // Source Code / SD_SPI.c

Abkürzungsverzeichnis

ACK Acknowledge

ADU Analog-Digital-Umsetzer

am/pm Ante meridiem (vor dem Mittag) / Post meridiem (nach dem Mittag)

ASAM-Bit ADC Sample AUTO-Start-Bit

ASIC Application-Specific Integrated Circuit

BCD Binary Coded Decimal

BEXUS Balloon-borne EXperiments for University Students

CCA Constant Current Anemometer

CPU Central Processing Unit

CS Chip Select

CTA Constant Temperature Anemometer

DATA Datenleitung

DCDC-Wandler Gleichspannungswandler

DIN Data Input

DMA Direct Memory Access

DOUT Data Output

EMV elektromagnetische Verträglichkeit

HK Lagedaten

I^2C Inter-Integrated Circuit

IAP Leibniz-Institut für Atomsphärenphysik

IRQ Interrupt-Quelle engl. Interrupt Request

ISR Interrupt Service Routine

LSB Least Significant Byte

MEMS Micro-Electro-Mechanical Systems

MISO Master In Slave Out

MOSI Master Out Slave In

MSB Most Significant Byte

msb Most Significant Bit

ND Data Ready

PC Personal Computer

PLL Phase Locked Loop

PSI Pound-Force Per Square Inch

PWM Pulsweitenmodulation

RAM Random Access Memory

RD Read Data

RISC Reduced Instruction Set Computer

RTCC Real-Time Clock und Calendar

RTOS Real-Time Operating System

SCK Taktleitung SHT-Sensor

SCLK Serial Clock

SD Memory Card Secure Digital Memory Card

SSNOT Slave Select Line

TTL Transistor-Transistor-Logik

UART Universal Asynchronous Receiver/Transmitter

Versicherung über die Selbstständigkeit

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

Hamburg, 14. September 2011	
Ort, Datum	Unterschrift