HAW HAMBURG

Bachelorarbeit

Torben Dierks

Entwicklung eines Beschleunigungssensors für hohe Frequenzen mit CAN-Schnittstelle

Fakultät Technik und Informatik *Faculty of Engineering and Computer Science*
Department Fahrzeugtechnik und Flugzeugbau *Department of Automotive and Aeronautical Engineering*

Torben Dierks

Entwicklung eines Beschleunigungssensors für hohe Frequenzen mit CAN-Schnittstelle

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Mechatronik
am Department Fahrzeugtechnik und Flugzeugbau
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüfer: Prof. Dr. Rasmus Rettig
Zweitprüfer: Prof. Dr.-Ing. Benedikt Plaumann

Abgabedatum: 03. Mai 2023

Kurzreferat

Torben Dierks

Thema der Arbeit

Entwicklung eines Beschleunigungssensors für hohe Frequenzen mit CAN-Schnittstelle

Stichworte

Beschleunigungssensor, Sensor, C, Python, Datenlogger, Controller Area Network, Mikrocontroller, Fast Fourier Transformation

Kurzreferat

In der angeführten Bachelorarbeit wird ein Beschleunigungssensor mit CAN-Schnittstelle entwickelt. Hierfür wird der Beschleunigungssensor unter Berücksichtigung der gegebenen Anforderungen programmiert. Besonderheit des verwendeten Beschleunigungssensors ist die „on-chip“ Berechnung einer FFT. Konzepte für eine effiziente Abfrage von Messwerten und Programmabläufe werden entwickelt und auf einem Mikrocontroller mit CAN-Schnittstelle implementiert. Der Beschleunigungssensor und Mikrocontroller mit CAN-Schnittstelle werden zusammen in einem selbstentwickelten Gehäuse verbaut. Abschließend werden Messungen zur Charakterisierung des Beschleunigungssensors durchgeführt.

Abstract

Torben Dierks

Title of Thesis

Development of an accelerometer for high frequencies with a CAN interface

Keywords

accelerometer, sensor, C, Python, data logger, Controller Area Network, microcontroller, fast fourier transform

Abstract

The present bachelor thesis an accelerometer with a CAN interface is developed. For this the accelerometer is programmed under consideration of the given requirements. Special feature of the used accelerometer is the on-chip calculation of a FFT. Concepts for an efficient query of measurement values and program sequences are developed and implemented on a microcontroller with CAN interface. Accelerometer and microcontroller with a CAN interface are packaged in a self-designed case. Finally, measurements are carried out to characterize the accelerometer.

Aufgabenstellung Abschlussarbeit

Name:
Torben Dierks

Thema:
Entwicklung eines Beschleunigungssensors für hohe Frequenzen mit CAN-Schnittstelle

1. Einführung:
Das Urban Mobility Lab an der HAW Hamburg hat sich unter anderem auf die Erhebung von verschiedensten Messwerten mit Datenloggern und eine anschließende Analyse dieser spezialisiert. Die Datenlogger erhalten von den verschiedenen Sensoren über den Controller Area Network (CAN bus) die Messwerte, speichern diese zwischen und leiten sie anschließend für die spätere Datenanalyse weiter.

Zu den aufgenommenen Messwerten zählt die Beschleunigung. Die aktuell verwendeten Sensoren sind in ihrem messbaren Frequenzbereich begrenzt. Für zukünftige Anwendungen der Beschleunigungssensoren sollte der messbare Frequenzbereich erweitert werden. Hierfür ist die Integration eines neuen Beschleunigungssensors notwendig.

2. Aufgabenstellung:

- Dokumentation über das Sensorelement ADcmXL3021 von Analog Devices
- Konfiguration des Beschleunigungssensors ADcmXL3021 von Analog Devices unter Beachtung der Anforderungen in den Betriebsmodi:
  - Zeitbereich
  - Frequenzbereich
    - Manuelles Anfordern von FFTs
    - automatische periodische Bereitstellung von FFTs
- Programmieren eines Mikrocontrollers für die Konfiguration des Beschleunigungssensors sowie das Abrufen der Messwerte
- Integration des Beschleunigungssensors, mit den weiteren Komponenten (Mikrocontroller, Spannungsversorgung, CAN-Schnittstelle), in ein Gehäuse für die Befestigung an z.B. Gabelstaplern
- Integration der CAN-Schnittstelle mit Datenlogger unter Beachtung der Randbedingungen gegeben durch den CAN-Bus

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

μC Mikrocontroller

AFFT automatic fast fourier transform

BRS bit rate switch

CAN Controller Area Network

CAN-FD Controller Area Network with flexible data rate

CRC cyclic redundancy check

EDL extended data length

ESI error state indicator

FFT fast fourier transform

FG Funktionsgenerator

FIR finite impulse response

IPT Information Processing Time

LED light emitting diode

MCC MPLAB Code Configurator

MEMS microelectromechanical systems

MFFT manual fast fourier transform

MTC manual time capture

PhaSeg1 Phase Segment 1

PhaSeg2 Phase Segment 2

PLL phase locked loop

PropSeg Propagation Segment

RTS real time streaming

SDI Serial Data In

SDO Serial Data Out

SJW Synchronization Jump Width

SM seismische Masse

SPI Serial Peripheral Interface

SyncSeg Synchronization Segment

TQ Time Quantum

1 Einleitung

Die Beschleunigung, also eine Veränderung der Geschwindigkeit über die Zeit, kann als physikalische Größe für viele verschiedenen Bereiche verwendet werden. In Handys wird sie verwendet um die Orientierung im Raum zu bestimmen, während sie in Autos verwendet wird um Unfälle zu erkennen und die Airbags auszulösen. Beide Anwendungsgebiete benötigen sehr unterschiedlichen Anforderungen an die Sensorik bezüglich Reaktionszeit und gemessener Beschleunigung. Ein weiteres Feld ist die Verwendung von Beschleunigungssensoren als ein Mittel zur Datenerhebung. Nach der Aufnahme von Messwerten können zum Beispiel Kenntnisse über auftretende dynamische Belastungen oder die Bodenbeschaffenheit gewonnen werden. Sie können aber auch dafür verwendet werden das Auftreten von Beschleunigungen mit bestimmten Frequenzen vor Ausfällen zu korrelieren. In dem vom Urban Mobility Lab an der HAW Hamburg für Industriekunden implementierten Datenloggerssystem, finden Beschleunigungssensoren in solch einem Umfeld Anwendung.

Der in dieser Arbeit betrachtet Beschleunigungssensor ADcmXL3021 von Analog Devices, besitzt die Besonderheit einer „on-chip“ Berechnung einer schnellen Fourier-Transformation. Im Verlauf dieser Arbeit wird zuerst eine Grundlage geschaffen, um eine erfolgreiche Integration des Beschleunigungssensors in das bestehende Datenloggerssystem zu ermöglichen. Hierfür wird im ersten Schritt umfangreiches Wissen über die Funktionsweise und Programmierung des verwendeten Beschleunigungssensor erarbeitet.

Für eine Integration in das bestehende Datenloggerssystem, wird eine Controller Area Network (CAN)-Schnittstelle benötigt. Über die CAN-Schnittstelle erfolgt die Konfiguration des Beschleunigungssensors, aber auch der Austausch von Messwerten. Dafür wird im zweiten Schritt ein Mikrocontroller implementiert. Dies erfordert die Entwicklung von Programmabläufen, die Entwicklung von Platinen für den Mikrocontroller und seine Programmierung unter Berücksichtigung der entwickelten Programmabläufe. Für eine effiziente Bereitstellung von Messwerten und die Kommunikation mit dem Beschleunigungssensor, wird ein System zur Unterscheidung von CAN-Nachrichten entwickelt. Eine

Verwendung im Datenloggerssystem erfordert außerdem die Integration aller Komponenten in einem Gehäuse.

Im letzten Schritt der Arbeit wird das entwickelte System bestehend aus dem Beschleunigungssensor, den Platinen und der CAN-Schnittstelle in dem Gehäuse getestet. Gegenstand der Messung sind die Charakterisierung des Messverhaltens des Beschleunigungssensors und eine Langzeitmessung. Die Charakterisierung erfolgt durch Messungen, mit definierten Parametern, auf einer Schwingprüfanlage. Für die Langzeitmessung werden die auftretenden Beschleunigung einer Hydraulikpumpe gemessen.

2 Stand der Technik

In diesem Kapitel wird der Stand der Technik erfasst. Zuerst werden Beschleunigungssensoren in ihrer grundsätzlichen Funktionsweise beschrieben. Außerdem wird die Verwendung von microelectromechanical systems (MEMS) als ein Fertigungsverfahren für Beschleunigungssensoren kurz beleuchtet. Es folgt die Betrachtung der beiden Bussysteme CAN und Serial Peripheral Interface (SPI). Beide Bussysteme zusammen bilden die Grundlage dafür, Messwerte schnell und zuverlässig übertragen zu können. Als letztes wird die Fourieranalyse von der Fourierreihenentwicklung bis zur fast fourier transform (FFT) beschrieben. Die FFT bildet die Grundlage für die Analyse der Messwerte.

2.1 Beschleunigungssensoren

Unterschieden wird bei der Messung von Beschleunigungen zwischen der relativen und absoluten Bestimmung der Beschleunigung. Die relative Beschleunigung beschreibt die Beschleunigung, die ein Körper im Bezug zu einem Bezugssystem erfährt. Bei einer absoluten Messung der Beschleunigung wird die Beschleunigung die ein Körper auf sich selbst erfährt bestimmt. Für relative Messungen kann die Geschwindigkeit v oder die zurückgelegte Strecke s in Relation zum Bezugssystem verwendet werden um die Beschleunigung zu bestimmen. Für die Bestimmung der Beschleunigung a gilt

$$a(t) = \dot{v}(t) = \ddot{s}(t) \tag{2.1}$$

Die Beschleunigung über die Zeit lässt sich folglich durch eine numerische Ableitung von v oder s bestimmen. Eine Bestimmung der Strecke kann zum Beispiel durch Licht erfolgen, wie es bei manchen Inkrementalgebern verwendet wird.

Eine absolute Bestimmung wird in Beschleunigungssensoren verwendet um zu bestimmen, welche Beschleunigung ein Körper absolut erfährt. Grundlage hierfür ist das zweite Newton'sche Axiom

$$F = m \cdot a \tag{2.2}$$

mit den Definitionen der Kraft F, Masse m und Beschleunigung a. Erfährt ein Körper mit der Masse m eine Beschleunigung, wirkt auf den Körper die Kraft F. Durch seine Trägheit wirkt ein Körper dieser Kraft entgegen. Für die Bestimmung von Beschleunigungen kann diese Trägheit verwendet werden. Wenn ein Körper elastisch mit einem Referenzsystem verbunden ist, welches die Beschleunigung erfährt, führt die Trägheit der Masse zu einer Verschiebung x gegenüber einem Referenzsystem. Das Referenzsystem ist zum Beispiel das Gehäuse des Beschleunigungssensors. Die Verschiebung wird bei Beschleunigungssensoren durch unterschiedliche Prinzipien gemessen und in eine Beschleunigung umgerechnet. Für Beschleunigungssensoren wird diese Masse als seismische Masse (SM) bezeichnet [32]. Die SM ist mit dem Referenzsystem über eine Aufhängung elastisch verbunden. Vereinfacht lässt sich die Aufhängung als Feder verstehen, die bei einer Beschleunigung gestaucht oder gestreckt wird. Die auf eine Feder wirkende Kraft F_F lässt sich berechnen aus

$$F_F = x \cdot k \tag{2.3}$$

dabei entspricht k der Federkonstante und x dem Federweg. Für x gilt $x > 0$ für eine Streckung und $x < 0$ für eine Stauchung bezogen auf einen Ruhezustand. Das beschriebene Messprinzip zeigt Abbildung 2.1. Hier erfährt das Referenzsystem durch die Kraft F_a eine Beschleunigung nach rechts. Die Trägheit der SM mit der Kraft F_T führt dazu, dass diese eine Verschiebung x gegenüber dem Mittelpunkt des Referenzsystems erfährt (Ruhelage im Mittelpunkt grau). Die Feder mit der Konstante k wird gestaucht, der Stauchung entgegen wirkt die Federkraft F_F. In einem Beschleunigungssensor entspricht die Feder der Aufhängung der SM.

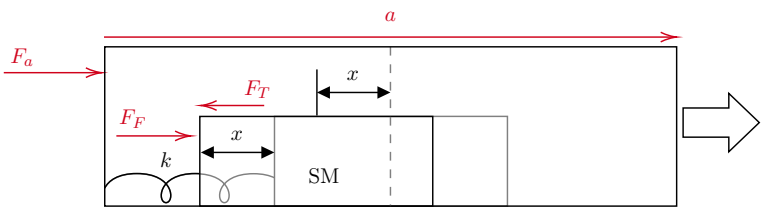


Abbildung 2.1: Grundsätzliches Messprinzip eines Beschleunigungssensors [32, S. 545]

Eine Bestimmung der Beschleunigung die das System in Abbildung 2.1 erfährt ist möglich, wenn x, m und k bekannt sind. Die Federkraft F_F wirkt der aus der Trägheit resultierenden Kraft F_T entgegen.

$$F_T = F_F \tag{2.4}$$

Setzt man Gleichung 2.2 und Gleichung 2.3 ein

$$m \cdot a = -k \cdot x \tag{2.5}$$

kann nach a umgestellt werden um die Beschleunigung zu erhalten

$$a = -\frac{k}{m} \cdot x \tag{2.6}$$

In einem vereinfachten System ließe sich so die Beschleunigung bestimmen.

In realen Systemen werden physikalische Effekte wie die Kapazität, Magnetismus, Induktion, Piezoelektrizität oder Piezoresistivität für eine Bestimmung der Verschiebung verwendet [32] [13]. Resultierende Größe aus den physikalischen Effekten ist meistens eine messbare Spannung, die proportional zur Verschiebung ist, angegeben durch die Sensitivität des Sensors. Für analoge Sensoren lässt sich die Spannung mit der Sensitivität in eine Beschleunigung umrechnen. Bei digitalen Sensoren ergibt sich die Sensitivität durch die Auflösung des Analog-Digital-Umsetzers, der die Spannung abtastet.

Bei Sensoren die auf der Piezoelektrizität beruhen, wird der piezoelektrische Effekt für die Bestimmung der Verschiebung verwendet. Eine Verformung führt zu einer Ladungsverschiebung die als Spannung messbar wird. Die Ladungsverschiebung ist nicht dauerhaft und baut sich über die Zeit ab. Wird bei einem Sensor der piezoresistive Effekt verwendet, führt eine Verformung des Materials zu einer Änderung des elektrischen Widerstandes. Eine Änderung lässt sich beispielsweise mit einer wheatstone'schen Messbrücke in eine Spannung umwandeln.

Verwendet ein Sensor die Kapazität für die Bestimmung der Beschleunigung, führt eine Verschiebung der SM zu einer Änderung in der Kapazität. Die Kapazität lässt sich in eine Spannung überführen. Verwendung findet häufig ein differenzieller Kondensator mit der Anordnung der Kondensatoren an zwei Enden der seismischen Masse, wie in Abbildung 2.2. Eine Verschiebung der SM, durch eine Beschleunigung, führt zu einer

Veränderung der Plattenabstände bei den Kondensatoren C_1 und C_2, Die Verschiebung beträgt $\pm d$ gegenüber dem Plattenabstand d_0 in Ruhelage. Dabei befindet sich eine Platte der beiden Kondensatoren an der SM, die andere ist fest an dem Referenzsystem. Damit führt eine Verschiebung der SM zu einer Veränderung der Kapazität.

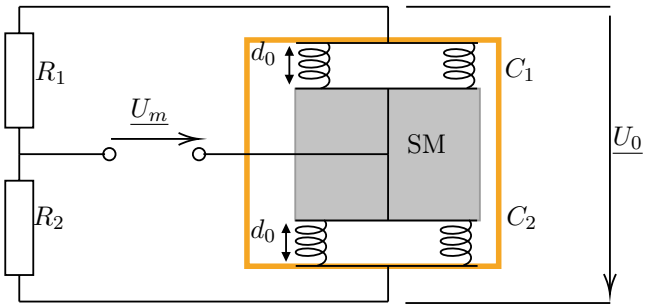


Abbildung 2.2: Differenzieller Kondensator für die Bestimmung der Beschleunigung bei kapazitiven Beschleunigungssensoren [32, S. 555]

Für das System in Abbildung 2.2 führt eine Verschiebung, durch eine Beschleunigung, zu einer proportionalen Änderung in $\underline{U_m}$ [32] [6].

$$\frac{\underline{U_m}}{\underline{U_0}} = \frac{1}{2} \cdot \frac{d}{d_0} \tag{2.7}$$

Durch die Proportionalität ist eine Bestimmung der Beschleunigung mit einem Umrechnungsfaktor aus der Ausgangsspannung $\underline{U_m}$ möglich.

Ein mittlerweile weit verbreitetes Fertigungsverfahren für die Herstellung von kapazitiven Beschleunigungssensoren ist die MEMS-Technik. Bei MEMS-Sensoren werden die erforderlichen Schaltungen und mechanischen Aufbauten im Mikrometerbereich und darunter hergestellt. Die Herstellung der erforderlichen Strukturen erfolgt mit ähnlichen Fertigungsverfahren, wie sie in der Herstellung von integrierten Schaltkreisen Verwendung finden. Wichtige Fertigungsverfahren sind hier das Bulk Micromachining, Surface Micromachining und High Aspect Ratio Micromachining [2]. Bulk Micromachining ist das älteste Fertigungsverfahren. MEMS-Strukturen entstehen durch das selektive Entfernen von Silizium von sogenannten Wafern. Das Silizium wird durch Ätzen entfernt. Es wird zwischen trockenem und nassem Ätzen unterschieden. Die zu ätzenden Strukturen werden durch Fotolithografie als Masken auf den Wafer übertragen. Bei der Fotolithografie werden auf die Stelle, die nicht durch das Ätzen entfernt werden sollen, resistive Schichten aufgebracht. Surface Micromachining ist im Gegensatz zum Bulk Micromachining ein Prozess bei dem Material aufgetragen wird. Zum Einen wird Stützmaterial aufgetragen,

was später entfernt wird. Zum Anderen wird Material aufgetragen, welches im Endzustand die MEMS-Strukturen darstellt. Das Stützmaterial wird wieder weggeätzt, womit nur die MEMS-Strukturen zurückbleiben. Es sind feinere freitragenden Strukturen als beim Bulk Micromachining möglich. Das gewählte Material für die MEMS-Strukturen muss entsprechend des späteren Anwendungsbereiches gewählt werden, sodass den auftretenden Belastungen standgehalten wird. High Aspect Ratio Micromachining fasst mehrere Fertigungsverfahren zusammen, die es ermöglichen Strukturen mit einer größeren Vertikalität herzustellen [2] [16] [17].
Eine MEMS-Struktur eines Beschleunigungssensors zeigt Abbildung 2.3. Es sind deutlich die Kämme zu erkennen, die die differenziellen Kondensatoren darstellen. Durch eine äußere Beschleunigung erfährt die MEMS-Struktur eine Verschiebung , die zu einer Veränderung der Kapazität in den Kondensatoren führt. Die Verwendung von mehreren Kondensatoren, in einer Beschleunigungsrichtung verbessern die Messgenauigkeit. Durch die Verwendung der MEMS ist es möglich kleinere, für die Verwendung auf Platine nutzbare, Beschleunigungssensoren zu bauen.

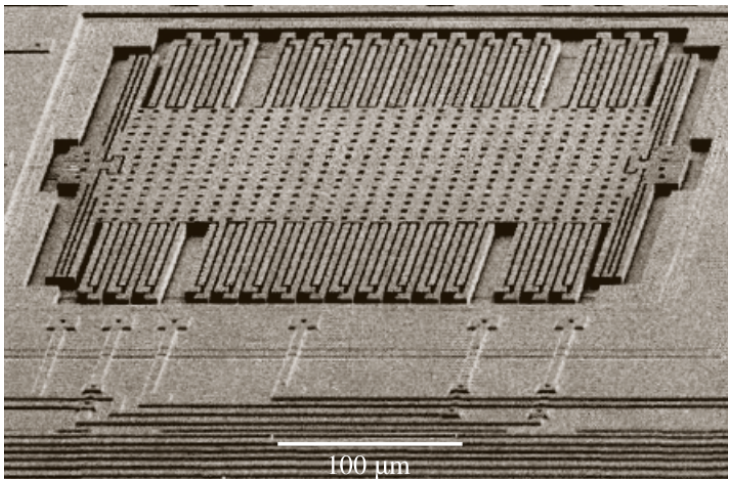


Abbildung 2.3: MEMS-Beschleunigungssensor unter dem Mikroskop [2, S. 421]

2.2 Controller Area Network

Der CAN-Bus ist ein standardisiertes serielles Feldbussystem. Es wurde bei der Robert Bosch GmbH in den 1980er Jahren entwickelt. Weite Verbreitung und Anwendung findet es bis heute in der Automobilindustrie. Für diese Anwendung zeichnet es sich durch seine hohe Zuverlässigkeit, geringe Fehlerwahrscheinlichkeit und Robustheit aus. Außerdem ist es echtzeitfähig und einfach erweiterbar. Neben dem Einsatz in der Automobilindustrie wird es in der Automatisierungstechnik aus den gleichen Gründen verwendet.
Abbildung 2.4 zeigt die Spannungen U_{CANH} und U_{CANL} der beiden differenziellen Signalleitungen CANH und CANL. Im Leerlauf beträgt die Spannung für beide Signalleitungen $U_{CANH} = U_{CANL} = 2,5\ V$. Die Differenz der beiden Signalleitungen geht in diesem Fall gegen Null. Dieser Buszustand stellt den rezessiven Buszustand dar, welcher einer logischen Eins entspricht. Eine logische Null ist der dominante Zustand und wird mit einer Differenz in der Spannung zwischen U_{CANH} und U_{CANL} von ($\approx 2\ V$) dargestellt.

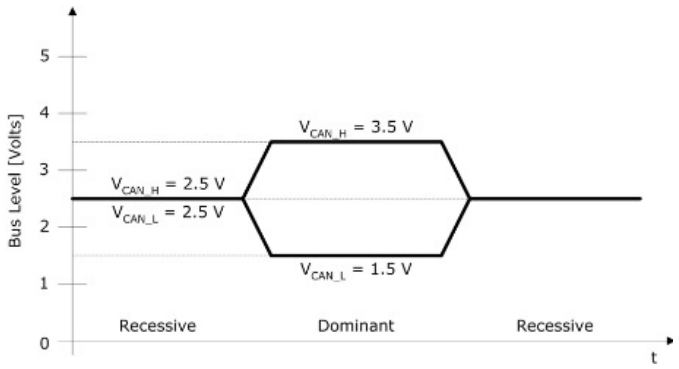


Abbildung 2.4: Logische Bitfolge (101) auf dem CAN-Bus mit den differenziellen Signalleitungen U_{CANH} und U_{CANL} [34]

Das gleiche Spannungsniveau für alle Teilnehmer in Abbildung 2.5 wird über die Verbindung der beiden Signalleitungen CANH und CANL über eine Terminierung von 120 Ω erreicht. Die Dominanz der Null wird durch die Ausführung der CAN-Transceiver als offene Kollektoren erreicht. In einem dominanten Zustand werden die beiden Signalleitung aktiv auf das Spannungsniveau der Versorgungsspannung für CANH und Masse für CANL getrieben. Die Spannung von CANH erhöht sich, während sich die Spannung von CANL verringert. Es ist ein Spannungsunterschied zwischen den beiden Leitungen messbar der als Null interpretiert wird. Ein Teilnehmer der eine Null versenden will, setzt sich durch diese Beschaltung immer durch [29] [3].

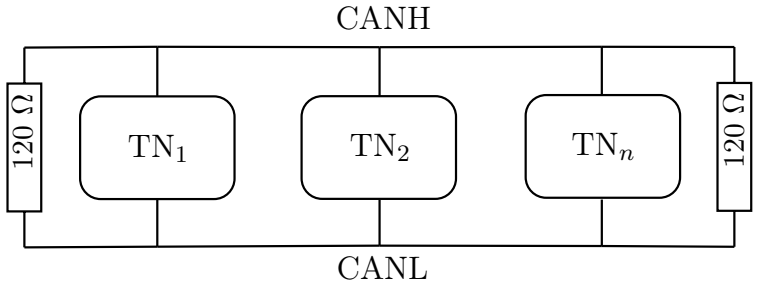


Abbildung 2.5: Schematische CAN-Bus Topologie mit n Teilnehmern und den Abschlusswiderständen

Eine CAN-Nachricht für die Übertragung von Daten setzt sich immer aus den sieben Feldern in Tabelle 2.1 zusammen. In dem cyclic redundancy check (CRC) Feld ist die 15 Bit lange Prüfsumme hinterlegt. Anhand dieser Prüfsumme kann ein Empfänger den Nachrichteninhalt überprüfen. Für die Identifier können 11 oder 29 Bit verwendet werden. In jedem Fall ist eine höher priorisierte Nachricht mit einem niedrigeren Identifier versehen. Die Adressierung ist dabei inhaltsbezogen, es wird nicht explizit ein Empfänger der Nachtricht bestimmt. Alle Teilnehmer empfangenen alle Nachrichten und filtern die für sie wichtige Inhalte anhand des Identifiers heraus.

Tabelle 2.1: Felder in einer CAN-Nachricht

Feldname	Beschreibung	Länge in Bit
Start of Frame	Start der Übertragung	1
Arbitration Field (Standard)	CAN-Identifier	11 + 1
Arbitration Field (Erweitert)	Standard CAN-Identifier mit zusätzlich erweiterndem Identifier	11 + 18 + 3
Control Field	Information über die Anzahl der Datenbytes	6
Data Field	0 - 8 Bytes mit Daten die übertragen werden sollen	0 - 64 Bit
CRC Field	Feld mit Sequenz des Cyclic redundancy check	15
ACK Field	Empfänger bestätigt Empfang mit Senden eines Bits	3
End of Frame	Sieben rezessive Bits signalisieren Ender einer Übertragung	7
Intermission Field	Drei rezessive Bits zwischen CAN-Nachrichten	3

Zusätzlich zu den angegebenen Längen kann sich die Nachricht durch Bit Stuffing verlängern. Bit Stuffing tritt ein, wenn der Sender fünfmal hintereinader ein Bit mit dem selben logischen Zustand versendet. Tritt dies auf ist das sechste Bit der gegenteilige logische Zustand zu den vorherigen fünf gleichen Bits. Bit Stuffing wird nur auf die Felder Start of Frame, Arbitration, Control, Data und CRC Field angewendet. Die Felder haben insgesamt eine Länge von 98 Bit. Sind alle Bits eine logische Eins, l müssten 19 Stuffing Bits eingesetzt werden. Mit einem Standardidentifier ist die längst mögliche CAN-Nachricht folglich 130 Bits lang, für einen erweiterten Identifier 150 Bit.

Eine Übertragung einer CAN-Nachricht ist korrekt abgeschlossen, wenn während der Übertragung bis zum Ende keine Fehler auftreten. Fehler im Laufe der Übertragung werden durch Fehlernachrichten signalisiert. Eine Übertragung kann von jedem Teilnehmer zu jedem Zeitpunkt gestartet werden, sofern sich der Bus im Leerlauf (rezessiv) befindet. Sollten zwei oder mehr Teilnehmer gleichzeitig eine Übertragung starten, wird über den Identifier bestimmt, welcher Teilnehmer seine Nachricht versenden darf. Die CAN-Treiber aller Teilnehmer die gleichzeitig versenden, vergleichen den Pegel der Signalleitungen mit dem Pegel den sie auf den Bus geben wollen. Ist der Pegel gleich sendet der Teilnehmer weiter. Sollte der Pegel der Signalleitungen dominant sein, der Pegel des Teilnehmers aber rezessiv, zieht sich der Teilnehmer mit dem rezessiven Bit zurück. Konnte ein Teilnehmer seine Nachricht nicht versenden, versucht er dies im Leerlauf erneut.

Der CAN-Bus bietet mehrere Übertragungsgeschwindigkeiten beziehungsweise Baudraten. Die maximale Baudrate beträgt $f_{\mathrm{CAN}} = 1\frac{MBit}{s}$. Typische Baudraten sind $f_{\mathrm{CAN}} = 1\frac{MBit}{s}; 800\frac{kBit}{s}; 500\frac{kBit}{s}; 250\frac{kBit}{s}; 125\frac{kBit}{s}$. Eine Anpassung der Baudraten wird über die Anpassung der Länge eines Bits vorgenommen. Ein einzelnes Bit teilt sich in die folgenden kleinere Unterelemente auf:

- Synchronization Segment (SyncSeg)
- Propagation Segment (PropSeg)
- Phase Segment 1 (PhaSeg1)
- Phase Segment 2 (PhaSeg2)

Die ersten beiden Segmente SyncSeg und PropSeg dienen der Synchronisierung der Teilnehmer. In dem SyncSeg findet auch der Wechsel des logischen Zustandes des Bus statt. Das PropSeg dient dazu, die Ausbreitung des elektronischen Signals zu kompensieren. Es ist in seiner Länge einstellbar. Bei Busnetzwerken mit großen Leitungslängen sollte eine längere Zeit vorgesehen werden um die längere Ausbreitung kompensieren zu können.

Die beiden Phase Segments gleichen Phasenunterschiede in den Flanken der Teilnehmer aus. Dafür kann das PhaSeg1 verlängert und das PhaSeg2 verkürzt werden. Der Zeitpunkt für die (erste) Bestimmung des logischen Zustandes des Bus ist immer am Ende von PhaSeg1. Dies ist der Sample Point. Sollte der Busteilnehmer so konfiguriert sein, dass der logische Zustand dreimal bestimmt werden soll, folgen nach der Bestimmung am Ende von PhaSeg1 noch zwei weitere Punkte für die Bestimmung.

Außerdem definiert sind die Information Processing Time (IPT) und Synchronization Jump Width (SJW). Die IPT ist die Zeit, die ein Prozessor für die Bestimmung des logischen Zustandes benötigt. Sollten Phasenunterschiede ausgeglichen werden müssen, ist die SJW die Grenze für die Veränderung der Längen-Segmente.
Jedes der Unterelemente setzt sich aus der Zeiteinheit des Time Quantum (TQ) zusammen. Das TQ leiten sich aus dem Prozessortakt auf dem Gerät mit der CAN-Schnittstelle ab. Für die Unterelemente gilt eine feste Länge nach Tabelle 2.2.

Tabelle 2.2: Länge der Unterelemente eines Bits für CAN

Unterelement	Formelzeichen	Länge in TQ
SyncSeg	t_{sync}	1
PropSeg	t_{prop}	1 - 8
PhaSeg1	t_{ps1}	1 - 8
PhaSeg2	t_{ps2}	2 - 8

Bei der Bestimmung der Länge der einzelnen Unterelemente müssen Anforderungen eingehalten werden

- PropSeg + PhaSeg1 $\geq$ PhaSeg2
- PropSeg + PhaSeg1 $\geq t_{\text{prop}}$
- PhaSeg2 $>$ SJW
- PhaSeg2 = 2· TQ

Das Einstellen einer Baudrate wird durch Auswahl der Längen der einzelnen Segmente unter Berücksichtigungen der Anforderungen durchgeführt. Die Baudrate ergibt sich aus der nominellen Bitrate NBR [26] [27].

$$NBR = f_{\text{CAN}} = \frac{1}{t_{bit}} \tag{2.8}$$

mit

$$t_{bit} = t_{\text{sync}} + t_{\text{prop}} + t_{\text{ps1}} + t_{\text{ps2}} \tag{2.9}$$

Für eine Erhöhung der Übertragungsgeschwindigkeit hat die Robert Bosch GmbH im Jahr 2012 Controller Area Network with flexible data rate (CAN-FD) als Weiterentwicklung von CAN präsentiert [28]. Es ist möglich, dass CAN und CAN-FD Komponenten am selben Bus hängen können. Teilnehmer deren Treiber nur CAN unterstützten, können die CAN-FD-Nachrichten nicht empfangen. Andersherum ist es Teilnehmern mit CAN-FD-Treibern möglich, CAN- und CAN-FD-Nachrichten zu empfangen.

Mit CAN-FD sind auch Übertragungen mit einer Übertragungsgeschwindigkeit $f_{\text{CAN}} > 1\frac{MBit}{s}$ möglich. Eine Erhöhung der Übertragungsgeschwindigkeit ist durch ein Wechsel der Übertragungsgeschwindigkeit innerhalb einer CAN-FD-Nachricht möglich. Für eine Implementierung von CAN-FD werden die Felder einer CAN-Nachricht um Bits erweitert.

- extended data length (EDL)
- r_0, r_1
- bit rate switch (BRS)
- error state indicator (ESI)

Alle diese Bits, bis auf r_1 im Arbitration Field, befinden sich im Control Field und nutzen vorher ungenutzte Bits. Eine Unterscheidung zwischen einer CAN- und CAN-FD-Nachricht ist anhand des EDL-Bits möglich. In CAN-FD-Nachrichten ist das Bit rezessiv.

Der Wechsel der Übertragungsgeschwindigkeit wird durch ein rezessives BRS-Bit signalisiert. Ist das BRS-Bit rezessiv, wechselt die Übertragungsgeschwindigkeit an dem Sample Point des BRS-Bits. Die Übertragung der Daten in dem Data Field ist damit schneller. Der Wechsel zurück in die alte langsamere Übertragungsgeschwindigkeit passiert mit dem Sample Point des letzten Bits im CRC-Field. Neben dem Wechsel der Übertragungsgeschwindigkeit bietet CAN-FD auch die Möglichkeit die Größe der zu übertragenden Daten auf 64 Byte zu erhöhen. Eine Erhöhung der Datenmenge erfordert das Verlängern der CRC-Prüfsumme auf 17 Polynome für bis zu 16 Byte und 21 Polynome für Nachrichten mit einer Datenmenge > 16 Byte. Für die gleichen Felder wie bei CAN wird auch bei CAN-FD Bit Stuffing angewendet. Nimmt man eine maximale Länge von Datenbytes mit 64 an, wo alle Bits mit Bit Stuffing den selben logischen Zustand haben, ergibt sich

eine maximale Länge einer Nachricht von 699 Bits. Die angegebene maximale Übertragungsgeschwindigkeit für die Daten ist nicht konsistent. Geläufig angegeben sind aber $f_{\text{CAN}} = 2\frac{MBit}{s}$ und $f_{\text{CAN}} = 5\frac{MBit}{s}$ [12] [28].

Eine Weiterentwicklung von der Robert Bosch GmbH im Jahr 2020 vorgeschlagen ist CAN XL. Mit CAN XL sollen Übertragungsgeschwindigkeiten von bis zu $f_{\text{CAN}} = 10\frac{MBit}{s}$ möglich werden, bei einer erhöhten Datenmenge auf bis zu 2048 Bytes. CAN XL verfolgt auch das Konzept des Wechsel der Übertragungsgeschwindigkeit innerhalb einer Nachricht [4].

2.3 Serial Peripheral Interface

SPI ist ein im Controller-Peripheral-Prinzip aufgebauter Bus. Es gibt immer eine zentrale Recheneinheit (Controller) die die untergeordnete Recheneinheiten (Peripheral) steuert. Für die synchrone Übertragung von Daten geht von dem Controller das Taktsignal aus welches für alle Busteilnehmer gilt. Aus dem Takt des Controllers leitet sich auch die Übertragungsgeschwindigkeit ab, was zur Folge hat, dass die Übertragungsgeschwindigkeiten nicht immer einheitlich sind. Angaben zur maximalen Übertragungsgeschwindigkeit variieren, üblich sind Datenraten im mittleren zweistelligen bis niedrigen dreistelligen Megahertzbereich. Auf dem Bus kann es immer nur eine Übertragung zwischen einem Teilnehmer und dem Controller geben. Dafür muss immer ein Teilnehmer über die Chip Select-Leitung mit ein niedrigen Spannungsniveau ausgewählt werden. Eine Adressierung von einzelnen Teilnehmer entfällt durch die Chip Select-Leitung.
Eine Implementierung mit einer oder zwei Datenleitungen ist möglich. Bei einer Implementierung mit einer Datenleitung versenden der Controller und Peripheral auf der selben Leitung ihre Nachrichten. In diesem Fall ist der Bus nur halb duplexfähig. Eine Übertragung in beide Richtungen ist möglich, aber nicht gleichzeitig. Wird der SPI-Bus mit zwei Leitungen implementiert, können Controller und Peripheral gleichzeitig Daten versenden, er ist voll duplexfähig.

Für das Senden und Empfangen von Daten wird häufig ein Register verwendet. Senden und Empfangen erfolgt wechselseitig. Bei allen Teilnehmern wird der logische Zustand eines Bits in der Übertragung immer an gleichen Stellen im Takt durchgeführt. Eine Bezeichnung der Datenleitungen ist nicht immer einheitlich, typisch sind aber Bezeichnungen wie COPI (Controller Out Peripheral In), CIPO (Controller In Peripheral Out) oder Serial Data In (SDI), Serial Data Out (SDO).

Mit den beschriebenen Leitungen ergibt sich für ein einfaches Netzwerk mit zwei Teilnehmern (Controller, Peripheral) eine Beschaltung nach Abbildung 2.6 [18].

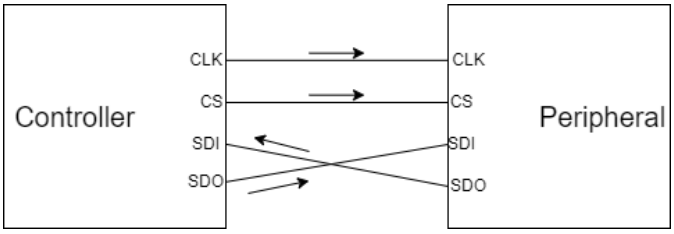


Abbildung 2.6: Beschaltung einer SPI-Leitung mit CS, CLK, SDI und SDO

Bei SPI wird zwischen vier Modi unterschieden. Diese unterscheiden sich in der Polarität des Taktsignals im Leerlauf CPOL und durch den Zeitpunkt für das Abtasten und Versenden von Daten CPHA. Bei der Polarität des Taktsignals im Leerlauf wird zwischen einem logisch hohen und niedrigen Zustand unterschieden. Das Abtasten und Versenden von Bits ist abhängig von der steigenden und fallenden Flanke des Taktsignals. Es wird unterschieden zwischen:

- Abtasten von Daten auf der fallenden Flanke und Versenden auf der steigenden Flanke
- Abtasten von Daten auf der steigenden Flanke und Versenden auf der fallenden Flanke

Aus diesen beiden Einstellungen ergeben sich insgesamt die vier unterschiedliche Kombinationen für CPOL und CPHA in Tabelle 2.3.

Tabelle 2.3: SPI-Modi und entsprechende Einstellungen [7, S. 1]

SPI-Modus	CPOL	CPHA	Polarität Taktsignal im Leerlauf	Zeitpunkt Abtasten/ Versenden
0	0	0	logisch niedrig	Abtasten von Daten auf der steigenden Flanke und Versenden auf der fallenden Flanke
1	0	1	logisch niedrig	Abtasten von Daten auf der fallenden Flanke und Versenden auf der steigenden Flanke
2	1	0	logisch hoch	Abtasten von Daten auf der steigenden Flanke und Versenden auf der fallenden Flanke
3	1	1	logisch hoch	Abtasten von Daten auf der fallenden Flanke und Versenden auf der steigenden Flanke

2.4 Hinführung zur Berechnung einer schnellen Fourier-Transformation

Grundlage einer schnellen Fourier-Transformation bilden die Fourierreihe und die (diskrete) Fourier-Transformation. Die FFT ist ein Algorithmus zur schnellen Berechnung der diskreten Darstellung einer Fourier-Transformation durch eine diskrete Fourier-Transformation.

2.4.1 Fourierreihenentwicklung

Grundgedanke der Reihenentwicklung einer Fourierreihe ist es, dass sich alle periodischen Signale durch eine Überlagerung von harmonischen Schwingungen darstellen lassen. Eine harmonische Schwingung $f(t)$ lässt sich mit einer Sinusfunktion mit der Amplitude A, der Kreisfrequenz ω und dem Phasenwinkel φ darstellen.

$$f(t) = A \cdot \sin(\omega t + \varphi) \tag{2.10}$$

Die harmonischen Schwingungen lassen sich addieren um ein anderes periodisches Signal zu bilden. Bei der Addition der harmonischen Schwingungen müssen die Kreisfrequenzen

ganze Vielfache der Kreisfrequenz der Grundfunktion ω_0 sein. Die Kreisfrequenz der Grundfunktion ist die Grundkreisfrequenz ω_0 mit der Periodendauer T.

$$\omega_0 = \frac{2\pi}{T} \tag{2.11}$$

Eine Funktion ist periodisch, wenn für jeden Zeitpunkt t in dem kontinuierlichen Signal gilt

$$f(t+T) = f(t) \tag{2.12}$$

Damit ein periodisches Signal $f(t)$ sich durch eine Fourierreihe darstellen lässt, muss sie unter anderem absolut integrierbar sein (Dirichletbedingung). Ist dies erfüllt lässt sich ein periodisches Signal durch Addition von harmonischen Schwingungen darstellen

$$f(t) = a_0 + \sum_{k=1}^{\infty} \left[a_k \cos\left(k\omega_0 t\right) + b_k \sin\left(k\omega_0 t\right)\right] \tag{2.13}$$

Die Fourierkoeffizienten a_0, a_k und b_k lassen sich aus der darzustellenden Funktion $f(t)$ berechnen. Der erste Koeffizient a_0 berechnet sich aus

$$a_0 = \frac{1}{T} \cdot \int_0^T f(t)\, dt \tag{2.14}$$

Er stellt einen Offset dar. Bei Darstellungen von elektrischen Spannungssignalen lässt sich a_0 mit einem Gleichspannungsanteil vergleichen. Für $k \geq 1$ lassen sich beide Koeffizienten a_k und b_k berechnen.

$$a_k = \frac{1}{T} \cdot \int_0^T f(t) \cdot \cos\left(k\omega_0 \cdot t\right)\, dt \tag{2.15}$$

$$b_k = \frac{1}{T} \cdot \int_0^T f(t) \cdot \sin\left(k\omega_0 \cdot t\right)\, dt \tag{2.16}$$

Die Gleichung 2.13 lässt sich mit der Eulerschen Formel in eine komplexe Darstellung überführen mit einer Änderung der unteren Grenze zu $-\infty$.

$$f(t) = \sum_{k=-\infty}^{\infty} c_k \cdot e^{jk\omega_0 t} \tag{2.17}$$

Der Koeffizient c_k errechnet sich aus

$$c_k = \frac{1}{T} \int_0^T f(t) e^{-jk\omega_0 t} \, dx \tag{2.18}$$

Eine Darstellung einer periodischen Funktion durch eine Fourierreihe führt zu einem Fourierspektrum. Die Amplituden A_k sind die Beträge aus den Fourierkoeffizienten a_k und b_k an der Stelle k [33] [11]. Über das Fourierspektrum verteilen sich die Amplituden A_k der Schwingungen k in Abbildung 2.7.

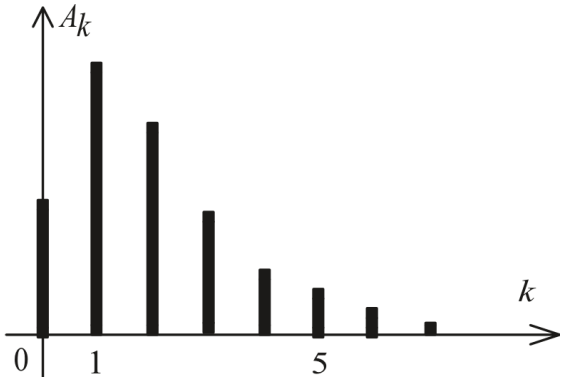


Abbildung 2.7: Fourierspektrum einer periodischen Funktion nach Fourierreihenbildung [33, S. 10]

2.4.2 Fourier-Transformation kontinuierlich und diskret

Die Darstellung einer Funktion durch eine Fourierreihe beschränkt sich auf periodische Signale. Ist ein Signal nicht mehr periodisch, wird für eine Darstellung des Spektrums die Fourier-Transformation verwendet. Wird die Periodendauer T eines Signals unendlich vergrößert, sodass gilt $T \to \infty$, geht ein periodisches Signal in ein nicht-periodisches Signal über. Die Frequenzen der harmonischen Schwingungen haben den Abstand $\omega_0 = \Delta \, \omega$. Setzt man diese Bedingung in den Vorfaktor $\frac{1}{T}$ von Gleichung 2.18 ein, folgt daraus

$$\frac{1}{T} = \frac{\omega_0}{2\pi} = \frac{\Delta\omega}{2\pi} \tag{2.19}$$

Setzt man Gleichung 2.18 und Gleichung 2.19 in Gleichung 2.17 ein, ergibt die Darstellung einer nicht-periodischen $f(t)_{T\to\infty}$. Mit $T \to \infty$ gehen die Summen in Integrale und die

diskreten Abstände, zwischen harmonischen Frequenzen, in ein kontinuierliches Signal über. Für die Darstellung eines nicht-periodischen Signals $f(t)$ ergibt sich

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \cdot e^{j\omega t} \, d\omega \tag{2.20}$$

Der Ausdruck $F(\omega)$ beschreibt das Fourierspektrum und errechnet sich aus

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot e^{-j\omega t} \, dt \tag{2.21}$$

Die Transformation eines Signals im Zeitbereich in den Frequenzbereich, der Frequenzgang, wird als Fouriertransformation bezeichnet. Im Gegensatz zum Spektrum der Fourierreihe besitzt das Spektrum der Fourier-Transformation keine diskreten Frequenzen sondern ein kontinuierliches Spektrum an Frequenzen. Die Betrachtung der Amplituden in dem Frequenzbereich wird als Amplitudengang bezeichnet. Die Berechnung eines Amplitudengangs führt zu dem Amplitudenspektrum in Abbildung 2.8, mit deutlich erkennbaren Spitzen für die Frequenzen $f = 3\ Hz$ und $f = 4\ Hz$.

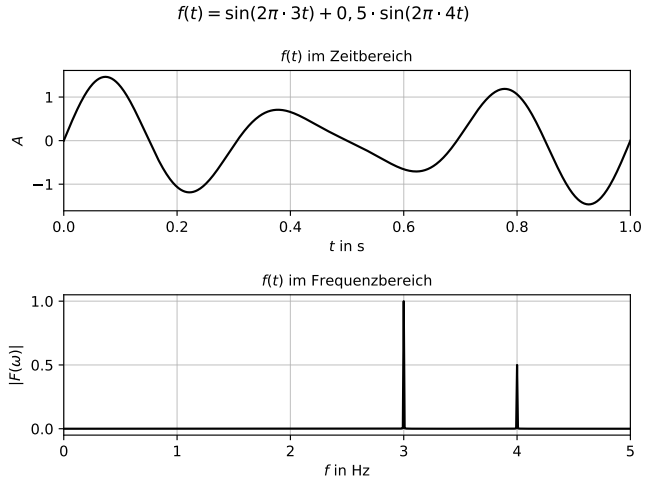


Abbildung 2.8: Beispiel des Amplitudengangs der Funktion $f(t) = \sin(2\pi \cdot 3t) + 0,5 \cdot \sin(2\pi \cdot 4t)$

Durch das Abtasten von analogen kontinuierlichen Signalen entsteht ein diskretes digitales Signal. Um auf diskrete Signale eine Fourier-Transformation anwenden zu können, muss eine diskrete Fourier-Transformation berechnet werden. Für ein Signal $x(t)$, welches mit der Periodendauer T abgetastet wird, ergibt sich für N Abtastpunkte in einem

Bereich von $0 \geq k \geq N-1$

$$x(t) = x(kT) = x[k] \tag{2.22}$$

Da das Signal nur an den Stellen kT einen Wert besitzt, kann das daraus resultierende Signal als eine Summe von k Impulsfunktionen $\delta(t)$ betrachtet werden. Die Impulsfunktionen haben immer den selben Abstand T für $0 \geq k \geq N-1$

$$x(t) = \sum_{k=0}^{N-1} x[k] \cdot \delta[t - kT] \tag{2.23}$$

Zusätzlich zu der Diskretisierung des Signals müssen die Frequenzen des Signales diskretisiert werden. Das Spektrum eines abgetasteten Signals ist periodisch mit der Periode $\frac{2\pi}{T}$.

Der Abstand γ zwischen den diskreten Frequenzen ergibt sich aus der Anzahl verteilter Abtastpunkte N im Spektrum.

$$\gamma = \frac{2\pi}{T \cdot N} \tag{2.24}$$

Alle Frequenzen die in dem diskreten Spektrum abgebildet werden können sind γ voneinander entfernt. Auf das gesamte Spektrum $\frac{2\pi}{T}$ verteilen sich n Werte im Bereich $0 \geq n \geq N-1$. Für ein diskretisiertes Frequenzspektrum $\omega[n]$ gilt somit

$$\omega[n] = n \cdot \gamma \tag{2.25}$$

$$\omega[n] = n \cdot \frac{2\pi}{T \cdot N} \tag{2.26}$$

Besitzt ein abgetastetes Signal Frequenzanteile ω_1 für die gilt

$$\omega[n] > \omega_1 > \omega[n+1] \tag{2.27}$$

wird ω_1 mit seinen Anteilen $\omega[n]$ zugeordnet. Das sogenannte Binning.

Die diskrete Darstellung einer Fourier-Transformation $X[n]$ ergibt sich durch Einsetzen von Gleichung 2.23 und Gleichung 2.26 in Gleichung 2.21 und anwenden der Ausblend-

eigenschaft der Impulsfunktion.

$$X(\omega) = \sum_{k=0}^{N-1} x[k] \cdot \int_{-\infty}^{\infty} \delta[t - kT] \cdot e^{-j\omega kT}\, dt \tag{2.28}$$

$$X(\omega[n]) = X[n] = \sum_{k=0}^{N-1} x[k] \cdot e^{-j2\pi \frac{n}{N} k} \tag{2.29}$$

Mit Hilfe einer Diskretisierung lassen sich Fourier-Transformationen auf digitalen Geräten wie Mikrocontroller (μC) oder Computern berechnen [33] [11].

2.4.3 Algorithmus zur Berechnung einer diskreten Fourier-Transformation

Das Problem mit der Verwendung von Gleichung 2.29 ist, dass für die Berechnung der diskreten Fourier-Transformation bei N Frequenzen N^2 Rechenoperationen notwendig sind [33] [11]. Um die Geschwindigkeit der Berechnung zu erhöhen, werden Algorithmen für die Berechnung der diskrete Fourier-Transformation verwendet. Die Algorithmen sind unter den Begriff der FFTs zusammengefasst. Der ursprüngliche Algorithmus geht auf Cooley-Turkey zurück und reduziert die Rechenoperationen auf $N \log(N)$ [5].

Das Grundprinzip des Algorithmus nach Cooley-Turkey ist es, ein Signal mit N Abtastpunkten in möglichst kleine Teile zu unterteilen und auf diese eine diskrete Fourier-Transformation anzuwenden. Die FFT ist folglich keine Transformation, sondern nur eine effiziente Implementierung einer diskreten Fourier-Transformation. Für Signale mit N Abtastwerten, für das gilt $N = 2^m$, kann das Signal in zwei gleiche Teile der Länge $\frac{N}{2}$ aufgeteilt werden.

Die Gleichung 2.29 lässt sich durch die Einführung des komplexen Drehfaktors W_N vereinfachen.

$$W_N = e^{-j\cdot\left(2\pi \frac{1}{N}\right)} \tag{2.30}$$

daraus ergibt sich für Gleichung 2.29

$$X[n] = \sum_{k=0}^{N-1} x[k] \cdot W_N^{n \cdot k} \tag{2.31}$$

Die Berechnung der diskreten Fourier-Transformation lässt sich in eine Berechnung in gerade $(2n)$ und ungeraden $(2n+1)$ Indizes aufteilen.

$$X[n] = \sum_{k=0}^{\frac{N}{2}-1} x[2n] \cdot W_N^{2n \cdot k} + \sum_{k=0}^{\frac{N}{2}-1} x[2n+1] \cdot W_N^{(2n+1) \cdot k} \tag{2.32}$$

Ausgedrückt mit dem komplexen Drehfaktor

$$X[n] = G(n) + W_N^k \cdot U(n) \tag{2.33}$$

Nach dem gleichen Schema lassen sich auch der gerade $G(n)$ und ungerade $U(n)$ Anteil weiter in Unterdatensätze nach geraden und ungeraden Indizes aufteilen. Diese Unterteilung lässt sich solange durchführen, bis für jeden Unterdatensatz gilt $n = 2$. Am Ende müssen demnach nur noch diskrete Fourier-Transformationen auf die Unterdatensätze mit den Längen $n = 2$ durchgeführt werden. Nach der Berechnung der diskrete Fourier-Transformation der Unterdatensätze lässt sich die gesamte diskrete Fourier-Transformation wieder zusammensetzen [33] [19].

3 Beschleunigungssensor ADcmXL3021 von Analog Devices

Die Haupthardwarekomponente ist der Beschleunigungssensor ADcmXL3021 von Analog Devices. Zusammen mit dem Mikrocontroller, der in Kapitel 5 betrachtet wird, ermöglicht er die Aufnahme von Messwerten und deren Bereitstellung über CAN. Auf die Zusammenarbeit beider Komponenten wird in Kapitel 7 eingegangen. Der Beschleunigungssensor wird in dem folgenden Kapitel näher in seinen Funktionen und Konfigurationen beleuchtet.

Der Sensor hat die Besonderheit eine FFT direkt auf dem Sensor zu berechnen. Nach der Berechnung der FFT werden die Beschleunigungen, aufgeteilt nach ihren Frequenzen, als Messwerte ausgegeben. Der Beschleunigungssensor bietet zwei Betriebsmodi für Beschleunigungen im Frequenzbereich (manual fast fourier transform (MFFT), automatic fast fourier transform (AFFT)) und zwei im Zeitbereich (manual time capture (MTC), real time streaming (RTS)). Neben der Berechnung der FFT bietet der Sensor auch die Möglichkeit Alarme zu konfigurieren. Hier wird zwischen Systemalarmen und Spektralalarmen unterschieden. Systemalarme hängen von der Temperatur und der Spannungsversorgung ab. Spektralalarme lösen bei dem Überschreiten von Beschleunigungsschwellwerte in einem bestimmten Frequenzbereich aus. Eine Konfiguration und das Abrufen von Messwerten des Sensors erfolgt über SPI durch Lesen und Schreiben von Registern [1].

3.1 Hardwareschnittstelle

Die Hardwareschnittstelle von und zum Sensor ist ein Stecker des Typs DF12(3.0)-14DS-0.5V(86). Der Stecker besitzt 16 Pins, wovon der Sensor 14 Pins für die Signale in Tabelle 3.1 verwendet. Sie gliedern sich in Eingänge, Ausgänge, Versorgung und SPI. Ein- und Ausgänge sind digital und haben verschiedenste Aufgaben um Betriebszustände zu

signalisieren oder um den Beschleunigungssensor zu steuern. Die Pins für SPI dienen der Kommunikation über diesen Bus. Die Kommunikation muss einem bestimmten Schema folgen, wie es in Abschnitt 3.5 beschrieben ist. Die letzten Pins dienen der Spannungsversorgung.

Tabelle 3.1: Pinbelegung Beschleunigungssensor ADcmXL3021

Typ	Pin	Bezeichnung	Beschreibung
Ausgang	2	ALM1	digitaler Ausgang der Alarme
	4	ALM2	digitaler Ausgang der Alarme
	5	$\overline{\text{BUSY}}$	logisch Null wenn der Sensor Messwerte aufnimmt und verarbeitet
	6	OUT_VDDM	logisch 0 Null wenn Temperaturschwellwert überschritten wird
Eingang	3	SYNC/RTS	Auslösen einer Messung über eine fallende Flanke
	7	$\overline{\text{RST}}$	Zurücksetzen des Beschleunigungssensors durch logisch Null für 130 ms
SPI	11	DIN	Dateneingang
	12	DOUT	Datenausgang
	13	SCLK	Takt
	14	$\overline{\text{CS}}$	Chip Select
Versorgung	1	GND	Masse
	8	VDD	Spannungsversorgung
	9	GND	Masse
	10	GND	Masse

3.2 Funktionsprinzip und Signalverarbeitung

Der Sensor ist als ein MEMS-Sensor ausgeführt. Um die Beschleunigung in allen drei Achsen messen zu können sind drei Beschleunigungssensoren ADXL1002 je Achse verbaut. Eine Ermittlung erfolgt nach dem in Abschnitt 2.1 beschriebenen Verfahren mit differenziellen Kondensatoren. Die Messachsen entsprechen den in Abbildung 3.1 eingezeichneten Achsen.

Die Messwerte die für jede Achse aufgenommen werden, durchlaufen verschiedene Verarbeitungsschritte bevor sie als Messwert abrufbar sind. Messwerte in den FFT-Modi durchlaufen die Schritte in Abbildung 3.2a, Messwerte im MTC-Modus die Schritte in

Abbildung 3.2b. Mit jedem Verarbeitungsschritt sind Register und Konfigurationen verbunden.
In allen Betriebsmodi ist der erste Schritt die Aufnahme von 4096 Messwerten. Die Aufnahme der Messwerte erfolgt durch das Abtasten mit einem Analog-Digital-Wandler mit einer Abtastrate von $f_s = 220\ kHz$. Die effektive Abtastrate reduziert sich durch einen Dezimierungsfilter. Der Sensor bietet ein automatisches Nullen aller Achsen. Daraus ergeben sich Offsets, die auf die Messwerte draufgerechnet werden. Sollten die Filter aktiviert sein, folgt nach dem Verrechnen mit dem Offset, die Filterung der gemessenen Beschleunigung.
Im MTC-Modus sind die Messwerte danach in Bufferregistern abrufbar. Sollten die Konditionen für einen Systemalarm erfüllt sein, werden die entsprechenden Flaggen gesetzt. Für die FFT-Betriebsmodi folgt nach der Filterung die Berechnung der FFT aus den 4096 Messwerten im Zeitbereich. Auch hier erfolgt eine Abfrage der Konditionen für die Alarme. Die Systemalarme werden durch Spektralalarme ergänzt.

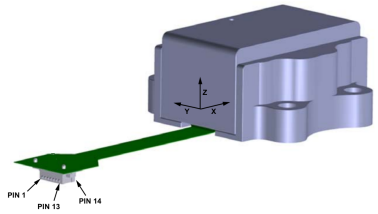
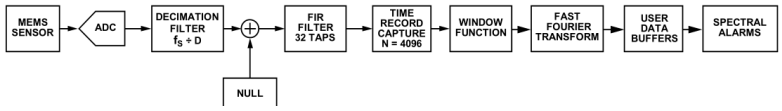
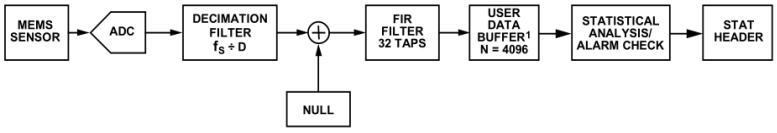


Abbildung 3.1: Achsenkreuz der Messachsen im Beschleunigungssensor ADcmXL3021 [1, S. 8]



(a) FFT



(b) MTC

Abbildung 3.2: Signalverarbeitungskette in den FFT-Betriebsmodi und MTC-Modus [1, S. 24]

3.3 Betriebsmodi und erweiternde Funktionen

Der Sensor biete insgesamt vier Betriebsmodi. Zwei von diesen Betriebsmodi geben die Beschleunigungswerte über die Zeit an (RTS und MTC). In den anderen zwei Betriebsmodi werden die gemessene Beschleunigungen ihren Frequenzanteilen über ein Spektrum ausgegeben (AFFT und MFFT).
In allen Betriebsmodi, außer dem RTS-Modus, dient der BUSY-Pin des Sensors dazu zu signalisieren, dass Messwerte abrufbar sind. Während der Aufnahme und Verarbeitung von Messwerten hat das BUSY-Signal ein niedriges Spannungsniveau. Dass Messwerte abrufbar sind, wird mit einem Wechsel von dem niedrigen auf ein hohes Spannungsniveau signalisiert. Um folglich Messwerte abrufen zu können, kann der steuerende Prozessor auf eine steigende Flanke an dem BUSY-Signal reagieren. Eine Messwertaufnahme kann entweder mit dem Setzen des Bits 11 im GLOB_CMD Register oder einer steigenden Flanke an dem SYNC-Pin gestartet werden.

Bei einer Aufnahme von Messwerten im MTC-Modus, werden 4096 Messwerte aufgenommen und bereitgestellt. In den Messwerten wird die Beschleunigung über die Zeit dargestellt. In dem MTC-Modus ist es außerdem möglich anstatt von Beschleunigungen die Geschwindigkeit als Messwert auszugeben. Der RTS-Modus bietet die Möglichkeit Beschleunigungswerte in Echtzeit über SPI auszugeben. In diesem Modus werden alle digitalen Signalverarbeitungsschritte und die Alarme umgangen. Eine Verarbeitung der Daten im RTS-Modus ist nicht möglich. Deswegen wird dieser nicht weiter betrachtet und sei nur erwähnt. Die Taktrate des SPI-Bus ist mit $f_{\text{SPI}} = 8\ MHz$ unter der geforderten Taktrate von mindestens $f_{\text{SPI}} = 12,5\ MHz$.

Befindet sich der Sensor im AFFT- oder MFFT-Modus reduziert sich die Anzahl von Messwerten die über die Buffer Register abgerufen werden können auf 2048. Im AFFT-Modus werden die Messwerte automatisch in einer festen Periodendauer aufgenommen. Der Sensor bestimmt selbstständig die abgelaufene Zeit und nach Ablauf der Periode werden selbstständig neue Messwerte aufgenommen und verarbeitet. Ein externes Starten ist nicht nötig.

Der messbare Frequenzbereich und die Binweite *bw* werden durch die einstellbaren Abtastraten in Tabelle 3.2 bestimmt. Die Binweite ergibt sich aus der Aufteilung des messbaren Frequenzspektrums auf 2048 Bins. Der messbare Frequenzbereich wird durch ein unteres $f_{\text{cutoff, low}}$ und oberes Frequenzlimit $f_{\text{cutoff, high}}$ begrenzt Alle Frequenzen im Bereich $f_{\text{cutoff, low}} \geq f \geq f_{\text{cutoff, high}}$ sind zuverlässig messbar. Theoretisch ist das untere

Frequenzlimit 0 Hz, jedoch ist der erste Bin einer Messung meistens nicht plausibel in den gemessenen Beschleunigungen. Aus dieser Beobachtung ergibt sich für das untere Frequenzlimit

$$f_{\text{cutoff, low}} = bw \tag{3.1}$$

Die maximal messbare Frequenz $f_{\text{cutoff, high}}$ ergibt sich nach Nyquist-Shannon aus der Abtastrate f_s

$$f_{\text{cutoff, high}} = \frac{f_s}{2} \tag{3.2}$$

Tabelle 3.2: Einstellbare Abtastraten mit daraus folgenden Binweiten für den Beschleunigungssensor

Registerwert für SR_n	Abtastrate in Hz	bw in Hz
0 (Default)	220k	53,8
1	110k	26,9
2	55k	13,4
3	27,5k	6,71
4	13,75k	3,35
5	6875	1,68
6	3438,5	0,839
7	1718,75	0,419617

3.4 Konfiguration

Um mit dem ADcmXL3021 Messwerte aufnehmen zu können müssen verschiedene Einstellungen in Registern getroffen werden. Die Einstellungen betreffen hauptsächlich die Abtastrate, die Periodendauer für den AFFT-Modus und die Mittelwertsbildung. Die wichtigsten Register für die Konfiguration des Sensors sind in Tabelle A.3 aufgelistet.
In erster Linie muss der Aufnahmemodus ausgewählt (AFFT, MFFT, MTC, RTS) werden. Wird der MTC-Modus ausgewählt, beschränken sich die Konfigurationsmöglichkeiten auf die Abtastrate. In den FFT-Betriebsmodi ist es möglich mehrere FFTs hintereinander aufzunehmen und den Mittelwert über diese zu bilden. Im AFFT-Modus wird das Aufnehmen von Beschleunigungen und Berechnen der FFT periodisch in einer voreingestellten Periodendauer durchgeführt. Als Perioden lassen sich Sekunden, Minuten

und Stunden mit einem Faktor von 1-255 auswählen.
Der Beschleunigungssensor lässt sich immer in den Leerlauf mit dem Schreiben eines escape codes versetzen. Dafür muss das GLOB_CMD-Register mit dem Wert *0x00E8* beschrieben werden. Sollte der Beschleunigungssensor gerade eine Messung durchführen wird diese abgebrochen.
Mit dem Dezimierungsfilter wird im MTC-, AFFT- und MFFT-Modus die Aufnahmedauer beeinflusst. Wird eine kleinere Abtastrate gewählt, dann können über einen längeren Zeitraum Messwerte aufgenommen werden. In den FFT-Modi lässt sich die Aufnahmedauer zusätzlich durch die Erhöhung der Anzahl der FFTs über die ein Mittelwert berechnet werden soll $n_{\overline{FFT}}$ erhöhen. Eine Verarbeitung der Messwerte dauert und kann zum Teil deutlich länger sein als die Aufnahmedauer. Eine Übersicht der Aufnahmedauer in den FFT-Modi t_{FFT} ist in Abbildung A.1 ablesbar, mit genauen Zahlenwerten in Tabelle A.6 bis Tabelle A.13. Die Aufnahmedauer im MTC-Modus t_{MTC} ist in Tabelle A.2 ersichtlich.

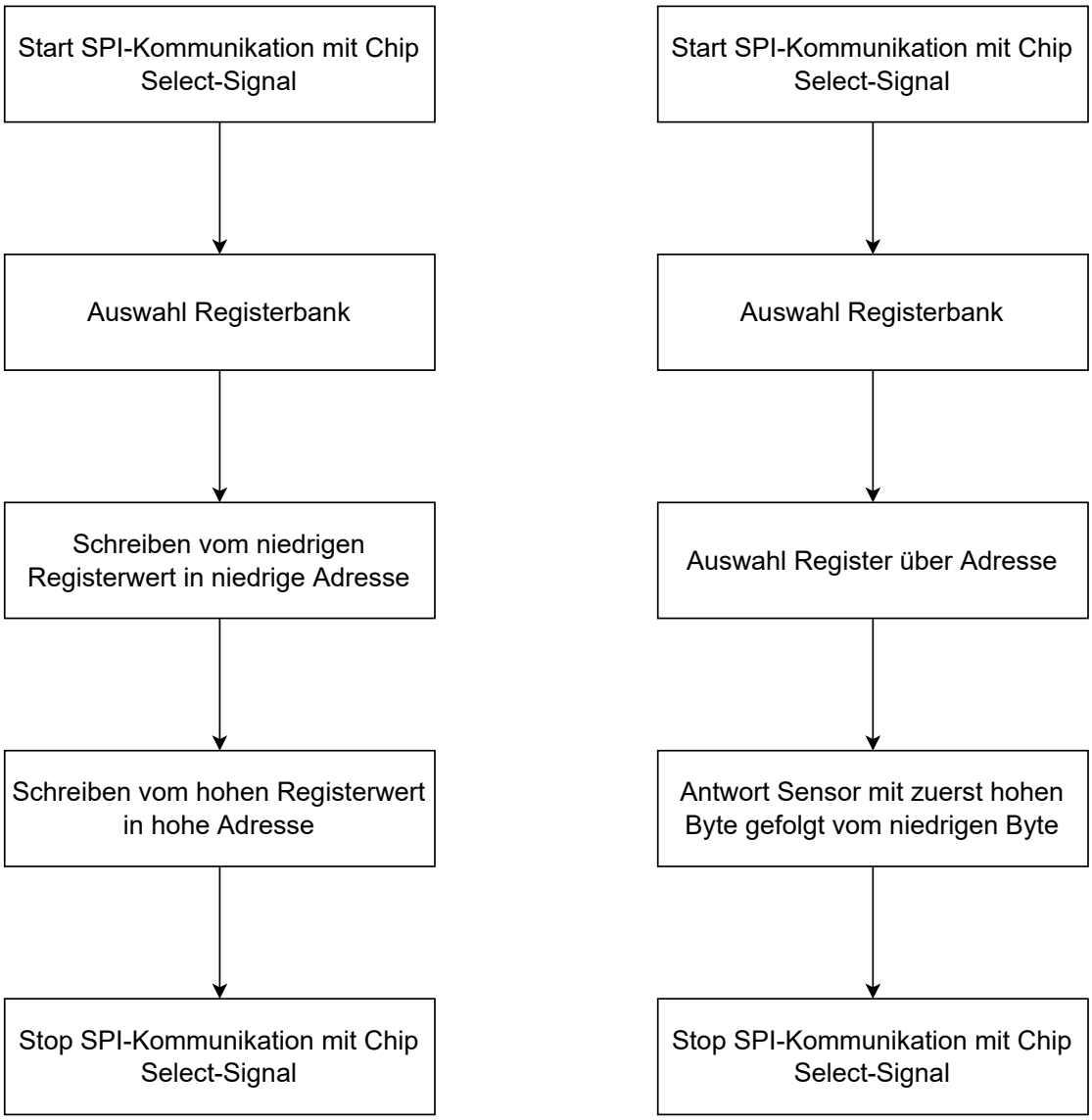
3.5 SPI-Kommunikation mit dem Beschleunigungssensor

Für die SPI-Kommunikation muss eine festgelegte Reihenfolge an Schritten beachtet werden. Werden diese Schritte in ihrem Ablauf nicht eingehalten, ist das Lesen und Schreiben von Registern nicht möglich.

Das Schreiben von Register erfolgt nach den Schritten in Abbildung 3.3a. Bei dem Schreiben an einer bestimmten Adresse, wird in der zwei Byte langen SPI-Nachricht, zuerst die Adresse und dann der Registerwert an der Adresse übertragen. Einem Register mit einem zwei Byte großen Registerwert, werden immer zwei ein Byte große Adressen zugeordnet. An einer Registeradresse kann nur ein ein Byte langer Wert gespeichert werden, dies bedingt die Aufteilung nach hohen und niedrigem Byte.

Die Schritte für das Lesen ändern sich geringfügig gegenüber dem Schreiben, wie es in Abbildung 3.3b ersichtlich ist. Die Schritte Zwei bis Vier beim Lesen und Schreiben beinhalten eine SPI-Kommunikation mit einer Länge von zwei Byte. Sollte die zu versendende Nachricht kürzer als zwei Byte sein, müssen trotzdem die kompletten zwei Byte versendet werden. Der Inhalt der verbleibenden Bits ist irrelevant. Zwischen jeder SPI-Kommunikation mit dem Sensor muss eine Pause von $t_{stall} = 16\ \mu s$ liegen.

Für die Auswahl der Registerbank und Adresse wird mit dem Setzen oder Nichtsetzen des höchsten Bits eingestellt, ob ein Register beschrieben oder gelesen werden soll. Die Auswahl der Registerbank kann übersprungen werden, wenn sich die Registerbank nicht verändert hat, da die zuletzt eingestellte Registerbank gespeichert wird.



(a) Schreiben (b) Lesen

Abbildung 3.3: Ablaufdiagramme SPI-Kommunikation zum Schreiben und Lesen von Register auf ADcmXL3021 Beschleunigungssensor

3.6 Verarbeitungskette für den Erhalt von Beschleunigungsmesswerten

Das Abrufen von Messwerten erfolgt nach den Schritten in Abbildung 3.4. Messwerte für die x-, y- und z-Achse sind im MTC-, AFFT- und MFFT-Modus über Bufferregister (X_BUF, Z_BUF, Y_BUF) abrufbar. Mit einer steigenden Flanke am BUSY-Signal kann mit dem Auslesen der Bufferregister begonnen werden. Bevor die Messwerte ausgelesen werden können muss das BUF_PNTR-Register mit dem Hexadezimalwert *0x0000* beschrieben werden. Dies führt dazu, dass der 0-te Wert der Messreihe in die Bufferregister der Achsen geladen wird. Es ist auch möglich andere Messwerte in der Reihe in das Bufferregister zu laden. Dafür muss der entsprechende Wert in Hexadezimal in das BUF_PNTR-Register geschrieben werden. Wenn ein Wert aus einem Bufferregister gelesen wird, wird in alle Bufferregister der nächste Wert automatisch geladen. Nach einer Übertragung aller Messwerte (4096 MTC, 2048 AFFT/ MFFT) können die Messwerte weiter verarbeitet werden. Die Umrechnung der Hexadezimalwerte unterscheidet zwischen MTC-Modus und den FFT-Modi. Im MTC-Modus wird zwischen einer Umrechnung von Beschleunigungs- und Geschwindigkeitsmesswerten unterschieden. Die Umrechungsgleichungen können Abschnitt A.2 entnommen werden.

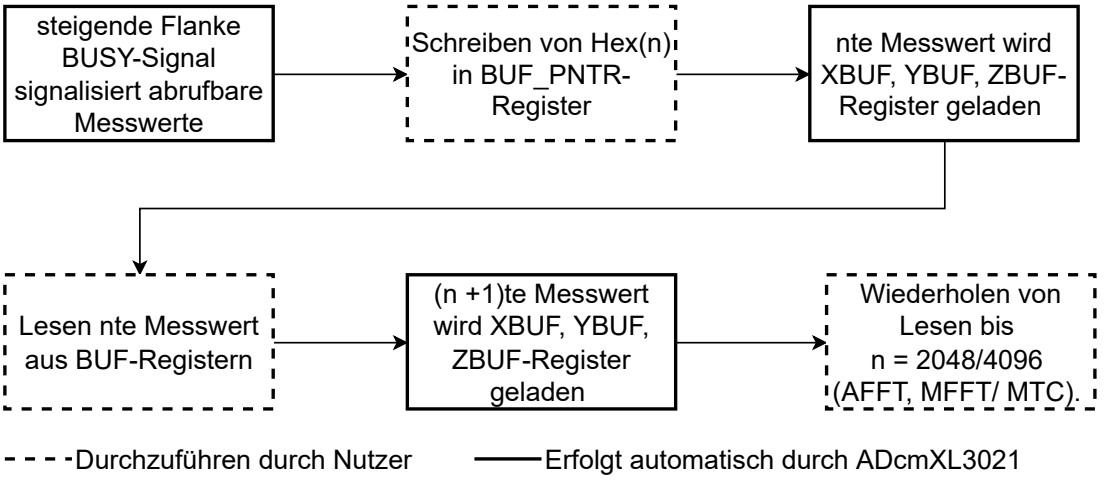


Abbildung 3.4: Schematischer Ablauf zum Abrufen von Messwerten

3.7 Filter

Der Beschleunigungssensor besitzt eingebaute finite impulse response (FIR)-Filter, die sich über Register konfigurieren lassen. Insgesamt können sechs Filter über Koeffizienten mit ihren Grenzfrequenzen f_g im FILT_CTRL-Register aktiviert werden. Die Koeffizienten mit ihren Registeradressen sind in den Filterbänken A - F hinterlegt. Für einen aktivierten Filter lässt sich sich einer der sechs Filter mit den Koeffizienten aus den Filterbänken A - F auswählen.

Mit den Standardwerten der Register sind die Filter als Hochpass und Tiefpass mit $f_g = 1\ kHz; f_g = 5\ kHz; f_g = 10\ kHz$ konfiguriert. Eine Veränderung der Grenzfrequenzen ist durch Verändern der Koeffizienten möglich. Der Wert eines Koeffizienten ist immer 16 Bit und als Zweierkomplement codiert. Wenn der Beschleunigungssensor mit den Standardwerten konfiguriert ist, ist kein Filter aktiv.

3.8 Alarmfunktionen

Der Beschleunigungssensor bietet Alarmfunktionen, die sich konfigurieren lassen und deren Status abgefragt werden kann. Bei den Alarmen wird zwischen Systemalarmen und Spektralalarmen unterschieden. Die Systemalarme beziehen sich auf die Spannungsversorgung oder die Temperatur des internen Temperatursensors. Spektralalarme werden ausgelöst, wenn in einem festgelegten Frequenzband festgelegte Beschleunigungswerte überschritten werden.
Für die Systemalarme kann entweder die Temperatur oder die Spannung als Alarmquelle verwendet werden. Ist der Alarm aktiviert und einer der beiden Quellen ausgewählt, wird die entsprechende Flagge bei dem Überschreiten eines Schwellwertes gesetzt. Für jede Achse des Beschleunigungssensors lassen sich Spektralalarme für ein definiertes Frequenzband aktivieren. Ein Spektralalarm wird ausgelöst, wenn in einem begrenzten Frequenzband Beschleunigungsschwellwerte überschritten werden. Das Frequenzband wird durch ein oberes und unteres Frequenzlimit beschränkt. Innerhalb dieses Frequenzbandes können zwei Beschleunigungsschwellwerte festgelegten werden. Für das Überschreiten der jeweiligen Schwellwerte werden einzeln Flaggen gesetzt. Für eine detailliere Beschreibung sei auf Abschnitt A.1 verwiesen.

3.9 Bekannte Fehler

Der vorliegende Beschleunigungssensor besitzt wiederkehrende Fehler, die beachtet werden müssen. Diese waren im gesamten Entwicklungsprozess beobachtbar und nicht durch andere Konfiguration behebbar.

Das automatische Nullen muss öfters durchgeführt werden, andernfalls ist die x-Achse nicht genullt. Darüber hinaus muss nach einem Zurücksetzen des Beschleunigungssensors das AVG_CNT-Register mit seinem Standardwert beschrieben werden. Andernfalls ist keine weitere Konfiguration des Registers möglich. Bei allen Reihen mit Messwerten ist zu beachten, dass der Messwert an den Stellen $n = 0$ meistens nicht brauchbar ist. Für die Stelle $n = 1$ tritt dies seltener auf.

4 Beschleunigungssensor ADcmXL3021 im Vergleich

Im Vergleich mit anderen Beschleunigungssensoren fällt auf, dass der ADcmXL3021 von Analog Devices in seinem Funktionsumfang einzigartig ist. Kein anderer Sensor bietet die Möglichkeit direkt eine FFT der Beschleunigungen zu berechnen. Ausgenommen sei hier der ADIS16228 auch von Analog Devices, der nicht mehr hergestellt wird. Betrachtet man Beschleunigungssensoren existieren sehr viele mit sehr unterschiedlichen Eigenschaften. In Tabelle 4.1 ist nur eine kleine Auswahl zusammengefasst, die den Eigenschaften des ADcmXL3021 ähnlich sind.

Alle betrachteten Sensoren messen die Beschleunigung in der x-,y- und z-Achse. Am Besten mit dem ADcmXL3021 lässt sich der KX134-1211 von Kionix vergleichen. Beide Sensoren sind sich in ihrer Sensitivität und im Messbereich sehr ähnlich, nicht aber in der Bandbreite. Dabei ist diese für den KX134-1211 kleiner und von Achse zu Achse unterschiedlich. Der KX134-1211 kann zwar kleinere Auflösungen für die Beschleunigungen liefern, dann stimmen die Messbereiche jedoch nicht mehr überein. Dafür ist jedoch der Beschaffungspreis um den den Faktor 24 kleiner. Ähnlich vom Preis wie der KX134-1211 ist der IIM-42652 von InvenSense. Auch dieser bietet eine vergleichbare Bandbreite wie der ADcmXL3021, dafür in einem deutlichen kleineren Messbereich mit maximal $\pm 16\ g$ mit einer besseren Auflösung von $0,488 \frac{mg}{LSB}$. Die beste Auflösung aller Sensoren bietet der ADXL357 auch von Analog Devices. Seine Auflösung befindet sich im $\frac{\mu g}{LSB}$ Bereich, während die anderen Sensoren Auflösungen mit $\frac{mg}{LSB}$ besitzen. Er bietet auch einen ähnlichen Messbereich mit $\pm 40\ g$, dafür aber eine deutlich kleinere Bandbreite mit maximal $1\ kHz$.

In Punkten der Bandbreite und dem Messbereich lässt sich der ADcmXL3021 sehr gut mit dem analogen 830M1-0200 von TE Connectivity vergleichen. Die Bandbreite ist besser im Vergleich mit $2\ kHz$ bis $15\ kHz$, bei gleichem Messbereich. Bedingt durch die Bauform als piezoelektrischer Sensor besitzt der 830M1-0200 eine deutlich bessere Noise

Density mit $0,05\frac{\mu g}{\sqrt{Hz}}$. Eine analoger Ausgang erfordert jedoch weitere Signalaufbereitung.
In einem Vergleich mit anderen Sensoren zeigt sich, dass der ADcmXL3021 mit seiner Funktion der Berechnung der FFT einzigartig ist. Vergleicht man nur die Eigenschaften der Beschleunigungsmessung, dann lassen sich ähnliche Sensoren finden. Allerdings lässt sich kein zu Hundertprozent identischer Sensor finden. Alle Sensoren bieten Vor- und Nachteile, die für einen bestimmten Anwendungszweck abgewogen werden müssen. Der ADcmXL3021 scheint aber einen Mittelweg zu beschreiten. Er bietet in den Aspekten Messbereich, Bandbreite, Noise Density und Sensitivität nicht die besten Werte, aber auch nicht die Schlechtesten. Der große Vorteil in der Bereitstellung als FFT ist es, Beschleunigungsmesswerte mit Frequenzen, die größer als die Übertragungsgeschwindigkeit sind, übermitteln zu können.

Ein direkter Vergleich ist jedoch, bedingt durch unterschiedlichen Montagearten und Preise, nicht möglich. Mit der hohen Abtastrate von $f_s = 220\ kHz$ ist es mit dem ADcmXL3021 theoretisch möglich Beschleunigungen mit deutlich höheren Frequenzen zu messen. Eine lineare Frequenzantwort wird jedoch nur in der 3 dB Bandbreite bis $f = 10\ kHz$ garantiert.

Tabelle 4.1: Vergleich von Beschleunigungssensoren in ihren Eigenschaften

Name	Herrsteller	Preis in €	Montageart	Bandbreite in Hz	Messbereich in g	Noise Density	Sensitivität
ADcmXL3021	AnalogDevices	351,95	Schrauben	10k	±50	MTC $26\frac{\mu g}{\sqrt{Hz}}$	MTC $1,907\frac{mg}{LSB}$ FFT $0,9535\frac{mg}{LSB}$
ADXL357	Analog Devices	81,45	SMD	0,977 bis 1k oder 0,0095 bis 10	±10	$75\frac{\mu g}{\sqrt{Hz}}$	$19,5\frac{\mu g}{LSB}$
					±20	kA	$39\frac{\mu g}{LSB}$
					±40	$90\frac{\mu g}{\sqrt{Hz}}$	$78\frac{\mu g}{LSB}$
IIM-42652	InvenSense	10,54	SMD	max 8,4k	±2	$70\frac{\mu g}{\sqrt{Hz}}$	$0,061\frac{mg}{LSB}$
					±4		$0,122\frac{mg}{LSB}$
					±8		$0,244\frac{mg}{LSB}$
					±16		$0,488\frac{mg}{LSB}$
830M1-0200	TE Connectivity	148,84	SMD	2 bis 15k	±25	$0,02\frac{mg}{\sqrt{Hz}}$	$50\frac{mV}{g}$
					±50	$0,05\frac{mg}{\sqrt{Hz}}$	$25\frac{mV}{g}$
KX134-1211	Kionix, Inc	15,14	SMD	8,2k (x-Achse) 8,5k (y-Achse) 5,6k (z-Achse)	±8	$300\frac{\mu g}{\sqrt{Hz}}$	$0,244\frac{mg}{LSB}$
					±16		$0,488\frac{mg}{LSB}$
					±32		$0,976\frac{mg}{LSB}$
					±64		$1,953\frac{mg}{LSB}$

5 Mikrocontroller PIC18LF2480 von Microchip Technology

Nach der Betrachtung des verwendeten Beschleunigungssensors in Kapitel 3 folgt in diesem Kapitel die Betrachtung des Mikrocontrollers. Wie beide Komponenten zusammen agieren wird in Kapitel 7 erläutert.

Als Mikrocontroller wird der PIC18LF2480 von Microchip Technology verwendet. Wichtig ist, dass der PIC18**LF**2480 gewählt wird. Dieser kann in einem größeren Spannungsbereich von $2,0\ V \leq U \leq 5,5\ V$ betrieben werden. Die Variante PIC18**F**2480 kann nur in einem Spannungsbereich von $4,2\ V \leq U \leq 5,5\ V$ betrieben werden. Mit der Verwendung der Variante PIC18**LF**2480 lassen sich der Beschleunigungssensor und μC mit der selben Spannung $U = 3,3\ V$ versorgen. Dadurch muss nur eine Leitung für die Versorgung aller Komponenten verwendet werden.
Der μC wird in in der Programmiersprache C programmiert. Um genaue Einstellungen vornehmen zu können, werden Register auf dem μC konfiguriert. Dafür stellt der Hersteller Microchip Technology die Programmierumgebung MPLAB bereit. In dieser Programmierumgebung kann die Bibliothek, die den μC unterstützt, inkludiert werden. Ist dies der Fall, ermöglichen Makros den Zugriff auf Ein- und Ausgänge und Register. Damit der μC programmiert werden kann, wird eine Schnittstelle zwischen Computer und μC benötigt. Dies ist das PICKit 4.

Auf die 28 Pins des μC verteilen sich die Spannungsversorgung und die Ein- und Ausgänge. Die Ein- und Ausgänge des μC sind Tri-State Pins. Das heißt, dass jeder Pin für die Verwendung in einen definierten Zustand versetzt werden muss. Sie sind meistens mit mehreren Funktionen verbunden. Die genaue Funktion eines Pins wird durch die Programmierung festgelegt. Für wenige Pins ist nur eine Programmierung in der Funktion als digitaler beziehungsweise analoger Ein- oder Ausgang möglich. Die meisten Pins übernehmen Funktionen für die vier verfügbaren Bussysteme CAN, SPI, I^2C und UART.

Jeder der Bussysteme, hat ein Modul auf dem μC, welches sich mit den gewünschten Einstellungen für den jeweiligen Bus programmieren lässt.

Neben den Ein- und Ausgängen für die Bussysteme bietet der μC auch die Möglichkeit, einen externen Oszillator zu verbinden. Wenn ein externe Oszillator verwendet wird, erfolgt die Bereitstellung des Taktsignals des Prozessors, nicht mehr durch den internen Oszillator. Der Takt lässt sich durch die Verwendung eines phase locked loop (PLL) erhöhen.
Der Speicher auf dem μC gliedert sich in den Speicher für Programme Program Memory, einen SRAM-Speicher im laufenden Betrieb und einen EEPROM-Langzeitspeicher auf. In dem Program Memory stehen dem Programmierer 16000 Bytes für Programme zur Verfügung. Für den Programmierer als Speicher nutzbar sind 768 Bytes für den SRAM und 256 Bytes für den EEPROM. Werte die in dem SRAM gespeichert sind, werden gelöscht sobald die Spannungsversorgung unterbrochen ist.
Für einige digitale Ein- und Ausgänge, sowie für alle Bussysteme sind Interrupts konfigurierbar. Bei den Interrupts der digitalen Pins ist dies nur bei ausgewählten Pins möglich. Hier kann auf eine steigende oder fallende Flanke reagiert werden. Bei den Bussystemen sind mehr Interrupts verfügbar, die durch Fehler, empfangene oder versendete Nachrichten aktiviert werden [21].

6 Anforderungsentwicklung

Die meisten Anforderungen ergeben sich aus dem vorgesehenen Einsatzbereich des Beschleunigungssensors als Sensorelement in einem Datenloggersystem. Innerhalb dieses Datenloggerssystems soll der Beschleunigungssensor Messwerte der Beschleunigung, aufgelöst über ein Frequenzspektrum, in regelmäßigen Abständen bereitstellen. Alle Anforderungen werden im Gespräch mit dem Auftraggeber entwickelt und schriftlich in Tabelle 6.1, Tabelle 6.2 und Tabelle 6.3 festgehalten. Eine Anforderungen beinhaltet immer eine konkrete sachliche Ausführung der Anforderung und gegebenenfalls einen Wert mit Einheit. Unterschieden wird bei den Anforderungen zwischen Wunsch (W) und Pflicht (P). Unter Pflicht sind die Anforderungen zusammengefasst, die für eine erfolgreiche Umsetzung zwingend erforderlich sind. Wunschanforderungen sind „nice-to-have“ und erweitern den Beschleunigungssensor sinnvoll. Die Anforderungen gliedern sich in die Kategorien *Beschleunigungssensor*, *Gehäuse* und $\boldsymbol{\mu}C$. Die Kategorie *Beschleunigungssensor* bezieht sich auf das Aufnehmen von Messwerten und die Konfiguration des Beschleunigungssensors. Unter der Kategorie Gehäuse sind die mechanischen Anforderungen an das Gehäuse zusammengefasst. Die letzte Kategorie $\boldsymbol{\mu}C$ bezieht sich direkt auf den $\boldsymbol{\mu}$C. Aus den Anforderungen ergeben sich die Aufgaben, die das gesamte System, bestehend aus dem Beschleunigungssensor, $\boldsymbol{\mu}$C und CAN-Schnittstelle, erfüllen muss. Im Wesentlichen ist die Aufgabe das Aufnehmen von Beschleunigungswerten aufgelöst über die Frequenz in einem Bereich von $1\ kHz \geq f \geq 10\ kHz$. Eine entsprechende Konfiguration des Beschleunigungssensors ist notwendig. Damit die Messwerte in das Datenloggerssystem eingebunden werden können, müssen die Messwerte mithilfe des $\boldsymbol{\mu}$C über SPI abgerufen und mit Hilfe von CAN weiter versendet werden. Bei der Entwicklung sollte, wenn möglich, eine Modularität des Systems beachtet werden. Sie soll es ermöglichen das selbe Gehäuse mit CAN-Schnittstelle für andere Sensoren verwenden zu können. In dem folgenden Kapitel wird das System betrachtet, das entwickelt wird um diese Aufgabe zu erfüllen.

Tabelle 6.1: Anforderungsliste für die Kategorie μC

ID	P/ W	Anforderung	Wert
1.1	P	Kommunikation Datenlogger $\leftrightarrow$ Sensor über CAN	-
1.2	P	Konfiguration Sensor über CAN	-
1.3	W	Programmierung $\boldsymbol{\mu}$C über CAN	-
1.4	P	Darstellen des Betriebszustandes mit LEDs	
1.5	P	Anfordern und Versenden der Messwerte	$t_{\text{data}} < 1\ s$
1.6	P	Bereitstellen einer SPI Schnittstelle	
1.7	P	Bereitstellen einer CAN Schnittstelle	-
1.8	P	Integration in bestehendes Datenloggerssystem	-

Tabelle 6.2: Anforderungsliste für die Kategorie Beschleunigungssensor

ID	P/ W	Anforderung	Wert
2.1	P	mindestens messbarer Frequenzbereich	$1\ kHz \geq f \geq 10\ kHz$
2.2	P	Starten des Beschleunigungssensors mit gleicher Konfiguration	-
2.3	W	Verwendung von Bandpassfiltern	-
2.4	W	Auslesen des Temperatursensors	-
2.5	W	Kalibrieren des Sensors	-
2.6	W	Überprüfung Konfiguration Beschleunigungssensor von außen	-
2.7	P	Aufnahme von Messwerten im Frequenz- und Zeitbereich	-
2.8	P	Konfiguration des Beschleunigungssensors von außen	-
2.9	W	Konfiguration des Beschleunigungssensors aus der Ferne	-
2.10	P	Übertragen der Messwerte an den $\boldsymbol{\mu}$C	-
2.11	P	minimale Versorgungsspannung	$V_{min} = 3\ V$
2.12	P	maximale Versorgungsspannung	$V_{max} = 3,6\ V$
2.13	P	zu messende Eventlänge	$t_{event} = 100\ ms$

Tabelle 6.3: Anforderungsliste für die Kategorie Gehäuse

ID	P/ W	Anforderung	Wert
3.1	P	Vergießen der Komponenten in dem Gehäuse vorsehen	-
3.2	P	Gehäuse gegen Wasser/ Staub schützen	-
3.3	P	Steckerverbindung zwischen Beschleunigungssensor und μC sichern	-
3.4	P	Loch LEDs in Gehäusedeckel vorsehen	-
3.5	P	Befestigung an bestehenden Befestigungspunkten	-
3.6	P	Beschleunigungssensor mit Schrauben sichern	M 2.5
3.7	P	Ausrichten des Moduls in einer definierten Richtung	-
3.8	W	möglichst kleines Gehäuse	-
3.9	P	Integrieren aller Komponenten (μC, Spannungsversorgung, CAN-Schnittstelle, Beschleunigungssensor) in ein Gehäuse	-
3.10	P	CAN-Schnittstelle nach außen	-
3.11	P	Befestigungspunkte für Adapterplatte vorsehen	-
3.12	P	gute mechanische Kopplung des Gehäuses mit Befestigung	-
3.13	W	Integration in das bestehende System sollte möglichst reibungslos sein	-
3.14	W	Erweiterbarkeit mit anderen Sensoren vorsehen	-

7 Systementwicklung

Die aus den Anforderungen entwickelten Aufgaben sollen in dem folgenden Kapitel in eine Betrachtung des Systems umgewandelt werden. Als erstes wird eine Übersicht über das System gegeben. Es folgt die Betrachtung des Systems in dem geforderten Gehäuse. Als Letztes werden erste Programmabläufe entwickelt, anhand denen eine Übertragung von Messwerten und eine Konfiguration des Beschleunigungssensors möglich wird. Außerdem wird ein System entwickelt, welches es ermöglicht effizient mit dem Beschleunigungssensor kommunizieren zu können.

7.1 Systemübersicht

In einer Übersicht besteht das System aus den drei Komponenten:

- Beschleunigungssensor
- μC
- CAN-Schnittstelle

Sie bilden zusammen das System in Abbildung 7.1. Die Systemgrenze, die sich aus der Unterbringung aller Komponenten in einem Gehäuse ergibt, ist rot markiert. Innerhalb dieses Gehäuses befinden sich der μC und Beschleunigungssensor. Der Beschleunigungssensor hat als Schnittstelle zum μC den SPI-Bus. Der μC muss als Schnittstellen SPI und CAN aufweisen. Mit der CAN-Schnittstelle kann der μC Messwerte über CAN bereitstellen und als Brücke fungieren. In der Funktion als Brücke übersetzt er Einstellungen und Messwerte zwischen den verschiedenen Bussystemen. Mit dem Ziel, Messwerte aus dem Beschleunigungssensor zu erhalten, müssen über die zwei Bussysteme Messwerte und Einstellungen mit dem externen Busteilnehmer ausgetauscht werden. Eine Übertragung von Messwerten und Einstellungen ist notwendig damit der Beschleunigungssensor, ohne direkten Zugriff, aus der Ferne konfigurierbar ist.

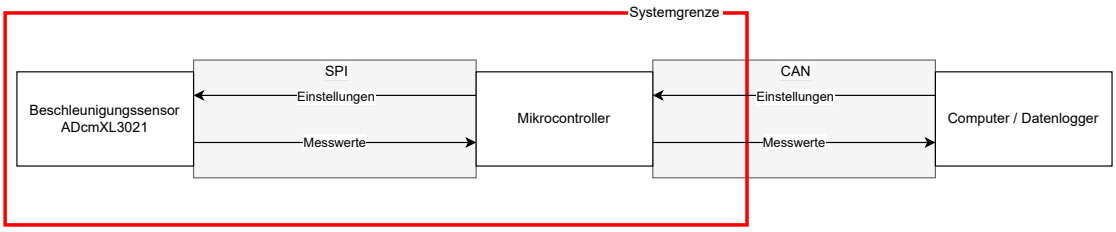


Abbildung 7.1: Übersicht des Gesamtsystems mit den drei Hauptkomponenten Beschleunigungssensor, μC und externer CAN-Teilnehmer

7.2 System innerhalb des Gehäuses

Das System innerhalb des Gehäuses soll detaillierter betrachtet werden. Hier ergeben sich aus den Anforderungen für den Betrieb weitere Komponenten und Signalflüsse. In dieser Systemsicht kommt die geforderte Modularität zum Tragen. Die benötigten Komponenten werden auf zwei Platinen aufgeteilt. Das Ziel mit der Aufteilung der Platinen ist es, die Platinen mit anderen Sensoren verwenden zu können. Dafür befinden sich auf der ersten Platine nur eine CAN-Schnittstelle, ein CAN-Transceiver und ein Spannungsregler. Der Spannungsregler stellt eine Versorgungsspannung bereit, während der CAN-Transceiver es einem μC ermöglicht am CAN-Bus teilzunehmen. Mit dem Spannungsregler und CAN-Transceiver kann beliebige Hardware auf der zweiten Platine platziert werden. CAN und die Spannungsversorgung sind auch die Schnittstellen zu der zweiten Platine, auf der sich ein μC befindet. Für den μC und Beschleunigungssensor gelten dieselben Schnittstellen für die Kommunikation wie in Abschnitt 7.1. Dazu kommen für den μC Schnittstellen, die für das Programmieren benötigt werden, sowie light emitting diode (LED) für das Anzeigen des Betriebszustandes. Damit alle Komponenten funktionieren müssen sie mit Spannung versorgt werden. Diese Spannung kann über die erste Platine für alle Komponenten bereitgestellt werden. In diesem Fall ist dies eine Spannung von $U = 3,3\ V$, da dies die von dem Beschleunigungssensor benötigte Spannung ist. Die Spannung erfordert zwischen allen Komponenten weitere Schnittstellen. Das System welches sich daraus ergibt ist in Abbildung 7.2 zusammengefasst.

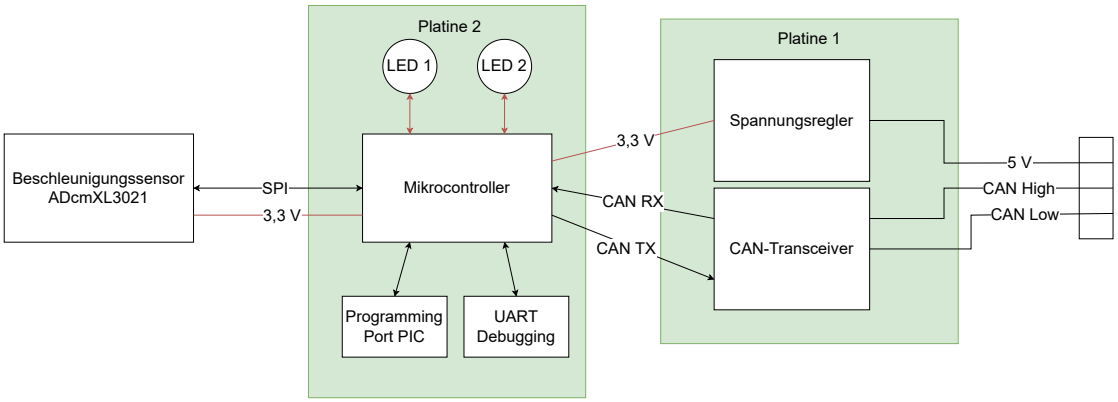


Abbildung 7.2: Übersicht der Hardwarekomponenten innerhalb des Gehäuses

7.3 Programmablauf

Aus der Funktion des μCs als Brücke kann ein Programmablauf in Abbildung 7.3 abgeleitet werden. Dafür wird auf den gewonnen Kenntnissen über die Funktionsweise von CAN, SPI, dem Beschleunigungssensor ADcmXL3021 und Mikrocontroller PIC18LF2480 aufgebaut. Zuerst muss der μC alle Schnittstellen und die verwendeten Interrupts initialisieren. Damit der μC auf Änderungen an den Schnittstellen zum Beschleunigungssensor und CAN reagieren kann bietet es sich an, die Interrupts des μCs zu verwenden. Danach befindet er sich in einer Endlosschleife, aus der er sofort auf die Interrupts durch CAN oder das BUSY-Signal reagieren kann. Tritt ein Interrupt auf, wechselt der μC vom Leerlauf in einen der beiden Programmabläufe. Das BUSY-Signal teilt dabei mit, dass neue Messwerte abrufbar sind. Interrupts für CAN werden ausgelöst, wenn eine CAN-Nachricht empfangen wird.

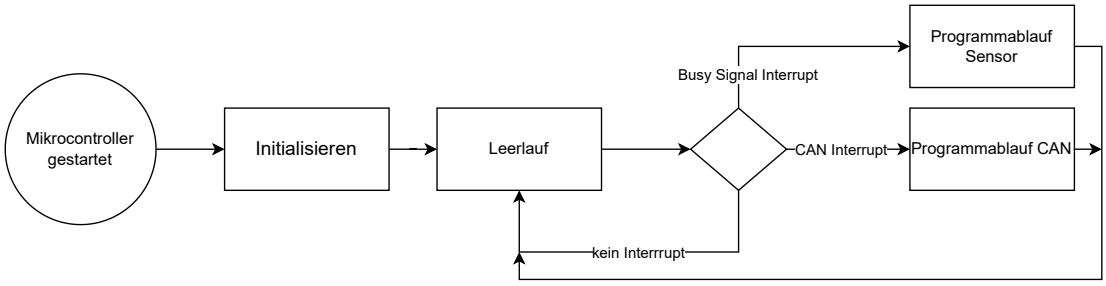


Abbildung 7.3: Programmablauf des Mikrocontrollers für die Initialisierung und das Warten auf Interrupts

Für einen Interrupt, ausgelöst durch das BUSY-Signal, ergibt sich der Programmablauf in Abbildung 7.4. Bei dem Abrufen von Messwerten muss zwischen den Betriebsmodi im Zeitbereich (MTC) oder den FFT-Modi (AFFT, MFFT) unterschieden werden. Die Abläufe sind grundsätzlich gleich, unterscheiden sich aber in der Anzahl an Messwerten die abgerufen werden müssen. Es ist notwendig alle Messwerte direkt über CAN weiter zu senden, da der Speicherplatz auf dem Mikrocontroller limitiert ist. Die Gesamtgröße der Messwerte überschreitet mit 12288/ 24576 Bytes deutlich den auf dem Mikrocontroller verfügbaren Speicher von 1024 Bytes. Wenn alle Messwerte Abgerufen und Übertragen wurden, kann der Mikrocontroller in den Leerlauf zurückkehren.
Wenn ein Interrupt durch den Erhalt von einer CAN-Nachricht auftritt, führt das zu dem Programmablauf in Abbildung 7.5. In diesem Programmablauf wird der Inhalt und der Typ der empfangenen CAN-Nachricht durch eine selbstdefinierte Kennung unterschieden. Wird eine CAN-Nachricht empfangen die die Kennung für das Schreiben eines Registers auf dem Beschleunigungssensor signalisiert, dann extrahiert der μC aus der empfangenen Nachricht das Register sowie den Registerwert und beschreibt das Register entsprechend. Außerdem werden Register auf dem Beschleunigungssensor ausgelesen, wenn dies gefordert ist. Um die aktuelle Konfiguration des Beschleunigungssensors überprüfen zu können, müssen alle dafür wichtigen Register mit einer Nachricht abgerufen werden. Gleichzeitig werden die Einstellungen in Variablen gespeichert um sie beispielsweise als Metadaten den CAN-Nachrichten mit Messwerten anhängen zu können. Für eine Änderung der Baudrate des CAN-Bus im laufenden Betrieb wird eine eigene Kennung definiert. Hier wird die gewünschte Baudrateneinstellung aus CAN-Nachricht extrahiert und interpretiert. In Abhängigkeit dieser Einstellung werden die entsprechenden Registerwerte in die dazugehörigen Register geschrieben. Mit dem beschriebenen Programmablauf ist es für den μC möglich, auf CAN-Nachrichten und neue Messwerte entsprechend zu reagieren. Wenn der Programmablauf in Code umgesetzt ist kann der μC als Brücke fungieren. Detailliertere Funktionen des Programms oder der Kennungen vereinfachen den Betrieb oder ergeben sich aus den beschriebenen Funktionen.

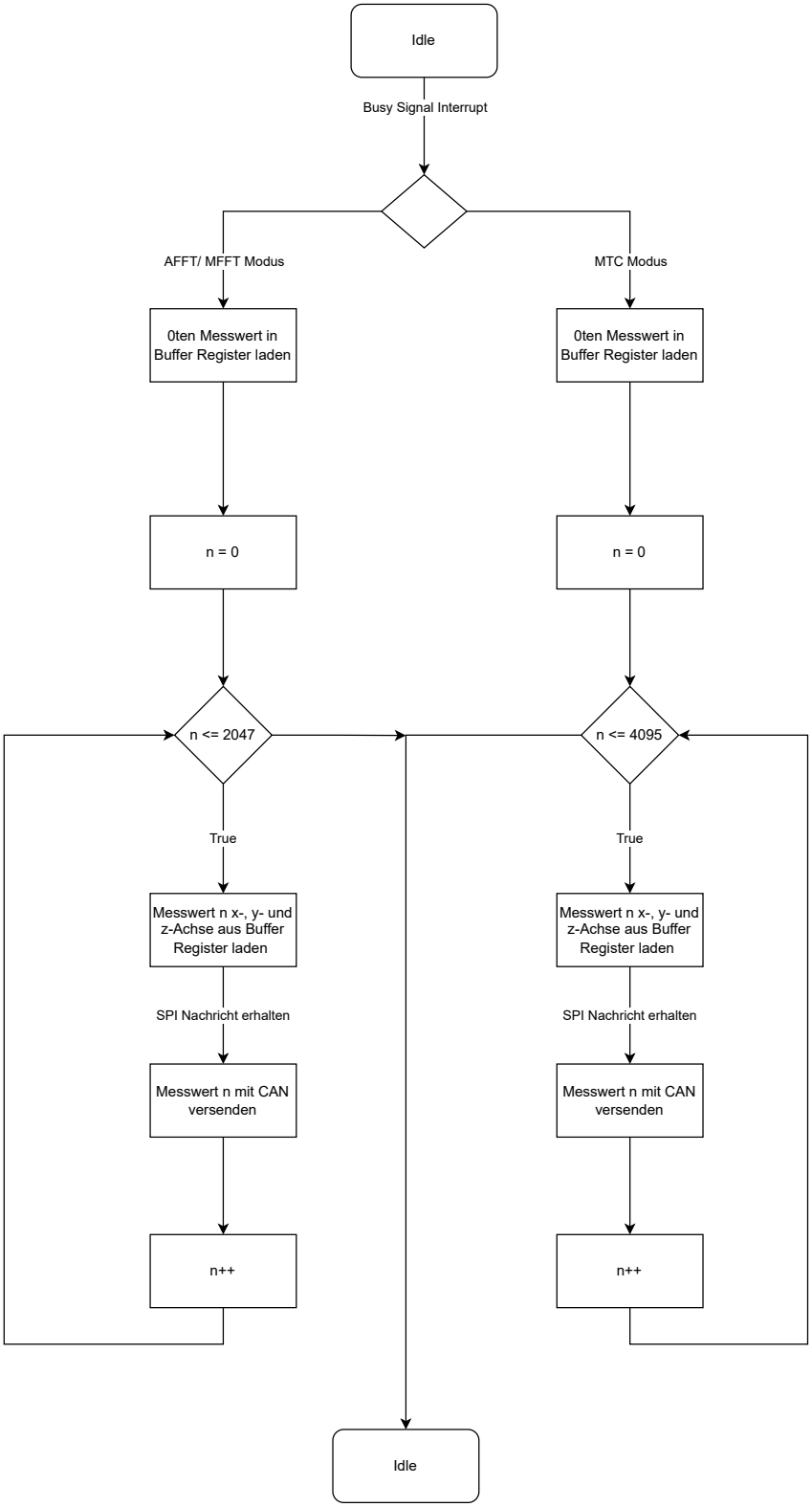


Abbildung 7.4: Programmablauf des Mikrocontrollers für neue Messwerte des Beschleunigungssensors

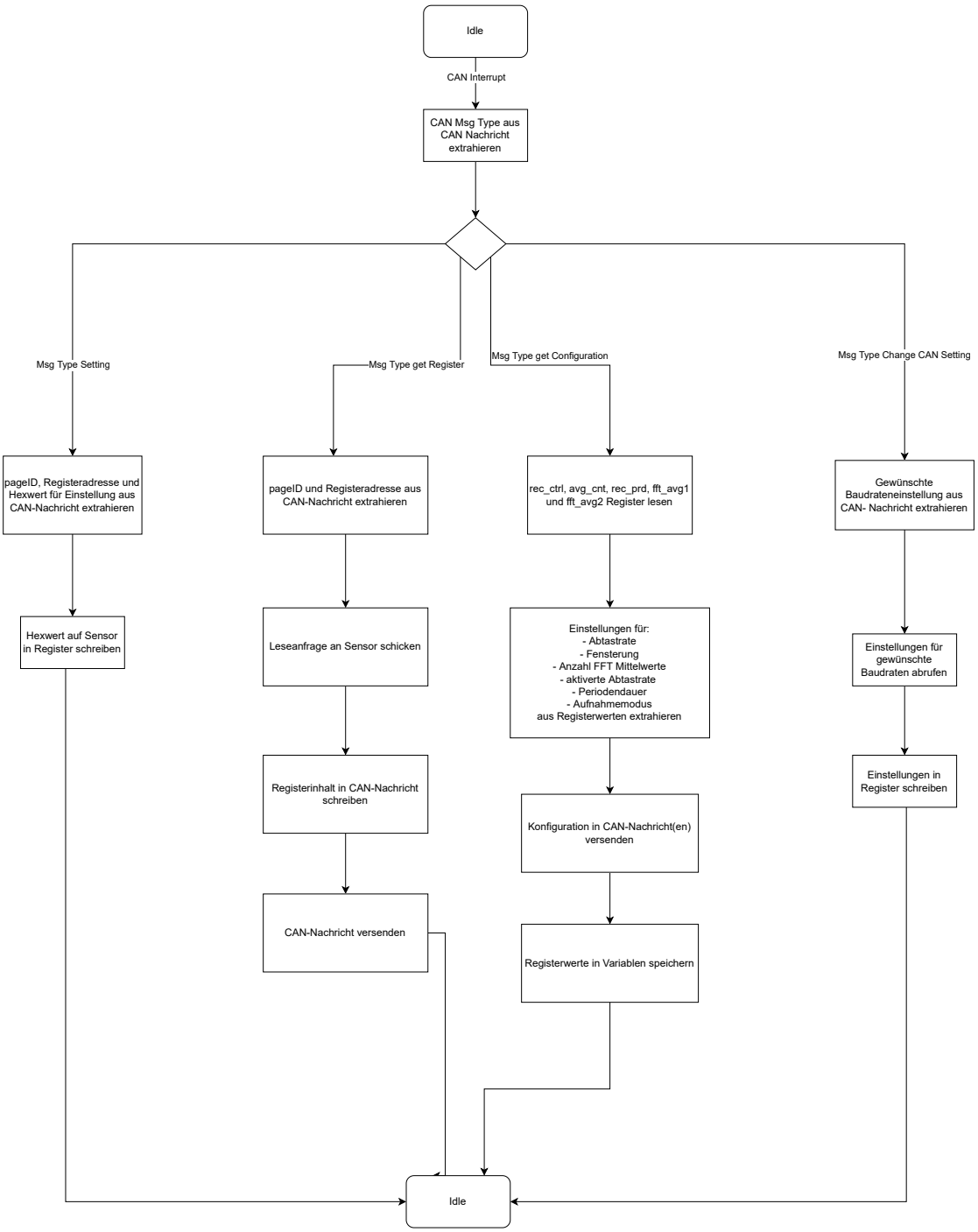


Abbildung 7.5: Programmablauf des Mikrocontrollers nach dem Erhalt von CAN-Nachrichten

7.4 Detailbetrachtung der selbstdefinierten Kennungen

Hinweis: *Um Verwechslungen zu vermeiden verweist der Begriff der Kennung ab hier, auf die selbstdefinierten Kennungen im ersten Datenbyte einer CAN-Nachricht. Die eigentliche Kennung (Identifier) einer CAN-Nachricht, die für jede Nachricht im Arbitration Field festgelegt ist, wird immer als Identifier bezeichnet.*

Auf dem μC können, in Abhängigkeit von den selbstdefinierten Kennungen, verschiedene Teile des Programms durchlaufen werden. Alle an den μC versendeten CAN-Nachrichten sind in ihrer Funktion eindeutig zuordenbar, was durch eine feste Zahl in Hex in dem ersten Byte aller Nachrichten erreicht wird. Soll beispielsweise das AVG_CNT-Register mit einem neuen Wert beschrieben werden, dann ergibt sich daraus der Nachrichteninhalt in Tabelle 7.1. Mit einem Wert von *0x01* im ersten Byte wird dem μC mitgeteilt, dass eine Nachricht für das Schreiben eines Registers folgt. Das zu schreibende Register befindet sich in der ersten Registerbank (*0x00* zweites Byte) mit der Adresse *0x3A* im dritten Byte. In diesem Fall ist der neue Wert für das Register *0x4375*. Die Länge von zwei Byte erfordert die Aufteilung auf die Datenbytes drei und vier.

Tabelle 7.1: Übersicht der selbstdefinierten Kennungen

	Datenbyte CAN-Nachricht				
	data0	data1	data2	data3	data4
Beschreibung	Kennung	Register-bank	Adresse	hohes Byte neuer Wert	niedriges Byte neuer Wert
Hexadezimalwert	*0x01*	*0x00*	*0x3A*	*0x43*	*0x75*

Das gerade aufgezeigte Beispiel ist eine von vielen Kennungen in Tabelle 7.2. Die restlichen Kennungen werden im folgenden Abschnitt beschrieben. Unterschieden wird bei den Kennungen zwischen Grundfunktionen, erweiternden Funktionen und Kennungen die der Kommunikation dienen.

Alle Kennungen haben eine Übertragungsrichtung. Manche Nachrichten, wie das Lesen von Registern, erfordern eine Nachricht von Extern an den μC mit einer Antwort des μC. Andere Nachrichten an den μC erfordern keine Antwort des μC. Die letzte Richtung ist von dem μC nach Extern, mit einer entsprechenden Verarbeitung.

Die Kennungen *0x1, 0x2, 0x4, 0x8* beziehen sich auf die Grundfunktionen aus Abbildung 7.5. Funktionen mit den Kennungen *0x3, 0x9, 0xA* erweitern die Funktionalität des Programms auf dem μC. Die restlichen Kennungen für die Kommunikation dienen der Kommunikation zwischen Extern und dem μC. Nachrichten mit Kennungen, die sich auf die Übertragung von Messwerten beziehen, dienen dazu, extern die Messwerte abspeichern zu können. Es wird zwischen Kennungen für Messwerte im Zeit- und Frequenzbereich unterschieden. Der Beginn einer Übertragung signalisiert dem Empfänger von Messwerten, dass dieser aus dem Leerlauf in einen aktiven wechseln muss um die Messwerte zu verarbeiten. Mit den Kennungen für die Konfiguration des Beschleunigungssensors wird der Beschleunigungssensor in den Leerlauf versetzt und der μC reagiert nicht mehr auf steigende Flanken am BUSY-Pin. Die Kennung *0xB* wird am Anfang der Konfiguration durch den μC versendet, sobald sich der Beschleunigungssensor im Leerlauf befindet.

Die Kennungen befinden sich bei allen Nachrichten von und zum μC in dem 0-ten Byte der CAN-Nachrichten. Sind weitere Werte für eine Nachricht notwendig, befinden sich diese immer in den nachfolgenden Bytes. Auf eine korrekte Reihenfolge ist hierbei zu achten. Für Nachrichten, die der μC empfangen und verarbeiten soll sind die Zuordnungen der Datenbytes nach Tabelle 7.3 einzuhalten. Die Registerbank und -adresse bezieht sich auf die Register auf dem Beschleunigungssensor. Ein Datenbyte einer CAN-Nachricht hat immer nur eine Größe von einem Byte, die Register des Beschleunigungssensors aber eine Größe von zwei Byte. Dies erfordert, dass die Registerinhalte bei dem Lesen und Schreiben in ein niedriges und hohes Byte aufgeteilt werden.

Die Zuteilung der Datenbytes für vom μC versendete CAN-Nachrichten kann der Tabelle 7.4 entnommen werden. Für das Abfragen von der aktuellen Konfiguration des Beschleunigungssensors sind zwei Nachrichten notwendig. Die Nachrichten, die den Start einer Übertragung von Messwerten signalisieren, beinhalten außerdem Werte aus der Konfiguration des Beschleunigungssensors, die für eine Auswertung der Messwerte notwendig sind. Bei Messwerten die im MTC-Modus aufgenommen wurden, kann außerdem unterschieden werden, ob es sich um Beschleunigungs- oder Geschwindigkeitswerte handelt. Jede Nachricht mit Messwerten beinhaltet immer die Messwerte der x-, y- und z-Achse aus dem selben Bin.

Tabelle 7.2: Übersicht der selbstdefinierten Kennungen

Richtung	Kennung	Beschreibung
		Grundfunktionen
Extern $\Rightarrow$ μC	0x1	Schreiben eines Registers auf dem Beschleunigungssensor
Extern $\Leftrightarrow$ μC	0x2	Lesen eines Registers auf dem Beschleunigungssensor
Extern $\Leftrightarrow$ μC	0x4	Rückmeldung über die aktuellen Einstellungen des Beschleunigungssensors
Extern $\Rightarrow$ μC	0x8	Verändern der CAN-Baudrate im laufenden Betrieb
		Erweiterte Funktionen
Extern $\Rightarrow$ μC	0x3	Initiiert das Aufnehmen von Messwerte im MTC-, AFFT- und MFFT-Modus
Extern $\Rightarrow$ μC	0x9	Starten des automatischen Nullen der Achsen des Beschleunigungssensors
Extern $\Rightarrow$ μC	0xA	Sensor auf Werkseinstellungen zurücksetzen
Extern $\Rightarrow$ μC	0xC	Speichern der aktuellen Registereinstellungen auf dem Beschleunigungssensor
		Kommunikation
Extern $\Rightarrow$ μC	0x6	Beginn Konfiguration Beschleunigungssensor
Extern $\Rightarrow$ μC	0x7	Stopp Konfiguration Beschleunigungssensor
Extern $\Leftarrow$ μC	0xB	Versenden nachdem escape code erfolgreich war
Extern $\Leftarrow$ μC	0x92	Signalisiert den Start der Übertragung von FFT-Messwerten
Extern $\Leftarrow$ μC	0x94	Messwerte für die x-, y- und z-Achse für Bin n ($0 \geq n \geq 2048$)
Extern $\Leftarrow$ μC	0x96	Signalisiert das Ende der Übertragung von FFT-Messwerten
Extern $\Leftarrow$ μC	0xE2	Signalisiert den Start der Übertragung von MTC-Messwerten
Extern $\Leftarrow$ μC	0xE4	Messwerte für die x-, y- und z-Achse für Bin n ($0 \geq n \geq 4096$)
Extern $\Leftarrow$ μC	0xE6	Signalisiert das Ende der Übertragung von MTC-Messwerten

Tabelle 7.3: Übersicht der Inhalte der Datenbytes bei den selbstdefinierten CAN-Nachrichtenkennungen für die Nachrichten von Python zum μC

Kennung	Datenbytes			
data0	data1	data2	data3	data4
0x1	Registerbank	Registeradresse	hohes Byte Einstellung	niedriges Byte Einstellung
0x2	Registerbank	Registeradresse	-	-
0x3		-	-	-
0x4	-	-	-	-
0x6	-	-	-	-
0x7	-	-	-	-
0x8	Baudratenauswahl	-	-	-
0x9	-	-	-	-
0xA	-	-	-	-
0xC	-	-	-	-

Tabelle 7.4: Übersicht der Inhalte der Datenbytes bei den selbstdefinierten CAN-Nachrichtenkennungen für die Nachrichten vom μC nach Python

Kennung	Datenbytes						
data0	data1	data2	data3	data4	data5	data6	data7
0x1	-	-	-	-	-	-	-
0x2	Register hohes Byte	Register niedriges Byte	-	-	-	-	-
0x3	-	-	-	-	-	-	-
0x4 Nachricht 1	Abtastrate	Fenster	Anzahl gemittelter FFTs	aktivierte Abtastrate	Periodendauer Einheit	Periodendauer Faktor	Aufnahmemodus
0x4 Nachricht 2	Beschleunigung/ Geschwindigkeit	-	-	-	-	-	-
0x6	-	-	-	-	-	-	-
0x7	-	-	-	-	-	-	-
0x8	-	-	-	-	-	-	-
0xB	-	-	-	-	-	-	-
0x92	Anzahl gemittelter FFTs	Abtastrate					
0x94	x-Achse hohes Byte	x-Achse niedriges Byte	y-Achse hohes Byte	y-Achse niedriges Byte	z-Achse hohes Byte	z-Achse niedriges Byte	
0x96	-	-					
0xE2	Abtastrate	Beschleunigung/ Geschwindigkeit	Anzahl gemittelter FFTs				
0xE4	x-Achse hohesByte	x-Achse niedriges Byte	y-Achse hohesByte	y-Achse niedriges Byte	z-Achse hohesByte	z-Achse niedriges Byte	
0xE6	-	-					

8 Platinenentwicklung

Die beiden Platinen werden in dem Programm KiCAD entwickelt [9]. Als Grundlage für die Platinen dienen Schaltpläne, die auch in KiCAD erstellt werden (Abschnitt A.5, Abschnitt A.6). Für die Komponenten, Traces und Vias sind Randbedingungen einzuhalten[1].

Auf beiden Platinen zusammen werden die Bauteile aus Tabelle 8.1 verbaut. Die erste Platine dient in erster Linie als Schnittstelle zum CAN-Bus mit zwei Rundsteckverbindungen mit Stecker und Buchse. Die Verwendung von einem Stecker und einer Buchse ermöglichen es, dass der Sensor an einer beliebigen Stelle in ein CAN-Netzwerk eingebunden werden kann. Für die Umwandlung der 5 V Spannung aus dem CAN-Kabel wird ein Spannungsregler verbaut. Dies ist vor allem nötig, da der verwendete Beschleunigungssensor nur eine Versorgungsspannung von $3,3$ V verträgt. Um weitere Busteilnehmer vor elektrostatischen Entladungen zu schützen, werden zwischen den CAN-Leitungen Schutzdioden verbaut.

Damit der μC als Teilnehmer auf dem CAN-Bus zugreifen, Nachrichten versenden und empfangen kann, ist außerdem ein CAN-Transceiver verbaut. Zwischen den beiden Platinen werden die Spannung, CAN und das Silent-Signal übertragen. Das Silent-Signal dient dazu, wenn vom μC gewünscht, das Versenden von CAN-Nachrichten vom CAN-Transceiver zu unterbinden.

Auf der zweiten Platine befinden sich in erster Linie ein μC, der mit allen anderen Komponenten verbunden wird. Er ist die zentrale Recheneinheit in dem gesamten System. Für einen höheren und stabilen Takt wird der μC mit einem externen Taktsignal aus einem CMOS Oszillator versorgt. Bei Platzierung des Oszillators ist darauf zu achten, dass der Weg zum μC möglichst kurz gehalten wird. Außerdem ist auf der zweiten Platine der Stecker für die Verbindung mit dem Beschleunigungssensor untergebracht. Mit diesem Stecker müssen alle, für die Kommunikation und den Betrieb des Beschleunigungssensors benötigten, Signalleitungen sowie Spannungsversorgung und Masse verbunden werden.

[1] `https://eu.beta-layout.com/specifications/`

Zur Darstellung des aktuellen Betriebszustandes sind LED vorgesehen. Damit der μC programmiert werden kann, sind Stecker für das Programmierinterface untergebracht.

Tabelle 8.1: Übersicht der verwendeten Hardwarekomponenten

Hersteller	Bezeichnung	Beschreibung	Position
Nexperia USA Inc.	PESD1CAN-UX	CAN-Bus ESD-Schutzdiode	Platine 1
Texas Instruments	TLV76733QWDRBRQ1	Spannungsregler von 5V auf 3,3V	Platine 1
NXP	TJA1441ATK	IC CAN-Transceiver	Platine 1
TE Connectivity	T4040014041-000	Stecker - Rundstecker-verbindung zum CAN-Bus	Platine 1
TE Connectivity	T4041017041-000	Buchse - Rundstecker-verbindung zum CAN-Bus	Platine 1
Microchip Technology	PIC18LF2480	Mikrocontroller	Platine 2
Hirose Connector	DF12NB(3.0)-14DP-0.5V(51)	Steckerverbindung Platine 2 und Beschleunigungssensor	Platine 2
ECS	2520MVLC-080-BN-TR3	8 MHz Oszillator	Platine 2
Analog Devices	ADcmXL3021	Beschleunigungssensor	Sensor

Abbildung 8.1 zeigt einen Auszug aus dem Schaltplan für die erste Platine. Neben den beiden integrierten Schaltkreisen sind die zusätzlich benötigten Widerstände und Kondensatoren zu erkennen. Die Bauteile lassen sich in Abbildung 8.2 wiederfinden. Dort sind diese jedoch auf der Platine angeordnet und durch Traces und Vias miteinander verbunden.

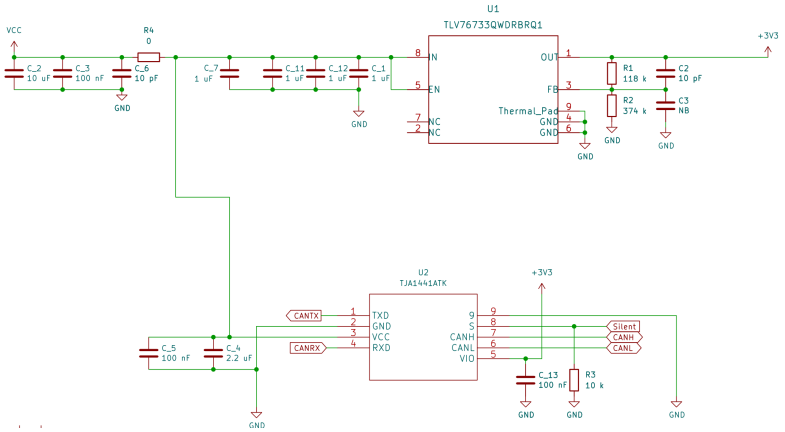


Abbildung 8.1: Auszug aus dem Schaltplan für die erste Platine mit den integrierten Schaltkreisen zum Wandeln der Versorgungsspannung oben und dem CAN-Transceiver unten

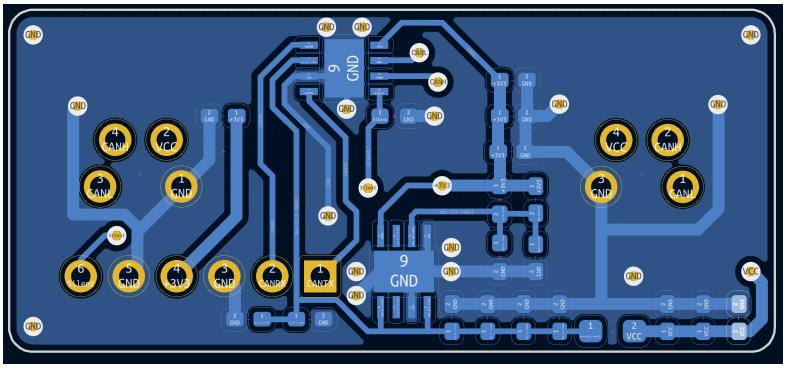


Abbildung 8.2: Platine 1 in der KiCAD Umgebung mit den integrierten Schaltkreisen zum Wandeln der Versorgungsspannung unten und dem CAN-Transceiver oben

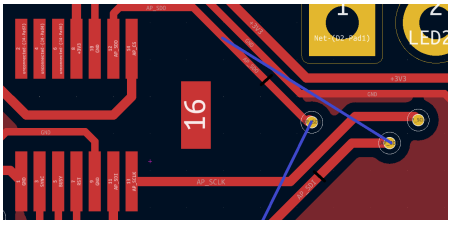
Neben den für den Betrieb benötigten integrierten Schaltkreisen müssen zusätzliche Komponenten vorgesehen werden. Für die Spannungsversorgung werden mehrere Kondensatoren für die Filterungen des Spannungssignals verbaut. Außerdem benötigen alle integrierten Schaltkreise weitere Komponenten für einen sicheren Betrieb. Die Beschaltung für den normalen Betrieb können mit genauen Angaben zu Kapazität und Widerstand den Datenblättern entnommen werden [1] [8] [21] [23] [24] [30]..

Als zentrale Recheneinheit muss der μC auf der Platine mit mehreren Eingangs- und Ausgangssignalen verbunden werden. Die Eingangs- und Ausgangssignale sind in Tabelle 8.2 zusammengefasst. Die Signale der Bussysteme beziehen sich in ihrer Richtung auf den μC.

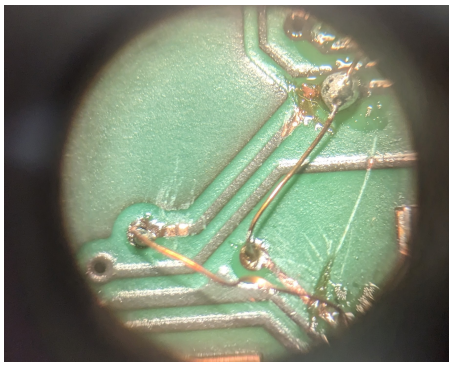
Tabelle 8.2: Übersicht der Signale von und zum Mikrocontroller

Signalname	Richtung	Beschreibung
Silent	Ausgang	Silent CAN-Transceiver
LED2-	Eingang	Minuspol LED2
AP_CS	Ausgang	Chip Select SPI
LED1-	Eingang	Minuspol LED1
LED1+	Ausgang	Pluspol LED1
AP_SCLK	Ausgang	Clock SPI
AP_SDI	Eingang	Data In SPI
AP_SDO	Ausgang	Data Out SPI
UART_TX	Ausgang	Ausgang UART
MCLR	Ausgang	MCLR PICKit4
PGD	Ausgang	PGD PICKit4
PGC	Ausgang	PGC PICKit4
PGM	Ausgang	PGM PICKit4
RST	Ausgang	Reset Beschleunigungssensor
CANRX	Eingang	Ausgang ECAN-Modul
CANTX	Ausgang	Eingang ECAN-Modul
SYNC	Ausgang	SYNC Beschleunigungssensor
BUSY	Eingang	BUSY Beschleunigungssensor

Bei der Entwicklung der Platine 2 ist nicht bedacht worden, dass die Datenleitungen von SPI vertauscht werden. Die beiden Leiterbahnen SDI und SDO werden an der in Abbildung 8.3 markierten Stelle getauscht. Für das Tauschen werden die beiden Leiterbahnen durchtrennt und einzelne Litzen als Brücken eingelötet.



(a) Stelle auf der Platine 2 für das Tauschen



(b) Getauschte Leiterbahnen auf der fertigen Platine

Abbildung 8.3: Tauschen von SDI und SDO auf Platine

9 Konstruktion des Gehäuses

Die Unterbringungen aller Komponenten erfordert ein Gehäuse für die Unterbringung des Beschleunigungssensors, den zwei Platinen und der CAN-Buchsen. Das Gehäuse wird in Creo konstruiert und im 3D-Druck Verfahren hergestellt [25].

Es teilt sich in zwei Teile auf, einen Teil mit den beiden Platine und einen Teil nur mit dem Beschleunigungssensor. Die Befestigungspunkte sind so angebracht, dass die Grundfläche möglichst klein gehalten wird. Die Befestigung des Gehäuses wird über eine Adapterplatte realisiert, wodurch es möglich ist, am Gehäuse immer dieselben Befestigungspunkte zu verwenden.

Für eine gute mechanische Kopplung wird der Beschleunigungssensor mit dem Gehäuse verschraubt. Um einen sicheren Halt von allen Schrauben zu gewährleisten, werden sogenannte Thread Inserts vorgesehen. Die Thread Inserts sind kleine Bauteile mit einem Innengewinde, aus zum Beispiel Messing, die in das Plastik des 3D-Drucks eingelassen werden können. Es werden Fasen für überschüssiges Material vorgesehen.

An dieser Stelle kommt auch die Modularität der Platinenentwicklung zum Tragen. Sollte das Gehäuse später für andere Sensoren verwendet, kann die erste Platine mit CAN-Schnittstelle weiter verwendet werden. Der Teil des Gehäuses in dem sich der Beschleunigungssensor befindet ist ausreichend groß für andere Sensoren dimensioniert.
Weiterhin muss bei dem Gehäuse darauf geachtet werden, dass es möglichst gut gegen Wasser und Staub geschützt ist. Bei der Befestigung des Sensors könnten Durchgangsbohrungen vorgesehen werden. Durch diese würden die Befestigungsschrauben geführt und mit Muttern gesichert. Um möglichst wenige Öffnungen zu haben, wird diese Befestigungsart nicht realisiert. Stattdessen ist das Gehäuse mit einem dickeren Boden konstruiert. In diesen Boden können die Thread Inserts voll eingelassen werden.

Die einzige Öffnung ist der Deckel, damit ist er auch die einzige Stelle, wo Fremdkörper eindringen können. Deswegen ist ein Überhang und ein späteres Abdichten mit Silikon vorgesehen. An der Unterseite des Deckels befindet sich außerdem ein Steg an der Stelle des Steckers zwischen Beschleunigungssensor und der zweiten Platine. Mit diesem Steg wird der Stecker gesichert.
In Abbildung 9.1 ist der Zusammenbau des Gehäuses mit Platinen und Beschleunigungssensor dargestellt. In der rechten Hälfte befindet sich der Beschleunigungssensor sowie die Hauptbefestigungspunkte. Für die Befestigungen werden unter anderem die Ösen ober- und unterhalb des Beschleunigungssensors verwendet. In der Mitte des Gehäuses befindet sich ein Durchlass für das Verbindungskabel von dem Sensor zu den Platinen. Außerdem sind hier die zwei weiteren Hauptbefestigungspunkte angebracht, welche das selbe Lochmaß, wie das bestehende System besitzen. Die Befestigungspunkte verstärken auch weiter das Gehäuse. In der linken Hälfte sind die beiden Platinen untergebracht.

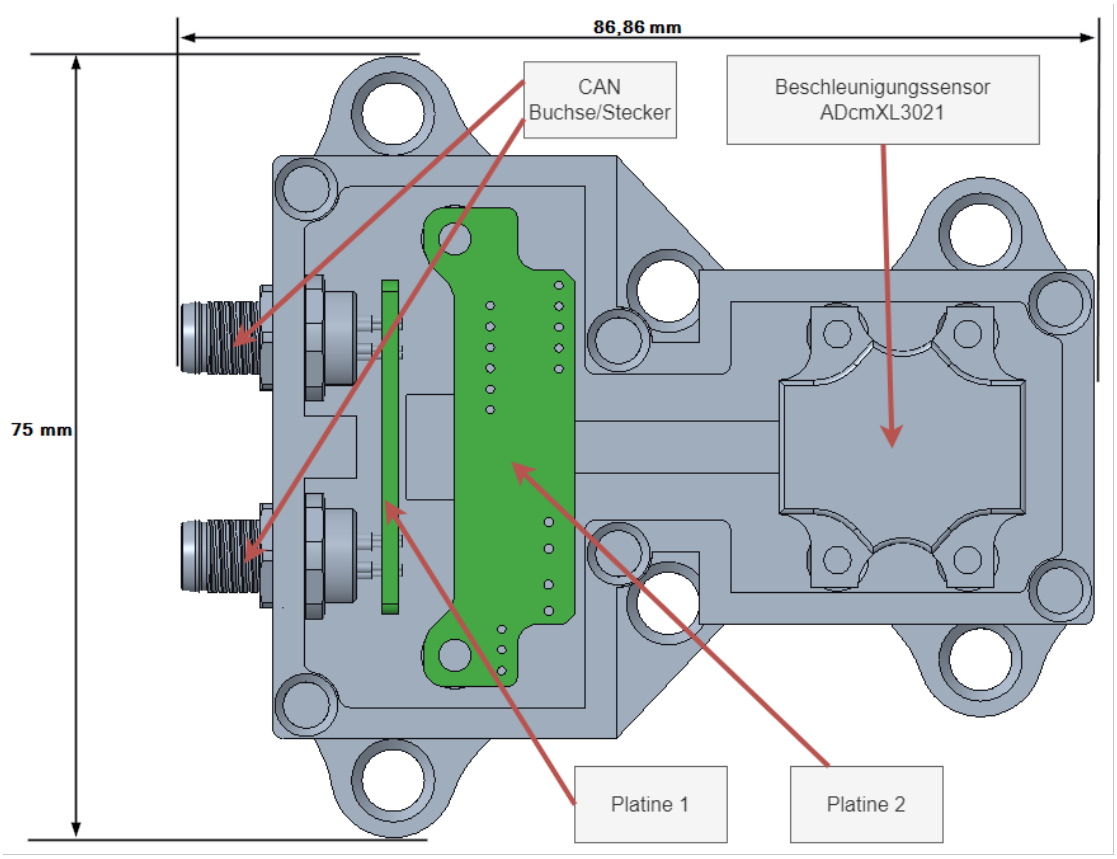


Abbildung 9.1: Fertig konstruiertes Gehäuse in CAD mit Beschleunigungssensor und Platinen in der Draufsicht

10 Software

Die Software für den Betrieb des Beschleunigungssensors gliedert sich in zwei Hauptteile. Der erste Teil ist das Programm auf dem verwendeten μC in der Programmiersprache C. Mit der Programmierung auf dem μC wird die Brückenfunktion realisiert. Hierfür müssen die beiden Bussysteme CAN und SPI konfiguriert und Funktionen geschrieben werden die es ermöglichen, Nachrichten zu versenden und zu empfangen. Mit dem Programmieren werden die Programmabläufe aus Abschnitt 7.3 implementiert.

Der zweite Teil sind die in Python geschriebene Scripte für verschiedene Aufgaben. Die Aufgaben sind dabei das Konfigurieren des Beschleunigungssensors, das Erstellen der Konfiguration und das Empfangen von SPI-Nachrichten mit Messwerten. Python dient auch dazu die Messwerte zu visualisieren.

10.1 Programmieren des Mikrocontrollers

Wenn der Mikrocontroller gestartet wird, durchläuft das Programm auf dem μC immer den selben Ablauf.

1. Initialisierung Pins, Schnittstellen und Interrupts
2. Initialisieren aller Variablen
3. Hauptteil des Programms in Endlosschleife mit Überprüfen der Interrupts

Die Endlosschleife stellt den normalen Programmzustand dar, in dem empfangene CAN-Nachrichten und fertige Messwerte reagiert wird.

10.1.1 Initialisierung der Pins, Schnittstellen und Interrupts

Grundsätzliche Einstellungen des μC lassen sich über die Configuration Bits vornehmen. Von den Standardeinstellungen abweichend sind, die Einstellungen für die Quelle der Taktrate und den Watch Dog. Mit der Einstellung HSPLL ist das Taktsignal des Mikrocontrollers durch den externen Oszillator gegeben. Das Taktsignal wird um den Faktor Vier auf 32 MHz erhöht.

```
1 #pragma config OSC = HSPLL      // Oszillator Selection bits (HS
    Oszillator, PLL enabled (Clock Frequency = 4 x FOSC1))
```

Außerdem wird der Watch Dog des μC deaktiviert, damit der μC im Leerlauf nicht neugestartet wird.

```
1 #pragma config WDT = OFF        // Watchdog Timer Enable bit (WDT
    disabled (control is placed on the SWDTEN bit))
```

Im ersten Schritt der Initialisierung werden alle Pins in einen definierten Zustand gebracht. Der definierter Zustand ist die Konfiguration als Eingang mit dem Schreiben einer Eins in das TRISx-Register mit gleichzeitigem Leeren der Speicher (Latches). Weiter werden diejenigen Pins, die als Ausgang fungieren sollen, als solche mit dem Schreiben einer Null als Ausgänge konfiguriert. Einzige Ausnahme ist hier der Pin für SDI, da dieser von dem SPI-Modul gesteuert wird.

```
1 TRISA = 0xFF; // set every pin as output
2 LATA = 0; // clear all latches
3 TRISAbits.RA0 = 0; // specifically set pin RA0 as output
```

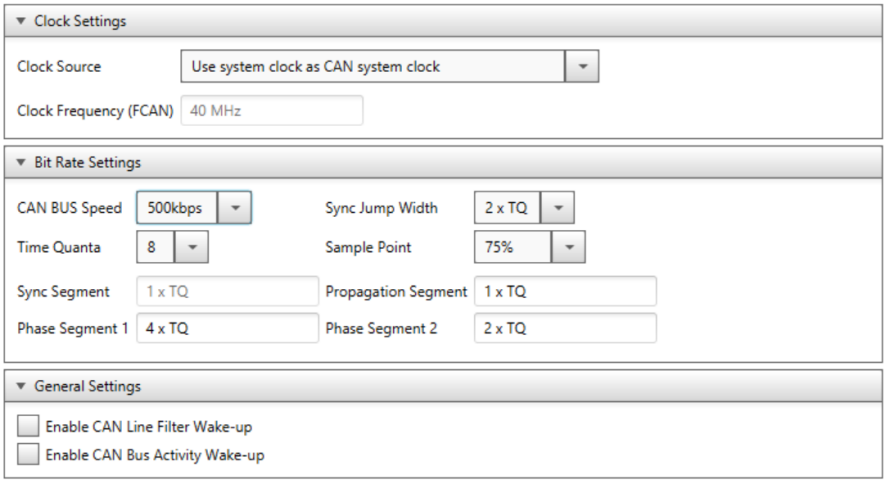
Nach dem Konfigurieren der Ein- und Ausgänge folgt die Konfiguration der beiden verwendeten Bussysteme. Beide Bussysteme bieten verschiedenste Einstellmöglichkeiten über die Register. Grundlage der SPI-Funktionen ist eine, auf Github von Microchip Technology veröffentlichte, Bibliothek [22]. Es sind Anpassungen notwendig, da ein anderer Mikrocontroller verwendet wird.

Der vom Beschleunigungssensor geforderte SPI-Mode 3 wird über die Register SMP, CKE und CKP konfiguriert. Damit der SPI-Bus grundsätzlich funktioniert muss das SSPEN-Register mit einer Eins beschrieben werden. Das Schreiben des Binärwertes *0000* konfiguriert den SPI-Treiber als Controller mit einer Taktrate $f_{\mathrm{SPI}} = \frac{FOSC}{4}$.

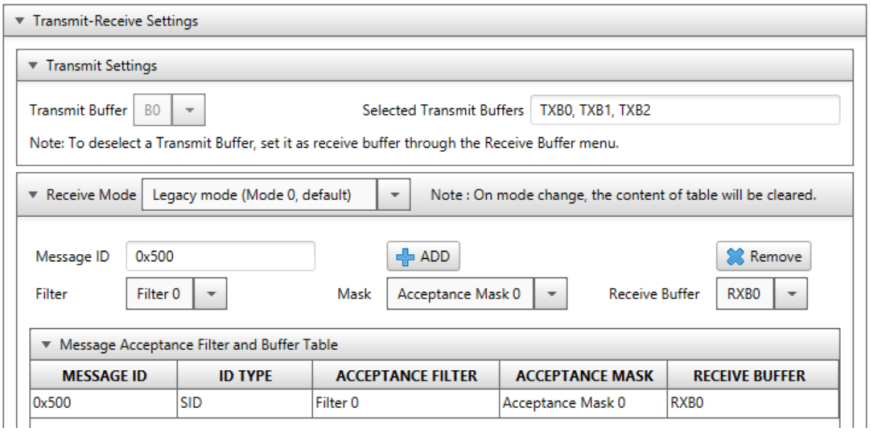
```
1 SSPSTATbits.SMP = 0; // data sample on middle
2 SSPSTATbits.CKE = 0; // transmit on transition from idle to active
3 SSPCON1bits.CKP = 1; // idle state is high level
4
5 SSPCON1bits.SSPEN = 1; // enable spi on pins
6 SSPCON1bits.SSPM = 0000; //f_spi = 32 MHz/ 4 = 8 MHz
```

Die Konfiguration des für den CAN-Bus verwendeten ECAN-Modul ist umfangreicher. Für die Konfiguration und die gesamte Bibliothek kann aber MPLAB Code Configurator (MCC) von Microchip Technology verwendet werden [20]. Mit diesem Unterprogramm für die verwendete Entwicklungsumgebung ist es möglich in einer grafischen Oberfläche die erforderlichen Einstellungen zu treffen. MCC erstellt automatisch aus den getroffenen Einstellungen die erforderlichen source- und header-Dateien. Der verwendete $\boldsymbol{\mu}$C wird nicht von MCC unterstützt, jedoch ist das verbaute CAN-Modul in unterstützten $\boldsymbol{\mu}$C verbaut, die simuliert werden können. Die in der Simulation erstellten Dateien lassen sich mit geringfügigen Änderungen von Registernamen übernehmen. Abbildung 10.1a und Abbildung 10.1b zeigen die getroffenen Einstellungen. Wenn die erstellten Dateien in das Projekt eingefügt werden, hat der CAN-Bus eine Baudrate von $f_{\mathrm{CAN}} = 500\frac{kBit}{s}$ und reagiert auf empfangene Nachrichten mit dem Identifier *0x500*.

Um eine Änderung der Baudrate im laufenden Betrieb zu ermöglichen, ist hierfür eine eigene CAN-Nachrichtenkennung vorgesehen. Wird diese Nachricht empfangen werden die Register BRGCON1, BRGCON2 und BRGCON3 mit neuen Werten beschrieben. Als Baudraten sind $f_{\mathrm{CAN}} = 1\frac{MBit}{s}; f_{\mathrm{CAN}} = 800\frac{kBit}{s}; f_{\mathrm{CAN}} = 500\frac{kBit}{s}; f_{\mathrm{CAN}} = 250\frac{kBit}{s}$ einstellbar. Eine Unterscheidung welche Baudrate eingestellt werden soll, erfolgt durch einen switch-case. Die Baudrate richtet sich nach dem Nachrichteninhalt in Tabelle 7.3, für eine Nachricht mit der Kennung *0x8*. Nach einer Veränderung der Baudrate ist kein Neustart des $\boldsymbol{\mu}$C notwendig.



(a) Konfiguration des ECAN-Moduls in MCC für die Baudrate



(b) Konfiguration des ECAN-Moduls in MCC für die gesendeten Nachrichten

10.1.2 Initialisierung Interrupts

Damit der μC sofort auf Änderungen reagieren kann, müssen die Interrupts entsprechend konfiguriert werden. Der folgende Programmausschnitt zeigt das Konfigurieren der Interrupts für den Anwendungsfall.

Interrupts generell werden mit dem GIE-Register aktiviert. Für den Empfang von Messwerten muss der μC auf eine steigende Flanke am BUSY-Signal reagieren. Interrupts für diesen Pin werden im INTEDG0-Register aktiviert mit der Sensitivität für eine steigende Flanke im INTEDG0-Register. Für den CAN-Bus werden die folgenden Interrupts aktiviert:

- Falsche Nachricht empfangen (IRXIE)
- Nachricht in Empfangsbuffer Eins empfangen (RXB1IE)
- Nachricht in Empfangsbuffer Null empfangen (RXB0IE)

Sind die Interrupts aktiviert, werden Flaggen mit einer Eins gesetzt, wenn das zugehörige Ereignis eingetreten ist. Ist ein Interrupt aufgetreten und wird bearbeitet, ist es unbedingt erforderlich, dass die Flagge sofort wieder auf Null gesetzt wird.

```
1 INTCONbits.GIE = 1; // enable interrupts in general
2 INTCONbits.PEIE = 1; // enable peripheral interrupts
3 RCONbits.IPEN = 0; //disables priorities for interrupts
4 INTCONbits.INT0E = 1; //pin RA0 interrupt
5 INTCON2bits.INTEDG0 = 1; // trigger on rising edge of busy signal
    signaling data is ready
6 PIE1bits.SSPIE = 1; // enable interrupts for SPI
7 PIE3bits.IRXIE = 1; // CAN invalid message received
8 PIE3bits.RXB1IE = 1; // gets set when a new CAN message is received.
    receive buffer 1
9 PIE3bits.RXB0IE = 1; // gets set when a new CAN message is received.
    receive buffer 0
```

10.1.3 Senden und Empfangen auf den Bussystemen CAN und SPI

Nachdem die beiden Bussysteme CAN und SPI richtig konfiguriert sind, können beide für das Senden und Empfangen von Nachrichten verwendet werden. Hierfür sind für beide Bussysteme Bibliotheken hinterlegt.

Nachrichten des CAN-Bus sind in einem Struct organisiert, welches von der im vorherigen Abschnitt erstellten Bibliothek definiert wird.

```
typedef union {
        Struct {
                uint8_t idType; // extended or normal length CAN ID
                uint32_t id; // CAN id of received or to be sent msg
                uint8_t dlc; // length in byte of CAN msg
                uint8_t data0; // data byte 0
                uint8_t data1; // data byte 1
                uint8_t data2; // data byte 2
                uint8_t data3; // data byte 3
                uint8_t data4; // data byte 4
                uint8_t data5; // data byte 5
                uint8_t data6; // data byte 6
                uint8_t data7; // data byte 7
        } frame;
}
uint8_t array[14];
```

Für das Versenden einer Nachricht müssen immer id, idtype und dlc defineirt sein. Analog zu den zu versendenden Nachrichten, werden die empfangenen Nachrichten auch in ein Struct der selben Struktur geschrieben. Um das Schreiben und Lesen der Struct möglichst schnell zu gestalten, werden Pointer auf die Structs für Senden und Empfangen initialisiert. Pointer zeigen auf die Adresse an dem das Struct in dem Speicher liegt. Die beiden Funktionen CAN_transmit und CAN_receive sind vom Aufbau identisch und haben als Eingabewert die Pointer auf die Structs.
Soll eine CAN-Nachricht mit CAN_transmit versendet werden wird in der Funktion überprüft, ob eines der Sendebuffer leer ist. Ist dies der Fall wird die zu versendende Nachricht in das freie Buffer geschrieben, der Rückgabewert der Funktion ist Eins. Das bewirkt, dass das Programm solange an dieser Stelle verharrt, bis die CAN-Nachricht in einen der Buffer geschrieben werden konnte. Die Funktion CAN_receive für das Empfangen von CAN-Nachrichten überprüft die Empfangsbuffer auf neue Nachrichten. Wenn sich in

einem Buffer eine Nachricht befindet, ist der Rückgabewert der Funktion CAN_receive, die empfangene Nachricht.

Das Senden und Empfangen von Nachrichten auf dem SPI-Bus kann mit einer Funktion abgedeckt werden. Die Funktion für das Versenden und Empfangen hat als Eingabewert die zu versendende Nachricht mit dem Rückgabewert der empfangenen Nachricht. Das Programm wird solange in der Funktion für das Versenden der SPI-Nachricht gehalten, bis das Versenden einer Nachricht vollständig abgeschlossen ist. Dies wird mit einer while-Schleife überprüft. Die Flagge SSPIF wird gesetzt nachdem eine Nachricht versendet wird, die Flagge BF-Flagge nach dem Empfang einer Nachricht.

```
1 uint8_t inline spiExchangeByte(uint8_t data){
2         SSPBUF = data; // write data to be sent to buffer
3         while (!PIR1bits.SSPIF){;} // wait until transmission is
              completed
4         SSPSTATbits.BF = 0; // clear buffer full flag
5         PIR1bits.SSPIF = 0; // clear transmission flag
6         return SSPBUF;
7 }
```

Mit der beschriebenen Funktion, wird die in Abschnitt 3.5 erläuterte SPI-Kommunikation mit dem Beschleunigungssensor, implementiert. Da der μC nur ein Byte beim Versenden und Empfangen speichern kann, ist es notwendig zwei spiExchangeByte Funktionen zu verketten.

10.1.4 Hauptprogramm - Endlosschleife

Wenn alle Initialisierungen abgeschlossen sind, läuft das Programm auf dem μC in einer Endlosschleife. Hierbei werden die Flaggen für den Erhalt einer CAN-Nachricht und einer steigenden Flanke am BUSY-Signal durchgehende auf ihren Zustand geprüft. Sollte eine der beiden Flaggen gesetzt werden, wird der dazugehörige Programmteil ausgeführt. Dies entspricht dem Ablauf in Abschnitt 7.3. Auf die Flagge für eine steigende Flanke wird nur reagiert, wenn der Beschleunigungssensor vollständig konfiguriert ist. Wenn ein Interrupt für das Empfangen einer CAN-Nachricht aufgetreten ist, kann auf die Nachricht entsprechend ihres Inhaltes reagiert werden. Hier wird auf CAN-Nachrichten mit den Kennungen *0x1 bis 0xC* aus der Tabelle 7.2 reagiert und der entsprechende Code ausgeführt.

10.2 Python

Python Scripte werden für verschiedenste Aufgaben verwendet. Damit ein Computer über Python mit dem Beschleunigungssensor/ μC über CAN kommunizieren kann ist in erster Linie ein Konverter zu CAN notwendig, hier der USB-to-CAN V2 von Ixxat. Auf diesen Konverter greift eine Bibliothek zu, die eine Kommunikation ermöglicht [31]. Die Bibliothek erfordert das Anlegen einer Konfigurationsdatei can.ini mit mindestens dem Herrsteller des Interfaces und dem Kanal. Auf diese Datei greift die Bibliothek standardmäig bei dem Initialisieren der CAN-Verbindung zu. Für ein Initialisieren einer CAN-Verbindung, wird die Funktion can.Bus() benötigt.

```
1 bus = can.Bus(interface='ixxat', channel='0', bitrate=500000)
```

Mit dem Eingabewert bitrate = 500000 lässt sich die Baudrate des CAN-Bus einstellen. Die hier eingestellte Bitrate muss identisch zu der auf dem μC eingestellten Baudrate sein. Für das Versenden wird die Funktion bus.send(msg) verwendet. Das Objekt msg muss vorher erstellt werden.

```
1 msg = can.Message(arbitration_id=canID,data=[data0, data1, ...],
    is_extended_id=False, is_fd=False)
```

Die arbitration_id ist der Identifier der CAN-Nachricht. Für den Identifier kann außerdem mit is_extended_id festgelegt werden, ob es sich um einen normale oder erweiterteten Identifier handelt. Unter data können die Bytes eingetragen werden, die versendet werden sollen. Nachrichten lassen sich mit der Funktion bus.recv(None) empfangen, deren Rückgabewert die empfangene Nachricht ist. Anstelle von None kann auch ein Zahlenwert eingetragen werden, der Zahlenwert entspricht dann der Zeit in Sekunden, in der nicht auf neue Nachrichten reagiert wird.
Damit alle Nachrichten empfangen werden können, kann die Funktion bus.recv(None) in einer Endlosschleife aufgerufen werden. Aufbauend auf diesen Funktionen werden fünf Python-Scripte für die Kommunikation mit dem Beschleunigungssensor entwickelt.

1. *receiveData.py* Empfangen von Messwerte
2. *jsonCreator.py* Erstellen einer JSON-Datei mit allen Einstellungen für den Beschleunigungssensor
3. *configure_ sensor.py* Konfiguration des Beschleunigungssensors
4. *adcmxl3021_ registers.py* Berechnung neuer Registerwerte, Adressen
5. *definitions.py* Definition von festen Werten. Identisch zu Definition auf μC

10.2.1 Empfangen von Messwerten

Vor dem Empfangen aller Messwerte werden alle Nachrichten auf dem Bus empfangen. Die Nachrichten mit Messwerten werden anhand der selbstdefinierten Kennungen in Tabelle 7.2 herausgefiltert. Empfangene Messwerte erhalten einen Zeitstempel und werden in einem eigenen Ordner zusammen mit den Metadaten gespeichert. Die empfangenen Nachrichten werden einer Liste hinzugefügt und erst mit dem Erhalt aller Messwerte in einer csv-Datei gespeichert. Da ein Messwert eine Größe von zwei Byte hat, in CAN-Nachrichten ein Datenbyte aber nur ein Byte groß ist, müssen die Messwerte wieder zusammengesetzt werden. Gleichzeitig zu dem Zusammensetzen werden die Messwerte von Hexadezimalwerten in Dezimalwerte konvertiert.

```
# save data to csv and stitch it together
writer.writerow([((msgData[1] <<8) | msgData[2]), ((msgData[3] <<8) |
    msgData[4]) , ((msgData[5] <<8) | msgData[6])])
```

Da die Messwerte für den MTC-Modus im Zweierkomplement formatiert sind, werden die Messwerte vor dem Abspeichern in einen Dezimalwert mit Vorzeichen konvertiert.

```
for msgData in msgBuffer:
        # stitch data together
        x_data_raw = ((msgData[1] << 8) | msgData[2])
        # convert received data from two complements
        x_data_converted = twos_complement(x_data_raw, 16)
        # save data to csv
        writer.writerow([x_data_converted, y_data_raw_converted,
           z_data_raw_converted])
```

Nach dem Abspeichern ist der Empfang von Messwerten beendet, das Script kann wieder passiv alle Nachrichten auf dem CAN-Bus empfangen.

10.2.2 Erstellen einer JSON-Datei mit Einstellungen

Für das Erstellen der JSON-Datei sind alle schreibbaren Register in einer eigenen Bibliothek in Python hinterlegt. Nicht-Schreibbare Register sind ausgelassen. In der Bibliothek sind die Standardwerte der Register hinterlegt. Außerdem können über Funktionen neue Registerwerte definiert werden. Die Funktionen der Register haben als Eingabewerte die einzelnen Einstellungen der Register. Der neue Registerwert wird aus Bitoperationen der einzelnen Einstellungen zusammengesetzt.

```
settingsIMU = imuRegisterSettings()
settingsIMU.rec_ctrl(sr0_enable=1, sr1_enable=0, sr2_enable=0,
    sr3_enable=0, recording_mode="MTC")
```

Nachdem alle gewünschten Änderungen von Einstellungen durchgeführt wurden, werden alle Register mit ihren Einstellungen in der JSON-Datei gespeichert. Für ein Register werden die getätigten Einstellungen, der Standardwert, der geänderte Registerwert, die Adresse und die Registerbank gespeichert. Ein Beispiel für eine JSON-Datei befindet sich in Unterabschnitt A.9.4.

10.2.3 Konfiguration des Beschleunigungssensors aus der JSON-Datei

Um den Beschleunigungssensor zu konfigurieren wird die erstellte JSON-Datei eingelesen. Der Registerwert, für alle schreibbaren Register, wird der JSON-Datei entnommen und auf den Beschleunigungssensor geschrieben. Bevor alle Register beschrieben werden, wird eine CAN-Nachricht mit der Kennung *0x6* im ersten Datenbyte versendet. Dies signalisiert dem μC, dass die Konfiguration des Beschleunigungssensors folgt. Solange die Konfiguration stattfindet, ist deaktiviert, dass der μC auf eine steigende Flanke am BUSY-Signal reagiert. Außerdem wird mit dem Start der Konfiguration der escape code von dem μC an den Beschleunigungssensor versendet. Damit ist sichergestellt, dass sich der Beschleunigungssensor im Leerlauf befindet. Ist dies der Fall, versendet der μC eine CAN-Nachricht mit der Kennung *0xB*. Nach ihrem Erhalt kann mit der Konfiguration begonnen werden. Aus der eingelesenen JSON-Datei werden alle Registerwerte hintereinander eingelesen und in das entsprechende Register auf dem Beschleunigungssensor geschrieben. Die Adresse und Registerbank des Registers werden auch der JSON-Datei entnommen. Ist die Konfiguration beendet, wird eine Ende-Nachricht verschickt, mit der die Sensitivität auf die steigende Flanke wieder aktiviert wird. Außerdem wird mit jeder neuen Konfiguration, wenn gewünscht, der Beschleunigungssensor genullt. Am Ende werden die aktuellen Registerwerte auf dem Beschleunigungssensor gespeichert, sodass dieser auch wenn die Spannungsversorgung unterbrochen wurde, wieder mit den selben Einstellungen startet.

11 Übertragungsdauer für Messwerten

Um alle Messwerte in den FFT-Betriebsmodi und dem MTC-Betriebsmodus übertragen zu können, wird eine nicht insignifikante Zeit benötigt. Für eine Übertragung von Messwerten aus den FFT-Betriebsmodi werden insgesamt 2050, im MTC- Modus 4098 CAN-Nachrichten benötigt. Die Länge der CAN-Nachrichten ist variabel und muss für die Kennungen *0x92, 0x94, 0x96, 0xE2, 0xE4* und *0xE6* einzeln bestimmt werden. Die Berechnung der Zeit pro Nachricht t_{CAN} ist über einen auf Github verfügbaren Rechner möglich [10]. Für diese Berechnungen wird eine Übertragungsgeschwindigkeit von $f_{\mathrm{CAN}} = 500\frac{kBit}{s}$ angenommen. Der Inhalt der Nachricht wird so gewählt, dass die höchste Anzahl an Stuffing Bits notwendig ist.
Für CAN ergeben sich damit die Übertragungsdauern für eine Nachricht t_{CAN} aller Kennungen für Messwerte. Aus der Anzahl der Nachrichten pro Kennungen n_{msg} ergibt sich eine Gesamtübertragungsdauer aller Nachrichten pro Kennung $t_{\mathrm{CAN,\ T}}$

$$t_{\mathrm{CAN,\ T}} = n_{\mathrm{msg}} \cdot t_{\mathrm{CAN}} \tag{11.1}$$

Die Länge einer SPI-Übertragung t_{SPI} lässt sich aus der übertragenden Anzahl an Bits n_{bits}, der Taktrate von SPI f_{SPI} und der Ruhezeit t_{stall} berechnen

$$t_{\mathrm{SPI}} = \frac{n_{\mathrm{bits}}}{f_{\mathrm{SPI}}} + 2 \cdot t_{stall} \tag{11.2}$$

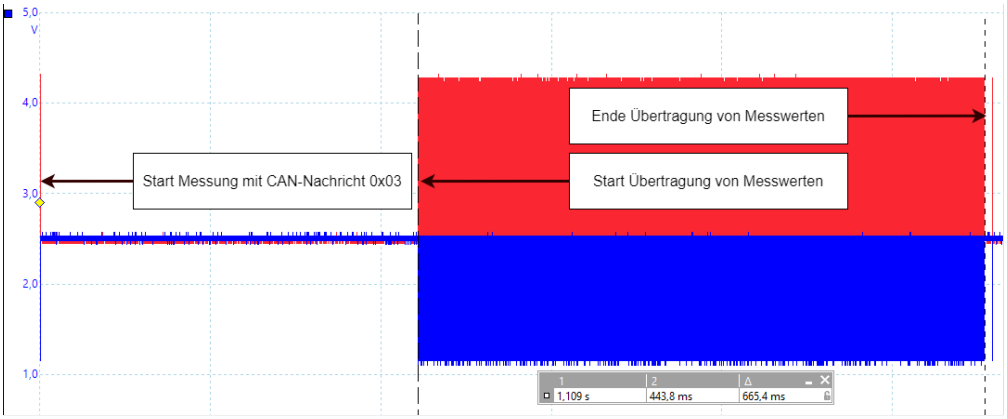
Es sind nur insgesamt 32 Bits notwendig, da die Auswahl der Registerbank spätestens mit der ersten Übertragung entfällt. Für jede Nachricht mit Messwerten aus den drei Achsen sind drei Leseaufforderungen notwendig. Die Gesamtzeit $t_{\mathrm{SPI,\ T}}$ für alle Nachrichten mit Messwerten ergit sich somit zu

$$t_{\mathrm{SPI,\ T}} = 3 \cdot n_{\mathrm{msg}} \cdot t_{\mathrm{SPI}} \tag{11.3}$$

Die Gesamtdauer einer Messwertsübertragung t_{data} ergibt sich aus den Übertragungszeiten pro Nachricht beider Bussysteme.

$$t_{\text{data}} = t_{\text{CAN, T}} + t_{\text{SPI, T}} \tag{11.4}$$

Berechnet man daraus die Übertragungsdauer von Messwerte, ergeben sich daraus die Werte in Tabelle A.5. Die errechneten Werte für die Übertragungsdauer für t_{data} im MTC- und FFT-Modus bestätigen sich in einer Messung in Abbildung 11.1. Mit einer Übertragungsdauer von $t_{\text{data}} = 1,34\ s$ im MTC-Modus und $t_{\text{data}} = 665,4\ ms$ im MTC-Modus sind die Übertragungsdauern sogar besser als die erechneten Werte. Einer Übertragung von Messwerten unter einer Sekunde ist möglich, jedoch gibt es dabei nicht mehr viel Spielraum gegeben.



(a) FFT-Betriebsmodi mit CANH (rot) und CANL (blau)

(b) MTC-Betriebsmodus mit CANH (rot) und CANL (blau)

Abbildung 11.1: Reale Übertragungsdauer von Messwerten im MTC- und FFT-Modus mit den beiden Signalleitungen CANH (rot) und CANL (blau)

12 Inbetriebnahme

Für eine erste Inbetriebnahme und die Übertragung von einzelnen CAN-Nachrichten, kann das Programm canAnalyser Mini 3 verwendet werden, welches zusammen mit dem Treiber für den USB-to-CAN V2 installiert wird [14]. Das Programm canAnalyser Mini 3 ist in Abbildung 12.1 dargestellt. Der obere Teil des Programms zeigt alle gesendeten und empfangenen Nachrichten. Im unteren Teil des Programmes können eigenen CAN-Nachrichten mit Kennung und Datenbytes definiert und versendet werden.

Bei der ersten Inbetriebnahme tritt der Fehler auf, dass bei höheren Baudraten des CAN-Bus $f_{\text{CAN}} > 500\frac{kBit}{s}$, es zu Bit stuffing Errors kommt. Der Fehler tritt nur auf, wenn der $\boldsymbol{\mu}$C Nachrichten versendet. Er lässt sich auf das Verhältnis zwischen den Längen von Eins und Null zurückführen. Ein Beheben es Fehlers ist nicht möglich, sodass die Baudrate auf $f_{\text{CAN}} = 500\frac{kBit}{s}$ begrenzt wird. Mit dieser Baudrate ist ein stabiler Betrieb möglich.

Es ist möglich mit dem in Abschnitt 7.4 beschrieben Konzept für die Identifikation von CAN-Nachrichten die empfangenen Nachrichten zu unterscheiden. Dies lässt sich mit einem Auslesen → Schreiben → Auslesen von Registern auf dem Beschleunigungssensor in Abbildung 12.2 testen. In diesem Fall wird das AVG_CNT-Register mit der Adresse *0x3A* zuerst ausgelesen. Dafür muss eine Nachricht mit der Kennung *0x2* im 0-ten Datenbyte, Registerbank *0x00* im ersten Datenbyte und Adresse *0x3A* im zweiten Datenbyte an den $\boldsymbol{\mu}$C versendet werden. Wie erwartet, antwortet der $\boldsymbol{\mu}$C mit dem Standardwert von *0x7421*. Eine Nachricht mit Kennung *0x1* und den zu schreibenden Daten in dem dritten und vierten Datenbyte, vom Computer an den $\boldsymbol{\mu}$C, beschreibt das Register mit einem neuen Wert. Das korrekte Schreiben des Registers lässt sich durch ein erneutes Auslesen des Registers verifizieren.

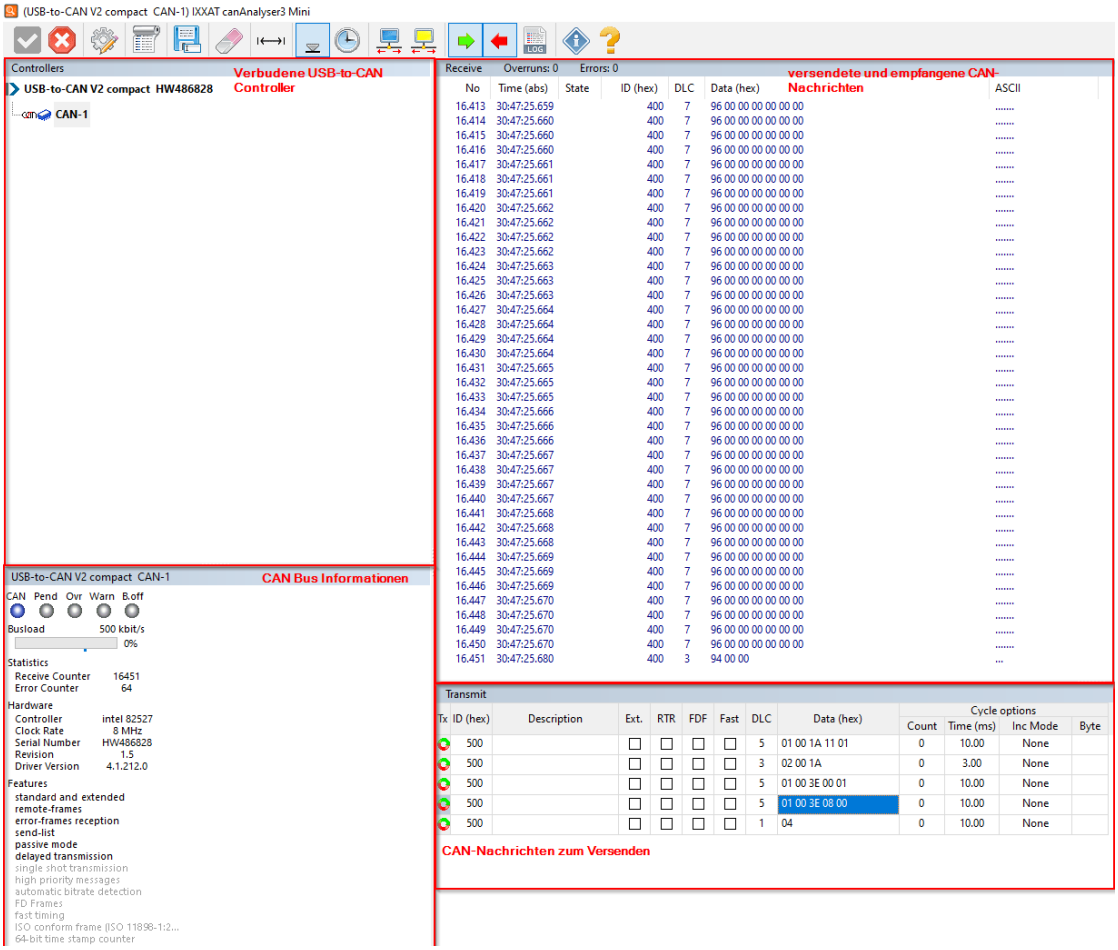
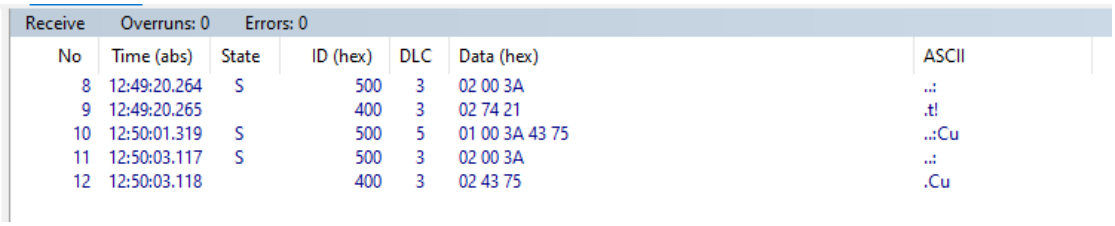


Abbildung 12.1: Übersicht des Programms canAnalyser Mini 3 zum Versenden und Empfangen von CAN-Nachrichten



Receive Overruns: 0 Errors: 0

No	Time (abs)	State	ID (hex)	DLC	Data (hex)	ASCII
8	12:49:20.264	S	500	3	02 00 3A	..:
9	12:49:20.265		400	3	02 74 21	.t!
10	12:50:01.319	S	500	5	01 00 3A 43 75	..:Cu
11	12:50:03.117	S	500	3	02 00 3A	..:
12	12:50:03.118		400	3	02 43 75	.Cu

Abbildung 12.2: Auslesen, Schreiben und erneutes Auslesen eins Registers auf dem Beschleunigungssensor über CAN

Für den Einbau in das Gehäuse werden die beiden Platinen nochmal neu verlötet. Hierbei treten einige Probleme auf, die behoben werden müssen. Zum Einen ist das Starten des μCs nach einer Unterbrechung der Spannungsversorgung oder einem Neuprogrammieren instabil. Zum Anderen ist es nicht möglich, Anworten über CAN von dem μC zu erhalten.
Das erste Fehlerbild äußert sich dadurch, dass sich der μC nach einem Zurücksetzen im darauffolgenden Startvorgang für Spannungen $U > 3,2\ V$ aufhängt. Ein Zurücksetzen wird entweder durch Unterbrechen der Spannungsversorgung oder durch Ziehen des MCLR-Pins auf ein niedriges Spannungsniveau ausgelöst. Dies lässt sich durch das Verlöten eines externen 8 MHz statt 40 MHz Oszillator beheben. Um den Takt wieder zu erhöhen wird die Taktrate durch die Verwendung eines PLL Multiplikators auf 32 MHz erhöht. Als Ursache für das Problem ist zu vermuten, dass der μC mit einem Takt von 40 MHz und einer Spannung von $U = 3,3\ V$, außerhalb der empfohlenen Betriebsbedingungen betrieben wird [21, S. 422].
Für das zweite Problem ist die Ursache nicht bestimmbar, es wird aber ein Workaround erarbeitet. Damit ein problemloser Betrieb möglich, ist muss das Starten des μCs in den folgenden Schritten erfolgen:

1. Starten des μC, ohne dass der Beschleunigungssensor mit der Platine verbunden ist
2. Versenden von zwei CAN-Nachrichten an den μC
   a) CAN meldet einen Bit Suffing Error und μC startet neu
   b) μC anwortet auf die zweite CAN-Nachricht mit Verzögerung
3. Verbinden des Beschleunigungssensors mit der Platine

Als eine Ursache für das Problem ist entweder eine falsche Programmierung oder eine Limitierung durch den μC anzunehmen. Durch Tests können andere Fehlerquellen ausgeschlossen werden. Dieser Fehler bedingt, dass ein externe Betrieb nicht möglich ist, womit ein Einsatz im Datenloggerssystem nicht möglich ist.

Alle Komponenten (Platine 1, Platine 2 und Beschleunigungssensor) in Abbildung 12.3 werden in dem in Kapitel 9 konstuierten Gehäuse verbaut. Bei dem Einbau der Komponenten muss eine bestimmte Reihenfolge eingehalten werden und beide Platinen vorher vollständig verlötet sein. Daher müssen alle Komponenten sich auf den Platinen befinden und beide Platinen müssen über Kabel oder Stecker miteinander verbunden sein. Um Kabel und die Brücken für das Tauschen von SDI und SDO zu sichern, wird auf die kritischen Stellen Silikon aufgebracht. Wenn beide Platinen fertig für den Einbau sind, muss zuerst Platine 1 eingebaut und die CAN-Stecker am Gehäuse befestigt werden. Danach wird die zweite Platine an den vorgesehenen Stellen verschraubt. Als Letztes wird der Beschleunigungssensor verbaut und mit Platine 2 verbunden.

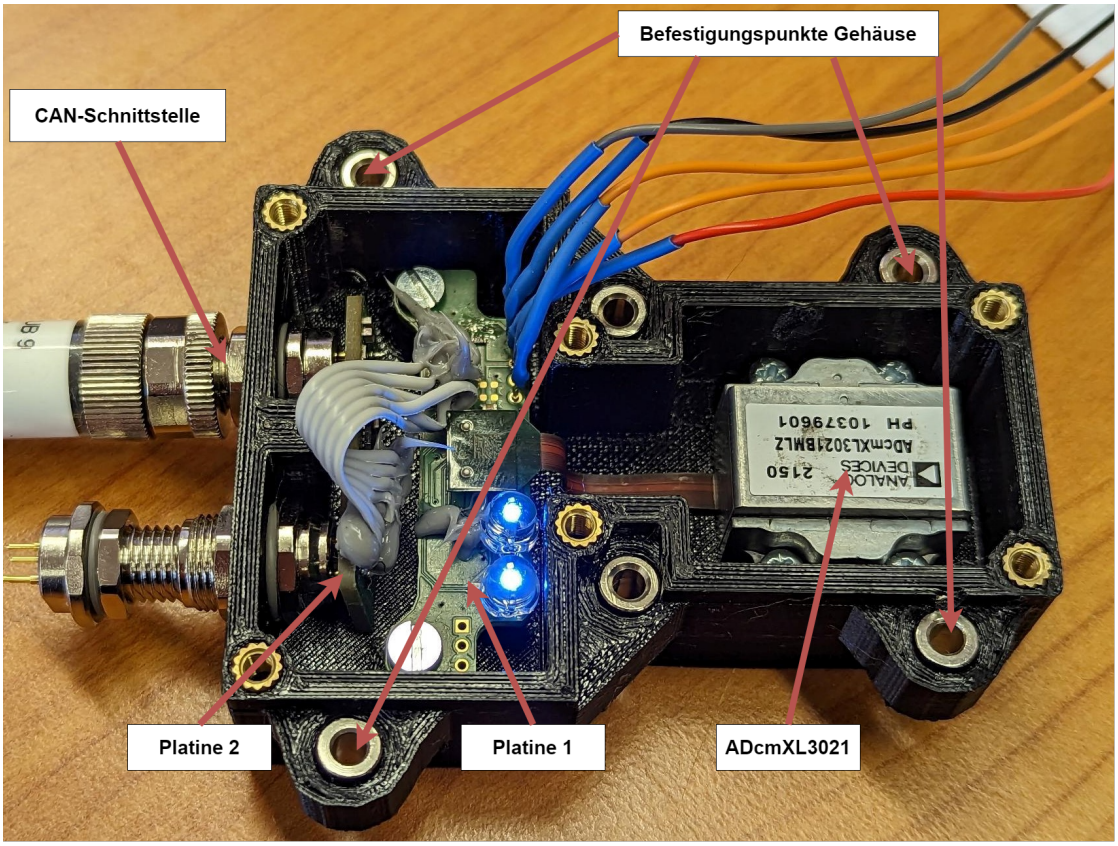


Abbildung 12.3: Zusammengebautes Gehäuse mit allen Komponenten

13 Abschlussmessung

Für den Abschluss dieser Arbeit wird eine Schwingprüfanlage dafür verwendet das gesamte Gehäuse mit Beschleunigungssensor und CAN-Schnittstelle mit hohen Frequenzen belasten zu können. Gleichzeitig werden verschiedenen Messung durchgeführt. Die Messungen unterscheiden sich untereinander und haben zum Ziel den Beschleunigungssensor in verschiedenen Aspekten charakterisieren zu können. Außerdem wird eine Langzeitmessung der auftretenden Beschleunigungen in einer Hydraulikpumpe durchgeführt.

Schwingprüfanlagen sind Maschinen für die Bauteil- oder Baugruppenprüfung. Auf diesen werden die Bauteil- oder Baugruppen auf ihre Widerstandsfähigkeit gegenüber Vibrationen und sich (zeitlich) verändernden Beschleunigungen getestet. Die Schwingungen können in einer oder mehreren Achsen gleichzeitig vorkommen.

13.1 Messaufbau

Die Messungen mit der Schwingprüfanlage werden im Leichtbaulabor des Departments Fahrzeugtechnik und Flugzeugbau an der Hochschule für angewandte Wissenschaften Hamburg durchgeführt. Die von dem Leichtbaulabor bereitgestellten Geräte für die Messung mit der Schwingprüfanlage sind:

- Schwingprüfanlage *LDS V406* von *Brüel & Kjær*
- Leistungsverstärker *19 Z/ 500* von *FG Elektronik*
- Funktionsgenerator *HM8131-2* von *HAMEG Instruments*
- Referenzsensor *Type 4335* von *Brüel & Kjær*

Die Schwingprüfanlage setzt das von dem Funktionsgenerator (FG) erzeugte Signal in eine mechanische Auslenkung um. Eine sich zeitlich verändernde Auslenkung führt zu einer Beschleunigung.

Für die Messung ergibt sich der schematische Ablauf der Signale durch alle Geräte in Abbildung 13.1. Mit dem Funktionsgenerator können verschiedenste periodische Signale in einem Frequenzbereich von 100 $\mu Hz \geq f \geq 15\ MHz$ generiert werden. Die Amplitude der Beschleunigung ist dabei von dem Leistungsverstärker abhängig. Für den geschwünschten Schwingungsverlauf ein einstellbares, durch den Funktionsgenerator generiertes Signal, mit dem Leistungsverstärker geziellt verstärkt. Der Beschleunigungssensor wird in seiner z-Achse beschleunigt. Das gesamte System ist dabei gesteuert und nicht geregelt. Die Schwingprüfanlage hat, bei einer Luftkühlung ohne Lüfter, eine maximale Beschleunigung von $a = 490\frac{m}{s^2} = 49,95\ g$, mit einer maximale Frequenz von $f = 9000\ Hz$ [15].

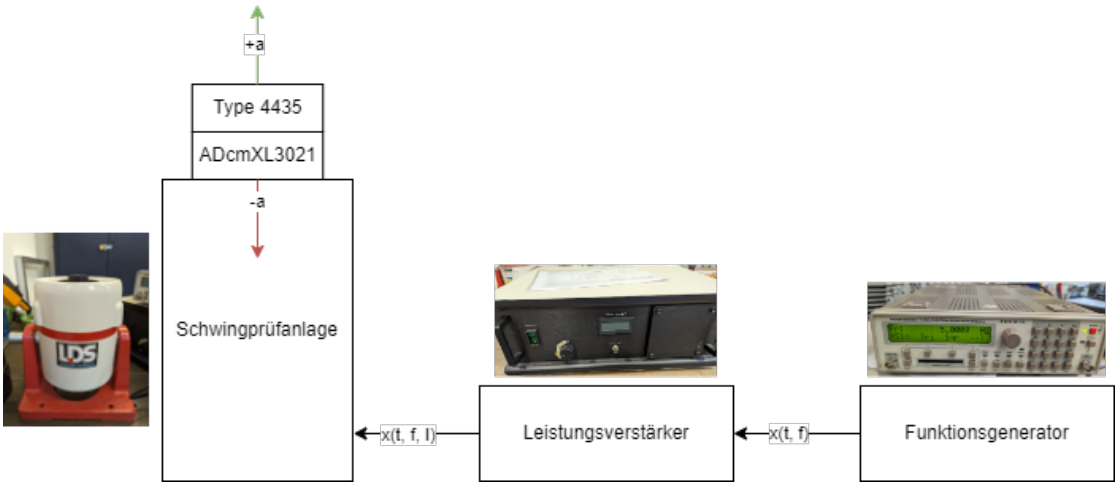


Abbildung 13.1: Schematischer Signalfluss für die Erzeugung einer Beschleunigung am Gehäuse mit ADcmXL3021 und Referenzsensor

Um die gemessene Beschleunigungen des ADcmXL3021 Beschleunigungssensors zu verifizieren wird ein Referenzsensor verwendet. Dieser wird auf dem Geäuse des ADcmXL3021 Beschleunigungssensors mit Wachs befestigt. Dabei handelt es sich um einen piezoelektrischen Sensor mit einer Sensitivität von $19,2\frac{mV}{g}$, dessen analoges Ausgangsspannungssignal zunächst verstärkt wird. Das verstärkte Signal wird mit einem Oszilloskop (Picoscope 3404D) aufgenommen. Mit dem Oszilloskop können der Spitzen-Spitzen-Wert, das quadratische Mittel (RMS) und Frequenzen aufgenommen werden.

Den Aufbau für die Messung mit dem ADcmXL3021 und dem Referenzsensor zeigt Abbildung 13.2. Das Gehäuse mit beiden Platinen und dem Beschleunigungssensor ist mit einem Adapter mit der Aufnahme der Schwingprüfanlage verschraubt. Der Adapter muss zuerst konstruiert werden und wird auch im 3D-Druck Verfahren hergestellt.

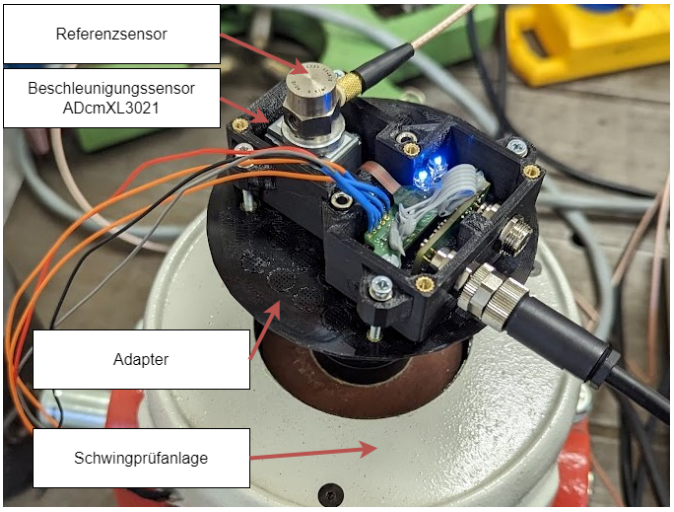


Abbildung 13.2: Beschleunigungssensor ADcmXL3021 und Referenzsensor auf Schwingprüfanlage

13.2 Beschreibung der durchzuführenden Messungen

Zur Verifikation der korrekten Funktionsweise des Beschleunigungssensors zusammen mit den μC und der CAN-Schnittstelle, sind mehrere Messungen durchzuführen. Die Messungen mit einer kurzen Beschreibung sind in Tabelle 13.1 aufgestellt. Für die genauen Messparameter in jeder Messung sei auf die Tabellen Tabelle A.14 bis Tabelle A.20 im Anhang verwiesen. Die Messparameter ergeben sich aus den verschiedenen Aspekten die untersucht werden sollen. Für die FFT-Betriebsmodi werden verschieden Abtastraten gewählt, meistens $f_s = 27,5\ kHZ$. Mit $f_s = 27,5\ kHZ$ wird am Besten der in den Anforderungen (Tabelle 6.2) festgelegte Frequenzbereich von $1\ kHz \geq f \geq 10\ kHz$ abgedeckt, sodass das Verhalten hier von besonderem Interesse ist. Für die Messungen im MTC-Modus wird die niedrigste Abtastrate $f_s = 1718,75\ Hz$ gewählt, da mit dieser Messungen über einen längeren Zeitraum möglich sind. Die meisten Messungen werden in den FFT-Betriebsmodi durchgeführt. Mit diesen Messungen soll in erster Linie die Genauigkeit der FFT-Messungen bestimmt werden. Die Untersuchungen betreffen die Genauigkeit, bezüglich der gemessene Frequenz, sowie der Amplitude der Beschleunigung.

Mit der ersten Messung wird die Auswirkung der der Abtastrate untersucht. Dazu werden in dem Frequenzbereich von $0,5\ Hz \geq f \geq 10\ kHz$ für vier Abtastraten ($f_s = 220\ kHz; f_s = 27,5\ kHz; f_s = 1718,75\ Hz; f_s = 3437,5\ Hz$) die Beschleunigung an den selben Frequenzen gemessen. Die Abtastraten $f_s = 220\ kHz; f_s = 1718,75\ Hz$ werden als die maximal und minimal einstellbare Abtastrate für diese Messung ausgewählt. Die Beschleunigungen werden dabei für eine Zeit von $t_{\mathrm{FFT}} \approx 2\ s$ gemessen. Für eine Untersuchung des Verhaltens der unteren Frequenzlimits $f_{\mathrm{cutoff,\ low}} = bw$, sind die ersten Frequenzen, an die unteren Frequenzlimits der Abtastraten angepasst. Das obere Frequenzlimit $f_{\mathrm{cutoff,\ high}}$ kann mit dem Messaufbau nicht für $f_s = 220\ kHz; f_s = 27,5\ kHz$ erreicht werden.
Um die Linearität und Genauigkeit der gemessenen Beschleunigung zu untersuchen, werden die Messungen 2 und 3 durchgeführt. In beiden Messungen werden jeweils dieselben Einstellungen für Amplitude und Frequenz der Beschleunigung eingestellt. Die Messungen unterscheiden sich jedoch im Betriebsmodus. In Messung 2 wird der Sensor im MFFT-Modus betrieben, in Messung 3 im MTC-Modus.
In der Messung 4 werden die Messparamter innerhalb einer längeren Messung im MFFT-Modus verändert. Damit soll untersucht werden, wie unterschiedliche Bedingungen sich innerhalb einer Messung darstellen. Damit der Sensor in dem Datenlogger eingesetzt werden kann, gilt es zu überprüfen, ob die Messungen konsistent sind. Dies wird mit der Messung 5 erreicht.
In der Messung 5 werden im ersten Schritt die Messparamter über mehrere Messungen in einem festen Zeitintervall im AFFT-Modus nicht verändert. Im zweitem Schritt werden während einer Messung die Einstellungen, am Funktionsgenerator, für Frequenz und Amplitude über den Messzeitraum verändert. Da das Messen von längeren Messzeiträumen durch eine Mittelwertsbildung der FFTs erreicht wird, muss überprüft werden wie sich die Mittelwertsbildung auf die Messungen auswirkt.
Dies wird in Messung 6 durch Erhöhen von $n_{\overline{FFT}}$, bei konstanter Frequenz und Amplitude der Beschleunigung, untersucht. Die Messung 7 fasst mehrere unterschiedliche Messungen mit unterschiedlichen Parametern zusammen. Die Messung 10 ist eine Langzeitmessung der auftretenden Beschleunigungen in einer Hydraulikpumpe. Für die Durchführung der Messung wird der Beschleunigungssensor im AFFT-Modus mit einer Abtastrate von $f_s = 27,5\ kHZ$ betrieben. Die Messung wird über einen Zeitraum von 30 Minuten durchgeführt. Über diesen Zeitraum werden einem Abstand von 30 Sekunden die Beschleunigungen für $t_{\mathrm{FFT}} = 1,49\ s$ gemessen.

Tabelle 13.1: Zusammenfassung der abschließenden Messungen mit Kurzbeschreibungen

ID	Betriebsmodus	Kurzbeschreibung	Messparameter
1	MFFT	Sweep durch festgelegte Frequenzbereiche für alle Abtastraten unter Berücksichtigung Frequenzlimits	Tabelle A.14
2	MFFT	Einstellen von verschiedenen Beschleunigungen bei gleichen Frequenzen im MFFT-Modus	Tabelle A.15
3	MTC	Einstellen von verschiedenen Beschleunigungen bei gleichen Frequenzen im MTC-Modus	Tabelle A.15
4	MFFT	Verändern der Frequenz und Amplitude der Beschleunigung innerhalb einer Messung	Tabelle A.16
5	AFFT	Konstante Frequenzen und Amplituden der Beschleunigung über mehrere Messungen im AFFT-Modus	Tabelle A.17
6	MFFT	Erhöhen von $n_{\overline{FFT}}$ bei konstanter Frequenz und Amplitude der Beschleunigung	Tabelle A.18
7	MTC und MFFT	Verschiedenste Messungen	Tabelle A.19
10	AFFT	Messen der auftretenden Beschleunigungen bei einer Hydraulikpumpe über einen längeren Messzeitraum	Tabelle A.20

13.3 Auswertung der Messergebnisse der Abschlussmessung

In diesem Abschnitt werden die Messungen aus Tabelle 13.1 ausgewertet und diskutiert. Die Analyse klärt Fragen bezüglich der Messgenauigkeit und Leistungsfähigkeiten des Beschleunigungssensors ADcmXL3021. Es folgt die Diskussion der Messergebnisse mit einem abschließenden Fazit.

13.3.1 Auswertung Messung 1

Die Messergebnisse aus der ersten Messung zeigt Abbildung 13.3. Neben den gemessenen Beschleunigungen für alle Frequenzen sind oberen Frequenzlimits für $f_s = 1718,75\ Hz; f_s = 3437,5\ Hz$ markiert.

Die Abbildung 13.3 zeigt eine gute Übereinstimmung der gemessen Beschleunigungen bei den verschiedenen Abtastraten. Beschleunigungsspitzen der Schwingprüfanlage für gleiche Frequenzen sind in allen Messung mit verschiedenen Abtastraten erkennbar. Ab einer Frequenz von $f = 5\ Hz$ messen die Abtastraten $f_s = 27,5\ kHz; f_s = 1718,75Hz; f_s = 3437,5\ Hz$ nicht die exakt gleiche Beschleunigung, aber Beschleunigungen mit geringen Abweichungen zueinander. Mit $f = 100\ Hz$ weichen die mit $f_s = 220\ kHz$ gemessenen nur noch geringfügig von den anderen Beschleunigungen ab. Der Anstieg in der Beschleunigung ab $f = 5\ Hz$ ist auch für $f_s = 220\ kHz$ erkennbar, weicht aber deutlich ab. Abweichungen in der Beschleunigung betragen für die meisten Frequenzen $a =\approx 0,3\ g$. Besonders sticht hier die Beschleunigungsspitze bei $f = 600\ Hz$ hervor. An dieser Frequenz wird mit allen Abtastraten eine Beschleunigung von $a \approx 6,2\ g$ gemessen. Hiervon weicht nur die Beschleunigung gemessen mit $f_s = 1718,75\ Hz$ ab. Sie liegt mit $a = 5\ g$ deutlich unter den anderen Beschleunigungen. Ab dieser Frequenz, bis zum Erreichen von $f_{\text{cutoff, high}}$ für $f_s = 1718,75\ Hz$, weist die Messung mit $f_s = 1718,75\ Hz$ geringere Beschleunigunngen als die anderen Abtastraten auf. Ein ähnliches Verhalten lässt sich nicht für $f_s = 3437,5\ Hz$, wo $f_{\text{cutoff, high}}$ auch im Laufe der Messungen überschritten wird, beobachten. Bis zum Überschreiten von $f_{\text{cutoff, high}}$ sind die Abweichungen in der gemessenen Beschleunigung geringfügig. Für die höheren Frequenzen $f > 1\ kHz$ sind die gemessenen Beschleunigungen für $f_s = 220\ kHz; f_s = 27,5\ kHz$ fast identisch.

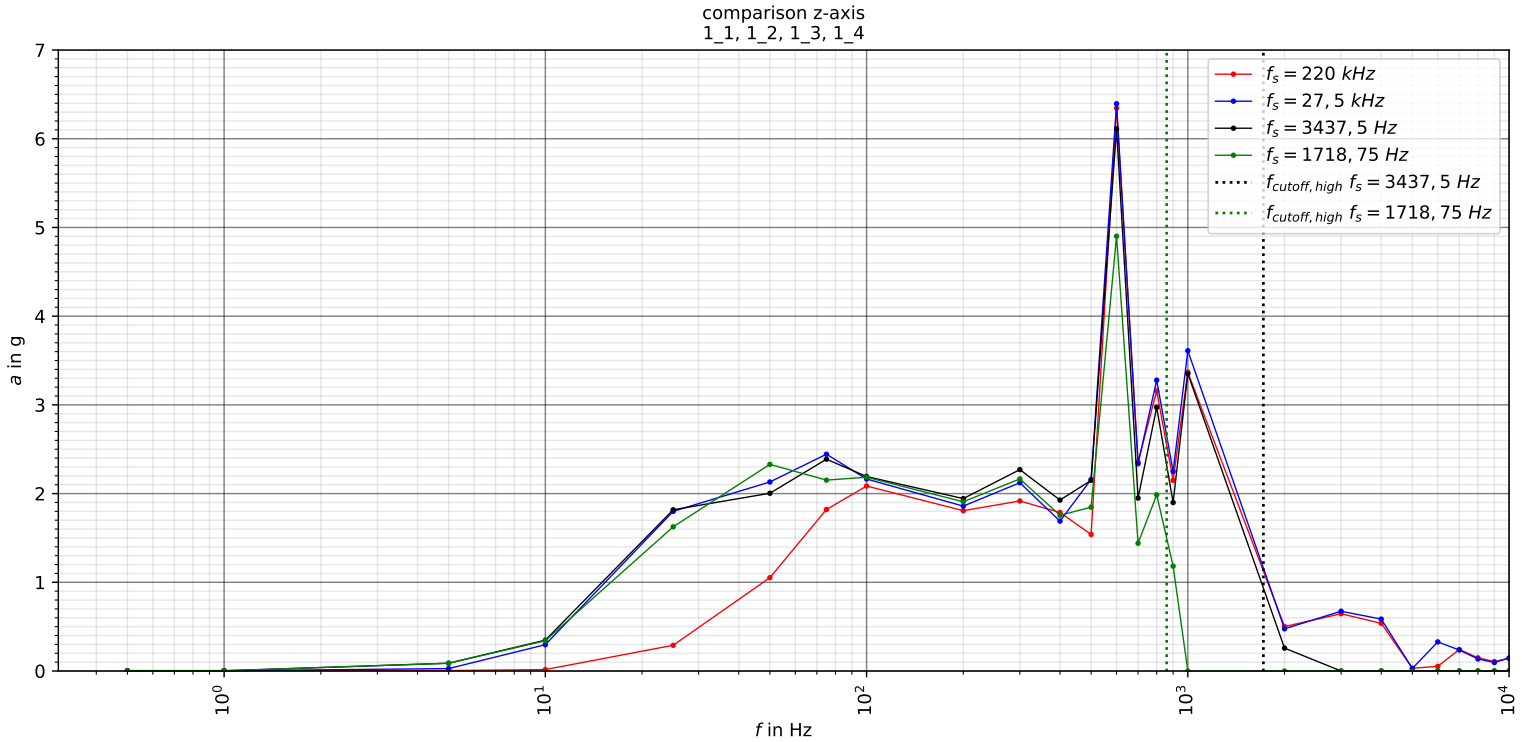


Abbildung 13.3: Vergleich der gemessenen Beschleunigungen in Messung 1 für die Abtastraten $f_s = 220\ kHz; f_s = 27,5\ kHz; f_s = 1718,75\ Hz; f_s = 3437,5\ Hz$

Mit Hilfe der Messwerten aus Messung 1 lässt sich auch das Verhalten der gemessenen Abtastraten für das untere Frequenzlimit $f_{\text{cutoff, low}}$ analysieren. Dafür werden die Messergbnisse aller Abtastraten in Messung 1 für die Frequenzen $f = 0,5\ Hz; f = 1\ Hz; f = 5\ Hz; f = 50\ Hz$ näher betrachtet. Die gewählten Frequenzen sind diejenigen Frequenzen, die den unteren Frequenzlimits der Abtastraten am nächsten sind. In Abbildung 13.4 ist erkennbar, dass mit einer niedrigen Abtastrate von $f_s = 1718,75\ Hz$ das Messen bis an die untere Grenze möglich ist. Eine Beschleunigungsspitze bei $f = 0,5\ Hz$ ist zu erkennen. Für $f_s = 3437,5\ Hz$ ist die Beschleunigungsspitze ebenfalls erkennbar. Aus den beiden höheren Abtastraten $f_s = 220\ kHz; f_s = 27,5\ kHz$ lässt sich keine Beschleunigungsspitze erkennen. Die Beschleunigungen liegen im Verhältnis deutlich höher. Für $f = 1\ Hz$ in Abbildung 13.5 gelten dieselben, gerade beschriebenen Beobachtungen bezüglich der Spitzen. Auffällig ist aber, dass $f_s = 220\ kHz; f_s = 27,5\ kHz$ keine Veränderung in den ersten beiden Werten zeigen. Dass die Werte sich nicht verändern setzt sich in Abbildung 13.6 für $f_s = 220\ kHz$ fort. Für $f_s = 27,5\ kHz$ ist die Spitze ab $f = 5\ Hz$ auch erkennbar, dabei aber mit einer Zuordnung zu $f = 6\ Hz$. Vergleichbare Beschleunigungen zwischen allen Abtastraten lassen sich erst in Abbildung 13.7 erkennen. In dieser Abbildung wird auch deutlich, dass mit einer geringeren Abtastrate die Beschleunigungsspitzen schmaler im Graphen werden.

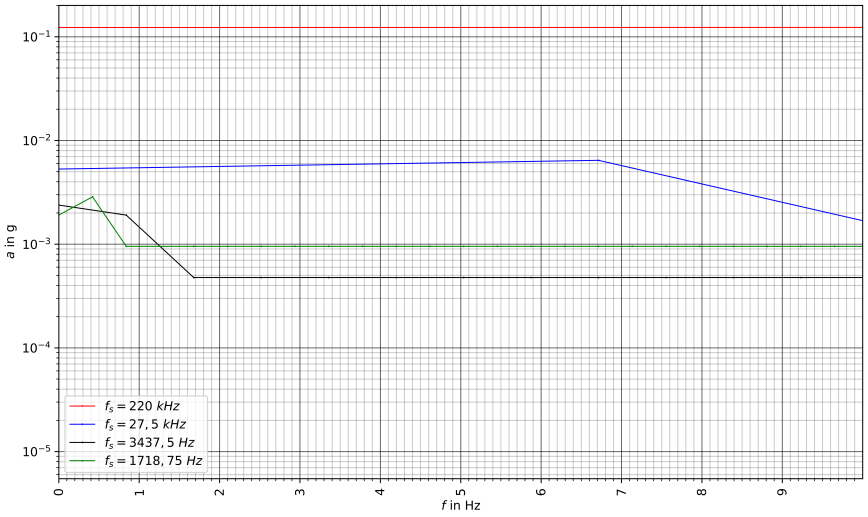


Abbildung 13.4: Gemessene Beschleunigung für alle vier Abtastraten in Messung 1 bei $f = 0,5\ Hz$

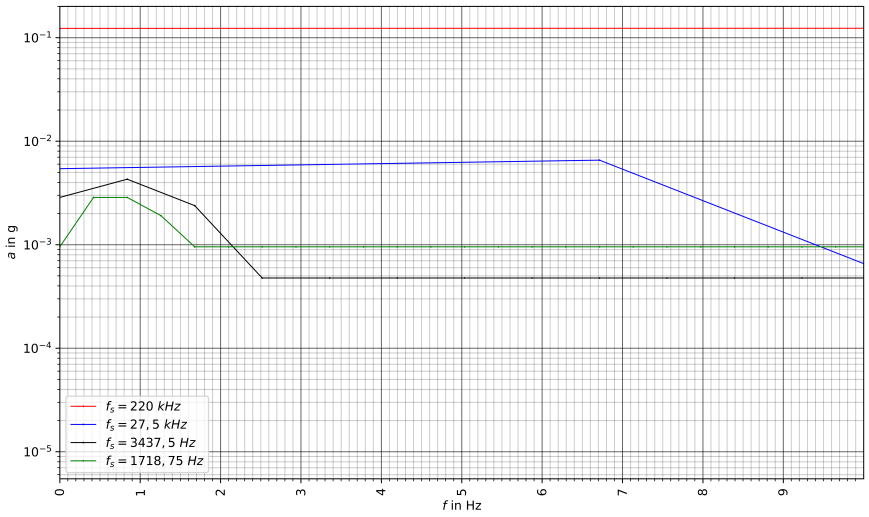


Abbildung 13.5: Gemessene Beschleunigung für alle vier Abtastraten in Messung 1 bei $f = 1\ Hz$

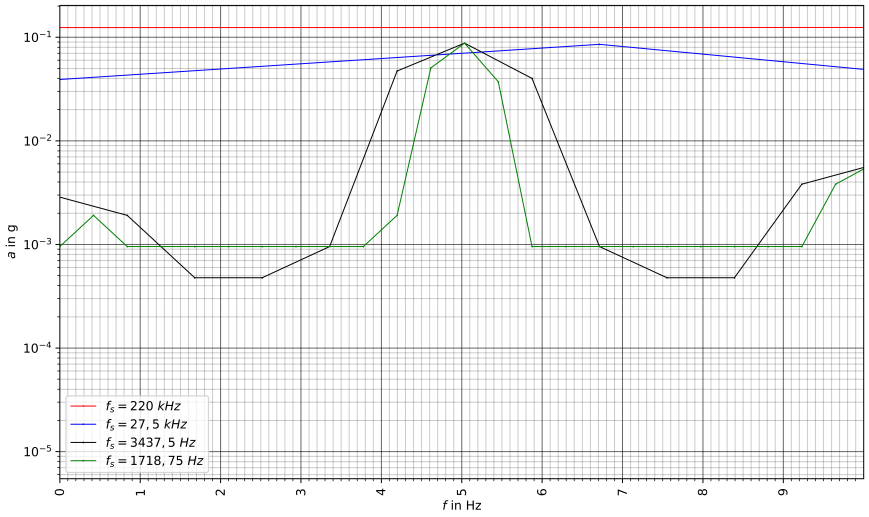


Abbildung 13.6: Gemessene Beschleunigung für alle vier Abtastraten in Messung 1 bei $f = 5\ Hz$

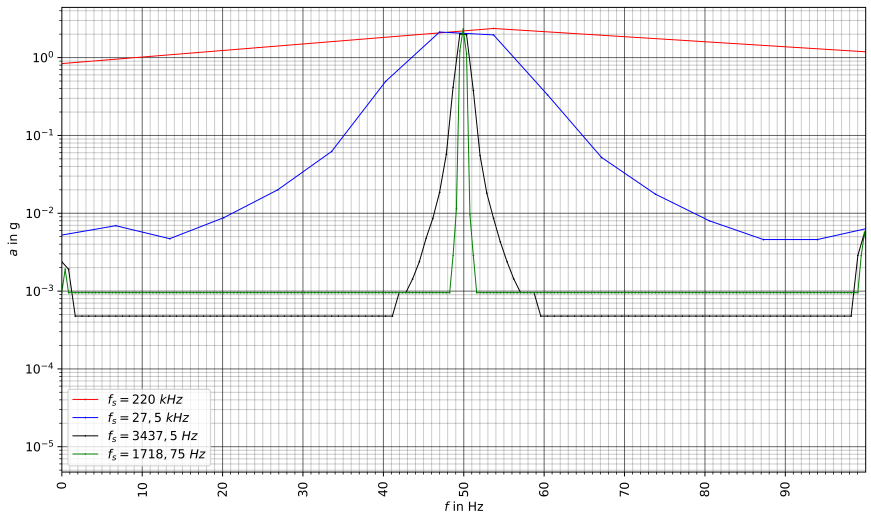


Abbildung 13.7: Gemessene Beschleunigung für alle vier Abtastraten in Messung 1 bei $f = 50\ Hz$

13.3.2 Auswertung Messung 2 und 3

Die Messungen 2 und 3 lassen sich zusammenfassen, da hier für $f = 100\ Hz; f = 1000\ Hz; f = 4600\ Hz$ in beiden Messungen dieselben Parameter gelten. In Abbildung 13.8 sind die gemessenen Beschleunigungen von dem Beschleunigungssensor ADcmXL3021 und dem Referenzsensor gegenübergestellt. Die gemessene Beschleunigung des Referenzsensors ist der Spitzenwert der Beschleunigung geteilt durch die Sensitivität des Sensors. Für eine Untersuchung der Linearität wird eine lineare Regression für alle Messungen berechnet. Es fällt sofort auf, dass bei allen drei gemessenen Frequenzen die im MFFT-Modus gemessene Beschleunigung von den restlichen Messung bei gleichen Paramtern abweicht. Die anderen Messungen messen immer ähnliche Beschleunigungen. Aus dieser Beobachtung fällt nur die Messung für $f = 4600\ Hz$ im MTC-Modus des ADcmXL3021 raus (adcmxl3021_3_3). Alle Messungen weisen einen linearen Anstieg in der gemessenen Beschleunigung auf. Die einzelnen Beschleunigungen liegen auf der Regressionsgerade mit Ausnahme der Messung 2_2. Die gemessen Beschleunigungen weisen hier einen größere Streuung auf. Hier zeigen sich die Abweichungen von der Linearität in den Messungen beider Beschleunigungssensoren. Für den Referenzsensor fällt die Abweichung von der Regressionsgerade geringer aus. Im Vergleich der Regressionsgeraden ist auch erkennbar, dass die Steigungen zwar nicht gleich sind, ein linearer Anstieg in beiden Fällen gegeben ist.

Die Messwerte des ADcmXL3021 aus dem FFT-Modus lassen sich an die Messungen mit dem Referenzsensor und im MTC-Modus angleichen. Der Faktor beträgt $\approx \frac{2}{\pi}$. Besonders für Messung 2_1, 3_1 führt das Anwenden dieses Faktors zu, im Grunde genommen, identischen Beschleunigung für beide Sensoren. Die Verbesserung ist für Messung 2_2, 3_2 und 2_3, 3_3 nicht so ausgeprägt, aber vorhanden. In beiden Fällen nähert sich die Beschleunigung im FFT-Modus an die anderen Beschleunigungen an. Mit der Anpassung durch den Faktor sind die Steigungen der Regressionsgeraden beider Sensoren, innerhalb einer Messung, annähernd gleich.

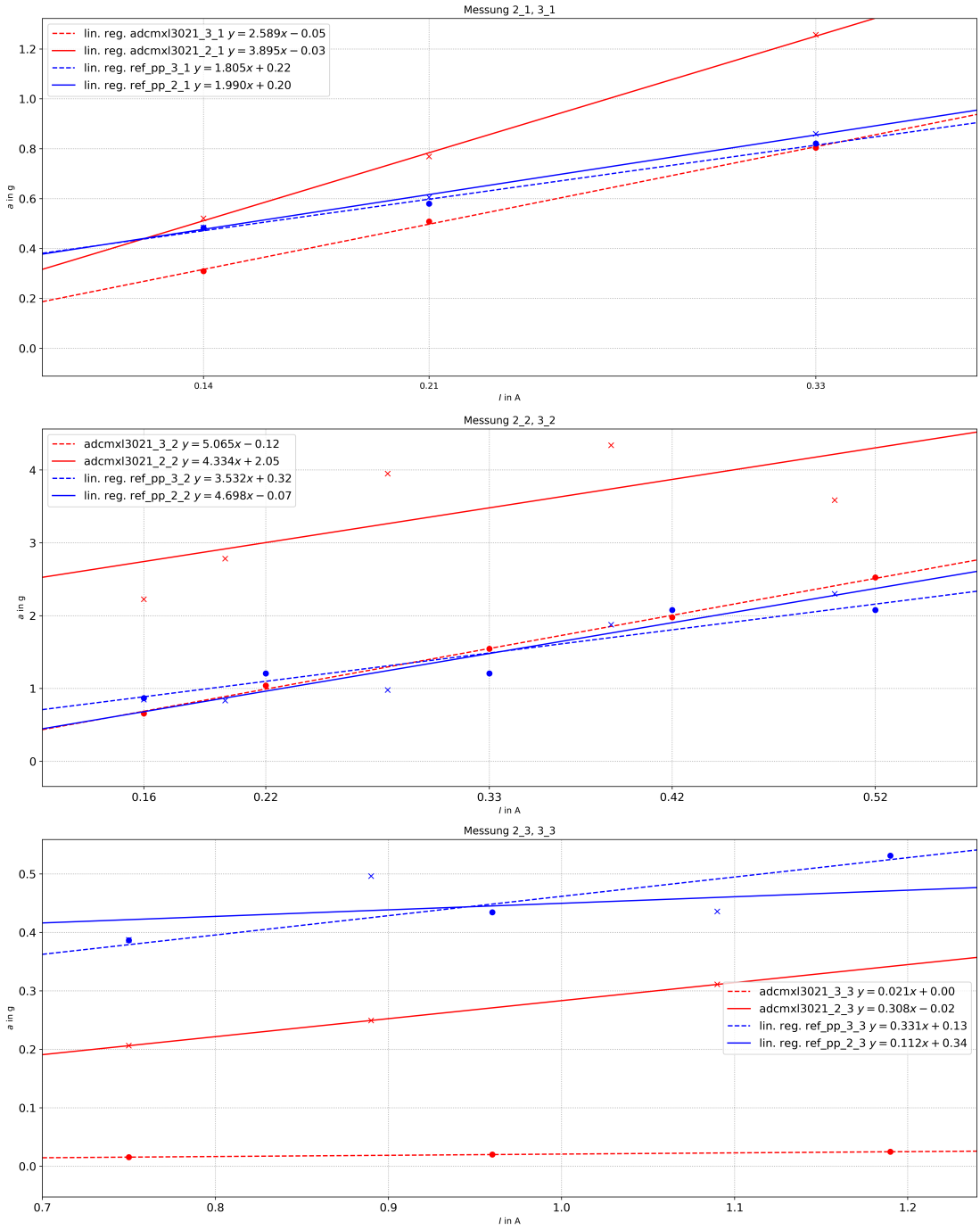


Abbildung 13.8: Linear Regressionen der Beschleunigungen des ADcmXL3021 und Referenzsensors für die Messungen 2_1, 2_2, 2_3, 3_1, 3_2 und 3_3

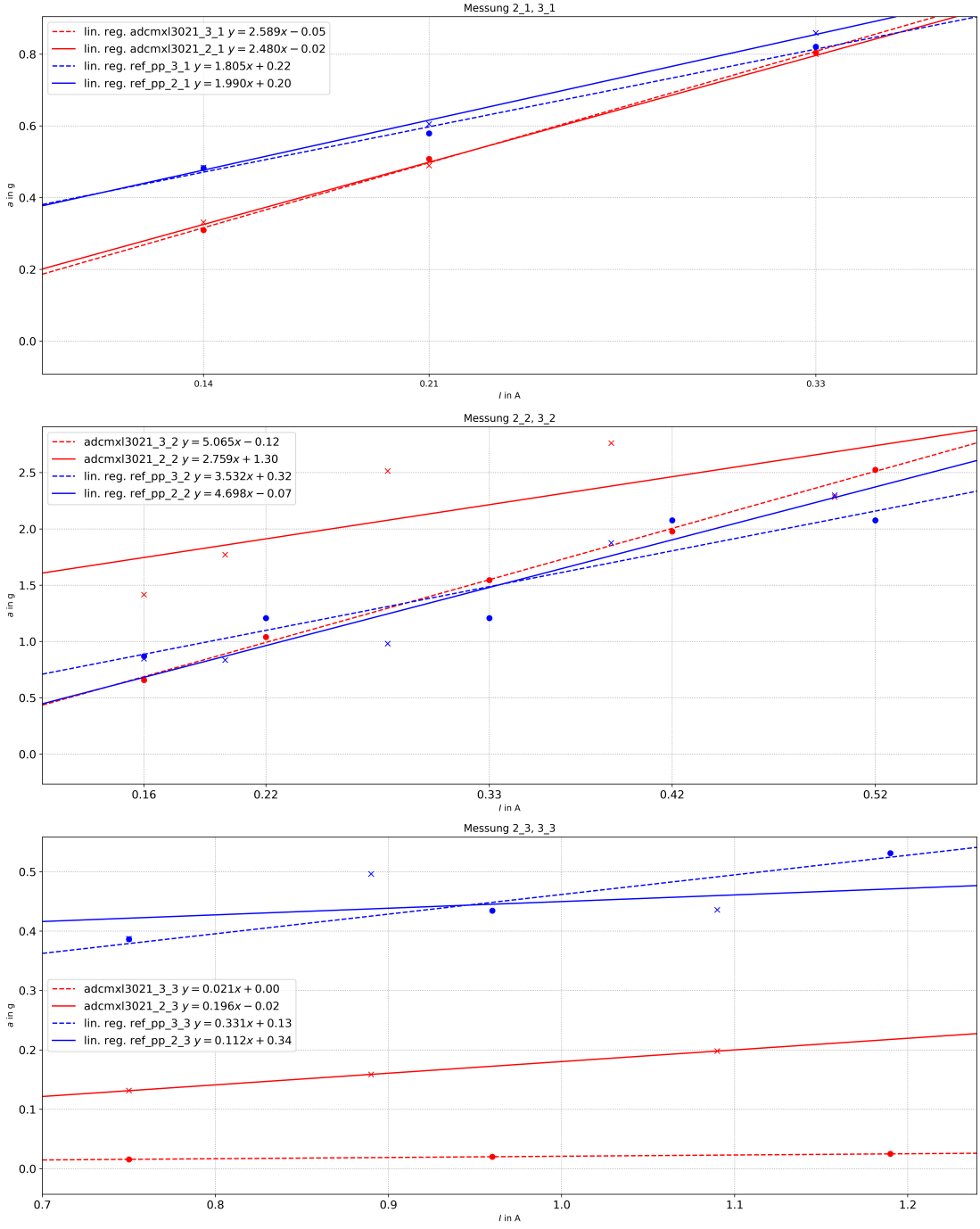


Abbildung 13.9: Linear Regressionen der Beschleunigungen des ADcmXL3021 und Referenzsensors für die Messungen 2_1, 2_2, 2_3, 3_1, 3_2 und 3_3 mit Faktor $\approx \frac{2}{\pi}$

13.3.3 Auswertung Messung 4

Für die Messung 4_1 in Abbildung 13.10 sind die beiden Frequenzen $f = 100\ Hz; f = 500\ Hz$ in dem Graphen erkennbar. Für beide Frequenzen ist eine Beschleunigungsspitze zu beobachten und befindet sich bei den erwarteten Frequenzen $f = 100\ Hz; f = 500\ Hz$. Es treten keine anderen Frequenzen in deutlichen Spitzen auf. Auffällig ist, dass für $f = 500\ Hz$ die x-Achse eine größere Beschleunigung misst als die z-Achse, in die eigentlich beschleunigt wird. Dies tritt aber nur hier auf. Die Ergebnisse der Messung 4_2 zeigt Abbildung 13.11. Auch hier tritt die Spitze in der Beschleunigung bei der erwarteten Frequenz $f = 250\ Hz$ auf. Die Frequenz der Schwingprüfanlage ist identisch zu der Frequenz in Messung 5_1, aber mit einer geringeren Amplitude. Dies spiegelt sich auch in einer kleineren gemessen Beschleunigung wider.

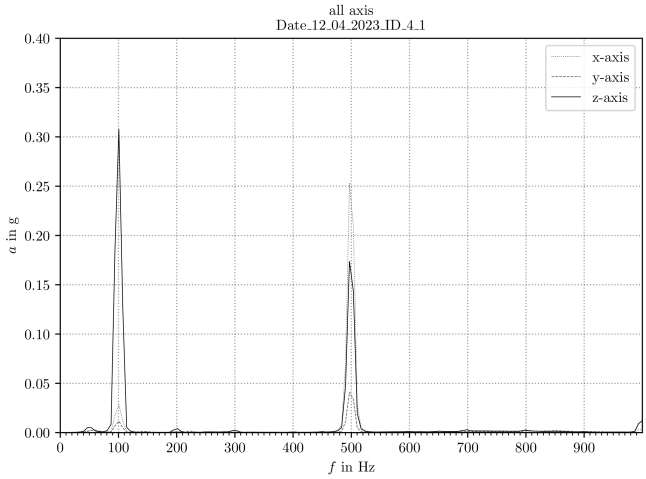


Abbildung 13.10: Gemessene Amplituden im relevanten Frequenzbereich für Messung 4_1

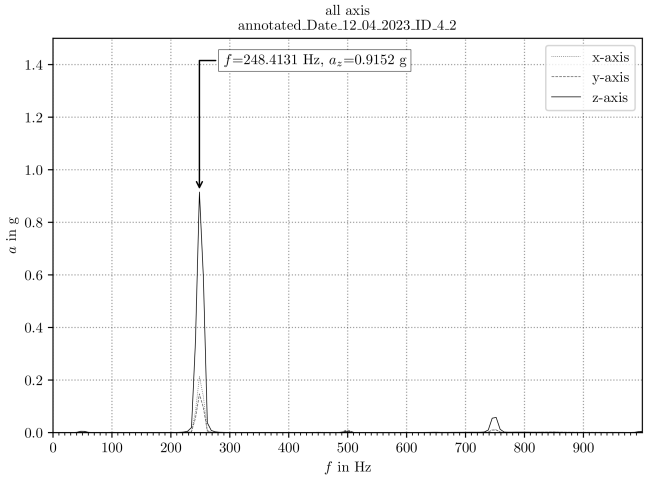


Abbildung 13.11: Gemessene Amplituden im relevanten Frequenzbereich für Messung 4_2

13.3.4 Auswertung Messung 5

Die gemessene Amplitude und Frequenz für alle hintereinander durchgeführte Messungen in Messung 5 zeigt Tabelle 13.2. Bei allen Messungen wird die gemessene Beschleunigungspitze einem erwarten Bin für $f = 248,4131\ Hz$ zugeordnet. Die gemessenen Beschleunigungen sind nicht immer gleich und weichen im geringen Maße voneinander ab. Diese geringe Abweichung spiegelt sich auch in einer Standardabweichung von $\sigma = 0,0167$ bei einem Mittelwert von $\overline{a} = 3,109\ g$ wider.

Tabelle 13.2: Gemessene Frequenzen und Beschleunigungen für Messung 5

ID	f in Hz	a in g	ID	f in Hz	a in g
5_1_1	248,4131	3,09292	5_1_9	248,4131	3,134
5_1_2	248,4131	3,081	5_1_10	248,4131	3,1214
5_1_3	248,4131	3,0981	5_1_11	248,4131	3,1255
5_1_4	248,4131	3,1013	5_1_12	248,4131	3,0918
5_1_5	248,4131	3,1139	5_1_13	248,4131	3,0981
5_1_6	248,4131	3,1213	5_1_14	248,4131	3,096
5_1_7	248,4131	3,1202	5_1_15	248,4131	3,0992
5_1_8	248,4131	3,135			

13.3.5 Auswertung Messung 6

Die Ergebnisse aus der Messung 6 sind in Tabelle 13.3 dargestellt. Für die Messungen bei einer gleichen Frequenz von $f = 250\ Hz$ wird die Beschleunigungspitze auch wieder dem erwarteten Bin zugeordnet. Es zeigen sich gegenüber Messung 5 größere Abweichungen in den gemessenen Beschleunigungen für $n_{\overline{FFT}} = 8, 10$. Alle anderen Beschleunigung sind jedoch immer noch ähnlich im Wert mit einer Standardabweichung von $\sigma = 0, 145411575$ bei einem Mittelwert von $\overline{a} = 3, 03962\ g$. Im Vergleich gegenüber den Messwerten in Tabelle 13.2 werden gleiche Beschleunigungen gemessen, die sich alle um $3, 1\ g$ bewegen.

Tabelle 13.3: Gemessene Frequenzen und Beschleunigungen für Messung 6

ID	$n_{\overline{FFT}}$	f in Hz	a in g
6_1_1	1	248,4131	3,1037
6_1_2	2	248,4131	3,0932
6_1_3	3	248,4131	3,1047
6_1_4	4	248,4131	3,1079
6_1_5	5	248,4131	3,1149
6_1_6	6	248,4131	3,1110
6_1_7	7	248,4131	3,1137
6_1_8	8	248,4131	2,7254
6_1_9	9	248,4131	3,1142
6_1_10	10	248,4131	2,8075

13.3.6 Auswertung Messung 7

Eine Anregung der Schwingprüfanlage mit einem White Noise Signal führt zu den messbaren Beschleunigungen in Abbildung 13.12. Die Resonanzfrequenz der Schwingprüfanlage zwischen $f = 9\ kHz$ und $f = 10\ kHz$ ist sehr gut zu erkennen.

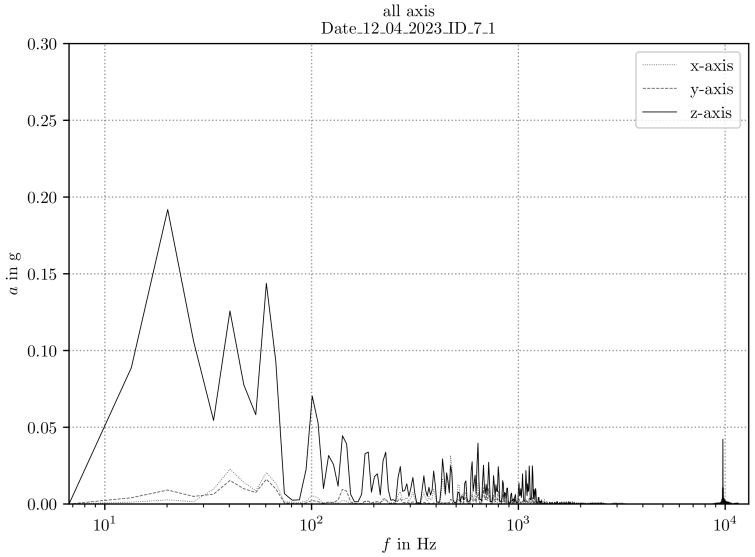


Abbildung 13.12: Gesamtes messbare Spektrum in Messung 7_1 mit $f_s = 27,5\ kHz$ mit Resonanz der Schwingprüfanlage

Wird die Schwingprüfanlage innerhalb eines Messzeitraums gestoppt, führt dies im MTC-Modus zu den gemessenen Beschleunigungen in Abbildung 13.13. Das Auftreten einer Beschleunigung lässt sich in allen Achsen erkennen. Der Beschleunigungssensor wird in der z-Achse beschleunigt, hier weist die Messung auch die größte Amplitude auf. Bei $t \approx 0,8\ s$ wird die Schwingprüfanlage gestoppt. In dem Verlauf der Beschleunigung ist noch ein Nachschwingen zu erkennen. Nach dem Start der Schwingprüfanlage bei $t \approx 2\ s$ wird dieselbe Amplitude, wie vor dem Stoppen, nach zwei Perioden wieder erreicht. Auch ist in den Messungen ein „schöner“ Sinus zu erkennen.

Berechnet man die FFT der Messung der z-Achse in Abbildung 13.13 ergibt dies die auftretenden Beschleunigungen über die Frequenz in Abbildung 13.14. Es ist eine deutliche Spitze in der Beschleunigung bei $f = 50\ Hz$ zu erkennen. Um die Beschleunigungspitze treten im Verhältnis deutlich kleinere Frequenzen auf. Allerdings sind keine weiteren Frequenzspitzen zu erkennen.

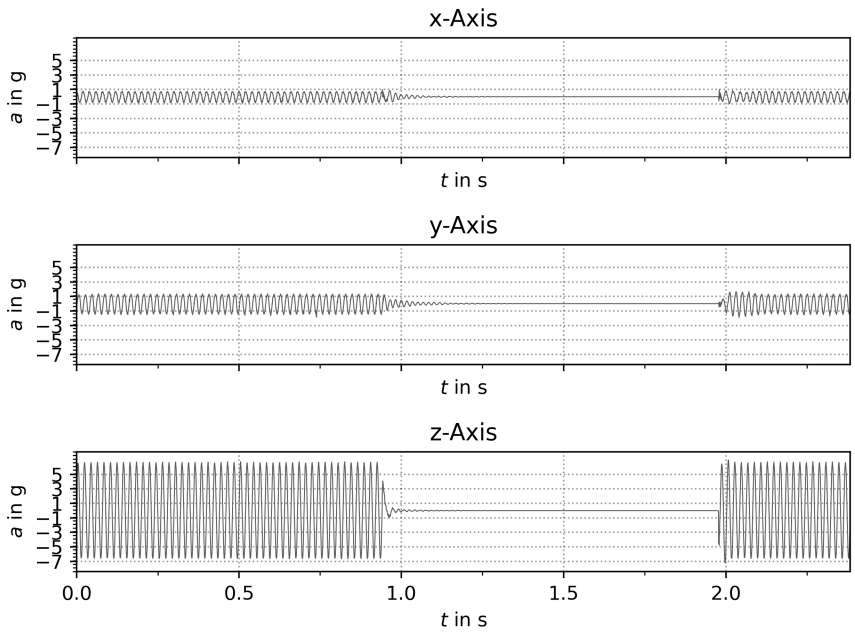


Abbildung 13.13: Messung 7_3

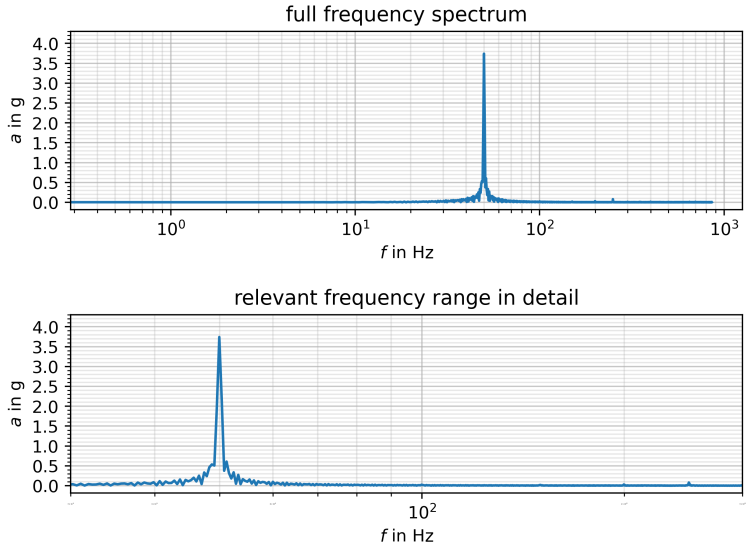


Abbildung 13.14: FFT der z-Achse für Messung 7_3 im gesamten Frequenzspektrum und im relevanten Frequenzbereich

13.3.7 Auswertung Messung 10

In einer letzten Messung werden die in einer Hydraulikpumpe auftretenden Beschleunigungen gemessen. Die Hydraulikpumpe ist das Modell 3521-048 der Firma Instron. Sie dient der Versorgung eines Ermüdungsprüfsystems. Für die Durchführung der Messung wird der Beschleunigungssensor an die Außenwand der Hydraulikpumpe, wie in Abbildung 13.15 zu sehen, geklebt.

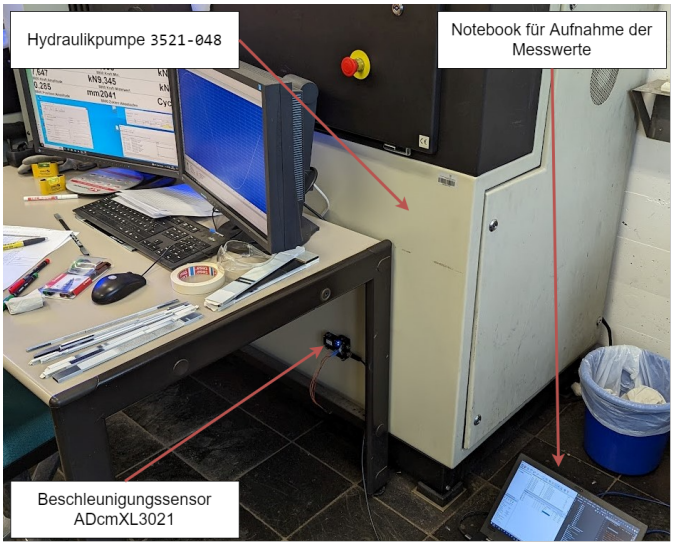


Abbildung 13.15: Messaufbau an der Hydraulikpumpe 3521-048

Die in dieser Messung auftretenden Beschleunigung sind in den Spektogrammen in Abbildung 13.16 (x), Abbildung 13.17 (y) und Abbildung 13.18 (z) dargestellt. Über den gesamten Messverlauf werden Beschleunigungen mit kleinen Amplituden im Bereich $4\ kHz \geq f \geq 8\ kHz$ gemessen. Im Verlauf der Messung wird einmal auf das Gehäuse der Hydraulikpumpe von außen geschlagen. Diese Spitze in der Beschleunigung ist vor allem in Abbildung 13.18 bei Minute Sieben zu erkennen. Die Amplitude der Beschleunigung ist deutlich höher als die der restlichen gemessenen Beschleunigungen. Auffällig ist, dass sich die höhere gemessene Beschleunigung auf das gesamte Spektrum auswirkt. Es werden für alle Frequenzen im Verhältnis extrem hohe Beschleunigungen gemessen. Dies zeigt sich auch wieder in Abbildung 13.19.

Der Schlag mit einer niedrigen Frequenz wird in den niedrigen Frequenzbins als größte Amplitude gemessen. Für die z-Achse ergibt sich ein insgesamt hohes Level an gemessenen Beschleunigungen, welches sich nicht in den anderen Achsen widerspiegelt. Die y- und

z-Achse zeigen aber weiterhin Spitzen an den selben Stellen im Spektrum. Die x-Achse hingegen zeigt über das gesamte Spektrum das Auftreten von Beschleunigung $> 1\ g$ für den gesamten Messzeitraum.

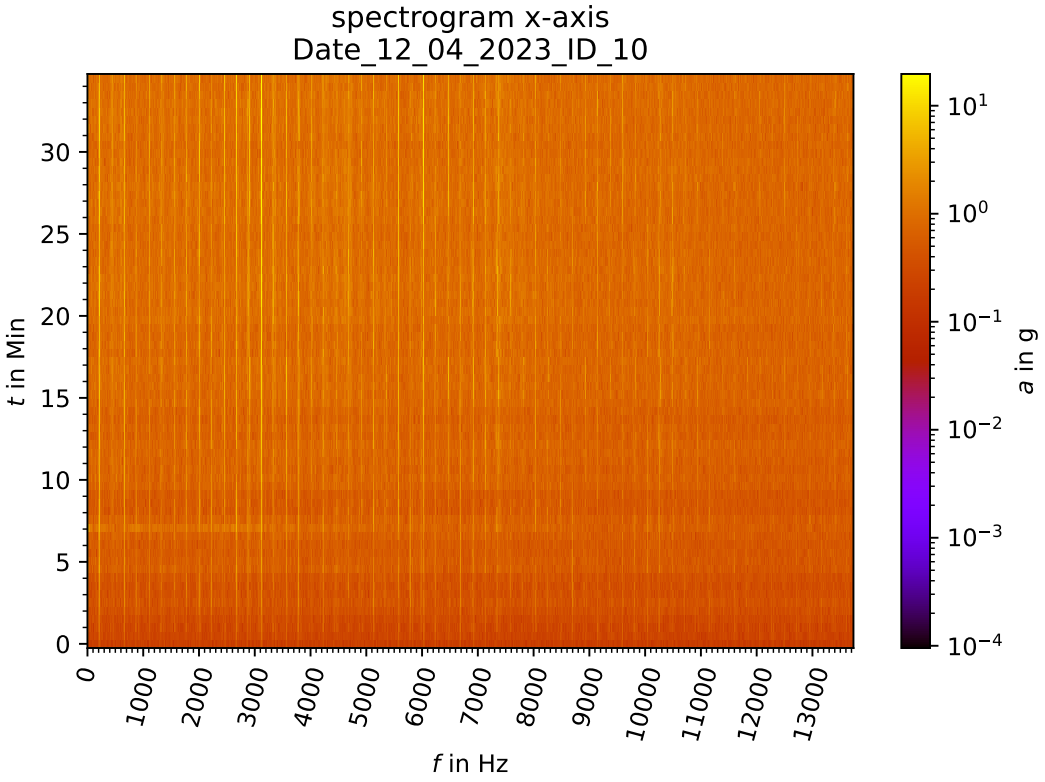


Abbildung 13.16: Spektogramm x-Achse Messung 10 (Normierung der Beschleunigung auf auftretenden maximalen und minimalen Beschleunigungen)

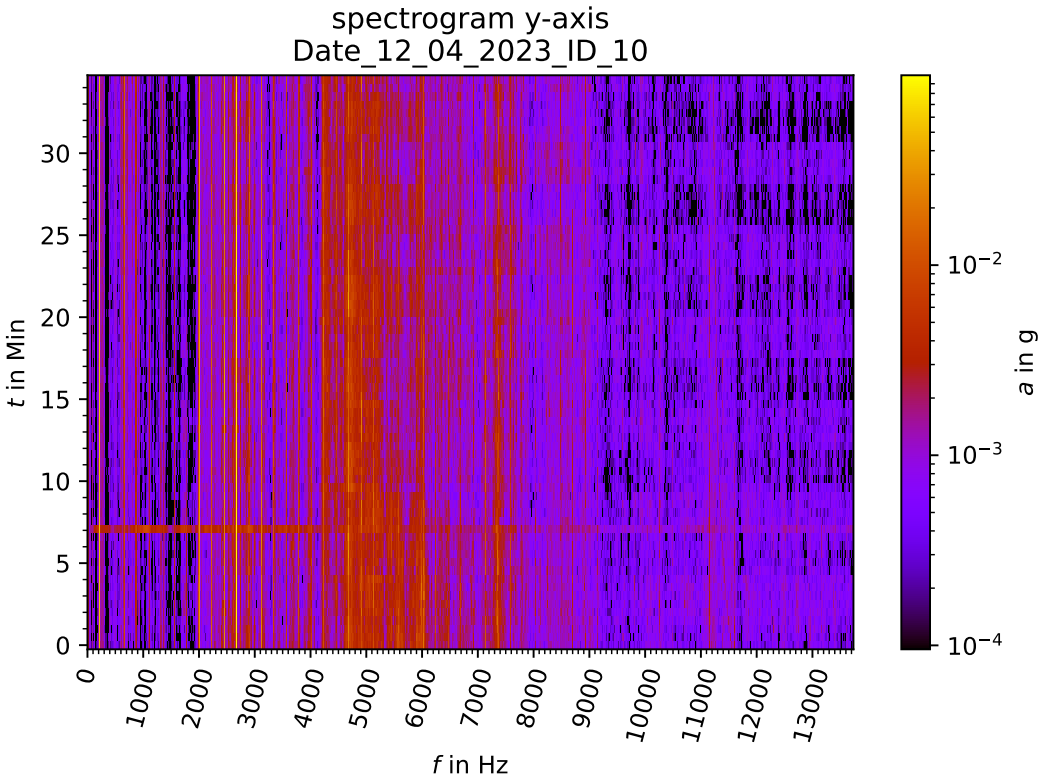


Abbildung 13.17: Spektogramm y-Achse Messung 10 (Normierung der Beschleunigung auf auftretenden maximalen und minimalen Beschleunigungen)

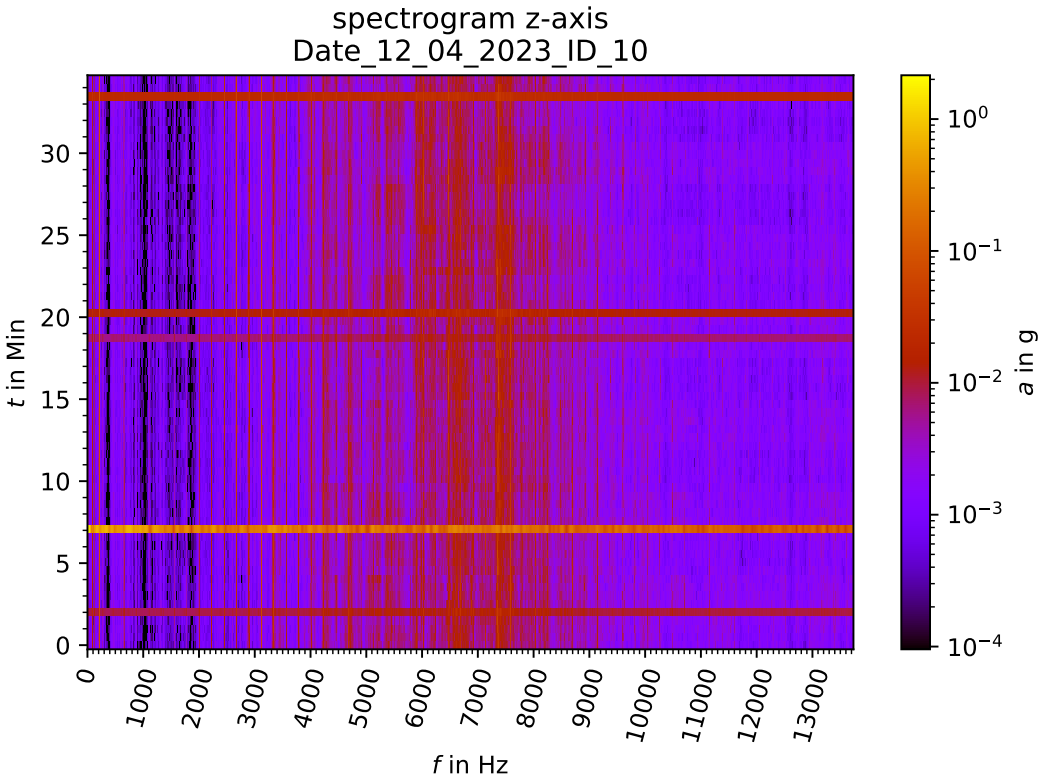


Abbildung 13.18: Spektogramm z-Achse Messung 10 (Normierung der Beschleunigung auf auftretenden maximalen und minimalen Beschleunigungen)

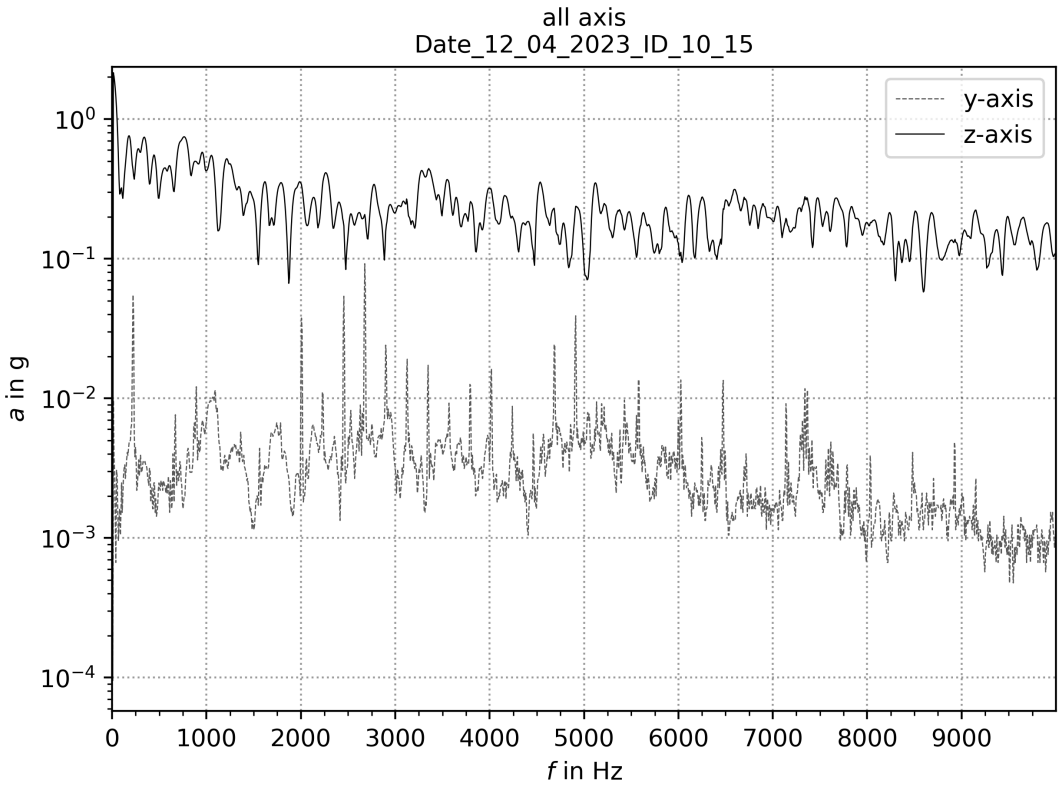


Abbildung 13.19: Gemessene Beschleunigungen über das gesamte Spektrum beim Schlag von außen auf das Gehäuse

13.4 Diskussion der Messergebnisse

Bei kontrollierten Bedingungen zeigt sich für den Beschleunigungssensor eine sehr gute Messgenauigkeit für Frequenzen. Über alle Messungen hinweg, wenn definierte Frequenzen eingestellt werden, lassen sich diese auch in den Auswertungen wiederfinden. In Tabelle 13.4 sind für Messungen mit einzeln auftretenden Frequenzen ihre Binzuordnung zusammengefasst. Es ist zu erkennen, dass eine Zuordung immer zu dem Bin passiert, der am nächsten an der eingestellten Frequenz liegt. Durch die Binweiten ist nie eine genaue Zuordnung der Frequenz möglich. Die genauen Zuordnungen der Frequnzen werden auch in Messung 1 deutlich, wo immer die Beschleunigung an der erwarteten Frequenz auftritt.

Tabelle 13.4: Frequenz Funktionsgenerator und Zuordung zu Frequenz durch Beschleunigungssensor

Messung	f in Hz	Zuordnung	Messung in Hz	f in Hz	Zuordnung
2_1	100	100,708	6_1	250	248,4131
2_2	1000	1000,366	5_1	250	
2_3	4600	4598,99			

In dem gesamten Frequenzspektrum, was durch die Anforderungsliste gefordert ist, ist ein Messen von Beschleunigungen möglich. Auch kleinste Beschleunigungen lassen sich aus den Messergebnissen herausfiltern. Am Besten wird dies mit der Resonanzfrequenz der Schwingprüfanlage deutlich. Hier wird auch für hohe Frequezen am Rande des geforderten Frequenzbereichs eine kleine Beschleunigung gemessen.

Die Messgenauigkeit für Beschleunigungen ist schwerer zu verifizieren. Vor allem die Messergebnisse in Abbildung 13.8 zeigen eine Diskrepanz in der gemessenen Beschleunigung von der Messung im MFFT-Modus zu den anderen Messungen. Die Diskrepanz kann aber einem Faktor verbessert werden, sodass hier plausible Ergebnisse erzielt werden. Ein Faktor von $\frac{2}{\pi}$ lässt darauf schließen, dass dieser durch die Berechung der FFT auftritt. Unterstützt wird diese Annahme durch den Fakt, dass die Messungen im MTC-Modus fast deckungsgleich mit den vom Referenzsensor gemessenen Beschleunigungen sind. Außerdem verbessert der Faktor für Messung 2_1, 3_1 die gemessenen Beschleunigungen so gut, dass hier die Abweichungen im Rahmen der Messungenauigkeit liegen. Auch gut zu erkennen ist, dass die Messergebnisse linear sind. Wie es bei einer konstanten Frequenz mit einer Erhöhung der Amplitude erwartet wird. Der Unterschied in der gemessenen

Beschleunigung wird mit dem Faktor verbessert. Messung 3_3, aufgenommen mit dem ADcmXL3021 fällt raus, da hier ein Fehler in der Konfiguration des ADcmXL3021 vorliegt. Der ADcmXL3021 wurde mit einer Abtastrate von $f_s = 1718,75\ Hz$ konfiguriert, die Frequenz in Messung 3_3 liegt aber bei $f = 4600\ Hz$. Aufgrund dessen kann die Beschleunigung bei dieser Frequenz nicht gemessen werden.

In Messung 1 wird aber deutlich, dass bei einem Betrieb im MFFT-Modus die gemessenen Beschleunigungen geringe Streuungen aufweisen. Die geringen Abweichungen liegen im Rahmen von Messungenauigkeiten. Außerdem ist durch die nicht vorhandene Regelung der Schwingprüfanlage nicht sichergestellt, dass immer die exakt selbe Beschleunigungsamplitude vorherrscht. Eine Abweichung von $a = 0,3\ g$ entpricht, auf den gesamten Messbereich des Beschleunigungssensors ($\pm 50\ g$), einer Abweichung von 0,3%.

Bei $f_s = 220\ kHz$ entspricht ein Messsbereich von $10\ Hz \geq f \geq 100\ Hz$ zwei Bins. Daraus resultiert der flachere Anstieg der Beschleunigung in diesem Bereich. Es werden mehr Frequenzen, die $a \approx 0\ g$ sind, in einem Bin zusammengefasst. Dadurch ergibt sich im Durchschnitt eine geringere Beschleunigung. Das führt auch zu den schmaleren Spitzen bei der Betrachtung des unteren Frequenzlimits. Mit $f > 1\ kHz$ sind die Auswirkungen der Binweiten nicht mehr so gravierend.

In der Betrachtung der Frequenzlimits wird deutlich, dass ein Messen bis an $f_{\text{cutoff, low}}$ möglich ist. Es sind keine größeren Beeinträchtigung durch das Annähern erkennbar. Eine vergleichbar präzise Aussage ist für das untere Frequenzlimit nicht möglich. Aus den Messungen geht hervor, dass für niedrige Abtastraten auch ein Messen von niedrigen Frequenzen möglich ist. Für höhere Abtastraten ist es, durch die Binweite, nicht möglich die Spitzen zu erkennen. Außerdem ist es für $f_s = 220\ kHz; f_s = 27,5\ kHz$ sehr auffällig, dass die ersten beiden Bins dieselben Werte besitzen. Wenn hier die ersten Bin sinnvolle Werte beinhalten würden, dann sollten die gemessenen Beschleunigungen unter den mit niedrigen Abtastraten gemessenen Beschleunigungen liegen.

Die Betrachtung der gemessenen Beschleunigungen in Messung 5_1 und 6_1 zeigt eine sehr gute Messwiederholgenauigkeit für die gemessene Beschleunigung. Auch hier liegen die Ergebnisse im Rahmen der Messungenauigkeit und weisen geringe Standardabweichungen auf. Die Genauigkeit in der gemessenen Beschleunigung in Mesung 6_1 zeigt auch, dass das Erhöhen von $n_{\overline{FFT}}$ keine Auswirkung auf die gemessene Beschleunigung hat.

Die Auswirkungen durch den Messaufbau können auch als gering angenommen werden. In Messung 7_3 sind kleine bis gar keine Auswirkungen erkennbar. Die Beschleunigung weist eine Sinuskurve auf. Außerdem ist nicht erkennbar, dass der Adapter signifikaten Auswirkungen auf die Messung hat. Es ist ein Abklingen in der Amplitude der Beschleunigung erkennbar, aber dies lässt sich eher als das Abklingen der Schwingprüfanlage interpretieren. Die einzige erkennbare theoretische Auswirkung ist die Resonanzfrequenz der Schwingprüfanlage bei $f \approx 9500\ Hz$.

Die Messung an der Hydraulikpumpe weist gute Messergebnisse auf. Es sind kleinste Beschleunigungen auch für hohe Frequenzen messbar. Das konstante Auftreten von Beschleunigungen im selben Frequenzbereich lässt darauf schließen, dass diese durch die Pumpe hervorgerufen werden. Eine Verifizierung dieser Annahme ist leider nicht möglich da die Drehzahl der Pumpe nicht aus den Datenblättern hervorgeht. Auch der Schlag, der mit einer geringen Frequenz auftritt, ist sehr gut zu erkennen. Er hebt sich deutlich von den restlichen Beschleunigungen ab. Der Schlag kann als ein Dirac-Impuls bei der Aufnahme der Messwerte angenommen werden. In einem Dirac-Impuls sind alle Frequenzen vertreten, was das Messen von im Verhältnis höheren Beschleunigungen, im gesamten Spektrum erklärt. In einem großen Frequenzbereich lassen sich auch Beschleunigungsspitzen mit niedrigen Frequenzen erkennen.

Eine Beeinträchtigung der Messergebnisse ergibt sich durch das Auftreten von Messfehlern des Beschleunigungssensors. In Abbildung 13.16 weisen die gemessenen Beschleunigungen keine Korrelation zu den Beschleunigungen in den anderen Achsen auf. Die gemessenen Beschleunigungen sind unrealistisch in ihrem periodischen Auftreten über das gesamte Spektrum mit hohen Amplituden.

Insgesamt weist der Beschleunigungssensor ADcmXL3021 gute Messergebnisse auf. Das Messen von auch kleinsten Beschleunigungen über das gesamte Spektrum sind möglich. Es zeigt sich auch, dass der Beschleunigungssensor die gemessenen Beschleunigungen sehr genau ihrer Frequenz zuordnet. Unerfreulich ist nur, dass Messfehler auftreten.

13.5 Empfehlungen für die Konfiguration aus den Messergebnissen

Aus den Messergebnissen können Empfehlungen für die Konfiguration des Beschleunigungssensors abgeleitet werden. Allgemein gilt natürlich, dass die gewählte Abtastrate so gewählt werden muss, dass sie den in einer Messung erwarteten Frequenzen entspricht. Bei der Auswahl der Abtastrate muss aber beachtet werden, dass eine gewisse Unsicherheit bezüglich der Messwerte der ersten beiden Bins erkennbar ist. Der erste Bin sollte immer als falsch angenommen und nicht beachtet werden. Die Betrachtung des zweiten Bin sollte mit Vorsicht passieren, kann aber für eine Auswertung von Messergebnissen verwendet werden. Für die gemessenen Beschleunigungen im FFT muss der Faktor von $\frac{2}{\pi}$ beachtet werden.

Der auftretende Fehler mit periodischen Signalen kann auf ein Fehler des Sensors bei der Autonullfunktion zurückgeführt werden. Der Fehler wird erst mit der Durchführung der Messungen erkannt und kann behoben werden. Das Nullen muss mehrmals durchgeführt werden bis sichergestellt ist, dass alle Achsen genullt sind.

Mit einer Abtastrate von $f_s = 27,5\ kHz$ lässt sich der komplette geforderte Messbereich von $1\ kHz \geq f \geq 10\ kHz$ für Frequenzen abdecken. Dies wird vor allem bei der Messung an der Hydraulikpumpe deutlich. Hier wurde der Schlag mit einer niedrigen Frequenz, aber auch hohe Frequenzen, ausgehend von der Hydraulikpumpe, gemessen. Der messbare Frequenzbereich verbessert sich sogar auf $bw\ Hz \geq f \geq 10\ kHz$. Dies geht aus den Messung für das untere Frequenzlimit hervor.

Außerdem zeigt das Auftreten der Resonanzfrequenz in den Messungen, dass auch Frequenzen mit geringen Beschleunigungen, am oberen Ende des geforderten Frequenzbereich messbar sind. Aus den Messungen lässt sich ebenfalls erkennen, dass die Verlängerung der Messdauer einer Messung, durch ein Erhöhen von $n_{\overline{FFT}}$, keine Auswirkungen auf die gemessenen Beschleunigungen und Frequenzen hat. Ein Erhöhen von $n_{\overline{FFT}}$ kann bedenkenklos durchgeführt werden. Auch ein Betrieb im AFFT-Modus ist ohne Probleme möglich, da hier der Senosr zuverlässig funktioniert und ein periodisches Berechnen einer FFT keine Auswrikung auf die Messungen hat.

14 Auswertung der Anforderungsliste

Nach der Abschlussmessung lässt sich erörtern, ob alle Anforderugen erfüllt wurden. Eine Übersicht welche Anforderungen erfüllt ✓, zum Teil erfüllt (✓) und nicht erfüllt ✗ wurden befindet sich in Tabelle 14.1, Tabelle 14.2 und Tabelle 14.3.

Insgesamt lässt sich festhalten, dass alle bis auf eine Pflichtanforderung erfüllt wurden. Es konnte nicht erfüllt werden, dass eine Integration in das bestehende Datenloggerssystem möglich ist. Ein Starten des Mikrocontrollers ist nur möglich, wenn die Steckerverbindung zwischen ihm und dem Beschleunigungssensor getrennt ist. Dadurch ist kein sicherer Betrieb in der Ferne möglich. Die Wunschanforderung, dass eine Befestigung an bestehenden Punkte möglich ist, konnte nicht durch das größere Gehäuse umgesetzt werden. Die mittleren beiden Befestigungspunkte haben jedoch das selbe Lochmaß, wie alte Gehäuse für Beschleunigungssensoren. Damit ist zumindest an dieser Stelle eine Befestigung an alten Punkten möglich, sofern das größere Gehäuse Platz findet.

Sobald der Steckerverbindungsfehler beseitigt ist, ist eine Integration in das Datenloggerssystem möglich, da alle Grundlagen dafür geschaffen sind. Die Abtastrate $f_s = 27,5\ kHz$, die den geforderten Messbereich am Besten abdeckt, hat einen messbaren Frequenzbereich von $0Hz \geq f \geq 13,75\ kHz$. Wovon der Frequenzbereich $6,71\ Hz \geq f \geq 10\ kHz$ zuverlässig und genau den erwartenden Bins zugeordnet wird. Aus den Messergebnissen geht hervor, dass sich der Beschleunigungssensor sich für den angestrebeten Einsatzbereich im Datenloggerssystem eignet. In allen Frequenzbereichen konnten definierte Beschleunigungen an definierten Frequenzen gemessen werden.

Eine Konfiguration des Beschleunigungssensors ist mit den entwickelten Python Scripten möglich. Das entwickelte System, für eine Unterscheidung von CAN-Nachrichten, ermöglicht es Messwerte und Konfigurationen mit dem Beschleunigungssensor zuverlässig auszutauschen. Hierfür erfüllt der μC seine Funktion als Brücke einwandfrei. Für eine gute mechanische Kopplung und exakte Ausrichtung, sind alle Komponenten mit dem Gehäuse verschraubt. Hier und an anderen Stellen, wie bei den Platinen, konnte darauf

geachtet werden, dass das System möglichst modular ist. Die Verwendung der ersten Platine und des Gehäuses, ist für andere Sensoren möglich.

Tabelle 14.1: Auswertung der Anforderungsliste für die Kategorie μC

ID	P/ W	Anforderung	Erfüllt?
1.1	P	Kommunikation Datenlogger $\leftrightarrow$ Sensor über CAN	✓
1.2	P	Konfiguration Sensor über CAN	✓
1.3	W	Programmierung $\boldsymbol{\mu}$C über CAN	(✓)
1.4	P	Darstellen des Betriebszustandes mit LEDs	✓
1.5	P	Anfordern und Versenden der Messwerte	✓
1.6	P	Bereitstellen einer SPI Schnittstelle	✓
1.7	P	Bereitstellen einer CAN Schnittstelle	✓
1.8	P	Integration in bestehendes Datenloggerssystem	✗

Tabelle 14.2: Auswertung der Anforderungsliste für die Kategorie Beschleunigungssensor

ID	P/ W	Anforderung	Erfüllt?
2.1	P	mindestens messbarer Frequenzbereich 1 $kHz \geq f \geq 10\ kHz$	✓
2.2	P	Starten des Beschleunigungssensors mit gleicher Konfiguration	✓
2.3	W	Verwendung von Bandpassfiltern	(✓)
2.4	W	Auslesen des Temperatursensors	(✓)
2.5	W	Kalibrieren des Sensors	✓
2.6	W	Überprüfung Konfiguration Beschleunigungssensor von außen	✓
2.7	P	Aufnahme von Messwerten im Frequenz- und Zeitbereich	✓
2.8	P	Konfiguration des Beschleunigungssensors von außen	✓
2.9	W	Konfiguration des Beschleunigungssensors aus der Ferne	(✓)
2.10	P	Übertragen der Messwerte an den $\boldsymbol{\mu}$C	✓
2.11	P	minimale Versorgungsspannung	✓
2.12	P	maximale Versorgungsspannung	✓
2.13	P	zu messende Eventlänge	✓

Tabelle 14.3: Auswertung der Anforderungsliste für die Kategorie Gehäuse

ID	P/ W	Anforderung	Erfüllt?
3.1	P	Vergießen der Komponenten in dem Gehäuse vorsehen	✓
3.2	P	Gehäuse gegen Wasser/ Staub schützen	(✓)
3.3	P	Steckerverbindung zwischen Beschleunigungssensor und μC sichern	✓
3.4	P	Loch LEDs in Gehäusedeckel vorsehen	✓
3.5	P	Befestigung an bestehenden Befestigungspunkten	✗
3.6	P	Beschleunigungssensor mit Schrauben sichern	✓
3.7	P	Ausrichten des Moduls in einer definierten Richtung	✓
3.8	W	möglichst kleines Gehäuse	✓
3.9	P	Integrieren aller Komponenten (μC, Spannungsversorgung, CAN-Schnittstelle, Beschleunigungssensor) in ein Gehäuse	✓
3.10	P	CAN-Schnittstelle nach außen	✓
3.11	P	Befestigungspunkte für Adapterplatte vorsehen	✓
3.12	P	gute mechanische Kopplung des Gehäuses mit Befestigung	✓
3.13	W	Integration in das bestehende System sollte möglichst reibungslos sein	(✓)
3.14	W	Erweiterbarkeit mit anderen Sensoren vorsehen	✓

15 Fazit und Ausblick

Die gegeben Zielsetzung dieser Arbeit war es den Beschleunigungssensor ADcmXL3021 von Analog Devices in ein Paket zu verpacken, dass eine Implementierung in ein Datenloggersystem ermöglicht. Um den Beschleunigungssensor verstehen und konfigurieren zu können, wurde zuerst Wissen über Funktiosnweise von Beschleunigungssensoren angeeinget. Außerdem wurde ein Verstädnis für die mathematischen Grundlagen für das Erechnen einer FFT gewonnen. Im selben Schritt wurden die beiden verwendeten Bussystem SPI und CAN verstanden. Der letzte vorbereitenden Schritt war da Verständis über die Möglichkeiten und Limitierungen des Mikrocontrollers.

Mit einem Verständis darüber, wie der CAN-Bus, der Beschleunigungssensor und Mikrocontroller funktionieren, wurden Programmabläufe entwickelt. In diesen Programmabläufen wurde festgelegt, wie ein Zusammenspiel von Mikrocontroller, Beschleunigungssensor und CAN-Bus aussehen muss. Die Limitierungen, gegeben durch den CAN-Bus und den Mikrocontroller, erforderten die Entwicklung eines eigenen Systems für die Unterscheidung von CAN-Nachrichten. Limitierung ergaben sich vorallem durch den begrenzen Speicherplatz auf dem Mikrocontroller. Es konnte ein System entwickelt werden mit dem es möglich ist CAN-Nachrichten zuverlässig zu unterscheiden und entsprechend des Inhalts bestimmte Programmabläufe durchführen zu können.

Damit der Mikrocontroller verwendet werden konnte, musste zuerst die Platinen entwickelt und gebaut werden. Nachdem dies abgeschlossen war, konnte der Mikrocontroller nach den entwickelten Programmabläufen programmiert werden. Dies umfasste die Implementierung von beiden Bussystemen und das Verarbeiten der Nachrichten von beiden Bussystemen. Außerdem ist es möglich die Baudrate von CAN im laufenden Betrieb zu verändern. In der Programmierung konnten die dabei gewonnen Kenntnisse angewendet werden. Im Laufe der Programmierung wurden erfolgreich Probleme mit dem ursprünglichen Konzepten und Hardware beseitigt.

Die gesamte Hardware wurde erfolgreich in einem Gehäuse untergebracht. Das Gehäuse ist dabei klein geblieben ohne die Einfachheit der Montage zu beeinträchtigen. Bei der

Konstruktion wurde darauf geachtet, dass das Gehäuse in der Zukunft gegen Staub und Wasser geschützt werden kann.

Leider konnte das Problem, dass die Verbindung zwischen Beschleunigungssensor und Mikrocontroller nach jedem Neustart des Mikrocontrollers getrennt werden muss, nicht beseitigt werden. Dies impliziert, dass das aktuelle System nicht in einem externen Datenloggersystem verwendet werden kann, da ein stabiler Betrieb nicht gewährleistet ist. Denoch eignet sich das entwickelt System dazu den Beschleunigungssensor auf seine Tauglichkeit für das angestrebete Einsatzgebiet zu Testen. Sobald die Fehler beseitigt sind, kann das entwickelte System in einem Datenloggerssystem eingesetzt werden.

Das Testen des Systems stellte den letzten Teil dieser Arbeit dar. In den Tests konnte ermittelt werden, dass der Beschleunigungssensor sich gut dafür eignet Beschleunigungen dem geforderten Frequenzbereich zu messen. Außerdem zeigen die Messungen, dass der Beschleunigungssensor eine sehr gute Genauigkeit bei der Zuordung von Frequenzen zu ihren entsprechenden Bins aufweist. Aus den Messungen konnten auch wichtige Erkenntisse darüber gewonnen werden, wie der Beschleunigungssensor sich für messbare Frequenzlimits und Abtastraten verhält. Die gewonnen Erkenntisse können für eine fundierte Wahl von Konfigurationsparamter für ein vorgesehenes Einsatzgebiet verwendet werden. Die gemessenen Beschleunigungen konnten mit dem Fakor verbessert werden. Das Anwenden so eines Faktors ist nicht zufriedenstellend.

In weiteren Messungen sollte der Faktor in den Beschleunigungsmessung im FFT-Modus näher untersucht und beseitigt werden. Außerdem sollte der auftretenden Fehler, der das Trennen der Verbindung zwischen Beschleunigungssensor und Mikrocontroller erfordert, beseitigt werden. Der Grund für diesen Fehler konnte leider nicht gefunden werden, hier sind weitere Untersuchungen notwendig. Ein weitere Aspekt, der in der Zukunft bearbeitet werden sollte, ist die Optimierung der Datenübertragung. Im Moment ist eine Datenübertragung zwar unter 1 s möglich, es ist aber wenig Puffer vorhanden. Ein letzter Aspekt der in der Zukunft betrachtet werden sollte ist die Messgenauigkeit der Beschleunigung. Weitere Messung sollten auf einem Versuchsaufbau durchgeführt werden bei dem die Beschleunigung geregelt wird. Wenn diese verbliebenen Fehler behoben und die letzten Fragen bezüglich der gemessenen Beschleunigung geklärt werden können, ist eine Integration in das Datenloggerssystem möglich.

Literaturverzeichnis

[1] Analog Devices: *ADcmXL3021 Wide Bandwidth, Low Noise, Triaxial Vibration Sensor.* – URL https://www.analog.com/media/en/technical-documentation/data-sheets/adcmxl3021.pdf. – Zugriffsdatum: 11.03.2023

[2] Bhushan, Bharat (Hrsg.): *Springer handbook of nanotechnology.* 2nd rev. & extended ed. Springer. – ISBN 978-3-540-29855-7

[3] Borgeest, Kai: *Elektronik in der Fahrzeugtechnik: Hardware, Software, Systeme und Projektmanagement.* 4., aktualisierte und erweiterte Auflage. Springer Vieweg (ATZ/MTZ-Fachbuch). – ISBN 978-3-658-23663-2

[4] CAN in Automation: *Controller Area Network Extra Long CAN XL).* – URL https://www.can-cia.org/can-knowledge/can/can-xl/. – Zugriffsdatum: 08.04.2023

[5] Cooley, James W. ; Tukey, John W.: *An Alogorithm for the Machine Calculations of Complex Fourier Series.* – URL https://web.stanford.edu/class/cme324/classics/cooley-tukey.pdf. – Zugriffsdatum: 05.04.2023

[6] Dadafshar, Majid: *ACCELEROMETER AND GYROSCOPES SENSORS: OPERATION, SENSING, AND APPLICATIONS.* – URL https://www.analog.com/media/en/technical-documentation/tech-articles/accelerometer-and-gyroscopes-sensors-operation-sensing-and-applications.pdf. – Zugriffsdatum: 07.04.2023

[7] Dhaker, Piyu: *Introduction to SPI Interface.* – URL https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf. – Zugriffsdatum: 04.04.2023

[8] ECS: *ECS-2520MVLC SMD MultiVolt Low Current Crystal Oscillator.* – URL https://ecsxtal.com/store/pdf/ECS-2520MVLC.pdf. – Zugriffsdatum: 19.03.2023

[9] Fernandez Bautista, Roberto ; Charras, Jean-Pierre ; Evans, Jon ; Hillbrand, Seth ; McInerney, Ian ; Pointhuber, Thomas ; Roszko, Mark ; Stambaugh, Wayne ; Wielgus, Mikolaj ; Wlostowski, Tomasz ; Young, Jeff: *KiCad.* – URL https://www.kicad.org/. – Zugriffsdatum: 19.03.2023

[10] Furlan, Nicola F. ; Matjaž, Guštin: *Downloads und Dokumentation.* – URL https://nik89.github.io/CanOverhead/. – Zugriffsdatum: 08.04.2023

[11] Guicking, Dieter: *Schwingungen.* Springer Fachmedien Wiesbaden. – URL http://link.springer.com/10.1007/978-3-658-14136-3. – ISBN 978-3-658-14135-6 978-3-658-14136-3

[12] Hartwich, Florian: *CAN with Flexible Data-Rate.* – URL https://www.can-cia.org/fileadmin/resources/documents/proceedings/2012_hartwich.pdf. – Zugriffsdatum: 08.04.2023

[13] Hesse, Stefan ; Schnell, Gerhard: *Sensoren für die Prozess- und Fabrikautomation: Funktion – Ausführung – Anwendung.* Springer Fachmedien Wiesbaden. – URL http://link.springer.com/10.1007/978-3-658-21173-8. – ISBN 978-3-658-21172-1 978-3-658-21173-8

[14] HMS Industrial Networks: *USB-to-CAN V2 - Aktives USB-Interface.* – URL https://www.ixxat.com/de/produkte/pc-interfaces-uebersicht/produkt-details/usb-to-can-v2-professional. – Zugriffsdatum: 27.04.2023

[15] Hottinger Brüel and Kjaer: *Vibration Test Solutions.* – URL https://www.bksv.com/-/media/literature/Product-Data/bu3105.ashx. – Zugriffsdatum: 26.03.2023

[16] Lin, Che-Hsin ; Li, Dongqing (Hrsg.): *Bulk Micromachining.* Springer US. – 164–173 S. – URL http://link.springer.com/10.1007/978-0-387-48998-8_138. – Zugriffsdatum: 07.04.2023. – ISBN 978-0-387-32468-5 978-0-387-48998-8

[17] Mamilla, Venkata R. ; Chakradhar, Kommuri.Sai: Micro Machining for Micro Electro Mechanical Systems (MEMS). 6, S. 1170–1177. – URL https://

linkinghub.elsevier.com/retrieve/pii/S2211812814005550. – Zugriffsdatum: 07.04.2023. – ISSN 22118128

[18] Meroth, Ansgar ; Sora, Petre: *Sensornetzwerke in Theorie und Praxis: Embedded Systems-Projekte erfolgreich realisieren.* Springer Fachmedien Wiesbaden. – URL https://link.springer.com/10.1007/978-3-658-31709-6. – ISBN 978-3-658-31708-9 978-3-658-31709-6

[19] Meyer, Martin: *Signalverarbeitung: Analoge und digitale Signale, Systeme und Filter.* Springer Fachmedien Wiesbaden. – URL https://link.springer.com/10.1007/978-3-658-32801-6. – ISBN 978-3-658-32800-9 978-3-658-32801-6

[20] Microchip Technology Inc.: *MPLAB Code Configurator.* – URL https://www.microchip.com/en-us/tools-resources/configure/mplab-code-configurator/melody. – Zugriffsdatum: 19.03.2023

[21] Microchip Technology Inc.: *PIC18F2480/2580/4480/4580 Data Sheet 28/40/44-Pin Enhanced Flash Microcontrollers with ECAN™ Technology, 10-Bit A/D and nanoWatt Technology.* – URL http://ww1.microchip.com/downloads/en/devicedoc/39637d.pdf. – Zugriffsdatum: 11.03.2023

[22] Microchip Technology Inc.: *PIC18F47Q10 Sending Data as a Master SPI Device with Multiple Slaves.* – URL https://www.microchip.com/en-us/tools-resources/configure/mplab-code-configurator/melody. – Zugriffsdatum: 11.03.2023

[23] Nexperia: *PESD1CAN-U CAN bus ESD protection diode.* – URL https://assets.nexperia.com/documents/data-sheet/PESD1CAN-U.pdf. – Zugriffsdatum: 15.03.2023

[24] NXP: *TJA1441 High-speed CAN transceiver.* – URL https://www.nxp.com/docs/en/data-sheet/TJA1441.pdf. – Zugriffsdatum: 19.03.2023

[25] PTC Inc.: *Creo Parametric.* – URL https://www.ptc.com/en/products/creo

[26] Richards, Pat: *Understanding Microchip's CAN Module Bit Timing.* – URL https://ww1.microchip.com/downloads/en/Appnotes/00754.pdf. – Zugriffsdatum: 26.03.2023

[27] Robert Bosch GmbH: *CAN Specification.* – URL http://esd.cs.ucr.edu/webres/can20.pdf. – Zugriffsdatum: 28.03.2023

[28] Robert Bosch GmbH: *CAN with Flexible Data-Rate Specification Version 1.0.* – URL https://web.archive.org/web/20151211125301/http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can_fd_spec.pdf. – Zugriffsdatum: 08.04.2023

[29] Schnell, Gerhard ; Wiedemann, Bernhard: *Bussysteme in der Automatisierungs- und Prozesstechnik: Grundlagen, Systeme und Anwendungen der industriellen Kommunikation.* 9. Springer Vieweg. – ISBN 978-3-658-23688-5

[30] Texas Instruments: *TLV767-Q1 1-A, 16-V Linear Voltage Regulator.* – URL https://www.ti.com/lit/ds/symlink/tlv767-q1.pdf?ts=1679244534046&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTLV767-Q1%252Fpart-details%252FTLV76733QWDRBRQ1. – Zugriffsdatum: 22.03.2023

[31] Thorne, Brian: *python-can.* – URL https://github.com/hardbyte/python-can. – Zugriffsdatum: 23.03.2023

[32] Tränkler, Hans-Rolf (Hrsg.) ; Reindl, Leo (Hrsg.): *Sensortechnik: Handbuch für Praxis und Wissenschaft.* Springer Berlin Heidelberg (VDI-Buch). – URL http://link.springer.com/10.1007/978-3-642-29942-1. – ISBN 978-3-642-29941-4 978-3-642-29942-1

[33] Ulrich, Helmut ; Ulrich, Stephan: *Laplace-Transformation, Diskrete Fourier-Transformation und z-Transformation: Grundlagen und Anwendungen zu Elektrotechnik, Informatik, Kommunikations- und Regelungstechnik.* Springer Fachmedien Wiesbaden. – URL https://link.springer.com/10.1007/978-3-658-31877-2. – ISBN 978-3-658-31876-5 978-3-658-31877-2

[34] Voss, Wilfried: *CAN Bus And SAE J1939 Bus Voltage.* – URL https://copperhilltech.com/blog/can-bus-and-sae-j1939-bus-voltage/. – Zugriffsdatum: 24.04.2023

A Anhang

A.1 Detaillierte Beschreibung des Beschleunigungssensors ADcmXL3021

In diesem Kapitel werden detaillierte Betrachtungen des Beschleunigungssensors ADcmXL3021 zusammengefasst.

A.1.1 Systemalarme

Die Systemalarme für die Spannungsversorgung und die Temperatur nutzen die selben Register. Es kann immer nur einer der beiden Werte als Alarmquelle verwendet werden. Die Alarmquelle wird im ALM_CTRL-Register ausgewählt. Im selben Register kann auch ausgewählt werden, ob der Alarm auslösen soll, wenn der Schwellwert unter- oder überschritten wird. Der Schwellwert wird in das ALM_S_MAG-Register geschrieben und richtet sich danach, ob er für die Spannung oder Temperatur gilt. Die Berechnung der Werte entspricht der Berechnung der Registerwerte von TEMP_OUT und SUPPLY_OUT. Beide Register lassen sich immer auslesen und aktualisieren ihre Werte mit jeder abgeschlossenen Messung. Der Wert des Registers SUPPLY_OUT lässt sich mit dem Faktor $LSB = 3,22\ mV$ umrechnen. Die Beziehung zwischen dem Registerwert von TEMP_OUT und der gemessenen Temperatur T_{sense} lautet

$$TEMP\_OUT = \frac{T_{\text{sense}} - 460°C}{-0,46\frac{°C}{LSB}} \tag{A.1}$$

Sie muss umgestellt werden um die gemessene Temperatur zu erhalten

$$T_{\text{sense}} = TEMP\_OUT \cdot \left(-0,46\frac{°C}{LSB}\right) + 460°C \tag{A.2}$$

Der Registerwert von TEMP_OUT ist im Zweierkomplement gegeben und muss entsprechend umgerechnet werden. Löst der Systemalarm aus, wird dies durch Setzen des entsprechenden Bits im DIAG_STAT-Register signalisiert.

A.1.2 Spektralalarme

Mehrere Register sind mit den Spektralalarmen verbunden. Die Spektralalarme lassen sich für jede Achse einzeln konfigurieren. Spektralalarme werden in dem ALM_CTRL-Register für jede Achse aktiviert. Alle diese Register sind in Tabelle A.3 zusammengefasst. Bei Dopplungen der Register für die x-, y- und z-Achse ist in dem Registernamen die Achse durch i ersetzt. Für die Spektralalarme lassen sich ein oberes $f_{\text{alarm, max}}$ und unteres $f_{\text{alarm, min}}$ Frequenzlimit in den Registern ALM_F_LOW und ALM_F_HIGH festlegen. Der Wert der Register ist von der Abtastrate des Beschleunigungssensors abhängig und wird als Frequenzbin fb im Bereich $0 \geq fb \geq 2047$ angegeben. Eine Berechnung von fb für $f_{\text{alarm, max}}$ und $f_{\text{alarm, min}}$ erfolgt in Abhängigkeit von bw in Tabelle 3.2.

$$fb = floor\left(\frac{f_{alarm,min/max}}{bw}\right) \tag{A.3}$$

Ein Spektralalarm löst aus, wenn in dem Frequenzbereich eine festgelegte Beschleunigung überschritten wird. Die Schwellwerte lassen sich in den Registern ALM_i_MAG1 a_{mag1} und ALM_i_MAG2 a_{mag2} festlegen. Es gilt

$$a_{\text{mag2}} \geq a_{\text{mag1}} \tag{A.4}$$

Die Berechnung erfolgt rückwärts zu der Berechnung von Beschleunigungswerten nach Gleichung A.8. Der Wert ist auch von der Einstellung für $n_{\overline{FFT}}$ abhängig.
Die Werte ALM_i_MAG1, ALM_i_MAG2, ALM_F_LOW und ALM_F_HIGH lassen sich für jede Abtastrate in SR_n für sechs Spektralbänder festlegen. Für jede Achse lassen sich 24 unterschiedliche Spektralalarme konfigurieren. Die Auswahl welcher Spektralalarm, durch Schreiben von Werten in ALM_i_MAG1, ALM_i_MAG2, ALM_F_LOW und ALM_F_HIGH konfiguriert werden soll, erfolgt in dem ALM_PNTR-Register. Hier werden SR_n und das Spektralband ausgewählt.
Sollte einer der Alarme für eine Messung auslösen, wird dies mithilfe von Flaggen signalisiert. Die Flaggen sind auf zwei Register mit groben und detaillierten Informationen aufgeteilt.

Grobe Informationen liefert das DIAG_STAT-Register. Hier wird signalisiert, dass eine Alarm für einer der drei Achsen und a_{mag2} oder a_{mag1} ausgelöst hat. Detaillierte Informationen liefern die ALM_i_STAT-Register. Hier ist genau angegeben für welches Spektralband a_{mag2} oder a_{mag1} ausgelöst hat. Für das Auslösen eines Alarms in einem Spektralband wird die höchste Beschleunigung im innerhalb des Frequenzbereichs $f_{\text{alarm, min}} \geq f \geq f_{\text{alarm, max}}$ in den ALM_i_PEAK-Registern gespeichert. Die Umrechnung des Wertes entspricht Gleichung A.8.

A.2 Umrechnung von Messwerten in allen Betriebsmodi

Mit einer Größe von 2 Bytes für jeden Messwerte der x-, y- und z-Achse, ergeben sich die Speicherbedarfe in den verschiedene Betriebsmodi in Tabelle A.1.

Nachdem die Messwerte über SPI übertragen wurden und abgespeichert sind, können sie weiter verarbeitet werden. In allen Betriebsmodi müssen die Werte aus den Bufferregistern von einem Hexwert in einen Dezimalwert umgerechnet werden. Die Umrechnung der Werte in einen Dezimalwert unterscheidet sich zwischen den FFT- und MTC-Betriebsmodi. Bei den Messwerten in den FFT-Modi genügt eine einfache Umrechnung. Da auch Messwerte mit einem negativen Vorzeichen im MTC-Modus möglich sind, sind die Messwerte im Zweierkomplement formatiert. Eine Umrechnung ist entsprechend erforderlich und im Folgenden kurz beschrieben.
Bei einer Darstellung im Zweierkomplement wird ein negatives Vorzeichen durch eine Eins im höchstwertigen Bit codiert. Bei positiven Vorzeichen ist das höchstwertige Bit Null. Wird beispielsweise ein Messwert von *0xE666* übertragen, kann dieser in drei Schritten in den gemessenen Beschleunigungswert umgerechnet werden. Für eine bessere Veranschaulichung der Schritte wird der Hexwert in einen Binärwert umgerechnet.

$$E666_{16} \longrightarrow 1110\ 0110\ 0110\ 0110_2$$

Im ersten Schritt müssen alle Bits des Messwertes invertiert werden.

$$0001\ 1001\ 1001\ 1001_2$$

Nach dem Invertieren wird der Wert mit 1 addiert

$$\begin{array}{r} \overset{1}{} \\ 0001\,1001\,1001\,1001 \\ +\,0000\,0000\,0000\,0001 \\ \hline 0001\,1001\,1001\,1010 \end{array}$$

Nach einer Umrechnung in einen Dezimalwert und Multiplikation mit -1 kann der Wert für die Berechnung der Beschleunigung verwendet werden.

$$0001\ 1001\ 1001\ 1010_2 \longrightarrow -1 \cdot 6554_{10}$$

Dieser Wert wird mithilfe von Gleichung A.5 in einen Beschleunigungswert umgerechnet. Bei Messwerten im Zeitbereich muss zur Umrechnung unterschieden werden zwischen Beschleunigungs- und Geschwindigkeitsmesswerten. Handelt es sich um Beschleunigungsmesswerte, lässt sich die Beschleunigung an der Stelle n $(0 \geq n \geq 4095)$ in dem Bufferregister BR mit der folgenden Gleichung berechnen.

$$a[n] = BR[n] \cdot 1,907\ mg \tag{A.5}$$

Geschwindigkeitswerte werden mit einem Faktor umgerechnet. Der Faktor cv ergibt sich aus der eingestellten Abtastrate

$$cv = \frac{2^{\text{AVG\_CNT Value}}}{f_s} \cdot 18,70 \frac{mm}{s} \tag{A.6}$$

Die Geschwindigkeit im Bufferregister BR lässt sich mit dem Faktor cv umrechnen

$$v[n] = BR[n] \cdot cv \tag{A.7}$$

Die Messwerte in den beiden FFT-Modi müssen über die Anzahl der FFT-Mittelwerte $n_{\overline{FFT}}$ umgerechnet werden. Für Beschleunigungen in beiden FFT-Modi errechnet sich die Beschleunigung an der Stelle n $(0 \geq n \geq 2047)$ mithilfe von

$$a[n] = \left(\frac{2^{\frac{BR[n]}{2048}}}{n_{\overline{FFT}}} \right) \cdot 0,9535\ mg \tag{A.8}$$

Tabelle A.1: Speicherbedarf aller Messwerte für die x-, y- und z-Achse in den Betriebsmodi

Betriebsmodus	Anzahl an Messwerten	Größe in Bytes
MFFT AFFT	2048	12 288
MTC	4096	24 576

A.3 Berechnung von $t_{\mathbf{MTC}}$ und $t_{\mathbf{FFT}}$

Die Messdauer MTC-Modus t_{MTC} errechnet sich nur aus der Abtastrate f_s

$$t_{\mathrm{MTC}} = 4096 \cdot \frac{1}{f_s} \tag{A.9}$$

Bei der Mittelwertsbildung werden n-einzelne Messungen hintereinander durchgeführt und am Ende der Mittelwert aller gebildet. Die Anzahl an $n_{\overline{FFT}}$ hat einen direkten Einfluss auf die Länge des Messzeitraumes t_{FFT} im AFFT- und MFFT-Modus.

$$t_{\mathrm{FFT}} = 4096 \cdot \frac{1}{f_s} \cdot n_{\overline{FFT}} \tag{A.10}$$

Für ein periodisches Erstellen von Messwerten ist zu beachten, dass gilt

$$t_{period} > t_{\mathrm{FFT}} \tag{A.11}$$

Für eine Messdauer im MTC-Modus ergeben sich aus Gleichung A.9 für t_{MTC} die Werte in Tabelle A.2 Die Messdauer im MFFT- und MFFT-Modus lässt sich feiner einstellen. Aus allen möglichen Konfigurationsmöglichkeiten ergeben sich für t_{FFT} die Werte in Tabelle A.6 bis Tabelle A.13.

Tabelle A.2: Messzeitraum im MTC-Modus

f_s in Hz	t_{MTC} in s	f_s in Hz	t_{MTC} in s
220k	0,0186	13,75k	0,2979
110k	0,0372	6875	0,5958
55k	0,0745	3437,5	1,1916
27,5k	0,1489	1718,75	2,3831

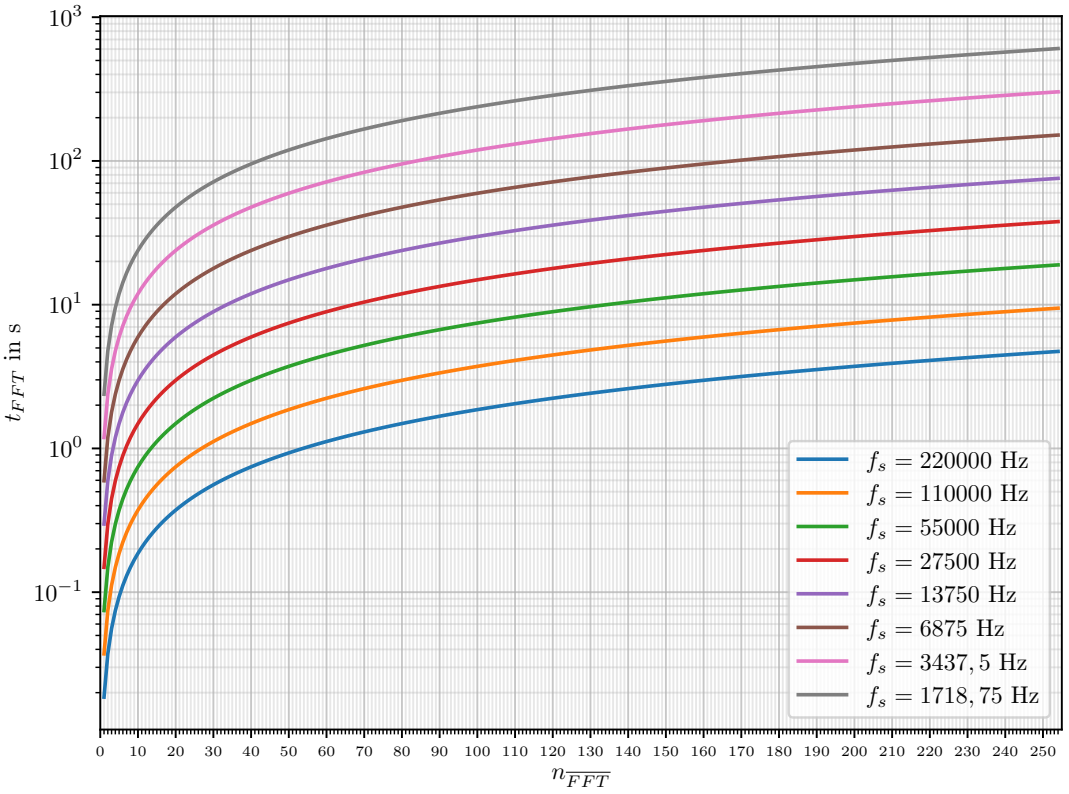


Abbildung A.1: Messzeitraum für die Aufnahme von FFT-Messwerten für alle f_s und $n_{\overline{FFT}}$

A.4 Tabellen

Tabelle A.3: Wichtige Register für die Konfiguration des Beschleunigungssensors

Registername	Beschreibung
ALM_CTRL	Allgemeine Einstellungen Spektralalarme
ALM_F_HIGH	Oberes Frequenzlimit für einen Spektralalarm
ALM_F_LOW	Unteres Frequenzlimit für einen Spektralalarm
ALM_i_MAG1	Grenzwert 1 für das Auslösen eines Alarmes Achse i
ALM_i_MAG2	Grenzwert 2 für das Auslösen eines Alarmes Achse i
ALM_i_PEAK	Spitzenwert der Beschleunigung im Frequenzbereich bei dem Auslösen eines Spektralalarms
ALM_i_STAT	Detaillierte Flaggen für das Auslösen des Spektralalarms der Achse i
ALM_PNTR	Auswahl der Parameter des Spektralalarms
AVG_CNT	Skalierungsfaktor für die Abtastrate
BUF_PNTR	Bin von Messwerten der sich aktuell in den x-,y- und z-Achse Buffern befindet
DIAG_STAT	Grobe Flaggen für das Auslösen eines Spektralalarms
FFT_AVG1	Anzahl an FFT aus denen der Mittelwert errechnet werden soll für Abtastraten SR0 und SR1
FFT_AVG2	Anzahl an FFTs aus denen der Mittelwert errechnet werden soll für Abtastraten SR2 und SR3
FILT_CTRL	Aktivieren von insgesamt 6 FIR-Filtern für jede Achse
GLOB_CMD	Verschiedenste globale Einstellungen
REC_CTRL	Aufnahmemodus, Aktivierung von Abtastraten, Geschwindigkeit anstatt von Beschleunigung und Fenster
REC_PRD	Periodendauer zur Bestimmung vonFFTs im AFFT-Modus
X_ANULL	Korrekturwert aus Autonullfunktion x-Achse
X_BUF	Messwert aktueller Bin x-Achse
Y_ANULL	Korrekturwert aus Autonullfunktion y-Achse
Y_BUF	Messwert aktueller Bin y-Achse
Z_ANULL	Korrekturwert aus Autonullfunktion z-Achse
Z_BUF/RSS_BUF	Messwert aktueller Bin z-Achse

Tabelle A.4: Symbolverzechnis

Symbol	Einheit	Beschreibung
a	$\frac{m}{s^2}$	Beschleunigung
a_{mag1}	g	Beschleunigung für Spektralalarm in ALM_i_MAG1
a_{mag2}	g	Beschleunigung für Spektralalarm in ALM_i_MAG2
bw	Hz	Weite Frequenzbins
Bit	1	Einheit für den Informationsgehalt einer Nachricht
Byte	$8 \cdot$ Bit	Einheit für den Informationsgehalt einer Nachricht
f	Hz	Frequenz
$f_{\text{alarm, min}}$	Hz	unteres Frequenzlimit Spektralalarm
$f_{\text{alarm, max}}$	Hz	oberes Frequenzlimit Spektralalarm
f_{CAN}	Hz	Baudrate CAN-Bus
$f_{\text{cutoff, low}}$	Hz	unteres Frequenzlimit des durch den Beschleunigungssensor messbaren Frequenzbereichs
$f_{\text{cutoff, high}}$	Hz	oberes Frequenzlimit des durch den Beschleunigungssensor messbaren Frequenzbereichs
f_g	Hz	Grenzfrequenz Filter
f_{SPI}	Hz	Taktrate SPI-Bus
f_s	Hz	Abtastrate
fb	1	Frequenzbin
g	$9,81 \frac{m}{s^2}$	Erdbeschleunigung
I	A	elektrischer Strom
n_{bits}	1	Anzahl der zu übertragenden Bits in einer CAN- oder SPI-Nachricht
$n_{\overline{FFT}}$	1	Anzahl der FFT-Mittelwerte
n_{msg}	1	Nachrichten pro selbstdefinierter Kennung
t	s	Zeit
t_{CAN}	s	Übertragungsdauer für eine CAN-Nachricht
$t_{\text{CAN, T}}$	s	Gesamtübertragungsdauer für alle CAN-Nachrichten
t_{data}	s	Gesamtübertragungsdauer für die Übertragung von Messwerten über CAN
t_{MTC}	s	Messdauer MTC-Modus
t_{FFT}	s	Messdauer FFT-Modi
t_{SPI}	s	Übertragungsdauer für eine SPI-Nachricht
$t_{\text{SPI, T}}$	s	Gesamtübertragungsdauer für alle SPI-Nachrichten
t_{sync}	s	Zeit SyncSeg CAN-Bus
t_{prop}	s	Zeit PropSeg CAN-Bus
t_{ps1}	s	Zeit PhaSeg1 CAN-Bus
t_{ps2}	s	Zeit PhaSeg2 CAN-Bus
T	s	Periodendauer
T_{sense}	$^\circ C$	gemessene Temperatur im TEMP_OUT-Register
U	V	elektrische Spannung
v	$\frac{m}{s}$	Geschwindigkeit
ω	$\frac{1}{s}$	Kreisfrequenz

Tabelle A.5: Übertragungsdauer für Messwerte in den FFT-Betriebsmodi und dem MTC-Betriebsmodus für $f_{\text{CAN}} = 500\frac{kBit}{s}$ und $f_{\text{SPI}} = 8\ MHz$

	Kennung	Worst Case Inhalt	n_{msg}	t_{CAN} in ms	$t_{\text{CAN, T}}$ in ms	t_{SPI} in μs	$t_{\text{SPI, T}}$ in ms	t_{data} in ms
FFT	0x92	928000	1	0,152	471,306	-	-	692,49
	0x94	94000000000000	2048	471,04		36	221,184	
	0x96	96	1	0,114		-	-	
MTC	0xE2	E2000001	1	0,17	958,748	-	-	1400
	0xE4	E4000000000000	4096	958,464		36	442,368	
	0xE6	E6	1	0,114		-	-	

Tabelle A.6: Messzeiträume für FFT-Modi für $1 \geq n_{\overline{FFT}} \geq 34$

	f_s in Hz							
$n_{\overline{FFT}}$	*220000*	*110000*	*55000*	*27500*	*13750*	*6875*	*3437,5*	*1718,75*
1	0,0186	0,0372	0,0745	0,1489	0,2979	0,5958	1,1916	2,3831
2	0,0372	0,0745	0,1489	0,2979	0,5958	1,1916	2,3831	4,7663
3	0,0559	0,1117	0,2234	0,4468	0,8937	1,7873	3,5747	7,1494
4	0,0745	0,1489	0,2979	0,5958	1,1916	2,3831	4,7663	9,5325
5	0,0931	0,1862	0,3724	0,7447	1,4895	2,9789	5,9578	11,9156
6	0,1117	0,2234	0,4468	0,8937	1,7873	3,5747	7,1494	14,2988
7	0,1303	0,2607	0,5213	1,0426	2,0852	4,1705	8,3409	16,6819
8	0,1489	0,2979	0,5958	1,1916	2,3831	4,7663	9,5325	19,0650
9	0,1676	0,3351	0,6703	1,3405	2,6810	5,3620	10,7241	21,4481
10	0,1862	0,3724	0,7447	1,4895	2,9789	5,9578	11,9156	23,8313
11	0,2048	0,4096	0,8192	1,6384	3,2768	6,5536	13,1072	26,2144
12	0,2234	0,4468	0,8937	1,7873	3,5747	7,1494	14,2988	28,5975
13	0,2420	0,4841	0,9681	1,9363	3,8726	7,7452	15,4903	30,9807
14	0,2607	0,5213	1,0426	2,0852	4,1705	8,3409	16,6819	33,3638
15	0,2793	0,5585	1,1171	2,2342	4,4684	8,9367	17,8735	35,7469
16	0,2979	0,5958	1,1916	2,3831	4,7663	9,5325	19,0650	38,1300
17	0,3165	0,6330	1,2660	2,5321	5,0641	10,1283	20,2566	40,5132
18	0,3351	0,6703	1,3405	2,6810	5,3620	10,7241	21,4481	42,8963
19	0,3537	0,7075	1,4150	2,8300	5,6599	11,3199	22,6397	45,2794
20	0,3724	0,7447	1,4895	2,9789	5,9578	11,9156	23,8313	47,6625
21	0,3910	0,7820	1,5639	3,1279	6,2557	12,5114	25,0228	50,0457
22	0,4096	0,8192	1,6384	3,2768	6,5536	13,1072	26,2144	52,4288
23	0,4282	0,8564	1,7129	3,4257	6,8515	13,7030	27,4060	54,8119
24	0,4468	0,8937	1,7873	3,5747	7,1494	14,2988	28,5975	57,1951
25	0,4655	0,9309	1,8618	3,7236	7,4473	14,8945	29,7891	59,5782
26	0,4841	0,9681	1,9363	3,8726	7,7452	15,4903	30,9807	61,9613
27	0,5027	1,0054	2,0108	4,0215	8,0431	16,0861	32,1722	64,3444
28	0,5213	1,0426	2,0852	4,1705	8,3409	16,6819	33,3638	66,7276
29	0,5399	1,0799	2,1597	4,3194	8,6388	17,2777	34,5553	69,1107
30	0,5585	1,1171	2,2342	4,4684	8,9367	17,8735	35,7469	71,4938
31	0,5772	1,1543	2,3087	4,6173	9,2346	18,4692	36,9385	73,8769
32	0,5958	1,1916	2,3831	4,7663	9,5325	19,0650	38,1300	76,2601
33	0,6144	1,2288	2,4576	4,9152	9,8304	19,6608	39,3216	78,6432
34	0,6330	1,2660	2,5321	5,0641	10,1283	20,2566	40,5132	81,0263

Tabelle A.7: Messzeiträume für FFT-Modi für $35 \geq n_{\overline{FFT}} \geq 69$

	f_s in Hz							
$n_{\overline{FFT}}$	*220000*	*110000*	*55000*	*27500*	*13750*	*6875*	*3437,5*	*1718,75*
35	0,6516	1,3033	2,6065	5,2131	10,4262	20,8524	41,7047	83,4095
36	0,6703	1,3405	2,6810	5,3620	10,7241	21,4481	42,8963	85,7926
37	0,6889	1,3777	2,7555	5,5110	11,0220	22,0439	44,0879	88,1757
38	0,7075	1,4150	2,8300	5,6599	11,3199	22,6397	45,2794	90,5588
39	0,7261	1,4522	2,9044	5,8089	11,6177	23,2355	46,4710	92,9420
40	0,7447	1,4895	2,9789	5,9578	11,9156	23,8313	47,6625	95,3251
41	0,7633	1,5267	3,0534	6,1068	12,2135	24,4271	48,8541	97,7082
42	0,7820	1,5639	3,1279	6,2557	12,5114	25,0228	50,0457	100,0913
43	0,8006	1,6012	3,2023	6,4047	12,8093	25,6186	51,2372	102,4745
44	0,8192	1,6384	3,2768	6,5536	13,1072	26,2144	52,4288	104,8576
45	0,8378	1,6756	3,3513	6,7025	13,4051	26,8102	53,6204	107,2407
46	0,8564	1,7129	3,4257	6,8515	13,7030	27,4060	54,8119	109,6239
47	0,8751	1,7501	3,5002	7,0004	14,0009	28,0017	56,0035	112,0070
48	0,8937	1,7873	3,5747	7,1494	14,2988	28,5975	57,1951	114,3901
49	0,9123	1,8246	3,6492	7,2983	14,5967	29,1933	58,3866	116,7732
50	0,9309	1,8618	3,7236	7,4473	14,8945	29,7891	59,5782	119,1564
51	0,9495	1,8991	3,7981	7,5962	15,1924	30,3849	60,7697	121,5395
52	0,9681	1,9363	3,8726	7,7452	15,4903	30,9807	61,9613	123,9226
53	0,9868	1,9735	3,9471	7,8941	15,7882	31,5764	63,1529	126,3057
54	1,0054	2,0108	4,0215	8,0431	16,0861	32,1722	64,3444	128,6889
55	1,0240	2,0480	4,0960	8,1920	16,3840	32,7680	65,5360	131,0720
56	1,0426	2,0852	4,1705	8,3409	16,6819	33,3638	66,7276	133,4551
57	1,0612	2,1225	4,2449	8,4899	16,9798	33,9596	67,9191	135,8383
58	1,0799	2,1597	4,3194	8,6388	17,2777	34,5553	69,1107	138,2214
59	1,0985	2,1969	4,3939	8,7878	17,5756	35,1511	70,3023	140,6045
60	1,1171	2,2342	4,4684	8,9367	17,8735	35,7469	71,4938	142,9876
61	1,1357	2,2714	4,5428	9,0857	18,1713	36,3427	72,6854	145,3708
62	1,1543	2,3087	4,6173	9,2346	18,4692	36,9385	73,8769	147,7539
63	1,1729	2,3459	4,6918	9,3836	18,7671	37,5343	75,0685	150,1370
64	1,1916	2,3831	4,7663	9,5325	19,0650	38,1300	76,2601	152,5201
65	1,2102	2,4204	4,8407	9,6815	19,3629	38,7258	77,4516	154,9033
66	1,2288	2,4576	4,9152	9,8304	19,6608	39,3216	78,6432	157,2864
67	1,2474	2,4948	4,9897	9,9793	19,9587	39,9174	79,8348	159,6695
68	1,2660	2,5321	5,0641	10,1283	20,2566	40,5132	81,0263	162,0527
69	1,2847	2,5693	5,1386	10,2772	20,5545	41,1089	82,2179	164,4358

Tabelle A.8: Messzeiträume für FFT-Modi für $70 \geq n_{\overline{FFT}} \geq 103$

	f_s in Hz							
$n_{\overline{FFT}}$	*220000*	*110000*	*55000*	*27500*	*13750*	*6875*	*3437,5*	*1718,75*
70	1,3033	2,6065	5,2131	10,4262	20,8524	41,7047	83,4095	166,8189
71	1,3219	2,6438	5,2876	10,5751	21,1503	42,3005	84,6010	169,2020
72	1,3405	2,6810	5,3620	10,7241	21,4481	42,8963	85,7926	171,5852
73	1,3591	2,7183	5,4365	10,8730	21,7460	43,4921	86,9841	173,9683
74	1,3777	2,7555	5,5110	11,0220	22,0439	44,0879	88,1757	176,3514
75	1,3964	2,7927	5,5855	11,1709	22,3418	44,6836	89,3673	178,7345
76	1,4150	2,8300	5,6599	11,3199	22,6397	45,2794	90,5588	181,1177
77	1,4336	2,8672	5,7344	11,4688	22,9376	45,8752	91,7504	183,5008
78	1,4522	2,9044	5,8089	11,6177	23,2355	46,4710	92,9420	185,8839
79	1,4708	2,9417	5,8833	11,7667	23,5334	47,0668	94,1335	188,2671
80	1,4895	2,9789	5,9578	11,9156	23,8313	47,6625	95,3251	190,6502
81	1,5081	3,0161	6,0323	12,0646	24,1292	48,2583	96,5167	193,0333
82	1,5267	3,0534	6,1068	12,2135	24,4271	48,8541	97,7082	195,4164
83	1,5453	3,0906	6,1812	12,3625	24,7249	49,4499	98,8998	197,7996
84	1,5639	3,1279	6,2557	12,5114	25,0228	50,0457	100,0913	200,1827
85	1,5825	3,1651	6,3302	12,6604	25,3207	50,6415	101,2829	202,5658
86	1,6012	3,2023	6,4047	12,8093	25,6186	51,2372	102,4745	204,9489
87	1,6198	3,2396	6,4791	12,9583	25,9165	51,8330	103,6660	207,3321
88	1,6384	3,2768	6,5536	13,1072	26,2144	52,4288	104,8576	209,7152
89	1,6570	3,3140	6,6281	13,2561	26,5123	53,0246	106,0492	212,0983
90	1,6756	3,3513	6,7025	13,4051	26,8102	53,6204	107,2407	214,4815
91	1,6943	3,3885	6,7770	13,5540	27,1081	54,2161	108,4323	216,8646
92	1,7129	3,4257	6,8515	13,7030	27,4060	54,8119	109,6239	219,2477
93	1,7315	3,4630	6,9260	13,8519	27,7039	55,4077	110,8154	221,6308
94	1,7501	3,5002	7,0004	14,0009	28,0017	56,0035	112,0070	224,0140
95	1,7687	3,5375	7,0749	14,1498	28,2996	56,5993	113,1985	226,3971
96	1,7873	3,5747	7,1494	14,2988	28,5975	57,1951	114,3901	228,7802
97	1,8060	3,6119	7,2239	14,4477	28,8954	57,7908	115,5817	231,1633
98	1,8246	3,6492	7,2983	14,5967	29,1933	58,3866	116,7732	233,5465
99	1,8432	3,6864	7,3728	14,7456	29,4912	58,9824	117,9648	235,9296
100	1,8618	3,7236	7,4473	14,8945	29,7891	59,5782	119,1564	238,3127
101	1,8804	3,7609	7,5217	15,0435	30,0870	60,1740	120,3479	240,6959
102	1,8991	3,7981	7,5962	15,1924	30,3849	60,7697	121,5395	243,0790
103	1,9177	3,8353	7,6707	15,3414	30,6828	61,3655	122,7311	245,4621

Tabelle A.9: Messzeiträume für FFT-Modi für $104 \geq n_{\overline{FFT}} \geq 136$

	f_s in Hz							
$n_{\overline{FFT}}$	*220000*	*110000*	*55000*	*27500*	*13750*	*6875*	*3437,5*	*1718,75*
104	1,9363	3,8726	7,7452	15,4903	30,9807	61,9613	123,9226	247,8452
105	1,9549	3,9098	7,8196	15,6393	31,2785	62,5571	125,1142	250,2284
106	1,9735	3,9471	7,8941	15,7882	31,5764	63,1529	126,3057	252,6115
107	1,9921	3,9843	7,9686	15,9372	31,8743	63,7487	127,4973	254,9946
108	2,0108	4,0215	8,0431	16,0861	32,1722	64,3444	128,6889	257,3777
109	2,0294	4,0588	8,1175	16,2351	32,4701	64,9402	129,8804	259,7609
110	2,0480	4,0960	8,1920	16,3840	32,7680	65,5360	131,0720	262,1440
111	2,0666	4,1332	8,2665	16,5329	33,0659	66,1318	132,2636	264,5271
112	2,0852	4,1705	8,3409	16,6819	33,3638	66,7276	133,4551	266,9103
113	2,1039	4,2077	8,4154	16,8308	33,6617	67,3233	134,6467	269,2934
114	2,1225	4,2449	8,4899	16,9798	33,9596	67,9191	135,8383	271,6765
115	2,1411	4,2822	8,5644	17,1287	34,2575	68,5149	137,0298	274,0596
116	2,1597	4,3194	8,6388	17,2777	34,5553	69,1107	138,2214	276,4428
117	2,1783	4,3567	8,7133	17,4266	34,8532	69,7065	139,4129	278,8259
118	2,1969	4,3939	8,7878	17,5756	35,1511	70,3023	140,6045	281,2090
119	2,2156	4,4311	8,8623	17,7245	35,4490	70,8980	141,7961	283,5921
120	2,2342	4,4684	8,9367	17,8735	35,7469	71,4938	142,9876	285,9753
121	2,2528	4,5056	9,0112	18,0224	36,0448	72,0896	144,1792	288,3584
122	2,2714	4,5428	9,0857	18,1713	36,3427	72,6854	145,3708	290,7415
123	2,2900	4,5801	9,1601	18,3203	36,6406	73,2812	146,5623	293,1247
124	2,3087	4,6173	9,2346	18,4692	36,9385	73,8769	147,7539	295,5078
125	2,3273	4,6545	9,3091	18,6182	37,2364	74,4727	148,9455	297,8909
126	2,3459	4,6918	9,3836	18,7671	37,5343	75,0685	150,1370	300,2740
127	2,3645	4,7290	9,4580	18,9161	37,8321	75,6643	151,3286	302,6572
128	2,3831	4,7663	9,5325	19,0650	38,1300	76,2601	152,5201	305,0403
129	2,4017	4,8035	9,6070	19,2140	38,4279	76,8559	153,7117	307,4234
130	2,4204	4,8407	9,6815	19,3629	38,7258	77,4516	154,9033	309,8065
131	2,4390	4,8780	9,7559	19,5119	39,0237	78,0474	156,0948	312,1897
132	2,4576	4,9152	9,8304	19,6608	39,3216	78,6432	157,2864	314,5728
133	2,4762	4,9524	9,9049	19,8097	39,6195	79,2390	158,4780	316,9559
134	2,4948	4,9897	9,9793	19,9587	39,9174	79,8348	159,6695	319,3391
135	2,5135	5,0269	10,0538	20,1076	40,2153	80,4305	160,8611	321,7222
136	2,5321	5,0641	10,1283	20,2566	40,5132	81,0263	162,0527	324,1053

Tabelle A.10: Messzeiträume für FFT-Modi für $137 \geq n_{\overline{FFT}} \geq 170$

	f_s in Hz							
$n_{\overline{FFT}}$	*220000*	*110000*	*55000*	*27500*	*13750*	*6875*	*3437,5*	*1718,75*
137	2,5507	5,1014	10,2028	20,4055	40,8111	81,6221	163,2442	326,4884
138	2,5693	5,1386	10,2772	20,5545	41,1089	82,2179	164,4358	328,8716
139	2,5879	5,1759	10,3517	20,7034	41,4068	82,8137	165,6273	331,2547
140	2,6065	5,2131	10,4262	20,8524	41,7047	83,4095	166,8189	333,6378
141	2,6252	5,2503	10,5007	21,0013	42,0026	84,0052	168,0105	336,0209
142	2,6438	5,2876	10,5751	21,1503	42,3005	84,6010	169,2020	338,4041
143	2,6624	5,3248	10,6496	21,2992	42,5984	85,1968	170,3936	340,7872
144	2,6810	5,3620	10,7241	21,4481	42,8963	85,7926	171,5852	343,1703
145	2,6996	5,3993	10,7985	21,5971	43,1942	86,3884	172,7767	345,5535
146	2,7183	5,4365	10,8730	21,7460	43,4921	86,9841	173,9683	347,9366
147	2,7369	5,4737	10,9475	21,8950	43,7900	87,5799	175,1599	350,3197
148	2,7555	5,5110	11,0220	22,0439	44,0879	88,1757	176,3514	352,7028
149	2,7741	5,5482	11,0964	22,1929	44,3857	88,7715	177,5430	355,0860
150	2,7927	5,5855	11,1709	22,3418	44,6836	89,3673	178,7345	357,4691
151	2,8113	5,6227	11,2454	22,4908	44,9815	89,9631	179,9261	359,8522
152	2,8300	5,6599	11,3199	22,6397	45,2794	90,5588	181,1177	362,2353
153	2,8486	5,6972	11,3943	22,7887	45,5773	91,1546	182,3092	364,6185
154	2,8672	5,7344	11,4688	22,9376	45,8752	91,7504	183,5008	367,0016
155	2,8858	5,7716	11,5433	23,0865	46,1731	92,3462	184,6924	369,3847
156	2,9044	5,8089	11,6177	23,2355	46,4710	92,9420	185,8839	371,7679
157	2,9231	5,8461	11,6922	23,3844	46,7689	93,5377	187,0755	374,1510
158	2,9417	5,8833	11,7667	23,5334	47,0668	94,1335	188,2671	376,5341
159	2,9603	5,9206	11,8412	23,6823	47,3647	94,7293	189,4586	378,9172
160	2,9789	5,9578	11,9156	23,8313	47,6625	95,3251	190,6502	381,3004
161	2,9975	5,9951	11,9901	23,9802	47,9604	95,9209	191,8417	383,6835
162	3,0161	6,0323	12,0646	24,1292	48,2583	96,5167	193,0333	386,0666
163	3,0348	6,0695	12,1391	24,2781	48,5562	97,1124	194,2249	388,4497
164	3,0534	6,1068	12,2135	24,4271	48,8541	97,7082	195,4164	390,8329
165	3,0720	6,1440	12,2880	24,5760	49,1520	98,3040	196,6080	393,2160
166	3,0906	6,1812	12,3625	24,7249	49,4499	98,8998	197,7996	395,5991
167	3,1092	6,2185	12,4369	24,8739	49,7478	99,4956	198,9911	397,9823
168	3,1279	6,2557	12,5114	25,0228	50,0457	100,0913	200,1827	400,3654
169	3,1465	6,2929	12,5859	25,1718	50,3436	100,6871	201,3743	402,7485
170	3,1651	6,3302	12,6604	25,3207	50,6415	101,2829	202,5658	405,1316

Tabelle A.11: Messzeiträume für FFT-Modi für $171 \geq n_{\overline{FFT}} \geq 204$

	f_s in Hz							
$n_{\overline{FFT}}$	*220000*	*110000*	*55000*	*27500*	*13750*	*6875*	*3437,5*	*1718,75*
171	3,1837	6,3674	12,7348	25,4697	50,9393	101,8787	203,7574	407,5148
172	3,2023	6,4047	12,8093	25,6186	51,2372	102,4745	204,9489	409,8979
173	3,2209	6,4419	12,8838	25,7676	51,5351	103,0703	206,1405	412,2810
174	3,2396	6,4791	12,9583	25,9165	51,8330	103,6660	207,3321	414,6641
175	3,2582	6,5164	13,0327	26,0655	52,1309	104,2618	208,5236	417,0473
176	3,2768	6,5536	13,1072	26,2144	52,4288	104,8576	209,7152	419,4304
177	3,2954	6,5908	13,1817	26,3633	52,7267	105,4534	210,9068	421,8135
178	3,3140	6,6281	13,2561	26,5123	53,0246	106,0492	212,0983	424,1967
179	3,3327	6,6653	13,3306	26,6612	53,3225	106,6449	213,2899	426,5798
180	3,3513	6,7025	13,4051	26,8102	53,6204	107,2407	214,4815	428,9629
181	3,3699	6,7398	13,4796	26,9591	53,9183	107,8365	215,6730	431,3460
182	3,3885	6,7770	13,5540	27,1081	54,2161	108,4323	216,8646	433,7292
183	3,4071	6,8143	13,6285	27,2570	54,5140	109,0281	218,0561	436,1123
184	3,4257	6,8515	13,7030	27,4060	54,8119	109,6239	219,2477	438,4954
185	3,4444	6,8887	13,7775	27,5549	55,1098	110,2196	220,4393	440,8785
186	3,4630	6,9260	13,8519	27,7039	55,4077	110,8154	221,6308	443,2617
187	3,4816	6,9632	13,9264	27,8528	55,7056	111,4112	222,8224	445,6448
188	3,5002	7,0004	14,0009	28,0017	56,0035	112,0070	224,0140	448,0279
189	3,5188	7,0377	14,0753	28,1507	56,3014	112,6028	225,2055	450,4111
190	3,5375	7,0749	14,1498	28,2996	56,5993	113,1985	226,3971	452,7942
191	3,5561	7,1121	14,2243	28,4486	56,8972	113,7943	227,5887	455,1773
192	3,5747	7,1494	14,2988	28,5975	57,1951	114,3901	228,7802	457,5604
193	3,5933	7,1866	14,3732	28,7465	57,4929	114,9859	229,9718	459,9436
194	3,6119	7,2239	14,4477	28,8954	57,7908	115,5817	231,1633	462,3267
195	3,6305	7,2611	14,5222	29,0444	58,0887	116,1775	232,3549	464,7098
196	3,6492	7,2983	14,5967	29,1933	58,3866	116,7732	233,5465	467,0929
197	3,6678	7,3356	14,6711	29,3423	58,6845	117,3690	234,7380	469,4761
198	3,6864	7,3728	14,7456	29,4912	58,9824	117,9648	235,9296	471,8592
199	3,7050	7,4100	14,8201	29,6401	59,2803	118,5606	237,1212	474,2423
200	3,7236	7,4473	14,8945	29,7891	59,5782	119,1564	238,3127	476,6255
201	3,7423	7,4845	14,9690	29,9380	59,8761	119,7521	239,5043	479,0086
202	3,7609	7,5217	15,0435	30,0870	60,1740	120,3479	240,6959	481,3917
203	3,7795	7,5590	15,1180	30,2359	60,4719	120,9437	241,8874	483,7748
204	3,7981	7,5962	15,1924	30,3849	60,7697	121,5395	243,0790	486,1580

Tabelle A.12: Messzeiträume für FFT-Modi für $205 \geq n_{\overline{FFT}} \geq 238$

	f_s in Hz							
$n_{\overline{FFT}}$	*220000*	*110000*	*55000*	*27500*	*13750*	*6875*	*3437,5*	*1718,75*
205	3,8167	7,6335	15,2669	30,5338	61,0676	122,1353	244,2705	488,5411
206	3,8353	7,6707	15,3414	30,6828	61,3655	122,7311	245,4621	490,9242
207	3,8540	7,7079	15,4159	30,8317	61,6634	123,3268	246,6537	493,3073
208	3,8726	7,7452	15,4903	30,9807	61,9613	123,9226	247,8452	495,6905
209	3,8912	7,7824	15,5648	31,1296	62,2592	124,5184	249,0368	498,0736
210	3,9098	7,8196	15,6393	31,2785	62,5571	125,1142	250,2284	500,4567
211	3,9284	7,8569	15,7137	31,4275	62,8550	125,7100	251,4199	502,8399
212	3,9471	7,8941	15,7882	31,5764	63,1529	126,3057	252,6115	505,2230
213	3,9657	7,9313	15,8627	31,7254	63,4508	126,9015	253,8031	507,6061
214	3,9843	7,9686	15,9372	31,8743	63,7487	127,4973	254,9946	509,9892
215	4,0029	8,0058	16,0116	32,0233	64,0465	128,0931	256,1862	512,3724
216	4,0215	8,0431	16,0861	32,1722	64,3444	128,6889	257,3777	514,7555
217	4,0401	8,0803	16,1606	32,3212	64,6423	129,2847	258,5693	517,1386
218	4,0588	8,1175	16,2351	32,4701	64,9402	129,8804	259,7609	519,5217
219	4,0774	8,1548	16,3095	32,6191	65,2381	130,4762	260,9524	521,9049
220	4,0960	8,1920	16,3840	32,7680	65,5360	131,0720	262,1440	524,2880
221	4,1146	8,2292	16,4585	32,9169	65,8339	131,6678	263,3356	526,6711
222	4,1332	8,2665	16,5329	33,0659	66,1318	132,2636	264,5271	529,0543
223	4,1519	8,3037	16,6074	33,2148	66,4297	132,8593	265,7187	531,4374
224	4,1705	8,3409	16,6819	33,3638	66,7276	133,4551	266,9103	533,8205
225	4,1891	8,3782	16,7564	33,5127	67,0255	134,0509	268,1018	536,2036
226	4,2077	8,4154	16,8308	33,6617	67,3233	134,6467	269,2934	538,5868
227	4,2263	8,4527	16,9053	33,8106	67,6212	135,2425	270,4849	540,9699
228	4,2449	8,4899	16,9798	33,9596	67,9191	135,8383	271,6765	543,3530
229	4,2636	8,5271	17,0543	34,1085	68,2170	136,4340	272,8681	545,7361
230	4,2822	8,5644	17,1287	34,2575	68,5149	137,0298	274,0596	548,1193
231	4,3008	8,6016	17,2032	34,4064	68,8128	137,6256	275,2512	550,5024
232	4,3194	8,6388	17,2777	34,5553	69,1107	138,2214	276,4428	552,8855
233	4,3380	8,6761	17,3521	34,7043	69,4086	138,8172	277,6343	555,2687
234	4,3567	8,7133	17,4266	34,8532	69,7065	139,4129	278,8259	557,6518
235	4,3753	8,7505	17,5011	35,0022	70,0044	140,0087	280,0175	560,0349
236	4,3939	8,7878	17,5756	35,1511	70,3023	140,6045	281,2090	562,4180
237	4,4125	8,8250	17,6500	35,3001	70,6001	141,2003	282,4006	564,8012
238	4,4311	8,8623	17,7245	35,4490	70,8980	141,7961	283,5921	567,1843

Tabelle A.13: Messzeiträume für FFT-Modi für $239 \geq n_{\overline{FFT}} \geq 255$

	f_s in Hz							
$n_{\overline{FFT}}$	*220000*	*110000*	*55000*	*27500*	*13750*	*6875*	*3437,5*	*1718,75*
239	4,4497	8,8995	17,7990	35,5980	71,1959	142,3919	284,7837	569,5674
240	4,4684	8,9367	17,8735	35,7469	71,4938	142,9876	285,9753	571,9505
241	4,4870	8,9740	17,9479	35,8959	71,7917	143,5834	287,1668	574,3337
242	4,5056	9,0112	18,0224	36,0448	72,0896	144,1792	288,3584	576,7168
243	4,5242	9,0484	18,0969	36,1937	72,3875	144,7750	289,5500	579,0999
244	4,5428	9,0857	18,1713	36,3427	72,6854	145,3708	290,7415	581,4831
245	4,5615	9,1229	18,2458	36,4916	72,9833	145,9665	291,9331	583,8662
246	4,5801	9,1601	18,3203	36,6406	73,2812	146,5623	293,1247	586,2493
247	4,5987	9,1974	18,3948	36,7895	73,5791	147,1581	294,3162	588,6324
248	4,6173	9,2346	18,4692	36,9385	73,8769	147,7539	295,5078	591,0156
249	4,6359	9,2719	18,5437	37,0874	74,1748	148,3497	296,6993	593,3987
250	4,6545	9,3091	18,6182	37,2364	74,4727	148,9455	297,8909	595,7818
251	4,6732	9,3463	18,6927	37,3853	74,7706	149,5412	299,0825	598,1649
252	4,6918	9,3836	18,7671	37,5343	75,0685	150,1370	300,2740	600,5481
253	4,7104	9,4208	18,8416	37,6832	75,3664	150,7328	301,4656	602,9312
254	4,7290	9,4580	18,9161	37,8321	75,6643	151,3286	302,6572	605,3143
255	4,7476	9,4953	18,9905	37,9811	75,9622	151,9244	303,8487	607,6975

Tabelle A.14: Messparameter Messung 1

	Funktionsgenerator f in Hz													ADcmXL3021				
ID	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_s in Hz	bw in Hz	Modus	$n_{\overline{FFT}}$	t_{FFT} in s
1_1	0,5	1	5	10	25	50	75	100	200	300	400	500	600	220k	53,71	MFFT	109	2,03
1_2														27,5k	6,71	MFFT	16	2,38
1_3														1718,75	0,42	MFFT	1	2,38
1_4														3437,5	0,84	MFFT	2	2,38
	Funktionsgenerator f in Hz													ADcmXL3021				
ID	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}	f_{22}	f_{23}	f_{24}	f_{25}	f_{26}	f_s in Hz	bw in Hz	Modus	$n_{\overline{FFT}}$	t_{FFT} in s
1_1	700	800	900	1k	2k	3k	4k	5k	6k	7k	8k	9k	10k	220k	53,71	MFFT	109	2,03
1_2														27,5k	6,71	MFFT	16	2,38
1_3														1718,75	0,42	MFFT	1	2,38
1_4														3437,5	0,84	MFFT	2	2,38

Tabelle A.15: Messparameter Messung 2 und 3

	FG		ADcmXL3021				
ID	f	I in A	f_s in Hz	bw in Hz	Modus	$n_{\overline{FFT}}$	t_{FFT} in s
2_1_1	100	0,14	27,5k	6,71	MFFT	33	4,92
2_1_2	100	0,21	27,5k	6,71	MFFT	33	4,92
2_1_3	100	0,33	27,5k	6,71	MFFT	33	4,92
ID	f	I in A	f_s in Hz	bw in Hz	Modus	$n_{\overline{FFT}}$	t_{FFT} in s
2_2_1	1000	0,16	27,5k	6,71	MFFT	33	4,92
2_2_2	1000	0,2	27,5k	6,71	MFFT	33	4,92
2_2_3	1000	0,28	27,5k	6,71	MFFT	33	4,92
2_2_4	1000	0,39	27,5k	6,71	MFFT	33	4,92
2_2_5	1000	0,5	27,5k	6,71	MFFT	33	4,92
ID	f	I in A	f_s in Hz	bw in Hz	Modus	$n_{\overline{FFT}}$	t_{FFT} in s
2_3_1	4600	0,75	27,5k	6,71	MFFT	33	4,92
2_3_2	4600	0,89	27,5k	6,71	MFFT	33	4,92
2_3_3	4600	1,09	27,5k	6,71	MFFT	33	4,92
	FG		ADcmXL3021				
ID	f	I in A	f_s in Hz	bw in Hz	Modus	t_{MTC} in s	
3_1_1	100	0,14	1718,75	0,42	MTC	2,38	
3_1_2	100	0,21	1718,75	0,42	MTC	2,38	
3_1_3	100	0,26	1718,75	0,42	MTC	2,38	
3_1_4	100	0,33	1718,75	0,42	MTC	2,38	
3_1_5	100	0,46	1718,75	0,42	MTC	2,38	
ID	f	I in A	f_s in Hz	bw in Hz	Modus	t_{MTC} in s	
3_2_1	1000	0,16	3437,5	0,84	MTC	1,19	
3_2_2	1000	0,22	3437,5	0,84	MTC	1,19	
3_2_3	1000	0,33	3437,5	0,84	MTC	1,19	
3_2_4	1000	0,42	3437,5	0,84	MTC	1,19	
3_2_5	1000	0,52	3437,5	0,84	MTC	1,19	
ID	f	I in A	f_s in Hz	bw in Hz	Modus	t_{MTC} in s	
3_3_1	4600	0,75	1718,75	0,42	MTC	2,38	
3_3_2	4600	0,96	1718,75	0,42	MTC	2,38	
3_3_3	4600	1,19	1718,75	0,42	MTC	2,38	

Tabelle A.16: Messparameter Messung 4

	FG			ADcmXL3021				
ID	f_1 in Hz	f_2 in Hz	I in A	f_s in Hz	bw in Hz	Modus	$n_{\overline{FFT}}$	t_{FFT} in s
4_1	100	500	const	27,5k	6,71	MFFT	34	5,06
ID	I_1 in A	I_2 in A	f in Hz	f_s in Hz	bw in Hz	Modus	$n_{\overline{FFT}}$	t_{FFT} in s
4_2	0,38	0,53	250	27,5k	6,71	MFFT	34	5,06

Tabelle A.17: Messparameter Messung 5

	FG		ADcmXL3021					
ID	f in Hz	I in A	f_s in Hz	bw in Hz	Modus	$n_{\overline{FFT}}$	t_{FFT} in s	T in s
5_1	250	0,6	27,5k	6,713867188	AFFT	5	0,744727273	5
5_2	zufällig	zufällig	27,5k	6,713867188	AFFT	5	0,744727273	5

Tabelle A.18: Messparameter Messung 6

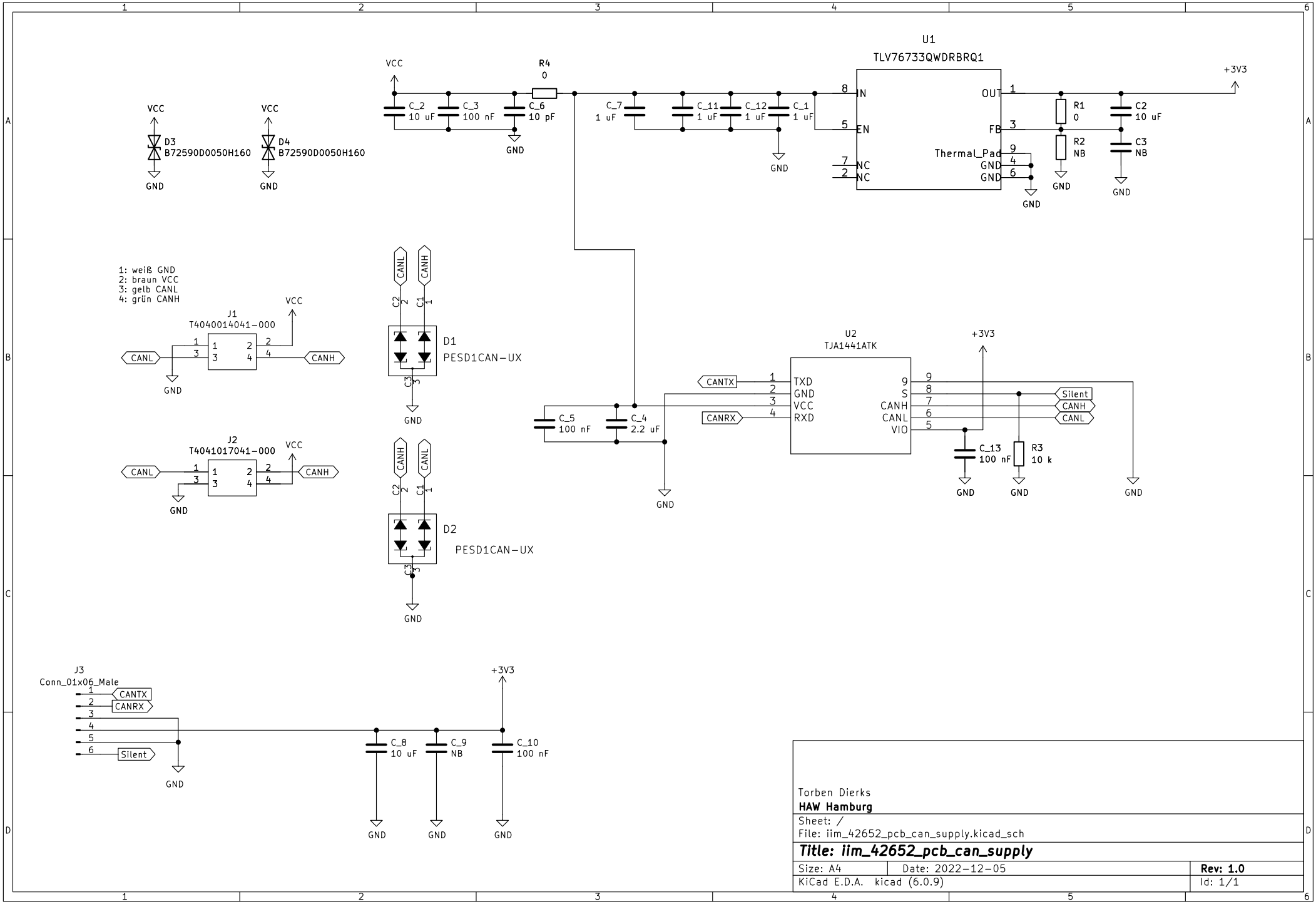
	FG		ADcmXL3021				
ID	f	I in A	f_s in Hz	bw in Hz	Modus	$n_{\overline{FFT}}$	t_{FFT} in s
6_1	250	0,6	27,5k	6,71	MFFT	1	0,15
6_2	250	0,6	27,5k	6,71	MFFT	2	0,30
6_3	250	0,6	27,5k	6,71	MFFT	3	0,45
6_4	250	0,6	27,5k	6,71	MFFT	4	0,60
6_5	250	0,6	27,5k	6,71	MFFT	5	0,74
6_6	250	0,6	27,5k	6,71	MFFT	6	0,89
6_7	250	0,6	27,5k	6,71	MFFT	7	1,04
6_8	250	0,6	27,5k	6,71	MFFT	8	1,19
6_9	250	0,6	27,5k	6,71	MFFT	9	1,34
6_10	250	0,6	27,5k	6,71	MFFT	10	1,49

Tabelle A.19: Messparameter Messung 7

		FG		ADcmXL3021				
ID	Anmerkung	f	I in A	f_s in Hz	bw in Hz	Modus	$n_{\overline{FFT}}$	t_{FFT} in s
7_1		White Noise	const	27,5k	6,71	MFFT	67	9,98
7_2	Verändern der Einstellungen	zufällig	zufällig	27,5k	6,71	MFFT	134	19,96
ID		f	I in A	f_s in Hz	bw in Hz	Modus	t_{MTC} in s	
7_3	An/Aus Schwingprüfanlage	zufällig	zufällig	1718,75	0,42	MTC	2,38	

Tabelle A.20: Messparameter Messung 10

		ADcmXL3021					
ID		f_s in Hz	bw in Hz	Modus	T in s	$n_{\overline{FFT}}$	t_{FFT} in s
10	Vibration erzeugt durch Hydraulikpumpe	27,5k	6,71	AFFT	30	10	1,49

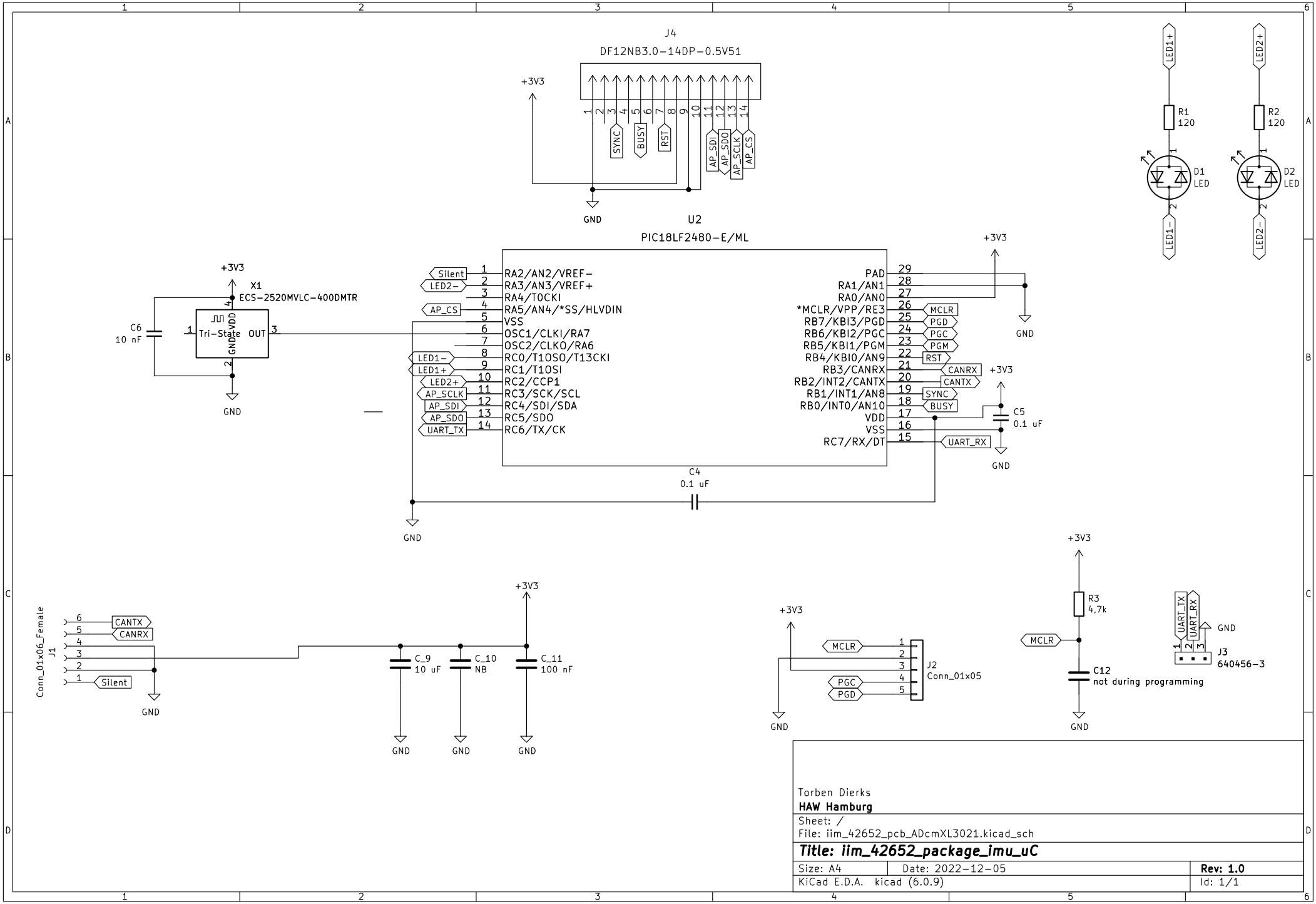


Torben Dierks
HAW Hamburg

Sheet: /
File: iim_42652_pcb_can_supply.kicad_sch

Title: iim_42652_pcb_can_supply

Size: A4	Date: 2022-12-05	**Rev: 1.0**
KiCad E.D.A. kicad (6.0.9)		Id: 1/1


J4
DF12NB3.0-14DP-0.5V51
+3V3
GND
U2
PIC18LF2480-E/ML
X1
ECS-2520MVLC-400DMTR
Tri-State
VDD
GND
OUT
C6
10 nF
C4
0.1 uF
C5
0.1 uF
Silent
LED2-
AP_CS
LED1-
LED1+
LED2+
AP_SCLK
AP_SDI
AP_SDO
UART_TX
RA2/AN2/VREF-
RA3/AN3/VREF+
RA4/T0CKI
RA5/AN4/*SS/HLVDIN
VSS
OSC1/CLKI/RA7
OSC2/CLKO/RA6
RC0/T1OSO/T13CKI
RC1/T1OSI
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
PAD
RA1/AN1
RA0/AN0
*MCLR/VPP/RE3
RB7/KBI3/PGD
RB6/KBI2/PGC
RB5/KBI1/PGM
RB4/KBI0/AN9
RB3/CANRX
RB2/INT2/CANTX
RB1/INT1/AN8
RB0/INT0/AN10
VDD
VSS
RC7/RX/DT
MCLR
PGD
PGC
PGM
RST
CANRX
CANTX
SYNC
BUSY
UART_RX
SYNC
BUSY
RST
AP_SDI
AP_SDO
AP_SCLK
AP_CS
LED1+
LED2+
R1
120
R2
120
D1
LED
D2
LED
LED1-
LED2-
Conn_01x06_Female
J1
CANTX
CANRX
Silent
C_9
10 uF
C_10
NB
C_11
100 nF
J2
Conn_01x05
R3
4,7k
C12
not during programming
J3
640456-3
Torben Dierks
HAW Hamburg
Sheet: /
File: iim_42652_pcb_ADcmXL3021.kicad_sch
Title: iim_42652_package_imu_uC
Size: A4
Date: 2022-12-05
Rev: 1.0
KiCad E.D.A. kicad (6.0.9)
Id: 1/1

A.7 Adapter Schwingprüfanlage

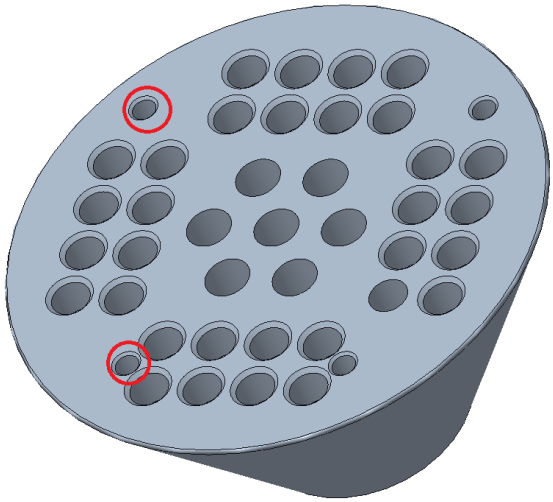


Abbildung A.2: Draufsicht Adapter Schwingprüfanlage mit rot markierten Befestigungspunkten für das Gehäuse

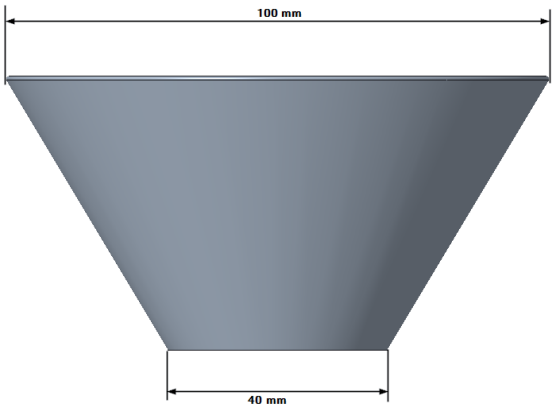
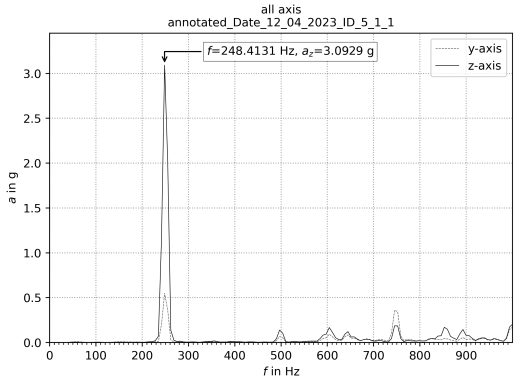


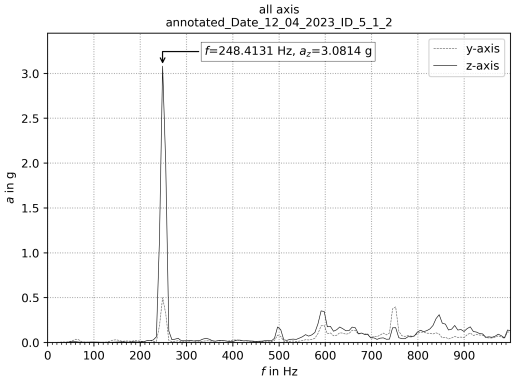
Abbildung A.3: Seitenansicht Adapter Schwingprüfanlage mit Bemaßung

A.8 Messergebnisse

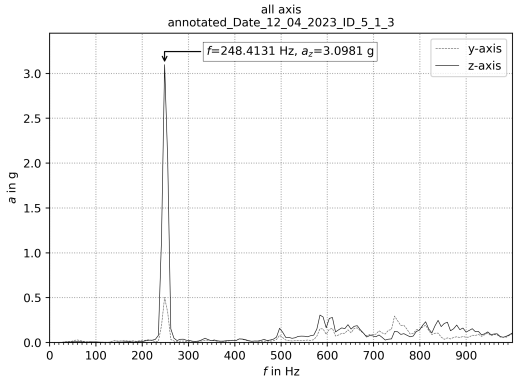
A.8.1 Messung 5_1



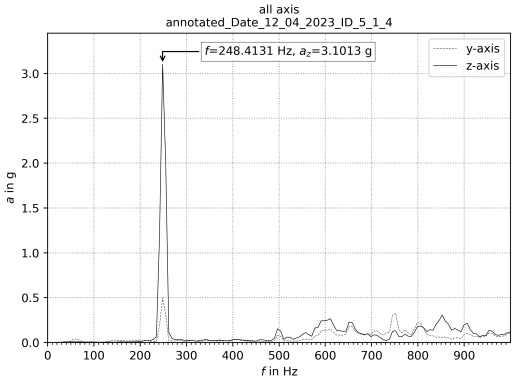
(a) Messung 5_1_1



(b) Messung 5_1_2

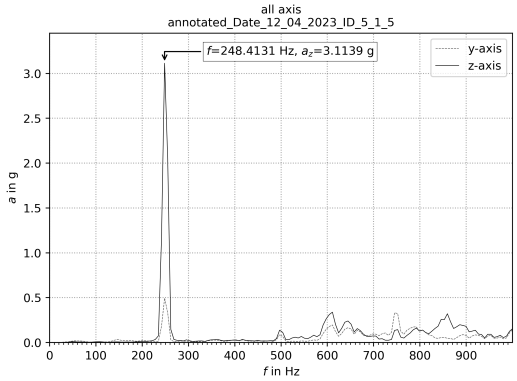


(c) Messung 5_1_3

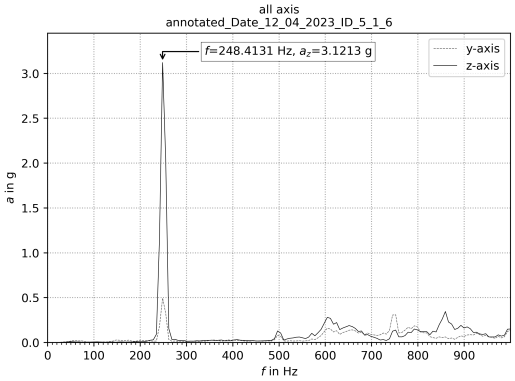


(d) Messung 5_1_4

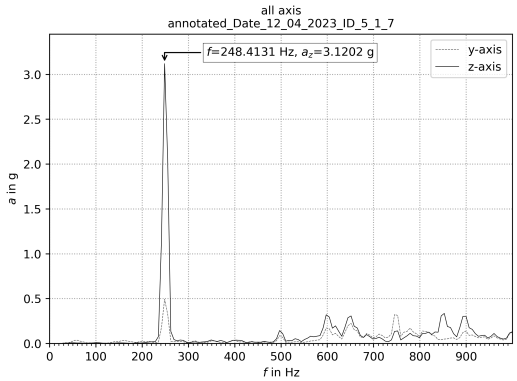
Abbildung A.4: Messung 5_1_1 bis Messung 5_1_4



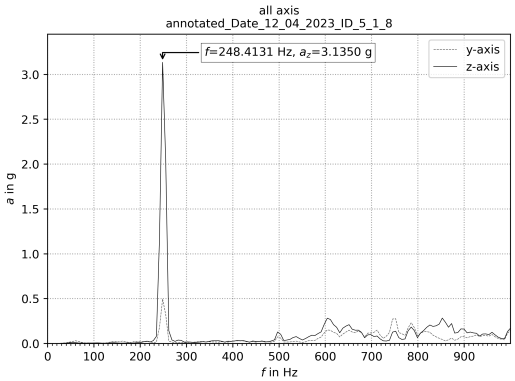
(a) Messung 5_1_5



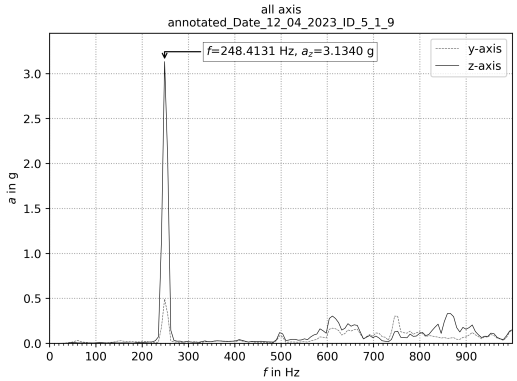
(b) Messung 5_1_6



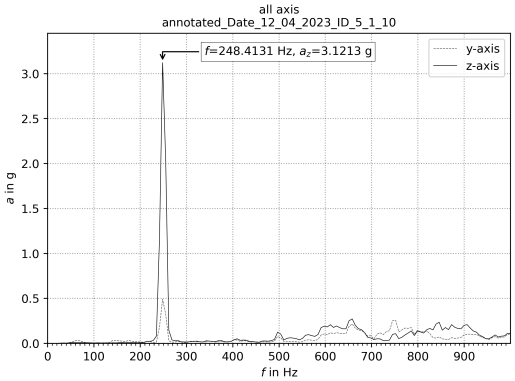
(c) Messung 5_1_7



(d) Messung 5_1_8

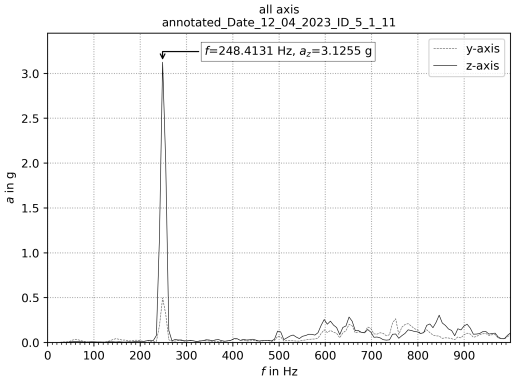


(e) Messung 5_1_9

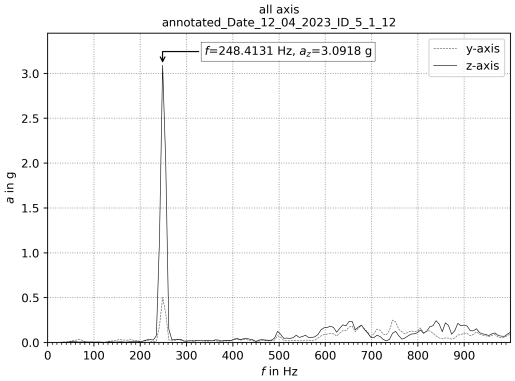


(f) Messung 5_1_10

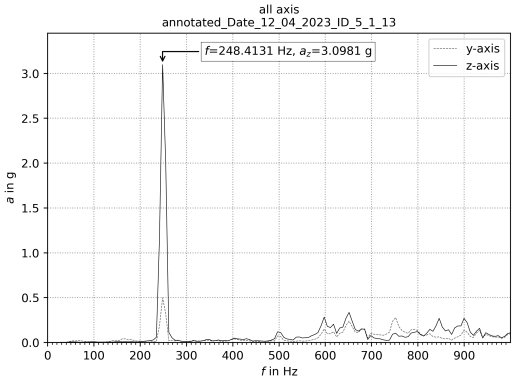
Abbildung A.5: Messung 5_1_5 bis Messung 5_1_10



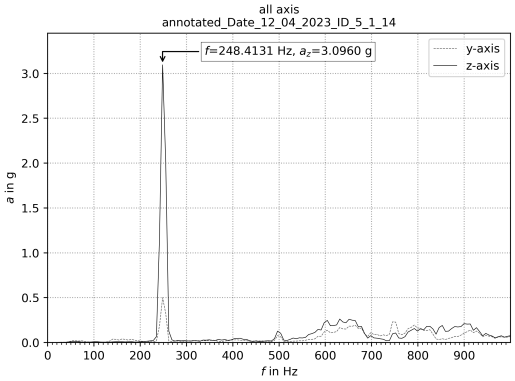
(a) Messung 5_1_11



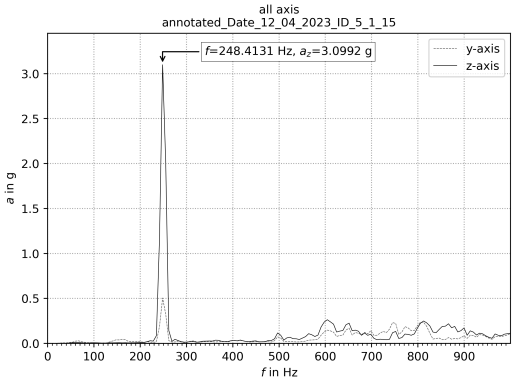
(b) Messung 5_1_12



(c) Messung 5_1_13



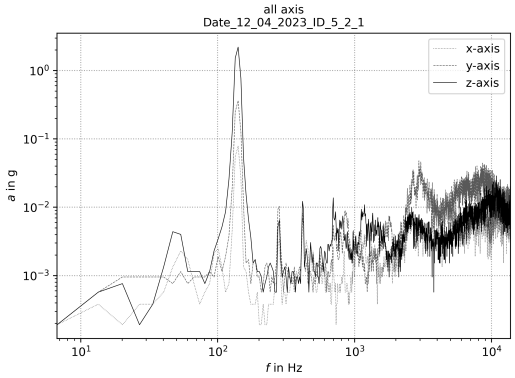
(d) Messung 5_1_14



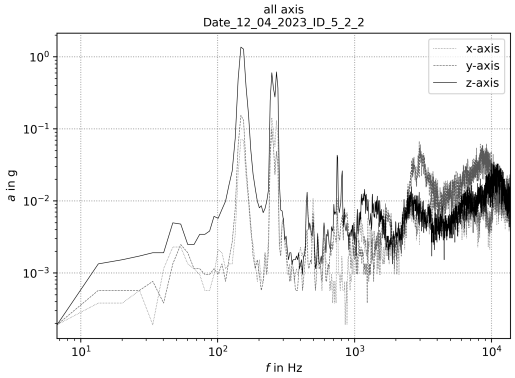
(e) Messung 5_1_15

Abbildung A.6: Messung 5_1_11 bis Messung 5_1_15

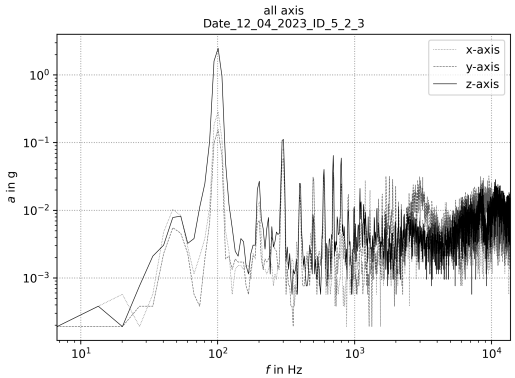
A.8.2 Messung 5_2



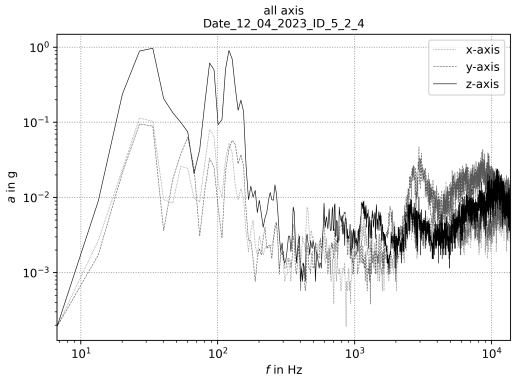
(a) Messung 5_2_1



(b) Messung 5_2_2

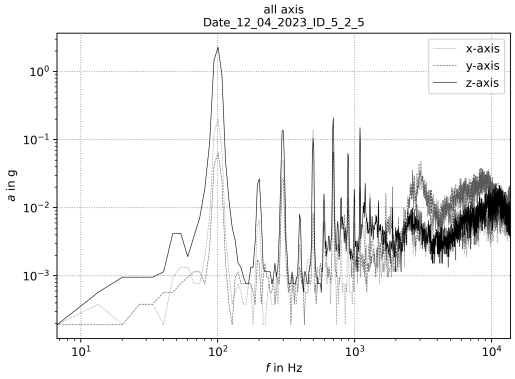


(c) Messung 5_2_3

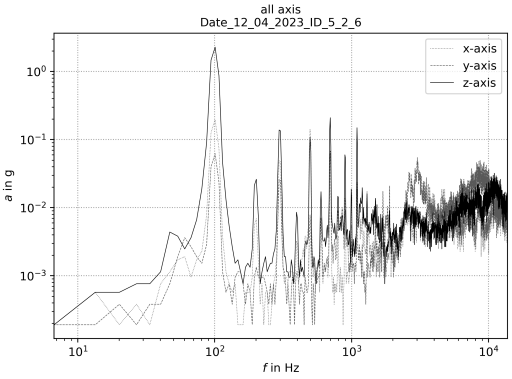


(d) Messung 5_2_4

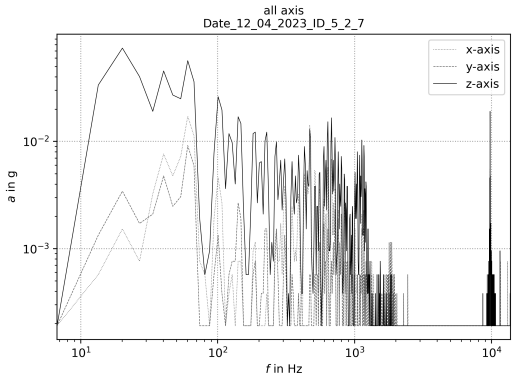
Abbildung A.7: Messung 5_2_1 bis Messung 5_2_4



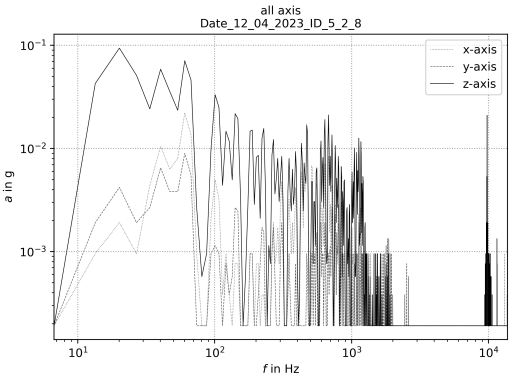
(a) Messung 5_2_5



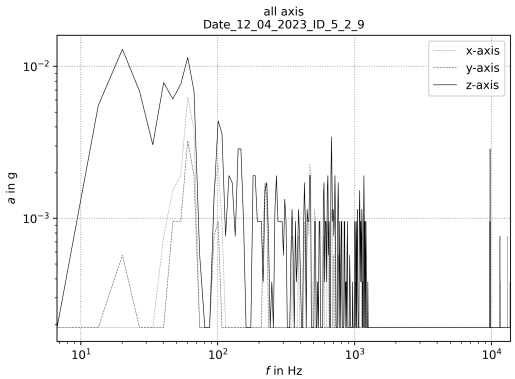
(b) Messung 5_2_6



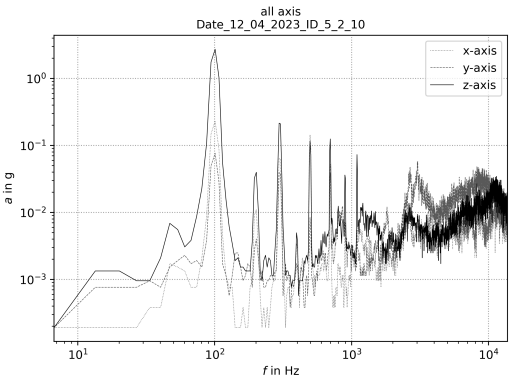
(c) Messung 5_2_7



(d) Messung 5_2_8



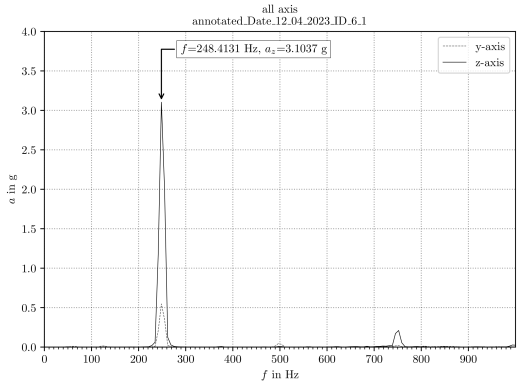
(e) Messung 5_2_9



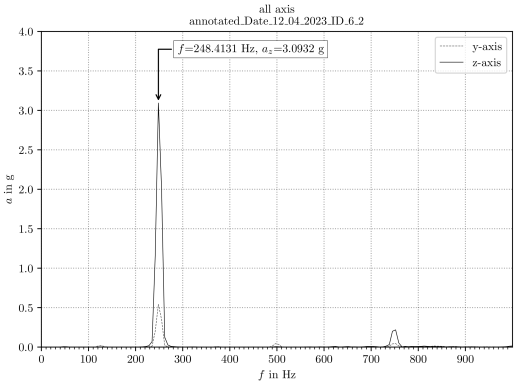
(f) Messung 5_2_10

Abbildung A.8: Messung 5_2_5 bis Messung 5_2_10

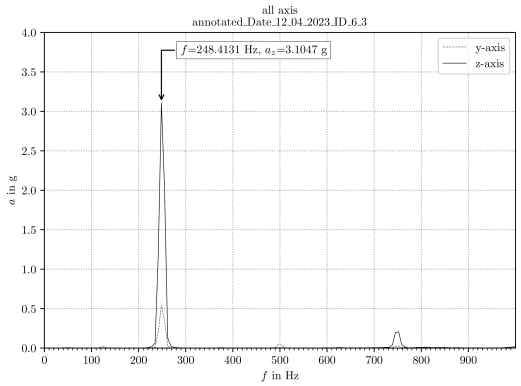
A.8.3 Messung 6



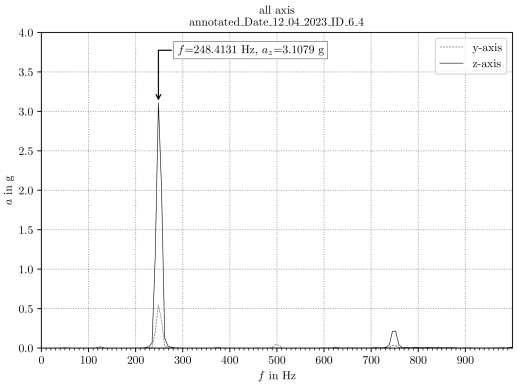
(a) Messung 6_1



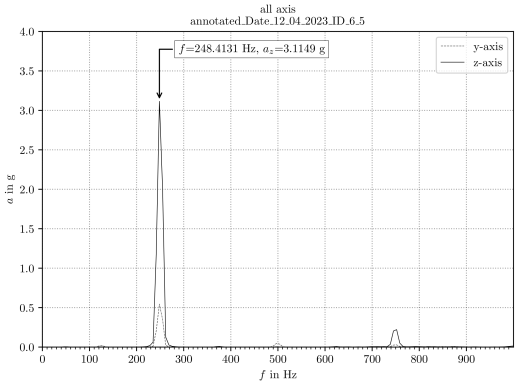
(b) Messung 6_2



(c) Messung 6_3

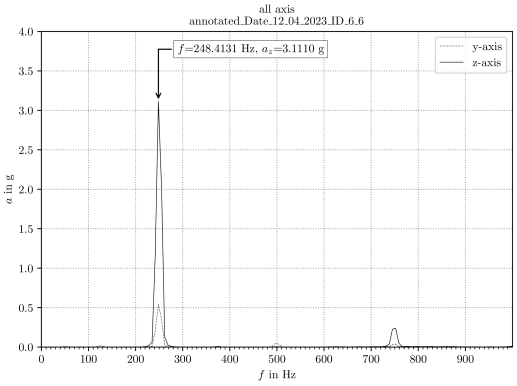


(d) Messung 6_4



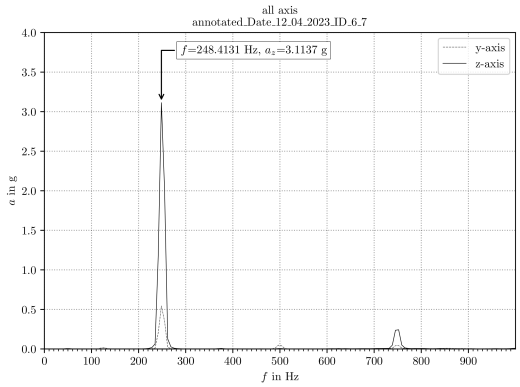
1

(e) Messung 6_5

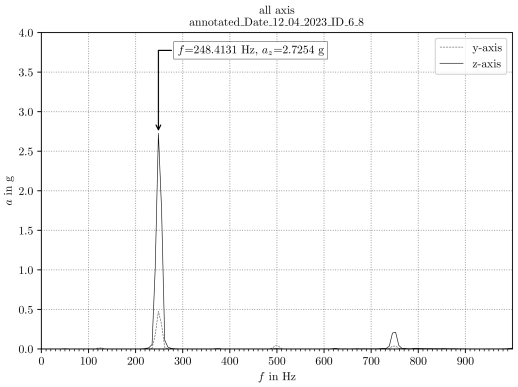


(f) Messung 6_6

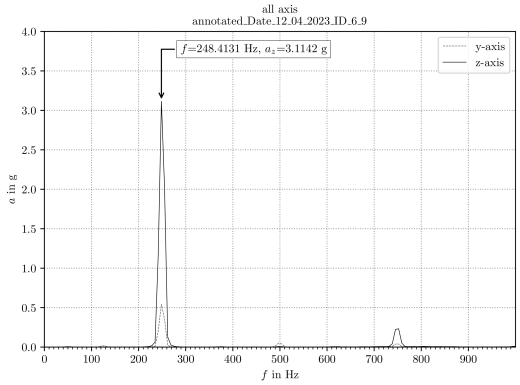
Abbildung A.9: Messung 6_1 bis Messung 6_6



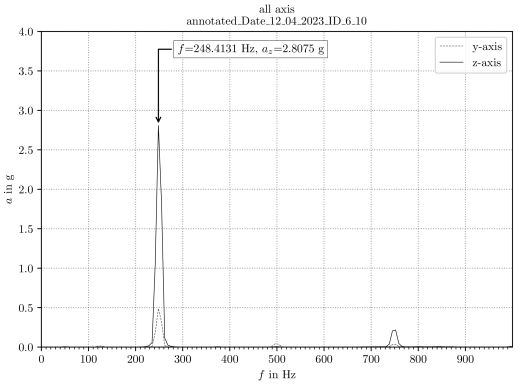
(a) Messung 6_7



(b) Messung 6_8



(c) Messung 6_9



(d) Messung 6_10

Abbildung A.10: Messung 6_7 bis Messung 6_10

A.8.4 Messung 7_1

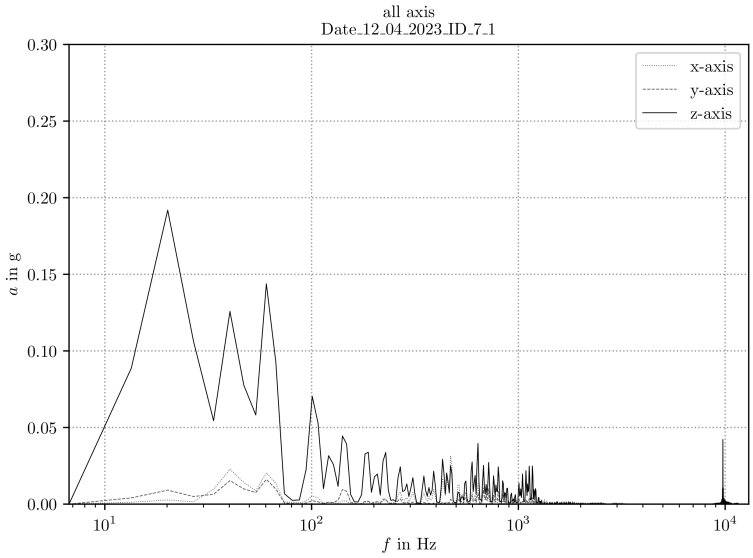


Abbildung A.11: Messung 7_1

A.8.5 Messung 7_2

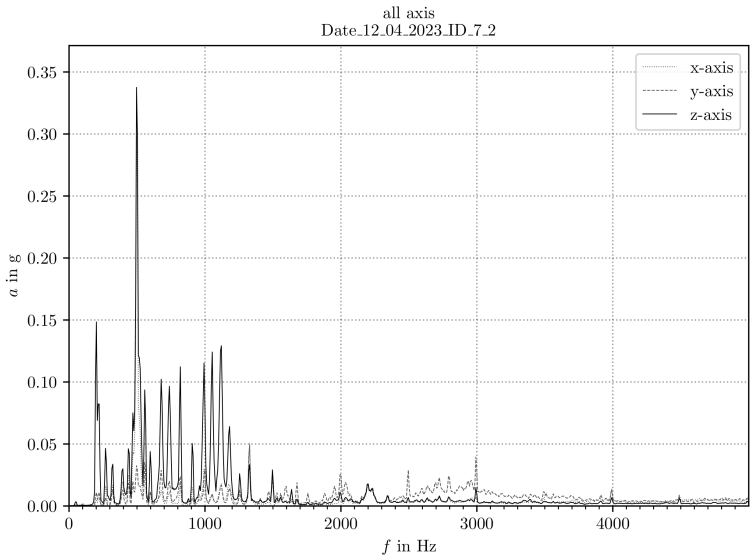


Abbildung A.12: Messung 7_2

A.9 Python Code

A.9.1 definitions.py

```
class imuRegister:
    def __init__(self, pageId, address, defaultValue, currentRegisterValue,
          readFlag, writeFlag):
        self._pageId = int(pageId)
        self._address = int(address)
        self._defaultValue = hex(defaultValue)
        self._currentRegisterValue = hex(currentRegisterValue)
        self._readFlag = readFlag
        self._writeFlag = writeFlag
    @property
    def pageId(self):
        return self._pageId
    @property
    def address(self):
        return self._address
    @property
    def defaultValue(self):
        return self._defaultValue
    @property
    def readFlag(self):
        return self._readFlag
    @property
    def writeFlag(self):
        return self._writeFlag


class defines:
   # defines for sent can message identfier
    CAN_MSG_SENSOR_IDENTFIER_SET_REGISTER = 1
    CAN_MSG_SENSOR_IDENTFIER_READ_REGISTER = 2
    CAN_MSG_TYPE_START_CAPTURE = 3
    CAN_MSG_TYPE_GET_SETTINGS = 4
    CAN_MSG_TYPE_STOPP_AFFT = 5
    CAN_MSG_TYPE_CONFIGURATION_IMU_START = 6
    CAN_MSG_TYPE_CONFIGURATION_IMU_STOP = 7
    CAN_MSG_TYPE_CHANGE_CAN_SETTINGS = 8
    CAN_MSG_TYPE_AUTONULL = 9
    CAN_MSG_TYPE_RESET_SENSOR = 10
    CAN_MSG_TYPE_ESCAPED = 11
    CAN_MSG_TYPE_SAVE_REGISTER = 12
    SR0_ENABLED = 1
    SR1_ENABLED = 2
    SR2_ENABLED = 4
    SR3_ENABLED = 8
    START_FFT_DATA_TRANSMISSON = 146
    STOP_FFT_DATA_TRANSMISSON = 148
    FFT_DATA_TRANSMISSON = 150
    START_MTC_DATA_TRANSMISSON = 226
    STOP_MTC_DATA_TRANSMISSON = 228
    MTC_DATA_TRANSMISSON = 230
    DATA_TRANSMISSON_ERROR = 246
    CAN_BAUD_1_MBPS = 161
    CAN_BAUD_800_kbps = 162
    CAN_BAUD_500_kbps = 163
    CAN_BAUD_500_kbps = 164
    SAMPLES_COUNT_FFT = 2048
    SAMPLES_COUNT_MTC = 4096
```

A.9.2 adcmxl3021_registers.py

```
from definitions import imuRegister


# Register definitions
page_id = imuRegister(pageId=0x00, address=0x00, defaultValue=0x0000,
      currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
temp_out = imuRegister(pageId=0x00, address=0x02, defaultValue=0x8000,
      currentRegisterValue=0x8000, readFlag=True, writeFlag=False)
supply_out = imuRegister(pageId=0x00, address=0x04, defaultValue=0x8000,
      currentRegisterValue=0x8000, readFlag=True, writeFlag=False)
fft_avg1 = imuRegister(pageId=0x00, address=0x06, defaultValue=0x0108,
      currentRegisterValue=0x0108, readFlag=True, writeFlag=True)
fft_avg2 = imuRegister(pageId=0x00, address=0x08, defaultValue=0x0101,
      currentRegisterValue=0x0101, readFlag=True, writeFlag=True)
buf_pntr = imuRegister(pageId=0x00, address=0x0A, defaultValue=0x0000,
      currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
rec_pntr = imuRegister(pageId=0x00, address=0x0C, defaultValue=0x0000,
      currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
x_buf = imuRegister(pageId=0x00, address=0x0E, defaultValue=0x8000,
      currentRegisterValue=0x8000, readFlag=True, writeFlag=True)
y_buf = imuRegister(pageId=0x00, address=0x10, defaultValue=0x8000,
      currentRegisterValue=0x8000, readFlag=True, writeFlag=True)
z_buf = imuRegister(pageId=0x00, address=0x12, defaultValue=0x8000,
      currentRegisterValue=0x8000, readFlag=True, writeFlag=True)
x_anull = imuRegister(pageId=0x00, address=0x14, defaultValue=0x0000,
      currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
y_anull = imuRegister(pageId=0x00, address=0x16, defaultValue=0x0000,
      currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
```

```
17 z_anull = imuRegister(pageId=0x00, address=0x18, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
18 rec_ctrl = imuRegister(pageId=0x00, address=0x1A, defaultValue=0x1102,
       currentRegisterValue=0x1102, readFlag=True, writeFlag=True)
19 rec_prd = imuRegister(pageId=0x00, address=0x1E, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
20 alm_F_low = imuRegister(pageId=0x00, address=0x20, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
21 alm_F_high = imuRegister(pageId=0x00, address=0x22, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
22 alm_x_mag1 = imuRegister(pageId=0x00, address=0x24, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
23 alm_y_mag1 = imuRegister(pageId=0x00, address=0x26, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
24 alm_z_mag1 = imuRegister(pageId=0x00, address=0x28, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
25 alm_x_mag2 = imuRegister(pageId=0x00, address=0x2A, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
26 alm_y_mag2 = imuRegister(pageId=0x00, address=0x2C, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
27 alm_z_mag2 = imuRegister(pageId=0x00, address=0x2E, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
28 alm_pntr = imuRegister(pageId=0x00, address=0x30, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
29 alm_S_mag = imuRegister(pageId=0x00, address=0x32, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
30 alm_ctrl = imuRegister(pageId=0x00, address=0x34, defaultValue=0x0080,
       currentRegisterValue=0x0080, readFlag=True, writeFlag=True)
31 filt_ctrl = imuRegister(pageId=0x00, address=0x38, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
32 avg_cnt = imuRegister(pageId=0x00, address=0x3A, defaultValue=0x7421,
       currentRegisterValue=0x7421, readFlag=True, writeFlag=True)
33 diag_stat = imuRegister(pageId=0x00, address=0x3C, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
34 glob_cmd = imuRegister(pageId=0x00, address=0x3E, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
35 alm_x_stat = imuRegister(pageId=0x00, address=0x40, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
36 alm_y_stat = imuRegister(pageId=0x00, address=0x42, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
37 alm_z_stat = imuRegister(pageId=0x00, address=0x44, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
38 alm_x_peak = imuRegister(pageId=0x00, address=0x46, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
39 alm_y_peak = imuRegister(pageId=0x00, address=0x48, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
40 alm_z_peak = imuRegister(pageId=0x00, address=0x4A, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
41 time_stamp_L = imuRegister(pageId=0x00, address=0x4C, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
42 time_stamp_H = imuRegister(pageId=0x00, address=0x4E, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
43 day_rev = imuRegister(pageId=0x00, address=0x52, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
44 year_mon = imuRegister(pageId=0x00, address=0x54, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
45 prod_id = imuRegister(pageId=0x00, address=0x56, defaultValue=0x0BCD,
       currentRegisterValue=0x0BCD, readFlag=True, writeFlag=True)
46 serial_num = imuRegister(pageId=0x00, address=0x58, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
47 user_scratch = imuRegister(pageId=0x00, address=0x5A, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
48 rec_flash_cnt = imuRegister(pageId=0x00, address=0x5C, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
49 misc_ctrl = imuRegister(pageId=0x00, address=0x64, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
50 rec_info1 = imuRegister(pageId=0x00, address=0x66, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
51 rec_info2 = imuRegister(pageId=0x00, address=0x68, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
52 rec_cnt = imuRegister(pageId=0x00, address=0x6A, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
53 alm_x_freq = imuRegister(pageId=0x00, address=0x6C, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
54 alm_y_freq = imuRegister(pageId=0x00, address=0x6E, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
55 alm_z_freq = imuRegister(pageId=0x00, address=0x70, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
56 stat_pntr = imuRegister(pageId=0x00, address=0x72, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
57 x_statistic = imuRegister(pageId=0x00, address=0x74, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
58 y_statistic = imuRegister(pageId=0x00, address=0x76, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
59 z_statistic = imuRegister(pageId=0x00, address=0x78, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
60 fund_freq = imuRegister(pageId=0x00, address=0x7A, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=True)
61 flash_cnt_L = imuRegister(pageId=0x00, address=0x7C, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
62 flash_cnt_U = imuRegister(pageId=0x00, address=0x7E, defaultValue=0x0000,
       currentRegisterValue=0x0000, readFlag=True, writeFlag=False)
63
64
65 class imuRegisterSettings:
66
```

```
67      def __init__(self):
68          # fft_avg1
69          self.numberOfRecordsSR0 = 8  # value range 0 - 255
70          self.numberOfRecordsSR1 = 1  # value range 0 - 255
71          self.fft_avg1_value = int(fft_avg1.defaultValue, 16)
72          # fft_avg2
73          self.numberOfRecordsSR2 = 1  # value range 0 - 255
74          self.numberOfRecordsSR3 = 1  # value range 0 - 255
75          self.fft_avg2_value = int(fft_avg2.defaultValue, 16)
76          # x_anull
77          self.x_anull_value = int(x_anull.defaultValue, 16)
78          # y_anull
79          self.y_anull_value = int(y_anull.defaultValue, 16)
80          # z_anull
81          self.z_anull_value = int(z_anull.defaultValue, 16)
82          # rec_ctrl
83          self.rts_timeout_enable = 0  # True or False
84          self.window_setting = 1  # 1 hanning, 0 rectangular, 2 flat_top
85          self.sr3_enable = 0  # True or False
86          self.sr2_enable = 0  # True or False
87          self.sr1_enable = 0  # True or False
88          self.sr0_enable = 1  # True or False
89          self.automatic_power_down = 0  # True or False
90          self.enable_compute_stats_mtc = 0  # True or False
91          self.enable_vel_calc = 0  # True or False
92          self.calc_root_sum_square = 0  # True or False
93          self.flash_memory_record_storage_method = 0  # 0 none, 1 alarm_triggered
                , 2 all
94          self.recording_mode = 2  # 0 MFFT, 1 AFFT, 2 MTC, 3 RTS
95          self.rec_ctrl_value = int(rec_ctrl.defaultValue, 16)
96          # rec_prd
97          self.scale = 0  # 0 seconds, 1 minute, 2 hour
98          self.rate = 1  # value range 0 - 255
99          self.rec_prd_value = int(rec_prd.defaultValue, 16)
100         # alm_F_low
101         self.lower_frequency_bin = 0  # value range 0 - 2047
102         self.alm_F_low_value = int(alm_F_low.defaultValue, 16)
103         # alm_F_high
104         self.upper_frequency_bin = 0  # value range 0 - 2047
105         self.alm_F_high_value = int(alm_F_high.defaultValue, 16) # value range 0
                x0000- 0xFFFF
106         # alm_x_mag1
107         self.alm_x_mag1_value = int(alm_x_mag1.defaultValue, 16)  # value range
                0x0000- 0xFFFF
108         # alm_y_mag1
109         self.alm_y_mag1_value = int(alm_y_mag1.defaultValue, 16) # value range 0
                x0000- 0xFFFF
110         # alm_z_mag1
111         self.alm_z_mag1_value = int(alm_z_mag1.defaultValue, 16) # value range 0
                x0000- 0xFFFF
112         # alm_x_mag2
113         self.alm_x_mag2_value = int(alm_x_mag2.defaultValue, 16) # value range 0
                x0000- 0xFFFF
114         # alm_y_mag2
115         self.alm_y_mag2_value = int(alm_y_mag2.defaultValue, 16) # value range 0
                x0000- 0xFFFF
116         # alm_z_mag2
117         self.alm_z_mag2_value = int(alm_z_mag2.defaultValue, 16) # value range 0
                x0000- 0xFFFF
118         # alm_pntr
119         self.sample_rate_setting_for = 0  # 0 SR0 01 SR1 02 SR2 03 SR3
120         self.spectral_band_number = 0  # 1,2,3,4,5,6
121         self.alm_pntr_value = int(alm_pntr.defaultValue, 16)
122         # alm_S_mag
123         self.alm_S_mag_value = int(alm_S_mag.defaultValue, 16)
124         # alm_ctrl
125         self.disable_automatic_clearing_spectral_alarm = 0  # True or False
126         self.response_delay = 0  # range 0 - 15
127         self.latch_diag_stat = 1  # True or False
128         self.enable_alm1_alm2 = 0  # True or False
129         self.system_alm_prio = 0  # 0 trigger for greater alm_s_mag 1 trigger
                for less alm_s_mag
130         self.system_alm_selection = 0  # 0 power 1 temperature
131         self.system_alarm_enable = 0  # True or False
132         self.x_axis_alm = 0  # true or false
133         self.y_axis_alm = 0  # true or false
134         self.z_axis_alm = 0  # true or false
135         self.alm_ctrl_value = int(alm_ctrl.defaultValue, 16)
136         # filt_ctrl
137         self.x_axis_fir_filter_selection = 0  # 0 none, 1 bank_a, 2 bank_b, 3
                bank_c, 4 bank_d, 5 bank_e, 6 bank_f
138         self.y_axis_fir_filter_selection = 0 # 0 none, 1 bank_a, 2 bank_b, 3
                bank_c, 4 bank_d, 5 bank_e, 6 bank_f
139         self.z_axis_fir_filter_selection = 0  # 0 none, 1 bank_a, 2 bank_b, 3
                bank_c, 4 bank_d, 5 bank_e, 6 bank_f
140         self.filt_ctrl_value = int(filt_ctrl.defaultValue, 16)
141         # avg_cnt
142         self.sr0_sample_rate_scale_factor = 1
143         self.sr1_sample_rate_scale_factor = 2
144         self.sr2_sample_rate_scale_factor = 4
145         self.sr3_sample_rate_scale_factor = 7
146         self.avg_cnt_value = int(avg_cnt.defaultValue, 16)
147         # glob_cmd
148         self.clear_auto_null_correction = 0  # 0 or 1
149         self.retrieve_spectral_alarm_band = 0  # 0 or 1
150         self.retrieve_record_data_from_flash = 0  # 0 or 1
```

```
151         self.save_spectral_alarm_band_registers_to_flash = 0  # 0 or 1
152         self.record_start_stop = 0  # 0 or 1
153         self.reset_buf_ptr = 0  # 0 or 1
154         self.clear_spectral_alarm_band_registers_from_flash = 0  # 0 or 1
155         self.clear_all_records = 0  # 0 or 1
156         self.software_reset = 0  # 0 or 1
157         self.save_registers_to_flash_memory = 0  # 0 or 1
158         self.flash_test = 0  # 0 or 1
159         self.clear_diag_stat = 0  # 0 or 1
160         self.factory_reset = 0  # 0 or 1
161         self.self_test = 0  # 0 or 1
162         self.power_down = 0  # 0 or 1
163         self.autonull = 0  # 0 or 1
164         self.glob_cmd_value = int(glob_cmd.defaultValue, 16)
165         # user_scratch
166         self.user_id = 0  # 0x0000 to 0xFFFF
167         self.user_id = int(user_scratch.defaultValue, 16)
168         # misc_ctrl
169         self.enable_sync_sensitivity = 0  # 0 or 1
170         self.transfers_statistics_record_from_flash_memory_to_sram = 0  # 0 or 1
171         self.transfer_statistics_from_sram_to_flash_record = 0  # 0 or 1
172         self.clear_time_domain_statistics = 0  # 0 or 1
173         self.set_self_test = 0  # 0 or 1
174         self.clear_self_test = 0  # 0 or 1
175         self.misc_ctrl_value = int(misc_ctrl.defaultValue, 16)
176
177     """
178     Each method in this class returns two byte hex values corresponding to the
            taken settings.
179     """
180
181     def fft_avg1(self, numberOfRecordsSR1=8, numberOfRecordsSR0=1):
182         self.numberOfRecordsSR0 = numberOfRecordsSR0
183         self.numberOfRecordsSR1 = numberOfRecordsSR1
184         self.fft_avg1_value = ((self.numberOfRecordsSR1 << 8) | self.
                numberOfRecordsSR0)
185         return self.fft_avg1_value
186     def fft_avg2(self, numberOfRecordsSR2=1, numberOfRecordsSR3=1):
187         self.numberOfRecordsSR2 = numberOfRecordsSR2
188         self.numberOfRecordsSR3 = numberOfRecordsSR3
189         self.fft_avg2_value = ((self.numberOfRecordsSR3 << 8) | self.
                numberOfRecordsSR2)
190         return self.fft_avg2_value
191     def x_anull(self, x_anull_value=0x0000):
192         self.x_anull_value = x_anull_value
193         return (self.x_anull_value)
194     def y_anull(self, y_anull_value=0x0000):
195         self.y_anull_value = y_anull_value
196         return (self.y_anull_value)
197     def z_anull(self, z_anull_value=0x0000):
198         self.z_anull_value = z_anull_value
199         return (self.z_anull_value)
200     def rec_ctrl(self, rts_timeout_enable=0, window_setting="hanning",
            sr3_enable=0, sr2_enable=0, sr1_enable=0, sr0_enable=1,
            automatic_power_down=0, enable_compute_stats_mtc=0, enable_vel_calc=0,
            calc_root_sum_square=0, flash_memory_record_storage_method="none",
            recording_mode="MTC"):
201         self.rts_timeout_enable = rts_timeout_enable
202         self.window_setting = window_setting
203         self.sr3_enable = sr3_enable
204         self.sr2_enable = sr2_enable
205         self.sr1_enable = sr1_enable
206         self.sr0_enable = sr0_enable
207         self.automatic_power_down = automatic_power_down
208         self.enable_compute_stats_mtc = enable_compute_stats_mtc
209         self.enable_vel_calc = enable_vel_calc
210         self.calc_root_sum_square = calc_root_sum_square
211         self.flash_memory_record_storage_method =
                flash_memory_record_storage_method
212         self.recording_mode = recording_mode
213         if self.window_setting == "hanning" or self.window_setting == 1:self.
                window_setting = 1
214         elif self.window_setting == "rectangular" or self.window_setting == 0:
                self.window_setting = 0
215         elif self.window_setting == "flat_top" or self.window_setting == 2:self.
                window_setting = 2
216         else:raise ValueError("Invalid input for window_setting")
217         if self.flash_memory_record_storage_method == "none" or self.
                flash_memory_record_storage_method == 0:self.
                flash_memory_record_storage_method = 0
218         elif self.flash_memory_record_storage_method == "alarm_triggered" or
                self.flash_memory_record_storage_method == 1:self.
                flash_memory_record_storage_method = 1
219         elif self.flash_memory_record_storage_method == "all" or self.
                flash_memory_record_storage_method == 2:self.
                flash_memory_record_storage_method = 2
220         else:raise ValueError(    "Invalid input for
                flash_memory_record_storage_method")
221         if self.recording_mode == "MFFT" or self.recording_mode == 0:self.
                recording_mode = 0
222         elif self.recording_mode == "AFFT" or self.recording_mode == 1:self.
                recording_mode = 1
223         elif self.recording_mode == "MTC" or self.recording_mode == 2:self.
                recording_mode = 2
224         elif self.recording_mode == "RTS" or self.recording_mode == 3:self.
                recording_mode = 3
```

```
        else:raise ValueError("Invalid input for recording_mode")
        self.rec_ctrl_value = ((self.rts_timeout_enable << 15) | (self.
            window_setting << 12) | (self.sr3_enable << 11) | (self.sr2_enable
            << 10) | (self.sr1_enable << 9) | (self.sr0_enable << 8) | (self.
            automatic_power_down << 7) | (self.enable_compute_stats_mtc << 6) |
             (self.enable_vel_calc << 5) | (self.calc_root_sum_square << 4) |
            self.flash_memory_record_storage_method << 2) | (self.
            recording_mode << 0))
        return self.rec_ctrl_value
    def rec_prd(self, scale="second", rate=1):
        self.scale = scale
        self.rate = rate
        if self.scale == "second" or self.scale == 0:self.scale = 0
        elif self.scale == "minute" or self.scale == 1:self.scale = 1
        elif self.scale == "hour" or self.scale == 2:self.scale = 2
        else:raise ValueError("Invalid input for scale")
        self.rec_prd_value = ((self.scale << 8) | (self.rate))
        return self.rec_prd_value
    def alm_F_low(self, lower_frequency_bin=0):
        self.lower_frequency_bin = lower_frequency_bin
        self.alm_F_low_value = (self.lower_frequency_bin)
        return self.alm_F_low_value
    def alm_F_high(self, upper_frequency_bin=0):
        self.upper_frequency_bin = upper_frequency_bin
        self.alm_F_high_value = (self.upper_frequency_bin)
        return self.alm_F_high_value
    def alm_x_mag1(self, alm_x_mag1_value=0x0000):
        self.alm_x_mag1_value = (alm_x_mag1_value)
        return self.alm_x_mag1_value
    def alm_y_mag1(self, alm_y_mag1_value=0x0000):
        self.alm_y_mag1_value = (alm_y_mag1_value)
        return self.alm_y_mag1_value
    def alm_z_mag1(self, alm_z_mag1_value=0x0000):
        self.alm_z_mag1_value = (alm_z_mag1_value)
        return self.alm_z_mag1_value
    def alm_x_mag2(self, alm_x_mag2_value=0x0000):
        self.alm_x_mag2_value = (alm_x_mag2_value)
        return self.alm_x_mag2_value
    def alm_y_mag2(self, alm_y_mag2_value=0x0000):
        self.alm_y_mag2_value = (alm_y_mag2_value)
        return self.alm_y_mag2_value
    def alm_z_mag2(self, alm_z_mag2_value=0x0000):
        self.alm_z_mag2_value = (alm_z_mag2_value)
        return self.alm_z_mag2_value
    def alm_pntr(self, sample_rate_setting_for=0, spectral_band_number=0):
        self.sample_rate_setting_for = sample_rate_setting_for  # 0 SR0 01 SR1
             02 SR2 03 SR3
        self.spectral_band_number = spectral_band_number  # 1,2,3,4,5,6
        self.alm_pntr_value = (self.sample_rate_setting_for << 8) | (self.
            spectral_band_number)
        return self.alm_pntr_value
    def alm_S_mag(self, alm_S_mag_value=0x0000):
        self.alm_S_mag_value = (alm_S_mag_value)
        return self.alm_S_mag_value
    def alm_ctrl(self, disable_automatic_clearing_spectral_alarm=0,
        response_delay=0, latch_diag_stat=1, enable_alm1_alm2=0,
        system_alm_prio=0, system_alm_selection=0, system_alarm_enable=0,
        x_axis_alm=0, y_axis_alm=0, z_axis_alm=0):
        self.disable_automatic_clearing_spectral_alarm =
            disable_automatic_clearing_spectral_alarm  # True or False
        self.response_delay = response_delay  # range 0 - 15
        self.latch_diag_stat = latch_diag_stat  # True or False
        self.enable_alm1_alm2 = enable_alm1_alm2  # True or False
        self.system_alm_prio = system_alm_prio # 0 trigger for greater alm_s_mag
             1 trigger for less alm_s_mag
        self.system_alm_selection = system_alm_selection  # 0 power 1
            temperature
        self.system_alarm_enable = system_alarm_enable  # True or False
        self.x_axis_alm = x_axis_alm  # true or false
        self.y_axis_alm = y_axis_alm  # true or false
        self.z_axis_alm = z_axis_alm  # true or false
        self.alm_ctrl_value = (self.disable_automatic_clearing_spectral_alarm <<
             12) | (self.response_delay << 8) | (self.latch_diag_stat << 7) | (
            self.enable_alm1_alm2 << 6) | (self.system_alm_prio << 5) | (self.
            system_alm_selection << 4) | (self.system_alarm_enable << 3) | (
            self.z_axis_alm << 2) | (self.y_axis_alm << 1) | (self.x_axis_alm)
        return self.alm_ctrl_value
    def filt_ctrl(self, x_axis_fir_filter_selection="none",
        y_axis_fir_filter_selection="none", z_axis_fir_filter_selection="none")
        :
        self.x_axis_fir_filter_selection = x_axis_fir_filter_selection  #
            factory default value of 0x0000 means no filter is used
        self.y_axis_fir_filter_selection = y_axis_fir_filter_selection
        self.z_axis_fir_filter_selection = z_axis_fir_filter_selection
        if self.x_axis_fir_filter_selection == "none" or self.
            x_axis_fir_filter_selection == 0:self.x_axis_fir_filter_selection =
             0
        elif self.x_axis_fir_filter_selection == "bank_a" or self.
            x_axis_fir_filter_selection == 1:self.x_axis_fir_filter_selection =
             1
        elif self.x_axis_fir_filter_selection == "bank_b" or self.
            x_axis_fir_filter_selection == 2:self.x_axis_fir_filter_selection =
             2
        elif self.x_axis_fir_filter_selection == "bank_c" or self.
            x_axis_fir_filter_selection == 3:self.x_axis_fir_filter_selection =
             3
```

```
        elif self.x_axis_fir_filter_selection == "bank_d" or self.x_axis_fir_filter_selection == 4:self.x_axis_fir_filter_selection = 4
        elif self.x_axis_fir_filter_selection == "bank_e" or self.x_axis_fir_filter_selection == 5:self.x_axis_fir_filter_selection = 5
        elif self.x_axis_fir_filter_selection == "bank_f" or self.x_axis_fir_filter_selection == 6:self.x_axis_fir_filter_selection = 6
        else:raise ValueError("Invalid input for x_axis_fir_filter_selection")
        if self.y_axis_fir_filter_selection == "none" or self.y_axis_fir_filter_selection == 0:self.y_axis_fir_filter_selection = 0
        elif self.y_axis_fir_filter_selection == "bank_a" or self.y_axis_fir_filter_selection == 1:self.y_axis_fir_filter_selection = 1
        elif self.y_axis_fir_filter_selection == "bank_b" or self.y_axis_fir_filter_selection == 2:self.y_axis_fir_filter_selection = 2
        elif self.y_axis_fir_filter_selection == "bank_c" or self.y_axis_fir_filter_selection == 3:self.y_axis_fir_filter_selection = 3
        elif self.y_axis_fir_filter_selection == "bank_d" or self.y_axis_fir_filter_selection == 4:self.y_axis_fir_filter_selection = 4
        elif self.y_axis_fir_filter_selection == "bank_e" or self.y_axis_fir_filter_selection == 5:self.y_axis_fir_filter_selection = 5
        elif self.y_axis_fir_filter_selection == "bank_f" or self.y_axis_fir_filter_selection == 6:self.y_axis_fir_filter_selection = 6
        else:raise ValueError("Invalid input for y_axis_fir_filter_selection")
        if self.z_axis_fir_filter_selection == "none" or self.z_axis_fir_filter_selection == 0:self.z_axis_fir_filter_selection = 0
        elif self.z_axis_fir_filter_selection == "bank_a" or self.z_axis_fir_filter_selection == 1:self.z_axis_fir_filter_selection = 1
        elif self.z_axis_fir_filter_selection == "bank_b" or self.z_axis_fir_filter_selection == 2:self.z_axis_fir_filter_selection = 2
        elif self.z_axis_fir_filter_selection == "bank_c" or self.z_axis_fir_filter_selection == 3:self.z_axis_fir_filter_selection = 3
        elif self.z_axis_fir_filter_selection == "bank_d" or self.z_axis_fir_filter_selection == 4:self.z_axis_fir_filter_selection = 4
        elif self.z_axis_fir_filter_selection == "bank_e" or self.z_axis_fir_filter_selection == 5:self.z_axis_fir_filter_selection = 5
        elif self.z_axis_fir_filter_selection == "bank_f" or self.z_axis_fir_filter_selection == 6:self.z_axis_fir_filter_selection = 6
        else:raise ValueError("Invalid input for z_axis_fir_filter_selection")
        self.filt_ctrl_value = ((self.z_axis_fir_filter_selection << 8) | (self.y_axis_fir_filter_selection << 4) | (self.x_axis_fir_filter_selection))
        return self.filt_ctrl_value
    def avg_cnt(self, sr0_sample_rate_scale_factor=1, sr1_sample_rate_scale_factor=2, sr2_sample_rate_scale_factor=4, sr3_sample_rate_scale_factor=7):
        self.sr0_sample_rate_scale_factor = sr0_sample_rate_scale_factor
        self.sr1_sample_rate_scale_factor = sr1_sample_rate_scale_factor
        self.sr2_sample_rate_scale_factor = sr2_sample_rate_scale_factor
        self.sr3_sample_rate_scale_factor = sr3_sample_rate_scale_factor
        self.avg_cnt_value = ((self.sr3_sample_rate_scale_factor << 12) | (self.sr2_sample_rate_scale_factor << 8) | (self.sr1_sample_rate_scale_factor << 4) | (self.sr0_sample_rate_scale_factor))
        return self.avg_cnt_value
    def glob_cmd(self, clear_auto_null_correction=0, retrieve_spectral_alarm_band=0, retrieve_record_data_from_flash=0, save_spectral_alarm_band_registers_to_flash=0, record_start_stop=0, reset_buf_ptr=0, clear_spectral_alarm_band_registers_from_flash=0, clear_all_records=0, software_reset=0, save_registers_to_flash_memory=0, flash_test=0, clear_diag_stat=0, factory_reset=0, self_test=0, power_down=0, autonull=0):
        self.clear_auto_null_correction = clear_auto_null_correction
        self.retrieve_spectral_alarm_band = retrieve_spectral_alarm_band
        self.retrieve_record_data_from_flash = retrieve_record_data_from_flash
        self.save_spectral_alarm_band_registers_to_flash = save_spectral_alarm_band_registers_to_flash
        self.record_start_stop = record_start_stop
        self.reset_buf_ptr = reset_buf_ptr
        self.clear_spectral_alarm_band_registers_from_flash = clear_spectral_alarm_band_registers_from_flash
        self.clear_all_records = clear_all_records
        self.software_reset = software_reset
        self.save_registers_to_flash_memory = save_registers_to_flash_memory
        self.flash_test = flash_test
        self.clear_diag_stat = clear_diag_stat
        self.factory_reset = factory_reset
        self.self_test = self_test
        self.power_down = power_down
        self.autonull = autonull
```

```
338         self.glob_cmd_value = ((self.clear_auto_null_correction << 15) | (self.
                retrieve_spectral_alarm_band << 14) | (self.
                retrieve_record_data_from_flash << 13) | (self.
                save_spectral_alarm_band_registers_to_flash << 12) | (self.
                record_start_stop << 11) | (self.reset_buf_ptr << 10) | (self.
                clear_spectral_alarm_band_registers_from_flash << 9) | (self.
                clear_all_records << 8) | (self.software_reset << 7) | (self.
                save_registers_to_flash_memory << 6) | (self.flash_test << 5) | (
                self.clear_diag_stat << 4) | (self.factory_reset << 3) | (self.
                self_test << 2) | (self.power_down << 1) | (self.autonull))
339         return self.glob_cmd_value
340     def user_scratch(self, user_id=0x0000):
341         self.user_id = user_id # create own unique user id
342         return (self.user_id)
343     def misc_ctrl(self, enable_sync_sensitivity=0,
              transfers_statistics_record_from_flash_memory_to_sram=0,
              transfer_statistics_from_sram_to_flash_record=0,
              clear_time_domain_statistics=0, set_self_test=0, clear_self_test=0):
344         self.enable_sync_sensitivity = enable_sync_sensitivity
345         self.transfers_statistics_record_from_flash_memory_to_sram =
                transfers_statistics_record_from_flash_memory_to_sram
346         self.transfer_statistics_from_sram_to_flash_record =
                transfer_statistics_from_sram_to_flash_record
347         self.clear_time_domain_statistics = clear_time_domain_statistics
348         self.set_self_test = set_self_test
349         self.clear_self_test = clear_self_test
350         self.misc_ctrl_value = ((self.enable_sync_sensitivity << 12) | (self.
                transfers_statistics_record_from_flash_memory_to_sram << 10) | (
                self.transfer_statistics_from_sram_to_flash_record << 9) | (self.
                clear_time_domain_statistics << 8) | (self.set_self_test << 3) | (
                self.clear_self_test << 2))
351         return self.misc_ctrl_value
```

A.9.3 configure_sensor.py

```
1  #!/usr/bin/env python
2  import can
3  import time
4  from definitions import defines
5  import adcmxl3021_registers
6  import json
7  from adcmxl3021_registers import imuRegisterSettings
8
9
10 def send_one(bus, msg):
11     """Sends a single message."""
12     try:
13
14         bus.send(msg)
15         print(msg)
16         print(f"Message sent on {bus.channel_info}")
17
18     except can.CanError:
19
20         print("Message NOT sent")
21
22
23 def receive_all(bus):
24     """Receives all messages and prints them to the console until Ctrl+C is
           pressed."""
25     # set to read-only, only supported on some interfaces
26     try:
27         while True:
28             msg = bus.recv(0.0)
29             if msg is not None:
30                 print(msg)
31     except KeyboardInterrupt:
32         pass  # exit normally
33
34
35 def splitSettings(value):
36     dataHigh = value >> 8
37     dataLow = value & 0xFF
38     return dataHigh, dataLow
39
40
41 def buildWriteRegMsg(address, pageID, data, canID):
42     dataHigh, dataLow = splitSettings(data)
43     msg = can.Message(arbitration_id=canID, data=[ defines.
           CAN_MSG_SENSOR_IDENTFIER_SET_REGISTER, pageID, address, dataHigh,
           dataLow], is_extended_id=False, is_fd=False)
44     return msg
45
46
47 def buildReadRegMsg(address, pageID, canID):
48     msg = can.Message(arbitration_id=canID, data=[defines.
           CAN_MSG_SENSOR_IDENTFIER_READ_REGISTER, pageID, address],
           is_extended_id=False, is_fd=False)
49     return msg
50
51
52 def startRecordingIMU(bus):
53
```

```
    msg = can.Message(arbitration_id=canID, data=[defines.
          CAN_MSG_SENSOR_IDENTFIER_SET_REGISTER,adcmxl3021_registers.glob_cmd.
          pageId, adcmxl3021_registers.glob_cmd.address, 0x08, 0x00],
          is_extended_id=False, is_fd=False)
    send_one(bus, msg)


def checkForValueChanges(bus):
    jsonSettingsFile = json.load(open('imuSettingsJSON.json'))

    msgStartConfig = can.Message(arbitration_id=canID, data=[defines.
          CAN_MSG_TYPE_CONFIGURATION_IMU_START], is_extended_id=False, is_fd=
          False)
    send_one(bus, msgStartConfig)
    while True:
        msg = bus.recv(None)
        # FFT
        if msg.data[0] == defines.CAN_MSG_TYPE_ESCAPED:
            msg = buildWriteRegMsg(address=adcmxl3021_registers.avg_cnt.address,
                  pageID=adcmxl3021_registers.avg_cnt.pageId, canID=canID, data=0
                  x7421)
            send_one(bus, msg)
            msg = buildWriteRegMsg(address=adcmxl3021_registers.rec_ctrl.address
                  ,pageID=adcmxl3021_registers.rec_ctrl.pageId, canID=canID, data
                  =0x1102)
            send_one(bus, msg)
            time.sleep(3)

            for i in jsonSettingsFile:
                setValue = jsonSettingsFile[i][0].get("reg_value")
                address = (jsonSettingsFile[i][0]["register"].get("address"))
                pageID = (jsonSettingsFile[i][0]["register"].get("pageID"))
                msg = buildWriteRegMsg(address, pageID, setValue, 0x500)
                send_one(bus, msg)
                time.sleep(0.1)

            endMsg = can.Message(arbitration_id=canID, data=[defines.
                  CAN_MSG_TYPE_AUTONULL], is_extended_id=False, is_fd=False)
            send_one(bus, endMsg)

            endMsg = can.Message(arbitration_id=canID, data=[defines.
                  CAN_MSG_TYPE_SAVE_REGISTER], is_extended_id=False, is_fd=False)
            send_one(bus, endMsg)
            time.sleep(2)
            endMsg = can.Message(arbitration_id=canID, data=[defines.
                  CAN_MSG_TYPE_CONFIGURATION_IMU_STOP],is_extended_id=False,
                  is_fd=False)
            send_one(bus, endMsg)

            return

if __name__ == "__main__":
    # can bus
    canID = 0x500
    bus = can.Bus(interface='ixxat', channel='0', bitrate=500000)
    checkForValueChanges(bus)
```

A.9.4 imuSettingsJSON.py

```
{
    "fft_avg1": [
        {
            "input_values": "SR0, SR1 # value range 0 - 255",
            "numberOfRecordsSR0": 2,
            "numberOfRecordsSR1": 255,
            "register": {
                "pageID": 0,
                "address": 6,
                "default_value": "0x108"
            },
            "reg_value": 65282
        }
    ],
    "fft_avg2": [
        {
            "input_values": "SR1, SR3 # value range 0 - 255",
            "numberOfRecordsSR0": 255,
            "numberOfRecordsSR1": 255,
            "register": {
                "pageID": 0,
                "address": 8,
                "default_value": "0x101"
            },
            "reg_value": 65535
        }
    ],
    "x_anull": [
        {
            "register": {
                "pageID": 0,
                "address": 20,
                "default_value": "0x0"
            },
            "reg_value": 0
```

```
        }
    ],
    "y_anull": [
        {
            "register": {
                "pageID": 0,
                "address": 22,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "z_anull": [
        {
            "register": {
                "pageID": 0,
                "address": 24,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "rec_ctrl": [
        {
            "input values": "all except window_setting and recording mode True
                  or False 0, window_setting 1 hanning, 0 rectangular, 2
                 flat_top, recording_mode 0 MFFT, 1 AFFT, 2 MTC, 3 RTS",
            "rts_timeout_enable": 0,
            "window_setting": 1,
            "sr3_enable": 0,
            "sr2_enable": 0,
            "sr1_enable": 0,
            "sr0_enable": 1,
            "automatic_power_down": 0,
            "enable_compute_stats_mtc": 0,
            "enable_vel_calc": 0,
            "calc_root_sum_square": 0,
            "flash_memory_record_storage_method": 0,
            "recording_mode": 0,
            "register": {
                "pageID": 0,
                "address": 26,
                "default_value": "0x1102"
            },
            "reg_value": 4352
        }
    ],
    "rec_prd": [
        {
            "input values": "scale 0 seconds, 1 minute, 2 hour, rate value range
                  0 - 255",
            "scale": 0,
            "rate": 30,
            "register": {
                "pageID": 0,
                "address": 30,
                "default_value": "0x0"
            },
            "reg_value": 30
        }
    ],
    "alm_F_low": [
        {
            "input values": "# value range 0 - 2047",
            "lower_frequency_bin": 0,
            "register": {
                "pageID": 0,
                "address": 32,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "alm_F_high": [
        {
            "input values": "# value range 0 - 2047",
            "upper_frequency_bin": 0,
            "register": {
                "pageID": 0,
                "address": 34,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "alm_x_mag1": [
        {
            "register": {
                "pageID": 0,
                "address": 36,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "alm_y_mag1": [
```

```
        {
            "register": {
                "pageID": 0,
                "address": 38,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "alm_z_mag1": [
        {
            "register": {
                "pageID": 0,
                "address": 40,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "alm_x_mag2": [
        {
            "register": {
                "pageID": 0,
                "address": 42,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "alm_y_mag2": [
        {
            "register": {
                "pageID": 0,
                "address": 44,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "alm_z_mag2": [
        {
            "register": {
                "pageID": 0,
                "address": 46,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "alm_pntr": [
        {
            "sample_rate_setting_for": 0,
            "spectral_band_number": 0,
            "register": {
                "pageID": 0,
                "address": 48,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "alm_S_mag": [
        {
            "register": {
                "pageID": 0,
                "address": 50,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "alm_ctrl": [
        {
            "disable_automatic_clearing_spectral_alarm": 0,
            "response_delay": 0,
            "latch_diag_stat": 1,
            "enable_alm1_alm2": 0,
            "system_alm_prio": 0,
            "system_alm_selection": 0,
            "system_alarm_enable": 0,
            "x_axis_alm": 0,
            "y_axis_alm": 0,
            "z_axis_alm": 0,
            "register": {
                "pageID": 0,
                "address": 52,
                "default_value": "0x80"
            },
            "reg_value": 128
        }
    ],
    "filt_ctrl": [
        {
            "input values": "all: 0 none, 1 bank_a, 2 bank_b, 3 bank_c, 4 bank_d
                 , 5 bank_e, 6 bank_f",
            "x_axis_fir_filter_selection": 0,
```

```
            "y_axis_fir_filter_selection": 0,
            "z_axis_fir_filter_selection": 0,
            "register": {
                "pageID": 0,
                "address": 56,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "avg_cnt": [
        {
            "input values": "all see table 87 datasheet adcmxl3021",
            "sr0_sample_rate_scale_factor": 6,
            "sr1_sample_rate_scale_factor": 1,
            "sr2_sample_rate_scale_factor": 1,
            "sr3_sample_rate_scale_factor": 1,
            "register": {
                "pageID": 0,
                "address": 58,
                "default_value": "0x7421"
            },
            "reg_value": 4374
        }
    ],
    "glob_cmd": [
        {
            "input values": "all True 1 or False 0",
            "clear_auto_null_correction": 0,
            "retrieve_spectral_alarm_band": 0,
            "retrieve_record_data_from_flash": 0,
            "save_spectral_alarm_band_registers_to_flash": 0,
            "record_start_stop": 0,
            "reset_buf_ptr": 0,
            "clear_spectral_alarm_band_registers_from_flash": 0,
            "clear_all_records": 0,
            "software_reset": 0,
            "save_registers_to_flash_memory": 0,
            "flash_test": 0,
            "clear_diag_stat": 0,
            "factory_reset": 0,
            "self_test": 0,
            "power_down": 0,
            "autonull": 0,
            "register": {
                "pageID": 0,
                "address": 62,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "user_scratch": [
        {
            "input values": "0x0000 to 0xFFFF",
            "user_id": 0,
            "register": {
                "pageID": 0,
                "address": 90,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ],
    "misc_ctrl": [
        {
            "input values": "all True 1 or False 0",
            "enable_sync_sensitivity": 0,
            "transfers_statistics_record_from_flash_memory_to_sram": 0,
            "transfer_statistics_from_sram_to_flash_record": 0,
            "clear_time_domain_statistics": 0,
            "set_self_test": 0,
            "clear_self_test": 0,
            "register": {
                "pageID": 0,
                "address": 100,
                "default_value": "0x0"
            },
            "reg_value": 0
        }
    ]
}
```


A.9.5 jsonCreator.py

```
import json
import adcmxl3021_registers
from adcmxl3021_registers import imuRegisterSettings

settingsIMU = imuRegisterSettings()
settingsIMU.rec_ctrl(sr0_enable=1, sr1_enable=0, sr2_enable=0, sr3_enable=0,
      recording_mode="MTC")
settingsIMU.avg_cnt(sr0_sample_rate_scale_factor=7, sr1_sample_rate_scale_factor
      =2,sr2_sample_rate_scale_factor=3, sr3_sample_rate_scale_factor=4)
```

```
settingsIMU.fft_avg1(numberOfRecordsSR0=2, numberOfRecordsSR1=4)
settingsIMU.fft_avg2(numberOfRecordsSR2=6, numberOfRecordsSR3=8)
settingsIMU.rec_prd(scale="second", rate=10)

imuSettingsJSON = {
    "fft_avg1": [
        {
            "input_values": "SR0, SR1 # value range 0 - 255",
            "numberOfRecordsSR0": settingsIMU.numberOfRecordsSR0,
            "numberOfRecordsSR1": settingsIMU.numberOfRecordsSR1,
            "register": {
                "pageID": adcmxl3021_registers.fft_avg1.pageId,
                "address": adcmxl3021_registers.fft_avg1.address,
                "default_value": adcmxl3021_registers.fft_avg1.defaultValue
            },
            "reg_value": settingsIMU.fft_avg1_value
        }
    ],
    "fft_avg2": [
        {
            "input_values": "SR1, SR3 # value range 0 - 255",
            "numberOfRecordsSR0": settingsIMU.numberOfRecordsSR2,
            "numberOfRecordsSR1": settingsIMU.numberOfRecordsSR3,
            "register": {
                "pageID": adcmxl3021_registers.fft_avg2.pageId,
                "address": adcmxl3021_registers.fft_avg2.address,
                "default_value": adcmxl3021_registers.fft_avg2.defaultValue
            },
            "reg_value": settingsIMU.fft_avg2_value
        }
    ],
    "x_anull": [
        {
            "register": {
                "pageID": adcmxl3021_registers.x_anull.pageId,
                "address": adcmxl3021_registers.x_anull.address,
                "default_value": adcmxl3021_registers.x_anull.defaultValue
            },
            "reg_value": settingsIMU.x_anull_value
        }
    ],
    "y_anull": [
        {
            "register": {
                "pageID": adcmxl3021_registers.y_anull.pageId,
                "address": adcmxl3021_registers.y_anull.address,
                "default_value": adcmxl3021_registers.y_anull.defaultValue
            },
            "reg_value": settingsIMU.y_anull_value
        }
    ],
    "z_anull": [
        {
            "register": {
                "pageID": adcmxl3021_registers.z_anull.pageId,
                "address": adcmxl3021_registers.z_anull.address,
                "default_value": adcmxl3021_registers.z_anull.defaultValue
            },
            "reg_value": settingsIMU.z_anull_value
        }
    ],
    "rec_ctrl": [
        {
            "input values": "all except window_setting and recording mode True 1
                  or False 0, window_setting 1 hanning, 0 rectangular, 2
                 flat_top, recording_mode 0 MFFT, 1 AFFT, 2 MTC, 3 RTS",
            "rts_timeout_enable": settingsIMU.rts_timeout_enable,
            "window_setting": settingsIMU.window_setting,
            "sr3_enable": settingsIMU.sr3_enable,
            "sr2_enable": settingsIMU.sr2_enable,
            "sr1_enable": settingsIMU.sr1_enable,
            "sr0_enable": settingsIMU.sr0_enable,
            "automatic_power_down": settingsIMU.automatic_power_down,
            "enable_compute_stats_mtc": settingsIMU.enable_compute_stats_mtc,
            "enable_vel_calc": settingsIMU.enable_vel_calc,
            "calc_root_sum_square": settingsIMU.calc_root_sum_square,
            "flash_memory_record_storage_method": settingsIMU.
                 flash_memory_record_storage_method,
            "recording_mode": settingsIMU.recording_mode,
            "register": {
                "pageID": adcmxl3021_registers.rec_ctrl.pageId,
                "address": adcmxl3021_registers.rec_ctrl.address,
                "default_value": adcmxl3021_registers.rec_ctrl.defaultValue
            },
            "reg_value": settingsIMU.rec_ctrl_value
        }
    ],
    "rec_prd": [
        {
            "input values": "scale 0 seconds, 1 minute, 2 hour, rate value range
                  0 - 255",
            "scale": settingsIMU.scale,
            "rate": settingsIMU.rate,
            "register": {
                "pageID": adcmxl3021_registers.rec_prd.pageId,
                "address": adcmxl3021_registers.rec_prd.address,
```

```
            "default_value": adcmxl3021_registers.rec_prd.defaultValue
        },
        "reg_value": settingsIMU.rec_prd_value
    }
],
"alm_F_low": [
    {
        "input values": "# value range 0 - 2047",
        "lower_frequency_bin": settingsIMU.lower_frequency_bin,
        "register": {
            "pageID": adcmxl3021_registers.alm_F_low.pageId,
            "address": adcmxl3021_registers.alm_F_low.address,
            "default_value": adcmxl3021_registers.alm_F_low.defaultValue
        },
        "reg_value": settingsIMU.alm_F_low_value
    }
],
"alm_F_high": [
    {
        "input values": "# value range 0 - 2047",
        "upper_frequency_bin": settingsIMU.upper_frequency_bin,
        "register": {
            "pageID": adcmxl3021_registers.alm_F_high.pageId,
            "address": adcmxl3021_registers.alm_F_high.address,
            "default_value": adcmxl3021_registers.alm_F_high.defaultValue
        },
        "reg_value": settingsIMU.alm_F_high_value
    }
],
"alm_x_mag1": [
    {
        "register": {
            "pageID": adcmxl3021_registers.alm_x_mag1.pageId,
            "address": adcmxl3021_registers.alm_x_mag1.address,
            "default_value": adcmxl3021_registers.alm_x_mag1.defaultValue
        },
        "reg_value": settingsIMU.alm_x_mag1_value
    }
],
"alm_y_mag1": [
    {
        "register": {
            "pageID": adcmxl3021_registers.alm_y_mag1.pageId,
            "address": adcmxl3021_registers.alm_y_mag1.address,
            "default_value": adcmxl3021_registers.alm_y_mag1.defaultValue
        },
        "reg_value": settingsIMU.alm_y_mag1_value
    }
],
"alm_z_mag1": [
    {
        "register": {
            "pageID": adcmxl3021_registers.alm_z_mag1.pageId,
            "address": adcmxl3021_registers.alm_z_mag1.address,
            "default_value": adcmxl3021_registers.alm_z_mag1.defaultValue
        },
        "reg_value": settingsIMU.alm_z_mag1_value
    }
],
"alm_x_mag2": [
    {
        "register": {
            "pageID": adcmxl3021_registers.alm_x_mag2.pageId,
            "address": adcmxl3021_registers.alm_x_mag2.address,
            "default_value": adcmxl3021_registers.alm_x_mag2.defaultValue
        },
        "reg_value": settingsIMU.alm_x_mag2_value
    }
],
"alm_y_mag2": [
    {
        "register": {
            "pageID": adcmxl3021_registers.alm_y_mag2.pageId,
            "address": adcmxl3021_registers.alm_y_mag2.address,
            "default_value": adcmxl3021_registers.alm_y_mag2.defaultValue
        },
        "reg_value": settingsIMU.alm_y_mag2_value
    }
],
"alm_z_mag2": [
    {
        "register": {
            "pageID": adcmxl3021_registers.alm_z_mag2.pageId,
            "address": adcmxl3021_registers.alm_z_mag2.address,
            "default_value": adcmxl3021_registers.alm_z_mag2.defaultValue
        },
        "reg_value": settingsIMU.alm_z_mag2_value
    }
],
"alm_pntr": [
    {
        "sample_rate_setting_for": settingsIMU.sample_rate_setting_for,
        "spectral_band_number": settingsIMU.spectral_band_number,
        "register": {
            "pageID": adcmxl3021_registers.alm_pntr.pageId,
            "address": adcmxl3021_registers.alm_pntr.address,
```

```
            "default_value": adcmxl3021_registers.alm_pntr.defaultValue
        },
        "reg_value": settingsIMU.alm_pntr_value
    }
],
"alm_S_mag": [
    {
        "register": {
            "pageID": adcmxl3021_registers.alm_S_mag.pageId,
            "address": adcmxl3021_registers.alm_S_mag.address,
            "default_value": adcmxl3021_registers.alm_S_mag.defaultValue
        },
        "reg_value": settingsIMU.alm_S_mag_value
    }
],
"alm_ctrl": [
    {
        "disable_automatic_clearing_spectral_alarm": settingsIMU.
             disable_automatic_clearing_spectral_alarm,
        "response_delay": settingsIMU.response_delay,
        "latch_diag_stat": settingsIMU.latch_diag_stat,
        "enable_alm1_alm2": settingsIMU.enable_alm1_alm2,
        "system_alm_prio": settingsIMU.system_alm_prio,
        "system_alm_selection": settingsIMU.system_alm_selection,
        "system_alarm_enable": settingsIMU.system_alarm_enable,
        "x_axis_alm": settingsIMU.x_axis_alm,
        "y_axis_alm": settingsIMU.y_axis_alm,
        "z_axis_alm": settingsIMU.z_axis_alm,
        "register": {
            "pageID": adcmxl3021_registers.alm_ctrl.pageId,
            "address": adcmxl3021_registers.alm_ctrl.address,
            "default_value": adcmxl3021_registers.alm_ctrl.defaultValue
        },
        "reg_value": settingsIMU.alm_ctrl_value
    }
],
"filt_ctrl": [
    {
        "input values": "all: 0 none, 1 bank_a, 2 bank_b, 3 bank_c, 4 bank_d
             , 5 bank_e, 6 bank_f",
        "x_axis_fir_filter_selection": settingsIMU.
             x_axis_fir_filter_selection,
        "y_axis_fir_filter_selection": settingsIMU.
             y_axis_fir_filter_selection,
        "z_axis_fir_filter_selection": settingsIMU.
             z_axis_fir_filter_selection,
        "register": {
            "pageID": adcmxl3021_registers.filt_ctrl.pageId,
            "address": adcmxl3021_registers.filt_ctrl.address,
            "default_value": adcmxl3021_registers.filt_ctrl.defaultValue
        },
        "reg_value": settingsIMU.filt_ctrl_value
    }
],
"avg_cnt": [
    {"input values": "all see table 87 datasheet adcmxl3021",
        "sr0_sample_rate_scale_factor": settingsIMU.
             sr0_sample_rate_scale_factor,
        "sr1_sample_rate_scale_factor": settingsIMU.
             sr1_sample_rate_scale_factor,
        "sr2_sample_rate_scale_factor": settingsIMU.
             sr2_sample_rate_scale_factor,
        "sr3_sample_rate_scale_factor": settingsIMU.
             sr3_sample_rate_scale_factor,
     "register": {
         "pageID": adcmxl3021_registers.avg_cnt.pageId,
         "address": adcmxl3021_registers.avg_cnt.address,
         "default_value": adcmxl3021_registers.avg_cnt.defaultValue
     },
     "reg_value": settingsIMU.avg_cnt_value
     }
],
"glob_cmd": [
    {
        "input values": "all True 1 or False 0",
        "clear_auto_null_correction": settingsIMU.clear_auto_null_correction
             ,
        "retrieve_spectral_alarm_band": settingsIMU.
             retrieve_spectral_alarm_band,
        "retrieve_record_data_from_flash": settingsIMU.
             retrieve_record_data_from_flash,
        "save_spectral_alarm_band_registers_to_flash": settingsIMU.
             save_spectral_alarm_band_registers_to_flash,
        "record_start_stop": settingsIMU.record_start_stop,
        "reset_buf_ptr": settingsIMU.reset_buf_ptr,
        "clear_spectral_alarm_band_registers_from_flash": settingsIMU.
             clear_spectral_alarm_band_registers_from_flash,
        "clear_all_records": settingsIMU.clear_all_records,
        "software_reset": settingsIMU.software_reset,
        "save_registers_to_flash_memory": settingsIMU.
             save_registers_to_flash_memory,
        "flash_test": settingsIMU.flash_test,
        "clear_diag_stat": settingsIMU.clear_diag_stat,
        "factory_reset": settingsIMU.factory_reset,
        "self_test": settingsIMU.self_test,
        "power_down": settingsIMU.power_down,
```

```
            "autonull": settingsIMU.autonull,
            "register": {
                "pageID": adcmxl3021_registers.glob_cmd.pageId,
                "address": adcmxl3021_registers.glob_cmd.address,
                "default_value": adcmxl3021_registers.glob_cmd.defaultValue
            },
            "reg_value": settingsIMU.glob_cmd_value
        }
    ],
    "user_scratch": [
        {
            "input values": "0x0000 to 0xFFFF",
            "user_id": settingsIMU.user_id,
            "register": {
                "pageID": adcmxl3021_registers.user_scratch.pageId,
                "address": adcmxl3021_registers.user_scratch.address,
                "default_value": adcmxl3021_registers.user_scratch.defaultValue
            },
            "reg_value": settingsIMU.user_id
        }
    ],
    "misc_ctrl": [
        {
            "input values": "all True 1 or False 0",
            "enable_sync_sensitivity": settingsIMU.enable_sync_sensitivity,
            "transfers_statistics_record_from_flash_memory_to_sram": settingsIMU
                .transfers_statistics_record_from_flash_memory_to_sram,
            "transfer_statistics_from_sram_to_flash_record": settingsIMU.
                transfer_statistics_from_sram_to_flash_record,
            "clear_time_domain_statistics": settingsIMU.
                clear_time_domain_statistics,
            "set_self_test": settingsIMU.set_self_test,
            "clear_self_test": settingsIMU.clear_self_test,
            "register": {
                "pageID": adcmxl3021_registers.misc_ctrl.pageId,
                "address": adcmxl3021_registers.misc_ctrl.address,
                "default_value": adcmxl3021_registers.misc_ctrl.defaultValue
            },
            "reg_value": settingsIMU.misc_ctrl_value
        }
    ]
}

with open('imuSettingsJSON.json', 'w', encoding='utf-8') as f:
    json.dump(imuSettingsJSON, f, ensure_ascii=False, indent=4)
```

A.9.6 receiveData.py

```
#!/usr/bin/env python
import can
import time
from definitions import defines
import csv
import os

# taken from https://stackoverflow.com/a/6727975/20937488 and modified


def twos_complement(value, bits):
    if value & (1 << (bits - 1)):
        value -= 1 << bits
    return value


def receiveFFTData(bus, id):
    """Receives all messages and prints them to the console until Ctrl+C is
        pressed."""
    msgBuffer = []
    try:
        while True:
            msg = bus.recv(None)
            # FFT
            if msg.data[0] == defines.START_FFT_DATA_TRANSMISSON:
                fftAvg = msg.data[1]
                f_s = msg.data[2]
                current_time = time.strftime("%Y%m%d_%H_%M_%S")
                name = "fft_ID_" + str(id) + "_time_" + current_time + "_data.
                    csv"
                nameMeta = "meta_fft" + current_time + "_data.csv"
                pathFFT = os.path.join(filePath)
                if not os.path.exists(pathFFT):
                    os.makedirs(pathFFT)
                path = os.path.join(pathFFT, name)
                pathMeta = os.path.join(pathFFT, nameMeta)
                while True:
                    msg = bus.recv(None)
                    if msg.data[0] == defines.STOP_FFT_DATA_TRANSMISSON:
                        with open(path, 'w', newline='') as file:
                            writer = csv.writer(file)
                            writer.writerow(["x_data", "y_data", "z_data"])
                            for msgData in msgBuffer:
                                writer.writerow([((msgData[1] << 8) | msgData
                                    [2]), ((msgData[3] << 8) | msgData[4]), ((
                                    msgData[5] << 8) | msgData[6])])
```

```
                    msgBuffer.clear()
                with open(pathMeta, 'w', newline='') as file:
                    writer = csv.writer(file)
                    writer.writerow(["fftAvg", "f_s"])
                    writer.writerow([fftAvg, f_s])
                return
            elif msg is not None and msg.data[0] == defines.
                 FFT_DATA_TRANSMISSON:
                msgBuffer.append(msg.data)
    # MTC
    if msg.data[0] == defines.START_MTC_DATA_TRANSMISSON:
        fftAvg = msg.data[3]
        aOrV = msg.data[2]
        f_s = msg.data[1]
        current_time = time.strftime("%Y%m%d_%H_%M_%S")
        name = "mtc_ID_" + str(id) + "_time_" + current_time + "_data.
             csv"
        nameMeta = "mtc_meta_" + current_time + "_data.csv"
        pathMTC = os.path.join(filePath)
        pathMeta = os.path.join(pathMTC, nameMeta)
        if not os.path.exists(pathMTC):
            os.makedirs(pathMTC)
        path = os.path.join(pathMTC, name)
        while True:
            msg = bus.recv(None)
            if msg.data[0] == defines.STOP_MTC_DATA_TRANSMISSON:

                with open(path, 'w', newline='') as file:
                    writer = csv.writer(file)
                    writer.writerow(["x_data", "y_data", "z_data"])
                    for msgData in msgBuffer:
                        x_data_raw = ((msgData[1] << 8) | msgData[2])
                        y_data_raw = ((msgData[3] << 8) | msgData[4])
                        z_data_raw = ((msgData[5] << 8) | msgData[6])
                        x_data_converted = twos_complement(x_data_raw,
                             16)
                        y_data_raw_converted = twos_complement(
                             y_data_raw, 16)
                        z_data_raw_converted = twos_complement(
                             z_data_raw, 16)
                        writer.writerow([x_data_converted,
                             y_data_raw_converted, z_data_raw_converted
                             ])
                    msgBuffer.clear()
                with open(pathMeta, 'w', newline='') as file:
                    writer = csv.writer(file)
                    writer.writerow(["fftAvg", "f_s", "acc or v"])
                    writer.writerow([fftAvg, f_s, aOrV])

                    return
                elif msg is not None and msg.data[0] == defines.
                     MTC_DATA_TRANSMISSON:
                    msgBuffer.append(msg.data)
    except KeyboardInterrupt:
        pass  # exit normally

if __name__ == "__main__":
    # can bus
    canID = 0x500
    id = "dritte_1_4"
    # create folder for .csv
    filePath = os.path.join("data", "20231804", id)
    if not os.path.exists(filePath):
        os.makedirs(filePath)
    bus = can.Bus(interface='ixxat', channel='0', bitrate=500000)
    while True:
        # wait for new FFT Data
        receiveFFTData(bus, id)
        print("wait")
```

A.10 Mikrocontroller Code

A.10.1 main.c

```
/**
 * author:
 * Torben Dierks
 * 2404082
 * Supervising Professor:
 * Prof. Dr. Rasmus Rettig
 * description:
 * main file bachelor thesis
 *
 */

// CONFIG1H
#pragma config OSC = HSPLL // Oscillator Selection bits (HS oscillator, PLL
     enabled (Clock Frequency = 4 x FOSC1))
#pragma config FCMEN = OFF // Fail-Safe Clock Monitor Enable bit (Fail-Safe
     Clock Monitor disabled)
#pragma config IESO = OFF  // Internal/External Oscillator Switchover bit (
     Oscillator Switchover mode disabled)
```

```
16
17 // CONFIG2L
18 #pragma config PWRT = OFF   // Power-up Timer Enable bit (PWRT disabled)
19 #pragma config BOREN = BOHW // Brown-out Reset Enable bits (Brown-out Reset
       enabled in hardware only (SBOREN is disabled))
20 #pragma config BORV = 3     // Brown-out Reset Voltage bits (VBOR set to 2.1V)
21
22 // CONFIG2H
23 #pragma config WDT = OFF     // Watchdog Timer Enable bit (WDT disabled (control
        is placed on the SWDTEN bit))
24 #pragma config WDTPS = 32768 // Watchdog Timer Postscale Select bits (1:32768)
25
26 // CONFIG3H
27 #pragma config PBADEN = ON   // PORTB A/D Enable bit (PORTB<4:0> pins are
       configured as analog input channels on Reset)
28 #pragma config LPT1OSC = OFF // Low-Power Timer 1 Oscillator Enable bit (Timer1
       configured for higher power operation)
29 #pragma config MCLRE = ON    // MCLR Pin Enable bit (MCLR pin enabled; RE3 input
        pin disabled)
30
31 // CONFIG4L
32 #pragma config STVREN = ON  // Stack Full/Underflow Reset Enable bit (Stack full
       /underflow will cause Reset)
33 #pragma config LVP = ON     // Single-Supply ICSP Enable bit (Single-Supply ICSP
        enabled)
34 #pragma config BBSIZ = 1024 // Boot Block Size Select bit (1K words (2K bytes)
       boot block)
35 #pragma config XINST = OFF  // Extended Instruction Set Enable bit (Instruction
       set extension and Indexed Addressing mode disabled (Legacy mode))
36
37 // CONFIG5L
38 #pragma config CP0 = OFF // Code Protection bit (Block 0 (000800-001FFFh) not
       code-protected)
39 #pragma config CP1 = OFF // Code Protection bit (Block 1 (002000-003FFFh) not
       code-protected)
40
41 // CONFIG5H
42 #pragma config CPB = OFF // Boot Block Code Protection bit (Boot block
       (000000-0007FFh) not code-protected)
43 #pragma config CPD = OFF // Data EEPROM Code Protection bit (Data EEPROM not
       code-protected)
44
45 // CONFIG6L
46 #pragma config WRT0 = OFF // Write Protection bit (Block 0 (000800-001FFFh) not
       write-protected)
47 #pragma config WRT1 = OFF // Write Protection bit (Block 1 (002000-003FFFh) not
       write-protected)
48
49 // CONFIG6H
50 #pragma config WRTC = OFF // Configuration Register Write Protection bit (
       Configuration registers (300000-3000FFh) not write-protected)
51 #pragma config WRTB = OFF // Boot Block Write Protection bit (Boot block
       (000000-0007FFh) not write-protected)
52 #pragma config WRTD = OFF // Data EEPROM Write Protection bit (Data EEPROM not
       write-protected)
53
54 // CONFIG7L
55 #pragma config EBTR0 = OFF // Table Read Protection bit (Block 0 (000800-001FFFh
       ) not protected from table reads executed in other blocks)
56 #pragma config EBTR1 = OFF // Table Read Protection bit (Block 1 (002000-003FFFh
       ) not protected from table reads executed in other blocks)
57
58 // CONFIG7H
59 #pragma config EBTRB = OFF // Boot Block Table Read Protection bit (Boot block
       (000000-0007FFh) not protected from table reads executed in other blocks)
60
61 // #pragma config statements should precede project file includes.
62 // Use project enums instead of #define for ON and OFF.
63
64 #include <xc.h>
65
66 #include "setup.h"
67 #include "definitions.h"
68 #include "ecan.h"
69 #include "interfacingSPI.h"
70 #include "lib/eeprom.h"
71
72 // function prototypes
73 void LED1on();
74 void LED1off();
75 void LED2on();
76 void LED2off();
77 void resetsensor();
78
79 void main(void)
80 {
81     portSetup();
82     interruptSetup();
83     ECAN_Initialize();
84     spiInitialize();
85
86     uCAN_MSG *canPtrSent, *canPtrReceive, canMsgSent, canMsgReceive;
87     canPtrSent = &canMsgSent;
88     canPtrReceive = &canMsgReceive;
89
90     int msgType = 0;
```

```
91  // variable declaration
92  uint8_t pageID = 0;
93  uint8_t address = 0;
94  bool fftConfigured = false;
95  uint8_t sampleRateOption = 254;
96  uint8_t windowSetting = 254;
97  uint8_t fftAverages = 254;
98  uint8_t avgCNTSetting = 254;
99  uint16_t recCtrlContent = 254;
100 uint16_t avgCntContent = 254;
101 uint16_t fftAvg1Content = 254;
102 uint16_t fftAvg2Content = 254;
103 uint16_t recPrdContent = 254;
104 uint16_t zBuffer = 0;
105 uint16_t xBuffer = 0;
106 uint16_t yBuffer = 0;
107 uint16_t zBufferMTC = 0;
108 uint16_t xBufferMTC = 0;
109 uint16_t yBufferMTC = 0;
110 int iData;
111 uint16_t bufPtr;
112 uint8_t txData = 0;
113 // set state for sensor SPI CS
114 CS = 1;
115 uint8_t baudrate = 0;
116 uint8_t settingHigh = 0;
117 uint8_t settingLow = 0;
118 uint16_t registerContent = 0;
119 uint8_t accelerationORvelocity = 0; // 0 a 1 v
120 uint8_t recordingMode = RECORDING_MODE_NOT_DEFINED;
121
122 while (1)
123 {
124     LED1on();
125     if (PORTBbits.RB0 == 0)
126     {
127         LED2off();
128     }
129     else
130     {
131         LED2on();
132     }
133     if (PIR3bits.RXB0IF)
134     {
135
136         CAN_receive(canPtrReceive);
137         // determine action from msg type
138         int msgType = canPtrReceive->frame.data0;
139         // clear interrupt
140         PIR3bits.RXB0IF = 0;
141         if (msgType == CAN_MSG_TYPE_SETTING)
142         {
143             // get data to be written from received can msg and write it to
                    register on sensor
144             pageID = canPtrReceive->frame.data1;
145             address = canPtrReceive->frame.data2;
146             settingHigh = canPtrReceive->frame.data3;
147             settingLow = canPtrReceive->frame.data4;
148             writeRegister(pageID, address, settingHigh, settingLow);
149         }
150         if (msgType == CAN_MSG_TYPE_REGISTER_CONTENT)
151         {
152             // get register content and write content to can bus
153             pageID = canPtrReceive->frame.data1;
154             address = canPtrReceive->frame.data2;
155             registerContent = readRegister(pageID, address);
156             // build can msg to be sent
157             canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
158             canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
159             canPtrSent->frame.dlc = CAN_DATA_LENGTH_3_BYTES; // data length
                    register 0000 0 Byte 1000 8 byte
160             canPtrSent->frame.data0 = CAN_MSG_TYPE_REGISTER_CONTENT;
161             canPtrSent->frame.data1 = registerContent >> 8;
162             canPtrSent->frame.data2 = registerContent & 0xFF;
163             // send data via CAN
164             // stay in while loop as long as can msg can not be written to
                    transfer buffer
165             while (!CAN_transmit(canPtrSent))
166                 ;
167         }
168         if (msgType == CAN_MSG_TYPE_START_CAPTURE)
169         {
170             writeRegister(PAGE_ID, ADDRESS_GLOB_CMD, 0x08, 0x00);
171         }
172
173         if (msgType == CAN_MSG_TYPE_GET_SETTINGS)
174         {
175             // retrieve current sensor settings and send them via can
176             fftConfigured = true;
177             // rec_ctrl
178             recCtrlContent = readRegister(PAGE_ID, ADDRESS_REC_CTRL);
179             sampleRateOption = (recCtrlContent >> 8) & 0xF;
180             windowSetting = (recCtrlContent >> 12) & 0x3;
181             recordingMode = recCtrlContent & 0x3;
182             accelerationORvelocity = (recCtrlContent >> 5) & 0x1;
183             // avg_cnt
```

```
184             avgCntContent = readRegister(PAGE_ID, ADDRESS_AVG_CNT);
185             // rec_prd
186             recPrdContent = readRegister(PAGE_ID, ADDRESS_REC_PRD);
187             // fft_avg1
188             fftAvg1Content = readRegister(PAGE_ID, ADDRESS_FFT_AVG1);
189             // fft_avg2
190             fftAvg2Content = readRegister(PAGE_ID, ADDRESS_FFT_AVG2);
191             if (sampleRateOption == SR0_ENABLED)
192             {
193                 fftAverages = fftAvg1Content & 0xFF;
194             }
195             else if (sampleRateOption == SR1_ENABLED)
196             {
197                 fftAverages = fftAvg1Content >> 8;
198             }
199             else if (sampleRateOption == SR2_ENABLED)
200             {
201                 fftAverages = fftAvg2Content & 0xFF;
202             }
203             else if (sampleRateOption == SR3_ENABLED)
204             {
205                 fftAverages = fftAvg2Content >> 8;
206             }
207             else
208             {
209                 fftAverages = 0xFF;
210             }
211             if (sampleRateOption == SR0_ENABLED)
212             {
213                 avgCNTSetting = avgCntContent & 0xF;
214             }
215             else if (sampleRateOption == SR1_ENABLED)
216             {
217                 avgCNTSetting = (avgCntContent >> 4) & 0xF;
218             }
219             else if (sampleRateOption == SR2_ENABLED)
220             {
221                 avgCNTSetting = (avgCntContent >> 8) & 0xF;
222             }
223             else if (sampleRateOption == SR3_ENABLED)
224             {
225                 avgCNTSetting = (avgCntContent >> 12) & 0xF;
226             }
227             else
228             {
229                 avgCNTSetting = 0xFF;
230             }
231             // get register content and write content to can bus
232             canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
233             canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
234             canPtrSent->frame.dlc = CAN_DATA_LENGTH_8_BYTES; // data length
                    register 0000 0 Byte 1000 8 byte
235             canPtrSent->frame.data0 = CAN_MSG_TYPE_GET_SETTINGS;
236             canPtrSent->frame.data1 = sampleRateOption;
237             canPtrSent->frame.data2 = windowSetting;
238             canPtrSent->frame.data3 = fftAverages;
239             canPtrSent->frame.data4 = avgCNTSetting;
240             canPtrSent->frame.data5 = (recPrdContent >> 8) & 0x3; // seconds
                    , minutes, hours
241             canPtrSent->frame.data6 = recPrdContent & 0xFF;       // scale
                    factor
242             canPtrSent->frame.data7 = recordingMode;
243             // send data via CAN
244             // stay in while loop as long as can msg can not be written to
                    transfer buffer
245             while (!CAN_transmit(canPtrSent))
246                 ;
247             // get register content and write content to can bus
248             canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
249             canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
250             canPtrSent->frame.dlc = CAN_DATA_LENGTH_2_BYTES; // data length
                    register 0000 0 Byte 1000 8 byte
251             canPtrSent->frame.data0 = CAN_MSG_TYPE_GET_SETTINGS;
252             canPtrSent->frame.data1 = accelerationORvelocity;
253             // send data via CAN
254             // stay in while loop as long as can msg can not be written to
                    transfer buffer
255             while (!CAN_transmit(canPtrSent))
256                 ;
257             // writeEEPROM(MEM_LOC_RECORDING_MODE, recordingMode);
258         }
259         if (msgType == CAN_MSG_TYPE_CONFIGURATION_sensor_START)
260         {
261             // stop capture
262             fftConfigured = false;
263             while (PORTBbits.RB0 == 0)
264             {
265                 writeRegister(0x00, ADDRESS_GLOB_CMD, ESCPAE_CODE_HIGH,
                        ESCAPE_CODE_LOW);
266             }
267             // get register content and write content to can bus
268             canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
269             canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
270             canPtrSent->frame.dlc = CAN_DATA_LENGTH_2_BYTES; // data length
                    register 0000 0 Byte 1000 8 byte
271             canPtrSent->frame.data0 = CAN_MSG_TYPE_ESCAPED;
```

```
272             // send data via CAN
273             // stay in while loop as long as can msg can not be written to
                    transfer buffer
274             while (!CAN_transmit(canPtrSent))
275                 ;
276             // writeRegister(PAGE_ID, ADDRESS_GLOB_CMD, ESCPAE_CODE_HIGH,
                    ESCAPE_CODE_LOW);
277             recordingMode = RECORDING_MODE_CONFIGURATION;
278         }
279         if (msgType == CAN_MSG_TYPE_CONFIGURATION_sensor_STOP)
280         {
281             recCtrlContent = readRegister(PAGE_ID, ADDRESS_REC_CTRL);
282             recordingMode = recCtrlContent & 0x3;
283             writeRegister(0x00, ADDRESS_GLOB_CMD, 0x00, 0x40);
284             while (PORTBbits.RB0 == 0)
285             {
286             }
287             fftConfigured = true;
288         }
289         if (msgType == CAN_MSG_TYPE_START_CAPTURE)
290         {
291             writeRegister(PAGE_ID, ADDRESS_GLOB_CMD, 0x08, 0x00);
292             // get register content and write content to can bus
293             canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
294             canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
295             canPtrSent->frame.dlc = CAN_DATA_LENGTH_1_BYTES; // data length
                    register 0000 0 Byte 1000 8 byte
296             canPtrSent->frame.data0 = CAN_MSG_TYPE_START_CAPTURE;
297             // send data via CAN
298             // stay in while loop as long as can msg can not be written to
                    transfer buffer
299             while (!CAN_transmit(canPtrSent))
300                 ;
301         }
302
303         if (msgType == CAN_MSG_TYPE_AUTONULL)
304         {
305             writeRegister(0x00, ADDRESS_GLOB_CMD, 0x00, 0x01);
306             // get register content and write content to can bus
307             canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
308             canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
309             canPtrSent->frame.dlc = CAN_DATA_LENGTH_1_BYTES; // data length
                    register 0000 0 Byte 1000 8 byte
310             canPtrSent->frame.data0 = CAN_MSG_TYPE_AUTONULL;
311             // send data via CAN
312             // stay in while loop as long as can msg can not be written to
                    transfer buffer
313             while (!CAN_transmit(canPtrSent))
314                 ;
315         }
316         if (msgType == CAN_MSG_TYPE_SAVE_REGISTER)
317         {
318             writeRegister(0x00, ADDRESS_GLOB_CMD, 0x00, 0x40);
319             // get register content and write content to can bus
320             canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
321             canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
322             canPtrSent->frame.dlc = CAN_DATA_LENGTH_1_BYTES; // data length
                    register 0000 0 Byte 1000 8 byte
323             canPtrSent->frame.data0 = CAN_MSG_TYPE_SAVE_REGISTER;
324             // send data via CAN
325             // stay in while loop as long as can msg can not be written to
                    transfer buffer
326             while (!CAN_transmit(canPtrSent))
327                 ;
328         }
329         if (msgType == CAN_MSG_TYPE_RESET_SENSOR)
330         {
331             writeRegister(0x00, ADDRESS_GLOB_CMD, 0x00, 0x08);
332             // get register content and write content to can bus
333             canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
334             canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
335             canPtrSent->frame.dlc = CAN_DATA_LENGTH_1_BYTES; // data length
                    register 0000 0 Byte 1000 8 byte
336             canPtrSent->frame.data0 = CAN_MSG_TYPE_RESET_SENSOR;
337             // send data via CAN
338             // stay in while loop as long as can msg can not be written to
                    transfer buffer
339             while (!CAN_transmit(canPtrSent))
340                 ;
341         }
342         if (msgType == CAN_MSG_TYPE_CHANGE_CAN_SETTINGS)
343         {
344             // change settings for baudrate and identifier of CAN
345             // disable interrupts
346             /**
347              * change settings msg
348              * data0 CAN_MSG_TYPE_CHANGE_CAN_SETTINGS
349              * data1 baudrate settings preset
350              * data2 identifier preset
351              *
352              * stored settings:
353              * 1: 1Mbps 0xA1 atm not working
354              * 2: 800kbps 0xA2 atm not working
355              * 3: 500kbps 0xA3
356              * 4: 250kbps 0xA4
357              *
```

```
358                 **/
359
360             baudrate = canPtrReceive->frame.data1;
361
362             switch (baudrate)
363             {
364             case CAN_BAUD_1_MBPS:
365                 changeBaudrateCAN(CAN_BAUD_1_MBPS_BRGCON1,
                        CAN_BAUD_1_MBPS_BRGCON2, CAN_BAUD_1_MBPS_BRGCON3);
366                 break;
367             case CAN_BAUD_800_kbps:
368                 changeBaudrateCAN(CAN_BAUD_800_KBPS_BRGCON1,
                        CAN_BAUD_800_KBPS_BRGCON2, CAN_BAUD_800_KBPS_BRGCON3);
369                 break;
370             case CAN_BAUD_500_kbps:
371                 changeBaudrateCAN(CAN_BAUD_500_KBPS_BRGCON1,
                        CAN_BAUD_500_KBPS_BRGCON2, CAN_BAUD_500_KBPS_BRGCON3);
372                 break;
373             case CAN_BAUD_250_kbps:
374                 changeBaudrateCAN(CAN_BAUD_250_KBPS_BRGCON1,
                        CAN_BAUD_250_KBPS_BRGCON2, CAN_BAUD_250_KBPS_BRGCON3);
375                 break;
376
377             default:
378
379                 break;
380             }
381         }
382     }
383     else if ((INTCONbits.INT0F) && fftConfigured)
384     {
385
386         // clear interrupt flag
387         INTCONbits.INT0F = 0;
388
389         if ((recordingMode == RECORDING_MODE_AFFT) || (recordingMode ==
                RECORDING_MODE_MFFT))
390         {
391             uint8_t high = 0;
392             uint8_t low = 0;
393
394             /**
395              * The sensor signals that FFT data is available via a rising
                    edge on the busy bin.
396              * A low state on the busy pin signals that the sensor currently
                    gathering data.
397              *
398              * A sensor data transmission consits of 2050 CAN messages.
399              *
400              * 1. Start message signaling the receiving end that fft data is
                    about to be transmitted.
401              *  data0 identifier 0x92
402              * 2. 2048 messages containg the fft data bin 0 to 2047
403              *  data0 identifier 0x96
404              *  data1 fft data x axis bin n high byte
405              *  data2 fft data x axis bin n low byte
406              *  data3 fft data y axis bin n high byte
407              *  data4 fft data y axis bin n low byte
408              *  data5 fft data z axis bin n high byte
409              *  data6 fft data z axis bin n low byte
410              * 3. End message signaling the receiving end that fft data is
                    about to be transmitted.
411              *  data0 identifier 0x94
412              **/
413
414             // start msg
415             canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
416             canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
417             canPtrSent->frame.dlc = CAN_DATA_LENGTH_3_BYTES; // data length
                    register 0000 0 Byte 1000 8 byte
418             canPtrSent->frame.data0 = START_FFT_DATA_TRANSMISSON;
419             //                canPtrSent->frame.data1 = fftAverages; //
                    needed for interpretation of measurement data
420             //                canPtrSent->frame.data2 = avgCNTSetting;
421             while (!CAN_transmit(canPtrSent))
422                 ;
423
424             // write 0x0000 to buf_ptr register in order to load the 0th
                    sample
425             // for each axis into the accoding buffer registers
426             writeRegister(PAGE_ID, ADDRESS_BUF_PTR, 0x00, 0x00);
427             // iterate over all 2048 bins and retrieve fft data for x,y,z-
                    axis
428             for (iData = 0; iData < (SAMPLES_COUNT_FFT); iData++)
429             {
430                 // determine bin
431                 high = iData >> 8;
432                 low = iData & 0xFF;
433                 // write iData to buf_ptr register in order to load the
                        iDatath sample
434                 // a read from one of the buffer registers causes all
                        registers to increment one bin
435                 // in order to retrieve the same bin for x,y,z-data at the
                        same time a write to buf_ptr register is necessary
436                 writeRegister(0x00, 0x0A, high, low);
437                 xBuffer = readRegister(0x00, ADDRESS_X_Buffer);
```

```
            writeRegister(0x00, 0x0A, high, low);
            yBuffer = readRegister(0x00, ADDRESS_Y_Buffer);
            writeRegister(0x00, 0x0A, high, low);
            zBuffer = readRegister(0x00, ADDRESS_Z_Buffer);

            // build can msg to be send
            canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
            canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
            canPtrSent->frame.dlc = CAN_DATA_LENGTH_7_BYTES;
            // identifier to signal receiving end that the current can msg is a data msg
            canPtrSent->frame.data0 = FFT_DATA_TRANSMISSON;
            canPtrSent->frame.data1 = xBuffer >> 8;   // high byte
            canPtrSent->frame.data2 = xBuffer & 0xFF; // low byte
            canPtrSent->frame.data3 = yBuffer >> 8;   // high byte
            canPtrSent->frame.data4 = yBuffer & 0xFF; // low byte
            canPtrSent->frame.data5 = zBuffer >> 8;   // high byte
            canPtrSent->frame.data6 = zBuffer & 0xFF; // low byte
            while (!CAN_transmit(canPtrSent))
                ;
        }
        __delay_ms(10); // delay because otherwise end msg gets sent before last data msg
        // send end msg to signal end of fft data transmission
        bufPtr = readRegister(0x00, ADDRESS_BUF_PTR);
        canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
        canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
        canPtrSent->frame.dlc = CAN_DATA_LENGTH_3_BYTES; // data length register 0000 0 Byte 1000 8 byte
        canPtrSent->frame.data0 = STOP_FFT_DATA_TRANSMISSON;
        canPtrSent->frame.data1 = bufPtr >> 8; // buf ptr value at end of transmisson for debugging
        canPtrSent->frame.data2 = bufPtr & 0xFF;
        while (!CAN_transmit(canPtrSent))
            ;
    }
    else if (recordingMode == RECORDING_MODE_MTC)
    {
        uint8_t high = 0;
        uint8_t low = 0;

        /**
         * The sensor signals that MTC data is available via a rising edge on the busy bin.
         * A low state on the busy pin signals that the sensor currently gathering data.
         *
         * A sensor data transmission consits of 4098 CAN messages.
         *
         * 1. Start message signaling the receiving end that fft data is about to be transmitted.
         *  data0 identifier 0xE2
         * 2. 2048 messages containg the fft data bin 0 to 2047
         *  data0 identifier 0xE4
         *  data1 fft data x axis bin n high byte
         *  data2 fft data x axis bin n low byte
         *  data3 fft data y axis bin n high byte
         *  data4 fft data y axis bin n low byte
         *  data5 fft data z axis bin n high byte
         *  data6 fft data z axis bin n low byte
         * 3. End message signaling the receiving end that fft data is about to be transmitted.
         *  data0 identifier 0xE6
         **/

        // start msg
        canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
        canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
        canPtrSent->frame.dlc = CAN_DATA_LENGTH_4_BYTES; // data length register 0000 0 Byte 1000 8 byte
        canPtrSent->frame.data0 = START_MTC_DATA_TRANSMISSON;
        canPtrSent->frame.data1 = avgCNTSetting; // needed for interpretation of measurement data
        canPtrSent->frame.data2 = accelerationORvelocity;
        canPtrSent->frame.data3 = fftAverages;
        while (!CAN_transmit(canPtrSent))
            ;

        // write 0x0000 to buf_ptr register in order to load the 0th sample
        // for each axis into the accoding buffer registers
        writeRegister(PAGE_ID, ADDRESS_BUF_PTR, 0x00, 0x00);
        // iterate over all 2048 bins and retrieve fft data for x,y,z-axis
        for (iData = 0; iData < (SAMPLES_COUNT_MTC); iData++)
        {
            // determine bin
            high = iData >> 8;
            low = iData & 0xFF;
            // write iData to buf_ptr register in order to load the iDatath sample
            // a read from one of the buffer registers causes all registers to increment one bin
            // in order to retrieve the same bin for x,y,z-data at the same time a write to buf_ptr register is necessary
            writeRegister(0x00, 0x0A, high, low);
```

```
519                 xBufferMTC = readRegister(0x00, ADDRESS_X_Buffer);
520                 writeRegister(0x00, 0x0A, high, low);
521                 yBufferMTC = readRegister(0x00, ADDRESS_Y_Buffer);
522                 writeRegister(0x00, 0x0A, high, low);
523                 zBufferMTC = readRegister(0x00, ADDRESS_Z_Buffer);
524
525                 // build can msg to be send
526                 canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
527                 canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
528                 canPtrSent->frame.dlc = CAN_DATA_LENGTH_7_BYTES;
529                 // identifier to signal receiving end that the current can
                        msg is a data msg
530                 canPtrSent->frame.data0 = MTC_DATA_TRANSMISSON;
531                 canPtrSent->frame.data1 = xBufferMTC >> 8;   // high byte
532                 canPtrSent->frame.data2 = xBufferMTC & 0xFF; // low byte
533                 canPtrSent->frame.data3 = yBufferMTC >> 8;   // high byte
534                 canPtrSent->frame.data4 = yBufferMTC & 0xFF; // low byte
535                 canPtrSent->frame.data5 = zBufferMTC >> 8;   // high byte
536                 canPtrSent->frame.data6 = zBufferMTC & 0xFF; // low byte
537                 while (!CAN_transmit(canPtrSent))
538                     ;
539             }
540             __delay_ms(10); // delay because otherwise end msg gets sent
                    before last data msg
541             // send end msg to signal end of fft data transmission
542             bufPtr = readRegister(0x00, ADDRESS_BUF_PTR);
543             canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
544             canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
545             canPtrSent->frame.dlc = CAN_DATA_LENGTH_3_BYTES; // data length
                    register 0000 0 Byte 1000 8 byte
546             canPtrSent->frame.data0 = STOP_MTC_DATA_TRANSMISSON;
547             canPtrSent->frame.data1 = bufPtr >> 8; // buf ptr value at end
                    of transmisson for debugging
548             canPtrSent->frame.data2 = bufPtr & 0xFF;
549             while (!CAN_transmit(canPtrSent))
550                 ;
551         }
552         else if (recordingMode == RECORDING_MODE_CONFIGURATION)
553         {
554         }
555         else
556         {
557             canPtrSent->frame.id = CAN_IDENTIFIER_PYTHON;
558             canPtrSent->frame.idType = dSTANDARD_CAN_MSG_ID_2_0B;
559             canPtrSent->frame.dlc = CAN_DATA_LENGTH_8_BYTES; // data length
                    register 0000 0 Byte 1000 8 byte
560             canPtrSent->frame.data0 = DATA_TRANSMISSON_ERROR;
561             canPtrSent->frame.data1 = 0xFF;
562             canPtrSent->frame.data2 = 0xFF;
563             canPtrSent->frame.data3 = 0xFF;
564             canPtrSent->frame.data4 = 0xFF;
565             canPtrSent->frame.data5 = 0xFF;
566             canPtrSent->frame.data6 = 0xFF;
567             canPtrSent->frame.data7 = 0xFF;
568             while (!CAN_transmit(canPtrSent))
569                 ;
570         }
571     }
572 }
573
574
575
576 void LED1on()
577 {
578     LED1_P = 1;
579 }
580
581 void LED1off()
582 {
583     LED1_P = 0;
584 }
585
586 void LED2on()
587 {
588     LED2_P = 1;
589 }
590
591 void LED2off()
592 {
593     LED2_P = 0;
594 }
595
596
597 void resetsensor()
598 {
599     __delay_ms(20);
600     RST = 0;
601     __delay_ms(140);
602     RST = 1;
603     __delay_ms(230);
604 }
```

A.10.2 ecan.c

```
1 /**
```

```
 * mcc generated file
 * modified by:
 * Torben Dierks
 * 2404082
 * Supervising Professor:
 * Prof. Dr. Rasmus Rettig
 * description:
 * mcc generated source file for can communication
 *
 */

#include <xc.h>
#include "ecan.h"

static uint32_t convertReg2ExtendedCANid(uint8_t tempRXBn_EIDH, uint8_t
    tempRXBn_EIDL, uint8_t tempRXBn_SIDH, uint8_t tempRXBn_SIDL);
static uint32_t convertReg2StandardCANid(uint8_t tempRXBn_SIDH, uint8_t
    tempRXBn_SIDL);
static void convertCANid2Reg(uint32_t tempPassedInID, uint8_t canIdType, uint8_t
     *passedInEIDH, uint8_t *passedInEIDL, uint8_t *passedInSIDH, uint8_t *
    passedInSIDL);

void ECAN_Initialize(void)
{
    ECANCON = 0x00;

    /**
    Initialize CAN I/O
    */
    CIOCON = 0x00;

    /**
    Mask and Filter definitions
    ....................................................
    CAN ID              ID Type         Mask                        Filter
                Buffer
    ....................................................
    0x500               SID             Acceptance Mask 0           Filter 0
                RXB0
    ....................................................
    */

    /**
        Initialize Receive Masks
    */
    RXM0EIDH = 0xFF;
    RXM0EIDL = 0xFF;
    RXM0SIDH = 0xFF;
    RXM0SIDL = 0xE3;
    RXM1EIDH = 0xFF;
    RXM1EIDL = 0xFF;
    RXM1SIDH = 0xFF;
    RXM1SIDL = 0xE3;

    /**
    Initialize Receive Filters
    */
    RXF0EIDH = 0x00;
    RXF0EIDL = 0x00;
    RXF0SIDH = 0xA0;
    RXF0SIDL = 0x00;
    RXF1EIDH = 0x00;
    RXF1EIDL = 0x00;
    RXF1SIDH = 0x00;
    RXF1SIDL = 0x00;
    RXF2EIDH = 0x00;
    RXF2EIDL = 0x00;
    RXF2SIDH = 0x00;
    RXF2SIDL = 0x00;
    RXF3EIDH = 0x00;
    RXF3EIDL = 0x00;
    RXF3SIDH = 0x00;
    RXF3SIDL = 0x00;
    RXF4EIDH = 0x00;
    RXF4EIDL = 0x00;
    RXF4SIDH = 0x00;
    RXF4SIDL = 0x00;
    RXF5EIDH = 0x00;
    RXF5EIDL = 0x00;
    RXF5SIDH = 0x00;
    RXF5SIDL = 0x00;

    /**
    Initialize CAN Timings
    */

    /**
        Baud rate: 500kbps
        System frequency: 32000000
        ECAN clock frequency: 32000000
        Time quanta: 8
        Sample point: 1-1-4-2
        Sample point: 75%
    */

    BRGCON1 = 0x03;
```

```
92      BRGCON2 = 0x98;
93      BRGCON3 = 0x01;
94
95      // ECAN_SetWakeUpInterruptHandler(WakeUpDefaultInterruptHandler);
96      PIR3bits.WAKIF = 0;
97      PIE3bits.WAKIE = 1;
98
99      CANCON = 0x00;
100     while (0x00 != (CANSTAT & 0xE0))
101         ; // wait until ECAN is in Normal mode
102 }
103
104 void CAN_sleep(void)
105 {
106     CANCON = 0x20; // request disable mode
107     while ((CANSTAT & 0xE0) != 0x20)
108         ; // wait until ECAN is in disable mode
109     // Wake up from sleep should set the CAN module straight into Normal mode
110 }
111
112 uint8_t CAN_transmit(uCAN_MSG *tempCanMsg)
113 {
114     uint8_t tempEIDH = 0;
115     uint8_t tempEIDL = 0;
116     uint8_t tempSIDH = 0;
117     uint8_t tempSIDL = 0;
118
119     uint8_t returnValue = 0;
120
121     if (TXB0CONbits.TXREQ != 1)
122     {
123
124         convertCANid2Reg(tempCanMsg->frame.id, tempCanMsg->frame.idType, &
                tempEIDH, &tempEIDL, &tempSIDH, &tempSIDL);
125
126         TXB0EIDH = tempEIDH;
127         TXB0EIDL = tempEIDL;
128         TXB0SIDH = tempSIDH;
129         TXB0SIDL = tempSIDL;
130         TXB0DLC = tempCanMsg->frame.dlc;
131         TXB0D0 = tempCanMsg->frame.data0;
132         TXB0D1 = tempCanMsg->frame.data1;
133         TXB0D2 = tempCanMsg->frame.data2;
134         TXB0D3 = tempCanMsg->frame.data3;
135         TXB0D4 = tempCanMsg->frame.data4;
136         TXB0D5 = tempCanMsg->frame.data5;
137         TXB0D6 = tempCanMsg->frame.data6;
138         TXB0D7 = tempCanMsg->frame.data7;
139
140         TXB0CONbits.TXREQ = 1; // Set the buffer to transmit
141         returnValue = 1;
142     }
143     else if (TXB1CONbits.TXREQ != 1)
144     {
145
146         convertCANid2Reg(tempCanMsg->frame.id, tempCanMsg->frame.idType, &
                tempEIDH, &tempEIDL, &tempSIDH, &tempSIDL);
147
148         TXB1EIDH = tempEIDH;
149         TXB1EIDL = tempEIDL;
150         TXB1SIDH = tempSIDH;
151         TXB1SIDL = tempSIDL;
152         TXB1DLC = tempCanMsg->frame.dlc;
153         TXB1D0 = tempCanMsg->frame.data0;
154         TXB1D1 = tempCanMsg->frame.data1;
155         TXB1D2 = tempCanMsg->frame.data2;
156         TXB1D3 = tempCanMsg->frame.data3;
157         TXB1D4 = tempCanMsg->frame.data4;
158         TXB1D5 = tempCanMsg->frame.data5;
159         TXB1D6 = tempCanMsg->frame.data6;
160         TXB1D7 = tempCanMsg->frame.data7;
161
162         TXB1CONbits.TXREQ = 1; // Set the buffer to transmit
163         returnValue = 1;
164     }
165     else if (TXB2CONbits.TXREQ != 1)
166     {
167
168         convertCANid2Reg(tempCanMsg->frame.id, tempCanMsg->frame.idType, &
                tempEIDH, &tempEIDL, &tempSIDH, &tempSIDL);
169
170         TXB2EIDH = tempEIDH;
171         TXB2EIDL = tempEIDL;
172         TXB2SIDH = tempSIDH;
173         TXB2SIDL = tempSIDL;
174         TXB2DLC = tempCanMsg->frame.dlc;
175         TXB2D0 = tempCanMsg->frame.data0;
176         TXB2D1 = tempCanMsg->frame.data1;
177         TXB2D2 = tempCanMsg->frame.data2;
178         TXB2D3 = tempCanMsg->frame.data3;
179         TXB2D4 = tempCanMsg->frame.data4;
180         TXB2D5 = tempCanMsg->frame.data5;
181         TXB2D6 = tempCanMsg->frame.data6;
182         TXB2D7 = tempCanMsg->frame.data7;
183
184         TXB2CONbits.TXREQ = 1; // Set the buffer to transmit
```

```
185         returnValue = 1;
186     }
187
188     return (returnValue);
189 }
190
191 uint8_t CAN_receive(uCAN_MSG *tempCanMsg)
192 {
193     uint8_t returnValue = 0;
194
195     // check which buffer the CAN message is in
196     if (RXB0CONbits.RXFUL != 0) // CheckRXB0
197     {
198         if ((RXB0SIDL & 0x08) == 0x08) // If Extended Message
199         {
200             // message is extended
201             tempCanMsg->frame.idType = (uint8_t)dEXTENDED_CAN_MSG_ID_2_0B;
202             tempCanMsg->frame.id = convertReg2ExtendedCANid(RXB0EIDH, RXB0EIDL,
                    RXB0SIDH, RXB0SIDL);
203         }
204         else
205         {
206             // message is standard
207             tempCanMsg->frame.idType = (uint8_t)dSTANDARD_CAN_MSG_ID_2_0B;
208             tempCanMsg->frame.id = convertReg2StandardCANid(RXB0SIDH, RXB0SIDL);
209         }
210
211         tempCanMsg->frame.dlc = RXB0DLC;
212         tempCanMsg->frame.data0 = RXB0D0;
213         tempCanMsg->frame.data1 = RXB0D1;
214         tempCanMsg->frame.data2 = RXB0D2;
215         tempCanMsg->frame.data3 = RXB0D3;
216         tempCanMsg->frame.data4 = RXB0D4;
217         tempCanMsg->frame.data5 = RXB0D5;
218         tempCanMsg->frame.data6 = RXB0D6;
219         tempCanMsg->frame.data7 = RXB0D7;
220         RXB0CONbits.RXFUL = 0;
221         returnValue = 1;
222     }
223     else if (RXB1CONbits.RXFUL != 0) // CheckRXB1
224     {
225         if ((RXB1SIDL & 0x08) == 0x08) // If Extended Message
226         {
227             // message is extended
228             tempCanMsg->frame.idType = (uint8_t)dEXTENDED_CAN_MSG_ID_2_0B;
229             tempCanMsg->frame.id = convertReg2ExtendedCANid(RXB1EIDH, RXB1EIDL,
                    RXB1SIDH, RXB1SIDL);
230         }
231         else
232         {
233             // message is standard
234             tempCanMsg->frame.idType = (uint8_t)dSTANDARD_CAN_MSG_ID_2_0B;
235             tempCanMsg->frame.id = convertReg2StandardCANid(RXB1SIDH, RXB1SIDL);
236         }
237
238         tempCanMsg->frame.dlc = RXB1DLC;
239         tempCanMsg->frame.data0 = RXB1D0;
240         tempCanMsg->frame.data1 = RXB1D1;
241         tempCanMsg->frame.data2 = RXB1D2;
242         tempCanMsg->frame.data3 = RXB1D3;
243         tempCanMsg->frame.data4 = RXB1D4;
244         tempCanMsg->frame.data5 = RXB1D5;
245         tempCanMsg->frame.data6 = RXB1D6;
246         tempCanMsg->frame.data7 = RXB1D7;
247         RXB1CONbits.RXFUL = 0;
248         returnValue = 1;
249     }
250     return (returnValue);
251 }
252
253 uint8_t CAN_messagesInBuffer(void)
254 {
255     uint8_t messageCount = 0;
256     if (RXB0CONbits.RXFUL != 0) // CheckRXB0
257     {
258         messageCount++;
259     }
260     if (RXB1CONbits.RXFUL != 0) // CheckRXB1
261     {
262         messageCount++;
263     }
264
265     return (messageCount);
266 }
267
268 uint8_t CAN_isBusOff(void)
269 {
270     uint8_t returnValue = 0;
271
272     // COMSTAT bit 5 TXBO: Transmitter Bus-Off bit
273     // 1 = Transmit error counter > 255
274     // 0 = Transmit error counter less then or equal to 255
275
276     if (COMSTATbits.TXBO == 1)
277     {
278         returnValue = 1;
```

```
279     }
280     return (returnValue);
281 }
282
283 uint8_t CAN_isRXErrorPassive(void)
284 {
285     uint8_t returnValue = 0;
286
287     // COMSTAT bit 3 RXBP: Receiver Bus Passive bit
288     // 1 = Receive error counter > 127
289     // 0 = Receive error counter less then or equal to 127
290
291     if (COMSTATbits.RXBP == 1)
292     {
293         returnValue = 1;
294     }
295     return (returnValue);
296 }
297
298 uint8_t CAN_isTXErrorPassive(void)
299 {
300     uint8_t returnValue = 0;
301
302     // COMSTAT bit 4 TXBP: Transmitter Bus Passive bit
303     // 1 = Transmit error counter > 127
304     // 0 = Transmit error counter less then or equal to 127
305
306     if (COMSTATbits.TXBP == 1)
307     {
308         returnValue = 1;
309     }
310     return (returnValue);
311 }
312
313 static uint32_t convertReg2ExtendedCANid(uint8_t tempRXBn_EIDH, uint8_t
        tempRXBn_EIDL, uint8_t tempRXBn_SIDH, uint8_t tempRXBn_SIDL)
314 {
315     uint32_t returnValue = 0;
316     uint32_t ConvertedID = 0;
317     uint8_t CAN_standardLo_ID_lo2bits;
318     uint8_t CAN_standardLo_ID_hi3bits;
319
320     CAN_standardLo_ID_lo2bits = (uint8_t)(tempRXBn_SIDL & 0x03);
321     CAN_standardLo_ID_hi3bits = (uint8_t)(tempRXBn_SIDL >> 5);
322     ConvertedID = (uint32_t)(tempRXBn_SIDH << 3);
323     ConvertedID = ConvertedID + CAN_standardLo_ID_hi3bits;
324     ConvertedID = (ConvertedID << 2);
325     ConvertedID = ConvertedID + CAN_standardLo_ID_lo2bits;
326     ConvertedID = (ConvertedID << 8);
327     ConvertedID = ConvertedID + tempRXBn_EIDH;
328     ConvertedID = (ConvertedID << 8);
329     ConvertedID = ConvertedID + tempRXBn_EIDL;
330     returnValue = ConvertedID;
331     return (returnValue);
332 }
333
334 static uint32_t convertReg2StandardCANid(uint8_t tempRXBn_SIDH, uint8_t
        tempRXBn_SIDL)
335 {
336     uint32_t returnValue = 0;
337     uint32_t ConvertedID;
338     // if standard message (11 bits)
339     // EIDH = 0 + EIDL = 0 + SIDH + upper three bits SIDL (3rd bit needs to be
            clear)
340     // 1111 1111 111
341     ConvertedID = (uint32_t)(tempRXBn_SIDH << 3);
342     ConvertedID = ConvertedID + (uint32_t)(tempRXBn_SIDL >> 5);
343     returnValue = ConvertedID;
344     return (returnValue);
345 }
346
347 static void convertCANid2Reg(uint32_t tempPassedInID, uint8_t canIdType, uint8_t
        *passedInEIDH, uint8_t *passedInEIDL, uint8_t *passedInSIDH, uint8_t *
        passedInSIDL)
348 {
349     uint8_t wipSIDL = 0;
350
351     if (canIdType == dEXTENDED_CAN_MSG_ID_2_0B)
352     {
353
354         // EIDL
355         *passedInEIDL = 0xFF & tempPassedInID; // CAN_extendedLo_ID_TX1 = &HFF
                And CAN_UserEnter_ID_TX1
356         tempPassedInID = tempPassedInID >> 8;  // CAN_UserEnter_ID_TX1 =
                CAN_UserEnter_ID_TX1 >> 8
357
358         // EIDH
359         *passedInEIDH = 0xFF & tempPassedInID; // CAN_extendedHi_ID_TX1 = &HFF
                And CAN_UserEnter_ID_TX1
360         tempPassedInID = tempPassedInID >> 8;  // CAN_UserEnter_ID_TX1 =
                CAN_UserEnter_ID_TX1 >> 8
361
362         // SIDL
363         // push back 5 and or it
364         wipSIDL = 0x03 & tempPassedInID;
```

```
365         tempPassedInID = tempPassedInID << 3; // CAN_UserEnter_ID_TX1 =
                CAN_UserEnter_ID_TX1 << 3
366         wipSIDL = (0xE0 & tempPassedInID) + wipSIDL;
367         wipSIDL = (uint8_t)(wipSIDL + 0x08);       // TEMP_CAN_standardLo_ID_TX1
                 = TEMP_CAN_standardLo_ID_TX1 + &H8
368         *passedInSIDL = (uint8_t)(0xEB & wipSIDL); // CAN_standardLo_ID_TX1 = &
                HEB And TEMP_CAN_standardLo_ID_TX1
369
370         // SIDH
371         tempPassedInID = tempPassedInID >> 8;
372         *passedInSIDH = 0xFF & tempPassedInID;
373     }
374     else //(canIdType == dSTANDARD_CAN_MSG_ID_2_0B)
375     {
376
377         *passedInEIDH = 0;
378         *passedInEIDL = 0;
379         tempPassedInID = tempPassedInID << 5;
380         *passedInSIDL = 0xFF & tempPassedInID;
381         tempPassedInID = tempPassedInID >> 8;
382         *passedInSIDH = 0xFF & tempPassedInID;
383     }
384
385     return;
386 }
```

A.10.3 ecan.h

```
1  /**
2   * mcc generated file
3   * modified by:
4   * Torben Dierks
5   * 2404082
6   * Supervising Professor:
7   * Prof. Dr. Rasmus Rettig
8   * description:
9   * mcc generated header file for can communication
10  *
11  */
12
13 #ifndef ECAN_H
14 #define ECAN_H
15
16 #include <stdbool.h>
17 #include <stdint.h>
18 #include "definitions.h"
19
20 typedef union
21 {
22     struct
23     {
24         uint8_t idType;
25         uint32_t id;
26         uint8_t dlc;
27         uint8_t data0;
28         uint8_t data1;
29         uint8_t data2;
30         uint8_t data3;
31         uint8_t data4;
32         uint8_t data5;
33         uint8_t data6;
34         uint8_t data7;
35     } frame;
36     uint8_t array[14];
37 } uCAN_MSG;
38
39 #define dSTANDARD_CAN_MSG_ID_2_0B 1
40 #define dEXTENDED_CAN_MSG_ID_2_0B 2
41
42 void ECAN_Initialize(void);
43 void CAN_sleep(void);
44 uint8_t CAN_transmit(uCAN_MSG *tempCanMsg);
45 uint8_t CAN_receive(uCAN_MSG *tempCanMsg);
46 uint8_t CAN_messagesInBuffer(void);
47 uint8_t CAN_isBusOff(void);
48 uint8_t CAN_isRXErrorPassive(void);
49 uint8_t CAN_isTXErrorPassive(void);
50 void ECAN_SetWakeUpInterruptHandler(void (*handler)(void));
51 #endif // ECAN_H
```

A.10.4 eeprom.c

```
1
2  /**
3   * author:
4   * Torben Dierks
5   * 2404082
6   * Supervising Professor:
7   * Prof. Dr. Rasmus Rettig
8   * description:
9   * source file with functions accessing the eeprom
10  *
```

```
 * code sequences taken from PIC18F2480/2580/4480/4580 datasheet p. 113
 */

/**
  Section: Included Files
*/
#include <xc.h>
#include "eeprom.h"

/**
 * writes data to the long term eeprom storage
 * @param uint8_t address: address in eeprom memory to write to. address range
       00h to FFh
 * @param uint8_t data: data to be written
 */
void writeEEPROM(uint8_t address, uint8_t data)
{
    // load address to write to
    EEADR = address;
    // load data ?to be written
    EEDATA = data;

    // clear EEPGD to access eeprom memory
    EECON1bits.EEPGD = 0;
    // clear CFGS to access eeprom memory
    EECON1bits.CFGS = 0;
    // enable writes
    EECON1bits.WREN = 1;

    // disable interrupts
    INTCONbits.GIE = 0;
    // exact code sequence to be followed according to datasheet p. 113
    EECON2 = 0x55;
    EECON2 = 0xAA;
    // initiate write
    EECON1bits.WR = 1;
    // wait until data is written. should only take one clock cycle
    // WR is cleared in hardware if data has been written
    while (1 == EECON1bits.WR)
        ;
    // enable interrupts

    PIR2bits.EEIF = 0; // clear interrupt signaling a succesful write

    // clear WREN to inhibit writes to eeprom memory
    EECON1bits.WREN = 0;

    INTCONbits.GIE = 1;

    // enable interrupts in general
    INTCONbits.GIE = 1;
}
/**
 * returns data at a given address in eeprom memory
 * @param uint8_t address: address in eeprom memory to read from. address range
       00h to FFh
 */
uint8_t readEEPROM(uint8_t address)
{
    // write address
    EEADR = address;
    // clear EEPGD to access eeprom memory
    EECON1bits.EEPGD = 0;
    // clear CFGS to access eeprom memory
    EECON1bits.CFGS = 0;
    // initiate read
    EECON1bits.RD = 1;

    // wait until data is ready. should only take one clock cycle
    // RD is cleared in hardware if data is available
    while (1 == EECON1bits.RD)
        ;

    return (EEDATA);
}
```

A.10.5 eeprom.h

```
/**
 * author:
 * Torben Dierks
 * 2404082
 * Supervising Professor:
 * Prof. Dr. Rasmus Rettig
  * description:
 * header file with functions accessing the eeprom
 */
// This is a guard condition so that contents of this file are not included
// more than once.
#ifndef eeprom
#define eeprom
/**
  Section: Included Files
*/
```

```
17 void writeEEPROM(uint8_t address, uint8_t data);
18 uint8_t readEEPROM(uint8_t address);
19 #endif // eeprom
```

A.10.6 interfacingSPI.c

```
1  /**
2   * author:
3   * Torben Dierks
4   * 2404082
5   * Supervising Professor:
6   * Prof. Dr. Rasmus Rettig
7   * description:
8   * source file with functions using the spi interface to communicate with the
        sensor
9   *
10  */
11
12 #include "interfacingSPI.h"
13
14 /**
15  * reads and returns the register content of sensor
16  * @param uint8_t pageID: register bank pageID on sensor
17  * @param uint8_t address: register address on sensor
18  */
19 uint16_t readRegister(uint8_t pageID, uint8_t address)
20 {
21     // check if the pageId of the given register is equal to the previous
22     // if the pageId is not equal change register bank
23     if (prevPageID != pageID)
24     {
25         // page select
26         CS = 0;
27         spiExchangeByte(WRITE_REGISTER);
28         spiExchangeByte(pageID);
29         CS = 1;
30         __delay_us(stallTime);
31     }
32     // save last used pageID
33     prevPageID = pageID;
34     // tell sensor which register to read
35     CS = 0;
36     spiExchangeByte(READ_REGISTER | address);
37     spiExchangeByte(0x00);
38     CS = 1;
39     // stall time between spi commands according to datasheet
40     __delay_us(stallTime);
41     // receiving sensor data
42     CS = 0;
43     uint16_t responseHighByte = ReadSPI();
44     uint16_t responseLowByte = ReadSPI();
45     CS = 1;
46     __delay_us(stallTime);
47     // stitching sensor data together
48     uint16_t resp = (responseHighByte << 8) | (responseLowByte);
49     return resp;
50 }
51 /**
52  * writes given data to register on sensor
53  * @param uint8_t pageID: register bank pageID on sensor
54  * @param uint8_t address: register address on sensor
55  * @param uint8_t registerDataHigh: high byte of data to be written to sensor
56  *  @param uint8_t registerDataHigh: high byte of data to be written to sensor
57  */
58 void inline writeRegister(uint8_t pageID, uint8_t address, uint8_t
        registerDataHigh, uint8_t registerDataLow)
59 {
60     /**
61      * sent lower byte first. lower byte == lower address value. Little Endian!
62      * registerDate has to be declared accordingly
63      */
64     // check if the pageId of the given register is equal to the previous
65     // if the pageId is not equal change register bank
66     if (prevPageID != pageID)
67     {
68         // page select
69         CS = 0;
70         spiExchangeByte(WRITE_REGISTER);
71         spiExchangeByte(pageID);
72         CS = 1;
73         __delay_us(stallTime);
74     }
75     prevPageID = pageID;
76     // write lower byte to address
77     CS = 0;
78     spiExchangeByte(WRITE_REGISTER | address);
79     spiExchangeByte(registerDataLow);
80     CS = 1;
81     __delay_us(stallTime);
82     // write higher byte to address + 1
83     CS = 0;
84     spiExchangeByte(WRITE_REGISTER | (address + 1));
85     spiExchangeByte(registerDataHigh);
86     CS = 1;
```

```
87      __delay_us(stallTime);
88      return;
89  }
```

A.10.7 interfacingSPI.h

```
1  /**
2   * author:
3   * Torben Dierks
4   * 2404082
5   * Supervising Professor:
6   * Prof. Dr. Rasmus Rettig
7   * description:
8   * header file with functions using the spi interface to communicate with the
        sensor
9   */
10
11 #ifndef interfacingSPI
12 #define interfacingSPI
13
14 /**
15   Section: Included Files
16 */
17 #include <xc.h>
18 #include <stdio.h>
19 #include <stdint.h>
20 #include "definitions.h"
21 #include "spi.h"
22
23 static uint8_t prevPageID = 0x00;
24 int stallTime = 16; // us t_stall datasheet sensor
25 uint16_t readRegister(uint8_t pageID, uint8_t address);
26 void inline writeRegister(uint8_t pageID, uint8_t address, uint8_t
       registerDataHigh, uint8_t registerDataLow);
27 void inline getMeasurementData(uint16_t *bufferArray, int samplesCount, uint8_t
       pageID, uint8_t address);
28 void inline testMeasurementData(int samplesCount, uint8_t pageID, uint8_t
       address);
29
30 #endif // interfacingSPI
```

A.10.8 setup.c

```
1  /**
2   * author:
3   * Torben Dierks
4   * 2404082
5   * Supervising Professor:
6   * Prof. Dr. Rasmus Rettig
7   * description:
8   * source file for setup functions
9   *
10  */
11
12 #include <xc.h>
13 #include "setup.h"
14 #include "definitions.h"
15
16 /**
17  *| Signal      | Direction | Port |   | Signal  | Direction | Port |
18 |-------------|-----------|------|---|---------|-----------|------|
19 | Silent      | O         | RA2  |   | BUSY    | I         | RB0  |
20 | LED2-       | O         | RA3  |   | CAN_RX  | I         | RB3  |
21 | Chip Select | O         | RA5  |   | SDI     | I         | RC4  |
22 | SYNC        | O         | RB1  |   | UART_RX | I         | RC7  |
23 | CAN_TX      | O         | RB2  |   |         |           |      |
24 | RST         | O         | RB4  |   |         |           |      |
25 | LED1-       | O         | RC0  |   |         |           |      |
26 | LED1+       | O         | RC1  |   |         |           |      |
27 | LED2+       | O         | RC2  |   |         |           |      |
28 | SCLK        | O         | RC3  |   |         |           |      |
29 | SDO         | O         | RC5  |   |         |           |      |
30 | UART_TX     | O         | RC6  |   |         |           |      |
31  *
32  *
33  *
34  */
35 /**
36  * settings for I/O ports used
37  */
38 void portSetup()
39 {
40      // every IO is an INPUT initially and set output latches 0
41      TRISA = 0xFF;
42      TRISB = 0xFF;
43      TRISC = 0xFF;
44      // clear ports
45      LATA = 0;
46      LATB = 0;
47      LATC = 0;
```

```
48  /*
        ********************************************************************************
49   *                          Output ports
50   *
51   ********************************************************************************
         */
52  // port A
53  // Silent
54  TRISAbits.RA2 = 0;
55  // CS
56  TRISAbits.RA5 = 0;
57
58  // Port B
59  //  Sync
60  TRISBbits.RB1 = 0;
61  // CAN TX
62  TRISBbits.RB2 = 0;
63  // RST
64  TRISBbits.RB4 = 0;
65  // port C
66  // UART_TX
67  TRISCbits.RC6 = 0;
68  // SCK
69  TRISCbits.RC3 = 0;
70  // SDO
71  TRISCbits.RC5 = 0;
72  // CS
73  TRISAbits.RA5 = 0;
74  // LEDs
75  // LED1
76  TRISCbits.RC1 = 0;
77  TRISCbits.RC0 = 0;
78  // LED2
79  TRISCbits.RC2 = 0;
80  TRISAbits.RA3 = 0;
81  // Predefined states
82  RST = 1; // active low
83  CS = 1;  // active low
84  /*
        ********************************************************************************
85   *                          Input ports
86   *
87   ********************************************************************************
         */
88  // port B
89  // BUSY
90      TRISBbits.RB0 = 1;
91  *   LATBbits.LATB0 = 0;
92      ADCON1bits.PCFG = 0b1111;
93      // CAN_RX; user must ensure bit is set
94      TRISBbits.RB3 = 1;
95      // port C
96      // CAN_RX
97      TRISCbits.RC7 = 1;
98  }
99  /**
100  * interrupt setup
101  */
102 void interruptSetup()
103 {
104     // enable interrupts in general
105     INTCONbits.GIE = 1;
106     // enable peripheral interrupts
107     INTCONbits.PEIE = 1;
108     // enables priorities for interrupts
109     RCONbits.IPEN = 0;
110     //    // interrupts from sensor busy pin
111     INTCONbits.INT0E = 1;
112     // 0 falling edge 1 rising edge
113     INTCON2bits.INTEDG0 = 1; // trigger on rising edge of busy signal signaling
            data is ready
114     // SPI
115     // enable interrupts for spi
116     PIE1bits.SSPIE = 1;
117
118     // MSSP interrupts are high priority
119     IPR1bits.SSPIP = 1;
120     // CAN
121     // ECAN_SetWakeUpInterruptHandler(WakeUpDefaultInterruptHandler);
122     PIR3bits.WAKIF = 0;
123     // CAN interrupt setup
124     // PIR interrupt flags. do not use in setup. needed to recognize
125     // interrupts and should be cleared after an interrupt has occurred
126     // PIE 3 contains main sources for can interrupts
127     // Receive related Interrupts
128     PIE3bits.IRXIE = 1;  // CAN invalid message received
129 *   PIE3bits.WAKIE = 1;  // CAN activity wake up
130     PIE3bits.ERRIE = 0;  // CAN bus error
131     PIE3bits.RXB1IE = 1; // get set when a new can message is received. receive
            buffer 1
132     PIE3bits.RXB0IE = 1; // get set when a new can message is received. receive
            buffer 0
133     // Transmit related interrupts
```

```
134     PIE3bits.TXB2IE = 0; // gets set when the associated buffer is empty.
            transmit buffer 2.
135     PIE3bits.TXB1IE = 0; // gets set when the associated buffer is empty.
            transmit buffer 1
136     PIE3bits.TXB0IE = 0; // gets set when the associated buffer is empty.
            transmit buffer 0
137     // CAN interrupt priority (1 == high priority, 0 == low priority))
138     IPR3bits.IRXIP = 0;  // CAN invalid message received
139     IPR3bits.WAKIP = 0;  // CAN activity wake up
140     IPR3bits.ERRIP = 0;  // CAN bus error
141     IPR3bits.RXB1IP = 1; // get set when a new can message is received. receive
            buffer 1
142     IPR3bits.RXB0IP = 1; // get set when a new can message is received. receive
            buffer 0
143     // Transmit related interrupts
144     IPR3bits.TXB2IP = 0; // gets set when the associated buffer is empty.
            transmit buffer 2.
145     IPR3bits.TXB1IP = 0; // gets set when the associated buffer is empty.
            transmit buffer 1
146     IPR3bits.TXB0IP = 0; // gets set when the associated buffer is empty.
            transmit buffer 0
147 }
148 void changeBaudrateCAN(uint8_t valueBRGCON1, uint8_t valueBRGCON2, uint8_t
        valueBRGCON3)
149 {
150     CANCON = 0x80;
151     while (0x80 != (CANSTAT & 0xE0))
152         ; // wait until ECAN is in config mode
153
154     ECANCON = 0x00;
155     CIOCON = 0x00;
156
157     RXM0EIDH = 0xFF;
158     RXM0EIDL = 0xFF;
159     RXM0SIDH = 0xFF;
160     RXM0SIDL = 0xE3;
161     RXM1EIDH = 0xFF;
162     RXM1EIDL = 0xFF;
163     RXM1SIDH = 0xFF;
164     RXM1SIDL = 0xE3;
165
166     RXF0EIDH = 0x00;
167     RXF0EIDL = 0x00;
168     RXF0SIDH = 0xA0;
169     RXF0SIDL = 0x00;
170     RXF1EIDH = 0x00;
171     RXF1EIDL = 0x00;
172     RXF1SIDH = 0x00;
173     RXF1SIDL = 0x00;
174     RXF2EIDH = 0x00;
175     RXF2EIDL = 0x00;
176     RXF2SIDH = 0x00;
177     RXF2SIDL = 0x00;
178     RXF3EIDH = 0x00;
179     RXF3EIDL = 0x00;
180     RXF3SIDH = 0x00;
181     RXF3SIDL = 0x00;
182     RXF4EIDH = 0x00;
183     RXF4EIDL = 0x00;
184     RXF4SIDH = 0x00;
185     RXF4SIDL = 0x00;
186     RXF5EIDH = 0x00;
187     RXF5EIDL = 0x00;
188     RXF5SIDH = 0x00;
189     RXF5SIDL = 0x00;
190
191     BRGCON1 = valueBRGCON1;
192     BRGCON2 = valueBRGCON2;
193     BRGCON3 = valueBRGCON3;
194
195     PIR3bits.WAKIF = 0;
196     PIE3bits.WAKIE = 1;
197
198     CANCON = 0x00;
199     while (0x00 != (CANSTAT & 0xE0))
200         ; // wait until ECAN is in Normal mode
201 }
```

A.10.9 setup.h

```
1 /**
2  * author:
3  * Torben Dierks
4  * 2404082
5  * Supervising Professor:
6  * Prof. Dr. Rasmus Rettig
7  * description:
8  * header file for setup functions
9  *
10 */
11
12 // This is a guard condition so that contents of this file are not included
13 // more than once.
14 #ifndef XC_HEADER_TEMPLATE_H
```

```
#define XC_HEADER_TEMPLATE_H

#include <xc.h> // include processor files - each processor file is guarded.

void portSetup();
void interruptSetup();
void writeBaudrateCANtoEEPROM();
void changeBaudrateCAN(uint8_t valueBRGCON1, uint8_t valueBRGCON2, uint8_t
      valueBRGCON3);
#endif /* XC_HEADER_TEMPLATE_H */
```

A.10.10 spi.c

```
/**
 * Library taken from: https://github.com/microchip-pic-avr-examples/pic18f47q10
      -cnano-spi-master-send-mcc/tree/master/pic18f47q10-cnano-spi-master-send-
      mcc.X
 * Last Accessed on: 08.03.2023
 * Adapated for the bachelor thesis by:
 * Torben Dierks
 * 2404082
 * Supervising Professor:
 * Prof. Dr. Rasmus Rettig
 *
 */

#include "spi.h"

typedef struct
{
    uint8_t con1;
    uint8_t stat;
    uint8_t add;
    uint8_t operation;
} spi1_configuration_t;

// con1 == SSPxCON1, stat == SSPxSTAT, add == SSPxADD, operation == Master/Slave
static const spi1_configuration_t spi1_configuration[] = {
    {0xa, 0x40, 0x1, 0}};

// initialize spi for use case
void spiInitialize()
{

    SSPSTATbits.SMP = 0; // data sample on middle
    SSPSTATbits.CKE = 0; // transmit on transition from idle to active
    SSPCON1bits.CKP = 1; // idle state is high level

    SSPCON1bits.SSPEN = 1;   // enable spi on pins
    SSPCON1bits.SSPM = 0000; // f_spi = 32 MHz/ 4 = 8 MHz
    // defined state of chip select line
    CS = 1;
}

uint8_t inline spiExchangeByte(uint8_t data)
{

    SSPBUF = data;
    while (!PIR1bits.SSPIF)
    {
    }

    SSPSTATbits.BF = 0;
    PIR1bits.SSPIF = 0;
    return SSPBUF;
}

uint8_t inline ReadSPI(void)
{
    unsigned char TempVar;
    TempVar = SSPBUF;   // Clear BF
    PIR1bits.SSPIF = 0; // Clear interrupt flag
    SSPBUF = 0x00;      // initiate bus cycle
    while (!PIR1bits.SSPIF)
        ;            // wait until cycle complete
    return (SSPBUF); // return with byte read
}

void spiClose(void)
{
    SSPCON1bits.SSPEN = 0;
}
```

A.10.11 spi.h

```
/**
 * Library taken from: https://github.com/microchip-pic-avr-examples/pic18f47q10
      -cnano-spi-master-send-mcc/tree/master/pic18f47q10-cnano-spi-master-send-
      mcc.X
 * Last Accessed on: 08.03.2023
 * Adapated for the bachelor thesis by:
 * Torben Dierks
```

```
 * 2404082
 * Supervising Professor:
 * Prof. Dr. Rasmus Rettig
 * Description:
 * header file for the spi bus interface
 *
 */
#ifndef SPI_H
#define SPI_H

#include <xc.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "definitions.h"

/* SPI interfaces */
typedef enum
{
  SPI1_DEFAULT
} spi1_modes_t;

void inline spiInitialize();
uint8_t inline spiExchangeByte(uint8_t data);
uint8_t inline ReadSPI(void);
void inline spiClose(void);

#endif // SPI_H
```

A.10.12 definitions.h

```
/**
 * author:
 * Torben Dierks
 * 2404082
 * Supervising Professor:
 * Prof. Dr. Rasmus Rettig
 * description:
 * header file containing global definitions
 *
 */

#ifndef HEADER_DEFINITIONS
#define HEADER_DEFINITIONS

#include <xc.h> // include processor files - each processor file is guarded.

/*===========================================================================
 * Output Pins
 *===========================================================================*/
#define SILENT LATAbits.LATA2
#define LED2_M LATAbits.LATA3
#define CS LATAbits.LATA5
#define IMU_SYNC LATBbits.LATB1
#define CAN_TX LATBbits.LATB2
#define RST LATBbits.LATB4
#define LED1_M LATCbits.LATC0
#define LED1_P LATCbits.LATC1
#define LED2_P LATCbits.LATC2
#define SCLK LATCbits.LATC3
// #define SDO LATCbits.LATC5
#define UART_TX LATCbits.LATC6

/*===========================================================================
 * IMU
 *===========================================================================*/
// recording modes

#define RECORDING_MODE_MFFT 0x0
#define RECORDING_MODE_AFFT 0x1
#define RECORDING_MODE_MTC 0x2
#define RECORDING_MODE_RTS 0x3
#define RECORDING_MODE_CONFIGURATION 0x4
#define RECORDING_MODE_NOT_DEFINED 0xF

// samples count
#define SAMPLES_COUNT_FFT 2048
#define SAMPLES_COUNT_MTC 4096

// vel or acc
#define MTC_MODE_ACCELERATION 0
#define MTC_MODE_VELOCITY 1

/*===========================================================================
 *external oscillator frequency
 *===========================================================================*/
#define _XTAL_FREQ 32000000

/*===========================================================================
 *interrupts
 *===========================================================================*/
```

```
 */
#define INT_FLAG_CAN_BUS_ERROR_MESSAGE_RECEIVE PIR3bits.IRXIF
#define INT_FLAG_CAN_BUS_ACTIVITY_WAKE_UP PIR3bits.WAKIF
#define INT_FLAG_CAN_MODULE_ERROR PIR3bits.ERRIF
#define INT_FLAG_CAN_TRANSMIT_BUFFER_2 PIR3bits.TXB2IF
#define INT_FLAG_CAN_TRANSMIT_BUFFER_1 PIR3bits.TXB1IF
#define INT_FLAG_CAN_TRANSMIT_BUFFER_0 PIR3bits.TXB0IF
#define INT_FLAG_CAN_REICEIVE_BUFFER_1 PIR3bits.RXB1IF
#define INT_FLAG_CAN_REICEIVE_BUFFER_0 PIR3bits.RXB0IF

/*==============================================================================
 * CAN
 *==============================================================================
 */
#define CAN_DATA_LENGTH_8_BYTES 8
#define CAN_DATA_LENGTH_7_BYTES 7
#define CAN_DATA_LENGTH_6_BYTES 6
#define CAN_DATA_LENGTH_5_BYTES 5
#define CAN_DATA_LENGTH_4_BYTES 4
#define CAN_DATA_LENGTH_3_BYTES 3
#define CAN_DATA_LENGTH_2_BYTES 2
#define CAN_DATA_LENGTH_1_BYTES 1
#define CAN_DATA_LENGTH_0_BYTES 0

#define CAN_MSG_TYPE_SETTING 0x1
#define CAN_MSG_TYPE_REGISTER_CONTENT 0x2
#define CAN_MSG_TYPE_START_CAPTURE 0x3
#define CAN_MSG_TYPE_GET_SETTINGS 0x4
#define CAN_MSG_TYPE_STOPP_AFFT 0x5
#define CAN_MSG_TYPE_CONFIGURATION_IMU_START 0x6
#define CAN_MSG_TYPE_CONFIGURATION_IMU_STOP 0x7
#define CAN_MSG_TYPE_CHANGE_CAN_SETTINGS 0x8
#define CAN_MSG_TYPE_AUTONULL 0x9
#define CAN_MSG_TYPE_RESET_SENSOR 0xA
#define CAN_MSG_TYPE_ESCAPED 0xB
#define CAN_MSG_TYPE_SAVE_REGISTER 0xC

#define CAN_IDENTIFIER_PYTHON 0x400

#define START_FFT_DATA_TRANSMISSON 0x92
#define STOP_FFT_DATA_TRANSMISSON 0x94
#define FFT_DATA_TRANSMISSON 0x96

#define START_MTC_DATA_TRANSMISSON 0xE2
#define STOP_MTC_DATA_TRANSMISSON 0xE4
#define MTC_DATA_TRANSMISSON 0xE6

#define DATA_TRANSMISSON_ERROR 0xF6

#define FFT_X_DATA 0x80
#define FFT_Y_DATA 0x81
#define FFT_Z_DATA 0x82

// predefined baudrate settings

#define CAN_BAUD_1_MBPS 0xA1
#define CAN_BAUD_800_kbps 0xA2
#define CAN_BAUD_500_kbps 0xA3
#define CAN_BAUD_250_kbps 0xA4
/**
  Baud rate: 1Mbps
  System frequency: 32000000
  ECAN clock frequency: 32000000
  Time quanta: 8
  Sample point: 1-1-4-2
  Sample point: 75%
*/

#define CAN_BAUD_1_MBPS_BRGCON1 0x01
#define CAN_BAUD_1_MBPS_BRGCON2 0x98
#define CAN_BAUD_1_MBPS_BRGCON3 0x01

#define MEM_LOC_1_MBPS_BRGCON1 0x01
#define MEM_LOC_1_MBPS_BRGCON2 0x02
#define MEM_LOC_1_MBPS_BRGCON3 0x03

/**
  Baud rate: 800kbps
  System frequency: 32000000
  ECAN clock frequency: 32000000
  Time quanta: 10
  Sample point: 1-1-6-2
  Sample point: 80.000%
*/

#define CAN_BAUD_800_KBPS_BRGCON1 0x01
#define CAN_BAUD_800_KBPS_BRGCON2 0xA8
#define CAN_BAUD_800_KBPS_BRGCON3 0x01

#define MEM_LOC_800_KBPS_BRGCON1 0x04
#define MEM_LOC_800_KBPS_BRGCON2 0x05
#define MEM_LOC_800_KBPS_BRGCON3 0x06

/**
  Baud rate: 500kbps
  System frequency: 32000000
```

```
160   ECAN clock frequency: 32000000
161   Time quanta: 8
162   Sample point: 1-1-4-2
163   Sample point: 75%
164   */
165
166 #define CAN_BAUD_500_KBPS_BRGCON1 0x03
167 #define CAN_BAUD_500_KBPS_BRGCON2 0x98
168 #define CAN_BAUD_500_KBPS_BRGCON3 0x01
169
170 #define MEM_LOC_500_KBPS_BRGCON1 0x07
171 #define MEM_LOC_500_KBPS_BRGCON2 0x08
172 #define MEM_LOC_500_KBPS_BRGCON3 0x09
173
174 /**
175   Baud rate: 250kbps
176   System frequency: 32000000
177   ECAN clock frequency: 32000000
178   Time quanta: 8
179   Sample point: 1-1-4-2
180   Sample point: 75%
181   */
182
183 #define CAN_BAUD_250_KBPS_BRGCON1 0x07
184 #define CAN_BAUD_250_KBPS_BRGCON2 0x98
185 #define CAN_BAUD_250_KBPS_BRGCON3 0x01
186
187 #define MEM_LOC_250_KBPS_BRGCON1 0x0A
188 #define MEM_LOC_250_KBPS_BRGCON2 0x0B
189 #define MEM_LOC_250_KBPS_BRGCON3 0x0C
190
191 #define MEM_LOC_RECORDING_MODE 0xA1
192
193 /*============================================================================
194  *Analog Devices ADcmXL3021
195  *============================================================================
196  */
197 // register access
198 #define WRITE_REGISTER 0x80
199 #define READ_REGISTER 0x00
200
201 // register addresses
202 #define ADDRESS_X_Buffer 0x0E
203 #define ADDRESS_Y_Buffer 0x10
204 #define ADDRESS_Z_Buffer 0x12
205 #define ADDRESS_BUF_PTR 0x0A
206 #define ADDRESS_REC_INFO1 0x66
207 #define ADDRESS_REC_INFO2 0x68
208 #define ADDRESS_FFT_AVG1 0x06
209 #define ADDRESS_FFT_AVG2 0x08
210 #define ADDRESS_REC_CTRL 0x1A
211 #define ADDRESS_REC_PRD 0x1E
212 #define ADDRESS_AVG_CNT 0x3A
213 #define ADDRESS_GLOB_CMD 0x3E
214
215 // page id config register
216 #define PAGE_ID 0x00
217
218 // escape code to end data acquisition of IMU
219 #define ESCPAE_CODE_HIGH 0x00
220 #define ESCAPE_CODE_LOW 0xE8
221
222 // sample rate options
223 #define SR0_ENABLED 0x01
224 #define SR1_ENABLED 0x02
225 #define SR2_ENABLED 0x04
226 #define SR3_ENABLED 0x08
227
228 #endif /*HEADER_DEFINITIONS */
```

B Inhalt der CD

- *bachelorarbeit_dierks_torben_2404082.pdf*
- *abschlussmessung.zip*
  Komprimiertes Verzeichnis mit allen Messdaten aus der Abschlussmessung.
- *Ordner Programme*
  - *adcmxl_3021*
    In C programmiertes Programm auf dem Mikrocontroller.
  - *can_python*
    Python Scripte für den Empfang von Messwerten und das Konfigurieren des Mikrocontrollers über CAN.
- *Ordner kicad_files*
  - *cad*
    Ordner mit allen Footprints und Symbols der verwendeten Bauteile.
  - *pcb_ADcmXL3021*
    Schaltplan und Platine für Platine 1
  - *pcb_can_supply*
    Schaltplan und Platine für Platine 1
- *Ordner cad_files*
  - *adapter*
    CAD-Dateien für Adapter zur Schwingprüfanlage
  - *case*
    CAD-Dateien für das Gehäuse

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Entwicklung eines Beschleunigungssensors für hohe Frequenzen mit CAN-Schnittstelle

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der Bachelorarbeit ist erfolgt durch:

Ort	Datum	Unterschrift im Original