Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Marc Gabriel Thom

Hard- und Softwareentwicklung von einem Beschleunigungs-Datenlogger für Kraftfahrzeuge

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Marc Gabriel Thom

Hard- und Softwareentwicklung von einem Beschleunigungs-Datenlogger für Kraftfahrzeuge

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

Betreuender Prüfer : Prof. Dr. Robert Heß
Zweitgutachter : Prof. Dr. –Ing. Karl-Ragmar Riemschneider

Abgegeben am 11. September 2014

Marc Gabriel Thom

Thema der Bachelorarbeit

Hard- und Softwareentwicklung von einem Beschleunigungs-Datenlogger für Kraftfahrzeuge

Stichworte

Drei-Achsen-Beschleunigungssensor, ARM Cortex-M4 Mikrocontroller, SD-Karte, Mikro-SD-Karte, GPS-Empfänger, Echt-Zeit-Uhr, Digital-Halbleitertemperatursensor

Kurzzusammenfassung

Diese Arbeit beschreibt die Hard- und Softwareentwicklung von einem Beschleunigungs-Datenlogger für Kraftfahrzeuge. Bei dem, für die Arbeit verwendeten, Mikrocontroller handelt es sich um einen ARM Cortex-M4. Dieser befindet sich auf dem Stellaris LM4F120 LaunchPad Evaluation Kit von Texas Instruments. Der Drei-Achsen-Beschleunigungssensor, die Echt-Zeit-Uhr, der Digital-Halbleitertemperatursensor, ein Steckplatz für eine Mikro-SD-Karte sowie Anschlussmöglichkeiten für den GPS-Empfänger, Steckplatz für eine SD-Karte und weitere Sensoren befinden sich auf einer Booster Platine, welche auf der Unterseite des Stellaris LM4F120 LaunchPad Evaluation Kit gesteckt wird.

Marc Gabriel Thom

Title of the paper

Hard- and software development of an acceleration data logger for motor vehicles

Keywords

Three-axis acceleration sensor, ARM Cortex-M4 microcontroller, SD-card, Micro-SD-card, GPS-receiver, Real-Time-Clock, digital semiconductor temperature sensor

Abstract

This paper describes the hard- and software development of an acceleration data logger for motor vehicles. The microcontroller which is used in this paper is an ARM Cortex-M4. This is located on the Stellaris Evaluation Kit LM4F120 LaunchPad from Texas Instruments. The three-axis accelerometer, Real-Time-Clock, the digital semiconductor temperature sensor, a slot for a micro SD card, such as facilities for the GPS receiver, a slot for an SD-card and other sensors are located on a booster board, which is plugged on the bottom of the Stellaris evaluation Kit LM4F120 LaunchPad.

Inhaltsverzeichnis

Vorwort

Mein Dank gilt meinen Eltern, die mir das Studium durch finanzielle Unterstützung ermöglicht haben.

Danken möchte ich auch Herrn Professor Dr. Heß, der mir während des gesamten Entwicklungsprozesses für die Bachelorthesis in zahlreichen persönlichen Gesprächen stets mit Rat zur Verfügung gestanden hat.

Des Weiteren danke ich Herrn Professor Dr. -Ing. Riemschneider und den Mitarbeitern vom Projekt BATSEN, die mir in Teamrunden und in persönlichen Gesprächen Ideen zur Durchführung meiner Arbeit lieferten.

Abschließend danke ich meinem ehemaligen Kommilitonen Simon Neugebauer, der mir im Bereich der Softwareentwicklung für Rückfragen zur Verfügung stand.

Kapitel 1

1 Einleitung

In diesem Kapitel wird die Motivation, die der Arbeit zugrunde liegt, dargelegt. Die sich darauf basierenden Ziele werden erörtert und eine Eingrenzung der Thematik wird vorgenommen. Am Ende des Kapitels wird der weitere Aufbau der Bachelorthesis aufgezeigt.

1.1 Motivation

Unter dem Aspekt, dass bei Diesel- und Benzinkraftstoffen die Bestandteilzusammensetzung hauptsächlich auf fossilen Stoffen basiert und sie somit eine begrenzte Verfügbarkeit aufweisen, besteht das Bedürfnis, eine zukunftsweisende Alternative als Energiequelle für Kraftfahrzeuge zu verwenden. Des Weiteren resultiert aus der Energieumwandlung von fossilen Stoffen ein generelles Umwelt- und Gesundheitsproblem. Dies äußert sich durch eine kontinuierlich ansteigende Schadstoffbelastung in Großstädten, da hier das Verkehrsaufkommen besonders hoch ist.

Eine dieser Alternativen sieht einen reinen Elektroantrieb mit Energiespeicher durch Batterien vor. Da diese Energiespeicherung aber nicht nur Vorteile mit sich bringt, wie die punktuelle Reduzierung von Schadstoffausstoß in den Städten, die mögliche Unabhängigkeit von fossilen Brennstoffen durch Energie aus regenerativen Kraftwerken und die reduzierte Lärm- und Geruchsbelastung, gibt es auch Nachteile wie z. B. geringere Reichweite, längere Energienachladezeit und mangelnde Vernetzung von öffentlichen Ladestationen. Aus diesem Grund werden momentan hauptsächlich, als Alternative, Kraftfahrzeuge mit Hybridantrieb verwendet. Solche sind beispielsweise Hybridbusse, die derzeit von der Freien und Hansestadt Hamburg für die Personenbeförderung in der Stadt getestet werden.

Die Abbildung 1.1 auf der folgenden Seite zeigt einen solchen Plug-In-Hybridbus (PHEV).

Abbildung 1.1: Plug-In-Hybridbus [1]

Diese haben bei einem Betrieb mit ausschließlich nur elektrischer Energie eine Mindestreichweite von 7 km. Bei zu geringer elektrischer Energie wird, um den Betrieb zu gewährleisten, ein Dieselmotor zugeschaltet. Zukünftig soll die Verwendung des Dieselmotors nur für den Notfall dienen. Die Batterien sollen dann an den Bushaltestellen über den Stromabnehmer am Dach des Busses nachgeladen werden.

Abbildung 1.2: Plug-In-Hybridbus Stromabnehmer [2]

Eine vollständige Nachladung erfolgt über Nacht. Hierfür werden ca. 6 h benötigt, wenn die Batterien entladen sind.

1.2 Zielsetzung

Ziel dieser Bachelorthesis ist, einen Beschleunigungs-Datenlogger zu entwickeln. Dieser soll später dazu verwendet werden, in Kraftfahrzeugen und speziell in den Stadtbussen der Freien und Hansestadt Hamburg auf verschiedenen Buslinien, Beschleunigungsdaten zu erfassen und zu speichern. Die Daten sollen dann unter dem Aspekt der Beschleunigung ausgewertet werden. Spezielles Augenmerk liegt hierbei auf der negativen Beschleunigung, die bei Bremsvorgängen auftreten. Elektrobetriebene Kraftfahrzeuge mit Batterie als Energiespeicher können diese auftretenden Kräfte durch Rekuperation zur kurzzeitigen Energierückgewinnung nutzen. Die Rekuperationsbremsung erfolgt hierbei durch die Verwendung des elektrischen Motors als Generator. Dadurch wird der Nachteil der geringen Reichweite dieser Antriebs- und dessen Energiespeicherform reduziert, wodurch die Energienachladung zeitlich nach verzögert werden kann. Anhand der gesammelten Daten soll im Vorfeld Aussage über eine mögliche Energierückgewinnung getroffen werden. Hierdurch lassen sich die Reichweiten und somit die Einsatzzeiten der Busse auf den verschiedenen Buslinien besser ermitteln und Positionen für Ladestationen zur Nachladung optimieren. Bei einem langsamen Umstieg auf Busse mit Elektroantrieb und Batteriespeicher, lassen sich hierdurch die geeignetsten Buslinien für die ersten Busse ermitteln, um einen möglichst finanziell wirtschaftlichen Umstieg zu erzielen.

1.3 Abgrenzung

Die Bachelorthesis beschäftigt sich primär mit der Herstellung des Beschleunigungs-Datenloggers und dessen Komponenten sowie die Softwarerealisierung. Sekundär wird Bezug auf den Test des Beschleunigungs-Datenloggers auf dessen Funktionalität genommen. Hierfür wurden unter Laborbedingungen und im Feldversuch, unter Verwendung eines Testkraftfahrzeugs, Testdaten erhoben und unter dem Aspekt der Funktionalität hin ausgewertet.

Der Einsatz in Kraftfahrzeugen bzw. in Stadtbussen in Hamburg, zwecks Datenerfassung in Bezug auf die Auswertung der Energierückgewinnung durch Rekuperation sowie die Ermittlung der günstigsten Buslinien und vorteilhafte Positionen für Ladestationen, wird aus Zeitgründen nicht vorgenommen.

1.4 Inhaltlicher Aufbau der Arbeit

Die Bachelorthesis setzt sich aus sieben Kapiteln zusammen.

Im zweiten Kapitel wird auf die Grundlagen eingegangen, die für den Entwurf des Beschleunigungs-Datenloggers notwendig sind.

Das dritte Kapitel handelt von der Spezifikation des Beschleunigungs-Datenloggers.

Hier werden die gesetzten Anforderungen dargelegt, Vorüberlegungen werden getroffen und eine Komponentenauswahl wird durchgeführt.

Im vierten Kapitel wird auf die Hardwareentwicklung eingegangen.

Das fünfte Kapitel stellt die Softwareentwicklung vor. Hier wird Bezug auf die unterschiedlichen Protokolle genommen, die zur Kommunikation zwischen Mikrocontroller und den Komponenten verwendet werden.

Das sechste Kapitel beschäftigt sich mit den Testphasen des Beschleunigungs-Datenloggers unter dem Aspekt der Funktionalität.

Die Bachelorthesis schließt mit dem siebten Kapitel ab. Hier folgt eine kurze Zusammenfassung. Es werden Optimierungsvorschläge getroffen und zukünftige, weitere Anwendungsmöglichkeiten erörtert.

Kapitel 2

2 Grundlagen

Die hier vermittelten Grundlagen stellen die Basis dar, auf der der Beschleunigungs-Datenlogger entwickelt wurde.

2.1 Formeln

Die Beschleunigung (a) setzt sich aus der Kraft (F) und der Masse (m) zusammen.

$$a = \frac{F}{m} \tag{2.1.1}$$

Die Geschwindigkeit (v) setzt sich aus dem Weg (s), der in einer Zeit (t) zurückgelegt wurde, zusammen.

$$v = \frac{s}{t} \tag{2.1.2}$$

Der Bremsweg (s) ist die Strecke, die ein bewegtes Objekt zurücklegt, um in seine Ruhelage zu kommen. Für ein Kraftfahrzeug ohne jegliche Verzögerung setzt sich dies aus der Geschwindigkeit (v) und der Beschleunigung (a) zusammen.

$$s = \frac{v^2}{2 \cdot a} \tag{2.1.3}$$

Die kinetische Energie (E_{kin}) eines Objekts setzt sich aus seiner Masse (m) und seiner Geschwindigkeit (v) zusammen.

$$E_{kin} = \frac{1}{2} \cdot \ m \cdot v^2 \tag{2.1.4}$$

Die Strom- Spannungskennlinie eines elektrischen Widerstands (R) verläuft im Idealfall linear. Er berechnet sich somit aus der Spannung (U) und dem Strom (I).

$$R = \frac{U}{I} \tag{2.1.5}$$

Die elektrische Energie (E_{elek}) ergibt sich bei konstantem Strom (I) und konstanter Spannung (U) aus der Zeitdifferenz ($\Delta(t)$).

$$E_{elek} = U \cdot I \cdot \Delta(t) \tag{2.1.6}$$

Die Kapazität (C) setzt sich aus der Ladung (Q) und der Spannung (U) zusammen.

$$C = \frac{Q}{U} \tag{2.1.7}$$

2.2 Mikrocontroller

Ein moderner Mikrocontroller ist in der Regel ein Halbleiterchip, der aus einem Mikroprozessor, einem Arbeitsspeicher und einem Programmspeicher besteht und auf dem Peripheriefunktionen wie z.B. SSI, I^2C, UART und CAN realisiert sind. Diese Peripheriefunktionen ermöglichen es ihm, mit anderen angeschlossenen Komponenten zu kommunizieren.

Für den Betrieb wird der Mikroprozessor in der Regel getaktet wobei der Takt mit einem Quarz Oszillator erzeugt wird. Seine Programmierung kann z.B. durch die Programmiersprache Assembler oder C erfolgen.

Verwendung findet der Mikrocontroller in fast allen technischen Artikeln wie z.B. im Handy, im Fernseher und vielem mehr.

2.3 Beschleunigung

Die Beschleunigungen stellt eine Bewegungsänderung eines Objekts dar. Bewegungsänderungen kommen durch Geschwindigkeitszunahme, Geschwindigkeitsabnahme und durch Richtungsänderungen zustande.

Das Formelzeichen für Beschleunigung ist a und die gängige Einheit sind m/s^2. Gebräuchlich sind auch die Einheit g für 9,81 m/s^2 oder auch Gal für 0,01 m/s^2.

Um Beschleunigungen zu messen, wird beispielsweise ein Beschleunigungssensor verwendet, bei denen in der Regel die wirkende Trägheitskraft auf einer Testmasse bestimmt wird. Für möglichst kleine Beschleunigungssensoren kommen sogenannte Mikro-elektro-mechanische-Systeme (MEMS) zum Einsatz. Diese bestehen vornehmlich aus einer Siliciumfeder, an der sich eine Siliciummasse befindet. Durch Beschleunigungsänderungen kommt es durch die veränderte Position der federnd aufgehängten Siliciummasse bezüglich zu einer festen Elektrode zu einer Kapazitätsänderung. Diese wird erfasst und zur Bestimmung der Beschleunigung verwendet.

2.4 Global-Positioning-System

Das Global-Positioning-System oder auch kurz GPS genannt ist ein Positionsbestimmungssystem. Zur Positionsbestimmung muss das GPS-Empfangsmodul von mindesten vier Satelliten deren Position und Uhrzeit gleichzeitig empfangen. Aus diesen Signalen werden im GPS-Empfangsmodul die Pseudo-Signallaufzeiten[1] gemessen. Durch die Positionskenntnis der Satelliten können hieraus die Ortsposition und die Höhenposition des Empfangsmoduls bestimmt werden. Positionsänderungen des bewegten Empfangsmoduls werden in seine Geschwindigkeit umgerechnet.

Das GPS-Empfangsmodul gibt in der Regel das Kommunikationsprotokoll NMEA 0183 [3] zurück. Dieses besteht minimal aus dem RMC Datensatz, der sich aus Uhrzeit, Datum, Geschwindigkeit und Positionsdaten zusammensetzt.

[1]Durch kleinste Uhrungenauigkeiten bei den Satelliten kommt es zu einer ungenauen Signallaufzeit, somit zu nicht richtigen, also Pseudo-Signallaufzeit.

Kapitel 3

3 Spezifikation

In diesem Kapitel werden die gesetzten Anforderungen dargelegt. Es werden Vorüberlegungen getroffen und eine Komponentenauswahl wird vorgenommen.

3.1 Anforderung

In der Anforderung werden die an den Beschleunigungs-Datenlogger gestellten Eigenschaften näher spezifiziert.

3.1.1 Mikrocontroller

Herzstück des Beschleunigungs-Datenloggers soll ein ARM-Mikrocontroller sein, der in der Programmiersprache C programmiert wird. Dieser soll sich auf einer eigenständigen Platine befinden und eine Kommunikation mit weiteren Komponenten über die Peripherien SSI, I^2C und UART durch Steckleisten ermöglichen.

3.1.2 Beschleunigungssensor & RTC

Für den Beschleunigungssensor soll ein 3-Achsen-Beschleunigungssensor verwendet werden der sich zusammen mit einer Real-Time-Clock auf einer Platine befindet, die zwecks Kombination mit der Mikrocontroller-Platine über eine Steckleiste verbunden wird.
Zu erfassende Daten sind dabei Datum, Uhrzeit und Beschleunigungsänderung in Richtungen der 3 Achsen. Diese sind in geeigneter Form auf einem Speichermedium zu sichern. Das verwendete Speichermedium soll hierbei mindestens für eine Sicherung von 168 h der erfassten Daten ausreichen. Des Weiteren soll das verwendete Speichermedium den Export der Daten zur Auswertung ermöglichen.

3.1.3 Energieversorgung

Die Energieversorgung des Beschleunigungs-Datenloggers soll autonom durch einen Sekundär-Energiespeicher erfolgen, wobei eine minimale Betriebszeit von 8 h gewährleistet sein muss. Bei Abschalten der Betriebsspannung soll das Programm kontrolliert beendet werden.

3.1.4 Erweiterung

Für die Energieversorgung ist als Erweiterung eine Versorgung durch einen Primär-Energiespeicher vorzusehen. Des Weiteren soll die Möglichkeit für weitere Datenerfassungen vorgesehen werden. Explizit ist hierfür hardwareseitig ein GPS-Empfangsmodul im Beschleunigungs-Datenlogger sowie ein Temperatursensor auf der Platine mit dem Beschleunigungssensor für interne Temperaturmessungen und ein Temperatursensor am Gehäuse für externe Temperaturmessungen zu integrieren. Zusätzlich sind Anschlussmöglichkeiten für Erweiterungen von noch nicht spezifizierten Komponenten vorzusehen.

3.1.5 Gehäuse

Um den Beschleunigungs-Datenlogger im Feldversuch und beim späteren Einsatz vor externen Einflüssen zu schützen, soll er über ein solides Gehäuse verfügen. Dieses soll für Vorführzwecke über eine Plexiglasabdeckung verfügen, die wahlweise durch eine robuste Aluminiumabdeckung ausgetauscht werden kann.

3.2 Vorüberlegung

Die aus der Vorüberlegung gewonnenen Informationen sind wichtig, um den Bereich, in dem die Messdaten liegen, theoretisch abzugrenzen und um die benötigten Komponenten und deren Bezug zueinander zu definieren.

3.2.1 Vorausberechnungen

Die auf der Erde herrschende Gravitation übt auf Objekte in der Regel eine Erdbeschleunigung von ca. 9,81 m/s^2 aus. Da die Haftreibung der Räder von Kraftfahrzeugen im Normalfall den Wirkungsgrad 1 nicht überschreitet, ist hierdurch die Kraftübertragung von der Antriebswelle über die Räder auf die Fahrbahn begrenzt. Daraus folgt, dass praktisch selbst unter der Annahme einer Kraftübertragung von 1:1 auf die Fahrbahn, die maximale positive Beschleunigung eines anfahrenden Kraftfahrzeugs 9,81 m/s^2 nicht überschreiten kann. Im Umkehrschluss ist auch eine maximale negative Beschleunigung beim regulären Abbremsen eines Kraftfahrzeugs von weniger als -9,81 m/s^2 nicht möglich.

Bei einem Kraftfahrzeug in der Stadt wird von einer maximalen Geschwindigkeit (v) von etwa 60 km/h ausgegangen, unter Berücksichtigung der Geschwindigkeitsüberschreitung von 10 km/h, bei der in der Regel geltenden 50 km/h.

Hieraus ergibt sich aus der Formel (2.1.3) ein Bremsweg von:

$$v = 60\ \frac{\text{km}}{\text{h}} \approx 16{,}7\ \frac{\text{m}}{\text{s}}$$

$$s_{Bremsweg} = \frac{16{,}7^2\ (\frac{\text{m}}{\text{s}})^2}{2 \cdot 9{,}81\ \frac{\text{m}}{\text{s}^2}} = 14{,}16\ \text{m}$$

Ein Kraftfahrzeug mit einer Masse von 1000 kg hat bei einer Geschwindigkeit von 60 km/h nach der Formel (2.1.4) eine kinetische Energie von:

$$E_{kin} = \frac{1}{2} \cdot 1000\ \text{kg} \cdot 16{,}7^2\ \left(\frac{\text{m}}{\text{s}}\right)^2 \approx 139{,}4\ \text{kJ}$$

Nach dem Umstellen der Formel (2.1.1) besitzt ein Kraftfahrzeug mit einer Masse von 1000 kg zum Zeitpunkt der maximalen Beschleunigung eine Kraft von:

$$F_{Fahrzeug} = 9{,}81\ \frac{\text{m}}{\text{s}^2} \cdot 1000\ \text{kg} = 9810\ \text{N}$$

Die folgende Tabelle 3.1 zeigt eine Auflistung der maximal auftretenden Beschleunigungen, die beim Bremsvorgang von 60 km/h auf 0 km/h unter verschiedenen Witterungsbedingungen theoretisch möglich sind, auf. Hierfür wurden die verschiedenen Beschleunigungen aus einer Tabelle, die vom ADAC für Reaktionswege und Bremswege aufgestellt wurden, verwendet [4]. Des Weiteren wird auf Basis dieser Erkenntnisse der Bremsweg und die hierbei erzeugte Kraft für ein Kraftfahrzeug mit einem Gewicht von 1000 kg errechnet.

Vorausberechnung					
Witterung	Beschleunigung in m/s^2	Fahrzeug-masse in kg	Anfangsgeschwindigkeit in m/s	Bremsweg in m	erzeugte Kraft in N
trocken	8	1000	16,7	17,43	8000
nass	5	1000	16,7	27,89	5000
Schnee	2	1000	16,7	69,72	2000
Eis	1	1000	16,7	139,45	1000

Tabelle 3.1: Vorausberechnung für Beschleunigungen

Hieraus ergibt sich, dass, realistisch betrachtet, auf geraden Fahrbahnen eine maximale Beschleunigung von 8 m/s^2 zu erwarten ist und eine Messdatenerfassung im Bereich von $\pm 9{,}81$ m/s^2 somit ausreichend ist. Um aber die Werte der Z-Achse für Erkenntnisse hinzuzuziehen zu können, auf der in Ruhelage durch die Erdbeschleunigung schon 9.81 m/s^2 wirken, wurde ein mehr als doppelt so großer Messbereich von ± 20 m/s^2 gewählt.

3.2.2 Konzeptentwurf

Der Konzeptentwurf stellt die Beziehung vom Mikrocontroller zu den einzelnen Komponenten und die dabei verwendeten Kommunikationsprotokolle dar.

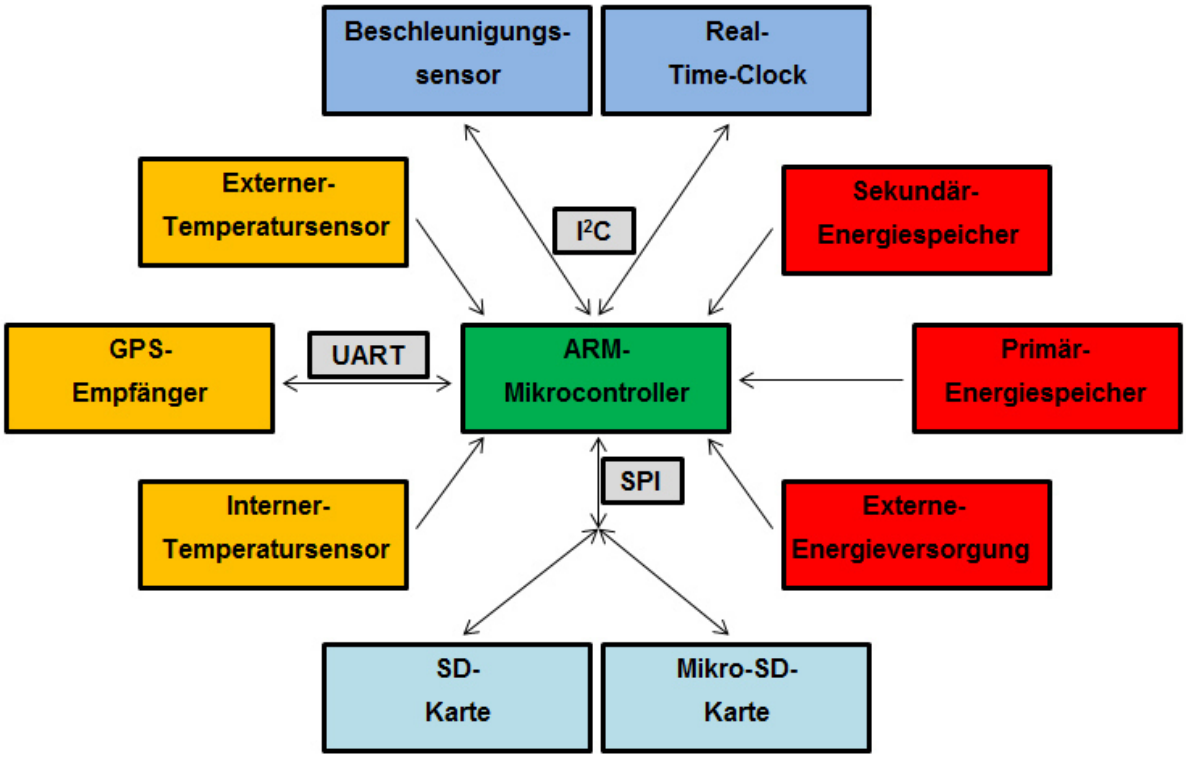


Abbildung 3.1: Konzeptentwurf Beschleunigungs-Datenlogger

3.3 Komponentenauswahl

Für die Entwicklung des Beschleunigungs-Datenloggers wurden die nachfolgenden Komponenten gewählt. Dies erfolgte unter dem Aspekt der Anforderungserfüllung und der kostengünstigsten Variante.

3.3.1 Mikrocontroller

Als Mikrocontroller wurde der Stellaris LM4F120H5QR Mikrocontroller gewählt. Dieser befindet sich auf dem Stellaris LM4F120 LaunchPad Evaluation Kit und gehört zu der ARM-Prozessor-Familie. Mit einem Preis von ca. 14 Euro für das Stellaris LM4F120 LaunchPad Evaluation Kit, handelt es sich hierbei um eine kostengünstige Variante. Durch seine umfangreichen Anwendungsmöglichkeiten bietet der LM4F120H5QR die Möglichkeit, den Beschleunigungs-Datenlogger für andere Messungen und Funktionen zu erweitern. Hierdurch wird der Beschleunigungs-Datenlogger flexibel in seinen Anwendungsmöglichkeiten und dessen Anwendungsbereichen. Das Stellaris LM4F120 LaunchPad Evaluation Kit verfügt über einen 3,3 V Regulator, wodurch ermöglicht wird, Komponenten direkt mit 5 V und 3,3 V Spannung zu versorgen, wobei der maximal zur Verfügung stehende Strom 300 mA beträgt. Der Eigenstrombedarf bei dem verwendeten Stellaris LM4F120 LaunchPad Evaluation Kit liegt bei Normalbetrieb lt. Messung bei ca. 52 mA. Die dreifarbige LED, die sich auf dem Board zu Testzwecken befindet, lässt den Strombedarf im Betrieb auf ca. 53 mA ansteigen. Die Spannungsversorgung darf lt. Herstellerdatenblatt [5] zwischen 4,85 V und 5,25 V schwanken, um einen sicheren Betrieb zu gewährleisten. Als Anschlussmöglichkeiten stehen ein USB 2.0 zur Datenübertragung und zur Energieversorgung sowie vier I^2C, vier SSI, acht UART und ein CAN zur Verfügung, wobei durch die multifunktionalen Pin-Belegungen nicht alle Anschlüsse gleichzeitig realisiert werden können.

Die Abbildung 3.2 auf der folgenden Seite stellt das Stellaris LM4F120 LaunchPad Evaluation Kit dar. Die 40 Steckpins auf der Platinen-Vorderseite verfügen auf der Platinen-Rückseite über Steckerbuchsen.

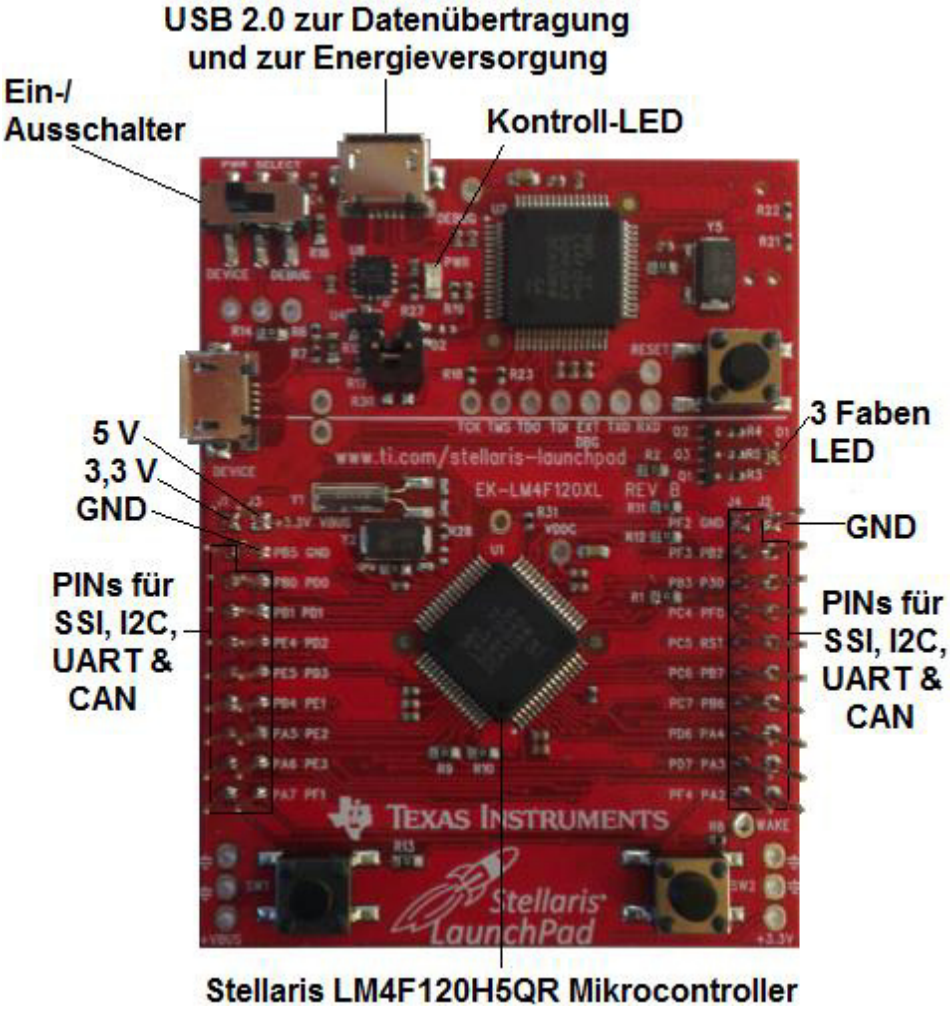


Abbildung 3.2: Stellaris LM4F120 LaunchPad Evaluation Kit

3.3.2 3D-BS Sensor-Modul

Für die Erfassung von Beschleunigungsdaten wurde das 3D-BS Sensor-Modul gewählt. Dieses ermöglicht durch seine kompakte Bauform einen platzsparenden Einbau. Mit einem Preis von ca. 7 Euro handelt es sich hierbei um ein preiswertes und fertig bestücktes Modul, welches aus dem BMA020 Beschleunigungssensor von Bosch Sensortech, einem Linearregler, einer Diode, Mosfet-Transistoren, Widerständen und Kondensatoren besteht.

Durch den Linearregler ist es möglich, das Sensor-Modul mit einer Spannung zwischen 2,5 V und 6 V zu betreiben. Es kann somit flexibel zwischen einer Spannungsversorgung von 3,3 V oder 5 V durch das Stellaris LM4F120 LaunchPad Evaluation Kit gewählt werden. Die Diode dient als Verpolungsschutz, um das Sensor- Modul zu schützen, wodurch eine sichere Handhabung gewährleistet wird. Die Mosfet-Transistoren werden jeweils in Kombination mit zwei 10 kΩ Pull-up-Widerständen als bidirektionale Pegelwandler verwendet, um an den Datenleitungen eine Versorgung mit High-Pegel zu gewährleisten. Eine zusätzliche externe Spannungsversorgung über Pull-up-Widerstände an den Datenleitungen entfällt somit. Die Kondensatoren dienen zur Spannungsglättung, wodurch mögliche Störeinflüsse seitens schwankender Spannung vorgebeugt wird. Der BMA020 ist ein digitaler 3-Achsen-Accelerometer-Sensor, welcher lt. Herstellerdatenblatt [6], je nach Konfigurierung, Beschleunigungsdaten in den Bereichen ±2 g, ±4 g und ±8 g erfassen kann und somit einen flexiblen Anwendungsbereich ermöglicht. Die Messdatenerzeugung ist hierbei zwischen 25 bis 1500 pro Sekunde einstellbar. Die Konfigurierung und Datenauslese kann wahlweise durch I^2C, SPI 3 und SPI 4 erfolgen. Der maximale Strombedarf des 3D-BS Sensor-Moduls liegt lt. Herstellerdatenblatt [7] unter 1 mA.

Abbildung 3.3: 3D-BS Sensor-Modul

3D-BS Sensor-Modul				
Pin	Bezeichnung	I2C Belegung	SPI 3 Belegung	SPI 4 Belegung
1	UIN	UIN	UIN	UIN
2	UENABLE	nicht benutzt	nicht benutzt	nicht benutzt
3	UPULLUP	UIN	nicht benutzt	nicht benutzt
4	INT	INT	INT	INT
5	GND	GND	GND	GND
6	CSB	UIN	CS	CS
7	SCK	SCL	SCK	SCK
8	SDI	SDA	SDA	MOSI
9	SDO	GND	GND	MISO
10	GND	GND	GND	GND

Tabelle 3.2: 3D-BS Sensor-Modul

3.3.3 SD-Karte

Um die erfassten Daten zu speichern und sie für die Auswertung leicht zu exportieren, wurde als Speichermedium die SD-Karte gewählt. Diese wird über SPI konfiguriert und beschrieben. Als Halterung für die SD-Karte wurde ein robustes SD-Kartenhaltermodul-Socket gewählt. Dieses verfügt über einen Linearregler, wodurch eine Spannungsversorgung durch das Stellaris LM4F120 LaunchPad Evaluation Kit wahlweise mit 3,3 V oder 5 V erfolgen kann. Des Weiteren sind die Pins für die Datenleitungen mit 10 kΩ Pull-up-Widerständen an 3,3 V versehen, um eine Versorgung mit High-Pegel zu gewährleisten, wodurch eine externe Spannungsversorgung über Pull-up-Widerstände an die Datenleitung entfällt. Um möglichen, störenden Spannungsschwankungen vorzubeugen, ist das SD-Kartenhaltermodul-Socket zusätzlich mit Kondensatoren zur Glättung ausgerüstet. Der Strombedarf für das SD-Kartenhaltermodul-Socket liegt nach Messung bei ca. 3 mA. Die SD-Karte hat lt. Herstellerdatenblatt [8] einen maximalen Strombedarf von bis zu 75 mA.

Abbildung 3.4: SD-Karten-Socket

Abbildung 3.5: SD-Karte Frontansicht

Abbildung 3.6: SD-Karte Rückansicht

3.3.4 Mikro-SD-Karte

Als Datenspeicheralternative für die SD-Karte wurde eine Mikro-SD-Karte vorgesehen. Als Halterung für die Mikro-SD-Karte befindet sich ein Mikro-SD-Kartenhaltermodul-Socket direkt auf dem 3D-Acceleration-Booster. Hierdurch kann der 3D-Acceleration-Booster mit dem Stellaris LM4F120 LaunchPad Evaluation Kit als eine leichte und platzsparende separate Einheit betreiben werden, wodurch alternative Anwendungsmöglichkeiten für das Erfassen von Beschleunigungsdaten ermöglicht werden. Die Mikro-SD-Karte wird, wie die SD-Karte, über SPI konfiguriert und beschrieben. Die Mikro-SD-Kartenhaltermodul-Socket-Pins für die Datenleitungen sind mit 47 kΩ Pull-up-Widerständen an 3,3 V separat auf dem 3D-Acceleration-Booster versehen, um eine Versorgung mit High-Pegel zu gewährleisten. Die Versorgungsspannung der Mikro-SD-Karte von 3,3 V kann durch den 3,3 V Ausgang des Stellaris LM4F120 LaunchPad Evaluation Kit realisiert werden. Je nach Kartenherstellertyp beträgt der maximale Strombedarf bis zu 65 mA.

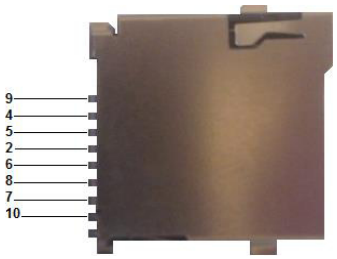


Abbildung 3.7: Mikro-SD-Socket

Abbildung 3.8: Mikro-SD-Karte

Abbildung 3.9: Mikro-SD-Karte Rückseite

SD-Karten-Socket und Mikro-SD-Karten-Socket				
Pin	Bezeichnung	SPI Belegung	SD-Karte	Mikro-SD-Karte
1	GND	nicht benutzt	GND	nicht benutzt
2	3,3 V	3,3 V	3,3 V	3,3 V
3	5 V	nicht benutzt	nicht benutzt	nicht benutzt
4	CS	Fss	CS	CS
5	MOSI	TX	Data In	Data In
6	SCK	CLK	Clock	Clock
7	MISO	RX	Data Out	Data Out
8	GND	GND	GND	GND
9	nicht benutzt	nicht benutzt	reserviert	reserviert
10	nicht benutzt	nicht benutzt	reserviert	reserviert

Tabelle 3.3: SD-Karten und Mikro-SD-Karten-Socket

3.3.5 Digital-Halbleitertemperatursensor

Um die Temperatur intern und extern vom Beschleunigungs-Datenlogger zu erfassen, wurden zwei TSIC 206 Digital-Halbleitertemperatursensoren gewählt. Diese benötigten jeweils eine Spannungsversorgung zwischen 2,95 V und 5,5 V, wodurch flexibel zwischen einer Spannungsversorgung von 3,3 V oder 5 V durch das Stellaris LM4F120 LaunchPad Evaluation Kit gewählt werden kann. Der Messbereich liegt lt. Herstellerdatenblatt [9] von -50 °C bis +150 °C, wobei eine Genauigkeit von ±0,5 °C erreicht wird. Um mögliche störende Spannungsschwankungen zu vermeiden, ist für den internen Digital-Halbleitertemperatursensor ein 100 nF Kondensator auf den 3D-Acceleration-Booster vorgesehen. Für den externen Digital-Halbleitertemperatursensor befindet sich ein 100 nF Kondensator in direkter Nähe vom Sensor.

3.3.6 Real-Time-Clock

Für die Echtzeiterfassung wurde die DS1307 Real-Time-Clock gewählt. Diese befindet sich auf dem 3D-Acceleration-Booster und benötigt lt. Herstellerdatenblatt [10] eine Versorgungsspannung von 5 V. Es besteht die Möglichkeit, dies durch den 5 V Ausgang des Stellaris LM4F120 LaunchPad Evaluation Kit zu realisieren. Die Konfigurierung und Auslesung erfolgt über I^2C. Die Pins für die Datenleitungen sind über 4,7 kΩ Pull-up-Widerstände an 3,3 V separat auf dem 3D-Acceleration-Booster versehen, um eine Versorgung mit High-Pegel zu gewährleisten. Um den Verlust der Uhrzeit und des Datums beim Ausschalten des Beschleunigungs-Datenloggers zu verhindern, wird die Real-Time-Clock mit einer Haltespannung, durch eine 3 V Lithium-Metall Knopfzelle mit einer Kapazität von 280 mAh versorgt. Diese befindet sich mit einer Halterung auf dem 3D-Acceleration-Booster. Die Kapazität von 280 mAh reicht aus, um einen Datenverlust für ca. 50 Jahre zu verhindern. Als Uhrenquarz wurde der CC5V-T1A gewählt, der lt. Herstellerdatenblatt [11] mit 32,768 kHz schwingt. Dieser befindet sich neben der RTC auf dem 3D-Acceleration-Booster.

3.3.7 GPS-Empfangsmodul

Als GPS-Empfangsmodul wurde das EM- 406a GPS-Engine-Board gewählt. Dieses verfügt über eine integrierte Antenne und ist in der Lage, lt. Herstellerdatenblatt [12], bis zu 20 Kanäle parallel zu verarbeiten. Mit ca. 30 Euro ist es eine der kostengünstigsten Variante für GPS-Empfangsmodule und kann durch seine kompakte, kleine Bauform platzsparend in den Beschleunigung-Datenlogger integriert werden. Seine Versorgungsspannung liegt zwischen 4,5 V und 6,5 V mit einem maximalen Strombedarf von 60 mA. Er kann somit durch den 5 V Ausgang des Stellaris LM4F120 LaunchPad Evaluation Kit betrieben werden.

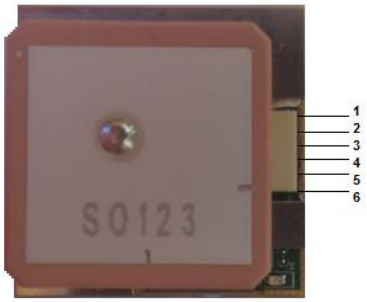


Abbildung 3.10: GPS-Empfangsmodul

GPS-Empfangsmodul		
Pin	Bezeichnung	UART Belegung
1	GND	GND
2	VCC	5V
3	RX	TX
4	TX	RX
5	GND	GND
6	PPS	nicht benutzt

Tabelle 3.4: GPS-Empfangsmodul

3.4 Komponenten-Energiebedarf

Der Energiebedarf der Booster-Platine wurde anhand der Widerstandsgrundbestückung durch Umstellen der Formel (2.1.5) rechnerisch ermittelt:

$$I_{Booster} = 4 \cdot \left(\frac{3,3\ \text{V}}{4,7\ \text{k}\Omega}\right) + 5 \cdot \left(\frac{3,3\ \text{V}}{47\ \text{k}\Omega}\right) \approx 3\ \text{mA}$$

Für die restlich verwendeten Komponenten wurde der Energiebedarf anhand der Herstellerdatenblätter oder bei fehlenden Angaben durch Messungen ermittelt. Die folgende Tabelle 3.5 stellt den Komponenten-Energiebedarf und den Gesamtverbrauch dar.

Komponenten Energiebedarf				
Komponenten	Umax	Umin	U verwendet	Imax Verbrauch
Stellaris LM4F120 Platine	5,25 V	4,75 V	5 V	53 mA
3D-BS Sensor-Modul	6 V	2,5 V	3,3 V	>1 mA
Real-Time-Clock	5,5 V	4,5 V	5 V	>1 mA
Micro SDHC-Karte 8G	3,6 V	2,7 V	3,3 V	60,0 mA
SD-Karten-Socket	5,5 V	2,7 V	3,3 V	3 mA
SDHC-Karte 8G	3,6 V	2,7 V	3,3 V	75,0 mA
TSIC 1	5,5 V	2,97 V	3,3 V	>1 mA
TSIC 2	5,5 V	2,97 V	3,3 V	>1 mA
Booster Platine	6,5 V	4,5 V	5 V	1 mA
GPS	6,5 V	4,5 V	5 V	70,0 mA
gesamt	X	X	X	206,0 mA

Tabelle 3.5: Komponenten-Energiebedarf

Für den Gesamtverbrauch wurde mit dem Stromverbrauch der SD-Karte gerechnet, da immer nur eine der Karten z.Zt. verwendet werden kann und diese den höheren Stromverbrauch hat. Der elektrische Energiebedarf des Beschleunigungs-Datenloggers liegt nach der Formel (2.1.6) für 1 h bei:

$$E_{elek} = \frac{5\ \text{V} \cdot 208\ \text{mA} \cdot 1\ \text{h} \cdot 1\ \text{A}}{1000\ \text{mA}} = 1,04\ \text{Wh}$$

3.5 Energieversorgung

Um den Beschleunigungs-Datenlogger mit der nötigen elektrischen Energie zu versorgen, wurden, um eine möglichst hohe Flexibilität zu erreichen, 3 Varianten wie folgt gewählt:

Variante 1: Energieversorgung durch einen Sekundär-Energiespeicher.

Variante 2: Energieversorgung durch einen Primär-Energiespeicher.

Variante 3: Energieversorgung durch eine externe Energieversorgung.

Jede dieser Varianten kann nur separat und nicht gleichzeitig eingesetzt werden.

3.5.1 Sekundär-Energiespeicher

Bei dem Sekundär-Energiespeicher handelt es sich um einen VTB-28 Mobil-Akku. Dieser kann über alle USB Schnittstellen, die über 5 V DC verfügen, geladen werden. Eine Verwendung als Energielieferant während des Ladezyklus ist nicht möglich, weshalb er nur als Hauptenergieversorger und nicht als Energiepuffer in Kombination mit den anderen Varianten verwendet werden kann. Um Bauteile mit Energie zu versorgen, verfügt der VTB-28 über 2 USB-Anschlussstellen.

Abbildung 3.11: VTB-28 Frontansicht

Der linke bei Frontansicht stellt hierbei 5 V DC und 1000 mA, der rechte 5 V DC und 2000 mA zur Verfügung. Der mittlere Mikro-USB Anschluss wird zum Laden des Akkus verwendet. Er besitzt lt. Herstellerdatenblatt [13] eine Akkukapazität von 10000 mAh, womit er eine Ausgangskapazität von 7400 mAh ermöglicht.

Somit stellt er nach der Formel (2.1.6) eine elektrische Energie zur Verfügung von:

$$E_{elek} = 5\ \mathrm{V} \cdot 7,4\ \mathrm{A} \cdot 1\ \mathrm{h} = 37\ \mathrm{Wh}$$

Er ermöglicht somit einen sicheren Betrieb des Beschleunigungs-Datenloggers von:

$$B_{Zeit} = \frac{37\ \mathrm{Wh}}{1,04\ \mathrm{Wh}} \cdot 1\ \mathrm{h} \approx 35,5\ \mathrm{h}$$

3.5.2 Primär-Energiespeicher

Als Primär-Energiespeicher dienen vier AA-Mignon-Lithium-Metall Batterien, die in Reihe geschaltet, eine Spannung von 6,3 V und eine Kapazität von 3000 mAh zur Verfügung stellen. Durch einen Präzisionsspannungsregler werden die 6,3 V auf 5 V ab geregelt. Somit stellt er nach der Formel (2.1.6) eine elektrische Energie zur Verfügung von:

$$E_{elek} = 5\ \mathrm{V} \cdot 3\ \mathrm{A} \cdot 1\ \mathrm{h} = 15\ \mathrm{Wh}$$

Er ermöglicht somit einen sicheren Betrieb des Beschleunigungs-Datenloggers von:

$$B_{Zeit} = \frac{15\ \mathrm{Wh}}{1,04\ \mathrm{Wh}} \cdot 1\ \mathrm{h} \approx 14,4\ \mathrm{h}$$

3.5.3 Externe Energieversorgung

Für eine externe Energieversorgung eignet sich eine Energiequelle, die eine Spannung zwischen 6 V und 20 V DC hat und die einen minimalen Permanentstrom von 208 mA nach der Abregelung auf 5 V zur Verfügung stellen kann.

3.6 Datenübertragungsmenge

In der nachfolgenden Tabelle 3.6 sind die Komponenten und ihre Datenübertragungsmenge aufgeführt.

Datenübertagungsmenge				
Komponenten	Takt	Data	Senden pro sec.	pro Stunde
3D-BS Sensor-Modul	1,5 kHz	10 Bit	1875 Byte	6,75 MB
Real-Time-Clock	X	X	64 Byte	30,4 kB
TSIC 1	10 Hz	11 Bit	13,75 Byte	49,5 kB
TSIC 2	10 Hz	11 Bit	13,75 Byte	49,5 kB
GPS	X	10 Bit	600 Byte	2,16 MB
maximale Datenmenge	X	X	2566,5 Byte	9,24 MB

Tabelle 3.6: Datenübertragungsmenge

Aus der Tabelle 3.6 ergibt sich eine maximale Datenerfassung für 168 h von:

$$\frac{9,24\ \text{MB} \cdot 168\ \text{h}}{1\ \text{h}} \approx 1.55\ \text{GB}$$

Eine 2 GB SD-Speicherkarte oder Mikro-SD-Speicherkarte ist somit ausreichend. Da Speicherkarten im Bereich von 2 GB bis 8 GB preislich nahe beieinander liegen, etwa bei ca. 6 Euro, wurden für den SD-Kartenhalter-Socket und den Mikro-SD-Kartenhalter-Socket jeweils Speicherkarten mit 8 GB gewählt.

Kapitel 4

4 Hardwareentwicklung

In diesem Kapitel wird die Hardwareentwicklung beschrieben. Der Beschleunigungs-Datenlogger setzt sich aus mehreren Einzelkomponenten und zwei Komponentenpaketen zusammen. Bei den Komponentenpaketen handelt es sich um den 3D-Acceleration-Booster und den Capacitor-Tower. Diese werden im Verlauf des Kapitels vorgestellt und der Beschleunigungs-Datenlogger, der aus der Komponentenzusammenfügung resultiert, wird erläutert.

4.1 3D-Acceleration-Booster

Die Platine des 3D-Acceleration-Boosters wurde in EAGLE 6.5.0 Light, einem frei verwendbaren Layout-Editor, entworfen und designt. Um dies zu realisieren, wurden unter EAGLE 6.5.0 Light für das 3D-BS Sensor- Modul, die DS1307 Real-Time-Clock, die Knopfzellenhalterung, der Mikro-SD-Kartenhaltermodul-Socket, die Header-Pin-Leisten, die Platinensteckverbinder und für den TSIC 206 Digital-Halbleitertemperatursensor, Komponentenbibliotheken angelegt. Die Komponentenmaße wurden hierfür aus den Herstellerdatenblättern oder durch Ausmessen mittels einer Schieblehre ermittelt. Ein Selbstätzen der Platine wurde nicht durchgeführt, da hierbei eine hohe Qualität der Platine nicht gewährleistet werden konnte. Die Platinenherstellung wurde somit an PCB-POOL abgegeben, die ca. zwei Wochen für Platinenanfertigung und Versand benötigen.

Auf den folgenden drei Seiten befindet sich die unter EAGLE 6.5.0 Light entworfene Version 1.0 des 3D-Acceleration-Booster-Schaltplans und das darauf basierende Platinen-Layout für die Platinenunterseite und für die Platinenoberseite.

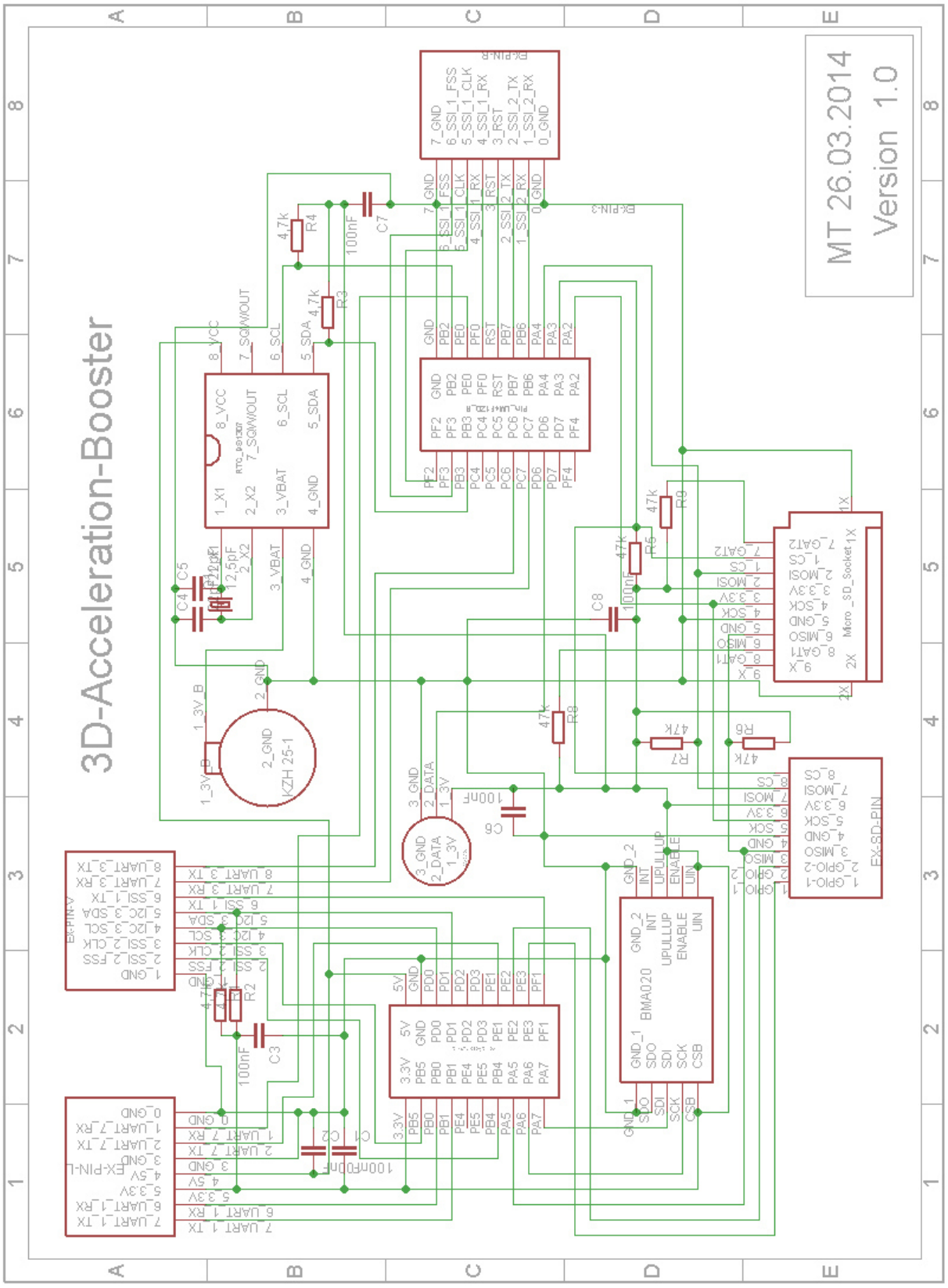


Abbildung 4.1: Schaltplan 3D-Acceleration-Booster

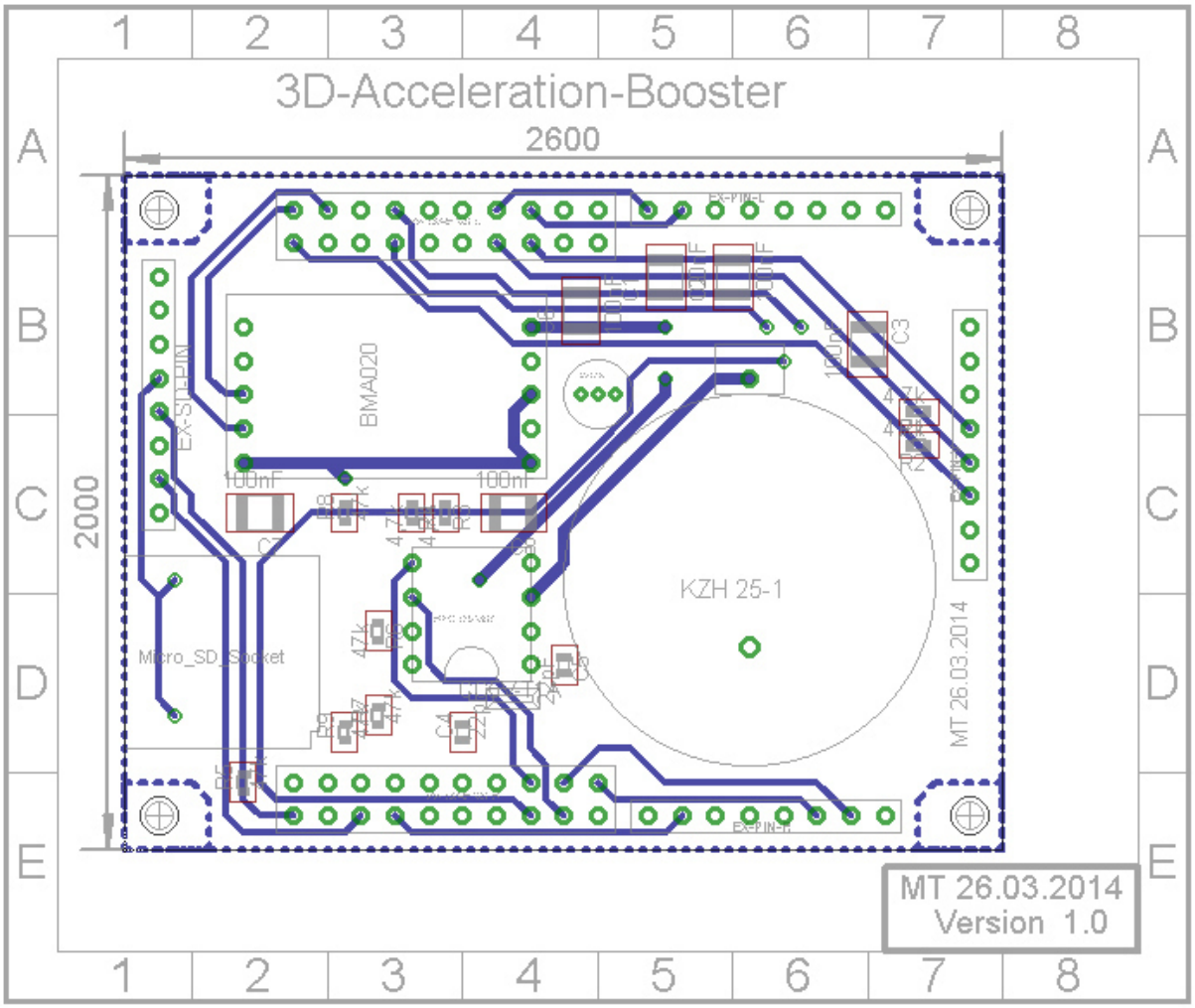


Abbildung 4.2: 3D-Acceleration-Booster Platinen-Layout Unterseite

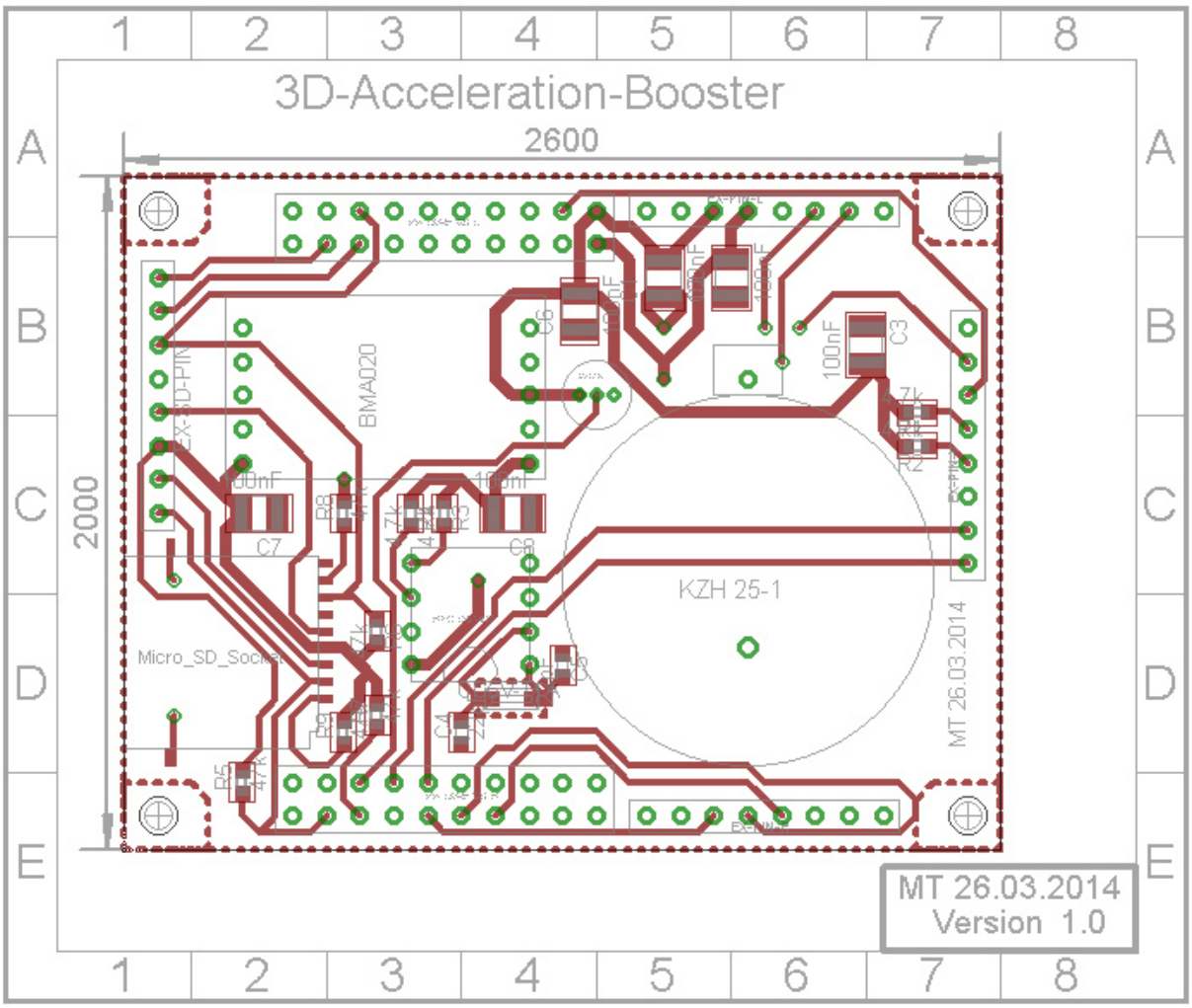


Abbildung 4.3: 3D-Acceleration-Booster Platinen-Layout Oberseite

Der 3D-Acceleration-Booster verfügt jeweils an den beiden Längsseiten über eine Header-Pin-Leiste mit 20 Pins, wodurch er an der Unterseite des Stellaris LM4F120 LaunchPad Evaluation Kit gesteckt werden kann. An allen vier Seiten der Platine befinden sich rechtwinklige Platinensteckverbindungen mit jeweils 8 Pins, an denen externe Komponenten angeschlossen werden können. Die untere abgewinkelte Steckleiste dient hierfür zum Anschließen des externen SD-Socket, der sich auf einer weiteren Platine befindet. Diese Pins sind parallel zu den Anschlusspins des Mikro-SD-Socket angeschlossen, das sich parallel zu der Steckleiste befindet. An der linken abgewinkelten Steckleiste befinden sich die Anschlussmöglichkeiten für das externen GPS-Empfangsmodul und den externen Temperatursensor. An den oben und rechts angeordneten, abgewinkelten Steckleisten besteht die Anschlussmöglichkeit für noch nicht spezifizierte externe Komponenten, welche wahlweise eine I^2C, SPI oder UART Schnittstelle benötigen. Die Datenleitungen für das I^2C Protokoll sind hierbei mit zwei 4,7 kΩ Pull-up-Widerständen versehen. Zwischen der linken und rechten abgewinkelten Steckleiste befindet sich eine Halterung für den Energiespeicher der RTC, der mit einer 3 V Lithium-Metall-Knopfzelle versehen ist. Hinter dem Mikro-SD-Socket befindet sich die RTC mit ihrem Uhrenquarz. Das 3D-BS Sensor-Modul befindet sich hinter der unten angeordneten abgewinkelten Steckleiste. Zwischen dem 3D-BS Sensor-Modul und der Halterung für den Energiespeicher der RTC befindet sich der TSIC für die Erfassung der internen Temperatur des Beschleunigungs-Datenloggers. Des Weiteren befinden sich auf der Platine fünf 47 kΩ Pull-up-Widerstände für die Datenleitung des Mikro-SD-Sockets, zwei 4,7 kΩ Pull-up-Widerstände für die Datenleitung der RTC sowie sechs 100 nF Kondensatoren zur Glättung der Spannung.

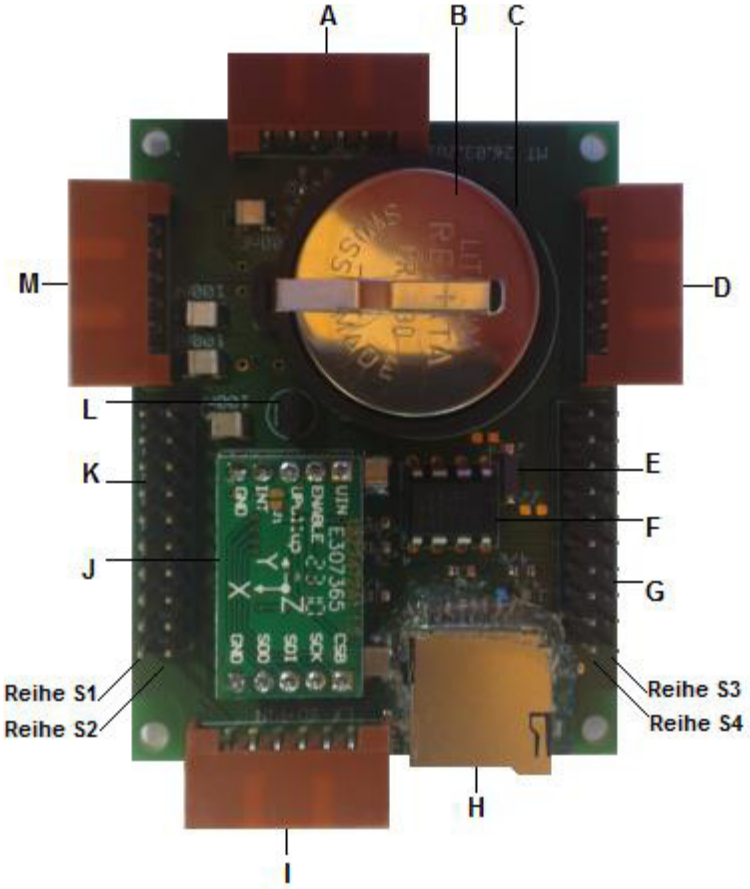


Abbildung 4.4: 3D-Acceleration-Booster

3D-Acceleration-Booster Beschriftung	
A = obere abgewinkelte Steckleiste	H = Mikro-SD-Socket
B = 3V Lithium-Metall Knopfzelle	I = untere abgewinkelte Steckleiste
C = Halterung für RTC Energiespeicher	J = 3D-BS Sensor-Modul
D = rechte abgewinkelte Steckleiste	K = Header-Pin-Leiste links
E = Uhrenquarz	L = TSIC
F = RTC	M = linke abgewinkelte Steckleiste
G= Header-Pin-Leiste rechts	

Tabelle 4.1: 3D-Acceleration-Booster Beschriftung

Header-Pin-Leiste links (K) mit Komponentenbezug											
Stellaris LM4F120H5QR LaunchPad Evalution Kit Pins				A = obere Steckerleiste	D = rechte Steckerleiste	F = RTC	H = Mikro-SD-Socket	I = untere Steckerleiste	J = 3D-BS Sensor-Modul	L = TSIC	M = linke Steckerleiste
1	**S1**	3.3V	max. 300mA				**X**	**SD**	**X**	**X**	**TSIC**
	S2	5V	max. 300mA			**X**					**GPS**
2	**S1**	PB5	SSI 2 Fss	**X**							
	S2	GND		**X**	**X**	**X**	**X**	**SD**	**X**	**X**	**X**
3	**S1**	PB0	UART 1 RX								**GPS**
	S2	PD0	I2C 3 SCL / SSI 3 CLK	**X**							
4	**S1**	PB1	UART 1 TX								**GPS**
	S2	PD1	I2C 3 SDA / SSI 3 FSS	**X**							
5	**S1**	PE4	I2C 2 SCL								
	S2	PD2	SSI 1 und 3 RX								
6	**S1**	PE5	I2C 2 SDA								
	S2	PD3	SSI 1 und 3 TX								
7	**S1**	PB4	SSI 2 CLK	**X**							
	S2	PE1	UART 7 TX								**X**
8	**S1**	PA5	SSI 0 TX				**X**	**SD**			
	S2	PE2	GPIO (AIN 1)					**X**			
9	**S1**	PA6	I2C 1 SCL						**X**		
	S2	PE3	GPIO (AIN 0)					**X**			
10	**S1**	PA7	I2C 1 SDA						**X**		
	S2	PF1	SSI 1 TX	**X**							

Tabelle 4.2: Header-Pin-Leiste links mit Komponentenbezug

Header-Pin-Leiste rechts (G) mit Komponentenbezug											
Stellaris LM4F120H5QR LaunchPad Evaluation Kit Pins				A = obere Steckerleiste	D = rechte Steckerleiste	F = RTC	H = Mikro-SD-Socket	I = untere Steckerleiste	J = 3D-BS Sensor-Modul	L = TSIC	M = linke Steckerleiste
11	**S3**	GND		**X**	**X**	**X**	**X**	**SD**	**X**	**X**	**X**
	S4	PF2	SSI 1 CLK		**X**						
12	**S3**	PB2	I2C 0 SCL			**X**					
	S4	PF3	SSI 1 Fss		**X**						
13	**S3**	PE0	UART 7 RX								**TSIC**
	S4	PB3	I2C 0 SDA			**X**					
14	**S3**	PF0	SSI 1 RX		**X**						
	S4	PC4	UART 1 RX								
15	**S3**	RST			**X**						
	S4	PC5	UART 1 TX								
16	**S3**	PB7	SSI 2 TX		**X**						
	S4	PC6	UART 3 RX	**X**							
17	**S3**	PB 6	SSI 2 RX		**X**						
	S4	PC7	UART 3 TX	**X**							
18	**S3**	PA4	SSI 0 RX				**X**	**SD**			
	S4	PD6	UART 2 RX							**X**	
19	**S3**	PA3	SSI 0 Fss				**X**	**SD**			
	S4	PD7	UART 2 TX								
20	**S3**	PA2	SSI 0 CLK				**X**	**SD**			
	S4	PF4	GPIO								

Tabelle 4.3: Header-Pin-Leiste rechts mit Komponentenbezug

Anmerkung:

Durch eine vertauschte Datenleiterbahn auf der 3D-Acceleration-Booster-Platine für den Mikro-SD-Kartenhalter-Socket, die erst bei der Inbetriebnahme des Beschleunigungs-Datenloggers entdeckt wurde, kann bei der in der Arbeit verwendeten 3D-Acceleration-Booster Version 1.0 das Mikro-SD-Kartenhalter-Socket nicht verwendet werden.

Im Anhang befinden sich ein Schaltplan und ein Platinen-Layout der Version 1.1 bei dem dieser Fehler behoben wurde. Die EAGLE-Dateien hierfür befinden sich auf der Zusatz-DVD im Ordner (Eagle/ 3D-Acceleration-Booster V.1.1).

4.2 Capacitor-Tower

Der Capacitor-Tower setzt sich aus drei übereinander angeordneten Lochrasterplatinen zusammen. Auf den unteren zwei Platinen sind jeweils Kondensatoren angeordnet, die eine Kapazität von 1 F haben. Diese sind lt. Herstellerdatenplatt [14] für eine maximale Spannung von 5,5 V ausgelegt und sind parallel zur 5 V Versorgungsleitung des Stellaris LM4F120 LaunchPad Evaluation Kit angeschlossen. Beim Aktivieren von einem der drei Energieversorgungsvarianten laden sich diese auf und dienen als Pufferspeicher. Nach der umgestellten Formel (2.7.1), hat der Capacitor-Tower eine Ladung von:

$$Q_{C.Tower} = 2 \cdot 1\ \mathrm{F} \cdot 5\ \mathrm{V} = 10\ \mathrm{As}$$

Der folgende Schaltplan wurde für den Capacitor-Tower unter EAGLE 6.5.0 Light entworfen.

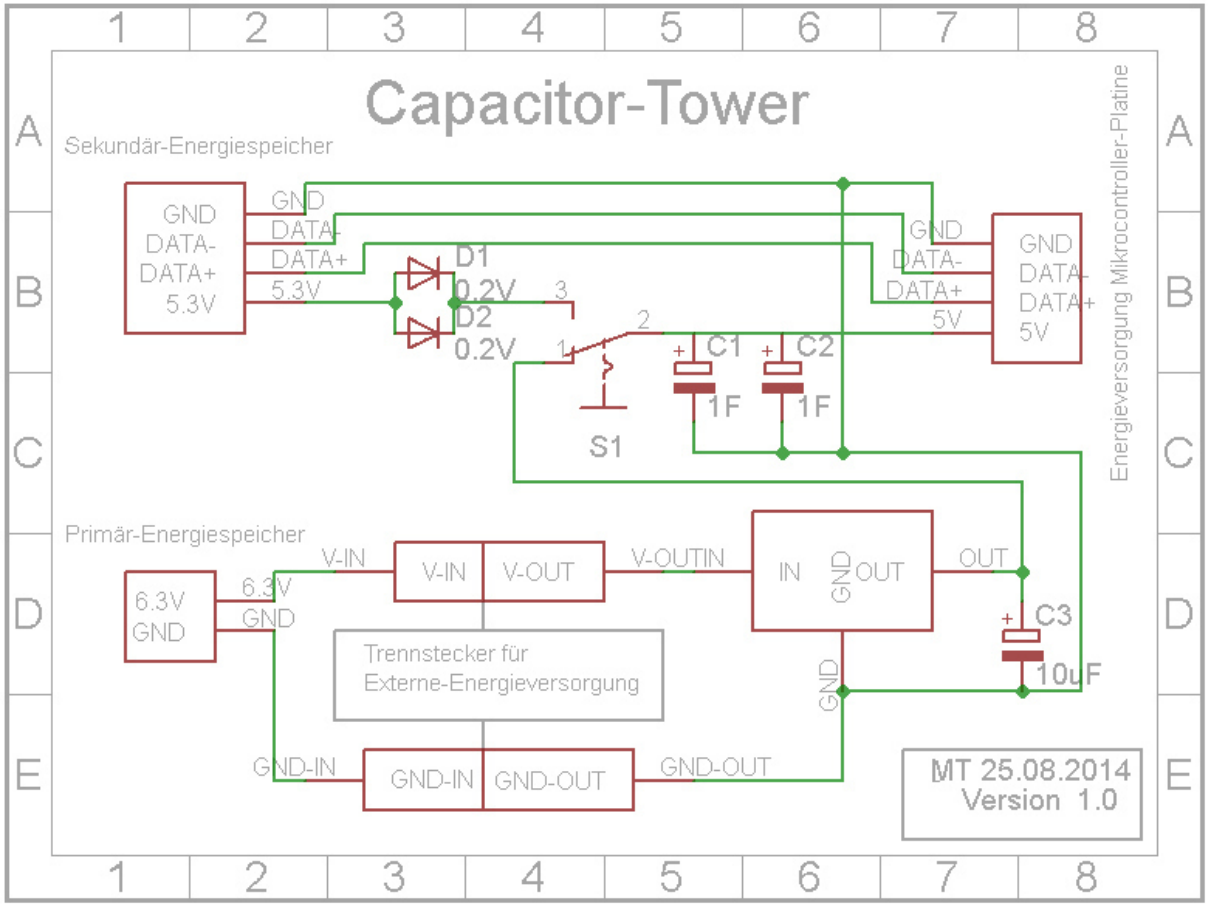


Abbildung 4.5: Schaltplan Capacitor-Tower

Auf der obersten Platine des Capacitor-Towers befindet sich ein Zweistufenschiebeschalter. Dieser ermöglicht es, zwischen dem Primär-Energiespeicher und dem Sekundär-Energiespeicher bzw. beim abgetrennten Primär-Energiespeicher und Verwendung der alternativ angeschlossenen externen Energieversorgung, zwischen zwei Energieversorgungsvarianten umzuschalten. Des Weiteren wird hierdurch durch Nichtbestückung der Primär-Energiespeicherhalterung und nicht angeschlossener externer Energieversorgung oder durch die Deaktivierung des Sekundär-Energiespeichers, ein sicheres Abschalten des Beschleunigungs-Datenloggers realisiert. Der Capacitor-Tower dient hierfür als Pufferspeicher, um beim Ausschalten des Beschleunigungs-Datenloggers oder beim Einbruch der Energieversorgung einen Black Out zu vermeiden, der undefinierte Zustände zur Folge haben kann und aus dem ein Datenverlust resultieren kann. Zum Schutz gegen eine Rückspeisung durch den Pufferspeicher in den Sekundär-Energiespeicher bei dessen Deaktivierung, befinden sich vor dem Schalter zwei parallel zueinander liegende Gleichrichter-Dioden, die eine Durchlassspannung von 0,2 V haben und lt. Herstellerdatenblatt [15] für einen Maximalstrom jeweils von 200 mA bestimmt sind. Ein Rückspeisen der Kondensatoren in den Primär-Energiespeicher oder in eine externe Energieversorgung wird durch den Präzisionsspannungsregler verhindert, der sich in der Energieversorgungsleitung befindet.

4.3 Beschleunigungs-Datenlogger

Der Beschleunigungs-Datenlogger (Abb.4.7- S.38) verfügt über ein robustes quaderförmiges Aluminiumgehäuse, welches mit einer hellgrauen, nicht leitenden Schutzbeschichtung lackiert ist. Er hat die Maße 200/200/72 Millimeter und kann wahlweise mit einer vier Millimeter dicken Plexiglasscheibe oder durch seinen Aluminiumdeckel abgedeckt werden. Für eine Unterteilung wurden sechs Rechteckaluminiumprofile im Innenraum des Beschleunigungs-Datenloggers angeordnet. Diese dienen gleichzeitig als Befestigung für die Halterung der Primär-Energiespeicher und dem GPS-Empfangsmodul sowie als Kabelkanal für die Leitung des Sekundär-Energiespeichers. Der Sekundär-Energiespeicher befindet sich, eingefasst von vier der Rechteckaluminiumprofile, unter einer Plexiglasabdeckung, die mit sechs Inbusschrauben versehen ist. Hierdurch ist der Sekundär-Energiespeicher fixiert und räumlich von den restlichen Komponenten getrennt. Für den Primär-Energiespeicher befinden sich auf der gegenüberliegenden Seite zwei Halterungen, die an dem Rechteckaluminiumprofil geschraubt sind und mit jeweils zwei AA-Mignon Zellen bestückt sind. Durch den geringen Platz empfiehlt es sich, beim Austausch der Zellen, das Rechteckaluminiumprofil, durch Entfernen der zwei Halteschrauben auf der Unterseite des Beschleunigungs-Datenloggers, das Rechteckaluminiumprofil nach oben hin, für eine Entnahme und das Bestücken der Halterung mit AA-Mignon Zellen, zu entfernen. Die Verbindungsleitung des Primär-Energiespeichers besitzt zwei Verbindungsstecker, die mit Schutzhülsen abgedeckt sind. Durch Trennung der zwei Leitungen an den Verbindungssteckern kann hierdurch der Primär-Energiespeicher schnell und kompakt mit seiner Halterung aus dem Beschleunigungs-Datenlogger entfernt werden oder eine alternative externe Energieversorgung zwischen 6 V und 20 V DC mit einem minimalen Permanentstrom von 208 mA an dieser Stelle abgeschlossen werden. Hierfür befindet sich in der Versorgungsleitung zum Capacitor-Tower ein Präzisionsspannungsregler, der lt. Herstellerdatenblatt [16] eine Spannung zwischen 6 V und 20 V auf 5 V umwandelt. Zu berücksichtigen ist hierbei, dass, je höher die zu wandelnde Spannung ist, desto mehr in Wärme umgewandelt wird. Um die Wärme optimal vom Präzisionsspannungsregler abzuführen, ist dieser direkt mit dem Bodengehäuse des Beschleunigungs-Datenloggers verbunden und halb unter einem Rechteckaluminiumprofil integriert. Um eine möglichst störungsfreie 5 V Spannung zu erzeugen, befindet sich direkt an dem Präzisionsspannungsregler ein 10 µF Kondensator, um die Spannung zu glätten. Für eine sofortige volle Funktionsfähigkeit des Beschleunigungs-Datenloggers muss der Capacitor-Tower geladen sein. Dies kann durch Abschalten durch den Ausschalter des Stellaris LM4F120 LaunchPad Evaluation Kit erfolgen, bis dieser geladen ist. Bei der Verwendung des Sekundär-Energiespeichers gilt hierbei, dass,

bei geladenem Capacitor-Tower, das Blinken der blauen Sekundär-Energiespeicher-Kontroll-LED in Dauerleuchten übergeht. Beim Aktivieren des Stellaris LM4F120 LaunchPad Evaluation Kit, fängt dieses wieder an zu blinken solange, der Beschleunigungs-Datenlogger in Betrieb ist.

Das folgende Oszilloskop-Bild stellt den Spannungsverlauf der 3,3 V und der 5 V Versorgungspannung vom Stellaris LM4F120 LaunchPad Evaluation Kit für die Komponenten beim Einschalten des Beschleunigungs-Datenlogger mit geladenem Capacitor-Tower und beim Ausschalten dar.

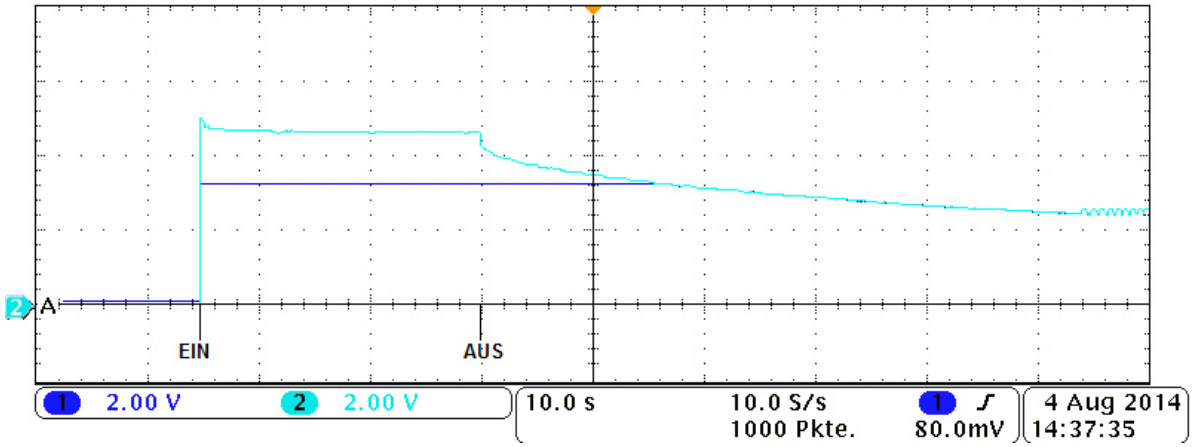


Abbildung 4.6: Beschleunigungs-Datenlogger Spannungsverlauf

Aus der Messung ergibt sich, dass der Capacitor-Tower noch ca. 15 s nach dem Ausschalten des Beschleunigungs-Datenloggers die 3,3 V an dem Versorgungsausgang aufrechterhalten kann, wodurch ein sicheres Speichern ermöglicht wird.

Unter dem Stellaris LM4F120 LaunchPad Evaluation Kit befindet sich der 3D-Acceleration-Booster. Dieser ist in Kombination mit der sich unter ihm befindenden Platine, auf der sich das SD-Kartenhalter-Socket befindet, durch vier Schrauben, fest mit dem Gehäuse des Beschleunigungs-Datenloggers verschraubt.

Durch die Parallelverdrahtung zwischen Mikro-SD-Kartenhalter-Socket und der Anschlussbuchse des externen SD-Kartenhalter-Sockets, kann hierbei immer nur eine Speichervariante z.Zt. verwendet werden. Um einen einwandfreien Betrieb zu gewährleisten, empfiehlt es sich, einen der beiden Sockets immer freizulassen.

Die nachfolgende Abbildung 4.7 stellt den Beschleunigungs-Datenlogger ohne Abdeckung dar.

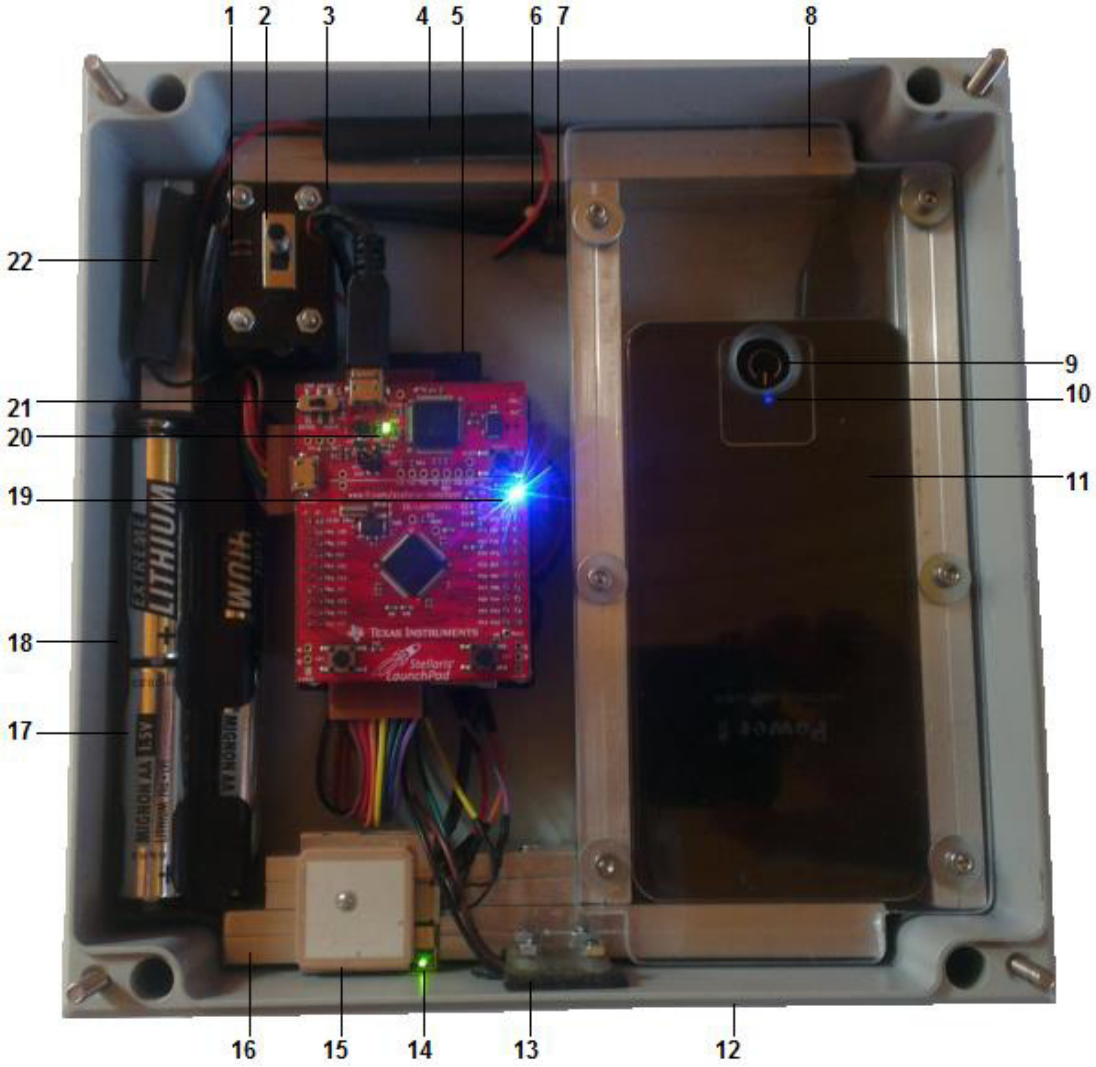


Abbildung 4.7: Beschleunigungs-Datenlogger

Beschleunigungs-Datenlogger	
Nummer	**Bezeichnung**
1	Gleichrichter-Dioden
2	Ein-/Ausschalter vom Beschleunigungs-Datenlogger
3	Capacitor-Tower
4	Plusverbindung mit Trennvorrichtung für Primär-Energiespeicher
5	SD-Karte
6	10 µF Glättungskondensator
7	5 V Präzisionsspannungsregler
8	Plexiglasabdeckung für den Sekundär-Energiespeicher
9	Einschalter des Sekundär-Energiespeichers
10	Kontroll-LED des Sekundär-Energiespeichers
11	Sekundär-Energiespeicher
12	Beschleunigungs-Datenlogger-Gehäuse
13	externer Temperatursensor
14	GPS-Kontroll-LED
15	GPS-Empfänger
16	Aluminiumschienen
17	Primär-Energiespeicher
18	Halterung für Primär-Energiespeicher
19	Kontroll-LED vom Beschleunigungs-Datenlogger
20	Kontroll-LED vom LM4f120
21	Ein-/Ausschalter vom LM4f120
22	Masseverbindung mit Trennvorrichtung für Primär-Energiespeicher

Tabelle 4.4: Beschleunigungs-Datenlogger

Nach dem Einschalten des Beschleunigungs-Datenloggers, leuchtet die Kontroll-LED des Beschleunigungs-Datenloggers (Abb.4.7 Nr.19 S.38) beim Aktivierten des Stellaris LM4F120 LaunchPad Evaluation Kit kurz rot, dann blau und dann grün auf und geht anschließend bei vollständigem Betrieb permanent in Blau über. Die Kontroll-LED des GPS-Empfangsmoduls (Abb.4.7 Nr.14 S.38) blinkt durchgehend grün, solange er keine Satelliten gefunden hat. Bei gefundenen Satelliten ist die Kontroll-LED permanent grün an. Die Kontroll-LED des Sekundär-Energiespeichers (Abb.4.7 Nr.10 S.38) blinkt so lange kontinuierlich blau, so lange Strom verbraucht wird. Beim Ausschalten des Beschleunigungs-Datenloggers erlischt die blaue Kontroll-LED des Beschleunigungs-Datenloggers. Die Kontroll-LED des Sekundär-Energiespeichers leuchtet noch für ca. 30 s durchgehend blau, danach schaltet sich der Sekun-

där-Energiespeicher selbstständig ab. Um ein Entladen des Capacitor-Towers zu vermeiden und somit Energie zu sparen, empfiehlt es sich, nach dem Abschalten des Beschleunigungs-Datenloggers, das Stellaris LM4F120 LaunchPad Evaluation Kit auch auszuschalten.
Um den Mikrocontroller auf dem Stellaris LM4F120 LaunchPad Evaluation Kit zu programmieren, muss der Capacitor-Tower durch Abziehen der Mikro-USB-Verbindung getrennt und mit dem Mikro-USB-Kabel vom Rechner verbunden werden.

4.4 Bedienungsanleitung

Der Beschleunigungs-Datenlogger ist wie folgt in Betrieb zu nehmen:

1. Die SD-Karte oder Mikro-SD-Karte in einen der vorgesehenen Sockets einlegen.
2. Das Stellaris LM4F120 LaunchPad Evaluation Kit ist durch dessen äußerste Schalterposition auf Aus zu schalten.
3. Den Capacitor-Tower durch Verwenden von einem der drei Energieversorgungvarianten laden. Der Schalter des Capacitor-Towers ist hierbei für die Verwendung des Sekundär-Energiespeichers auf die obere Position (Abb.4.7- Nr.2- S.38) zu legen und der Sekundär-Energiespeicher ist durch einmaligen Knopfdruck durch dessen Einschalter (Abb.4.7- Nr.9- S.38) zu aktivieren. Die Kontroll-LED des Sekundär-Energiespeichers blinkt so lange, bis der Capacitor-Tower geladen ist. Danach leuchtet die Kontroll-Led permanent blau. Für die Verwendung von einen der anderen beiden Varianten ist der Schalter des Capacitor-Towers auf die untere Position zu legen und wahlweise der Primär-Energiespeicher oder die externe Energieversorgung zu verwenden. Der Capacitor-Tower ist hierbei, bei Vollentladung, nach ca. 30 s geladen.
4. Nach Laden des Capacitor-Towers, das Stellaris LM4F120 LaunchPad Evaluation Kit durch dessen Schalterumlegung auf die Innenposition (Abb. 4.7 Nr.21 S.38) einschalten. Das permanente blaue Leuchten der Beschleunigungs-Datenlogger-Kontroll-LED (Abb. 4.7 Nr.19 S.38) signalisiert, dass der Beschleunigungs-Datenlogger erfolgreich aktiviert ist und Daten speichert.

Der Beschleunigungs-Datenlogger ist wie folgt auszuschalten:

1. Der Schalter des Capacitor-Towers (Abb.4.7- Nr.22- S.38) ist auf entgegengesetzte Position zu legen. Hier ist zu beachten, dass bei Verwendung des Sekundär-Energiespeichers, der Primär-Energiespeicher entweder durch unbestückte Primär-Energiespeicherhalterung bzw. durch dessen aufgetrennte Versorgungsleitung (Abb.4.7- Nr.4 & Nr.22- S.38) nicht auf der gegenüberliegenden Schalterposition anliegt oder eine externe Energieversorgung angeschlossen ist. Bei Verwendung des Primär-Energiespeichers oder einer externen Energieversorgung ist zu beachten, dass beim Umlegen des Schalters, der Sekundär-Energiespeicher nicht aktiviert ist.
2. Nach Umschalten des Capacitor-Tower-Schalters, kann die SD-Karte oder Mikro-SD-Karte aus ihrem Socket entfernt werden.

Achtung: Eine Nichteinhaltung der Ausschaltreihenfolge, ein Abtrennen des Mikro-USB-Steckers vom Stellaris LM4F120 LaunchPad Evaluation Kit, ohne den Beschleunigungs-Datenlogger auszuschalten oder ein Entfernen der Speicherkarte während des Betriebs kann einen Datenverlust zur Folge haben.

Kapitel 5

5 Softwareentwicklung

Die Software für den Beschleunigungs-Datenlogger wurde in der Programmiersprache C, primär unter IAR Embedded Workbanch for ARM 7.10.3 entwickelt und getestet. Sekundär wurde sie für die Entwicklerumgebung Code Composer Studio 5.5.0 angepasst und unter dieser getestet. Durch die Verwendung von zwei verschiedenen Entwicklerumgebungen wird Flexibilität bei der späteren Verwendungswahl erreicht. Die in der Software verwendeten Standartfunktionen von Texas Instruments für den Stellaris LM4F120H5QR Mikrocontroller wurden durch das Softwarepaket StellarisWare [17] realisiert. Für die entwickelte Software wurde die Literatur Real-Time Interfacing to ARM CortexTM –M Microcontrollers Embedded Systems [18], ARM Mikrocontroller [19], und C Programmieren von Anfang an [20] zurate gezogen.
Die Softwarerealisierung für die Mikro-SD-Karte und die SD-Karte wurde durch das Softwarepaket AVR-mmc-0.6.4 [21] durch dessen Anpassung für eine Verwendung mit dem Stellaris LM4F120H5QR Mikrocontroller realisiert.

5.1 Hauptroutine

In der Hauptroutine werden grundlegende Konfigurationen des Stellaris LM4F120H5QR Mikrokontrollers, unter Verwendung seines Herstellerdatenblatts [22], vorgenommen. Im ersten Schritt wird hier die System-Clock des Stellaris LM4F120H5QR Mikrocontroller auf 16 MHz gesetzt, um die maximale Geschwindigkeit des Mikrocontrollers zu nutzen. Im nächsten Schritt wird die Funktionen für Brown Out aufgerufen. Es folgen die Funktionsaufrufe für Serial-Peripheral-Interface, SD-Karten-Konfiguration, Inter-Integrated-Circuit, RTC-Konfiguration, Beschleunigungssensor-Konfiguration und Universal-Asynchronous-Receiver-Transmitter. Nach der Konfigurierung der RTC, der Mikro-SD oder alternativ der SD-Karte und des 3-Achsen-Beschleunigungssensor, werden in einer Endlosschleife die zu speichernden Daten der Reihe nach abgerufen. Nach dem jeweiligen Abruf der RTC-Daten, der 3-Achsen-Beschleunigungssensor-Daten, der TSIC-Daten und der GPS-Daten, werden diese für eine erleichterte Auswertung umgewandelt. Nach der Umwandlung der Daten, werden diese auf die Speicherkarte in einem sich selbst anlegenden und alle 10 Minuten ändernden CSV-File gespeichert.

5.2 Brown Out

Brown Out beschreibt den Störfall, der durch Spannungseinbrüche hervorgerufen wird. Anders als bei einem Black Out können diese Störungen durch geeignete Maßnahmen abgefangen werden. Dies ist wichtig, damit es nicht zu undefinierten Zuständen kommt, woraus sich Fehler generieren könnten. Texas Instrument sieht für das Stellaris LM4F120 LaunchPad Evaluation Kit hierfür schon Funktionen vor, wodurch bei einem Brown Out ein Reset des Programms durchgeführt werden kann oder durch Verwendung eines Interrupts, eine geeignete Funktion aufgerufen wird. Beim Beschleunigungs-Datenlogger ist es wichtig, dass bei einem Störfall das Schreiben auf die Mikro-SD-Karte oder auf die SD-Karte sicher beendet wird und das File geschlossen wird. Hierfür wird beim Auftreten von Brown Out ein Interrupt ausgelöst, wodurch eine Funktion aufgerufen wird, die diese realisiert. Die Funktionsweise, einen Brown Out zu erkennen, wird gleichzeitig für ein sicheres Ausschalten des Beschleunigungs-Datenloggers verwendet. Beim Ausschalten oder beim Einbruch der Versorgungsspannung sorgt der Capacitor-Tower dafür, dass noch genügend Energie vorhanden ist, um einen Black Out zu verhindern und um das Programm sicher zu beenden.

5.3 Serial-Peripheral-Interface

Das Serial-Peripheral-Interface oder auch kurz SPI wurde von Motorola entwickelt und gleicht dem Synchronous-Serial-Interface oder auch kurz SSI. Es handelt sich hierbei um ein Master Slave Prinzip mit vier Leitungen. Für den Beschleunigungs-Datenlogger findet dieses Protokoll Anwendung in der Kommunikation zwischen dem Mikrocontroller und Mikro-SD-Karte oder alternativ zwischen Mikrocontroller und SD-Karte, wobei der Mikrocontroller der Master ist und die Karte jeweils der Slave. Zur Realisierung wird in der Hauptroutine für die SD-Karte oder die Mikro-SD-Karte die Funktion mmc_init() aufgerufen. Dies geschieht in einer Schleife bis zur erfolgreichen Durchführung, da eine erfolgreiche Initialisierung der Speicherkarte nicht gleich beim ersten Versuch gewährleistet ist. Der Funktionsaufruf an sich beinhaltet wiederum Funktionensaufrufe, die eigens von Texas Instruments für eine Realisierung von SPI für das Stellaris LM4F120 LaunchPad Evaluation Kit vorgesehen sind. Hierbei wird der Reihe nach erst die Peripherie für SPI ermöglicht, die benötigten Ports werden konfiguriert und die Abtastrate wird definiert. Für die Speicherkarten wurde hier unter den vier möglichen SPIs, die das Stellaris LM4F120 LaunchPad Evaluation Kit vorsieht, die Peripherie SSI0 gewählt. Diese wird über den Port A realisiert, wo sie die Pins 2 bis 5 belegt. Die Abtastrate wird auf 400 kHz definiert. Dies ist lt. Sd Card Specification [23] die maximale Geschwindigkeit, mit der eine SD-Karte konfiguriert werden darf.

5.4 SD-Karten-Konfiguration

In der Funktion mmc_init(), die aus dem Softwarepaket AVR-mmc-0.6.4 [21] stammt, wird nach der Konfigurierung der SPI, die Konfigurierung der Speicherkarte vorgenommen. Hierfür wird durch einen Schleifenaufruf die CS-Leitung für mehr als 74 Durchläufe auf High gelegt, um die Speicherkarte in den SPI-Modus zu setzen. Im nächsten Schritt folgt die Initialisierung der Karte durch das Setzen von CMD- und ACMD-Kommandos. Die folgende Tabelle zeigt eine Auflistung von CMD- und ACMD-Kommandos, die bei der Kommunikation über SPI mit der Speicherkarte lt. Sd Card Specification [23] Verwendung findet.

CMD- und ACMD-Kommandos	
Kommandos	Beschreibung
CMD0	Rücksetzung der Software
CMD1	Initialisierungsprozess
ACMD41	Initialisierungsprozess für SDC
CMD8	Spannungsüberprüfung
CMD9	Lesen des CSD Registers
CMD10	Lesen des CID Registers
CMD12	Lesevorgang Stoppen
CMD13	Status senden
CMD16	Datenblockgröße festlegen
CMD17	Datenblock lesen
CMD18	mehrere Datenblöcke lesen
CMD24	Datenblock schreiben
CMD25	mehrere Datenblöcke schreiben
CMD55	Zulassen von ACMD-Kommandos
CMD58	Lesen des OCR Registers

Tabelle 5.1: CMD- & ACMD-Kommandos

Nach erfolgter Initialisierung wird die Abtastrate auf eine Geschwindigkeit, die halb so groß ist wie die maximale Geschwindigkeit der System Clock und nicht größer als 12,5 MHz, gesetzt. Hierdurch wird ein möglichst schnelles Schreiben auf die Speicherkarte ermöglicht.
Im nächsten Schritt wird für die Speicherkarte durch den Funktionsaufruf fat_loadFatData(), der aus dem Softwarepaket AVR-mmc-0.6.4 [21] stammt, das Format für Fat16 oder Fat32 je nach Speicherkarte realisiert, wonach die Konfigurierung der Speicherkarte abgeschlossen ist.

5.5 Inter-Integrated-Circuit

Das Inter-Integrated-Circuit oder auch kurz I^2C wurde von Phillips Semiconductor entwickelt und ist identisch mit Two-Wire-Interface oder auch kurz TWI.

Für den Beschleunigungs-Datenlogger finden diese Protokolle Anwendung in der Kommunikation zwischen RTC und Mikrocontroller sowie zwischen Beschleunigungssensor und Mikrocontroller. Hierbei ist der Mikrocontroller als Master und die RTC und der Beschleunigungssensor jeweils als Slave definiert. Um I^2C zu realisieren, wird in der Hauptroutine für die RTC die Funktion I2C0_Init() und für den Beschleunigungssensor die Funktion I2C1_Init() aufgerufen. Diese Funktionen beinhalten wiederum Funktionensaufrufe, die eigens von Texas Instruments für eine Realisierung von I^2C für das Stellaris LM4F120 LaunchPad Evaluation Kit vorgesehen sind. Hier wird die Peripherie für I^2C vorbereitet, die benötigten Ports werden konfiguriert und die Datenrate wird gesetzt. Für die RTC wurde hier unter den vier möglichen I^2C Peripherien die das Stellaris LM4F120 LaunchPad Evaluation Kit vorsieht, die Peripherie I2C0 gewählt. Diese wird über den Port B realisiert, wo sie die Pins 3 und 2 belegt. Die Datenrate, wird auf 100 kbps gesetzt. Dies ist lt. Herstellerdatenblatt [10] die maximale Datenrate mit der die RTC verwendet werden darf.

Für den 3-Achsen-Beschleunigungssensor wurde von den vier möglichen I^2C Peripherien des Stellaris LM4F120 LaunchPad Evaluation Kit die Peripherie I2C1 gewählt. Diese wird über den Port A realisiert, wo sie die Pins 6 und 7 belegt. Die Datenrate wird auf 400 kbps gesetzt. Dies ist lt. Herstellerdatenblatt [6] die maximale Datenrate mit der der 3-Achsen-Beschleunigungssensor verwendet werden darf.

5.6 RTC-Konfiguration

Im Funktionsaufruf RTC_config() wird nach der Konfigurierung der I^2C Peripherie die Konfigurierung der RTC vorgenommen. Hier wird eine Startzeit und ein Startdatum gesetzt. Für die RTC erfolgt diese Einstellung nur einmalig und wird, solange der RTC-Pufferspeicher funktionsfähig ist, nicht wiederholt. Aus diesem Grund wird die Funktion nach ihrer erstmaligen, erfolgreichen Ausführung, auskommentiert.

5.7 Beschleunigungssensor-Konfiguration

Im Funktionsaufruf BMA_config() wird nach der Konfigurierung der I^2C Peripherie die Konfigurierung des 3-Achsen-Beschleunigungssensors vorgenommen. Hier wird nach lt. Herstellerdatenblatt [6] die Sensorabtastzeit auf die maximale Frequenz von 1500 Hz gesetzt und der Messbereich wird auf ± 2 G eingestellt.

5.8 Universal-Asynchronous-Receiver-Transmitter

Der Universal-Asynchronous-Receiver-Transmitter oder auch kurz UART ist eine serielle Schnittstelle, über die digitale Signale verarbeitet werden. Für den Beschleunigungs-Datenlogger findet dieses Protokoll Anwendung in der Kommunikation zwischen dem Mikrocontroller und dem GPS-Empfangsmodul. Für die UART-Realisierung wird in der Hauptroutine die Funktion Gps_Init() aufgerufen. Diese Funktion beinhaltet wiederum Funktionsaufrufe, die eigens von Texas Instruments für eine Realisierung von UART für das Stellaris LM4F120 LaunchPad Evaluation Kit vorgesehen sind. Hier wird die Peripherie für UART vorbereitet, die benötigten Ports konfiguriert und die Abtastrate definiert. Für das GPS-Empfangsmodul wurde hier unter den acht möglichen UART Peripherien, die der Stellaris LM4F120 LaunchPad Evaluation Kit bietet, die Peripherie UART1 gewählt. Diese wird über den Port B realisiert, wo sie die Pins 0 und 1 belegt. Die Datenrate wird auf 4800 bps gesetzt. Dies ist lt. Herstellerdatenblatt [12] die maximale Abtastrate, mit der das GPS-Empfangsmodul verwendet werden darf.
Für den GPS- Empfangsmodul selbst wird keine zusätzliche Konfigurierung durchgeführt.

5.9 Datenumwandlung

Für eine erleichterte Auswertung werden die erhobenen Daten umgewandelt.
Hierfür wird durch den Funktionsaufruf RTC_lesen(RTC_DATA) die Daten der RTC ausgelesen. Diese werden dann in den darauffolgenden Funktionen, Anpassung_Datum(RTC_DATA) und Anpassung_Uhrzeit(RTC_DATA) durch Bitverschiebung, Bitnegierung und Bitumrechnungen vor ihrer Speicherung in eine lesbare Form gebracht.
Für den 3-Achsen-Beschleunigungssensor werden die Daten durch den Funktionsaufruf BMA_achsen_lesen(BMA_achsen_werte_10Bit) ausgelesen. Hier werden den Daten im Vorfeld aus zwei 8 Bit Sätzen durch Bitschieben zu 10 Bit zusammengefügt. Diese werden dann in der darauffolgenden Funktion, Anpassung_BMA(BMA_achsen_werte_10Bit) durch Bitverschiebung, Bitnegierung und Bitumrechnungen vor ihrer Speicherung in eine lesbare Form gebracht.
Für das GPS-Empfangsmodul werden die Daten durch den Funktionsaufruf Gps_emp(Gps_signal) ausgelesen. Hier werden die Daten im Vorfeld auf GPS-Signal Anfang und Ende geprüft, da der GPS-Empfänger verschiedene GPS-Signale zurückgibt. Diese werden dann in der darauffolgenden Funktion, Anpassung_Gps(Gps_signal) von nicht benötigten Informationen befreit.

Die Funktion der Temperaturerfassung wurde softwareseitig aus mangelnden Zeitgründen nicht realisiert. Hier wurden lediglich unter dem Funktionsaufruf Anpassung_Temp(Temp) die Anpassungsrechnung für das zu erwartende Bitmuster der Temperatursensoren durch Bitverschiebung, Bitnegierung und Bitumrechnungen realisiert. Um einen Platzhalter für die Temperaturdaten in der Messdatentabelle zu schaffen, wurde hier mit einem festen Temperaturwert gerechnet.

5.10 Datenspeicherung

Die Datenspeicherung erfolgt durch den Funktionsaufruf speichern(). Hier wird geprüft, ob das CSV-File existiert. Die Filebezeichnung setzt sich aus dem Datum, die Uhrzeit und der Endung csv zusammen, wodurch alle 10 min ein neues File angelegt wird. Bei nicht existierendem File wird eine neue Fileanlegung vorgenommen. Im ersten Schritt wird im CSV-File ein Tabellenkopf angelegt. Dieser setzt sich aus einem Tages- und Zeitstempel zusammen. Dann wird die Tabellenbezeichnung für Zeit, Beschleunigungsdaten, interner und externer Temperatur, und GPS-Koordinaten vorgenommen. Im letzten Schritt werden die gewandelten Daten der RTC für die Uhrzeit, die drei Beschleunigungswerte des 3-Achsen-Beschleunigungssensors, die interne und externe Temperatur der TSICs, und die empfangenen GPS-Signale abgespeichert. Bei einem existierenden CSV-File werden die zu speichernden Daten an die vorherigen angehängt.

Kapitel 6

6 Beschleunigungs-Datenlogger-Test

Um den Beschleunigungs-Datenlogger auf seine Funktion zu testen, wurde er im Labor und im Feldversuch einer Testreihe unterzogen. Bei den Versuchsreihen wurde auf eine Datenerfassung von Temperatur und GPS-Signal verzichtet, da es sich hierbei um Zusatzfunktionen des Beschleunigungs-Datenloggers handelt, für die ein Funktionstest nicht von Relevanz ist. Auf Grund des hohen Datenvolumens, das schon bei kurzen Messungen auftritt, befinden sich die Messdatentabellen zu den einzelnen Versuchen nur auf der Zusatz-DVD im Ordner (Messdaten).

6.1 Testdatenauszug

Da bei einer Klasse 10 SD-Karten ca. 4 Messdatensätze pro Sekunde in einer Tabelle, die sich in einem angelegten CSV File befindet, gespeichert werden, erfolgt hier nur ein kleiner Auszug der Testdatenerfassung. Des Weiteren wurden für eine bessere Darstellung die Messdaten auf zwei Tabellen aufgeteilt und überarbeitet.

Beschleunigungs-Datenlogger			**Datum:**	**09.09.2014**	**Uhrzeit:**
Uhrzeit	**X-Achse**	**Y-Achse**	**Z-Achse**	**Temp.(IN)**	**Temp.(OUT)**
14:22:14	0	0,94	9,62	025,5C°	025,5C°
14:22:14	0	-0,07	9,31	025,5C°	025,5C°
14:22:14	0	0,4	9,27	025,5C°	025,5C°
14:22:14	0	-0,15	9,35	025,5C°	025,5C°
14:22:14	0	0,55	9,89	025,5C°	025,5C°
14:22:14	0	-0,04	9,31	025,5C°	025,5C°
14:22:15	0	-0,15	9,54	025,5C°	025,5C°
14:22:15	0	-0,15	9,62	025,5C°	025,5C°
14:22:15	0	-0,46	9,23	025,5C°	025,5C°
14:22:15	0	0,43	9.,2	025,5C°	025,5C°

Tabelle 6.1: Testdatenauszug 1

14:22:13
GPS-Koordinaten
$GPVTG,36.94,T,,M,0.83,N,1.5,K,A.3587,E,0.83,36.94,090914,,,A
$GPGGA,122328.503,5348.5747,N,01000.3586,E,1,04,6.1,62.7,M,45.7,M,,0000
$GPGSA,A,3,18,27,08,16,,,,,,,,,9.2,6.1,6.9,1,04,6.1,62.7,M,45.7,M,,0000
$GPRMC,122328.503,A,5348.5747,N,01000.3586,E,0.72,34.82,090914,,,A,0000
$GPVTG,34.82,T,,M,0.72,N,1.3,K,A01000.3586,E,0.72,34.82,090914,,,A,0000
$GPGGA,122329.503,5348.5746,N,01000.3585,E,1,04,6.1,62.3,M,45.7,M,,0000
$GPGSA,A,3,18,27,08,16,,,,,,,,,9.2,6.1,6.9,1,04,6.1,62.3,M,45.7,M,,0000
$GPRMC,122329.503,A,5348.5746,N,01000.3585,E,0.68,34.20,090914,,,A,0000
$GPVTG,34.20,T,,M,0.68,N,1.3,K,A01000.3585,E,0.68,34.20,090914,,,A,0000
$GPGGA,122330.503,5348.5745,N,01000.3584,E,1,04,6.1,61.9,M,45.7,M,,0000

Tabelle 6.2: Testdatenauszug 2

Folgende GPS Signalarten werden gespeichert:

RMC = Recommended Minimum Sentence C

GGA = Fix Information

GSA = Overall Satellite Data

GSV = GPS Satellite in View

VTG = Vector track and Speed over the Ground

Die GPS-Koordinaten setzten sich beispielsweise für die RMC Signalart lt. The NMEA 0183 Protocol [3] wie in der folgenden Tabelle 6.3 zusammen.

GPS-Koordinaten-Signalart-RMC	
Zeichen	**Bedeutung**
$GPRMC	Signalart RMC
122328.503	Uhrzeit
A	Status OK
5348.5747	Breitengrad
N	Ausrichtung Norden
01000.3586	Längengrad
E	Ausrichtung Westen
0.72	Geschwindigkeit in Knoten
34.82	Kurs
090914	Datum

Tabelle 6.3: GPS-Koordinaten-Signalart-RMC

6.2 Labor-Ausrichtungstest

Im Labor wurde der Beschleunigungs-Datenlogger auf eine, sich in Waage ausgerichtete, Versuchsoberfläche positioniert. In der folgenden Abbildung 6.1 werden die hier erfassten Messdaten der 3-Achsen grafisch dargestellt.

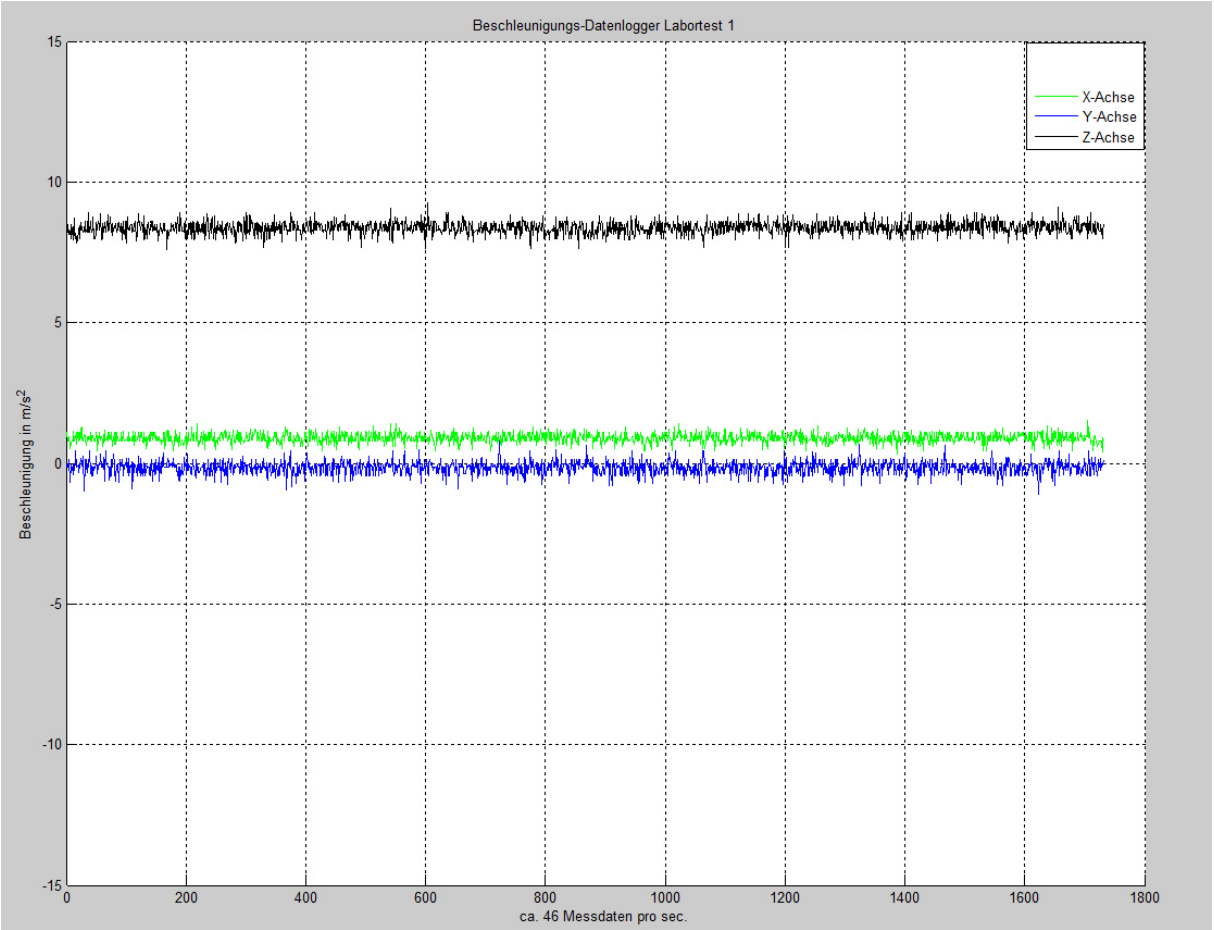


Abbildung 6.1: Beschleunigungs-Datenlogger Labortest 1

Aus den erfassten Daten ergibt sich:

Eine mittlere Abweichung der X-Achse in Ruhelage (0 m/s^2) von 0,89 m/s^2.

Eine mittlere Abweichung der Y-Achse in Ruhelage (0 m/s^2) von –0,16 m/s^2.

Eine mittlere Abweichung der Z-Achse zur Gravitation von 9,81 m/s^2 von 8,38 m/s^2.

Endbetrachtung:

Lt. Herstellerdatenblatt [6] ist eine Abweichung in Ruhelage bei allen Achsen um ±2,2 m/s^2 möglich, wodurch sich die starke Abweichung der X-Achse erklären lässt.

Für eine vereinfachte Auswertung und bessere Darstellung wurden die 3 Achsen nach dem Versuch jeweils durch eine mathematische Anpassung im Programm in ihrer Ruhelage möglichst nahe zu 0 verschoben und in diesem Bereich grob geglättet. Dies geschah unter Beach-

tung, dass bei voller positiv auftretender Gravitation auf die einzelnen Achsen, der sich hier erfasste Datenmittelwert, an 9,81 m/s^2 annähert und bei voller negativ auftretender Gravitation jeweils an -9,81 m/s^2.

6.3 Labor-Gravitationstest

Für den Gravitationstest wurden die auftretenden Beschleunigungen in Ruhelage und jeweils bei maximal und minimal auftretender Gravitation für die 3 Achsen separat erfasst und in den folgenden 3 Grafiken dargestellt.

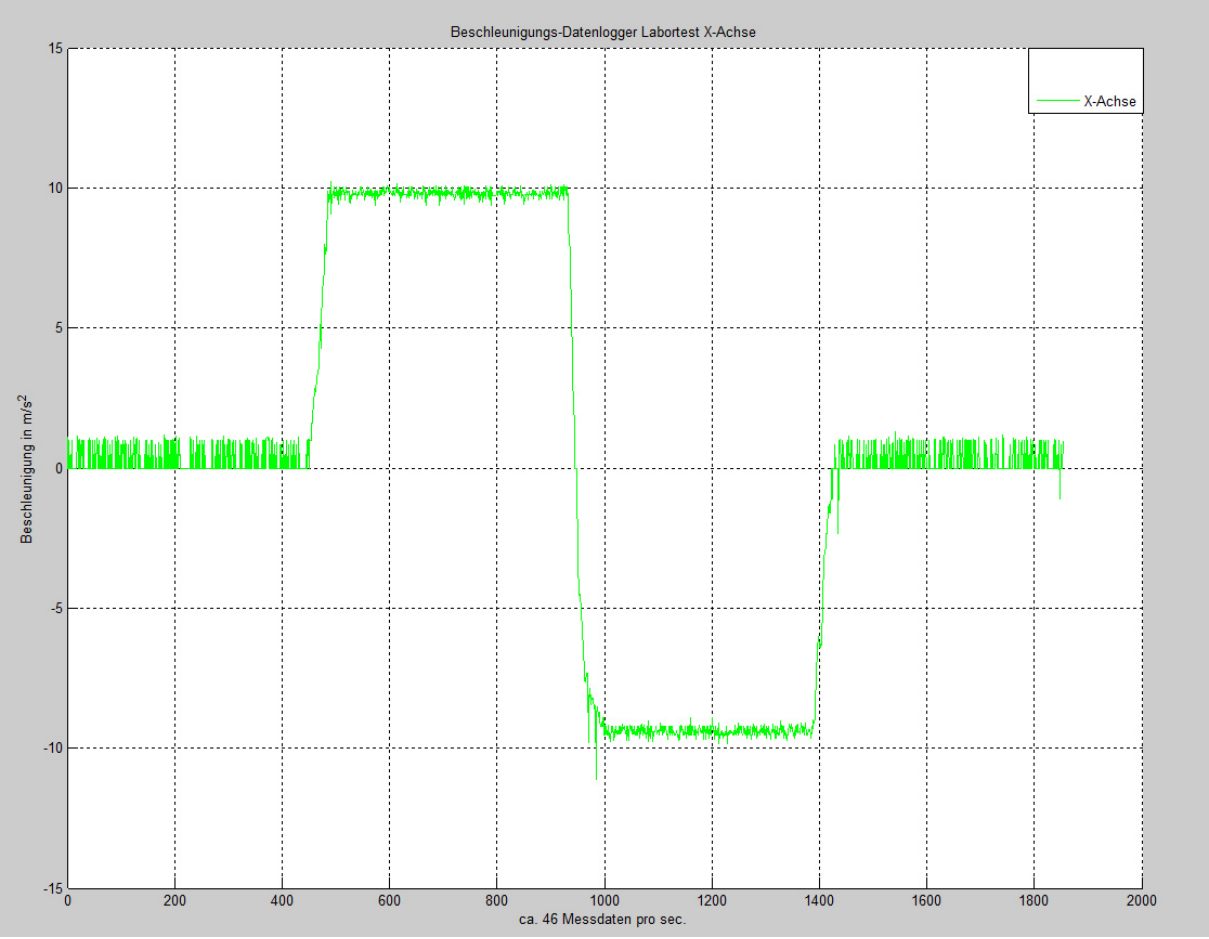


Abbildung 6.2: Beschleunigungs-Datenlogger Labortest X-Achse

Aus den erfassten Daten ergibt sich:

Ein Mittelwert in Ruhelage von 0,29 m/s^2.

Ein Mittelwert bei voller positiver Gravitationseinwirkung von 9,79 m/s^2.

Ein Mittelwert bei voller negativer Gravitationseinwirkung von -9,40 m/s^2.

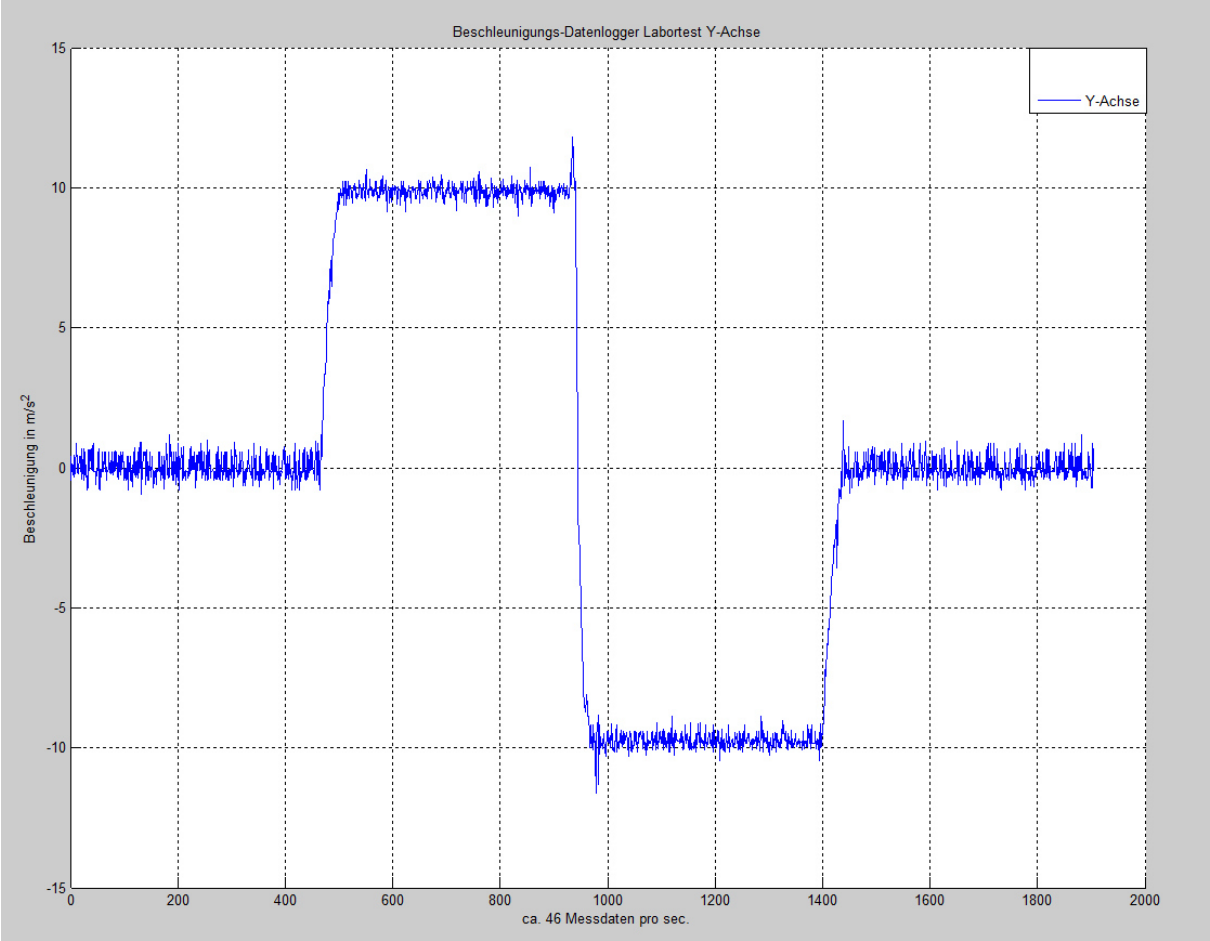


Abbildung 6.3: Beschleunigungs-Datenlogger Labortest Y-Achse

Aus den erfassten Daten ergibt sich:

Ein Mittelwert in Ruhelage von 0,01 m/s^2.

Ein Mittelwert bei voller positiver Gravitationseinwirkung von 9,86 m/s^2.

Ein Mittelwert bei voller negativer Gravitationseinwirkung von -9,74 m/s^2.

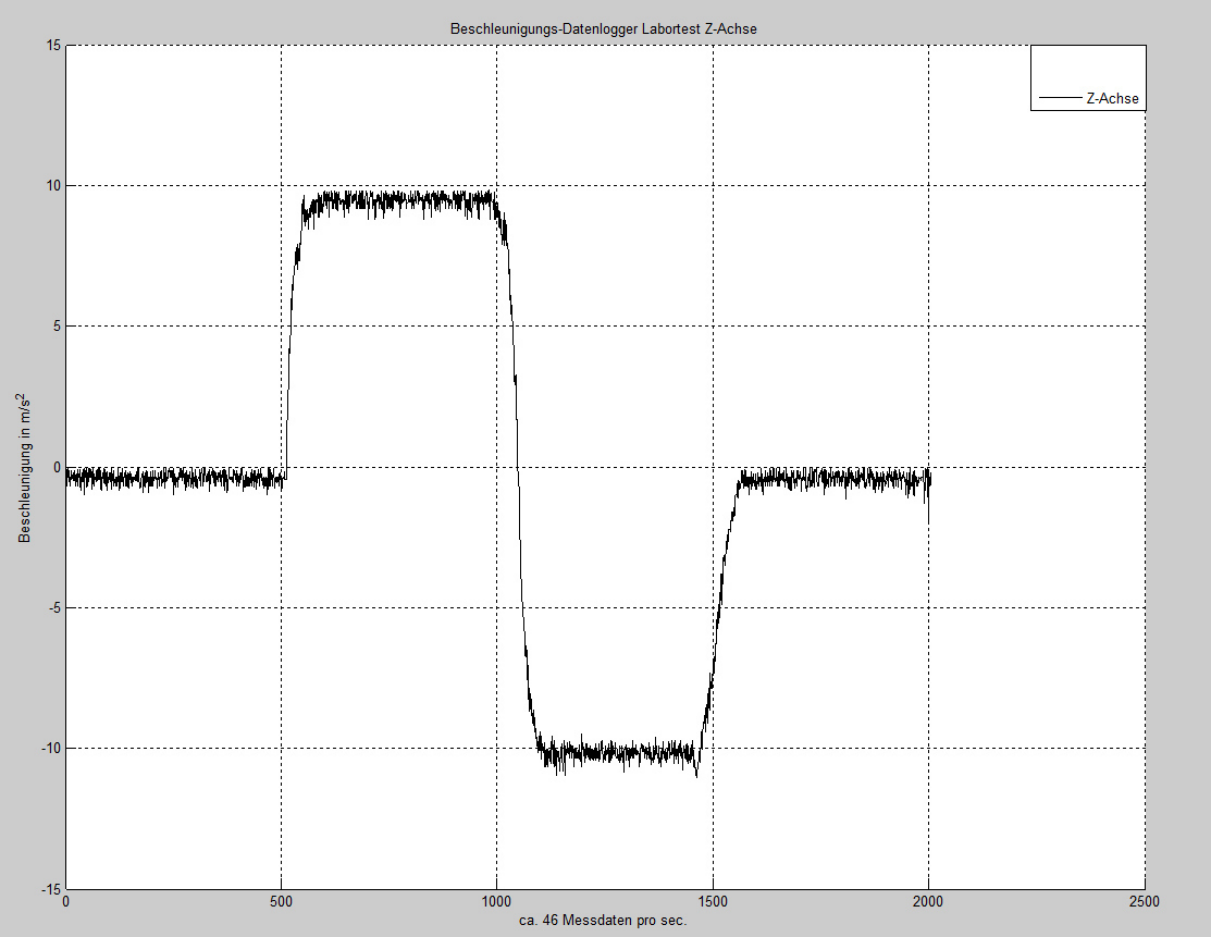


Abbildung 6.4: Beschleunigungs-Datenlogger Labortest Z-Achse

Aus den erfassten Daten ergibt sich:

Ein Mittelwert in Ruhelage von -0,4 m/s^2.

Ein Mittelwert bei voller positiver Gravitationseinwirkung von 9,45 m/s^2.

Ein Mittelwert bei voller negativer Gravitationseinwirkung von -10,16 m/s^2.

Endbetrachtung:

Bei den Beschleunigungssensoren handelt es sich um Silicium-Federn. Diese schwingen permanent, wodurch das Erfassen von einem gradlinigen Signal, selbst in Ruhelage, nicht möglich ist. Bei einer Auswertung darf somit kein Einzelsignal, sondern immer nur der Mittelwert aus mehreren Daten betrachtet werden. Prinzipiell lässt sich der Beschleunigungs-Datenlogger überall positionieren, da dieser aber das qualitativ hochwertigste Signal in Bezug auf Abweichung bei Ruhelage und beim Gravitationstest bei der Y-Achse hat, empfiehlt es sich, den Beschleunigungs-Datenlogger in Fahrtlichtung zu positionieren. Für eine positive

Messdatenerfassung an der Y-Achse bei Anfahren des Kraftfahrzeugs, muss hierbei beachtet werden, dass der Beschleunigungs-Datenlogger mit dem externen Temperatursensor entgegengesetzt zur Fahrtrichtung zeigt.

6.4 Feldversuch

Im Feldversuch wurden Testdaten durch ein Kraftfahrzeug auf einer Teststrecke in zwei unterschiedlichen Versuchen ermittelt und ausgewertet. Um die Reproduzierbarkeit der Testdaten zu gewährleisten, wurden die Versuche mehrfach wiederholt.

6.4.1 Testfahrzeug

Bei dem für den Feldversuch verwendeten Kraftfahrzeug handelt es sich um einen VW Polo. Dieser hat als Bereifung 145 R13 74S und eine Nennleistung von 40 kW. Das Fahrzeuggewicht mit Testfahrer lag zum Zeitpunkt der Tests bei ca. 905 kg. Für die Testdatenerfassung wurde der Beschleunigungs-Datenlogger auf der Rückbank des Kraftfahrzeugs in der Mitte positioniert und mit dem Anschnallgurt fixiert. Hier sind mögliche Störeinflüsse durch veränderte Fahrzeuglage, bedingt durch schnelle positive Beschleunigung und durch eine nicht starre Fahrzugaufhängung als gering einzuschätzen, da sich diese Position nahe an der Hinterachse befindet. Optimal ist eine mittig zwischen den beiden Achsen liegende Position, um die Beeinflussung bei negativer Beschleunigung zu minimieren. Dies ist aus mangelnder Fixierungsmöglichkeit bei dem Testkraftfahrzeug nicht möglich.

6.4.2 Teststrecke

Bei der Teststrecke handelt es sich um eine ca. 500 m lange grobkörnig asphaltierte Straße. Diese weist einen fast geradlinigen Verlauf auf. Die Versuche wurden auf den ersten 400 m durchgeführt. Durch nachträgliche Baumaßnahmen für Kabel und Rohrsysteme weist die Teststrecke an einigen Stellen horizontal verlaufende Nachbesserungen mit einer Breite von bis zu 1 m auf. Diese führen zu einer Fahrbahnanhebung oder Absenkung. Des Weiteren weist die Teststrecke an einigen Stellen ein leicht seitliches Gefälle auf. Die Fahrbahn ist durch die Witterungsverhältnisse in einem trockenen Zustand.

6.4.3 Feldversuch 1

Für diesen Test wurde das Testfahrzeug möglichst schnell von Stillstand auf 100 km/h beschleunigt und dann wieder unter Fahrzeug schonenden Bedingungen, schnellstmöglich zum Stillstand gebracht. Die nachfolgende Grafik 6.5 stellt hier den Testverlauf für die 3 Achsen dar.

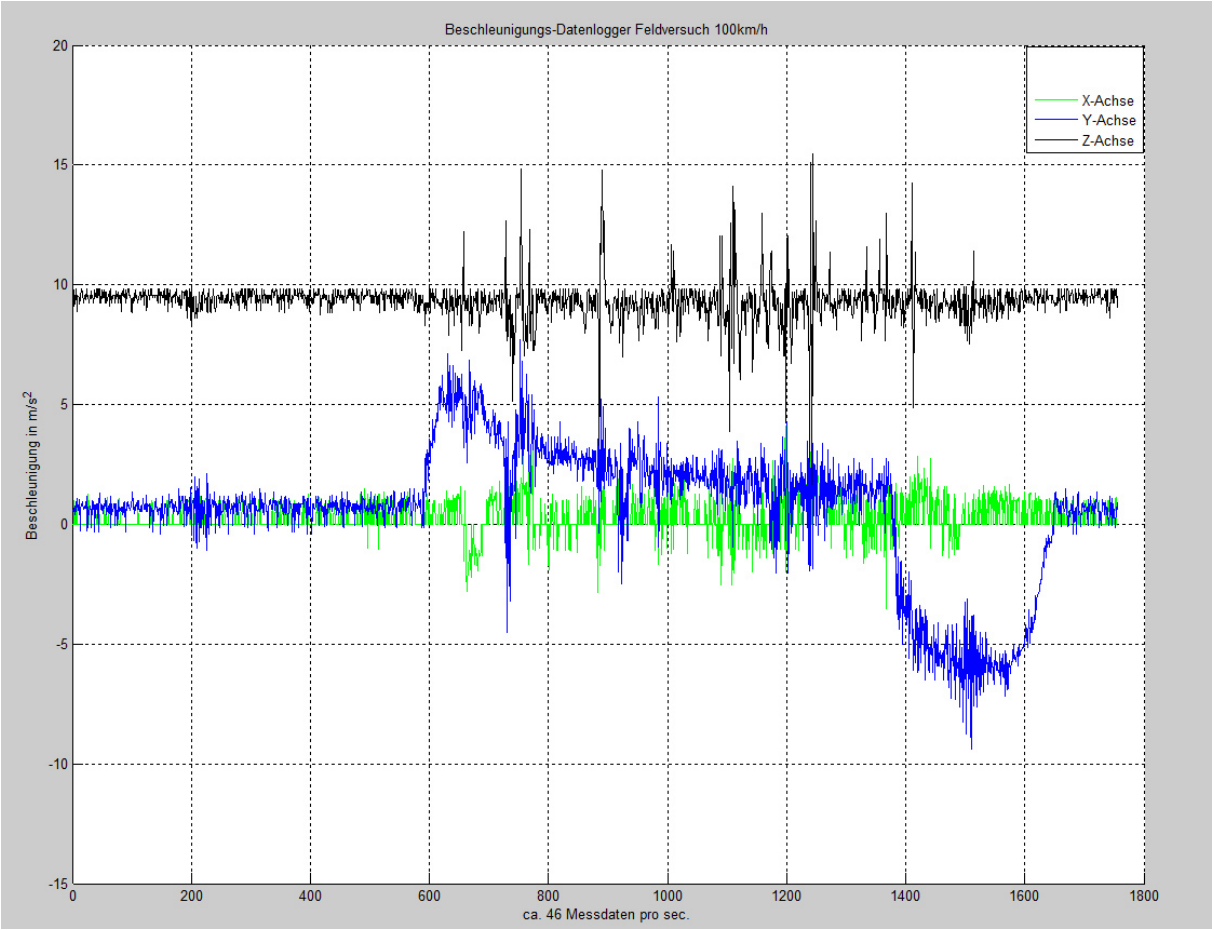


Abbildung 6.5: Beschleunigungs-Datenlogger Feldversuch 100 km/h

Die Y-Achse stellt die Beschleunigung in Fahrtrichtung dar.

Die X-Achse stellt die Beschleunigung in seitlicher Richtung dar.

Die Z-Achse stellt die Beschleunigung in der vertikalen Richtung dar.

Die Testdaten wurden in einem Zeitraum von 39 s erfasst, hierbei wurden ca. 46 Messdatensätze pro Sekunde erhoben.

Für die Analyse wurden die Z-Achse und die X-Achse gemeinsam dargestellt, die Y-Achse erfolgt separat in der darauffolgenden Grafik 6.7. Für die Auswertung wurden in beiden Grafiken 9 Sektoren von A.1 bis A.9 markiert.

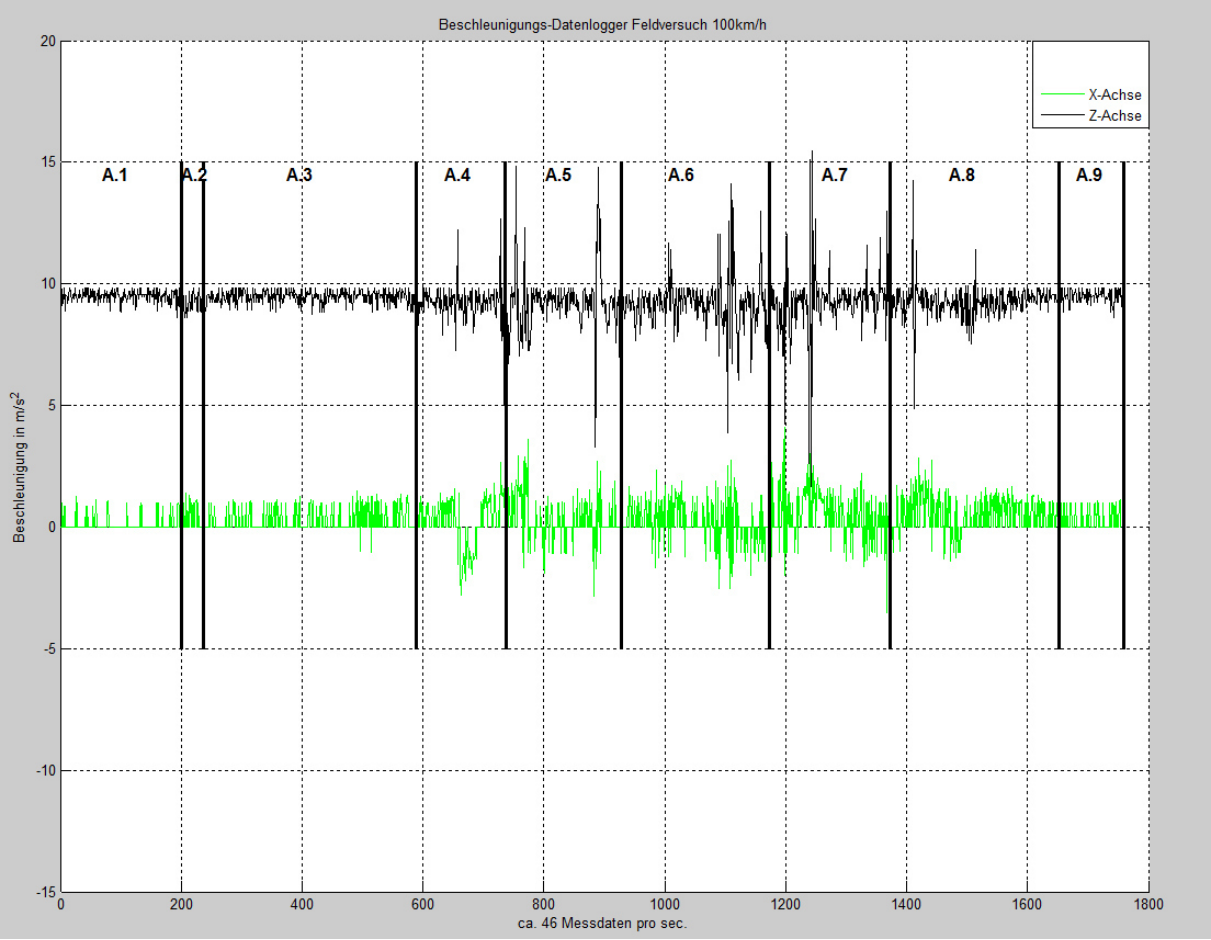


Abbildung 6.6: Beschleunigungs-Datenlogger Feldversuch 100 km/h Z- & X-Achsen

Da für eine Analyse nach dem Gesichtspunkt der Funktionalität für eine Auswertung in Bezug auf Rekuperation die Y-Achse die entscheidende ist, wird diese genau analysiert. Die Z-Achse und die X-Achse werden nur auf Besonderheiten untersucht.

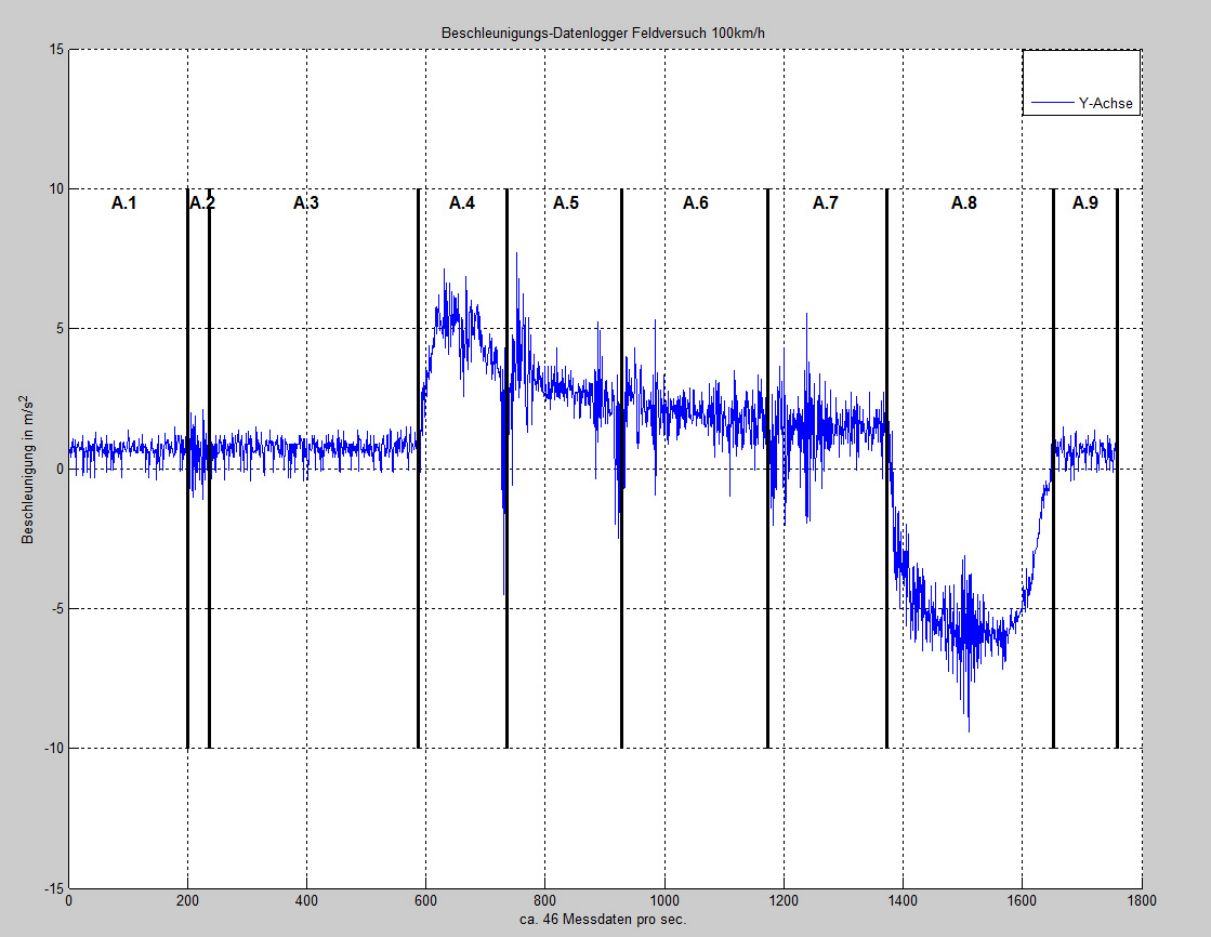


Abbildung 6.7: Beschleunigungs-Datenlogger Feldversuch 100 km/h Y-Achse

Sektor A.1

Kraftfahrzeug befindet sich im Stillstand, das Antriebsaggregat ist deaktiviert.

Zeitraum der Datenerfassung ca. 4 s.

Mittelwert der Beschleunigung bei 0,68 m/s^2.

Analyse: Abweichung von der 0 Achse durch eine, von der waagerechten abweichende Positionierung beim Einbau des Beschleunigungs-Datenloggers.

Sektor A.2

Antriebsaggregat des Kraftfahrzeugs wird aktiviert. Kraftfahrzeug befindet sich im Stillstand.

Zeitraum der Datenerfassung ca. 1 s.

Analyse: Bei der Aktivierung des Antriebsaggregats wird das Kraftfahrzeug für kurze Zeit in Vibration und Schwingung versetzt. Diese äußert sich durch einen leichten Ausschlag der Beschleunigungen in Richtung der Y-Achse.

Sektor A.3

Antriebsaggregat ist eingeschaltet, Kraftfahrzeug befindet sich im Stillstand.
Zeitraum der Datenerfassung ca. 8 s.
Mittelwert der Beschleunigung bei 0,72 m/s^2.
Analyse: Werte sind fast identisch mit Sektor A.1. Der geringe Mittelwertanstieg von 0,04 m/s^2 wird auf das Einfedern des Fahrzeugfahrwerks nach der Aktivierung des Antriebsaggregats zurückgeführt.
Fazit: Den Mittelwert der Beschleunigung von 0,72 m/s^2 muss bei der weiteren Auswertung berücksichtigt werden und von den positiven Beschleunigungen subtrahiert werden. Bei den negativen Beschleunigungen muss die 0,72 m/s^2 negativ aufaddiert werden.

Sektor A.4

Kraftfahrzeug wird im 1. Gang angefahren.
Zeitraum der Datenerfassung ca. 3 s.
Mittelwert der Beschleunigung bei 3,81 m/s^2.
Erreichte Geschwindigkeit vor dem Schalten in den 2. Gang ca. 40 km/h.
Beim Schalten in den 2. Gang Einbruch der Beschleunigung.
Die Schwankungen bei der Z-Achse und X-Achse sind auf Fahrbahnunebenheiten zurückzuführen.
Analyse: Die maximal auftretende Beschleunigung findet in den ersten Sekunden statt.
Der Mittelwert liegt hier bei 4,56 m/s^2.

Sektor A.5

Getriebe des Kraftfahrzeugs wird in den 2. Gang geschaltet.
Erfasster Zeitraum der Daten ca. 4 s.
Mittelwert der Beschleunigung bei 2,23 m/s^2.
Schalten in den 3. Gang bei ca. 60 km/h.
Analyse: Die Schwingung um den Mittelwert nimmt bei allen 3 Achsen zu. Fahrbahnunebenheiten machen sich mit steigender Geschwindigkeit bei der Z-Achse und X-Achse durch höhere Beschleunigungswerte bemerkbar.

Sektor A.6

Getriebe des Kraftfahrzeugs wird in den 3. Gang geschaltet.

Zeitraum der Datenerfassung ca. 6 s.

Mittelwert der Beschleunigung bei 1,28 m/s^2.

Schalten in den 4. Gang bei ca. 90 km/h.

Analyse: Keine weiteren Besonderheiten in Vergleich zu Sektor A.5.

Sektor A.7

Getriebe des Kraftfahrzeugs wird in den 4. Gang geschaltet.

Zeitraum der Datenerfassung ca. 4 s.

Mittelwert der Beschleunigung bei 0,71 m/s^2.

Endgeschwindigkeit bei 100 km/h.

Analyse: Keine weiteren Besonderheiten im Vergleich zu Sektor A.5.

Sektor A.8

Kontrollierter, Kfz schonender und schneller Bremsvorgang von 100 km/h auf 0 km/h.

Zeitraum der Datenerfassung ca. 6 s.

Mittelwert der Beschleunigung -5,42 m/s^2.

Analyse: Keine weiteren Besonderheiten im Vergleich zu Sektor A.5.

Sektor A.9

Antriebsaggregat ist aktiviert, Kraftfahrzeug befindet sich im Stillstand.

Zeitraum der Datenerfassung ca. 2 s.

Mittelwert der Beschleunigung 0,67 m/s^2.

Analyse: Werte sind fast identisch mit Sektor A.1. Die Abweichung zu Sektor A.3 wird der leichten Fahrbahnneigung am Ende der Teststrecke zugeschrieben.

Auswertung:

Bei Addition der gesamten positiven Beschleunigung über deren Zeiträume ergibt sich eine Geschwindigkeit von:

$$v = 3\,\mathrm{s} \cdot 3,81\,\frac{\mathrm{m}}{\mathrm{s}^2} + 4\,\mathrm{s} \cdot 2,23\,\frac{\mathrm{m}}{\mathrm{s}^2} + 6\,\mathrm{s} \cdot 1,28\,\frac{\mathrm{m}}{\mathrm{s}^2} + 4\,\mathrm{s} \cdot 0,71\,\frac{\mathrm{m}}{\mathrm{s}^2} = 30,87\,\frac{\mathrm{m}}{\mathrm{s}}$$

Werden die gesamten negativen Beschleunigungen über deren Zeitraume addiert, resultiert eine Geschwindigkeit von:

$$v = 6\,\mathrm{s} \cdot 5,42\,\frac{\mathrm{m}}{\mathrm{s}^2} = 32,52\,\frac{\mathrm{m}}{\mathrm{s}}$$

Die Abweichung beträgt:

$$32{,}52\frac{\mathrm{m}}{\mathrm{s}} - 30{,}87\frac{\mathrm{m}}{\mathrm{s}} = 1{,}65\frac{\mathrm{m}}{\mathrm{s}}$$

Theoretisch müssten beide Werte identisch sein, da es sich um ein geschlossenes Energiesystem handelt. Die in dem Test ermittelte Abweichung beträgt 1,65 m/s. Diese ist auf eine sich stärker ändernde Position des Beschleunigungs-Datenloggers beim Bremsvorgang zurückzuführen. Die Ursache hierfür liegt in der nahen Positionierung zur Hinterachse, wodurch beim Bremsen die Beschleunigungswerte leicht angehoben werden.

Um eine Messdatenverfälschung durch veränderte Lageposition zu minimieren, sollte der Beschleunigungs-Datenlogger nach Möglichkeit in der Kraftfahrzeugmitte, im gleichen Abstand zur Vorder- und Hinterachse, fest montiert werden. Da diese Möglichkeit bei dem verwendeten Kraftfahrzeug nicht bestand, wurden hier die positiven Beschleunigungen über den Zeitraum mit der negativen Beschleunigungen über den Zeitraum gegenübergestellt und der Mittelwert aus beiden gebildet.

Hieraus ergibt sich ein Mittelwert von:

$$\frac{(32,52\,\frac{\mathrm{m}}{\mathrm{s}} + 30,87\,\frac{\mathrm{m}}{\mathrm{s}})}{2} \approx 31,7\,\frac{\mathrm{m}}{\mathrm{s}}$$

Der Mittelwert dividiert durch die Bremszeit ergibt dann annähernd die Maximalbeschleunigung beim Bremsvorgang von:

$$a = \frac{31{,}7\ \frac{\mathrm{m}}{\mathrm{s}}}{6\ \mathrm{s}} \approx 5{,}28\ \frac{\mathrm{m}}{\mathrm{s}^2}$$

Nach Umstellen der Formel (2.1.1) ergibt sich hiernach eine Kraft von:

$$F = 5{,}28\ \frac{\mathrm{m}}{\mathrm{s}^2} \cdot 905\ \mathrm{kg} = 4{,}781\ \mathrm{kN}$$

Der hierbei zurückgelegte Bremsweg ist nach der Formel (2.1.3) eine Strecke von:

$$v = 100\ \frac{\mathrm{km}}{\mathrm{h}} \approx 27{,}78\ \frac{\mathrm{m}}{\mathrm{s}}$$

$$s = \frac{(27{,}78\ \frac{\mathrm{m}}{\mathrm{s}})^2}{2 \cdot\ 5{,}28\ \frac{\mathrm{m}}{\mathrm{s}^2}} \approx 73\mathrm{m}$$

Hieraus resultiert eine kinetische Energie von:

$$E_{kin} = 4{,}781\ \mathrm{kN} \cdot 73\mathrm{m}\ \approx 349\ \mathrm{kJ}$$

Als Vergleichsrechnung mit der Formel (2.1.4) ergibt sich für des Kraftfahrzeug eine kinetische Energie von:

$$E_{kin} = \frac{1}{2} \cdot 905\ \mathrm{kg} \cdot 27{,}78^2\ \left(\frac{\mathrm{m}}{\mathrm{s}}\right)^2 \approx 349{,}2\ \mathrm{kJ}$$

Die kinetische Energie, die aus den Messwerten errechnet wurde, ist fast mit dem in der Vergleichsrechnung rein rechnerisch ermittelten Wert identisch. Die Abweichung beträgt 0,2 kJ und ist auf Messungenauigkeit und Rundungsfehler zurückzuführen.
Der Beschleunigungs-Datenlogger hat im Feldversuch 1 erfolgreich Änderungen von Beschleunigungen erfasst und gespeichert, die gut auswertbar sind. Die bei der Auswertung er-

mittelten Werte sind plausibel, der Versuch hat die Funktionsfähigkeit des Beschleunigungsdatenloggers somit bestätigt.

6.4.4 Feldversuch 2

Für diesen Test wurde das Testfahrzeug mit einer Simulation von einem üblichen Fahrverhalten vom Stillstand auf 60 km/h beschleunigt und nach kurzer Fahrzeit langsam auf 0 km/h abgebremst. Die nachfolgende Grafik 6.8 stellt hier den Testverlauf für die 3 Achsen dar.

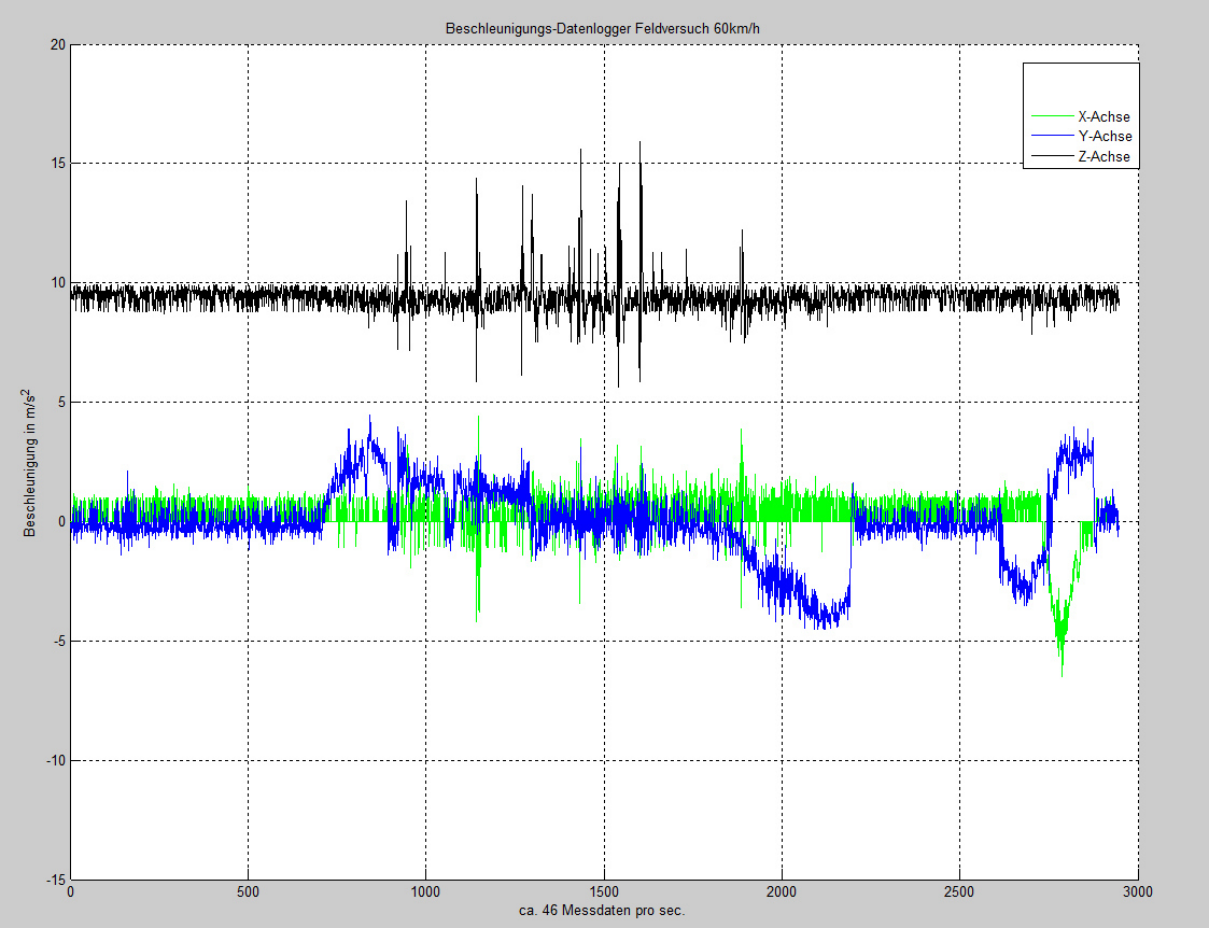


Abbildung 6.8: Beschleunigungs-Datenlogger Feldversuch 60 km/h

Die Y-Achse stellt die Beschleunigung in Fahrtrichtung dar.
Die X-Achse stellt die Beschleunigung in seitlicher Richtung dar.
Die Z-Achse stellt die Beschleunigung in vertikaler Richtung dar.
Die Testdaten wurden in einem Zeitraum von 65 s erfasst, hierbei wurden ca. 46 Messdatensätze pro Sekunde erhoben.

Für die Analyse wurden, wie bei dem ersten Feldversuch, die Z-Achse und die X-Achse zusammen dargestellt. Die Darstellung der Y-Achse erfolgt separat in den darauffolgenden Grafiken 6.10. Für die Auswertung wurden in beiden Grafiken 11 Sektoren von B.1 bis B.11 markiert.

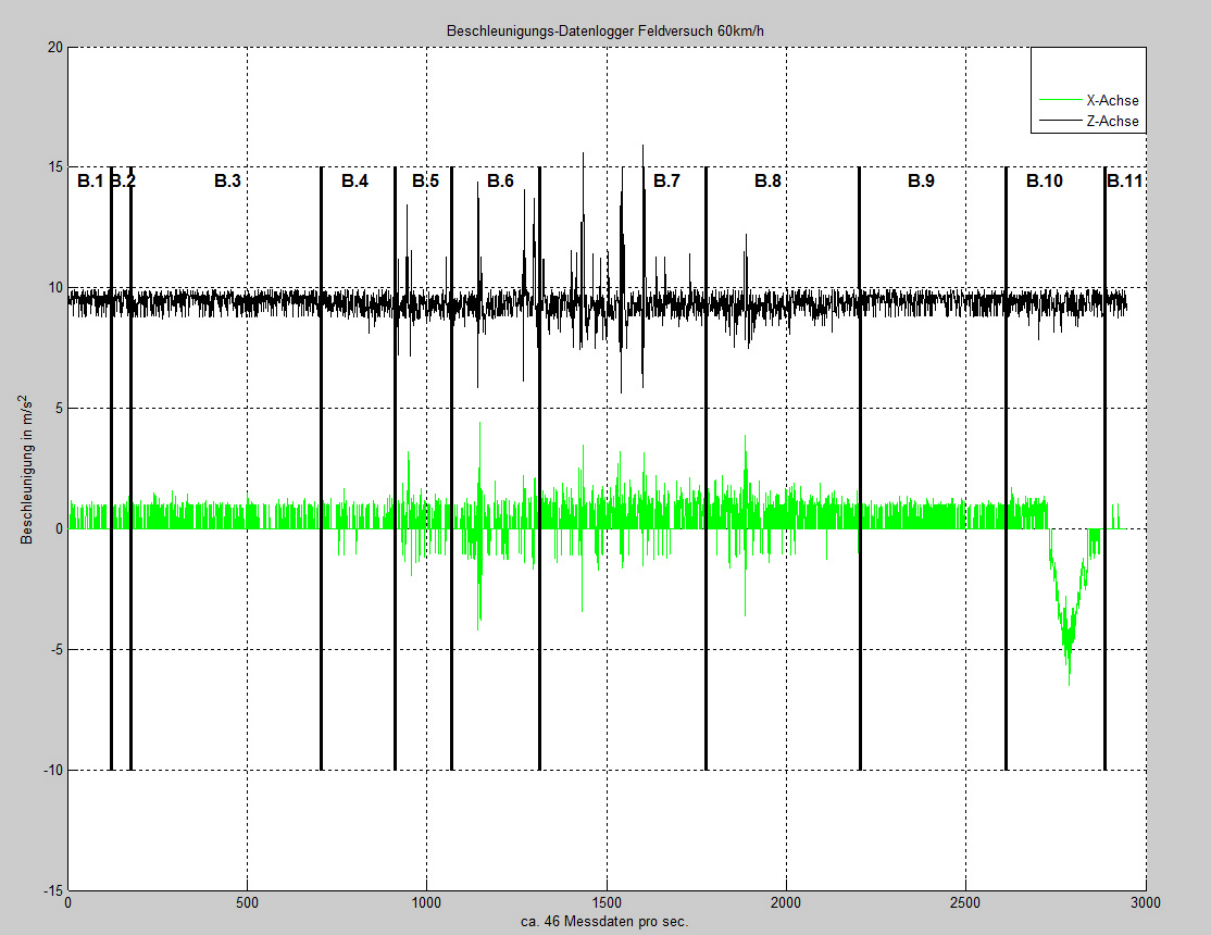


Abbildung 6.9: Beschleunigungs-Datenlogger Feldversuch 60 km/h Z-& X-Achsen

Da für eine Analyse, unter dem Aspekt der Funktionalität, für eine Auswertung in Bezug auf Rekuperation die Y-Achse die entscheidende ist, wird diese genauer analysiert. Auf die Z-Achse und die X-Achse werden wiederum nur Besonderheiten erwähnt.

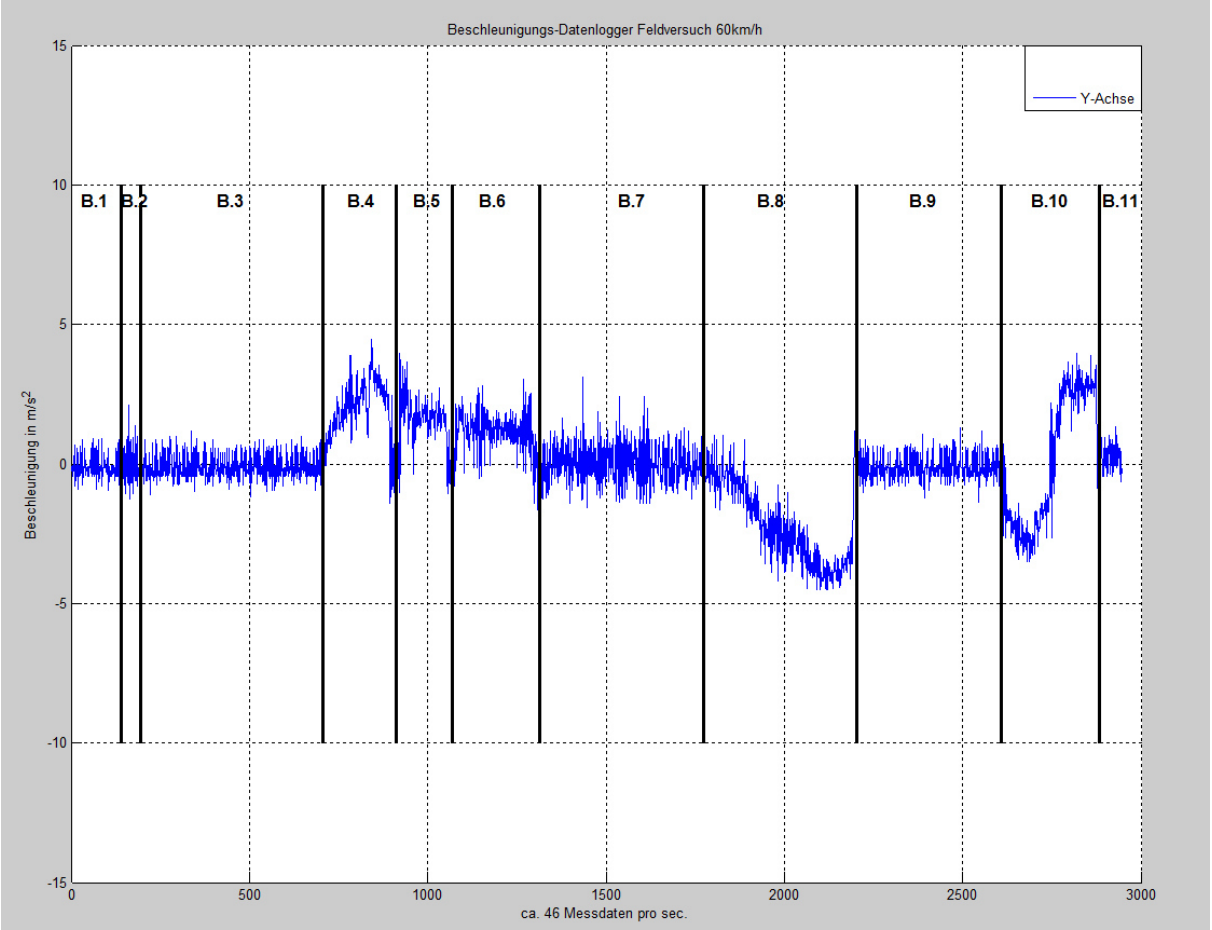


Abbildung 6.10: Beschleunigungs-Datenlogger Feldversuch 60 km/h Y-Achse

Sektor B.1

Kraftfahrzeug befindet sich im Stillstand das Antriebsaggregat ist deaktiviert.

Zeitraum der Datenerfassung ca. 2 s.

Mittelwert der Beschleunigung bei -0,12 m/s^2.

Analyse: Abweichung von der 0-Achse durch eine von der waagerecht abweichenden Positionierung beim Einbau des Beschleunigungs-Datenloggers.

Sektor B.2

Antriebsaggregat des Kraftfahrzeugs wird aktiviert, Kraftfahrzeug befindet sich im Stillstand.

Zeitraum der Datenerfassung ca. 1 s.

Analyse: Beim Starten des Motors wird das Kraftfahrzeug für kurze Zeit in Vibration und Schwingung versetzt. Dies äußert sich durch einen leichten Ausschlag der Beschleunigungen in Richtung der Y-Achse.

Sektor B.3

Antriebsaggregat ist aktiviert, Kraftfahrzeug befindet sich im Stillstand. Werte sind identisch mit Sektor B.1.

Mittelwert der Beschleunigung bei -0,12 m/s^2.

Zeitraum der Datenerfassung ca. 11 s.

Fazit: der Mittelwert der Beschleunigung von 0,12 m/s^2 muss bei der weiteren Auswertung berücksichtigt werden und auf die positiven Beschleunigungen addiert werden. Bei den negativen Beschleunigungen müssen die 0,12 m/s^2 subtrahiert werden.

Sektor B.4

Kraftfahrzeug wird im 1. Gang angefahren.

Zeitraum der Datenerfassung ca. 4 s.

Mittelwert der Beschleunigung bei 2,34 m/s^2.

Erreichte Geschwindigkeit vor dem Schalten in den 2. Gang, ca. 30 km/h.

Analyse: Beim Schalten in den 2. Gang bricht die Beschleunigung ein. Durch den schlagartigen Wegfall schwingen die Silicium-Sensorfeder leicht über, wodurch eine kurze, niedrige negative Beschleunigung gemessen wurde, die nicht existent ist.

Die Schwankungen bei der Z-Achse und Y-Achse sind auf Fahrbahnunebenheiten zurückzuführen.

Sektor B.5

Getriebe des Kraftfahrzeugs wird in den 2. Gang geschaltet.

Erfasster Zeitraum der Daten ca. 3 s.

Mittelwert der Beschleunigung bei 1,92 m/s^2.

Schalten in den 3. Gang bei ca. 40 km/h.

Analyse: Die Schwingung um den Mittelwert nimmt bei allen 3 Achsen zu. Fahrbahnunebenheiten machen sich bei steigender Geschwindigkeit bei der Z-Achse und Y-Achse durch Ausschläge bemerkbar.

Sektor B.6

Getriebe des Kraftfahrzeugs wird in den 3. Gang geschaltet.

Zeitraum der Datenerfassung ca. 4 s.

Mittelwert der Beschleunigung bei 1,38 m/s^2.

Schalten in den 4. Gang bei ca. 60 km/h.

Analyse: Keine weiteren Besonderheiten im Vergleich zu Sektor B.5.

Sektor B.7

Getriebe des Kraftfahrzeugs wird in den 4. Gang geschaltet.

Zeitraum der Datenerfassung ca. 12 s.

Mittelwert der Beschleunigung 0,18 m/s^2.

Endgeschwindigkeit 60 km/h.

Analyse: Keine weiteren Besonderheiten im Vergleich zu Sektor B.5.

Sektor B.8

Langsamer, normaler Bremsvorgang von 60 km/h auf 0 km/h.

Zeitraum der Datenerfassung ca. 11 s.

Mittelwert der Beschleunigung bei –1,96 m/s^2.

Sektor B.9

Antriebsaggregat ist aktiviert, Kraftfahrzeug befindet sich im Stillstand.

Zeitraum der Datenerfassung ca. 10 s.

Mittelwert der Beschleunigung bei -0,12 m/s^2.

Sektor B.10

Schnelle Rückwärtsfahrt mit 90° Einparkung.

Zeitraum der Datenerfassung ca. 5 s.

Hierbei ca. 3 s Rückwärtsbeschleunigung.

Mittelwert der Beschleunigung bei -1,84 m/s^2.

90° Einlenkung und ca. 2 s Bremsvorgang.

Mittelwert der Beschleunigung bei 2,7 m/s^2.

Sektor B.11

Antriebsaggregat ist aktiviert, Kraftfahrzeug befindet sich im Stillstand.

Zeitraum der Datenerfassung ca. 1 s.

Mittelwert der Beschleunigung bei 0,19 m/s^2.

Analyse: Der Anstieg des Mittelwerts am Ende der Messung ist auf die leichte Fahrbahnansteigung zurückzuführen. Des Weiteren ist eine minimale Positionsänderung des Beschleunigungs-Datenloggers nicht auszuschließen, da dieser nicht starr mit dem Fahrzeugchassis verbunden wurde.

Auswertung:

Bei Addition der gesamten positiven Beschleunigungen über deren Zeiträume, errechnet sich eine Geschwindigkeit von:

$$v = 4\,\mathrm{s} \cdot 2,34\,\frac{\mathrm{m}}{\mathrm{s}^2} + 3\,\mathrm{s} \cdot 1,92\,\frac{\mathrm{m}}{\mathrm{s}^2} + 4\,\mathrm{s} \cdot 1,38\,\frac{\mathrm{m}}{\mathrm{s}^2} + 3\,\mathrm{s} \cdot 0,18\,\frac{\mathrm{m}}{\mathrm{s}^2} = 21,18\,\frac{\mathrm{m}}{\mathrm{s}}$$

Wenn die gesamte negative Beschleunigung über deren Zeitraume addiert wird, ergibt sich eine Geschwindigkeit von:

$$v = 11\,\mathrm{s} \cdot 1,96\,\frac{\mathrm{m}}{\mathrm{s}^2} = 21,56\,\frac{\mathrm{m}}{\mathrm{s}}$$

Die Abweichung beträgt:

$$21,56\,\frac{\mathrm{m}}{\mathrm{s}} - 21,18\,\frac{\mathrm{m}}{\mathrm{s}} = 0,38\,\frac{\mathrm{m}}{\mathrm{s}}$$

Die in dem Test ermittelte Abweichung von 0,38 m/s^2 hat, wie bei dem Feldversuch 1 beschrieben, ihre Ursache in der Positionierung des Beschleunigungs-Datenloggers. Da die Fahrzeugneigung bei langsamem Bremsenvorgang geringer ist, ist hier die Abweichung kleiner. Für spätere Messungen mit dem Beschleunigungs-Datenlogger empfiehlt es sich, diesen möglichst mittig zwischen die Achsen zu positionieren. Da diese Möglichkeit bei dem verwendeten Kraftfahrzeug nicht bestand, wurden hier die positiven Beschleunigungen über den Zeitraum mit den negativen Beschleunigungen über den Zeitraum gegenübergestellt und aus beiden der Mittelwert gebildet.

Hieraus ergibt sich ein Mittelwert von:

$$\frac{(21,56\ \frac{\mathrm{m}}{\mathrm{s}} + 21,18\ \frac{\mathrm{m}}{\mathrm{s}})}{2} = 21,37\ \frac{\mathrm{m}}{\mathrm{s}}$$

Der Mittelwert dividiert durch die Bremszeit, ergibt dann annähernd die Maximalbeschleunigung beim Bremsvorgang von:

$$a\ = \frac{21,37\ \frac{\mathrm{m}}{\mathrm{s}}}{11\ \mathrm{s}} \approx 1,94\ \frac{\mathrm{m}}{\mathrm{s}^2}$$

Nach der Formel (2.1.1) ergibt sich hiernach eine Kraft von:

$$F = 1,94\ \frac{\mathrm{m}}{\mathrm{s}^2} \cdot 905\ \mathrm{kg} = 1,758\ \mathrm{kN}$$

Der hierbei zurückgelegte Bremsweg beträgt nach der Formel (2.1.3) eine Strecke von:

$$v = 60\ \frac{\mathrm{km}}{\mathrm{h}} \approx 16,7\ \frac{\mathrm{m}}{\mathrm{s}}$$

$$s = \frac{(16,7\ \frac{\mathrm{m}}{\mathrm{s}})^2}{2 \cdot\ 1,94\ \frac{\mathrm{m}}{\mathrm{s}^2}} \approx 72\ \mathrm{m}$$

Hieraus ergibt sich eine kinetische Energie von:

$$E_{kin} = 1{,}758\ \mathrm{kN} \cdot 72\ \mathrm{m}\ \approx 126{,}5\ \mathrm{kJ}$$

Als Vergleichsrechnung mit der Formel (2.1.4) ergibt sich für des Kraftfahrzeug eine kinetische Energie von:

$$E_{kin} = \frac{1}{2} \cdot 905\ \mathrm{kg} \cdot 16,7^2\ \left(\frac{\mathrm{m}}{\mathrm{s}}\right)^2 \approx 126,2\ \mathrm{kJ}$$

Die kinetische Energie, die aus den Messwerten errechnet wurde, ist fast mit dem in der Vergleichsrechnung rein rechnerisch ermittelten Wert identisch. Die Abweichung beträgt 0,3 kJ und ist auf Messungenauigkeit und Rundungsfehler zurückzuführen.
Der Beschleunigungs-Datenlogger hat im Feldversuch 2, genauso wie im Feldversuch 1, erfolgreich Änderungen von Beschleunigungen erfasst und gespeichert, die gut auswertbar sind.
Die bei der Auswertung ermittelten Werte sind plausibel und der Versuch hat die Funktionsfähigkeit des Beschleunigungsdatenloggers somit bestätigt.
Die erfolgreichen Feldversuche, bei dem gut auszuwertende und plausible Messdaten erfasst wurden, belegen, dass der Beschleunigungs-Datenlogger funktionsfähig ist.
Eine Reproduzierbarkeit der Messdaten ist gegeben. Dies wurde durch weitere Tests unter ähnlichem Fahrverhalten durch eine nahezu identische Grafik bewiesen.

Anmerkung:
Bei Messungen über einen längeren Zeitraum unter Laborbedingungen, sind bei der Datenspeicherung Fehler aufgetreten. Hier traten bei der Verwendung von einem täglich angelegten File, Fehler bei einer Tabellenlänge größer als 50000 Zeilen auf.
Bei einem alle 10 Minuten neu angelegtem File traten sporadisch Fehler nach mehrstündigem Betrieb bei der Fileanlegung auf.
Langzeittests, bei dem der Beschleunigungs-Datenlogger über einen längeren Zeitraum in einem Kraftfahrzeug Beschleunigungsdaten erfasst, wurden aus zeitlichen Gründen nicht durchgeführt.
Des Weiteren wurde der Beschleunigungs-Datenlogger nicht mit einer externen Energiequelle getestet.

Kapitel 7

7 Fazit und Ausblick

Dieses Kapitel beinhaltet eine kurze Zusammenfassung mit getroffenen Verbesserungsansätzen sowie das Fazit und eventuelle zukünftige Verwendungsmöglichkeiten für den Beschleunigungs-Datenlogger.

7.1 Zusammenfassung

Der Beschleunigungs-Datenlogger ist in seiner Grundfunktion, für das Erfassens von Beschleunigungsdaten und für deren Abspeicherung mit einem versehenen Zeitstempel voll funktionsfähig. Hierbei wurden bei Verwendung von einer 8 GB Klasse 10 Speicherkarte ca. 46 Messdatensätze pro Sekunde und bei der Verwendung von einer 8 GB Klasse 4 Speicherkarte ca. 20 Messdatensätze pro Sekunde abgespeichert. Die in den Feldversuchen erhobenen Datensätze ließen sich in Bezug auf veränderte Beschleunigungen für ein Kraftfahrzeug gut auswerten. Die Zusatzfunktion, die das Erfassen von GPS-Koordinaten vorsieht, ist ebenfalls voll funktionsfähig. Hierdurch sinkt jedoch die Abspeicherungsrate der Messdatensätze auf ca. 4 pro Sekunde ab, da für jede weitere Datenerfassung und die damit verbundene Datenverarbeitung der Programmdurchlauf verlängert wird. Das Programm kann somit nicht mehr so schnell durchlaufen werden, wodurch auch der Speichervorgang reduziert wird. Die Zusatzfunktion der Temperaturerfassung wurde aus zeitlichen Gründen softwaretechnisch nicht mehr realisiert. Hier wurde lediglich die Anpassungsrechnung für das zu erwartende Bitmuster der Temperatursensoren realisiert. Um einen Platzhalter für die Temperaturdaten in der Messdatentabelle zu schaffen, wurde hier mit einem festen Temperaturwert gerechnet. Des Weiteren konnten aus zeitlichen Gründen ein Platinenredisign der 3D-Acceleration-Booster-Patine, auf dem der Fehler einer vertauschten Datenleiterbahn des Mikro-SD-Sockets behoben wird, nicht durchgeführt werden, wodurch das Speichern auf einer alternativen Mirko-SD-Karte nicht benutzt werden kann.

Die autonome Energieversorgung ist durch den Sekundär-Energiespeicher oder durch den Primär-Energiespeicher mehr als ausreichend für einen Betrieb von 8 h ausgelegt.

Dasselbe trifft auch auf die gewählte 8 GB Speicherkarte zu, die weit über eine Messdatenerfassungszeit von 168 h liegt.

Für ein kontrolliertes Programmende steht durch den Capacitor-Tower ausreichend Energie zur Verfügung.

7.2 Verbesserungsansatz

Bei der Entwicklung der 3D-Acceleration-Booster-Platine sind im Nachhinein zwei Fehler beim Platinen-Layout lokalisiert worden. Der eine Fehler erfolgte bei der Dimensionierung für den Platz des Mikro-SD-Kartenhalter-Sockets, um diesen dennoch verwenden zu können, wurden für den Beschleunigungs-Datenlogger-Prototyp die Mikro-SD-Kartenhalter-Socket-Pins verlängert und der Mikro-SD-Kartenhalter-Socket wurde zusätzlich durch Heißkleber fixiert. Der zweite Fehler ist eine vertauschte Datenleiterbahn für den Mikro-SD-Kartenhalter-Socket, wodurch dieser nicht verwendet werden kann. Dieser Fehler muss, um den Mikro-SD-Kartenhalter-Socket verwenden zu können, bei der nächsten Platinen-Version behoben werden. Ein Redesign der Platine mit neu dimensioniertem Platz für den Mikro-SD-Kartenhalter-Socket befindet sich im Anhang und auf der Zusatz-DVD im Ordner (Eagle/ 3D-Acceleration-Booster V.1.1). Dieses sollte aber vor der Platinenanfertigung genau geprüft werden.

Um die Möglichkeit der Abspeicherung von Temperaturdaten nutzen zu können, muss hierfür die Abfrage der zwei Temperatursensoren softwaretechnisch realisiert werden.

Bei der Verwendung einer externen Energieversorgung sollte, wenn diese z.B. eine Autobatterie ist, die hohe Spannungsspitzen aufweisen kann, zum Schutz des Beschleunigungs-Datenloggers und seinem Präzisionsspannungsregler, eine geeignete Suppressordiode[2] nachgerüstet werden.

In dem Beschleunigungs-Datenlogger entwickelten Prototyp wurde für den GPS-Empfänger ein Empfangsmodul mit integrierter Antenne verwendet. Bei Verwendung der Metallabdeckung oder durch eine ungünstige Positionierung des Beschleunigungs-Datenloggers, kann dadurch kein GPS-Signal empfangen werden. Als Verbesserung für nächste Beschleunigungs-Datenlogger-Generationen sollte hier ein GPS-Empfangsmodul verwendet werden, das über eine externe Antenne verfügt.

Des Weiteren können noch softwaretechnische Verbesserungen vorgenommen werden, wie z.B. die Verwendung von Datenzwischenspeicherungen in das RAM des Mikrocontrollers und eine nicht zyklische Abfrage von Sensoren und GPS-Empfangsmodul, sondern eine durch Interrupt gesteuerte Abfrage, die eine Datenspeicherrate pro Sekunde optimiert.

---

[2] Schutzdiode gegen Überspannung die beim Durchbrennen eine Masseschluss verursacht

Da bei Testmessungen über mehrstündige Zeiträume sporadisch Fehler bei der Fileanlegung auftraten, empfiehlt es sich die Software so anzupassen, dass ein vorher existierendes File überschrieben wird.

7.3 Fazit

Der Beschleunigungs-Datenlogger ist in der Lage, durch seine autonome Energieversorgung flexibel in einem Kraftfahrzeug positioniert zu werden. Hier muss lediglich für eine Fixierungsmöglichkeit gesorgt werden. Die durchgeführten Tests waren reproduzierbar und die dabei erhobenen Daten waren gut auswertbar und plausibel, wodurch die Funktionsfähigkeit des Beschleunigungs-Datenloggers bewiesen ist.

7.4 Ausblick

Durch die zusätzliche Erfassung von GPS-Koordinaten kann der Beschleunigungs-Datenlogger nicht nur zur Analyse zwecks Rekuperation verwendet werden, sondern auch für eine Analyse des Fahrverhaltens und des Verkehrsaufkommens genutzt werden.

Des Weiteren kann durch die zusätzlichen Anschlussmöglichkeiten am 3D-Acceleration-Booster, die für eine Komponentenerweiterung vorgesehen sind, Komponenten nachgerüstet werden. Hier könnte beispielsweise die Nachrüstung eines GSM-Empfängers erfolgen, wodurch eine Kommunikation zwischen Beschleunigungs-Datenlogger und Mobiltelefon zur Datenübertragung realisiert werden kann.

Durch die unbestückten Header-Pin-Reihen des Stellaris LM4F120 LaunchPad Evaluation Kit können hier Erweiterungen aufgesteckt werden. Eine solche Erweiterung könnte beispielsweise ein LC-Display, wie das Stellaris LaunchPad LCD Booster-Pack EB-LM4F120-L35 von KENTEC sein. Hierfür müssten dann aber software- und hardwareseitige Anpassungen vorgenommen werden, da es hier, bei gleichzeitiger Pin-Verwendung, durch den 3D-Acceleration-Booster auf der Unterseite zu Funktionsproblemen des Beschleunigungs-Datenloggers führet.

Dadurch, dass die 3D-Acceleration-Booster Platine dieselbe Größe hat wie das Stellaris LM4F120 LaunchPad Evaluation Kit, können diese als eine kompakte kleine Einheit für andere Projekte betrieben werden, beispielsweise die Beschleunigungsdatenerfassung an Exoprothesen.

8 Verzeichnis der Abkürzungen

Formelzeichen und Symbole

A	Fläche $[m^2]$
a	Beschleunigung $\left[\frac{m}{s^2}\right]$
B_{Zeit}	Betriebszeit [h]
C	Kapazität [F]
E_{elek}	elektrische Energie [Wh]
E_{kin}	kinetische Energie [J]
F	Kraft [N]
I	Strom [A]
m	Masse [kg]
Q	Ladung [As]
R	Widerstand [Ω]
s	Weg [m]
t	Zeit [s]
U	Spannung [V]
v	Geschwindigkeit $\left[\frac{m}{s}\right]$
Δ(t)	Zeitdifferenz [s]

Abkürzungen

CAN	Controller Area Network
CCS	Code Composer Studio
CS	Chip Select
CSV	Comma Separated Values
EAGLE	Einfach Anzuwendender Grafischer Layout-Editor
FAT	File Allocation Table
GGA	Fix information
GPS	Global Positioning System
GSA	Overall Satellite Data
GSM	Global System for Mobile Communications
I^2C	Inter Integrated Circuit
MEMS	Micro Electro Mechanical Systems
MISO	Master In Slave Out
MOSI	Master Out Slave In
NMEA	National Marine Electronic Association
PHEV	Plug-In Hybrid Electric Vehicle
RAM	Random Access Memory
RMC	Recommended Minimum Sentence C
SCL	Serial Clock
SCK	Serial Clock
SDA	Serial Data
SDI	Serial Data In
SDO	Serial Data Out
SPI	Serial Peripheral Interface
SS	Slave Select
SSI	Synchronous Serial Interface
STE	Slave Transmit Enable
TWI	Two Wire Interface
UART	Universal Asynchronous Receiver Transmitter
VTG	Vector track and Speed over the Ground

9 Abbildungsverzeichnis

10 Tabellenverzeichnis

11 Literaturverzeichnis

[1] Hochbahn (Hrsg.): Foto, Plug In-Hybridbus, http://www.nahverkehrhamburg.de/busverkehr-hamburg/item/1150-hochbahn-und-volvo-starten-entwicklungspartnerschaft-fuer-e-busse –Abruf: 12.07.2014

[2] Volvo (Hrsg.): Foto, Plug In-Hybridbus, http://www.vdi-nachrichten.com/Archiv/Bild/316382/316312 –Abruf: 12.07.2014

[3] Klaus Betke (Hrsg.):, *The NMEA 0183 Protocol*, Version: 2001, http://fort21.ru/download/NMEAdescription.pdf –Abruf: 05.09.2014

[4] ADAC (Hrsg.): ADACsignale, *TIPPS FÜR DIE PRAXIS-UNTERRICHTSBEISPIEL* http://www.adac.de/_mmm/pdf/Verkehr_und_Mathe_Anhalteweg_45164.pdf –Abruf: 18.08.2014

[5] Texas Instruments (Hrsg.): *Stellaris LM4F120 LaunchPad Evaluation Board User Manual*, Version: 2013, http://www.ti.com/lit/ug/spmu289c/spmu289c.pdf –Abruf: 02.03.2014

[6] Bosch Sensortec (Hrsg.): *BMA020 Digital, triaxial acceleration sensor Data sheet*, Version: 2008, http://www.farnell.com/datasheets/1525403.pdf –Abruf: 02.03.2014

[7] ELV (Hrsg.): *Bau- und Bedienungsanleitung 3-Achsen-Beschleunigungssensor-Modul mit SPI- und I2C- Schnittstelle*, Version: 2010, http://www.emteka.de/WebRoot/StoreLDE/Shops/62398537/4C7B/4E29/455B/848B/EF86/C0A8/29BB/54C5/3D-BS_KM_G_100316.pdf –Abruf: 02.03.2014

[8] SanDisk (Hrsg.): *SanDisk Secure Digital Card,* Version: 2003.
http://www.circlemud.org/jelson/sdcard/SDCardStandardv1.9.pdf
–Abruf: 09.03.2014

[9] Hygrosens Instruments GmbH (Hrsg.): *TSICTM 206 Digitale Halbleiter- Temperatur-sensoren*, Version 2008.
https://cdn-reichelt.de/documents/datenblatt/B400/TSIC_Sensoren_dbd.pdf
–Abruf 03.03.2014

[10] Dallas Semiconductor (Hrsg.): *DS1307 64X8 Serial Real Time Clock*,
https://cdn-reichelt.de/documents/datenblatt/A200/DS_1307.pdf
–Abruf 03.03.2014

[11] Micro Crystal Switzerland (Hrsg.): *CC5V-T1A Tuning Fork Crystal 32.768 kHz*, Version: 2007, https://cdn-reichelt.de/documents/datenblatt/B400/CC5V-T1A.pdf
–Abruf: 14.03.2014

[12] Navilock (Hrsg.): *EM-406A GPS RECEIVER ENGIN BOARD*, Version: 2007,
http://www.navilock.de/produkte/F_1144_PPS_60407/dokumente.html
–Abruf: 18.08.2014

[13] Reichelt elektronik GmbH (Hrsg.): *Mobil-Akku VTB-28*, Version: 2012,
https://cdn-reichelt.de/documents/datenblatt/D600/VTB-28.pdf
–Abruf: 22.03.2014

[14] Panasonic (Hrsg.): *Electric Double Layer Capacitors/NF*,
https://cdn-reichelt.de/documents/datenblatt/B300/SPK1%2C0_SPK10_SPK22_SPK47_PAN.pdf
–Abruf: 22.03.2014

[15] Vishay Semiconductors (Hrsg.): Schottky Diode, Version: 2002
https://cdn-reichelt.de/documents/datenblatt/A400/BAT86_VIS.pdf
–Abruf: 05.09.2014

[16] Linear Technology (Hrsg.): LT1086 Series,
https://cdn-reichelt.de/documents/datenblatt/A200/LT1086%23LT.pdf
–Abruf: 07.09.2014

[17] Texas Instruments (Hrsg.): Softwareparket, StellarisWare, *SW-LM3S-LM4F*, Version: 2012, http://www.ti.com/tool/sw-lm3s
–Abruf: 19.05.2014

[18] Jonathan W. Valvano, *Real-Time Interfacing to ARM CortexTM –M Microcontrollers Embedded Systems*, Amazon Distribution GmbH, Leipzig 2013

[19] Bert von Dam, *ARM Mikrocontroller*, Elektor- Verlag GmbH, Aachen 2012

[20] Helmut Erlenköter, *C Programmieren von Anfang an*, Rowohlt Taschenbuch Verlag: GmbH, Reinbek bei Hamburg 2007

[21] AVR FAT32, Softwareparket, *AVR-mmc-0.6.4*,
http://www.mikrocontroller.net/articles/AVR_FAT32#Der_Status
–Abruf: 05.04.2014

[22] Texas Instruments (Hrsg.): *Stellaris LM4F120H5QR Microcontroller Data Sheet*. Version: 2013.
http://www.mouser.com/ds/2/405/lm4f120h5qr-124014.pdf
–Abruf: 02.03.2014

[23] SD Association (Hrsg.): *SD Card Specification*, Version: 2001
https://www.sdcard.org/downloads/pls/simplified_specs/archive/partE1_100.pdf
–Abruf: 18.08.2014

12 DVD Beschleunigungs-Datenlogger

Ordner	Aufgabenstellung
Ordner	Bachelorthesis
Ordner	Bilder
Ordner	Datenblätter
Ordner	Eagle
Ordner	Quellcode
Ordner	Tabellen

13 Anhang

13.1 Aufgabenstellung

Version: 1.1 Datum: 09.06.2014

Vergabe einer Bachelorarbeit:

Erstellung eines Beschleunigungs-Datenlogger für Kraftfahrzeuge

Hintergrund

Kraftfahrzeuge mit Elektroantrieb ermöglichen die Rückgewinnung von Energie beim Abbremsen des Fahrzeugs z.B. vor Ampeln oder bei Abwärtsfahrten. Wird diese Energie gespeichert, steht sie für eine spätere Beschleunigung als Unterstützung zur Verfügung. Als Vorbereitung für Entwicklungen auf diesem Gebiet soll eine Analyse des typischen Fahrverhaltes und der Möglichkeiten zur Energieeinsparung erfolgen. Dazu soll der, in dieser Bachelorarbeit zu entwickelnde Beschleunigungs-Datenlogger dienen.

Obligatorischer Inhalt der Bachelorarbeit

Es soll ein Beschleunigungs-Datenlogger, kurz BDL, entwickelt und hergestellt werden.

- Umsetzung mit einem ARM-Microcontroller, Programmierung mit C und geeigneten Bibliotheken (z.B. Stellaris von Texas Instruments)
- Verwendung eines drei Achsen Beschleunigungssensors
- Die Daten werden auf einem SD-Speicherchip in geeignetem Format gespeichert
- Realisierung von Echtzeiterfassung und Datum
- Autonome Stromversorgung durch Batterie
- Kontrolliertes Beenden des Programms bei Abschalten der Betriebsspannung

In einer ersten Phase wird der BDL spezifiziert und Komponenten werden ausgewählt. Die Datenmenge und die Datenrate soll abgeschätzt werden. Danach wird die Platine entworfen, die Komponenten bestellt, die Platine bestückt und getestet. Der BDL soll in ein solides Gehäuse integriert werden. Abschließend soll die grundsätzliche Funktionsfähigkeit durch Erfassung von Testdaten nachgewiesen werden. Die Datenauswertung soll exemplarisch grafisch durch MatLab dargestellt werden.

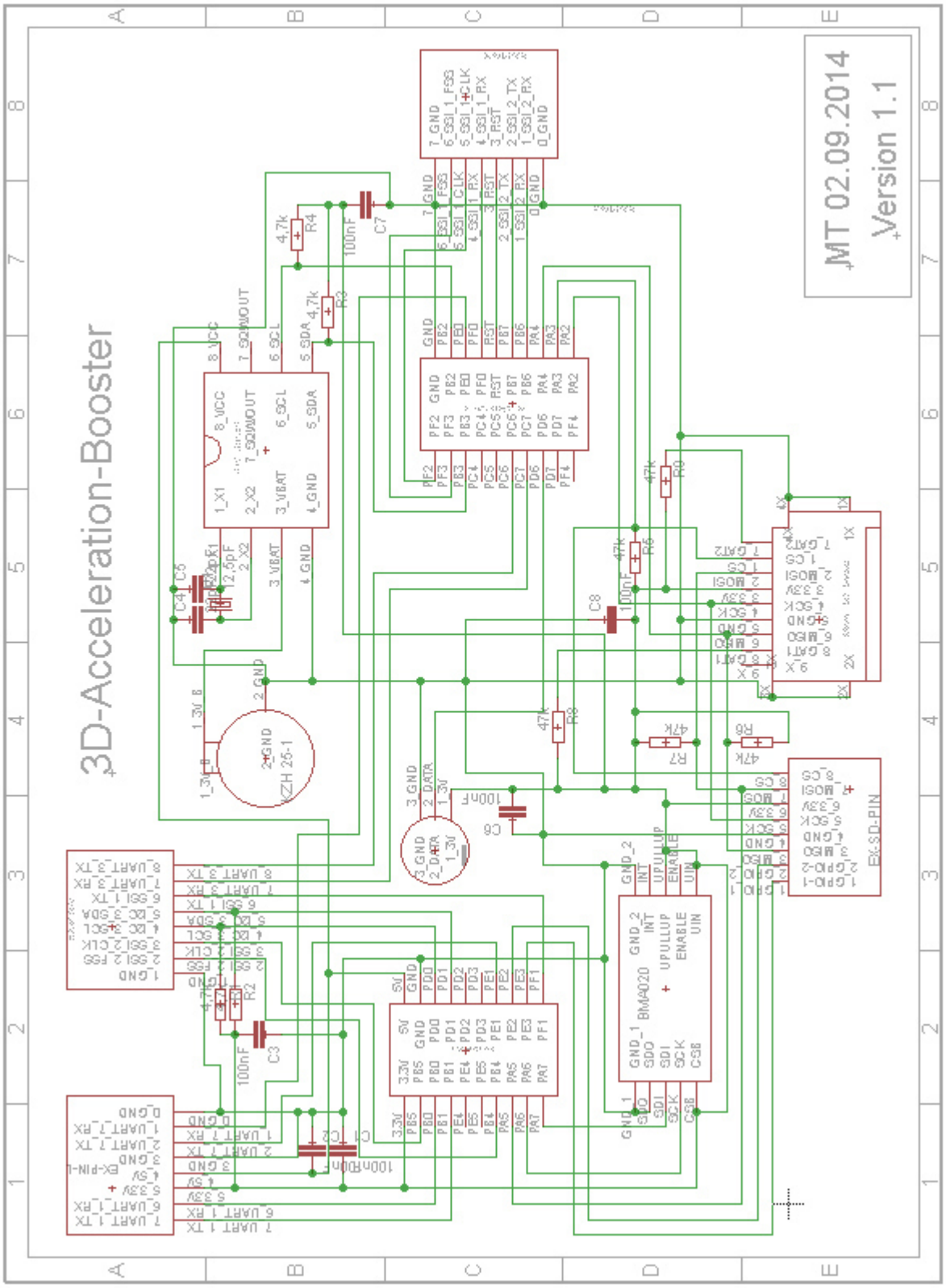
Optionale Inhalte der Bachelorarbeit

- Integration eines GPS-Moduls mit entsprechender Speicherung der Position
- Erfassung von externer und interner Temperatur des BDLs und Speicherung
- Stromversorgung durch Primärbatterien
- Testen des fertigen BDLs in einem Kraftfahrzeug
- Statistische Auswertung der gespeicherten Daten

Rahmenbedingungen

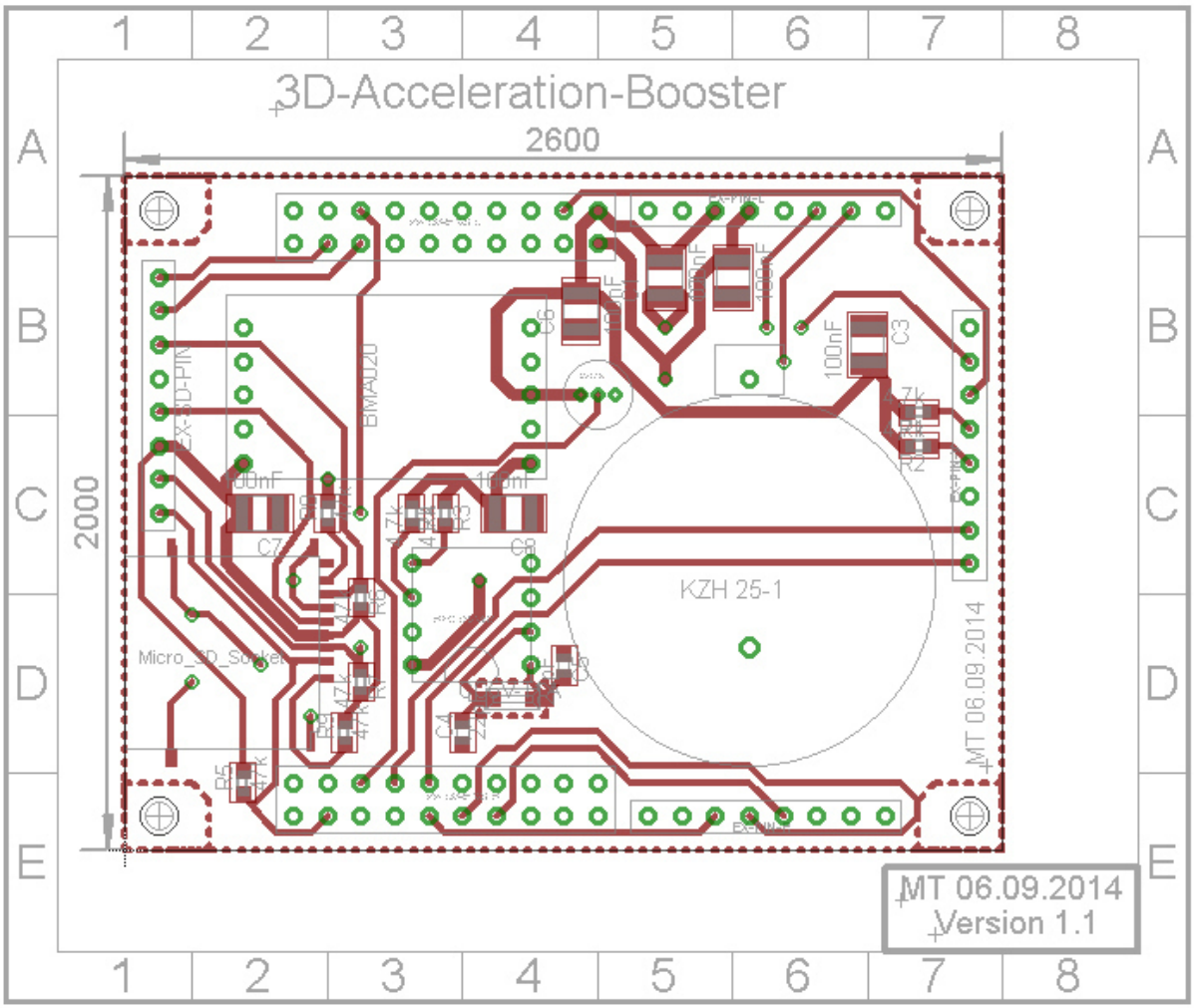
- Die Arbeit wird an der HAW in den Laboren von Prof. Riemschneider erstellt
- Betreuer: Prof. Heß (Betreuung) und Prof. Riemschneider (Zweitgutachter)

13.2 3D-Acceleration-Booster-Schaltplan

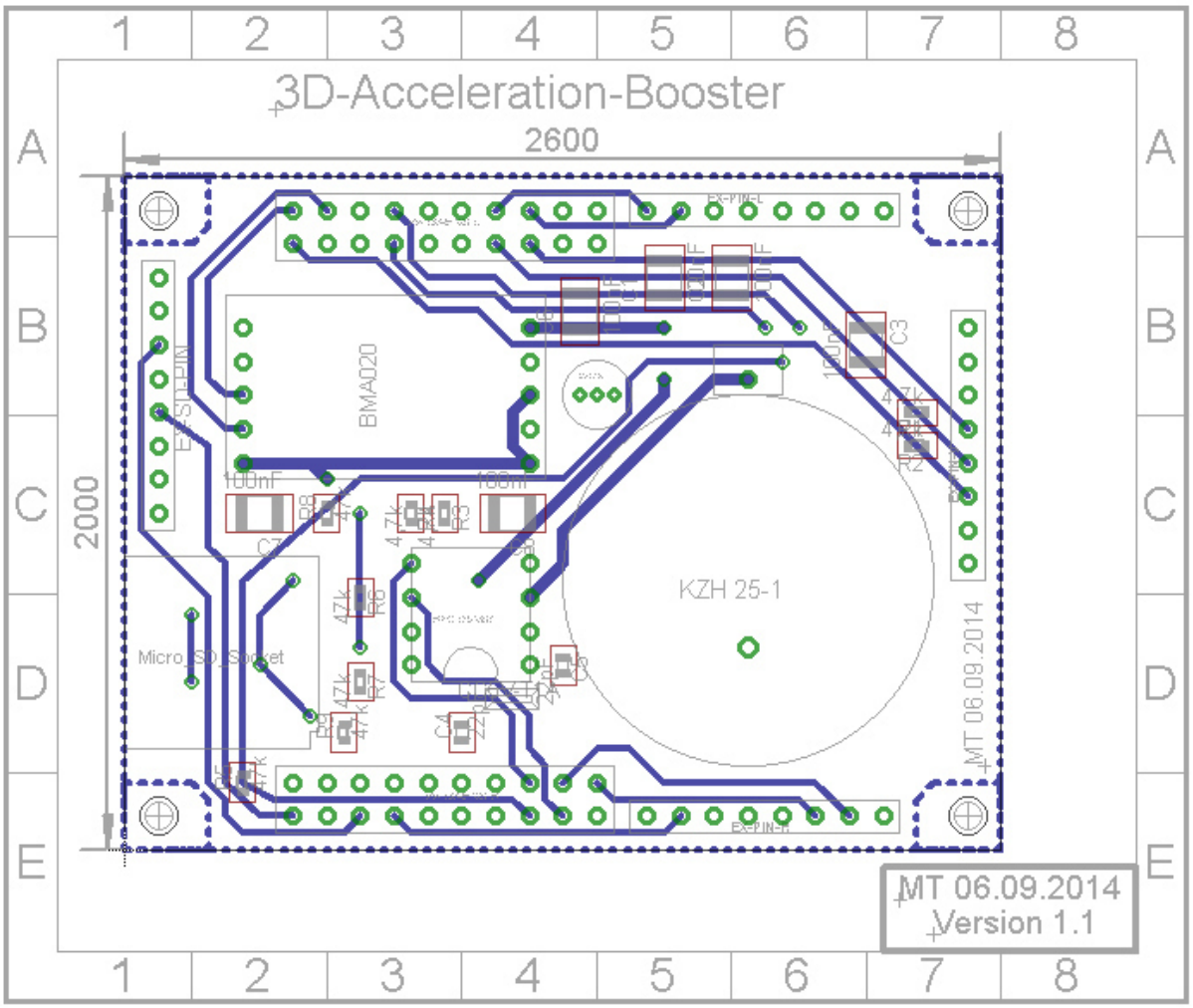


Schaltplan 3D-Acceleration-Booster Version 1.1

13.3 3D-Acceleration-Booster-Platinen-Layouts

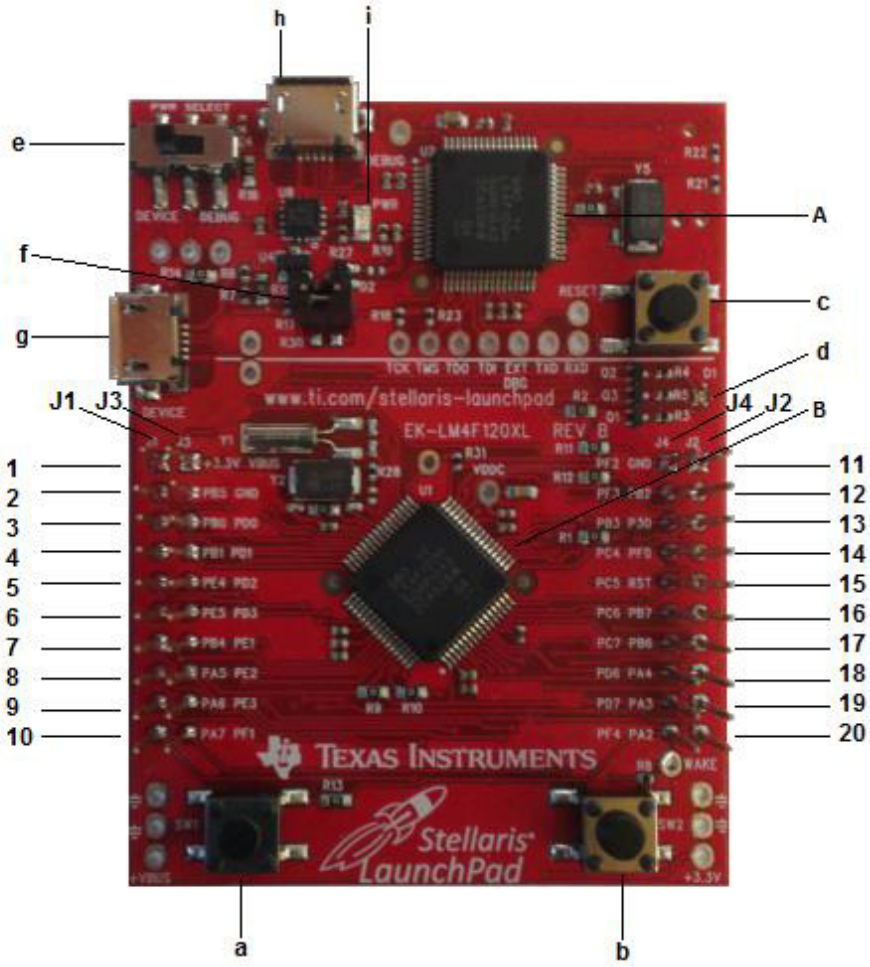


3D-Acceleration-Booster Platinen-Layout Oberseite Version 1.1



3D-Acceleration-Booster Platinen-Layout Unterseite Version 1.1

13.4 Stellaris LM4F120 LaunchPad Evaluation Kit



Stellaris LM4F120 LaunchPad Evaluation Kit Frontansicht

Stellaris LM4F120 LaunchPad Evaluation Kit Frontansicht	
A	Stellaris LM4F120H5QR Mikrocontroller 1 zum Programmieren des Mikrocontroller 2
B	Stellaris LM4F120H5QR Mikrocontroller 2
a	Benutzer Switch 1
b	Benutzer Switch 2
c	Reset Switch zum Löschen der Benutzer Switchs 1 und 2
d	RGB Nutzer LED
e	Power Select Switch
f	Jumper
g	Mikro USB Anschluss (Device)
h	Mikro USB Anschluss Power In-Circuit Debug Interface (ICDI)
i	Grüne power LED

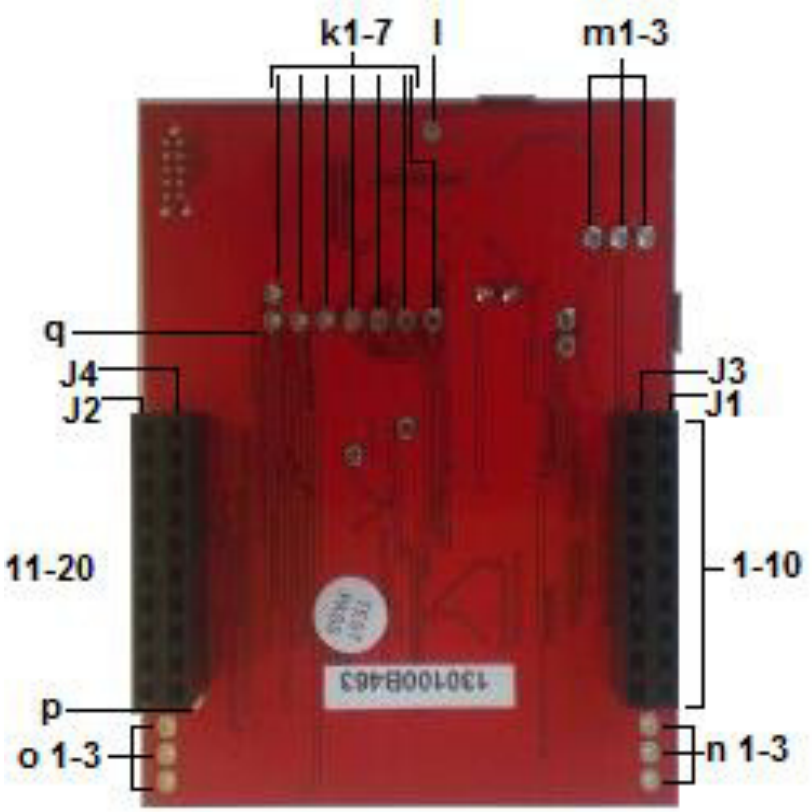
Stellaris LM4F120 LaunchPad Evaluation Kit Frontansicht

Stellaris LM4F120 LaunchPad Evaluation Kit Pin-Belegung links			
1	J1	3.3V	3.3V
	J3	VBUS	5V
2	J1	PB5	SSI 2 Fss
	J3	GND	
3	J1	PB0	UART 1 RX
	J3	PD0	I2C 3 SCL / SSI 1 und 3 CLK
4	J1	PB1	UART 1 TX
	J3	PD1	I2C 3 SDA / SSI 1 und 3 FSS
5	J1	PE4	I2C 2 SCL / UART 5 RX / CAN 0 RX
	J3	PD2	SSI 1 und 3 RX
6	J1	PE5	I2C 2 SDA / UART 5 TX / CAN 0 TX
	J3	PD3	SSI 1 und 3 TX
7	J1	PB4	SSI 2 CLK / CAN 0 RX
	J3	PE1	UART 7
8	J1	PA5	SSI 0 TX
	J3	PE2	GPIO (AIN 1)
9	J1	PA6	I2C 1 SCL
	J3	PE3	GPIO (AIN 0)
10	J1	PA7	I2C 1 SDA
	J3	PF1	SSI 1 TX

Stellaris LM4F120 LaunchPad Evaluation Kit Pin-Belegung links

Stellaris LM4F120 LaunchPad Evaluation Kit Pin-Belegung rechts			
11	J2	GND	
	J4	PF2	SSI 1 CLK
12	J2	PB2	I2C 0 SCL
	J4	PF3	SSI 1 Fss / CAN 0 TX
13	J2	PE0	UART 7 RX
	J4	PB3	I2C 0 SDA
14	J2	PF0	SSI 1 RX
	J4	PC 4	UART 4 RX / UART 1 RX
15	J2	RST	
	J4	PC5	UART 4 TX / UART 1 TX
16	J2	PB7	SSI 2 TX
	J4	PC6	UART 3 RX
17	J2	PB 6	SSI 2 RX
	J4	PC7	UART 3 TX
18	J2	PA4	SSI 0 RX
	J4	PD6	UART 2 RX
19	J2	PA3	SSI 0 Fss
	J4	PD7	UART 2 TX
20	J2	PA2	SSI 0 CLK
	J4	PF4	GPIO

Stellaris LM4F120 LaunchPad Evaluation Kit Pin-Belegung rechts



Stellaris LM4F120 LaunchPad Evaluation Kit Rückansicht

Stellaris LM4F120 LaunchPad Evaluation Kit Rückansicht	
k1	Reset
k2	VCP_TXD PA1 UART Module 0 receive
k3	EXTDBG PA7 I2C1 SDA
k4	ICDI_TDI (ICDI JTAG Pin 8) PC2 PWM 0
k5	ICDI_TDO (ICDI JTAG Pin 6) PC3 SWO
k6	ICDI_TMS (ICDI JTAG Pin 2) PC1 SWDIO
k7	ICDI_TCK (ICDI JTAG Pin 4) PC0 SWD CLK
l	DEBUGG
m1	Power Select Switch Modus In-Circuit Debug Interface (ICDI)
m2	Power Select Switch Modus aus
m3	Power Select Switch Modus DEVIC
n1	GND
n2	GND
n3	VBUS 5V
o1	GND
o2	GND
o3	3.3V
p	WAKE
q	VCP_RXD PA0 UART Module 0 transmit

Stellaris LM4F120 LaunchPad Evaluation Kit Rückansicht

Achtung: Die Pins des Stellaris LM4F120H5QR LaunchPad Evaluation Kit sind teilweise multifunktional. Die Angaben in dieser Arbeit beziehen sich hauptsächlich auf die für den Beschleunigungs-Datenlogger relevante Pin-Belegungen. Die vollständigen multifunktionalen Pin-Belegungen sind im Herstellerdatenblatt [19] verfügbar.

13.5 Quellcode

Hauptroutine (C-File)

```
1
2  /*****************************************************************************
3  ******************************************************************************
4  *
5  * File:                main.c
6  *
7  * Created on:          13.06.2014
8  * Author:              Marc Thom
9  * Bachelorthesis:      Hard- und Softwareentwicklung von einem Beschleunigungs-
10 *                      Datenlogger für Kraftfahrzeuge
11 *
12 *
13 * Primär:              unter IAR Embedded 7.10.3      entwickelt und getestet
14 * Sekunder:            unter Code Composert Studio 5.5.0   getestet
15 *
16 * Hardware:            Stellaris LM4F120 LaunchPad Evaluation Kit,
17 *                      Miko-SD-Karte FAT 32, SD-Karte FAT 16, SD-Karte FAT 32,
18 *                      BMA020, RTC, TSIC und GPS-Empfänger
19 *
20 * Anmerkung:           Die für den Mikrocontroller verwendeten Funktionen zu
21 *                      dessen Konfikurierung und Periferieverwendungen
22 *                      sind Standartfunktionen von Texas Instruments!
23 *                      Das, für die Funktionen, benötigte
24 *                      Softwarepaket ist das StellarisWare: SW-LM3S-LM4F
25 *                      und wurde unter http://www.ti.com/tool/sw-lm3s
26 *                      bezogen. Letzter Zugriff 19.05.2014
27 *
28 *                      Die für die SD-Karte oder Mikro-SD-Karte verwendeten
29 *                      Funktionen zu deren Konfikurierung oder Datenbeschreibung
30 *                      sind von Daniel R.und wurden unter
31 *                      http://www.mikrocontroller.net/articles/AVR_FAT32#Der_Status
32 *                      bezogen. Letzter Zugriff 2014/04/05
33 *
34 ******************************************************************************
35 *****************************************************************************/
36
37 //----------Eigene Header LED, BDL und ADC----------
38 #include "booster/LED.h"
39 #include "booster/BDL.h"
40 #include "booster/ADC_Init.h"
41
42 //----------Eigene Header BMA020----------
43 #include "bma/Master_BMA020.h"
44 #include "bma/Slave_BMA020.h"
45
46 //----------Eigene Header RTC----------
47 #include "rtc/Master_RTC.h"
48 #include "rtc/Slave_RTC.h"
49
50 //----------Eigene Header SD Karte----------
51 #include "sd/Mikro_SD_mmc_config.h"
52 #include "sd/Mikro_SD_mmc.h"
53 #include "sd/Mikro_SD_Fat.h"
54 #include "sd/Mikro_SD_File.h"
55 #include "sd/Master_SD_CARD.h"
56
57 //----------Eigene Header TSIC----------
58 #include "tsic/TSIC.h"
59
```

```
60  //----------Eigene Header GPS----------
61  #include "gps/gps.h"
62
63  //----------Standart Header----------
64  #include <stdio.h>
65  #include <string.h>
66  #include <stdlib.h>
67
68  //----------inc----------
69  #include "inc/lm4f120h5qr.h"
70  #include "inc/hw_ssi.h"
71  #include "inc/hw_i2c.h"
72  #include "inc/hw_sysctl.h"
73  #include "inc/hw_memmap.h"
74  #include "inc/hw_types.h"
75  #include "inc/hw_gpio.h"
76  #include "inc/hw_uart.h"
77  #include "inc/hw_ints.h"
78  //#include "inc/hw_adc.h"
79
80  //----------driverlib----------
81  #include "driverlib/gpio.h"
82  #include "driverlib/pin_map.h"
83  #include "driverlib/sysctl.h"
84  #include "driverlib/i2c.h"
85  #include "driverlib/ssi.h"
86  #include "driverlib/uart.h"
87  #include "driverlib/timer.h"
88  #include "driverlib/pwm.h"
89  #include "driverlib/rom.h"
90  #include "driverlib/debug.h"
91  #include "driverlib/fpu.h"
92  #include "driverlib/interrupt.h"
93  //#include "driverlib/adc.h"
94  //#include "driverlib/cpu.h"
95
96  //----------utils----------
97  #include "utils/uartstdio.h"
98
99  // Definition des verwendeten Mikrocontrollers
100 // wird für Header benötigt die für verschiedene Mikrocontroler geschrieben
101 // wurden.
102 #define PART_LM4F120H5QR
103
104 // Arrays für den BMA020
105 uint8_t X_Achse[8]={";-0,0000"};
106 uint8_t Y_Achse[8]={";-0,0000"};
107 uint8_t Z_Achse[8]={";-0,0000"};
108
109 // Arrays für die RTC
110 uint8_t Datum[11]={"00.00.00000"};
111 uint8_t Uhrzeit[9]={"00:00:000"};
112 uint8_t MessZeit[8]={"00000000"};
113
114 // Arrays für die TSIC
115 uint8_t Temp_IN[9]={";000.0c°0"};
116 uint8_t Temp_OUT[10]={";000.0c°0;"};
117
118 // Arrays für den GPS
```

```
119  uint8_t GPS[]={"                                                              "};
120
121  // Arrays für den Test
122  uint8_t TEST[8]={";00,0000"};
123
124  // Array für Filename darf nicht länger als 8 +3 sein!
125  unsigned char Filename[12]= {"0000000.csv"};
126
127  int schleife = 1;
128
129  // Hauptrutine
130  int main()
131  {
132    signed short BMA_achsen_werte_10Bit[3] = {0,0,0};
133    signed short RTC_DATA[7] = {0,0,0,0,0,0,0};
134    signed short Temp[2]= {0,0};
135    char Gps_signal[70];
136    signed short Test_test [1]= {0};
137    //unsigned long ADC_Wert[1]={0};
138
139    ////////////////////////////////////////////////////////////////////////////
140    // Clock setzen für die Lauf-Directory des cristal/oscillator auf 16Mhz
141    ////////////////////////////////////////////////////////////////////////////
142    SysCtlClockSet(SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ
143                   | SYSCTL_OSC_MAIN);
144
145    ////////////////////////////////////////////////////////////////////////////
146    // Interrupt für Brown Out
147    ////////////////////////////////////////////////////////////////////////////
148    SysCtlIntRegister(brown_out);
149    SysCtlIntEnable(SYSCTL_INT_BOR);
150    IntMasterEnable();
151
152    ////////////////////////////////////////////////////////////////////////////
153    // Brown Out Configuration auf Interrupt und nicht Reset
154    ////////////////////////////////////////////////////////////////////////////
155    SysCtlBrownOutConfigSet(SYSCTL_BOR_RESAMPLE, 1);
156
157    ////////////////////////////////////////////////////////////////////////////
158    // Testdurchlauf der LED`s
159    ////////////////////////////////////////////////////////////////////////////
160
161    LED_Rot();
162    LED_Blau();
163    LED_Gruen();
164    ////////////////////////////////////////////////////////////////////////////
165    // SD-KARTE Initialisierung in einer Schleife, da es möglich ist, dass der
166    // Vorgang nicht beim ersten mal richtig abgeschlossen wird.
167    ////////////////////////////////////////////////////////////////////////////
168    while(FALSE == mmc_init()){}
169
170    fat_loadFatData(); // Initialisirung der Fat
171
172    ////////////////////////////////////////////////////////////////////////////
173    // Für die RTC der Aufruf der Initialisierung von I2C0
174    ////////////////////////////////////////////////////////////////////////////
175    I2C0_Init();
176
177    ////////////////////////////////////////////////////////////////////////////
```

```
178     // Die folgende Funktion muss auskommentiert bleiben, da der RTC nur bei
179     // erstmaliger Inbetriebnahme konfiguriert wird! Funktion nur aktivieren, wenn
180     // RTC resetet werden muss oder nach RTC Batteriewechsel.
181     // Siehe hierfür in die rtc/Slave_RTC.h
182     ////////////////////////////////////////////////////////////////////////////
183     //RTC_config();
184
185     ////////////////////////////////////////////////////////////////////////////
186     // Drei-Achsen-Acceloreter
187     ////////////////////////////////////////////////////////////////////////////
188     I2C1_Init();                  //Aufruf der Initialisierung für I2C1
189     BMA_config();                 //Aufruf der Initialisierung vom Slave (BMA020)
190
191     ////////////////////////////////////////////////////////////////////////////
192     // Temperatur innen und aussen
193     ////////////////////////////////////////////////////////////////////////////
194     //TSIC_Init();
195
196     ////////////////////////////////////////////////////////////////////////////
197     // GPS
198     ////////////////////////////////////////////////////////////////////////////
199     Gps_Init();
200
201     ////////////////////////////////////////////////////////////////////////////
202     // ADC (angedachte Verwendung für sicherse Programmende bei Spannungsabfall)
203     // wird nicht verwendet, da sicheres Programmende durch Brown Out realisiert
204     // wurde!
205     ////////////////////////////////////////////////////////////////////////////
206     // ADC_Init();
207     // ADC_lesen(ADC_Wert);
208     // if(*ADC_Wert > 0x200)
209
210     ////////////////////////////////////////////////////////////////////////////
211     // Schleife in der die Daten zyklisch abgefragt, angepasst und gespeichert werden
212     ////////////////////////////////////////////////////////////////////////////
213     while(schleife)
214     {
215       RTC_lesen(RTC_DATA);
216       Anpassung_Datum(RTC_DATA);
217       Anpassung_Uhrzeit(RTC_DATA);
218
219       BMA_achsen_lesen(BMA_achsen_werte_10Bit);
220       Anpassung_BMA(BMA_achsen_werte_10Bit);
221
222      // Temp_lesen(Temp);
223       Anpassung_Temp(Temp);
224
225       Gps_emp(Gps_signal);
226       Anpassung_Gps(Gps_signal);
227
228       // rechen_test(Test_test);
229       speichern();
230
231       LED_Blau_an();
232     }
233   }
234
235   ////////////////////////////////////////////////////////////////////////////
236   // Funktion die beim Auslösen von Interrupt bei Brown Out aufgerufen wird, um das
```

```
237  // Programm sicher zu beenden
238  ////////////////////////////////////////////////////////////////////////////
239  void brown_out(void)
240  {
241    ffwrite ('\r');
242    // Datei schließen
243    ffclose();
244    // Master nicht erlauben
245    SSIDisable(SSI0_BASE);
246    schleife = 0;
247    while(1){
248      LED_Rot();
249    }
250  }
251
252  ////////////////////////////////////////////////////////////////////////////
253  // GPS Anpassung
254  ////////////////////////////////////////////////////////////////////////////
255  void Anpassung_Gps(char *Gps_signal)
256  {
257    int i =0;
258    int y=0;
259    GPS[0]='$';
260
261    for(i=0;Gps_signal[i] != '\0';i++)
262    {
263      if(Gps_signal[i] == '$')
264      {
265        i++;
266      }
267      if(Gps_signal[i] == 'G' )
268        for(y=1;Gps_signal[i] != '\0';i++,y++)
269        {
270          GPS[y] = Gps_signal[i];
271        }
272    }
273  }
274
275  ////////////////////////////////////////////////////////////////////////////
276  // Funktion zum Speichern der Daten als CSV Datei
277  ////////////////////////////////////////////////////////////////////////////
278  void speichern(void)
279  {
280    // Kopf der Tabelle
281    uint8_t DateiKopf1[] = "Beschleunigungs-Datenlogger;";
282    uint8_t DateiKopf2[] = ";;Datum:;";
283    uint8_t DateiKopf3[] = ";Uhrzeit:;";
284
285    // Bezeichnungen in der Tabelle
286    uint8_t ListenKopf1[]= "Uhrzeit;";
287    uint8_t ListenKopf2[]= "MessZeit;";
288    uint8_t ListenKopf3[]= "X-Achse;";
289    uint8_t ListenKopf4[]= "Y-Achse;";
290    uint8_t ListenKopf5[]= "Z-Achse;";
291    uint8_t ListenKopf6[]= "Temp.(IN);";
292    uint8_t ListenKopf7[]= "Temp.(OUT);";
293    uint8_t ListenKopf8[]= "GPS-Koordinaten;";
294
295    uint8_t Liste[]= ";";
```

```
296     // Prüfung ob Datei existiert
297     if (FALSE == ffileExsists(Filename))
298     {
299       //Datei wird anlegen
300       ffopen(Filename, 'c');
301 
302       // Den Kopf der Tabelle schreiben
303       ffwrites (DateiKopf1);
304       ffwrites (DateiKopf2);
305       ffwrites (Datum);
306       ffwrites (DateiKopf3);
307       ffwrites (Uhrzeit);
308       // Zeilenumbruch
309       ffwrite ('\n');
310       // Die Bezeichnungen in der Tabelle schreiben
311       ffwrites (ListenKopf1);
312       //ffwrites (ListenKopf2);
313       ffwrites (ListenKopf3);
314       ffwrites (ListenKopf4);
315       ffwrites (ListenKopf5);
316       ffwrites (ListenKopf6);
317       ffwrites (ListenKopf7);
318       ffwrites (ListenKopf8);
319       // Für MS-Windows Terminal '\r' anhängen
320       ffwrite ('\r');
321       // Datei schließen
322       ffclose();
323     }
324 
325     // Datei existiert, neue Daten werden angehängt
326     if (TRUE == ffileExsists(Filename))
327     {
328       ffopen(Filename, 'c');
329 
330       // Geht zum Dateiende um Überschreiben zu verhindern
331       ffseek (file.length);
332 
333       ffwrites(Uhrzeit);
334       //  ffwrites(MessZeit);
335       // ffwrites(TEST);
336       ffwrites(X_Achse);
337       ffwrites(Y_Achse);
338       ffwrites(Z_Achse);
339       ffwrites(Temp_IN);
340       ffwrites(Temp_OUT);
341       ffwrites(Liste);
342       ffwrites(GPS);
343 
344       // Für MS-Windows Terminal '\r' anhängen.
345       ffwrite ('\r');
346 
347       // Datei schließen
348       ffclose();
349     }
350   }
351 
352   ////////////////////////////////////////////////////////////////////////////
353   // Funktion zum Wandeln der 10 Bit Daten vom BMA020
354   ////////////////////////////////////////////////////////////////////////////
```

```
355  void Anpassung_BMA( signed short *Achse_10Bit)
356  {
357    signed short BMA_puffer[5] = {0,0,0,0,0};
358    uint8_t Achse_8Bit[3] = {0,0,0};
359    uint8_t Anpassung = 0;
360
361    ////////////////////////////////////////////////////////////////////////////
362    // positive Beschleunigung in Richtung der X-Achse
363    ////////////////////////////////////////////////////////////////////////////
364    if ( Achse_10Bit[0] < 0x200 )
365    {
366      // Glättung der Messerwerte auf 0 im Bereich 0 bis 0.09g der X-Achse
367      // Abweichender Mittelwert von 0 ist 0.088g  wert 0.09g für eine bessere
368      // Darstelung gewählt
369
370      if ( Achse_10Bit[0]<=0x018)
371      {
372        X_Achse[0] = ';';
373        X_Achse[1] = '0';
374        X_Achse[2] = '0';
375        X_Achse[3] = '.';
376        X_Achse[4] = '0';
377        X_Achse[5] = '0';
378        X_Achse[6] = '\0';
379        X_Achse[7] = '\0';
380      }
381      else
382      {
383        Achse_10Bit[0] = Achse_10Bit[0]-0x003; // Annäherung an 9.81 (1G)
384
385        BMA_puffer[3]= Achse_10Bit[0];
386        BMA_puffer[3] = BMA_puffer[3]*3.9;
387
388        BMA_puffer[3] = (BMA_puffer[3]-(BMA_puffer[3]%1000))/1000;
389        BMA_puffer[3] = BMA_puffer[3]+48;
390
391        BMA_puffer[2] = Achse_10Bit[0];
392        BMA_puffer[2] = BMA_puffer[2]*3.9;
393
394        BMA_puffer[2] = (BMA_puffer[2]%1000);
395        BMA_puffer[2] = (BMA_puffer[2]-(BMA_puffer[2]%100))/100;
396        BMA_puffer[2] = BMA_puffer[2]+48;
397
398        BMA_puffer[1] = Achse_10Bit[0];
399        BMA_puffer[1] = BMA_puffer[1]*3.9;
400
401        BMA_puffer[1] = ((BMA_puffer[1]%100)-(BMA_puffer[1]%10))/10;
402        BMA_puffer[1] = BMA_puffer[1]+48;
403
404        BMA_puffer[0] = Achse_10Bit[0];
405        BMA_puffer[0] = BMA_puffer[0]*3.9;
406
407        BMA_puffer[0] = BMA_puffer[0]%10;
408        BMA_puffer[0] = BMA_puffer[0]+48;
409
410        // Positive Darstellung der Messwerte
411        X_Achse[0] = ';';
412        X_Achse[1] = BMA_puffer[3];
413        X_Achse[2] = BMA_puffer[2];
```

```
414         X_Achse[3] = '.';
415         X_Achse[4] = BMA_puffer[1];
416         X_Achse[5] = BMA_puffer[0];
417         X_Achse[6] = '\0';
418         X_Achse[7] = '\0';                // String Terminator
419       }
420     }
421
422     ////////////////////////////////////////////////////////////////////////////
423     // Maximale negative Beschleunigung in Richtung der X-Achse
424     ////////////////////////////////////////////////////////////////////////////
425     if ( Achse_10Bit[0] == 0x200)
426     {
427       X_Achse[0] = ';';
428       X_Achse[1] = '-';
429       X_Achse[2] = '2';
430       X_Achse[3] = '0';
431       X_Achse[4] = '.';
432       X_Achse[5] = '0';
433       X_Achse[6] = '0';
434       X_Achse[7] = '\0';
435     }
436
437     ////////////////////////////////////////////////////////////////////////////
438     //negative Beschleunigung in Richtung der X-Achse
439     ////////////////////////////////////////////////////////////////////////////
440     if ( Achse_10Bit[0] > 0x200)
441     {
442       if (Achse_10Bit[0] >= 0x300)
443       {
444         BMA_puffer[3]= '0';
445       }
446       if (Achse_10Bit[0] < 0x300)
447       {
448         BMA_puffer[3]= '1';
449       }
450       Achse_8Bit[0] = Achse_10Bit[0];
451
452       if(Achse_8Bit[0]>=0xE7)
453       {
454         Anpassung = 96; // Annäherung an -9.81 (-1G)
455       }
456       if (Achse_8Bit[0] >= 0xE0)
457       {
458         Anpassung = 124;
459       }
460
461       Anpassung = 99;
462       BMA_puffer[2] = 255 - Achse_8Bit[0];
463       BMA_puffer[2] = BMA_puffer[2]*3.9;
464       BMA_puffer[2] = BMA_puffer[2]+Anpassung ;
465       BMA_puffer[2] = (BMA_puffer[2]%1000);
466       BMA_puffer[2] = (BMA_puffer[2]-(BMA_puffer[2]%100))/100;
467       BMA_puffer[2] = BMA_puffer[2]+48;
468
469       BMA_puffer[1] = 255 - Achse_8Bit[0];
470       BMA_puffer[1] = BMA_puffer[1]*3.9;
471       BMA_puffer[1] = BMA_puffer[1]+Anpassung ;
472       BMA_puffer[1] = ((BMA_puffer[1]%100)-(BMA_puffer[1]%10))/10;
```

```
473       BMA_puffer[1] = BMA_puffer[1]+48 ;
474
475       BMA_puffer[0] = 255 - Achse_8Bit[0];
476       BMA_puffer[0] = BMA_puffer[0]*3.9;
477       BMA_puffer[0] = BMA_puffer[0]+Anpassung;
478       BMA_puffer[0] = BMA_puffer[0]%10;
479       BMA_puffer[0] = BMA_puffer[0]+48;
480
481       // Negative Darstellung der Messwerte
482       X_Achse[0] = ';';
483       X_Achse[1] = '-';
484       X_Achse[2] = BMA_puffer[3];
485       X_Achse[3] = BMA_puffer[2];
486       X_Achse[4] = '.';
487       X_Achse[5] = BMA_puffer[1];
488       X_Achse[6] = BMA_puffer[0];
489       X_Achse[7] = '\0';
490     }
491
492     ////////////////////////////////////////////////////////////////////////////
493     // positive Beschleunigung in Richtung der Y-Achse
494     ////////////////////////////////////////////////////////////////////////////
495     if ( Achse_10Bit[1] < 0x200)
496     {
497       Anpassung = 40; // Annäherung an 9.81 (1G)
498       BMA_puffer[3] = Achse_10Bit[1];
499       BMA_puffer[3] = BMA_puffer[3]*3.9;
500       BMA_puffer[3] = BMA_puffer[3]+Anpassung;
501       BMA_puffer[3] = (BMA_puffer[3]-(BMA_puffer[3]%1000))/1000;
502       BMA_puffer[3] = BMA_puffer[3]+48;
503
504       BMA_puffer[2] = Achse_10Bit[1];
505       BMA_puffer[2] = BMA_puffer[2]*3.9;
506       BMA_puffer[2] = BMA_puffer[2]+Anpassung;
507       BMA_puffer[2] = (BMA_puffer[2]%1000);
508       BMA_puffer[2] = (BMA_puffer[2]-(BMA_puffer[2]%100))/100;
509       BMA_puffer[2] = BMA_puffer[2]+48;
510
511       BMA_puffer[1] = Achse_10Bit[1];
512       BMA_puffer[1] = BMA_puffer[1]*3.9;
513       BMA_puffer[1] = BMA_puffer[1]+Anpassung;
514       BMA_puffer[1] = ((BMA_puffer[1]%100)-(BMA_puffer[1]%10))/10;
515       BMA_puffer[1] = BMA_puffer[1]+48;
516
517       BMA_puffer[0] = Achse_10Bit[1];
518       BMA_puffer[0] = BMA_puffer[0]*3.9;
519       BMA_puffer[0] = BMA_puffer[0]+Anpassung;
520       BMA_puffer[0] = BMA_puffer[0]%10;
521       BMA_puffer[0] = BMA_puffer[0]+48;
522
523       Y_Achse[0] = ';';
524       Y_Achse[1] = BMA_puffer[3];
525       Y_Achse[2] = BMA_puffer[2];
526       Y_Achse[3] = '.';
527       Y_Achse[4] = BMA_puffer[1];
528       Y_Achse[5] = BMA_puffer[0];
529       Y_Achse[6] = '\0';
530       Y_Achse[7] = '\0';                // String Terminator
531     }
```

```
532
533     ////////////////////////////////////////////////////////////////////////////
534     // Maximale negative Beschleunigung in Richtung der Y-Achse
535     ////////////////////////////////////////////////////////////////////////////
536     if ( Achse_10Bit[1] == 0x200)
537     {
538       Y_Achse[0] = ';';
539       Y_Achse[1] = '-';
540       Y_Achse[2] = '2';
541       Y_Achse[3] = '0';
542       Y_Achse[4] = '.';
543       Y_Achse[5] = '0';
544       Y_Achse[6] = '0';
545       Y_Achse[7] = '\0';
546     }
547
548     ////////////////////////////////////////////////////////////////////////////
549     //negative Beschleunigung in Richtung der Y-Achse
550     ////////////////////////////////////////////////////////////////////////////
551     if ( Achse_10Bit[1] > 0x200)
552     {
553       if (Achse_10Bit[1] >= 0x300)
554       {
555         BMA_puffer[3]= '0';
556       }
557       if (Achse_10Bit[1] < 0x300)
558       {
559         BMA_puffer[3]= '1';
560       }
561       Achse_8Bit[1] = Achse_10Bit[1];
562       {
563         Anpassung = 4;
564         BMA_puffer[2] = 255 - Achse_8Bit[1];
565         BMA_puffer[2] = BMA_puffer[2]*3.9;
566         BMA_puffer[2] = BMA_puffer[2]+Anpassung;
567         BMA_puffer[2] = (BMA_puffer[2]%1000);
568         BMA_puffer[2] = (BMA_puffer[2]-(BMA_puffer[2]%100))/100;
569         BMA_puffer[2] = BMA_puffer[2]+48;
570
571         BMA_puffer[1] = 255 - Achse_8Bit[1];
572         BMA_puffer[1] = BMA_puffer[1]*3.9;
573         BMA_puffer[1] = BMA_puffer[1]+Anpassung;
574         BMA_puffer[1] = ((BMA_puffer[1]%100)-(BMA_puffer[1]%10))/10;
575         BMA_puffer[1] = BMA_puffer[1]+48;
576
577         BMA_puffer[0] = 255 - Achse_8Bit[1];
578         BMA_puffer[0] = BMA_puffer[0]*3.9;
579         BMA_puffer[0] = BMA_puffer[0]+Anpassung;
580         BMA_puffer[0] = BMA_puffer[0]%10;
581         BMA_puffer[0] = BMA_puffer[0]+48;
582
583         Y_Achse[0] = ';';
584         Y_Achse[1] = '-';
585         Y_Achse[2] = BMA_puffer[3];
586         Y_Achse[3] = BMA_puffer[2];
587         Y_Achse[4] = '.';
588         Y_Achse[5] = BMA_puffer[1];
589         Y_Achse[6] = BMA_puffer[0];
590         Y_Achse[7] = '\0';
```

```
591       }
592     }
593
594     ////////////////////////////////////////////////////////////////////////////
595     // positive Beschleunigung in Richtung der Z-Achse
596     ////////////////////////////////////////////////////////////////////////////
597     if ( Achse_10Bit[2] < 0x200)
598     {
599       {
600         if (Achse_10Bit[2] >= 0x019)
601         {
602           Anpassung = 96;
603         }
604         else
605         {
606           Anpassung = 0;
607         }
608         BMA_puffer[3] = Achse_10Bit[2];
609         BMA_puffer[3] = BMA_puffer[3]*3.9;
610         BMA_puffer[3] = BMA_puffer[3] -Anpassung;
611         BMA_puffer[3] = (BMA_puffer[3]-(BMA_puffer[3]%1000))/1000;
612         BMA_puffer[3] = BMA_puffer[3]+48;
613
614         BMA_puffer[2] = Achse_10Bit[2];
615         BMA_puffer[2] = BMA_puffer[2]*3.9;
616         BMA_puffer[2] = BMA_puffer[2] -Anpassung;
617         BMA_puffer[2] = (BMA_puffer[2]%1000);
618         BMA_puffer[2] = (BMA_puffer[2]-(BMA_puffer[2]%100))/100;
619         BMA_puffer[2] = BMA_puffer[2]+48;
620
621         BMA_puffer[1] = Achse_10Bit[2];
622         BMA_puffer[1] = BMA_puffer[1]*3.9;
623         BMA_puffer[1] = BMA_puffer[1] -Anpassung;
624         BMA_puffer[1] = ((BMA_puffer[1]%100)-(BMA_puffer[1]%10))/10;
625         BMA_puffer[1] = BMA_puffer[1]+48;
626
627         BMA_puffer[0] = Achse_10Bit[2];
628         BMA_puffer[0] = BMA_puffer[0]*3.9;
629         BMA_puffer[0] = BMA_puffer[0] -Anpassung;
630         BMA_puffer[0] = BMA_puffer[0]%10;
631         BMA_puffer[0] = BMA_puffer[0]+48;
632
633         // Positive Darstellung
634         //    Z_Achse[0] = ';';
635         //    Z_Achse[1] = BMA_puffer[3];
636         //    Z_Achse[2] = '.';
637         //    Z_Achse[3] = BMA_puffer[2];
638         //    Z_Achse[4] = BMA_puffer[1];
639         //    Z_Achse[5] = BMA_puffer[0];
640         //    Z_Achse[6] = '\0';
641         //    Z_Achse[7] = '\0';
642
643         // Für eine bessere Darstellung der Messwerte in der Auswertung wurde
644         // ein Vorzeichenwechsel durchgeführt für -2g wurde dies nicht zusätzlich
645         // angepasst, da dieser Messwert im Normalfall nicht eintritt !
646
647         Z_Achse[0] = ';';
648         Z_Achse[1] = '-';
649         Z_Achse[2] = BMA_puffer[3];
```

```
650         Z_Achse[3] = BMA_puffer[2];
651         Z_Achse[4] = '.';
652         Z_Achse[5] = BMA_puffer[1];
653         Z_Achse[6] = BMA_puffer[0];
654         Z_Achse[7] = '\0';
655       }
656     }
657
658     ////////////////////////////////////////////////////////////////////////////
659     // Maximale negative Beschleunigung in Richtung der X-Achse
660     ////////////////////////////////////////////////////////////////////////////
661     if ( Achse_10Bit[2] == 0x200)
662     {
663       Z_Achse[0] = ';';
664       Z_Achse[1] = '-';
665       Z_Achse[2] = '2';
666       Z_Achse[3] = '0';
667       Z_Achse[4] = '.';
668       Z_Achse[5] = '0';
669       Z_Achse[6] = '0';
670       Z_Achse[7] = '\0';
671     }
672
673     ////////////////////////////////////////////////////////////////////////////
674     //negative Beschleunigung in Richtung der Z-Achse
675     ////////////////////////////////////////////////////////////////////////////
676     if ( Achse_10Bit[2] > 0x200)
677     {
678       if (Achse_10Bit[2] >= 0x300)
679       {
680         BMA_puffer[3] = '0';
681       }
682       if (Achse_10Bit[2] < 0x300)
683       {
684         BMA_puffer[3] = '1';
685       }
686       Achse_8Bit[2] = Achse_10Bit[2];
687       if ( Achse_8Bit[2] >= 0x1f)
688       {
689         Anpassung = 120;
690       }
691       if ( Achse_8Bit[2] < 0x1f)
692       {
693         Anpassung = 4; // Annäherung an 9.81 (1G)
694       }
695       BMA_puffer[2] = 255 - Achse_8Bit[2];
696       BMA_puffer[2] = BMA_puffer[2]*3.9;
697       BMA_puffer[2] = BMA_puffer[2]+Anpassung;
698       BMA_puffer[2] = (BMA_puffer[2]%1000);
699       BMA_puffer[2] = (BMA_puffer[2]-(BMA_puffer[2]%100))/100;
700       BMA_puffer[2] = BMA_puffer[2]+48;
701
702       BMA_puffer[1] = 255 - Achse_8Bit[2];
703       BMA_puffer[1] = BMA_puffer[1]*3.9;
704       BMA_puffer[1] = BMA_puffer[1]+Anpassung;
705       BMA_puffer[1] = ((BMA_puffer[1]%100)-(BMA_puffer[1]%10))/10;
706       BMA_puffer[1] = BMA_puffer[1]+48;
707
708       BMA_puffer[0] = 255 - Achse_8Bit[2];
```

```
709       BMA_puffer[0] = BMA_puffer[0]*3.9;
710       BMA_puffer[0] = BMA_puffer[0]+Anpassung;
711       BMA_puffer[0] = BMA_puffer[0]%10;
712       BMA_puffer[0] = BMA_puffer[0]+48;
713
714       // Negative Darstellung
715       //    Z_Achse[0]=';';
716       //    Z_Achse[1]='-';
717       //    Z_Achse[2] = BMA_puffer[3];
718       //    Z_Achse[3] ='.';
719       //    Z_Achse[4] = BMA_puffer[2];
720       //    Z_Achse[5] = BMA_puffer[1];
721       //    Z_Achse[6] = BMA_puffer[0];
722       //    Z_Achse[7] = '\0';
723
724       // Für eine bessere Darstellung der Messwerte in der Auswertung wurde
725       // ein Vorzeichenwechsel durchgeführt für -2g wurde dies nicht zusätzlich
726       // angepasst da dieser Messwert im Normalfall nicht eintritt !
727       Z_Achse[0] = ';';
728       Z_Achse[1] = BMA_puffer[3];
729       Z_Achse[2] = BMA_puffer[2];
730       Z_Achse[3] = '.';
731       Z_Achse[4] = BMA_puffer[1];
732       Z_Achse[5] = BMA_puffer[0];
733       Z_Achse[6] = '\0';
734       Z_Achse[7] = '\0';
735     }
736   }
737
738   ////////////////////////////////////////////////////////////////////////////
739   // Funktion zum wandeln des Datum
740   ////////////////////////////////////////////////////////////////////////////
741   void Anpassung_Datum(signed short* RTC_DATA)
742   {
743     signed short RTC_puffer[6] = {0,0,0,0,0,0};
744
745     //Tag im Monat
746     RTC_puffer[5] = RTC_DATA[4];
747     RTC_puffer[5] = RTC_puffer[5]>>4;
748     RTC_puffer[4] = RTC_DATA[4];
749     RTC_puffer[4] &= ~(0xf0);
750
751     //Monat
752     RTC_puffer[3] = RTC_DATA[5];
753     RTC_puffer[3] = RTC_puffer[3]>>4;
754     RTC_puffer[2] = RTC_DATA[5];
755     RTC_puffer[2] &= ~(0xf0);
756
757     //Jahr
758     RTC_puffer[1] = RTC_DATA[6];
759     RTC_puffer[1] = RTC_puffer[1]>>4;
760     RTC_puffer[0] = RTC_DATA[6];
761     RTC_puffer[0] &= ~(0xf0);
762
763     //Datum
764     Datum[0] = (RTC_puffer[5]+48);
765     Datum[1] = (RTC_puffer[4]+48);
766     Datum[2] = '.';
767     Datum[3] = (RTC_puffer[3]+48);
```

```
768     Datum[4] = (RTC_puffer[2]+48);
769     Datum[5] = '.';
770     Datum[6] = '2';
771     Datum[7] = '0';
772     Datum[8] = (RTC_puffer[1]+48);
773     Datum[9] = (RTC_puffer[0]+48);
774     Datum[10] = '\0';
775
776   // Filename ändert sich jeden Tag
777   //  Filename[0] = 'B';
778   //  Filename[1] = 'D';
779   //  Filename[2] = 'L';
780   //  Filename[3] = (RTC_puffer[3]+48);
781   //  Filename[4] = (RTC_puffer[2]+48);
782   //  Filename[5] = (RTC_puffer[5]+48);
783   //  Filename[6] = (RTC_puffer[4]+48);
784   //  Filename[7] = '.';
785   //  Filename[8] = 'c';
786   //  Filename[9] = 's';
787   //  Filename[10]= 'v';
788
789   //  Filename ändert sich alle 10 Minuten
790   //  Filename die ersten 4 Stellen
791      Filename[0] = (RTC_puffer[3]+48);
792      Filename[1] = (RTC_puffer[2]+48);
793      Filename[2] = (RTC_puffer[5]+48);
794      Filename[3] = (RTC_puffer[4]+48);
795   }
796
797   ////////////////////////////////////////////////////////////////////////////
798   // Funktion zum Wandeln der Uhrzeit
799   ////////////////////////////////////////////////////////////////////////////
800   void Anpassung_Uhrzeit(signed short* RTC_DATA)
801   {
802
803     signed short RTC_puffer[6] = {0,0,0,0,0,0};
804
805     //Stunden
806     RTC_puffer[5] = RTC_DATA[2];
807     RTC_puffer[5] = RTC_puffer[5]>>4;
808     RTC_puffer[4] = RTC_DATA[2];
809     RTC_puffer[4] &= ~(0xf0);
810
811     //Minunten
812     RTC_puffer[3] = RTC_DATA[1];
813     RTC_puffer[3] = RTC_puffer[3]>>4;
814     RTC_puffer[2] = RTC_DATA[1];
815     RTC_puffer[2] &= ~(0xf0);
816
817     //Sekunden
818     RTC_puffer[1] = RTC_DATA[0];
819     RTC_puffer[1] = RTC_puffer[1]>>4;
820     RTC_puffer[0] = RTC_DATA[0];
821     RTC_puffer[0] &= ~(0xf0);
822
823     //Uhrzeit
824     Uhrzeit[0] = (RTC_puffer[5]+48);
825     Uhrzeit[1] = (RTC_puffer[4]+48);
826     Uhrzeit[2] = ':';
```

```
827     Uhrzeit[3] = (RTC_puffer[3]+48);
828     Uhrzeit[4] = (RTC_puffer[2]+48);
829     Uhrzeit[5] = ':';
830     Uhrzeit[6] = (RTC_puffer[1]+48);
831     Uhrzeit[7] = (RTC_puffer[0]+48);
832     Uhrzeit[8] = '\0';
833
834     //MessZeit
835     MessZeit[0] = ';';
836     MessZeit[1] = (RTC_puffer[5]+48);
837     MessZeit[2] = (RTC_puffer[4]+48);
838     MessZeit[3] = (RTC_puffer[3]+48);
839     MessZeit[4] = (RTC_puffer[2]+48);
840     MessZeit[5] = (RTC_puffer[1]+48);
841     MessZeit[6] = (RTC_puffer[0]+48);
842     MessZeit[7] = '\0';
843
844   // Filename ändert sich alle 10 Minuten
845   // Filename die letzten 7 Stellen
846     Filename[4] =(RTC_puffer[5]+48);
847     Filename[5] =(RTC_puffer[4]+48);
848     Filename[6] =(RTC_puffer[3]+48);
849     Filename[7] = '.';
850     Filename[8] = 'c';
851     Filename[9] = 's';
852     Filename[10]= 'v';
853   }
854
855   ////////////////////////////////////////////////////////////////////////////
856   // Funktion zum Wandeln der Innen- und Aussenthemperatur
857   ////////////////////////////////////////////////////////////////////////////
858   void Anpassung_Temp(signed short *Temp)
859   {
860       Temp[0] = 0x2ff;
861       Temp[1] = 0x2ff;
862     signed short Temp_puffer[4] = {0,0,0,0};
863
864     uint8_t Temp8[2] ={0,0};
865
866     // Achtung durch Anpassung der Daten kommt es bei Abweichung von  0 c° alle 5 °c
867     // um eine 0.1 c° Erhöhung (25.5 c° sind real 25 c°)
868     // die Genauigkeit des Sensors liegt lt. Datenblatt
869     // im Bereich von 10 c° bis 90 c° bei +-0.3 c°
870
871     // Bei Temperaturen unter -25c° wird nur noch U-25 c° angezeigt
872     if (Temp[0] < 0x0ff)
873     {
874       Temp_IN[0] = ';';
875       Temp_IN[1] = 'U';
876       Temp_IN[2] = '-';
877       Temp_IN[3] = '2';
878       Temp_IN[4] = '5';
879       Temp_IN[5] = 'c';
880       Temp_IN[6] = '°';
881       Temp_IN[7] = '\0';
882       Temp_IN[8] = '\0';
883     }
884
885     // Temperaturen von 0 c° bis -25 c°
```

```
886     if ( Temp[0]  < 0x200 && Temp[0]  > 0x0ff)
887     {
888       Temp8[0] = Temp[0];
889 
890       Temp_puffer[0]='-';
891 
892       Temp_puffer[1] = 255 - Temp8[0] ;
893       Temp_puffer[1] = (Temp_puffer[1]-(Temp_puffer[1]%100))/100;
894       Temp_puffer[1] = Temp_puffer[1]+48;
895 
896       Temp_puffer[2] = 255 - Temp8[0] ;
897       Temp_puffer[2] = ((Temp_puffer[2]%100)-(Temp_puffer[2]%10))/10;
898       Temp_puffer[2] = Temp_puffer[2]+48;
899 
900       Temp_puffer[3] = 255 - Temp8[0] ;
901       Temp_puffer[3] = (Temp_puffer[3]%10);
902       Temp_puffer[3] = Temp_puffer[3]+48;
903 
904       Temp_IN[0]=';';
905       Temp_IN[1] = Temp_puffer[0];
906       Temp_IN[2] = Temp_puffer[1];
907       Temp_IN[3] = Temp_puffer[2];
908       Temp_IN[4] = '.';
909       Temp_IN[5] = Temp_puffer[3];
910       Temp_IN[6] = 'c';
911       Temp_IN[7] = '°';
912       Temp_IN[7] = '\0';
913     }
914 
915     // Temperaturen von 0 c° bis 150 c°
916     if( Temp[0] >= 0x200)
917     {
918       Temp[0]=Temp[0]-0x200;
919 
920       Temp_puffer[0] = Temp[0];
921       Temp_puffer[0] = (Temp_puffer[0]-(Temp_puffer[0]%1000))/1000;
922       Temp_puffer[0] = Temp_puffer[0]+48;
923 
924       Temp_puffer[1] = Temp[0];
925       Temp_puffer[1] = ((Temp_puffer[1]%1000)-(Temp_puffer[1]%100))/100;
926       Temp_puffer[1] = Temp_puffer[1]+48;
927 
928       Temp_puffer[2] = Temp[0];
929       Temp_puffer[2] = ((Temp_puffer[2]%100)-(Temp_puffer[2]%10))/10;
930       Temp_puffer[2] = Temp_puffer[2]+48;
931 
932       Temp_puffer[3] = Temp[0];
933       Temp_puffer[3] = Temp_puffer[3]%10;
934       Temp_puffer[3] = Temp_puffer[3]+48;
935 
936       Temp_IN[0] = ';';
937       Temp_IN[1] = Temp_puffer[0];
938       Temp_IN[2] = Temp_puffer[1];
939       Temp_IN[3] = Temp_puffer[2];
940       Temp_IN[4] = '.';
941       Temp_IN[5] = Temp_puffer[3];
942       Temp_IN[6] = 'c';
943       Temp_IN[7] = '°';
944       Temp_IN[8] = '\0';
```

```
945     }
946
947     // Bei Temperaturen unter -25c° wird nur noch U-25 c° angezeigt
948     if (Temp[1] < 0x0ff)
949     {
950       Temp_OUT[0] = ';';
951       Temp_OUT[1] = 'U';
952       Temp_OUT[2] = '-';
953       Temp_OUT[3] = '2';
954       Temp_OUT[4] = '5';
955       Temp_OUT[5] = 'c';
956       Temp_OUT[6] = '°';
957       Temp_OUT[7] = '\0';
958       Temp_OUT[8] = '\0';
959     }
960
961     // Temperaturen von 0 c° bis -25 c°
962     if ( Temp[1]  < 0x200 && Temp[1]  > 0x0ff)
963     {
964       Temp8[1] = Temp[1];
965
966       Temp_puffer[0]='-';
967
968       Temp_puffer[1] = 255 - Temp8[1] ;
969       Temp_puffer[1] = (Temp_puffer[1]-(Temp_puffer[1]%100))/100;
970       Temp_puffer[1] = Temp_puffer[1]+48;
971
972       Temp_puffer[2] = 255 - Temp8[1] ;
973       Temp_puffer[2] = ((Temp_puffer[2]%100)-(Temp_puffer[2]%10))/10;
974       Temp_puffer[2] = Temp_puffer[2]+48;
975
976       Temp_puffer[3] = 255 - Temp8[1] ;
977       Temp_puffer[3] = (Temp_puffer[3]%10);
978       Temp_puffer[3] = Temp_puffer[3]+48;
979
980       Temp_OUT[0] = ';';
981       Temp_OUT[1] = Temp_puffer[0];
982       Temp_OUT[2] = Temp_puffer[1];
983       Temp_OUT[3] = Temp_puffer[2];
984       Temp_OUT[4] = '.';
985       Temp_OUT[5] = Temp_puffer[3];
986       Temp_OUT[6] = 'c';
987       Temp_OUT[7] = '°';
988       Temp_OUT[8] = '\0';
989     }
990
991     // Temperaturen von 0 c° bis 150 c°
992     if( Temp[1] >= 0x200)
993     {
994       Temp[1]=Temp[1]-0x200;
995
996       Temp_puffer[0] = Temp[1];
997       Temp_puffer[0] = (Temp_puffer[0]-(Temp_puffer[0]%1000))/1000;
998       Temp_puffer[0] = Temp_puffer[0]+48;
999
1000      Temp_puffer[1] = Temp[1];
1001      Temp_puffer[1] = ((Temp_puffer[1]%1000)-(Temp_puffer[1]%100))/100;
1002      Temp_puffer[1] = Temp_puffer[1]+48;
1003
```

```
1004        Temp_puffer[2] = Temp[1];
1005        Temp_puffer[2] = ((Temp_puffer[2]%100)-(Temp_puffer[2]%10))/10;
1006        Temp_puffer[2] = Temp_puffer[2]+48;
1007
1008        Temp_puffer[3] = Temp[1];
1009        Temp_puffer[3] = Temp_puffer[3]%10;
1010        Temp_puffer[3] = Temp_puffer[3]+48;
1011
1012        Temp_OUT[0] = ';';
1013        Temp_OUT[1] = Temp_puffer[0];
1014        Temp_OUT[2] = Temp_puffer[1];
1015        Temp_OUT[3] = Temp_puffer[2];
1016        Temp_OUT[4] = '.';
1017        Temp_OUT[5] = Temp_puffer[3];
1018        Temp_OUT[6] = 'c';
1019        Temp_OUT[7] = '°';
1020        Temp_OUT[8] = '\0';
1021      }
1022    }
1023    void rechen_test(signed short *test)
1024    {
1025
1026      signed short BMA_puffer[4] = {0,0,0,0};
1027      // char BMA_puffer[4] = {0,0,0,0};
1028      signed short testzahl = 0x300;
1029      uint8_t test1 = testzahl;
1030      //char testzahl = 0x3ff;
1031
1032      // BMA_puffer[3] = ~(testzahl);
1033      //    BMA_puffer[3] = BMA_puffer[3]*3.9;
1034      //    BMA_puffer[3] = BMA_puffer[3]+4;
1035      //    BMA_puffer[3] = (BMA_puffer[3]-(BMA_puffer[3]%1000))/1000;
1036      //    BMA_puffer[3] = BMA_puffer[3]+48;
1037      if (testzahl >= 0x300)
1038      {
1039        BMA_puffer[3]= '0';
1040      }
1041      if (testzahl < 0x300)
1042      {
1043        BMA_puffer[3]= '1';
1044      }
1045      BMA_puffer[2] = 255-test1;
1046      //BMA_puffer[2] = test1;
1047      BMA_puffer[2] = BMA_puffer[2]*3.9;
1048      BMA_puffer[2] = BMA_puffer[2]+4;
1049      BMA_puffer[2] = (BMA_puffer[2]%1000);
1050      BMA_puffer[2] = (BMA_puffer[2]-(BMA_puffer[2]%100))/100;
1051      BMA_puffer[2] = BMA_puffer[2]+48;
1052      //BMA_puffer[2] = BMA_puffer[2]+48;
1053
1054      BMA_puffer[1] = 255-test1;
1055      // BMA_puffer[2] = test1;
1056      BMA_puffer[1] = BMA_puffer[1]*3.9;
1057      BMA_puffer[1] = BMA_puffer[1]+4;
1058      BMA_puffer[1] = ((BMA_puffer[1]%100)-(BMA_puffer[1]%10))/10;
1059      BMA_puffer[1] = BMA_puffer[1]+48;
1060      //BMA_puffer[1] = BMA_puffer[1]+57;
1061
1062      BMA_puffer[0] = 255-test1;
```

```
1063    //BMA_puffer[2] = test1;
1064    BMA_puffer[0] = BMA_puffer[0]*3.9;
1065    BMA_puffer[0] = BMA_puffer[0]+4;
1066    BMA_puffer[0] = BMA_puffer[0]%10;
1067    BMA_puffer[0] = BMA_puffer[0]+48;
1068    //BMA_puffer[0] = BMA_puffer[0]+54;
1069    TEST[0] = ';';
1070    TEST[1] = '-';
1071    TEST[2] = BMA_puffer[3];
1072    TEST[3] = '.';
1073    TEST[4] = BMA_puffer[2];
1074    TEST[5] = BMA_puffer[1];
1075    TEST[6] = BMA_puffer[0];
1076    TEST[7] = '\0';
1077  }
1078
```

Hauptroutine (H-File)

```
1
2   /*****************************************************************************
3   *****************************************************************************
4   *
5   * File:                 BDL.h
6   *
7   * Created on:           25.04.2014
8   * Author:               Marc Thom
9   *
10  *****************************************************************************
11  *****************************************************************************/
12  #ifndef BDL_H_
13  #define BDL_H_
14
15  ////////////////////////////////////////////////////////////////////////////
16  // Externe Funktionen
17  ////////////////////////////////////////////////////////////////////////////
18  extern void speichern(void);
19  extern void Anpassung_BMA(signed short *Achse_10Bit);
20  extern void Anpassung_Datum(signed short *RTC_DATA);
21  extern void Anpassung_Uhrzeit(signed short *RTC_DATA);
22  extern void Anpassung_Temp(signed short *Temp);
23  extern void rechen_test(signed short *test);
24  extern void Temp_lesen (signed short *Temp);
25  extern void brown_out(void);
26
27  #endif /* BDL_H_ */
28
```

Serial-Peripheral-Interface & SD-Karte (H-File)

```
1  /*****************************************************************************
2  ******************************************************************************
3  *
4  * File:                 Master_Slave_SD_CARD.c
5  *
6  * Created on:           02.06.2014
7  * Author:               Marc Thom
8  *
9  *
10 * Anmerkung:            Die verwendeten Funktionen sind Standartfunktionen von
11 *                       Texas Instruments !
12 *                       Das für die Funktionen benötigte
13 *                       Softwarepaket ist das StellarisWare: SW-LM3S-LM4F
14 *                       und wurde unter http://www.ti.com/tool/sw-lm3s
15 *                       bezogen. Letzter Zugriff 19.05.2014
16 *
17 *                       Der Funktionsaufruf mmc_wait_ready()stammt aus dem
18 *                       Softwarepaket von Daniel R. das unter
19 *                       http://www.mikrocontroller.net/articles/AVR_FAT32#Der_Status
20 *                       bezogen wurde. Letzter Zugriff 2014/04/05
21 *
22 ******************************************************************************
23 *****************************************************************************/
24
25 //----------Eigene Header BMA020----------
26 #include "sd/Mikro_SD_mmc_config.h"
27 #include "sd/Mikro_SD_mmc.h"
28 #include "sd/Mikro_SD_Fat.h"
29 #include "sd/Mikro_SD_File.h"
30 #include "sd/Master_SD_CARD.h"
31
32 //----------Standart Header----------
33 #include "inc/lm4f120h5qr.h"
34 #include <stdio.h>
35 #include <string.h>
36
37 //----------inc----------
38 #include "inc/hw_ssi.h"             // SSI (SPI)
39 #include "inc/hw_sysctl.h"          // SYSCTL
40 #include "inc/hw_gpio.h"            // GPIO
41 #include "inc/hw_memmap.h"
42 #include "inc/hw_types.h"
43
44 //----------driverlib----------
45 #include "driverlib/gpio.h"
46 #include "driverlib/interrupt.h"
47 #include "driverlib/ssi.h"
48 #include "driverlib/sysctl.h"
49 #include "driverlib/systick.h"
50 #include "driverlib/pin_map.h"
51
52 ////////////////////////////////////////////////////////////////////////////
53 // Inizalisierung von SSI0
54 ////////////////////////////////////////////////////////////////////////////
55 void SSI0_Init(void)
56 {
57   //Die SSI0 Peripherie erlaubt
58   SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
59
```

```
119     return TRUE;
120   }
121
122   ////////////////////////////////////////////////////////////////////////////
123   // CS auf High
124   ////////////////////////////////////////////////////////////////////////////
125   void Deselect(void)
126   {
127     GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, GPIO_PIN_3);
128     SSI0_Lesen();
129   }
130
131   ////////////////////////////////////////////////////////////////////////////
132   // Byte über SSI0 an die SD Karte schreiben
133   ////////////////////////////////////////////////////////////////////////////
134   void SSI0_Schreiben(uint8_t data)
135   {
136     unsigned long buffer;
137     SSIDataPut(SSI0_BASE, data); //Daten in den fifo schreiben
138     SSIDataGet(SSI0_BASE, &buffer); //Daten lesen
139   }
140
141   ////////////////////////////////////////////////////////////////////////////
142   // Byte über SSI0 von der SD Karte lesen
143   ////////////////////////////////////////////////////////////////////////////
144   uint8_t SSI0_Lesen(void)
145   {
146     unsigned long buffer;
147
148     SSIDataPut(SSI0_BASE, 0xFF); // Dummy Daten schreiben
149     SSIDataGet(SSI0_BASE, &buffer); // Daten lesen
150     return (uint8_t)buffer;
151   }
152
```

Serial-Peripheral-Interface & SD-Karten (H-File)

```
1
2  /*****************************************************************************
3  *****************************************************************************
4  *
5  * File:                Master_SD_CARD.h
6  *
7  * Created on:          05.04.2014
8  * Author:              Marc Thom
9  *
10 *****************************************************************************
11 *****************************************************************************/
12
13 #ifndef MASTER_SD_CARD_H_
14 #define MASTER_SD_CARD_H_
15
16
17 // externen Funktionen für SSI0
18 extern void SSI0_Init(void);
19 extern void SSI0_max_speed (void);
20 extern uint8_t Select(void);
21 extern void Deselect(void);
22 extern void SSI0_Schreiben(uint8_t data);
23 extern uint8_t SSI0_Lesen(void);
24
25
26 ////////////////////////////////////////////////////////////////////////////
27 // Die externen Funktionsaufrufe stammen aus dem Softwarepaket von Daniel R. das
28 // unter http://www.mikrocontroller.net/articles/AVR_FAT32#Der_Status
29 // bezogen wurde. Letzter Zugriff 2014/04/05
30 ////////////////////////////////////////////////////////////////////////////
31 extern uint8_t mmc_wait_ready (void);
32 extern uint8_t mmc_init (void);
33 extern uint8_t mmc_send_cmd (uint8_t cmd, uint32_t arg);
34 extern uint8_t mmc_multi_block_stop_read (void);
35 extern uint8_t mmc_multi_block_stop_write (void);
36 extern void mmc_multi_block_read_sector (uint8_t *Buffer);
37 extern uint8_t mmc_multi_block_write_sector (uint8_t *Buffer);
38 extern uint8_t mmc_wait_ready (void);
39 extern uint8_t mmc_write_sector (uint32_t addr,uint8_t *buffer);
40 extern uint8_t mmc_read_sector (uint32_t addr,uint8_t *buffer);
41
42
43 #endif /* MASTER_SD_CARD_H_ */
44
```

Inter-Integrated-Circuit & RTC (C-File)

```
/*****************************************************************************
******************************************************************************
*
* File:                 Master_Slave_RTC.c
*
* Created on:           28.02.2014
* Author:               Marc Thom
*
*
* Anmerkung:            Die verwendeten Funktionen sind Standartfunktionen von
*                       Texas Instruments !
*                       Das für die Funktionen benötigte
*                       Softwarepaket ist das StellarisWare: SW-LM3S-LM4F
*                       und wurde unter http://www.ti.com/tool/sw-lm3s
*                       bezogen. Letzter Zugriff 19.05.2014
*
******************************************************************************
*****************************************************************************/

//----------Eigene Header RTC----------
#include "rtc/Master_RTC.h"
#include "rtc/Slave_RTC.h"

#include "booster/LED.h"

//----------inc----------
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"

//----------driverlib----------
#include "driverlib/i2c.h"
#include "driverlib/gpio.h"


#define GPIO_PB2_I2C0SCL        0x00010803

#define GPIO_PB3_I2C0SDA        0x00010C03

//*****************************************************************************
//Inizialisierung von I2C0
//*****************************************************************************
void I2C0_Init(void)
{
  //Die I2C0 Peripherie erlaubt
  SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);

  //Benutzte Port B [2:3]
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

  //Configurierung von I2C0 für die Funktion an Port B2
  GPIOPinConfigure(GPIO_PB2_I2C0SCL);
  GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);

  //Configurierung von I2C0 für die Funktion an Port B3
  GPIOPinConfigure(GPIO_PB3_I2C0SDA);
  GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);

  // Für RTC nur Standarteinstellung möglich !!
  // 0 = Datenrate auf 100kbps gesetzt (Standardeinstellung)
```

```
60    I2CMasterInitExpClk(I2C0_MASTER_BASE, SysCtlClockGet(), 0);
61
62    // Master erlauben
63    I2CMasterEnable(I2C0_MASTER_BASE);
64  }
65
66  ////////////////////////////////////////////////////////////////////////////
67  // Ersteinstellung der RTC (bei Neueinstellung Werte in rtc/Slave_RTC.h änden)
68  ////////////////////////////////////////////////////////////////////////////
69  void RTC_config(void)
70  {
71    // RTC Adresse und Einstellung von Uhrzeit und Datum
72    I2C0_Slave_register(RTC_ADDRESS, SEC, MIN, STUNDE, TAG, DATUM, MONAT, JAHR);
73  }
74
75  ////////////////////////////////////////////////////////////////////////////
76  // Funktion um Uhrzeit und Datum in der RTC zu setzen
77  // Die Kommentare für Register Sekunde setzen trifft auch auf die folgenden
78  // Register zu
79  ////////////////////////////////////////////////////////////////////////////
80  unsigned long I2C0_Slave_register(unsigned char adress, unsigned char sec,
81                                    unsigned char min, unsigned char stunde,
82                                    unsigned char tag, unsigned char datum,
83                                    unsigned char monat, unsigned char jahr)
84  {
85
86    ////////////////////////////////////////////////////////////////////////////
87    // Register Sekunde setzen
88    ////////////////////////////////////////////////////////////////////////////
89    // Master mitteilen, welche Adresse der Slave hat an den er schreiben soll
90    // 0 = Master an Slave schreiben
91    I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
92
93    // Register für Sekunde
94    I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Sec);
95
96    // Senden
97    I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
98
99    // Warten bis Master fertig ist mit der Übertragung
100   while(I2CMasterBusy(I2C0_MASTER_BASE)){}
101
102   // Daten welche ins Register geschrieben werden sollen
103   I2CMasterDataPut(I2C0_MASTER_BASE,sec);
104
105   // warten bis Master fertig ist mit der Übertragung
106   while(I2CMasterBusy(I2C0_MASTER_BASE)){}
107
108   // Ende senden
109   I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
110
111   // Warten bis Master fertig ist mit der Übertragung
112   while(I2CMasterBusy(I2C0_MASTER_BASE)){}
113
114   ////////////////////////////////////////////////////////////////////////////
115   // Register Minute setzen
116   ////////////////////////////////////////////////////////////////////////////
117   I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
118   I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Min);
```

```
119     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
120     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
121     I2CMasterDataPut(I2C0_MASTER_BASE,min);
122     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
123     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
124     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
125
126     ////////////////////////////////////////////////////////////////////////////
127     // Register Stunde setzen
128     ////////////////////////////////////////////////////////////////////////////
129     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
130     I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Stunde);
131     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
132     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
133     I2CMasterDataPut(I2C0_MASTER_BASE,stunde);
134     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
135     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
136     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
137
138     ////////////////////////////////////////////////////////////////////////////
139     // Register Tag setzen
140     ////////////////////////////////////////////////////////////////////////////
141     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
142     I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Tag);
143     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
144     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
145     I2CMasterDataPut(I2C0_MASTER_BASE,tag);
146     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
147     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
148     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
149
150     ////////////////////////////////////////////////////////////////////////////
151     // Register Datum setzen
152     ////////////////////////////////////////////////////////////////////////////
153     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
154     I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Datum);
155     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
156     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
157     I2CMasterDataPut(I2C0_MASTER_BASE,datum);
158     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
159     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
160     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
161
162     ////////////////////////////////////////////////////////////////////////////
163     // Register Monat setzen
164     ////////////////////////////////////////////////////////////////////////////
165     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
166     I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Monat);
167     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
168     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
169     I2CMasterDataPut(I2C0_MASTER_BASE,monat);
170     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
171     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
172     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
173
174     ////////////////////////////////////////////////////////////////////////////
175     // Register Jahr setzen
176     ////////////////////////////////////////////////////////////////////////////
177     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
```

```
178     I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Jahr);
179     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
180     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
181     I2CMasterDataPut(I2C0_MASTER_BASE,jahr);
182     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
183     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
184     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
185
186     return 1;
187   }
188
189   ////////////////////////////////////////////////////////////////////////////
190   // Funktion um Uhrzeit und das Datum von der RTC auszulesen
191   // Die Kommentare für Register Sekunde auslesen trifft auch auf die folgenden
192   // Register zu
193   ////////////////////////////////////////////////////////////////////////////
194   unsigned long I2C0_Slave_lesen(unsigned char adress, unsigned long* puffer)
195   {
196
197     ////////////////////////////////////////////////////////////////////////////
198     // Register Sekunde auslesen
199     ////////////////////////////////////////////////////////////////////////////
200     // Master mitteilen, welche Adresse der Slave hat an die er schreiben soll
201     // 0 = Master an Slave schreiben
202     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
203
204     // Register welches gelesen werden soll
205     I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Sec);
206
207     // Senden
208     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
209
210     // Warten das Senden abgeschlossen ist
211     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
212
213     // Dem Master mitteilen welche Adresse der Slave hat von dem er lesen soll
214     // 1 = Master vom Slave lesen
215     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 1);
216
217     // Empfangen
218     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
219
220     // Warten das Senden abgeschlossen ist
221     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
222
223     // Empfangenes Byte auslesen
224     puffer[0] = I2CMasterDataGet(I2C0_MASTER_BASE);
225
226     // Warten das Senden abgeschlossen ist
227     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
228
229     // Empfangen
230     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
231
232     // Warten das Senden abgeschlossen ist
233     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
234
235     ////////////////////////////////////////////////////////////////////////////
236     // Register Minute auslesen
```

```
237     ////////////////////////////////////////////////////////////////////////////
238     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
239     I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Min);
240     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
241     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
242     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 1);
243     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
244     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
245     puffer[1] = I2CMasterDataGet(I2C0_MASTER_BASE);
246     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
247     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
248     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
249
250     ////////////////////////////////////////////////////////////////////////////
251     // Register Stunde auslesen
252     ////////////////////////////////////////////////////////////////////////////
253     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
254     I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Stunde);
255     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
256     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
257     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 1);
258     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
259     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
260     puffer[2] = I2CMasterDataGet(I2C0_MASTER_BASE);
261     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
262     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
263     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
264
265     ////////////////////////////////////////////////////////////////////////////
266     // Register Tag auslesen
267     ////////////////////////////////////////////////////////////////////////////
268     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
269     I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Tag);
270     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
271     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
272     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 1);
273     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
274     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
275     puffer[3] = I2CMasterDataGet(I2C0_MASTER_BASE);
276     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
277     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
278     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
279
280     ////////////////////////////////////////////////////////////////////////////
281     // Register Datum auslesen
282     ////////////////////////////////////////////////////////////////////////////
283     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
284     I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Datum);
285     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
286     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
287     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 1);
288     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
289     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
290     puffer[4] = I2CMasterDataGet(I2C0_MASTER_BASE);
291     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
292     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
293     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
294
295     ////////////////////////////////////////////////////////////////////////////
```

```
296     // Register Monat auslesen
297     ////////////////////////////////////////////////////////////////////////////
298     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
299     I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Monat);
300     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
301     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
302     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 1);
303     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
304     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
305     puffer[5] = I2CMasterDataGet(I2C0_MASTER_BASE);
306     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
307     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
308     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
309
310     ////////////////////////////////////////////////////////////////////////////
311     // Register Jahr auslesen
312     ////////////////////////////////////////////////////////////////////////////
313     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 0);
314     I2CMasterDataPut(I2C0_MASTER_BASE, Reg_Jahr);
315     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
316     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
317     I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, adress, 1);
318     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
319     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
320     puffer[6] = I2CMasterDataGet(I2C0_MASTER_BASE);
321     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
322     I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
323     while(I2CMasterBusy(I2C0_MASTER_BASE)){}
324
325     return 1;
326   }
327
328   ////////////////////////////////////////////////////////////////////////////
329   //Daten von der RTC lesen
330   ////////////////////////////////////////////////////////////////////////////
331   void RTC_lesen(signed short* RTC_DATA)
332   {
333     unsigned long Daten[7] = {0,0,0,0,0,0,0};
334
335     I2C0_Slave_lesen(RTC_ADDRESS, Daten);
336
337     RTC_DATA[0] = Daten[0]; // Sekunden
338     RTC_DATA[1] = Daten[1]; // Minuten
339     RTC_DATA[2] = Daten[2]; // Stunden
340     RTC_DATA[3] = Daten[3]; // Wochentage
341     RTC_DATA[4] = Daten[4]; // Tage
342     RTC_DATA[5] = Daten[5]; // Monate
343     RTC_DATA[6] = Daten[6]; // Jahre
344   }
345
```

Inter-Integrated-Circuit & RTC (H-File)

```
1
2   /*****************************************************************************
3   ******************************************************************************
4   *
5   * File:                Slave_RTC.h
6   *
7   * Created on:          28.02.2014
8   * Author:              Marc Thom
9   *
10  ******************************************************************************
11  *****************************************************************************/
12  #ifndef SLAVE_RTC_H_
13  #define SLAVE_RTC_H_
14
15
16  #define RTC_ADDRESS     0x68
17
18  ////////////////////////////////////////////////////////////////////////////////
19  // Einstellung der ersten Inbetriebnahme des RTC nur ändern wenn RTC resetet
20  // werden muss oder nach Batteriewechsel siehe hierfür in die Kommentare
21  // am Ende der Header
22  ////////////////////////////////////////////////////////////////////////////////
23  #define SEC                     0x00
24  #define MIN                     0x30// 0x30
25  #define STUNDE                  0x11// 0x18
26  #define TAG                     0x02 // 0x07
27  #define DATUM                   0x10 //0x25
28  #define MONAT                   0x06 //0x05
29  #define JAHR                    0x14 //0x14
30
31
32  #define Reg_Sec                 0x00
33  #define Reg_Min                 0x01
34  #define Reg_Stunde              0x02
35  #define Reg_Tag                 0x03
36  #define Reg_Datum               0x04
37  #define Reg_Monat               0x05
38  #define Reg_Jahr                0x06
39  //#define Reg_Control             0x07
40  //#define Reg_Ram                 0x08
41  ////////////////////////////////////////////////////////////////////////////////
42
43
44  /*-----------------------------------------------------------------------------
45  * DS1307 RTC (als Slave über I2C0)
46  * Wichtige Daten
47  *
48  * Einstellung der Uhrzeit
49  *
50  * Einstellung von Sekunden in Register 0x00
51  * Bit 7 auf 0 für Oscillator erlauben auf 1 für nicht erlauben
52  * Bit 6 bis 4 für Sekunden in 10er Schritten
53  * Bit 3 bis 0 für die Sekunde 0 bis 9
54  *
55  * Einstellung von Minuten in Register 0x01
56  * Bit 7 wird nicht verwendet
57  * Bit 6 bis 4 für Minuten in 10er Schritten
58  * Bit 3 bis 0 für die Minute 0 bis 9
59  *
```

```
60  * Einstellung von Stunden in Register 0x02
61  * Bit 7 wird nicht verwendet
62  * Bit 6 auf 0 für 24 Stunden Modus auf 1 für 12 Stunden Modus
63  * Bit 5 im 12 Stunden Modus 1 für PM und 0 für AM Bit 4 wird nicht benutzt
64  * Bit 5 im 24 Stunden Modus für (Stunde 20 bis 23)
65  * Bit 4 für 10er Stunde
66  * Bit 3 bis 0 für die Stunde  0 bis 9
67  *
68  * Einstellung des Datums
69  *
70  * Einstellung vom Wochentag in Register 0x03
71  * Bit 7 bis 3 wird nicht verwendet
72  * Bit 2 bis 0 für die 7 Tage
73  *
74  * Einstellung vom Tag in Register 0x04
75  * Bit 7 und 6 wird nicht verwendet
76  * Bit 5 und 4 für Datum in 10er Schritten
77  * Bit 3 bis 0 für Datum 1 bis 9
78  *
79  * Einstellung vom Monat in Register 0x05
80  * Bit 7 bis 5 wird nicht verwendet
81  * Bit 4 für 10ter Monat
82  * Bit 3 bis 0 für Monat 1 bis 9
83  *
84  * Einstellung vom Jahr in Register 0x06
85  * Bit 7 bis 4 10er Jahre
86  * Bit 3 bis 0 für Jahr 0 bis 9
87  *(Einstellung 00 bis 99)
88  *-------------------------------------------------------------------------
89  */
90
91  #endif /* SLAVE_RTC_H_ */
92
```

```
1
2   /*****************************************************************************
3   ******************************************************************************
4   *
5   * File:         Master_RTC.h
6   *
7   * Created on:           28.02.2014
8   * Author:               Marc Thom
9   *
10  ******************************************************************************
11  *****************************************************************************/
12
13  #ifndef MASTER_RTC_H_
14  #define MASTER_RTC_H_
15
16  extern void I2C0_Init(void);
17  extern void RTC_config();
18  extern void RTC_lesen(signed short* RTC_DATA);
19
20  extern unsigned long I2C0_Slave_register(unsigned char adress, unsigned char sec,
21                                           unsigned char min, unsigned char stunde,
22                                           unsigned char tag, unsigned char datum,
23                                           unsigned char monat, unsigned char jahr);
24
25  extern unsigned long I2C0_Slave_lesen(unsigned char adress, unsigned long* puffer);
26
27
28  #endif /* MASTER_RTC_H_ */
29
```

Inter-Integrated-Circuit & BMA020 (C-File)

```
/*****************************************************************************
******************************************************************************
*
* File:                 Master_Slave_BMA020.c
*
* Created on:           28.02.2014
* Author:               Marc Thom
*
*
* Anmerkung:            Die verwendeten Funktionen sind Standartfunktionen von
*                       Texas Instruments !
*                       Das für die Funktionen benötigte
*                       Softwarepaket ist das StellarisWare: SW-LM3S-LM4F
*                       und wurde unter http://www.ti.com/tool/sw-lm3s
*                       bezogen. Letzter Zugriff 19.05.2014
*
******************************************************************************
*****************************************************************************/

//----------Eigene Header BMA020----------
#include "bma/Slave_BMA020.h"
#include "bma/Master_BMA020.h"

//----------inc----------
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"

//----------driverlib----------
#include "driverlib/i2c.h"
#include "driverlib/gpio.h"

//////////////////////////////////////////////////////////////////////////////
// Inizalisierung von I2C1
//////////////////////////////////////////////////////////////////////////////
void I2C1_Init(void)
{
  // Die I2C1 Peripherie erlaubt
  SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);

  // Port A realisieren  [7:6]
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

  // Configurierung von I2C1 für die Funktion an Port A6
  GPIOPinConfigure(GPIO_PA6_I2C1SCL);
  GPIOPinTypeI2CSCL(GPIO_PORTA_BASE, GPIO_PIN_6);

  // Configurierung von I2C1 für die Funktion an Port A7
  GPIOPinConfigure(GPIO_PA7_I2C1SDA);
  GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_7);

  // Erlauben und Initialisieren von I2C1 in Master Modus
  // Benutzung der System Clock für I2C1 (beide Einstellungen sind möglich)
  // 0 = Datenrate auf 100kbps gesetzt (Standard Einstellung)
  // 1 = Datenrate auf 400kbps gesetzt (fast mode Einstellung)
  I2CMasterInitExpClk(I2C1_MASTER_BASE, SysCtlClockGet(), 1);

  // Master erlauben
  I2CMasterEnable(I2C1_MASTER_BASE);
}
```

```
60
61  ////////////////////////////////////////////////////////////////////////////
62  // Registereinstellung von Slave (BMA020)
63  // Achtung nie mals von CONTROL_2 mehr als die ersten 5 Bit ändern !!
64  ////////////////////////////////////////////////////////////////////////////
65  void BMA_config(void)
66  {
67    // Einstellung 1500 Hz und +/- 2g
68    I2C1_Slave_register(SLAVE_ADDRESS, CONTROL_2, hz_1500_2g);
69  }
70
71  ////////////////////////////////////////////////////////////////////////////
72  // Master Registereinstellung auf Slave (BMA020)schreiben
73  ////////////////////////////////////////////////////////////////////////////
74  unsigned long I2C1_Slave_register(unsigned char adress, unsigned char reg,
75                                    unsigned char data)
76  {
77    // Master mitteilen, welche Adresse der Slave hat an den er schreiben soll
78    // 0 = Master an Slave schreiben
79    I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, adress, 0);
80
81    // Register welches beschrieben werden soll
82    I2CMasterDataPut(I2C1_MASTER_BASE, reg);
83
84    // Senden
85    I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
86
87    // Warten bis Master fertig ist mit der Übertragung
88    while(I2CMasterBusy(I2C1_MASTER_BASE)){}
89
90    // Daten welche ins Register geschrieben werden sollen
91    I2CMasterDataPut(I2C1_MASTER_BASE, data);
92
93    // Warten bis Master fertig ist mit der Übertragung
94    while(I2CMasterBusy(I2C1_MASTER_BASE)){}
95
96    // Ende senden
97    I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
98
99    // Warten bis Master fertig ist mit der Übertragung
100   while(I2CMasterBusy(I2C1_MASTER_BASE)){}
101
102   return 1;
103 }
104
105 ////////////////////////////////////////////////////////////////////////
106 // Daten von Slave lesen
107 ////////////////////////////////////////////////////////////////////////
108 unsigned long I2C1_Slave_lesen(unsigned char adress, unsigned char length,
109                                unsigned char start_reg, unsigned long* puffer)
110 {
111   unsigned char i = 0;
112
113   // Master mitteilen, welche Adresse der Slave hat an die er schreiben soll
114   // 0 = Master an Slave schreiben
115   I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, adress, 0);
116
117   // Register welches gelesen werden soll
118   I2CMasterDataPut(I2C1_MASTER_BASE, start_reg);
```

```
119
120     // Senden
121     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
122
123     // Warten das Senden abgeschlossen ist
124     while(I2CMasterBusy(I2C1_MASTER_BASE)){}
125
126     // Dem Master mitteilen welche Adresse der Slave hat von dem er lesen soll
127     // 1 = Master vom Slave lesen
128     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, adress, 1);
129
130     // Empfangen
131     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
132
133     // Warten das Senden abgeschlossen ist
134     while(I2CMasterBusy(I2C1_MASTER_BASE)){}
135
136     // Empfangenes Byte auslesen
137     puffer[0] = I2CMasterDataGet(I2C1_MASTER_BASE);
138
139     for(i = 1; i < length; i++)
140     {
141       // Empfangen
142       I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT);
143
144       // Warten das Senden abgeschlossen ist
145       while(I2CMasterBusy(I2C1_MASTER_BASE)){}
146
147       // Empfangenes Byte auslesen
148       puffer[i] = I2CMasterDataGet(I2C1_MASTER_BASE);
149     }
150
151     // Empfangen
152     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
153
154     // Warten das Senden abgeschlossen ist
155     while(I2CMasterBusy(I2C1_MASTER_BASE)){}
156
157     return 1;
158   }
159
160   ////////////////////////////////////////////////////////////////////////////
161   //Achsen auslesen und zu 10 Bit zusammenfügen
162   ////////////////////////////////////////////////////////////////////////////
163   void BMA_achsen_lesen(signed short *Achse_10Bit)
164   {
165     unsigned long Axenwert[6] = {0,0,0,0,0,0};
166
167     I2C1_Slave_lesen(SLAVE_ADDRESS, 6, 0x02, Axenwert);
168
169     // Werte für die X Achse und dann LSB und MSB zu 10 Bit zusammenfügen
170     Achse_10Bit[0] = ACHSE_10BIT(Axenwert[0], Axenwert[1]);
171
172     // Werte für die Y Achse und dann LSB und MSB zu 10 Bit zusammenfügen
173     Achse_10Bit[1] = ACHSE_10BIT(Axenwert[2], Axenwert[3]);
174
175     // Werte für die Z Achse und dann LSB und MSB zu 10 Bit zusammenfügen
176     Achse_10Bit[2] = ACHSE_10BIT(Axenwert[4], Axenwert[5]);
177   }
```

Inter-Integrated-Circuit & BMA020 (H-File)

```
1
2  /*****************************************************************************
3  ******************************************************************************
4  *
5  * File:                Slave_BMA020.h
6  *
7  * Created on:          18.02.2014
8  * Author:              Marc Thom
9  *
10 ******************************************************************************
11 *****************************************************************************/
12
13 #ifndef SLAVE_BMA020_H_
14 #define SLAVE_BMA020_H_
15
16 ////////////////////////////////////////////////////////////////////////////////
17 // 7 Bit Adresse im Format [A6:A5:A4:A3:A2:A1:A0:LB]
18 // Bei einer 0 in LB Master sendet, bei einer 1 Master lesen
19 ////////////////////////////////////////////////////////////////////////////////
20 #define SLAVE_ADDRESS       0x38
21
22 ////////////////////////////////////////////////////////////////////////////////
23 // Für die Registereinstellung vom BMA020
24 // es wurde vorerst nur mit CONTROL_2 Einstellung vorgenommen
25 ////////////////////////////////////////////////////////////////////////////////
26 #define CONTROL_1       0x15
27 #define CONTROL_2       0x14
28 #define CONTROL_3       0x0A
29 #define CONTROL_4       0x0B
30
31 ////////////////////////////////////////////////////////////////////////////////
32 // Definetion für CONTROL_2 (Einstellung des Messbereichs und der Abtastrate)
33 // Der Bereich von +/- 2g ist nach Vorrausberechnung ausreichend. Die Abtastrate
34 // kann nach Bedarf ausgewählt werden.
35 // Für andere G Bereiche muss der entsprechende Hexa Wert errechnet werden.
36 // Siehe hier für in die Kommentare am Ende der Header.
37 // Wichtig unter der Adresse CONTROL_2 nur die ersten 5 Bit (0 bis 4) ändern!!
38 ////////////////////////////////////////////////////////////////////////////////
39 #define hz_25_2g                0x00          // Einstellung 25 Hz und +/- 2g
40 #define hz_50_2g                0x01          // Einstellung 50 Hz und +/- 2g
41 #define hz_100_2g               0x02          // Einstellung 100 Hz und +/- 2g
42 #define hz_190_2g               0x03          // Einstellung 190 Hz und +/- 2g
43 #define hz_375_2g               0x04          // Einstellung 375 Hz und +/- 2g
44 #define hz_750_2g               0x05          // Einstellung 750 Hz und +/- 2g
45 #define hz_1500_2g              0x06          // Einstellung 1500 Hz und +/- 2g
46
47 ////////////////////////////////////////////////////////////////////////////////
48 // Definition für die 10 Bit die aus LSB und MSB geformt wurden.
49 ////////////////////////////////////////////////////////////////////////////////
50 #define ACHSE_10BIT(LSB,MSB)    ((((unsigned long) MSB) << 2) | LSB >> 6)
51
52 /*-------------------------------------------------------------------------------
53 * BMA020 3 Achesenbeschleuniger (als Slave über I2C1)
54 * Wichtige Daten !
55 *-------------------------Data zur Beschaltung---------------------------------
56 *
57 *           UIN - CSB
58 *           UIN - UPullup (über 10 kOhm mit 3.3V)
59 *           UIN - +2,5V bis +6V (3.3V Gewählt)
```

```
60  *           SDO - GND
61  *
62  *       Vom Mikrocontroller zum BMA020 Modul:
63  *           SCI - SCK
64  *           SDA - SDI
65  *           GND - GND
66  *
67  * Datenblatt S.9
68  * X-Achse 0x03 MSB <9:2> und 0x02 LSB <1:0> (Bit 0 = neue Daten)
69  * Y-Achse 0x05 und <9:2> und 0x04 LSB <1:0> (Bit 0 = neue Daten)
70  * Z-Achse 0x07 und <9:2> und 0x06 LSB <1:0> (Bit 0 = neue Daten)
71  *
72  * Datenblatt S.21
73  * acc_x, acc_y, acc_z auslesen für 10 Bit
74  * acc_x (0x02, Bit 6 und 7 (LSB), 0x03, Bit 7 bis 0 (MSB))
75  * acc_y (0x04, Bit 6 und 7 (LSB), 0x05, Bit 7 bis 0 (MSB))
76  * acc_z (0x06, Bit 6 und 7 (LSB), 0x07, Bit 7 bis 0 (MSB))
77  *
78  * Für 10 Bit Darstelung ergibt sich somit LSB muss 6 nach links
79  * und MSB muss 2 nach rechts geschoben werden.
80  *
81  * -2.000g : 10 0000 0000
82  * -1.996g : 10 0000 0001
83  * ......................
84  * -0.004g : 11 1111 1111
85  *  0.000g : 00 0000 0000
86  * +0.004g : 00 0000 0001
87  * ......................
88  * +1.992g : 01 1111 1110
89  * +1.996g : 01 1111 1111
90  *
91  * Datenblatt S.9
92  * Adresse 0x01
93  * Bit 0 bis 3 ml_version <3:0>
94  * Bit 4 bis 7 al_version <3:0>
95  *
96  * Datenblatt S.9
97  * Adresse 0x00
98  * Bit 0 bis 2 chip_id
99  *
100 * Default setting mit Binrär -----010
101 *
102 *----------------------Control-------------------------------------------------
103 *
104 * Datenblatt S.10 und S.11
105 * Die Adresse  0x14  !!!Achtung Bit 5 bis 7 nicht benutzen)!!!
106 * Bit 2 = 0 und Bit 1 = 0 und Bit 0 = 0 für 25 Hz     HEX. = 0x00
107 * Bit 2 = 0 und Bit 1 = 0 und Bit 0 = 1 für 50 Hz     HEX. = 0x01
108 * Bit 2 = 0 und Bit 1 = 1 und Bit 0 = 0 für 100 Hz    HEX. = 0x02
109 * Bit 2 = 0 und Bit 1 = 1 und Bit 0 = 1 für 190 Hz    HEX. = 0x03
110 * Bit 2 = 1 und Bit 1 = 0 und Bit 0 = 0 für 375 Hz    HEX. = 0x04
111 * Bit 2 = 1 und Bit 1 = 0 und Bit 0 = 1 für 750 Hz    HEX. = 0x05
112 * Bit 2 = 1 und Bit 1 = 1 und Bit 0 = 0 für 1500 Hz   HEX. = 0x06
113 * Bit 4 = 0 und Bit 3 = 0 für +/- 2g
114 * Bit 4 = 0 und Bit 3 = 1 für +/- 4g
115 * Bit 4 = 1 und Bit 3 = 0 für +/- 8g
116 *
117 * Default setting mit Binär xxx0 0000
118 *
```

```
119  * Für 0x14 -> 00000 für +/- 2g und 25 Hz
120  *
121  * Datenblatt S.9 und S.12
122  * Die Adresse 0X15
123  * Bit 0 wechseln des BMA020 vom Schlaf Mode zum Normal Mode
124  * Bit 2 = 0 und Bit 1 = 0 aufwachen nach 20 ms  HEX. = 0x00
125  * Bit 2 = 0 und Bit 1 = 1 aufwachen nach 80 ms  HEX. = 0x01
126  * Bit 2 = 1 und Bit 1 = 0 aufwachen nach 320 ms HEX. = 0x02
127  * Bit 2 = 1 und Bit 1 = 1 aufwachen nach 2560 ms HEX. = 0x03
128  * Bit 3 shadow_dis
129  * Bit 4 latch_INT    (wird für I2C nicht benutzt bzw. muss 0 sein)
130  * Bit 5 new_data_INT (wird für I2C nicht benutzt bzw. muss 0 sein)
131  * Bit 6 enable_adv_INT (wir für disable advanced interrupt benutzt)
132  * kein effekt auf den sensor bei IC Funktion)
133  * Bit 7 SPI4 (wird für I2C Nicht benutzt bzw. muss 0 sein)
134  *
135  * Default setting mit Binär 1000 0000
136  *
137  * Für 0x15 -> 000 für aufwachen in 20 ms
138  *
139  * Datenblatt S.9 und S.13
140  * Die Adresse 0x0B
141  * Bit 0 Enable_LG (erzeugt Interrupt)
142  * Bit 1 Enable_HG (erzeugt Interrupt)
143  * Bit 2 und 3 counter_LG
144  * Bit 4 und 5 counter_HG
145  * Bit 6 any_motion (generiert bei irgend einer Bewegung einen Interupt)
146  * Bit 7 Alert (weckt den BMA020 bei bewegung auf)
147  *
148  * Datenblatt S.9
149  * Die Adresse 0x0A
150  * Bit 0 sleep
151  * Bit 1 soft_reset
152  * Bit 2 self_test_0
153  * Bit 3 self_test_1
154  * Bit 6 rest_IMT
155  *
156  * Default setting mit Binär 00
157  * ------------------------------------------------------------------------------
158  * Datenblatt S.32
159  * Datentransfer beginnt bei fallender Flanke von SDA wenn SCK high ist
160  *
161  * Datenblatt S.34
162  * BMA020 als Slave Adresse 0x38
163  * schreiben  Adresse 0x70
164  * um Daten lesen zu können muss zuerst der Offset gesetzt werden
165  * Schiebe-Register (0x02)
166  * lesen Adresse 0x71
167  * Protocol Slave Adresse -> Register Adresse -> Register Datei
168  *------------------------------------------------------------------------------
169  */
170
171  #endif /* SLAVE_BMA020_H_ */
172
```

```
1
2   /*******************************************************************************
3   *******************************************************************************
4   *
5   * File:                Master_BMA020.h
6   *
7   * Created on:          18.02.2014
8   * Author:              Marc Thom
9   *
10  *******************************************************************************
11  *******************************************************************************/
12
13  #ifndef MASTER_BMA020_H_
14  #define MASTER_BMA020_H_
15
16  ////////////////////////////////////////////////////////////////////////////////
17  // Master ist der LM4F120H5QR Mikrocontroller
18  // Hier Definetion um die zugehörigen Definitionen in den driverlib zu erkennen
19  // die für verschiedene Mikrokontroler geeignet sind.
20  ////////////////////////////////////////////////////////////////////////////////
21  #define PART_LM4F120H5QR
22
23  ////////////////////////////////////////////////////////////////////////////////
24  // Externe funktionen
25  ////////////////////////////////////////////////////////////////////////////////
26  extern void I2C1_Init(void);
27  extern void BMA_config(void);
28  extern unsigned long I2C1_Slave_register(unsigned char adress, unsigned char reg,
29                                           unsigned char data);
30  extern unsigned long I2C1_Slave_lesen(unsigned char adress, unsigned char length,
31                                        unsigned char start_reg,
32                                        unsigned long* puffer);
33  extern void BMA_achsen_lesen(signed short* Achse_10Bit);
34
35  ////////////////////////////////////////////////////////////////////////////////
36  // I2C1 in Master Mode
37  // PA6 I2C1 SCL (Pin 23) Bitfeld 3 port A bit 6
38  // PA7 I2C1 SDA (Pin 24) Bitfeld 3 port A bit 7
39  ////////////////////////////////////////////////////////////////////////////////
40
41  #endif /* MASTER_BMA020_H_ */
42
```

Universal-Asynchronous-Receiver-Transmitter & GPS (C-File)

```
1
2  /*****************************************************************************
3  *****************************************************************************
4  *
5  * File:                 Master_Slave_GPS.c
6  *
7  * Created on:           16.08.2014
8  * Author:               Marc Thom
9  *
10 *
11 * Anmerkung:            Die verwendeten Funktionen sind Standartfunktionen von
12 *                       Texas Instruments !
13 *                       Das für die Funktionen benötigte
14 *                       Softwarepaket ist das StellarisWare: SW-LM3S-LM4F
15 *                       und wurde unter http://www.ti.com/tool/sw-lm3s
16 *                       bezogen. Letzter Zugriff 19.05.2014
17 *
18 *****************************************************************************
19 *****************************************************************************/
20
21 //----------Eigene Header GPS----------
22 #include "gps/gps.h"
23
24 //----------inc----------
25 #include "inc/hw_ints.h"
26 #include "inc/hw_memmap.h"
27 #include "inc/hw_types.h"
28
29 //----------driverlib----------
30 #include "driverlib/sysctl.h"
31 #include "driverlib/uart.h"
32 #include "driverlib/gpio.h"
33
34 //----------Standart Header----------
35 #include <stdio.h>
36 #include <stdbool.h>
37 #include <cstdlib>
38 #include <cstring>
39
40
41
42 #define PART_LM4F120H5QR
43
44 #define GPIO_PB0_U1RX           0x00010001
45 #define GPIO_PB1_U1TX           0x00010401
46
47
48
49 ////////////////////////////////////////////////////////////////////////////
50 // Initialisierung von UART
51 ////////////////////////////////////////////////////////////////////////////
52 void Gps_Init(void)
53 {
54
55   // Die UART1 Peripherie erlaubt
56   SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
57
58   // Port B realisieren [0:1]
59   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
```

```
60
61    // Configurierung von UART1 für die Funktion an Port B0
62    GPIOPinConfigure(GPIO_PB0_U1RX);
63
64    // Configurierung von UART1 für die Funktion an Port B1
65    GPIOPinConfigure(GPIO_PB1_U1TX);
66    GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
67
68    // Configurierung von UART für 4800, 8-N-1 Operation
69    UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 4800,
70                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
71                         UART_CONFIG_PAR_NONE));
72
73
74    UARTIntEnable(UART1_BASE, UART_INT_RX | UART_INT_RT);
75
76  }
77
78  ////////////////////////////////////////////////////////////////////////////
79  // GPS-Daten empfangen und nach Signalanfang und Signalende auswerten
80  ////////////////////////////////////////////////////////////////////////////
81  void Gps_emp(char *Gps_signal)
82  {
83    int i = 0;
84    while(UARTCharsAvail(UART1_BASE))
85    {
86      // Signalanfang
87      if (Gps_signal[i] == '$')
88      {
89        {
90          while (Gps_signal[i] != '*')
91          {
92
93            Gps_signal[++i] = UARTCharGet(UART1_BASE);
94          }
95          // Signalende
96          if (Gps_signal[i]=='*')
97          {
98            Gps_signal[i]='\0';
99          }
100       }
101     }
102     else
103     {
104       Gps_signal[i] = UARTCharGetNonBlocking(UART1_BASE);
105     }
106   }
107 }
108
```

Universal-Asynchronous-Receiver-Transmitter & GPS (H-File)

```
1
2  /*****************************************************************************
3  ******************************************************************************
4  *
5  * File:                gps.h
6  *
7  * Created on:          16.08.2014
8  * Author:              Marc Thom
9  *
10 ******************************************************************************
11 *****************************************************************************/
12
13 #ifndef GPS_H_
14 #define GPS_H_
15
16 extern void Gps_Init(void);
17 extern void Gps_emp(char *Gps_signal);
18 extern void Anpassung_Gps(char *Gps_signal);
19
20 #endif /* GPS_H_ */
```

Anmerkung:

Der vollständige Quellcode für den Beschleunigungs-Datenlogger befindet sich auf der Zusatz-DVD im Ordner (Quellcode).

Versicherung über die Selbständigkeit

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

Hamburg,11.Septemper 2014
Ort, Datum

Unterschrift