Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Felix Attila Groth

Datenlogger für Elektrobusse mit Mikrocontrollersteuerung, Inertialmesssystem sowie GPS- und GSM-Modulen

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Felix Attila Groth

Datenlogger für Elektrobusse mit Mikrocontrollersteuerung, Inertialmesssystem sowie GPS- und GSM-Modulen

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

Betreuender Prüfer : Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter : Prof. Dr.-Ing. Jürgen Vollmer

Abgegeben am 15. Mai 2015

Felix Attila Groth

Thema der Bachelorthesis

Datenlogger für Elektrobusse mit Mikrocontrollersteuerung, Inertialmesssystem sowie GPS- und GSM-Modulen

Stichworte

Elektrobus, Datenlogger, IMU, Inertialmesssystem, GPS, GSM, GPRS

Kurzzusammenfassung

Im Rahmen des Forschungsprojektes BATSEN wird ein Datenlogger zur Aufzeichnung von Positions- und Bewegungsdaten in Elektro- und Hybridbussen entwickelt. Neue Hardware- und Softwaremodule werden entworfen, gefertigt und diese mit weiteren Komponenten zu einem System integriert. Die Daten sollen zur Erstellung von Lade- und Entladekurven zur Batteriebeurteilung tauglich sein. Durch Testfahrten wird die Eignung des Messkonzeptes überprüft und erwartete Effekte werden bestätigt.

Felix Attila Groth

Title of the paper

Data Logger for Electric Busses with Microcontroller Control, Inertial Measurement Unit as well as GPS and GSM Modules

Keywords

Electric busses, data logger, IMU, inertial measurement unit, GPS, GSM

Abstract

A data-logger is developed within the research project BATSEN to record position- and motion data in electric and und hybrid busses. New hardware and software modules are designed, manufactured, and integrated to a system together with further components. The data should be suitable to generate meaningful charge- and discharge curves. With test drives the usability of the measurement concept is evaluated and expected effects are verified.

Inhaltsverzeichnis

1. Einführung und Motivation

1.1. Emissionsfreier öffentlicher Nahverkehr und das Projekt BEEDeL

> *„Hamburg ist einer der größten Buskäufer in Deutschland. Deshalb ist unsere Stadt als Entwicklungslabor für emissionsfreie Busse so wichtig [...] Der Senat hat die politische Entscheidung getroffen, ab 2020 ausschließlich emissionsfreie Busse durch seine Unternehmen, die Hochbahn und die Verkehrsbetriebe Hamburg-Holstein, für den öffentlichen Nahverkehr beschaffen zu lassen."* [30]

Der fortschreitende Klimawandel hat u.a. durch steigende Meeresspiegel und das häufigere Auftreten von Wetterextremen weitreichende Folgen für den Menschen und die Natur. Er ist im Wesentlichen Folge des Ausstoßes von Treibhausgasen und hat im vergangenen Jahrhundert zu einem Temperaturanstieg von etwa $0.7\,^{\circ}\mathrm{C}$ geführt. Wird der Treibhausgasausstoß nicht gebremst, wird eine Erwärmung von weiteren $1.4\,^{\circ}\mathrm{C}$ bis $5.8\,^{\circ}\mathrm{C}$ bis zum Jahr 2100 prognostiziert. [27]

Mit einem Anteil von etwa $23.1\,\%$ hat der Verkehr erheblichen Anteil an den CO_2-Emissionen in Hamburg. Wiederholtes Überschreiten der Immissionsgrenzwerte für z.B. Stickoxide an verkehrsnahen Messstationen im Hamburger Straßennetz zeigt zudem, dass Maßnahmen zur Verminderung der Belastung der Luftqualität durch den Straßenverkehr nötig werden. [3]

Großen Einfluss auf die durch den Straßenverkehr verursachten Umweltbelastungen hat die Verwendung fossiler Brennstoffe zum Antrieb von Kraftfahrzeugen. Der Hamburger Senat setzt deshalb auf die Verlagerung von Individualverkehr auf den öffentlichen Nahverkehr sowie den Einsatz innovativer Techniken. Die Anbieter des städtischen Nahverkehrs sollen zur Unterstützung dieser Ziele ab dem Jahr 2020 keine Linienbusse mehr beschaffen, die fossile Brennstoffe verwenden. [8]

Das Projekt BEEDeL soll betriebliche Fragestellungen in Bezug auf den Einsatz von Elektrobussen im öffentlichen Nahverkehr sowie deren wirtschaftliche und funktionelle Auswirkungen untersuchen. [23] Im Rahmen des Projektes arbeiten zusammen:

Abbildung 1.1.: Nahverkehrsbus beim Nachladen an der Ladestation der Innovationslinie 109 am Hamburger ZOB [7]

- Hamburger Hochbahn AG
- Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung (Fraunhofer-IVI)
- Hochschule für Angewandte Wissenschaften Hamburg

Es sollen mit den Ergebnissen Planungsaufgaben der HOCHBAHN unterstützt werden, indem u.a. die Tauglichkeit und sinnvolle Ausbaupläne einer dezentralen Ladeinfrastruktur untersucht werden. Außerdem sollen Auswirkungen von Ladezeiten und anderen Betriebsbedingungen auf die Lebenszeit der Batterie analysiert werden. Teilnetze, die zur Umrüstung aufgrund betrieblicher Rahmenbedingungen besonders geeignet sind, sollen Identifiziert werden. [24]

Um relevante Antriebstechnologien unter ähnlichen Rahmenbedingungen testen zu können, hat die HOCHBAHN im Jahr 2014 die sog. Innovationslinie 109 in Betrieb genommen. Auf dieser innerstädtischen Buslinie des Hamburger Nahverkehrs werden seitdem fünf verschieden Bustypen mit alternativen Antrieben eingesetzt. Darunter jeweils ein Brennstoffzellen- und Batterie-Brennstoffzellen-Bus, serieller und paralleler Diesel-Batterie-Hybridbus sowie ein sog. Plug-In-Bus, dessen Batterie an einer Ladestation an den Endhaltestellen aufgeladen wird (Abb. 1.1). [7]

1.2. Batterien in Nahverkehrsbussen und das Projekt BATSEN

Das Projekt BATSEN beschäftigt sich, unter Beteiligung der Hochschule für Angewandte Wissenschaften und Industriepartnern aus der Automobil-, Batterie- und Zulieferindustrie, mit der Entwicklung und Erprobung von drahtlosen Sensornetzen zur Überwachung von Batteriezellen. Die Sensoren sollen u.a. auf Zellen von Starter- und Traktionsbatterien in Fahrzeugen zum Einsatz kommen um Wirtschaftlichkeit, Sicherheit und Verfügbarkeit im Betrieb zu erhöhen. [22]

Gerade in Traktionsbatterien, also zum Antrieb des Fahrzeuges, müssen Batteriezellen hohen Anforderungen gerecht werden, um das von fossilen Brennstoffen gewohnte Betriebsverhalten möglichst anzunähern.
Lithium-Ionen-Batterien, die aufgrund vergleichsweise hoher Energie und Leistungsdichte häufige Verwendung als Traktionsbatterien in Fahrzeugen finden, halten ihre Lebensdauer in Zyklen und Jahren nach den Spezifikationen des Herstellers in der Regel nur bei Einhaltung enger Betriebsparameter ein. Dabei spielen insbesondere die Höhe der Lade- und Entladeströme sowie die Temperatur während des Ladens und Entladens eine Rolle. [2]

Beim Einsatz als Traktionsbatterien in Bussen sind zudem die Bedingungen des Busbetriebs zu beachten. Die Busse werden von der HOCHBAHN auf Linien mit einer Länge von bis zu etwa $35\,\mathrm{km}$ eingesetzt und befahren in einem Umlauf mehrere u.U. unterschiedliche Linien [24]. Um den bis zu $350\,\mathrm{km}$ langen Umläufen gerecht zu werden, sind die Nahverkehrsbusse wie z.B. der Mercedes Citaro mit Dieseltanks mit einem Volumen von $260\,\mathrm{l}$ und mehr ausgestattet [40]. Beim Vergleich der Energiedichte einer Lithium-Ionen-Batterie von etwa $120\,\frac{\mathrm{W\,h}}{\mathrm{kg}}$ bis $150\,\frac{\mathrm{W\,h}}{\mathrm{kg}}$ zu Dieselkraftstoff mit etwa $9.7\,\frac{\mathrm{kW\,h}}{\mathrm{l}} \approx 9.3\,\frac{\mathrm{kW\,h}}{\mathrm{kg}}$ lässt sich leicht erkennen, dass durch die sehr viel geringere Energiedichte der Batterien nur vergleichsweise kleine Energiemengen im Fahrzeug vorgehalten werden können [5]. Zusätzlich zu dem hohen Gewicht der Batterien, entstehen für große Kapazitäten auch deutlich höhere Anschaffungskosten.

Am Beispiel des auf der Innovationslinie 109 verwendeten Plug-In-Hybrid-Busses „Volvo 7900 Electric Hybrid“ bedeutet dies, dass nur eine Lithium-Ionen-Batterie von $19\,\mathrm{kW\,h}$ für eine rein elektrische Reichweite von etwa $7\,\mathrm{km}$ verbaut wird. [43] Es ist deshalb nötig, Konzepte zum Laden der Batterien in den Betriebsablauf zu integrieren. Kurze Standzeiten an den Endhaltestellen und damit der Erbringung eines möglichst großen Transportangebotes pro Fahrzeug führen dabei dazu, dass die Ladung der Batterien in möglichst kurzer Zeit und damit mit hohen Strömen erfolgen muss.

Um die Effizienz der Elektrobusse weiter zu verbessern, kommen in den Fahrzeugen zudem Systeme zur Rekuperation zum Einsatz. Diese sollen möglichst große Teile der beim

Bremsen abgeführten kinetischen Energie in elektrische Energie umwandeln und in der Traktionsbatterie speichern. So kann Energie zum Antrieb des Fahrzeugs verwendet werden, die von mechanischen Bremsen als Wärmeenergie abgeführt worden wäre.

Wie sich diese Eigenschaften auf die Lade- und Entladeprofile und damit auf die Lebenszeit und Betriebssicherheit der Batterien auswirken, soll im Rahmen der Projekte untersucht werden.

Im Rahmen des BATSEN Projekts wurde u.a. von Wisniewski [45] ein Zyklierprüfstand entwickelt, welcher es ermöglichen soll, verschiedene Batteriezellen unter praxisrelevanten Betriebsbedingungen zu testen. Die Verwendung von Daten mit Bezug zur Betriebspraxis von Nahverkehrsbussen und den daraus resultierenden speziellen Last- und Ladeprofilen für die Batterien wird zur Zeit von Schmidt [29] untersucht. Es ergibt sich für weitere Untersuchungen der Bedarf an Messdaten zum Fahr- und Betriebsverhalten von Nahverkehrsbussen sowie den tatsächlichen Last- und Ladeprofilen.

2. Theoretischer Hintergrund

2.1. Fahrzeugbeschleunigung

2.1.1. Zusammenhang zwischen Beschleunigung und Motorstrom

Zur Beschreibung der Fahrzeugbewegungen sollen Koordinatensysteme und Bezeichnungen wie in [20] zur Anwendung kommen. Diese sind in Abb. 2.1 dargestellt. Im Folgenden

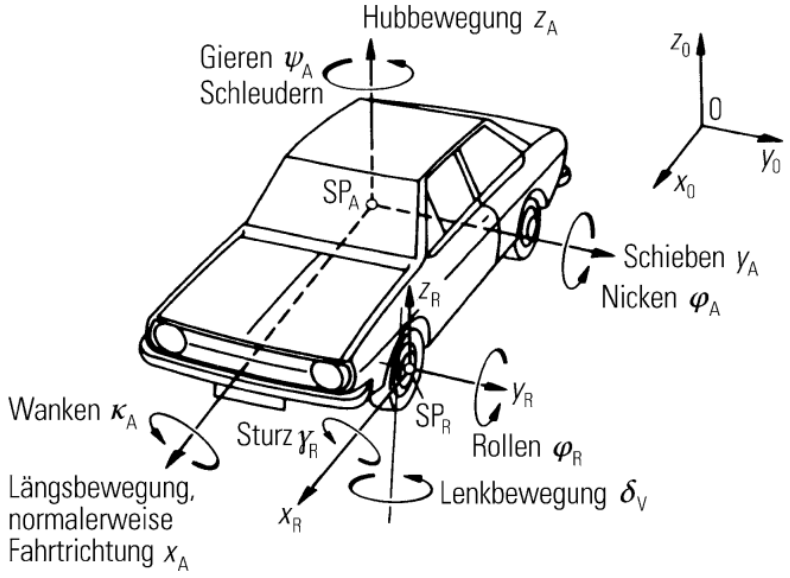


Abbildung 2.1.: Koordinatensysteme zur Beschreibung der Fahrzeugbewegungen und deren Benennung [20, S. 4]

soll zunächst gezeigt werden, dass die Beschleunigung eines Elektro-Fahrzeuges direkt mit dem Motorstrom in Zusammenhang steht. Abgesehen von Effekten die in 2.1.2 beschrieben werden, treten die energetisch relevanten Kräfte und Beschleunigungen entlang der Längsachse des Fahrzeuges auf. Nur hier wird motorisch Energie zu- oder durch Bremsen abgeführt. Es sollen zunächst die Kräfte und Momente am Fahrzeug gemäß Abb. 2.2, ohne

Berücksichtigung eines Anhängers, betrachtet werden.
Dabei sind

$$G_{Rj} \widehat{=} \text{ Gewicht aller Räder der Achse } j$$

$$\alpha \widehat{=} \text{ Längsneigung der Fahrbahn}$$

$$G \widehat{=} \text{ Fahrzeugaufbaugewicht}$$

$$F_{Zj} \widehat{=} \text{ Achslast}$$

$$F_{Lx} \widehat{=} \text{ Luftwiderstand}$$

$$\sum X_j \widehat{=} \text{ Summe aller Längskräfte an den Achsen}$$

$$F_{Xj} \widehat{=} \text{ Umfangskräfte}$$

Unter der Annahme, dass die translatorische Beschleunigung der Achsen gleich der des Aufbaues ist, können X_j und F_{Xj} eliminiert werden [20, S. 76]. Über die Radmomente und -radien kann die Zugkraft Z im unbeschleunigten Zustand gemäß [20, S. 76 Gl 5.4] wie folgt dargestellt werden:

$$Z = \sum_{j=1}^{n} \frac{M_{Rj}}{r_j} = \left(m + \sum_{j=1}^{n} \frac{J_{Rj}}{r_j R_j} \right) \ddot{x} + G \sin\alpha + F_{Lx} + \sum_{j=1}^{{}^{n}F_{Zj} \frac{e_j}{r_j}}$$

Wobei die Summanden auf der rechten Seite der Gleichung den Beschleunigungswiderstand F_B, den Steigungswiderstand F_{St}, den Luftwiderstand F_{Lx} sowie den Rollwiderstand F_R darstellen [20, S. 77 ff]. Wird die wirkende Zugkraft größer als die Widerstände, so wird das Fahrzeug durch die Überschusszugkraft F_a beschleunigt. Die Beschleunigung $a = \ddot{x}$ ergibt sich mit der Fahrzeugmasse m zu

$$a = \frac{F_a}{m} \text{ mit } F_a = Z - (F_B + F_{St} + F_{Lx} + F_R)$$

Die Beschleunigung hängt also direkt von den Radmomenten ab.

Unter der beispielhaften Annahme des Antriebs mit einer dreiphasigen Drehstrom-Synchronmaschine ergibt sich das Drehmoment nach [6, S. 117] zu

$$M = \frac{P_{mech}}{\omega}$$

wobei $\omega = 2\pi \cdot n$ die Kreisfrequenz und P_{ab} die abgegebene mechanische Leistung ist. Die aufgenommene elektrische Leistung P_{zu} ergibt sich als

$$P_{zu} = 3 \cdot U_{ph} \cdot I_{1ph} \cdot \cos\varphi$$

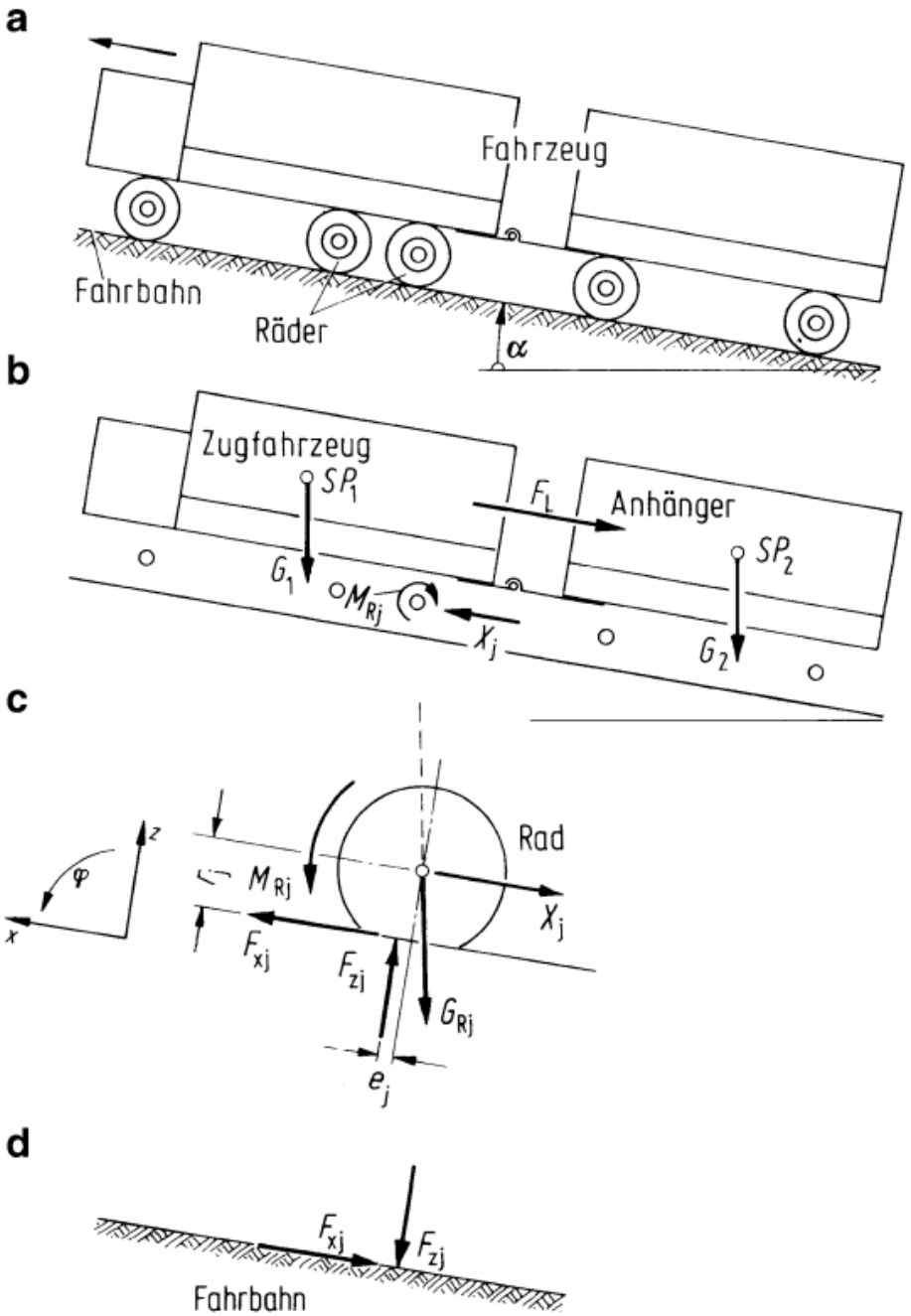


Abbildung 2.2.: **a** Fahrzeug, Rad und Fahrbahn; **b** Kräfte und Momente am Fahrzeugaufbau; **c** Kräfte und Momente am Rad; **d** Kräfte an der Fahrbahn [20, S. 77]

mit U_{ph} Ständerspannung, I_{1ph} Ständerstrom und dem Leistungsfaktor $\cos\varphi$ [6, S. 117]. Sie ist über den, von den Maschineneigenschaften abhängigen, Wirkungsgrad mit der abgegebenen mechanischen Leistung verknüpft und ergibt sich mit der Verlustleistung P_V (u.a. Kupfer-, Eisen- und Reibungsverluste) zu:

$$P_{zu} = P_{ab} + P_V$$

Es ist somit gezeigt, dass Ständerstrom und -spannung direkt ursächlich auf die Größe der Beschleunigung des Fahrzeuges einwirken. Die Zusammenhänge lassen sich auf den generatorischen Betrieb der Synchronmaschine bei der Rekuperation während des Schub- und Bremsbetriebs übertragen.

2.1.2. Beschleunigungen und Kräfte in stationären Fahrsituationen

Es sollen nun die am Fahrzeug auftretenden Beschleunigungen und ihre Richtung in relevanten stationären Fahrsituationen betrachtet werden, um im Weiteren bewerten zu können, wie diese sensorisch zu erfassen sind. Außerdem soll anhand der gewonnenen Erkenntnisse eine u.U. nötige Fehlerabschätzung bei Verwendung bestimmter Sensorik ermöglicht werden. Die Betrachtung erfolgt im Einzelnen für:

- Geradeausfahrt mit konstanter Geschwindigkeit
- Geradeausfahrt mit veränderlicher Geschwindigkeit
- Kreisfahrt mit konstanter Geschwindigkeit
- Kreisfahrt mit veränderlicher Geschwindigkeit

Nicht betrachtet werden sollen hier vor allem Schwingungen durch Unebenheits- und Motoranregung, da diese nahezu mittelwertfrei und damit energetisch und messtechnisch für den Anwendungsfall wenig relevant sind. Zudem wird die Regelkreis-Betrachtung, wie sie in der Literatur zur Dynamik von Fahrer und Fahrzeug verwendet wird, als Anlass genommen, die Frequenz der zu betrachtenden Signalanteile in der Längsbeschleunigung nach oben zu begrenzen. Die Literatur weist für die Übertragungsfunktion Fahrer- und Fahrzeug eine Schnittfrequenz von etwa $0.3\,\mathrm{Hz}$ in der Normalfahrt (Folgefahrt) und bis zu $3\,\mathrm{Hz}$ in kritischen Situationen aus [20, S. 743, 756]. Da der u.U. höherfrequente Eingriff in die Beschleunigung durch Fahrassistenzsysteme in kontrollierten Fahrsituationen nicht zu erwarten ist und eine Geschwindigkeitsregelung im städtischen Nahverkehr nicht zum Einsatz kommt, wird angenommen, dass deutlich oberhalb der angegebenen Frequenzen nur unerwünschte Signalanteile liegen. Diese Tatsache wird durch Messungen in Kapitel 7 belegt. Es wird zudem zur Vereinfachung das lineare Einspurmodell aus [20, S. 613 ff] verwendet werden,

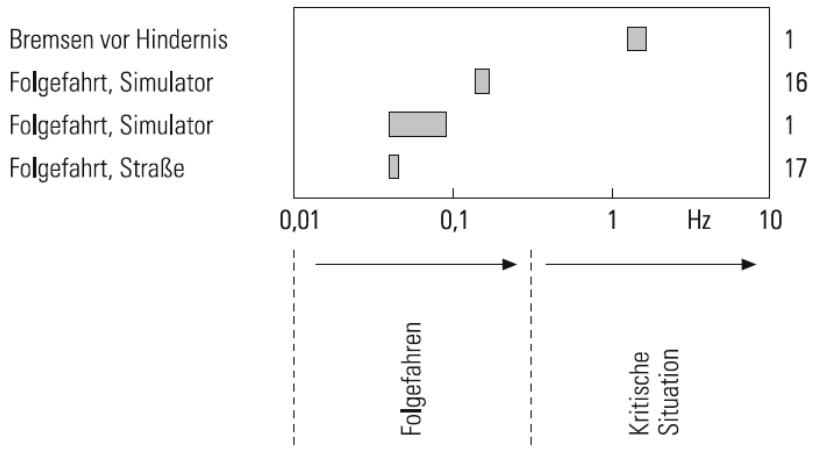


Abbildung 2.3.: Schnittfrequenzen des offenen Längsregelkreises [20, S. 756]

welches in Abb. 2.4 dargestellt ist. Wenn relevant, wird dabei angenommen, dass der Fahrzeugschwerpunkt in Fahrbahnhöhe liegt, sodass u.a. kein Wanken in der Kurvenfahrt auftritt. Die gemachten Vereinfachungen sind zulässig, wenn das Fahrverhalten in Normalsituationen beschrieben werden soll [20, S. 613]. Aufgrund der Anwendungsumgebung in Bussen des öffentlichen Nahverkehrs können Fahrsituationen an der Kraftschlussgrenze außenvor bleiben.

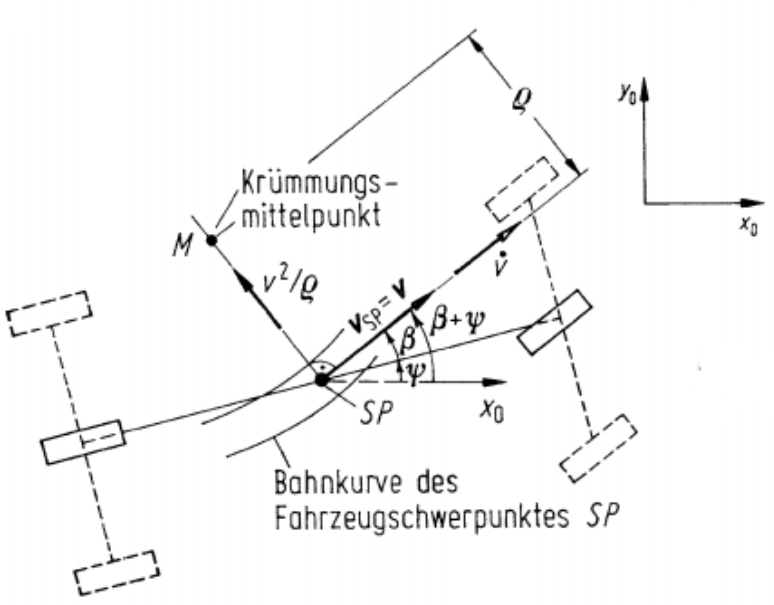


Abbildung 2.4.: Kinematische Größen an einem Einspurmodell [20, S. 615]

Geradeausfahrt mit konstanter Geschwindigkeit

Bei der Geradeausfahrt mit konstanter Geschwindigkeit liegt das in 2.1.1 beschriebene Kräftegleichgewicht zwischen Zugkraft und Widerstandskräften vor. Dabei wird die Erdbeschleunigung anteilig mit $a_{St} = \sin\alpha \cdot g$ auf der Längsachse des Fahrzeuges wirksam. Bei positiven Steigungen α ist diese durch die motorische Zugkraft zusätzlich zu überwinden ($F_{St} > 0$), bei negativen Steigungen beschleunigt sie in Fahrtrichtung, sodass die kinetische Energie durch mechanische Bremsen in Wärme umgewandelt oder z.B. zur Rekuperation abgeführt werden muss um die Geschwindigkeit konstant zu halten ($Z < 0$).
Es ist hier insbesondere der Fall des haltenden Fahrzeuges zu berücksichtigen. Während die Kraft durch die anteilig wirkende Erdbeschleunigung während der Fahrt durch die Zugkraft des Motors überwunden und während der Abwärtsfahrt durch Bremsen umgewandelt wird, wirkt während des Haltens an einer Steigung die Haftreibung der mechanischen Bremse der Beschleunigung entgegen. Da durch das Halten lediglich potentielle Energie erhalten, aber keine Energie umgewandelt wird, ist die Erdbeschleunigung in diesem Fall für die energetische Betrachtung des Antriebs nicht relevant.
Es treten in diesen Fahrsituationen lediglich Beschleunigungen entlang der Längsachse des Fahrzeugs auf.

Geradeausfahrt mit veränderlicher Geschwindigkeit

Für die Geradeausfahrt mit veränderlicher Geschwindigkeit gelten zunächst dieselben Annahmen wie für die mit konstanter Geschwindigkeit. Es tritt lediglich eine Überschusszugkraft, wie in 2.1.1 beschrieben, entlang der Längsachse auf, welche zu einer Geschwindigkeitsänderung des Fahrzeugs führt. Es wirkt eine Beschleunigung ungleich des Erdbeschleunigungsanteils. Auch hier treten in der Modellannahme lediglich Kräfte entlang der Längsachse auf.

Kreisfahrt mit konstanter Geschwindigkeit

Die Fahrt des Fahrzeuges auf einer Kreisbahn mit festem Radius ρ soll als Näherung für die Kurvenfahrt betrachtet werden. In der Kreisfahrt treten die Maximalwerte der während der Kurvenfahrt zu betrachtenden Einflüsse auf. In der Kreisfahrt tritt zusätzlich zu den bereits betrachteten Beschleunigungen entlang der Fahrzeuglängsachse eine Beschleunigung quer zum Fahrzeug auf, die Zentripetalbeschleunigung $\frac{v^2}{\rho}$. Sie wirkt vom Fahrzeugschwerpunkt zum Krümmungsmittelpunkt, wie in Abb. 2.4 dargestellt. [20, S. 625] Die Zentripetalbeschleunigung für einen Bus in Kreisfahrt soll in dieser Betrachtung auf ein Maximum von $0.4\,\mathrm{g} \approx 3.924\,\frac{\mathrm{m}}{\mathrm{s}^2}$ begrenzt werden, da dies als Maximalwert auf trockenen Straßen für LKW

nach [20, S. 627] und aufgrund des Schwerpunktes als Grenzwert für Reisebusse gemäß [26, S. 113] angenommen werden kann.

Wie in Abb. 2.4 dargestellt, weicht die Längsachse des Fahrzeugs um den sog. Schwimmwinkel β von der Tangente am Kreis ab. Entsprechend zeigt sich die Zentripetalbeschleunigung zu Anteilen sowohl auf der Längs- als auch der Querachse des Fahrzeugs. Der Schwimmwinkel kann nach [26, S. 113] am Beispiel eines Reisebusses für die gegebenen Querbeschleunigungen als auf $-2\,\mathrm{deg} < \beta < 2\,\mathrm{deg}$ begrenzt angenommen werden. Unter den angenommenen Maximalwerten liegt der Fehler der Längsbeschleunigung durch die Zentripetalbeschleunigung bei:

$$a_{L,err} = \frac{v^2}{\rho} \cdot \sin\beta = 3.924\,\frac{\mathrm{m}}{\mathrm{s}^2} \cdot \sin\pm 2^\circ \approx \pm 0.137\,\frac{\mathrm{m}}{\mathrm{s}^2}$$

Unter welchen Bedingungen dieser Fehler vernachlässigt werden kann, wird in 7.2.4 diskutiert.

Kreisfahrt mit veränderlicher Geschwindigkeit

Bei der Betrachtung einer Kreisfahrt mit veränderlicher Geschwindigkeit ist zudem die Beschleunigung $a = \dot{v}$ aus Abb. 2.4 zu berücksichtigen. Diese entsteht durch Bremsen oder motorisches Erhöhen der Zugkraft. Durch den Schwimmwinkel teilt sich diese Beschleunigung in einen Quer- und Längsbeschleunigungsanteil auf. Es liegen bei maximalem Schwimmwinkel

$$\cos\beta = \cos\pm 2\,\mathrm{deg} = 99.94\,\%$$

der relevanten Beschleunigung anteilig auf der Fahrzeuglängsachse an.

2.1.3. Ermittlung durch Positionsbestimmung

Die durch Energiezu- oder abfuhr hervorgerufene Geschwindigkeitsänderung des Fahrzeuges kann u.a. durch die periodische Positionsbestimmung ermittelt werden. Dazu wird verwendet, dass die Geschwindigkeit die Ableitung der Strecke nach der Zeit

$$v = \dot{s} = \frac{\mathrm{d}s}{\mathrm{d}t}$$

ist. Zur numerischen Berechnung ist die Voraussetzung dafür, dass die Zeitdauer Δt zwischen je zwei Positionsbestimmungen sowie der zurückgelegt Weg zwischen drei bestimmten Positionen jeweils als $\Delta s = s(t) - s(t - \Delta t)$ bekannt sind. Die mittlere Geschwindigkeit

auf dem zurückgelegten Streckenabschnitt kann dann mit dem Differenzenquotienten

$$v \approx \frac{\Delta s}{\Delta t}$$

bestimmt werden [44, S. 49].
Analog kann aus den Geschwindigkeiten zu zwei Zeitpunkten die mittlere Beschleunigung des Körpers aufgrund von

$$a = \dot{v} = \frac{\mathrm{d}v}{\mathrm{d}t}$$

durch

$$a \approx \frac{\Delta v}{\Delta t}$$

numerisch ermittelt werden. Oder zusammen, als zentraler Differenzenquotient für die zweite Ableitung [44, S. 50]:

$$a \approx \frac{s(t - 2\Delta t) - 2s(t - \Delta t) + s(t)}{(\Delta t)^2}$$

Für die Betrachtung des Fehlers ergibt sich bei numerischer Differentiation mittels des zentralen Differenzenquotienten [44, S. 49]:

$$f''(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} + \mathcal{O}(h^2)$$

Der Fehler wächst also quadratisch mit dem Zeitabstand zwischen zwei Positionsermittlungen.

Anhand des beschriebenen Vorgehens ist zu erkennen, dass Sprünge in der Positionsbestimmung zu einem einzelnen Zeitpunkt, also ein einzelner Fehler in der Positionsbestimmung, auch wenn schon die nächste Positionsbestimmung wieder korrekte Werte liefert, zu einem großen Δs und damit zu einer sehr großen mittleren Beschleunigung führen. Da eine große Beschleunigung, auch für einen kurzen Zeitraum zwischen zwei Positionsbestimmungen, eine Zufuhr großer Mengen kinetischer Energie bedeutet, sind diese Fehler bei einer energetischen Betrachtung der vom Motor aufgebrachten Drehmomente von erheblicher Bedeutung. Für den gleichen Positionsfehler wird der Einfluss mit höheren Abtastraten und damit kleinerem Δt größer.

2.2. Besonderheiten der Satellitennavigation

Eine globale Positionsbestimmung ist heute mittels globaler Navigationssatellitensysteme (GNSS) mit geringem Hardwareaufwand möglich. Die gängigen in Betrieb befindlichen Vertreter sind das amerikanische NAVSTAR GPS und das russische GLONASS, welche durch

voneinander unabhängige Infrastruktur die satellitengestützte Ortung erlauben. Dabei stehen im Falle von GPS 32 aktive [31, S. 46], bei GLONASS zur Zeit 24 Satelliten [31, S. 104] zur Verfügung. Durch die höhere Satellitenverfügbarkeit ist die Genauigkeit der Ortung durch GPS besser als die mittels GLONASS. Eine weitere Verbesserung ist durch die gemeinsame Verwendung von GLONASS und GPS in einem Empfänger zu erzielen [31, S. 105].

Beide GNSS setzen dabei Verfahren auf der Grundlage von Signallaufzeitmessungen zur Positionsbestimmung ein. Durch die Ermittlung der Distanz aus der Signallaufzeit des Funksignals von mindestens vier Satelliten und der Kenntnis über deren Position lässt sich die absolute Position des Empfängers dreidimensional bestimmen. GPS und GLONASS Satelliten übermitteln dabei die eigene und die Position anderer Satelliten sowie Informationen zur Satellitenumlaufbahn in den Nachrichten, die auch zur Laufzeitmessung am Empfänger verwendet werden. Die Genauigkeit der Positionsbestimmung ist dabei vor allem von der Genauigkeit der Laufzeitmessung abhängig [31, S. 5]. Dabei beziehen sich die Positionsangaben auf die Referenzellipsoide WGS 84 (GPS) und PZ-90 (GLONASS) [31, S. 108], werden aber je nach Empfänger u.U. in das jeweils andere Bezugssystem überführt. Die Referenzellipsoide stellen Näherungen an die Geoid-Form der Erde dar, sodass ortsabhängig vor allem in der Höhe über dem Geoid große Abweichung entstehen können.

Besonders die unterschiedlichen Frequenzen zur zivilen Verwendung von $1242.9375\,\mathrm{MHz}$ bis $1249.0625\,\mathrm{MHz}$ für GLONASS und $1575.42\,\mathrm{MHz}$ für GPS unterscheiden die Systeme [31, S. 64, 105]. Außerdem verwenden die Systeme unterschiedliche Verfahren zur Unterscheidung der einzelnen Satelliten: GPS verwendet verschiedene Codes auf der gleichen Trägerfrequenz (CDMA) während GLONASS für verschiedene Satelliten unterschiedliche Frequenzen verwendet [31, S. 107].

Fehler entstehen neben Abweichungen des Geiods vom Referenzellipsoids u.a. durch

- Die Abweichung der Satellitenbahnen von den vorberechneten
- Die Brechung der Signale an der Ionosphäre (Ionosphärische Refraktion) mit einem Fehler von 5 bis $15\,\mathrm{m}$ [31, S. 83]
- Laufzeitverzögerung durch die Troposphäre mit einem Fehler von bis zu $10\,\mathrm{m}$ [31, S. 84]
- Ungünstige Satellitenkonstellationen [31, S. 85]
- Mehrwegeffekte mit Fehlern von 1 bis $100\,\mathrm{m}$ [31, S. 86]

Die Abweichung der Satellitenbahnen ergibt sich z.B. durch das ungleichmäßige Schwerefeld der Erde und führt durch ungenaue Informationen über die Position des Satelliten auch zu einem Fehler in der Ortung des Empfängers [31, S. 81 ff].

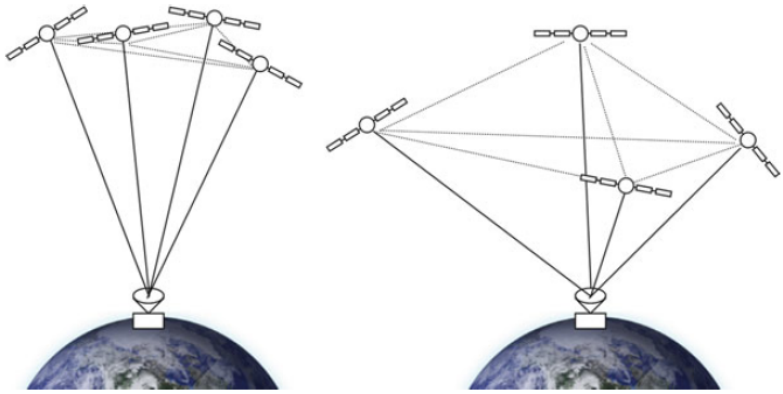


Abbildung 2.5.: links: Schlechte, rechts: Gute Satellitenkonstellation [31, S. 85]

Fehler durch Brechung des Signals an der Ionosphäre können durch Zusatzinformationen über den Einfluss der Ionosphäre, z.B. durch die Messung der Laufzeit eines Signals mit anderer Frequenz, korrigiert werden. Diese Möglichkeit steht im Moment nur unter Nutzung der militärischen Frequenzen L2 von GPS und GLONASS zur Verfügung [31, S. 81 ff].

Einflüsse der Troposphäre sind weitestgehend frequenzunabhängig und können daher nur schwer korrigiert werden. Fehler in der oben angegebenen Größenordnung treten allerdings nur dann auf, wenn das Signal einen vergleichsweise weiten Weg durch die Troposphäre zurücklegen muss, der empfangene Satellit also nur wenig über dem Horizont steht. Es ist deshalb die Verwendung von Satelliten in einer günstigeren Konstellation zu bevorzugen, wenn diese verfügbar sind. Es ist dabei eine Konstellation optimal, bei der die Satelliten nicht zu nah beieinander stehen, jedoch auch keinen zu kleinen Erhebungswinkel über dem Horizont haben. Dies ist z.B. gegeben, wenn ein Satellit direkt überhalb dem Empfänger im Zenit steht und drei weitere mit einem Winkel von je 120° darum. In Abb. 2.5 sind Satellitenpositionen im Vergleich dargestellt. Als Maß für die Qualität wird das Volumen des aufgespannten Körpers betrachtet, welches bei einer besonders guten Satellitenkonstellation maximal wird [31, S. 81 ff].

Zu den größten Fehlern in der Positionsbestimmung am Empfänger können Mehrwegeffekte durch die Reflexion der Satellitensignale an z.B. Gebäudewänden führen. Zur fehlerhaften Berechnung führen dabei die durch den „Umweg“ verlängerten Signallaufzeiten. Dieser Fehler kann häufig vom Empfänger erkannt und bei Verfügbarkeit genügender Satelliten korrigiert werden. Der Fehler reduziert sich dann auf etwa $5\ \mathrm{m}$ [31, S. 81 ff].

Viele GNSS-Empfänger verwenden zudem umfangreiche Filter und Zustandsschätzer um die Qualität der Positionsbestimmung zu verbessern. Es sei z.B. auf die Möglichkeit der Einstellung von sog. „Dynamic Platform Models“ bei GNSS-Empfängern des Herstellers µBlox

verwiesen, welche die zur Korrektur verwendeten Modellparameter an den Einsatzzweck auf Booten in Fahrzeugen, Flugzeugen oder bei Fußgängern anpassen [42, S. 9].

Nachteile bei der Betrachtung der Beschleunigung eines Fahrzeuges liegen hier vor allem in dem sehr großen Fehler durch Positionssprünge, wie in 2.1.3 beschrieben. Diese können durch die oben angeführten Fehler in der Positionsbestimmung durchaus auftreten. Der Fehler bleibt zunächst aber auf einzelne Messwerte und -zeitpunkte beschränkt, sodass sich daraus im Weiteren kein ansteigender Fehler in der Aussage über den kinetischen Zustand des Fahrzeugs ergibt.

2.3. Inertialmesssysteme

Eine andere Möglichkeit zur Messung der Fahrzeugbeschleunigung stellt die Messung durch Inertialsensoren dar. Dabei bestehen Sensorsysteme zur Erfassung von Bewegungen im dreidimensionalen Raum mit sechs Freiheitsgraden zumeist aus drei orthogonalen Beschleunigungssensoren sowie drei Drehratensensoren. Außerdem kommen häufig Magnetfeldsensoren zur Erfassung des Erdmagnetfeldes zum Einsatz.

Die Beschleunigungsmessung erfolgt dabei durch die Messung der Wirkung der Beschleunigung auf ein gedämpftes Feder-Masse-System. Abb. 2.6 stellt beispielhaft eine durch eine Balkenfeder gedämpfte seismische Masse dar. Eine Beschleunigung des Gehäuses in z-Richtung führt zu einer Verschiebung der Masse gegen das Gehäuse, deren Auslenkung, welche direkt abhängig von der Größe der Beschleunigung ist [9, S. 544]. Um das Signal-Rausch-Verhältnis zu erhöhen, werden teilweise rückgekoppelte Systeme eingesetzt. Bei diesen wird in einem Regelkreis durch eine elektrisch hervorgerufene Kraft die Rückstellung der seismischen Masse bewirkt [9, S. 546]. Unterschiedliche Sensoren, meist ausgeführt als mikroelektromechanische Systeme (MEMS), unterscheiden sich dabei vor allem durch die Sensorik zur Erfassung der Auslenkung der seismischen Masse. Diese sind zumeist kapazitiv ausgeführt, wodurch konstruktiv die Umsetzung eines Regelkreises zur Rückstellung der Masse begünstigt wird. Es kommen aber z.B. auch induktive, piezoresistive oder piezoelektrische Sensoren zum Einsatz [9, S. 554 ff].
Es ist zu beachten, dass die vorgestellten Beschleunigungssensoren stets auch die Erd- sowie eventuelle Zentrifugalbeschleunigung messen. In der Verwertung von Messdaten aus den Beschleunigungssensoren gilt es also zu bewerten, ob gemessene Beschleunigungen tatsächlich zu zu- oder abgeführter kinetischer Energie führen oder bei der energetischen Betrachtung nicht berücksichtigt werden dürfen. Um diese Bewertungen vorzunehmen sollen die Vorüberlegungen aus 2.1.2 dienen.

Mit Drehratensensoren wird die Rotationsgeschwindigkeit Ω in $\frac{\mathrm{deg}}{\mathrm{s}}$ gemessen. Dabei kommen im Wesentlichen MEMS zum Einsatz, welche die Drehrate über die Corioliskraft er-

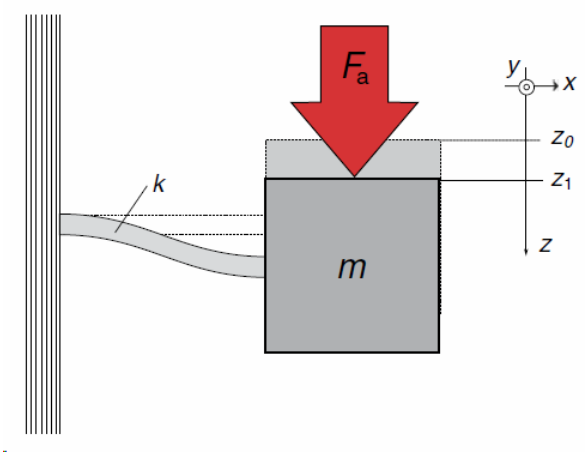


Abbildung 2.6.: Schematische Auslenkung einer Balkenfeder durch die Beschleunigung a der Masse m in z-Richtung [9, S. 545]

mitteln sowie Faser- und Ringlaser-Kreisel, die über den Sagnac-Effekt messen. Während die MEMS sehr günstig und klein ausgeführt werden können, erreichen optische Faser- oder Ringlaser-Kreisel als Drehratensensoren, bei größerer Bauform, eine deutlich höhere Genauigkeit und vor allem kleinere Bias-Fehler [9, S. 560]. Da zumeist nicht die Rotationsgeschwindigkeit, sondern der Rotationswinkel von Interesse ist, erfolgt eine Integration der Messwerte über die Zeit, sodass ein Bias-Fehler mit zunehmender Dauer der Integration zu einem größer werdenden Winkelfehler (Drift) führt. Diese Fehler liegen bei MEMS in der Größenordnung von $10\,\frac{\text{deg}}{\text{h}}$, bei Faser-Kreiseln bei etwa $1\,\frac{\text{deg}}{\text{h}}$ und Ringlaser-Kreisel erreichen eine Drift von $< 0.001\,\frac{\text{deg}}{\text{h}}$ [4]. Der Einsatz von Drehratensensoren zur Ermittlung der Rotationsgeschwindigkeit und -winkel, relativ zu einer definierten Null-Lage, ist z.B. zur korrekten Berücksichtigung der Erdbeschleunigung, an Steigungen und Gefällen, sowie zur Korrektur von Fehlern, durch einen Eintrag der Querbeschleunigung in die gemessene Längsbeschleunigung bei Kurvenfahrt, (Abschnitt 2.1.2) sinnvoll. Nur mit der zusätzlichen Drehrateninformation kann die tatsächliche Lage im Raum ermittelt werden.

Der Vorteil der Ermittlung von energetisch relevanten Beschleunigungen durch ein Inertialmesssystem liegt in der Verfügbarkeit der Messwerte, die nicht an den Empfang von Satelliten gebunden ist. Dadurch treten u.a. keine kurzzeitigen großen Fehler in der Beschleunigung auf, wie sie z.B. bei Sprüngen in der Positionsbestimmung zu erwarten sind.
Im Gegenzug muss dafür genau beachtet werden, welche gemessenen Beschleunigungsanteile tatsächlich zu einer Geschwindigkeitsänderung des Fahrzeuges führen. Werden andere Beschleunigungsanteile berücksichtigt, führt dies bei integrativer Verarbeitung der Messdaten zu einem über die Zeit zunehmenden Fehler.

3. Anforderungen

3.1. Anforderungen Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme (IVI)

Zu den relevanten Stakeholdern gehört das Fraunhofer IVI, welches für das Projekt BEEDeL, mittels der erfassten Daten die Tauglichkeit einzelner Buslinien der Hamburger Hochbahn für die Elektrifizierung untersuchen möchte. Es kommt dabei ein MATLAB-Modell zum Einsatz, welches das energetische Verhalten eines Elektrobusses annähern kann. Berücksichtigung finden dabei u.a.

- Der Streckenverlauf und die Position von Haltestellen zur Ermittlung von geeigneten Orten für Batterieladestationen
- Das Geschwindigkeitsprofil des Busses entlang der Strecke sowie Informationen über die Höhe, zur Bestimmung des Energiebedarfs auf Teilabschnitten und der Gesamtstrecke
- Strecke und Dauer der Fahrt auf der Linie
- Haltezeiten an Haltestellen und Wendepunkten um die möglichen Ladezeiten ohne Störung des Betriebs zu bestimmen
- Umgebungstemperaturen zur Kennzeichnung der Betriebsbedingungen für Batterien sowie des Bedarfs an Klimatisierung im Bus

Die in Tab. 3.1 angegebenen Daten fordert das Fraunhofer IVI aus dem Datenlogger.

Als Anforderungen an Speicherung und Datenübertragung werden die folgenden formuliert:

- Ausgabe in MATLAB-Datei
- Speicherung an den Endhaltestellen oder auf dem Betriebshof (z.B. bei Zündung aus)
- Dateiname: Stadt_Linie_Datum_Startzeit_Besonderheiten.mat
- Speicherung auf einem Wechseldatenträger mit Speicherkapazität für die Messungen von 1-2 Wochen

Datum	Abtastrate	Einheit	Auflösung
Geographische Länge	$10\,\mathrm{Samples/s}$	deg	$1 \times 10^{-6}\,\mathrm{deg}$
Geographische Breite	$10\,\mathrm{Samples/s}$	deg	$1 \times 10^{-6}\,\mathrm{deg}$
Höhe über NN	$10\,\mathrm{Samples/s}$	m	$0.1\,\mathrm{m}$
Geschwindigkeit aus GNSS	$10\,\mathrm{Samples/s}$	$\frac{\mathrm{km}}{\mathrm{h}}$	$0.1\,\frac{\mathrm{km}}{\mathrm{h}}$
UTC Datum und Startzeit d. Messung	$10\,\mathrm{Samples/s}$	-	$0.1\,\mathrm{s}$
Anzahl der Satelliten (GNSS)	$10\,\mathrm{Samples/s}$	-	-
Kennzeichnung von Haltestellen	$10\,\mathrm{Samples/s}$	-	-

Tabelle 3.1.: Anforderungen des Fraunhofer IVI an die Datenausgabe aus dem Datenlogger

Weiter werden zudem betriebsseitige Anforderungen, wie die Versorgung aus $18\,\mathrm{V}$ bis $32\,\mathrm{V}$ DC, Einhaltung des Automotive-Standards für Steckverbinder sowie an Temperaturbereiche und Staub-, Feuchtigkeits-, Schlagfestigkeit angegeben.

3.2. Anforderungen BATSEN

Auch das Projekt BATSEN der HAW Hamburg möchte durch Daten aus dem Datenlogger auf Nahverkehrsbussen einen Wissensgewinn erzielen. Dabei liegt der Fokus auf der in 2.1.1 beschriebenen direkten Beziehung zwischen Beschleunigung und Stromaufnahme des Motors sowie des, zusätzlich durch Nebenverbraucher beinflussten, Batteriestroms. Die ermittelten Beschleunigungswerte sollen, anhand eines im Rahmen von anderen Arbeiten im Projekt zu entwickelnden Modells, zur Zyklierung von Batterien verwendet werden können. Auf diese Weise soll das Verhalten verschiedener Batteriezellen unter dem Belastungsprofil von Elektrobussen untersucht werden.

Im BATSEN Projekt wird, aufgrund der in 1.2 und 2.1.1 festgestellten Zusammenhänge, der Bedarf an den in Tab. 3.2 dargestellten Daten herausgearbeitet. Dabei soll die Erfassung von Strömen und Spannungen an Batterie und Antriebsmotor als Ausbaumöglichkeit vorgesehen werden. Zusätzlich wird festgestellt, dass eine Möglichkeit zur Fernüberwachung des Systems bzw. Fernwarnung bei Funktionsfehlern wünschenswert ist. Das Bedienkonzept soll die Verwendung eines LCD-Touchscreens zulassen.

3.3. Ziel-Spezifikation

Schon bei der Formulierung der Zielspezifikation sollen hier Anforderungen mit sehr geringer Priorität von erfolgskritischen getrennt werden. Als Teil der Ziel-Spezifikation für den Daten-

Datum	Abtastrate	Einheit	Auflösung
Energetisch relevante Fahrzeugbeschleunigung	100 Samples/s	$\frac{m}{s^2}$	0.35 $\frac{m}{s^2}$
Batterietemperatur	0.2 Samples/s	°C	0.5 °C
Außentemperatur	0.2 Samples/s	°C	0.5 °C
Innentemperatur	0.2 Samples/s	°C	0.5 °C
Batteriestrom	–	A	–
Batteriespannung	–	V	–
Motorstrom	–	A	–
Motorspannung	–	V	–

Tabelle 3.2.: Anforderungen des BATSEN Projekts an die Datenausgabe aus dem Datenlogger

logger ergibt sich die in Tab. 3.3 aufgeführte Liste von zu erfassenden Daten, die kritisch für die Einsatztauglichkeit des Gerätes sind. Das Gerät soll zudem die folgenden Anforderungen erfüllen:

- Speicherung in MATLAB-Datei
- Speicherung der Daten auf einem Wechseldatenträger
- Möglichkeit zur Fernkommunikation mit dem Gerät
- Prinzipielle Erweiterbarkeit um vorhandene Strom- und Spannungssensoren

Alle weiteren Anforderungen aus den betroffenen Forschungsprojekten sind nicht erfolgskritisch, sollen allerdings bei der Entwicklung berücksichtigt werden.

Datum	Abtastrate	Einheit	Auflösung
Geographische Länge	10 Samples/s	deg	1×10^{-6} deg
Geographische Breite	10 Samples/s	deg	1×10^{-6} deg
Höhe über NN	10 Samples/s	m	0.1 m
Geschwindigkeit aus GNSS	10 Samples/s	$\frac{km}{h}$	0.1 $\frac{km}{h}$
UTC Datum und Startzeit d. Messung	10 Samples/s	-	0.1 s
Anzahl der Satelliten (GNSS)	10 Samples/s	-	-
Energetisch relevante Fahrzeugbeschleunigung	100 Samples/s	$\frac{m}{s^2}$	0.35 $\frac{m}{s^2}$
Batterietemperatur	0.2 Samples/s	°C	0.5 °C
Außentemperatur	0.2 Samples/s	°C	0.5 °C
Innentemperatur	0.2 Samples/s	°C	0.5 °C

Tabelle 3.3.: Kritische Anforderungen an die Datenausgabe aus dem Datenlogger gemäß Ziel-Spezifikation

3.4. Vorarbeiten

Im Projekt steht bereits ein von Thom [39] entwickelter Datenlogger zur Verfügung, welcher in der Lage ist Beschleunigungen auf drei orthogonalen Achsen zu messen. Zwar ist das Gerät als funktionsfähig zur Aufzeichnung von Beschleunigungswerten getestet worden, aufgrund folgender Feststellungen soll allerdings eine Neuentwicklung auf Basis der erlangten Erkenntnisse erfolgen:

- Empfangsprobleme durch eine im Gerät verbaute GPS-Antenne
- Es soll untersucht werden, ob umfangreichere Inertialmesssysteme zur Verfügung stehen, welche weitere Informationen über Bewegung und Lage des Fahrzeugs zulassen
- Die für den verwendeten Mikrocontroller zur Verfügung stehende Treiber-Bibliothek StellarisWare wird vom Hersteller Texas Instruments nicht mehr unterstützt

4. Lösungskonzept

Im Rahmen der Findung eines Lösungskonzepts sollen, auf Basis der herausgearbeiteten Anforderungen, Systemkomponenten identifiziert sowie Schnittstellen und Hardwarekomponenten bestimmt werden. Dabei werden Entscheidungen der Ausführung in Hard- und Software, die nicht relevant für die Konzeptfindung sind, in den Kapiteln 5 und 6 dokumentiert.

4.1. Identifikation nötiger Systemkomponenten

Zunächst wird anhand der Zielspezifikation festgestellt, dass zumindest folgende Systemkomponenten benötigt werden, um die Grundfunktionen des Datenloggers umzusetzen:

- Inertialsensoren zur Erfassung der Beschleunigung
- GNSS-Empfänger zur Positionsbestimmung
- Mobilfunk-Modul für Fernwartung/Datenabruf
- Temperatursensoren zur Messung von Innen-, Außen- und Batterietemperatur
- Controller zur Ablaufsteuerung und Zusammenführung der Daten
- Wechselspeicher zur Speicherung der anfallenden Daten
- Akkumulator zum Betrieb ohne externe Spannungsversorgung
- LCD-Touchscreen zur Umsetzung einer umfangreicheren Mensch-Maschine-Schnittstelle (HMI)

Als Ausgangspunkt für die Konzeptionierung wird angenommen, dass der Controller wie in Abb. 4.1 mit jeder Komponente über einen eigenen Kanal kommuniziert und die Ablaufsteuerung als zentrales Element übernimmt. Eine Festlegung der Hardware-Schnittstellen und Gruppierung zu Hardware-Modulen kann erst nach der Komponentenauswahl in Kapitel 5 erfolgen.

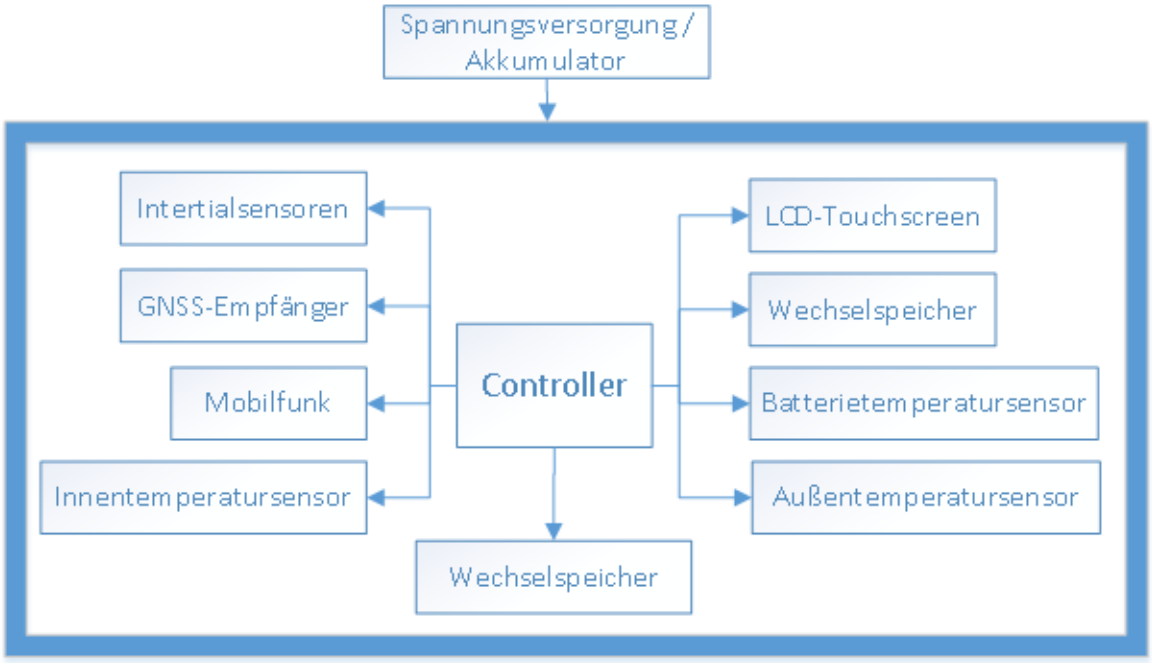


Abbildung 4.1.: Schematische Darstellung der nötigen Komponenten

4.2. Komponentenauswahl

4.2.1. Inertialsensoren

Zur Auswahl der Inertialsensoren werden zunächst verschiedene Typen integrierter Inertialmesssysteme in Tab. 4.1 in ihren entscheidungsrelevanten Eigenschaften verglichen. Wie in Abschnitt 2.3 dargelegt, birgt die Verfügbarkeit der Drehraten großes Potential zur Verbesserung der Messergebnisse zur tatsächlich zu- und abgeführten Energie, weshalb die Erfassung dieser Daten erfolgen soll. Die Nachteile eines Sensorsystems mit analogen Ausgängen sind vor allem die Schnittstellenbelegung durch den Bedarf von zumindest 6 ADC-Kanälen sowie ein erhöhter Schaltungsaufwand mit dem Risiko, durch externe Beschaltung die Signaleigenschaften z.B. durch höhere Störanfälligkeit zu beeinflussen. Der in der Arbeit von Thom [39] verwendete 3-Achsen-Beschleunigungssensor vom Typ BMA020 des Herstellers Bosch Sensortec könnte zwar zusammen mit einem weiteren 3-Achsen-Drehratensensor eingesetzt werden, es soll aber aufgrund des geringeren Strom- und Platzbedarfs sowie der besseren Verfügbarkeit auf ein einzelnes vollständig integriertes Sensorsystem zurückgegriffen werden. Diese sind preisgünstig zu beziehen, da derartige Sensorsysteme, meist ergänzt um Temperatur- und Magnetfeldsensoren, weite Verbreitung in Mobiltelefonen gefunden haben.

	3-Achsen Beschleunigung	3-Achsen Beschleunigung + 3-Achsen Drehrate
Beschleunigung Längsachse	Ja	Ja
Lage und Position in 6 Freiheitsgraden	Nein	Ja
Benötigte Schnittstellen (analog/digital)	3 ADC / 1 (SPI / I^2C)	6 ADC / 1 (SPI / I^2C)

Tabelle 4.1.: Vergleich der Eigenschaften verschiedener Typen von integrierten Inertialmessystemen

Bei der Auswahl des zu verwendenden ICs werden, nach Recherche, das MPU-9250 vom Hersteller InvenSense [10] und LSM330 von STMicroelectronics in einen Vergleich gezogen. Bei der Vorauswahl fand die Verfügbarkeit und vor allem die Verbreitung von quelloffener Anwendungssoftware verschiedenen Typs Berücksichtigung, sodass die zur Implementierung von Protokollen benötigten Informationen sicher vollständig zur Verfügung stehen.
In der Gegenüberstellung der beiden Sensorsysteme 4.2 zeigen sich deutliche Vorteile des MPU-9250 in der Auflösung der Beschleunigungssensoren, in anderen Eigenschaften stimmen die Produkte weitestgehend überein. Auch die Tatsache, dass im MPU-9250 drei Magentfeldsensoren zur Messung des Erdmagnetfeldes enthalten sind, macht diesen für den Einsatz attraktiver. Die Möglichkeiten in Zukunft eine weitreichendere Sensorfusion durchzuführen werden dadurch deutlich erweitert.

	MPU-9250	LSM330
Maximale Auflösung Beschleunigung	12 Bit	16 Bit
Maximale Auflösung Drehrate	16 Bit	16 Bit
Messbereich Beschleunigung	$\pm 2\,g$ bis $\pm 16\,g$	$\pm 2\,g$ bis $\pm 16\,g$
Messbereich Drehraten	$\pm 250\,\frac{deg}{s}$ bis $\pm 2000\,\frac{deg}{s}$	$\pm 250\,\frac{deg}{s}$ bis $\pm 2000\,\frac{deg}{s}$
Magnetfeldsensoren	Ja	Nein
Schnittstelle	SPI / I^2C	SPI / I^2C
Preis	$\approx 5.50\,EUR$	$\approx 5.00\,EUR$

Tabelle 4.2.: MPU-9250 (InvenSense) und LSM330 (STMicroelectronics) im Vergleich

4.2.2. Empfänger zur Satellitennavigation

Zunächst soll auch bei den GNSS-Empfängern ein Vergleich verschiedener Ausstattungen und Ausführungen, so wie am Markt verfügbar, erfolgen. Dabei sind vor allem die folgenden Fragen zu klären:

1. Sollen die Empfänger gemeinsam mit den Antennen verbaut werden?
2. Soll ein fertig aufgebautes Modul inklusive Antenne verwendet werden?
3. Wird die gleichzeitige Positionsbestimmung aus Satelliten des GPS und GLONASS gefordert?

Zur Beantwortung ist es nötig, Annahmen über die Einsatzbedingungen des Datenloggers anzustellen. Dazu wird auf Aussagen des Frauenhofer IVI sowie Gesprächsergebnisse aus dem Auftakt-Meeting zum BEEDeL Projekt zurückgegriffen. Da das Gerät beim Betrieb in Nahverkehrsbussen der Hochbahn AG vermutlich unter dem Dach und hinter Plastikverkleidung montiert wird, liegen direkt am Verbauort sehr ungünstige Bedingungen für den Empfang von GNSS-Satelliten vor. Damit ergibt sich, dass die GNSS-Antenne räumlich abgesetzt vom Hauptgerät an einem Ort mit besseren Empfangsbedingungen, z.B. auf dem Fahrzeugdach oder an der Windschutzscheibe zu montieren ist.
Da die Fahrzeuglänge je nach Einbauorten u.U. zu großen Kabellängen von mehr als $10\,\mathrm{m}$ führen kann, wird entschieden, Antenne und Empfänger gemeinsam zu verbauen. Es soll so der analoge Übertragungsweg der hochfrequenten und ohnehin schwachen Satellitensignale möglichst kurz (mm bis wenige cm) gehalten werden. Die Übertragung von Positionsinformationen aus dem Empfänger kann dann digital erfolgen und somit auf die nötige Übertragungsstrecke ausgelegt werden.
Bei Verwendung eines fertig aufgebauten Empfängermoduls inklusive Antenne können die Ergebnisse der mit Hochfrequenzsignalen erfahrenen Hersteller genutzt werden, sodass das Risiko minimiert wird, durch Verarbeitungsmängel bei der Bestückung und fehlerhafte Anpassung der Antenne die Satellitensignale ungünstig zu beeinflussen. Um diesen Erfahrungsvorsprung der Hersteller zu nutzen soll nach einem fertig aufgebauten Modul inklusive Antenne gesucht werden.
Im Kontext der Datenverarbeitung in Modellen zur Bestimmung von zu- und abgeführter kinetischer und potentieller Energie ist es nach Abschnitt 2.1.3 wichtig, dass die Positionsbestimmung möglichst lückenlos mit der geforderten Abtastrate erfolgt und keine großen Positionsabweichungen in Form von Sprüngen auftreten. Es ist deshalb wünschenswert durch möglichst viele verfügbare Satelliten eine genaue Bestimmung auch bei teilweiser Abschattung zu ermöglichen. Die Entscheidung fällt deshalb auf einen Empfänger, der GPS und GLONASS unterstützt.

Nach Vergleich vieler am Markt verfügbarer Empfänger-Module stellt sich heraus, dass die Anforderung an die Abtastrate von $10\,\mathrm{Samples/s}$ nur von sehr wenigen erfüllt wird. Zwei

Module, die die Anforderungen prinzipiell erfüllen, werden in Tab. 4.3 verglichen. Da lediglich das CAM-M8Q der Firma uBlox als fertiges Modul mit Antenne zu beziehen ist, soll dieses Verwendung finden. Über ein proprietäres, binäres Protokoll [42] können Einstellungen am Modul vorgenommen und Daten ausgelesen werden, sodass der Funktionsumfang den eines nur mit Protokoll nach NMEA 0183 zu verwendenden Empfängers weit überschreitet.

	uBlox CAM-M8Q	Furuno GV-87
GPS	ja	ja
GLONASS	ja	ja
Protokolle	NMEA / Proprietär	NMEA
Koppelnavigation	nein	ja
Anzahl Kanäle	72	26
Horizontale Genauigkeit	2.5 m	2.5 m
Schnittstelle	UART	UART

Tabelle 4.3.: GNSS-Empfänger uBlox CAM-M8Q und Furuno GV-87 im Vergleich

Die große Fahrzeuglänge der Busse soll zudem genutzt werden, indem zwei GNSS-Empfänger gleichen Typs an unterschiedlichen Stellen (z.B. vorne und hinten) montiert werden. Dadurch soll erreicht werden, dass auch bei Abschattungen wie z.B. Unterführungen, die nur einen Teil des Fahrzeugs betreffen, noch zuverlässig die Position bestimmt werden kann. Die Qualität der Positionsbestimmung eines Empfängers kann dabei durch seine Ausgabewerte, wie u.a. die ungefähre horizontale Genauigkeit oder die Anzahl verwendeter Satelliten bewertet werden.

4.2.3. Mobilfunk-Modul

Um die Möglichkeit zur Fernkommunikation zu erhalten, soll ein Mobilfunk-Modul zum Einsatz kommen. Unter Berücksichtigung der sehr guten Mobilfunkabdeckung in Stadtgebieten kann angenommen werden, dass der Datenlogger im Bus damit nahezu durchgehend kommunikationsfähig bleibt. Zur Auswahl stehen verschiedene Module, die u.a. die Technologien LTE, UMTS und GPRS zur Datenübertragung unterstützen. Einige Module verschiedener Hersteller werden in Tab. 4.4 verglichen. Da zur Übertragung von Steuerbefehlen sowie einer möglichen Statusübermittlung durch den Datenlogger an einen Server nur geringe Datenmengen anfallen, reicht die Datenrate unter Verwendung von GPRS aus. Eine spätere Betrachtung der voraussichtlich anfallenden Datenmengen an Messdaten in Abschnitt 4.4.2

zeigt zudem, dass selbst eine direkte Übertragung der Messdaten bei Verfügbarkeit der vollen GPRS-Datenrate möglich wäre. Das gewählte Modul SIM800H der Firma SIMCom ist zudem im Vergleich zu Modulen mit anderen Technologien preisgünstig und unterstützt SMTP zum Versand von E-Mails.

	uBlox LISA-U230	uBlox TOBY-L210	**SIMCom SIM800H**
Technologien	GPRS/UMTS	GPRS/UMTS/LTE	GPRS
Max. Datenrate (Up- / Download)	$5.76\,\frac{\text{MByte}}{\text{s}}$ / $21.1\,\frac{\text{MByte}}{\text{s}}$	$50\,\frac{\text{MByte}}{\text{s}}$ / $100\,\frac{\text{MByte}}{\text{s}}$	$85.6\,\frac{\text{kByte}}{\text{s}}$ / $85.6\,\frac{\text{kByte}}{\text{s}}$
Audio	digital	digital	digital / analog
Protokolle (Software)	TCP/IP, UDP/IP, HTTP/FTP/SSL	TCP/IP, UDP/IP, HTTP/FTP/SSL	TCP/IP, UDP/IP, HTTP/FTP/SMTP/ POP3/SSL
Preis	$\approx 50\,\text{EUR}$	$\approx 65\,\text{EUR}$	$\approx 20\,\text{EUR}$
Schnittstellen	UART / USB	UART / USB	UART / USB

Tabelle 4.4.: Mobilfunk-Module uBlox LISA-U230, uBlox TOBY-L210 und SIMCom SIM800H im Vergleich

4.2.4. Temperatursensor

Die Temperatursensoren zur Messung von Außen-, Innen- und Batterietemperatur müssen so ausgelegt werden, dass sie entfernt vom Datenlogger verbaut werden können. Aus diesem Grund können keine NTC- oder PTC-Widerstände in Spannungsteilern in Verbindung mit der Messung eines Spannungsabfalls durch einen ADC im Datenlogger verwendet werden, wie es im Batterie-Zykliergerät [45] umgesetzt ist. Kabellängen von mehreren Metern würden den Spannungsabfall erheblich beeinflussen und damit die Messwerte unbrauchbar oder eine komplexe Kalibrierung und Korrektur nötig machen. Die Wandlung der Temperatur in einen digitalen Wert soll stattdessen am Messort erfolgen und über eine digitale Schnittstelle zum Datenlogger übertragen werden.

In Anlehnung an den durch Thom [39] ausgewählten Temperatursensor TSic 206 [11] der Firma IST AG soll ein über den 1-Wire-Bus [18] angebundener Anwendung finden. Eine Nachbildung der Schnittstelle ist mittels einer UART-Hardware möglich, sodass damit der Rechenaufwand für die Kommunikation minimiert werden kann. Die geringe Datenrate des 1-Wire-Bus ist für die Temperaturmessung mit der in den Anforderungen in Abschnitt 3.2 festgelegten Abtastrate ausreichend, wie in Abschnitt 4.4.1 gezeigt wird, und ermöglicht eine zuverlässige Übertragung auch über einige $10\,\text{m}$ ohne hohe Anforderungen an die Verkabelung zu stellen.

Nach einem Vergleich zweier verfügbarer 1-Wire-Temperatursensoren in Tab. 4.5 wird auf den leicht verfügbaren DS18B20 der Firma Maxim Integrated zurückgegriffen. Dieser bietet eine einstellbare Auflösung von $9\,\text{bit}$ bis $12\,\text{bit}$ über einen Temperaturbereich von $-55\,^{\circ}\text{C}$ bis $125\,^{\circ}\text{C}$ und ist damit etwas flexibler und preisgünstiger einsetzbar als der TSic 206. Außerdem ist die Genauigkeit auch für Temperaturen unter $0\,^{\circ}\text{C}$ spezifiziert und bei der Messung wird nicht mit schnellen Temperaturwechseln gerechnet.
Durch die Verwendung der gängigen 1-Wire-Schnittstelle können mit geringem Aufwand und Kosten auch andere Temperatursensoren verwendet werden, weshalb die Wahl nicht kritisch ist.

	Maxim Integrated DS18B20	TSic 206
Auflösung	$9\,\text{bit}$ bis $12\,\text{bit}$	$11\,\text{bit}$
Messbereich	$-55\,^{\circ}\text{C}$ bis $125\,^{\circ}\text{C}$	$-50\,^{\circ}\text{C}$ bis $150\,^{\circ}\text{C}$
Genauigkeit	$\pm 0.5\,^{\circ}\text{C}$ bei $-10\,^{\circ}\text{C}$ bis $85\,^{\circ}\text{C}$	$\pm 0.5\,^{\circ}\text{C}$ bei $10\,^{\circ}\text{C}$ bis $90\,^{\circ}\text{C}$
Wandlungsdauer	$\leq 750\,\text{ms}$	$\geq 100\,\text{ms}$
Preis	$\approx 1.70\,\text{EUR}$	$\approx 4.50\,\text{EUR}$

Tabelle 4.5.: Temperatursensoren Maxim Integrated DS18B20 und TSic 206 [11] im Vergleich

4.2.5. Controller

Für die Implementierung des Programmablaufs und den Betrieb der Hardware-Schnittstellen sind im Wesentlichen zwei mögliche Ausführungsvarianten zu unterscheiden: Zum Einen die Umsetzung auf der Basis eines Ein-Platinen-PCs wie z.B. dem Raspberry Pi [25] und zum Anderen mit einem Mikrocontroller.

Bei der Vorerprobung verschieden ausgeführter Messsysteme wurde dazu u.a. eine Messung von Beschleunigungswerten mittels Google Android basiertem Mobiltelefon und einer einfachen Datenlogger-Anwendung durchgeführt. Abgesehen davon, dass die Messwerte aufgrund mangelhafter Befestigungsmöglichkeit im Bus nicht verwertbar sind, zeigten sich an den erfassten Daten Probleme, die auf das typische Verhalten von Betriebssystemen zurückzuführen sind und deshalb auch auf Ein-Platinen-PCs erwartet werden. Die Datenbasis stellt u.a. Zeitstempel in ms zu jedem erfassten Beschleunigungswert zur Verfügung, an denen sich schnell zeigt, dass die Abtastung nicht äquidistant erfolgt. In Abb. 4.2 ist die Dauer Δt in ms zwischen zwei Samples über die Samplefolge für die ersten 3000 Werte der Messung aufgetragen. Sehr deutlich zeigt sich dabei ein hoher Jitter in Höhe von $364\,\text{ms}$ bei

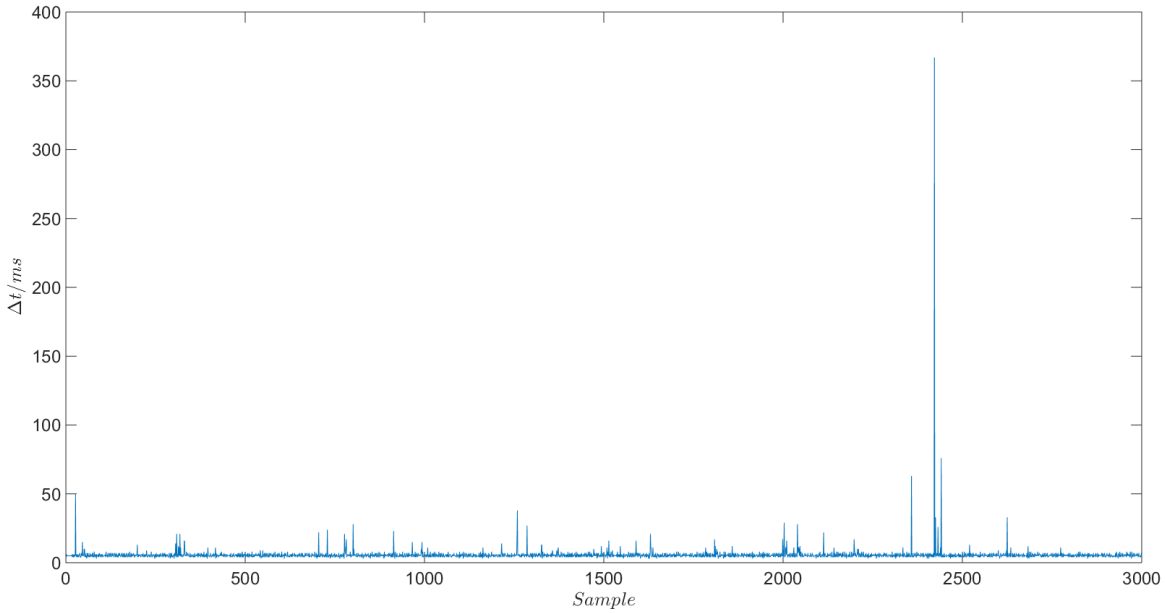


Abbildung 4.2.: Abtastintervalle im Verlauf einer Messung mit nicht-echtzeit Betriebssystem

einer durchschnittlichen Abtastperiode von $5.68\,\mathrm{ms}$. Das beobachtete Verhalten wird auf die Zeitscheibenzuteilung des Betriebssystems an die verwendete Anwendung zurückgeführt, weshalb ähnliches auch bei anderen Betriebssystemen zu erwarten ist, solange zur Abtastung kein stark optimierter Treiber entwickelt wird.

Die mangelnde Äquidistanz bei der Abtastung von Messwerten stellt ein ernsthaftes Problem bei der Weiterverarbeitung der Messwerte dar, da gängige Verfahren der digitalen Signalverarbeitung auf äquidistant abgetastete Samplefolgen ausgelegt sind. Während einfache numerische Integration oder Differentiation noch möglich sind, können vorhandene digitale Filter nicht ohne Weiteres angewendet werden. Es werden teils komplizierte und rechenaufwendige Interpolationsverfahren nötig, um die Daten an ein äquidistant abgetastetes Signal anzunähern.

Aus diesen Erkenntnissen begründet sich der Anspruch, an den Datenlogger, bei der Abtastung Echtzeitbedingungen zu erfüllen.

In Tab. 4.6 findet sich eine rein qualitative Gegenüberstellung der beiden Varianten mit ihren Vor- und Nachteilen. Im Wesentlichen ist dabei festzustellen, dass das Linux-Betriebssytem Bestandteile wie Dateisysteme für Wechselspeicher mitbringt. Im Vergleich zu größeren Mikrocontrollern ist dafür aber die Verfügbarkeit von Hardware-Schnittstellen wie SPI, I^2C oder UART eingeschränkt. Da u.a. die gestellten Echtzeitanforderungen mit einem auf Ein-Platinen-PCs gängigen Linux Betriebssystem nicht ohne großen Programmieraufwand erfüllt

werden können, soll als zentrale Komponente des Datenloggers ein entsprechend auszuwählender Mikrocontroller zum Einsatz kommen.

	Mikrocontroller	Ein-Platinen-PC
Echtzeitfähigkeit	+	-
Rechenleistung	-	+
Wechselspeicheranbindung	-	+
Hardware-Schnittstellen	+	-
Preis	$\approx 20\,\mathrm{EUR}$	$\approx 35\,\mathrm{EUR}$

Tabelle 4.6.: Qualitativer Vergleich Ein-Platinen-PC mit Mikrocontroller als Basis für den Datenlogger (+ pro / - contra)

Mit den bisher erfolgten konzeptionellen Festlegungen und der Forderung möglichst weitreichender Erweiterbarkeit ergibt sich ein großer Bedarf an Schnittstellen am Mikrocontroller. Außerdem soll zur sicheren Einhaltung der Echtzeitanforderungen auch in möglichen weiteren Ausbaustufen des Datenloggers ein leistungsstarker Mikrocontroller zum Einsatz kommen.

Aufgrund der Verbreitung im BATSEN-Projekt und den vorhandenen Erfahrungen kommt als Basis das Evaluationsboard „TM4C1294 Connected LaunchPad“ [34] der Firma Texas Instruments zum Einsatz (Abb. 4.3). Es bietet die Möglichkeit, auf die wichtigsten Schnittstellen des Mikrocontrollers über Stiftleisten zuzugreifen und eigene Erweiterungsplatinen darauf aufzustecken. Alle weiteren Anschlüsse des Mikrocontrollers können über eine nachzubestückende Stiftleiste an der Seite der Platine verwendet werden.
Der verbaute Mikrocontroller ist ein $120\,\mathrm{MHz}$ 32-bit ARM Cortex-M4 vom Typ TM4C1294NCPDT [35], welcher eine ausreichende Anzahl an seriellen Schnittstellen und Rechenleistung bietet.
Texas Instruments stellt für diesen und andere Mikrocontroller die umfangreiche Treiber-Bibliothek Tivaware [36] zur Verfügung. Mit dieser wird der Zugriff auf die Hardware des Mikrocontrollers in gut dokumentierten C-Funktionen abstrahiert.

4.2.6. Wechselspeicher

Mit der Festlegung auf die Verwendung des TM4C1294NCPDT Mikrocontrollers, wird die Auswahl des Wechselspeichers stark eingeschränkt. So bieten sich z.B. S-ATA Festplatten nicht zur Verwendung an, da diese nur mit zusätzlichem Aufwand an den Mikrocontroller angebunden werden können. Aufgrund ihrer Stoßfestigkeit und weil Texas Instruments im Rahmen von Tivaware passende Treiber und eine Implementierung des Dateisystems FatFS zur Verfügung stellt, wird als Datenspeicher eine microSD-Karte benutzt. MicroSD-Karten

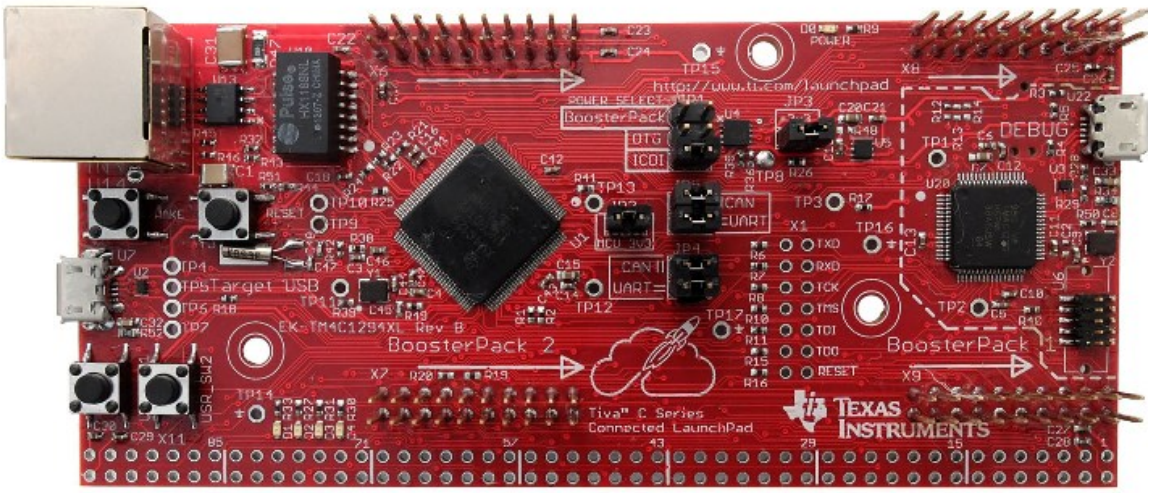


Abbildung 4.3.: Evaluationsboard: Tiva TM4C1294 Connected LaunchPad [38]

lassen sich über die SPI-Schnittstelle an den Mikrocontroller anbinden und sind mit Kapazitäten von einigen GByte preisgünstig erhältlich, da sie in vielen mobilen Geräten verwendet werden.

4.2.7. LCD-Touchscreen

Als einfache Mensch-Maschine-Schnittstelle soll, neben wenigen einfachen Tastern und LEDs, ein LCD-Touchscreen eingesetzt werden. Das Display des Herstellers Kentec vom Typ K350QVG-V2-F ist als Erweiterungsplatine EB-LM4F120-L35 [12] fertig aufgebaut zum Aufstecken auf die Evaluationsboards von Texas Instruments verfügbar. Mit einer Bildschirmdiagonale von 3.5 Zoll kann es gut im Gehäuse des Datenloggers untergebracht werden. Es verfügt über einen resistiven Touchscreen und Farbausgabe.

4.2.8. Akkumulator

Die Versorgung des Datenloggers soll aus einem Akkumulator oder der Boardspannung des Fahrzeugs erfolgen. Wie bereits im Beschleunigungs-Datenlogger von Thom [39] wird dafür ein Akkumulator zum USB-Anschluss verwendet, welcher ursprünglich für den mobilen Betrieb von USB-Geräten vorgesehen ist. Dieser hat, wie in Tab. 4.7 qualitativ dargestellt, Vorteile bei der Handhabung gegenüber einer einfachen Akkumulatorzelle, da die nötige Einhaltung von Spannungs- und Strombedingungen beim Lade- und Entladevorgang durch

integrierte Elektronik sichergestellt wird. Der zusätzliche Schaltungsaufwand verringert sich damit deutlich, es folgen daraus allerdings u.U. Einschränkungen für das Bedienkonzept.

	Akkumulatorzelle	**USB-Akkumulator**
Geregelte Ausgangsspannung	-	+
Flexibles Bedienkonzept	+	-
Effektive Ausnutzung	+	-
Laderegelung	-	+
Preis	$\approx 30\,\mathrm{EUR}$	$\approx 40\,\mathrm{EUR}$

Tabelle 4.7.: Qualitativer Vergleich Akkumulatorzelle mit USB-Akkumulator (+ pro / - contra)

Ausgewählt wird die VTB-28A des Herstellers Vitebo, da das Verhalten aus der Arbeit von Thom [39] bekannt ist. Über USB-Anschlüsse wird ein maximaler Strom von $2.1\,\mathrm{A}$ bei einer Ausgangsspannung von $5\,\mathrm{VDC}$ bis $5.5\,\mathrm{VDC}$ zur Verfügung gestellt. Eine nähere Betrachtung der Spannungsversorgung erfolgt in Abschnitt 5.2.

4.3. Konstruktive Festlegungen

Orientiert an den weiteren in Kapitel 3 formulierten Anforderungen werden konstruktive Festlegungen getroffen:

- Gehäuse

  Als Gehäuse für den Datenlogger wird ein Aluminium-Druckgussgehäuse der Firma Rose mit den Maßen L x B x H 200 x 280 x $72\,\mathrm{mm}$ verwendet.

- Steckverbinder

  Für den Anschluss der Sensoren und Empfänger an den Datenlogger werden vom Typ AMP Superseal der Firma TE Connectivity verwendet.

- Anschluss

  Die Anschlusskabel werden als sog. Pigtails ausgeführt, welche durch Verschraubungen aus dem Gehäuse herausgeführt werden.

4.4. Grobabschätzung der Belegung von Systemressourcen

Um zu verifizieren, dass die Echtzeitanforderungen der Ziel-Spezifikation (Abschnitt 3.3) mit den ausgewählten Komponenten eingehalten werden können, soll eine Grobanalyse der zu erwartenden Belegung von Systemressourcen durchgeführt werden. Dabei soll insbesondere eine Betrachtung der Datenraten und Datenübertragungsraten aus den Anforderungen der Ziel-Spezifikation abgeleitet werden.

4.4.1. Datenraten und Busbelegung

Datenrate GNSS-Empfänger

Eine Abschätzung der Datenrate des GNSS-Empfängers kann anhand der Spezifikation für das Binärprotokoll [42] erfolgen, welche die Länge einer Nachricht, die Uhrzeit, Position und Satelliteninformationen enthält, mit $92\,\mathrm{Byte}$ angibt. Es ist daraus außerdem die maximale Baudrate der UART-Schnittstelle mit $460\,800\,\mathrm{Bd}$ und wegen der binären Übertragung auch die Datenübertragungsrate mit $460\,800\,\frac{\mathrm{bit}}{\mathrm{s}}$ zu entnehmen. Bei einer Samplerate von $10\,\mathrm{Samples/s}$ ergeben sich

$$10\,\mathrm{Samples/s} \cdot 92\,\mathrm{Byte} = 920\,\frac{\mathrm{Byte}}{\mathrm{s}}$$

an Nutzdaten.
Die Auslastung des Kommunikationskanals ergibt sich damit zu etwa

$$\frac{920\,\frac{\mathrm{Byte}}{\mathrm{s}}}{460\,800\,\frac{\mathrm{bit}}{\mathrm{s}}} \approx 1.6\,\%$$

und die Übertragungsdauer für eine Nachricht von

$$92\,\mathrm{Byte} \cdot \frac{1}{460\,800\,\frac{\mathrm{bit}}{\mathrm{s}}} \approx 1.6\,\mathrm{ms}$$

Auch unter der Annahme eines Worst-Case-Overhead durch das Kommunikationsprotokoll von $100\,\%$ ist die Datenübertragungsrate ausreichend, für die Übertragung einer Nachricht stehen $100\,\mathrm{ms}$ zur Verfügung.

Datenrate Inertialsensoren

Das ausgewählte Sensorsystem MPU-9250 bietet gemäß Datenblatt [10] die Möglichkeit, die aktuellen Messwerte aller Beschleunigungs-, Drehraten- und des Temperatursensors in direkter Folge als jeweils $16\,\text{bit}$ Werte im Zweierkomplement auszulesen. Damit fällt außer der $7 \cdot 16\,\text{bit} = 112\,\text{bit}$ nur der je nach verwendeter Schnittstelle (SPI/I^{2}C) vorhandene Protokoll-Overhead der Kommunikation an. Bei einer Abtastrate von $100\,\text{Samples/s}$ fallen demnach

$$100\,\text{Samples/s} \cdot 112\,\text{bit} = 11.2\,\frac{\text{kbit}}{\text{s}}$$

Nutzdaten an.
Bei der langsamer zu verwendenden I^{2}C-Schnittstelle mit maximal $400\,\frac{\text{kbit}}{\text{s}}$ gemäß Datenblatt des MPU-9250 [10], würde dies zur einer Belegung der Kommunikationsschnittstelle von etwa

$$\frac{11.2\,\frac{\text{kbit}}{\text{s}}}{400\,\frac{\text{kbit}}{\text{s}}} = 2.8\,\%$$

führen. Selbst mit $100\,\%$ Protokoll-Overhead würde die Datenrate des Busses für die Übertragung ausreichen.

Im Worst-Case-Fall eines $100\,\%$ Protokoll-Overhead würde die Datenübertragung für ein Sample etwa

$$224\,\text{bit} \cdot \frac{1}{400\,\frac{\text{kbit}}{\text{s}}} = 0.56\,\text{ms}$$

dauern. Sie liegt damit deutlich unter den maximal verfügbaren $10\,\text{ms}$ bei der vorgegebenen Abtastrate.

Datenrate Temperatursensoren

Für den verwendeten 1-Wire-Bus ergibt sich aus der Spezifikation durch das einzuhaltende Bit-Timing [18] eine Datenrate von etwa $15\,\text{kbit}$. Nach Angabe des Datenblattes zum verwendeten Sensor DS18B20 [18] sind zum Auslösen einer Messung der $1\,\text{Byte}$ „READ-ROM-COMMAND“, $8\,\text{Byte}$ zur Adressierung des Sensors und der $1\,\text{Byte}$ „CONVERT-TEMPERATURE“ Befehl zu übermitteln. Die Wandlung auf dem Sensor dauert bei höchster Auflösung etwa $750\,\text{ms}$, worauf der Messwert mit dem $1\,\text{Byte}$ „READ-ROM-COMMAND“, $8\,\text{Byte}$ zur Adressierung des Sensors und dem $1\,\text{Byte}$ Befehl „READ-SCRATCHPAD“ zur Übermittlung von zumindest $2\,\text{Byte}$ aufgefordert wird.
Damit ergibt sich eine Datenrate von

$$0.2\,\text{Samples/s} \cdot 22\,\text{Byte} = 4.4\,\frac{\text{Byte}}{\text{s}}$$

und somit eine Auslastung der verfügbaren Datenübertragungsrate von

$$\frac{4.4\,\frac{\text{Byte}}{\text{s}}}{15\,\frac{\text{kbit}}{\text{s}}} = 0.23\,\%$$

Unter der Berücksichtigung der im Vergleich zur Kommunikation großen Wandlungsdauer, ergibt sich für Dauer einer einzelnen Messung:

$$22\,\text{Byte} \cdot \frac{1}{15\,\frac{\text{kbit}}{\text{s}}} + 750\,\text{ms} = 11.7\,\text{ms} + 750\,\text{ms} = 761.7\,\text{ms}$$

Auch wenn der 1-Wire-Bus von den zunächst vorgesehenen drei Sensoren gemeinsam verwendet wird, ist die geforderte Abtastrate bei einer Busbelegung und Messdauer von insgesamt etwa $2.3\,\text{s}$ einzuhalten.

Datenrate Wechselspeicher

Eine sehr grobe Abschätzung für die Datenrate der auf den Wechselspeicher zu schreibenden Messdaten ergibt sich anhand der geforderten Abtastraten, Auflösungen und der gewählten Komponenten etwa wie folgt

- 3-Achsen-Beschleunigung: $3 \cdot 200\,\frac{\text{Byte}}{\text{s}}$
- 3-Achsen-Drehrate: $3 \cdot 200\,\frac{\text{Byte}}{\text{s}}$
- $3\cdot$ Temperatur: $3 \cdot 0.4\,\frac{\text{Byte}}{\text{s}}$
- $2\cdot$ Geographische Länge: $2 \cdot 40\,\frac{\text{Byte}}{\text{s}}$
- $2\cdot$ Geographische Breite: $2 \cdot 40\,\frac{\text{Byte}}{\text{s}}$
- $2\cdot$ Höhe über NN: $2 \cdot 40\,\frac{\text{Byte}}{\text{s}}$
- $2\cdot$ Geschwindigkeit: $2 \cdot 40\,\frac{\text{Byte}}{\text{s}}$
- $2\cdot$ Anzahl der Satelliten: $2 \cdot 10\,\frac{\text{Byte}}{\text{s}}$
- UTC-Uhrzeit: $70\,\frac{\text{Byte}}{\text{s}}$

Dabei wird jeweils angenommen, dass eine Speicherung der Daten im nativ von den Sensoren zur Verfügung gestellten Format erfolgt. Overhead durch Dateiformate und Dateisystem wird hier zunächst vernachlässigt. Insgesamt fallen damit etwa $1611.2\,\frac{\text{Byte}}{\text{s}}$ an.

Da die SD-Karte als Wechselspeicher über die SPI-Schnittstelle angesprochen wird, ist gemäß Spezifikation [32] nur die Einhaltung der Class-0-Geschwindigkeit anzunehmen. Diese

liegt bei einer SPI-Taktrate von $12.5\,\mathrm{MHz}$. Die ohne Overhead mögliche Datenübertragungsrate von $12.5\,\mathrm{Mbit}$ liegt weit über der anfallenden Nutzdatenrate.

4.4.2. Speicherbedarf

Der Speicherbedarf durch die aufgezeichneten Nutzdaten ergibt sich direkt aus der Datenrate für Nutzdaten auf den Wechselspeicher. Dabei soll die Datenmenge pro Stunde und Tag betrachtet werden.
Pro Stunde fallen demnach etwa

$$1611.2\,\frac{\mathrm{Byte}}{\mathrm{s}} \cdot 3600\,\mathrm{s} = 5.53\,\mathrm{MiByte}$$

Daten an und pro Tag

$$5.53\,\mathrm{MiByte} \cdot 24 = 132.72\,\mathrm{MiByte}$$

Damit wäre eine Aufzeichnungsdauer von mehr als einer Woche auf gängigen SD-Karten möglich.

4.4.3. Zeitverhalten

Das Zeitverhalten kann anhand der erfolgten konzeptionellen Festlegungen zunächst nur über die Übertragungsdauer der Daten abgeschätzt werden, an deren Erfassung Echtzeitanforderungen gestellt werden. Dabei wird als Worst-Case angenommen, dass der gesamte Übertragungsvorgang den Controller belegt, sodass die Abhandlung sequentiell erfolgen muss. In Abb. 4.4 wird zur Verdeutlichung eine $10\,\mathrm{ms}$ Zeitscheibe dargestellt, in welcher die längsten ermittelten Nachrichten auf den Schnittstellen übertragen werden. Die Länge der Zeitscheibe wurde anhand der kürzesten Abtastrate gewählt. An der Abbildung kann abgelesen werden, dass die Übermittlung der Nachrichten in einer Zeitscheibe erfolgen kann. Die Speicherung auf der SD-Karte muss offensichtlich in einer weniger belegten Zeitscheibe erfolgen. Es ist zu erkennen, dass unter Berücksichtigung der zum Aufbau der Kommunikationskanäle benötigen Rechenzeit die Einhaltung der Echtzeitanforderungen kaum gewährleistet werden kann. Deshalb ist es nötig, durch die Ausnutzung der in Hardware vorhandenen FIFOs der Kommunikations-Schnittstellen, eine Parallelisierung der Vorgänge zu erreichen. Weitere Rechenzeit wird für die Ablaufsteuerung und Benutzerinteraktion belegt, welche keine Echtzeitbedingungen erfüllt werden muss.

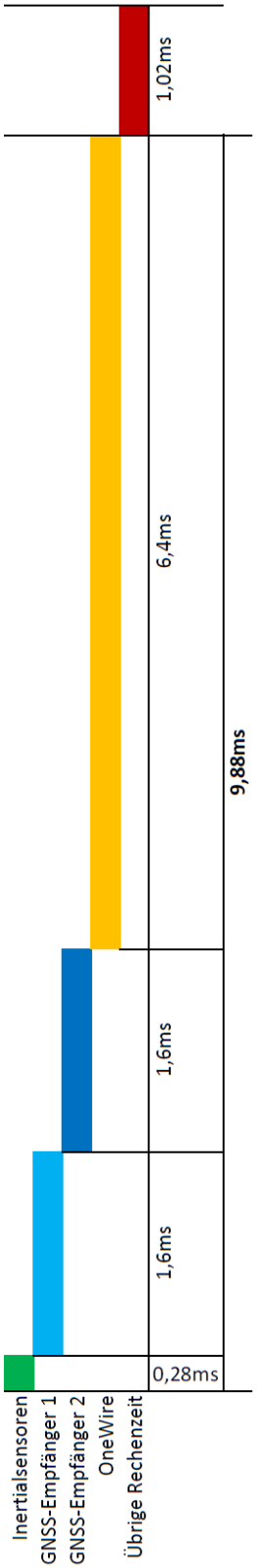


Abbildung 4.4.: Grobabschätzung des Zeitverhaltens anhand der Übertragungsdauer von Nachrichten

5. Hardware-Entwurf und Inbetriebnahme

5.1. Hardware-Module und Schnittstellen

Nach der Identifikation und Auswahl der Systemkomponenten in Kapitel 4, sollen die Systemkomponenten und -funktionen nun auf eigenständigen Platinen zu Hardware-Modulen zusammengefasst werden.
Dabei wird für die Zusammenfassung zu Hardware-Modulen vor allem die Erweiterbarkeit und der prototypische Charakter des Datenloggers berücksichtigt, während sich die zu verwendenden Schnittstellen aus den in Kapitel 4 gemachten Festlegungen ergeben.

Es wird die Gruppierung in die folgenden Hardware-Module gewählt:

- Datenlogger-Erweiterungsplatine
- GSM-/GPRS-Erweiterungsplatine
- GNSS-Empfänger
- Spannungsversorgung / Eingangsschutz

In Abb. 5.1 ist die Zuordnung der Systemfunktionen zu den Hardware-Modulen dargestellt.

Alle Module werden so ausgelegt, dass sie mit einer Eingangsspannung von $5\,\mathrm{V}$ versorgt werden können. Eingangsschutz und nötige Komponenten im Zusammenhang mit der Spannungsversorgung sollen auf einer eigenen Platine aufgebaut werden, sodass Austausch oder Anpassung ohne Eingriff in die restlichen Module möglich ist.

Auf der Datenlogger-Erweiterungsplatine sollen alle Komponenten Platz finden, die für die hauptsächliche Datenaufzeichnungsfunktion des Systems nötig sind. Dank dieses Aufbaus kann dieses Modul gemeinsam mit dem Evaluationsboard zur Erfassung der nötigen Daten eingesetzt werden. Dafür ist lediglich eine Spannungsquelle anzuschließen.

Die GSM-/GPRS-Erweiterungsplatine bietet als weitere Aufsteckplatine die Möglichkeit der Kommunikation über Mobilfunk. Die RTC des Moduls soll zudem durch eine Primärzelle gestützt werden, sodass bei Verwendung der Erweiterungsplatine auch eine Absolutzeit zur

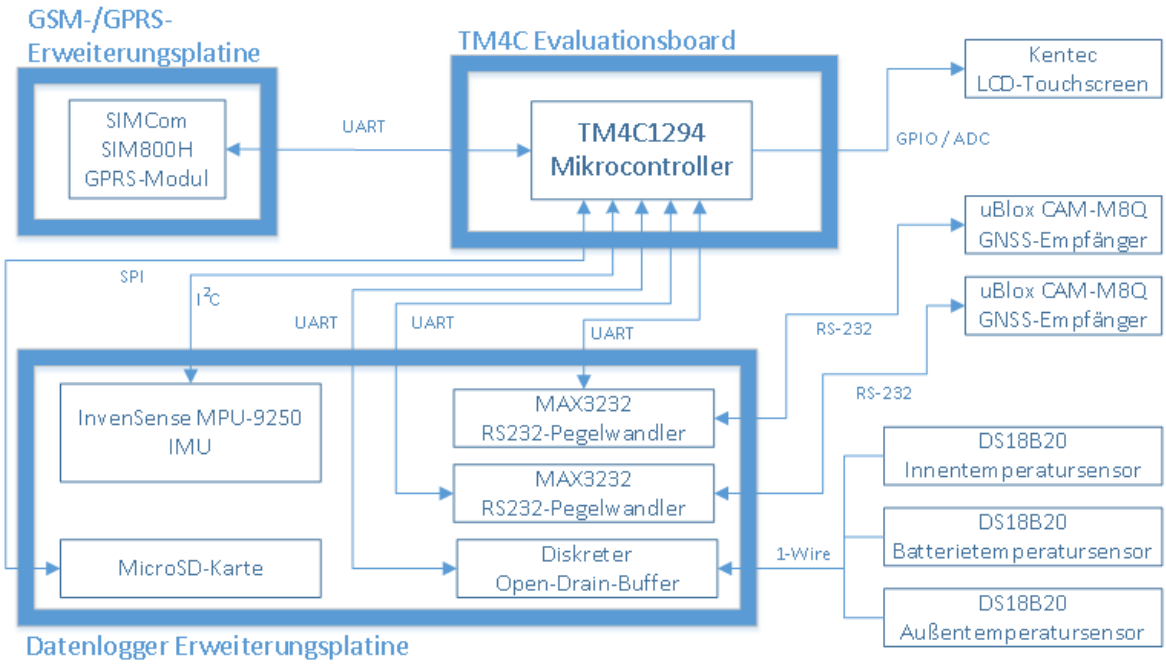


Abbildung 5.1.: Schematische Darstellung der Erweiterungsplatinen, Funktionsmodule und Kommunikationsschnittstellen

Verfügung steht. Für den Betrieb des Datenloggers ist dieses Modul nicht zwangsweise nötig.

Als externe Module werden die GNSS-Empfänger ausgelegt. Diese werden vorrangig zur Erweiterung um die Satellitenortung des Datenloggers benötigt und sollen die Betriebsfähigkeit des Gerätes nicht bedingen. Durch die Anbindung einer Taktleitung, durch den Hersteller als „Timepulse“ bezeichnet, soll eine Synchronisation der Datenaufzeichnung zur UTC-Zeit ermöglicht werden. Das Verfahren dazu wird in Abschnitt 6.3.2 beschrieben.

Damit die Hardware-Module gemeinsam an das Evaluationsboard angeschlossen werden können, ist außerdem die Pin-Belegung an den Stiftleisten und die Nutzung der in Hardware vorhandenen Kommunikationsschnittstellen des Mikrocontrollers abzustimmen. Aus dem Konzept in Abb. 5.1 und weiteren Festlegungen in den folgenden Abschnitten entsteht die Pinbelegung wie in Anhang B.1 aufgeführt. Dabei findet insbesondere Berücksichtigung, dass die geplanten Hardware-Module sich weitestmöglich auf einen der zwei standardisierten Erweiterungssteckplätze beschränkt. So steht der zweite Anschluss für die Ansteuerung des Displays und eventuelle Erweiterungen zur Verfügung.

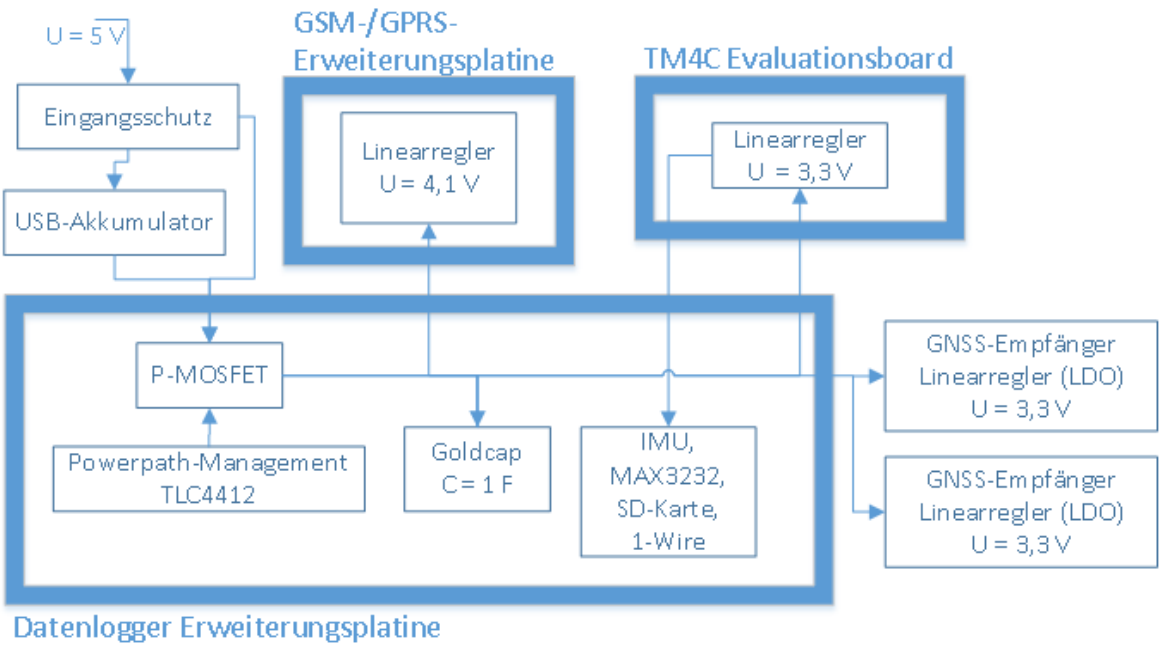


Abbildung 5.2.: Schematische Darstellung der Spannungsversorgung der Module

5.2. Spannungsversorgung und Eingangsschutz

Der Entwurf der Spannungsversorgung wird durch die folgenden Vorgaben geformt:

- Versorgung aus dem USB-Akkumulator und externer 5 V Spannungsversorgung
- Schutz gegen Überspannung
- Schutz gegen Überstrom

Bedingt durch diese Vorgaben und die Anforderungen der einzelnen Hardware-Module ergibt sich die schematische Darstellung der Spannungsversorgung in Abb. 5.2, welche in den folgenden Abschnitten erläutert und hergeleitet werden soll.

Insbesondere aus der Verwendung des USB-Akkumulators ergeben sich, aufgrund dessen Bedienkonzepts, Auswirkungen auf den Entwurf. Der USB-Akkumulator kann über seinen Eingang mit 5 V geladen werden, stellt aber währenddessen an seinem Ausgang keine Spannung zur Verfügung. Nach Betätigung des Einschalters liegt am Ausgang des USB-Akkumulators eine Spannung von 5 V an, welche nicht aktiv abgeschaltet werden kann. Der USB-Akkumulator schaltet die Spannung am Ausgang erst dann automatisch wieder ab, wenn über einen Zeitraum von etwa 10 s nahezu kein Strom mehr geflossen ist, oder eine Spannung an seinen Eingang angelegt wird. Es ist zu beachten, dass bei Abschaltung

am Ausgang masseseitig durch einen N-Kanal-MOSFET getrennt wird und keine geregelte Spannung mehr anliegt.
Direkte Folge dieser Vorgaben sind, dass nach dem Abschalten einer externen Spannungsquelle der Datenlogger ohne Strom ist, bis der USB-Akkumulator durch den Anwender eingeschaltet, oder wieder eine externe Spannung angelegt wird. Außerdem kann der Datenlogger durch seine Software nur eine Abschaltung veranlassen, indem er die Stromaufnahme stark senkt. Es bietet sich daher an, dass der Datenlogger in die Lage versetzt wird, sich von der Spannungsversorgung zu trennen. Das gewählte Verfahren wird in Abschnitt 5.3 beschrieben.

Um die Bedienung möglichst einfach zu gestalten, soll das Debug-Interface des Mikrocontrollers, die externe Spannungsversorgung sowie die Ladespannung über ein einzelnes aus dem Gehäuse herausgeführtes USB-Kabel angeschlossen werden. Dabei ist es nötig, folgende Bedingungen einzuhalten:

- Der Lade-Eingang des USB-Akkumulators darf nicht mit der Masse des Datenloggers verbunden werden, da zu diesem sonst am Ausgang des USB-Akkumulators ständig eine nicht definierte Spannung anliegt.
- Es darf kein Strom aus dem Datenlogger oder Debugger in den Lade-Eingang des USB-Akkumulators fließen.
- Aufgrund des $4.1\,\mathrm{V}$-Linearreglers auf dem GSM-/GPRS-Modul ist der Spannungsabfall von einigen $100\,\mathrm{mV}$ durch eine Diode im Strompfad nicht tolerierbar.

Ein vereinfachter Schaltplan der Spannungsversorgung ist in Abb. 5.3 dargestellt. Daran ist zu erkennen, dass anstelle einer Diode ein P-Kanal-MOSFET verhindert, dass ein Strom aus dem Datenlogger in den Lade-Eingang des USB-Akkumulators fließen kann. Große Relevanz hat dabei das IC LTC4412 [14] des Herstellers Linear Technology, welches bei Anliegen einer externen Spannung das Gate des MOSFET so steuert, dass sich über die Drain-Source-Strecke ein Spannungsabfall von nur $20\,\mathrm{mV}$ einstellt.

In Abb. 5.3 ist zudem zu erkennen, dass als Schutz vor Spannungsspitzen und Überströmen eine Suppressordiode sowie eine Schmelzsicherung, ausgelegt auf $2\,\mathrm{A}$, zum Einsatz kommen.

5.2.1. Betrachtung des maximalen Stroms

Um zu ermitteln, ob die Auslegung der Spannungsversorgung ausreichend ist, soll der maximale Strom in den Datenlogger grob abgeschätzt werden. Dazu sind in Tabelle 5.1 die Datenblattwerte für die Ströme in die einzelnen Systemkomponenten aufgeführt. Da lediglich Linearregler eingesetzt werden, entspricht deren Summe im Wesentlichen dem Strom

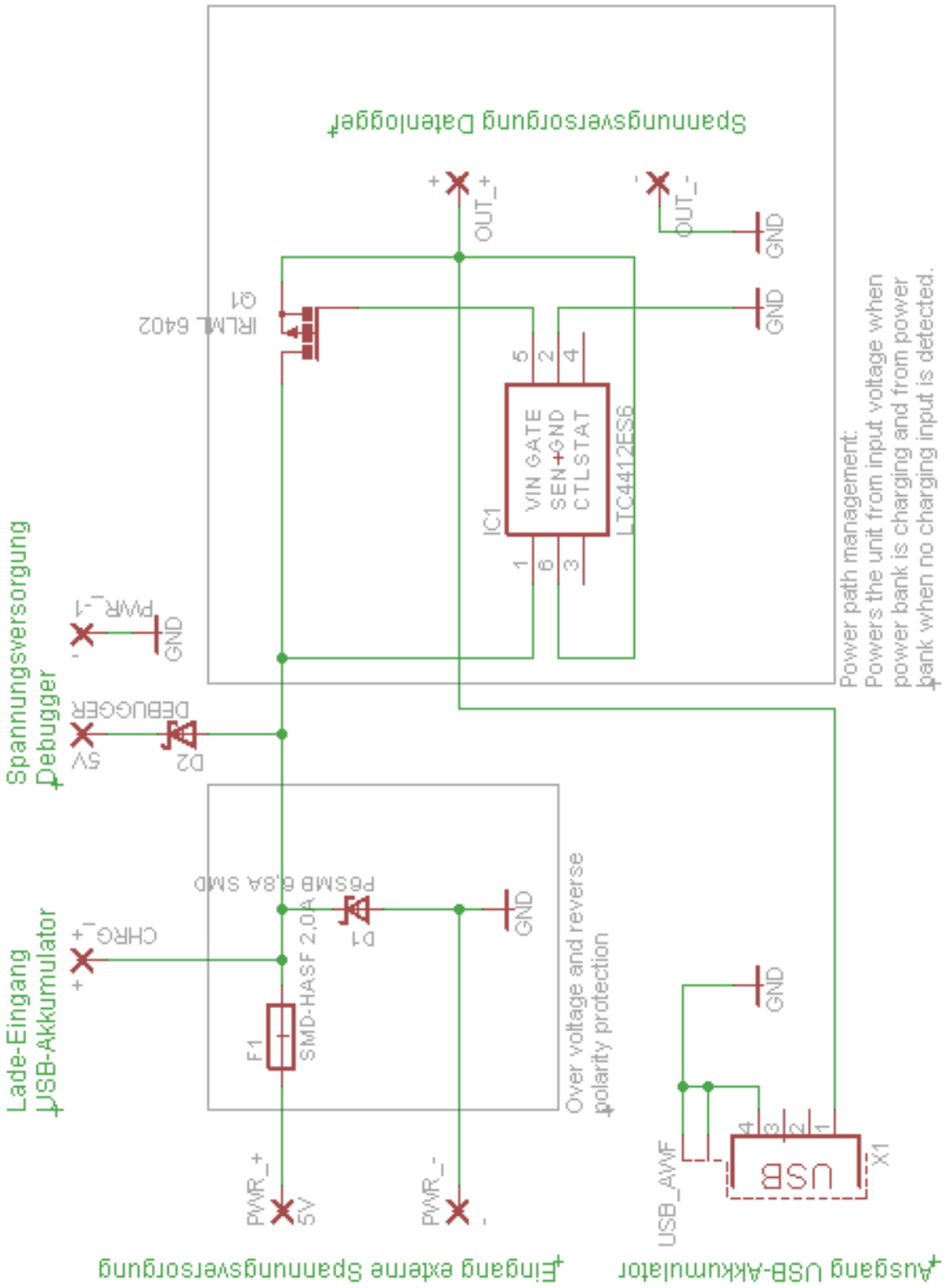


Abbildung 5.3.: Vereinfachter Schaltplan der Spannungsversorgungs-Platine

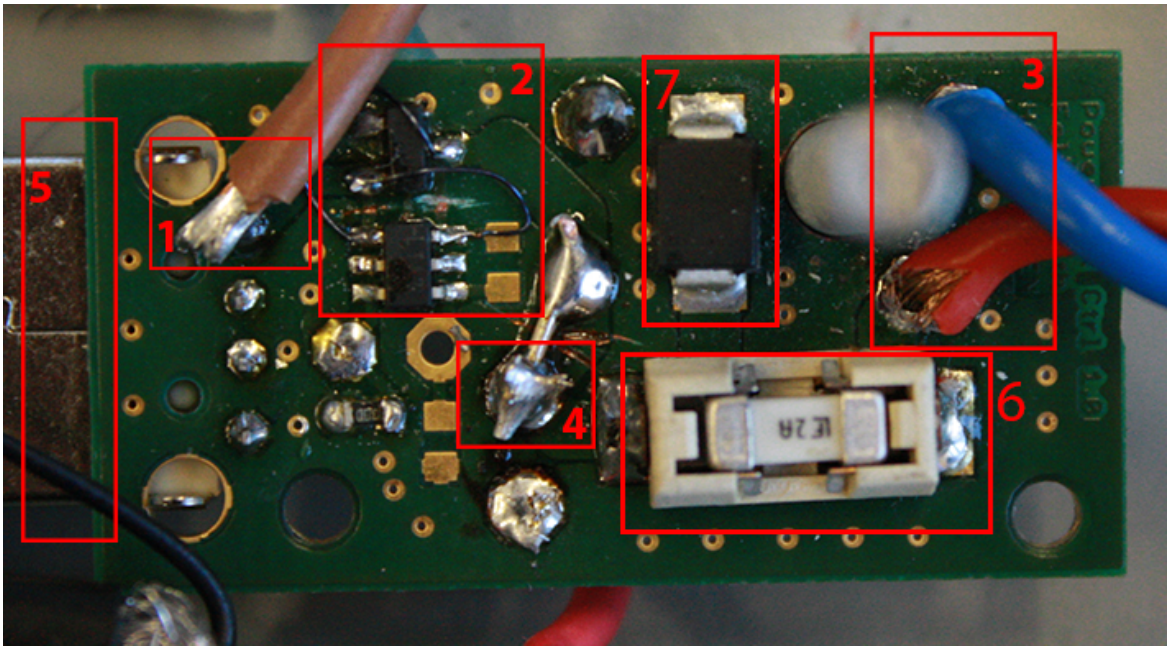


Abbildung 5.4.: Platine zur Spannungsversorgung, 1: Spannung zur Versorgung des Dataloggers, 2: Korrekturen des PCB-Layouts, 3: Anschluss externe Spannungsversorgung, 4: Lade-Eingang des USB-Akkumulators, 5: USB-Anschluss zum Ausgang des USB-Akkumulators, 6: 2 A Schmelzsicherung, 7: Suppressordiode

aus der Spannungsquelle, unabhängig von der Spannung an der Komponente. Diese ergibt sich zu $994.2\,\mathrm{mA}$.
Selbst unter der Worst-Case-Annahme, dass die Komponenten durchgängig und gleichzeitig die maximale Leistung aufnehmen, ist die Ausgangsleistung des USB-Akkumulators ausreichend. In diesem Worst-Case würde zudem bei einer Kapazität von $12\,\mathrm{A\,h}$ und aufgerundetem Maximalstrom etwa eine Laufzeit von

$$\frac{12\,\mathrm{A\,h}}{1\,\mathrm{A}} = 12\,\mathrm{h}$$

erreicht.
Es ist anzunehmen, dass die Leistungsaufnahme der Systemkomponenten im normalen Betrieb weit unter den Maximalwerten bleibt.

Komponente	Maximaler Strom
GNSS-Empfänger (2 Stk.)	142 mA
GSM-/GPRS-Modul	454 mA
Inertialsensoren	2.7 mA
MicroSD-Karte	45 mA
RS232-Pegelwandler (4 Stk.)	240 mA
Mikrocontroller	106 mA
Temperatursensoren (3 Stk.)	4.5 mA
Summe	994.2 mA

Tabelle 5.1.: Grobabschätzung zum maximalen Stromverbrauch des Datenloggers

5.2.2. Inbetriebnahme

Die Inbetriebnahme der Modul-Platine soll nach dem Bestücken gemäß der Tab. 5.2 durchgeführt werden, sodass Fehlfunktionen frühzeitig erkannt werden können.

5.3. Datenlogger-Erweiterungsplatine

5.3.1. Entwurf

Der entstandene Schaltplan zur Datenlogger-Erweiterungsplatine findet sich in Anhang B.3. Die bestückte Platine ist in Abb. 5.5 mit Markierungen zur Herausstellung der wichtigen Funk-

	OK?
Anlegen von $5\,\mathrm{V}$ an PWR	
$5\,\mathrm{V}$ an CHRG?	
$5\,\mathrm{V}$ an OUT?	
$\approx 3.3\,\mathrm{V}$ an STAT?	
CTL an $5\,\mathrm{V}$	
$0\,\mathrm{V}$ an OUT?	
Anlegen von $5\,\mathrm{V}$ an USB	
$5\,\mathrm{V}$ an OUT?	
$0\,\mathrm{V}$ an CHRG?	
$\approx 0.3\,\mathrm{V}$ an STAT?	

Tabelle 5.2.: Inbetriebnahme-Plan für Spannungsversorgungs-Modul

tionsgruppen versehen. Im Weiteren werden wesentliche Entwurfsentscheidungen beleuchtet.

Eine wichtige im Entwurf der Datenlogger-Erweiterungsplatine zu berücksichtigende Rahmenbedingung ist das Betriebsverhalten des USB-Akkumulators. Um eine Abschaltung durch Software realisieren zu können, muss die Schaltung sich selbst von der Spannungsversorgung trennen können. Dafür wird ein P-Kanal-MOSFET als High-Side-Schalter eingesetzt und über den bereits auf der Spannungsversorgungs-Platine eingesetzten LTC4412 [14] angesteuert. Dabei wird der Kontroll-Eingang des MOSFET-Treibers über einen Öffner-Kontakt des Einschalters und einen Pull-Up-Widerstand auf einen High-Pegel gelegt, sodass der MOSFET sperrt. Beim Tasten wird der MOSFET aufgesteuert, die Software muss dann den Kontroll-Eingang aktiv auf Low-Pegel ziehen, um damit eine Selbsthaltung umzusetzen. Durch einen High-Pegel kann die Software dann die gesamte Platine von der Spannungsversorgung trennen. Die Verbindung des Kontroll-Eingangs zum Mikrocontroller erfolgt über an dem Pin angelötete Litze und wird auf die seitliche Stiftleiste des Evaluationsboards aufgesteckt, da alle Pins an der verwendeten Erweiterungs-Stiftleiste belegt sind.

Da während des Betriebs des Datenloggers auf einen Wechselspeicher mit Dateisystem geschrieben wird, muss für den Fall eines unerwarteten Wegfallens der Versorgungsspannung gewährleistet werden, dass schreibende Dateisystemoperationen erfolgreich beendet werden können. Dazu wird an die Spannung von $5\,\mathrm{V}$ ein Goldcap-Kondensator mit hoher Kapazität von $1\,\mathrm{F}$ angeschlossen. Aus diesem kann der Datenlogger betrieben werden, während unnötige Komponenten abgeschaltet und die Dateisystemoperationen beendet werden.

Auf der Datenlogger-Erweiterungsplatine wird zudem ein Sockel für MicroSD-Karten vorgesehen und gemäß Spezifikation angeschlossen. Die Karte arretiert mittels eines Federme-

chanismus in dem gewählten Sockel [21] und wird erst durch nochmaliges Drücken ausgeworfen. So ist ein stabiler Sitz der Karte auch im Fahrzeugbetrieb gewährt.

Zur Umsetzung der 1-Wire-Schnittstelle wird ein diskreter Open-Drain-Buffer aus zwei N-Kanal-MOSFETs aufgebaut und an eine UART-Schnittstelle des Mikrocontrollers angeschlossen. Dabei wird im Wesentlichen der Appnote 214 [16] der Firma Maxim Integrated gefolgt. Außerdem wird ein P-Kanal-MOSFET vorgesehen, mit dem der Mikrocontroller die Datenleitung des 1-Wire-Busses auf einen High-Pegel ziehen kann. Auf diese Weise wird der sog. parasitäre Betrieb der angeschlossenen Sensoren ermöglicht, bei dem diese sich über die Datenleitung mit Spannung versorgen. Die Ansteuerung erfolgt über eine Litze, die auf die seitliche Stiftleiste des Evaluationsboards führt.

Auch das Inertialmessystem wird auf der Datenlogger-Erweiterungsplatine untergebracht. Der Anschluss erfolgt mit wenigen passiven Bauteilen gemäß Datenblatt [10]. Über die wahlweise Bestückung von R2 oder R3 kann das LSB der I^2C-Adresse eingestellt werden, sodass es theoretisch möglich ist, ein weiteres Inertialmessystem am gleichen I^2C-Bus zu betreiben. Außerdem werden Jumper JP1 und JP2 vorgesehen, die es zulassen, das Inertialmessystem alternativ an der SPI-Schnittstelle zu betreiben, sollten die höheren Datenübertragungsraten in Zukunft nötig werden.

Mittels eines 10-poligen Steckverbinders können die GNSS-Empfänger an die Datenlogger-Erweiterungsplatine angeschlossen werden. Dafür wurden jeweils ein RS232-Pegelwandler gemäß Datenblatt beschaltet und an die UART-Schnittstellen des Mikrocontrollers angeschlossen. Die Wahl der höheren RS232-Pegel ermöglicht längere Kabelwege zu den GNSS-Empfängern als ein $3.3\,\mathrm{V}$-Logik-Pegel. Über die Pegelwandler wird auch das Timepulse-Signal übertragen und gemäß der gewählten Pin-Belegung (Anhang B.1) verbunden.

Um eine einfache Mensch-Maschine-Schnittstelle auch ohne den LCD-Touchscreen umsetzen zu können, werden zudem Treiber für drei Leuchtdioden sowie Anschlüsse für drei Taster vorgesehen. Mittels einfacher RC-Glieder werden die Taster in Hardware entprellt, um einen zuverlässigen Betrieb zu gewährleisten.

5.3.2. Inbetriebnahme

Die Inbetriebnahme nach der Bestückung erfolgt gemäß des Inbetriebnahme-Plans in Tab. 5.3 und wird durch einen Funktionstest mittels des Software-Modultests (Abschnitt 6.5) abgeschlossen.

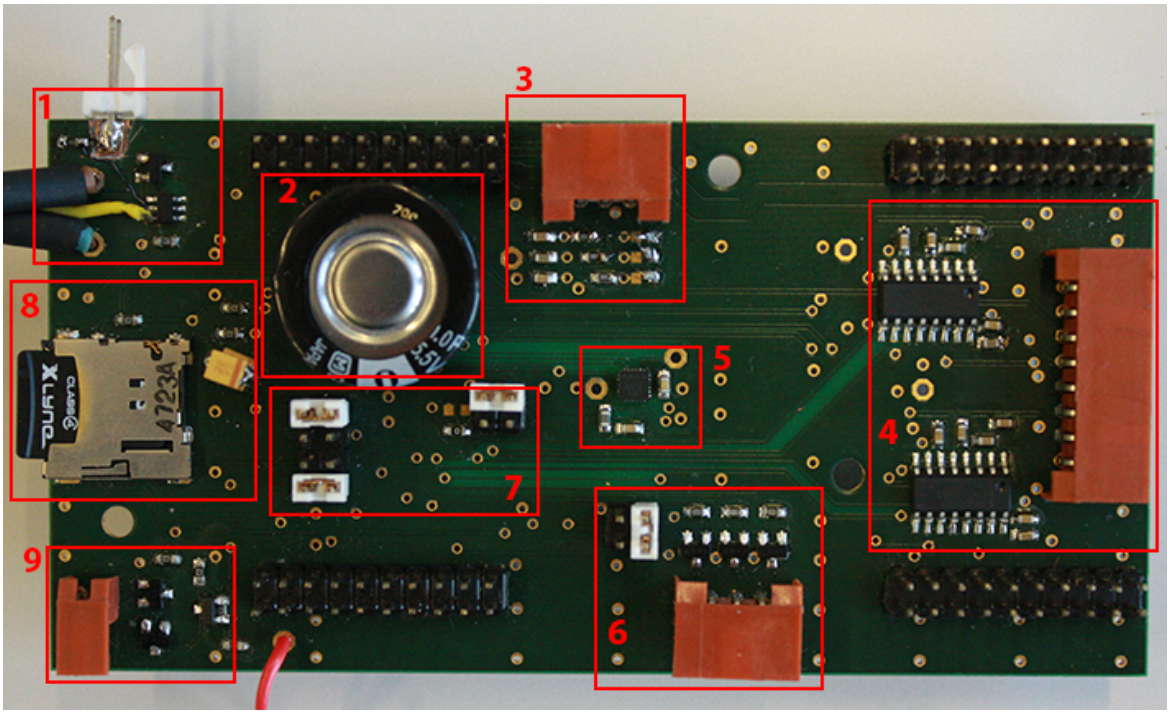


Abbildung 5.5.: Datenlogger-Erweiterungsplatine, 1: P-Kanal-MOSFET mit Ansteuerung und Stiftleiste für Ein-Taster, 2: Goldcap zur Pufferung der Versorgungsspannung, 3: Steckverbinder für Taster mit RC-Gliedern, 4: Steckverbinder für GNSS-Empfänger und RS232-Pegelwandler, 5: Inertialmesssystem, 6: Steckverbinder für LEDs mit Transistor-Treibern, 7: Jumper zum Wechsel zwischen I^2C und SPI am Inertialmesssystem, 8: MicroSD-Karte mit Tantal-Kondensatoren, 9: Steckverbinduer und Open-Drain-Buffer für 1-Wire

	OK?
Anlegen von $5\,\mathrm{V}$ an V+	
$5\,\mathrm{V}$ an +5V?	
Kontakte PWR-SWITCH kurzschließen	
$0\,\mathrm{V}$ an +5V?	
Aufstecken auf Evaluationsboard	
Anlegen von $5\,\mathrm{V}$ an V+	
$3.3\,\mathrm{V}$ an +3V3?	
GPS1_TXD auf Low-Pegel	
X1RED $\approx -5\,\mathrm{V}$?	
X1ORG an $5\,\mathrm{V}$	
GPS1_RXD $\approx 3.3\,\mathrm{V}$?	
X1YLW an $5\,\mathrm{V}$	
GPS1_TIME $\approx 3.3\,\mathrm{V}$?	
GPS2_TXD auf Low-Pegel	
X1GRN $\approx -5\,\mathrm{V}$?	
X1BLU an $5\,\mathrm{V}$	
GPS2_RXD $\approx 3.3\,\mathrm{V}$?	
X1PPL an $5\,\mathrm{V}$	
GPS2_TIME $\approx 3.3\,\mathrm{V}$?	

Tabelle 5.3.: Inbetriebnahme-Plan für Datenlogger-Erweiterungsplatine

Fehler in der 1-Wire-Übertragung

Bei den Software-Tests für die 1-Wire-Schnittstelle zeigt sich, dass mit einem ursprünglichen Entwurf des diskreten Open-Drain-Buffers in Abb. 5.6, nach der Appnote [16] der Firma Maxim Integrated, der 1-Wire-Bus nicht funktional ist. Ein angeschlossener Temperatursensor kann mittels des Modultests nicht ausgelesen werden.

Eine Messung mit dem Oszilloskop am Datenausgang der UART-Schnittstelle (OW_TXD) und der Datenleitung des 1-Wire (K12) ergibt die Darstellung in Abb. 5.7 während der Übertragung eines Bits. Es ist daran zu erkennen, dass der Pegel auf der Datenleitung des 1-Wire dem Low-Pegel der UART nur stark verzögert und mit einer breiten Flanke folgt. Das Bit-Timing des 1-Wire nach [18] ist damit nicht mehr eingehalten und die Kommunikation schlägt fehl.
Der Fehler kann behoben werden, indem der Widerstand $R16$ kleiner gewählt wird. Mit $R16 = 27\,\mathrm{k\Omega}$ konnte die Flankensteilheit wie in Abb. 5.8 deutlich verbessert werden, sodass eine fehlerfreie Kommunikation mit den Temperatursensoren möglich ist.

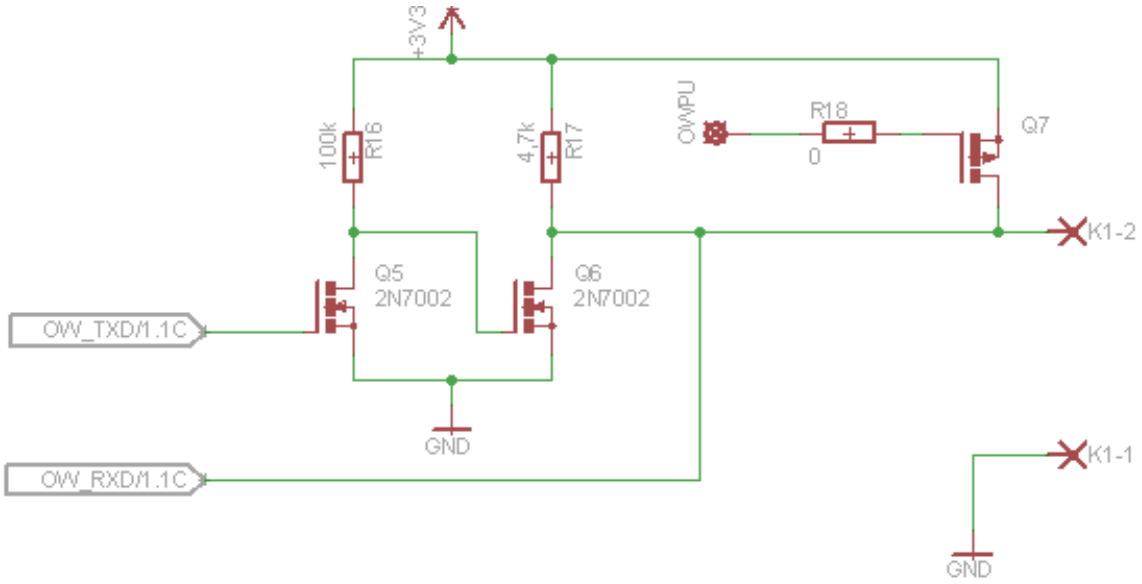


Abbildung 5.6.: Diskreter Open-Drain-Buffer für 1-Wire-Bus

Vermutlich ist die Beispielschaltung aus der Appnote undokumentiert auf eine Versorgungsspannung von 5 V ausgelegt.

Reset durch Einschaltstrom der MicroSD-Karte

Weiter konnte bei den Software-Tests für den Betrieb der MicroSD-Karte festgestellt werden, dass beim Einstecken der Karte in das laufende System ein Reset des Mikrocontrollers stattfindet. Dieser begründet sich in einem Brown-Out-Reset durch ein Einbrechen der 3.3 V Spannungsversorgung, ausgelöst durch den hohen Einschaltstrom der MicroSD-Karte. Eine Messung der 3.3 V mittels Oszilloskop ergibt den Spannungseinbruch in Abb. 5.9 von fast 1 V.
Erst durch den Einsatz von Tantalkondensatoren mit einer Gesamtkapazität von 141 µF direkt vor dem Pin zur Spannungsversorgung der MicroSD-Karte konnte der Spannungseinbruch soweit reduziert werden, dass zuverlässig kein Brown-Out-Reset beim Einstecken auftritt. Der Einschaltvorgang führt damit, wie in Abb. 5.10, nur noch zu einem Spannungseinbruch von etwa 280 mV.
Um im Weiteren einen sicheren Betrieb des Datenloggers zu gewährleisten, wird ein 220 µF Tantalkondensator vor den Sockel der MicroSD-Karte platziert.

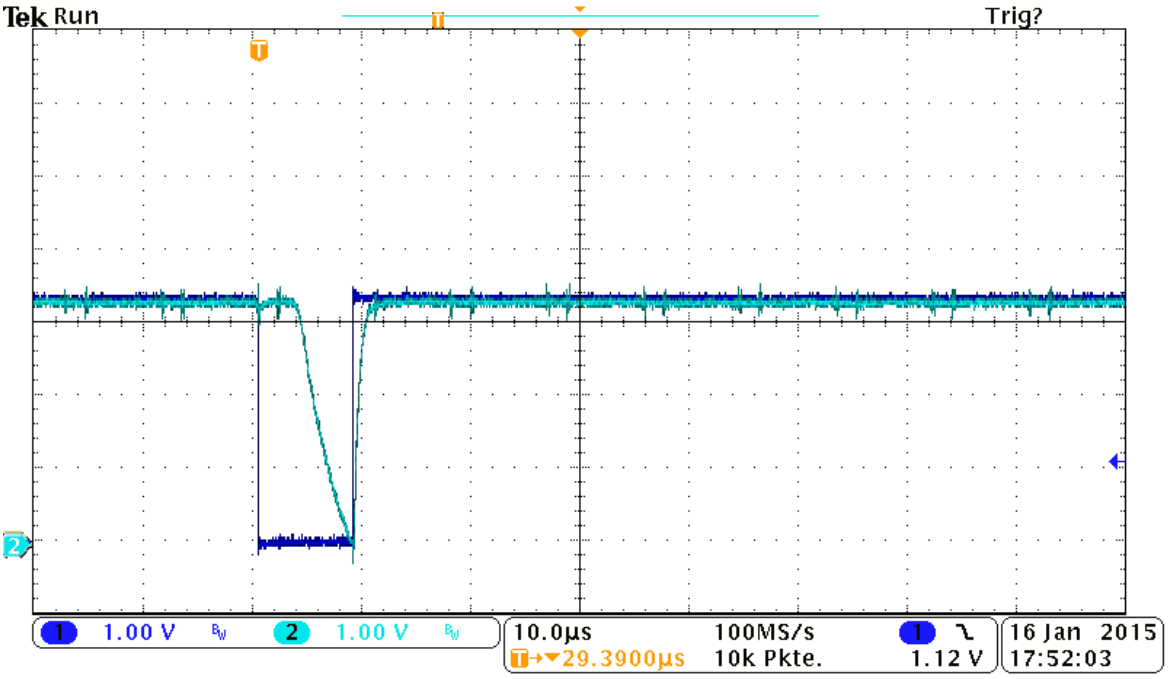


Abbildung 5.7.: Fehlerhafte Übertragung auf dem 1-Wire-Bus

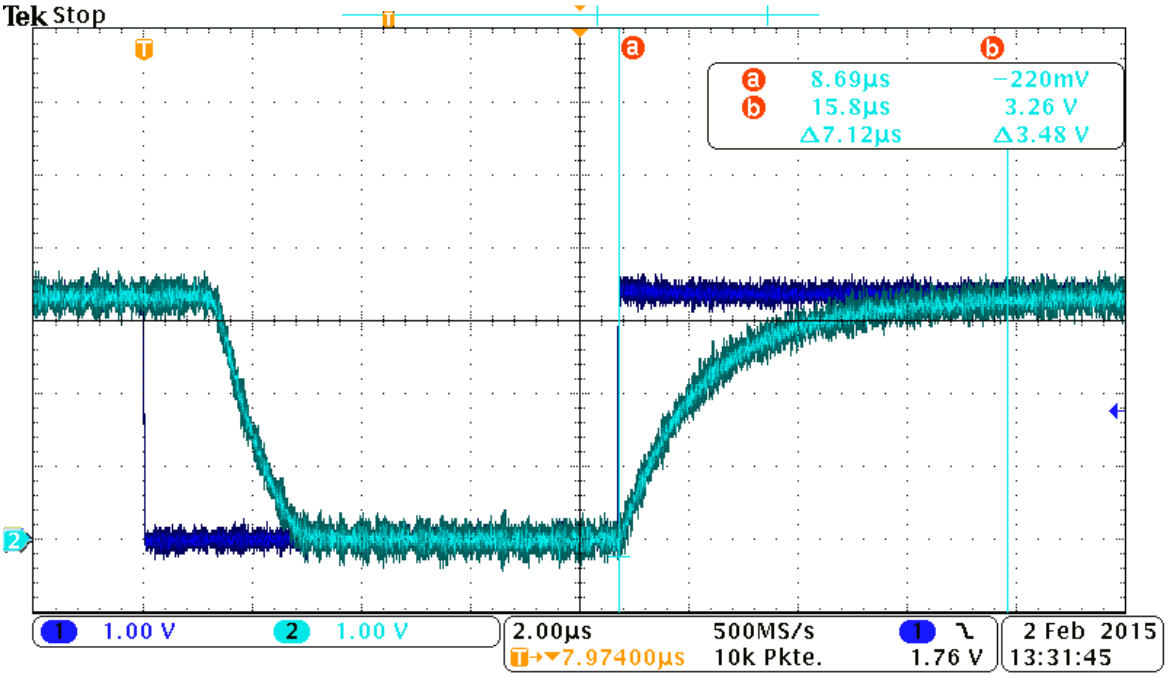


Abbildung 5.8.: Fehlerfreie Übertragung auf dem 1-Wire-Bus

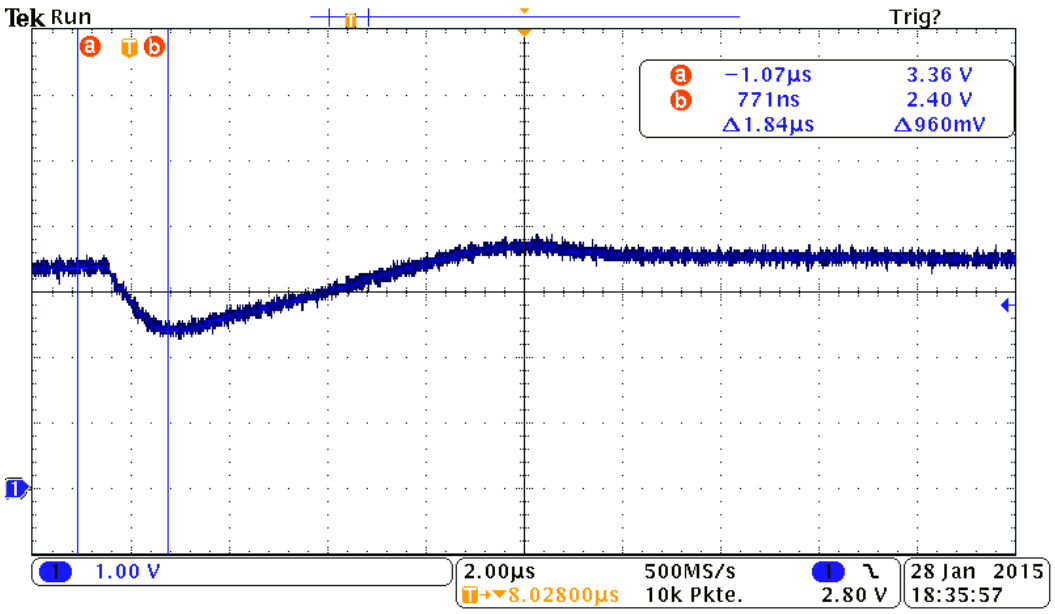


Abbildung 5.9.: Einbruch der Versorgungsspannung durch den Einschaltstrom der MicroSD-Karte

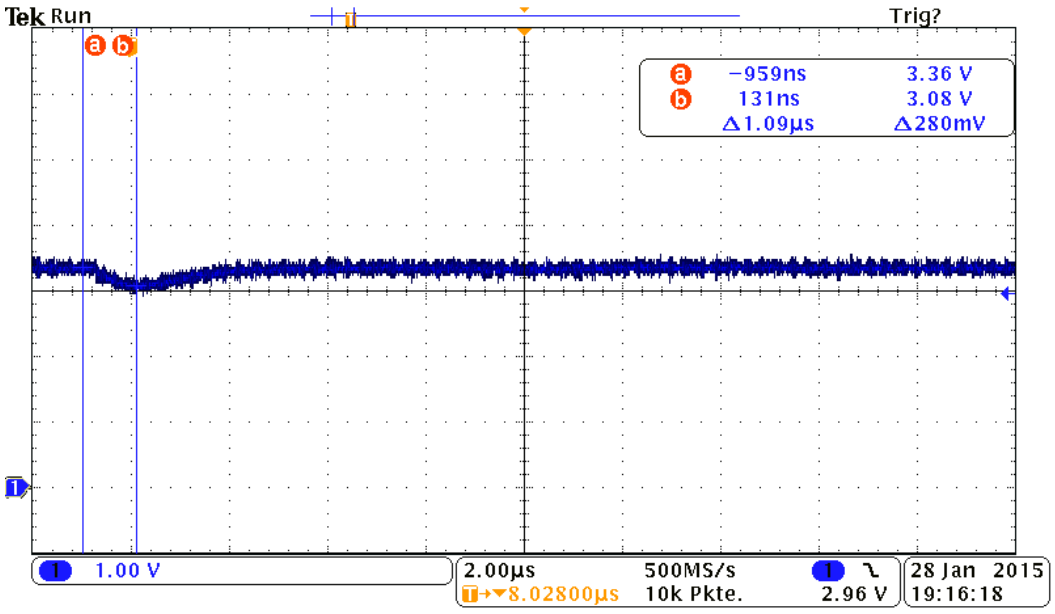


Abbildung 5.10.: Einbruch der Versorgungsspannung durch den Einschaltstrom der MicroSD-Karte mit 141 µF

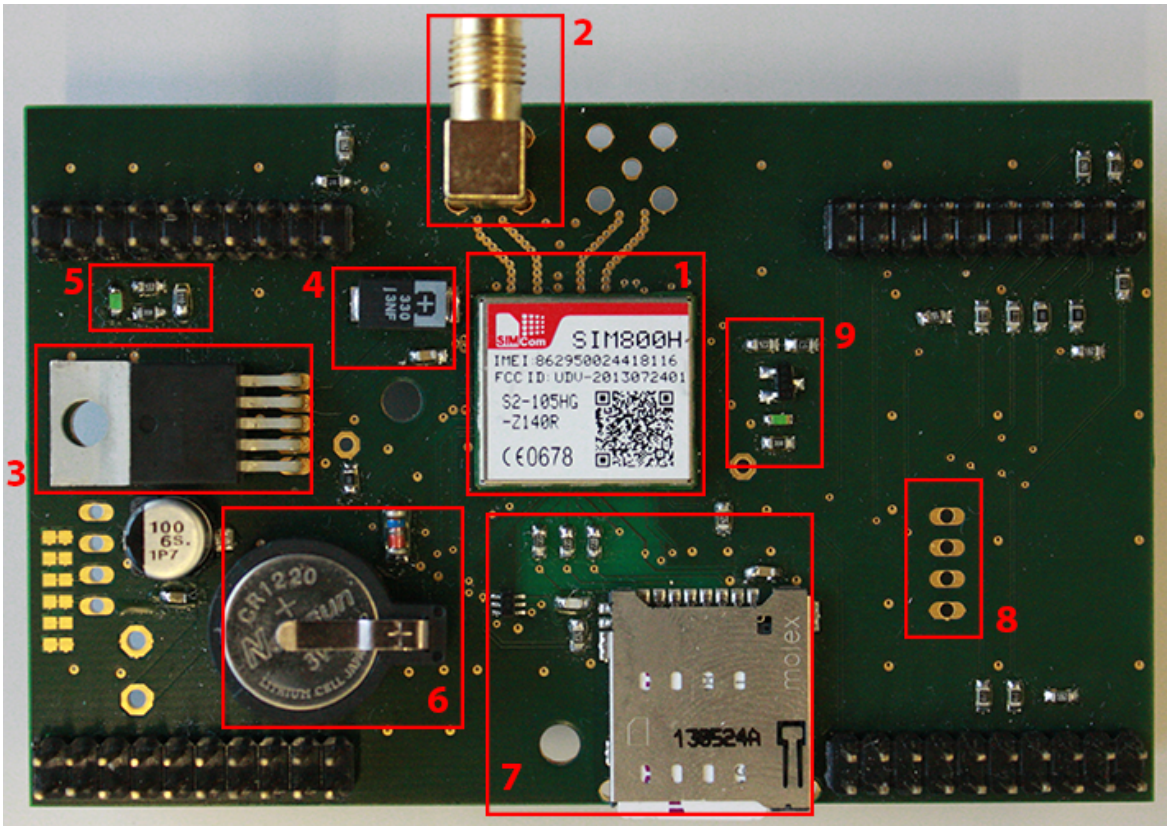


Abbildung 5.11.: GSM-/GPRS-Erweiterungsplatine, 1: GSM-/GPRS-Modul, 2: SMA-Buchse zum Anschluss einer GSM-Antenne, 3: Linearregler zur Versorgung des GSM-/GPRS-Modul, 4: 330 µF Tantal-Kondensator, 5: Power LED, 6: Puffer-Batterie für RTC, 7: Sockel für SIM-Karte mit Schutzbeschaltung, 8: Lötanschluss für USB-Verbindung, 9: LED und Treiber als Indikator für den Netzstatus

5.4. GSM-/GPRS-Erweiterungsplatine

5.4.1. Entwurf

Der Schaltplan zur GSM-/GPRS-Erweiterungsplatine findet sich in Anhang B.4. Die bestückte Platine ist in Abb. 5.11 mit Markierungen zur Herausstellung der wichtigen Funktionsgruppen versehen. Im Weiteren werden auch für dieses Hardware-Modul wesentliche Entwurfsentscheidungen beleuchtet.

Schaltplan und Platinenlayout richten sich im Wesentlichen nach den Empfehlung des Herstellers in [33].

Insbesondere ist die Schaltung zur Spannungsversorgung mittels eines Linearreglers [19] und eines Tantalkondensators mit einer Kapazität von $330\,\mu F$ nahe am Eingangspin des GSM-/GPRS-Moduls übernommen worden. Durch die große Kapazität sollen Stromspitzen aufgefangen werden, die durch Bursts in der GPRS-Übertragung hervorgerufen werden. Das Datenblatt gibt diese Stromspitzen mit bis zu $2\,A$, für eine Dauer von $577\,\mu s$ an und fordert einen Spannungseinbruch von maximal $350\,mV$ währenddessen.
Um zu überprüfen, ob diese Bedingung voraussichtlich eingehalten werden kann, wird aus dem Platinenlayout der Datenlogger-Erweiterungsplatine und der GSM-/GPRS-Erweiterungsplatine das in Abb. 5.12 dargestellte PSpice-Modell abgeleitet. Es finden dabei Übergangs- und Leiterbahnwiderstände sowie die Kapazität des Goldcaps auf der Datenlogger-Erweiterungsplatine und der Tantal-Kondensator am Eingang des GSM-/GPRS-Modul Berücksichtigung. Vernachlässigt wird die Sprungantwort des Linearreglers und deshalb, unter der Annahme, dass die Ausgangsspannung des Linearreglers im Worstcase der Eingangsspannung folgt, nur die $5\,V$ Versorgungsspannung betrachtet. Aus der Simulation ergibt sich Abb. 5.13, an der zu erkennen ist, dass die Spannung um weniger als $210\,mV$ einbricht. Dieses Verhalten liegt im Rahmen der geforderten Spezifikation.

Der Anschluss des Moduls an den Mikrocontroller erfolgt über eine UART-Schnittstelle mit den nötigen Anschlüssen für eine Hardware-Flow-Control. Damit soll bei eventuell anfallenden größeren Datenmengen der Mikrocontroller entlastet werden. Außerdem wird der Kontroll-Eingang des Linearreglers an einen GPIO-Pin angeschlossen, sodass aus der Software heraus die Versorgungsspannung des GSM-/GPRS-Moduls komplett abgeschaltet werden kann. Die Auswahl von Widerständen zur Pegelanpassung erfolgt gemäß Herstellerangabe in [33].
Es werden außerdem die USB-Schnittstelle des Moduls sowie der analoge Audio-Ein- und Ausgang auf Stiftleisten verfügbar gemacht, um einen flexiblen Einsatz der Erweiterungsplatine zu ermöglichen.

Sowohl GSM- als auch Bluetooth-Antennenanschlüsse sind im Platinenlayout vorgesehen und gemäß Vorschlägen des Herstellers SIMCom als „coplanar Waveguide“ ausgeführt um möglichst eine Impedanz von $50\,\Omega$ zu erreichen.

Die RTC des GSM-/GPRS-Moduls wird durch eine Primärzelle gestützt, sodass auch nach längerer Trennung von externer Versorgungsspannung eine Absolutzeit als Referenz zur Verfügung steht. Dafür wird eine CR1025-Knopfzelle mit $32\,mA\,h$ in einem Knopfzellenhalter verbaut. Mit der Datenblattangabe von maximal $5\,\mu A$ Stromaufnahme der RTC ergibt sich eine Laufzeit von etwa:

$$\frac{32\,mA\,h}{5\,\mu A} = 6400\,h \approx 266\,d$$

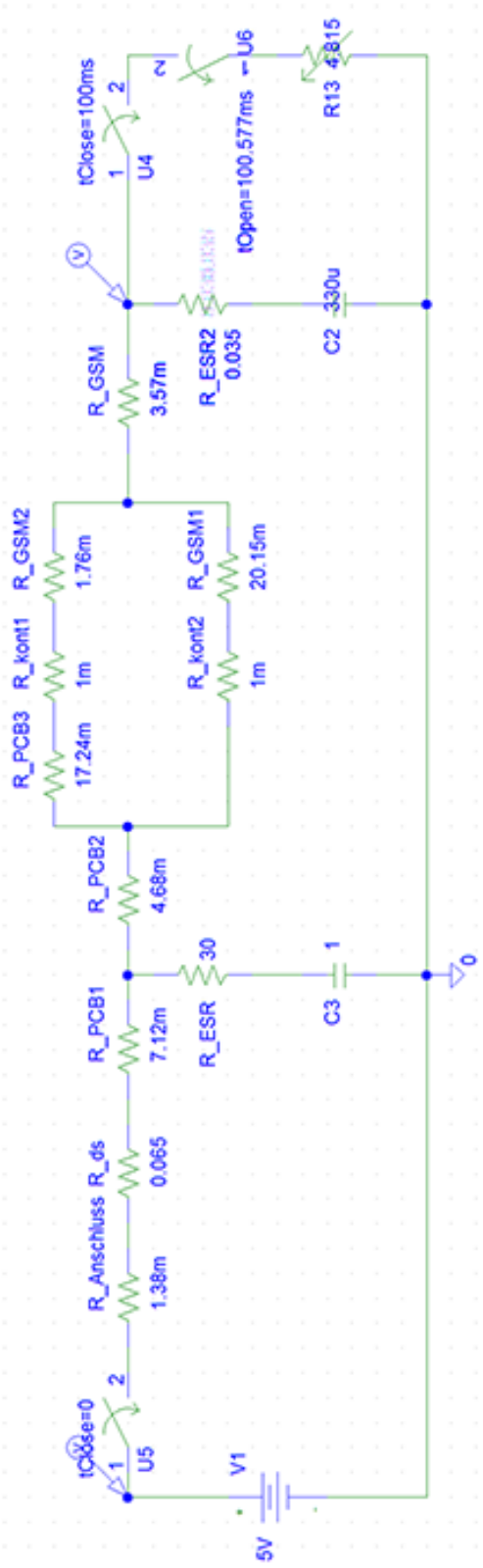


Abbildung 5.12.: Simulationsmodell zum Spannungsabfall der Versorgungsspannung bei GPRS-Übertragungs-Burst

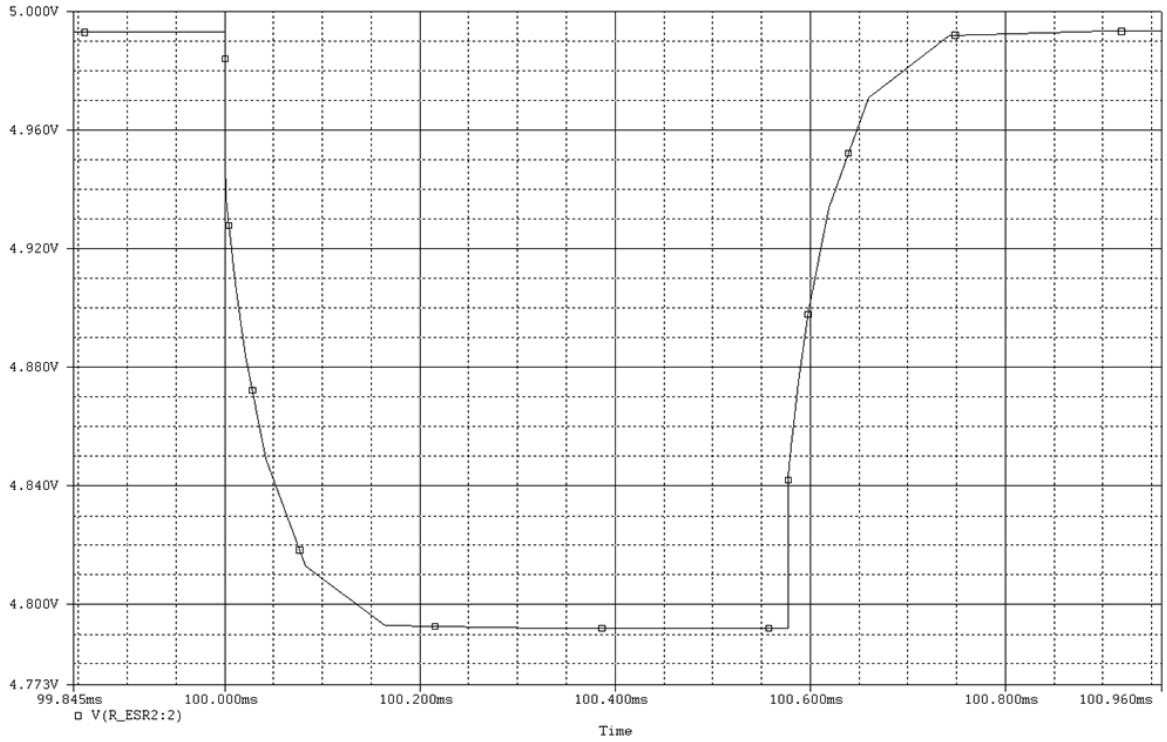


Abbildung 5.13.: Simulierter Spannungsabfall der Versorgungsspannung bei GPRS-Übertragungs-Burst

5.4.2. Inbetriebnahme

Die Inbetriebnahme nach der Bestückung erfolgt gemäß des Inbetriebnahme-Plans in Tab. 5.4 und wird durch einen Funktionstest mittels des Software-Modultests (Abschnitt 6.5) abgeschlossen.

	OK?
Anlegen von $5\,\mathrm{V}$ an +5V	
$0\,\mathrm{V}$ an V_GSM?	
GSM_ON an $3.3\,\mathrm{V}$ anlegen	
$4.1\,\mathrm{V}$ an V_GSM?	
LED2 leuchtet?	
GSM_PWRKEY für $5\,\mathrm{s}$ an $3.3\,\mathrm{V}$	
Abwarten von etwa $30\,\mathrm{s}$	
LED1 blinkt gleichmäßig?	

Tabelle 5.4.: Inbetriebnahme-Plan für GSM-/GPRS-Erweiterungsplatine

5.5. GNSS-Empfänger

5.5.1. Entwurf

Der Schaltplan zu den GNSS-Empfängern findet sich in Anhang B.5. Die bestückte Platine ist in Abb. 5.14 mit Markierungen zur Herausstellung der wichtigen Funktionsgruppen versehen. Im Weiteren werden wesentliche Entwurfsentscheidungen für dieses Hardware-Modul angeführt.

Um den GNSS-Empfänger auch bei größeren Kabellängen noch mit der nötigen Betriebsspannung versorgen zu können, liegt an der Zuleitung eine Spannung von $5\,\mathrm{V}$ an, welche erst im GNSS-Empfänger durch einen Low-Dropout-Linearregler [13] auf die benötigten $3.3\,\mathrm{V}$ heruntergeregelt wird. Der Linearregler kann die Ausgangsspannung selbst bei einer Eingangsspannung von nur $3.6\,\mathrm{V}$ zur Verfügung stellen. Die Komponenten im GNSS-Empfänger können bis zu einer Spannung von etwa $2.7\,\mathrm{V}$ innerhalb ihrer Spezifikation betrieben werden, sodass ein Betrieb selbst mit einem größeren Spannungsabfall in der Zuleitung möglich ist.

Da die erste Positionsbestimmung durch das GNSS-Modul deutlich beschleunigt wird, wenn das Modul die UTC-Zeit sowie die Positionen und Umlaufbahnen möglicher Satelliten bereits kennt, bietet es sich an, den Speicher des Moduls durch einen Kondensator zu stützen.

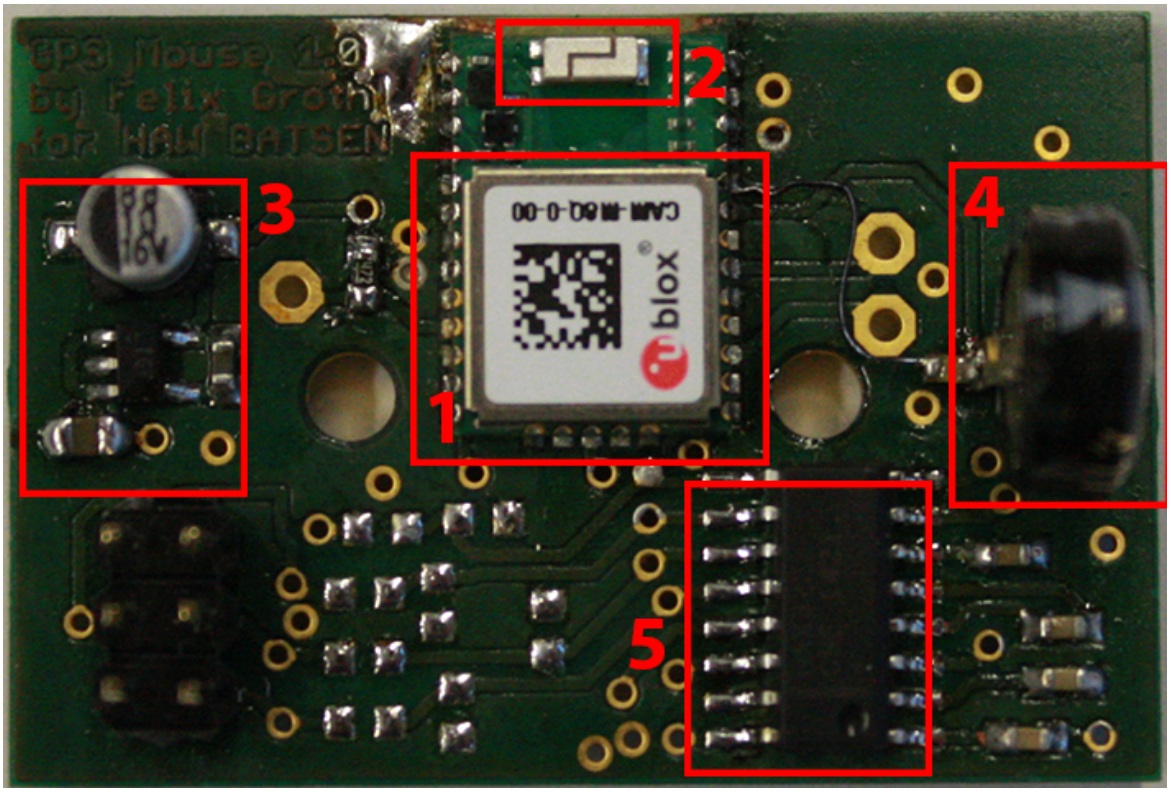


Abbildung 5.14.: GNSS-Empfänger, 1: GNSS-Modul, 2: Chipantenne auf dem Modul, 3: Low-Dropout-Linearregler zur Versorgung mit 3.3 V, 4: Goldcap zur kurzzeitigen Stützung des Speichers im GNSS-Modul, 5: RS232-Pegelwandler

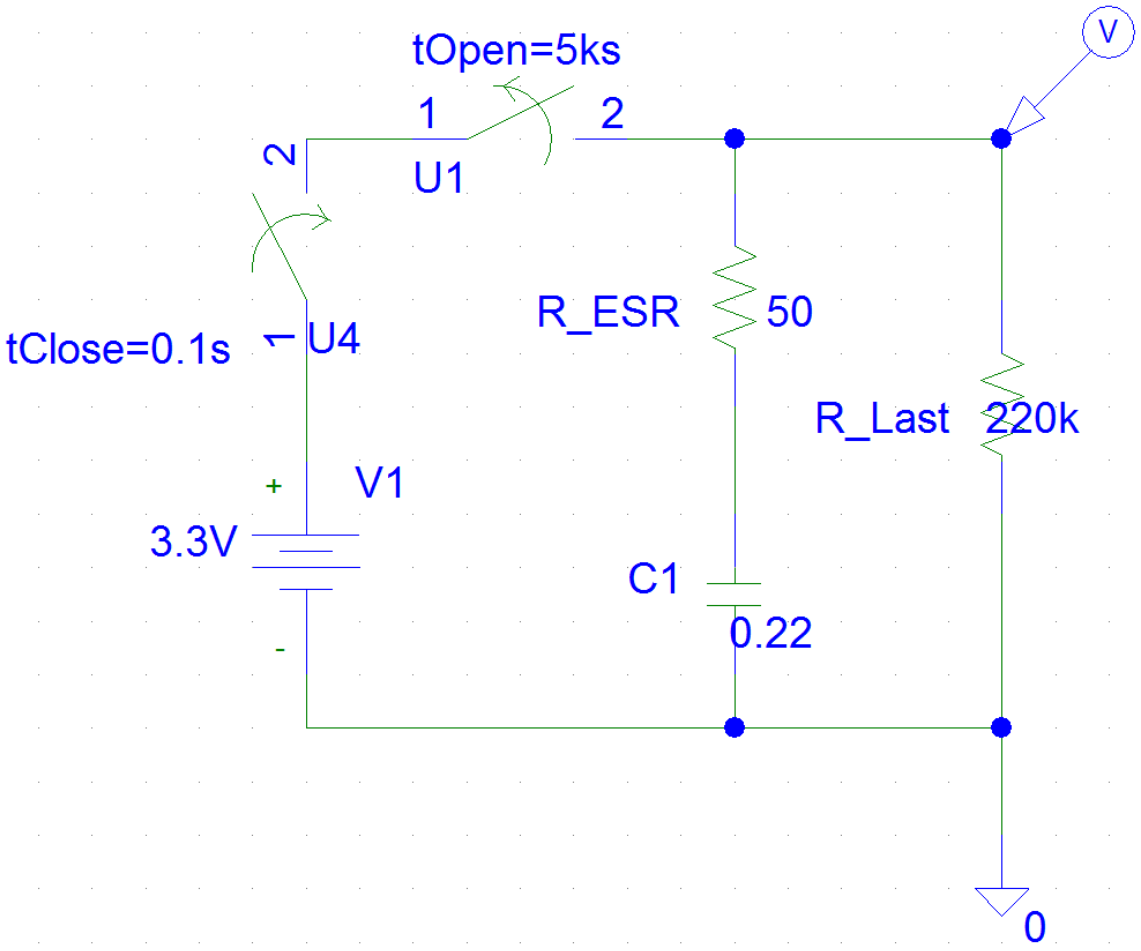


Abbildung 5.15.: Simulationsmodell zur Entladekurve des Stützkondensators für RTC und Speicher im GNSS-Empfänger

Da der Datenlogger über eine batteriegestützte RTC bereits auf dem GSM-/GPRS-Modul verfügt, reicht es aus, kürzere Unterbrechungen der Spannungsversorgung, z.B. beim Aus- und wieder Einschalten des Datenloggers, zu überbrücken. Aus dem verwendeten Goldcap-Kondensator, welcher eine Kapazität von $220\,\mathrm{mF}$ hat, fließt zur Stützung des Speichers und der RTC ein Strom von $15\,\mu\mathrm{A}$, laut Datenblatt [41], bei einer minimalen Spannung von $1.4\,\mathrm{V}$. Die Entladekurve wurde in PSpice mit dem Modell in Abb. 5.15 simuliert, wobei sich die Kurve in Abb. 5.16 ergab. Voraussichtlich können der Speicher und die RTC aus dem Goldcap also für etwa $44\,000\,\mathrm{s} \approx 12\,\mathrm{h}$ gestützt werden.

Als Gegenstück zu den RS232-Pegelwandlern auf der Datenlogger-Erweiterungsplatine kommen auch im GNSS-Empfänger MAX3232 [17] der Firma Maxim Integrated zum Ein-

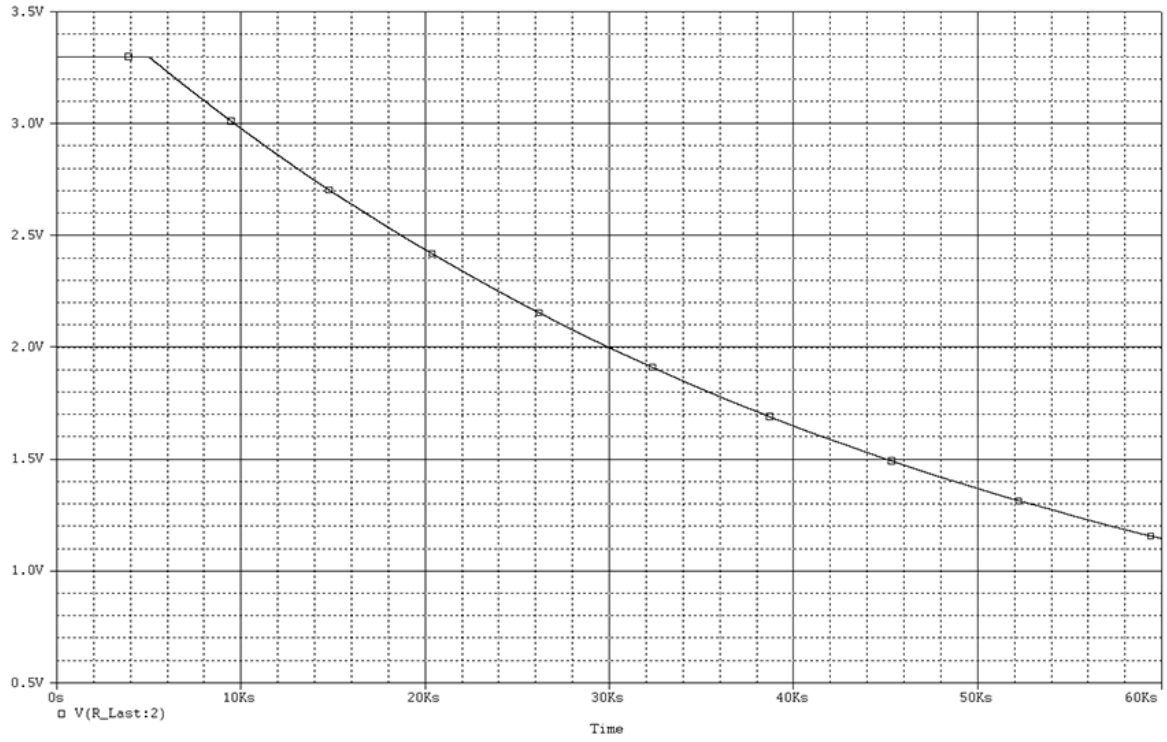


Abbildung 5.16.: Simulierte Entladekurve des Stützkondensators für RTC und Speicher im GNSS-Empfänger

satz, die die Pegel auf den verwendeten Datenleitungen mit Pegeln zwischen $0\,\mathrm{V}$ bis $3.3\,\mathrm{V}$ für das GNSS-Modul zur Verfügung stellen.

5.5.2. Inbetriebnahme

Die Inbetriebnahme nach der Bestückung erfolgt gemäß des Inbetriebnahme-Plans in Tab. 5.5 und wird durch einen Funktionstest mittels des Software-Modultests (Abschnitt 6.5) abgeschlossen.

	OK?
Anlegen von $5\,\mathrm{V}$ an +5V	
$3.3\,\mathrm{V}$ an +3V3?	
$5\,\mathrm{V}$ anlegen an RXD	
GPS_RXD $\approx 3.3\,\mathrm{V}$?	
$5\,\mathrm{V}$ anlegen an INT	
GPS_EXTINT $\approx 3.3\,\mathrm{V}$?	

Tabelle 5.5.: Inbetriebnahme-Plan für GNSS-Empfänger

Antenne

Die Erprobung der GNSS-Empfänger mittels des Software-Tests in Abschnitt 6.5 zeigt eine sehr geringe Anzahl von empfangenen Satelliten und eine Positionsbestimmung ist nur stationär nach einigen Minuten Wartezeit möglich. Wird der Empfänger am Fahrzeug oder der Person mitgeführt, ist eine Positionsbestimmung nur in seltenen Fällen möglich.
Um die Empfangsqualität zu verbessern, wird die kurzfristig verfügbare Patchantenne 1590R-A des Herstellers Abracon [1] auf einem Leiterplattenstück verbaut, welches an das Gehäuse des GNSS-Empfängers angepasst ist. In Abb. 5.17 ist die Antenne im Gehäuse abgebildet. Wie in Abb. 5.18 zu sehen ist, wird die Antenne mit einem Stück Koaxialleitung an das GNSS-Modul angeschlossen.

Mit der neuen Antenne stellt sich eine erheblich verbesserte Empfangsleistung ein. Eine Positionsbestimmung ist nun sogar innen am Fenster von Fahrzeugen problemlos möglich, wie in den Messungen zur Erprobung in Kapitel 7 gezeigt wird.

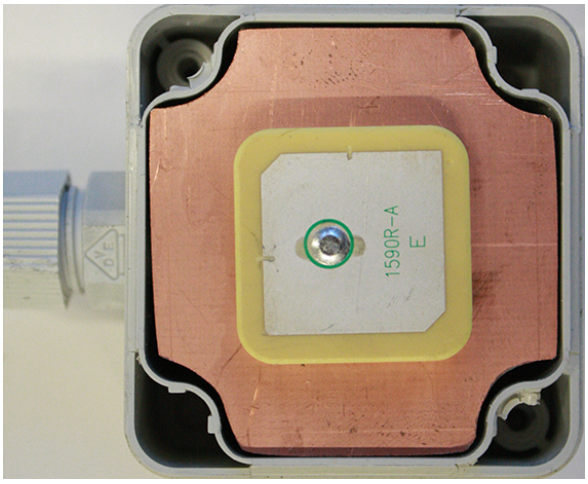


Abbildung 5.17.: Passive GNSS-Antenne

Abbildung 5.18.: Passive Antenne am GNSS-Modul

6. Software-Entwurf und Implementierung

6.1. Funktionsmodule und Schnittstellen

Die Entwicklung der Software für den Mikrocontroller erfolgt in „Code Composer Studio“, einer von Texas Instruments an die Entwicklung für deren Mikrocontroller angepassten Version der Entwicklungsumgebung „Eclipse“. Jegliche Software für den Mikrocontroller wird in C implementiert, lediglich für die PC-Software zum Datenauslesen (Abschnitt 6.4.1) und der Funktionsprüfung des Inertialmesssystems (Abschnitt 6.5) kommt C++ zum Einsatz.

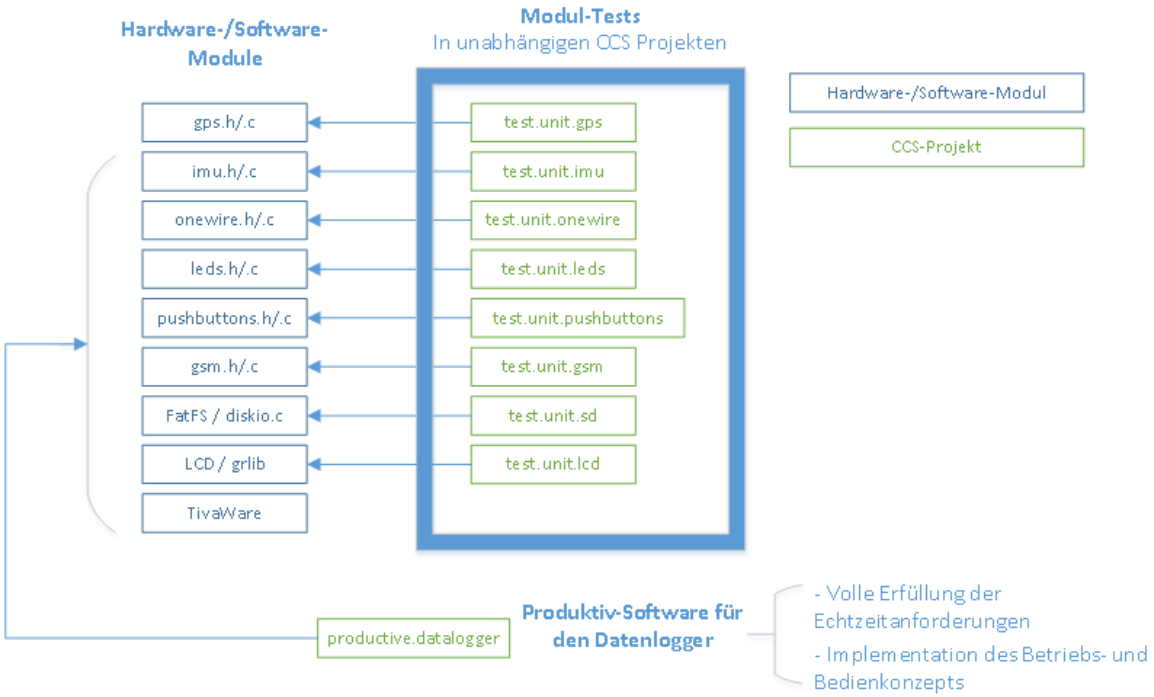


Abbildung 6.1.: Software-Entwicklungskonzept: Funktionsmodule und CCS-Projekte

Bei der Software-Entwicklung für den Datenlogger werden die Implementierungen für die einzelnen Funktionsmodule jeweils in eigenen Dateien ausgeführt. Außerdem werden für die Funktionen Modultests implementiert, welche in Abschnitt 6.5 beschrieben weden.
Wie in Abb. 6.1 dargestellt, werden die Software-Module dann sowohl im Hauptprogramm des Datenloggers (Abschnitt 6.4) als auch in den Modultests eingebunden, sodass die Tests anhand derselben Implementierung durchgeführt werden können. Für die Produktiv-Software des Datenloggers sowie die Modultests werden jeweils eigenen CCS-Projekte angelegt, welche unabhängig voneinander kompiliert und ausgeführt werden können.

Als Grundlage verwenden alle Module die Treiberbibliothek „TivaWare“, welche die Zugriffe auf die Hardware des Mikrocontrollers in C-Funktionen abstrahiert. Die Header-Dateien der Bibliothek werden je nach Bedarf in den einzelnen Software-Modulen eingebunden, sodass ein verwendendes Hauptprogramm nur die Header der Software-Module und aus der TivWare lediglich selbst explizit verwendete Funktionsteile einbinden muss.

6.1.1. Inertialmesssystem

Für das Inertialmesssystem wurden folgende Schnittstellen implementiert:

- Funktionen zur Initialisierung und Konfiguration des Messsystems
- Datenstrukturen zur Abbildung des Übertragungsformats
- Globaler Ringbuffer zur Aufnahme der Messwerte
- Funktion zum Abrufen des aktuellen Messwertes aus dem Hardware-Modul in den Ringbuffer

Der Betriebsmodus des Inertialmesssystems ist dabei eine periodische Messung der Beschleunigungen und Drehraten mit $1\,\mathrm{kHz}$. Die Signale werden auf dem Inertialmesssystem durch ein digitales Tiefpassfilter vorgefiltert. Die Grenzfrequenzen sind dabei auf $184\,\mathrm{Hz}$ für die Beschleunigungsmessung und $188\,\mathrm{Hz}$ für die Drehratensensoren eingestellt, sodass diese deutlich oberhalb der Nyquist-Freuqenz der Abtastung durch den Datenlogger liegen. Es gehen durch die Tiefpassfilter also keine Informationen verloren. Damit stellen die gewählten Grenzfrequenzen eine sinnvolle Wahl aus den zehn auf dem Inertialmesssystem auswählbaren dar.

Da das Inertialmesssystem keine Synchronisation seiner Abtastung auf ein externes Signal ermöglicht, ruft das Hauptprogramm des Datenlogger zu den gewählten Abtastzeitpunkten den jeweils aktuellen Messwert ab. Mit der Annahme, dass die Taktbasis von Inertialmesssystem und Datenlogger-Hauptprogramm von einander leicht abweichen, ergibt sich, dass der tatsächliche Abstand zweier Messungen bei $9\,\mathrm{ms}$, $10\,\mathrm{ms}$ und $11\,\mathrm{ms}$ liegen kann. Um die gewünschen $10\,\mathrm{ms}$ entsteht damit u.U. ein Jitter von $2\,\mathrm{ms}$.

Datenstrukturen im globalen Ringbuffer enthalten die Information, ob neue Messdaten erhalten wurden. Die gelesene Datenstruktur muss durch das Hauptprogramm wieder freigegeben werden.

6.1.2. GNSS-Empfänger

Für die GNSS-Empfänger wurden folgende Schnittstellen implementiert:

- Funktion zur Initialisierung und Konfiguration des Empfängers
- Datenstrukturen zur Abbildung des uBlox Nachrichtenformats [42]
- Globaler Ringbuffer zur Kommunikation in Sende und Empfangsrichtung
- Funktionen zur Konfiguration und Synchronisation des Timepulse-Signals wie in Abschnitt 6.3.2 beschrieben

Die GNSS-Empfänger werden bei der Initialisierung so konfiguriert, dass sie eine Nachricht mit den aktuellen Positions- und Bewegungsinformationen und der zugehörigen UTC-Zeit alle $100\,\mathrm{ms}$ selbständig über die UART-Schnittstelle übertragen. Diese werden in den entsprechenden Datenstrukturen des Ringbuffers abgelegt.

Datenstrukturen im globalen Ringbuffer enthalten die Information, ob neue Nachrichten des GNSS-Empfängers vorliegen. Die gelesene Datenstruktur muss durch das Hauptprogramm wieder freigegeben werden.

6.1.3. Temperatursensoren

Für die Temperatursensoren wurden folgende Schnittstellen implementiert:

- Funktion zur Suche nach angeschlossenen Sensoren
- Datenstrukturen zur Abbildung der Messdaten
- Funktion zum Auslösen der Messung auf einem bestimmten Sensor
- Funktion zum Auslesen der letzten durchgeführten Messung aus einem Sensor

Die Messung auf den einzelnen Sensoren muss nacheinander erfolgen, wenn die Sensoren, wie im aktuellen Hardware-Aufbau, über die Datenleitung mit der Betriebsspannung versorgt werden. Messungen können zu beliebigen Zeiten ausgeführt werden und das Messergebnis später vom Sensor abgerufen werden.

6.1.4. Weitere Software-Module

Für die Taster sind Funktionen zum Abruf der Tasterzustände sowie dem Festlegen einer Interrupt-Service-Routine für Tastendrücke implementiert.

Die Leuchtdioden können über einfache Funktionen gesetzt und deren Status gelesen werden.

Aus dem GSM-Modul können Zeit und Datum aus der RTC in die des Mikrocontrollers übernommen werden. Das Modul kann über Funktionsaufrufe eingeschaltet und gestartet werden.

Das FatFS Dateisystem wurde aus der TivaWare übernommen und der Treiber zum Zugriff auf die SD-Karte mit DMA Unterstützung von Magnuson [37] so verändert, dass er die Hardware-Schnittstellen des verwendeten Evaluationsboards korrekt verwendet.

Alle Ansteuerungsfunktionen für den LCD-Touchscreen stehen in der Grafik-Bibliothek „grlib“ als Teil der TivaWare zur Verfügung. Lediglich die Festlegung der zu verwendenden Hardware-Schnittstellen des Mikrocontrollers wurde angepasst.

6.2. Formatierung der Ausgabedaten

6.2.1. Dateiformat

Die Datenausgabe soll in einem MATLAB-Dateiformat erfolgen. Dazu wird auf Grundlage des mit TivaWare [36] veröffentlichten FatFS-Dateisystems eine Implementierung des Level-4-MAT-Dateiformates [15] vorgenommen. Dieses Dateiformat kann in allen aktuellen MATLAB-Versionen geöffnet werden, ist aber deutlich einfacher zu implementieren als neuere Formate.
Mit der Wahl des Level-4-Dateiformates entstehen die folgenden relevanten Einschränkungen:

- Vektoren und Matrizen können nur einen Datentyp enthalten.
- Es können keine „komplexen“ Datenstrukturen abgelegt werden.

Das Level-4-Dateiformat speichert zudem die Werte von Vektoren oder Matrizen hintereinander und kann diese nicht in der Datei verschachteln. Bei der mit der Messdauer wachsenden Anzahl von Elementen in den zu speichernden Vektoren hat dies zur Folge, dass alle in der Datei folgenden Strukturen verschoben werden müssten, wenn im davor liegenden Vektor keine freien Datenfelder mehr zur Verfügung stehen. Zwar könnten alle Datenstrukturen mit vielen leeren Elementen vorbelegt werden, dies hätte allerdings auch für kurze Messungen

sehr große Dateien zur Folge. Zudem müsste das Erreichen der leer vorbelegten Anzahl an Elementen dazu führen, dass während der Messung eine weitere Datei erstellt und die Felder vorbelegt würden, bevor die Messung fortgeführt werden kann. Dieser Vorgang würde zum Verlust von Messwerten führen.

Zur Lösung wird deshalb Folgendes gewählt und festgelegt:

- Für jeden Vektor / jede Matrix wird eine eigene Datei angelegt und während der Messung offen gehalten.
- Jede Datenstruktur wird anhand der vorgesehenen Abtastrate mit genug leeren Elementen für eine Messdauer von 24 h vorbelegt.
- Nach der Beendigung einer Messung werden die Dateien auf die nötige Länge gekürzt, um keinen unnötigen Speicherplatz zu belegen.
- Die Speicherung der Messdaten erfolgt im nativen Datentyp und muss ggf. offline in SI-Einheiten / andere Bezugsgrößen umgerechnet werden.

6.2.2. Dateien und Datentypen

Die Dateien und Datentypen, wie in Tab. 6.1 bis 6.4 dargestellt, werden, soweit die entsprechenden Module angeschlossen und betriebsbereit sind, bei der Aufzeichnung durch den Datenlogger erstellt. Ausgabedatentypen sind dabei an den intern verwendeten Festkommaformaten orientiert, sodass u.U. eine Umrechnung der Daten vor der Weiterverarbeitung erfolgen muss. Mit diesem Vorgehen wird die Erfüllung der Echtzeitanforderungen vereinfacht, da keine Rechenzeit des Mikrocontrollers für die Umrechnung aufgewandt werden muss.
Jede Datei enthält einen Vektor oder eine Matrix, deren Name dem Dateinamen ohne Endung entspricht.
Für jede Messung wird auf der Speicherkarte ein neuer Ordner mit fortlaufender Nummerierung angelegt und die Dateien werden darin abgelegt.

In Tab. 6.1 sind die Ausgabedateien eines GNSS-Empfängers aufgelistet. Diese werden mit der Endung „2.MAT“ genauso auch für den zweiten GNSS-Empfänger geschrieben. Die in den Tab. 6.3 und 6.4 gelisteten Ausgabedaten verwenden die angegebenen Zeitstempel als Zeitbasis mit, sodass der Zugriff auf weitere dauerhaft geöffnete Dateien nicht nötig wird.

Dateiname	Inhalt	Datentyp	Einheit
TIME1.MAT	Vektor, Zeitstempel zur Positionsbestimmung	32-bit unsigned Integer	ms
LAT1.MAT	Vektor, Breitengrad	32-bit signed Integer	1×10^7 deg
LON1.MAT	Vektor, Längengrad	32-bit signed Integer	1×10^7 deg
HEIGHT1.MAT	Vektor, Höhe über NN	32-bit signed Integer	mm
SPEED1.MAT	Vektor, Geschwindigkeit über Grund	32-bit signed Integer	$\frac{\text{mm}}{\text{s}}$
NUMSV1.MAT	Vektor, Anzahl verwendeter Satelliten	8-bit unsigned Integer	Stk.
FIX1.MAT	Vektor, Typ der Positionisbestimmung	8-bit unsigned Integer	0: Kein Fix, 2: 2D-Fix, 3: 3D-Fix, 5: Zeit

Tabelle 6.1.: Dateien, Datentypen und Einheiten der Datenausgabe des Datenloggers: Satellitenortung

Dateiname	Inhalt	Datentyp	Einheit
TIME3.MAT	Vektor, Zeitstempel zur Inertialmessung	32-bit unsigned Integer	ms
AX.MAT	Vektor, Beschleunigung X-Achse	16-bit signed Integer	$\frac{2 \cdot 9{,}81}{2^{15}} \frac{\text{m}}{\text{s}}$
AY.MAT	Vektor, Beschleunigung Y-Achse	16-bit signed Integer	$\frac{2 \cdot 9{,}81}{2^{15}} \frac{\text{m}}{\text{s}}$
AZ.MAT	Vektor, Beschleunigung Z-Achse	16-bit signed Integer	$\frac{2 \cdot 9{,}81}{2^{15}} \frac{\text{m}}{\text{s}}$
GX.MAT	Vektor, Drehrate X-Achse	16-bit signed Integer	$\frac{250}{2^{15}} \frac{\text{deg}}{\text{s}}$
GY.MAT	Vektor, Drehrate Y-Achse	16-bit signed Integer	$\frac{250}{2^{15}} \frac{\text{deg}}{\text{s}}$
GZ.MAT	Vektor, Drehrate Z-Achse	16-bit signed integer	$\frac{250}{2^{15}} \frac{\text{deg}}{\text{s}}$

Tabelle 6.2.: Dateien, Datentypen und Einheiten der Datenausgabe des Datenloggers: Inertialmesssystem

Dateiname	Inhalt	Zeitbasis	Datentyp	Einheit
TEMP1.MAT	Vektor, Temperatur 1	TIME1.MAT	16-bit signed Integer	0.0625 °C
TEMP2.MAT	Vektor, Temperatur 2	TIME1.MAT	16-bit signed Integer	0.0625 °C
TEMP3.MAT	Vektor, Temperatur 3	TIME1.MAT	16-bit signed Integer	0.0625 °C

Tabelle 6.3.: Dateien, Datentypen und Einheiten der Datenausgabe des Datenloggers: Temperatursensoren

Dateiname	Inhalt	Zeitbasis	Datentyp	Einheit
YEAR.MAT	Vektor Jahr nach UTC-Zeit	TIME1.MAT	16-bit unsigned Integer	a
DATETIME.MAT	Matrix, 5 Zeilen	TIME1.MAT	8-bit unsigned Integer	
	Monat	-“-	-“-	mon
	Tag	-“-	-“-	d
	Stunde	-“-	-“-	h
	Minute	-“-	-“-	min
	Sekunde	-“-	-“-	s

Tabelle 6.4.: Datenausgabe des Datenloggers: UTC-Zeit, Aufschlüsselung der Spaltenvektoren

6.3. Systemzeitquellen und Synchronisation

6.3.1. Systemzeitquellen für relative Messdauer und UTC-Zeit

Wie bereits in 4.2.5 gezeigt, ist für die weitere Verarbeitung der Messwerte eine äquidistante Abtastung wichtig. Bei der Realisierung der Funktionalität in Software muss deshalb eine gemeinsame, möglichst stabile Zeitbasis für die Abtastung der verschiedenen Sensoren und Empfänger eingesetzt werden. Außerdem soll eine Synchronisation der Abtastzeitpunkte zur UTC-Zeit erreicht werden.

Eine Interrupt-Service-Routine soll die relative Dauer der aktuellen Messung in ms zählen, sodass der Zählerwert verwendet werden kann, um abgetastete Werte aller Signalquellen mit einem Zeitstempel auf gemeinsamer Basis zu versehen. Zumindest für einen dieser Zeitstempel muss die UTC-Zeit bekannt sein, für jeden anderen Zeitstempel kann sie dann daraus abgeleitet werden.

Da die Zeitpunkte der Positionsbestimmung durch die GNSS-Module äquidistant sowie zwangsweise synchron zur UTC-Zeit sind und durch Konfiguration lediglich in ihrer Häufigkeit

beeinflusst werden können, bietet es sich an, die Erfassung an den anderen Sensoren daran zu synchronisieren. Deshalb wird die Zeitbasis eines GNSS-Empfängers, wenn verfügbar, für das ganze System des Datenloggers verwendet. Dazu kann der GNSS-Empfänger zur Ausgabe eines auf die UTC-Zeit synchronisierten „Timepulse“-Rechtecksignals mit einstellbarer Frequenz konfiguriert werden. Das Rechtecksignal wird auf eine Frequenz von $1\,\mathrm{kHz}$ eingestellt und die steigende Flanke zur Auslösung der Interrupt-Service-Routine verwendet.

Da für den Betrieb des Datenloggers nicht zwangsweise GNSS-Empfänger anzuschließen sind, werden alternative Zeitbasen gebraucht. Damit also der Datenlogger in verschiedener Hardware-Konfiguration betrieben werden kann, ist die folgende Verwendung von Taktgebern für die Interrupt-Service-Routine, mit absteigender Priorität, umgesetzt:

- Priorität 1: Timepulse-Signal des GNSS-Empfängers 1
- Priorität 2: Timepulse-Signal des GNSS-Empfängers 2
- Priorität 3: Interner Timer des Mikrocontrollers

Dazu wird beim Initialisieren der Funktionsmodule im Hauptprogramm die Verfügbarkeit der jeweiligen Taktquelle überprüft und bei Nichtverfügbarkeit auf die Nächste zurückgefallen.

Als Quelle für die absolute UTC-Zeit wird zunächst während der Initialisierung die batteriegestützte RTC des GSM-/GPRS-Moduls abgefragt und Zeit sowie Datum in die RTC des Mikrocontrollers übernommen. Da aber die GNSS-Empfänger, wenn Daten von zumindest einem Satelliten empfangen werden, mit den Positionsnachrichten eine besser synchronisierte UTC-Zeitinformation übertragen, wird diese mit höherer Priorität verwendet:

- Priorität 1: UTC-Information des GNSS-Empfängers 1
- Priorität 2: UTC-Information des GNSS-Empfängers 2
- Priorität 3: RTC des Mikrocontrollers bzw. GSM-/GPRS-Moduls

Wie in Abschnitt 6.3.2 beschrieben, ist die UTC-Zeitinformation der GNSS-Empfänger zudem auf $1\,\mathrm{ms}$ genau zur relativen Messdauer synchronisiert.

6.3.2. Synchronisation der GNSS-Nachrichten

Gemäß dem Datenblatt des Herstellers [42] findet zwar die Positionsbestimmung anhand von äquidistant erfassten Satellitensignalen statt, die regelmäßige Übertragung der berechneten Positionsdaten an den Mikrocontroller ist dazu allerdings um eine variable Berechnungsdauer verzögert. Anhand einer angepassten Test-Software in Quellcode D.27 wurde mittels der

Messung einer Hüllkurve am Oszilloskop der Jitter der Datenübertragung betrachtet. Als Referenz diente dabei das periodische Timepulse-Signal. Wie in Abb. 6.2 dargestellt, ergibt sich ein Jitter von mehreren ms.
Diese Tatsache hat zur Folge, dass eine eingehende Nachricht mit Positionsinformationen vom GNSS-Empfänger nicht ohne Weiteres zuverlässig einer bestimmten Flanke des Timepulse-Signals zugeordnet werden kann. Der genaue Abtastzeitpunkt ist also zunächst unbekannt. Um diese Zuordnung zu ermöglichen, wird die Tatssache ausgenutzt, dass sowohl das Timepulse-Signal, als auch die Positionsbestimmung auf dem GNSS-Modul am Beginn der UTC-Sekunden ausgerichtet sind.

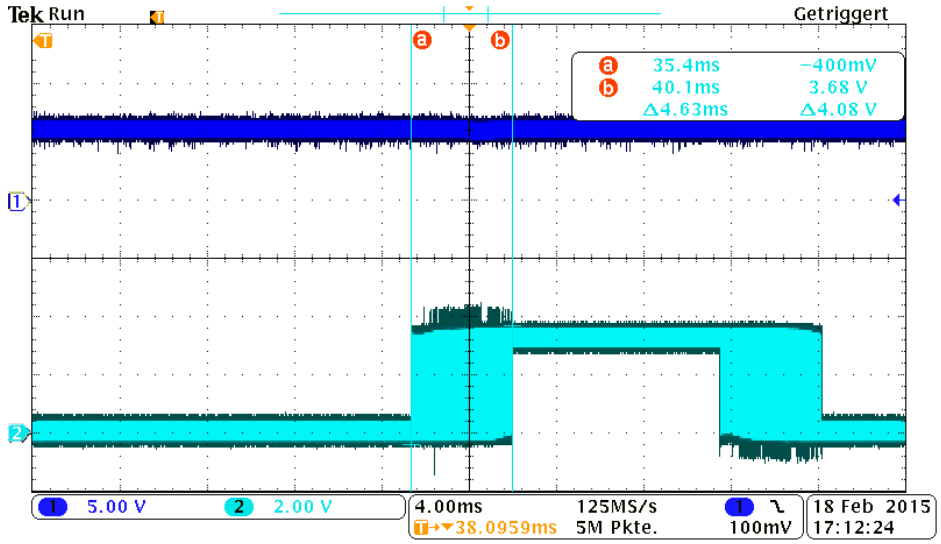


Abbildung 6.2.: Jitter-Messung der GNSS-Nachrichten mittels Hüllkurve, türkis: Controller-Pin signalisiert Empfangs-Interrupt

Das implementierte Verfahren zum Versehen von Positions- und UTC-Datensätzen des GNSS-Empfängers mit Zeitstempeln auf Basis der relativen Messdauer ist in Abb. 6.3 als vereinfachtes Struktogramm dargestellt.
Einmalig beim Einschalten des Datenloggers wird dazu wie folgt vorgegangen: Das Timepulse-Signal wird auf eine Taktrate von 1 Hz konfiguriert, sodass nur zum Beginn jeder UTC-Sekunde eine Taktflanke anliegt. Mit Hilfe eines internen Timers wird nun vom Beginn einer durch den Timepulse detektierten UTC-Sekunde mit dem Systemtakt gezählt. Außerdem wird die Taktrate des Timepulse-Signals nun auf die Zielrate von 1 kHz erhöht. Bei jeder nun detektierten Flanke wird anhand des Zählerwertes des internen Timers überprüft, ob seit der Detektierung des UTC-Sekundenbeginns

$$n \cdot 1\,\mathrm{s} \pm 0.5\,\mathrm{ms} \quad \text{mit} \quad n \in \mathbb{R}$$

vergangen sind. Ist die Bedingung erfüllt, hat die detektierte Taktflanke zum Beginn einer UTC-Sekunde stattgefunden. Mit dem Wissen, dass bei einer Abtastrate von $10\,\mathrm{Samples/s}$ alle $100\,\mathrm{Takte}$ eine Positionsbestimmung folgt, kann nun regelmäßig die aktuelle relative Messdauer in ms gespeichert und die nächste eingehende Nachricht des GNSS-Empfängers mit diesem Zeitstempel versehen werden.
Der zeitliche Verlauf der Zählerwerte sowie des Timepulse-Signals während des Verfahrens sind in Abb. 6.4 und 6.5 dem Prinzip nach dargestellt. Innerhalb des rot markierten Bereichs wird eine steigende Flanke korrekt als Erste einer Sekunde identifiziert.
Die Berücksichtigung eines Bereichs von Zählerwerten für die Erkennung ist der Tatsache geschuldet, dass Timer und Timepulse-Signal in der Realität vermutlich nicht synchron sind.

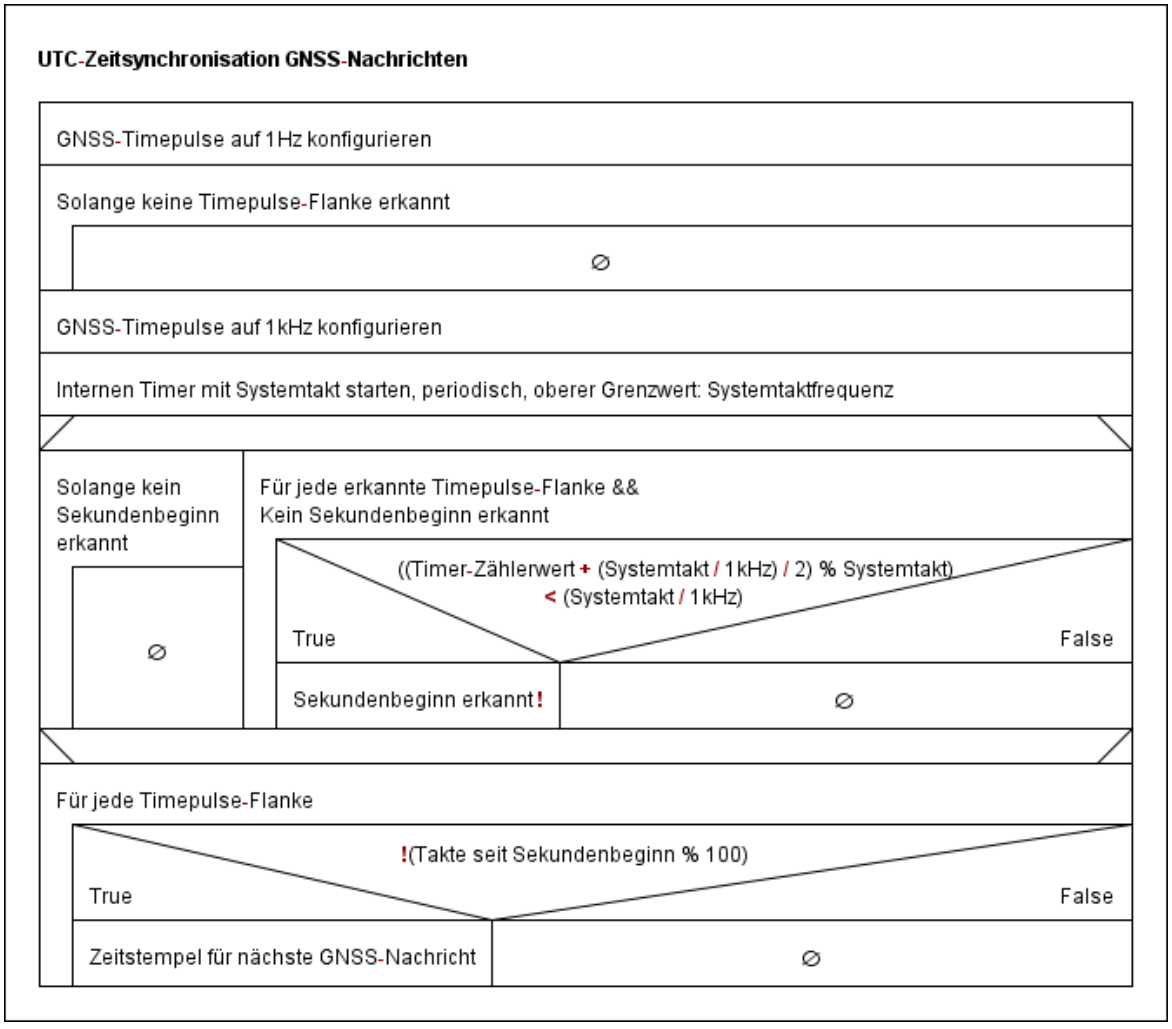


Abbildung 6.3.: Struktogramm zum Programmablauf der Synchronisation von GNSS-Nachrichten zum GNSS-Timepulse

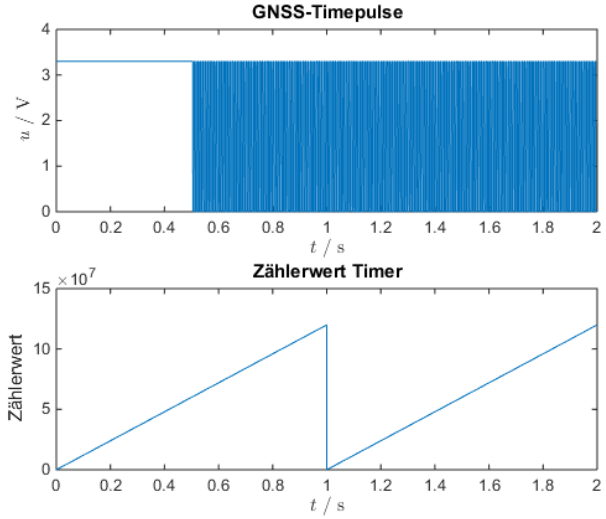


Abbildung 6.4.: Timepulse-Signal und Zählerwert während der Zeitsynchronisation: Das Timepulse-Signal zu beginn mit 1 Hz und wird dann auf 1 kHz erhöht.

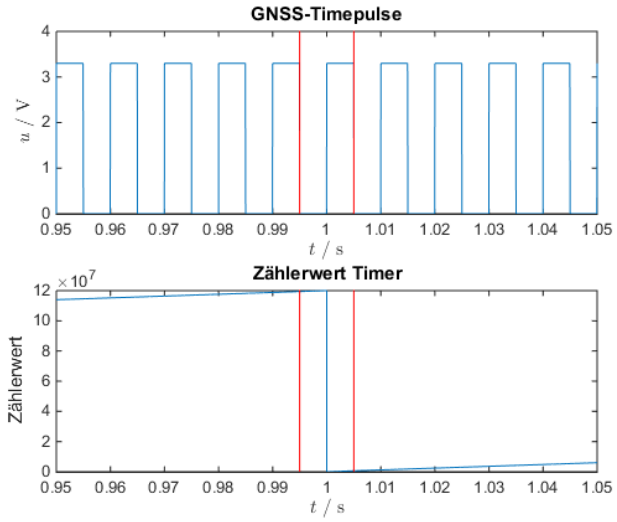


Abbildung 6.5.: Timepulse-Signal und Zählerwert während der Zeitsynchronisation: Zählerwerte innerhalb des roten Bereichs markieren eine Taktflanke zum Beginn einer UTC-Sekunde

6.4. Hauptprogramm und Bedienkonzept

Das Hauptprogramm des Datenloggers folgt im Wesentlichen dem Zustandsautomaten in Abb. 6.6, welches auch das Bedienkonzept beschreibt. Bevor in den Anfangszustand des Automaten gesprungen wird, wird anhand der Kontroll- und Status-Pins des LTC4412 überprüft, ob tatsächlich Versorgungsspannung zur Verfügung steht oder lediglich der Goldcap-Kondensator noch geladen ist. Es folgt die Initialisierung aller Funktionsmodule, soweit angeschlossen.

Im gezeigten Zustandsdiagramm werden in einzelnen Zuständen Makros verwendet, welche umfangreichere Programmabläufe beschreiben. Zudem treten, besonders im Zustand „mainRecord“, durch den Gebrauch von Interrupts und Hardware-Puffern teilweise parallele Vorgänge auf. Abb. 6.7 enthält zum besseren Verständnis der Abläufe eine Darstellung des Verhaltens im Zustand „mainRecord“ mit den Interrupt-Service-Routinen und einer Kurzbeschreibung der Vorgänge. Dabei wird im Hauptprogramm in der Funktion „writeData“ ständig überprüft, ob zu schreibende Daten in den Speicherfeldern der einzelnen Software-Module liegen. Außerdem wird durch den aktiven Zeitgeber für die Relativzeit die Interrupt-Service-Routine „timerMsIsr“ ausgelöst, welche den Relativzeitstempel erhöht und die regelmäßige Messung der Inertial- und Temperatursensoren auslöst. Auch die Kommunikation mit den Modulen findet Interrupt gesteuert und unter Ausnutzung von Hardware-FIFOs statt, sodass u.a. bei der Festlegung der Prioritäten weitere Interruptquellen berücksichtigt werden müssen. Die Interrupt-Service-Routinen sind in Abb. 6.7 mit absteigender Priorität aufgetragen. Prinzipiell haben dabei die Interrupts mit hoher Häufigkeit und Echtzeit-Anforderungen aufgrund der Äquidistanz der Messung die höhere Priorität. So kann eine weitgehende Parallelisierung der Kommunikation zum Zwecke der Auflösung von in 4.4.3 beschriebenen Problemen erreicht werden.

Das Makro „fileTransfer“ wird in Abschnitt 6.4.1 und das Makro „calibrateImu“ in Abschnitt 6.6 beschrieben.

6.4.1. Übertragung von Messdaten über USB

Eine Möglichkeit die Messdaten vom Datenlogger auf den PC zu übertragen, ist die Öffnung des Gehäuses unter Werkzeugeinsatz und die Entnahme der Speicherkarte aus dem Gerät. Die Speicherkarte kann dann mit einem Lesegerät am PC verwendet werden, um die Ordner und Dateien zu kopieren. Gerade bei langen Messungen und damit großen Mengen angefallener Messdaten bietet sich dieses Verfahren an, da die meisten Lesegeräte mit einer hohen Datenübertragungsrate auf die Speicherkarte zugreifen können.

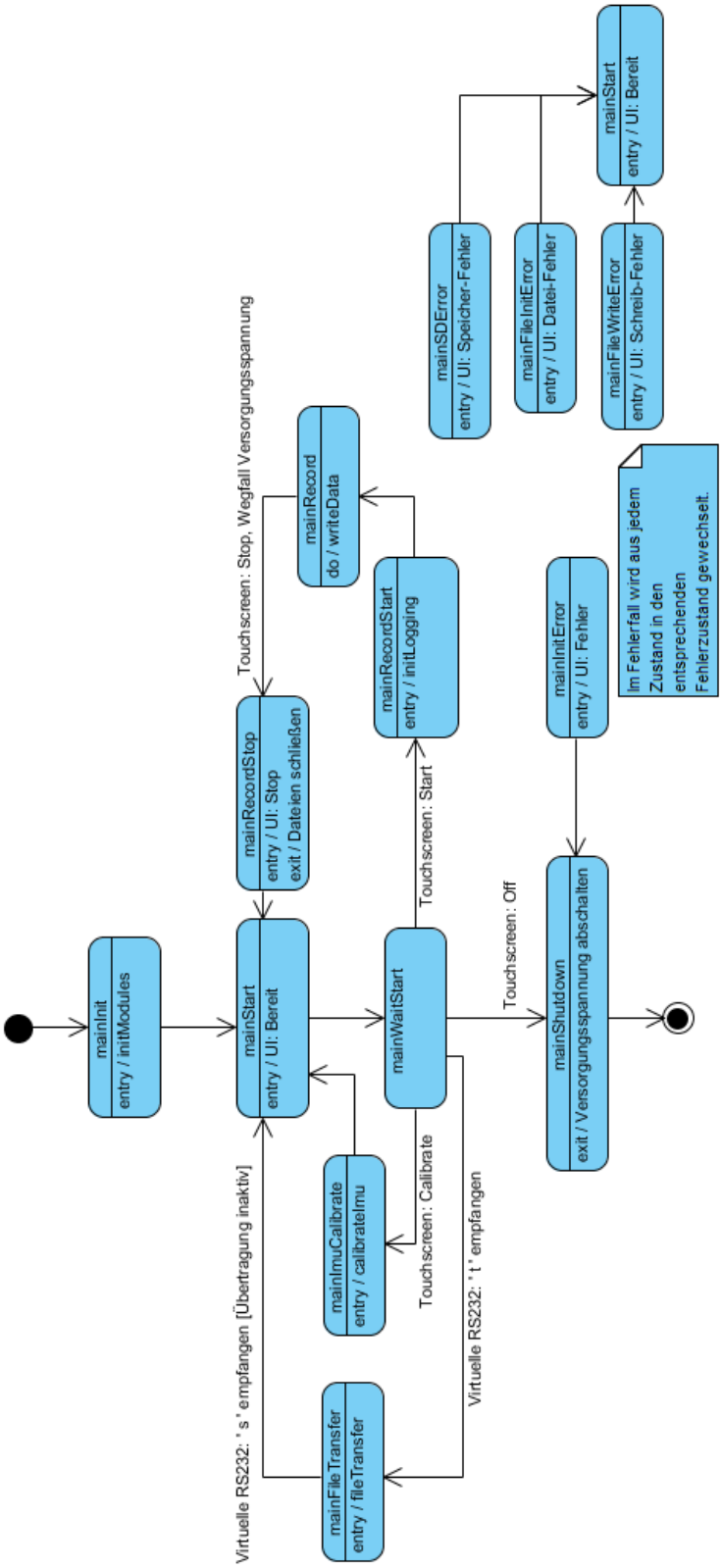


Abbildung 6.6.: Zustandsdiagramm zum Ablauf des Hauptprogramms

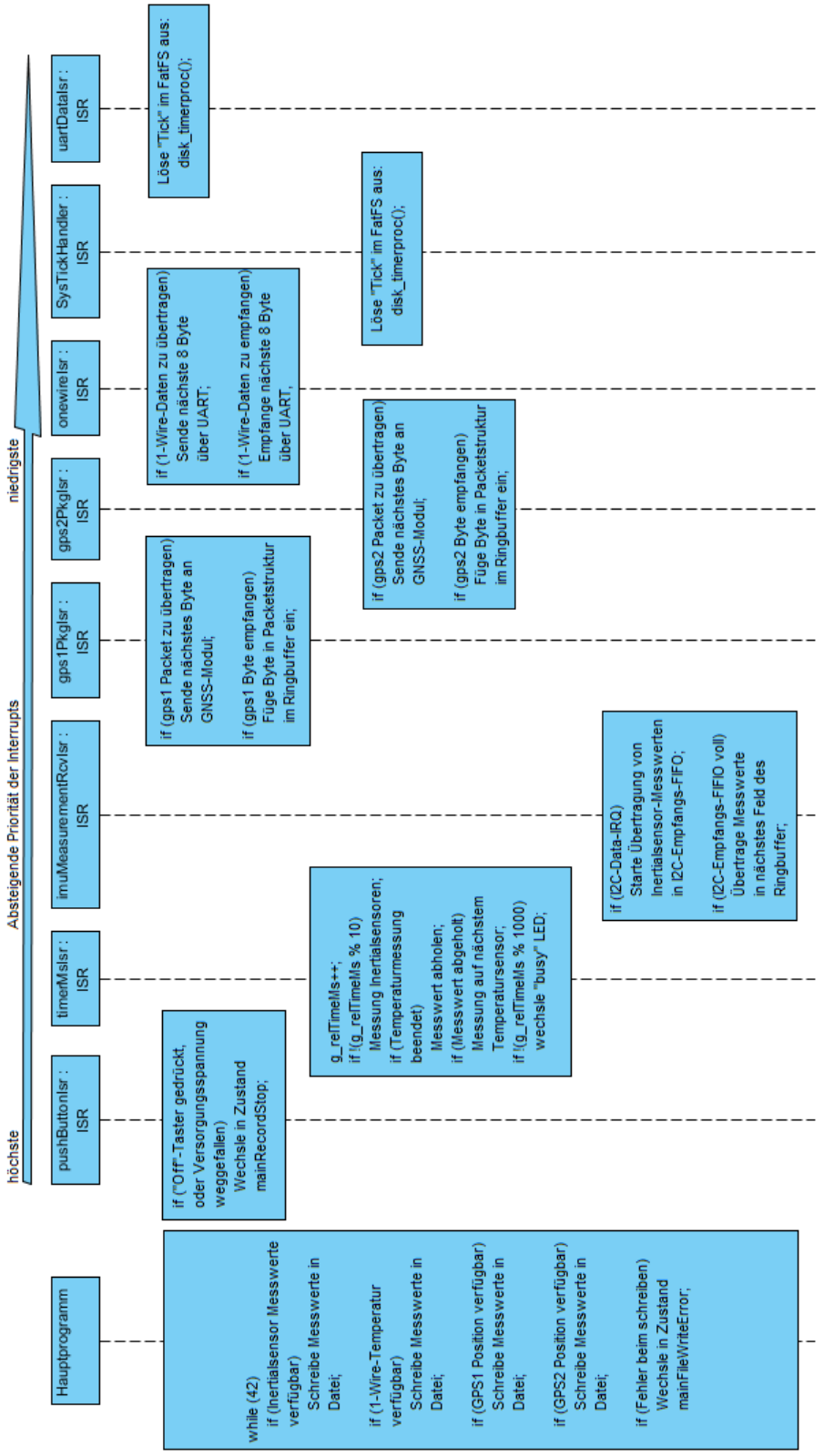


Abbildung 6.7.: Ablauf und Interrupt-Service-Routinen während der Datenaufzeichnung mit Echtzeitanforderungen

Bei kürzeren Messungen stellt es sich als unpraktisch heraus, das Gehäuse des Datenloggers zum Auslesen der Messdaten häufig und in kurzen Abständen öffnen und schließen zu müssen. Um ein einfacheres Zugriffsverfahren anbieten zu können, ist deshalb in der Software des Datenloggers ein sehr einfaches Protokoll implementiert worden, um die Messdaten über die virtuelle RS232-Schnittstelle an den PC übertragen zu können. Dieses Protokoll soll hier kurz erläutert werden.

Aus dem Ruhezustand kann der eingeschaltete Datenlogger durch die Übertragung des Zeichens 't' in den Dateiübertragungs-Modus (Makro „fileTransfer“ in Abb. 6.6) verbracht werden. Mit dem Zeichen 'g' wird nun die Übertragung gestartet, sendet der PC das Zeichen 'd', werden alle Daten auf der Speicherkarte gelöscht und mit dem Zeichen 's' der Dateiübertragungs-Modus beendet.
Wird die Übertragung gestartet, beginnt der Datenlogger die auf der Speicherkarte befindlichen Messungen, nach dem Protokoll in Struktogramm Abb. 6.8, byteweise über die serielle Schnittstelle zu senden.

Für den PC wird eine auf Qt und C++ basierende Anwendung „DataLogger File Extractor“ entwickelt (Anhang D.4), um gemäß des beschriebenen Protokolls die Daten aus dem Datenlogger auszulesen. Die Oberfläche der Anwendung ist in Abb. 6.9 zu sehen. Sie erlaubt den Verbindungsaufbau mit dem Datenlogger sowie das Kopieren der Messdaten in einen anzugebenden Ordner und das Leeren des Datenspeichers.
Die Übertragung erfolgt mit $921\,600\,\frac{\text{bit}}{\text{s}}$, sodass die Messdaten eines Tages in etwa

$$\frac{132.72\,\text{MiByte}}{921\,600\,\frac{\text{bit}}{\text{s}}} \approx \frac{1\,113\,336\,054\,\text{bit}}{921\,600\,\frac{\text{bit}}{\text{s}}} \approx 20.13\,\text{min}$$

auf den PC übertragen werden können.

6.5. Modultests

Damit im Rahmen der Inbetriebnahme neu aufgebauter Hardware-Module die Betriebsfähigkeit einzelner Funktionsgruppen überprüft werden kann und um die Funktionsfähigkeit der einzelnen Software-Module unabhängig voneinander prüfen zu können, werden einfache Modultests implementiert. Diese bestehen aus eigenen Hauptprogrammen mit eingeschränktem Funktionsumfang und verwenden regelmäßig nur die TivaWare-Treiber und das zu überprüfende Software-Modul, wie in Abb. 6.1 dargestellt.
Programmablauf, Aus- und Eingaben unterscheiden sich dabei zwischen den einzelnen Modultests und werden deshalb im Folgenden kurz beschrieben. Der Quellcode zu allen implementierten Modultests ist im Anhang D.3 abgedruckt.

- GNSS-Empfänger (gps.h/c):

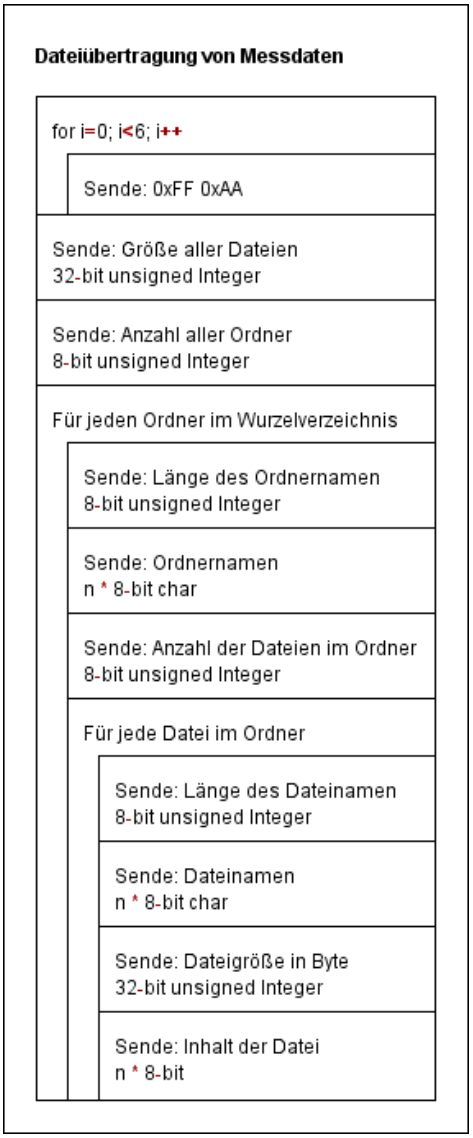


Abbildung 6.8.: Protokoll zur Übertragung der Messdaten über die virtuelle RS232-Schnittstelle

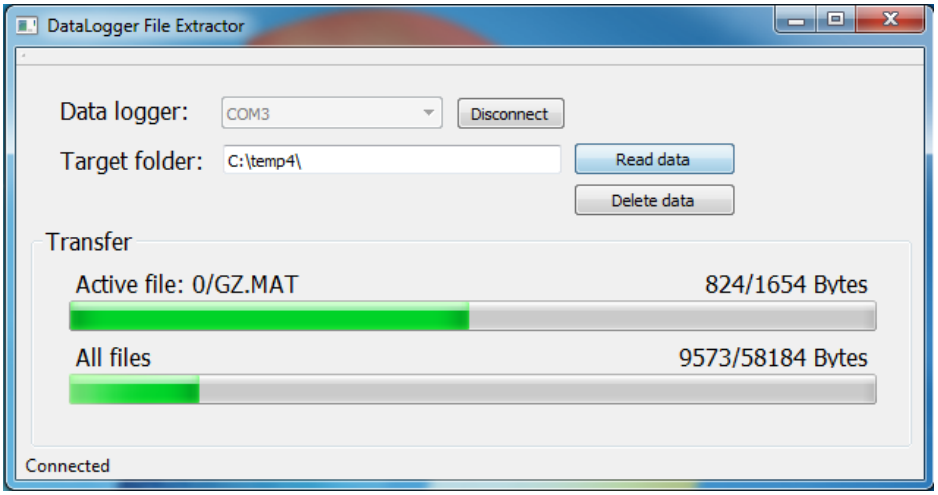


Abbildung 6.9.: PC-Anwendung zum Auslesen der Messdaten

Konfiguriert und Initialisiert den GNSS-Empfänger, sodass dieser alle 100 ms die aktuelle Position bestimmt. Bei Erhalt einer Nachricht des Empfängers wird die Position in Längen- und Breitengrad über die virtuelle RS232-Schnittstelle ausgegeben.

- Inertialmesssystem (imu.h/c):

  Das Inertialmesssystem wird zur periodischen Messung konfiguriert. Die Messwerte werden abgeholt und binär über die virtuelle RS232-Schnittstelle übertragen. Sie können mit der eigens entwickelten PC-Anwendung „MPU9250 Raw Data“ visualisiert werden. (Anhang D.4)

- 1-Wire Temperatursensoren (onewire.h/c):

  Der 1-Wire-Bus wird nach angeschlossenen Sensoren durchsucht, jeder Sensor führt abwechselnd Messungen aus und die Ergebnisse werden leserlich über die virtuelle RS232-Schnittstelle übertragen.

- GSM-/GPRS-Erweiterung (gsm.h/c):

  Das GSM-/GPRS-Modul wird gestartet und initialisiert. Danach steht die serielle Schnittstelle des Moduls getunnelt an der virtuellen RS232-Schnittstelle des Datenloggers zur Verfügung. Es können die AT-Kommandos des Moduls verwendet werden, um die Funktionalität zu prüfen.

- Leuchtdioden (leds.h/c):

  Die nötigen Pins des Mikrocontrollers werden konfiguriert und eine Lauflichtdarstellung auf den Leuchtdioden ausgegeben.

- Taster (buttons.h/c):

  Die nötigen Pins des Mikrocontrollers werden konfiguriert und Tastendrücke über die virtuelle RS232-Schnittstelle angefordert und quittiert.

- Speicherkarte (FatFS / diskio.c):

  Auf der SD-Karte wird eine Testdatei angelegt und mit dem Text „TEST“ beschrieben.

- LCD-Touchscreen (LCD / grlib):

  Das Display und der Touchscreen werden initialisiert und eine einfache Schaltfläche angezeigt. Ein Druck auf die Schaltfläche wird durch Farbwechsel quittiert.

6.6. Kalibrierverfahren zur Einbaulage

Ein besonderes Problem bei der Auswertung von Messungen der Inertialsensoren entsteht, wenn durch den Einbau des Datenloggers eine Abweichung zwischen dem Bezugs-Koordinatensystem des Fahrzeugs und den Koordinaten-Achsen des Inertialmesssystems auftritt. Wenn der Datenlogger im Raum gegen das Fahrzeugkoordinatensystem in Abb. 2.1 verdreht ist, kann eine Längsbeschleunigung des Fahrzeugs nicht mehr einfach von der ihr zugeordneten Achse des Messsystems abgelesen werden. In diesem Fall verteilt sich die Längsbeschleunigung zu unterschiedlichen Anteilen auf alle drei Achsen des Messsystems. Die Vernachlässigung von Beschleunigungen auf der Quer- und Hochachse des Fahrzeugs bei der Aus- und Verwertung der Messdaten ist damit nicht mehr durch das alleinige Ausschließen zweier Messachsen zu bewerkstelligen.

Um nun dennoch den Betrag der Längsbeschleunigung ermitteln zu können, muss Kenntnis über die Lage der Fahrzeuglängsachse im Bezugssystem des Inertialmesssystems erlangt werden. Eine reine Beschleunigung auf der Längsachse lässt sich allerdings nicht messen, da immer zumindest die Erdbeschleunigung zusätzlich wirkt.
Zunächst ist also, durch Messung der Beschleunigung im Stillstand und auf ebener Fläche, der Raumvektor der Erdbeschleunigung $\vec{g}$ zu ermitteln. Das Fahrzeug ist danach in der Ebene entlang der Längsachse zu beschleunigen, sodass die Längs- und Erdbeschleunigung gemeinsam, ohne Beitrag einer Querbeschleunigung, gemessen werden kann. Es entstehen dabei die in Abb. 6.11 a exemplarisch und ohne Berücksichtigung von Größenordnungen dargestellten Beschleunigungsvektoren. Dabei setzt sich der während des Beschleunigungsvorgangs gemessene Beschleunigungsvektor zusammen als

$$\vec{a} = \vec{a}_l + \vec{g}$$

mit der der Längsbeschleunigungskomponente $\overrightarrow{a_l}$.
Da nun, wie in Abb. 6.11 b gezeigt, $\overrightarrow{a}_l$ bestimmt werden kann, kann die Fahrzeuglängsachse im Koordinatensystem des Messsystems nun als Gerade durch den Ursprung in Richtung des Stützvektors $\overrightarrow{a}_l$ beschrieben werden.

Ein gemessener Beschleunigungsvektor während des normalen Betriebs setzt sich zusammen aus der Längs-, Quer- und Hochachsenbeschleunigung sowie der Erdbeschleunigung:

$$\overrightarrow{a}_{mess} = \overrightarrow{a}_{l,mess} + \overrightarrow{a}_{q,mess} + \overrightarrow{a}_{h,mess} + \overrightarrow{g}$$

Wie in Abb. 6.11 c dargestellt, kann nun die Längsachsenkomponente durch Orthogonalprojektion des gemessenen Beschleunigungsvektors auf die Längsachsen-Gerade berechnet werden:

$$a_{l,mess} = \left\| \frac{\overrightarrow{a}_{mess} \cdot \overrightarrow{a}_l}{\overrightarrow{a}_l \cdot \overrightarrow{a}_l} \overrightarrow{a}_l \right\| \cdot \cos\left(\measuredangle(\overrightarrow{a}_l, \overrightarrow{a}_{mess})\right)$$

Die Software des Datenloggers bietet ein experimentelles Verfahren an, um die Vektoren $\overrightarrow{g}$ und $\overrightarrow{a}_l$ zu ermitteln. Abgelegt werden die Vektoren in „OFFSET/ACC.MAT“ und „ROLL/ACC.MAT“.
Eine Korrektur der Messwerte anhand der Kalibrierwerte muss anschließend offline in der Nachbearbeitung der Daten erfolgen.
Da die Implementierung erst spät in der Entwicklung des Datenloggers durchgeführt wurde, bleibt das Kalibrierverfahren ungetestet. Sowohl das theoretische Verfahren, als auch die Implementierung, sind deshalb lediglich ein Vorschlag zum Lösungsansatz für das vorliegende Problem.

Im Struktogramm Abb. 6.10 ist die Implementierung des Verfahrens dargestellt. Der Programmablauf setzt dabei voraus, dass das Fahrzeug zunächst auf ebener Fläche stillsteht und ermittelt den Erdbeschleunigungsvektor als Mittelwert von 128 Messungen. Darauf wartet das Programm auf eine Beschleunigung ohne Querbeschleunigungskomponente. Dazu wird überprüft, ob die gemessene Drehrate während der Messung einen gegebenen Schwellwert nicht überschreitet. Außerdem muss zur Messung zumindest an einer Achse die Beschleunigungsmessung für alle Messzeitpunkte einen Schwellwert überschreiten, d.h. das Fahrzeug also tatsächlich beschleunigen. Der Mittelwert aus 128 Messungen und der Erdbeschleunigungsvektor werden abgespeichert und mit den Messdaten ausgegeben.

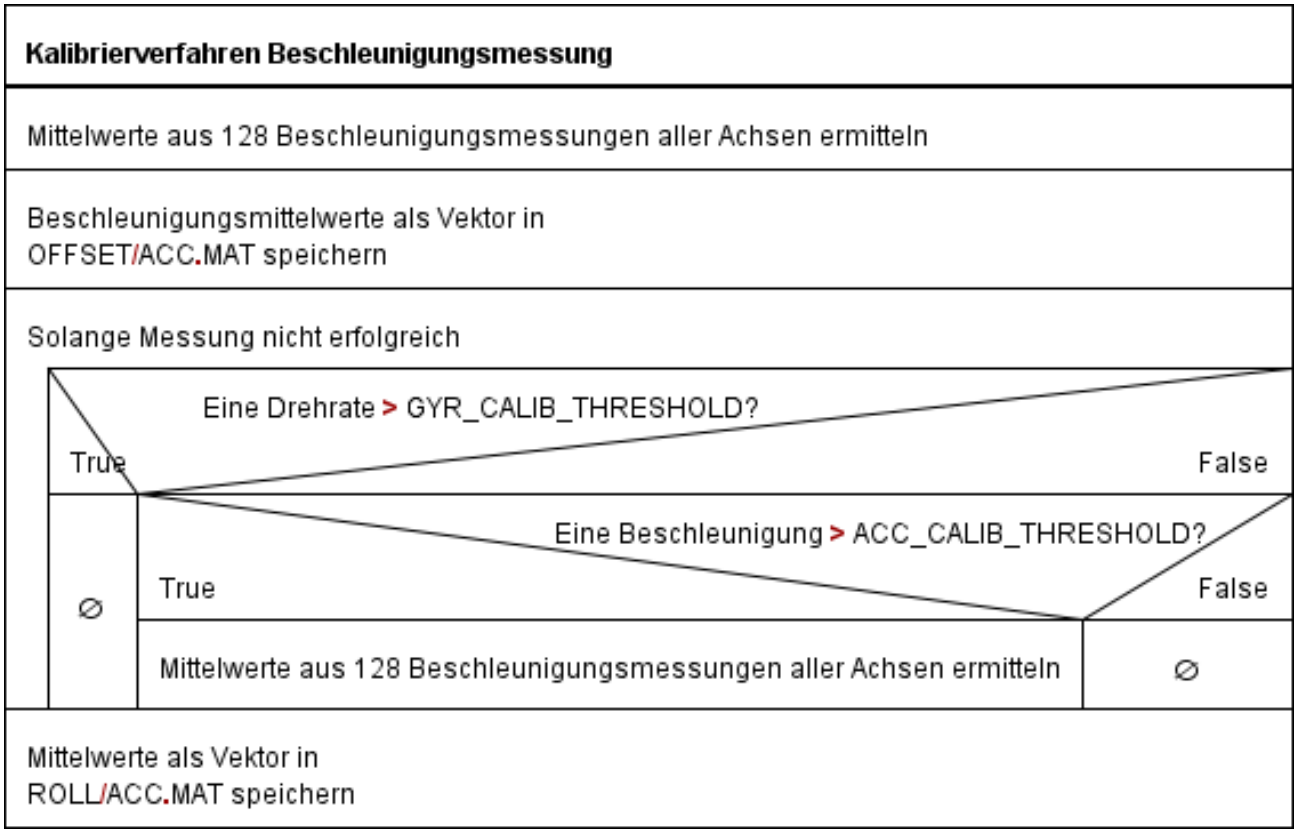


Abbildung 6.10.: Struktogramm zum Programmablauf des Kalibrierverfahrens zur Einbaulage

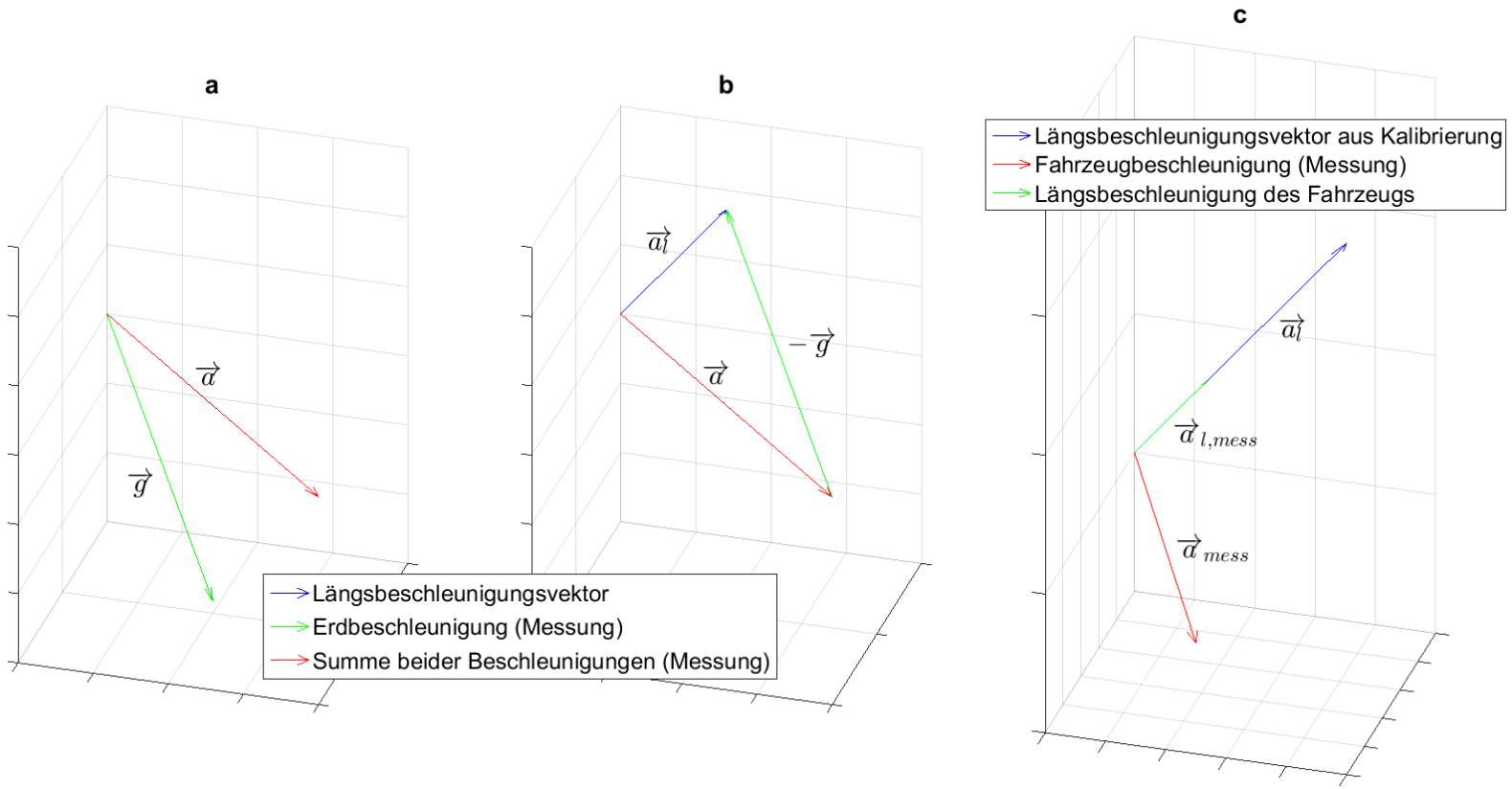


Abbildung 6.11.: Kalibrierverfahren zur Bestimmung der Einbaulage: Gemessene und berechnete Vektoren im $\mathbb{R}^3$, a: Gemessene Fahrzeug- und Erdbeschleunigung, b: Ermittlung des Längsbeschleunigungsvektors, c: Ermittlung der Längsbeschleunigung durch Orthogonalprojektion auf die Fahrzeuglängsachse

7. Erprobung und Erfassung von Messdaten

7.1. Messplan

Um die Funktionsfähigkeit des Datenloggers zu überprüfen, die tatsächliche Qualität der Messergebnisse zu bewerten und die in der Theorie gemachten Abschätzungen zu verifizieren erfolgen verschiedene Messungen mit dem Datenlogger. Messungen, die vorgegebene Fahrsituationen untersuchen sollen, werden dabei in einem PKW durchgeführt, da im Rahmen dieser Arbeit kein Nahverkehrsbus für derartige Messungen zur Verfügung steht.

Im Einzelnen erfolgen die in Tab. 7.1 aufgelisteten Messungen in einem PKW und die in Tab. 7.2 in einem Nahverkehrsbus im normalen Linienbetrieb. Dabei wird jeweils der Datenlogger möglichst achsengerecht ausgerichtet, da eine nachträgliche Korrektur der Messwerte vorerst nicht möglich ist. Im Verlauf dieser Arbeit konnten Verifizierung und Anpassung des Kalibrierverfahrens aus zeitlichen Gründen nicht mehr durchgeführt werden.
Außerdem wird, wie in Tab. 7.3 gelistet, ein Messversuch mit den Temperatursensoren durchgeführt.

Bei den Messungen im PKW soll die prinzipielle Betriebsfähigkeit des Datenloggers geprüft werden. Außerdem sollen erste Bewertungen über die Qualität der Messung erfolgen. Um dabei die Annahmen aus 2.1.2 in der Praxis bewerten zu können, werden Fahrsituationen gewählt, die den in der Theorie betrachteten stationären Fahrzuständen möglichst nahe kommen.

Die Messungen im Nahverkehrsbus sollen erste repräsentative Ergebnisse im beabsichtigten Messeinsatz des Datenloggers liefern. Anhand der Messungen soll u.a. eine Bewertung der nötigen Betriebsbedingungen der GNSS-Empfänger im Bus sowie der erreichbaren Genauigkeit möglich sein.

Die Temperaturmessung erfolgt mit zwei Sensoren, von denen einer die Raumtemperatur des BATSEN-Projektlabors und der andere die Temperatur am Vorlauf des Heizkörpers misst. Dabei sollen unterschiedliche Temperaturen und vor allem Temperaturänderungen beobachtet und so die Funktionsfähigkeit der Sensoren und der Kommunikation bestätigt werden.

Messung-Nr.	Bezeichnung	Randbedingungen
1	Geradeausfahrt mit konstanter Geschwindigkeit	• Möglichst achsengerechte Ausrichtung des Datenloggers • GNSS-Empfänger außerhalb am Fahrzeug • Fahrt mit Tempomat
2	Geradeausfahrt veränderlicher Geschwindigkeit	• Möglichst achsengerechte Ausrichtung des Datenloggers • GNSS-Empfänger außerhalb am Fahrzeug • Geschwindigkeitsänderung durch Fahrer
3	Kreisfahrt mit konstanter Geschwindigkeit	• Möglichst achsengerechte Ausrichtung des Datenloggers • GNSS-Empfänger außerhalb am Fahrzeug • Fahrt mit Tempomat
4	Kreisfahrt mit veränderlicher Geschwindigkeit	• Möglichst achsengerechte Ausrichtung des Datenloggers • GNSS-Empfänger außerhalb am Fahrzeug • Geschwindigkeitsänderung durch Fahrer

Tabelle 7.1.: Messplan zu Messungen im PKW

Messung-Nr.	Bezeichnung	Randbedingungen
5	Innovationslinie 109 (Abschnitt 1.1)	• Möglichst achsengerechte Ausrichtung des Datenloggers • Starthaltestelle: Hauptbahnhof/ZOB • Endhaltestelle: U-Alsterdorf • GNSS-Empfänger innen am Fenster
6	Innovationslinie 109	• Möglichst achsengerechte Ausrichtung des Datenloggers • Starthaltestelle: U-Alsterdorf • Endhaltestelle: Hauptbahnhof/ZOB • GNSS-Empfänger innen am Fenster

Tabelle 7.2.: Messplan zu Messungen im Nahverkehrsbus

Messung-Nr.	Bezeichnung	Randbedingungen
7	Temperaturmessung	• Gleichzeitige Messung von zwei Temperaturen • Messung von Raumtemperatur • Messung der Temperatur des Heizungsvorlaufs

Tabelle 7.3.: Messplan zu Messungen mit Temperatursensoren

7.2. Messungen im PKW

7.2.1. Durchführung

Für die Durchführung der Messungen im PKW wird der Datenlogger im Kofferraum des Fahrzeugs auf einer rutschfesten Gummiunterlage fixiert und so ausgerichtet, dass die X-Achse des Inertialmesssystems möglichst genau der Längsachse des Fahrzeugs entspricht (Abb. C.1). Die Leitungen zu den GNSS-Empfängern werden jeweils links und rechts in Höhe des Fahrzeugdachs durch die Dichtung der Heckklappe geführt, sodass die Empfänger mit der Antenne nach oben gerichtet an der Fahrzeugseite fixiert sind (Abb. C.2).

Da sich eine Verbindung der jeweiligen Messungen mit und ohne Geschwindigkeitsänderung anbietet, werden die vier Fahrzustände nur in zwei getrennten Messungen erfasst. Bei der Auswertung können dann die entsprechenden Zeitabschnitte aus der Messung im Detail betrachtet und ausgewertet werden. Es wird also eine zusammenhängende Messung für die Geradeausfahrt mit konstanter und veränderlicher Geschwindigkeit sowie eine weitere für ebendiese Vorgänge in der Kreisfahrt durchgeführt.

Für die Geradeausfahrt wird dazu auf etwa $40\,\frac{\mathrm{km}}{\mathrm{h}}$ beschleunigt und darauf wieder abgebremst. Die Fahrt wird dann auf einer geraden Straße mit einer Geschwindigkeit von $50\,\frac{\mathrm{km}}{\mathrm{h}}$ unter Verwendung der Geschwindigkeitsregelung des Fahrzeugs fortgesetzt. Damit sind die relevanten Fahrzustände in der Geradeausfahrt in der Messung enthalten.

Die Kreisfahrt wird in einem Kreisverkehr durchgeführt. Das Fahrzeug wird in den Kreisverkehr eingefahren und mittels der Geschwindigkeitsregelung des Fahrzeugs bei $30\,\frac{\mathrm{km}}{\mathrm{h}}$ und möglichst konstantem Lenkwinkel durch den Kreisverkehr bewegt. Das Fahrzeug wird dann abgebremst, wieder beschleunigt und aus dem Kreisverkehr ausgefahren.

Bei allen Messungen wurde das Fahrzeug zu Beginn auf einer möglichst ebenen Fläche abgestellt. Diese wird bei der Auswertung der Messungen als „Nullbezug“ zur Offsetkorrektur verwendet.

7.2.2. Geradeausfahrt mit konstanter Geschwindigkeit

Die Geradeausfahrt erfolgt entlang der in Abb. C.3 dargestellten Route. Für die Darstellung wurden die bei der Aufzeichnung erhobenen Positionsinformationen beider GNSS-Empfänger mit Hilfe von „Google Earth“ visualisiert. An dem Ergebnis ist zunächst zu erkennen, dass die Positionsinformationen der GNSS-Empfänger nah beieinander liegen und die Positionsgenauigkeit aufgrund des guten Empfangs sogar die Identifikation der Fahrspur

zulässt. Mittels der Messfunktionen von „Google Earth“ kann die größte Abweichung der Positionen beider GNSS-Empfänger in dieser Messung, wie in Abb. C.4 dargestellt zu etwa

$$s_{\Delta GNSS} \approx 2.4\,\mathrm{m}$$

bestimmt werden. Sie liegt damit in der Größenordnung der Positionsgenauigkeit laut Hersteller.

Zu der befahrenen Strecke ergeben sich Geschwindigkeit laut GNSS-Positionen und Beschleunigung nach Beschleunigungssensor auf der Längsachse wie in Abb. C.5. Zunächst kann daran vor allem ein deutliches Rauschen in der Beschleunigungsmessung, insbesondere während der Fahrt, festgestellt werden. Hier äußern sich Schwingungen aus Unebenheits- und Motoranregung.

Um die Arbeit mit den Messwerten einfacher zu gestalten, soll versucht werden, die unerwünschten Schwingungen aus dem Signal zu filtern. Dazu wird eine über die bereits in Abschnitt 2.1.2 erfolgte theoretische Betrachtung hinausgehende Analyse des Signals durchgeführt. Mittels DFT werden die Amplitudenspektren in Abb. C.6 zum einen für die gesamte Messung, zum anderen nur für den Fahrtteil mit konstanter Geschwindigkeit berechnet. Aus der Detailbetrachtung in Abb. C.7 geht hervor, dass die durch den Fahrer eingebrachten Anteile sich im Wesentlichen unter einer Frequenz von $1\,\mathrm{Hz}$ bemerkbar machen. Diese Beobachtung entspricht den Erwartungen aus der Theorie. In höheren Frequenzen zeigt sich zwischen den Amplitudenspektren kaum eine Differenz, sodass angenommen wird, dass hier lediglich fahrerunabhängige Vorgänge auftreten.

Im Weiteren wird das Beschleunigungs-Signal deshalb vor der Abbildung wie folgt vorverarbeitet:

- Digitales Tiefpassfilter
- $f_g = 10\,\mathrm{Hz}$
- Verwendung eines sog. Zero-Phase-Filter

Ein Zero-Phase-Filter wird verwendet um eine Verzögerung der dargestellten Beschleunigung zu anderen Messwerten zu vermeiden. Dabei wird das nicht kausale Verhalten eines solchen Filters akzeptiert, muss aber bei der Auswertung u.U. berücksichtigt werden.

In Abb. C.8 findet sich die gemessene Längsbeschleunigung nach der Anwendung des Filters. An der in Abb. C.9 ist zudem zu erkennen, dass die Messung des Drehratensensors für die Gierachse während der Geradeausfahrt zwischen etwa $108\,\mathrm{s}$ bis $138\,\mathrm{s}$ nahezu bei $0\,\frac{\mathrm{deg}}{\mathrm{s}}$ liegt. Die ablesbaren kleineren Ausschläge sind vermutlich vor allem durch kleine Lenkkorrekturen zu begründen, welche sich zur sicheren Fahrzeugführung nicht vermeiden lassen. Eine Detailbetrachtung des Zeitbereichs in Abb. 7.1 zeigt, dass sowohl Geschwindigkeit nach GNSS als auch die gemessene Längsbeschleunigung nicht konstant sind. Zum einen ist hier

sicherlich das Regelverhalten der automatischen Geschwindigkeitsregelung zu beobachten, da aber z.B. zwischen 120 s bis 125 s ein leichter Anstieg der Geschwindigkeit mit einer tendenziell negativen Beschleunigung gemessen wird, müssen weitere Effekte eine Rolle spielen. Mit hoher Wahrscheinlichkeit liegt eine leichte Abwärtsneigung der Fahrbahn vor, sodass wie in Abschnitt 2.1.2 beschrieben, ein Anteil der Erdbeschleunigung in die Messung der Längsbeschleunigung eingeht.

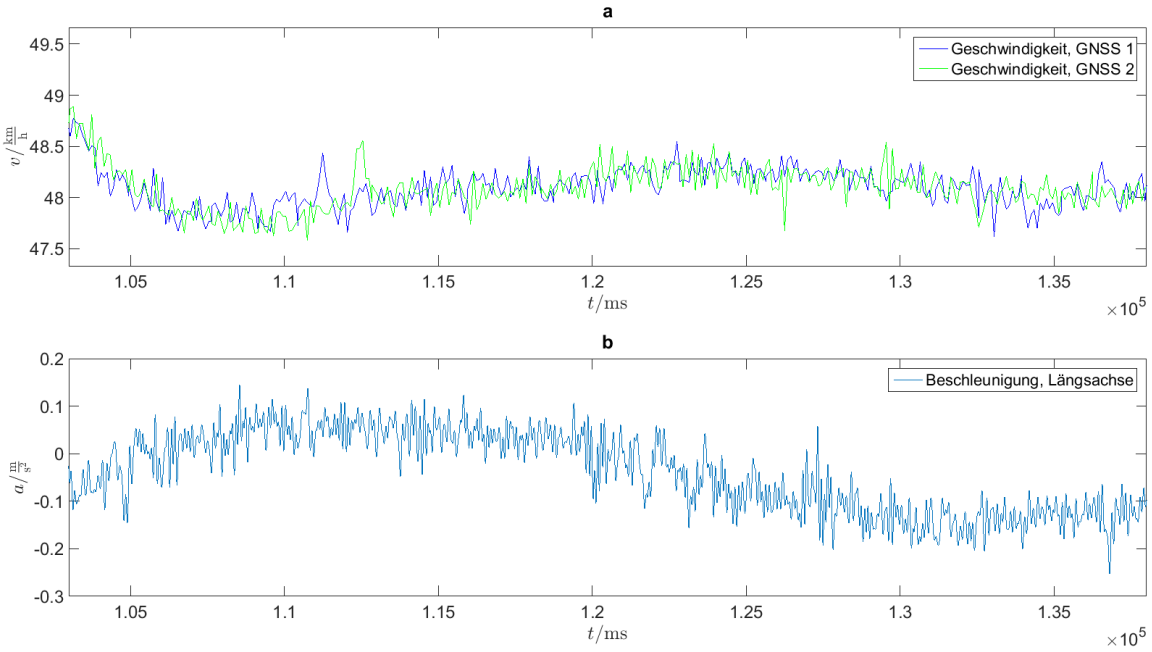


Abbildung 7.1.: Messung-Nr. 1: Geschwindigkeit und Beschleunigung ($f_{g,TP} = 10\,\mathrm{Hz}$) bei Geradeausfahrt mit konstanter Geschwindigkeit, **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem, digital tiefpassgefiltert

Zusammenfassend ist nach der Messung zur Geradeausfahrt mit konstanter Geschwindigkeit festzustellen:

- Die Fahrereinflüsse auf die Fahrzeuglängsregelung liegen in der normalen Fahrt tatsächlich unterhalb von 1 Hz
- Selbst geringe Steigungen wirken sich sichtbar auf die gemessene Beschleunigung aus, auch wenn keine Geschwindigkeitsänderung stattfindet

7.2.3. Geradeausfahrt mit veränderlicher Geschwindigkeit

Auch die Beobachtungen zur Geradeausfahrt mit veränderlicher Geschwindigkeit finden anhand der bereits in Abschnitt 7.2.2 beschriebene Messfahrt statt. Dazu wird die in Abb. C.8 dargestellte Beschleunigung und Geschwindigkeit nun im Zeitbereich von etwa 15 s bis 36 s betrachtet. Die Abb. C.9 zeigt für diesen Zeitabschnitt der Messung nur geringe Drehraten um die Gierachse, welche durch nötige Lenkkorrekturen entstehen. Es liegt also eine Beschleunigung und Abbremsung bei nahezu gerader Fahrt vor.

Die Detailansicht in Abb. 7.2 zeigt, dass sich die Beschleunigung wie erwartet nach der Änderung der mittels GNSS bestimmten Geschwindigkeit verhält. Während die Geschwindigkeit zunimmt, wird eine positive Beschleunigung gemessen, beim Bremsen eine negative.

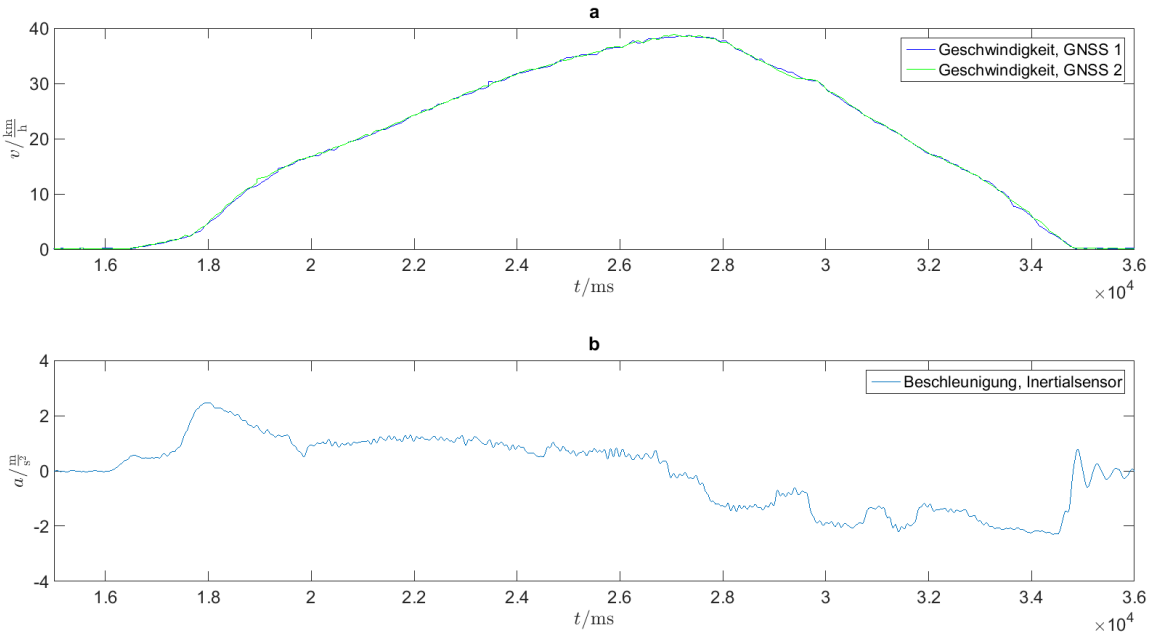


Abbildung 7.2.: Messung-Nr. 2: Geschwindigkeit und Beschleunigung ($f_{g,TP} = 10\,\mathrm{Hz}$) bei Geradeausfahrt mit veränderlicher Geschwindigkeit, **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem, digital tiefpassgefiltert

Um das Messergebnis zu überprüfen ist in Abb. 7.3 zusätzlich die Beschleunigung aus der durch einen der GNSS-Empfänger bestimmten Geschwindigkeit aufgetragen. Dazu wurde wie in Abschnitt 2.1.3 theoretisch beschrieben, die Beschleunigung durch numerische Differentiation bestimmt.
Aus der Abbildung ist zu entnehmen, dass die gemessene Beschleunigung in ihrem zeitlichen Verlauf und der Größenordnung gut zu der durch das GNSS-Modul bestimmten Ge-

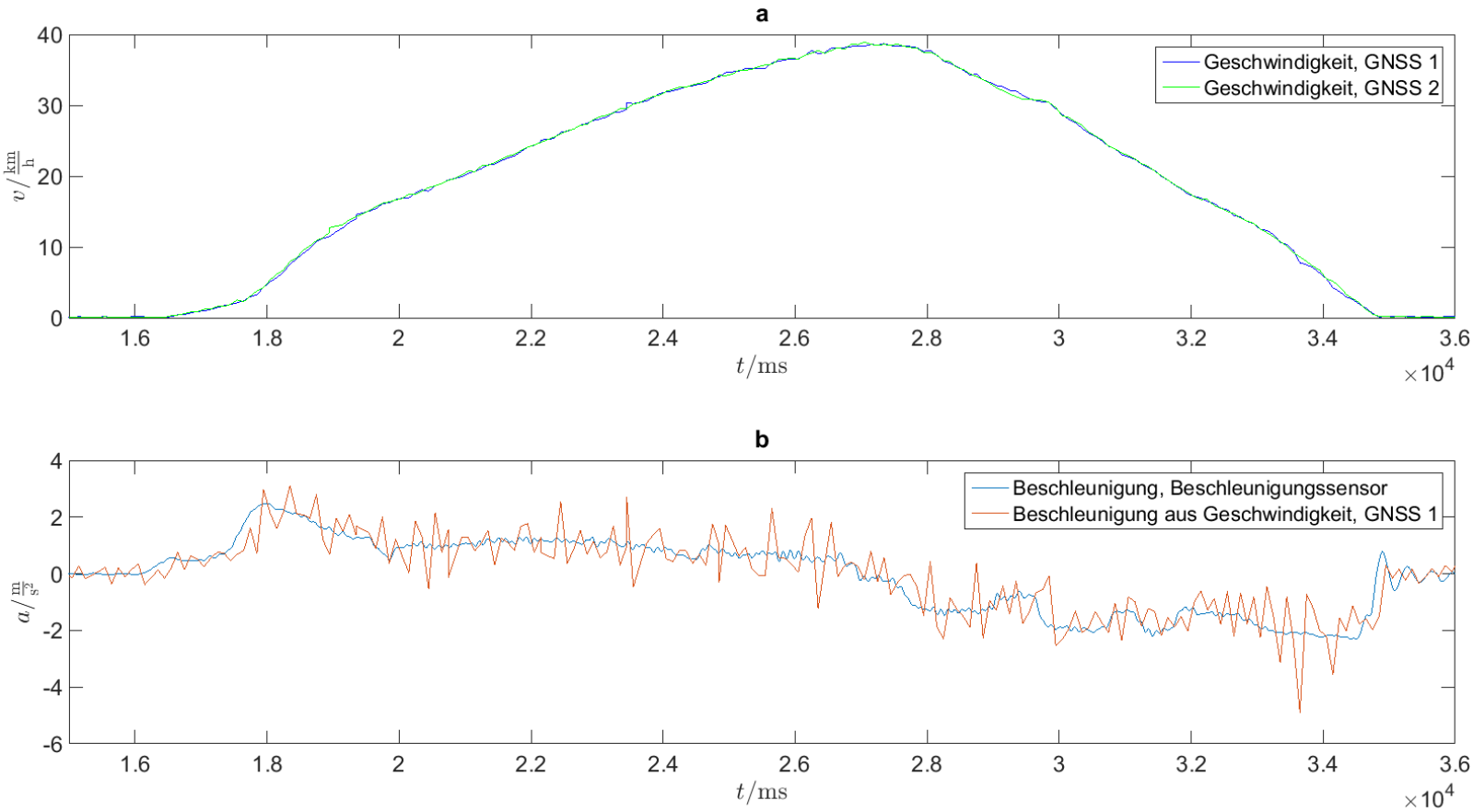


Abbildung 7.3.: Messung-Nr. 2: Geschwindigkeit und Beschleunigung ($f_{g,TP} = 10\,\mathrm{Hz}$) bei Geradeausfahrt mit veränderlicher Geschwindigkeit, **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem, digital tiefpassgefiltert und Beschleunigung als numerisches Differential der Geschwindigkeit von GNSS 1

schwindigkeitsänderung passt. Außerdem lässt sich feststellen, dass die aus der Geschwindigkeit abgeleitete Beschleunigung deutlich um die gemessene rauscht, und dabei Spitzen erreicht, die den Betrag der gemessenen Beschleunigung um mehr als das doppelte übersteigen. Dieses Verhalten ist auf Sprünge in der Positionsbestimmung, wie in Abschnitt 2.1.3 beschrieben, zurückzuführen. Diese machen sich schon in der Geschwindigkeit, welche die erste Ableitung der Strecke darstellt, als steile Sprünge bemerkbar.
Abb. 7.4 zeigt die aus der, mittels GNSS-Empfänger 1 bestimmten, Geschwindigkeit abgeleitete Beschleunigung aufgetragen über die mit dem Beschleunigungssensor gemessene. Dabei wurde die abgeleitete Geschwindigkeit zudem durch ein Tiefpassfilter geglättet, sodass aus dem Verfahren entstehende Spitzen weniger eingehen. Die Punktwolke bildet im wesentlichen den erwarteten Zusammenhang ab, es wird aber auch deutlich, dass die aus den unterschiedlichen Verfahren ermittelten Beschleunigungen nicht identisch sind.

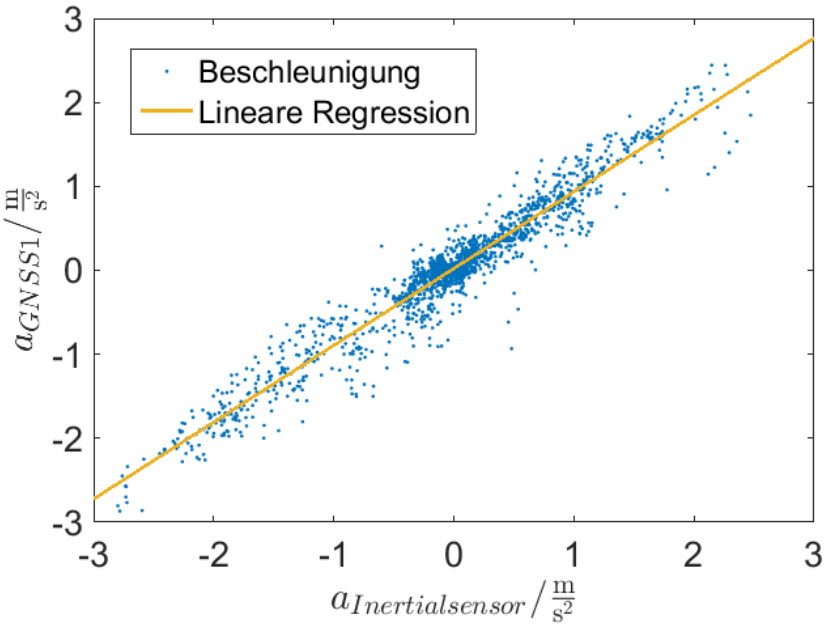


Abbildung 7.4.: Messung-Nr. 1 und 2: Beschleunigung ($f_{g,TP} = 1\,\mathrm{Hz}$) nach GNSS-Empfänger 1 über der Beschleunigung nach Beschleunigungssensor ($f_{g,TP} = 10\,\mathrm{Hz}$)

Steigungen sind anhand einer reinen Betrachtung der Längsbeschleunigung in dieser Fahrsituation nicht mehr herauszurechnen.

Zusammenfassend ist nach der Messung zur Geradeausfahrt mit veränderlicher Geschwindigkeit festzustellen:

- Der Datenlogger misst die Längsbeschleunigung in Geradeausfahrt korrekt

- Die numerische Differentiation der Geschwindigkeit aus den GNSS-Empfängern führt für einzelne Abtastzeitpunkte zu teils großen Abweichungen

7.2.4. Kreisfahrt mit konstanter Geschwindigkeit

Die Kreisfahrt erfolgt entlang der in Abb. C.10 dargestellten Route. Für die Darstellung wurden die bei der Aufzeichnung erhobenen Positionsinformationen eines GNSS-Empfängers mit Hilfe von „Google Earth“ visualisiert. Der Kreisradius ergibt sich durch Messung mittels „Google Earth“ in Abb. C.11 zu etwa

$$\rho \approx 19.5\,\mathrm{m}$$

In den Abb. C.12 und C.13 sind die Geschwindigkeiten nach Satellitenortung, die auf der Fahrzeuglängsachse gemessene Beschleunigung sowie die Drehrate um die Gierachse dargestellt. Dabei lässt sich anhand der Drehrate sehr gut die Fahrt durch den Kreisverkehr erkennen: Bei etwa $104.5\,\mathrm{s}$ bis $110\,\mathrm{s}$ der Messung wird mit einer Rechtskurve in den Kreisverkehr eingefahren. Danach wird das Fahrzeug in den Linkskreis gefahren. Zwischen etwa $126\,\mathrm{s}$ und $138\,\mathrm{s}$ kann aus der Geschwindigkeit nach GNSS eine nahezu konstante Geschwindigkeit abgelesen werden, welche durch die Geschwindigkeitsregelung des Fahrzeugs auf etwa $30\,\frac{\mathrm{km}}{\mathrm{h}}$ gehalten wird.
Eine Detaildarstellung des Zeitbereichs enthalten die Abb. 7.5 und 7.6.
Es fällt auf, dass die in den GNSS-Empfängern ermittelten Geschwindigkeiten voneinander um etwa $1.5\,\frac{\mathrm{km}}{\mathrm{h}}$ abweichen. Die Abweichung in dieser Größenordnung lässt sich mit den unterschiedlichen Bahngeschwindigkeiten des inneren und äußeren Empfängers erklären. Sie bedeutet eine Differenz im Radius von etwa $1\,\mathrm{m}$, welche unterhalb der absoluten Genauigkeit der Empfänger liegt.

Im Zeitraum der nahezu konstanten Geschwindigkeit in der Kreisfahrt, liegt die gemessene Drehrate im Mittel bei

$$\omega \approx 24.5\,\frac{\mathrm{deg}}{\mathrm{s}}$$

Für die Dauer einer Kreisfahrt ergibt sich damit

$$t_{Kreis} = \frac{360\,\mathrm{deg}}{24.5\,\frac{\mathrm{deg}}{\mathrm{s}}} \approx 14.69\,\mathrm{s}$$

bei einer Kreisbahn von

$$s_{Kreis} = 2\pi\rho = 2\pi 19.5\,\mathrm{m} \approx 122.52\,\mathrm{m}$$

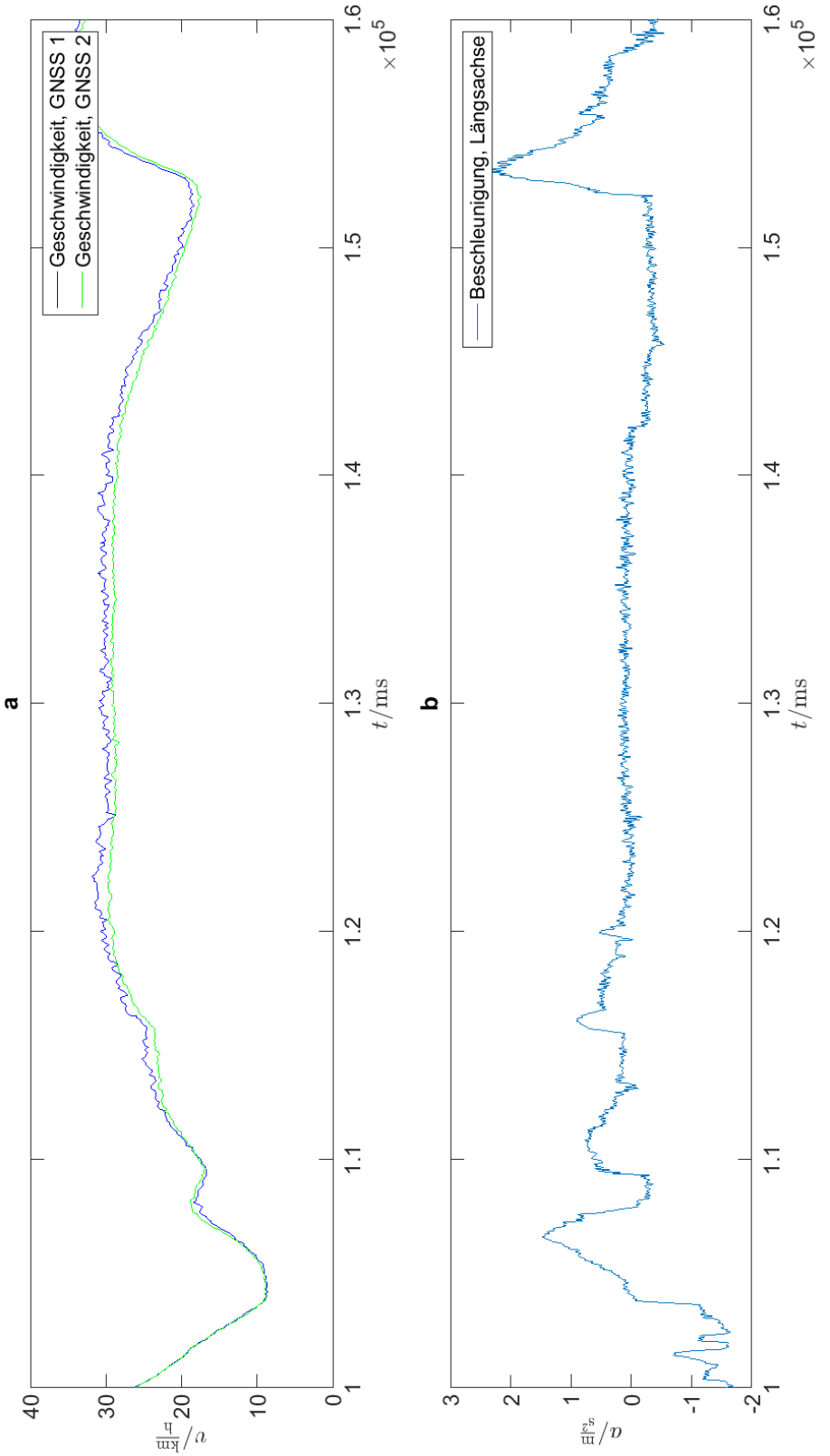


Abbildung 7.5.: Messung-Nr. 3: Geschwindigkeit und Beschleunigung ($f_{g,TP} = 10\,\mathrm{Hz}$) bei Kreisfahrt mit konstanter Geschwindigkeit, **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem, digital tiefpassgefiltert

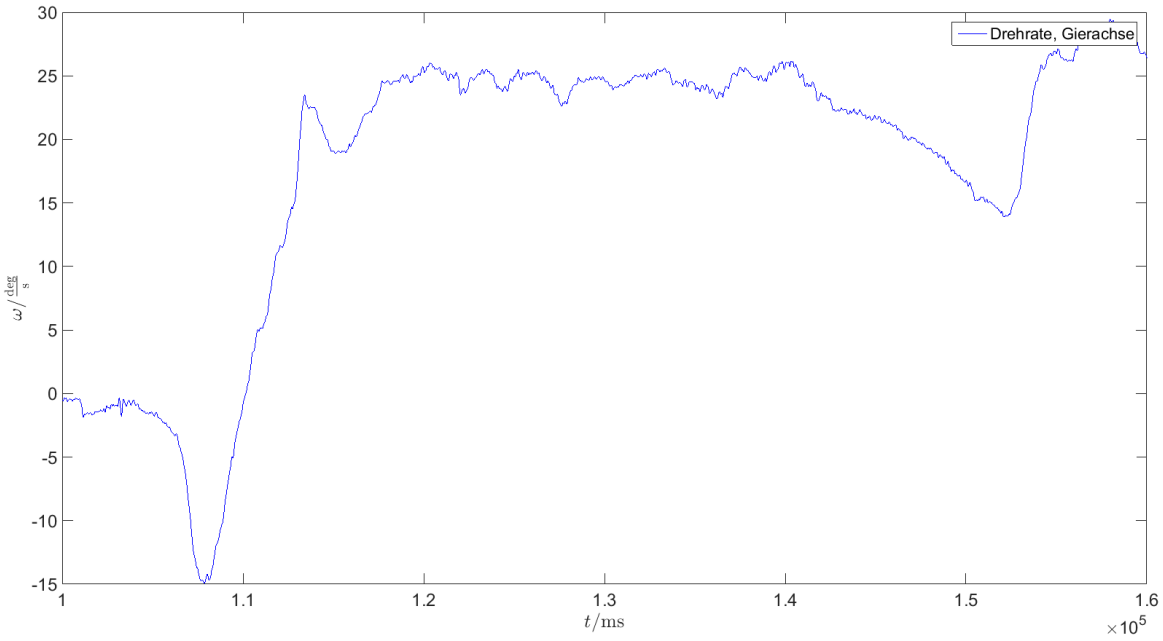


Abbildung 7.6.: Messung-Nr. 3: Drehrate um die Gierachse bei Kreisfahrt mit konstanter Geschwindigkeit

Die daraus ermittelbare Bahngeschwindigkeit

$$v_{Kreis} = \frac{122.52\,\mathrm{m}}{14.69\,\mathrm{s}} \approx 8.34\,\frac{\mathrm{m}}{\mathrm{s}} = 30.02\,\frac{\mathrm{km}}{\mathrm{h}}$$

entspricht nahezu der mittels GNSS bestimmten und am Tempomat eingestellten Geschwindigkeit.

Auch die gemessene Beschleunigung auf der Längsachse verhält sich in diesem Zeitraum nahezu konstant. Sie ergibt sich im Mittel zu

$$a_{l,Kreis} \approx 0.1\,\frac{\mathrm{m}}{\mathrm{s}^2}$$

bei einer Zentripetalbeschleunigung von

$$a_q = \frac{v^2}{\rho} = \frac{30\,\frac{\mathrm{km}}{\mathrm{h}}}{19.5\,\mathrm{m}} \approx 3.56\,\frac{\mathrm{m}}{\mathrm{s}^2}$$

und damit einem Schwimmwinkel von

$$\beta = \arcsin\left(\frac{a_{l,Kreis}}{a_q}\right) \approx 1.61\,\mathrm{deg}$$

Damit liegen die Werte in etwa in der aus Abschnitt 2.1.2 theoretisch erwarteten Größenordnung. Dabei ist zu berücksichtigen, dass eine Ungenauigkeit im achsengerechten Einbau in dieser Messung nicht vom Schwimmwinkel unterschieden werden kann. Beide Winkel addieren sich.

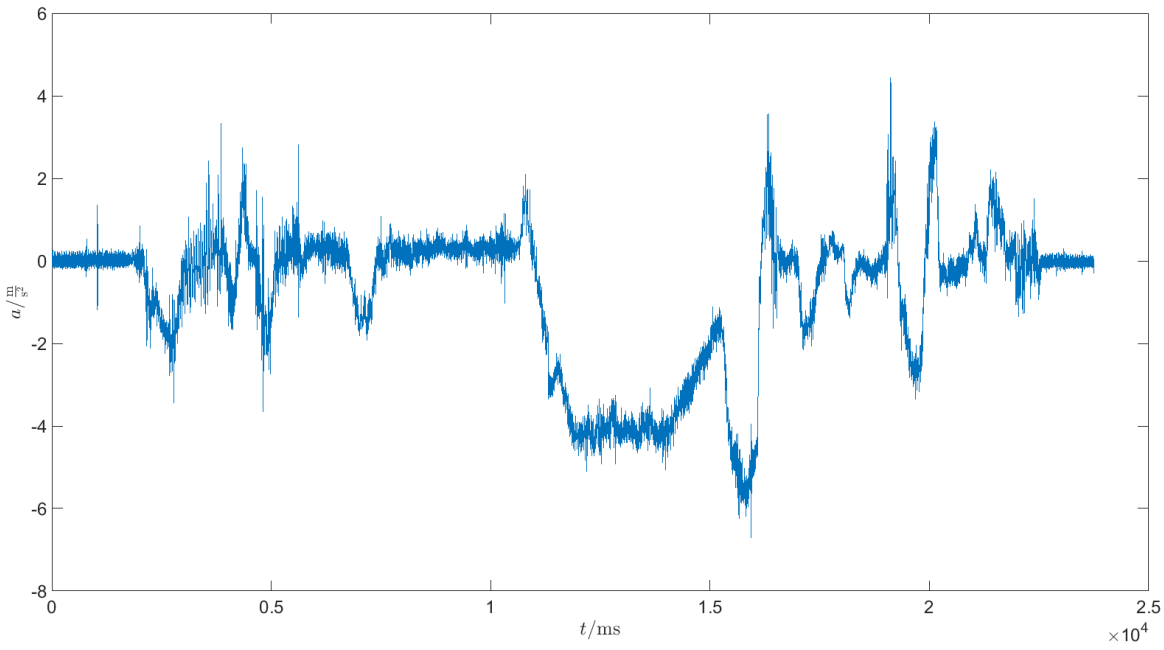


Abbildung 7.7.: Messung-Nr. 3: Gemessene Querbeschleunigung bei Kreisfahrt mit konstanter Geschwindigkeit, Inertialmesssystem

In Abb. 7.7 ist die gemessene Querbeschleunigung ungefiltert dargestellt. Für den gewählten Zeitbereich ergibt sich eine mittlere Querbeschleunigung von:

$$a_{q,mess} \approx -4.06\,\frac{\mathrm{m}}{\mathrm{s}^2}$$

Die gemessene Querbeschleunigung ist damit betragsmäßig größer derer, die sich rechnerisch aus den Parametern der Kreisfahrt ermitteln lässt. Einer der Gründe kann das Wanken des Fahrzeugs, also eine Drehung um die Längsachse sein, sodass auch hier ein Anteil der Erdbeschleunigung mitgemessen wird.

Zusammenfassend ist nach der Messung zur Kreisfahrt mit konstanter Geschwindigkeit festzustellen:

- Die Drehrate um die Gierachse wird korrekt gemessen
- Messung des Schwimmwinkels und Eintrag der Zentripetalbeschleunigung in die Längsbeschleunigung passen zu den theoretischen Erwartungen

- Der Eintrag der Zentripetalbeschleunigung liegt auch in der Praxis in einer relevanten Größenordnung. Da er in normaler Fahrt nur sehr kurzfristig während der Kurvenfahrt auftritt, kann er in den weiteren Betrachtungen vorerst vernachlässigt werden.

7.2.5. Kreisfahrt mit veränderlicher Geschwindigkeit

Die Auswertung einer Kreisfahrt mit veränderlicher Geschwindigkeit findet anhand der bereits in Abschnitt 7.2.4 behandelten Messfahrt statt. Aus den Messungen in den Abb. C.12 und C.13 wird nun der Zeitraum zwischen $138\,\mathrm{s}$ und $165\,\mathrm{s}$ ausgewertet.
In diesem Zeitraum wird das Fahrzeug zunächst während der Kreisfahrt abgebremst auf eine Geschwindigkeit von etwa $18\,\frac{\mathrm{km}}{\mathrm{h}}$ und dann wieder auf etwa $35\,\frac{\mathrm{km}}{\mathrm{h}}$ beschleunigt. Die Detaildarstellungen des Zeitabschnitts sind in Abb. 7.8 und 7.9 zu finden.

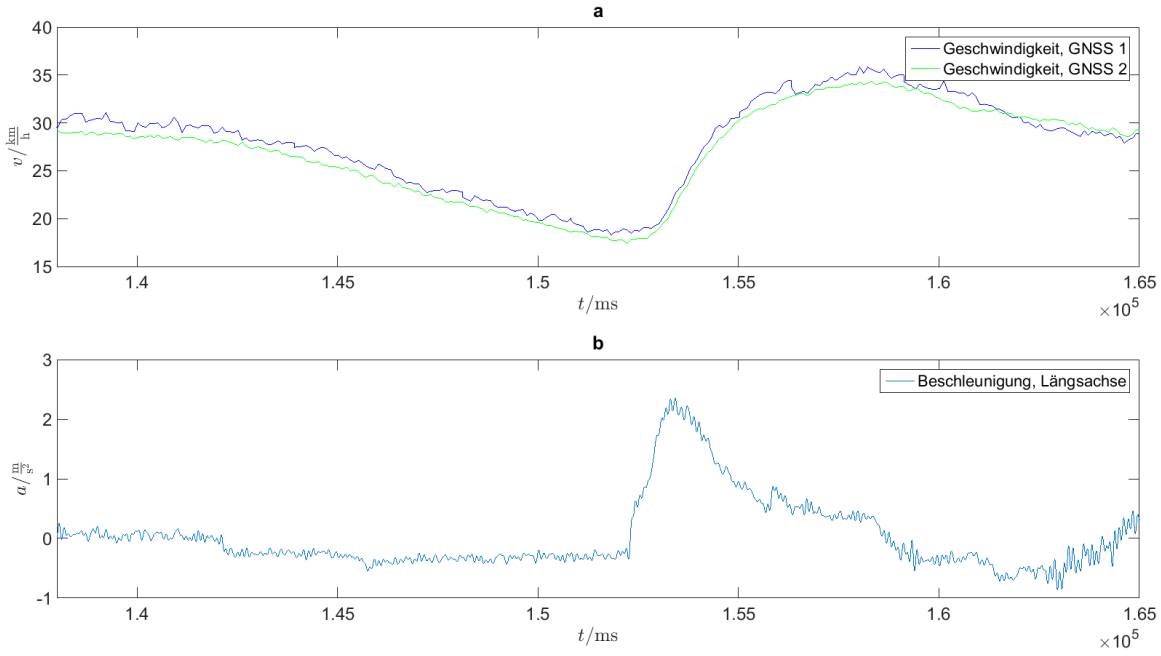


Abbildung 7.8.: Messung-Nr. 4: Geschwindigkeit und Beschleunigung ($f_{g,TP} = 10\,\mathrm{Hz}$) bei Kreisfahrt mit veränderlicher Geschwindigkeit, **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem, digital tiefpassgefiltert

An der Drehrate um die Gierachse ist dabei deutlich zu erkennen, wie sie mit der Geschwindigkeit abnimmt. Außerdem fällt an der Beschleunigung auf der Längsachse auf, dass die Verzögerung des Fahrzeugs mit einer ähnlichen Größenordnung eingeht wie die Querbeschleunigung. Hier wird also mit einem relativ großen Fehler gemessen.

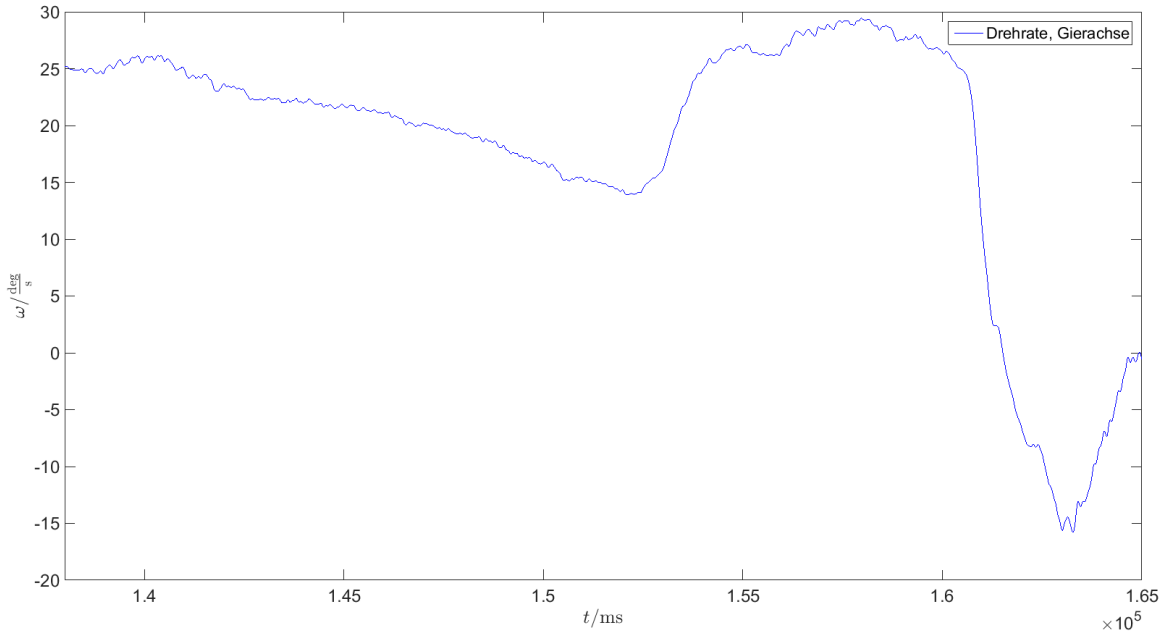


Abbildung 7.9.: Messung-Nr. 4: Drehrate um die Gierachse bei Kreisfahrt mit veränderlicher Geschwindigkeit

Bei der vergleichsweise großen Beschleunigung dagegen, fällt der Eintrag der Querbeschleunigung weniger ins Gewicht, da er lediglich etwa $5\,\%$ der durch den Fahrer verursachten Längsbeschleunigung ausmacht.

Zusammenfassend ist nach der Messung zur Kreisfahrt mit veränderlicher Geschwindigkeit festzustellen:

- Die Drehrate um die Gierachse verhält sich wie erwartet abhängig mit der Kreisbahngeschwindigkeit
- Mit zunehmender Kurvenbahngeschwindigkeit und Drehrate werden kleine Beschleunigungen deutlich verfälscht

7.3. Messungen im Nahverkehrsbus

Durchführung

Die Messungen im Nahverkehrsbus erfolgten im normalen Linienbetrieb der Busse. Der Datenlogger konnte deshalb nicht fest verbaut werden, sondern wurde auf einem der hinteren Sitze in Fensternähe abgelegt. Durch eine rutschfeste Unterlage konnte verhindert werden,

dass sich das Gerät auf dem Sitz bewegt. Außerdem ermöglichte die Unterlage ein schnelles, möglichst achsengerechtes Ausrichten vor dem Fahrtbeginn. Die Ausrichtung erfolgt lediglich nach Augenmaß.
Um den GNSS-Empfängern den Satellitenempfang zu ermöglichen, wurden diese während der gesamten Fahrt an das Seitenfenster des Busses gehalten. Auch hier konnte eine weitere Befestigung nicht erfolgen.
Eine Fotodokumentation des Messaufbaus entfiel, da die Bedienung der Messung dies nicht zuließ.

Die Datenaufzeichnung zu beiden Messungen beginnt jeweils kurz vor der Fahrt und Platzierung des Gerätes, da sichergestellt werden soll, dass der Bus nicht anfährt bevor die Aufzeichnung läuft. Dies hat zur Folge, dass die einzelnen aufgezeichneten Daten mit einem „ungültigen“ Zeitraum von einigen Sekunden beginnen. Dieser ist in der Auswertung zu vernachlässigen.

Gemessen wurde jeweils von Start- zu Endhaltestelle der Buslinie, in jede Richtung einmal. Dabei fand die Fahrt in den folgenden Busmodellen statt:

- Messung 5: Volvo 7900 Hybrid mit parallelem Elektro- und Dieselmotor
- Messung 6: Mercedes Benz Citaro mit Dieselmotor

Die mit dem Datenlogger aufgezeichnete Fahrtroute aus Messung 5 ist in Abb. C.14 auf Luftbildaufnahmen dargestellt. In Messung 6 wird dieselbe Strecke in umgekehrter Richtung abgefahren.

Auswertung

Bei der Auswertung der erhobenen Messdaten sollen die aus der Satellitenortung bestimmten Pfade in relevanten Detailausschnitten betrachtet und bewertet werden. Außerdem sind als Grundlage für die weitere Auswertung die gemessene Längsbeschleunigung sowie die aus der Satellitenortung bestimmte Geschwindigkeit für die Messungen 5 und 6 in den Abb. C.15 resp. C.16 über die Messdauer aufgetragen. Die ersten Sekunden der Messung, in denen der Datenlogger noch nicht im Bus positioniert ist, sind daran zu erkennen, dass die Geschwindigkeit laut beider GNSS-Empfänger nahezu null ist, während das Inertialmesssystem auf allen Achsen große Ausschläge aufweist. Das System ist bei beiden Messungen vor dem ersten Anfahren des Fahrzeugs in Ruhelage.

Zunächst fallen an den, mittels der GNSS-Empfänger aufgenommenen, Pfaden auf georeferenzierten Luftbildern einzelne Besonderheiten auf. So ist z.B. in Abb. C.17 zu erkennen, dass nach Beginn der Messung noch recht große Abweichungen in den ermittelten Positionen vorliegen. Die Positionsangaben aus beiden Empfängern nähern sich dann innerhalb

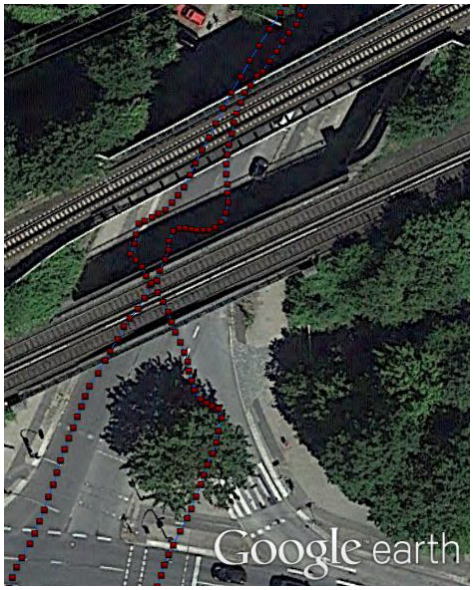


Abbildung 7.10.: Messfahrt zu Messung-Nr. 6: Positionsfehler beim Unterfahren einer Brücke, Quelle: Google Earth

kurzer Zeit bis nahezu auf den gleichen Startpunkt an. Noch bevor der Bus an der Starthaltestelle abfährt ist ein Ergebnis erreicht, dass innerhalb der möglichen Positionsgenauigkeit der Empfänger liegt.

An den Abb. C.18 und 7.10 ist der Einfluss von Brücken auf die Satellitenortung zu erkennen, unter welchen der Empfänger verschattet wird. In das Berechnungsverfahren auf dem Empfänger gehen kurzzeitig fehlende oder falsche Messwerte ein, sodass sich ein Fehler in der festgestellten Position ergibt. Der Fehler wird erst wirksam korrigiert, wenn der Empfang wiederhergestellt ist. Außerdem ist am Verhalten der GNSS-Module zu erkennen, dass vermutlich Filterverfahren wie z.B. Kalman-Filter zum Einsatz kommen. Aufeinanderfolgende Positionsbestimmungen unter schlechter werdenden Bedingungen führen zwar zu einem zunehmenden Fehler, es treten aber keine unplausiblen Sprünge in der Positionsbestimmung auf. Das Filterverhalten ist auch an Abb. 7.11 zu erkennen: Einer der GNSS-Empfänger errechnet als Ergebnis, dass der Bus nicht unmittelbar an der Haltestelle anhielt, sondern zunächst weiterfuhr und dann zurückkehrte. Dieses Verhalten legt nahe, dass ein Filter zum Einsatz kommt, dass das Systemverhalten anhand der zurückliegenden Messwerte schätzt.

Abb. 7.12 zeigt, dass insbesondere beim Fahrzeughalt ein langsames Abdriften der ermittelten Position auftritt. Ein längerer Halt, wie z.B. an einer Haltestelle führt hier zu einem größeren Fehler. Dieses Verhalten tritt zuverlässig an allen angefahrenen Haltestellen auf. In Abb. C.15 und C.16 ist zu sehen, dass die Berechnungsverfahren zur Ermittlung der Geschwin-

Abbildung 7.11.: Messfahrt zu Messung-Nr. 5: Positionsfehler durch Filterverfahren des GNSS-Moduls, Quelle: Google Earth

digkeit trotzdem nur zu sehr kleinen Fehlern im Stillstand führen. Wie auch in Abb. 7.13 zu erkennen, liegt der Fehler in den meisten Haltephasen deutlich unter $1\,\frac{\mathrm{km}}{\mathrm{h}} \approx 0.28\,\frac{\mathrm{m}}{\mathrm{s}}$. Nur in seltenen Fällen steigt der Fehler für wenige Abtastwerte auf einige $\frac{\mathrm{km}}{\mathrm{h}}$. Das hier tatsächlich keine Geschwindigkeitsänderung vorliegt, ist an der gemessenen Längsbeschleunigung zu erkennen.

Die Abb. C.19 und C.20 zeigen, dass die unterschiedlich ermittelten Positionen der beiden GNSS-Empfänger zu teils deutlich unterschiedlichen Geschwindigkeitsprofilen führen. Zeitlich äquidistant abgetastete Positionen zeigen hier für beide GNSS-Empfänger deutlich unterschiedliche Strecken auf der Karte.
Der Fehler in der Positionsbestimmung nimmt augenscheinlich im Laufe der Messung 6 zu, Beobachtungen während der Messung waren zunehmende Bewölkung und beginnender Regen. In Abb. C.21 ist über den Verlauf der Messung 6 jeweils der Mittelwert der verwendeten Satelliten über je $1\,\mathrm{min}$ als Balkendiagramm dargestellt. Es ist eine abnehmende Tendenz zu erkennen.

Eine Betrachtung der gemessenen Längsbeschleunigung zeigt zunächst, dass die positiven und negativen Beschleunigungen in ähnlichen Größenordnungen auftreten. Außerdem wird an Abb. 7.14 deutlich, dass bei der Auswertung der gemessenen Beschleunigung zur Ermittlung der kinetischen Energie des Fahrzeugs die Zeiträume, in denen das Fahrzeug steht, besonders beachtet werden müssen. Obwohl das Fahrzeug steht und ihm keine kinetische Energie zu- oder abgeführt wird, wird die Erdbeschleunigung anteilig auf der Längsachse des Inertialmesssystems gemessen.

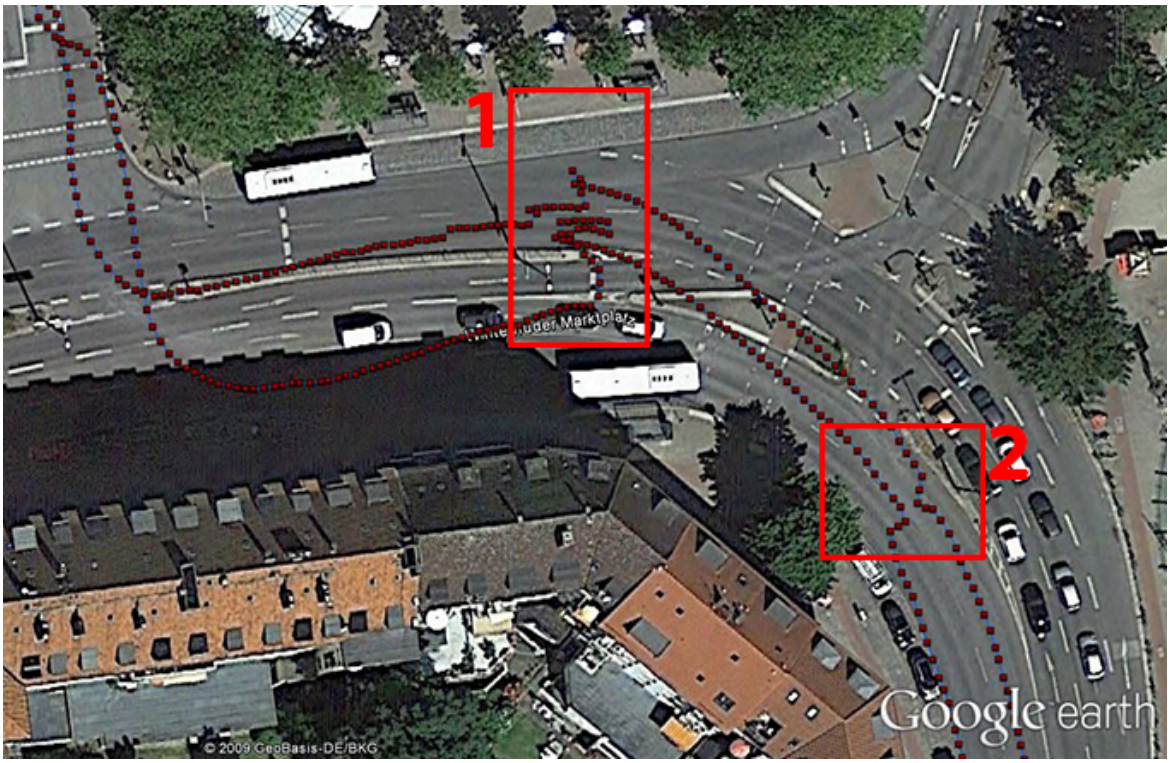


Abbildung 7.12.: Messfahrt zu Messung-Nr. 6: Positionsfehler im Stand, 1: Halt an einer Haltestelle, 2: Halt an einer Ampel, Quelle: Google Earth

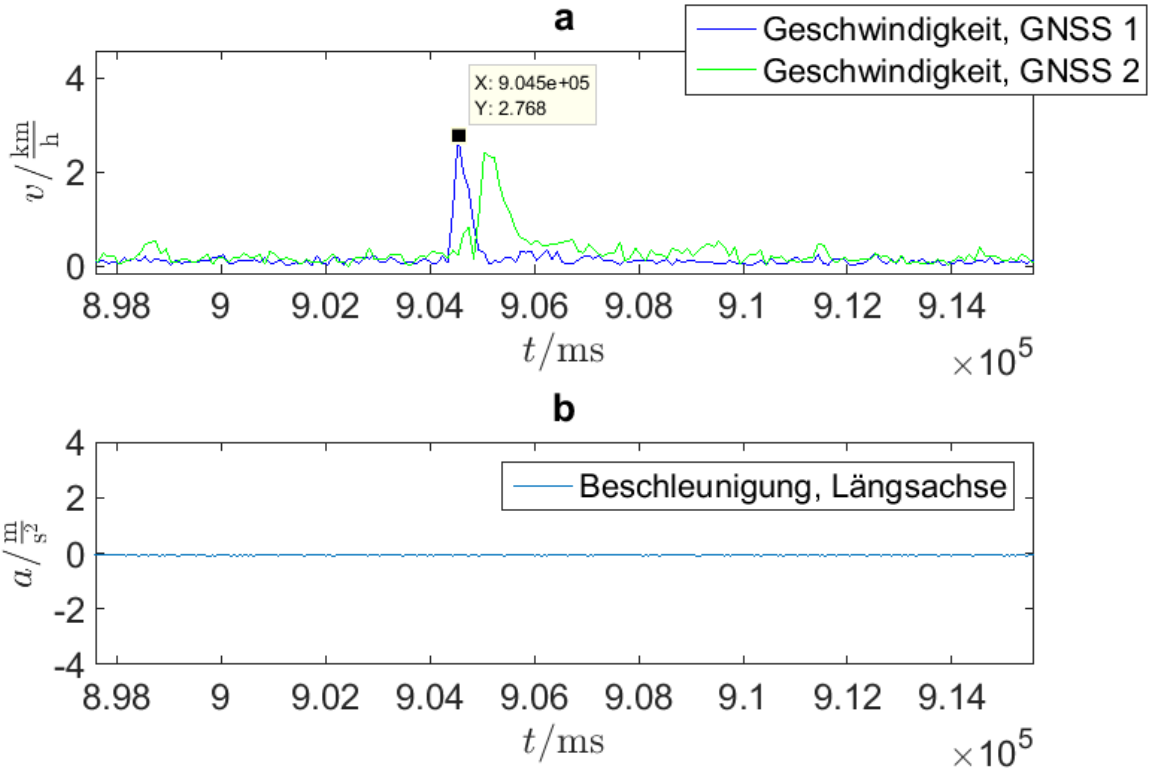


Abbildung 7.13.: Messfahrt zu Messung-Nr. 5: Geschwindigkeitsfehler im Stand, a: , **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem ($f_{g,TP} = 10\,\mathrm{Hz}$)

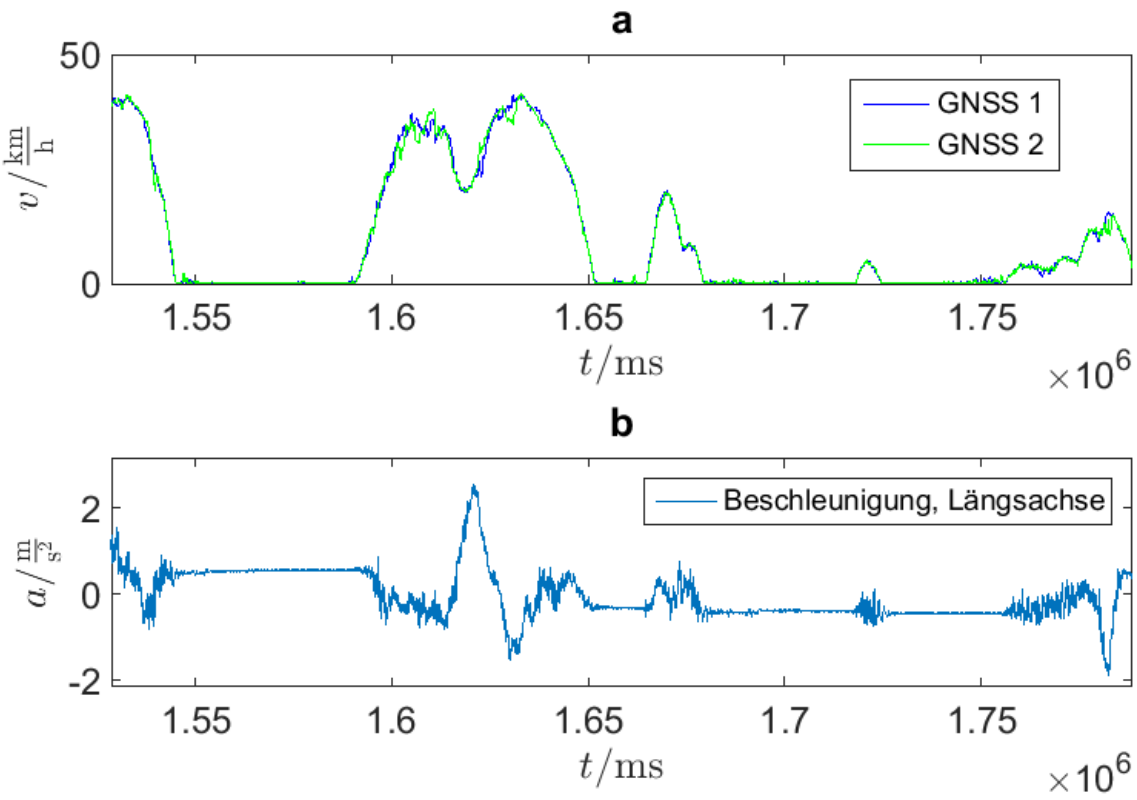


Abbildung 7.14.: Messfahrt zu Messung-Nr. 5: Beschleunigungsfehler im Stand, a: , **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem ($f_{g,TP} = 10\,\mathrm{Hz}$)

Die Höhenmessung mittels Satellitenortung während beider Messungen führt zu den Ergebnissen in Abb. C.24. Selbst unter Vernachlässigung der ersten Sekunden, in denen die Satellitenortung noch zu einer sehr ungenauen Position führt, sind über den gesamten Fahrtverlauf große Abweichungen zwischen den Messungen beider GNSS-Empfänger zu beobachten. Zudem weichen die Ergebnisse beider Messfahrten stark voneinander ab, obwohl dieselbe Strecke nur in umgekehrter Richtung befahren wurde.
Weiter enthalten die Höhenmessungen unplausible Werte, wie z.B. negative Höhen über NN oder große Höhenunterschiede. Die höchste Erhebung Hamburgs, welche zudem nicht auf der Strecke liegt, hat eine Höhe über NN von etwa $116\,\mathrm{m}$ über NN.

Zuletzt soll noch die Drehrate um die Gierachse, wie in Abb. C.22 und C.23 dargestellt, Beachtung finden.
An den Aufzeichnungen beider Fahrten ist zu erkennen, dass größere Drehraten nur in kurzen Spitzen, z.B. bei Abbiegevorgängen, auftreten. Selbst in diesen Spitzen liegen die Drehraten meist deutlich unter $10\,\frac{\mathrm{deg}}{\mathrm{s}}$ und damit weit unter den in Abschnitt 7.2.4 verwendeten Drehraten. Da kein Nahverkehrsbus für entsprechende Messungen zur Verfügung steht, können auch die tatsächlichen Schwimmwinkel in den für eine normale Busfahrt relevanten Fahrzuständen nicht ermittelt werden.

Zusammenfassend ist nach den Messungen im Nahverkehrsbus festzustellen:

- Satellitenortung und darauf basierende Geschwindigkeitsmessung sind auch im Nahverkehrsbus und durch Fahrzeugscheiben funktionsfähig
- Es treten teils deutliche Unterschiede in Position und Geschwindigkeit zwischen den Messergebnissen der beiden GNSS-Empfänger auf
- Die Beschleunigungsmessung auf der Längsachse führt im Stillstand auch im Nahverkehrsbus zu deutlichen Fehlern
- Gemessene Drehraten und damit vermutlich auch der Fehler durch Querbeschleunigung sind deutlich kleiner als die im PKW gemessenen und in der Theorie als maximal angenommenen
- Die Höhenmessung mittels Satellitenortung führt zu Fehlern von mehreren $10\,\mathrm{m}$
- Abweichungen zwischen den GNSS-Empfängern fallen deutlich größer aus, als bei den Messungen im PKW

7.4. Messungen mit den Temperatursensoren

Während einer der Temperatursensoren frei auf dem Labortisch liegt, wird der andere mittels Gewebeklebeband am Rohr des Heizungsvorlaufs befestigt. Die Messung am Datenlogger

wird gestartet und über meherere Stunden laufengelassen.
Es ergibt sich aus den Messergebnissen die Abb. C.25. Daran ist deutlich zu erkennen, wie die Raumtemperatur der Temperatur des Heizungsvorlaufs in der Nachtabsenkung bis etwa 3:00 Uhr folgt und danach mit dem deutlichen Anstieg der Vorlauftemperatur, zur Beheizung der Räume vor deren Nutzung, wieder zunimmt.

Zusammenfassend ist nach der Messung mit den Temperatursensoren festzustellen:

- Die Temperaturmessung des Datenloggers ist funktionsfähig
- Mehrere Temperaturen können korrekt erfasst werden
- Bei Temperaturen kleinerer Amplituden wird die Quantisierung durch die Digitalisierung des Messwerts sichtbar

8. Bewertung und Ausblick

8.1. Bewertung in Bezug auf die Anforderungen

Die durchgeführten Versuche zeigen, dass die Messung der Längsbeschleunigung eines Fahrzeuges im Allgemeinen und eines Nahverkehrsbusses im Speziellen mit dem entwickelten Datenlogger möglich sind. Auch die angeforderte Abtastrate wird dabei erreicht. Weitere Betrachtungen zeigen allerdings, dass diese, für durch den Fahrer verursachte Längsbeschleunigungen, vermutlich keine Bedeutung haben.
Berechnungen und Versuche zeigen, dass die Ermittlung der tatsächlich energetisch relevanten Beschleunigung aus der gemessenen Längsbeschleunigung unter Zuhilfenahme weiterer Messdaten erfolgen muss, da die Messwerte durch andere als die gewünschten Einflüsse verfälscht werden. Dafür stellt der Datenlogger mit den Drehratensensoren eine vermutlich geeignete Datenbasis zur Verfügung.

Die Aufzeichnung der durch Satellitenortung bestimmten Position erfolgt mit einer um eine Größenordnung feineren Auflösung als gefordert. Dabei ist die absolute Genauigkeit der Positionsbestimmung mit etwa $2.5\ \mathrm{m}$ deutlich schlechter.
Insbesondere die Messung der Höhe über NN durch Satellitenortung erreicht keine ausreichende Genauigkeit. Die Zuordnung der Zeitstempel in Millisekunden zur UTC-Zeit erlaubt deren bis zu Millisekunden genaue Bestimmung bei Verwendung der GNSS-Empfänger als Zeitbasis.
Geschwindigkeiten werden vom Datenlogger mit mehr als geforderter Auflösung aufgezeichnet, ihre tatsächliche Genauigkeit ist, wie in den Versuchen in Abschnitt 7.3 festgestellt, stark abhängig von der Qualität der Satellitenortung. Außerdem müssen u.U. Einflüsse von Filterverfahren innerhalb des GNSS-Empfängers berücksichtigt werden.

Anhand von Messungen ist belegt, dass die Temperaturmessung mittels mehrerer angeschlossener Temperatursensoren möglich ist.

Die Speicherung der Daten erfolgt in einem von MATLAB lesbaren Format, aufgrund der Echtzeitbedingungen bei der Datenaufzeichnung wird allerdings in mehr als nur eine Datei geschrieben. Die Zusammenführung der Daten muss anschließend in MATLAB erfolgen.

Durch die äquidistante und auf einer Zeitbasis synchronisierte Abtastung verschiedener Messgrößen können die Ergebnisse von Messungen mit gängigen Verfahren der digitalen Signalverarbeitung weiterverarbeitet werden.

Möglichkeiten zur Fernkommunikation mit dem Datenlogger wurden zwar in Hardware geschaffen, eine Implementierung in der Software des Datenloggers erfolgte allerdings nicht.

Da das verwendete Evaluationsboard noch weitere, bisher ungenutzte Hardware-Schnittstellen verschiedenen Typs zur Verfügung stellt, kann die Anbindung von drahtlosen Batteriesensoren aus dem Projekt BATSEN unter Verwendung der entsprechenden Empfänger-Hardware bei Bedarf erfolgen.

Nicht erfüllt werden konnte im Speziellen die Anforderung aus dem BEEDeL-Projekt zur Kennzeichnung von Haltestellen sowie die Benennung der Ausgabedatei anhand der befahrenen Buslinie.

8.2. Ausblick

Mit dem Datenlogger können nun Daten aus tatsächlichen Fahrten von Nahverkehrsbussen erhoben und für die Untersuchung von Batteriezellen eingesetzt werden. Dafür ist es, wie in [29] von Schmidt beschrieben, nötig, Verfahren zu entwickeln, die aus den kinetischen Energiezuständen des Fahrzeugs Lade- und Entladeströme der Traktionsbatterie ermitteln.
Anhand der einfachen theoretischen Betrachtung in Abschnitt 2.1.1 und den Versuchsergebnissen aus Abschnitt 7.3 kann geschlossen werden, dass die Ströme der Entladung, beim Beschleunigen, sowie der Ladung, durch Rekuperation beim Abbremsen, eine ähnliche Größenordnung haben. Diese Tatsache stellt hohe Anforderungen an die Traktionsbatterie, da Lithium-Ionen-Batterien in der Regel mit deutlich höheren Strömen entladen, als geladen werden können.

Um die Beziehung zwischen der Veränderung des kinetischen Energiezustands des Fahrzeugs und den Batterieströmen und -spannungen messen zu können, ist die parallele Messung dieser Größen sinnvoll. Die im BATSEN-Projekt u.a. von Sassano [28] entwickelten drahtlosen Batteriesensoren können dazu prinzipiell verwendet werden. Die nötige Empfangs-Hardware muss dafür an den Mikrocontroller angeschlossen und die Software entsprechend erweitert werden.

Die, mit teils relevanter Größe, in die Messung der Längsbeschleunigung eingehende Querbeschleunigung bei Kurvenfahrt sollte entweder in der Software des Datenloggers oder in der Nachbereitung der Messdaten durch die Entwicklung und Implementierung geeigneter Filterverfahren herausgerechnet werden. Auch sollte ein derartiges Verfahren in der Lage

sein die in der energetischen Betrachtung sehr großen Fehler, durch gemessene Erdbeschleunigung im Stillstand des Fahrzeugs, zu vermindern. Hierfür stellen die durch den Datenlogger aufgezeichneten Drehraten u.U. geeignete Messgrößen dar, da sie zusammen mit den Beschleunigungssensoren Aufschluss über die Raumlageveränderung des Fahrzeugs zulassen. Eine weitere Zusammenführung der Daten aus der Satellitenortung mit denen des Inertialmesssystems erscheint zudem erstrebenswert.
Soll das vorgeschlagene und in Abschnitt 6.6 theoretisch vorgestellte Kalibrierverfahren zukünftig Anwendung finden, ist das Verfahren zu überprüfen und insbesondere in der Praxis auf Tauglichkeit zu testen. Im Rahmen dieser Arbeit fand lediglich eine experimentelle Implementierung ohne Durchführung von Praxistests statt.

Beim Aufbau weiterer und u.U. überarbeiteter Datenlogger sollten die Platinen gemäß der Schaltpläne in Anhang B neu entworfen werden, sodass die auf den im Prototypen verwendeten Platinen durchgeführten manuellen Änderungen an den Schaltungen in das Platinenlayout übernommen werden.
Weiterentwicklungen des Datenloggers sollten zudem das Versorgungskonzept mittels des verwendeten USB-Akkumulators überdenken und u.U. eine eigene Lade- und Entladeregelung einer Akkumulatorzelle zur Versorgung des Gerätes vorsehen. Die Steuerung der Spannungsversorgung wird damit flexibler und eine erweiterte Kontrolle darüber aus der Software des Mikrocontrollers möglich.

Literaturverzeichnis

[1] ABRACON: *APAE1590R2540AKDB1-T*, März 2014. – URL http://www.abracon.com/patchantenna/APAE1590R2540AKDB1-T.pdf. – Zugriffsdatum: 05. 05. 2015. – Revision 02.03.2014

[2] ANDREA, Davide: *Battery Management Systems for Large Lithium Ion Battery Packs*. Norwood, MA : Artech House, 2010. – ISBN 978-1-60807-104-3

[3] BEHÖRDE FÜR STADTENTWICKLUNG UND UMWELT: Luftreinhalteplan für die Freie und Hansestadt Hamburg. (2004), Oktober. – URL http://www.hamburg.de/contentblob/143556/data/luftreinhalteplan.pdf. – Zugriffsdatum: 10. 05. 2015

[4] DEPPNER, Heinz G.: Drehratenmessgeber. In: *Technische Berichte* 32 (1999), Dezember, S. 10–15. – URL http://www.etech.fh-hamburg.de/tb/tb32/Deppner.pdf. – Zugriffsdatum: 13. 04. 2015

[5] FRIEDRICH, Dr.-Ing. Jürgen K.-H. ; BRAESS, Hans-Hermann (Hrsg.) ; SEIFFERT, Ulrich (Hrsg.): *Vieweg Handbuch Kraftfahrzeugtechnik*. 7. Wiesbaden : Srpinger Vieweg, September 2013. – ISBN 978-3658016906

[6] FUEST, Klaus ; DÖRING, Peter: *Elektrische Maschinen und Antriebe: Lehr- und Arbeitsbuch für Gleich-, Wechsel- und Drehstrommaschinen sowie Elektronische Antriebstechnik*. 7. Wiesbaden : Friedr. Vieweg & Sohn Verlag, August 2007. – ISBN 978-3834800985

[7] HAMBURGER HOCHBAHN AG: *Innovationslinie 109*. 2015. – URL http://www.hochbahn.de/wps/portal/de/home/hochbahn/aktuelles/Innovationslinie+Site?1dmy&current=true&urile=wcm:path:/Hochbahn_Content/home/hochbahn/aktuelles/Innovationslinie+Site. – Zugriffsdatum: 10. 05. 2015

[8] HAMBURGER SENAT: Mobilitätsprogramm 2013. (2013), September. – URL http://www.hamburg.de/contentblob/4119700/data/mobilitaetsprogramm-2013.pdf. – Zugriffsdatum: 11. 05. 2015

[9] HORN, Michael ; HÖFLINGER, Fabian ; TRÄNKLER, Hans-Rolf (Hrsg.) ; REINDL, Leonhard M. (Hrsg.): *Sensortechnik - Handbuch für Praxis und Wissenschaft*. 2. Berlin Heidelberg : Springer Vieweg, September 2014. – 541– S. – ISBN 978-3-642-29941-4

[10] INVENSENSE: *MPU-9250 Register Map and Descriptions*, September 2013. – URL http://www.invensense.com/mems/gyro/documents/RM-MPU-9250A-00.pdf. – Zugriffsdatum: 27. 04. 2015. – Revision 1.4

[11] IST AG: *TSic Digitale Halbleitertemperatursensoren TSIC206/306*, April 2008. – URL https://cdn-reichelt.de/documents/datenblatt/B400/TSIC_Sensoren_dbd.pdf. – Zugriffsdatum: 27. 04. 2015

[12] KENTEC DISPLAY: *Stellaris LaunchPad LCD Boosterpack EB-LM4F120-L35*, Februar 2014. – URL http://www.farnell.com/datasheets/1653560.pdf. – Zugriffsdatum: 27. 04. 2015. – Revision 03

[13] LINEAR TECHNOLOGY: *LT1761 Series*. Milpitas, CA, USA: , Mai 2010. – URL http://cds.linear.com/docs/en/datasheet/1761sff.pdf. – Zugriffsdatum: 05. 05. 2015. – Revision F

[14] LINEAR TECHNOLOGY: *Low Loss PowerPath Controller in ThinSOT*, Februar 2015. – URL http://cds.linear.com/docs/en/datasheet/4412fb.pdf. – Zugriffsdatum: 04. 05. 2015. – Revision B

[15] MATHWORKS: *MAT-File Format*. Natick, MA: , März 2015. – URL https://www.mathworks.com/help/pdf_doc/matlab/matfile_format.pdf. – Zugriffsdatum: 10. 05. 2015. – Revision R2015a

[16] MAXIM INTEGRATED: *Using a UART to Implement a 1-Wire Bus Master*, September 2002. – URL http://www.maximintegrated.com/en/app-notes/index.mvp/id/214. – Zugriffsdatum: 27. 04. 2015. – Revision 10.09.2002

[17] MAXIM INTEGRATED: *MAX3222/MAX3232/MAX3237/MAX3241**, Januar 2007. – URL http://pdfserv.maximintegrated.com/en/ds/MAX3222-MAX3241.pdf. – Zugriffsdatum: 05. 05. 2015. – Revision 7

[18] MAXIM INTEGRATED: *DS18B20 Programmable Resolution 1-Wire Digital Thermometer*, April 2008. – URL http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf. – Zugriffsdatum: 27. 04. 2015. – Revision 042208

[19] MICREL INC.: *MIC29150/29300/29500/29750*. San Jose, CA, USA: , December 2012. – URL http://www.micrel.com/_PDF/mic29150.pdf. – Zugriffsdatum: 05. 05. 2015. – Revision M9999-122012-B

[20] MITSCHKE, Manfred ; WALLENTOWITZ, Henning: *Dynamik der Kraftfahrzeuge*. 5. Berlin Heidelberg : Springer Vieweg, August 2014. – ISBN 978-3-658-05067-2

[21] MOLEX: *Micro SD Conn. normal ultra low profile assy*, Januar 2012. – URL http://www.mouser.com/ds/2/276/5031821852_sd-339188.pdf. – Zugriffsdatum: 05. 05. 2015. – Revision G

[22] N.N.: *BATSEN: Department Informations- und Elektrotechnik: HAW Hamburg*. 2011. – URL http://www.haw-hamburg.de/department-informations-und-elektrotechnik/forschung/forschungsschwerpunkt-adys/batsen.html. – Zugriffsdatum: 10. 05. 2015

[23] N.N.: Jahresbericht 2014. (2014). – URL http://www.now-gmbh.de/fileadmin/user_upload/RE_Publikationen_NEU_2013/Publikationen_NOW_Berichte/NOW_Jahresbericht_2014.pdf. – Zugriffsdatum: 09. 05. 2015

[24] N.N.: Bewertung des Einsatzes von Elektrobussen mit dezentraler Ladeinfrastruktur. (2015)

[25] RASPBERRY PI FOUNDATION: *Raspberry Pi Website*. – URL https://www.raspberrypi.org/. – Zugriffsdatum: 27. 04. 2015

[26] RILL, Georg: *Fahrzeugdynamik*. 2001. – URL http://www.autogumi.com/FDV_Skript.pdf. – Zugriffsdatum: 13. 04. 2015

[27] RODT, Stefan ; GEORGI, Birgit ; HUCKESTEIN, Burkhard ; MÖNCH, Lars ; HERBENER, Reinhard ; JAHN, Helge ; KOPPE, Katharina ; LINDMAIER, Jörn: CO2-Emissionsminderung im Verkehr in Deutschland. (2010), März. – URL http://www.umweltbundesamt.de/uba-info-medien/mysql_medien.php?anfrage=Kennummer&Suchwort=3773. – Zugriffsdatum: 10. 05. 2015. – ISSN 1862-4804

[28] SASSANO, Nico: *Hard- und Softwareentwicklung für einen drahtlos kommunizierenden Batterie-Zellensensor mit funksynchronisierter Messung*. Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorthesis, August 2013

[29] SCHMIDT, Oliver: *Auslegung und Erprobung eines Lithiumbatterie-Prüfstands für typische Lade- und Lastprofile von Elektrobussen*. Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorthesis, Juni 2015. – in Bearbeitung

[30] SCHOLZ, Olaf: *Eröffnung der Innovationslinie 109*. Dezember 2014. – URL http://www.hamburg.de/contentblob/4427238/data/2014-12-18-innovationslinie-109.pdf. – Zugriffsdatum: 07. 04. 2015. – Ansprache des ersten Bürgermeisters Olaf Scholz zur Eröffnung der Innovationslinie 109 der Hamburger Hochbahn

[31] SCHÜTTLER, Tobias: *Satellitennavigation - Wie sie funktioniert und wie sie unseren Alltag beeinflusst*. Berlin Heidelberg : Springer Vieweg, 2014. – ISBN 978-3-642-53886-5

[32] SD GROUP: *SD Specifications Part 1 Physical Layer*, September 2006. – URL https://www.sdcard.org/downloads/pls/simplified_specs/archive/part1_200.pdf. – Zugriffsdatum: 27. 04. 2015. – Revision 2.00

[33] SIMCOM: *SIM800H Hardware Design*, August 2013. – URL http://wm.sim.com/upfile/2013828142822f.pdf. – Zugriffsdatum: 05. 05. 2015. – Revision 1.01

[34] TEXAS INSTRUMENTS: *Tiva C Series TM4C1294 Connected LaunchPad Evaluation Kit*, März 2014. – URL http://www.ti.com/lit/ug/spmu365a/spmu365a.pdf. – Zugriffsdatum: 27. 04. 2015

[35] TEXAS INSTRUMENTS: *Tiva TM4C1294NCPDT Microcontroller*, Juni 2014. – URL http://www.ti.com/lit/ds/spms433b/spms433b.pdf. – Zugriffsdatum: 27. 04. 2015. – Revision 15863.2743

[36] TEXAS INSTRUMENTS: *TivaWare Peripheral Driver Library*, Februar 2014. – URL http://www.ti.com.cn/cn/lit/ug/spmu298a/spmu298a.pdf. – Zugriffsdatum: 27. 04. 2015. – Revision 2.1.0.12573

[37] TEXAS INSTRUMENTS ; MAGNUSON, Jon (Hrsg.): *FatFS diskio.h/diskio.c*. 2015. – URL https://github.com/jmagnuson/fatfs-tiva-cm4f/blob/master/src/third_party/fatfs/port/mmc-tiva-cm4f.c. – Zugriffsdatum: 15. 02. 2015

[38] TEXAS INSTRUMENTS: *TM4C1294 Connected LaunchPad*. 2015. – URL http://www.ti.com/tool/ek-TM4C1294xl. – Zugriffsdatum: 27. 04. 2015

[39] THOM, Marc G.: *Hard- und Softwareentwicklung von einem Beschleunigungs-Datenlogger für Kraftfahrzeuge*. Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorthesis, September 2014

[40] TSCHAKERT, Wolfgang: IBC 2014: Der Citaro gegen den Rest der Welt. In: *BUS fahrt* (2014), Mai. – URL http://busfahrt.com/images/stories/testberichte/ibc_2014.pdf. – Zugriffsdatum: 05. 05. 2015

[41] U-BLOX: *CAM-M8Q*, November 2014. – URL https://www.u-blox.com/images/downloads/Product_Docs/CAM-M8Q_DataSheet_%28UBX-13004081%29.pdf. – Zugriffsdatum: 17. 04. 2015. – Revision R06

[42] U-BLOX: *u-blox M8 Receiver Description*, Dezember 2014. – URL https://www.u-blox.com/images/downloads/Product_Docs/u-bloxM8_ReceiverDescriptionProtocolSpec_%28UBX-13003221%29_Public.pdf. – Zugriffsdatum: 17. 04. 2015. – Revision R08

[43] VOLVO: *Volvo 7900 Electric Hybrid*, 2014. – URL http://www.volvobuses.com/bus/germany/de-de/products_services/buses/City%20buses/volvo_7900_electric_hybrid/Documents/Technische-Daten-Volvo-7900-Electric-Hybrid_DE.pdf. – Zugriffsdatum: 08. 05. 2015

[44] WESTERMANN, Thomas: *Mathematik für Ingenieure*. 5. Berlin Heidelberg : Springer Vieweg, April 2008. – ISBN 978-3540777304

[45] WISNIEWSKI, Thomas: *Zyklierprüfstand für Batteriezellen mit Steuerung durch einen ARM-Controller sowie Messdatenverwaltung und Netzwerkanbindung*. Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorthesis, Juli 2013

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis und Glossar

(Micro-) SD-Karte	Speicherkarte
Abb.	Abbildung
ADC	Analog-Digital-Umsetzer
ARM	Prozessorarchitektur
BATSEN	Forschungsprojekt; „Drahtlose Zellensensoren für Fahrzeugbatterien“
BEEDeL	Forschungsprojekt; „Bewertung des Einsatzes von Elektrobussen mit dezentraler Ladeinfrastruktur in Metropolen am Beispiel der HOCHBAHN“
bzw.	beziehungsweise
CCS	Code Composer Studio; Entwicklungsumgebung für Mikrocontroller des Herstellers Texas Instruments
CDMA	Code Division Multiple Access; Multiplexverfahren
CO_2	Kohlenstoffdioxid
DFT	Diskrete Fourier-Transformation
DMA	Direct Memory Access; Speicherdirektzugriff
Eclipse	Quelloffene Entwicklungsumgebung
FatFS	Dateisystem
Fraunhofer IVI	Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme IVI
GPS (NAVSTAR)	Global Positioning System; Satellitennavigationssystem betrieben durch das Verteidigungsministerium der USA
ggf.	gegebenenfalls
Ringbuffer	Ringpuffer; Im Ring angeordnete Warteschlange
GLONASS	Satellitennavigationssystem betrieben durch das russische Verteidigungsministerium
GNSS	Globales Navigationssatellitensystem
GPIO	General Purpose Input/Output
GPRS	General Packet Radio Service; Paketorientierter Datenübertragungsdienst in GSM-Netzen
GSM	Global System for Mobile Communications
HAW	Hochschule für Angewandte Wissenschaften
HMI	Human-Machine-Interface; Mensch-Maschine-Schnittstelle
Höhe über NN	Höhe über dem mittleren Meeresspiegel

IMU	Inertial Measurement Unit; Inertialmesssystem
Jitter	Amplitude der zeitlichen Schwankung im Taktsignal
LCD	Liquid Crystal Display; Flüssigkristallanzeige
LED	Light Emitting Diode; Leuchtdiode
LKW	Lastkraftwagen
LSB	Least Signifikant Bit; Niederwertigstes Bit
LTE	Long Term Evolution; Mobilfunkstandard
MATLAB	Software des Herstellers MathWorks
MEMS	Microelectomechanical System; Mikroelektromechanisches System
NMEA 0183	Standard zur Kommunikation von Navigationsgeräten auf Schiffen
NTC Widerstand	Negativ Temperature Coefficient (Thermistor); Heißleiter
PCB	Printed Circuit Board; Gedruckte Schaltung oder Platine
PKW	Personenkraftwagen
PTC Widerstände	Positive Temperature Coefficient (Thermistor); Kaltleiter
Qt	Platformübergreifende C++ -Bibliothek für grafische Oberflächen
Rekuperation	Energierückgewinnung
resp.	respektive
RTC	Real Time Clock
S-ATA	Serial Advanced Technology Attachment; Schnittstelle zwischen Prozessor und Datenträgern
SIM-Karte	Subscriber Identity Module; Chipkarte zur Nutzeridentifikation in Mobilfunknetzen
sog.	sogenannte(n)
SPI	Serial Peripheral Interface
TivaWare	C-basierte Treiber-Bibliothek des Herstellers Texas Instruments für dessen Mikrocontroller
u.a.	unter anderem
u.U.	unter Umständen
UART	Universal Asynchronus Receiver Transmitter; Schnittstelle zur seriellen Datenübertragung
UMTS	Universal Mobile Telecommunication System; Mobilfunkstandard
USB	Universal Serial Bus
UTC	Coordinated Universal Time; Koordinierte Weltzeit
WGS 84	Referenzellipsoid bei Positionsbestimmung mittels GPS
z.B.	zum Beispiel
ZOB	Zentraler Omnibus-Bahnhof

Quellcodeverzeichnis

A. Aufgabenstellung

Hochschule für Angewandte Wissenschaften Hamburg
Department Informations- und Elektrotechnik
Prof. Dr.-Ing. Karl-Ragmar Riemschneider 16. Februar 2015

Datenlogger für Elektrobusse mit Mikrocontrollersteuerung, Inertialmesssystem sowie GPS- und GSM-Modulen

Motivation

In der Forschungsgruppe Batteriesensoren (BATSEN) an der HAW Hamburg werden Batteriesysteme in verschiedenen Einsatzbereichen untersucht. Aktuell werden in Hamburg Elektrobusse im Stadtverkehr erprobt. Bei den Lade- und Lastverläufen für die Batteriesysteme sind beim Busbetrieb zahlreiche Besonderheiten relevant. Für diese Besonderheiten wird eine detaillierte Datenbasis benötigt.
Mit messtechnisch erfassten Daten-Grundlagen sollen Lade- und Entladeprofile für Batterietests erstellt werden. Außerdem sollen betriebliche Analysen und Optimierungen (Fahrpläne, Ladezeiten, Planung der Ladestationen u.a.) bei Projektpartnern unterstützt werden.

Aufgabe

Herr Felix Groth erhält die Aufgabe ein Datenlogger-System zu entwickeln, dass

1. im Rahmen der Erprobung der Elektro- und Hybridbusse im Hamburger Verkehrsverbund verwendet kann
2. die Positions- und Bewegungsdaten der Fahrzeuge als Grundfunktion erfasst
3. die Rohdaten für energetische Analysen (z.B. Rekuperation) liefern kann
4. für die Aufzeichung von Batteriemessdaten vorbereitet ist

Das Datenlogger-System soll auf einem ARM-Cortex-M4-Prozessor basieren und flexibel die Ergänzung von Sensor- und Kommunikationsmodulen unterstützen. Das System hat prototypischen Charakter, daher steht zunächst der Erfahrungsgewinnn für die Systemauslegung und nicht die Aufwandsminimierung im Vordergrund. Die Anforderungen sind zunächst durch Recherche zu analysieren und für die Komponetenauswahl zu spezifizieren. Durch Voruntersuchung im Busbetrieb sollen Rahmenbedingungen festgestellt und bewertet werden.
Der Datenlogger ist für die praktische Nutzung im Rahmen von Forschungsprojekten zu Elektromobilität vorzubereiten und daher möglichst offen für spätere Anpassungen und Erweiterungen zu konzipieren.

Gliederung

Die Aufgabe der Bachelorarbeit gliedert sich wie folgt:

1) **Vorarbeiten und Analyse der Anforderungen**
   - Einarbeitung und Recherche zu Elektrobussen und deren Betriebsabläufen
   - Erarbeitung von Funktionsgrundlagen von GPS- und Inertialmesssystemen

- Ziel-Spezifikation des Fraunhofer-Institut IVI Dresden
- Analyse von Vorarbeiten, Aufwandabschätzungen

2) **Konzeption des Datenloggers**

- Modulare Systemauslegung mit Modulgrenzen, Funktionszuweisungen
- Komponentenauswahl: GPS-, GSM- und IMU-Module, Temperatursensor
- Zeitverhalten, Speicherbedarf, Datenraten usw.
- Grobabschätzung des Energiebedarfs, Konzeption der Versorgung
- Konstruktive Festlegungen, ggfs. Aufteilung in Teilsysteme und Platinen
- Bedienkonzept mit einer optionalen Erweiterung mit einem Display
- Optionales Erweiterungskonzept für Batteriedaten

3) **Hardwareentwurf**

Schaltungsentwurf für Platinen mit folgenden Modulen:

- GPS-Modul, GSM-Modul, Inertial-Modul (IMU), SD-Karte
- Versorgungschaltung, inkl. Eingangschutz und USB-Akkumlator
- optional: Display, Funkschnittstelle zur Batteriedatenerfassung
- Schnittstellen-Definition, Pinbelegungen
- Platinenentwurf, Bestückung
- Inbetriebnahme mit Testplan, Modul- und Gesamtfunktionstest

4) **Softwareentwurf und Implementierung**

- Software für Modul-, Schnittstellen- und Gesamtfunktionstest
- Entwicklung der Softwaregrundstruktur und Partitionierung
- Entwicklung der zeitlichen Funktionssteuerung auf Basis unterschiedlicher Systemzeit-Quellen (RTC, GPS, Timerfunktion)
- Implementierung der Schnittstellen, Datenstrukturen und Datenformate

5) **Erprobung und Datenerfassung**

- Laborerprobung, Kalibrierungen, Fehlerkorrekturen, ggf. Modifikationen
- Erprobung im Busbetrieb
- Messplanung, Durchführung und Dokumentation von Versuchsreihen

6) **Auswertung und Bewertung**

- Exemplarische Auswertungen der Versuchergebnisse, Bewertung der Messreihen
- Bewertung der erzielten Ergebnisse in Hinblick auf die Anforderungen
- Analyse von Verbesserungs- und Optimierungspotential, Vorschläge für weitere Arbeiten
- Bewertung bezüglich der Anwendung und der Forschungszielstellung

Dokumentation

Die Vorarbeiten und die kommerziellen Unterlagen sind zielgerichtet zu recherchieren. Die gewählten Lösungen sind gut nachvollziehbar zu dokumentieren. Die gesetzten Rahmenbedingungen und wesentlichen Entwurfsentscheidungen sollen beschrieben werden. Die Ergebnisse sind systematisch zu erfassen und grafisch auszuwerten.

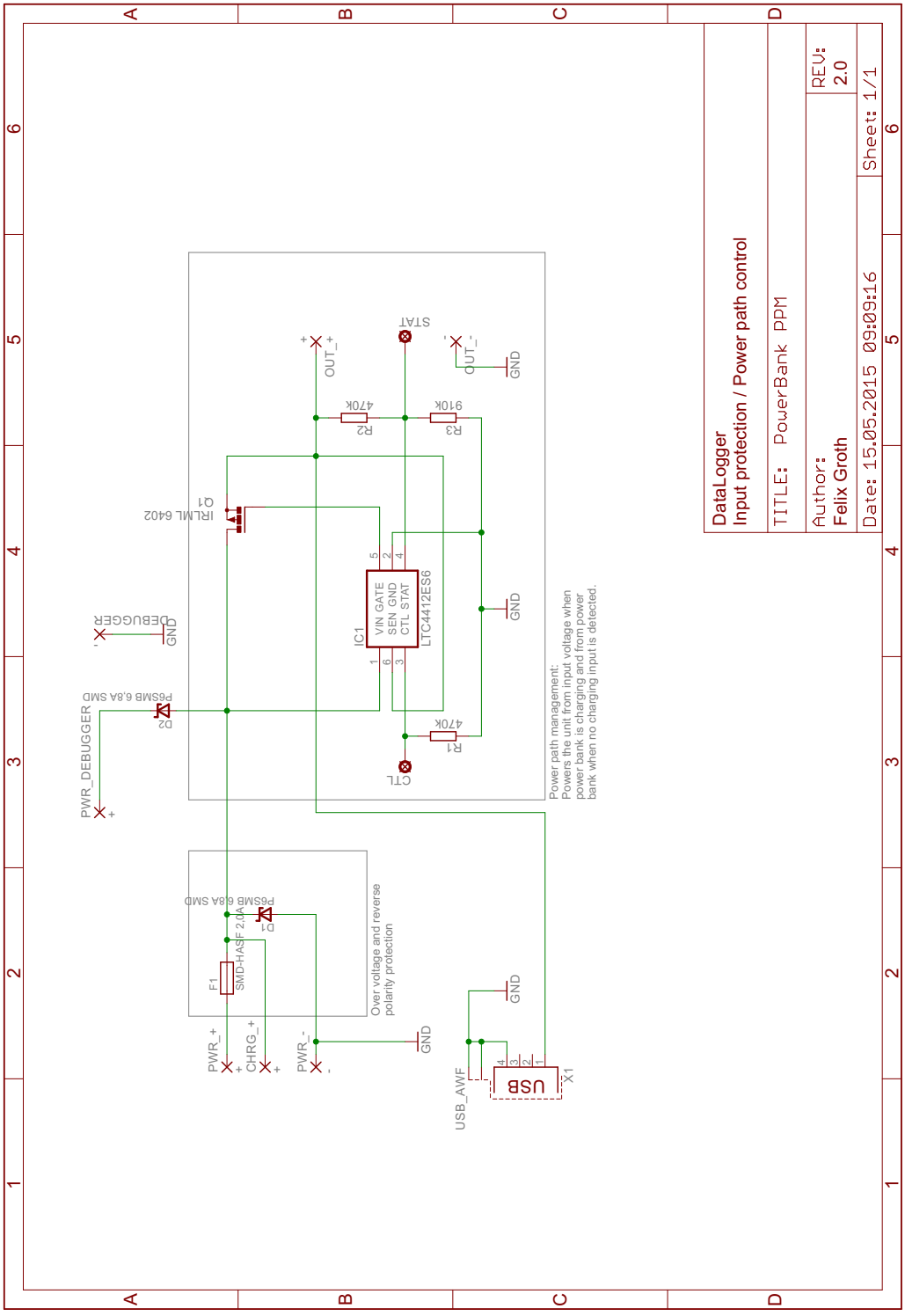
B. Schaltpläne

B.1. Pinbelegung

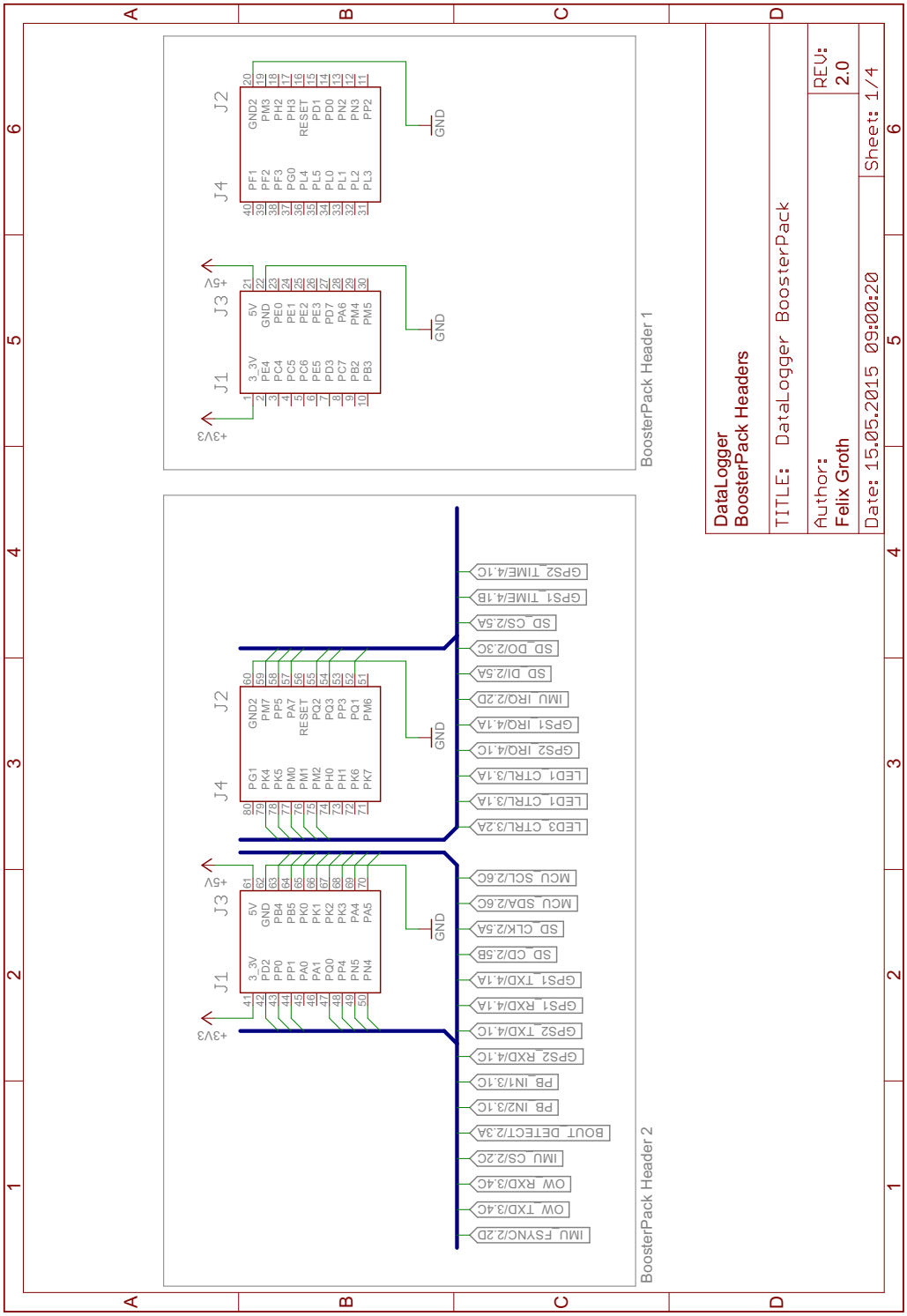
Header	Pin	GPIO	Funktion
A2	2	PD2	Inertialmesssystem FSYNC
A2	3	PP0	1-Wire RX
A2	4	PP1	1-Wire TX
A2	5	PA0	GSM UART RX
A2	6	PA1	GSM UART TX
A2	7	PQ0	MicroSD SPI CLK
A2	8	PP4	MicroSD Card Detect
A2	9	PN5	Inertialmesssystem I^2C SCL
A2	10	PN4	Inertialmesssystem I^2C SDA
Header	Pin	GPIO	Funktion
B2	3	PB4	Inertialmessystem CS
B2	4	PB5	Taster 1
B2	5	PK0	GNSS 2 UART RX
B2	6	PK1	GNSS 2 UART TX
B2	7	PK2	Taster 2
B2	8	PK3	LTC4412 STAT
B2	9	PA4	GNSS 1 UART RX
B2	10	PA5	GNSS 1 UART TX
Header	Pin	GPIO	Funktion
C2	1	PG1	GSM ON
C2	2	PK4	GNSS 1
C2	3	PK5	GNSS 2
C2	4	PM0	Inertialmesssystem IRQ
C2	5	PM1	GNSS 1 Timepulse
C2	6	PM2	GNSS 2 Timepulse
C2	7	PH0	GSM UART RTS
C2	8	PH1	GSM UART CTS
C2	9	PK6	GSM RESET
C2	10	PK7	GSM UART RI
Header	Pin	GPIO	Funktion
D2	2	PM7	LED 1
D2	3	PP5	LED 2
D2	4	PA7	LED 3
D2	6	PQ2	MicroSD MOSI
D2	7	PQ3	MicroSD MISO
D2	8	PP3	GSM UART DCD
D2	9	PQ1	MicroSD CS
D2	10	PM6	GSM PWRKEY
Header	Pin	GPIO	Funktion
A1 - D1	-	-	LCD-Touchscreen
-	-	PA2	LTC4412 CTL
-	-	PA3	1-Wire Pull-Up

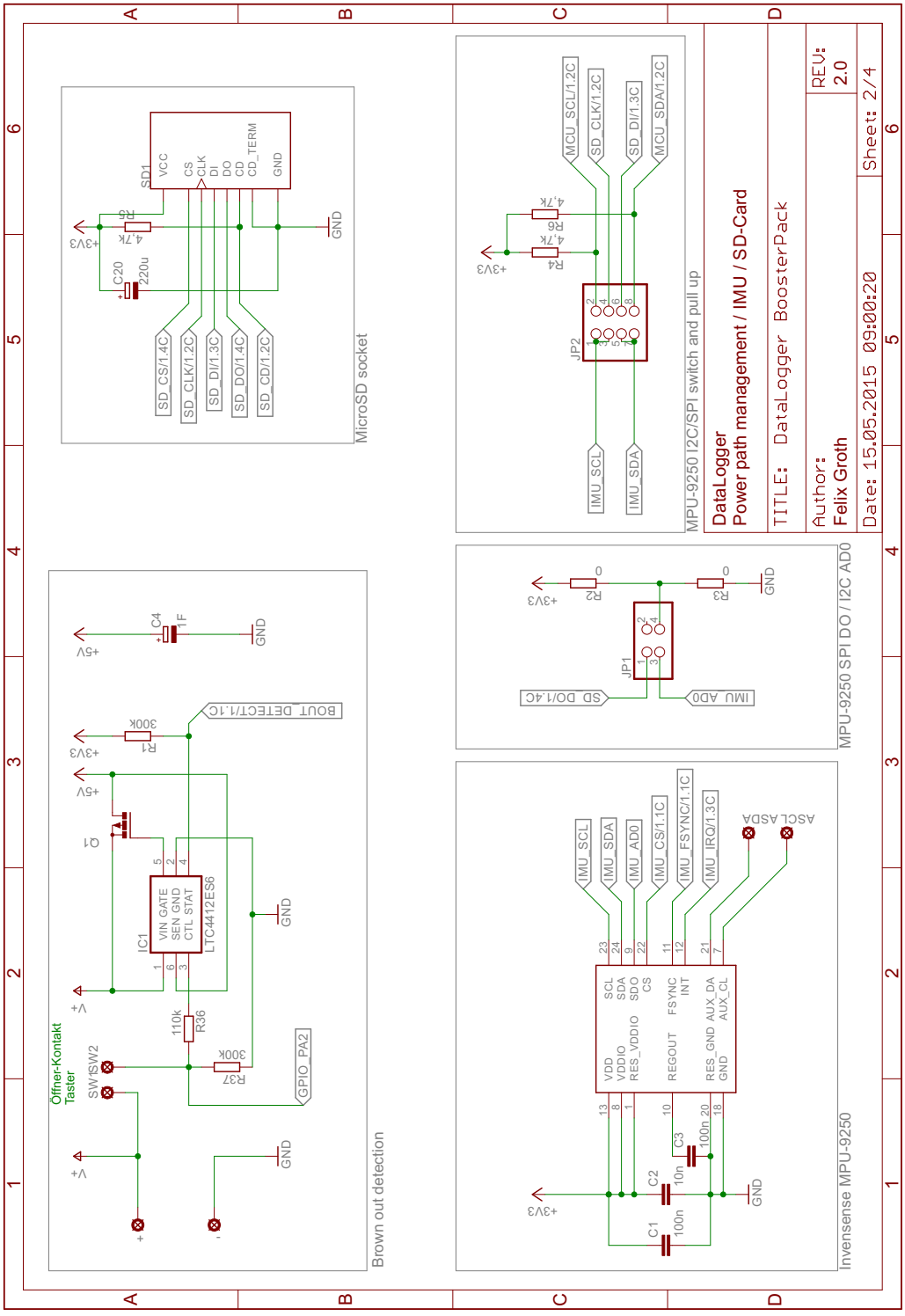
Tabelle B.1.: Belegung der Anschlüsse an Mikrocontroller und Evaluationsboard

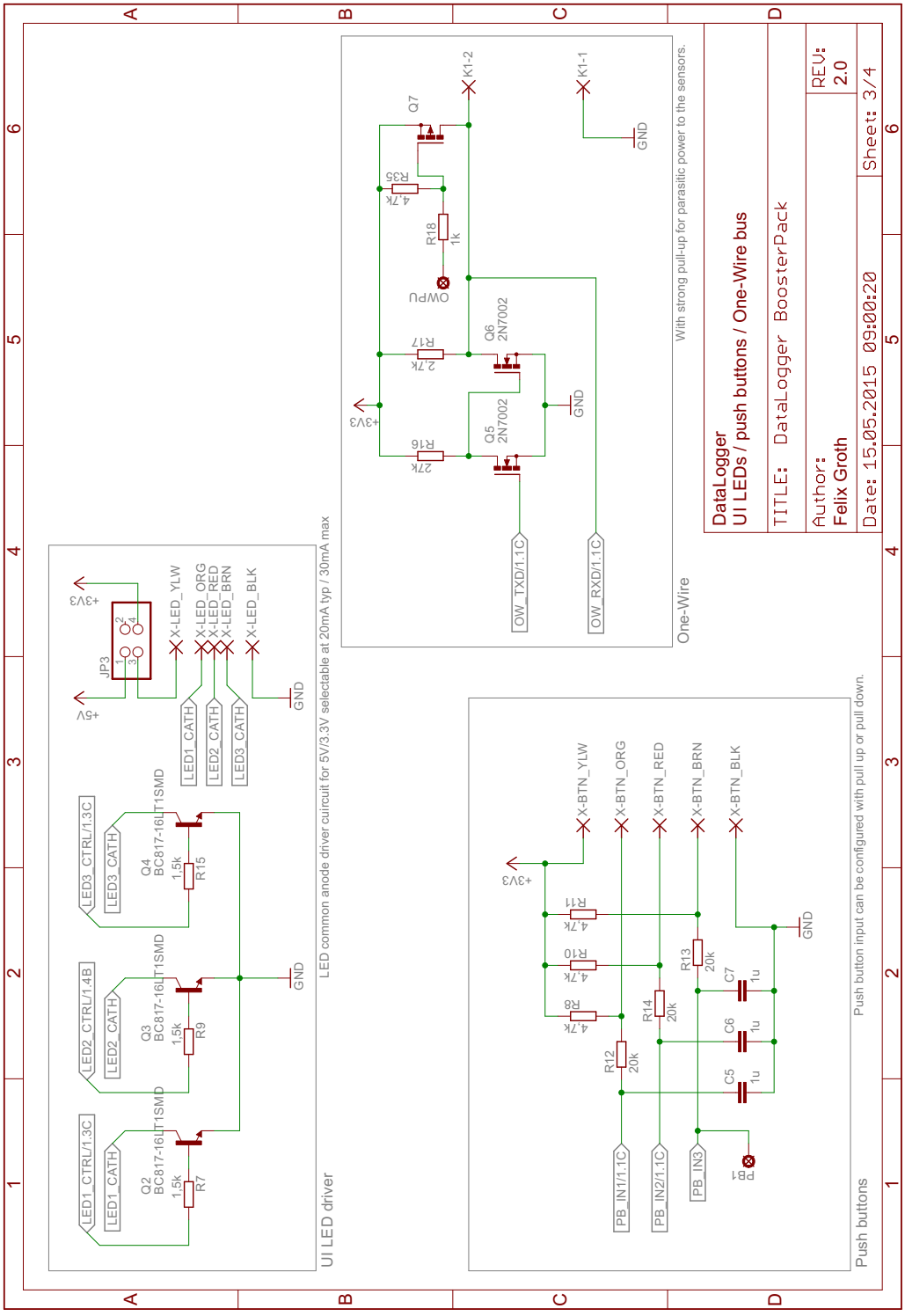
B.2. Spannungsversorgung und Eingangsschutz

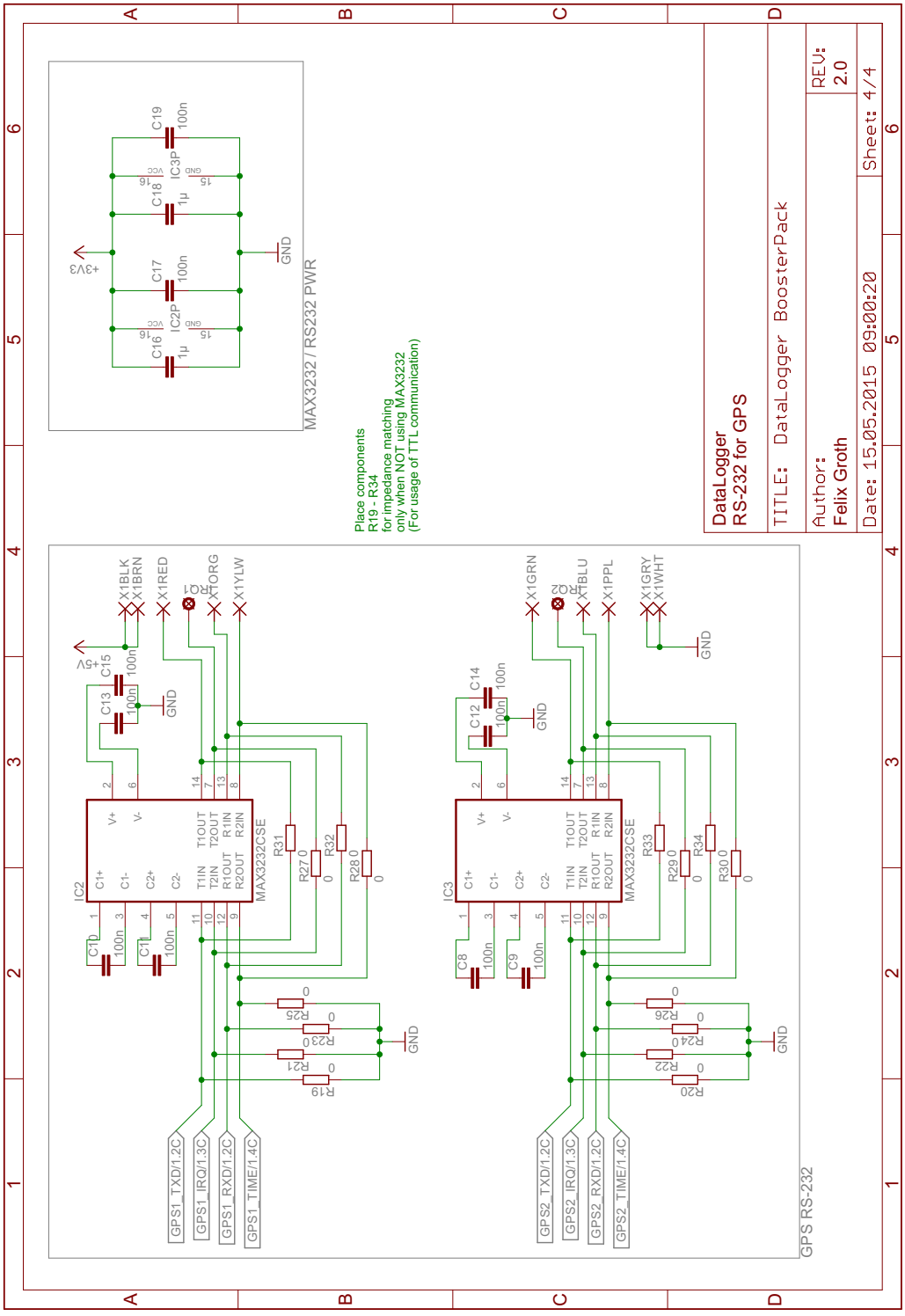

DataLogger
Input protection / Power path control
TITLE: PowerBank PDM
Author:
Felix Groth
REV:
2.0
Date: 15.05.2015 09:09:16
Sheet: 1/1
Power path management:
Powers the unit from input voltage when
power bank is charging and from power
bank when no charging input is detected.
Over voltage and reverse
polarity protection
IC1
LTC4412ES6
VIN GATE
SEN GND
CTL STAT
Q1
IRLML 6402
D1
D2
P6SMB 6.8A SMD
F1
SMD-HASF 2.0A
R1
R2
R3
470k
470k
910k
STAT
CTL
OUT_+
OUT_-
GND
PWR_+
PWR_-
CHRG_+
PWR_DEBUGGER
DEBUGGER
USB_AWF
USB
X1

B.3. Datenlogger-Erweiterungsplatine

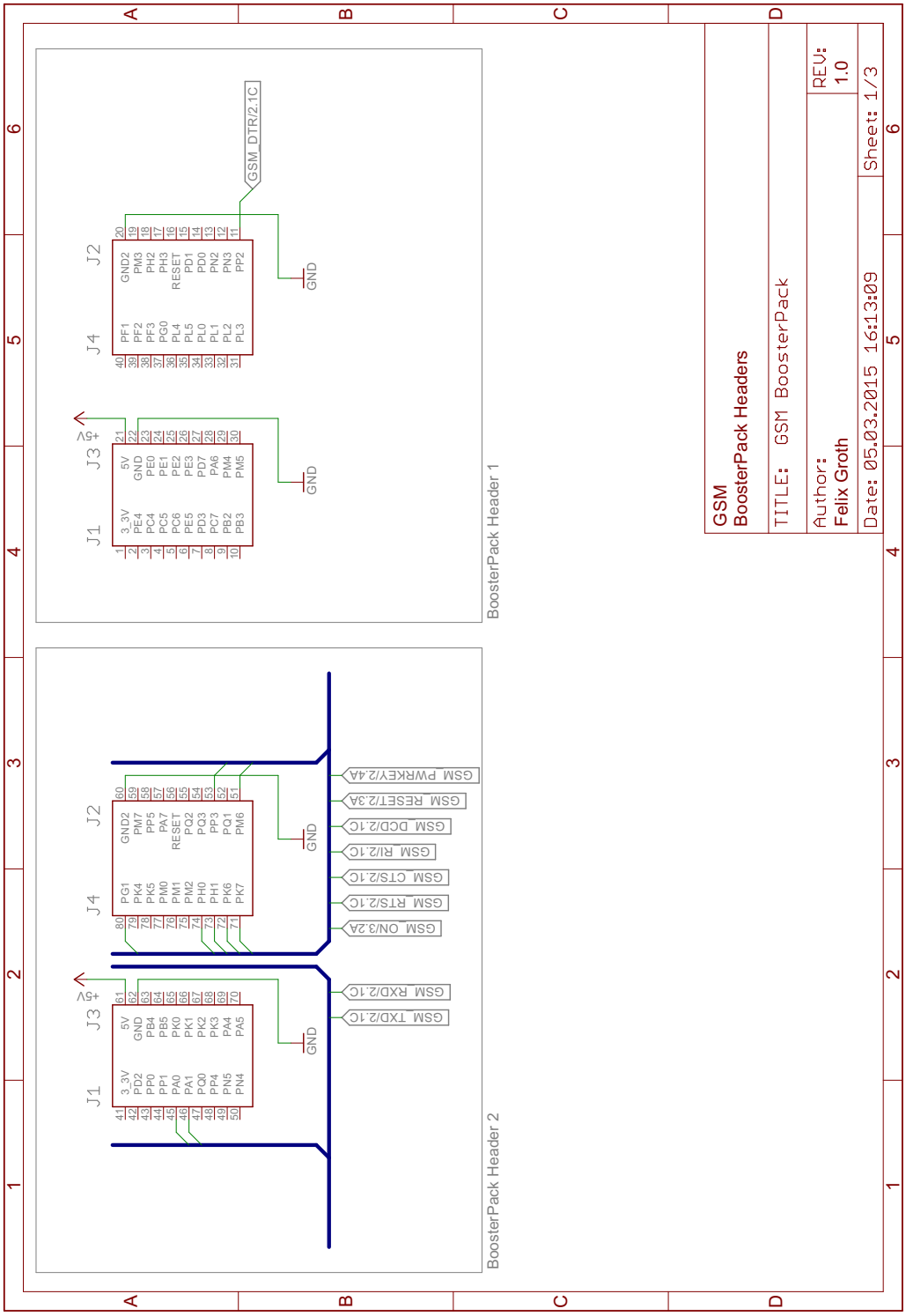

DataLogger
BoosterPack Headers
TITLE: DataLogger BoosterPack
Author:
Felix Groth
REV:
2.0
Date: 15.05.2015 09:00:20
Sheet: 1/4
BoosterPack Header 1
BoosterPack Header 2
J1
J2
J3
J4
+3V3
+5V
GND
3_3V
5V
GND
PD2
PP0
PP1
PA0
PA1
PQ0
PP4
PN5
PN4
PB4
PB5
PK0
PK1
PK2
PK3
PA4
PA5
PG1
PK4
PK5
PM0
PM1
PM2
PH0
PH1
PK6
PK7
GND2
PM7
PP5
PA7
RESET
PQ2
PQ3
PP3
PQ1
PM6
PF1
PF2
PF3
PG0
PL4
PL5
PL0
PL1
PL2
PL3
PP2
PM3
PH2
PH3
RESET
PD1
PD0
PN2
PN3
PE0
PE1
PE2
PE3
PD7
PA6
PM4
PM5
PE4
PC4
PC5
PC6
PE5
PD3
PC7
PB2
PB3
GPS2_TIME/4.1C
GPS1_TIME/4.1B
SD_CS/2.5A
SD_DO/2.3C
SD_DI/2.5A
IMU_IRQ/2.2D
GPS1_IRQ/4.1A
GPS2_IRQ/4.1C
LED1_CTRL/3.1A
LED1_CTRL/3.1A
LED3_CTRL/3.2A
MCU_SCL/2.6C
MCU_SDA/2.6C
SD_CLK/2.5A
SD_CD/2.5B
GPS1_TXD/4.1A
GPS1_RXD/4.1A
GPS2_TXD/4.1C
GPS2_RXD/4.1C
PB_IN1/3.1C
PB_IN2/3.1C
BOUT_DETECT/2.3A
IMU_CS/2.2C
OW_RXD/3.4C
OW_TXD/3.4C
IMU_FSYNC/2.2D

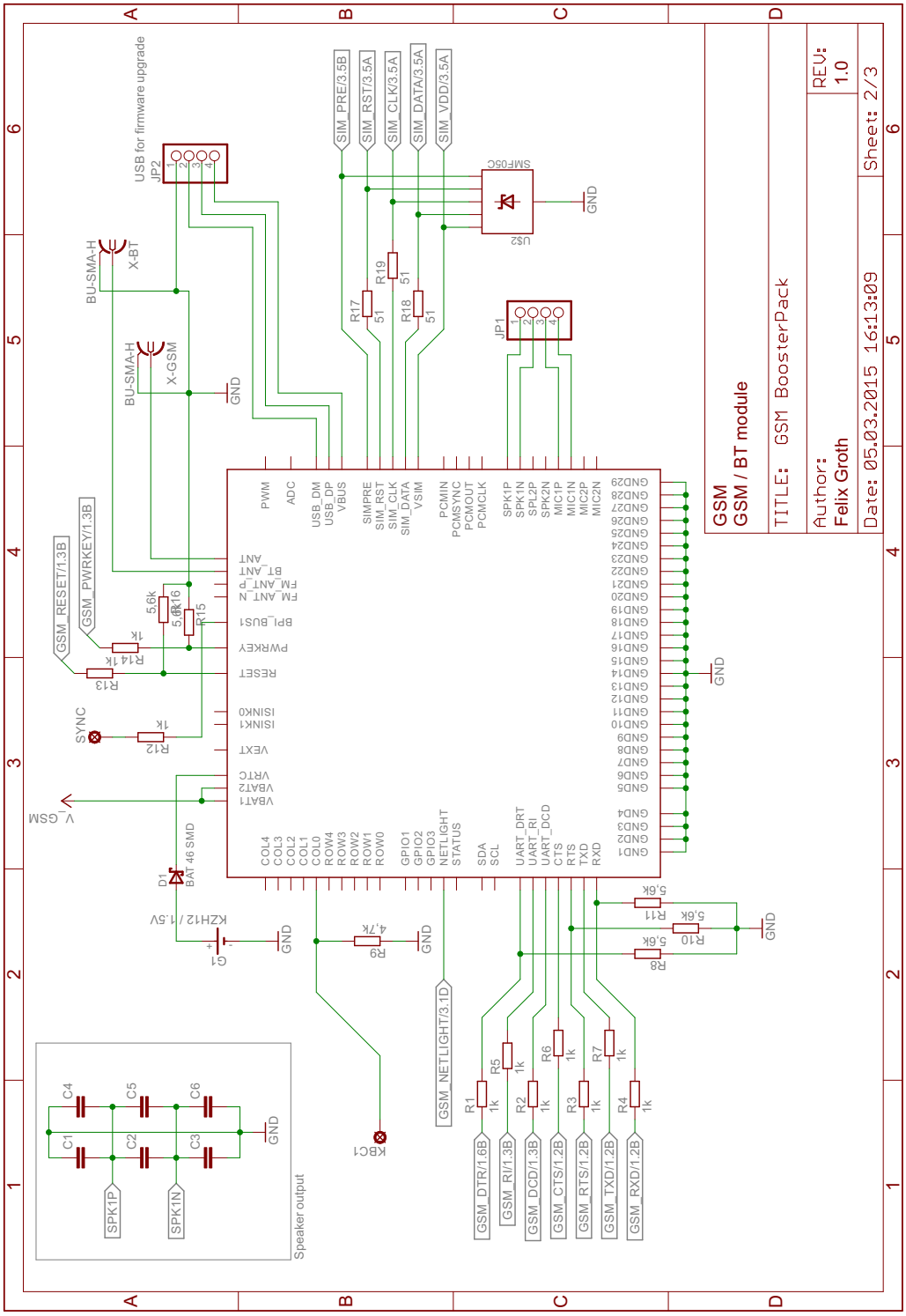

DataLogger
Power path management / IMU / SD-Card
TITLE: DataLogger BoosterPack
Author: Felix Groth
REV: 2.0
Date: 15.05.2015 09:00:20
Sheet: 2/4
MicroSD socket
MPU-9250 I2C/SPI switch and pull up
MPU-9250 SPI DO / I2C AD0
Invensense MPU-9250
Brown out detection
SD1
JP2
JP1
IC1
LTC4412ES6
SW1
SW2
Öffner-Kontakt
Taster
MCU_SCL/1.2C
SD_CLK/1.2C
SD_DI/1.3C
MCU_SDA/1.2C
IMU_SCL
IMU_SDA
SD_CS/1.4C
SD_DO/1.4C
SD_CD/1.2C
IMU_AD0
IMU_FSYNC/1.1C
IMU_IRQ/1.3C
BOUT_DETECT/1.1C
GPIO_PA2
+3V3
+5V
GND
V+

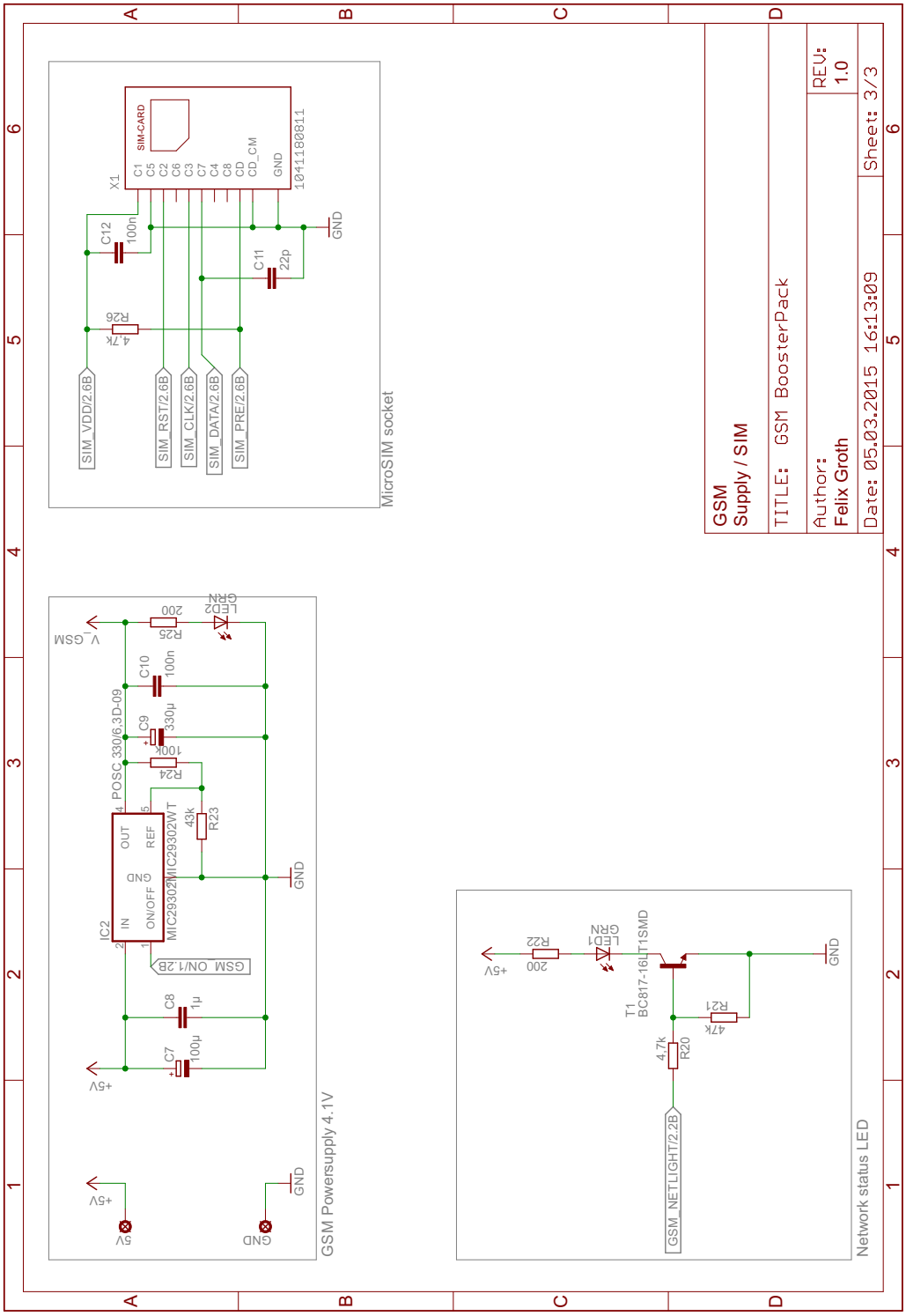

DataLogger
UI LEDs / push buttons / One-Wire bus
TITLE: DataLogger BoosterPack
Author:
Felix Groth
REV: 2.0
Date: 15.05.2015 09:00:20
Sheet: 3/4
UI LED driver
LED common anode driver cuircuit for 5V/3.3V selectable at 20mA typ / 30mA max
JP3
+5V
+3V3
GND
X-LED_YLW
X-LED_ORG
X-LED_RED
X-LED_BRN
X-LED_BLK
LED1_CATH
LED2_CATH
LED3_CATH
LED1_CTRL/1.3C
LED2_CTRL/1.4B
LED3_CTRL/1.3C
Q2 BC817-16LT1SMD
Q3 BC817-16LT1SMD
Q4 BC817-16LT1SMD
1.5k
R7
R9
R15
One-Wire
With strong pull-up for parasitic power to the sensors.
OW_TXD/1.1C
OW_RXD/1.1C
Q5 2N7002
Q6 2N7002
Q7
R16 27k
R17 2.7k
R18 1k
R35 4.7k
OWPU
K1-1
K1-2
Push buttons
Push button input can be configured with pull up or pull down.
X-BTN_YLW
X-BTN_ORG
X-BTN_RED
X-BTN_BRN
X-BTN_BLK
PB_IN1/1.1C
PB_IN2/1.1C
PB_IN3
R8 4.7k
R10 4.7k
R11 4.7k
R12 20k
R13 20k
R14 20k
C5 1u
C6 1u
C7 1u
PB1


DataLogger
RS-232 for GPS
TITLE: DataLogger BoosterPack
Author:
Felix Groth
REV: 2.0
Date: 15.05.2015 09:00:20
Sheet: 4/4
Place components
R19 - R34
for impedance matching
only when NOT using MAX3232
(For usage of TTL communication)
MAX3232 / RS232 PWR
GPS RS-232
MAX3232CSE
GPS1_TXD/1.2C
GPS1_IRQ/1.3C
GPS1_RXD/1.2C
GPS1_TIME/1.4C
GPS2_TXD/1.2C
GPS2_IRQ/1.3C
GPS2_RXD/1.2C
GPS2_TIME/1.4C
GND
+3V3
+5V

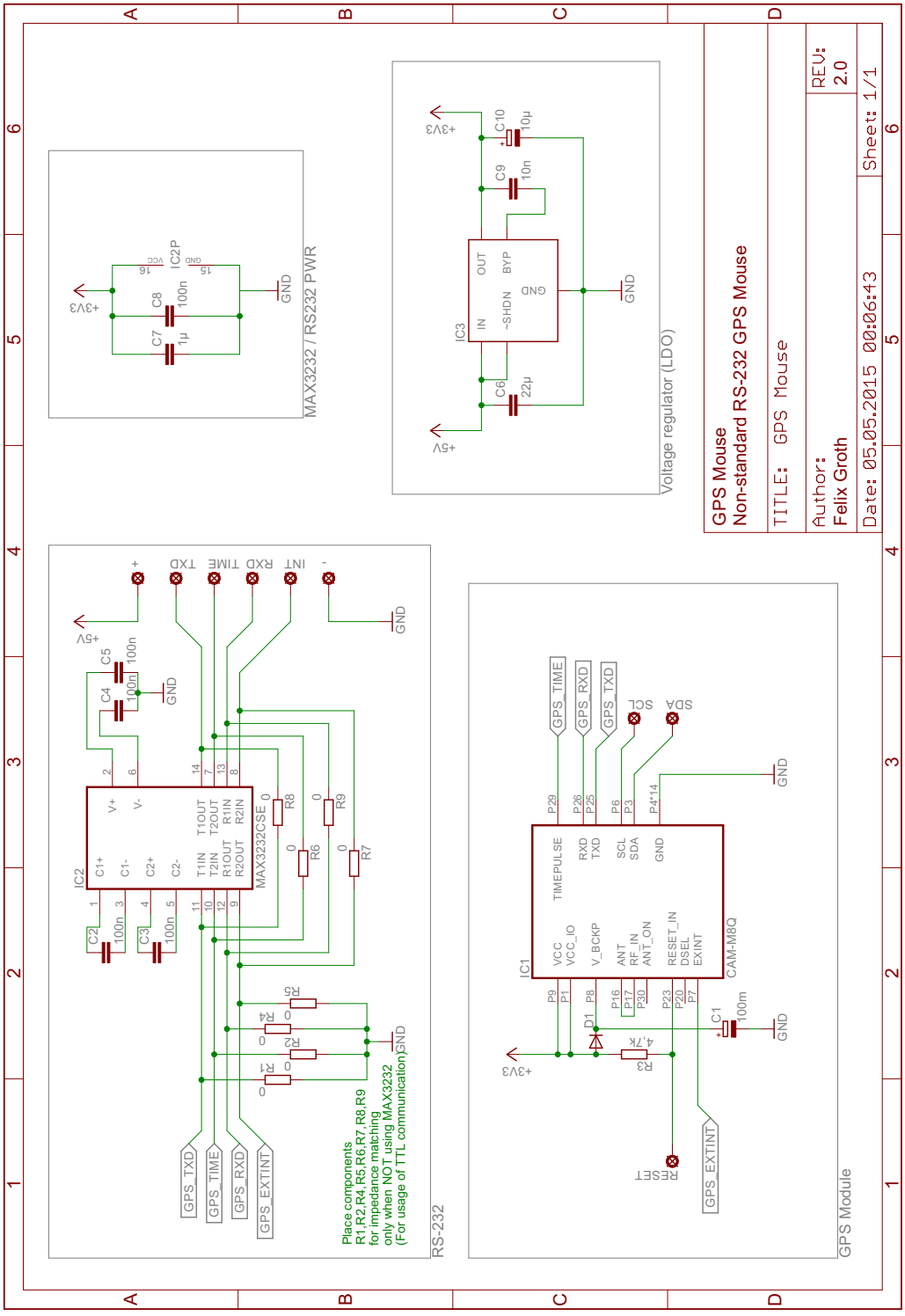
B.4. GSM-/GPRS-Erweiterungsplatine


GSM
BoosterPack Headers
TITLE: GSM BoosterPack
Author:
Felix Groth
REV:
1.0
Date: 05.03.2015 16:13:09
Sheet: 1/3
BoosterPack Header 1
BoosterPack Header 2
GSM_DTR/2.1C
GSM_PWRKEY/2.4A
GSM_RESET/2.3A
GSM_DCD/2.1C
GSM_RI/2.1C
GSM_CTS/2.1C
GSM_RTS/2.1C
GSM_ON/3.2A
GSM_RXD/2.1C
GSM_TXD/2.1C
GND
5V


GSM
GSM / BT module
TITLE: GSM BoosterPack
Author: Felix Groth
Date: 05.03.2015 16:13:09
REV: 1.0
Sheet: 2/3
USB for firmware upgrade
JP2
SIM_PRE/3.5B
SIM_RST/3.5A
SIM_CLK/3.5A
SIM_DATA/3.5A
SIM_VDD/3.5A
SM712
U$2
GND
BU-SMA-H
X-BT
X-GSM
JP1
R17 51
R19 51
R18 51
GSM_RESET/1.3B
GSM_PWRKEY/1.3B
R16 5,6k
R15 5,6k
R14 1k
R13 1k
SYNC
R12 1k
V_GSM
D1 BAT 46 SMD
KZH12 / 1,5V
G1
R9 4,7k
R11 5,6k
R10 5,6k
R8 5,6k
GSM_NETLIGHT/3.1D
KBC1
R1 1k
R5 1k
R2 1k
R6 1k
R3 1k
R7 1k
R4 1k
GSM_DTR/1.6B
GSM_RI/1.3B
GSM_DCD/1.3B
GSM_CTS/1.2B
GSM_RTS/1.2B
GSM_TXD/1.2B
GSM_RXD/1.2B
C4
C5
C6
C1
C2
C3
SPK1P
SPK1N
Speaker output
PWM
ADC
USB_DM
USB_DP
VBUS
SIMPRE
SIM_RST
SIM_CLK
SIM_DATA
VSIM
PCMIN
PCMSYNC
PCMOUT
PCMCLK
SPK1P
SPK1N
SPL2P
SPK2N
MIC1P
MIC1N
MIC2P
MIC2N
FM_ANT_N
FM_ANT_P
BT_ANT
ANT
BPI_BUS1
PWRKEY
RESET
ISINK0
ISINK1
VEXT
VRTC
VBAT2
VBAT1
COL4
COL3
COL2
COL1
COL0
ROW4
ROW3
ROW2
ROW1
ROW0
GPIO1
GPIO2
GPIO3
NETLIGHT
STATUS
SDA
SCL
UART_DRT
UART_RI
UART_DCD
CTS
RTS
TXD
RXD


GSM
Supply / SIM
TITLE: GSM BoosterPack
Author:
Felix Groth
REV:
1.0
Date: 05.03.2015 16:13:09
Sheet: 3/3
MicroSIM socket
SIM-CARD
X1
C1
C5
C2
C6
C3
C7
C4
C8
CD
CD_CM
GND
10411808811
C12
100n
C11
22p
R26
4.7k
SIM_VDD/2.6B
SIM_RST/2.6B
SIM_CLK/2.6B
SIM_DATA/2.6B
SIM_PRE/2.6B
GSM Powersupply 4.1V
V_GSM
R25
200
LED2
GRN
C10
100n
C9
330µ
POSC 330/6.3D-09
R24
100k
IC2
MIC29302WT
OUT
REF
GND
IN
ON/OFF
MIC29302
43k
R23
GSM_ON/1.2B
C8
1µ
C7
100µ
+5V
GND
Network status LED
R22
200
LED1
GRN
T1
BC817-16LT1SMD
R21
47k
4.7k
R20
GSM_NETLIGHT/2.2B

B.5. GNSS-Empfänger


GPS Mouse
Non-standard RS-232 GPS Mouse
TITLE: GPS Mouse
Author:
Felix Groth
Date: 05.05.2015 00:06:43
REV: 2.0
Sheet: 1/1
MAX3232 / RS232 PWR
Voltage regulator (LDO)
RS-232
GPS Module
Place components
R1,R2,R4,R5,R6,R7,R8,R9
for impedance matching
only when NOT using MAX3232
(For usage of TTL communication)
GPS_TXD
GPS_TIME
GPS_RXD
GPS_EXTINT
RESET
SDA
SCL
GND
+3V3
+5V

C. Messungen und Messergebnisse

C.1. Geradeausfahrt

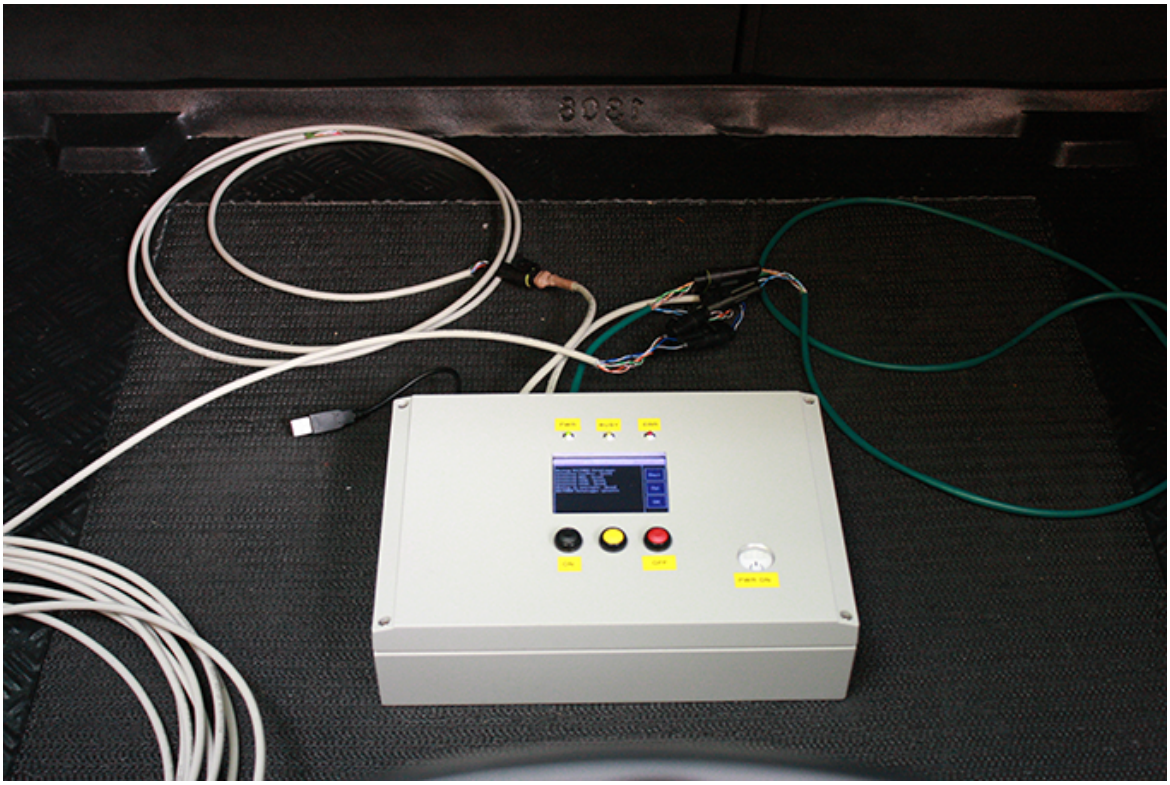


Abbildung C.1.: Einbaulage des Datenloggers während der Messungen im PKW

Abbildung C.2.: Einbau der GNSS-Empfängers während der Messungen im PKW, Beispiel: Rechte Fahrzeugseite

Abbildung C.3.: Fahrtroute der Messfahrt eines PKW in der Geradeausfahrt (Messung-Nr. 1 und 2), Quelle: Google Earth

Abbildung C.4.: Exemplarisch: Abweichung der Positionsbestimmung durch Satellitenortung (Messung-Nr. 1 und 2), Quelle: Google Earth

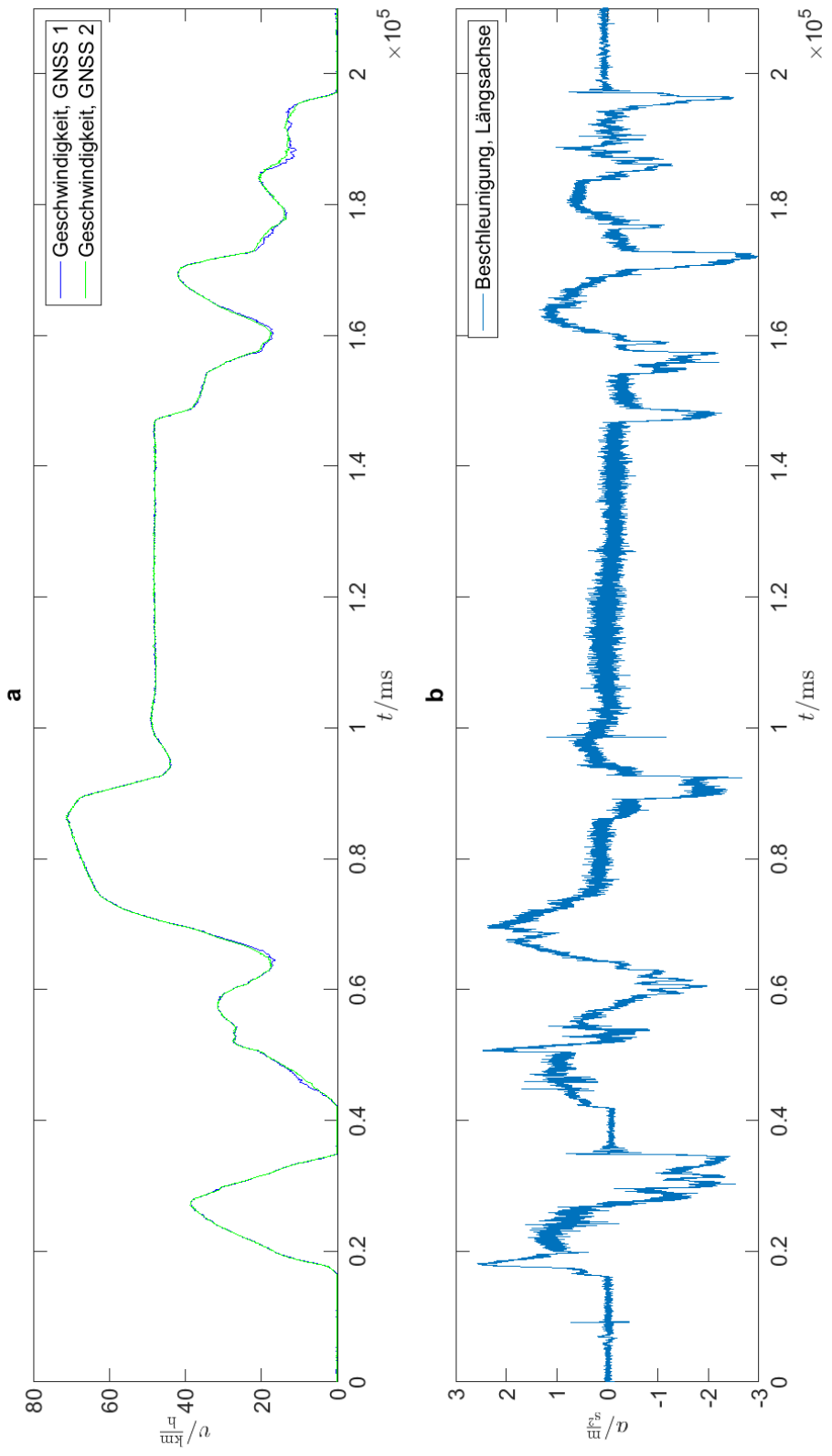


Abbildung C.5.: Messfahrt zu Messung-Nr. 1 und 2: Geschwindigkeit und Beschleunigung bei Geradeausfahrt, **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem

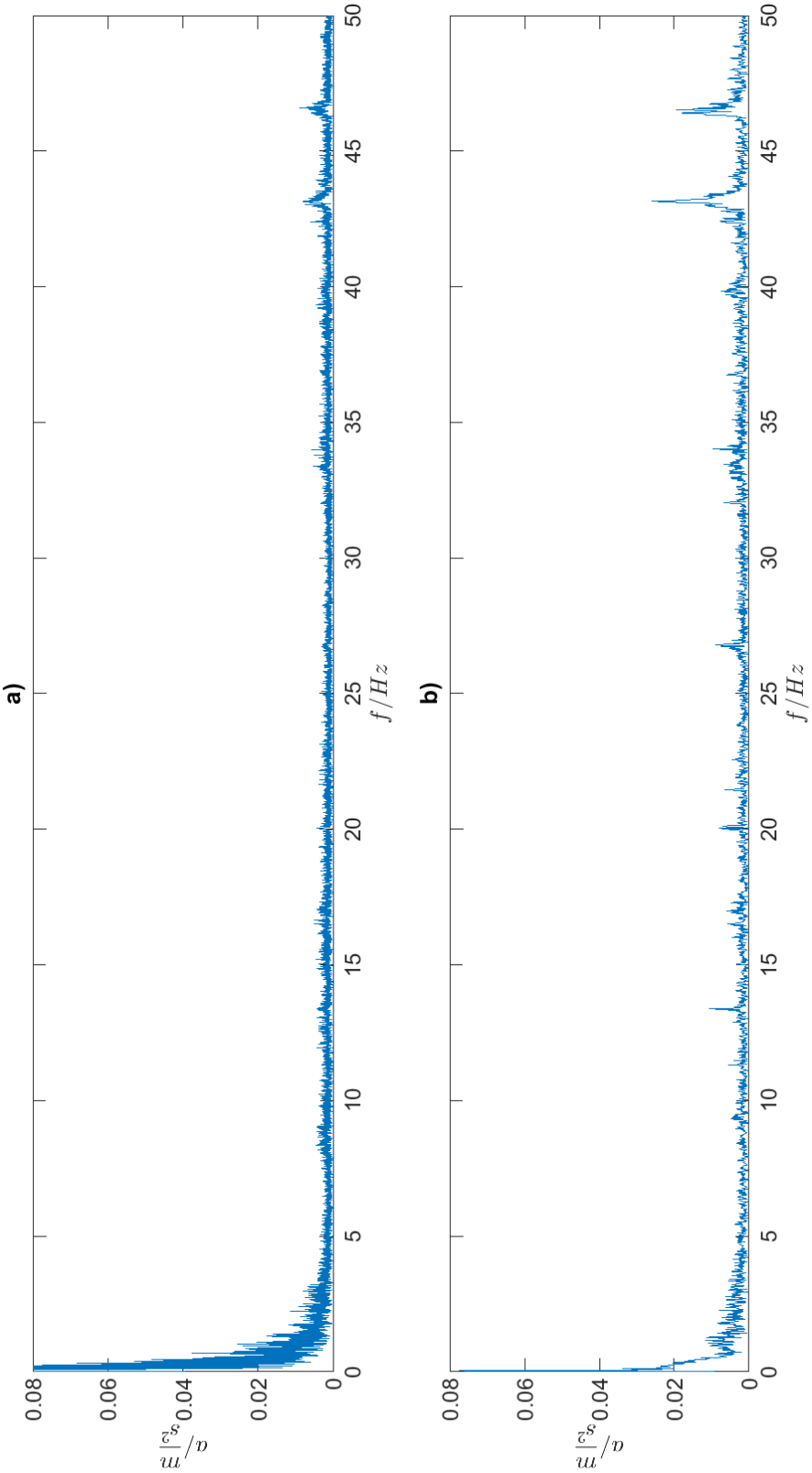


Abbildung C.6.: Amplitudenspektren der Beschleunigung bei Geradeausfahrt, **a)** Amplitudenspektrum der Längsbeschleunigung bei Geradeausfahrt mit Geschwindigkeitsänderung durch den Fahrer, **b)** Amplitudenspektrum der Längsbeschleunigung bei Geradeausfahrt mit konstanter Geschwindigkeit

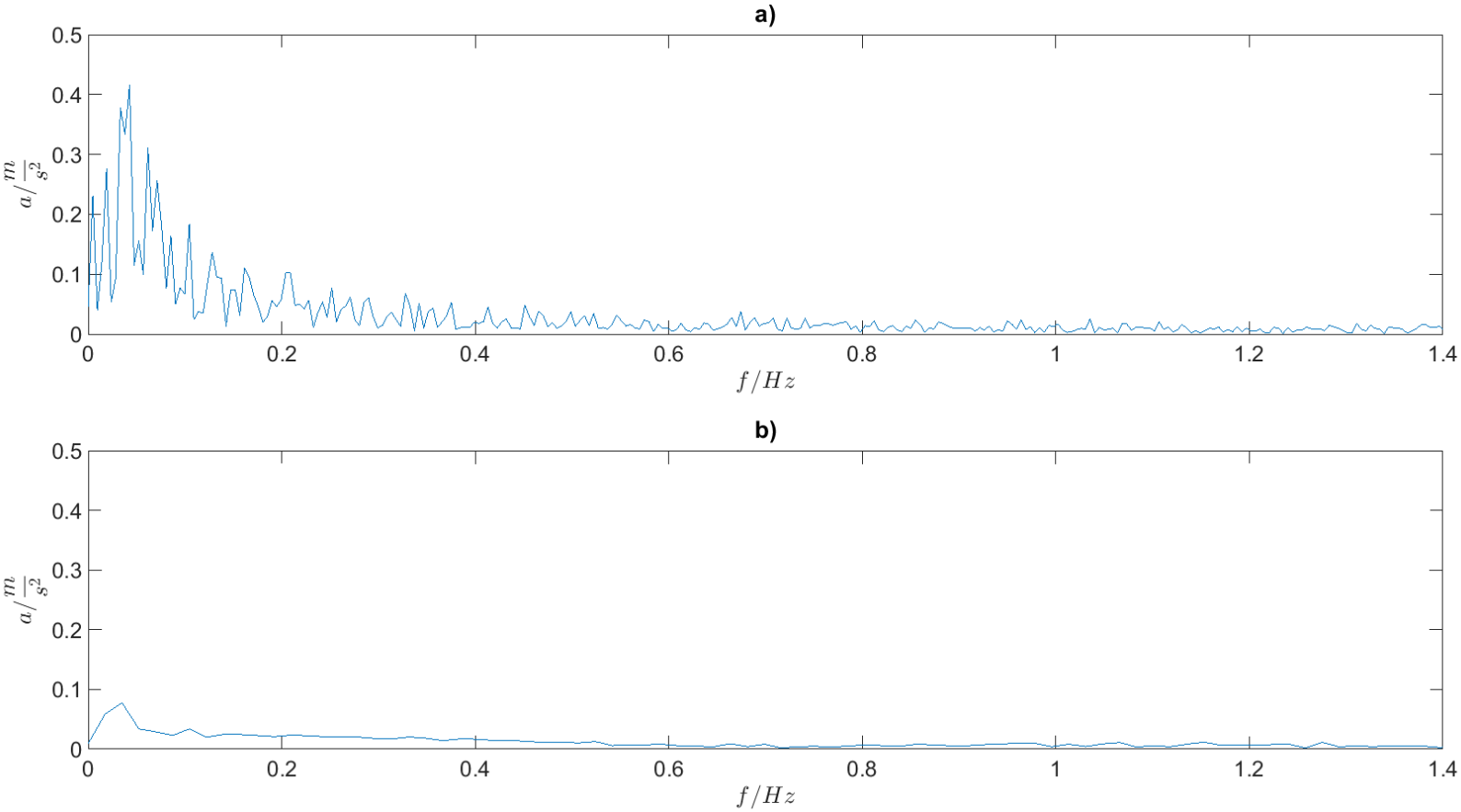


Abbildung C.7.: Amplitudenspektren der Beschleunigung bei Geradeausfahrt im unteren Frequenzbereich, **a)** Amplitudenspektrum der Längsbeschleunigung bei Geradeausfahrt mit Geschwindigkeitsänderung durch den Fahrer, **b)** Amplitudenspektrum der Längsbeschleunigung bei Geradeausfahrt mit konstanter Geschwindigkeit

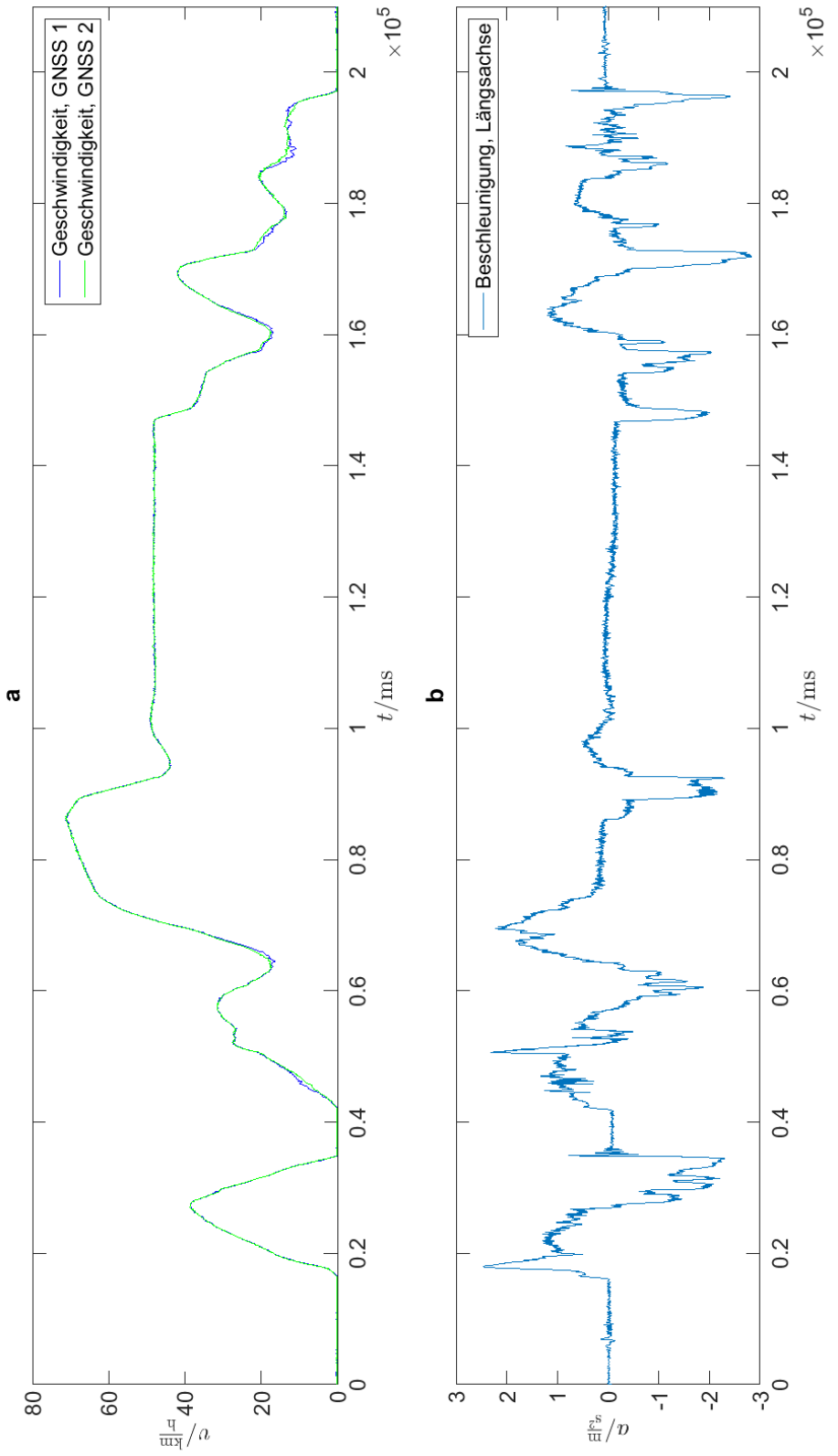


Abbildung C.8.: Messfahrt zu Messung-Nr. 1 und 2: Geschwindigkeit und Beschleunigung ($f_{g,TP} = 10\,\mathrm{Hz}$) bei Geradeausfahrt, **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem, digital tiefpassgefiltert

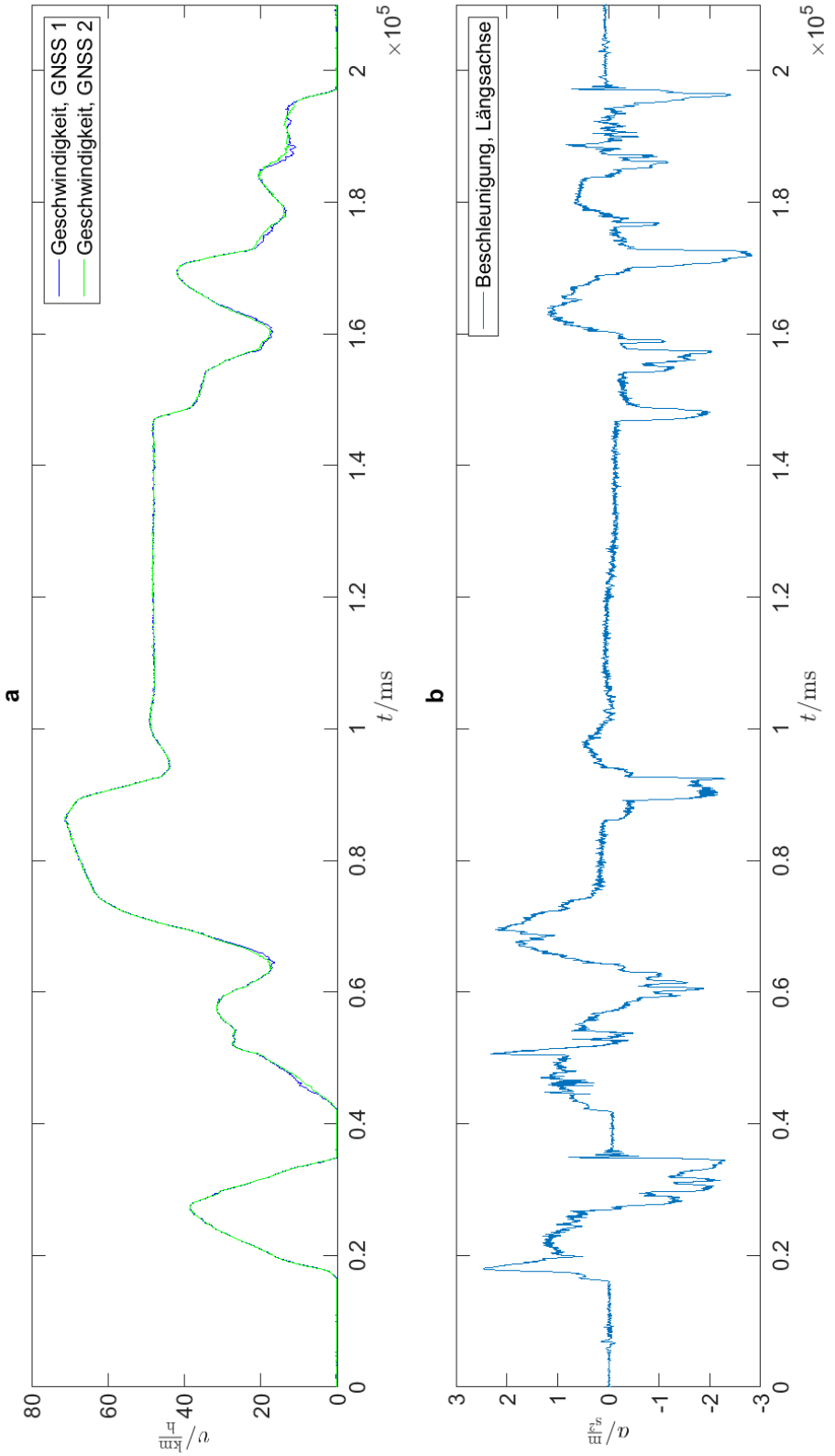


Abbildung C.9.: Messfahrt zu Messung-Nr. 1 und 2: Drehrate um die Gierachse bei Geradeausfahrt

C.2. Kreisfahrt

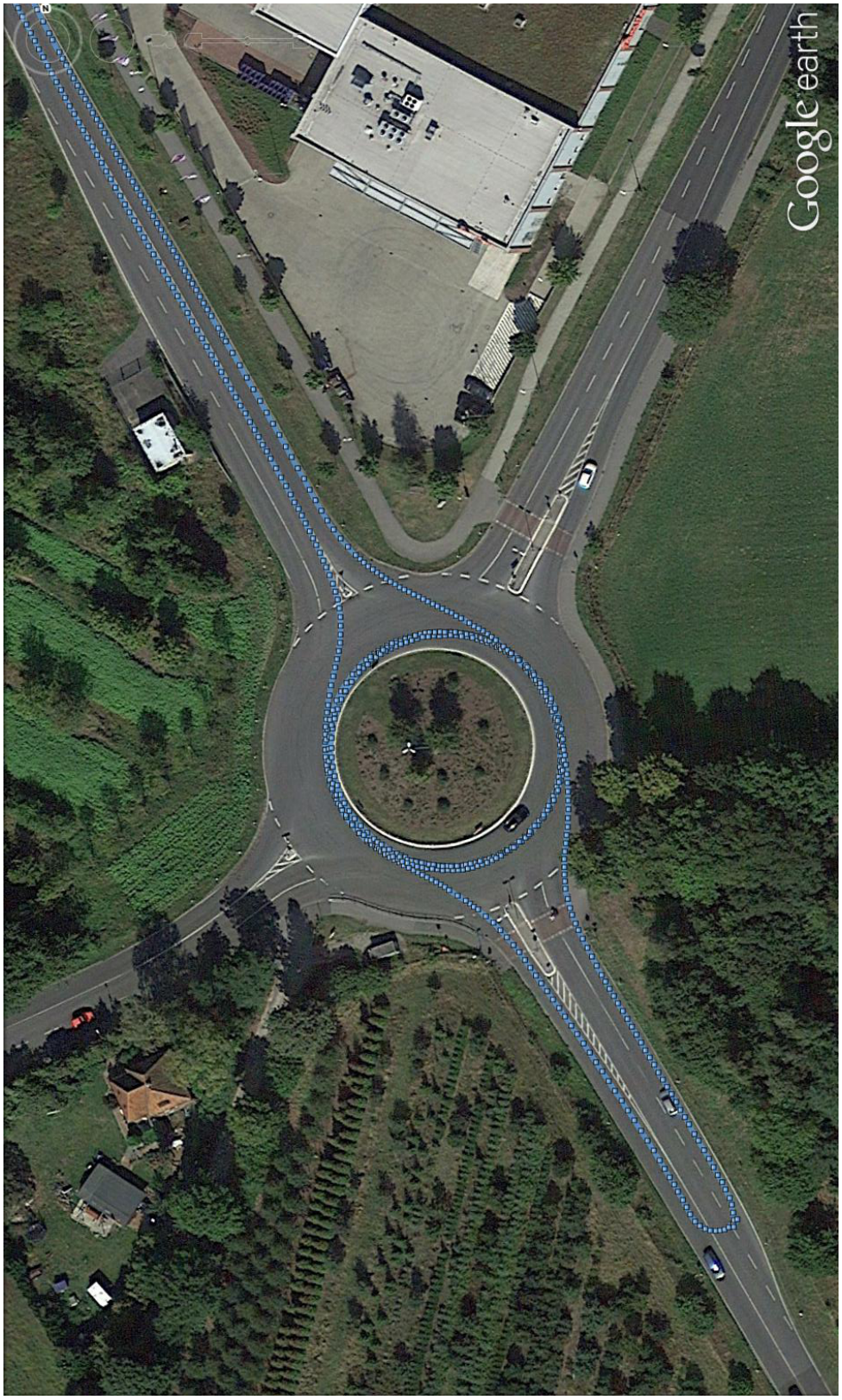


Abbildung C.10.: Fahrtroute der Messfahrt eines PKW in der Kreisfahrt (Messung-Nr. 3 und 4), Quelle: Google Earth

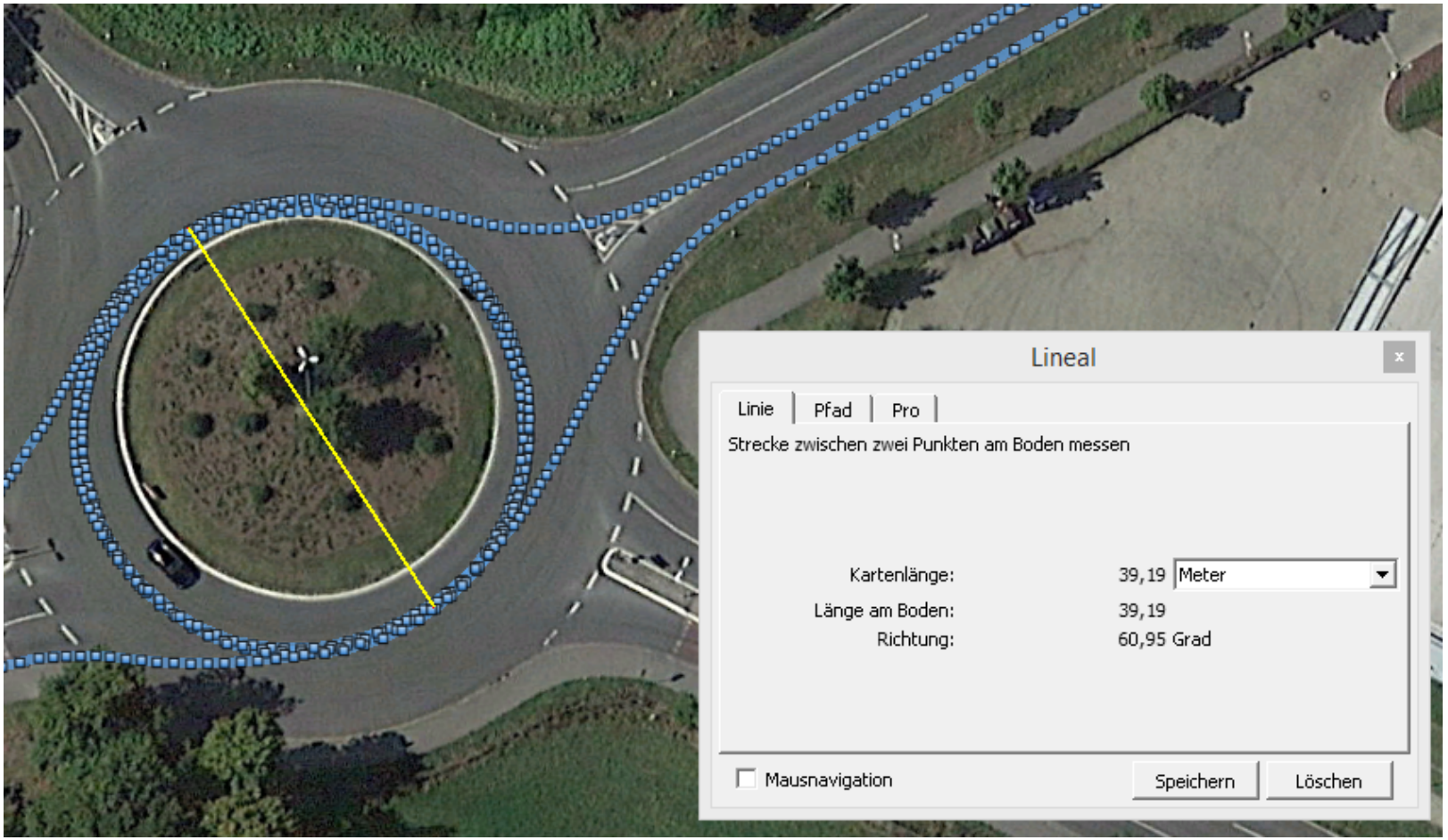


Abbildung C.11.: Messfahrt eines PKW in der Kreisfahrt: Messung des Kreisdurchmessers (Messungen-Nr. 3 und 4), Quelle: Google Earth

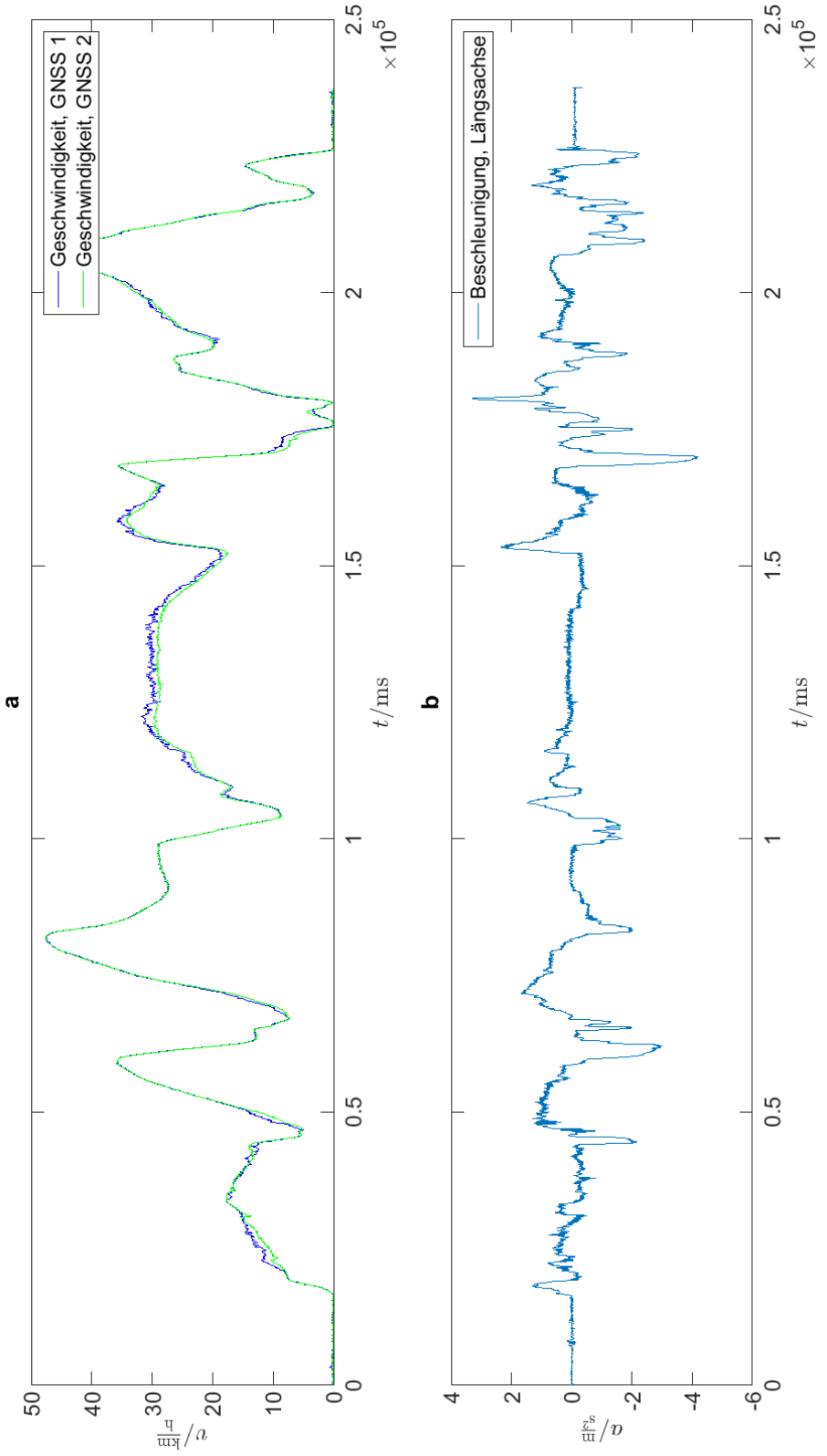


Abbildung C.12.: Messfahrt zu Messung-Nr. 3 und 4: Geschwindigkeit und Beschleunigung bei Kreisfahrt, **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem

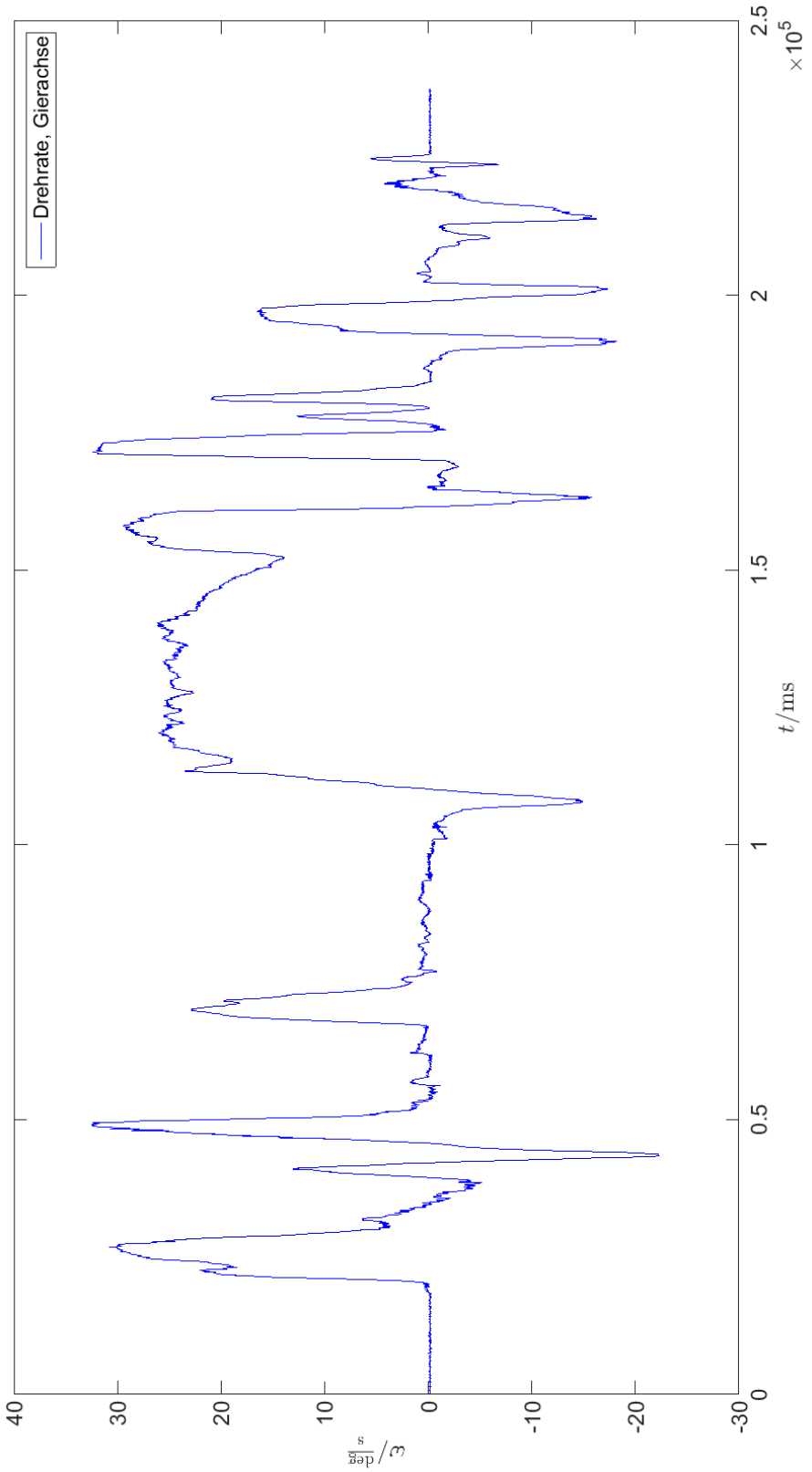


Abbildung C.13.: Messfahrt zu Messung-Nr. 3 und 4: Drehrate um die Gierachse bei Kreisfahrt

C.3. Nahverkehrsbus Linie 109

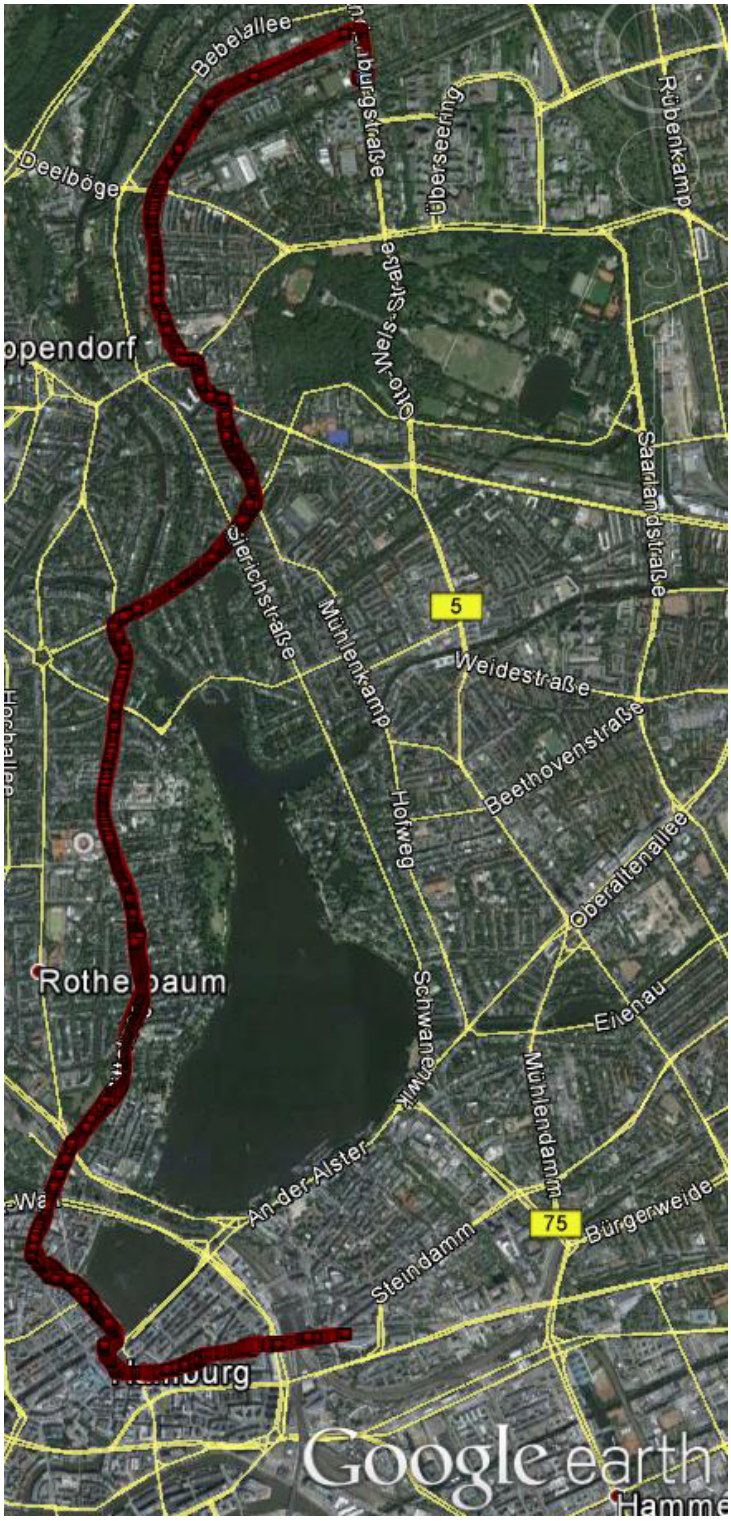


Abbildung C.14.: Fahrtroute zu Messung-Nr. 5 und 6: Luftbild, Messung 5: Süden nach Norden, Messung 6: Norden nach Süden, Quelle: Google Earth

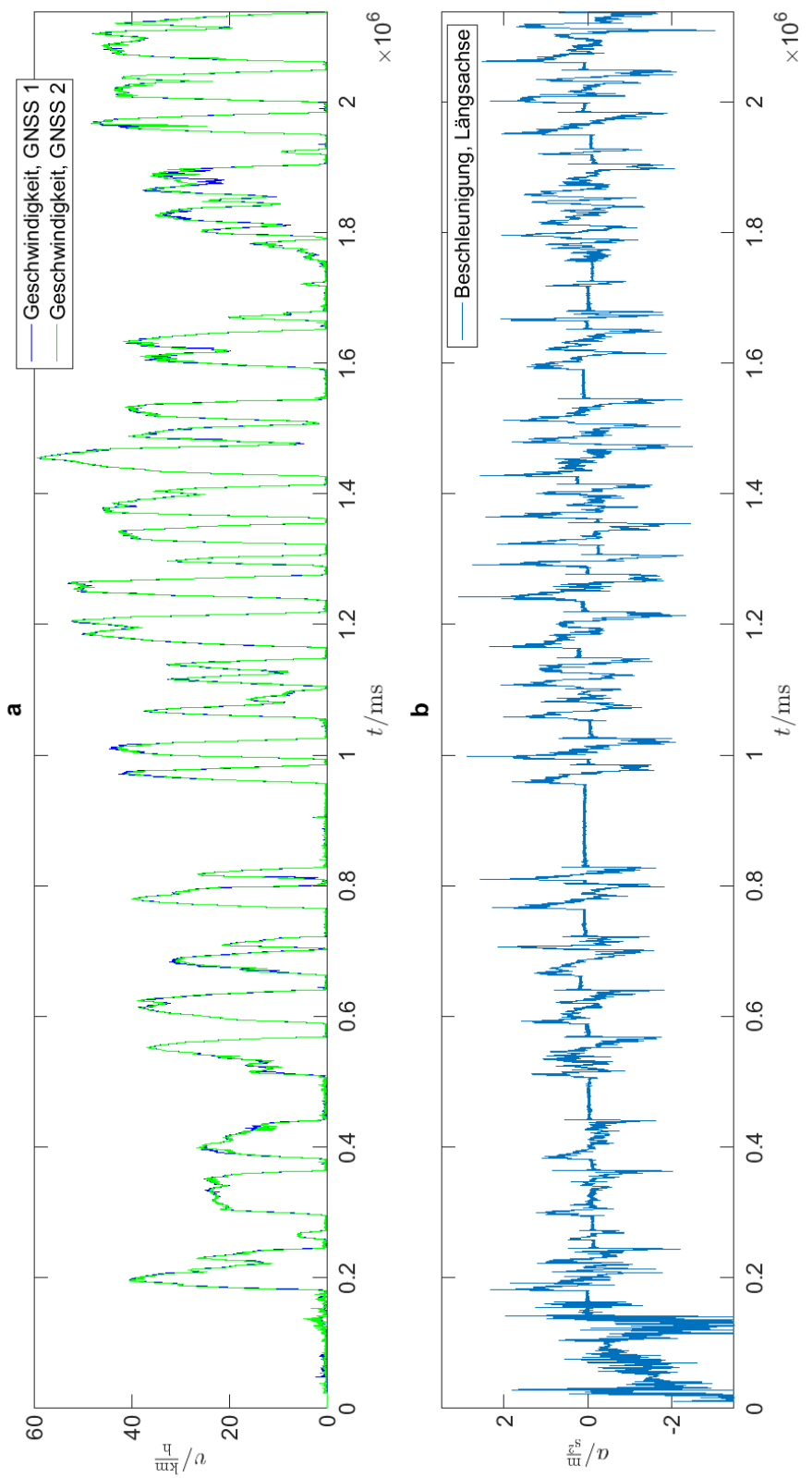


Abbildung C.15.: Messfahrt zu Messung-Nr. 5: Geschwindigkeit und Beschleunigung im Nahverkehrsbus, **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem ($f_{g,TP} = 10$ Hz)

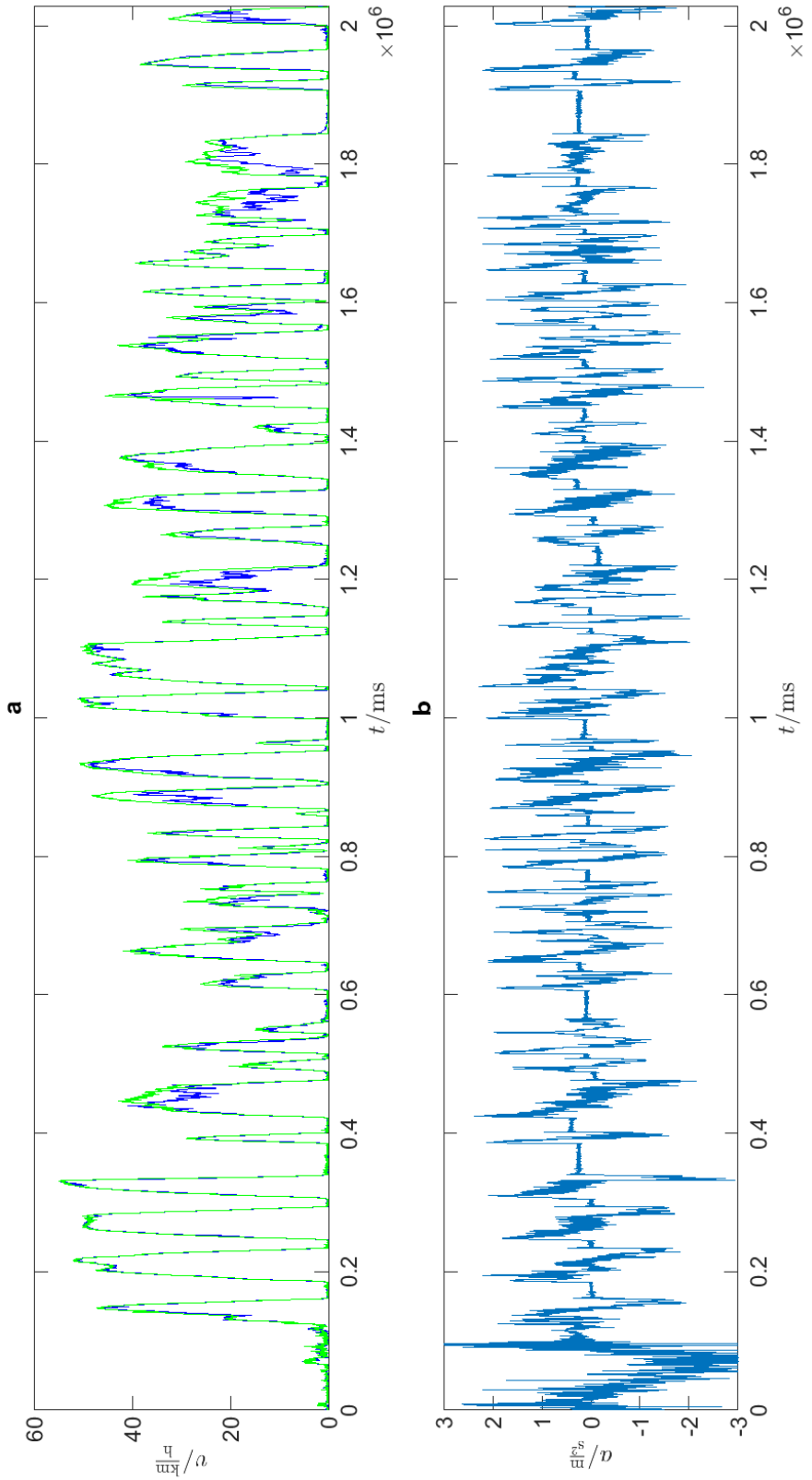


Abbildung C.16.: Messfahrt zu Messung-Nr. 6: Geschwindigkeit und Beschleunigung im Nahverkehrsbus, **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem ($f_{g,TP} = 10$ Hz)

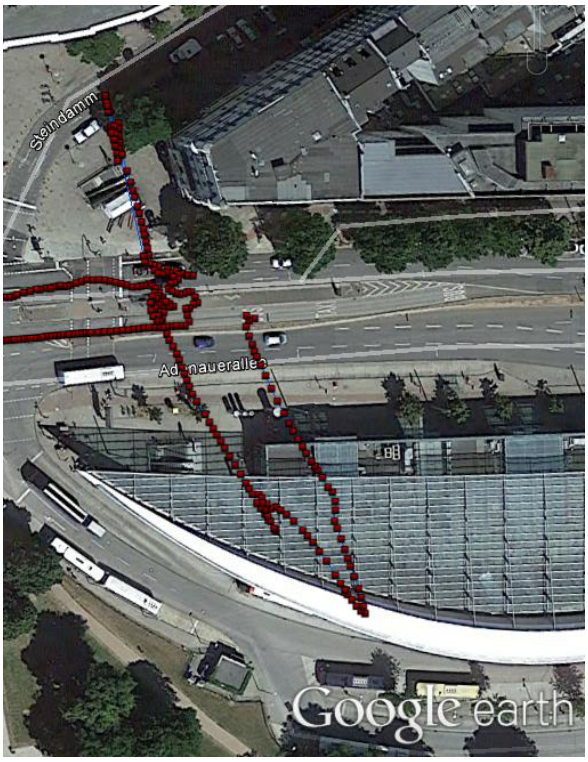


Abbildung C.17.: Messfahrt zu Messung-Nr. 5: Positionsfehler nach dem Einschalten des Datenloggers, Quelle: Google Earth

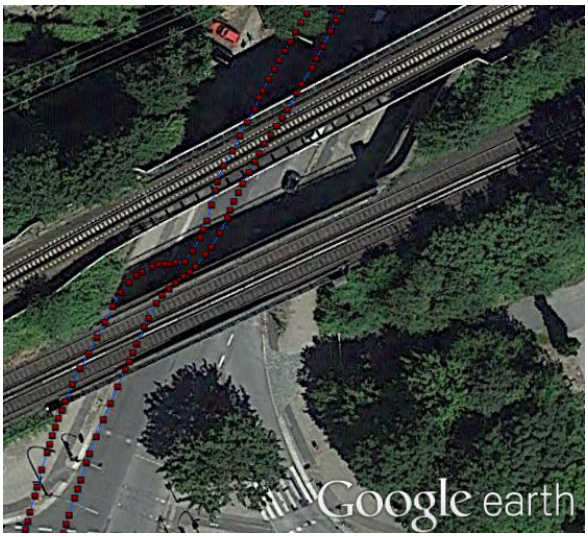


Abbildung C.18.: Messfahrt zu Messung-Nr. 5: Positionsfehler beim Unterfahren einer Brücke, Quelle: Google Earth

Abbildung C.19.: Messfahrt zu Messung-Nr. 6: Positionsunterschiede beider Empfänger, Quelle: Google Earth

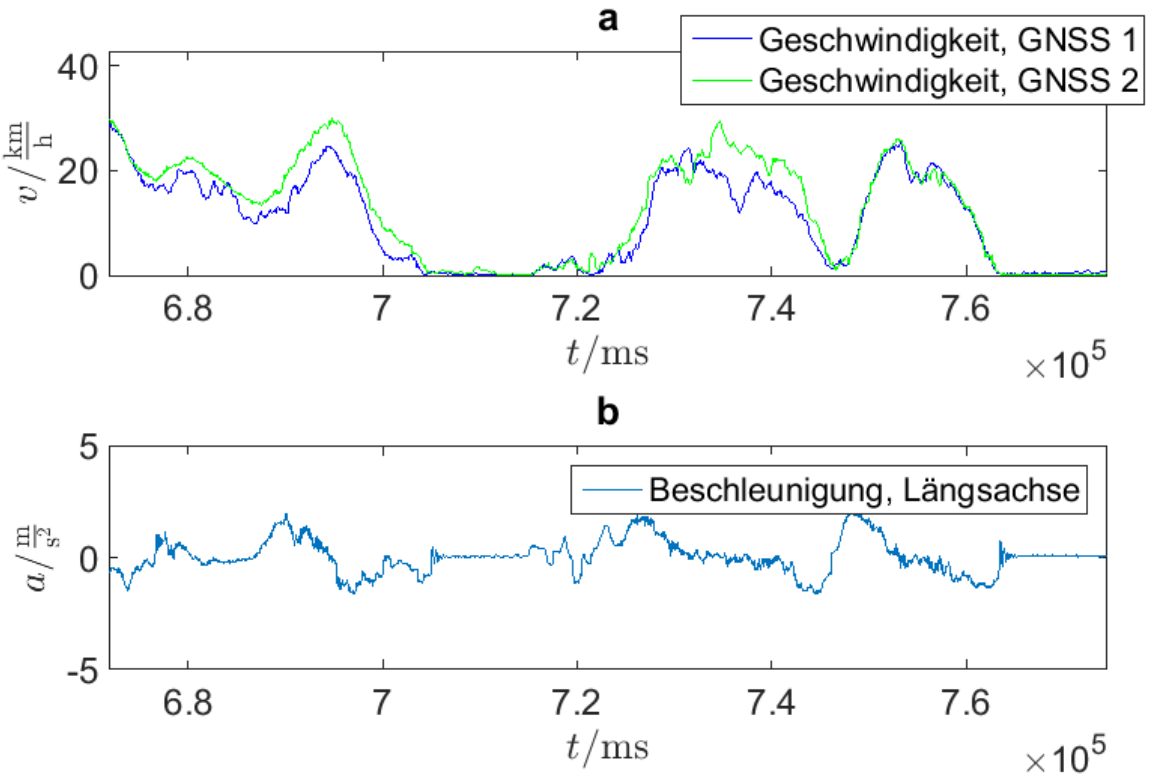


Abbildung C.20.: Messfahrt zu Messung-Nr. 6: Geschwindigkeitsunterschiede beider Empfänger, **a)** Geschwindigkeiten nach GNSS-Empfängern, **b)** Beschleunigung auf der Längsachse nach Inertialmesssystem ($f_{g,TP} = 10\,\mathrm{Hz}$)

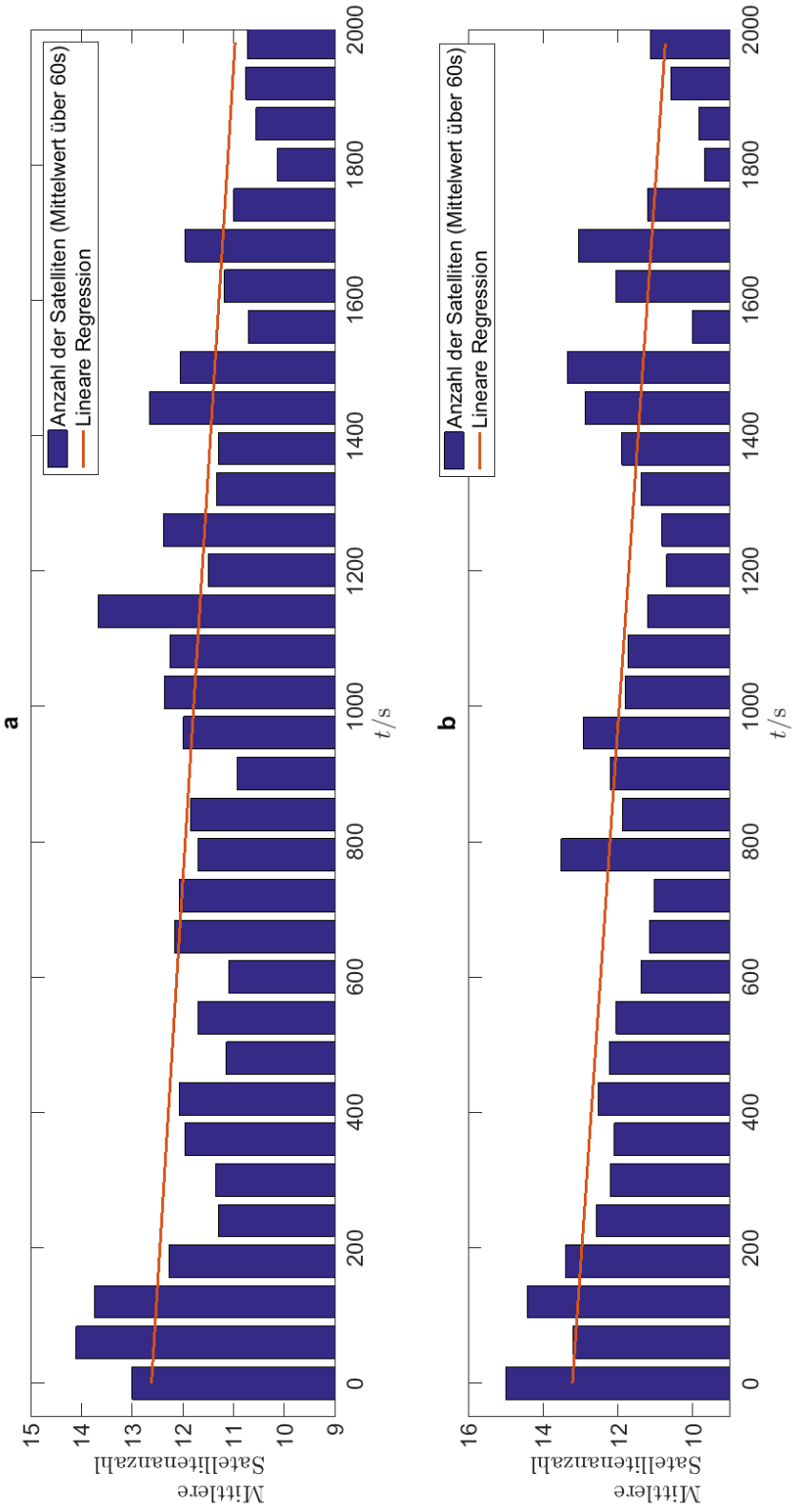


Abbildung C.21.: Messfahrt zu Messung-Nr. 6: Verwendete Satelliten zur Ortung, **a)** GNSS-Empfänger 1, **b)** GNSS-Empfänger 2

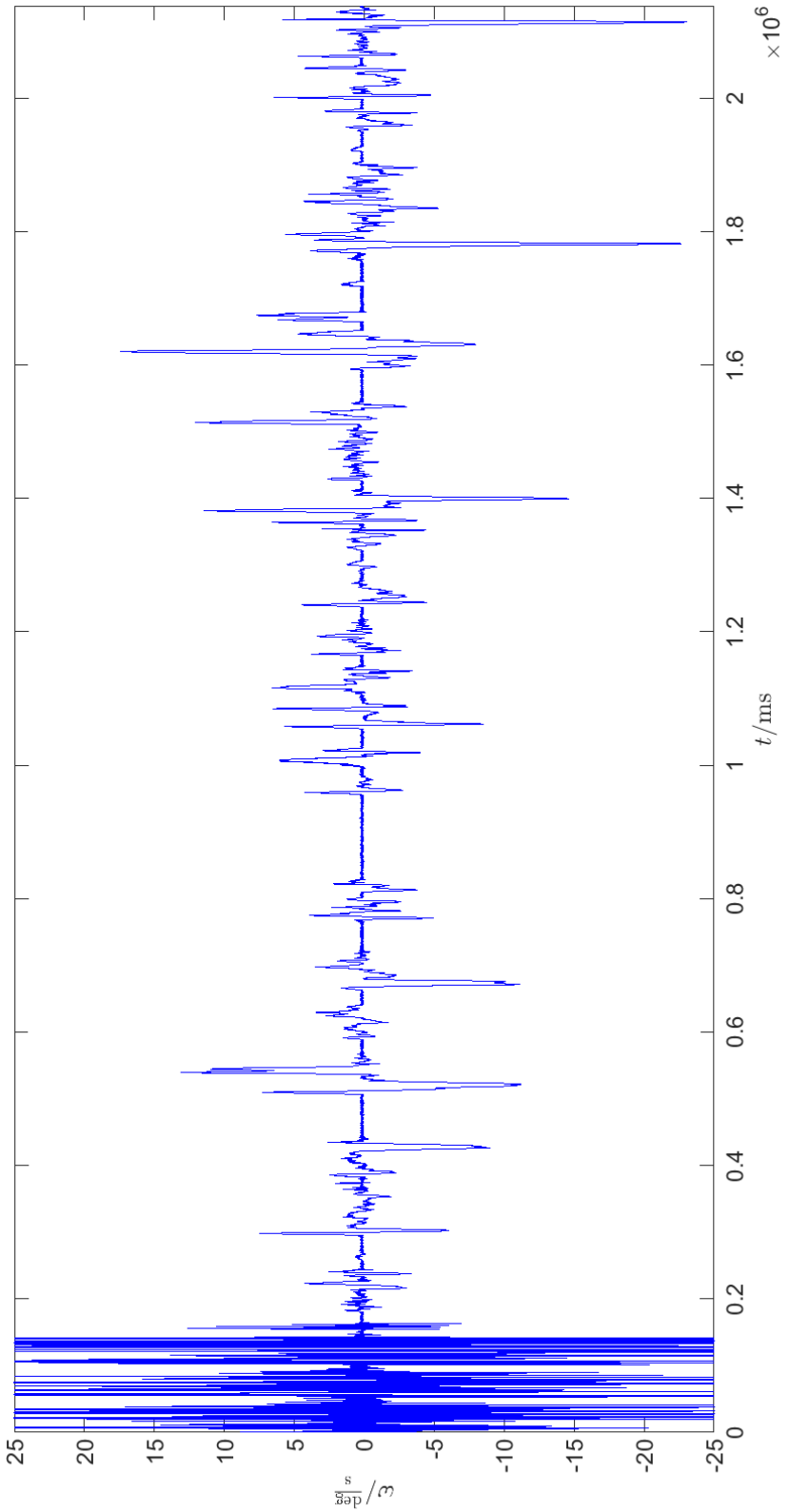


Abbildung C.22.: Messfahrt zu Messung-Nr. 5: Drehrate um die Gierachse im Nahverkehrsbus

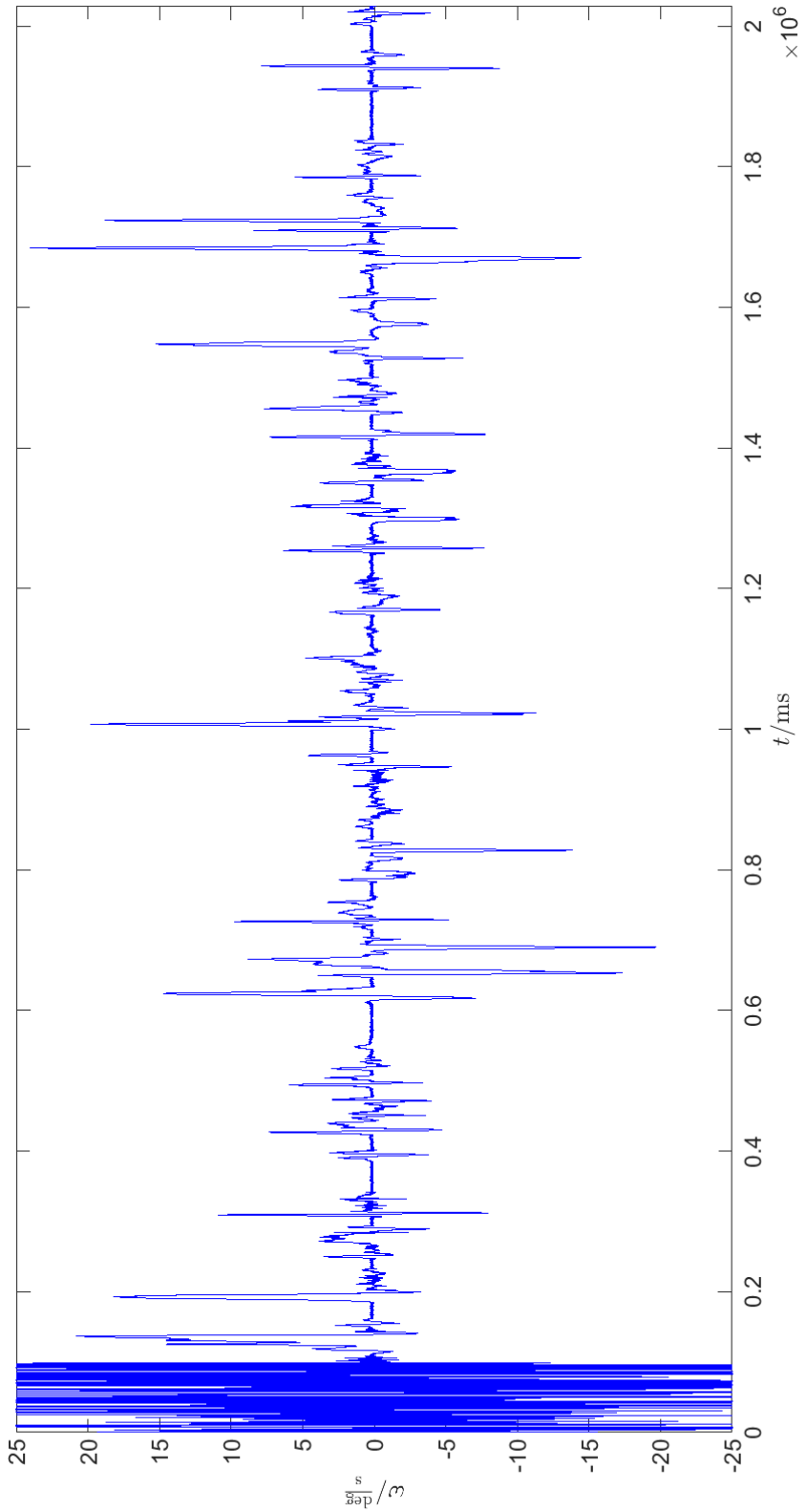


Abbildung C.23.: Messfahrt zu Messung-Nr. 6: Drehrate um die Gierachse im Nahverkehrsbus

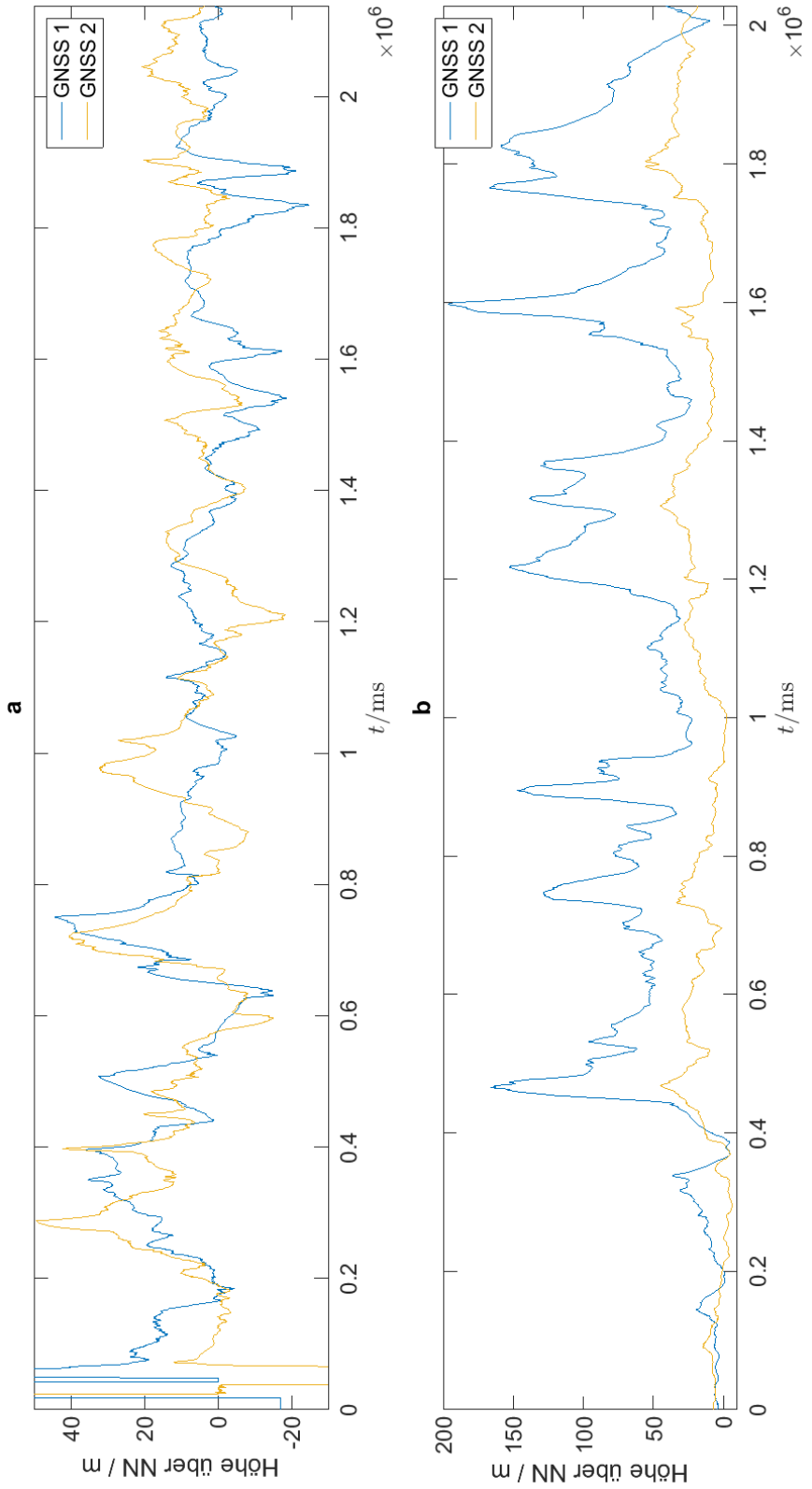


Abbildung C.24.: Messfahrt zu Messung-Nr. 5 und 6: Höhenmessung mittels Satellitenortung im Nahverkehrsbus, **a)** Messung 5, **b)** Messung 6

C.4. Temperatursensoren

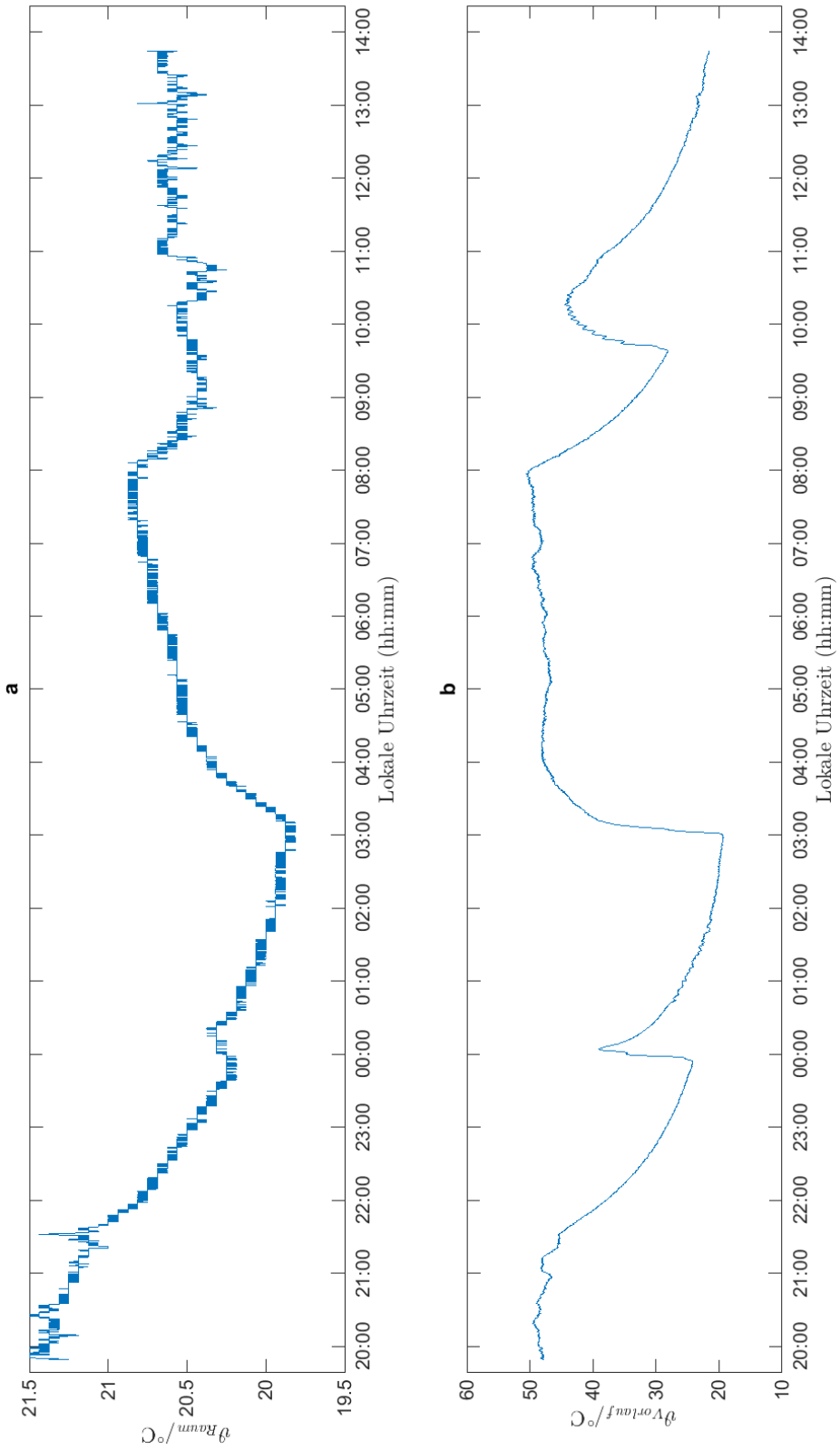


Abbildung C.25.: Messung-Nr. 7: Temperatur im Raum und Heizungsvorlauf, **a)** Raumtemperatur im Projektlabor, **b)** Temperatur des Heizungsvorlaufs am Heizkörper

D. Quellcode

D.1. Hauptprogramm

Quellcode D.1: Hauptprogramm des Datenloggers

```
1  /*
2   * main.c -- Datalogger productive main programm
3   *
4   * Author: Felix Groth
5   */
6
7  /*
8   * Number of samples to preallocate for the given samplerates
9   */
10 #define MAT_PREALLOC_1HZ  86400
11 #define MAT_PREALLOC_10HZ 864000
12 #define MAT_PREALLOC_100HZ  8640000
13
14 /*
15  * Sample rates
16  */
17 #define PERIOD_TIMEBASE    1000  //Period in us for the timebase counter
18 #define PERIOD_IMU_MEAS    10000 //Period in us of the IMU measurement (multiple of TIMEBASE)
19 #define PERIOD_TEMP_MEAS   1000000 //Period in us of the temperature measurement (multiple of TIMEBASE)
20
21 /*
22  * Settings for GPS communication setup
23  */
24 #define INIT_GPS_RETRY     20
25 #define INIT_GPS_WAIT    1000000
26
27 /*
28  * Thresholds for calibration
29  */
30 #define GYR_CALIB_THRESHOLD 200
31 #define ACC_CALIB_THRESHOLD 600
32
33 /*
34  * Debug level
35  */
36 #define DEBUG 3
37
38
39 //Standard includes
40 #include <stdint.h>
41 #include <stdbool.h>
42 #include <stdio.h>
43
44 //Hardware includes
45 #include "inc/tm4c1294ncpdt.h"
46 #include "inc/hw_nvic.h"
47
48 //Driver includes
49 #include "driverlib/sysctl.h"
50 #include "driverlib/timer.h"
51
52 //Module includes
53 #include "utils/uartstdio.h"
54 #include "buttons/buttons.h"
55 #include "leds/leds.h"
56 #include "onewire/onewire.h"
57 #include "imu/imu.h"
58 #include "gps/gps.h"
```

```
59  #include "matlab/mat4.h"
60  #include "display/gui.h"
61
62
63  /*
64   * Global variables
65   */
66  //System clock
67  unsigned int g_sysClock;
68  //Status information for each functional module
69  uint8_t g_gps1Ready = 0;
70  uint8_t g_gps2Ready = 0;
71  uint8_t g_oneWireReady = 0;
72  uint8_t g_imuReady = 0;
73  //OneWire
74  volatile maximROMCode g_onewireDevices[3];
75  volatile uint8_t g_onewireDeviceCount = 0;
76  volatile int16_t g_onewireDeviceActive = 0;
77  int16_t g_temperature[2] = {0x0550, 0x0550};
78  extern volatile onewireBuffer g_onewireRxBuffer;
79  extern volatile onewireBuffer g_onewireTxBuffer;
80  //FatFS
81  static FATFS g_fatFs;
82  static FRESULT g_fatFsResult;
83  //Relative time counter (milliseconds)
84  volatile uint32_t g_relTimeMs = 0;
85  //GPS uBlox buffer
86  extern volatile ubxPacketBufferRx* g_ubxPacketBufferRxActiveRead[2];
87  extern volatile ubxPacketBufferRx* g_ubxPacketBufferRxActiveWrite[2];
88  extern volatile ubxPacketBufferTx* g_ubxPacketBufferTxActiveRead[2];
89  extern volatile ubxPacketBufferTx* g_ubxPacketBufferTxActiveWrite[2];
90  //GPS Timepulse training helper
91  extern volatile gpsTOSHelper tosHelper;
92  //IMU buffer
93  extern volatile ImuDataBuffer g_imuDataBuffer[IMU_RINGBUFFER_SIZE];
94  extern volatile uint8_t g_imuReadPosition;
95  //Variables for IMU calibration
96  volatile MemsData g_accOffset;
97  volatile MemsData g_gyrOffset;
98
99  /*
100  * MATLAB output file configuration
101  */
102 #define MAT_FILE_COUNT     26
103 #define MAT_FILEID_LAT1    0
104 #define MAT_FILEID_LON1    1
105 #define MAT_FILEID_HEIGHT1  2
106 #define MAT_FILEID_NUMSV1 3
107 #define MAT_FILEID_SPEED1 4
108 #define MAT_FILEID_TIME1   5
109 #define MAT_FILEID_FIX1    6
110 #define MAT_FILEID_DAYTIME  7
111 #define MAT_FILEID_YEAR    8
112 #define MAT_FILEID_LAT2    9
113 #define MAT_FILEID_LON2    10
114 #define MAT_FILEID_HEIGHT2  11
115 #define MAT_FILEID_NUMSV2 12
116 #define MAT_FILEID_SPEED2 13
117 #define MAT_FILEID_TIME2   14
118 #define MAT_FILEID_FIX2    15
119 #define MAT_FILEID_TEMP1   16
120 #define MAT_FILEID_TEMP2   17
121 #define MAT_FILEID_TEMP3   18
122 #define MAT_FILEID_AX    19
123 #define MAT_FILEID_AY    20
124 #define MAT_FILEID_AZ    21
125 #define MAT_FILEID_GX    22
126 #define MAT_FILEID_GY    23
127 #define MAT_FILEID_GZ    24
128 #define MAT_FILEID_TIME3   25
129 //Files
130 mat4File matFiles[MAT_FILE_COUNT];
131 mat4File g_offsetFile;
132 //Filenames
133 const char* g_matFileNames[] = { "lat1.mat", "lon1.mat", "height1.mat", "numsv1.mat", "speed1.mat", "time1.mat", "fix1.mat", ↩
        "datetime.mat", "year.mat",
134                  "lat2.mat", "lon2.mat", "height2.mat", "numsv2.mat", "speed2.mat", "time2.mat", "fix2.mat",
135                  "temp1.mat", "temp2.mat", "temp3.mat",
136                  "ax.mat", "ay.mat", "az.mat", "gx.mat", "gy.mat", "gz.mat", "time3.mat"
137 };
138 //Variable names
139 const char* g_matVariableNames[] = { "lat1", "lon1", "height1", "numsv1", "speed1", "time1", "fix1", "datetime","year",
140                    "lat2", "lon2", "height2", "numsv2", "speed2", "time2", "fix2",
141                    "temp1", "temp2", "temp3",
142                    "ax", "ay", "az", "gx", "gy", "gz", "time3"
```

```
143 };
144 //Filetypes
145 const uint32_t g_matFileTypes[] = { MAT_TYPE_INT32, MAT_TYPE_INT32, MAT_TYPE_INT32, MAT_TYPE_UINT8, MAT_TYPE_INT32, MAT_TYPE_INT32, ↩
        MAT_TYPE_UINT8, MAT_TYPE_UINT8, MAT_TYPE_UINT16,
146                     MAT_TYPE_INT32, MAT_TYPE_INT32, MAT_TYPE_INT32, MAT_TYPE_UINT8, MAT_TYPE_INT32, MAT_TYPE_INT32, MAT_TYPE_UINT8,
147                       MAT_TYPE_INT16, MAT_TYPE_INT16, MAT_TYPE_INT16,
148                       MAT_TYPE_INT16, MAT_TYPE_INT16, MAT_TYPE_INT16, MAT_TYPE_INT16, MAT_TYPE_INT16, MAT_TYPE_INT16, MAT_TYPE_INT32
149 };
150 //Matrix row dimensions
151 const uint8_t g_matFileRows[] = { 1, 1, 1, 1, 1, 1, 1, 5, 1,
152                     1, 1, 1, 1, 1, 1, 1,
153                     1, 1, 1,
154                     1, 1, 1, 1, 1, 1, 1
155 };
156 //File preallocation in vector length
157 const uint32_t g_matFilePrealloc[] = { MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ, ↩
        MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ,
158                       MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ, ↩
                              MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ,
159                       MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ, MAT_PREALLOC_10HZ,
160                       MAT_PREALLOC_100HZ, MAT_PREALLOC_100HZ, MAT_PREALLOC_100HZ, MAT_PREALLOC_100HZ, MAT_PREALLOC_100HZ, ↩
                              MAT_PREALLOC_100HZ, MAT_PREALLOC_100HZ
161 };
162
163 /*
164  * Enumeration for main state machine
165  */
166 typedef enum {
167   mainInit,
168   mainImuCalibrate,
169   mainStart,
170   mainWaitStart,
171   mainRecordStart,
172   mainRecord,
173   mainRecordStop,
174   mainShutdown,
175   mainInitError,
176   mainSDError,
177   mainFileInitError,
178   mainFileWriteError,
179   mainFileTransfer,
180   mainHold
181 } mainProgramState;
182
183 /*
184  * Enumeration for file transfer state machine
185  */
186 typedef enum {
187   fileTransferInactive,
188   fileTransferActive,
189   fileTransferDelete
190 } fileTransferState;
191
192
193 /*
194  * Main state machine state
195  */
196 static volatile mainProgramState g_progState = mainInit;
197 // FSM state for file transfer
198 static volatile fileTransferState g_transferState = fileTransferInactive;
199
200 /*
201  * Handler for the systick interrupt. This should be called every 10ms
202  * to signal a tick to the FatFS file system
203  */
204 void SysTickHandler(void)
205 {
206     //Call the FatFs tick timer.
207     disk_timerproc();
208 }
209
210
211 /*
212  * Millisecond counter interrupt handler
213  * Counts milliseconds and may trigger high priority interrupt operations.
214  * This is the interrupt handler for the active timebase (GPS1 / GPS2 / Timer)
215  */
216 void timerMsIsr(void) {
217   //Clear interrupt flag
218   GPIOIntClear(GPS1_TIMEPULSE_BASE, GPS1_TIMEPULSE_INT_PIN);
219   GPIOIntClear(GPS2_TIMEPULSE_BASE, GPS2_TIMEPULSE_INT_PIN);
220
221   if (!(g_gps1Ready || g_gps2Ready))
222     TimerIntClear(TIMER4_BASE, TIMER_TIMA_TIMEOUT);
223
```

```
224   if (!(g_progState == mainRecord))
225     return;
226
227   g_relTimeMs++;
228
229   //Take measurement from IMU
230   if (!(g_relTimeMs % (PERIOD_IMU_MEAS / PERIOD_TIMEBASE))) {
231     if (imuTriggerMeasurement() != IMU_BUFFER_OK)
232 #if DEBUG > 1
233       UARTCharPut(UART2_BASE, 'E');
234 #endif
235   }
236
237     //Measure a temperature every PREIOD_TEMP_MEAS
238   if (g_oneWireReady) {
239     if (!((g_relTimeMs + (PERIOD_TEMP_MEAS / PERIOD_TIMEBASE)/2) % (PERIOD_TEMP_MEAS / PERIOD_TIMEBASE)))
240       ds1820StartConversionInt((maximROMCode *)&(g_onewireDevices[g_onewireDeviceActive]));
241     if (!(g_relTimeMs % (PERIOD_TEMP_MEAS / PERIOD_TIMEBASE))) {
242       ds1820ReadTemperatureInt((maximROMCode *)&(g_onewireDevices[g_onewireDeviceActive]));
243       if (g_onewireDeviceActive < (g_onewireDeviceCount - 1))
244         g_onewireDeviceActive++;
245       else
246         g_onewireDeviceActive = 0;
247     }
248   }
249
250   //Blinking UI feedback when measurement is active (1s)
251   if (!(g_relTimeMs%1000))
252     ledsToggle(LED_YELLOW);
253 }
254
255
256 /*
257  * Writes available data to the sd-card
258  */
259 void writeData(void) {
260   // Check if IMU data is ready
261   switch (g_imuDataBuffer[g_imuReadPosition].state) {
262   case imuDataReady:
263 #if DEBUG > 7
264     UARTCharPut(UART2_BASE, 'I');
265 #endif
266     //Write IMU measurement to files
267     g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_AX]), (uint8_t*)&(g_imuDataBuffer[g_imuReadPosition].imuData.acc.x));
268     g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_AY]), (uint8_t*)&(g_imuDataBuffer[g_imuReadPosition].imuData.acc.y));
269     g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_AZ]), (uint8_t*)&(g_imuDataBuffer[g_imuReadPosition].imuData.acc.z));
270     g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_GX]), (uint8_t*)&(g_imuDataBuffer[g_imuReadPosition].imuData.gyr.x));
271     g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_GY]), (uint8_t*)&(g_imuDataBuffer[g_imuReadPosition].imuData.gyr.y));
272     g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_GZ]), (uint8_t*)&(g_imuDataBuffer[g_imuReadPosition].imuData.gyr.z));
273     g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TIME3]), (uint8_t*)&(g_imuDataBuffer[g_imuReadPosition].timestamp));
274     imuPacketRead();
275     break;
276   default:
277     break;
278   }
279
280   // Check if 1-Wire data is ready
281   switch (g_onewireRxBuffer.state) {
282   case owBufferReady:
283 #if DEBUG > 5
284     UARTCharPut(UART2_BASE, 'T');
285 #endif
286     //Simply copy the temperature into our buffers.
287     //This is only 16-bit and not timecritical due to slow temperature changes.
288     //We will write these with the GPS data, so that we don't need another timebase.
289     if (g_onewireRxBuffer.onewireData.romCode.raw == g_onewireDevices[0].raw)
290       g_temperature[0] = *(g_onewireRxBuffer.onewireData.temperature);
291     else if (g_onewireRxBuffer.onewireData.romCode.raw == g_onewireDevices[1].raw)
292       g_temperature[1] = *(g_onewireRxBuffer.onewireData.temperature);
293     g_onewireRxBuffer.state = owBufferFree;
294
295     //Write data if GPS does not do it
296     if (!g_gps1Ready && !g_gps2Ready) {
297       g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TIME1]), (uint8_t*)&(g_relTimeMs));
298       g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TEMP1]), (uint8_t*)&(g_temperature[0])); //Write temperatures
299       g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TEMP2]), (uint8_t*)&(g_temperature[1])); //Write temperatures
300       g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TEMP3]), (uint8_t*)&(g_temperature[2])); //Write temperatures
301     }
302     break;
303   default:
304     break;
305   }
306
307   // Check if GPS1 data is ready
308   switch (g_ubxPacketBufferRxActiveRead[gps1]->rxState) {
```

```
309     case ubxRxDataReady:
310       //Position information?
311       if ((g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->class == UBX_NAV) && (g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->id ↩
              == UBX_NAV_PVT)) {
312         //Write GPS data to files
313         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_LON1]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->ubxNavPvt.lon));
314         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_LAT1]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->ubxNavPvt.lat));
315         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_HEIGHT1]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->ubxNavPvt.hMSL));
316         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TIME1]), (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps1]->timestamp));
317         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_YEAR]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->ubxNavPvt.year));
318         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_DAYTIME]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->ubxNavPvt.month));
319         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_FIX1]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->ubxNavPvt.fixType));
320         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_NUMSV1]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->ubxNavPvt.numSV));
321         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_SPEED1]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->ubxNavPvt.gSpeed));
322
323         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TEMP1]), (uint8_t*)&(g_temperature[0])); //Write temperatures
324         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TEMP2]), (uint8_t*)&(g_temperature[1])); //Write temperatures
325         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TEMP3]), (uint8_t*)&(g_temperature[2])); //Write temperatures
326       } //TODO: Take care of other packets received?
327   #if DEBUG > 5
328       UARTprintf("G1");
329   #endif
330       ubxBufferRxPkgRead(gps1, g_ubxPacketBufferRxActiveRead[gps1]);
331       break;
332     case ubxRxError:
333       ubxBufferRxPkgRead(gps1, g_ubxPacketBufferRxActiveRead[gps1]);
334       UARTprintf("GPS1 data had CRC errors!\n");
335       break;
336     default:
337       break;
338     }
339
340     // Check if GPS2 data is ready
341     switch (g_ubxPacketBufferRxActiveRead[gps2]->rxState) {
342     case ubxRxDataReady:
343       //Position information?
344       if ((g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->class == UBX_NAV) && (g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->id ↩
              == UBX_NAV_PVT)) {
345         //Write GPS data to files
346         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_LON2]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->ubxNavPvt.lon));
347         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_LAT2]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->ubxNavPvt.lat));
348         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_HEIGHT2]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->ubxNavPvt.hMSL));
349         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_FIX2]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->ubxNavPvt.fixType));
350         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_NUMSV2]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->ubxNavPvt.numSV));
351         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_SPEED2]), ↩
                (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->ubxNavPvt.gSpeed));
352         g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TIME2]), (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps2]->timestamp));
353
354         if (!g_gps1Ready) {
355           g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TIME1]), (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps2]->timestamp));
356           g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_DAYTIME]), ↩
                  (uint8_t*)&(g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->ubxNavPvt.month));
357           g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TEMP1]), (uint8_t*)&(g_temperature[0])); //Write temperatures
358           g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TEMP2]), (uint8_t*)&(g_temperature[1])); //Write temperatures
359           g_fatFsResult |= matFileAddValue(&(matFiles[MAT_FILEID_TEMP3]), (uint8_t*)&(g_temperature[2])); //Write temperatures
360         }
361       } //TODO: Take care of other packets received.
362   #if DEBUG > 5
363       UARTprintf("G2");
364   #endif
365       ubxBufferRxPkgRead(gps2, g_ubxPacketBufferRxActiveRead[gps2]);
366       break;
367     case ubxRxError:
368       ubxBufferRxPkgRead(gps2, g_ubxPacketBufferRxActiveRead[gps2]);
369       UARTprintf("gps2 data had CRC errors!\n");
370       break;
371     default:
372       break;
373     }
374
375     //Check for file write errors
376     if (g_fatFsResult != FR_OK)
```

```
377       g_progState = mainFileWriteError;
378   }
379
380   /*
381    * Callback functions for the touchscreen
382    */
383   // Touchscreen start button
384   void lcdBtnStartClick(tWidget *pWidget) {
385     if (g_progState == mainWaitStart) {
386       g_progState = mainRecordStart;
387       PushButtonTextSet((tPushButtonWidget*)pWidget, "Stop");
388     }
389     if (g_progState == mainRecord) {
390       g_progState = mainRecordStop;
391       PushButtonTextSet((tPushButtonWidget*)pWidget, "Start");
392     }
393   }
394   // Touchscreen calibrate button
395   void lcdBtnCalibrateClick(tWidget *pWidget) {
396     if (g_progState == mainWaitStart)
397       g_progState = mainImuCalibrate;
398   }
399   // Touchscreen off button
400   void lcdBtnOffClick(tWidget *pWidget) {
401     if (g_progState == mainWaitStart)
402       g_progState = mainShutdown;
403   }
404
405
406   /*
407    * Callback functions for buttons and uart UI
408    */
409   // ISR to handle received data from virtual serial port
410   void uartDataIsr(void) {
411     uint8_t uartChar;
412
413     UARTIntClear(UART2_BASE, UARTIntStatus(UART2_BASE, true));
414     uartChar = UARTCharGet(UART2_BASE);
415
416     switch(g_progState) {
417     case mainWaitStart:
418       if (uartChar == 'g')
419         g_progState = mainRecordStart;
420       else if (uartChar == 't') {
421         g_progState = mainFileTransfer;
422         lcdGuiShowTransferActive();
423       }
424       break;
425     case mainRecord:
426       if (uartChar == 's')
427         g_progState = mainRecordStop;
428       break;
429     case mainFileTransfer:
430       if (g_transferState == fileTransferInactive) {
431         if (uartChar == 'g')
432           g_transferState = fileTransferActive;
433         else if (uartChar == 's') {
434           lcdGuiHideTransferActive();
435           g_progState = mainStart;
436         } else if (uartChar == 'd')
437           g_transferState = fileTransferDelete;
438       }
439       break;
440     }
441   }
442   // ISR to handle push button presses
443   // Since the TLC4412 STAT pin shares PORTK, this may also indicate
444   // a loss of power!
445   void pushButtonIsr(void) {
446     pushButtonsIntClear();
447     switch(g_progState) {
448     case mainWaitStart:
449       if (!(pushButtonsRead() & BTN2_BIT))
450         g_progState = mainRecordStart;
451       else if (!(pushButtonsRead() & BTN1_BIT))
452         g_progState = mainShutdown;
453       break;
454     case mainRecord:
455       if (!(pushButtonsRead() & BTN1_BIT))
456         g_progState = mainRecordStop;
457       break;
458     }
459
460     if (GPIOIntStatus(GPIO_PORTK_BASE, true) & GPIO_INT_PIN_3) {
461       //Power loss!
```

```
462       if (g_progState == mainRecord)
463         g_progState = mainRecordStop;
464     }
465     GPIOIntClear(GPIO_PORTK_BASE, GPIO_INT_PIN_3);
466   }
467
468
469   /*
470    * Inline functions for main program sequences
471    */
472   // Initialize all functional modules
473   inline void initModules() {
474     //Loop iterator
475     uint32_t i;
476
477       // Print something to make sure UART works
478       UARTprintf("\n\n\n\n****Booting BATSEN DataLogger:\n");
479
480       // Initialize buttons
481       UARTprintf("Initializing buttons........[busy]");
482       pushButtonsInit();
483       UARTprintf("\b\b\b\b\bdone]\n");
484
485       // Initialize LEDs
486       UARTprintf("Initializing LEDs........[busy]");
487       ledsInit();
488       ledsWrite(LED_GREEN | LED_YELLOW);
489       UARTprintf("\b\b\b\b\bdone]\n");
490
491       // Initialize LCD
492       UARTprintf("Initializing LCD........[busy]");
493       //Register int handler for touchscreen
494       lcdGuiInit(7);
495       UARTprintf("\b\b\b\b\bdone]\n");
496
497       //Ouput log to LCD
498       lcdGuiAppendLogLine("Booting BATSEN DataLogger:");
499
500       // Initialize OneWire
501     UARTprintf("Initializing OneWire........[busy]");
502     lcdGuiAppendLogLine("Initializing OneWire........[busy]");
503     initOneWireUART();
504     g_onewireDeviceCount = searchOneWire((maximROMCode *)g_onewireDevices, 3, 0);
505     if (!(g_onewireDeviceCount > 0)) {
506       UARTprintf("\b\b\b\b\bfailed]\n");
507       lcdGuiUpdateLogLine("Initializing OneWire........[failed]");
508       g_oneWireReady = 0;
509     } else {
510       g_oneWireReady = 1;
511       UARTprintf("\b\b\b\b\bdone]\n");
512       lcdGuiUpdateLogLine("Initializing OneWire........[done]");
513     }
514
515       // Initialize IMU
516     UARTprintf("Initializing IMU........[busy]");
517     lcdGuiAppendLogLine("Initializing IMU........[busy]");
518     if (imuInitI2C() != IMU_COM_READY) {
519       UARTprintf("\b\b\b\b\bfailed]\n");
520       lcdGuiUpdateLogLine("Initializing IMU........[failed]");
521       g_progState = mainInitError;
522       g_imuReady = 0;
523       return;
524     } else {
525       imuInit(MPU_VAL_FS_500, MPU_VAL_AFS_2G);
526       g_imuReady = 1;
527       lcdGuiUpdateLogLine("Initializing IMU........[done]");
528       UARTprintf("\b\b\b\b\bdone]\n");
529     }
530
531       // Initialize GPS 1
532       UARTprintf("Initializing GPS1........[busy]");
533       lcdGuiAppendLogLine("Initializing GPS1........[busy]");
534       for (i = INIT_GPS_RETRY; i; i--) {
535         if (gpsInitModule(gps1, GPS_BAUDRATE, 4) == GPS_INIT_SUCCESS) {
536           g_gps1Ready = 1;
537           break;
538         }
539         SysCtlDelay(INIT_GPS_WAIT);
540       }
541       if (g_gps1Ready) {
542         UARTprintf("\b\b\b\b\bdone]\n");
543         lcdGuiUpdateLogLine("Initializing GPS1........[done]");
544       } else {
545         UARTprintf("\b\b\b\b\bfailed]\n");
546         lcdGuiUpdateLogLine("Initializing GPS1........[failed]");
```

```
547       }
548
549       //Initialize GPS 2
550       UARTprintf("Initializing GPS2........[busy]");
551       lcdGuiAppendLogLine("Initializing GPS2........[busy]");
552       for (i = INIT_GPS_RETRY; i; i--) {
553         if (gpsInitModule(gps2, GPS_BAUDRATE, 4) == GPS_INIT_SUCCESS) {
554           g_gps2Ready = 1;
555           break;
556         }
557         SysCtlDelay(INIT_GPS_WAIT);
558       }
559       if (g_gps2Ready) {
560         UARTprintf("\b\b\b\b\bdone]\n");
561         lcdGuiUpdateLogLine("Initializing GPS2........[done]");
562       } else {
563         UARTprintf("\b\b\b\b\bfailed]\n");
564         lcdGuiUpdateLogLine("Initializing GPS2........[failed]");
565       }
566
567       //Setup interrupt settings
568       UARTprintf("Setting up interrupts........[busy]");
569       lcdGuiAppendLogLine("Setting up interrupts........[busy]");
570       //GPS as timebase in milliseconds, else use local timer
571       //Measurement timebase interrupt has highest priority
572       if (g_gps1Ready) {
573       if (gpsTimepulseIntSetup(gps1, 1000, timerMsIsr, 2) != GPS_INIT_SUCCESS) {
574         UARTprintf("\b\b\b\b\bfailed]\n");
575         lcdGuiUpdateLogLine("Setting up interrupts........[failed]");
576         g_progState = mainInitError;
577         return;
578       }
579       } else if (g_gps2Ready) {
580       if (gpsTimepulseIntSetup(gps2, 1000, timerMsIsr, 2) != GPS_INIT_SUCCESS) {
581         UARTprintf("\b\b\b\b\bfailed]\n");
582         lcdGuiUpdateLogLine("Setting up interrupts........[failed]");
583         g_progState = mainInitError;
584         return;
585       }
586       } else {
587         //Setup local timer to get millisecond ticks
588       SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER4);
589       TimerConfigure(TIMER4_BASE, TIMER_CFG_PERIODIC);
590       TimerLoadSet(TIMER4_BASE, TIMER_A, g_sysClock/1000);
591       TimerPrescaleSet(TIMER4_BASE, TIMER_A, 0);
592       TimerControlStall(TIMER4_BASE, TIMER_A, true);  //Stop timer in debug mode.
593       IntEnable(INT_TIMER4A);
594       TimerIntRegister(TIMER4_BASE, TIMER_A, timerMsIsr);
595       TimerIntEnable(TIMER4_BASE, TIMER_TIMA_TIMEOUT);
596       IntPrioritySet(INT_TIMER4A, 2);
597       TimerEnable(TIMER4_BASE, TIMER_A);
598       }
599     //Systick has a medium priority because it manages FatFS timeouts but is not awfully time critical
600     IntPrioritySet(INT_SYSCTL, 6);
601     SysTickPeriodSet(g_sysClock / 100);
602     SysTickEnable();
603     SysTickIntRegister(SysTickHandler);
604     SysTickIntEnable();
605     //Interrupt for onewire is not time critical
606     onewireIntEnable(5);
607     //Interrupt for IMU communication has rather high priority because of its high frequency
608     imuSetupRcvInt(3);
609     //Interrupt for push buttons - has to have high priority, because of TLC4412 STAT pin on same port!
610     pushButtonsIsrRegister(pushButtonIsr, 1);
611       //Virtual serial port interrupts
612     UARTFIFODisable(UART2_BASE);
613     IntPrioritySet(INT_UART2, 10);
614     UARTIntRegister(UART2_BASE, uartDataIsr);
615     UARTIntEnable(UART2_BASE, UART_INT_RX);
616
617     UARTprintf("\b\b\b\b\bdone]\n");
618     lcdGuiUpdateLogLine("Setting up interrupts........[done]");
619
620     //If we reached this point, the init was successfull
621     UARTprintf("BATSEN DataLogger operative\n\n");
622     lcdGuiAppendLogLine("BATSEN DataLogger operative");
623   }
624
625   // Get calibration vectors for IMU
626   inline void calibrateImu(void) {
627     //Iterator
628     uint32_t i;
629     //Variables for IMU calibration temp values
630     int32_t axMid, ayMid, azMid,
631         gxMid, gyMid, gzMid;
```

```
632
633   ledsWrite(ledsRead() | LED_YELLOW);
634   UARTprintf("IMU offset measurement........[busy]");
635   lcdGuiAppendLogLine("IMU offset measurement........[busy]");
636   axMid = ayMid = azMid = 0;
637   gxMid = gyMid = gzMid = 0;
638
639   //Measure the acceleration 128 times and calculate mean
640   for (i=0; i<128; i++) {
641     imuTriggerMeasurement();
642     while (g_imuDataBuffer[g_imuReadPosition].state != imuDataReady)
643       ;
644     axMid += g_imuDataBuffer[g_imuReadPosition].imuData.acc.x;
645     ayMid += g_imuDataBuffer[g_imuReadPosition].imuData.acc.y;
646     azMid += g_imuDataBuffer[g_imuReadPosition].imuData.acc.z;
647     gxMid += g_imuDataBuffer[g_imuReadPosition].imuData.gyr.x;
648     gyMid += g_imuDataBuffer[g_imuReadPosition].imuData.gyr.y;
649     gzMid += g_imuDataBuffer[g_imuReadPosition].imuData.gyr.z;
650     imuPacketRead();
651     SysCtlDelay(2000000);
652   }
653   g_accOffset.x = axMid / 128;
654   g_accOffset.y = ayMid / 128;
655   g_accOffset.z = azMid / 128;
656   g_gyrOffset.x = gxMid / 128;
657   g_gyrOffset.y = gyMid / 128;
658   g_gyrOffset.z = gzMid / 128;
659
660   UARTprintf("\b\b\b\b\bdone]\n");
661   lcdGuiUpdateLogLine("IMU offset measurement........[done]");
662
663   //Mount the SD card file system
664   UARTprintf("Initializing SD card file system........[busy]");
665   lcdGuiAppendLogLine("Initializing SD card file system........[busy]");
666   if(f_mount(0, &g_fatFs) == FR_OK) {
667     UARTprintf("\b\b\b\b\bdone]\n");
668     lcdGuiUpdateLogLine("Initializing SD card file system........[done]");
669   } else {
670     UARTprintf("\b\b\b\b\bfailed]\n");
671     lcdGuiUpdateLogLine("Initializing SD card file system........[failed]");
672     g_progState = mainSDError;
673     return;
674   }
675
676   //Create offset vector file
677   UARTprintf("Creating file /offset/acc.mat........[busy]");
678   lcdGuiAppendLogLine("Creating file /offset/acc.mat........[busy]");
679
680   //Create folder if neccessary
681   f_mkdir("/offset");
682
683   if (matFileMatrixCreate(&g_offsetFile, "/offset/acc.mat", "accoffset", 1, 3, MAT_TYPE_INT16) == FR_OK) {
684     UARTprintf("\b\b\b\b\bdone]\n");
685     lcdGuiUpdateLogLine("Creating file /offset/acc.mat........[done]");
686   } else {
687     UARTprintf("\b\b\b\b\bfailed]\n");
688     lcdGuiUpdateLogLine("Creating file /offset/acc.mat........[failed]");
689     g_progState = mainFileInitError;
690     return;
691   }
692
693   UARTprintf("Writing acc offset vector........[busy]");
694   lcdGuiUpdateLogLine("Writing acc offset vector........[busy]");
695   //Write offset vector to file
696   g_fatFsResult |= matFileAddValue(&g_offsetFile, (uint8_t*)&(g_accOffset.z));
697
698   //Close mat file
699   matFileClose(&g_offsetFile);
700
701   UARTprintf("\b\b\b\b\bdone]\n");
702   lcdGuiUpdateLogLine("Writing acc offset vector........[done]");
703
704
705   UARTprintf("Please accelerate vehicle!........[busy]");
706   lcdGuiUpdateLogLine("Please accelerate vehicle!........[busy]");
707
708   for (i=0; i<128; i++) {
709     imuTriggerMeasurement();
710     while (g_imuDataBuffer[g_imuReadPosition].state != imuDataReady)
711       ;
712     //Is the vehicle turning?
713     if (abs(((g_imuDataBuffer[g_imuReadPosition].imuData.gyr.x - g_gyrOffset.x)) > GYR_CALIB_THRESHOLD) ||
714       abs(((g_imuDataBuffer[g_imuReadPosition].imuData.gyr.y - g_gyrOffset.y)) > GYR_CALIB_THRESHOLD) ||
715       abs(((g_imuDataBuffer[g_imuReadPosition].imuData.gyr.z - g_gyrOffset.z)) > GYR_CALIB_THRESHOLD)) {
716         lcdGuiUpdateLogLine("Please accelerate vehicle!........[turning]");
```

```
717           i = 0;
718           axMid = ayMid = azMid = 0;
719       } else {
720         //Is the vehicle accelerating enough?
721           if (abs(((g_imuDataBuffer[g_imuReadPosition].imuData.acc.x - g_accOffset.x)) > ACC_CALIB_THRESHOLD) ||
722             abs(((g_imuDataBuffer[g_imuReadPosition].imuData.acc.y - g_accOffset.y)) > ACC_CALIB_THRESHOLD) ||
723           abs(((g_imuDataBuffer[g_imuReadPosition].imuData.acc.z - g_accOffset.z)) > ACC_CALIB_THRESHOLD)) {
724               UARTprintf("\b\b\b\b\bdone]\n");
725             axMid += g_imuDataBuffer[g_imuReadPosition].imuData.acc.x;
726             ayMid += g_imuDataBuffer[g_imuReadPosition].imuData.acc.y;
727             azMid += g_imuDataBuffer[g_imuReadPosition].imuData.acc.z;
728           } else {
729             i = 0;
730             axMid = ayMid = azMid = 0;
731             lcdGuiUpdateLogLine("Please accelerate vehicle!........[busy]");
732           }
733         }
734       imuPacketRead();
735       SysCtlDelay(58593);   //TODO: Define this - measurement period ~ 1/512s - measurement time ~ 0.25s
736     }
737     g_accOffset.x = axMid / 128;
738     g_accOffset.y = ayMid / 128;
739     g_accOffset.z = azMid / 128;
740
741     //Create roll-axis vector file
742     UARTprintf("Creating file /roll/acc.mat........[busy]");
743     lcdGuiAppendLogLine("Creating file /roll/acc.mat........[busy]");
744
745     //Create folder if neccessary
746     f_mkdir("/roll");
747
748     if (matFileMatrixCreate(&g_offsetFile, "/roll/acc.mat", "accroll", 1, 3, MAT_TYPE_INT16) == FR_OK) {
749       UARTprintf("\b\b\b\b\bdone]\n");
750       lcdGuiUpdateLogLine("Creating file /roll/acc.mat........[done]");
751     } else {
752       UARTprintf("\b\b\b\b\bfailed]\n");
753       lcdGuiUpdateLogLine("Creating file /roll/acc.mat........[failed]");
754       g_progState = mainFileInitError;
755       return;
756     }
757
758     UARTprintf("Writing acc roll vector........[busy]");
759     lcdGuiUpdateLogLine("Writing acc roll vector........[busy]");
760     //Write offset vector to file
761     g_fatFsResult |= matFileAddValue(&g_offsetFile, (uint8_t*)&(g_accOffset.z));
762
763     //Close mat file
764     matFileClose(&g_offsetFile);
765
766     UARTprintf("\b\b\b\b\bdone]\n");
767     lcdGuiUpdateLogLine("Writing acc roll vector........[done]");
768
769     ledsWrite(ledsRead() & ~LED_YELLOW);
770   }
771
772   // Setup files and prepare modules for logging
773   inline void initLogging(void) {
774     //Iterator
775     uint32_t i, j;
776
777     //Multipurpose string buffer
778     char stringBuffer[512];
779     char stringBuffer2[512];
780
781     //FatFS
782     FILINFO nfo;
783
784     //LED code for "busy"
785     ledsWrite(LED_GREEN | LED_YELLOW);
786
787     //Mount the SD card file system
788     UARTprintf("Initializing SD card file system........[busy]");
789     lcdGuiAppendLogLine("Initializing SD card file system........[busy]");
790     if(f_mount(0, &g_fatFs) == FR_OK) {
791       UARTprintf("\b\b\b\b\bdone]\n");
792       lcdGuiUpdateLogLine("Initializing SD card file system........[done]");
793     } else {
794       UARTprintf("\b\b\b\b\bfailed]\n");
795       lcdGuiUpdateLogLine("Initializing SD card file system........[failed]");
796       g_progState = mainSDError;
797       return;
798     }
799
800     //Identify the next unused folder
801     UARTprintf("Creating folder........[busy]");
```

```
802     lcdGuiAppendLogLine("Creating folder........[busy]");
803     for (j=0; j<255; j++) {
804       sprintf(stringBuffer, "/%d", j);
805       if (f_stat(stringBuffer, &nfo) != FR_OK)
806         break;
807     }
808     //Create the folder when it is not the last one available
809     if ((j >= 254) || (f_mkdir(stringBuffer) != FR_OK)) {
810       UARTprintf("\b\b\b\b\bfailed]\n");
811       lcdGuiUpdateLogLine("Creating folder........[failed]");
812       g_progState = mainFileInitError;
813       return;
814     }
815     UARTprintf("\b\b\b\b\bdone]\n");
816     lcdGuiUpdateLogLine("Creating folder........[done]");
817
818     //Create the needed MAT files
819     for (i = 0; i < MAT_FILE_COUNT; i++) {
820       sprintf(stringBuffer, "/%d/%s", j, g_matFileNames[i]);
821       sprintf(stringBuffer2, "Creating file %s........[busy]", stringBuffer);
822       UARTprintf("%s", stringBuffer2);
823       lcdGuiUpdateLogLine(stringBuffer2);
824       if (matFileMatrixCreate(&(matFiles[i]), stringBuffer, g_matVariableNames[i], g_matFilePrealloc[i], g_matFileRows[i], ↩
              g_matFileTypes[i]) == FR_OK) {
825         UARTprintf("\b\b\b\b\bdone]\n");
826         sprintf(stringBuffer2, "Creating file %s........[done]", stringBuffer);
827         lcdGuiUpdateLogLine(stringBuffer2);
828       } else {
829         UARTprintf("\b\b\b\b\bfailed]\n");
830         sprintf(stringBuffer2, "Creating file %s........[failed]", stringBuffer);
831         lcdGuiUpdateLogLine(stringBuffer2);
832         g_progState = mainFileInitError;
833         return;
834       }
835     }
836
837     if (g_progState == mainFileInitError)
838       return;
839
840     //Start logging, if all files could be created
841     UARTprintf("Initiating logging process........[busy]");
842     lcdGuiAppendLogLine("Initiating logging process........[busy]");
843     //Reset file access error flag
844     g_fatFsResult = FR_OK;
845     //Filter CFG messages while measuring
846     ubxFilterClassSet(UBX_CFG);
847     //Clear GPS buffers
848     ubxInitBuffers(gps1);
849     ubxInitBuffers(gps2);
850     //Enable measurement timebase
851     gpsTimepulseIntEnable();
852     //Reset relative time counter
853     g_relTimeMs = 0;
854     UARTprintf("\b\b\b\b\bdone]\n");
855     lcdGuiUpdateLogLine("Initiating logging process........[done]");
856     UARTprintf("\nDataLogging started!\nREMOVING THE CARD WILL KILL YOUR DATA!\nPress 's' or button2 to stop.\n\n");
857     //UI
858     ledsWrite(LED_GREEN);
859   }
860
861   //File transfer sub-fsm
862   inline void fileTransfer(void) {
863     //Iterator
864     uint32_t i;
865     //Variables for file transfer
866     FILINFO nfo, subnfo;
867     DIR dir, subdir;
868     FIL fil;
869     uint32_t sizeAll;
870     uint32_t fileSize;
871     uint8_t fileNameLen;
872     char path[255];
873     uint8_t fileCount, folderCount;
874     uint8_t byte;
875     uint32_t bytesRead;
876
877     switch (g_transferState) {
878     case fileTransferInactive:
879       break;
880     case fileTransferActive:
881       //Mount the SD card file system
882       if(f_mount(0, &g_fatFs) != FR_OK) {
883         lcdGuiHideTransferActive();
884         g_progState = mainSDError;
885         break;
```

```
886       }
887
888       //Summarize all file sizes
889       sizeAll = 0;
890       fileCount = 0;
891       folderCount = 0;
892       if (f_opendir(&dir, "/") == FR_OK)
893         while ((f_readdir(&dir, &nfo) == FR_OK) && (nfo.fname[0] != 0)) {
894           if (nfo.fname[0] == '.')
895             continue;
896           //Is it a directory?
897           if (nfo.fattrib & AM_DIR) {
898             folderCount++;
899             if (f_opendir(&subdir, nfo.fname) == FR_OK)
900               while ((f_readdir(&subdir, &subnfo) == FR_OK) & (subnfo.fname[0] != 0)) {
901                 if (subnfo.fname[0] == '.')
902                   continue;
903                 //Is it a file?
904                 if (!(subnfo.fattrib & AM_DIR)) {
905                   sizeAll += subnfo.fsize;
906                   fileCount++;
907                 }
908               }
909           }
910         }
911
912       //Send start sequence
913       for (i=0; i<6; i++) {
914         UARTCharPut(UART2_BASE, 0xFF);
915         UARTCharPut(UART2_BASE, 0xAA);
916       }
917       g_transferState = fileTransferActive;
918
919         //Binary summary of all file sizes
920         for (i=0; i<4; i++)
921           UARTCharPut(UART2_BASE, *((uint8_t*)(&sizeAll)) + i));
922         //Binary count of directories to transmit
923       UARTCharPut(UART2_BASE, folderCount);
924
925       //Open root directory
926       if (f_opendir(&dir, "/") == FR_OK) {
927         while ((f_readdir(&dir, &nfo) == FR_OK) && (nfo.fname[0] != 0) && (UARTCharGetNonBlocking(UART2_BASE) != 's')) {
928             if (nfo.fname[0] == '.')
929               continue;
930             //Is it a directory?
931             if (nfo.fattrib & AM_DIR) {
932               fileNameLen = strlen(nfo.fname);
933               if (f_opendir(&subdir, nfo.fname) == FR_OK) {
934                   //Binary directory name length
935                   UARTCharPut(UART2_BASE, fileNameLen);
936                   //ASCII directory name
937                   UARTprintf("%s", nfo.fname);
938                   //Identify count of files in this directory
939                   fileCount = 0;
940               while ((f_readdir(&subdir, &subnfo) == FR_OK) & (subnfo.fname[0] != 0)) {
941                 if (subnfo.fname[0] == '.')
942                   continue;
943                 //Is it a file?
944                 if (!(subnfo.fattrib & AM_DIR))
945                   fileCount++;
946               }
947                 //Binary count of files to transmit
948               UARTCharPut(UART2_BASE, fileCount);
949
950               //Transfer files
951               f_readdir(&subdir, NULL);
952               while ((f_readdir(&subdir, &subnfo) == FR_OK) & (subnfo.fname[0] != 0)) {
953                 if (subnfo.fname[0] == '.')
954                   continue;
955                 //Is it a file?
956                 if (!(subnfo.fattrib & AM_DIR)) {
957                         fileNameLen = strlen(subnfo.fname);
958                         sprintf(path, "%s/%s", nfo.fname, subnfo.fname);
959                   if (f_open(&fil, path, FA_READ) != FR_OK)
960                     continue;
961                       fileSize = f_size(&fil);
962                   ////Transfer
963                   //Binary file name length
964                   UARTCharPut(UART2_BASE, fileNameLen);
965                   //ASCII file name
966                   UARTprintf("%s", subnfo.fname);
967                   //Binary file size in bytes
968                   for (i=0; i<4; i++)
969                     UARTCharPut(UART2_BASE, *((uint8_t*)(&fileSize)) + i));
970                   //Binary file content byte wise
```

```
971                 for (i=0; i<fileSize; i++) {
972                   f_read(&fil, &byte, 1, &bytesRead);
973                   UARTCharPut(UART2_BASE, byte);
974                 }
975                 f_close(&fil);
976               }
977             }
978             }
979           }
980       }
981     }
982     g_transferState = fileTransferInactive;
983     break;
984   case fileTransferDelete:
985     //Delete all files and folders
986     //Mount the SD card file system
987     if(f_mount(0, &g_fatFs) != FR_OK) {
988       g_progState = mainSDError;
989       break;
990     }
991
992     //Walk through directories and subdirectories and delete it all.
993     //This is non recursive on purpose and restricts deletion to a maximum depth of 2
994     if (f_opendir(&dir, "/") == FR_OK)
995       while ((f_readdir(&dir, &nfo) == FR_OK) && (nfo.fname[0] != 0)) {
996         if (nfo.fname[0] == '.')
997           continue;
998         //Is it a directory?
999         if (nfo.fattrib & AM_DIR) {
1000          if (f_opendir(&subdir, nfo.fname) == FR_OK)
1001            while ((f_readdir(&subdir, &subnfo) == FR_OK) & (subnfo.fname[0] != 0)) {
1002              if (subnfo.fname[0] == '.')
1003                continue;
1004              sprintf(path, "%s/%s", nfo.fname, subnfo.fname);
1005              //Try to delete
1006              f_unlink(path);
1007            }
1008        }
1009        //Delete
1010        f_unlink(nfo.fname);
1011      }
1012    g_transferState = fileTransferInactive;
1013    break;
1014  default:
1015    g_transferState = fileTransferInactive;
1016    break;
1017  }
1018 }
1019
1020 /*
1021  * Main
1022  */
1023 int main(void) {
1024   uint32_t i;
1025
1026   //Set system clock to 120MHz
1027     g_sysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
1028             SYSCTL_OSC_MAIN |
1029             SYSCTL_USE_PLL |
1030             SYSCTL_CFG_VCO_480), 120000000);
1031
1032     //TLC4412 STAT pin
1033   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
1034   GPIOPinTypeGPIOInput(GPIO_PORTK_BASE, GPIO_PIN_3);
1035   //This high priority interrupt indicates when power is cut off
1036   GPIOIntTypeSet(GPIO_PORTK_BASE, GPIO_PIN_3, GPIO_FALLING_EDGE);
1037   GPIOIntRegister(GPIO_PORTK_BASE, pushButtonIsr);
1038   GPIOIntEnable(GPIO_PORTK_BASE, GPIO_PIN_3);
1039   IntPrioritySet(INT_GPIOK, 1);
1040
1041   //TLC4412 CTL pin
1042   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
1043   GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_2);
1044
1045   //Make sure LCD backlight is off
1046   SysCtlPeripheralEnable(LCD_BACKLIGHT_PERIPH);
1047   GPIOPinTypeGPIOOutput(LCD_BACKLIGHT_BASE, LCD_BACKLIGHT_PIN);
1048   LED_OFF();
1049
1050     //Configure UART for virtual serial port
1051     UARTStdioConfig(2, 921600, g_sysClock);
1052
1053     //Check if power is available and a startup "request" has been received
1054     //Startup is initiated by pulling TLC4412 CTL pin to low level or using UART char 'g'
1055     //Startup is not possible when TLC4412 STAT pin has low level (explicit shutdown conditions
```

```
1056      //e.g. system is powered only from the goldcap)
1057    while ((GPIOPinRead(GPIO_PORTA_BASE, GPIO_PIN_2) || !GPIOPinRead(GPIO_PORTK_BASE, GPIO_PIN_3))
1058        && UARTCharGetNonBlocking(UART2_BASE) != 'g')
1059      ;
1060
1061    GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_2);
1062    GPIOPadConfigSet(GPIO_PORTA_BASE, GPIO_PIN_2, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD);
1063    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_2, 0);
1064
1065      //Main loop
1066      while (42) {
1067        WidgetMessageQueueProcess();
1068        switch (g_progState) {
1069        case mainInit:
1070          initModules();
1071          g_progState = mainStart;
1072          break;
1073        case mainStart:
1074        ledsWrite(LED_GREEN);
1075        UARTprintf("\nDataLogging will begin, when you press 'g' or button1!\n");
1076        lcdGuiShowMainMenu();
1077        g_progState = mainWaitStart;
1078          break;
1079        case mainWaitStart:
1080          break;
1081        case mainImuCalibrate:
1082          calibrateImu();
1083          g_progState = mainStart;
1084          break;
1085        case mainRecordStart:
1086          initLogging();
1087        g_progState = mainRecord;
1088        break;
1089        case mainRecord:
1090        writeData();
1091        break;
1092        case mainRecordStop:
1093        ledsWrite(LED_GREEN | LED_YELLOW);
1094        lcdGuiAppendLogLine("Stop logging........[busy]");
1095        //Disable measurement timebase
1096        gpsTimepulseIntDisable();
1097        //Close all MAT files
1098        for (i=0; i<MAT_FILE_COUNT; i++)
1099          matFileClose(&(matFiles[i]));
1100        UARTprintf("\n\nFiles closed.\nRemoval of card is safe now!\n");
1101        lcdGuiUpdateLogLine("Stop logging........[done]");
1102        g_progState = mainStart;
1103          break;
1104        case mainShutdown:
1105          //Cut power
1106          GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_2, GPIO_PIN_2);
1107          //Display backlight off
1108          LED_OFF();
1109          //uC software Reset
1110          HWREG(NVIC_APINT) = NVIC_APINT_VECTKEY | NVIC_APINT_SYSRESETREQ;
1111          break;
1112        case mainInitError:
1113          ledsWrite(LED_RED);
1114          SysCtlDelay(80000000);
1115          g_progState = mainShutdown;
1116          break;
1117        case mainSDError:
1118          ledsWrite(LED_RED);
1119          SysCtlDelay(80000000);
1120          g_progState = mainStart;
1121          break;
1122        case mainFileInitError:
1123          ledsWrite(LED_RED);
1124          SysCtlDelay(80000000);
1125          g_progState = mainStart;
1126        break;
1127        case mainFileWriteError:
1128        ledsWrite(LED_RED);
1129        //Disable measurement timebase
1130        gpsTimepulseIntDisable();
1131        //Close all MAT files
1132        for (i=0; i<MAT_FILE_COUNT; i++)
1133          matFileClose(&(matFiles[i]));
1134        UARTprintf("\n\nFiles closed because of an error!\nWHAT DID I SAY ABOUT REMOVING THE CARD?\n\n");
1135        g_progState = mainStart;
1136          break;
1137        case mainFileTransfer:
1138          fileTransfer();
1139          break;
1140        default:
```

```
1141            break;
1142          }
1143        }
1144    }
```

D.2. Software-Module

Quellcode D.2: Software-Modul: Inertialmesssystem (Header)

```
1   /*
2    * imu.h
3    *
4    * Author: Felix Groth
5    */
6
7   #ifndef IMU_H_
8   #define IMU_H_
9
10  #include <stdbool.h>
11  #include <stdint.h>
12  #include "inc/tm4c1294ncpdt.h"
13  #include "inc/hw_memmap.h"
14
15  #include "inc/hw_types.h"
16  #include "inc/hw_gpio.h"
17  #include "inc/hw_i2c.h"
18
19  #include "driverlib/sysctl.h"
20  #include "driverlib/rom.h"
21  #include "driverlib/gpio.h"
22  #include "driverlib/pin_map.h"
23  #include "driverlib/i2c.h"
24  #include "driverlib/interrupt.h"
25
26  //IMU Ringbuffer
27  #define IMU_RINGBUFFER_SIZE    250
28
29  //IMU tweakable configuration
30  #define IMU_CONF_AFS   MPU_VAL_AFS_2G
31  #define IMU_ACC_1G_VAL 32767/(0x02<<IMU_CONF_AFS) //Value for 1g depending on full scale
32  #define IMU_CONF_NUM_CALIB 8 //As in: Calibration with 2^N measurements
33  #define IMU_CONF_GFS MPU_VAL_FS_2000
34
35
36  /** IMU Hardware port settings **/
37  #define IMU_PERIPH_I2C      SYSCTL_PERIPH_I2C2
38  #define IMU_PERIPH_GPIO     SYSCTL_PERIPH_GPION
39  #define IMU_PINMUX_SCL      GPIO_PN5_I2C2SCL
40  #define IMU_PINMUX_SDA      GPIO_PN4_I2C2SDA
41  #define IMU_PIN_SCL         GPIO_PIN_5
42  #define IMU_PIN_SDA         GPIO_PIN_4
43  #define IMU_PORT_BASE       GPIO_PORTN_BASE
44  #define IMU_I2C_BASE        I2C2_BASE
45  #define IMU_I2C_ADDRESS     MPU_I2C_ADDRESS
46  #define IMU_I2C_INT         INT_I2C2
47  #define IMU_ENABLE_PERIPH   SYSCTL_PERIPH_GPIOB
48  #define IMU_ENABLE_BASE     GPIO_PORTB_BASE
49  #define IMU_ENABLE_PIN      GPIO_PIN_4
50
51  /** IMU function returns **/
52  #define IMU_COM_READY           0x71
53  #define IMU_COM_FAIL            0x00
54  #define IMU_BUFFER_OK           0x00
55  #define IMU_BUFFER_OVF            0x01
56
57  //MPU address
58  #define MPU_I2C_ADDRESS                 0x68
59
60  //MPU config registers
61  #define MPU_REG_WHO_AM_I        0x75
62
63  #define MPU_REG_PWR_MGMT_1              0x6B
64  #define MPU_MASK_CLKSEL                 0x07
65  #define MPU_MASK_SLEEP                  0x40
66
67  #define MPU_REG_CONFIG                  0x1A
68  #define MPU_MASK_DLPF_CFG               0x07
```

```
69
70  #define MPU_REG_ACCEL_CONFIG2       0x1D
71  #define MPU_MASK_A_DLPF_CFG                 0x07
72
73  #define MPU_REG_GYRO_CONFIG                 0x1B
74  #define MPU_MASK_FSSEL                      0x18
75
76  #define MPU_REG_ACCEL_CONFIG                0x1C
77  #define MPU_MASK_AFSSEL                     0x18
78
79  #define MPU_REG_INT_PIN_CFG                 0x37
80  #define MPU_BIT_I2C_BYPASS_EN               0x01
81  #define MPU_LEN_I2C_BYPASS_EN               0x01
82
83  #define MPU_REG_INT_STATUS          0x3A
84  #define MPU_MASK_RAW_DATA_RDY_INT     0x01
85
86  #define MPU_REG_INT_ENABLE          0x38
87  #define MPU_MASK_RAW_RDY_EN         0x01
88
89  //Config register values
90  #define MPU_VAL_WHO_AM_I          0x71
91
92  #define MPU_VAL_CLK_SEL_AUTO        0x01
93  #define MPU_VAL_SLEEP_DIS         0x00
94
95  #define MPU_VAL_DLPF_184                    0x01
96  #define MPU_VAL_DLPF_5            0x06
97
98  #define MPU_VAL_A_DLPF_184          0x01
99
100 #define MPU_VAL_AFS_2G                      0x00
101 #define MPU_VAL_AFS_4G                      0x08
102 #define MPU_VAL_AFS_8G                      0x10
103 #define MPU_VAL_AFS_16G                     0x18
104
105 #define MPU_VAL_FS_2000                     0x18
106 #define MPU_VAL_FS_1000                     0x10
107 #define MPU_VAL_FS_500                      0x08
108 #define MPU_VAL_FS_250                      0x00
109
110 //Data registers
111 #define MPU_REG_ACCEL_XOUT_H                0x3B
112 #define MPU_REG_ACCEL_XOUT_L                0x3C
113 #define MPU_REG_ACCEL_YOUT_H                0x3D
114 #define MPU_REG_ACCEL_YOUT_L                0x3E
115 #define MPU_REG_ACCEL_ZOUT_H                0x3F
116 #define MPU_REG_ACCEL_ZOUT_L                0x40
117 #define MPU_REG_TEMP_OUT_H                  0x41
118 #define MPU_REG_TEMP_OUT_L                  0x42
119 #define MPU_REG_GYRO_XOUT_H                 0x43
120 #define MPU_REG_GYRO_XOUT_L                 0x44
121 #define MPU_REG_GYRO_YOUT_H                 0x45
122 #define MPU_REG_GYRO_YOUT_L                 0x46
123 #define MPU_REG_GYRO_ZOUT_H                 0x47
124 #define MPU_REG_GYRO_ZOUT_L                 0x48
125
126 //Magnetometer config
127 #define MPU_I2C_MAG_ADDRESS                 0x0C
128 //Magnetometer config registers and bit positions
129 #define MPU_REG_MAG_CNTL                    0x0A
130 #define MPU_REG_MAG_ST1                     0x02
131 #define MPU_BIT_MAG_DRDY                    0x00
132 //Magnetometer Config register values
133 #define MPU_VAL_MAG_MEAS                    0x01
134 //Magnetometer data registers
135 #define MPU_REG_MAG_XOUT_L                  0x03
136 #define MPU_REG_MAG_XOUT_H                  0x04
137 #define MPU_REG_MAG_YOUT_L                  0x05
138 #define MPU_REG_MAG_YOUT_H                  0x06
139 #define MPU_REG_MAG_ZOUT_L                  0x07
140 #define MPU_REG_MAG_ZOUT_H                  0x08
141
142
143
144 typedef struct MemsData {
145   int16_t z;
146   int16_t y;
147   int16_t x;
148 } MemsData;
149
150 /*****************************************************************/
151 /* Type: Struct                                                  */
152 /* Name: temp                                                    */
153 /* Description: Data structure to hold a single 16-bit temperature */
```

```
154  /*              measurement.                                            */
155  /*****************************************************************/
156  struct Temp {
157    int16_t meas;
158   };
159
160  /*****************************************************************/
161  /* Type: Struct                                                      */
162  /* Name: imuData                                                     */
163  /* Description: Holds 9DOF data from the whole IMU plus one temperature */
164  /*              measurement.                                         */
165  /*****************************************************************/
166  typedef struct {
167    struct MemsData gyr; //Gyroscope data
168    struct Temp     tmp; //Temperature data
169    struct MemsData acc; //Accelerometer data
170    //struct MemsData mag; //Magnetometer data
171  } ImuData;
172
173  typedef enum {
174    imuFree = 0,
175    imuBusy = 1,
176    imuDataReady = 2
177  } ImuBufferState;
178
179  typedef struct {
180    ImuData imuData;
181    ImuBufferState state;
182    uint8_t* readPosition;
183    uint32_t timestamp;
184  } ImuDataBuffer;
185
186  typedef struct {
187    uint8_t buffer[12];
188    uint8_t* position;
189    ImuBufferState state;
190  } ImuTxBuffer;
191
192  uint8_t imuInitI2C(void);
193  uint8_t imuReadByte(uint8_t addr);
194  void imuWriteByte(uint8_t addr, uint8_t data);
195  void imuReadModifyWrite(uint8_t addr, uint8_t mask, uint8_t value);
196  void imuReadBuffer(uint8_t addr, uint8_t* buffer, uint8_t len);
197  void imuAddressRegister(uint8_t addr);
198  void imuWaitReady(void);
199  void imuReadBufferReverse(uint8_t addr, uint8_t* buffer, uint8_t len);
200  void imuPollMeasurement(ImuData* imuData);
201  void imuInit(uint8_t gyroFullScale, uint8_t accelFullScale);
202
203  void imuMeasurementRcvIsr(void);
204  void imuSetupRcvInt(uint8_t priority);
205  uint8_t imuTriggerMeasurement(void);
206  void imuPacketRead(void);
207
208  #endif /* IMU_H_ */
```

Quellcode D.3: Software-Modul: Inertialmesssystem)

```
1  /*
2   * imu.c
3   *
4   * Author: Felix Groth
5   */
6
7  #include "imu.h"
8
9  extern unsigned int g_sysClock;
10 //Global relaive time in ms
11 extern volatile uint32_t g_relTimeMs;
12 volatile ImuDataBuffer g_imuDataBuffer[IMU_RINGBUFFER_SIZE];
13 //volatile ImuDataBuffer* g_imuReadBuffer = g_imuDataBuffer;
14 //volatile ImuDataBuffer* g_imuWriteBuffer = g_imuDataBuffer;
15 volatile uint8_t g_imuReadPosition = 0;
16 volatile uint8_t g_imuWritePosition = 0;
17 volatile static ImuTxBuffer g_imuTxBuffer;
18 volatile static uint8_t g_addressed = 0;
19
20 uint8_t imuInitI2C(void) {
21     //enable I2C module 2
22     SysCtlPeripheralEnable(IMU_PERIPH_I2C);
23
```

```
24      //reset module
25      SysCtlPeripheralReset(IMU_PERIPH_I2C);
26
27      //enable GPIO peripheral that contains I2C 2
28      SysCtlPeripheralEnable(IMU_PERIPH_GPIO);
29
30      // Configure the pin muxing for I2C2 functions on port N4 and N5.
31      GPIOPinConfigure(IMU_PINMUX_SCL);
32      GPIOPinConfigure(IMU_PINMUX_SDA);
33
34      // Select the I2C function for these pins.
35      GPIOPinTypeI2CSCL(IMU_PORT_BASE, IMU_PIN_SCL);
36      GPIOPinTypeI2C(IMU_PORT_BASE, IMU_PIN_SDA);
37
38      // Enable and initialize the I2C2 master module.  Use the system clock for
39      // the I2C2 module.  The last parameter sets the I2C data transfer rate.
40      // If false the data rate is set to 100kbps and if true the data rate will
41      // be set to 400kbps.
42      I2CMasterInitExpClk(IMU_I2C_BASE, SysCtlClockGet(), true);
43
44      //clear I2C FIFO -- TODO: Is this needed?
45      HWREG(IMU_I2C_BASE + I2C_O_FIFOCTL) = 80008000;
46
47      //Make IMU available
48      SysCtlPeripheralEnable(IMU_ENABLE_PERIPH);
49      GPIOPinTypeGPIOOutput(IMU_ENABLE_BASE, IMU_ENABLE_PIN);
50      GPIOPinWrite(IMU_ENABLE_BASE, IMU_ENABLE_PIN, IMU_ENABLE_PIN);
51
52      //Check if IMU answers correctly
53      return imuReadByte(MPU_REG_WHO_AM_I); //Defined as IMU_COM_READY or IMU_COM_FAIL
54  }
55
56
57  void imuAddressRegisterNoWait(uint8_t addr) {
58    //specify that we are writing (a register address) to the slave device
59    I2CMasterSlaveAddrSet(IMU_I2C_BASE, IMU_I2C_ADDRESS, false);
60
61    //specify register to be read
62    I2CMasterDataPut(IMU_I2C_BASE, addr);
63
64    //send control byte and register address byte to slave device
65    I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_BURST_SEND_START);
66  }
67
68
69  void imuAddressRegister(uint8_t addr) {
70    //specify that we are writing (a register address) to the slave device
71    I2CMasterSlaveAddrSet(IMU_I2C_BASE, IMU_I2C_ADDRESS, false);
72
73    //specify register to be read
74    I2CMasterDataPut(IMU_I2C_BASE, addr);
75
76    //send control byte and register address byte to slave device
77    I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_BURST_SEND_START);
78
79    //wait for MCU to finish transaction
80    //TODO: This is busy waiting!
81    while(I2CMasterBusy(IMU_I2C_BASE));
82  }
83
84  uint8_t imuReadByte(uint8_t addr) {
85    //***READ
86    imuAddressRegister(addr);
87
88    //if(error = I2CMasterErr(I2C2_BASE))
89    //  UARTprintf("IMU I2C error: %x...\n", error);
90
91    //specify that we are going to read from slave device
92    I2CMasterSlaveAddrSet(IMU_I2C_BASE, MPU_I2C_ADDRESS, true);
93
94    //send control byte and read from the register we specified
95    I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
96
97    //wait for MCU to finish transaction
98    //TODO: This is busy waiting!
99    while(I2CMasterBusy(I2C2_BASE));
100
101   //if(error = I2CMasterErr(I2C2_BASE))
102   //  UARTprintf("IMU I2C error: %x...\n", error);
103
104   //return data pulled from the specified register
105   return I2CMasterDataGet(I2C2_BASE);
106 }
107
108 void imuWriteByte(uint8_t addr, uint8_t data) {
```

```
109     //**WRITE
110     imuAddressRegister(addr);
111
112     //if(error = I2CMasterErr(IMU_I2C_BASE))
113     //  UARTprintf("IMU I2C error: %x...\n", error);
114
115     //specify that we are writing to the slave device
116     I2CMasterSlaveAddrSet(IMU_I2C_BASE, MPU_I2C_ADDRESS, false);
117
118     //specify the data to be written
119     I2CMasterDataPut(IMU_I2C_BASE, data);
120
121     //send control byte and register data byte to slave device
122     I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
123
124     //wait for MCU to finish transaction
125     while(I2CMasterBusy(IMU_I2C_BASE));
126
127     //if(error = I2CMasterErr(I2C2_BASE))
128     //  UARTprintf("IMU I2C error: %x...\n", error);
129   }
130
131   void imuReadModifyWrite(uint8_t addr, uint8_t mask, uint8_t value) {
132     uint8_t data;
133
134     //***READ
135     data = imuReadByte(addr);
136     //***MODIFY
137     data |= (value & mask);
138     data &= ~(~value & mask);
139
140     //**WRITE
141     imuWriteByte(addr, data);
142   }
143
144   void imuWaitReady(void) {
145     //Wait for the data to be ready on IMU
146     while (!(imuReadByte(MPU_REG_INT_STATUS) & MPU_MASK_RAW_DATA_RDY_INT))
147       ;
148   }
149
150   void imuReadBuffer(uint8_t addr, uint8_t* buffer, uint8_t len) {
151     imuAddressRegister(addr);
152
153     //if(error = I2CMasterErr(I2C2_BASE))
154       //UARTprintf("IMU I2C error: %x...\n", error);
155
156     //***READ DATA FROM IMU
157     //specify that we are going to read from slave device
158     I2CMasterSlaveAddrSet(IMU_I2C_BASE, IMU_I2C_ADDRESS, true);
159
160     //send control byte and read from the register we specified
161     I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
162
163     //wait for MCU to finish transaction
164     while(I2CMasterBusy(IMU_I2C_BASE));
165
166     //Fill received data into the receive buffer
167     *(buffer++) =  I2CMasterDataGet(IMU_I2C_BASE);
168
169     for(len = len-2; len>0; len--) {
170       //send control byte and read from the register we specified
171       I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT);
172
173       //wait for MCU to finish transaction
174       while(I2CMasterBusy(IMU_I2C_BASE));
175
176       //Fill data we received into our buffer
177       *(buffer++) =  I2CMasterDataGet(IMU_I2C_BASE);
178     }
179
180     //send control byte and read from the register we specified
181     I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
182
183     //wait for MCU to finish transaction
184     while(I2CMasterBusy(IMU_I2C_BASE));
185
186     //Fill data we received into our buffer
187     *buffer = I2CMasterDataGet(IMU_I2C_BASE);
188   }
189
190   void imuReadBufferReverse(uint8_t addr, uint8_t* buffer, uint8_t len) {
191     //Prepare buffer pointer for reverse writing
192     buffer += (len-1);
193
```

```
194     //Address the register
195     imuAddressRegister(addr);
196
197     //if(error = I2CMasterErr(I2C2_BASE))
198       //UARTprintf("IMU I2C error: %x...\n", error);
199
200     //***READ DATA FROM IMU
201     //specify that we are going to read from slave device
202     I2CMasterSlaveAddrSet(IMU_I2C_BASE, IMU_I2C_ADDRESS, true);
203
204     //send control byte and read from the register we specified
205     I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
206
207     //wait for MCU to finish transaction
208     while(I2CMasterBusy(IMU_I2C_BASE));
209
210     //Fill received data into the receive buffer
211     *(buffer--) =  I2CMasterDataGet(IMU_I2C_BASE);
212
213     for(len = len-2; len>0; len--) {
214       //send control byte and read from the register we specified
215       I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT);
216
217       //wait for MCU to finish transaction
218       while(I2CMasterBusy(IMU_I2C_BASE));
219
220       //Fill data we received into our buffer
221       *(buffer--) =  I2CMasterDataGet(IMU_I2C_BASE);
222     }
223
224     //send control byte and read from the register we specified
225     I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
226
227     //wait for MCU to finish transaction
228     while(I2CMasterBusy(IMU_I2C_BASE));
229
230     //Fill data we received into our buffer
231     *buffer = I2CMasterDataGet(IMU_I2C_BASE);
232   }
233
234   void imuPollMeasurement(ImuData* imuData) {
235     //Wait for the IMU to get the current measurement done
236     imuWaitReady();
237     //Reading reverse because the IMU is big endian -- the easiest way of byte swapping
238     imuReadBufferReverse(MPU_REG_ACCEL_XOUT_H, (uint8_t*)imuData, sizeof(ImuData));
239   }
240
241
242   //WARNING: This needs the FIFO threshold to be an int divisor to the number of bytes to read
243   void imuMeasurementRcvIsr(void) {
244     uint32_t status;
245     uint8_t bytesRead = 0;
246
247     status = I2CMasterIntStatusEx(IMU_I2C_BASE, true);
248     I2CMasterIntClearEx(IMU_I2C_BASE, status);
249
250     if ((status & I2C_MASTER_INT_DATA) && g_addressed) {
251       g_addressed = 0;
252       //Read 14 bytes from I2C slave
253       I2CMasterBurstLengthSet(IMU_I2C_BASE, 14);     //TODO: Define this
254       //Where to read the data from
255       I2CMasterSlaveAddrSet(IMU_I2C_BASE, IMU_I2C_ADDRESS, true);
256       //This is as close as we get to a good relative timestamp for this measurement
257       g_imuDataBuffer[g_imuWritePosition].timestamp = g_relTimeMs;
258       //Start burst receive using FIFO_SINGLE_RECEIVE
259       //This seems to be a workaround, because it looses bus arbitration between bytes
260       I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_FIFO_SINGLE_RECEIVE);
261     }
262
263     if (status & I2C_MASTER_INT_RX_FIFO_REQ) {
264       while (I2CFIFODataGetNonBlocking(IMU_I2C_BASE, g_imuDataBuffer[g_imuWritePosition].readPosition)) {
265         g_imuDataBuffer[g_imuWritePosition].readPosition--;
266         bytesRead++;
267       }
268       if (!I2CMasterBurstCountGet(IMU_I2C_BASE) && bytesRead) {
269         //End the transfer for the slave without reading another byte.
270         I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_FIFO_BURST_RECEIVE_FINISH);
271         g_imuDataBuffer[g_imuWritePosition].state = imuDataReady;
272         //Shift ring buffer write position
273         g_imuWritePosition++;
274         g_imuWritePosition %= IMU_RINGBUFFER_SIZE;
275         /*if ((g_imuWriteBuffer - g_imuDataBuffer) < IMU_RINGBUFFER_SIZE-1)
276           g_imuWriteBuffer++;
277         else
278           g_imuWriteBuffer = g_imuDataBuffer;*/
```

```
279       }
280     }
281   }
282
283   void imuSetupRcvInt(uint8_t priority) {
284     //g_imuDataBuffer = imuDataBuffer;
285     I2CRxFIFOFlush(IMU_I2C_BASE);
286     //Setup receive FIFO to trigger the interrupt when 7 bytes are received
287     //We can do this exactly twice to receive all 14 bytes of measurement data
288     I2CRxFIFOConfigSet(IMU_I2C_BASE, I2C_FIFO_CFG_RX_MASTER | I2C_FIFO_CFG_RX_TRIG_7);
289     I2CRxFIFOFlush(IMU_I2C_BASE);
290     I2CIntRegister(IMU_I2C_BASE, imuMeasurementRcvIsr);
291     IntPrioritySet(IMU_I2C_INT, priority);
292     I2CMasterIntEnableEx(IMU_I2C_BASE, I2C_MASTER_INT_RX_FIFO_REQ | I2C_MASTER_INT_DATA);
293     //I2CMasterIntEnable(IMU_I2C_BASE);
294   }
295
296
297   //There is no checking if the buffer is available!
298   uint8_t imuTriggerMeasurement(void) {
299     if (g_imuDataBuffer[g_imuWritePosition].state == imuFree) {
300       //Setup the receive buffer
301       g_imuDataBuffer[g_imuWritePosition].readPosition = (uint8_t*)&(g_imuDataBuffer[g_imuWritePosition].imuData) + 13; //TODO: ↩
              Clarify this!
302       g_imuDataBuffer[g_imuWritePosition].state = imuBusy;
303       //Wait for the IMU to get the current measurement done
304       //imuWaitReady();
305       //Flush RX FIFO to make sure we start out clean.
306       I2CRxFIFOFlush(IMU_I2C_BASE);
307       //Address the start register
308       imuAddressRegisterNoWait(MPU_REG_ACCEL_XOUT_H);   //TODO: We might be able to further improve this
309       g_addressed = 1;
310       /*//Read 14 bytes from I2C slave
311       I2CMasterBurstLengthSet(IMU_I2C_BASE, 14);     //TODO: Define this
312       //Where to read the data from
313       I2CMasterSlaveAddrSet(IMU_I2C_BASE, IMU_I2C_ADDRESS, true);
314       //This is as close as we get to a good relative timestamp for this measurement
315       g_imuWriteBuffer->timestamp = g_relTimeMs;
316       //Start burst receive using FIFO_SINGLE_RECEIVE
317       //This seems to be a workaround, because it looses bus arbitration between bytes
318       I2CMasterControl(IMU_I2C_BASE, I2C_MASTER_CMD_FIFO_SINGLE_RECEIVE);*/
319
320       return IMU_BUFFER_OK;
321     } else
322       return IMU_BUFFER_OVF;
323   }
324
325   void imuPacketRead(void) {
326     //Mark as read
327     g_imuDataBuffer[g_imuReadPosition].state = imuFree;
328     //Shift ring buffer read position
329     g_imuReadPosition++;
330     g_imuReadPosition %= IMU_RINGBUFFER_SIZE;
331     /*if ((g_imuReadBuffer - g_imuDataBuffer) < IMU_RINGBUFFER_SIZE-1)
332       g_imuReadBuffer++;
333     else
334       g_imuReadBuffer = g_imuDataBuffer;*/
335   }
336
337   void imuInit(uint8_t gyroFullScale, uint8_t accelFullScale) {
338     //Configure IMU
339     //Select best (PLL) clock source automatically
340     imuReadModifyWrite(MPU_REG_PWR_MGMT_1, MPU_MASK_CLKSEL, MPU_VAL_CLK_SEL_AUTO);
341     //Set gyro full scale range
342     imuReadModifyWrite(MPU_REG_GYRO_CONFIG, MPU_MASK_FSSEL, gyroFullScale);
343     //Set accel full scale range
344     imuReadModifyWrite(MPU_REG_ACCEL_CONFIG, MPU_MASK_AFSSEL, accelFullScale);
345     //Disable sleep
346     imuReadModifyWrite(MPU_REG_PWR_MGMT_1, MPU_MASK_SLEEP, MPU_VAL_SLEEP_DIS);
347     //Set low pass filter to ~184Hz for gyro and accel
348     imuReadModifyWrite(MPU_REG_CONFIG, MPU_MASK_DLPF_CFG, MPU_VAL_DLPF_184);
349     imuReadModifyWrite(MPU_REG_ACCEL_CONFIG2, MPU_MASK_A_DLPF_CFG, MPU_VAL_A_DLPF_184);
350   }
```

Quellcode D.4: Software-Modul: 1-Wire (Header)

```
1   /*
2    * onewire.h
3    *
4    * Author: Felix Groth
5    */
```

```
6
7
8   #ifndef ONEWIRE_H_
9   #define ONEWIRE_H_
10
11
12  //Standard includes
13  #include <stdbool.h>
14  #include <stdint.h>
15
16  //Hardware includes
17  #include "inc/tm4c1294ncpdt.h"
18  #include "inc/hw_gpio.h"
19  #include "inc/hw_memmap.h"
20
21  //Driver includes
22  #include "driverlib/sysctl.h"
23  #include "driverlib/rom.h"
24  #include "driverlib/gpio.h"
25  #include "driverlib/uart.h"
26  #include "driverlib/pin_map.h"
27  #include "driverlib/interrupt.h"
28
29
30  /** Defines for one wire hardware port **/
31  #define ONEWIRE_UART_BASE     UART6_BASE
32  #define ONEWIRE_GPIO_BASE     GPIO_PORTP_BASE
33  #define ONEWIRE_GPIO_PERIPH     SYSCTL_PERIPH_GPIOP
34  #define ONEWIRE_UART_PERIPH     SYSCTL_PERIPH_UART6
35  #define ONEWIRE_UART_RX       GPIO_PP0_U6RX
36  #define ONEWIRE_UART_TX       GPIO_PP1_U6TX
37  #define ONEWIRE_UART_RX_PIN     GPIO_PIN_0
38  #define ONEWIRE_UART_TX_PIN     GPIO_PIN_1
39  #define ONEWIRE_PULLUP_PERIPH SYSCTL_PERIPH_GPIOA
40  #define ONEWIRE_PULLUP_BASE     GPIO_PORTA_BASE
41  #define ONEWIRE_PULLUP_PIN      GPIO_PIN_3
42  #define ONEWIRE_UART_INT      INT_UART6
43
44  #define ONEWIRE_ENABLE_PULLUP()     GPIOPinWrite(ONEWIRE_PULLUP_BASE, ONEWIRE_PULLUP_PIN, 0)
45  #define ONEWIRE_DISABLE_PULLUP()    GPIOPinWrite(ONEWIRE_PULLUP_BASE, ONEWIRE_PULLUP_PIN, ONEWIRE_PULLUP_PIN)
46
47
48  /** Defines for one wire protocol **/
49  // ROM commands
50  #define ONEWIRE_ROM_SEARCH      0xF0
51  #define ONEWIRE_ROM_READ      0x33
52  #define ONEWIRE_ROM_MATCH     0x55
53  #define ONEWIRE_ROM_SKIP      0xCC
54  #define ONEWIRE_ROM_ALARM     0xEC
55  // DS18B20 functions
56  #define ONEWIRE_DS18_CONVERT    0x44
57  #define ONEWIRE_DS18_READ     0xBE
58  // DS18B20 constants
59  #define ONEWIRE_DS18_FAMILY     0x28
60
61  /** Defines for function return values **/
62  #define ONEWIRE_INIT_SUCCESS    0x00
63  #define ONEWIRE_INIT_FAIL     0x01
64  #define ONEWIRE_CRC_FAIL      0x02
65
66
67  typedef union {
68    struct {
69      uint8_t family;
70      uint8_t serial[6];
71      uint8_t crc;
72    };
73    uint64_t raw;
74    //uint8_t rawBytes[8];
75  } maximROMCode;
76
77  typedef enum {
78    owFree,
79    owInit,
80    owWriting,
81    owReading,
82    owError
83  } onewireState;
84
85  /**
86   * WARNING!
87   * These dummy bytes in the struct are there for only one reason:
88   *    The maximROMCode struct is aligned to a 64-bit boundary.
89   *    Thus we need an offset for the buffer to be contignuos.
90   *  This is heavily compiler dependent and might break on another one.
```

```
91    *  Also take care changing the struct, you might break this!
92    * WARNING!
93    */
94   typedef struct {
95     uint8_t dummy[7];
96     uint8_t romMatch;
97     maximROMCode romCode;
98     uint8_t command;
99     uint8_t scratchpad[8];
100    //Multiple interpretations of the data
101    //DS18B20
102    uint16_t* temperature;
103  } onewireData;
104
105  typedef enum {
106    owBufferFree,
107    owBufferBusy,
108    owBufferReady
109  } onewireBufferState;
110
111  /**
112   * WARNING!
113   * These dummy bytes in the struct are there for only one reason:
114   *    The maximROMCode struct is aligned to a 64-bit boundary.
115   *    Thus we need an offset for the buffer to be contignuos.
116   *  This is heavily compiler dependent and might break on another one.
117   *  Also take care changing the struct, you might break this!
118   * WARNING!
119   */
120  typedef struct {
121    union {
122      struct {
123        uint8_t dummy[7];
124        uint8_t data[32];
125      };
126      onewireData onewireData;
127    };
128    uint8_t* position;
129    uint8_t writeLength;
130    uint8_t readLength;
131    onewireBufferState state;
132  } onewireBuffer;
133
134  void initOneWireUART(void);
135  uint8_t initOneWire(void);
136  void writeOneWire(uint8_t byte);
137  uint8_t readOneWire(void);
138  uint8_t searchOneWire(maximROMCode* devices, uint8_t maxDevices, uint8_t isAlarm);
139  uint8_t matchOneWireDevice(maximROMCode* addr);
140  uint8_t ds1820StartConversion(maximROMCode* addr);
141  uint8_t ds1820ReadTemperature(maximROMCode* addr, int16_t* temperature);
142  uint8_t ds1820GetCRC(uint8_t crc, uint8_t byte);
143  void onewireIsr(void);
144  void ds1820CmdInt(maximROMCode* addr, uint8_t command);
145  void onewireIntEnable(uint8_t priority);
146
147  void ds1820StartConversionInt(maximROMCode* addr);
148  void ds1820ReadTemperatureInt(maximROMCode* addr);
149
150
151  #endif /* ONEWIRE_H_ */
```

Quellcode D.5: Software-Modul: 1-Wire

```
1   /*
2    * onewire.c
3    *
4    * Author: Felix Groth
5    */
6
7   #include "onewire.h"
8
9   extern unsigned int g_sysClock;
10
11  //CRC Lookup table according to MAXIM appnote 27
12  const uint8_t g_maximCRCLookup[] = {0, 94, 188, 226, 97, 63, 221, 131, 194, 156, 126, 32, 163, 253, 31, 65,
13      157, 195, 33, 127, 252, 162, 64, 30, 95, 1, 227, 189, 62, 96, 130, 220,
14      35, 125, 159, 193, 66, 28, 254, 160, 225, 191, 93, 3, 128, 222, 60, 98,
15      190, 224, 2, 92, 223, 129, 99, 61, 124, 34, 192, 158, 29, 67, 161, 255,
16      70, 24, 250, 164, 39, 121, 155, 197, 132, 218, 56, 102, 229, 187, 89, 7,
17      219, 133, 103, 57, 186, 228, 6, 88, 25, 71, 165, 251, 120, 38, 196, 154,
```

```
18      101, 59, 217, 135, 4, 90, 184, 230, 167, 249, 27, 69, 198, 152, 122, 36,
19      248, 166, 68, 26, 153, 199, 37, 123, 58, 100, 134, 216, 91, 5, 231, 185,
20      140, 210, 48, 110, 237, 179, 81, 15, 78, 16, 242, 172, 47, 113, 147, 205,
21      17, 79, 173, 243, 112, 46, 204, 146, 211, 141, 111, 49, 178, 236, 14, 80,
22      175, 241, 19, 77, 206, 144, 114, 44, 109, 51, 209, 143, 12, 82, 176, 238,
23      50, 108, 142, 208, 83, 13, 239, 177, 240, 174, 76, 18, 145, 207, 45, 115,
24      202, 148, 118, 40, 171, 245, 23, 73, 8, 86, 180, 234, 105, 55, 213, 139,
25      87, 9, 235, 181, 54, 104, 138, 212, 149, 203, 41, 119, 244, 170, 72, 22,
26      233, 183, 85, 11, 136, 214, 52, 106, 43, 117, 151, 201, 74, 20, 246, 168,
27      116, 42, 200, 150, 21, 75, 169, 247, 182, 232, 10, 84, 215, 137, 107, 53};
28
29  //OneWire communication state
30  volatile onewireState g_onewireState = owFree;
31  //OneWire communication buffers
32  volatile onewireBuffer g_onewireRxBuffer;
33  volatile onewireBuffer g_onewireTxBuffer;
34
35
36  //*****************************************************************************
37  //
38  // Initialize one-wire bus
39  //
40  //*****************************************************************************
41  void initOneWireUART(void) {
42    //Enable peripherals
43    SysCtlPeripheralEnable(ONEWIRE_GPIO_PERIPH);
44      SysCtlPeripheralEnable(ONEWIRE_UART_PERIPH);
45      //Configure pin muxing
46      GPIOPinConfigure(ONEWIRE_UART_RX);
47      GPIOPinConfigure(ONEWIRE_UART_TX);
48      //Configure pin setup for functions
49      GPIOPinTypeUART(ONEWIRE_GPIO_BASE, ONEWIRE_UART_RX_PIN | ONEWIRE_UART_TX_PIN);
50      //Enable UART
51      UARTEnable(ONEWIRE_UART_BASE);
52
53      //Make strong Pull-Up available and enable
54      SysCtlPeripheralEnable(ONEWIRE_PULLUP_PERIPH);
55      GPIOPinTypeGPIOOutput(ONEWIRE_PULLUP_BASE, ONEWIRE_PULLUP_PIN);
56      ONEWIRE_ENABLE_PULLUP();
57  }
58
59  //*****************************************************************************
60  //
61  // Reset one-wire devices and detect presence
62  //
63  //*****************************************************************************
64  uint8_t initOneWire(void) {
65    char presence = 0x00;
66
67    //Change UART speed to 9600 baud for one-wire bit timing
68    UARTDisable(ONEWIRE_UART_BASE);
69      UARTConfigSetExpClk(ONEWIRE_UART_BASE, g_sysClock, 9600,
70                                (UART_CONFIG_PAR_NONE | UART_CONFIG_STOP_ONE |
71                                 UART_CONFIG_WLEN_8));
72      UARTEnable(ONEWIRE_UART_BASE);
73
74      //Empty UART FIFO in case anything unwanted is left behind
75      while(UARTCharGetNonBlocking(ONEWIRE_UART_BASE) != -1)
76        ;
77
78    //Disable strong pull-up
79      ONEWIRE_DISABLE_PULLUP();
80
81    //Initiate presence detection sequence
82    UARTCharPut(ONEWIRE_UART_BASE, 0xF0);
83    presence = UARTCharGet(ONEWIRE_UART_BASE);
84    //If successfull
85    if (presence != 0xF0) {
86      //Setup UART for faster bit timing
87      UARTDisable(ONEWIRE_UART_BASE);
88      UARTConfigSetExpClk(ONEWIRE_UART_BASE, g_sysClock, 115200,
89                    (UART_CONFIG_PAR_NONE | UART_CONFIG_STOP_ONE |
90                     UART_CONFIG_WLEN_8));
91        UARTEnable(ONEWIRE_UART_BASE);
92        //if (enablePullup)
93        //Enable strong pull-up
94        //GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_4, 0);
95        return ONEWIRE_INIT_SUCCESS;
96    } else {
97      //if (enablePullup)
98        //Enable strong pull-up
99        //GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_4, 0);
100     return ONEWIRE_INIT_FAIL;
101   }
102 }
```

```
103
104 //*****************************************************************************
105 //
106 // Write to the one-wire bus
107 //
108 //*****************************************************************************
109 void writeOneWire(uint8_t byte) {
110   int i;
111
112   //Disable strong pull-up
113   ONEWIRE_DISABLE_PULLUP();
114
115   //Write one byte per information bit to UART
116   for (i=0; i < 8; i++) {
117     UARTCharPut(ONEWIRE_UART_BASE, (0x01<<i & byte) ? 0xFF : 0x00);
118     UARTCharGet(ONEWIRE_UART_BASE);
119   }
120
121   //Enable strong pull-up
122   ONEWIRE_ENABLE_PULLUP();
123 }
124
125 //*****************************************************************************
126 //
127 // Read from one-wire bus
128 //
129 //*****************************************************************************
130 uint8_t readOneWire(void) {
131   int i;
132   char buffer, byte = 0x00;
133
134   //Disable strong pull-up
135   ONEWIRE_DISABLE_PULLUP();
136
137   //Read one byte per information bit from UART
138   for (i=0; i < 8; i++) {
139     UARTCharPut(ONEWIRE_UART_BASE, 0xFF);
140     buffer = UARTCharGet(ONEWIRE_UART_BASE);
141     byte |= (buffer & 0x01)<<i;
142   }
143
144   //Enable strong pull-up
145   ONEWIRE_ENABLE_PULLUP();
146
147   return byte;
148 }
149
150 //*****************************************************************************
151 // Identify devices on the bus
152 // Implemented as of the MAXIM app note!
153 //*****************************************************************************
154 uint8_t searchOneWire(maximROMCode* devices, uint8_t maxDevices, uint8_t isAlarm) {
155   uint8_t lastDevice = 0;
156   uint32_t idBitNumber = 0;
157   uint32_t lastZero = 0;
158   uint8_t idBit, compIdBit;
159   uint32_t lastDiscrepancy = 0;
160   uint8_t searchDirection;
161   uint8_t deviceCount = 0;
162
163   //Disable strong pull-up
164   ONEWIRE_DISABLE_PULLUP();
165
166   for (deviceCount = 0; !lastDevice && (deviceCount < maxDevices); deviceCount++) {
167     if (initOneWire() != ONEWIRE_INIT_SUCCESS) {
168       //Enable strong pull-up
169       ONEWIRE_ENABLE_PULLUP();
170       return 0;
171     }
172
173     idBitNumber = 0;
174     lastZero = 0;
175
176     //Only search for devices in alarm state?
177     writeOneWire(isAlarm ? ONEWIRE_ROM_ALARM : ONEWIRE_ROM_SEARCH);
178     ONEWIRE_DISABLE_PULLUP();
179
180     //Start searching
181     devices[deviceCount].raw = 0x00000000;
182     while (idBitNumber < 64) {
183       //Read ID-Bit and complementary ID-Bit
184       UARTCharPut(ONEWIRE_UART_BASE, 0xFF);
185       idBit = UARTCharGet(ONEWIRE_UART_BASE) & 0x01;
186       UARTCharPut(ONEWIRE_UART_BASE, 0xFF);
187       compIdBit = UARTCharGet(ONEWIRE_UART_BASE) & 0x01;
```

```
188
189         if (idBit & compIdBit) {
190           lastDiscrepancy = 0;
191           lastDevice = 0;
192           //Enable strong pull-up
193           ONEWIRE_ENABLE_PULLUP();
194           return 0;
195         }
196
197         if (!(idBit | compIdBit)) {
198           if (idBitNumber == lastDiscrepancy) {
199             searchDirection = 0x01;
200           } else  {
201             if (idBitNumber > lastDiscrepancy) {
202               searchDirection = 0x00;
203               lastZero = idBitNumber;
204             } else
205               searchDirection = (devices[deviceCount-1].raw >> idBitNumber) & 0x01;
206           }
207         } else {
208           searchDirection = idBit;
209         }
210
211         devices[deviceCount].raw |= (uint64_t)searchDirection<<idBitNumber;
212         UARTCharPut(ONEWIRE_UART_BASE, searchDirection ? 0xFF : 0x00);
213         UARTCharGet(ONEWIRE_UART_BASE);
214         idBitNumber++;
215       }
216
217       lastDiscrepancy = lastZero;
218       if (!lastDiscrepancy)
219         lastDevice = 1;
220     }
221
222     //Enable strong pull-up
223     ONEWIRE_ENABLE_PULLUP();
224
225     return deviceCount;
226   }
227
228   // Match one-wire device by ROM-Code
229   uint8_t matchOneWireDevice(maximROMCode* addr) {
230     uint8_t i;
231
232     //Disable strong pull-up
233     ONEWIRE_DISABLE_PULLUP();
234
235     if (initOneWire() != ONEWIRE_INIT_SUCCESS)
236       return ONEWIRE_INIT_FAIL;
237
238     //Tell the slaves that we will address one of them
239     writeOneWire(ONEWIRE_ROM_MATCH);
240
241     //Send every single byte of the 64 bit address
242     for (i=0; i < 8; i++)
243       writeOneWire(((uint8_t *)addr)[i]);
244
245     //Enable strong pull-up
246     ONEWIRE_ENABLE_PULLUP();
247
248     return ONEWIRE_INIT_SUCCESS;
249   }
250
251   uint8_t ds1820StartConversion(maximROMCode* addr) {
252     //Address the sensor
253     if (matchOneWireDevice(addr) == ONEWIRE_INIT_SUCCESS) {
254       //Do temperature conversion NOW!
255       writeOneWire(ONEWIRE_DS18_CONVERT);
256       return ONEWIRE_INIT_SUCCESS;
257     } else
258       return ONEWIRE_INIT_FAIL;
259   }
260
261   void onewireIntEnable(uint8_t priority) {
262     //Setup UART interrupt
263     IntPrioritySet(ONEWIRE_UART_INT, priority);
264     UARTIntRegister(ONEWIRE_UART_BASE, onewireIsr);
265     UARTFIFOLevelSet(ONEWIRE_UART_BASE, UART_FIFO_TX1_8, UART_FIFO_RX1_8);
266     UARTTxIntModeSet(ONEWIRE_UART_BASE, UART_TXINT_MODE_EOT);
267     UARTFIFOEnable(ONEWIRE_UART_BASE);
268     UARTIntEnable(ONEWIRE_UART_BASE, UART_INT_TX | UART_INT_RX);
269   }
270
271   void onewireTxTriggerInt(void) {
272     //Prepare buffer to be sent via onewire
```

```
273     g_onewireTxBuffer.position = (uint8_t*)g_onewireTxBuffer.data;
274     //Mark bus state as init
275     g_onewireState = owInit;
276
277     //Change UART speed to 9600 baud for one-wire bit timing
278       UARTConfigSetExpClk(ONEWIRE_UART_BASE, g_sysClock, 9600,
279                                   (UART_CONFIG_PAR_NONE | UART_CONFIG_STOP_ONE |
280                                    UART_CONFIG_WLEN_8));
281
282     //Disable strong pull-up
283       ONEWIRE_DISABLE_PULLUP();
284     //Initiate presence detection sequence
285     UARTCharPut(ONEWIRE_UART_BASE, 0xF0);
286   }
287
288   void onewireBufferTxInit(void) {
289     g_onewireTxBuffer.state = owBufferFree;
290     g_onewireTxBuffer.position = (uint8_t*)g_onewireTxBuffer.data;
291     g_onewireTxBuffer.readLength = 0;
292     g_onewireTxBuffer.writeLength = 0;
293   }
294
295   void onewireBufferRxInit(void) {
296     g_onewireRxBuffer.state = owBufferFree;
297     g_onewireRxBuffer.position = (uint8_t*)g_onewireRxBuffer.data;
298     g_onewireRxBuffer.readLength = 0;
299     g_onewireRxBuffer.writeLength = 0;
300
301     //Initialize data interpretation pointers
302     g_onewireRxBuffer.onewireData.temperature = (uint16_t*)g_onewireRxBuffer.onewireData.scratchpad;
303   }
304
305   void onewireBufferTxInsert(uint8_t byte) {
306     g_onewireTxBuffer.state = owBufferBusy;
307     *(g_onewireTxBuffer.position) = byte;
308     g_onewireTxBuffer.writeLength++;
309     g_onewireTxBuffer.position++;
310   }
311
312   void ds1820StartConversionInt(maximROMCode* addr) {
313     uint8_t i;
314
315     //Reset buffer
316     onewireBufferTxInit();
317     onewireBufferRxInit();
318
319     //Insert match ROM command
320     onewireBufferTxInsert(ONEWIRE_ROM_MATCH);
321     //Insert 64 bit ROM address
322     for (i=0; i < 8; i++)
323       onewireBufferTxInsert(*((uint8_t*)addr + i));
324     //Insert Convert T command
325     onewireBufferTxInsert(ONEWIRE_DS18_CONVERT);
326
327     //Trigger interrupt driven transmit
328     onewireTxTriggerInt();
329   }
330
331   void ds1820ReadTemperatureInt(maximROMCode* addr) {
332     uint8_t i;
333
334     //Reset buffer
335     onewireBufferTxInit();
336     onewireBufferRxInit();
337
338     //Insert match ROM command
339     onewireBufferTxInsert(ONEWIRE_ROM_MATCH);
340     //Insert 64 bit ROM address
341     for (i=0; i < 8; i++)
342       onewireBufferTxInsert(*((uint8_t*)addr + i));
343     //Insert Convert T command
344     onewireBufferTxInsert(ONEWIRE_DS18_READ);
345     //Setup to read 8 bytes
346     g_onewireTxBuffer.readLength = 8;
347
348     //Trigger interrupt driven transmit
349     onewireTxTriggerInt();
350   }
351
352   void onewireIsr(void) {
353     uint32_t status;
354     uint8_t i;
355
356     status = UARTIntStatus(ONEWIRE_UART_BASE, true);
357     UARTIntClear(ONEWIRE_UART_BASE, status);
```

```
358
359   if (status & UART_INT_RX) {
360     switch (g_onewireState) {
361     case owWriting:
362       //Alwyays receive a complete byte
363       //Clear data
364       *(g_onewireRxBuffer.position) = 0x00;
365       for (i=0; i<8; i++)
366         *(g_onewireRxBuffer.position) |= (UARTCharGet(ONEWIRE_UART_BASE) & 0x01)<<i;
367       g_onewireRxBuffer.position++;
368       break;
369     case owReading:
370       //Alwyays receive a complete byte
371       //Clear data
372       *(g_onewireRxBuffer.position) = 0x00;
373       for (i=0; i<8; i++)
374         *(g_onewireRxBuffer.position) |= (UARTCharGet(ONEWIRE_UART_BASE) & 0x01)<<i;
375       g_onewireRxBuffer.readLength++;
376
377       //When all data has been received
378       if (g_onewireRxBuffer.readLength == g_onewireRxBuffer.writeLength) {
379         g_onewireRxBuffer.state = owBufferReady;
380         //Make the FIFO trigger at every RX byte, so we can keep it clean
381         UARTFIFOLevelSet(ONEWIRE_UART_BASE, UART_FIFO_TX1_8, UART_FIFO_RX1_8);
382         ONEWIRE_ENABLE_PULLUP();
383         g_onewireState = owFree;
384       } else
385         g_onewireRxBuffer.position++;
386       break;
387     default:
388       //Empty UART FIFO in case anything unwanted is left behind
389       while(UARTCharsAvail(ONEWIRE_UART_BASE))
390         UARTCharGet(ONEWIRE_UART_BASE);
391       break;
392     }
393   }
394   if (status & UART_INT_TX) {
395     switch (g_onewireState) {
396     case owInit:
397       //If successfull
398       if (UARTCharGet(ONEWIRE_UART_BASE) != 0xF0) {
399         //Setup UART for faster bit timing
400         UARTConfigSetExpClk(ONEWIRE_UART_BASE, g_sysClock, 115200,
401                             (UART_CONFIG_PAR_NONE | UART_CONFIG_STOP_ONE |
402                              UART_CONFIG_WLEN_8));
403
404         UARTFIFOLevelSet(ONEWIRE_UART_BASE, UART_FIFO_TX1_8, UART_FIFO_RX4_8);
405         g_onewireRxBuffer.state = owBufferBusy;
406
407         //Write first 8-bytes to UART FIFO (8-bits on onewire)
408         for (i=0; i < 8; i++)
409           UARTCharPut(ONEWIRE_UART_BASE, (0x01<<i & *(g_onewireTxBuffer.position)) ? 0xFF : 0x00);
410         g_onewireTxBuffer.position++;
411         g_onewireTxBuffer.writeLength--;
412
413         g_onewireState = owWriting;
414       } else {
415         g_onewireState = owError;
416         g_onewireTxBuffer.state = owBufferReady;
417       }
418       break;
419     case owWriting:
420       //Write 8-bytes to UART FIFO (8-bits on onewire)
421       for (i=0; i < 8; i++)
422         UARTCharPut(ONEWIRE_UART_BASE, (0x01<<i & *(g_onewireTxBuffer.position)) ? 0xFF : 0x00);
423
424       //Have all bytes been written?
425       if (--g_onewireTxBuffer.writeLength)
426         g_onewireTxBuffer.position++;
427       else
428         if (g_onewireTxBuffer.readLength) {
429           g_onewireState = owReading;
430           //g_onewireRxBuffer.position = g_onewireRxBuffer.data;
431           g_onewireRxBuffer.writeLength = g_onewireTxBuffer.readLength;
432           g_onewireRxBuffer.readLength = 0;
433         } else {
434           ONEWIRE_ENABLE_PULLUP();
435           g_onewireState = owFree;
436           g_onewireTxBuffer.state = owBufferReady;
437           UARTFIFOLevelSet(ONEWIRE_UART_BASE, UART_FIFO_TX1_8, UART_FIFO_RX1_8);
438         }
439       break;
440     case owReading:
441       //Write 8-bytes of 0xFF to the FIFO, to make the onewire slave write a data byte
442       for (i=0; i < 8; i++)
```

```
443           UARTCharPut(ONEWIRE_UART_BASE, 0xFF);
444         //Have all bytes been read?
445         if (!(g_onewireTxBuffer.readLength--))
446           g_onewireState = owFree;
447         break;
448       case owFree:
449         ONEWIRE_ENABLE_PULLUP();
450         break;
451       default:
452         break;
453       }
454     }
455   }
456
457   uint8_t ds1820ReadTemperature(maximROMCode* addr, int16_t* temperature) {
458     uint8_t i, crc = 0x00;
459     //Address the sensor
460     if (matchOneWireDevice(addr) == ONEWIRE_INIT_SUCCESS) {
461       //Read first two bytes of scratchpad
462       writeOneWire(ONEWIRE_DS18_READ);
463       for (i=0; i<9; i++) {
464         if (i<2)
465           crc = ds1820GetCRC(crc, (((uint8_t*)temperature)[i] = readOneWire()));
466         else if (i<8)
467           crc = ds1820GetCRC(crc, readOneWire());
468         else
469           if (crc != readOneWire())
470             return ONEWIRE_CRC_FAIL;
471       }
472       return ONEWIRE_INIT_SUCCESS;
473     } else
474       return ONEWIRE_INIT_FAIL;
475   }
476
477   uint8_t ds1820GetCRC(uint8_t crc, uint8_t byte) {
478     return g_maximCRCLookup[crc ^ byte];
479   }
```

Quellcode D.6: Software-Modul: GPS / GNSS (Header)

```
1   /*
2    * gps.h
3    *
4    * Author: Felix Groth
5    */
6
7   #ifndef GPS_H_
8   #define GPS_H_
9
10  //Standard includes
11  #include <stdint.h>
12  #include <stdio.h>
13  #include <stdbool.h>
14
15  //Hardware and driver includes
16  #include "inc/tm4c1294ncpdt.h"
17  #include "inc/hw_gpio.h"
18  #include "inc/hw_memmap.h"
19
20  #include "driverlib/sysctl.h"
21  #include "driverlib/rom.h"
22  #include "driverlib/gpio.h"
23  #include "driverlib/uart.h"
24  #include "driverlib/pin_map.h"
25  #include "driverlib/timer.h"
26  #include "driverlib/interrupt.h"
27
28
29  /** GPS setup definitions - you may change these **/
30  //TODO: Put these into a seperate file - unit tests and production system might need them different
31  #define GPS_BAUDRATE      460800
32  #define GPS_FIX_PERIOD    100    //Period time in ms between position fixes
33  #define GPS_POSITION_RATE 1    //Send rate for position information - multiples of GPS_FIX_PERIOD
34  #define GPS_TIME_ALIGN    0    //Align to UTC
35  #define GPS_SATELLITE_RATE  1    //Send rate for satellite information - multiples of GPS_FIX_PERIOD
36
37  /** Defines for GPS hardware ports **/
38  #define GPS1_UART_BASE      UART3_BASE
39  #define GPS1_GPIO_BASE      GPIO_PORTA_BASE
40  #define GPS1_GPIO_PERIPH    SYSCTL_PERIPH_GPIOA
41  #define GPS1_UART_PERIPH    SYSCTL_PERIPH_UART3
```

```
42  #define GPS1_UART_RX       GPIO_PA4_U3RX
43  #define GPS1_UART_TX       GPIO_PA5_U3TX
44  #define GPS1_UART_RX_PIN    GPIO_PIN_4
45  #define GPS1_UART_TX_PIN    GPIO_PIN_5
46  #define GPS1_UART_INT      INT_UART3
47  #define GPS1_TIMEPULSE_PERIPH  SYSCTL_PERIPH_GPIOM
48  #define GPS1_TIMEPULSE_BASE     GPIO_PORTM_BASE
49  #define GPS1_TIMEPULSE_PIN      GPIO_PIN_1
50  #define GPS1_TIMEPULSE_INT_PIN    GPIO_INT_PIN_1
51  #define GPS1_TIMEPULSE_INT      INT_GPIOM
52
53  #define GPS2_UART_BASE      UART4_BASE
54  #define GPS2_GPIO_BASE      GPIO_PORTK_BASE
55  #define GPS2_GPIO_PERIPH    SYSCTL_PERIPH_GPIOK
56  #define GPS2_UART_PERIPH    SYSCTL_PERIPH_UART4
57  #define GPS2_UART_RX       GPIO_PK0_U4RX
58  #define GPS2_UART_TX       GPIO_PK1_U4TX
59  #define GPS2_UART_RX_PIN    GPIO_PIN_0
60  #define GPS2_UART_TX_PIN    GPIO_PIN_1
61  #define GPS2_UART_INT      INT_UART4
62  #define GPS2_TIMEPULSE_PERIPH  SYSCTL_PERIPH_GPIOM
63  #define GPS2_TIMEPULSE_BASE     GPIO_PORTM_BASE
64  #define GPS2_TIMEPULSE_PIN      GPIO_PIN_2
65  #define GPS2_TIMEPULSE_INT_PIN    GPIO_INT_PIN_2
66  #define GPS2_TIMEPULSE_INT      INT_GPIOM
67
68  #define GPS_MAX_BAUDRATE   460800
69
70  //Return defines for gps functions
71  #define GPS_INIT_FAIL     0x01
72  #define GPS_INIT_SUCCESS   0x00
73
74
75  //GPS module identifier enum
76  typedef enum {
77    gps1 = 0,
78    gps2 = 1
79  } gpsId;
80
81  typedef enum {
82    gpsTOSInit = 0,
83    gpsTOSBusy = 1,
84    gpsTOSWaiting = 2,
85    gpsTOSTrained = 3
86  } gpsTOSTrainingState;
87
88  typedef struct {
89    uint32_t precisionCounter;
90    uint32_t frequency;
91    gpsTOSTrainingState state;
92    uint8_t isTOS;
93  } gpsTOSHelper;
94
95
96  /** Defines for uBlox protocol **/
97  #define UBX_MAX_PAYLOAD       1024
98  #define UBX_SYNC1          0xB5
99  #define UBX_SYNC2          0x62
100 #define UBX_FRAME_SIZE        8    //Frame size in bytes: 2 bytes sync + 1 byte class + 1 byte id + 2 bytes length + 2 bytes checksum
101 #define UBX_FRAME_SIZE_NO_CHK 6    //Frame size in bytes: 2 bytes sync + 1 byte class + 1 byte id + 2 bytes length
102 #define UBX_CHECKSUM_BYTES      2
103 #define UBX_SYNC_OFFSET       2
104 #define UBX_SYNC_WAIT       5000000
105
106 #define UBX_RXBUFF_COUNT      10
107 #define UBX_TXBUFF_COUNT      1
108
109 //Return defines for uBlox protocol synchronisation
110 #define UBX_SYNC_TIMEOUT      0x02  //For blocking synchronisation only
111 #define UBX_SYNC_FAIL       0x01
112 #define UBX_SYNC_MATCH        0x00
113 //Return defines for uBlox package reception
114 #define UBX_OUT_OF_MEM        0x02
115 #define UBX_CHK_FAIL        0x01
116 #define UBX_CHK_MATCH       0x00
117 //Return defines for uBlox acknowledge packages
118 #define UBX_ACK_NONE        0x03
119 #define UBX_ACK_OTHER       0x02
120 #define UBX_ACK_NOT         0x01
121 #define UBX_ACK_MATCH       0x00
122 //Return defines for RX ringbuffer FFW search
123 #define UBX_PKG_FOUND       0x00
124 #define UBX_PKG_NOT_FOUND     0x01
125
126
```

```
127 //UBX message classes and ids
128 #define UBX_CFG       0x06     //CLASS: Configuration
129 #define UBX_CFG_PRT    0x00      //ID: Port settings
130 #define UBX_CFG_TP5    0x31      //ID: Timepulse configuration
131 #define UBX_CFG_RATE   0x08      //ID: Update rate configuration
132 #define UBX_CFG_MSG    0x01      //ID: Output message rate configuration
133 #define UBX_ACK       0x05     //CLASS: Acknowledge
134 #define UBX_ACK_NAK    0x00      //ID: Not acknowledged
135 #define UBX_ACK_ACK    0x01      //ID: Acknowledged
136 #define UBX_NAV       0x01     //CLASS: Navigation
137 #define UBX_NAV_POSLLH  0x02      //ID: Geodetic Position Solution
138 #define UBX_NAV_TIMEUTC 0x21      //ID: UTC Time Solution
139 #define UBX_NAV_PVT    0x07      //ID: Position Velocity Time Solution
140 #define UBX_TIM       0x0D     //CLASS: Time
141 #define UBX_TIM_TOS    0x12      //ID: Time Pulse Time and Frequency Data
142 #define UBX_TIM_TP     0x01      //ID: Time Pulse Timedata
143 #define UBX_MON       0x0A     //CLASS: Monitoring Messages
144 #define UBX_MON_RXR    0x21      //ID: Receiver Status Information
145 #define UBX_RXM       0x02     //CLASS: Receiver manager Messages
146 #define UBX_RXM_PMREQ 0x41      //ID: Requests a Power Management Task
147
148 /* UBX packet data for ids */
149 //UBX all standard polls have no payload
150 #define UBX_POLL_LTH       0
151
152 //UBX CFG-PRT
153 #define UBX_CFG_PRT_POLL       1
154 #define UBX_CFG_PRT_LEN        20
155 //portId
156 #define UBX_CFG_PRT_ID_OS      0
157 #define UBX_CFG_PRT_ID_TYPE      uint8_t    //8 bit unsigned int
158 //txReady
159 #define UBX_CFG_PRT_TXR_OS       2
160 #define UBX_CFG_PRT_TXR_TYPE     uint16_t   //16 bit field
161 #define UBX_CFG_PRT_TXR_EN       0x0001     //Enable TX ready feature for this port
162 #define UBX_CFG_PRT_TXR_POL      0x0002     //Pin polarity (high active when set)
163 #define UBX_CFG_PRT_TXR_PIN_OS     2      //Bit offset for the value to assign here
164 #define UBX_CFG_PRT_TXR_THR_OS     7      //Bit offset for the threshold field
165 //mode
166 #define UBX_CFG_PRT_MODE_OS      4
167 #define UBX_CFG_PRT_MODE_TYPE    uint32_t       //32 bit field
168 #define UBX_CFG_PRT_MODE_8BIT    (0x00000003<<6)    //Value for 8-Bit character length
169 #define UBX_CFG_PRT_MODE_NOPAR     (0x00000004<<9)    //Value for no parity
170 #define UBX_CFG_PRT_MODE_1STP    (0x00000000<<12)   //Value for 1 stop bit
171 //baudRate
172 #define UBX_CFG_PRT_RATE_OS      8
173 #define UBX_CFG_PRT_RATE_TYPE    uint32_t   //32 bit unsigned int
174 //InProtoMask
175 #define UBX_CFG_PRT_INPRO_OS     12
176 #define UBX_CFG_PRT_INPRO_TYPE     uint16_t   //16 bit field
177 #define UBX_CFG_PRT_INPRO_UBX    0x0001     //Value for UBX input protocol
178 #define UBX_CFG_PRT_INPRO_NMEA     0x0002     //Value for NMEA input protocol
179 #define UBX_CFG_PRT_INPRO_RTCM     0x0004     //Value for RTCM input protocol
180 //OutProtoMask
181 #define UBX_CFG_PRT_OUTPRO_OS    14
182 #define UBX_CFG_PRT_OUTPRO_TYPE    uint16_t   //16 bit field
183 #define UBX_CFG_PRT_OUTPRO_UBX     0x0001     //Value for UBX output protocol
184 #define UBX_CFG_PRT_OUTPRO_NMEA    0x0002     //Value for NMEA output protocol
185 //flags
186 #define UBX_CFG_PRT_FLGS_OS      16
187 #define UBX_CFG_PRT_FLGS_TYOE    uint16_t   //16 bit field
188 #define UBX_CFG_PRT_FLGS_TIOUT     0x0002     //Value to enable extended timeout
189
190
191 //UBX CFG-TP5
192 #define UBX_CFG_TP5_POLL     1
193 #define UBX_CFG_TP5_LEN      32
194 //tpIdx
195 #define UBX_CFG_TP5_IDX_OS     0
196 #define UBX_CFG_TP4_IDX_TYPE   uint8_t    //8 bit unsigned int
197 //version
198 #define UBX_CFG_TP5_VRS_OS     1
199 #define UBX_CFG_TP5_VRS_TYPE   uint8_t    //8 bit unsigned int
200 //freqPeriod
201 #define UBX_CFG_TP5_FRQP_OS    8
202 #define UBX_CFG_TP5_FRQP_TYPE uint32_t   //32 bit unsigned int
203 //freqPeriodLock
204 #define UBX_CFG_TP5_FRQPL_OS   12
205 #define UBX_CFG_TP5_FRQPL_TYPE   uint32_t   //32 bit unsigned int
206 //pulseLenRatio
207 #define UBX_CFG_TP5_PLR_OS     16
208 #define UBX_CFG_TP5_PLR_TYPE   uint32_t   //32 bit unsigned int
209 //pulseLenRatioLock
210 #define UBX_CFG_TP5_PLRL_OS    20
211 #define UBX_CFG_TP5_PLRL_TYPE uint32_t   //32 bit unsigned int
```

```
212   //flags
213   #define UBX_CFG_TP5_FLGS_OS    28
214   #define UBX_CFG_TP5_FLGS_TYPE  uint32_t  //32 bit field
215   #define UBX_CFG_TP5_FLGS_ACT   0x00000001  //Activate
216   #define UBX_CFG_TP5_FLGS_GPS   0x00000002  //Lock to GPS as soon as possible
217   #define UBX_CFG_TP5_FLGS_LOC   0x00000004  //Switch to "locked" settings when GPS time is valid
218   #define UBX_CFG_TP5_FLGS_FRQ   0x00000008  //Supply frequency, else period time
219   #define UBX_CFG_TP5_FLGS_LEN   0x00000010  //Supply length, else duty cycle
220   #define UBX_CFG_TP5_FLGS_ALG   0x00000020  //Align to top of second
221   #define UBX_CFG_TP5_FLGS_POL   0x00000040  //Polarity of timepulse, rising edge at top of second if set
222   #define UBX_CFG_TP5_FLGS_GRD   0x00000080  //Align to GPS timegrid if set, else UTC
223
224   //UBX CFG-RATE
225   #define UBX_CFG_RATE_LEN     6
226   //measRate
227   #define UBX_CFG_RATE_MSR_OS    0
228   #define UBX_CFG_RATE_MSR_TYPE uint16_t  //16 bit unsigned int
229   //navRate
230   #define UBX_CFG_RATE_NVR_OS    2
231   #define UBX_CFG_RATE_NVR_TYPE uint16_t  //16 bit unsigned int
232   //timeRef
233   #define UBX_CFG_RATE_TREF_OS   4
234   #define UBX_CFG_RATE_TREF_TYPE  uint16_t  //16 bit unsigned int (0 = UTC, 1 = GPS)
235
236
237   //UBX CFG-MSG
238   #define UBX_CFG_MSG_POLL     2
239   #define UBX_CFG_MSG_LEN      3
240   #define UBX_CFG_MSG_LEN_ALL    8
241   //msgClass
242   #define UBX_CFG_MSG_CLS_OS     0
243   #define UBX_CFG_MSG_CLS_TYPE   uint8_t   //8 bit unsigned int as defined above
244   //msgId
245   #define UBX_CFG_MSG_ID_OS    1
246   #define UBX_CFG_MSG_ID_TYPE    uint8_t   //8 bit unsigned int as defined above
247   //rate
248   #define UBX_CFG_MSG_RATE_OS    2
249   #define UBX_CFG_MSG_RATE_TYPE uint8_t   //8 bit signed int (send msg every X navigation epoch)
250
251   //UBX NAV-POSLLH
252   #define UBX_NAV_POSLLH_LEN       28
253   //iTOW
254   #define UBX_NAV_POSLLH_ITOW_OS     0
255   #define UBX_NAV_POSLLH_ITOW_TYPE   uint32_t  //32 bit unsigned int (ms)
256   //lon
257   #define UBX_NAV_POSLLH_LON_OS    4
258   #define UBX_NAV_POSLLH_LON_TYPE    int32_t   //32 bit signed int (deg * 10e7)
259   //lat
260   #define UBX_NAV_POSLLH_LAT_OS    8
261   #define UBX_NAV_POSLLH_LAT_TYPE    int32_t   //32 bit signed int (deg * 10e7)
262   //hMSL
263   #define UBX_NAV_POSLLH_MSL_OS    16
264   #define UBX_NAV_POSLLH_MSL_TYPE    int32_t   //32 bit signed int --- [mm] above mean sea level
265
266   //UBX RXM-PMREQ
267   #define UBX_RXM_PMREQ_LEN      8
268   //flags
269   #define UBX_RXM_PMREQ_FLGS_BKP     0x02
270   /** END of UBX protocol defines **/
271
272   /** UBX protocol types **/
273   //GNSS identifiers
274   typedef enum {
275     i2c = 0,
276     uart = 1,
277     uart2 = 2,
278     usb = 3,
279     spi = 4
280   } ubxPortId;
281
282   //uBlox Port identifiers
283   typedef enum {
284     gps = 0,
285     sbas = 1,
286     galileo = 2,
287     beidou = 3,
288     imes = 4,
289     qzss = 5,
290     glonass = 6
291   } ubxGNSSId;
292
293   //uBlox package buffer states
294   typedef enum {
295     ubxRxDataReady = 0,
296     ubxRxSyncing = 1,
```

```
297     ubxRxSynced = 2,
298     ubxRxReading = 3,
299     ubxRxChecking = 4,
300     ubxRxFree = 5,
301     ubxRxError = 6
302   } ubxPacketStateRx;
303
304   //uBlox package buffer states
305   typedef enum {
306     ubxTxRTS = 0,    //Ready to send
307     ubxTxSyncing = 1,
308     ubxTxWriting = 2,
309     ubxTxDone = 3,
310     ubxTxACK = 4,
311     ubxTxNAK = 5,
312     ubxTxFree = 6,
313     ubxTxError = 7
314   } ubxPacketStateTx;
315
316   //UBX-NAV-POSLLH
317   typedef struct {
318     uint32_t itow;
319     int32_t lon;
320     int32_t lat;
321     int32_t height;
322     int32_t hMsl;
323     uint32_t hAcc;
324     uint32_t vAcc;
325   } ubxNavPosllh;
326
327   //UBX-CFG-PRT
328   typedef struct {
329     uint8_t portID;
330     uint8_t reserved1;
331     uint16_t txReady;
332     uint32_t mode;
333     uint32_t baudRate;
334     uint16_t inProtoMask;
335     uint16_t outProtoMask;
336     uint16_t flags;
337     uint16_t reserved2;
338   } ubxCfgPrtUART;
339
340   //UBX-CFG-RATE
341   typedef struct {
342     uint16_t measRate;
343     uint16_t navRate;
344     uint16_t timeRef;
345   } ubxCfgRateSettings;
346
347   //UBX-CFG-MSG
348   typedef struct {
349     uint8_t msgClass;
350     uint8_t msgID;
351     uint8_t rate;
352   } ubxCfgMsgRate;
353
354   //UBX-CFG-TP5
355   typedef struct {
356     uint8_t tpIdx;
357     uint8_t version;
358     uint8_t reserved[2];
359     int16_t antCableDelay;
360     int16_t rfGroupDelay;
361     uint32_t freqPeriod;
362     uint32_t freqPeriodLock;
363     uint32_t pulseLenRatio;
364     uint32_t pulseLenRatioLock;
365     int32_t userConfigDel;
366     uint32_t flags;    //There are two versions of how these are used
367   } ubxCfgTp5;
368
369   //UBX-NAV-PVT
370   typedef struct {
371     uint32_t iTOW;
372     uint16_t year;
373     uint8_t month;
374     uint8_t day;
375     uint8_t hour;
376     uint8_t min;
377     uint8_t sec;
378     uint8_t valid;
379     uint32_t tAcc;
380     int32_t nano;
381     uint8_t fixType;
```

```
382     uint8_t flags;
383     uint8_t reserved1;
384     uint8_t numSV;
385     int32_t lon;
386     int32_t lat;
387     int32_t height;
388     int32_t hMSL;
389     uint32_t hAcc;
390     uint32_t vAcc;
391     int32_t velN;
392     int32_t velE;
393     int32_t velD;
394     int32_t gSpeed;
395     int32_t headMot;
396     uint32_t sAcc;
397     uint32_t headAcc;
398     uint16_t pDOP;
399     uint8_t reserved2[6];
400     int32_t headVeh;
401     uint8_t reserved3[4];
402   } ubxNavPvt;
403
404   //UBX-RXM-PMREQ
405   typedef struct {
406     uint32_t duration;
407     uint32_t flags;
408   } ubxRxmPmreq;
409
410   typedef struct {
411     union {
412       struct {
413         uint8_t class;
414         uint8_t id;
415       };
416       uint16_t classAndId;
417     };
418     uint16_t length;
419     union {
420       uint8_t payload[UBX_MAX_PAYLOAD];
421       ubxCfgPrtUART ubxCfgPrtUART;
422       ubxCfgRateSettings ubxCfgRateSettings;
423       ubxCfgMsgRate ubxCfgMsgRate;
424       ubxNavPosllh ubxNavPosllh;
425       ubxNavPvt ubxNavPvt;
426       ubxCfgTp5 ubxCfgTp5;
427       ubxRxmPmreq ubxRxmPmreq;
428     };
429     uint8_t checksum[2];
430   } ubxPacket;
431
432   typedef struct ubxPacketBufferRx ubxPacketBufferRx;
433   struct ubxPacketBufferRx {
434     ubxPacket* ubxPacket;
435     uint8_t* writePosition;
436     ubxPacketStateRx rxState;
437     uint8_t calcChecksum[2];
438     //The measurement time relative timestamp in ms
439     uint32_t timestamp;
440     //These can be concatenated
441     ubxPacketBufferRx* prev;
442     ubxPacketBufferRx* next;
443   };
444
445   typedef struct ubxPacketBufferTx ubxPacketBufferTx;
446   struct ubxPacketBufferTx {
447     ubxPacket* ubxPacket;
448     uint8_t* readPosition;
449     ubxPacketStateTx txState;
450     //These can be concatenated
451     ubxPacketBufferTx* prev;
452     ubxPacketBufferTx* next;
453   };
454
455
456
457   /** External globals to use **/
458   extern unsigned int g_sysClock;
459
460   /** Function prototypes **/
461   void ubxBufferRxPkgRead(gpsId id, volatile ubxPacketBufferRx* rxBuffer);
462   void gpsInitUART(gpsId id, uint32_t baudRate);
463   uint8_t gpsConfigTiempulseInt(gpsId id, uint32_t period, void (*pfnIntHandler)(void));
464   uint8_t gpsSetMsgRate(gpsId id, uint8_t ubxClass, uint8_t ubxId, uint8_t rate);
465   uint8_t gpsInitModule(gpsId id, uint32_t baudRate, uint8_t interruptPriority);
466   void gpsTimepulseWrapperIsr(void);
```

```
467 inline void gpsTrainTopOfSecond(gpsId id);
468 uint8_t gpsTimepulseIntSetup(gpsId id, uint32_t frequency, void (*pfnIntHandler)(void), uint8_t priority);
469 void gps1PkgIsr(void);
470 void gps2PkgIsr(void);
471 void gpsEnablePkgInt(gpsId id, uint8_t priority);
472 void gpsDisablePkgInt(gpsId id);
473 uint8_t ubxPollRead(gpsId id, volatile ubxPacket* packet);
474 void ubxSwitchBuffer(volatile ubxPacketBufferRx* rxBuffer, volatile ubxPacketBufferTx* txBuffer);
475 void ubxPacketResetChecksum(volatile ubxPacket* packet);
476 ubxPacketStateTx ubxPacketByteWrite(gpsId id, volatile ubxPacketBufferTx* txBuffer);
477 ubxPacketStateRx ubxPacketByteRead(gpsId id, volatile ubxPacketBufferRx* rxBuffer);
478 uint8_t ubxPacketCheckAck(gpsId id, volatile ubxPacketBufferTx* txPkg);
479 void gpsPkgTriggerWrite(gpsId id);
480 ubxPacketBufferRx* ubxRxPacketFind(gpsId gpsId, uint8_t class, uint8_t id);
481 void gpsTimepulseIntEnable(void);
482 void gpsTimepulseIntDisable(void);
483 void ubxInitBuffers(gpsId gpsId);
484 void ubxFilterClassSet(uint8_t class);
485 void gpsPowerDown(gpsId id);
486
487 #endif /* GPS_H_ */
```

Quellcode D.7: Software-Modul: GPS / GNSS

```
1  /*
2   * gps.c
3   *
4   * Author: Felix Groth
5   */
6
7  //Include the header file
8  #include "gps.h"
9
10 /** Global variables **/
11 //We will be using a very basic double buffering scheme here
12 volatile ubxPacket g_ubxPacketBuffer[2][UBX_RXBUFF_COUNT + UBX_TXBUFF_COUNT];
13 volatile ubxPacketBufferRx g_ubxPacketBufferRx[2][UBX_RXBUFF_COUNT];
14 volatile ubxPacketBufferTx g_ubxPacketBufferTx[2][UBX_TXBUFF_COUNT];
15 //Global relaive time in ms
16 extern volatile uint32_t g_relTimeMs;
17 //Timestamp based on g_relTimeMs belonging to the last 1/10 of a second (last nav epoch)
18 volatile uint32_t g_gpsTimestamp;
19 //These pointers handle the ringbuffer positions for reading and writing
20 volatile ubxPacketBufferRx* g_ubxPacketBufferRxActiveRead[2];
21 volatile ubxPacketBufferRx* g_ubxPacketBufferRxActiveWrite[2];
22 volatile ubxPacketBufferTx* g_ubxPacketBufferTxActiveRead[2];
23 volatile ubxPacketBufferTx* g_ubxPacketBufferTxActiveWrite[2];
24 //A filter to remove exactly ONE message class from the data flow
25 volatile uint8_t g_ubxPacketFilterClass = 0x00;
26 //Helper to identify the top of second timepulse
27 volatile gpsTOSHelper tosHelper;
28 //Holds the pointer to the users timepulse ISR
29 void (*g_gpsTimepulseIsr)(void);
30 //Sets whether the timepulse interrupt shall trigger the user function
31 uint8_t g_gpsTimepulseIntEnabled = 0x00;
32
33
34 void gpsTimepulseIntEnable(void) {
35   g_gpsTimepulseIntEnabled = 0x01;
36 }
37
38 void gpsTimepulseIntDisable(void) {
39   g_gpsTimepulseIntEnabled = 0x00;
40 }
41
42 void ubxFilterClassSet(uint8_t class) {
43   g_ubxPacketFilterClass = class;
44 }
45
46 void ubxBufferRxPkgRead(gpsId id, volatile ubxPacketBufferRx* rxBuffer) {
47   if (rxBuffer == NULL)
48     return;
49
50   rxBuffer->rxState = ubxRxFree;
51   if (rxBuffer == g_ubxPacketBufferRxActiveRead[id])
52     g_ubxPacketBufferRxActiveRead[id] = g_ubxPacketBufferRxActiveRead[id]->next;
53   else {
54     //Reorder
55     //Remove
56     rxBuffer->prev->next = rxBuffer->next;
57     rxBuffer->next->prev = rxBuffer->prev;
```

```
58      //Insert
59      rxBuffer->prev = g_ubxPacketBufferRxActiveRead[id]->prev;
60      rxBuffer->prev->next = (ubxPacketBufferRx*)rxBuffer;
61      rxBuffer->next = (ubxPacketBufferRx*)g_ubxPacketBufferRxActiveRead[id];
62      rxBuffer->next->prev = (ubxPacketBufferRx*)rxBuffer;
63    }
64  }
65
66  void ubxInitBuffers(gpsId gpsId) {
67    uint8_t bufferId;
68
69    g_ubxPacketBufferRxActiveRead[gpsId] = g_ubxPacketBufferRx[gpsId];
70    g_ubxPacketBufferRxActiveWrite[gpsId] = g_ubxPacketBufferRx[gpsId];
71    g_ubxPacketBufferTxActiveRead[gpsId] = g_ubxPacketBufferTx[gpsId];
72    g_ubxPacketBufferTxActiveWrite[gpsId] = g_ubxPacketBufferTx[gpsId];
73    for (bufferId = 0; bufferId < UBX_RXBUFF_COUNT; bufferId++) {
74      g_ubxPacketBufferRx[gpsId][bufferId].ubxPacket = (ubxPacket*)g_ubxPacketBuffer[gpsId] + bufferId;
75      g_ubxPacketBufferRx[gpsId][bufferId].rxState = ubxRxFree;
76      //Concatenate into a ring
77      if (bufferId == 0)
78        g_ubxPacketBufferRx[gpsId][bufferId].prev = (ubxPacketBufferRx*)g_ubxPacketBufferRx[gpsId] + (UBX_RXBUFF_COUNT - 1);
79      else
80        g_ubxPacketBufferRx[gpsId][bufferId].prev = (ubxPacketBufferRx*)g_ubxPacketBufferRx[gpsId] + (bufferId - 1);
81
82      if (bufferId < (UBX_RXBUFF_COUNT - 1))
83        g_ubxPacketBufferRx[gpsId][bufferId].next = (ubxPacketBufferRx*)g_ubxPacketBufferRx[gpsId] + (bufferId + 1);
84      else
85        g_ubxPacketBufferRx[gpsId][bufferId].next = (ubxPacketBufferRx*)g_ubxPacketBufferRx[gpsId];
86    }
87
88    for (; bufferId < (UBX_RXBUFF_COUNT + UBX_TXBUFF_COUNT); bufferId++) {
89      g_ubxPacketBufferTx[gpsId][bufferId - UBX_RXBUFF_COUNT].ubxPacket = (ubxPacket*)g_ubxPacketBuffer[gpsId] + bufferId;
90      g_ubxPacketBufferTx[gpsId][bufferId - UBX_RXBUFF_COUNT].txState = ubxTxFree;
91    }
92  }
93
94  //*****************************************************************************
95  //
96  // Initialize GPS UART
97  //
98  //*****************************************************************************
99  void gpsInitUART(gpsId id, uint32_t baudRate) {
100   //Which module do we update the port settings for?
101   if (id == gps1) {
102     SysCtlPeripheralEnable(GPS1_GPIO_PERIPH);
103     SysCtlPeripheralEnable(GPS1_UART_PERIPH);
104     GPIOPinConfigure(GPS1_UART_RX);
105     GPIOPinConfigure(GPS1_UART_TX);
106     GPIOPinTypeUART(GPS1_GPIO_BASE, GPS1_UART_RX_PIN | GPS1_UART_TX_PIN);
107     UARTConfigSetExpClk(GPS1_UART_BASE, g_sysClock, baudRate,
108                         (UART_CONFIG_PAR_NONE | UART_CONFIG_STOP_ONE |
109                          UART_CONFIG_WLEN_8));
110     UARTEnable(GPS1_UART_BASE);
111   } else if (id == gps2) {
112     SysCtlPeripheralEnable(GPS2_GPIO_PERIPH);
113     SysCtlPeripheralEnable(GPS2_UART_PERIPH);
114     GPIOPinConfigure(GPS2_UART_RX);
115     GPIOPinConfigure(GPS2_UART_TX);
116     GPIOPinTypeUART(GPS2_GPIO_BASE, GPS2_UART_RX_PIN | GPS2_UART_TX_PIN);
117     UARTConfigSetExpClk(GPS2_UART_BASE, g_sysClock, baudRate,
118                         (UART_CONFIG_PAR_NONE | UART_CONFIG_STOP_ONE |
119                          UART_CONFIG_WLEN_8));
120     UARTEnable(GPS2_UART_BASE);
121   }
122 }
123
124 //*****************************************************************************
125 //
126 // Initialize GPS Timepulse interrupt
127 // TODO: There is a problem if you initialize both of these
128 //     you will have to check which one fired in your handler routine
129 //
130 //*****************************************************************************
131 uint8_t gpsConfigTiempulseInt(gpsId id, uint32_t period, void (*pfnIntHandler)(void)) {
132   ubxPacketBufferRx* rxBufferTemp;
133
134   //Timepulse signal setup
135   //Try to receive the active timepulse settings
136   g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->class = UBX_CFG;
137   g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->id = UBX_CFG_TP5;
138   g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->length = UBX_POLL_LTH;
139   g_ubxPacketBufferTxActiveWrite[id]->txState = ubxTxRTS;
140   gpsPkgTriggerWrite(id);
141   while (g_ubxPacketBufferTxActiveWrite[id]->txState != ubxTxDone)
142     ;
```

```
143     SysCtlDelay(10000000);    //TODO: Define these
144     //See if we received a correct uBlox package
145     if ((rxBufferTemp = ubxRxPacketFind(id, UBX_CFG, UBX_CFG_TP5)) == NULL)
146       return GPS_INIT_FAIL;
147
148     //Switch RX and TX packet buffer
149     ubxSwitchBuffer(rxBufferTemp, g_ubxPacketBufferTxActiveWrite[id]);
150     ubxBufferRxPkgRead(id, rxBufferTemp); //The RX buffer is now "dirty" and should be marked free and read
151     //Read the ACK, that followed the poll request
152     ubxPacketCheckAck(id, g_ubxPacketBufferTxActiveWrite[id]);
153     //Change the timepulse settings with the given values
154     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgTp5.freqPeriod = period;
155     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgTp5.freqPeriodLock = period;
156     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgTp5.pulseLenRatio = period>>1;
157     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgTp5.pulseLenRatioLock = period>>1;
158     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgTp5.version = 0; //This fits our module
159     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgTp5.flags |= (UBX_CFG_TP5_FLGS_ACT | UBX_CFG_TP5_FLGS_GPS | ↩
            UBX_CFG_TP5_FLGS_LEN | UBX_CFG_TP5_FLGS_ALG);
160     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgTp5.flags &= ~(UBX_CFG_TP5_FLGS_LOC | UBX_CFG_TP5_FLGS_FRQ | ↩
            UBX_CFG_TP5_FLGS_GRD);
161     g_ubxPacketBufferTxActiveWrite[id]->txState = ubxTxRTS;
162     gpsPkgTriggerWrite(id);
163     while (g_ubxPacketBufferTxActiveWrite[id]->txState != ubxTxDone)
164       ;
165     //Wait for answer
166     SysCtlDelay(10000000);
167     //Check if GPS module acknowledged the new settings
168     ubxPacketCheckAck(id, g_ubxPacketBufferTxActiveWrite[id]);
169     if (g_ubxPacketBufferTxActiveWrite[id]->txState != ubxTxACK)
170       return GPS_INIT_FAIL;
171
172     //Which module do we update the interrupt settings for?
173     if (id == gps1) {
174       SysCtlPeripheralEnable(GPS1_TIMEPULSE_PERIPH);
175       GPIOPinTypeGPIOInput(GPS1_TIMEPULSE_BASE, GPS1_TIMEPULSE_PIN);
176       GPIOIntTypeSet(GPS1_TIMEPULSE_BASE, GPS1_TIMEPULSE_PIN, GPIO_RISING_EDGE);
177       GPIOIntRegister(GPS1_TIMEPULSE_BASE, pfnIntHandler);
178       GPIOIntEnable(GPS1_TIMEPULSE_BASE, GPS1_TIMEPULSE_INT_PIN);
179     } else if (id == gps2) {
180       SysCtlPeripheralEnable(GPS2_TIMEPULSE_PERIPH);
181       GPIOPinTypeGPIOInput(GPS2_TIMEPULSE_BASE, GPS2_TIMEPULSE_PIN);
182       GPIOIntTypeSet(GPS2_TIMEPULSE_BASE, GPS2_TIMEPULSE_PIN, GPIO_RISING_EDGE);
183       GPIOIntRegister(GPS2_TIMEPULSE_BASE, pfnIntHandler);
184       GPIOIntEnable(GPS2_TIMEPULSE_BASE, GPS2_TIMEPULSE_INT_PIN);
185     }
186
187     return GPS_INIT_SUCCESS;
188   }
189
190
191   void gpsPkgTriggerWrite(gpsId id) {
192     //Trigger a one-byte write, the rest will be interrupt controlled
193     ubxPacketByteWrite(id, g_ubxPacketBufferTxActiveRead[id]);
194   }
195
196   void gps1PkgIsr(void) {
197     uint32_t status;
198
199     status = UARTIntStatus(GPS1_UART_BASE, true);
200     UARTIntClear(GPS1_UART_BASE, status);
201
202     if (status & UART_INT_RX) {
203       //TODO: Check for the buffer state to maybe set an error flag when missing bytes.
204       while (UARTCharsAvail(GPS1_UART_BASE))
205         if (ubxPacketByteRead(gps1, g_ubxPacketBufferRxActiveWrite[gps1]) == ubxRxDataReady) {
206           //Tag the buffer with the timestamp
207           g_ubxPacketBufferRxActiveWrite[gps1]->timestamp = g_gpsTimestamp;
208           //If a packet has been received, use next buffer for receiving data
209           g_ubxPacketBufferRxActiveWrite[gps1] = g_ubxPacketBufferRxActiveWrite[gps1]->next;
210           //If next place buffer has not yet been read, we outran the reading process, thus the old package is lost
211           if (g_ubxPacketBufferRxActiveWrite[gps1]->rxState != ubxRxFree)
212             ubxBufferRxPkgRead(gps1, g_ubxPacketBufferRxActiveWrite[gps1]);
213         }
214     } if (status & UART_INT_TX) {
215       ubxPacketByteWrite(gps1, g_ubxPacketBufferTxActiveRead[gps1]);
216     }
217   }
218
219
220   void gps2PkgIsr(void) {
221     uint32_t status;
222
223     status = UARTIntStatus(GPS2_UART_BASE, true);
224     UARTIntClear(GPS2_UART_BASE, status);
225
```

```
226     if (status & UART_INT_RX) {
227       //TODO: Check for the buffer state to maybe set an error flag when missing bytes.
228       while (UARTCharsAvail(GPS2_UART_BASE))
229         if (ubxPacketByteRead(gps2, g_ubxPacketBufferRxActiveWrite[gps2]) == ubxRxDataReady) {
230           //Tag the buffer with the timestamp
231           g_ubxPacketBufferRxActiveWrite[gps2]->timestamp = g_gpsTimestamp;
232           //If a packet has been received, use next buffer for receiving data
233           g_ubxPacketBufferRxActiveWrite[gps2] = g_ubxPacketBufferRxActiveWrite[gps2]->next;
234           //If next place buffer has not yet been read, we outran the reading process, thus the old package is lost
235           if (g_ubxPacketBufferRxActiveWrite[gps2]->rxState != ubxRxFree)
236             ubxBufferRxPkgRead(gps2, g_ubxPacketBufferRxActiveWrite[gps2]);
237         }
238     } if (status & UART_INT_TX) {
239       ubxPacketByteWrite(gps2, g_ubxPacketBufferTxActiveRead[gps2]);
240     }
241   }
242
243
244   void gpsEnablePkgInt(gpsId id, uint8_t priority) {
245     switch (id) {
246     case gps1:
247       IntPrioritySet(GPS1_UART_INT, priority);
248       UARTIntRegister(GPS1_UART_BASE, gps1PkgIsr);
249       UARTFIFOLevelSet(GPS1_UART_BASE, UART_FIFO_TX1_8, UART_FIFO_RX1_8);
250       UARTTxIntModeSet(GPS1_UART_BASE, UART_TXINT_MODE_EOT);
251       UARTFIFOEnable(GPS1_UART_BASE);
252       UARTIntEnable(GPS1_UART_BASE, UART_INT_RX | UART_INT_TX);
253       break;
254     case gps2:
255       IntPrioritySet(GPS2_UART_INT, priority);
256       UARTIntRegister(GPS2_UART_BASE, gps2PkgIsr);
257       UARTFIFOLevelSet(GPS2_UART_BASE, UART_FIFO_TX1_8, UART_FIFO_RX1_8);
258       UARTTxIntModeSet(GPS2_UART_BASE, UART_TXINT_MODE_EOT);
259       UARTFIFOEnable(GPS2_UART_BASE);
260       UARTIntEnable(GPS2_UART_BASE, UART_INT_RX | UART_INT_TX);
261       break;
262     default:
263       break;
264     }
265   }
266
267
268   void gpsDisablePkgInt(gpsId id) {
269     switch (id) {
270     case gps1:
271       UARTIntDisable(GPS1_UART_BASE, UART_INT_RX | UART_INT_TX);
272       break;
273     case gps2:
274       UARTIntDisable(GPS2_UART_BASE, UART_INT_RX | UART_INT_TX);
275       break;
276     default:
277       break;
278     }
279   }
280
281   uint8_t gpsSetMsgRate(gpsId id, uint8_t ubxClass, uint8_t ubxId, uint8_t rate) {
282     //Navigation fix updates
283     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->class = UBX_CFG;
284     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->id = UBX_CFG_MSG;
285     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->length = UBX_CFG_MSG_LEN;
286     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgMsgRate.msgClass = ubxClass;
287     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgMsgRate.msgID = ubxId;
288     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgMsgRate.rate = rate;
289     g_ubxPacketBufferTxActiveWrite[id]->txState = ubxTxRTS;
290     gpsPkgTriggerWrite(id);
291     while (g_ubxPacketBufferTxActiveWrite[id]->txState != ubxTxDone)
292       ;
293     //Wait for ACK
294     SysCtlDelay(10000000);
295     ubxPacketCheckAck(id, g_ubxPacketBufferTxActiveWrite[id]);
296     if (g_ubxPacketBufferTxActiveWrite[id]->txState != ubxTxACK)
297       return GPS_INIT_FAIL;
298
299     return GPS_INIT_SUCCESS;
300   }
301
302
303   ubxPacketBufferRx* ubxRxPacketFind(gpsId gpsId, uint8_t class, uint8_t id) {
304     ubxPacketBufferRx* rxBufferTemp;
305
306     rxBufferTemp = (ubxPacketBufferRx*)g_ubxPacketBufferRxActiveRead[gpsId];
307     while (rxBufferTemp != g_ubxPacketBufferRxActiveWrite[gpsId]) {
308       if ((rxBufferTemp->ubxPacket->class == class) &&
309         (rxBufferTemp->ubxPacket->id == id))
310         return rxBufferTemp;
```

```
311       rxBufferTemp = rxBufferTemp->next;
312     }
313
314     return NULL;
315   }
316
317   uint8_t gpsSetBaudrate(gpsId id, uint32_t baudRate) {
318     ubxPacketBufferRx* rxBufferTemp;
319
320     //Start communication with 9600 baud
321     gpsInitUART(id, 9600);
322
323     //Try to receive the active port settings for UART
324     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->class = UBX_CFG;
325     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->id = UBX_CFG_PRT;
326     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->length = UBX_POLL_LTH;
327     g_ubxPacketBufferTxActiveWrite[id]->txState = ubxTxRTS;
328     gpsPkgTriggerWrite(id);
329     while (g_ubxPacketBufferTxActiveWrite[id]->txState != ubxTxDone)
330       ;
331     SysCtlDelay(10000000);     //TODO: Define these
332
333     //See if we received a correct uBlox package
334     if ((rxBufferTemp = ubxRxPacketFind(id, UBX_CFG, UBX_CFG_PRT)) == NULL) {
335       //If not, see if the baudrate has already been set
336       //TODO: This would be the place to do autobauding
337       gpsInitUART(id, baudRate);
338       //And retransmit the poll request
339       g_ubxPacketBufferTxActiveWrite[id]->txState = ubxTxRTS;
340       gpsPkgTriggerWrite(id);
341       //Still no data?
342       while (g_ubxPacketBufferTxActiveWrite[id]->txState != ubxTxDone)
343         ;
344       SysCtlDelay(10000000);
345       if ((rxBufferTemp = ubxRxPacketFind(id, UBX_CFG, UBX_CFG_PRT)) == NULL)
346         return GPS_INIT_FAIL;
347     }
348
349     //Switch RX and TX packet buffer
350     ubxSwitchBuffer(rxBufferTemp, g_ubxPacketBufferTxActiveWrite[id]);
351     ubxBufferRxPkgRead(id, rxBufferTemp); //The RX buffer is now "dirty" and should be marked free and read
352     //Read the ACK, that followed the poll request
353     ubxPacketCheckAck(id, g_ubxPacketBufferTxActiveWrite[id]);
354     //Change the baudrate to the given value and disable NMEA to unclutter the UART
355     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgPrtUART.outProtoMask = UBX_CFG_PRT_OUTPRO_UBX;
356     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgPrtUART.baudRate = baudRate;
357     g_ubxPacketBufferTxActiveWrite[id]->txState = ubxTxRTS;
358     gpsPkgTriggerWrite(id);
359     while (g_ubxPacketBufferTxActiveWrite[id]->txState != ubxTxDone)
360       ;
361     //Increase UART speed
362     gpsInitUART(id, baudRate);
363
364     //Wait for answer
365     SysCtlDelay(10000000);
366
367     //Check if GPS module acknowledged the new settings
368     ubxPacketCheckAck(id, g_ubxPacketBufferTxActiveWrite[id]);
369     if (g_ubxPacketBufferTxActiveWrite[id]->txState != ubxTxACK)
370       return GPS_INIT_FAIL;
371
372     return GPS_INIT_SUCCESS;
373   }
374
375   //*****************************************************************************
376   //
377   // Initialize GPS module
378   //
379   //*****************************************************************************
380   uint8_t gpsInitModule(gpsId id, uint32_t baudRate, uint8_t interruptPriority) {
381     ubxInitBuffers(id);
382       gpsInitUART(id, 9600);
383     //Filter navigation messages
384     ubxFilterClassSet(UBX_NAV);
385       gpsEnablePkgInt(id, interruptPriority);
386
387     if (gpsSetBaudrate(id, baudRate) != GPS_INIT_SUCCESS)
388       return GPS_INIT_FAIL;
389
390     //Navigation fix updates
391     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->class = UBX_CFG;
392     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->id = UBX_CFG_RATE;
393     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->length = UBX_CFG_RATE_LEN;
394     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgRateSettings.measRate = GPS_FIX_PERIOD;
395     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgRateSettings.navRate = 1;  //This is per definition
```

```
396     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxCfgRateSettings.timeRef = GPS_TIME_ALIGN;
397     g_ubxPacketBufferTxActiveWrite[id]->txState = ubxTxRTS;
398     gpsPkgTriggerWrite(id);
399     while (g_ubxPacketBufferTxActiveWrite[id]->txState != ubxTxDone)
400       ;
401     //Wait for ACK
402     SysCtlDelay(10000000);
403     //Check if GPS module acknowledged the new settings
404     ubxPacketCheckAck(id, g_ubxPacketBufferTxActiveWrite[id]);
405     if (g_ubxPacketBufferTxActiveWrite[id]->txState != ubxTxACK)
406       return GPS_INIT_FAIL;
407
408     if (gpsSetMsgRate(id, UBX_NAV, UBX_NAV_PVT, 1) == GPS_INIT_FAIL)
409       return GPS_INIT_FAIL;
410
411     return GPS_INIT_SUCCESS;
412   }
413
414   //TODO: Can it be woken up?
415   void gpsPowerDown(gpsId id) {
416     //Filter navigation messages
417     ubxFilterClassSet(UBX_NAV);
418     //Set backup power mode
419     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->class = UBX_RXM;
420     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->id = UBX_RXM_PMREQ;
421     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->length = UBX_RXM_PMREQ_LEN;
422     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxRxmPmreq.duration = 0;  //Stay there forever
423     g_ubxPacketBufferTxActiveWrite[id]->ubxPacket->ubxRxmPmreq.flags = UBX_RXM_PMREQ_FLGS_BKP;  //Stay there forever
424     g_ubxPacketBufferTxActiveWrite[id]->txState = ubxTxRTS;
425     gpsPkgTriggerWrite(id);
426     while (g_ubxPacketBufferTxActiveWrite[id]->txState != ubxTxDone)
427       ;
428   }
429
430   void gpsTimepulseWrapperIsr(void) {
431     switch (tosHelper.state) {
432     case gpsTOSInit:
433       TimerEnable(TIMER2_BASE, TIMER_A);
434       tosHelper.state = gpsTOSBusy;
435       break;
436     case gpsTOSWaiting:
437       //This checks if (with high probability) this is a TOS timepulse
438       if (((TimerValueGet(TIMER2_BASE, TIMER_A) + ((g_sysClock / tosHelper.frequency)>>1)) % g_sysClock) < (g_sysClock / ↩
              tosHelper.frequency)) {
439         TimerDisable(TIMER2_BASE, TIMER_A);
440         tosHelper.state = gpsTOSTrained;
441         tosHelper.isTOS = 1;
442       }
443       break;
444     case gpsTOSTrained:
445       tosHelper.precisionCounter++;
446       //If this is TOS timepulse, mark that!
447       if (!(tosHelper.precisionCounter %= tosHelper.frequency))
448         tosHelper.isTOS = 1;
449       else
450         tosHelper.isTOS = 0;
451       //The timestamping is hardcoded for now (every 1/10 of a second)
452       if (!(tosHelper.precisionCounter % 100))
453         g_gpsTimestamp = g_relTimeMs;
454       //Call the user defined ISR
455       if (g_gpsTimepulseIntEnabled)
456         g_gpsTimepulseIsr();
457       break;
458     default:
459       break;
460     }
461     GPIOIntClear(GPS1_TIMEPULSE_BASE, GPS1_TIMEPULSE_INT_PIN);
462     GPIOIntClear(GPS2_TIMEPULSE_BASE, GPS2_TIMEPULSE_INT_PIN);
463   }
464
465   inline void gpsTrainTopOfSecond(gpsId id) {
466     //Reset values in TOS helper
467     tosHelper.state = gpsTOSInit;
468     tosHelper.precisionCounter = 0;
469     //Setup the local reference timer
470     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
471     TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);
472     //TimerEnable(TIMER1_BASE, TIMER_BOTH);
473     TimerLoadSet(TIMER2_BASE, TIMER_A, g_sysClock);
474     TimerPrescaleSet(TIMER2_BASE, TIMER_A, 0);
475     TimerControlStall(TIMER2_BASE, TIMER_A, true);  //Stop timer in debug mode.
476     //Generate timepulse every second
477     if (gpsConfigTiempulseInt(id, 1000000, gpsTimepulseWrapperIsr) != GPS_INIT_SUCCESS)
478       return;
479
```

```
480   while (tosHelper.state != gpsTOSBusy)
481     ;
482   SysCtlDelay(100000);
483   //Increase timepulse to target frequency
484   if (gpsConfigTiempulseInt(id, 1000000/tosHelper.frequency, gpsTimepulseWrapperIsr) != GPS_INIT_SUCCESS)
485     return;
486
487   //Wait for identification of a TOS timepulse
488   tosHelper.state = gpsTOSWaiting;
489   while (tosHelper.state != gpsTOSTrained)
490     ;
491 }
492
493 uint8_t gpsTimepulseIntSetup(gpsId id, uint32_t frequency, void (*pfnIntHandler)(void), uint8_t priority) {
494   switch (id) {
495   case gps1:
496     IntPrioritySet(GPS1_TIMEPULSE_INT, priority);
497     break;
498   case gps2:
499     IntPrioritySet(GPS2_TIMEPULSE_INT, priority);
500     break;
501   default:
502     break;
503   }
504
505   g_gpsTimepulseIsr = pfnIntHandler;
506   tosHelper.frequency = frequency;
507   gpsTrainTopOfSecond(id);
508
509   return GPS_INIT_SUCCESS;
510 }
511
512 void ubxSwitchBuffer(volatile ubxPacketBufferRx* rxBuffer, volatile ubxPacketBufferTx* txBuffer) {
513   ubxPacket* temp = rxBuffer->ubxPacket;
514   rxBuffer->ubxPacket = txBuffer->ubxPacket;
515   txBuffer->ubxPacket = temp;
516   rxBuffer->rxState = ubxRxFree;
517 }
518
519 void ubxPacketResetChecksum(volatile ubxPacket* packet) {
520   //Clear checksum
521   *((uint16_t*)packet->checksum) = 0x0000;
522 }
523
524 ubxPacketStateTx ubxPacketByteWrite(gpsId id, volatile ubxPacketBufferTx* txBuffer) {
525   //TODO: This is a function mainly called from interrupts, thus variables should not be declared here.
526   uint32_t uartBase;
527
528   //Decide which UART_BASE to use
529   if (id == gps1)
530     uartBase = GPS1_UART_BASE;
531   else if (id == gps2)
532     uartBase = GPS2_UART_BASE;
533
534   switch (txBuffer->txState) {
535   case ubxTxWriting:
536     //Write until all of the payload has been sent
537     if ((txBuffer->readPosition - txBuffer->ubxPacket->payload) < txBuffer->ubxPacket->length) {
538       txBuffer->ubxPacket->checksum[0] += *(txBuffer->readPosition);
539       txBuffer->ubxPacket->checksum[1] += txBuffer->ubxPacket->checksum[0];
540     } else if ((txBuffer->readPosition - txBuffer->ubxPacket->payload) < (txBuffer->ubxPacket->length + 1)) //Writing first ↩
          checksum byte
541       txBuffer->readPosition = (uint8_t*)txBuffer->ubxPacket->checksum;
542     else
543       txBuffer->txState = ubxTxDone;  //Writing last checksum byte
544
545     UARTCharPut(uartBase, *(txBuffer->readPosition++));
546     break;
547   case ubxTxSyncing:
548     txBuffer->txState = ubxTxWriting;
549     UARTCharPut(uartBase, UBX_SYNC2);
550     break;
551   case ubxTxRTS:
552     txBuffer->readPosition = (uint8_t*)&(txBuffer->ubxPacket->class);
553     txBuffer->ubxPacket->checksum[0] = 0x00;
554     txBuffer->ubxPacket->checksum[1] = 0x00;
555     txBuffer->txState = ubxTxSyncing;
556     UARTCharPut(uartBase, UBX_SYNC1);
557     break;
558   default:
559     break;
560   }
561
562   return txBuffer->txState;
563 }
```

```
564
565 ubxPacketStateRx ubxPacketByteRead(gpsId id, volatile ubxPacketBufferRx* rxBuffer) {
566   //TODO: This is a function mainly called from interrupts, thus variables should not be declared here.
567   uint32_t uartBase;
568   uint8_t byte;
569
570   //Decide which UART_BASE to use
571   if (id == gps1)
572     uartBase = GPS1_UART_BASE;
573   else if (id == gps2)
574     uartBase = GPS2_UART_BASE;
575
576   byte = UARTCharGet(uartBase);
577
578   switch (rxBuffer->rxState) {
579   case ubxRxChecking:
580     rxBuffer->ubxPacket->checksum[1] = byte;
581     //Check if this is a filtered message class
582     if (!(rxBuffer->ubxPacket->class & ~g_ubxPacketFilterClass)) {
583       rxBuffer->rxState = ubxRxFree;
584       break;
585     }
586     //Check if checksum is correct
587     if ((rxBuffer->ubxPacket->checksum[0] == rxBuffer->calcChecksum[0]) &&
588       (rxBuffer->ubxPacket->checksum[1] == rxBuffer->calcChecksum[1]))
589       rxBuffer->rxState = ubxRxDataReady;
590     else
591       rxBuffer->rxState = ubxRxError;
592     break;
593   case ubxRxReading:
594     //Receive until all of the payload has been read
595     if ((rxBuffer->writePosition - rxBuffer->ubxPacket->payload) < rxBuffer->ubxPacket->length) {
596       *(rxBuffer->writePosition++) = byte;
597       rxBuffer->calcChecksum[0] += byte;
598       rxBuffer->calcChecksum[1] += rxBuffer->calcChecksum[0];
599     } else {
600       rxBuffer->ubxPacket->checksum[0] = byte;
601       rxBuffer->rxState = ubxRxChecking;
602     }
603     break;
604   case ubxRxSyncing:
605     if (byte == UBX_SYNC2) {
606       rxBuffer->calcChecksum[0] = 0x00;
607       rxBuffer->calcChecksum[1] = 0x00;
608       rxBuffer->ubxPacket->length = 0;
609       rxBuffer->writePosition = (uint8_t*)&(rxBuffer->ubxPacket->class);
610       rxBuffer->rxState = ubxRxReading;
611     } else
612       rxBuffer->rxState = ubxRxFree;
613     break;
614   case ubxRxFree:
615     if (byte == UBX_SYNC1)
616       rxBuffer->rxState = ubxRxSyncing;
617     break;
618   default:
619     break;
620   }
621
622   return rxBuffer->rxState;
623 }
624
625 uint8_t ubxPacketCheckAck(gpsId id, volatile ubxPacketBufferTx* txPkg) {
626   ubxPacketBufferRx* rxBufferTemp;
627
628   rxBufferTemp = (ubxPacketBufferRx*)g_ubxPacketBufferRxActiveRead[id];
629   while (rxBufferTemp != g_ubxPacketBufferRxActiveWrite[id]) {
630     if ((rxBufferTemp->ubxPacket->class == UBX_ACK) &&
631       (rxBufferTemp->ubxPacket->id == UBX_ACK_ACK) &&
632       (rxBufferTemp->ubxPacket->payload[0] == txPkg->ubxPacket->class) &&
633       (rxBufferTemp->ubxPacket->payload[1] == txPkg->ubxPacket->id)) {
634         txPkg->txState = ubxTxACK;
635         ubxBufferRxPkgRead(id, rxBufferTemp);
636         return UBX_ACK_MATCH;
637       }
638     rxBufferTemp = rxBufferTemp->next;
639   }
640
641   //ID for NAK received or malformed package
642   return UBX_ACK_NOT;
643 }
```

Quellcode D.8: Software-Modul: MATLAB-Dateiformat (Header)

```
1  /*
2   * mat4.h
3   *
4   * Author: Felix Groth
5   */
6
7  #ifndef MAT4_H_
8  #define MAT4_H_
9
10 //Standard includes
11 #include <stdint.h>
12 #include <stdbool.h>
13 #include <string.h>
14
15 //Hardware includes
16 #include "inc/hw_memmap.h"
17 #include "inc/hw_ssi.h"
18 #include "inc/hw_types.h"
19 #include "inc/hw_gpio.h"
20
21 //Driver includes
22 #include "driverlib/sysctl.h"
23 #include "driverlib/rom.h"
24 #include "driverlib/gpio.h"
25 #include "driverlib/pin_map.h"
26 #include "driverlib/ssi.h"
27 #include "driverlib/interrupt.h"
28 #include "driverlib/systick.h"
29
30 //Module includes
31 #include "fatfs/src/ff.h"
32 #include "fatfs/src/diskio.h"
33
34
35 /** Defines for matlab files **/
36 #define MAT_FILE_HEADER_SIZE  20
37 //#define MAT_FILE_COUNT       10
38 #define MAT_MAX_VARNAME      255
39
40 //These need to be added together when constructing type information
41 #define MAT_TYPE_LE      0
42 #define MAT_TYPE_BE      1000
43 #define MAT_TYPE_DOUBLE   0
44 #define MAT_TYPE_FLOAT    10
45 #define MAT_TYPE_INT32    20
46 #define MAT_TYPE_INT16    30
47 #define MAT_TYPE_UINT16   40
48 #define MAT_TYPE_UINT8    50
49 #define MAT_TYPE_MATFULL  0
50 #define MAT_TYPE_MATTEXT  1
51 #define MAT_TYPE_MATSPARSE  2
52
53 /** Struct for mat-file ver. 4 header -- mathworks documentation for matfile-format **/
54 typedef struct {
55   uint32_t type;
56   uint32_t rows;
57   uint32_t columns;
58   uint32_t imagflag;
59   uint32_t namelength;
60 } mat4Header;
61
62 typedef struct {
63   FIL file;
64   uint32_t type;
65   DWORD dataStart;
66   uint32_t elementCount;
67   char varname[MAT_MAX_VARNAME];
68   mat4Header header;
69 } mat4File;
70
71
72 FRESULT matFileHeaderWrite(mat4File* matFile);
73 FRESULT matFileVectorCreate(mat4File* matFile, const char* filename, const char* varname, uint32_t columns, uint32_t type);
74 FRESULT matFileMatrixCreate(mat4File* matFile, const char* filename, const char* varname, uint32_t columns, uint32_t rows, uint32_t ↩
       type);
75 FRESULT matFileAddValue(mat4File* matFile, uint8_t* val);
76 void matFileClose(mat4File* matFile);
77
78 #endif /* MAT4_H_ */
```

Quellcode D.9: Software-Modul: MATLAB-Dateiformat

```
/*
 * mat4.c
 *
 * Author: Felix Groth
 */


#include "matlab/mat4.h"


/** Global variables **/
//Lookup table for the byte size of the given matlab type
static uint8_t g_matByteSizeLookup[] = {8, 4, 4, 2, 2, 1};
//Temporarly holds the bytes written with f_write
static uint8_t bytesWritten;
//Temporarly holds return values for file operations
static FRESULT g_fResult;

// Write a MAT-File header into the file
FRESULT matFileHeaderWrite(mat4File* matFile) {
  g_fResult = f_write(&(matFile->file), (UINT*)&(matFile->header), MAT_FILE_HEADER_SIZE, (UINT*)&bytesWritten);
  if (g_fResult == FR_OK)
    g_fResult = f_write(&(matFile->file), (UINT*)matFile->varname, matFile->header.namelength, (UINT*)&bytesWritten);

  matFile->dataStart = f_tell(&(matFile->file));
  return g_fResult;
}

// Create a vector file
FRESULT matFileVectorCreate(mat4File* matFile, const char* filename, const char* varname, uint32_t columns, uint32_t type) {
  //Open the file
  g_fResult = f_open(&(matFile->file), filename, FA_WRITE | FA_READ | FA_CREATE_ALWAYS);
  //Check for file open errors
  if (g_fResult != FR_OK)
    return g_fResult;

  //Setup the header
  if ((matFile->header.namelength = strlen(varname)) < MAT_MAX_VARNAME) {
    matFile->header.columns = 0;
    matFile->header.rows = 1;
    matFile->header.imagflag = 0;
    matFile->header.type = MAT_TYPE_LE + type + MAT_TYPE_MATFULL;
    matFile->type = type;
    strcpy(matFile->varname, varname);
  } else
    return FR_INVALID_PARAMETER;

  //Write header information
  g_fResult = matFileHeaderWrite(matFile);
  if (g_fResult != FR_OK)
    return g_fResult;

  //Pre allocate space
  g_fResult = f_lseek(&(matFile->file), f_tell(&(matFile->file)) + (g_matByteSizeLookup[type/10] * columns));
  //Bail out if seeking failed
  if (g_fResult != FR_OK)
    return g_fResult;
  //Check for successfull allocation
  if (f_tell(&(matFile->file)) != matFile->dataStart + (g_matByteSizeLookup[type/10] * columns))
    return FR_INVALID_PARAMETER;

  //Move back to correct position in file
  f_lseek(&(matFile->file), matFile->dataStart);

  return FR_OK;
}

// Add values to a file
FRESULT matFileAddValue(mat4File* matFile, uint8_t* val) {
  g_fResult = f_write(&(matFile->file), (UINT*)val, g_matByteSizeLookup[matFile->type/10] * matFile->header.rows, ↩
        (UINT*)&bytesWritten);
  if (g_fResult == FR_OK)
    matFile->header.columns++;

  return g_fResult;
}

// Create a matrix file
FRESULT matFileMatrixCreate(mat4File* matFile, const char* filename, const char* varname, uint32_t columns, uint32_t rows, uint32_t ↩
      type) {
  //Open the file
  g_fResult = f_open(&(matFile->file), filename, FA_WRITE | FA_READ | FA_CREATE_ALWAYS);
  //Check for file open errors
  if (g_fResult != FR_OK)
```

```
83      return g_fResult;
84
85    //Setup the header
86    if ((matFile->header.namelength = strlen(varname)) < MAT_MAX_VARNAME) {
87      matFile->header.columns = 0;
88      matFile->header.rows = rows;
89      matFile->header.imagflag = 0;
90      matFile->header.type = MAT_TYPE_LE + type + MAT_TYPE_MATFULL;
91      matFile->type = type;
92      strcpy(matFile->varname, varname);
93    } else
94      return FR_INVALID_PARAMETER;
95
96    //Write header information
97    g_fResult = matFileHeaderWrite(matFile);
98    if (g_fResult != FR_OK)
99      return g_fResult;
100
101   //Pre allocate space
102   g_fResult = f_lseek(&(matFile->file), f_tell(&(matFile->file)) + (g_matByteSizeLookup[type/10] * columns * rows));
103   //Bail out if seeking failed
104   if (g_fResult != FR_OK)
105     return g_fResult;
106   //Check for successfull allocation
107   if (f_tell(&(matFile->file)) != matFile->dataStart + (g_matByteSizeLookup[type/10] * columns * rows))
108     return FR_INVALID_PARAMETER;
109
110   //Move back to correct position in file
111   f_lseek(&(matFile->file), matFile->dataStart);
112
113   return FR_OK;
114 }
115
116 // Close the MAT-File
117 void matFileClose(mat4File* matFile) {
118   //Truncate the file to where we have written
119   //Matlab wants this or it will complain about the file format
120   f_truncate(&(matFile->file));
121
122   //Update header information
123   f_lseek(&(matFile->file), 0);
124   matFileHeaderWrite(matFile);
125
126   //Close file when done
127   f_close(&(matFile->file));
128 }
```

Quellcode D.10: Software-Modul: Taster (Header)

```
1  /*
2   * buttons.h
3   *
4   * Author: Felix Groth
5   */
6
7  #ifndef BUTTONS_H_
8  #define BUTTONS_H_
9
10 //Standard includes
11 #include <stdint.h>
12 #include <stdbool.h>
13
14 //Hardware includes
15 #include "inc/tm4c1294ncpdt.h"
16 #include "inc/hw_gpio.h"
17 #include "inc/hw_memmap.h"
18
19 //Driver includes
20 #include "driverlib/sysctl.h"
21 #include "driverlib/gpio.h"
22 #include "driverlib/interrupt.h"
23
24 /** Defines for buttons hardware ports **/
25 #define BTN1_PORT_BASE   GPIO_PORTB_BASE
26 #define BTN2_PORT_BASE   GPIO_PORTK_BASE
27
28 #define BTN1_PORT_SYSCTL   SYSCTL_PERIPH_GPIOB
29 #define BTN2_PORT_SYSCTL   SYSCTL_PERIPH_GPIOK
30
31 #define BTN1_PIN   GPIO_PIN_5
32 #define BTN2_PIN   GPIO_PIN_2
```

```
33
34 #define BTN1_SHIFT  5
35 #define BTN2_SHIFT  2
36
37 #define BTN1_INT  INT_GPIOB
38 #define BTN2_INT  INT_GPIOK
39
40
41 /** Defines for functions return values **/
42 #define BTN1_BIT  0x01
43 #define BTN2_BIT  0x02
44
45
46 uint8_t pushButtonsRead(void);
47 void pushButtonsInit(void);
48 void pushButtonsIsrRegister(void (*pfnIntHandler)(void), uint8_t priority);
49 void pushButtonsIntClear(void);
50
51
52 #endif /* BUTTONS_H_ */
```

Quellcode D.11: Software-Modul: Taster

```
1  /*
2   * buttons.c
3   *
4   * Author: Felix Groth
5   */
6
7  #include "buttons.h"
8
9  //Read bit values from buttons
10 uint8_t pushButtonsRead(void)
11 {
12   uint8_t data;
13
14   data = (GPIOPinRead(BTN1_PORT_BASE, BTN1_PIN) & 0xFF)>>BTN1_SHIFT;
15   data |= (GPIOPinRead(BTN2_PORT_BASE, BTN2_PIN) & 0xFF)>>(BTN2_SHIFT-1);
16
17   return data;
18 }
19
20 //Read bit values from buttons
21 void pushButtonsInit(void) {
22     SysCtlPeripheralEnable(BTN1_PORT_SYSCTL);
23     SysCtlPeripheralEnable(BTN2_PORT_SYSCTL);
24
25     GPIOPinTypeGPIOInput(BTN1_PORT_BASE, BTN1_PIN);
26     GPIOPinTypeGPIOInput(BTN2_PORT_BASE, BTN2_PIN);
27 }
28
29 void pushButtonsIsrRegister(void (*pfnIntHandler)(void), uint8_t priority) {
30   //Enable interrupt for button 1
31   SysCtlPeripheralEnable(BTN1_PORT_SYSCTL);
32   GPIOPinTypeGPIOInput(BTN1_PORT_BASE, BTN1_PIN);
33   //GPIOPadConfigSet(GPS1_TIMEPULSE_BASE,GPS1_TIMEPULSE_PIN,GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_STD_WPU);
34   GPIOIntTypeSet(BTN1_PORT_BASE, BTN1_PIN, GPIO_FALLING_EDGE);
35   GPIOIntRegister(BTN1_PORT_BASE, pfnIntHandler);
36   GPIOIntEnable(BTN1_PORT_BASE, BTN1_PIN);
37   IntPrioritySet(BTN1_INT, priority);
38
39   //Enable interrupt for button 2
40   SysCtlPeripheralEnable(BTN2_PORT_SYSCTL);
41   GPIOPinTypeGPIOInput(BTN2_PORT_BASE, BTN2_PIN);
42   //GPIOPadConfigSet(GPS1_TIMEPULSE_BASE,GPS1_TIMEPULSE_PIN,GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_STD_WPU);
43   GPIOIntTypeSet(BTN2_PORT_BASE, BTN2_PIN, GPIO_FALLING_EDGE);
44   GPIOIntRegister(BTN2_PORT_BASE, pfnIntHandler);
45   GPIOIntEnable(BTN2_PORT_BASE, BTN2_PIN);
46   IntPrioritySet(BTN2_INT, priority);
47 }
48
49 void pushButtonsIntClear(void) {
50   GPIOIntClear(BTN1_PORT_BASE, BTN1_PIN);
51   GPIOIntClear(BTN2_PORT_BASE, BTN2_PIN);
52 }
```

Quellcode D.12: Software-Modul: Angepasster Display-Treiber (Header) [36]

```
1  //*****************************************************************************
2  //
3  // Kentec320x240x16_ssd2119_8bit.h - Prototypes for the Kentec K350QVG-V2-F
4  //                                   display driver with an SSD2119
5  //                                   controller.
6  //
7  // Copyright (c) 2012-2014 Texas Instruments Incorporated.  All rights reserved.
8  // Software License Agreement
9  //
10 // Texas Instruments (TI) is supplying this software for use solely and
11 // exclusively on TI's microcontroller products. The software is owned by
12 // TI and/or its suppliers, and is protected under applicable copyright
13 // laws. You may not combine this software with "viral" open-source
14 // software in order to form a larger program.
15 //
16 // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
17 // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
18 // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
19 // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
20 // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
21 // DAMAGES, FOR ANY REASON WHATSOEVER.
22 //
23 // This is part of revision 2.1.0.12573 of the EK-TM4C1294XL Firmware Package.
24 //
25 //*****************************************************************************
26
27 #ifndef __KENTEC320X240X16_SSD2119_8BIT_H__
28 #define __KENTEC320X240X16_SSD2119_8BIT_H__
29
30 //*****************************************************************************
31 //
32 // Bit definitions for the LCD control registers in the SRAM/Flash daughter
33 // board.
34 //
35 //*****************************************************************************
36 #define LCD_CONTROL_NRESET      0x04
37 #define LCD_CONTROL_YN          0x02
38 #define LCD_CONTROL_XN          0x01
39
40 //*****************************************************************************
41 //
42 // EPI addresses used to access the LCD when the SRAM/Flash daughter board is
43 // installed.
44 //
45 //*****************************************************************************
46 #define LCD_COMMAND_PORT      0x6C000002
47 #define LCD_DATA_PORT         0x6C000003
48 #define LCD_CONTROL_SET_REG 0x6C000000
49 #define LCD_CONTROL_CLR_REG 0x6C000001
50
51 //
52 // Read start bit.  This is ORed with LCD_COMMAND_PORT or LCD_DATA_PORT to
53 // initiate a read request from the LCD controller.
54 //
55 #define LCD_READ_START        0x00000004
56
57 //*****************************************************************************
58 //
59 // Backlight control GPIO used with the Flash/SRAM/LCD daughter board.
60 // Defined in the header, because we need them in the main program flow.
61 //
62 //*****************************************************************************
63 #define LCD_BACKLIGHT_PERIPH      SYSCTL_PERIPH_GPIOF
64 #define LCD_BACKLIGHT_BASE        GPIO_PORTF_AHB_BASE
65 #define LCD_BACKLIGHT_PIN         GPIO_PIN_1
66
67 //*****************************************************************************
68 //
69 // Prototypes for the globals exported by this driver.
70 //
71 //*****************************************************************************
72 extern void Kentec320x240x16_SSD2119Init(uint32_t ui32SysClockSpeed);
73 extern const tDisplay g_sKentec320x240x16_SSD2119;
74 extern void Kentec320x240x16_SSD2119SetLCDControl(uint8_t ui8Mask,
75                                                   uint8_t ui8Val);
76 extern void LED_ON(void);
77 extern void LED_OFF(void);
78 #endif // __KENTEC320X240X16_SSD2119_H__
```

Quellcode D.13: Software-Modul: Angepasster Display-Treiber [36]

```
1  //*****************************************************************************
2  //
3  // Kentec320x240x16_ssd2119_8bit.c - Display driver for the Kentec
4  //                                   K350QVG-V2-F TFT display with an SSD2119
5  //                                   controller.  This version assumes an
6  //                                   8080-8bit interface between the micro
7  //                                   and display (PS3-0 = 0011b).
8  //
9  // Copyright (c) 2012-2014 Texas Instruments Incorporated.  All rights reserved.
10 // Software License Agreement
11 //
12 // Texas Instruments (TI) is supplying this software for use solely and
13 // exclusively on TI's microcontroller products. The software is owned by
14 // TI and/or its suppliers, and is protected under applicable copyright
15 // laws. You may not combine this software with "viral" open-source
16 // software in order to form a larger program.
17 //
18 // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
19 // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
20 // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
21 // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
22 // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
23 // DAMAGES, FOR ANY REASON WHATSOEVER.
24 //
25 // This is part of revision 2.1.0.12573 of the EK-TM4C1294XL Firmware Package.
26 //
27 //*****************************************************************************
28
29 //*****************************************************************************
30 //
31 //! \addtogroup display_api
32 //! @{
33 //
34 //*****************************************************************************
35
36 #include <stdbool.h>
37 #include <stdint.h>
38 #include "inc/hw_gpio.h"
39 #include "inc/hw_ints.h"
40 #include "inc/hw_memmap.h"
41 #include "inc/hw_types.h"
42 #include "driverlib/gpio.h"
43 #include "driverlib/epi.h"
44 #include "driverlib/interrupt.h"
45 #include "driverlib/sysctl.h"
46 #include "driverlib/timer.h"
47 #include "driverlib/rom.h"
48 #include "grlib/grlib.h"
49 #include "display/Kentec320x240x16_ssd2119_8bit.h"
50
51 //*****************************************************************************
52 //
53 // This driver operates in four different screen orientations.  They are:
54 //
55 // * Portrait - The screen is taller than it is wide, and the flex connector is
56 //              on the left of the display.  This is selected by defining
57 //              PORTRAIT.
58 //
59 // * Landscape - The screen is wider than it is tall, and the flex connector is
60 //               on the bottom of the display.  This is selected by defining
61 //               LANDSCAPE.
62 //
63 // * Portrait flip - The screen is taller than it is wide, and the flex
64 //                   connector is on the right of the display.  This is
65 //                   selected by defining PORTRAIT_FLIP.
66 //
67 // * Landscape flip - The screen is wider than it is tall, and the flex
68 //                    connector is on the top of the display.  This is
69 //                    selected by defining LANDSCAPE_FLIP.
70 //
71 // These can also be imagined in terms of screen rotation; if portrait mode is
72 // 0 degrees of screen rotation, landscape is 90 degrees of counter-clockwise
73 // rotation, portrait flip is 180 degrees of rotation, and landscape flip is
74 // 270 degress of counter-clockwise rotation.
75 //
76 // If no screen orientation is selected, "landscape flip" mode will be used.
77 //
78 //*****************************************************************************
79 #if ! defined(PORTRAIT) && ! defined(PORTRAIT_FLIP) && \
80     ! defined(LANDSCAPE) && ! defined(LANDSCAPE_FLIP)
81 #define LANDSCAPE
82 //#define PORTRAIT
83 #endif
84
```

```
85
86  //*****************************************************************************
87  //
88  // Various definitions controlling coordinate space mapping and drawing
89  // direction in the four supported orientations.
90  //
91  //*****************************************************************************
92  #ifdef PORTRAIT
93  #define HORIZ_DIRECTION 0x28
94  #define VERT_DIRECTION 0x20
95  #define MAPPED_X(x, y) (319 - (y))
96  #define MAPPED_Y(x, y) (x)
97  #endif
98  #ifdef LANDSCAPE
99  #define HORIZ_DIRECTION 0x00
100 #define VERT_DIRECTION  0x08
101 #define MAPPED_X(x, y) (319 - (x))
102 #define MAPPED_Y(x, y) (239 - (y))
103 #endif
104 #ifdef PORTRAIT_FLIP
105 #define HORIZ_DIRECTION 0x18
106 #define VERT_DIRECTION 0x10
107 #define MAPPED_X(x, y) (y)
108 #define MAPPED_Y(x, y) (239 - (x))
109 #endif
110 #ifdef LANDSCAPE_FLIP
111 #define HORIZ_DIRECTION 0x30
112 #define VERT_DIRECTION  0x38
113 #define MAPPED_X(x, y) (x)
114 #define MAPPED_Y(x, y) (y)
115 #endif
116
117 //*****************************************************************************
118 //
119 // Defines for the pins that are used to communicate with the SSD2119.
120 //
121 //*****************************************************************************
122
123 //
124 // LCD Data line GPIO definitions.
125 //
126 #define LCD_DATAH_PERIPH_D      SYSCTL_PERIPH_GPIOD
127 #define LCD_DATAH_BASE_D        GPIO_PORTD_BASE
128 #define LCD_DATAH_D7            0x4005B008
129 #define LCD_DATAH_D7_S(a)        ((a) >> 6)
130 #define LCD_DATAH_D6            0x4005B004
131 #define LCD_DATAH_D6_S(a)        ((a) >> 6)
132 #define LCD_DATAH_D4            0x4005B020
133 #define LCD_DATAH_D4_S(a)        ((a) >> 1)
134 #define LCD_DATAH_PERIPH_E      SYSCTL_PERIPH_GPIOE
135 #define LCD_DATAH_BASE_E        GPIO_PORTE_AHB_BASE
136 #define LCD_DATAH_D5_S(a)        ((a) >> 1)
137 #define LCD_DATAH_D5            0x4005C040
138 #define LCD_DATAH_PERIPH_F      SYSCTL_PERIPH_GPIOF
139 #define LCD_DATAH_BASE_F        GPIO_PORTF_BASE
140 #define LCD_DATAH_D3_S(a)        (a)
141 #define LCD_DATAH_D3            0x4005D020
142 #define LCD_DATAH_PERIPH_M      SYSCTL_PERIPH_GPIOM
143 #define LCD_DATAH_BASE_M        GPIO_PORTM_BASE
144 #define LCD_DATAH_D2_S(a)        ((a) << 1)
145 #define LCD_DATAH_D2            0x40063020
146 #define LCD_DATAH_PERIPH_C      SYSCTL_PERIPH_GPIOC
147 #define LCD_DATAH_BASE_C        GPIO_PORTC_BASE
148 #define LCD_DATAH_D1_S(a)        ((a) << 4)
149 #define LCD_DATAH_D1            0x4005A080
150 #define LCD_DATAH_D0_S(a)        ((a) << 4)
151 #define LCD_DATAH_D0            0x4005A040
152
153 #define LCD_DATAH_PIN_7             GPIO_PIN_1
154 #define LCD_DATAH_PIN_6             GPIO_PIN_0
155 #define LCD_DATAH_PIN_5             GPIO_PIN_4
156 #define LCD_DATAH_PIN_4             GPIO_PIN_3
157 #define LCD_DATAH_PIN_3             GPIO_PIN_3
158 #define LCD_DATAH_PIN_2             GPIO_PIN_3
159 #define LCD_DATAH_PIN_1             GPIO_PIN_5
160 #define LCD_DATAH_PIN_0             GPIO_PIN_4
161
162 //
163 // LCD control line GPIO definitions.
164 //
165 #define LCD_CS_PERIPH           SYSCTL_PERIPH_GPIOB
166 #define LCD_CS_BASE             GPIO_PORTB_BASE
167 #define LCD_CS_PIN              GPIO_PIN_3
168 #define LCD_CS_PIN_REG            0x40059020
169 #define LCD_DC_PERIPH           SYSCTL_PERIPH_GPIOB
```

```
170 #define LCD_DC_BASE             GPIO_PORTB_BASE
171 #define LCD_DC_PIN              GPIO_PIN_2
172 #define LCD_DC_PIN_REG            0x40059010
173 #define LCD_WR_PERIPH           SYSCTL_PERIPH_GPIOC
174 #define LCD_WR_BASE             GPIO_PORTC_BASE
175 #define LCD_WR_PIN              GPIO_PIN_7
176 #define LCD_WR_PIN_REG              0x4005A200
177 #define LCD_RD_PERIPH           SYSCTL_PERIPH_GPION
178 #define LCD_RD_BASE             GPIO_PORTN_BASE
179 #define LCD_RD_PIN              GPIO_PIN_2
180 #define LCD_RD_PIN_REG              0x40064010
181
182 //*****************************************************************************
183 //
184 // Speed of the communication with the display
185 //
186 //*****************************************************************************
187 #define LCD_WRITE_DELAY    1
188
189 //*****************************************************************************
190 //
191 // Backlight control GPIO used with the Flash/SRAM/LCD daughter board.
192 //
193 //*****************************************************************************
194 //#define LCD_BACKLIGHT_PERIPH    SYSCTL_PERIPH_GPIOF
195 //#define LCD_BACKLIGHT_BASE      GPIO_PORTF_AHB_BASE
196 //#define LCD_BACKLIGHT_PIN       GPIO_PIN_1
197
198 #define USER_LED_PERIPH         SYSCTL_PERIPH_GPION
199 #define USER_LED_BASE           GPIO_PORTN_BASE
200 #define USER_LED_1_PIN          GPIO_PIN_0
201 #define USER_LED_2_PIN          GPIO_PIN_1
202
203 //*****************************************************************************
204 //
205 // Macro used to set the LCD data bus in preparation for writing a byte to the
206 // device.
207 //
208 //*****************************************************************************
209 #define SET_LCD_DATA(ui8Byte)                       \
210 {                                                   \
211     HWREG(LCD_DATAH_D7) = LCD_DATAH_D7_S(ui8Byte);    \
212     HWREG(LCD_DATAH_D6) = LCD_DATAH_D6_S(ui8Byte);    \
213     HWREG(LCD_DATAH_D5) = LCD_DATAH_D5_S(ui8Byte);     \
214     HWREG(LCD_DATAH_D4) = LCD_DATAH_D4_S(ui8Byte);    \
215     HWREG(LCD_DATAH_D3) = LCD_DATAH_D3_S(ui8Byte);     \
216     HWREG(LCD_DATAH_D2) = LCD_DATAH_D2_S(ui8Byte);     \
217     HWREG(LCD_DATAH_D1) = LCD_DATAH_D1_S(ui8Byte);    \
218     HWREG(LCD_DATAH_D0) = LCD_DATAH_D0_S(ui8Byte);    \
219 }
220
221 //*****************************************************************************
222 //
223 // Various internal SD2119 registers name labels
224 //
225 //*****************************************************************************
226 #define SSD2119_DEVICE_CODE_READ_REG  0x00
227 #define SSD2119_OSC_START_REG         0x00
228 #define SSD2119_OUTPUT_CTRL_REG       0x01
229 #define SSD2119_LCD_DRIVE_AC_CTRL_REG 0x02
230 #define SSD2119_PWR_CTRL_1_REG        0x03
231 #define SSD2119_DISPLAY_CTRL_REG      0x07
232 #define SSD2119_FRAME_CYCLE_CTRL_REG  0x0B
233 #define SSD2119_PWR_CTRL_2_REG        0x0C
234 #define SSD2119_PWR_CTRL_3_REG        0x0D
235 #define SSD2119_PWR_CTRL_4_REG        0x0E
236 #define SSD2119_GATE_SCAN_START_REG   0x0F
237 #define SSD2119_SLEEP_MODE_REG        0x10
238 #define SSD2119_ENTRY_MODE_REG        0x11
239 #define SSD2119_GEN_IF_CTRL_REG       0x15
240 #define SSD2119_PWR_CTRL_5_REG        0x1E
241 #define SSD2119_RAM_DATA_REG          0x22
242 #define SSD2119_FRAME_FREQ_REG        0x25
243 #define SSD2119_VCOM_OTP_1_REG        0x28
244 #define SSD2119_VCOM_OTP_2_REG        0x29
245 #define SSD2119_GAMMA_CTRL_1_REG      0x30
246 #define SSD2119_GAMMA_CTRL_2_REG      0x31
247 #define SSD2119_GAMMA_CTRL_3_REG      0x32
248 #define SSD2119_GAMMA_CTRL_4_REG      0x33
249 #define SSD2119_GAMMA_CTRL_5_REG      0x34
250 #define SSD2119_GAMMA_CTRL_6_REG      0x35
251 #define SSD2119_GAMMA_CTRL_7_REG      0x36
252 #define SSD2119_GAMMA_CTRL_8_REG      0x37
253 #define SSD2119_GAMMA_CTRL_9_REG      0x3A
254 #define SSD2119_GAMMA_CTRL_10_REG     0x3B
```

```
255 #define SSD2119_V_RAM_POS_REG          0x44
256 #define SSD2119_H_RAM_START_REG        0x45
257 #define SSD2119_H_RAM_END_REG          0x46
258 #define SSD2119_X_RAM_ADDR_REG         0x4E
259 #define SSD2119_Y_RAM_ADDR_REG         0x4F
260
261 #define ENTRY_MODE_DEFAULT 0x6830
262 #define MAKE_ENTRY_MODE(x) ((ENTRY_MODE_DEFAULT & 0xFF00) | (x))
263
264 //*****************************************************************************
265 //
266 // The dimensions of the LCD panel.
267 //
268 //*****************************************************************************
269 #define LCD_VERTICAL_MAX 240
270 #define LCD_HORIZONTAL_MAX 320
271
272 //*****************************************************************************
273 //
274 // Translates a 24-bit RGB color to a display driver-specific color.
275 //
276 // \param c is the 24-bit RGB color.  The least-significant byte is the blue
277 // channel, the next byte is the green channel, and the third byte is the red
278 // channel.
279 //
280 // This macro translates a 24-bit RGB color into a value that can be written
281 // into the display's frame buffer in order to reproduce that color, or the
282 // closest possible approximation of that color.
283 //
284 // \return Returns the display-driver specific color.
285 //
286 //*****************************************************************************
287 #define DPYCOLORTRANSLATE(c)    ((((c) & 0x00f80000) >> 8) |               \
288         (((c) & 0x0000fc00) >> 5) |                \
289         (((c) & 0x000000f8) >> 3))
290
291 //*****************************************************************************
292 //
293 // Function pointer types for low level LCD controller access functions.
294 //
295 //*****************************************************************************
296 typedef void (*pfnWriteData)(uint16_t ui16Data);
297 typedef void (*pfnWriteCommand)(uint8_t ui8Data);
298
299 //*****************************************************************************
300 //
301 // Function pointers for low level LCD controller access functions.
302 //
303 //*****************************************************************************
304
305 static void WriteDataGPIO(uint16_t ui16Data);
306 static void WriteCommandGPIO(uint8_t ui8Data);
307
308 pfnWriteData WriteData = WriteDataGPIO;
309 pfnWriteCommand WriteCommand = WriteCommandGPIO;
310
311 void LED_OFF(void)
312 {
313     HWREG(LCD_BACKLIGHT_BASE + GPIO_O_DATA + (LCD_BACKLIGHT_PIN << 2)) = 0;
314     //HWREG(USER_LED_BASE + GPIO_O_DATA + (USER_LED_1_PIN << 2)) =
315     //     USER_LED_1_PIN;
316 }
317
318 void LED_ON(void)
319 {
320     HWREG(LCD_BACKLIGHT_BASE + GPIO_O_DATA + (LCD_BACKLIGHT_PIN << 2)) =
321         LCD_BACKLIGHT_PIN;
322     //HWREG(USER_LED_BASE + GPIO_O_DATA + (USER_LED_1_PIN << 2)) = 0;
323 }
324
325 //*****************************************************************************
326 //
327 // Writes a data word to the SSD2119.  This function implements the basic GPIO
328 // interface to the LCD display.
329 //
330 //*****************************************************************************
331 static void
332 WriteDataGPIO(uint16_t ui16Data)
333 {
334     //
335     // Write the most significant byte of the data to the bus.
336     //
337     SET_LCD_DATA(ui16Data >> 8);
338     //SysCtlDelay(LCD_WRITE_DELAY);
339
```

```
340         //
341         // Pull CS Low.
342         //
343         HWREG(LCD_CS_PIN_REG) = 0;
344         //SysCtlDelay(LCD_WRITE_DELAY);
345
346         //
347         // Assert the write enable signal.
348         // Delay for at least 60nS (at 120 Mhz) to meet Display timing.
349         //
350         HWREG(LCD_WR_PIN_REG) = 0;
351         //SysCtlDelay(LCD_WRITE_DELAY);
352
353         //
354         // Deassert the write enable signal.
355         //
356         HWREG(LCD_WR_PIN_REG) = 0xFF;
357         //SysCtlDelay(LCD_WRITE_DELAY);
358
359         //
360         // Write the least significant byte of the data to the bus.
361         //
362         SET_LCD_DATA(ui16Data);
363         //SysCtlDelay(LCD_WRITE_DELAY);
364
365         //
366         // Assert the write enable signal.
367         // Delay for at least 60nS (at 120 Mhz) to meet Display timing.
368         //
369         HWREG(LCD_WR_PIN_REG) = 0;
370         //SysCtlDelay(LCD_WRITE_DELAY);
371
372         //
373         // Deassert the write enable signal.
374         //
375         HWREG(LCD_WR_PIN_REG) = 0xFF;
376         //SysCtlDelay(LCD_WRITE_DELAY);
377
378         //
379         // Deassert the chip select pin.
380         //
381         HWREG(LCD_CS_PIN_REG) = 0xFF;
382         //SysCtlDelay(LCD_WRITE_DELAY);
383     }
384
385     //*****************************************************************************
386     //
387     // Writes a command to the SSD2119.  This function implements the basic GPIO
388     // interface to the LCD display.
389     //
390     //*****************************************************************************
391     static void
392     WriteCommandGPIO(uint8_t ui8Data)
393     {
394         //
395         // Write the most significant byte of the data to the bus. This is always
396         // 0 since commands are no more than 8 bits currently.
397         //
398         SET_LCD_DATA(0);
399         //SysCtlDelay(LCD_WRITE_DELAY);
400
401         //
402         // Pull CS Low
403         //
404         HWREG(LCD_CS_PIN_REG) = 0;
405         //SysCtlDelay(LCD_WRITE_DELAY);
406
407         //
408         // Assert the write enable and DC signals.  Do this 3 times to slow things
409         // down a bit.
410         //
411         HWREG(LCD_DC_PIN_REG) = 0;
412         HWREG(LCD_WR_PIN_REG) = 0;
413         //SysCtlDelay(LCD_WRITE_DELAY);
414
415         //
416         // Deassert the write enable and DC signals. We need to leave WR high for
417         // at least 50nS so, again, stick in dummy writes to pad the timing.
418         //
419         HWREG(LCD_DC_PIN_REG) = 0xFF;
420         HWREG(LCD_WR_PIN_REG) = 0xFF;
421         //SysCtlDelay(LCD_WRITE_DELAY);
422
423         //
424         // Write the least significant byte of the data to the bus.
```

```
425         //
426         SET_LCD_DATA(ui8Data);
427         //SysCtlDelay(LCD_WRITE_DELAY);
428
429         //
430         // Assert the write enable signal. With delay to meet timing requirements.
431         //
432         HWREG(LCD_DC_PIN_REG) = 0;
433         HWREG(LCD_WR_PIN_REG) = 0;
434         //SysCtlDelay(LCD_WRITE_DELAY);
435
436
437         //
438         // Deassert the write enable and DC signals.  Make sure we add padding here
439         // too since some compilers inline this function.
440         //
441         HWREG(LCD_DC_PIN_REG) = 0xFF;
442         HWREG(LCD_WR_PIN_REG) = 0xFF;
443         //SysCtlDelay(LCD_WRITE_DELAY);
444
445         HWREG(LCD_CS_PIN_REG) = 0xFF;
446         //SysCtlDelay(LCD_WRITE_DELAY);
447     }
448
449     //*****************************************************************************
450     //
451     // Initializes the pins required for the GPIO-based LCD interface.
452     //
453     // This function configures the GPIO pins used to control the LCD display
454     // when the basic GPIO interface is in use.  On exit, the LCD controller
455     // has been reset and is ready to receive command and data writes.
456     //
457     // \return None.
458     //
459     //*****************************************************************************
460     static void
461     InitGPIOLCDInterface(uint32_t ui32ClockMS)
462     {
463         //
464         // Configure the pins that connect to the LCD as GPIO outputs.
465         //
466         SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOD);
467         SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOE);
468         SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOF);
469         SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOM);
470         SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOC);
471         SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOB);
472         SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPION);
473
474         GPIOPinTypeGPIOOutput(LCD_DATAH_BASE_E, LCD_DATAH_PIN_5);
475         GPIOPinTypeGPIOOutput(LCD_DATAH_BASE_D, LCD_DATAH_PIN_4 |
476                 LCD_DATAH_PIN_7 | LCD_DATAH_PIN_6);
477         GPIOPinTypeGPIOOutput(LCD_DATAH_BASE_F, LCD_DATAH_PIN_3);
478         GPIOPinTypeGPIOOutput(LCD_DATAH_BASE_M, LCD_DATAH_PIN_2);
479         GPIOPinTypeGPIOOutput(LCD_DATAH_BASE_C, LCD_DATAH_PIN_1 | LCD_DATAH_PIN_0);
480
481         GPIOPadConfigSet(LCD_DATAH_BASE_E, LCD_DATAH_PIN_5, GPIO_STRENGTH_12MA,
482                 GPIO_PIN_TYPE_STD);
483         GPIOPadConfigSet(LCD_DATAH_BASE_D, LCD_DATAH_PIN_7 | LCD_DATAH_PIN_6 |
484                 LCD_DATAH_PIN_4, GPIO_STRENGTH_12MA, GPIO_PIN_TYPE_STD);
485         GPIOPadConfigSet(LCD_DATAH_BASE_F, LCD_DATAH_PIN_3, GPIO_STRENGTH_12MA,
486                 GPIO_PIN_TYPE_STD);
487         GPIOPadConfigSet(LCD_DATAH_BASE_M, LCD_DATAH_PIN_2, GPIO_STRENGTH_12MA,
488                 GPIO_PIN_TYPE_STD);
489         GPIOPadConfigSet(LCD_DATAH_BASE_C, LCD_DATAH_PIN_1 | LCD_DATAH_PIN_0,
490                 GPIO_STRENGTH_12MA, GPIO_PIN_TYPE_STD);
491
492         GPIOPinTypeGPIOOutput(LCD_DC_BASE, LCD_DC_PIN);
493         GPIOPinTypeGPIOOutput(LCD_RD_BASE, LCD_RD_PIN);
494         GPIOPinTypeGPIOOutput(LCD_WR_BASE, LCD_WR_PIN);
495         GPIOPinTypeGPIOOutput(LCD_CS_BASE, LCD_CS_PIN);
496         GPIOPinTypeGPIOOutput(LCD_BACKLIGHT_BASE, LCD_BACKLIGHT_PIN);
497
498         //
499         // Set the LCD control pins to their default values.
500         //
501         GPIOPinWrite(LCD_CS_BASE, LCD_CS_PIN, LCD_CS_PIN);
502         GPIOPinWrite(LCD_DATAH_BASE_E, LCD_DATAH_PIN_5,  0x00);
503         GPIOPinWrite(LCD_DATAH_BASE_D, LCD_DATAH_PIN_7 | LCD_DATAH_PIN_6 |
504                 LCD_DATAH_PIN_4,  0x00);
505         GPIOPinWrite(LCD_DATAH_BASE_F, LCD_DATAH_PIN_3,  0x00);
506         GPIOPinWrite(LCD_DATAH_BASE_M, LCD_DATAH_PIN_2,  0x00);
507         GPIOPinWrite(LCD_DATAH_BASE_C, LCD_DATAH_PIN_1 | LCD_DATAH_PIN_0, 0x00);
508
509         GPIOPinWrite(LCD_DC_BASE, LCD_DC_PIN, 0x00);
```

```
510         GPIOPinWrite(LCD_RD_BASE, LCD_RD_PIN, LCD_RD_PIN);
511         GPIOPinWrite(LCD_WR_BASE, LCD_WR_PIN, LCD_WR_PIN);
512         GPIOPinWrite(LCD_CS_BASE, LCD_CS_PIN, 0);
513
514         //
515         // Delay for 1ms.
516         //
517         SysCtlDelay(ui32ClockMS);
518
519         //
520         // Deassert the LCD reset signal.
521         //
522         //GPIOPinWrite(LCD_RST_BASE, LCD_RST_PIN, LCD_RST_PIN);
523
524         //
525         // Delay for 1ms while the LCD comes out of reset.
526         //
527         SysCtlDelay(ui32ClockMS);
528     }
529
530     //*****************************************************************************
531     //
532     //! Initializes the display driver.
533     //!
534     //! \param ui32SysClockSpeed is the system clock speed of the MCU.
535     //!
536     //! This function initializes the SSD2119 display controller on the panel,
537     //! preparing it to display data.
538     //!
539     //! \return None.
540     //
541     //*****************************************************************************
542     void
543     Kentec320x240x16_SSD2119Init(uint32_t ui32SysClockSpeed)
544     {
545         uint32_t ui32ClockMS, ui32Count;
546
547         //
548         // Get the current processor clock frequency.
549         //
550         ui32ClockMS = ui32SysClockSpeed / (3 * 1000);
551
552         //
553         // Enable the GPIO peripherals used to interface to the SSD2119.
554         //
555         SysCtlPeripheralEnable(LCD_DATAH_PERIPH_E);
556         SysCtlPeripheralEnable(LCD_DATAH_PERIPH_D);
557         SysCtlPeripheralEnable(LCD_DATAH_PERIPH_F);
558         SysCtlPeripheralEnable(LCD_DATAH_PERIPH_M);
559         SysCtlPeripheralEnable(LCD_DATAH_PERIPH_C);
560         SysCtlPeripheralEnable(LCD_DC_PERIPH);
561         SysCtlPeripheralEnable(LCD_RD_PERIPH);
562         SysCtlPeripheralEnable(LCD_WR_PERIPH);
563         SysCtlPeripheralEnable(LCD_CS_PERIPH);
564         SysCtlPeripheralEnable(LCD_BACKLIGHT_PERIPH);
565
566         //
567         // Perform low level interface initialization depending upon how the LCD
568         // is connected to the Tiva C Series microcontroller.  This varies
569         // depending upon the daughter board connected it is possible that a
570         // daughter board can drive the LCD directly rather than via the basic GPIO
571         // interface.
572         //
573         {
574             //
575             // Initialize the GPIOs used to interface to the LCD controller.
576             //
577             InitGPIOLCDInterface(ui32ClockMS);
578         }
579
580         //
581         // Enter sleep mode (if we are not already there).
582         //
583         WriteCommand(SSD2119_SLEEP_MODE_REG);
584         WriteData(0x0001);
585
586         //
587         // Set initial power parameters.
588         //
589         WriteCommand(SSD2119_PWR_CTRL_5_REG);
590         WriteData(0x00BA);
591         WriteCommand(SSD2119_VCOM_OTP_1_REG);
592         WriteData(0x0006);
593
594         //
```

```
595     // Start the oscillator.
596     //
597     WriteCommand(SSD2119_OSC_START_REG);
598     WriteData(0x0001);
599
600     //
601     // Set pixel format and basic display orientation (scanning direction).
602     //
603     WriteCommand(SSD2119_OUTPUT_CTRL_REG);
604     WriteData(0x30EF);
605     WriteCommand(SSD2119_LCD_DRIVE_AC_CTRL_REG);
606     WriteData(0x0600);
607
608     //
609     // Exit sleep mode.
610     //
611     WriteCommand(SSD2119_SLEEP_MODE_REG);
612     WriteData(0x0000);
613
614     //
615     // Delay 30mS
616     //
617     SysCtlDelay(30 * ui32ClockMS);
618
619     //
620     // Configure pixel color format and MCU interface parameters.
621     //
622     WriteCommand(SSD2119_ENTRY_MODE_REG);
623     WriteData(ENTRY_MODE_DEFAULT);
624
625     //
626     // Enable the display.
627     //
628     WriteCommand(SSD2119_DISPLAY_CTRL_REG);
629     WriteData(0x0033);
630
631     //
632     // Set VCIX2 voltage to 6.1V.
633     //
634     WriteCommand(SSD2119_PWR_CTRL_2_REG);
635     WriteData(0x0005);
636
637     //
638     // Configure gamma correction.
639     //
640     WriteCommand(SSD2119_GAMMA_CTRL_1_REG);
641     WriteData(0x0000);
642     WriteCommand(SSD2119_GAMMA_CTRL_2_REG);
643     WriteData(0x0400);
644     WriteCommand(SSD2119_GAMMA_CTRL_3_REG);
645     WriteData(0x0106);
646     WriteCommand(SSD2119_GAMMA_CTRL_4_REG);
647     WriteData(0x0700);
648     WriteCommand(SSD2119_GAMMA_CTRL_5_REG);
649     WriteData(0x0002);
650     WriteCommand(SSD2119_GAMMA_CTRL_6_REG);
651     WriteData(0x0702);
652     WriteCommand(SSD2119_GAMMA_CTRL_7_REG);
653     WriteData(0x0707);
654     WriteCommand(SSD2119_GAMMA_CTRL_8_REG);
655     WriteData(0x0203);
656     WriteCommand(SSD2119_GAMMA_CTRL_9_REG);
657     WriteData(0x1400);
658     WriteCommand(SSD2119_GAMMA_CTRL_10_REG);
659     WriteData(0x0F03);
660
661     //
662     // Configure Vlcd63 and VCOMl.
663     //
664     WriteCommand(SSD2119_PWR_CTRL_3_REG);
665     WriteData(0x0007);
666     WriteCommand(SSD2119_PWR_CTRL_4_REG);
667     WriteData(0x3100);
668
669     //
670     // Set the display size and ensure that the GRAM window is set to allow
671     // access to the full display buffer.
672     //
673     WriteCommand(SSD2119_V_RAM_POS_REG);
674     WriteData((LCD_VERTICAL_MAX-1) << 8);
675     WriteCommand(SSD2119_H_RAM_START_REG);
676     WriteData(0x0000);
677     WriteCommand(SSD2119_H_RAM_END_REG);
678     WriteData(LCD_HORIZONTAL_MAX-1);
679     WriteCommand(SSD2119_X_RAM_ADDR_REG);
```

```
680         WriteData(0x00);
681         WriteCommand(SSD2119_Y_RAM_ADDR_REG);
682         WriteData(0x00);
683 
684         //
685         // Clear the contents of the display buffer.
686         //
687         WriteCommand(SSD2119_RAM_DATA_REG);
688         for(ui32Count = 0; ui32Count < (320 * 240); ui32Count++)
689         {
690             WriteData(0x0000);
691         }
692     }
693 
694     //****************************************************************************
695     //
696     //! Draws a pixel on the screen.
697     //!
698     //! \param pvDisplayData is a pointer to the driver-specific data for this
699     //! display driver.
700     //! \param i32X is the X coordinate of the pixel.
701     //! \param i32Y is the Y coordinate of the pixel.
702     //! \param ui32Value is the color of the pixel.
703     //!
704     //! This function sets the given pixel to a particular color.  The coordinates
705     //! of the pixel are assumed to be within the extents of the display.
706     //!
707     //! \return None.
708     //
709     //****************************************************************************
710     static void
711     Kentec320x240x16_SSD2119PixelDraw(void *pvDisplayData, int32_t i32X,
712             int32_t i32Y,
713             uint32_t ui32Value)
714     {
715         //
716         // Set the X address of the display cursor.
717         //
718         WriteCommand(SSD2119_X_RAM_ADDR_REG);
719         WriteData(MAPPED_X(i32X, i32Y));
720 
721         //
722         // Set the Y address of the display cursor.
723         //
724         WriteCommand(SSD2119_Y_RAM_ADDR_REG);
725         WriteData(MAPPED_Y(i32X, i32Y));
726 
727         //
728         // Write the pixel value.
729         //
730         WriteCommand(SSD2119_RAM_DATA_REG);
731         WriteData(ui32Value);
732     }
733 
734     //****************************************************************************
735     //
736     //! Draws a horizontal sequence of pixels on the screen.
737     //!
738     //! \param pvDisplayData is a pointer to the driver-specific data for this
739     //! display driver.
740     //! \param i32X is the X coordinate of the first pixel.
741     //! \param i32Y is the Y coordinate of the first pixel.
742     //! \param i32X0 is sub-pixel offset within the pixel data, which is valid for
743     //! 1 or 4 bit per pixel formats.
744     //! \param i32Count is the number of pixels to draw.
745     //! \param i32BPP is the number of bits per pixel; must be 1, 4, or 8,
746     //! optionally OR'ed with flags that a driver may use to aid performance.
747     //! \param pui8Data is a pointer to the pixel data.  For 1 and 4 bit per pixel
748     //! formats, the most significant bit(s) represent the left-most pixel.
749     //! \param pui8Palette is a pointer to the palette used to draw the pixels.
750     //!
751     //! This function draws a horizontal sequence of pixels on the screen, using
752     //! the supplied palette.  For 1 bit per pixel format, the palette contains
753     //! pre-translated colors; for 4 and 8 bit per pixel formats, the palette
754     //! contains 24-bit RGB values that must be translated before being written to
755     //! the display.
756     //!
757     //! \return None.
758     //
759     //****************************************************************************
760     static void
761     Kentec320x240x16_SSD2119PixelDrawMultiple(void *pvDisplayData, int32_t i32X,
762             int32_t i32Y, int32_t i32X0,
763             int32_t i32Count,
764             int32_t i32BPP,
```

```
765         const uint8_t *pui8Data,
766         const uint8_t *pui8Palette)
767 {
768     uint32_t ui32Byte;
769 
770     //
771     // Set the cursor increment to left to right, followed by top to bottom.
772     //
773     WriteCommand(SSD2119_ENTRY_MODE_REG);
774     WriteData(MAKE_ENTRY_MODE(HORIZ_DIRECTION));
775 
776     //
777     // Set the starting X address of the display cursor.
778     //
779     WriteCommand(SSD2119_X_RAM_ADDR_REG);
780     WriteData(MAPPED_X(i32X, i32Y));
781 
782     //
783     // Set the Y address of the display cursor.
784     //
785     WriteCommand(SSD2119_Y_RAM_ADDR_REG);
786     WriteData(MAPPED_Y(i32X, i32Y));
787 
788     //
789     // Write the data RAM write command.
790     //
791     WriteCommand(SSD2119_RAM_DATA_REG);
792 
793     //
794     // Determine how to interpret the pixel data based on the number of bits
795     // per pixel.
796     //
797     switch(i32BPP & 0xFF)
798     {
799         //
800         // The pixel data is in 1 bit per pixel format.
801         //
802         case 1:
803             {
804                 //
805                 // Loop while there are more pixels to draw.
806                 //
807                 while(i32Count)
808                 {
809                     //
810                     // Get the next byte of image data.
811                     //
812                     ui32Byte = *pui8Data++;
813 
814                     //
815                     // Loop through the pixels in this byte of image data.
816                     //
817                     for(; (i32X0 < 8) && i32Count; i32X0++, i32Count--)
818                     {
819                         //
820                         // Draw this pixel in the appropriate color.
821                         //
822                         WriteData(((uint32_t *)pui8Palette)[(ui32Byte >>
823                                    (7 - i32X0)) & 1]);
824                     }
825 
826                     //
827                     // Start at the beginning of the next byte of image data.
828                     //
829                     i32X0 = 0;
830                 }
831 
832                 //
833                 // The image data has been drawn.
834                 //
835                 break;
836             }
837 
838             //
839             // The pixel data is in 4 bit per pixel format.
840             //
841         case 4:
842             {
843                 //
844                 // Loop while there are more pixels to draw.  "Duff's device"
845                 // is used to jump into the middle of the loop if the first
846                 // nibble of the pixel data should not be used.  Duff's device
847                 // makes use of the fact that a case statement is legal
848                 // anywhere within a sub-block of a switch statement.  See
849                 // http://en.wikipedia.org/wiki/Duff's_device for detailed
```

```
850                 // information about Duff's device.
851                 //
852                 switch(i32X0 & 1)
853                 {
854                     case 0:
855                         while(i32Count)
856                         {
857                             //
858                             // Get the upper nibble of the next byte of pixel
859                             // data and extract the corresponding entry from
860                             // the palette.
861                             //
862                             ui32Byte = (*pui8Data >> 4) * 3;
863                             ui32Byte = (*(uint32_t *)(pui8Palette + ui32Byte) &
864                                     0x00ffffff);
865
866                             //
867                             // Translate this palette entry and write it to the
868                             // screen.
869                             //
870                             WriteData(DPYCOLORTRANSLATE(ui32Byte));
871
872                             //
873                             // Decrement the count of pixels to draw.
874                             //
875                             i32Count--;
876
877                             //
878                             // See if there is another pixel to draw.
879                             //
880                             if(i32Count)
881                             {
882                                 case 1:
883                                     //
884                                     // Get the lower nibble of the next byte of
885                                     // pixel data and extract the corresponding
886                                     // entry from the palette.
887                                     //
888                                     ui32Byte = (*pui8Data++ & 15) * 3;
889                                     ui32Byte = (*(uint32_t *)(pui8Palette +
890                                             ui32Byte) & 0x00ffffff);
891
892                                     //
893                                     // Translate this palette entry and write
894                                     // it to the screen.
895                                     //
896                                     WriteData(DPYCOLORTRANSLATE(ui32Byte));
897
898                                     //
899                                     // Decrement the count of pixels to draw.
900                                     //
901                                     i32Count--;
902                             }
903                         }
904                 }
905
906                 //
907                 // The image data has been drawn.
908                 //
909                 break;
910             }
911
912             //
913             // The pixel data is in 8 bit per pixel format.
914             //
915         case 8:
916             {
917                 //
918                 // Loop while there are more pixels to draw.
919                 //
920                 while(i32Count--)
921                 {
922                     //
923                     // Get the next byte of pixel data and extract the
924                     // corresponding entry from the palette.
925                     //
926                     ui32Byte = *pui8Data++ * 3;
927                     ui32Byte = *(uint32_t *)(pui8Palette + ui32Byte) &
928                         0x00ffffff;
929
930                     //
931                     // Translate this palette entry and write it to the screen.
932                     //
933                     WriteData(DPYCOLORTRANSLATE(ui32Byte));
934                 }
```

```
935
936                 //
937                 // The image data has been drawn.
938                 //
939                 break;
940             }
941
942             //
943             // We are being passed data in the display's native format.  Merely
944             // write it directly to the display.  This is a special case which
945             // is not used by the graphics library but which is helpful to
946             // applications which may want to handle, for example, JPEG images.
947             //
948         case 16:
949             {
950                 uint16_t ui16Byte;
951
952                 //
953                 // Loop while there are more pixels to draw.
954                 //
955                 while(i32Count--)
956                 {
957                     //
958                     // Get the next byte of pixel data and extract the
959                     // corresponding entry from the palette.
960                     //
961                     ui16Byte = *((uint16_t *)pui8Data);
962                     pui8Data += 2;
963
964                     //
965                     // Translate this palette entry and write it to the screen.
966                     //
967                     WriteData(ui16Byte);
968                 }
969             }
970     }
971 }
972
973 //*****************************************************************************
974 //
975 //! Draws a horizontal line.
976 //!
977 //! \param pvDisplayData is a pointer to the driver-specific data for this
978 //! display driver.
979 //! \param i32X1 is the X coordinate of the start of the line.
980 //! \param i32X2 is the X coordinate of the end of the line.
981 //! \param i32Y is the Y coordinate of the line.
982 //! \param ui32Value is the color of the line.
983 //!
984 //! This function draws a horizontal line on the display.  The coordinates of
985 //! the line are assumed to be within the extents of the display.
986 //!
987 //! \return None.
988 //
989 //*****************************************************************************
990 static void
991 Kentec320x240x16_SSD2119LineDrawH(void *pvDisplayData, int32_t i32X1,
992         int32_t i32X2, int32_t i32Y,
993         uint32_t ui32Value)
994 {
995     //
996     // Set the cursor increment to left to right, followed by top to bottom.
997     //
998     WriteCommand(SSD2119_ENTRY_MODE_REG);
999     WriteData(MAKE_ENTRY_MODE(HORIZ_DIRECTION));
1000
1001     //
1002     // Set the starting X address of the display cursor.
1003     //
1004     WriteCommand(SSD2119_X_RAM_ADDR_REG);
1005     WriteData(MAPPED_X(i32X1, i32Y));
1006
1007     //
1008     // Set the Y address of the display cursor.
1009     //
1010     WriteCommand(SSD2119_Y_RAM_ADDR_REG);
1011     WriteData(MAPPED_Y(i32X1, i32Y));
1012
1013     //
1014     // Write the data RAM write command.
1015     //
1016     WriteCommand(SSD2119_RAM_DATA_REG);
1017
1018     //
1019     // Loop through the pixels of this horizontal line.
```

```
1020        //
1021        while(i32X1++ <= i32X2)
1022        {
1023            //
1024            // Write the pixel value.
1025            //
1026            WriteData(ui32Value);
1027        }
1028    }
1029
1030    //*****************************************************************************
1031    //
1032    //! Draws a vertical line.
1033    //!
1034    //! \param pvDisplayData is a pointer to the driver-specific data for this
1035    //! display driver.
1036    //! \param i32X is the X coordinate of the line.
1037    //! \param i32Y1 is the Y coordinate of the start of the line.
1038    //! \param i32Y2 is the Y coordinate of the end of the line.
1039    //! \param ui32Value is the color of the line.
1040    //!
1041    //! This function draws a vertical line on the display.  The coordinates of the
1042    //! line are assumed to be within the extents of the display.
1043    //!
1044    //! \return None.
1045    //
1046    //*****************************************************************************
1047    static void
1048    Kentec320x240x16_SSD2119LineDrawV(void *pvDisplayData, int32_t i32X,
1049            int32_t i32Y1,
1050            int32_t i32Y2, uint32_t ui32Value)
1051    {
1052        //
1053        // Set the cursor increment to top to bottom, followed by left to right.
1054        //
1055        WriteCommand(SSD2119_ENTRY_MODE_REG);
1056        WriteData(MAKE_ENTRY_MODE(VERT_DIRECTION));
1057
1058        //
1059        // Set the X address of the display cursor.
1060        //
1061        WriteCommand(SSD2119_X_RAM_ADDR_REG);
1062        WriteData(MAPPED_X(i32X, i32Y1));
1063
1064        //
1065        // Set the starting Y address of the display cursor.
1066        //
1067        WriteCommand(SSD2119_Y_RAM_ADDR_REG);
1068        WriteData(MAPPED_Y(i32X, i32Y1));
1069
1070        //
1071        // Write the data RAM write command.
1072        //
1073        WriteCommand(SSD2119_RAM_DATA_REG);
1074
1075        //
1076        // Loop through the pixels of this vertical line.
1077        //
1078        while(i32Y1++ <= i32Y2)
1079        {
1080            //
1081            // Write the pixel value.
1082            //
1083            WriteData(ui32Value);
1084        }
1085    }
1086
1087    //*****************************************************************************
1088    //
1089    //! Fills a rectangle.
1090    //!
1091    //! \param pvDisplayData is a pointer to the driver-specific data for this
1092    //! display driver.
1093    //! \param pRect is a pointer to the structure describing the rectangle.
1094    //! \param ui32Value is the color of the rectangle.
1095    //!
1096    //! This function fills a rectangle on the display.  The coordinates of the
1097    //! rectangle are assumed to be within the extents of the display, and the
1098    //! rectangle specification is fully inclusive (in other words, both i16XMin
1099    //! and i16XMax are drawn, along with i16YMin and i16YMax).
1100    //!
1101    //! \return None.
1102    //
1103    //*****************************************************************************
1104    static void
```

```
1105 Kentec320x240x16_SSD2119RectFill(void *pvDisplayData, const tRectangle *pRect,
1106         uint32_t ui32Value)
1107 {
1108     int32_t i32Count;
1109 
1110     //
1111     // Write the Y extents of the rectangle.
1112     //
1113     WriteCommand(SSD2119_ENTRY_MODE_REG);
1114     WriteData(MAKE_ENTRY_MODE(HORIZ_DIRECTION));
1115 
1116     //
1117     // Write the X extents of the rectangle.
1118     //
1119     WriteCommand(SSD2119_H_RAM_START_REG);
1120 #if (defined PORTRAIT) || (defined LANDSCAPE)
1121     WriteData(MAPPED_X(pRect->i16XMax, pRect->i16YMax));
1122 #else
1123     WriteData(MAPPED_X(pRect->i16XMin, pRect->sMin));
1124 #endif
1125 
1126     WriteCommand(SSD2119_H_RAM_END_REG);
1127 #if (defined PORTRAIT) || (defined LANDSCAPE)
1128     WriteData(MAPPED_X(pRect->i16XMin, pRect->i16YMin));
1129 #else
1130     WriteData(MAPPED_X(pRect->i16XMax, pRect->i16YMax));
1131 #endif
1132 
1133     //
1134     // Write the Y extents of the rectangle
1135     //
1136     WriteCommand(SSD2119_V_RAM_POS_REG);
1137 #if (defined LANDSCAPE_FLIP) || (defined PORTRAIT)
1138     WriteData(MAPPED_Y(pRect->i16XMin, pRect->i16YMin) |
1139               (MAPPED_Y(pRect->i16XMax, pRect->i16YMax) << 8));
1140 #else
1141     WriteData(MAPPED_Y(pRect->i16XMax, pRect->i16YMax) |
1142               (MAPPED_Y(pRect->i16XMin, pRect->i16YMin) << 8));
1143 #endif
1144 
1145     //
1146     // Set the display cursor to the upper left of the rectangle (in
1147     // application coordinate space).
1148     //
1149     WriteCommand(SSD2119_X_RAM_ADDR_REG);
1150     WriteData(MAPPED_X(pRect->i16XMin, pRect->i16YMin));
1151 
1152     WriteCommand(SSD2119_Y_RAM_ADDR_REG);
1153     WriteData(MAPPED_Y(pRect->i16XMin, pRect->i16YMin));
1154 
1155     //
1156     // Tell the controller we are about to write data into its RAM.
1157     //
1158     WriteCommand(SSD2119_RAM_DATA_REG);
1159 
1160     //
1161     // Loop through the pixels of this filled rectangle.
1162     //
1163     for(i32Count = ((pRect->i16XMax - pRect->i16XMin + 1) *
1164                     (pRect->i16YMax - pRect->i16YMin + 1));
1165             i32Count >= 0; i32Count--)
1166     {
1167         //
1168         // Write the pixel value.
1169         //
1170         WriteData(ui32Value);
1171     }
1172 
1173     //
1174     // Reset the X extents to the entire screen.
1175     //
1176     WriteCommand(SSD2119_H_RAM_START_REG);
1177     WriteData(0x0000);
1178     WriteCommand(SSD2119_H_RAM_END_REG);
1179     WriteData(0x013F);
1180 
1181     //
1182     // Reset the Y extent to the full screen
1183     //
1184     WriteCommand(SSD2119_V_RAM_POS_REG);
1185     WriteData(0xEF00);
1186 }
1187 
1188 //*****************************************************************************
1189 //
```

```
1190 //! Translates a 24-bit RGB color to a display driver-specific color.
1191 //!
1192 //! \param pvDisplayData is a pointer to the driver-specific data for this
1193 //! display driver.
1194 //! \param ui32Value is the 24-bit RGB color.  The least-significant byte is
1195 //! the blue channel, the next byte is the green channel, and the third byte is
1196 //! the red channel.
1197 //!
1198 //! This function translates a 24-bit RGB color into a value that can be
1199 //! written into the display's frame buffer in order to reproduce that color,
1200 //! or the closest possible approximation of that color.
1201 //!
1202 //! \return Returns the display-driver specific color.
1203 //
1204 //*****************************************************************************
1205 static uint32_t
1206 Kentec320x240x16_SSD2119ColorTranslate(void *pvDisplayData,
1207         uint32_t ui32Value)
1208 {
1209     //
1210     // Translate from a 24-bit RGB color to a 5-6-5 RGB color.
1211     //
1212     return(DPYCOLORTRANSLATE(ui32Value));
1213 }
1214
1215 //*****************************************************************************
1216 //
1217 //! Flushes any cached drawing operations.
1218 //!
1219 //! \param pvDisplayData is a pointer to the driver-specific data for this
1220 //! display driver.
1221 //!
1222 //! This functions flushes any cached drawing operations to the display.  This
1223 //! is useful when a local frame buffer is used for drawing operations, and the
1224 //! flush would copy the local frame buffer to the display.  For the SSD2119
1225 //! driver, the flush is a no operation.
1226 //!
1227 //! \return None.
1228 //
1229 //*****************************************************************************
1230 static void
1231 Kentec320x240x16_SSD2119Flush(void *pvDisplayData)
1232 {
1233     //
1234     // There is nothing to be done.
1235     //
1236 }
1237
1238 //*****************************************************************************
1239 //
1240 //! The display structure that describes the driver for the Kentec
1241 //! K350QVG-V2-F TFT panel with an SSD2119 controller.
1242 //
1243 //*****************************************************************************
1244 const tDisplay g_sKentec320x240x16_SSD2119 =
1245 {
1246     sizeof(tDisplay),
1247     0,
1248 #if defined(PORTRAIT) || defined(PORTRAIT_FLIP)
1249     240,
1250     320,
1251 #else
1252     320,
1253     240,
1254 #endif
1255     Kentec320x240x16_SSD2119PixelDraw,
1256     Kentec320x240x16_SSD2119PixelDrawMultiple,
1257     Kentec320x240x16_SSD2119LineDrawH,
1258     Kentec320x240x16_SSD2119LineDrawV,
1259     Kentec320x240x16_SSD2119RectFill,
1260     Kentec320x240x16_SSD2119ColorTranslate,
1261     Kentec320x240x16_SSD2119Flush
1262 };
1263
1264 //*****************************************************************************
1265 //
1266 // Close the Doxygen group.
1267 //! @}
1268 //
1269 //*****************************************************************************
```

Quellcode D.14: Software-Modul: Angepasster Touchscreen-Treiber (Header) [36]

```
1  //*****************************************************************************
2  //
3  // touch.c - Touch screen driver for the EK-TM4C1294XL board.
4  //
5  // Copyright (c) 2013-2014 Texas Instruments Incorporated.  All rights reserved.
6  // Software License Agreement
7  //
8  // Texas Instruments (TI) is supplying this software for use solely and
9  // exclusively on TI's microcontroller products. The software is owned by
10 // TI and/or its suppliers, and is protected under applicable copyright
11 // laws. You may not combine this software with "viral" open-source
12 // software in order to form a larger program.
13 //
14 // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
15 // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
16 // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
17 // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
18 // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
19 // DAMAGES, FOR ANY REASON WHATSOEVER.
20 //
21 // This is part of revision 2.1.0.12573 of the EK-TM4C1294XL Firmware Package.
22 //
23 //*****************************************************************************
24
25 //*****************************************************************************
26 //
27 //! \addtogroup touch_api
28 //! @{
29 //
30 //*****************************************************************************
31
32 #include <stdbool.h>
33 #include <stdint.h>
34 #include "inc/hw_adc.h"
35 #include "inc/hw_gpio.h"
36 #include "inc/hw_ints.h"
37 #include "inc/hw_memmap.h"
38 #include "inc/hw_timer.h"
39 #include "inc/hw_types.h"
40 #include "driverlib/adc.h"
41 #include "driverlib/gpio.h"
42 #include "driverlib/interrupt.h"
43 #include "driverlib/sysctl.h"
44 #include "driverlib/timer.h"
45 #include "grlib/grlib.h"
46 #include "grlib/widget.h"
47 #include "display/touch.h"
48
49 //*****************************************************************************
50 //
51 // This driver operates in four different screen orientations.  They are:
52 //
53 // * Portrait - The screen is taller than it is wide, and the flex connector is
54 //              on the left of the display.  This is selected by defining
55 //              PORTRAIT.
56 //
57 // * Landscape - The screen is wider than it is tall, and the flex connector is
58 //               on the bottom of the display.  This is selected by defining
59 //               LANDSCAPE.
60 //
61 // * Portrait flip - The screen is taller than it is wide, and the flex
62 //                   connector is on the right of the display.  This is
63 //                   selected by defining PORTRAIT_FLIP.
64 //
65 // * Landscape flip - The screen is wider than it is tall, and the flex
66 //                    connector is on the top of the display.  This is
67 //                    selected by defining LANDSCAPE_FLIP.
68 //
69 // These can also be imagined in terms of screen rotation; if portrait mode is
70 // 0 degrees of screen rotation, landscape is 90 degrees of counter-clockwise
71 // rotation, portrait flip is 180 degrees of rotation, and landscape flip is
72 // 270 degress of counter-clockwise rotation.
73 //
74 // If no screen orientation is selected, "landscape" mode will be used.
75 //
76 //*****************************************************************************
77 #if ! defined(PORTRAIT) && ! defined(PORTRAIT_FLIP) && \
78     ! defined(LANDSCAPE) && ! defined(LANDSCAPE_FLIP)
79 #define LANDSCAPE
80 #endif
81
82 //*****************************************************************************
83 //
84 // The GPIO pins and SDC channels to which the touch screen is connected.
```

```
85  //
86  //*****************************************************************************
87  #define TS_P_PERIPH             SYSCTL_PERIPH_GPIOE
88  #define TS_P_BASE               GPIO_PORTE_BASE
89  #define TS_XP_PIN                GPIO_PIN_0//U
90  #define TS_XP_ADC                ADC_CTL_CH3
91  #define TS_YP_PIN                GPIO_PIN_5//R
92  #define TS_YP_ADC                ADC_CTL_CH8
93  #define TS_XN_PERIPH             SYSCTL_PERIPH_GPION
94  #define TS_XN_BASE               GPIO_PORTN_BASE
95  #define TS_XN_PIN                GPIO_PIN_3//L
96  #define TS_YN_PERIPH             SYSCTL_PERIPH_GPIOP
97  #define TS_YN_BASE               GPIO_PORTP_BASE
98  #define TS_YN_PIN                GPIO_PIN_2//D
99
100 //*****************************************************************************
101 //
102 // Touchscreen calibration parameters.  Screen orientation is a build time
103 // selection.
104 //
105 //*****************************************************************************
106 const int32_t g_pi32TouchParameters[7] =
107 {
108 #ifdef PORTRAIT
109     3840,                       // M0
110     318720,                     // M1
111     -297763200,                 // M2
112     328576,                     // M3
113     -8896,                      // M4
114     -164591232,                 // M5
115     3100080,                    // M6
116 #endif
117 #ifdef LANDSCAPE
118     //328192,                     // M0
119     430000,                     // M0
120     //-4352,                      // M1
121     -6400,                      // M1
122     -178717056,                 // M2
123     1488,                       // M3
124     -314592,                    // M4
125     1012670064,                 // M5
126     3055164,                    // M6
127 #endif
128 #ifdef PORTRAIT_FLIP
129     1728,                       // M0
130     -321696,                    // M1
131     1034304336,                 // M2
132     -325440,                    // M3
133     1600,                       // M4
134     1161009600,                 // M5
135     3098070,                    // M6
136 #endif
137 #ifdef LANDSCAPE_FLIP
138     -326400,                    // M0
139     -1024,                      // M1
140     1155718720,                 // M2
141     3768,                       // M3
142     312024,                     // M4
143     -299081088,                 // M5
144     3013754,                    // M6
145 #endif
146 };
147
148 //*****************************************************************************
149 //
150 // A pointer to the current touchscreen calibration parameter set.
151 //
152 //*****************************************************************************
153 const int32_t *g_pi32ParmSet;
154
155 //*****************************************************************************
156 //
157 // The minimum raw reading that should be considered valid press.
158 //
159 //*****************************************************************************
160 int16_t g_i16TouchMin = TOUCH_MIN;
161
162 //*****************************************************************************
163 //
164 // The current state of the touch screen driver's state machine.  This is used
165 // to cycle the touch screen interface through the powering sequence required
166 // to read the two axes of the surface.
167 //
168 //*****************************************************************************
169 static uint32_t g_ui32TSState;
```

```
170 #define TS_STATE_INIT            0
171 #define TS_STATE_READ_X          1
172 #define TS_STATE_READ_Y          2
173 #define TS_STATE_SKIP_X          3
174 #define TS_STATE_SKIP_Y          4
175
176 //*****************************************************************************
177 //
178 // The most recent raw ADC reading for the X position on the screen.  This
179 // value is not affected by the selected screen orientation.
180 //
181 //*****************************************************************************
182 volatile int16_t g_i16TouchX;
183
184 //*****************************************************************************
185 //
186 // The most recent raw ADC reading for the Y position on the screen.  This
187 // value is not affected by the selected screen orientation.
188 //
189 //*****************************************************************************
190 volatile int16_t g_i16TouchY;
191
192 //*****************************************************************************
193 //
194 // A pointer to the function to receive messages from the touch screen driver
195 // when events occur on the touch screen (debounced presses, movement while
196 // pressed, and debounced releases).
197 //
198 //*****************************************************************************
199 static int32_t (*g_pfnTSHandler)(uint32_t ui32Message, int32_t i32X,
200                                  int32_t i32Y);
201
202 //*****************************************************************************
203 //
204 // The current state of the touch screen debouncer.  When zero, the pen is up.
205 // When three, the pen is down.  When one or two, the pen is transitioning from
206 // one state to the other.
207 //
208 //*****************************************************************************
209 static uint8_t g_ui8State = 0;
210
211 //*****************************************************************************
212 //
213 // The queue of debounced pen positions.  This is used to slightly delay the
214 // returned pen positions, so that the pen positions that occur while the pen
215 // is being raised are not send to the application.
216 //
217 //*****************************************************************************
218 static int16_t g_pi16Samples[8];
219
220 //*****************************************************************************
221 //
222 // The count of pen positions in g_pi16Samples.  When negative, the buffer is
223 // being pre-filled as a result of a detected pen down event.
224 //
225 //*****************************************************************************
226 static int8_t g_i8Index = 0;
227
228 //*****************************************************************************
229 //
230 //! Debounces presses of the touch screen.
231 //!
232 //! This function is called when a new X/Y sample pair has been captured in
233 //! order to perform debouncing of the touch screen.
234 //!
235 //! \return None.
236 //
237 //*****************************************************************************
238 static void
239 TouchScreenDebouncer(void)
240 {
241     int32_t i32X, i32Y, i32Temp;
242
243     //
244     // Convert the ADC readings into pixel values on the screen.
245     //
246     i32X = g_i16TouchX;
247     i32Y = g_i16TouchY;
248     i32Temp = (((i32X * g_pi32TouchParameters[0]) +
249                 (i32Y * g_pi32TouchParameters[1]) + g_pi32TouchParameters[2]) /
250                g_pi32TouchParameters[6]);
251     i32Y = (((i32X * g_pi32TouchParameters[3]) +
252              (i32Y * g_pi32TouchParameters[4]) + g_pi32TouchParameters[5]) /
253             g_pi32TouchParameters[6]);
254     i32X = i32Temp;
```

```
255
256         //
257         // See if the touch screen is being touched.
258         //
259         if((g_i16TouchX < g_i16TouchMin) || (g_i16TouchY < g_i16TouchMin))
260         {
261             //
262             // If there are no valid values yet then ignore this state.
263             //
264             if((g_ui8State & 0x80) == 0)
265             {
266                 g_ui8State = 0;
267             }
268
269             //
270             // See if the pen is not up right now.
271             //
272             if(g_ui8State != 0x00)
273             {
274                 //
275                 // Decrement the state count.
276                 //
277                 g_ui8State--;
278
279                 //
280                 // See if the pen has been detected as up three times in a row.
281                 //
282                 if(g_ui8State == 0x80)
283                 {
284                     //
285                     // Indicate that the pen is up.
286                     //
287                     g_ui8State = 0x00;
288
289                     //
290                     // See if there is a touch screen event handler.
291                     //
292                     if(g_pfnTSHandler)
293                     {
294                         //
295                         // If we got caught pre-filling the values, just return the
296                         // first valid value as a press and release.  If this is
297                         // not done there is a perceived miss of a press event.
298                         //
299                         if(g_i8Index < 0)
300                         {
301                             g_pfnTSHandler(WIDGET_MSG_PTR_DOWN, g_pi16Samples[0],
302                                            g_pi16Samples[1]);
303                             g_i8Index = 0;
304                         }
305
306                         //
307                         // Send the pen up message to the touch screen event
308                         // handler.
309                         //
310                         g_pfnTSHandler(WIDGET_MSG_PTR_UP, g_pi16Samples[g_i8Index],
311                                        g_pi16Samples[g_i8Index + 1]);
312                     }
313                 }
314             }
315         }
316         else
317         {
318             //
319             // If the state was counting down above then fall back to the idle
320             // state and start waiting for new values.
321             //
322             if((g_ui8State & 0x80) && (g_ui8State != 0x83))
323             {
324                 //
325                 // Restart the release count down.
326                 //
327                 g_ui8State = 0x83;
328             }
329
330             //
331             // See if the pen is not down right now.
332             //
333             if(g_ui8State != 0x83)
334             {
335                 //
336                 // Increment the state count.
337                 //
338                 g_ui8State++;
339
```

```
340             //
341             // See if the pen has been detected as down three times in a row.
342             //
343             if(g_ui8State == 0x03)
344             {
345                 //
346                 // Indicate that the pen is down.
347                 //
348                 g_ui8State = 0x83;
349 
350                 //
351                 // Set the index to -8, so that the next 3 samples are stored
352                 // into the sample buffer before sending anything back to the
353                 // touch screen event handler.
354                 //
355                 g_i8Index = -8;
356 
357                 //
358                 // Store this sample into the sample buffer.
359                 //
360                 g_pi16Samples[0] = i32X;
361                 g_pi16Samples[1] = i32Y;
362             }
363         }
364         else
365         {
366             //
367             // See if the sample buffer pre-fill has completed.
368             //
369             if(g_i8Index == -2)
370             {
371                 //
372                 // See if there is a touch screen event handler.
373                 //
374                 if(g_pfnTSHandler)
375                 {
376                     //
377                     // Send the pen down message to the touch screen event
378                     // handler.
379                     //
380                     g_pfnTSHandler(WIDGET_MSG_PTR_DOWN, g_pi16Samples[0],
381                                    g_pi16Samples[1]);
382                 }
383 
384                 //
385                 // Store this sample into the sample buffer.
386                 //
387                 g_pi16Samples[0] = i32X;
388                 g_pi16Samples[1] = i32Y;
389 
390                 //
391                 // Set the index to the next sample to send.
392                 //
393                 g_i8Index = 2;
394             }
395 
396             //
397             // Otherwise, see if the sample buffer pre-fill is in progress.
398             //
399             else if(g_i8Index < 0)
400             {
401                 //
402                 // Store this sample into the sample buffer.
403                 //
404                 g_pi16Samples[g_i8Index + 10] = i32X;
405                 g_pi16Samples[g_i8Index + 11] = i32Y;
406 
407                 //
408                 // Increment the index.
409                 //
410                 g_i8Index += 2;
411             }
412 
413             //
414             // Otherwise, the sample buffer is full.
415             //
416             else
417             {
418                 //
419                 // See if there is a touch screen event handler.
420                 //
421                 if(g_pfnTSHandler)
422                 {
423                     //
424                     // Send the pen move message to the touch screen event
```

```
425                     // handler.
426                     //
427                     g_pfnTSHandler(WIDGET_MSG_PTR_MOVE,
428                                    g_pi16Samples[g_i8Index],
429                                    g_pi16Samples[g_i8Index + 1]);
430                 }
431
432                 //
433                 // Store this sample into the sample buffer.
434                 //
435                 g_pi16Samples[g_i8Index] = i32X;
436                 g_pi16Samples[g_i8Index + 1] = i32Y;
437
438                 //
439                 // Increment the index.
440                 //
441                 g_i8Index = (g_i8Index + 2) & 7;
442             }
443         }
444     }
445 }
446
447 //*****************************************************************************
448 //
449 //! Handles the ADC interrupt for the touch screen.
450 //!
451 //! This function is called when the ADC sequence that samples the touch screen
452 //! has completed its acquisition.  The touch screen state machine is advanced
453 //! and the acquired ADC sample is processed appropriately.
454 //!
455 //! It is the responsibility of the application using the touch screen driver
456 //! to ensure that this function is installed in the interrupt vector table for
457 //! the ADC3 interrupt.
458 //!
459 //! \return None.
460 //
461 //*****************************************************************************
462 void
463 TouchScreenIntHandler(void)
464 {
465     //
466     // Clear the ADC sample sequence interrupt.
467     //
468     HWREG(ADC0_BASE + ADC_O_ISC) = 1 << 3;
469
470     //
471     // Determine what to do based on the current state of the state machine.
472     //
473     switch(g_ui32TSState)
474     {
475         //
476         // The new sample is an X-axis sample that should be discarded.
477         //
478         case TS_STATE_SKIP_X:
479         {
480             //
481             // Read and throw away the ADC sample.
482             //
483             HWREG(ADC0_BASE + ADC_O_SSFIFO3);
484
485             //
486             // Set the analog mode select for the YP pin.
487             //
488             HWREG(TS_P_BASE + GPIO_O_AMSEL) =
489                   HWREG(TS_P_BASE + GPIO_O_AMSEL) | TS_YP_PIN;
490
491             //
492             // Configure the Y-axis touch layer pins as inputs.
493             //
494             HWREG(TS_P_BASE + GPIO_O_DIR) =
495                   HWREG(TS_P_BASE + GPIO_O_DIR) & ~TS_YP_PIN;
496             HWREG(TS_YN_BASE + GPIO_O_DIR) =
497                   HWREG(TS_YN_BASE + GPIO_O_DIR) & ~TS_YN_PIN;
498
499             //
500             // The next sample will be a valid X-axis sample.
501             //
502             g_ui32TSState = TS_STATE_READ_X;
503
504             //
505             // This state has been handled.
506             //
507             break;
508         }
509
```

```
510             //
511             // The new sample is an X-axis sample that should be processed.
512             //
513             case TS_STATE_READ_X:
514             {
515                 //
516                 // Read the raw ADC sample.
517                 //
518                 g_i16TouchX = HWREG(ADC0_BASE + ADC_O_SSFIFO3);
519
520                 //
521                 // Clear the analog mode select for the YP pin.
522                 //
523                 HWREG(TS_P_BASE + GPIO_O_AMSEL) =
524                     HWREG(TS_P_BASE + GPIO_O_AMSEL) & ~TS_YP_PIN;
525
526                 //
527                 // Configure the X- and Y-axis touch layers as outputs.
528                 //
529                 HWREG(TS_P_BASE + GPIO_O_DIR) =
530                     HWREG(TS_P_BASE + GPIO_O_DIR) | TS_XP_PIN | TS_YP_PIN;
531                 HWREG(TS_XN_BASE + GPIO_O_DIR) =
532                       HWREG(TS_XN_BASE + GPIO_O_DIR) | TS_XN_PIN;
533                 HWREG(TS_YN_BASE + GPIO_O_DIR) =
534                       HWREG(TS_YN_BASE + GPIO_O_DIR) | TS_YN_PIN;
535
536                 //
537                 // Drive the positive side of the Y-axis touch layer with VDD and
538                 // the negative side with GND.  Also, drive both sides of the X-
539                 // axis layer with GND to discharge any residual voltage (so that
540                 // a no-touch condition can be properly detected).
541                 //
542                 HWREG(TS_XN_BASE + GPIO_O_DATA + (TS_XN_PIN << 2)) = 0;
543                 HWREG(TS_YN_BASE + GPIO_O_DATA + (TS_YN_PIN << 2)) = 0;
544                 HWREG(TS_P_BASE + GPIO_O_DATA + ((TS_XP_PIN | TS_YP_PIN) << 2)) =
545                     TS_YP_PIN;
546
547                 //
548                 // Configure the sample sequence to capture the X-axis value.
549                 //
550                 HWREG(ADC0_BASE + ADC_O_SSMUX3) = TS_XP_ADC;
551
552                 //
553                 // The next sample will be an invalid Y-axis sample.
554                 //
555                 g_ui32TSState = TS_STATE_SKIP_Y;
556
557                 //
558                 // This state has been handled.
559                 //
560                 break;
561             }
562
563             //
564             // The new sample is a Y-axis sample that should be discarded.
565             //
566             case TS_STATE_SKIP_Y:
567             {
568                 //
569                 // Read and throw away the ADC sample.
570                 //
571                 HWREG(ADC0_BASE + ADC_O_SSFIFO3);
572
573                 //
574                 // Set the analog mode select for the XP pin.
575                 //
576                 HWREG(TS_P_BASE + GPIO_O_AMSEL) =
577                       HWREG(TS_P_BASE + GPIO_O_AMSEL) | TS_XP_PIN;
578
579                 //
580                 // Configure the X-axis touch layer pins as inputs.
581                 //
582                 HWREG(TS_P_BASE + GPIO_O_DIR) =
583                       HWREG(TS_P_BASE + GPIO_O_DIR) & ~TS_XP_PIN;
584                 HWREG(TS_XN_BASE + GPIO_O_DIR) =
585                       HWREG(TS_XN_BASE + GPIO_O_DIR) & ~TS_XN_PIN;
586
587                 //
588                 // The next sample will be a valid Y-axis sample.
589                 //
590                 g_ui32TSState = TS_STATE_READ_Y;
591
592                 //
593                 // This state has been handled.
594                 //
```

```
595             break;
596         }
597 
598         //
599         // The new sample is a Y-axis sample that should be processed.
600         //
601         case TS_STATE_READ_Y:
602         {
603             //
604             // Read the raw ADC sample.
605             //
606             g_i16TouchY = HWREG(ADC0_BASE + ADC_O_SSFIFO3);
607 
608             //
609             // The next configuration is the same as the initial configuration.
610             // Therefore, fall through into the initialization state to avoid
611             // duplicating the code.
612             //
613         }
614 
615         //
616         // The state machine is in its initial state
617         //
618         case TS_STATE_INIT:
619         {
620             //
621             // Clear the analog mode select for the XP pin.
622             //
623             HWREG(TS_P_BASE + GPIO_O_AMSEL) =
624                 HWREG(TS_P_BASE + GPIO_O_AMSEL) & ~TS_XP_PIN;
625 
626             //
627             // Configure the X- and Y-axis touch layers as outputs.
628             //
629             HWREG(TS_P_BASE + GPIO_O_DIR) =
630                   HWREG(TS_P_BASE + GPIO_O_DIR) | TS_XP_PIN | TS_YP_PIN;
631             HWREG(TS_XN_BASE + GPIO_O_DIR) =
632                   HWREG(TS_XN_BASE + GPIO_O_DIR) | TS_XN_PIN;
633             HWREG(TS_YN_BASE + GPIO_O_DIR) =
634                   HWREG(TS_YN_BASE + GPIO_O_DIR) | TS_YN_PIN;
635 
636             //
637             // Drive one side of the X-axis touch layer with VDD and the other
638             // with GND.  Also, drive both sides of the Y-axis layer with GND
639             // to discharge any residual voltage (so that a no-touch condition
640             // can be properly detected).
641             //
642             HWREG(TS_P_BASE + GPIO_O_DATA + ((TS_XP_PIN | TS_YP_PIN) << 2)) =
643                 TS_XP_PIN;
644             HWREG(TS_XN_BASE + GPIO_O_DATA + ((TS_XN_PIN) << 2)) = 0;
645             HWREG(TS_YN_BASE + GPIO_O_DATA + ((TS_YN_PIN) << 2)) = 0;
646 
647             //
648             // Configure the sample sequence to capture the Y-axis value.
649             //
650             HWREG(ADC0_BASE + ADC_O_SSMUX3) = TS_YP_ADC;
651 
652             //
653             // If this is the valid Y sample state, then there is a new X/Y
654             // sample pair.  In that case, run the touch screen debouncer.
655             //
656             if(g_ui32TSState == TS_STATE_READ_Y)
657             {
658                 TouchScreenDebouncer();
659             }
660 
661             //
662             // The next sample will be an invalid X-axis sample.
663             //
664             g_ui32TSState = TS_STATE_SKIP_X;
665 
666             //
667             // This state has been handled.
668             //
669             break;
670         }
671     }
672 }
673 
674 //*****************************************************************************
675 //
676 //! Initializes the touch screen driver.
677 //!
678 //! \param ui32SysClock is the frequency of the system clock.
679 //!
```

```
680 //! This function initializes the touch screen driver, beginning the process of
681 //! reading from the touch screen.  This driver uses the following hardware
682 //! resources:
683 //!
684 //! - ADC sample sequence 3
685 //! - Timer 1 subtimer A
686 //!
687 //! \return None.
688 //
689 //*****************************************************************************
690 void
691 TouchScreenInit(uint32_t ui32SysClock)
692 {
693     //
694     // Set the initial state of the touch screen driver's state machine.
695     //
696     g_ui32TSState = TS_STATE_INIT;
697
698     //
699     // Determine which calibration parameter set we will be using.
700     //
701     g_pi32ParmSet = g_pi32TouchParameters;
702
703     //
704     // There is no touch screen handler initially.
705     //
706     g_pfnTSHandler = 0;
707
708     //
709     // Enable the peripherals used by the touch screen interface.
710     //
711     SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
712     SysCtlPeripheralEnable(TS_P_PERIPH);
713     SysCtlPeripheralEnable(TS_XN_PERIPH);
714     SysCtlPeripheralEnable(TS_YN_PERIPH);
715     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
716
717     //
718     // Configure the ADC sample sequence used to read the touch screen reading.
719     //
720     ADCHardwareOversampleConfigure(ADC0_BASE, 16);
721     ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0);
722     ADCSequenceStepConfigure(ADC0_BASE, 3, 0,
723                              TS_YP_ADC | ADC_CTL_END | ADC_CTL_IE |
724                              ADC_CTL_SHOLD_256);
725     ADCSequenceEnable(ADC0_BASE, 3);
726
727     //
728     // Enable the ADC sample sequence interrupt.
729     //
730     ADCIntEnable(ADC0_BASE, 3);
731     IntEnable(INT_ADC0SS3);
732
733     //
734     // Configure the GPIOs used to drive the touch screen layers.
735     //
736     GPIOPinTypeGPIOOutput(TS_P_BASE, TS_XP_PIN | TS_YP_PIN);
737     GPIOPinTypeGPIOOutput(TS_XN_BASE, TS_XN_PIN);
738     GPIOPinTypeGPIOOutput(TS_YN_BASE, TS_YN_PIN);
739
740     GPIOPinWrite(TS_P_BASE, TS_XP_PIN | TS_YP_PIN, 0x00);
741     GPIOPinWrite(TS_XN_BASE, TS_XN_PIN, 0x00);
742     GPIOPinWrite(TS_YN_BASE, TS_YN_PIN, 0x00);
743
744     //
745     // See if the ADC trigger timer has been configured, and configure it only
746     // if it has not been configured yet.
747     //
748     if((HWREG(TIMER1_BASE + TIMER_O_CTL) & TIMER_CTL_TAEN) == 0)
749     {
750         //
751         // Configure the timer to trigger the sampling of the touch screen
752         // every millisecond.
753         //
754         TimerConfigure(TIMER1_BASE, (TIMER_CFG_SPLIT_PAIR |
755                                      TIMER_CFG_A_PERIODIC |
756                                      TIMER_CFG_B_PERIODIC));
757         TimerLoadSet(TIMER1_BASE, TIMER_A, (ui32SysClock / 400) - 1);
758         TimerControlTrigger(TIMER1_BASE, TIMER_A, true);
759
760         //
761         // Enable the timer.  At this point, the touch screen state machine
762         // will sample and run once per millisecond.
763         //
764         TimerEnable(TIMER1_BASE, TIMER_A);
```

```
765         }
766     }
767
768     //*****************************************************************************
769     //
770     //! Sets the callback function for touch screen events.
771     //!
772     //! \param pfnCallback is a pointer to the function to be called when touch
773     //! screen events occur.
774     //!
775     //! This function sets the address of the function to be called when touch
776     //! screen events occur.  The events that are recognized are the screen being
777     //! touched (``pen down''), the touch position moving while the screen is
778     //! touched (``pen move''), and the screen no longer being touched (``pen
779     //! up'').
780     //!
781     //! \return None.
782     //
783     //*****************************************************************************
784     void
785     TouchScreenCallbackSet(int32_t (*pfnCallback)(uint32_t ui32Message,
786                                                   int32_t i32X, int32_t i32Y))
787     {
788         //
789         // Save the pointer to the callback function.
790         //
791         g_pfnTSHandler = pfnCallback;
792     }
793
794     //*****************************************************************************
795     //
796     // Close the Doxygen group.
797     //! @}
798     //
799     //*****************************************************************************
```

Quellcode D.15: Software-Modul: Angepasster Touchscreen-Treiber [36]

```
1   //*****************************************************************************
2   //
3   // touch.h - Prototypes for the touch screen driver.
4   //
5   // Copyright (c) 2012-2014 Texas Instruments Incorporated.  All rights reserved.
6   // Software License Agreement
7   //
8   // Texas Instruments (TI) is supplying this software for use solely and
9   // exclusively on TI's microcontroller products. The software is owned by
10  // TI and/or its suppliers, and is protected under applicable copyright
11  // laws. You may not combine this software with "viral" open-source
12  // software in order to form a larger program.
13  //
14  // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
15  // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
16  // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
17  // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
18  // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
19  // DAMAGES, FOR ANY REASON WHATSOEVER.
20  //
21  // This is part of revision 2.1.0.12573 of the EK-TM4C1294XL Firmware Package.
22  //
23  //*****************************************************************************
24
25  #ifndef __TOUCH_H__
26  #define __TOUCH_H__
27
28  //*****************************************************************************
29  //
30  // The lowest ADC reading assumed to represent a press on the screen.  Readings
31  // below this indicate no press is taking place.
32  //
33  //*****************************************************************************
34  #define TOUCH_MIN 300
35
36  //*****************************************************************************
37  //
38  // Prototypes for the functions exported by the touch screen driver.
39  //
40  //*****************************************************************************
41  extern volatile int16_t g_i16TouchX;
42  extern volatile int16_t g_i16TouchY;
43  extern int16_t g_i16TouchMin;
```

```
44 extern void TouchScreenIntHandler(void);
45 extern void TouchScreenInit(uint32_t ui32SysClock);
46 //extern void TouchScreenInit(uint32_t ui32SysClock);
47 extern void TouchScreenCallbackSet(int32_t (*pfnCallback)(uint32_t ui32Message,
48                                                           int32_t i32X,
49                                                           int32_t i32Y));
50
51 #endif // __TOUCH_H__
```

Quellcode D.16: Software-Modul: Leuchtdioden (Header)

```
1  /*
2   * leds.h
3   *
4   * Author: Felix Groth
5   */
6
7  #ifndef LEDS_H_
8  #define LEDS_H_
9
10
11 #include <stdint.h>
12 #include <stdbool.h>
13
14 #include "inc/hw_gpio.h"
15 #include "inc/hw_memmap.h"
16
17 #include "driverlib/sysctl.h"
18 #include "driverlib/gpio.h"
19
20 /** Defines for LED hardware ports **/
21 #define LED1_PORT_BASE      GPIO_PORTM_BASE
22 #define LED2_PORT_BASE      GPIO_PORTP_BASE
23 #define LED3_PORT_BASE      GPIO_PORTA_BASE
24 #define LED1_PORT_SYSCTL    SYSCTL_PERIPH_GPIOM
25 #define LED2_PORT_SYSCTL    SYSCTL_PERIPH_GPIOP
26 #define LED3_PORT_SYSCTL    SYSCTL_PERIPH_GPIOA
27 #define LED1_PIN        GPIO_PIN_7
28 #define LED2_PIN        GPIO_PIN_5
29 #define LED3_PIN        GPIO_PIN_7
30
31 /** Defines for LED functions **/
32 #define LED_YELLOW        0x02
33 #define LED_GREEN       0x04
34 #define LED_RED         0x01
35
36 void ledsInit(void);
37 uint8_t ledsWrite(uint8_t output);
38 uint8_t ledsRead(void);
39 uint8_t ledsToggle(uint8_t output);
40 void ledsChase(uint32_t speed);
41
42 #endif /* LEDS_H_ */
```

Quellcode D.17: Software-Modul: Leuchtdioden

```
1  /*
2   * leds.c
3   *
4   * Author: Felix Groth
5   */
6
7  #include "leds.h"
8
9  // Init LED ports and pins
10 void ledsInit(void) {
11     SysCtlPeripheralEnable(LED1_PORT_SYSCTL);
12     SysCtlPeripheralEnable(LED2_PORT_SYSCTL);
13     SysCtlPeripheralEnable(LED3_PORT_SYSCTL);
14
15     GPIOPinTypeGPIOOutput(LED1_PORT_BASE, LED1_PIN);
16     GPIOPinTypeGPIOOutput(LED2_PORT_BASE, LED2_PIN);
17     GPIOPinTypeGPIOOutput(LED3_PORT_BASE, LED3_PIN);
18 }
19
20
21 // Write three bits to the LEDs
```

```
22  uint8_t ledsWrite(uint8_t output) {
23    GPIOPinWrite(LED1_PORT_BASE, LED1_PIN, (output & 0x01) ? LED1_PIN : 0);
24    GPIOPinWrite(LED2_PORT_BASE, LED2_PIN, (output & 0x02) ? LED2_PIN : 0);
25    GPIOPinWrite(LED3_PORT_BASE, LED3_PIN, (output & 0x04) ? LED3_PIN : 0);
26    return output;
27  }
28
29  uint8_t ledsRead(void) {
30    return ((GPIOPinRead(LED1_PORT_BASE, LED1_PIN) ? 0x01 : 0x00) |
31        (GPIOPinRead(LED2_PORT_BASE, LED2_PIN) ? 0x02 : 0x00) |
32        (GPIOPinRead(LED3_PORT_BASE, LED3_PIN) ? 0x04 : 0x00));
33  }
34
35  uint8_t ledsToggle(uint8_t output) {
36    return ledsWrite(ledsRead() ^ output);
37  }
38
39  void ledsChase(uint32_t speed) {
40    uint32_t i;
41
42    ledsWrite(LED_GREEN);
43    for(i=0; i<speed; i++)
44      ;
45    ledsWrite(LED_YELLOW);
46    for(i=0; i<speed; i++)
47      ;
48    ledsWrite(LED_RED);
49    for(i=0; i<speed; i++)
50      ;
51    ledsWrite(LED_YELLOW);
52    for(i=0; i<speed; i++)
53      ;
54    ledsWrite(LED_GREEN);
55  }
```

Quellcode D.18: Software-Modul: Modifizierter FatFS-Treiber für SD-Karte [37]

```
1   /*-----------------------------------------------------------------------*/
2   /* MMC/SDC (in SPI mode) control module  (C)ChaN, 2007                   */
3   /*-----------------------------------------------------------------------*/
4   /* Only rcvr_spi(), xmit_spi(), disk_timerproc() and some macros         */
5   /* are platform dependent.                                               */
6   /*-----------------------------------------------------------------------*/
7
8   /*
9    * (C) 2014, Jon Magnuson <my.name at google's email service>
10  * This file is based on the sample driver provided by TI, and uses DMA
11  * for sector transmission.
12  */
13
14  /* Standard includes */
15  #include <stdint.h>
16  #include <stdbool.h>
17
18  /* Platform includes */
19  #include "inc/hw_memmap.h"
20  #include "inc/hw_types.h"
21  #include "inc/hw_ints.h"
22  #include "inc/hw_ssi.h"
23  #include "driverlib/gpio.h"
24  #include "driverlib/rom.h"
25  #include "driverlib/ssi.h"
26  #include "driverlib/sysctl.h"
27  #include "driverlib/interrupt.h"
28  #include "driverlib/udma.h"
29  #include "driverlib/pin_map.h"  //TODO: Reference
30
31  /* Debug includes */
32  #include "utils/uartstdio.h"
33
34  /* FatFs includes */
35  #include "diskio.h"
36
37  #if defined (USE_FREERTOS)
38  /* FreeRTOS Includes */
39  #include "FreeRTOS.h"
40  #include "task.h"
41  #include "queue.h"
42  #include "semphr.h"
43  #endif
44
```

```
45  /* Definitions for MMC/SDC command */
46  #define CMD0     (0x40+0)     /* GO_IDLE_STATE */
47  #define CMD1     (0x40+1)     /* SEND_OP_COND */
48  #define CMD8     (0x40+8)     /* SEND_IF_COND */
49  #define CMD9     (0x40+9)     /* SEND_CSD */
50  #define CMD10    (0x40+10)    /* SEND_CID */
51  #define CMD12    (0x40+12)    /* STOP_TRANSMISSION */
52  #define CMD16    (0x40+16)    /* SET_BLOCKLEN */
53  #define CMD17    (0x40+17)    /* READ_SINGLE_BLOCK */
54  #define CMD18    (0x40+18)    /* READ_MULTIPLE_BLOCK */
55  #define CMD23    (0x40+23)    /* SET_BLOCK_COUNT */
56  #define CMD24    (0x40+24)    /* WRITE_BLOCK */
57  #define CMD25    (0x40+25)    /* WRITE_MULTIPLE_BLOCK */
58  #define CMD41    (0x40+41)    /* SEND_OP_COND (ACMD) */
59  #define CMD55    (0x40+55)    /* APP_CMD */
60  #define CMD58    (0x40+58)    /* READ_OCR */
61
62  /* Peripheral definitions for DK-TM4C123G board */
63  // SSI port
64  #define SDC_SSI_BASE              SSI3_BASE
65  #define SDC_SSI_SYSCTL_PERIPH     SYSCTL_PERIPH_SSI3
66  #define SDC_SSI_INT               INT_SSI3
67  #define SDC_SSI_TX_UDMA_CHAN      UDMA_CHANNEL_ADC1
68  #define SDC_SSI_RX_UDMA_CHAN      UDMA_CHANNEL_ADC0
69
70  // GPIO for SSI pins
71  #define SDC_GPIO_PORT_BASE        GPIO_PORTQ_BASE
72  #define SDC_GPIO_SYSCTL_PERIPH    SYSCTL_PERIPH_GPIOQ
73  #define SDC_SSI_CLK               GPIO_PIN_0
74  #define SDC_SSI_TX                GPIO_PIN_2
75  #define SDC_SSI_RX                GPIO_PIN_3
76  #define SDC_SSI_FSS               GPIO_PIN_1
77  #define SDC_SSI_PINS              (SDC_SSI_TX | SDC_SSI_RX | SDC_SSI_CLK |  \
78                                     SDC_SSI_FSS)
79
80  //#define USE_DMA_TX
81  //#define USE_DMA_RX
82
83
84  void init_dma(uint8_t send);
85  unsigned int sector_send_dma(uint8_t *buff, uint32_t len);
86  unsigned int sector_receive_dma(uint8_t *buff, uint32_t len);
87  static uint32_t dma_complete=0;
88  static uint8_t dummy_rx = 0x00;
89  static uint8_t dummy_tx = 0xff;
90
91  // asserts the CS pin to the card
92  static
93  void SELECT (void)
94  {
95      ROM_GPIOPinWrite(SDC_GPIO_PORT_BASE, SDC_SSI_FSS, 0);
96  }
97
98  // de-asserts the CS pin to the card
99  static
100 void DESELECT (void)
101 {
102     ROM_GPIOPinWrite(SDC_GPIO_PORT_BASE, SDC_SSI_FSS, SDC_SSI_FSS);
103 }
104
105 /*-----------------------------------------------------------------------
106
107    Module Private Functions
108
109 -----------------------------------------------------------------------*/
110
111 static volatile
112 DSTATUS Stat = STA_NOINIT;    /* Disk status */
113
114 static volatile
115 BYTE Timer1, Timer2;    /* 100Hz decrement timer */
116
117 static
118 BYTE CardType;            /* b0:MMC, b1:SDC, b2:Block addressing */
119
120 static
121 BYTE PowerFlag = 0;     /* indicates if "power" is on */
122
123 /*-----------------------------------------------------------------------*/
124 /* Transmit a byte to MMC via SPI  (Platform dependent)                  */
125 /*-----------------------------------------------------------------------*/
126
127 static
128 void xmit_spi(BYTE dat)
129 {
```

```
130     uint32_t ui32RcvDat;
131
132     ROM_SSIDataPut(SDC_SSI_BASE, dat); /* Write the data to the tx fifo */
133
134     ROM_SSIDataGet(SDC_SSI_BASE, &ui32RcvDat); /* flush data read during the write */
135 }
136
137
138 /*-----------------------------------------------------------------------*/
139 /* Receive a byte from MMC via SPI  (Platform dependent)                 */
140 /*-----------------------------------------------------------------------*/
141
142 static
143 BYTE rcvr_spi (void)
144 {
145     uint32_t ui32RcvDat;
146
147     ROM_SSIDataPut(SDC_SSI_BASE, 0xFF); /* write dummy data */
148
149     ROM_SSIDataGet(SDC_SSI_BASE, &ui32RcvDat); /* read data frm rx fifo */
150
151     return (BYTE)ui32RcvDat;
152 }
153
154
155 static
156 void rcvr_spi_m (BYTE *dst)
157 {
158     *dst = rcvr_spi();
159 }
160
161 /*-----------------------------------------------------------------------*/
162 /* Wait for card ready                                                   */
163 /*-----------------------------------------------------------------------*/
164
165 static
166 BYTE wait_ready (void)
167 {
168     BYTE res;
169
170
171     Timer2 = 50;    /* Wait for ready in timeout of 500ms */
172     rcvr_spi();
173     do
174         res = rcvr_spi();
175     while ((res != 0xFF) && Timer2);
176
177     return res;
178 }
179
180 /*-----------------------------------------------------------------------*/
181 /* Send 80 or so clock transitions with CS and DI held high. This is     */
182 /* required after card power up to get it into SPI mode                  */
183 /*-----------------------------------------------------------------------*/
184 static
185 void send_initial_clock_train(void)
186 {
187     unsigned int i;
188     uint32_t ui32Dat;
189
190     /* Ensure CS is held high. */
191     DESELECT();
192
193     /* Switch the SSI TX line to a GPIO and drive it high too. */
194     ROM_GPIOPinTypeGPIOOutput(SDC_GPIO_PORT_BASE, SDC_SSI_TX);
195     ROM_GPIOPinWrite(SDC_GPIO_PORT_BASE, SDC_SSI_TX, SDC_SSI_TX);
196
197     /* Send 10 bytes over the SSI. This causes the clock to wiggle the */
198     /* required number of times. */
199     for(i = 0 ; i < 10 ; i++)
200     {
201         /* Write DUMMY data. SSIDataPut() waits until there is room in the */
202         /* FIFO. */
203         ROM_SSIDataPut(SDC_SSI_BASE, 0xFF);
204
205         /* Flush data read during data write. */
206         ROM_SSIDataGet(SDC_SSI_BASE, &ui32Dat);
207     }
208
209     /* Revert to hardware control of the SSI TX line. */
210     ROM_GPIOPinConfigure(GPIO_PQ2_SSI3XDAT0);
211     ROM_GPIOPinTypeSSI(SDC_GPIO_PORT_BASE, SDC_SSI_TX);
212     GPIOPadConfigSet(SDC_GPIO_PORT_BASE, SDC_SSI_CLK | SDC_SSI_TX | SDC_SSI_FSS,
213                      GPIO_STRENGTH_4MA, GPIO_PIN_TYPE_STD);
214 }
```

```
215
216 /*-----------------------------------------------------------------------*/
217 /* Power Control  (Platform dependent)                                   */
218 /*-----------------------------------------------------------------------*/
219 /* When the target system does not support socket power control, there   */
220 /* is nothing to do in these functions and chk_power always returns 1.   */
221
222 static
223 void power_on (void)
224 {
225     /*
226      * This doesn't really turn the power on, but initializes the
227      * SSI port and pins needed to talk to the card.
228      */
229
230     /* Enable the peripherals used to drive the SDC on SSI */
231     ROM_SysCtlPeripheralEnable(SDC_SSI_SYSCTL_PERIPH);
232     ROM_SysCtlPeripheralEnable(SDC_GPIO_SYSCTL_PERIPH);
233
234     /*
235      * Configure Pin-Muxing!
236      */
237     //TODO: Make these definable!
238     ROM_GPIOPinConfigure(GPIO_PQ0_SSI3CLK);
239     ROM_GPIOPinConfigure(GPIO_PQ1_SSI3FSS);
240     ROM_GPIOPinConfigure(GPIO_PQ3_SSI3XDAT1);
241     ROM_GPIOPinConfigure(GPIO_PQ2_SSI3XDAT0);
242
243     /*
244      * Configure the appropriate pins to be SSI instead of GPIO. The FSS (CS)
245      * signal is directly driven to ensure that we can hold it low through a
246      * complete transaction with the SD card.
247      */
248     ROM_GPIOPinTypeSSI(SDC_GPIO_PORT_BASE, SDC_SSI_TX | SDC_SSI_RX | SDC_SSI_CLK);
249     ROM_GPIOPinTypeGPIOOutput(SDC_GPIO_PORT_BASE, SDC_SSI_FSS);
250
251     /*
252      * Set the SSI output pins to 4MA drive strength and engage the
253      * pull-up on the receive line.
254      */
255     GPIOPadConfigSet(SDC_GPIO_PORT_BASE, SDC_SSI_RX, GPIO_STRENGTH_4MA,
256                      GPIO_PIN_TYPE_STD_WPU);
257     GPIOPadConfigSet(SDC_GPIO_PORT_BASE, SDC_SSI_CLK | SDC_SSI_TX | SDC_SSI_FSS,
258                      GPIO_STRENGTH_4MA, GPIO_PIN_TYPE_STD);
259
260     /* Configure the SSI0 port */
261     SSIConfigSetExpClk(SDC_SSI_BASE, g_sysClock,
262                        SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, 400000, 8);
263     ROM_SSIEnable(SDC_SSI_BASE);
264
265     /* Set DI and CS high and apply more than 74 pulses to SCLK for the card */
266     /* to be able to accept a native command. */
267     send_initial_clock_train();
268
269     PowerFlag = 1;
270 }
271
272 // set the SSI speed to the max setting
273 static
274 void set_max_speed(void)
275 {
276     unsigned long i;
277
278     /* Disable the SSI */
279     ROM_SSIDisable(SDC_SSI_BASE);
280
281     /* Set the maximum speed as half the system clock, with a max of 12.5 (30) MHz. */
282     i = g_sysClock / 2;
283     if(i > 30000000)  //TODO: Changed this from 12500000 / Seems to run fine on all my uSDs
284     {
285         i = 30000000;
286     }
287
288     /* Configure the SSI0 port to run at 12.5MHz */
289     ROM_SSIConfigSetExpClk(SDC_SSI_BASE, g_sysClock,
290                            SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, i, 8);
291
292     /* Enable the SSI */
293     ROM_SSIEnable(SDC_SSI_BASE);
294 }
295
296 static
297 void power_off (void)
298 {
299     PowerFlag = 0;
```

```
300 }
301
302 static
303 int chk_power(void)        /* Socket power state: 0=off, 1=on */
304 {
305     return PowerFlag;
306 }
307
308
309
310 /*-----------------------------------------------------------------------*/
311 /* Receive a data packet from MMC                                        */
312 /*-----------------------------------------------------------------------*/
313
314 static
315 BOOL rcvr_datablock (
316     BYTE *buff,            /* Data buffer to store received data */
317     UINT btr            /* Byte count (must be even number) */
318 )
319 {
320     BYTE token;
321
322
323     Timer1 = 100;
324     do {                            /* Wait for data packet in timeout of 100ms */
325         token = rcvr_spi();
326     } while ((token == 0xFF) && Timer1);
327     if(token != 0xFE) return FALSE;    /* If not valid data token, retutn with error */
328
329 #if defined(USE_DMA_RX)
330         sector_receive_dma((uint8_t*)buff, 512);
331 #else
332     do {                            /* Receive the data block into buffer */
333         rcvr_spi_m(buff++);
334         rcvr_spi_m(buff++);
335     } while (btr -= 2);
336 #endif
337     rcvr_spi();                        /* Discard CRC */
338     rcvr_spi();
339
340     return TRUE;                    /* Return with success */
341 }
342
343
344
345 /*-----------------------------------------------------------------------*/
346 /* Send a data packet to MMC                                             */
347 /*-----------------------------------------------------------------------*/
348
349 #if _READONLY == 0
350 static
351 BOOL xmit_datablock (
352     const BYTE *buff,    /* 512 byte data block to be transmitted */
353     BYTE token            /* Data/Stop token */
354 )
355 {
356     BYTE resp, wc;
357
358
359     if (wait_ready() != 0xFF) return FALSE;
360
361     xmit_spi(token);                    /* Xmit data token */
362     if (token != 0xFD) {    /* Is data token */
363         wc = 0;
364
365 #if defined(USE_DMA_TX)
366         sector_send_dma((uint8_t*)buff, 512);
367 #else
368         do {                            /* Xmit the 512 byte data block to MMC */
369             xmit_spi(*buff++);
370             xmit_spi(*buff++);
371         } while (--wc);
372 #endif
373         xmit_spi(0xFF);                    /* CRC (Dummy) */
374         xmit_spi(0xFF);
375         resp = rcvr_spi();
376         if ((resp & 0x1F) != 0x05) {    /* If not accepted, return with error */
377             return FALSE;
378         }
379     }
380
381
382     return TRUE;
383 }
384 #endif /* _READONLY */
```

```
385
386
387
388 /*-----------------------------------------------------------------------*/
389 /* Send a command packet to MMC                                          */
390 /*-----------------------------------------------------------------------*/
391
392 static
393 BYTE send_cmd (
394     BYTE cmd,        /* Command byte */
395     DWORD arg        /* Argument */
396 )
397 {
398     BYTE n, res;
399
400
401     if (wait_ready() != 0xFF) return 0xFF;
402
403     /* Send command packet */
404     xmit_spi(cmd);                        /* Command */
405     xmit_spi((BYTE)(arg >> 24));        /* Argument[31..24] */
406     xmit_spi((BYTE)(arg >> 16));        /* Argument[23..16] */
407     xmit_spi((BYTE)(arg >> 8));            /* Argument[15..8] */
408     xmit_spi((BYTE)arg);                /* Argument[7..0] */
409     n = 0xff;
410     if (cmd == CMD0) n = 0x95;            /* CRC for CMD0(0) */
411     if (cmd == CMD8) n = 0x87;            /* CRC for CMD8(0x1AA) */
412     xmit_spi(n);
413
414     /* Receive command response */
415     if (cmd == CMD12) rcvr_spi();        /* Skip a stuff byte when stop reading */
416     n = 10;                                /* Wait for a valid response in timeout of 10 attempts */
417     do
418         res = rcvr_spi();
419     while ((res & 0x80) && --n);
420
421     return res;            /* Return with the response value */
422 }
423
424 /*-----------------------------------------------------------------------*
425  * Send the special command used to terminate a multi-sector read.
426  *
427  * This is the only command which can be sent while the SDCard is sending
428  * data. The SDCard spec indicates that the data transfer will stop 2 bytes
429  * after the 6 byte CMD12 command is sent and that the card will then send
430  * 0xFF for between 2 and 6 more bytes before the R1 response byte.  This
431  * response will be followed by another 0xFF byte.  In testing, however, it
432  * seems that some cards don't send the 2 to 6 0xFF bytes between the end of
433  * data transmission and the response code.  This function, therefore, merely
434  * reads 10 bytes and, if the last one read is 0xFF, returns the value of the
435  * latest non-0xFF byte as the response code.
436  *
437  *-----------------------------------------------------------------------*/
438
439 static
440 BYTE send_cmd12 (void)
441 {
442     BYTE n, res, val;
443
444     /* For CMD12, we don't wait for the card to be idle before we send
445      * the new command.
446      */
447
448     /* Send command packet - the argument for CMD12 is ignored. */
449     xmit_spi(CMD12);
450     xmit_spi(0);
451     xmit_spi(0);
452     xmit_spi(0);
453     xmit_spi(0);
454     xmit_spi(0);
455
456     /* Read up to 10 bytes from the card, remembering the value read if it's
457        not 0xFF */
458     for(n = 0; n < 10; n++)
459     {
460         val = rcvr_spi();
461         if(val != 0xFF)
462         {
463             res = val;
464         }
465     }
466
467     return res;            /* Return with the response value */
468 }
469
```

```
/*-----------------------------------------------------------------------*/

   Public Functions

/*-----------------------------------------------------------------------*/


/*-----------------------------------------------------------------------*/
/* Initialize Disk Drive                                                 */
/*-----------------------------------------------------------------------*/

DSTATUS disk_initialize (
    BYTE drv        /* Physical drive nmuber (0) */
)
{
    BYTE n, ty, ocr[4];


    if (drv) return STA_NOINIT;           /* Supports only single drive */
    if (Stat & STA_NODISK) return Stat;    /* No card in the socket */

    power_on();                           /* Force socket power on */
    //send_initial_clock_train();          /* Ensure the card is in SPI mode */

    SELECT();               /* CS = L */
    ty = 0;
    if (send_cmd(CMD0, 0) == 1) {            /* Enter Idle state */
        Timer1 = 100;                        /* Initialization timeout of 1000 msec */
        if (send_cmd(CMD8, 0x1AA) == 1) {    /* SDC Ver2+ */
            for (n = 0; n < 4; n++) ocr[n] = rcvr_spi();
            if (ocr[2] == 0x01 && ocr[3] == 0xAA) {    /* The card can work at vdd range of 2.7-3.6V */
                do {
                    if (send_cmd(CMD55, 0) <= 1 && send_cmd(CMD41, 1UL << 30) == 0)    break;    /* ACMD41 with HCS bit */
                } while (Timer1);
                if (Timer1 && send_cmd(CMD58, 0) == 0) {    /* Check CCS bit */
                    for (n = 0; n < 4; n++) ocr[n] = rcvr_spi();
                    ty = (ocr[0] & 0x40) ? 6 : 2;
                }
            }
        } else {                            /* SDC Ver1 or MMC */
            ty = (send_cmd(CMD55, 0) <= 1 && send_cmd(CMD41, 0) <= 1) ? 2 : 1;    /* SDC : MMC */
            do {
                if (ty == 2) {
                    if (send_cmd(CMD55, 0) <= 1 && send_cmd(CMD41, 0) == 0) break;    /* ACMD41 */
                } else {
                    if (send_cmd(CMD1, 0) == 0) break;                                /* CMD1 */
                }
            } while (Timer1);
            if (!Timer1 || send_cmd(CMD16, 512) != 0)    /* Select R/W block length */
                ty = 0;
        }
    }
    CardType = ty;
    DESELECT();            /* CS = H */
    rcvr_spi();            /* Idle (Release DO) */

    if (ty) {            /* Initialization succeded */
        Stat &= ~STA_NOINIT;        /* Clear STA_NOINIT */
        set_max_speed();
    } else {            /* Initialization failed */
        power_off();
    }

    return Stat;
}



/*-----------------------------------------------------------------------*/
/* Get Disk Status                                                       */
/*-----------------------------------------------------------------------*/

DSTATUS disk_status (
    BYTE drv        /* Physical drive nmuber (0) */
)
{
    if (drv) return STA_NOINIT;        /* Supports only single drive */
    return Stat;
}



/*-----------------------------------------------------------------------*/
/* Read Sector(s)                                                        */
/*-----------------------------------------------------------------------*/
```

```
555
556 DRESULT disk_read (
557     BYTE drv,           /* Physical drive nmuber (0) */
558     BYTE *buff,           /* Pointer to the data buffer to store read data */
559     DWORD sector,       /* Start sector number (LBA) */
560     BYTE count           /* Sector count (1..255) */
561 )
562 {
563     if (drv || !count) return RES_PARERR;
564     if (Stat & STA_NOINIT) return RES_NOTRDY;
565
566     if (!(CardType & 4)) sector *= 512;    /* Convert to byte address if needed */
567
568     SELECT();           /* CS = L */
569
570     if (count == 1) {    /* Single block read */
571         if ((send_cmd(CMD17, sector) == 0)    /* READ_SINGLE_BLOCK */
572             && rcvr_datablock(buff, 512))
573             count = 0;
574     }
575     else {               /* Multiple block read */
576         if (send_cmd(CMD18, sector) == 0) {    /* READ_MULTIPLE_BLOCK */
577             do {
578                 if (!rcvr_datablock(buff, 512)) break;
579                 buff += 512;
580             } while (--count);
581             send_cmd12();               /* STOP_TRANSMISSION */
582         }
583     }
584
585     DESELECT();           /* CS = H */
586     rcvr_spi();           /* Idle (Release DO) */
587
588     return count ? RES_ERROR : RES_OK;
589 }
590
591
592
593 /*-----------------------------------------------------------------------*/
594 /* Write Sector(s)                                                       */
595 /*-----------------------------------------------------------------------*/
596
597 #if _READONLY == 0
598 DRESULT disk_write (
599     BYTE drv,           /* Physical drive nmuber (0) */
600     const BYTE *buff,    /* Pointer to the data to be written */
601     DWORD sector,       /* Start sector number (LBA) */
602     BYTE count           /* Sector count (1..255) */
603 )
604 {
605     if (drv || !count) return RES_PARERR;
606     if (Stat & STA_NOINIT) return RES_NOTRDY;
607     if (Stat & STA_PROTECT) return RES_WRPRT;
608
609     if (!(CardType & 4)) sector *= 512;    /* Convert to byte address if needed */
610
611     SELECT();           /* CS = L */
612
613     if (count == 1) {    /* Single block write */
614         if ((send_cmd(CMD24, sector) == 0)    /* WRITE_BLOCK */
615             && xmit_datablock(buff, 0xFE))
616             count = 0;
617     }
618     else {               /* Multiple block write */
619         if (CardType & 2) {
620             send_cmd(CMD55, 0); send_cmd(CMD23, count);    /* ACMD23 */
621         }
622         if (send_cmd(CMD25, sector) == 0) {    /* WRITE_MULTIPLE_BLOCK */
623             do {
624                 if (!xmit_datablock(buff, 0xFC)) break;
625                 buff += 512;
626             } while (--count);
627             if (!xmit_datablock(0, 0xFD))    /* STOP_TRAN token */
628                 count = 1;
629         }
630     }
631
632     DESELECT();           /* CS = H */
633     rcvr_spi();           /* Idle (Release DO) */
634
635     return count ? RES_ERROR : RES_OK;
636 }
637 #endif /* _READONLY */
638
639
```

```
640
641 /*-----------------------------------------------------------------------*/
642 /* Miscellaneous Functions                                               */
643 /*-----------------------------------------------------------------------*/
644
645 DRESULT disk_ioctl (
646     BYTE drv,        /* Physical drive nmuber (0) */
647     BYTE ctrl,       /* Control code */
648     void *buff       /* Buffer to send/receive control data */
649 )
650 {
651     DRESULT res;
652     BYTE n, csd[16], *ptr = buff;
653     WORD csize;
654
655
656     if (drv) return RES_PARERR;
657
658     res = RES_ERROR;
659
660     if (ctrl == CTRL_POWER) {
661         switch (*ptr) {
662         case 0:         /* Sub control code == 0 (POWER_OFF) */
663             if (chk_power())
664                 power_off();        /* Power off */
665             res = RES_OK;
666             break;
667         case 1:         /* Sub control code == 1 (POWER_ON) */
668             power_on();             /* Power on */
669             res = RES_OK;
670             break;
671         case 2:         /* Sub control code == 2 (POWER_GET) */
672             *(ptr+1) = (BYTE)chk_power();
673             res = RES_OK;
674             break;
675         default :
676             res = RES_PARERR;
677         }
678     }
679     else {
680         if (Stat & STA_NOINIT) return RES_NOTRDY;
681
682         SELECT();       /* CS = L */
683
684         switch (ctrl) {
685         case GET_SECTOR_COUNT :    /* Get number of sectors on the disk (DWORD) */
686             if ((send_cmd(CMD9, 0) == 0) && rcvr_datablock(csd, 16)) {
687                 if ((csd[0] >> 6) == 1) {    /* SDC ver 2.00 */
688                     csize = csd[9] + ((WORD)csd[8] << 8) + 1;
689                     *(DWORD*)buff = (DWORD)csize << 10;
690                 } else {                     /* MMC or SDC ver 1.XX */
691                     n = (csd[5] & 15) + ((csd[10] & 128) >> 7) + ((csd[9] & 3) << 1) + 2;
692                     csize = (csd[8] >> 6) + ((WORD)csd[7] << 2) + ((WORD)(csd[6] & 3) << 10) + 1;
693                     *(DWORD*)buff = (DWORD)csize << (n - 9);
694                 }
695                 res = RES_OK;
696             }
697             break;
698
699         case GET_SECTOR_SIZE :    /* Get sectors on the disk (WORD) */
700             *(WORD*)buff = 512;
701             res = RES_OK;
702             break;
703
704         case CTRL_SYNC :    /* Make sure that data has been written */
705             if (wait_ready() == 0xFF)
706                 res = RES_OK;
707             break;
708
709         case MMC_GET_CSD :    /* Receive CSD as a data block (16 bytes) */
710             if (send_cmd(CMD9, 0) == 0        /* READ_CSD */
711                 && rcvr_datablock(ptr, 16))
712                 res = RES_OK;
713             break;
714
715         case MMC_GET_CID :    /* Receive CID as a data block (16 bytes) */
716             if (send_cmd(CMD10, 0) == 0        /* READ_CID */
717                 && rcvr_datablock(ptr, 16))
718                 res = RES_OK;
719             break;
720
721         case MMC_GET_OCR :    /* Receive OCR as an R3 resp (4 bytes) */
722             if (send_cmd(CMD58, 0) == 0) {    /* READ_OCR */
723                 for (n = 0; n < 4; n++)
724                     *ptr++ = rcvr_spi();
```

```
725                 res = RES_OK;
726             }
727
728 //        case MMC_GET_TYPE :    /* Get card type flags (1 byte) */
729 //            *ptr = CardType;
730 //            res = RES_OK;
731 //            break;
732
733         default:
734             res = RES_PARERR;
735         }
736
737         DESELECT();             /* CS = H */
738         rcvr_spi();             /* Idle (Release DO) */
739     }
740
741     return res;
742 }
743
744
745
746 /*-----------------------------------------------------------------------*/
747 /* Device Timer Interrupt Procedure  (Platform dependent)                 */
748 /*-----------------------------------------------------------------------*/
749 /* This function must be called in period of 10ms                        */
750
751 void disk_timerproc (void)
752 {
753 //    BYTE n, s;
754     BYTE n;
755
756
757     n = Timer1;                         /* 100Hz decrement timer */
758     if (n) Timer1 = --n;
759     n = Timer2;
760     if (n) Timer2 = --n;
761
762 }
763
764 /*-----------------------------------------------------------------------*/
765 /* User Provided Timer Function for FatFs module                         */
766 /*-----------------------------------------------------------------------*/
767 /* This is a real time clock service to be called from                   */
768 /* FatFs module. Any valid time must be returned even if                 */
769 /* the system does not support a real time clock.                        */
770
771 DWORD get_fattime (void)
772 {
773
774     return    ((2007UL-1980) << 25)     // Year = 2007
775             | (6UL << 21)               // Month = June
776             | (5UL << 16)               // Day = 5
777             | (11U << 11)               // Hour = 11
778             | (38U << 5)                // Min = 38
779             | (0U >> 1)                 // Sec = 0
780             ;
781
782 }
783
784 /*****************************************************************************
785  *
786  *                           SD DMA FUNCTIONS
787  *
788  ****************************************************************************/
789 void
790 SDCSSIIntHandler(void)
791 {
792 #if defined (USE_FREERTOS)
793     portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;
794 #endif
795
796     uint32_t ui32Status;
797     uint32_t ui32Mode;
798
799     /* Get status */
800     ui32Status = ROM_SSIIntStatus(SDC_SSI_BASE, TRUE);
801
802     /* Clear status */
803     ROM_SSIIntClear(SDC_SSI_BASE, ui32Status);
804
805     ui32Mode = ROM_uDMAChannelModeGet(SDC_SSI_RX_UDMA_CHAN | UDMA_PRI_SELECT);
806
807     if(ui32Mode == UDMA_MODE_STOP /*UDMA_MODE_BASIC*/)
808     {
809         /* Signal txfer complete */
```

```
810         dma_complete = 1;
811 #if defined (USE_FREERTOS)
812         /* TODO: Implement RTOS semaphore for process yielding */
813 #endif
814     }
815
816     /* If the SSI DMA TX channel is disabled, that means the TX DMA txfer is complete */
817     if(!ROM_uDMAChannelIsEnabled(SDC_SSI_TX_UDMA_CHAN))
818     {
819
820     }
821
822 #if defined (USE_FREERTOS)
823     /* Switch tasks if necessary. */
824     if ( xHigherPriorityTaskWoken != pdFALSE ) {
825         portEND_SWITCHING_ISR( xHigherPriorityTaskWoken );
826     }
827 #endif
828
829 }
830
831 unsigned int
832 sector_send_dma(uint8_t *buff, uint32_t len)
833 {
834
835     volatile uint32_t discard;
836
837     dma_complete = 0;
838
839     /* Re-initialize DMA every transmission */
840     init_dma(1);
841
842     ROM_uDMAChannelTransferSet(SDC_SSI_RX_UDMA_CHAN | UDMA_PRI_SELECT,
843                                UDMA_MODE_BASIC,
844                                (void *)(SSI0_BASE + SSI_O_DR),
845                                &dummy_rx,
846                                len /*512*/);
847
848     ROM_uDMAChannelTransferSet(SDC_SSI_TX_UDMA_CHAN | UDMA_PRI_SELECT,
849                                UDMA_MODE_BASIC,
850                                buff,
851                                (void *)(SDC_SSI_BASE + SSI_O_DR),
852                                len);
853
854     /* Initiate DMA txfer */
855     ROM_uDMAChannelEnable(SDC_SSI_RX_UDMA_CHAN);
856     ROM_uDMAChannelEnable(SDC_SSI_TX_UDMA_CHAN);
857
858     while (!dma_complete);
859
860
861     while(1)
862     {
863         /* Double-check to make sure DMA txfer is done (can probably remove) */
864         uint32_t evFlags = ROM_uDMAChannelModeGet(SDC_SSI_RX_UDMA_CHAN | UDMA_PRI_SELECT);
865         if (evFlags==UDMA_MODE_STOP) break;
866     }
867
868     //for (discard=100; discard; discard--);
869
870     ROM_uDMAChannelDisable(SDC_SSI_RX_UDMA_CHAN);
871     ROM_uDMAChannelDisable(SDC_SSI_TX_UDMA_CHAN);
872     ROM_SSIDMADisable(SDC_SSI_BASE, SSI_DMA_TX | SSI_DMA_RX);
873
874     return 0;
875 }
876
877 unsigned int
878 sector_receive_dma(uint8_t *buff, uint32_t len)
879 {
880
881     volatile uint32_t discard;
882
883     dma_complete = 0;
884
885     /* Re-initialize DMA every transmission */
886     init_dma(0);
887
888     ROM_uDMAChannelTransferSet(SDC_SSI_RX_UDMA_CHAN | UDMA_PRI_SELECT,
889                                UDMA_MODE_BASIC,
890                                (void *)(SSI0_BASE + SSI_O_DR),
891                                buff,
892                                len /*512*/);
893
894     ROM_uDMAChannelTransferSet(SDC_SSI_TX_UDMA_CHAN | UDMA_PRI_SELECT,
```

```
895                                 UDMA_MODE_BASIC,
896                                 &dummy_tx,
897                                 (void *)(SDC_SSI_BASE + SSI_O_DR),
898                                 len);
899
900     /* Initiate DMA txfer */
901     ROM_uDMAChannelEnable(SDC_SSI_RX_UDMA_CHAN);
902     ROM_uDMAChannelEnable(SDC_SSI_TX_UDMA_CHAN);
903
904     while (!dma_complete);
905
906     while(1)
907     {
908         /* Double-check to make sure DMA txfer is done (can probably remove) */
909         uint32_t evFlags = ROM_uDMAChannelModeGet(SDC_SSI_RX_UDMA_CHAN | UDMA_PRI_SELECT);
910         if (evFlags==UDMA_MODE_STOP) break;
911     }
912
913     //for (discard=100; discard; discard--);
914
915     ROM_uDMAChannelDisable(SDC_SSI_RX_UDMA_CHAN);
916     ROM_uDMAChannelDisable(SDC_SSI_TX_UDMA_CHAN);
917     ROM_SSIDMADisable(SDC_SSI_BASE, SSI_DMA_TX | SSI_DMA_RX);
918
919     return 0;
920 }
921
922 void
923 init_dma(uint8_t send)
924 {
925
926     /* Init SPI DMA & SPI interrupt */
927     ROM_SSIDMAEnable(SDC_SSI_BASE, SSI_DMA_TX | SSI_DMA_RX);
928     ROM_IntEnable(SDC_SSI_INT);
929
930     //TODO: Make these definable
931     ROM_uDMAChannelAssign(UDMA_CH14_SSI3RX);
932     ROM_uDMAChannelAssign(UDMA_CH15_SSI3TX);
933
934     if (send) {
935
936         /* RX */
937         ROM_uDMAChannelAttributeDisable(SDC_SSI_RX_UDMA_CHAN, UDMA_ATTR_ALL);
938         ROM_uDMAChannelControlSet(SDC_SSI_RX_UDMA_CHAN | UDMA_PRI_SELECT,
939                                   UDMA_SIZE_8 | UDMA_SRC_INC_NONE | UDMA_DST_INC_NONE | UDMA_ARB_4);
940
941         /* TX */
942         ROM_uDMAChannelAttributeDisable(SDC_SSI_TX_UDMA_CHAN,
943                                         UDMA_ATTR_ALTSELECT
944                                         | UDMA_ATTR_HIGH_PRIORITY
945                                         | UDMA_ATTR_REQMASK);
946         ROM_uDMAChannelControlSet(SDC_SSI_TX_UDMA_CHAN | UDMA_PRI_SELECT,
947                                   UDMA_SIZE_8 | UDMA_SRC_INC_8 | UDMA_DST_INC_NONE | UDMA_ARB_4);
948     }
949     else { /* receive */
950
951         /* RX */
952         ROM_uDMAChannelAttributeDisable(SDC_SSI_RX_UDMA_CHAN, UDMA_ATTR_ALL);
953         ROM_uDMAChannelControlSet(SDC_SSI_RX_UDMA_CHAN | UDMA_PRI_SELECT,
954                                   UDMA_SIZE_8 | UDMA_SRC_INC_NONE | UDMA_DST_INC_8 | UDMA_ARB_4);
955
956
957         /* TX */
958         ROM_uDMAChannelAttributeDisable(SDC_SSI_TX_UDMA_CHAN,
959                                         UDMA_ATTR_ALTSELECT
960                                         | UDMA_ATTR_HIGH_PRIORITY
961                                         | UDMA_ATTR_REQMASK);
962         ROM_uDMAChannelControlSet(SDC_SSI_TX_UDMA_CHAN | UDMA_PRI_SELECT,
963                                   UDMA_SIZE_8 | UDMA_SRC_INC_NONE | UDMA_DST_INC_NONE | UDMA_ARB_4);
964     }
965
966     /* Clear SSI0 FIFO just to be safe */
967     while((HWREG(SSI0_BASE + SSI_O_SR) & SSI_SR_RNE))
968     {
969         uint32_t discard = HWREG(SSI0_BASE + SSI_O_DR);
970     }
971
972 }
```

D.3. Modultests

Quellcode D.19: Hauptprogramm des Modul-Tests: Inertialmesssystem

```
/*
 * main.c - Inertialmesssystem-Modultest
 *
 * Author: Felix Groth
 */

//Standard includes
#include <stdint.h>
#include <stdbool.h>

//Board includes
#include "inc/tm4c1294ncpdt.h"

//Driver includes
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"

//Module includes
#include "utils/uartstdio.h"
#include "imu/imu.h"

//Globals
unsigned int g_sysClock;
uint32_t g_relTimeMs = 0;

int main(void) {
    int j;
    uint8_t isCalibrated = 0;
    ImuData imuData;

    SysCtlVoltageEventConfig(SYSCTL_VEVENT_VDDABO_NONE | SYSCTL_VEVENT_VDDBO_NONE);

    g_sysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
            SYSCTL_OSC_MAIN |
            SYSCTL_USE_PLL |
            SYSCTL_CFG_VCO_480), 100000000);

    //Setup virtual serial port UART
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    GPIOPinConfigure(GPIO_PD4_U2RX);
    GPIOPinConfigure(GPIO_PD5_U2TX);
    GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5);
    UARTStdioConfig(2, 115200, g_sysClock);

    UARTprintf("Initializing IMU\t");

    if (imuInitI2C() == IMU_COM_READY) {
      imuInit(MPU_VAL_FS_250, MPU_VAL_AFS_2G);
    UARTprintf("[success]\n");
    } else {
      UARTprintf("[failed]\n");
      while(42)
        ;
    }

    while(42) {
      //Check for UART input telling us to recallibrate
      if (UARTCharsAvail(UART2_BASE))
        if(UARTCharGet(UART2_BASE) == 'c')
          isCalibrated = 0;

    imuPollMeasurement(&imuData);

    //Output data header
    for (j=0; j<14; j++)
      UARTCharPut(UART2_BASE, 0xAA);

    UARTCharPut(UART2_BASE, (uint8_t) (imuData.gyr.z & 0xff));
    UARTCharPut(UART2_BASE, (uint8_t) (imuData.gyr.z >> 8));
    UARTCharPut(UART2_BASE, (uint8_t) (imuData.gyr.y & 0xff));
    UARTCharPut(UART2_BASE, (uint8_t) (imuData.gyr.y >> 8));
    UARTCharPut(UART2_BASE, (uint8_t) (imuData.gyr.x & 0xff));
    UARTCharPut(UART2_BASE, (uint8_t) (imuData.gyr.x >> 8));

    UARTCharPut(UART2_BASE, (uint8_t) (imuData.tmp.meas & 0xff));
    UARTCharPut(UART2_BASE, (uint8_t) (imuData.tmp.meas >> 8));

    UARTCharPut(UART2_BASE, (uint8_t) (imuData.acc.z & 0xff));
    UARTCharPut(UART2_BASE, (uint8_t) (imuData.acc.z >> 8));
    UARTCharPut(UART2_BASE, (uint8_t) (imuData.acc.y & 0xff));
    UARTCharPut(UART2_BASE, (uint8_t) (imuData.acc.y >> 8));
    UARTCharPut(UART2_BASE, (uint8_t) (imuData.acc.x & 0xff));
    UARTCharPut(UART2_BASE, (uint8_t) (imuData.acc.x >> 8));
```

```
85
86      //Reduce output rate -- this is non critical
87      for (j=0; j<100000; j++)
88          ;
89      }
90  }
```

Quellcode D.20: Hauptprogramm des Modul-Tests: 1-Wire Temperatursensoren

```
1   /*
2    * main.c -- 1-Wire-Modultest
3    *
4    * Author: Felix Groth
5    */
6
7   #include <stdint.h>
8   #include <stdio.h>
9   #include <stdbool.h>
10
11  //----------inc----------
12  #include "inc/tm4c1294ncpdt.h"
13  #include "inc/hw_gpio.h"
14  #include "inc/hw_memmap.h"
15  #include "inc/hw_types.h"
16
17  #include "driverlib/sysctl.h"
18  #include "driverlib/rom.h"
19  #include "driverlib/gpio.h"
20  #include "driverlib/uart.h"
21  #include "driverlib/pin_map.h"
22
23  #include "utils/uartstdio.h"
24  #include "onewire/onewire.h"
25
26  //Globals
27  unsigned int g_sysClock;
28
29  int main(void) {
30      uint32_t i,j;
31      int16_t temperature;
32      double temperature_flt;
33      uint8_t temperature_string[10];
34      uint8_t deviceCount = 0;
35      maximROMCode onewireDevices[12];
36
37      SysCtlVoltageEventConfig(SYSCTL_VEVENT_VDDABO_NONE | SYSCTL_VEVENT_VDDBO_NONE);
38
39      g_sysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
40              SYSCTL_OSC_MAIN |
41              SYSCTL_USE_PLL |
42              SYSCTL_CFG_VCO_480), 120000000);
43
44      SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);
45      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
46      GPIOPinConfigure(GPIO_PD4_U2RX);
47      GPIOPinConfigure(GPIO_PD5_U2TX);
48      GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5);
49
50      UARTStdioConfig(2, 115200, g_sysClock);
51
52      UARTprintf("Initializing OneWire UART...\n");
53      initOneWireUART();
54
55    while(42) {
56      UARTprintf("Searching for OneWire devices...\n");
57      deviceCount = searchOneWire(onewireDevices, 12, 0);
58      UARTprintf("Found %i OneWire devices...\n", deviceCount);
59      for(i=0; i<deviceCount; i++) {
60        if (onewireDevices[i].family == ONEWIRE_DS18_FAMILY) {
61          UARTprintf("OneWire device Nr. %d is of type DS18B20 digital temperature sensor.\n", deviceCount);
62          UARTprintf("Starting temperature conversion on sensor %d...", i);
63          if (ds1820StartConversion(&(onewireDevices[i])) != ONEWIRE_INIT_SUCCESS)
64            UARTprintf("[failed]\n");
65          else
66            UARTprintf("[success]\n");
67
68          //Wait a long time for the conversion -- making this at least 750ms is critical!
69          for(j=0; j<10000000; j++)
70            ;
71
72          UARTprintf("Reading temperature measurement from sensor %d...", i);
```

```
73          if (ds1820ReadTemperature(&(onewireDevices[i]), &temperature) != ONEWIRE_INIT_SUCCESS)
74            UARTprintf("[failed]\n");
75          else
76            UARTprintf("[success]\n");
77
78          //Put values together and clear unneeded sign bits
79          //temperature[0] = (g_oneWireScratchpad[0] | g_oneWireScratchpad[1]<<8) & 0x87FF;
80          temperature_flt = temperature * 0.0625;
81
82          sprintf(temperature_string, "%4f", temperature_flt);
83
84          //Print entire scratchpad to UARTStdio
85          UARTprintf("Temperature at sensor %d: %s degC...\n",
86              i,
87              temperature_string);
88        } else
89          UARTprintf("OneWire device Nr. %d is of unknown type.\n", deviceCount);
90      }
91    }
92  }
```

Quellcode D.21: Hauptprogramm des Modul-Tests: Satellitenortung (GNSS)

```
1   /*
2    * main.c – GNSS-Modultest
3    *
4    * Author: Felix Groth
5    */
6
7   #include <stdint.h>
8   #include <stdio.h>
9   #include <stdbool.h>
10
11  //----------inc----------
12  #include "inc/tm4c1294ncpdt.h"
13  #include "inc/hw_gpio.h"
14  #include "inc/hw_memmap.h"
15
16  #include "driverlib/sysctl.h"
17  #include "driverlib/rom.h"
18  #include "driverlib/gpio.h"
19  #include "driverlib/uart.h"
20  #include "driverlib/pin_map.h"
21  #include "utils/uartstdio.h"
22
23  #include "gps/gps.h"
24
25  //Globals
26  unsigned int g_sysClock;
27  uint32_t g_relTimeMs = 0;
28
29  //External globals
30  //GPS uBlox buffer
31  extern volatile ubxPacketBufferRx* g_ubxPacketBufferRxActiveRead[2];
32  extern volatile ubxPacketBufferRx* g_ubxPacketBufferRxActiveWrite[2];
33  extern volatile ubxPacketBufferTx* g_ubxPacketBufferTxActiveRead[2];
34  extern volatile ubxPacketBufferTx* g_ubxPacketBufferTxActiveWrite[2];
35
36  void timepulseIsr(void) {
37    GPIOIntClear(GPS1_TIMEPULSE_BASE, GPS1_TIMEPULSE_INT_PIN);
38
39    // Check if GPS1 data is ready
40    switch (g_ubxPacketBufferRxActiveRead[gps1]->rxState) {
41    case ubxRxDataReady:
42      //Position information?
43      if ((g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->class == UBX_NAV) && (g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->id ↩
            == UBX_NAV_PVT)) {
44        UARTprintf("GNSS 1 data received:\n Lat: %d, Lon: %d, Number of sats: %d\n\n",
45            g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->ubxNavPvt.lat,
46            g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->ubxNavPvt.lon,
47            g_ubxPacketBufferRxActiveRead[gps1]->ubxPacket->ubxNavPvt.numSV);
48      }
49      ubxBufferRxPkgRead(gps1, g_ubxPacketBufferRxActiveRead[gps1]);
50      break;
51    case ubxRxError:
52      ubxBufferRxPkgRead(gps1, g_ubxPacketBufferRxActiveRead[gps1]);
53      UARTprintf("GNSS 1 data had CRC errors!\n");
54      break;
55    default:
56      break;
57    }
```

```

  // Check if GPS2 data is ready
  switch (g_ubxPacketBufferRxActiveRead[gps2]->rxState) {
  case ubxRxDataReady:
    //Position information?
    if ((g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->class == UBX_NAV) && (g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->id ↩
        == UBX_NAV_PVT)) {
      UARTprintf("GNSS 2 data received:\n Lat: %d, Lon: %d, Number of sats: %d\n\n",
          g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->ubxNavPvt.lat,
          g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->ubxNavPvt.lon,
          g_ubxPacketBufferRxActiveRead[gps2]->ubxPacket->ubxNavPvt.numSV);
    }
    ubxBufferRxPkgRead(gps2, g_ubxPacketBufferRxActiveRead[gps2]);
    break;
  case ubxRxError:
    ubxBufferRxPkgRead(gps2, g_ubxPacketBufferRxActiveRead[gps2]);
    UARTprintf("GNSS 2 data had CRC errors!\n");
    break;
  default:
    break;
  }
}

int main(void) {
    SysCtlVoltageEventConfig(SYSCTL_VEVENT_VDDABO_NONE | SYSCTL_VEVENT_VDDBO_NONE);

    g_sysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
            SYSCTL_OSC_MAIN |
            SYSCTL_USE_PLL |
            SYSCTL_CFG_VCO_480), 120000000);

    //Initialize the virtual serial port UART
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    GPIOPinConfigure(GPIO_PD4_U2RX);
    GPIOPinConfigure(GPIO_PD5_U2TX);
    GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5);

    UARTStdioConfig(2, 115200, g_sysClock);

    //Initialize GPS 1
    UARTprintf("Initializing GPS 1...\n");
    if (gpsInitModule(gps1, 460800, 4) == GPS_INIT_SUCCESS)
      UARTprintf("Successfully initialized GPS 1 with %d bit/s!\n", 460800);
    else
      UARTprintf("Initialization of GPS 1 failed!\n");

    //Initialize GPS 2
    UARTprintf("Initializing GPS 2...\n");
    if (gpsInitModule(gps2, 460800, 4) == GPS_INIT_SUCCESS)
      UARTprintf("Successfully initialized GPS 2 with %d bit/s!\n", 460800);
    else
      UARTprintf("Initialization of GPS 2 failed!\n");

    //Setup timepulse interrupt
  if (gpsTimepulseIntSetup(gps1, 1000, timepulseIsr, 2) != GPS_INIT_SUCCESS)
    UARTprintf("Initialization of GPS 1 timepulse failed!\n");
  else {
    ubxFilterClassSet(UBX_CFG);
    gpsTimepulseIntEnable();
  }

    //Forever
    while(42)
      ;
}
```

Quellcode D.22: Hauptprogramm des Modul-Tests: LCD-Touchscreen

```
/*
 * main.c -- LCD-Modultest
 *
 * Author: Felix Groth
 */

//Standard includes
#include <stdint.h>
#include <stdbool.h>

//Hardware includes
#include "inc/tm4c1294ncpdt.h"
```

```
13  //#include "inc/hw_gpio.h"
14
15  //Driver includes
16  #include "driverlib/sysctl.h"
17
18  //Module includes
19  #include "utils/uartstdio.h"
20
21  #include "grlib/grlib.h"
22  //#include "display/frame.h"
23  #include "display/Kentec320x240x16_ssd2119_8bit.h"
24  //#include "display/pinout.h"
25  //#include "display/touch.h"
26  #include "grlib/context.c"
27  #include "grlib/rectangle.c"
28
29  unsigned int g_sysClock;
30
31  int main(void) {
32    //Variables for basic LCD UI
33      tContext sContext;
34      tRectangle sRect;
35
36    //Set system clock to 120MHz
37      g_sysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
38              SYSCTL_OSC_MAIN |
39              SYSCTL_USE_PLL |
40              SYSCTL_CFG_VCO_480), 120000000);
41
42      //Configure UART for virtual serial port
43      UARTStdioConfig(2, 921600, g_sysClock);
44
45      SysCtlDelay(g_sysClock>>1);
46
47      //Initialize LCD
48      UARTprintf("Initializing LCD........[busy]");
49      Kentec320x240x16_SSD2119Init(g_sysClock);
50      SysCtlDelay(g_sysClock>>1);
51      // Initialize the graphics context.
52      GrContextInit(&sContext, &g_sKentec320x240x16_SSD2119);
53      // Fill the top 24 rows of the screen with blue to create the banner.
54      sRect.i16XMin = 0;
55      sRect.i16YMin = 0;
56      sRect.i16XMax = GrContextDpyWidthGet(&sContext) - 1;
57      sRect.i16YMax = 23;
58      GrContextForegroundSet(&sContext, ClrDarkBlue);
59      GrRectFill(&sContext, &sRect);
60      // Put a white box around the banner.
61      GrContextForegroundSet(&sContext, ClrWhite);
62      GrRectDraw(&sContext, &sRect);
63      // Put the application name in the middle of the banner.
64      GrContextFontSet(&sContext, g_psFontCm14);
65      GrStringDrawCentered(&sContext, "BATSEN DataLogger", -1,
66                           GrContextDpyWidthGet(&sContext) / 2, 10, 0);
67      UARTprintf("\b\b\b\bdone]\n");
68
69      while(42)
70        ;
71  }
```

Quellcode D.23: Hauptprogramm des Modul-Tests: Leuchtdioden

```
1   /*
2    * main.c - LED-Modultest
3    *
4    * Author: Felix Groth
5    */
6   //Standard includes
7   #include <stdint.h>
8   #include <stdbool.h>
9
10  //Hardware includes
11  #include "inc/tm4c1294ncpdt.h"
12  #include "inc/hw_gpio.h"
13
14  //Driver includes
15  #include "driverlib/sysctl.h"
16  #include "driverlib/rom.h"
17  #include "driverlib/gpio.h"
18  #include "driverlib/pin_map.h"
19
```

```
20  //Module includes
21  #include "utils/uartstdio.h"
22  #include "leds/leds.h"
23
24  //Globals
25  unsigned int g_sysClock;
26
27  int main(void) {
28    uint32_t i,j;
29
30      SysCtlVoltageEventConfig(SYSCTL_VEVENT_VDDABO_NONE | SYSCTL_VEVENT_VDDBO_NONE);
31
32      g_sysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
33              SYSCTL_OSC_MAIN |
34              SYSCTL_USE_PLL |
35              SYSCTL_CFG_VCO_480), 120000000);
36
37      SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);
38      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
39      GPIOPinConfigure(GPIO_PD4_U2RX);
40      GPIOPinConfigure(GPIO_PD5_U2TX);
41      GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5);
42
43      UARTStdioConfig(2, 9600, g_sysClock);
44
45      UARTprintf("Init LED ports and pins...[success]\n");
46      ledsInit();
47      ledsWrite(LED_RED | LED_GREEN | LED_YELLOW);
48
49      while(42) {
50        ledsChase(3000000);
51        ledsChase(3000000);
52        ledsChase(3000000);
53        ledsWrite(LED_RED | LED_GREEN | LED_YELLOW);
54        for (i=0; i<20; i++) {
55        ledsToggle(LED_GREEN);
56        for (j=0; j<1000000; j++)
57          ;
58        }
59      }
60  }
```

Quellcode D.24: Hauptprogramm des Modul-Tests: Taster

```
1   /*
2    * main.c - Taster-Modultest
3    *
4    * Author: Felix Groth
5    */
6
7   //Standard includes
8   #include <stdint.h>
9   #include <stdbool.h>
10
11  //Hardware includes
12  #include "inc/tm4c1294ncpdt.h"
13  #include "inc/hw_gpio.h"
14
15  //Driver includes
16  #include "driverlib/sysctl.h"
17  #include "driverlib/rom.h"
18  #include "driverlib/gpio.h"
19  #include "driverlib/pin_map.h"
20
21  //Module includes
22  #include "utils/uartstdio.h"
23  #include "buttons/buttons.h"
24
25
26  //Globals
27  unsigned int g_sysClock;
28
29
30  int main(void) {
31      SysCtlVoltageEventConfig(SYSCTL_VEVENT_VDDABO_NONE | SYSCTL_VEVENT_VDDBO_NONE);
32
33      g_sysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
34              SYSCTL_OSC_MAIN |
35              SYSCTL_USE_PLL |
36              SYSCTL_CFG_VCO_480), 120000000);
37
```

```
38      SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);
39      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
40      GPIOPinConfigure(GPIO_PD4_U2RX);
41      GPIOPinConfigure(GPIO_PD5_U2TX);
42      GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5);
43
44      UARTStdioConfig(2, 115200, g_sysClock);
45
46      pushButtonsInit();
47      UARTprintf("Going through buttons over and over again...\n");
48
49      while(42) {
50        UARTprintf("Please press button 1!\n");
51        while (pushButtonsRead() & BTN1_BIT)
52          ;
53        UARTprintf("Button 1 pressed!\n");
54        UARTprintf("Please press button 2!\n");
55        while (pushButtonsRead() & BTN2_BIT)
56            ;
57        UARTprintf("Button 2 pressed!\n");
58      }
59  }
```

Quellcode D.25: Hauptprogramm des Modul-Tests: GSM / GPRS

```
1   /*
2    * main.c - GSM-Modultest
3    *
4    * Author: Felix Groth
5    */
6
7   #include <stdint.h>
8   #include <stdio.h>
9   #include <stdbool.h>
10
11  #include "inc/tm4c1294ncpdt.h"
12  #include "inc/hw_gpio.h"
13  #include "inc/hw_memmap.h"
14
15  #include "driverlib/sysctl.h"
16  #include "driverlib/rom.h"
17  #include "driverlib/gpio.h"
18  #include "driverlib/uart.h"
19  #include "driverlib/pin_map.h"
20  #include "utils/uartstdio.h"
21
22  #include "gsm/gsm.h"
23
24  //Globals
25  unsigned int g_sysClock;
26
27  int main(void) {
28      g_sysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
29              SYSCTL_OSC_MAIN |
30              SYSCTL_USE_PLL |
31              SYSCTL_CFG_VCO_480), 120000000);
32
33      UARTStdioConfig(2, 115200, g_sysClock);
34
35      UARTprintf("Initializing GSM...\n");
36      gsmInit();
37      gsmWaitAutobaud();
38      UARTprintf("AutoBauding done...\n");
39
40
41      UARTprintf("Setting BaudRate to 115200...\n");
42      if(gsmRaiseBaudrate())
43        UARTprintf("Setting BaudRate failed...\n");
44      else
45        UARTprintf("Setting BaudRate successfull...\n");
46
47      while(42) {
48        //Check for RING, else just tunnel
49        if(UARTCharsAvail(UART0_BASE))
50          UARTprintf("%c", UARTCharGet(UART0_BASE));
51        if(UARTCharsAvail(UART2_BASE))
52          UARTCharPut(UART0_BASE, UARTCharGet(UART2_BASE));
53    }
54  }
```

Quellcode D.26: Hauptprogramm des Modul-Tests: MicroSD-Karte / FatFS

```
1  /*
2   * main.c - FatFS/SD-Modultest
3   *
4   * Author: Felix Groth
5   */
6
7  //Standard includes
8  #include <stdint.h>
9  #include <stdbool.h>
10
11 //Hardware includes
12 #include "inc/tm4c1294ncpdt.h"
13 #include "inc/hw_memmap.h"
14 #include "inc/hw_ssi.h"
15 #include "inc/hw_types.h"
16 #include "inc/hw_gpio.h"
17
18 //Driver includes
19 #include "driverlib/sysctl.h"
20 #include "driverlib/rom.h"
21 #include "driverlib/gpio.h"
22 #include "driverlib/pin_map.h"
23 #include "driverlib/ssi.h"
24 #include "driverlib/interrupt.h"
25 #include "driverlib/systick.h"
26
27 //Module includes
28 #include "utils/uartstdio.h"
29 #include "fatfs/src/ff.h"
30 #include "fatfs/src/diskio.h"
31 #include "matlab/mat4.h"
32
33 //Globals
34 unsigned int g_sysClock;
35 static FATFS g_sFatFs;
36 static FIL g_fileObject;
37
38 //*****************************************************************************
39 //
40 // This is the handler for this SysTick interrupt.  FatFs requires a timer tick
41 // every 10 ms for internal timing purposes.
42 //
43 //*****************************************************************************
44 void SysTickHandler(void)
45 {
46     //Call the FatFs tick timer.
47     disk_timerproc();
48 }
49
50 int main(void) {
51   int16_t i;
52   FRESULT iFResult;
53   mat4File test;
54
55     SysCtlVoltageEventConfig(SYSCTL_VEVENT_VDDABO_NONE | SYSCTL_VEVENT_VDDBO_NONE);
56
57     g_sysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
58             SYSCTL_OSC_MAIN |
59             SYSCTL_USE_PLL |
60             SYSCTL_CFG_VCO_480), 120000000);
61
62     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);
63     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
64     GPIOPinConfigure(GPIO_PD4_U2RX);
65     GPIOPinConfigure(GPIO_PD5_U2TX);
66     GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5);
67     UARTStdioConfig(2, 115200, g_sysClock);
68
69     //Configure SysTick for a 100Hz interrupt.  The FatFs driver wants a 10 ms tick.
70     SysTickPeriodSet(g_sysClock / 100);
71     SysTickEnable();
72     SysTickIntRegister(SysTickHandler);
73     SysTickIntEnable();
74     //Enable Interrupts
75     IntMasterEnable();
76
77     //Mount the file system, using logical disk 0.
78     UARTprintf("Trying to mount the SD card filesystem...[busy]");
79     iFResult = f_mount(0, &g_sFatFs);
80     if(iFResult != FR_OK) {
81         UARTprintf("\b\b\b\b\bfailed] - Error: %i\n", iFResult);
82         while(42)
83           ;
84     } else
```

```
85        UARTprintf("\b\b\b\b\bdone]\n");
86
87      UARTprintf("Creating file test.mat and writing 5000 to 1...[busy]");
88      if (matFileVectorCreate(&test, "test.mat", "test", 15000, MAT_TYPE_INT16) != FR_OK)
89        UARTprintf("\b\b\b\b\bfailed]\n");
90      else {
91      for (i=5000; i>0; i--)
92        matFileAddValue(&test, (uint8_t*)(&i));
93      UARTprintf("\b\b\b\b\bdone]\n");
94      }
95
96      //Open a file called "test.txt"
97      UARTprintf("Trying to mount open file test.txt...[busy]");
98      iFResult = f_open(&g_fileObject, "test.txt", FA_WRITE | FA_READ | FA_CREATE_ALWAYS);
99      if(iFResult != FR_OK)
100     {
101       UARTprintf("\b\b\b\b\bfailed] - Error: %i\n", iFResult);
102         while(42)
103           ;
104     } else
105       UARTprintf("\b\b\b\b\bdone]\n");
106
107     //Print "Test\r\n" to the file
108     UARTprintf("Trying to write Test\\r\\n to the file...[busy]");
109     i = f_puts("Test\r\n", &g_fileObject);
110     if (i == 6)
111       UARTprintf("\b\b\b\b\bdone]\n");
112     else
113       UARTprintf("\b\b\b\b\bfailed] - Wrote %i bytes\n", i);
114     UARTprintf("f_puts error: %i\n", i);
115
116     //Close the file
117     UARTprintf("Closing the file...[busy]");
118     f_close(&g_fileObject);
119     if(iFResult != FR_OK)
120     {
121       UARTprintf("\b\b\b\b\bfailed] - Error: %i\n", iFResult);
122         while(42)
123           ;
124     } else
125       UARTprintf("\b\b\b\b\bdone]\n");
126
127
128     UARTprintf("Writing 0 to 5999 to test.mat and closing...[busy]");
129     for (i=0; i<6000; i++)
130       matFileAddValue(&test, (uint8_t*)(&i));
131     matFileClose(&test);
132     UARTprintf("\b\b\b\b\bdone]\n");
133
134     UARTprintf("test.unit.sd done\n");
135
136     while(42)
137       ;
138 }
```

Quellcode D.27: Hauptprogramm zur Messung: GNSS-Jitter

```
1   /*
2    * main.c -- GNSS-Jitter-Messung
3    *
4    * Author: Felix Groth
5    */
6   #include <stdint.h>
7   #include <stdio.h>
8   #include <stdbool.h>
9
10  //----------inc----------
11  #include "inc/tm4c1294ncpdt.h"
12  #include "inc/hw_gpio.h"
13  #include "inc/hw_memmap.h"
14  //#include "inc/hw_ints.h"
15
16  #include "driverlib/sysctl.h"
17  #include "driverlib/rom.h"
18  #include "driverlib/gpio.h"
19  #include "driverlib/uart.h"
20  #include "driverlib/pin_map.h"
21  #include "driverlib/interrupt.h"
22
23  #include "gps/gps.h"
24  #include "utils/uartstdio.h"
```

```
25  #include "leds/leds.h"
26
27  //Globals
28  unsigned int g_sysClock;
29  uint32_t g_relTimeMs = 0;
30
31  //External globals
32  //GPS uBlox buffer
33  extern volatile ubxPacketBufferRx* g_ubxPacketBufferRxActiveRead[2];
34  extern volatile ubxPacketBufferRx* g_ubxPacketBufferRxActiveWrite[2];
35  extern volatile ubxPacketBufferTx* g_ubxPacketBufferTxActiveRead[2];
36  extern volatile ubxPacketBufferTx* g_ubxPacketBufferTxActiveWrite[2];
37
38  void none(void) {
39    GPIOIntClear(GPS1_TIMEPULSE_BASE, GPS1_TIMEPULSE_INT_PIN);
40  }
41
42  void gpsMessageISR(void) {
43    int i;
44    GPIOPinWrite(LED1_PORT_BASE, LED1_PIN, LED1_PIN);
45    for (i=0; i< 100000; i++)
46      ;
47    GPIOPinWrite(LED1_PORT_BASE, LED1_PIN, 0);
48    while(UARTCharGetNonBlocking(UART3_BASE) != -1)
49      ;
50    //Clear interrupt flag
51    UARTIntClear(UART3_BASE, UART_INT_RX);
52  }
53
54  int main(void) {
55      SysCtlVoltageEventConfig(SYSCTL_VEVENT_VDDABO_NONE | SYSCTL_VEVENT_VDDBO_NONE);
56
57      g_sysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
58              SYSCTL_OSC_MAIN |
59              SYSCTL_USE_PLL |
60              SYSCTL_CFG_VCO_480), 120000000);
61
62      SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);
63      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
64      GPIOPinConfigure(GPIO_PD4_U2RX);
65      GPIOPinConfigure(GPIO_PD5_U2TX);
66      GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5);
67
68      UARTStdioConfig(2, 115200, g_sysClock);
69
70      SysCtlPeripheralEnable(LED1_PORT_SYSCTL);
71      GPIOPinTypeGPIOOutput(LED1_PORT_BASE, LED1_PIN);
72
73      //Initialize GPS 1
74      UARTprintf("Initializing GPS1........[busy]");
75      if (gpsInitModule(gps1, GPS_BAUDRATE, 4) == GPS_INIT_SUCCESS)
76        UARTprintf("\b\b\b\b\bdone]\n");
77      else
78        UARTprintf("\b\b\b\b\bfailed]\n");
79
80      gpsConfigTiempulseInt(gps1, 100000, none);
81
82      //Enable UART3 (GPS1-UART) interrupt for RX
83      IntEnable(INT_UART3);
84      UARTIntEnable(UART3_BASE, UART_INT_RX);
85      IntMasterEnable();
86
87      while(42)
88        ;
89  }
```

D.4. PC-Anwendungen

Quellcode D.28: PC-Anwendung „DataLogger Data Extractor“: Hauptfenster

```
1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3  #include <QLabel>
4  #include <RS232Worker.h>
5
6  MainWindow::MainWindow(QWidget *parent) :
7      QMainWindow(parent),
```

```
8       ui(new Ui::MainWindow)
9   {
10      ui->setupUi(this);
11
12      int i;
13      QStringList comPorts;
14      for (i=0; i<30; i++)
15          comPorts.append(QString("COM%1").arg(i+1));
16
17      this->findChild<QComboBox *>("cmbComPort")->addItems(comPorts);
18  }
19
20  void MainWindow::setFileSizeSum(unsigned int sizeAll) {
21      QLabel * allBytesLabel = this->findChild<QLabel *>("allBytesLabel");
22      QProgressBar * transferProgress = this->findChild<QProgressBar *>("transferProgress");
23
24      allBytesLabel->setText(QString("0/%1 Bytes").arg(sizeAll));
25      transferProgress->setMaximum(sizeAll);
26  }
27
28  void MainWindow::setFile(QString fileName, unsigned int filesize) {
29      QLabel * fileBytesLabel = this->findChild<QLabel *>("fileBytesLabel");
30      QProgressBar * fileProgress = this->findChild<QProgressBar *>("fileProgress");
31      QLabel * filenameLabel = this->findChild<QLabel *>("filenameLabel");
32
33      fileBytesLabel->setText(QString("0/%1 Bytes").arg(filesize));
34      fileProgress->setMaximum(filesize);
35      filenameLabel->setText(fileName);
36  }
37
38  void MainWindow::setPositions(unsigned int pos, unsigned int posAll) {
39      QLabel * allBytesLabel = this->findChild<QLabel *>("allBytesLabel");
40      QLabel * fileBytesLabel = this->findChild<QLabel *>("fileBytesLabel");
41      QProgressBar * fileProgress = this->findChild<QProgressBar *>("fileProgress");
42      QProgressBar * transferProgress = this->findChild<QProgressBar *>("transferProgress");
43
44      allBytesLabel->setText(QString("%1/%2 Bytes").arg(posAll).arg(transferProgress->maximum()));
45      fileBytesLabel->setText(QString("%1/%2 Bytes").arg(pos).arg(fileProgress->maximum()));
46      fileProgress->setValue(pos);
47      transferProgress->setValue(posAll);
48  }
49
50  void MainWindow::toggleCom() {
51      QPushButton * btnComCtl = this->findChild<QPushButton *>("btnComCtl");
52      QPushButton * dataReadBtn = this->findChild<QPushButton *>("dataReadBtn");
53      QPushButton * dataDeleteBtn = this->findChild<QPushButton *>("dataDeleteBtn");
54      QComboBox * cmbComPort = this->findChild<QComboBox *>("cmbComPort");
55      QStatusBar * statusBar = this->findChild<QStatusBar *>("statusBar");
56      QLineEdit * dataPathEdit = this->findChild<QLineEdit *>("dataPathEdit");
57
58      if(!btnComCtl->text().compare("Connect")) {
59          this->rs232workerThread = new RS232Worker(cmbComPort->currentIndex(),this->baudRate);
60          if (this->rs232workerThread->openComPort()) {
61              this->rs232workerThread = NULL;
62              return;
63          }
64          this->rs232workerThread->dataPath = dataPathEdit->text();
65          btnComCtl->setText("Disconnect");
66          cmbComPort->setEnabled(false);
67          dataReadBtn->setEnabled(true);
68          dataDeleteBtn->setEnabled(true);
69          statusBar->showMessage("Connected");
70
71          QObject::connect(this->rs232workerThread, SIGNAL(updateSizeAll(unsigned int)), this, SLOT(setFileSizeSum(unsigned int)), ↩
                Qt::QueuedConnection);
72          QObject::connect(this->rs232workerThread, SIGNAL(updatePos(unsigned int, unsigned int)), this, SLOT(setPositions(unsigned ↩
                int, unsigned int)), Qt::QueuedConnection);
73          QObject::connect(this->rs232workerThread, SIGNAL(updateFile(QString, unsigned int)), this, SLOT(setFile(QString, unsigned ↩
                int)), Qt::QueuedConnection);
74          QObject::connect(dataReadBtn, SIGNAL(clicked()), this->rs232workerThread, SLOT(startTransfer()), Qt::QueuedConnection);
75          QObject::connect(dataDeleteBtn, SIGNAL(clicked()), this->rs232workerThread, SLOT(deleteAll()), Qt::QueuedConnection);
76          this->rs232workerThread->start();
77      } else if(!btnComCtl->text().compare("Disconnect")) {
78          QObject::disconnect(this->rs232workerThread, SIGNAL(updateSizeAll(unsigned int)), this, SLOT(setFileSizeSum(unsigned int)));
79          QObject::disconnect(this->rs232workerThread, SIGNAL(updatePos(unsigned int, unsigned int)), this, ↩
                SLOT(setPositions(unsigned int, unsigned int)));
80          QObject::disconnect(this->rs232workerThread, SIGNAL(updateFile(QString, unsigned int)), this, SLOT(setFile(QString, ↩
                unsigned int)));
81          QObject::disconnect(dataReadBtn, SIGNAL(clicked()), this->rs232workerThread, SLOT(startTransfer()));
82          QObject::disconnect(dataDeleteBtn, SIGNAL(clicked()), this->rs232workerThread, SLOT(deleteAll()));
83          this->rs232workerThread->quit();
84          this->rs232workerThread->terminate();
85          this->rs232workerThread->wait();
86          btnComCtl->setText("Connect");
87          cmbComPort->setEnabled(true);
```

```
88          dataReadBtn->setEnabled(false);
89          dataDeleteBtn->setEnabled(false);
90          statusBar->showMessage("Disconnected");
91          this->rs232workerThread = NULL;
92      }
93  }
94
95  MainWindow::~MainWindow()
96  {
97      delete ui;
98  }
```

Quellcode D.29: PC-Anwendung „DataLogger Data Extractor“: RS232-Thread

```
1   #include "RS232Worker.h"
2   #include "rs232.h"
3   #include <QSharedPointer>
4   #include <QLabel>
5   #include <QDir>
6   #include <QFile>
7   #include <QMessageBox>
8   #include <QDebug>
9   #include <stdint.h>
10
11  RS232Worker::RS232Worker(int comPort, int baudRate)
12  {
13      this->comPort = comPort;
14      this->baudRate = baudRate;
15      this->bailOut = 0;
16      this->comStatus = 1;
17      this->allDone = 0;
18  }
19
20  int RS232Worker::openComPort() {
21      int result;
22
23      result = (this->comStatus = RS232_OpenComport(this->comPort, this->baudRate));
24
25      //Put system into file transfer mode
26      if (!this->comStatus)
27          RS232_SendByte(this->comPort, 't');
28
29      return result;
30  }
31
32  void RS232Worker::run()
33  {
34      unsigned int i, j;
35      unsigned char buffer[1024];
36      int headerRead = 0;
37      //int dataRead = 0;
38      int count;
39      unsigned int fileSize, fileSizeCounter;
40      unsigned char fileCount, folderCount;
41      QString fileName, folderName;
42      unsigned char nameLength;
43      unsigned int fileBytesReceived = 0;
44      unsigned int bytesReceived = 0;
45      //QFile* outputFile;
46      bool skipFile = false;
47      QDir directory(dataPath);
48
49      if (this->comStatus)
50          return;
51
52      while(!this->bailOut) {
53          headerRead = 0;
54
55          while ((headerRead < 12) && !this->bailOut) {
56              count = RS232_PollComport(this->comPort, buffer+headerRead, 1);
57              if (count == 1) {
58                  qDebug() << "Receiving as header: " << *(buffer+headerRead);
59                  if (headerRead % 2) {
60                      if (*(buffer+headerRead) == 0xAA) {
61                          headerRead++;
62                          qDebug() << "Got header byte: " << headerRead;
63                      }
64                  } else {
65                      if (*(buffer+headerRead) == 0xFF) {
66                          headerRead++;
67                          qDebug() << "Got header byte: " << headerRead;
```

```
68                      }
69                  }
70              }
71          }
72
73          qDebug() << "Header recognized and received.";
74
75          if (this->bailOut)
76              break;
77
78          qDebug() << "Receiving data now.";
79          bytesReceived = 0;
80
81          //Read size summary of all files
82          count = 0;
83          while (count < 4)
84              count += RS232_PollComport(this->comPort, buffer+count, 4-count);
85          emit(updateSizeAll(*((unsigned int*)buffer)));
86          //Read folder count
87          count = 0;
88          while (count < 1)
89              count += RS232_PollComport(this->comPort, buffer, 1);
90          folderCount = *(buffer);
91
92          for (i=0; i<folderCount; i++) {
93              //Read length of directory name
94              count = 0;
95              while (count < 1)
96                  count += RS232_PollComport(this->comPort, buffer, 1);
97              nameLength = *(buffer);
98
99              //Read directory name
100             count = 0;
101             while (count < nameLength)
102                 count += RS232_PollComport(this->comPort, buffer+count, nameLength-count);
103             buffer[nameLength] = '\0';
104             folderName = QString((const char*)buffer);
105
106             qDebug() << "Receiving folder: " << folderName;
107
108             //Create directory
109             directory.mkdir(folderName);
110
111             //Read file count
112             count = 0;
113             while (count < 1)
114                 count += RS232_PollComport(this->comPort, buffer, 1);
115             fileCount = *(buffer);
116
117             for (j=0; j<fileCount; j++) {
118                 //Read length of file name
119                 count = 0;
120                 while (count < 1)
121                     count += RS232_PollComport(this->comPort, buffer, 1);
122                 nameLength = *(buffer);
123
124                 //Read file name
125                 count = 0;
126                 while (count < nameLength)
127                     count += RS232_PollComport(this->comPort, buffer+count, nameLength-count);
128                 buffer[nameLength] = '\0';
129                 fileName = QString((const char*)buffer);
130
131                 //Read file size
132                 count = 0;
133                 while (count < 4)
134                     count += RS232_PollComport(this->comPort, buffer+count, 4-count);
135                 fileSize = *((unsigned int*)buffer);
136                 emit(updateFile(folderName+QString("/")+fileName, *((unsigned int*)buffer)));
137
138                 //Receive file content
139                 fileBytesReceived = 0;
140                 this->outputFile = new QFile(dataPath+QString("/")+folderName+QString("/")+fileName);
141                 if (!this->outputFile->open(QIODevice::WriteOnly)) {
142                     skipFile = true;
143                     qDebug() << "Could not open: " << fileName;
144                 } else
145                     skipFile = false;
146
147                 fileSizeCounter = fileSize;
148                 while (fileSize) {
149                     count = 0;
150                     while (count < 1) {
151                         if (fileSize >= 1024)
152                             count = RS232_PollComport(this->comPort, buffer, 1024);
```

```
153                         else
154                             count = RS232_PollComport(this->comPort, buffer, fileSize);
155                     }
156                     if (!skipFile)
157                         this->outputFile->write((const char*)buffer, count);
158                     fileBytesReceived += count;
159                     bytesReceived += count;
160                     fileSize -= count;
161                     emit(updatePos(fileBytesReceived, bytesReceived));
162                 }
163                 //bytesReceived += fileSize;
164                 //emit(updatePos(fileBytesReceived, bytesReceived));
165                 if (!skipFile)
166                     this->outputFile->close();
167                 delete(this->outputFile);
168             }
169         }
170     }
171     this->allDone = 1;
172     return;
173 }
174
175 int RS232Worker::startTransfer() {
176     if (this->comStatus)
177         return 0;
178
179     RS232_SendByte(this->comPort, 'g');
180     return 1;
181 }
182
183 int RS232Worker::deleteAll() {
184     if (this->comStatus)
185         return 0;
186
187     RS232_SendByte(this->comPort, 'd');
188     return 1;
189 }
190
191 int RS232Worker::stopTransfer() {
192     if (this->comStatus)
193         return 0;
194
195     RS232_SendByte(this->comPort, 's');
196     return 1;
197 }
198
199 void RS232Worker::quit() {
200     this->bailOut = 1;
201
202     this->stopTransfer();
203     RS232_CloseComport(this->comPort);
204 }
```

Quellcode D.30: PC-Anwendung „MPU9250 Raw Data“: Hauptfenster

```
1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3  #include <QLabel>
4  #include <RS232Worker.h>
5
6  MainWindow::MainWindow(QWidget *parent) :
7      QMainWindow(parent),
8      ui(new Ui::MainWindow)
9  {
10     ui->setupUi(this);
11
12     int i;
13     QStringList comPorts;
14     for (i=0; i<30; i++)
15         comPorts.append(QString("COM%1").arg(i+1));
16
17     this->findChild<QComboBox *>("cmbComPort")->addItems(comPorts);
18 }
19
20 void MainWindow::setText(short gx, short gy, short gz, short ax, short ay, short az, short temperature) {
21     QString buffer;
22     double outputTemperature;
23     QProgressBar * prgrXGyr = this->findChild<QProgressBar *>("prgrXGyr");
24     QProgressBar * prgrYGyr = this->findChild<QProgressBar *>("prgrYGyr");
25     QProgressBar * prgrZGyr = this->findChild<QProgressBar *>("prgrZGyr");
26     QProgressBar * prgrXAcc = this->findChild<QProgressBar *>("prgrXAcc");
```

```
27      QProgressBar * prgrYAcc = this->findChild<QProgressBar *>("prgrYAcc");
28      QProgressBar * prgrZAcc = this->findChild<QProgressBar *>("prgrZAcc");
29
30      prgrXGyr->setValue(gx);
31      prgrYGyr->setValue(gy);
32      prgrZGyr->setValue(gz);
33      prgrXAcc->setValue(ax);
34      prgrYAcc->setValue(ay);
35      prgrZAcc->setValue(az);
36
37      //Calculate and output decimal temperature
38      outputTemperature = ((double)(temperature - MPU_TEMP_OFFSET)/MPU_TEMP_RES) + MPU_TEMP_STD;
39      buffer = QString::number(outputTemperature);
40      this->findChild<QLabel *>("tempValueLbl")->setText(buffer.append(" degC"));
41  }
42
43  void MainWindow::toggleCom() {
44      QPushButton * btnComCtl = this->findChild<QPushButton *>("btnComCtl");
45      QPushButton * btnCalibrate = this->findChild<QPushButton *>("btnCalibrate");
46      QComboBox * cmbComPort = this->findChild<QComboBox *>("cmbComPort");
47      QStatusBar * statusBar = this->findChild<QStatusBar *>("statusBar");
48
49      if(!btnComCtl->text().compare("Start")) {
50          this->rs232workerThread = new RS232Worker(cmbComPort->currentIndex(),this->baudRate);
51          if (this->rs232workerThread->openComPort()) {
52              this->rs232workerThread = NULL;
53              return;
54          }
55          btnComCtl->setText("Stop");
56          cmbComPort->setEnabled(false);
57          btnCalibrate->setEnabled(true);
58          statusBar->showMessage("RS232 Connected");
59          QObject::connect(this->rs232workerThread, SIGNAL(updateData(short,short,short,short,short,short,short)), this, ↩
                SLOT(setText(short,short,short,short,short,short,short)), Qt::QueuedConnection);
60          QObject::connect(btnCalibrate, SIGNAL(clicked()), this->rs232workerThread, SLOT(reCalibrate()), Qt::QueuedConnection);
61          this->rs232workerThread->setOutputDiv(1);
62          this->rs232workerThread->reCalibrate();
63          this->rs232workerThread->start();
64      } else {
65          QObject::disconnect(this->rs232workerThread, SIGNAL(updateData(short,short,short,short,short,short,short)), this, ↩
                SLOT(setText(short,short,short,short,short,short,short)));
66          QObject::disconnect(btnCalibrate, SIGNAL(clicked()), this->rs232workerThread, SLOT(reCalibrate()));
67          this->rs232workerThread->quit();
68          this->rs232workerThread->wait();
69          btnComCtl->setText("Start");
70          cmbComPort->setDisabled(false);
71          btnCalibrate->setEnabled(false);
72          statusBar->showMessage("RS232 Disconnected");
73          this->rs232workerThread = NULL;
74      }
75  }
76
77  MainWindow::~MainWindow()
78  {
79      delete ui;
80  }
```

Quellcode D.31: PC-Anwendung „MPU9250 Raw Data“: RS232-Thread

```
1   #include "RS232Worker.h"
2   #include "rs232.h"
3   #include <QSharedPointer>
4   #include <QLabel>
5   #include <QMessageBox>
6   #include <QDebug>
7   #include <stdint.h>
8
9   RS232Worker::RS232Worker(int comPort, int baudRate)
10  {
11      this->comPort = comPort;
12      this->baudRate = baudRate;
13      this->bailOut = 0;
14      this->comStatus = 1;
15      this->allDone = 0;
16  }
17
18  int RS232Worker::openComPort() {
19      return (this->comStatus = RS232_OpenComport(this->comPort, this->baudRate));
20  }
21
22  int RS232Worker::setOutputDiv(unsigned char div) {
```

```
23      if (this->comStatus)
24          return 0;
25
26      unsigned char buffer[6];
27      RS232_SendByte(this->comPort, 'd');
28      while(RS232_PollComport(this->comPort, buffer, 1) == 1)
29          ;
30      RS232_SendByte(this->comPort, div);
31      while(RS232_PollComport(this->comPort, buffer, 1) == 0)
32          ;
33      return (int)*(uint8_t*)buffer;
34  }
35
36  void RS232Worker::run()
37  {
38      unsigned char buffer[6];
39      int headerRead = 0;
40      int dataRead = 0;
41      int count;
42
43      if (this->comStatus)
44          return;
45
46      int displayCount = 0;
47      while(!this->bailOut) {
48          while (headerRead < 14) {
49              count = RS232_PollComport(this->comPort, buffer+headerRead, 1);
50              if ((count == 1) && (*(buffer+headerRead) == IMU_HEADER_BYTE))
51                  headerRead++;
52              else if(count == 1) {
53                  qDebug() << "Lost track" << displayCount;
54                  displayCount++;
55                  headerRead=0;
56              }
57          }
58
59          while (dataRead < 14) {
60              count = RS232_PollComport(this->comPort, buffer+dataRead, 1);
61              if (count == 1)
62                  dataRead++;
63          }
64          emit(updateData( *(short*)(buffer+4), *(short*)(buffer+2), *(short*)buffer, *(short*)(buffer+12), *(short*)(buffer+10), ↩
                *(short*)(buffer+8), *(short*)(buffer+6)) );
65          headerRead = 0;
66          dataRead = 0;
67      }
68      this->allDone = 1;
69      return;
70  }
71
72  int RS232Worker::reCalibrate() {
73      if (this->comStatus)
74          return 0;
75
76      RS232_SendByte(this->comPort, 'c');
77      return 1;
78  }
79
80  void RS232Worker::quit() {
81      this->bailOut = 1;
82      while(!this->allDone)
83          ;
84      RS232_CloseComport(this->comPort);
85  }
```

Versicherung über die Selbstständigkeit

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

Hamburg, 15. Mai 2015	
Ort, Datum	Unterschrift