PG 522: AutoLab

Endbericht

Projektgruppe 522

AutoLab

- Eine Experimentierplattform für automotive Softwareentwicklung -

http://ess.cs.tu-dortmund.de/Teaching/PGs/autolab

Wintersemester 2007/2008 und Sommersemester 2008

Teilnehmer:
Alexander Berger, Oliver Botschkowski, Stephan Braun, Sabrina Hecke, Florian Hohnsbehn, Christian Horn, Gregor Kaleta, Boris Konrad, Sebastian Kosch, Matthias Meier, Robert Neue, Thomas Romanek

Betreuer:
Dr. Michael Engel, Horst Schirmeier, Jochen Streicher

Technische Universität Dortmund
Fachbereich Informatik
LSXII
Prof. Dr.-Ing. Olaf Spinczyk

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Moderne Kraftfahrzeuge sind heterogene, verteilte, eingebettete Rechnersysteme auf Rädern. Der in Abbildung 1.1 gezeigte Maybach verfügt beispielsweise über 76 Steuergeräte (engl. Electronic Control Units ECUs) auf Basis von 8-32 Bit Mikrocontrollern, die über nicht weniger als fünf Bussysteme miteinander vernetzt sind. Man geht heute davon aus, dass etwa 40 Prozent der Produktionskosten eines Autos in die Fahrzeugelektronik und die auf den Steuergeräten laufende Software fließen [Bro06].
Der Grund für diese enorm großen und stetig wachsenden Aufwendungen liegt in der immer größer werdenden Bedeutung moderner Fahrzeugfunktionen, die ohne Einsatz von Software technisch nicht mehr beherrschbar wären. Typische Beispiele sind über mehrere Steuergeräte verteilt realisierte Fahrzeugfunktionen wie die elektronische Parkbremse oder der automatische Abstandsregeltempomat. Man schätzt, dass Fahrzeuge der Premium-Klasse heute mit Software im Umfang von etwa 10.000.000 Programmzeilen ausgestattet sind. Seit dem Einzug von Software ins Automobil vor ca. 30 Jahren stieg diese Zahl exponentiell, und der Trend wird wohl auch in den nächsten Jahren noch anhalten.
Obwohl in der Summe etliche Megabytes an Software im Automobil bewegt werden, dürfen keine Ressourcen verschwendet werden. Um dem hohen Konkurrenz- und damit Kostendruck in der Automobilbranche Rechnung zu tragen, wird daher dedizierte Hardware eingesetzt und die Systemsoftware anwendungsspezifisch konfiguriert. Zudem werden typischerweise maschinennahe Programmiersprachen wie C oder Assembler eingesetzt.
Die kontinuierlich steigende Komplexität der Software im Kraftfahrzeug und die oben genannten zusätzlichen Erschwernisse, die für heterogene, verteilte, eingebettete Systeme typisch sind, stellen die Automobilhersteller und ihre Zulieferer vor große Herausforderungen. Ohne eine Neuorientierung bezüglich der eingesetzten Entwicklungsprozesse, Entwicklungswerkzeuge, Programmiersprachen und Testverfahren wird es wohl kaum möglich sein, die derzeit stetig wachsende Zahl der Pannen, die durch Elektronik und insbesondere Softwarefehler verursacht werden, in den Griff zu bekommen. [Dud04]

1.2 Wissenschaftlicher Kontext

Inzwischen hat sich Automotive Software Engineering als eigenständiges Forschungsgebiet herausgebildet, welches bereits einige Ansätze zur Bewältigung dieser Probleme hervorgebracht hat

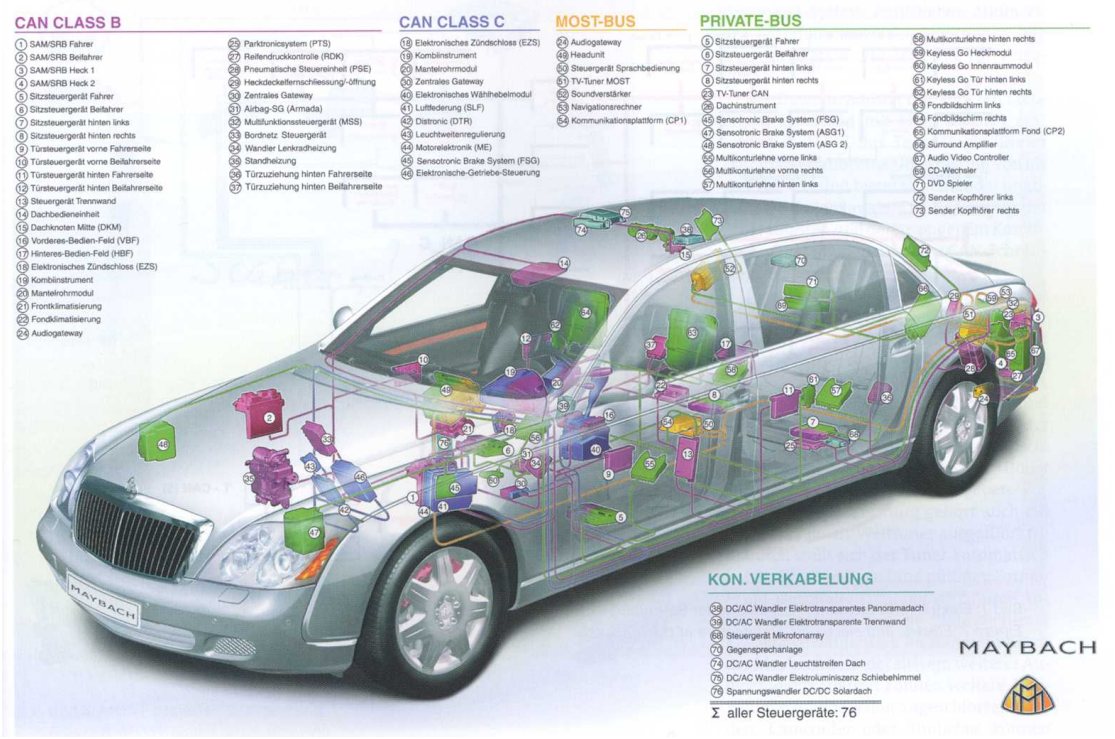


Abbildung 1.1: Vernetzte Steuergeräte in einem modernen Kraftfahrzeug [DC02]

[MR06]. Zur Beherrschung der steigenden Komplexität werden neben etablierten Methoden der Dekomposition und Trennung der Belange auch neuere, modellbasierte Entwicklungsansätze untersucht [Bro06]. Der riesigen Variantenvielfalt automotiver Software und deren Konfigurierbarkeit versucht man durch systematische Wiederverwendung mit Hilfe von Software-Produktlinien [LMN04] Herr zu werden. Die neue Arbeitsgruppe „Eingebettete Systemsoftware“ ist in diesem Zusammenhang speziell im Bereich der automotiven Systemsoftware aktiv (z.B.[OS03]). Ziel ist es, neue Entwicklungsmethoden zu erarbeiten und zu bewerten, die es erlauben, hochgradig anwendungsspezifisch konfigurierbare Systemsoftware zu bauen, die trotz Wiederverwendbarkeit und Wartbarkeit die extrem harten Effizienzanforderungen erfüllt. Zum einen setzen wir dabei auf merkmalbasierte Produktableitung [OS06], einer Technik, die die systematische Entwicklung von Software-Produktlinien unterstützt. Ein weiterer wichtiger Ansatz ist die Aspektorientierte Programmierung [GK97] (mittels AspectC++ [OS05]), ein Entwurfs- und Implementierungsansatz, der eine bessere Modularisierung des Programmcodes durch Trennung der Belange erlaubt und so die Konfigurierung der Software erleichtert.

1.3 Ziele der PG

Um die eigenen Systementwicklungen besser bewerten zu können und generelle Probleme automotiver Systemsoftware zu untersuchen, soll innerhalb der nächsten Jahre ein Labor entstehen, für das die Projektgruppe AutoLab den Grundstein legen sollte. Ziel war dabei, einen realistischen Brettaufbau einer Fahrzeugelektronik vorzunehmen und diesen in eine Steuer- und Messumge-

bung einzubetten. Die folgenden Einzelziele konnten erreicht werden:

- Schaffung einer Entwicklungsplattform, die Analysen unter realistischen Lastverhältnissen erlaubt,
- Betrachtung des Fahrzeugs als Gesamtheit. Globale Gesichtspunkte wie Lastverteilung (u.a. bei der Flash-Programmierung), Energieverbrauch und Startzeiten könnten untersucht werden,
- Untersuchung alternativer Fahrzeugtopologien.

1.3.1 Vision

Heutzutage ist ein modernes Auto ein heterogenes, verteiltes, eingebettetes System auf Rädern. Infolgedessen sind die informatikbezogenen Forschungs- und Entwicklungsschwerpunkte vor allem in

- der Programmierung verteilter Fahrzeugfunktionen,
- der Analyse der Lastverteilung,
- der Minimierung des Stromverbrauchs,
- der Untersuchung von Echtzeitfragestellungen und
- der Maßschneiderung von Infrastruktur

zu sehen.

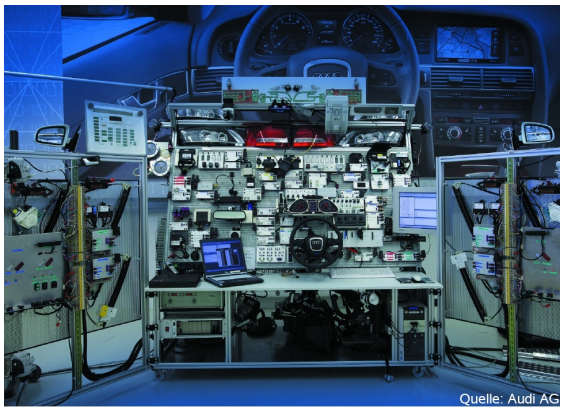


Abbildung 1.2: Brettaufbau bei Audi

Um diese Probleme untersuchen zu können, benötigt man einen möglichst realistischen Aufbau eines Fahrzeugnetzes (siehe Abbildung 1.2). Als Grundlage soll ein CAN-Bus-Netzwerk verwendet werden, über welches die Fahrzeugknoten des Netzes kommunizieren können. Langfristig sollen aber auch FlexRay-Netze, welche heutzutage die Regel im Antriebs- und Fahrwerksbereich sind, verwendet werden.
Das Fahrzeugnetz ist in eine Steuer- und Messumgebung eingebettet, mit dessen Hilfe das Fahrzeug programmiert, gestoppt, gestartet sowie beobachtet werden kann. Die Messumgebung dient

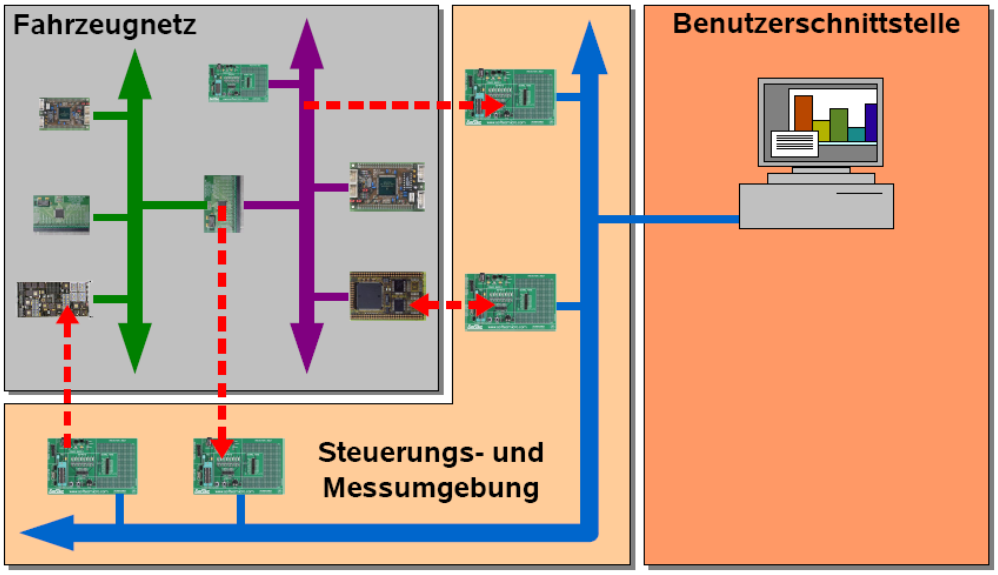


Abbildung 1.3: Fahrzeugnetz, Steuer- und Messumgebung, Benutzerschnittstelle

dazu, die Ausgabesignale der Steuergeräte und deren Energieverbrauch sowie Bus-Nachrichten zeitgenau zu erfassen. Des weiteren sollen Eingabesignalleitungen der Steuergeräte programmiert und Testabläufe zum Zwecke der Automatisierung aufgezeichnet werden.
Um mit dem System interagieren zu können, gibt es eine graphische Benutzerschnittstelle. In ihr werden Daten der Messumgebung visualisiert. Dies sind beispielsweise Daten, die Aufschluss über die Lastverteilung oder das Startup-Verhalten geben. Die Messdaten können aufgezeichnet, gespeichert und wiedergegeben werden. Auch die Steuerung des Systems ist ebenfalls über die Benutzerschnittstelle möglich.
Es ist vorgesehen, das System in der Lehre und in der Forschung einzusetzen. Sowohl Wissenschaftler der Technischen Universität Dortmund als auch externe Wissenschaftler sollen das System für ihre Forschung nutzen können. Um den externen Forschern die Nutzung des Systems, zu ermöglichen ohne physikalisch anwesend sein zu müssen, besteht die Möglichkeit, einen Remote-Zugriff über das Internet einzurichten. Dieser entfernte Zugriff ist im Rahmen der Projektgruppe aus zeitlichen Gründen nicht verwirklicht worden.
In der Lehre soll das System unter anderem in vorlesungsbegleitenden Übungen genutzt werden, in denen die Studierenden z.B. eigene Software für die Steuergeräte schreiben können.

1.3.2 Minimalziele

Die Minimalziele der Projektgruppe können in vier Kategorien unterteilt werden: die Erstellung eines grundlegenden Konzeptes, die Inbetriebnahme eines Versuchsaufbaus, die Entwicklung einer Steuer- und Messumgebung, sowie den Entwurf und die Realisierung eines Demonstrators.
Konkret sind somit die folgenden Ziele von der Projektgruppe zu realisieren:

1. **Erstellung eines Konzeptes:** Es soll ein detaillierter Entwurf
   - eines einfachen Fahrzeugelektronik-Versuchsaufbaus (Grundlage ist ein CAN-basiertes Karosserienetzwerk) *und*

- einer Steuerungs- und Messumgebung (Starten/Stoppen/Flash-Programmierung; zeitgenaues Erfassen von Änderungen an digitalen Ausgabeleitungen und CAN-Bus- Nachrichten)

erarbeitet werden.

2. **Inbetriebnahme eines Versuchsaufbaus:** Die vorhandenen Steuergeräte sollen miteinander vernetzt und mit Hilfe der dazugehörigen Entwicklungswerkzeuge und Standardsoftware in Betrieb genommen werden.
3. **Steuerungs- und Messumgebung:** Die entworfene Umgebung soll in Hard- und Software implementiert werden.
4. **Demonstrator:** Es soll eine geeignete Beispielanwendung für das zu entwickelnde Fahrzeugnetz entworfen und implementiert werden. Diese Beispielanwendung soll in der Lage sein, die Funktionstüchtigkeit des Fahrzeugnetz-Versuchsaufbaus und die zuvor spezifizierten Eigenschaften der Steuerungs-und Messumgebung zu demonstrieren.

1.4 Mitglieder

Auf Seiten des Lehrstuhls 12 wird diese Projektgruppe betreut von

- Prof. Dr.-Ing. Olaf Spinczyk
- Dr. Michael Engel
- Horst Schirmeier
- Jochen Streicher

Auf studentischer Seite besteht die Projektgruppe AutoLab aus

- Alexander Berger
- Oliver Botschkowski
- Stephan Braun
- Sabrina Hecke
- Florian Hohnsbehn
- Christian Horn
- Gregor Kaleta
- Boris Konrad
- Sebastian Kosch
- Matthias Meier
- Robert Neue
- Thomas Romanek

Alle Teilnehmer an der Projektgruppe sind zugleich gemeinschaftlich Autoren dieses Berichts.

1.5 Aufbau dieses Dokumentes

Neben dieser ersten Einführung (Kapitel 1) ist dieses Dokument in zwölf weitere Abschnitte gegliedert.
Zunächst wird die Organisation und der Ablauf des Projekts (Kapitel 2) näher erläutert. Anschließend erfolgt ein Überblick über das im Rahmen der Vorbereitung durchgeführte Seminar (Kapitel 3). Die aus den Zielen abgeleiteten Anforderungen an das Projekt werden im 4. Kapitel detailliert aufgeführt. Die von uns verwendete Hard- und Software sowie die eingesetzten Werkzeuge werden im 5. Kapitel (Grundlagen) beschrieben. Aufbauend auf diesen Grundlagen werden im 6. Kapitel die ersten Entwürfe für das Projekt vorgestellt. Im Anschluss an diese Entwürfe wird in Kapitel 7 der von uns erstellte erste Prototyp präsentiert. In Kapitel 8 wird der darauf aufbauende 2. Prototyp vorgestellt, bevor im 9. Kapitel der erste öffentliche Auftritt erläutert wird. Daran anschließend wird in Kapitel 10 das Endprodukt beschrieben. Der Abgleich des Erreichten mit den gesetzten Zielen findet im 11. Kapitel statt. Die im Laufe der Projektgruppe zu Tage getretenen allgemeinen Schwierigkeiten hinsichtlich der Durchführung und Planung eines Projekts werden in Kapitel 12 zusammengefasst. Das 13. Kapitel fasst noch einmal die Entwicklungsstufen zusammen, bewertet das Projektergebnis und liefert Anregungen für mögliche zukünftige Weiterentwicklungen.

Kapitel 2

Organisation und Ablauf des Projekts

Im Folgenden wird die Organisation und der zeitliche Ablauf der Projektgruppe näher erläutert.

2.1 Zeitraum und Umfang der PG

Die Projektgruppe AutoLab fand im Wintersemester 2007/2008 und im Sommersemester 2008 mit jeweils acht Semesterwochenstunden an der Technischen Universität Dortmund statt. Mit diesem Endbericht endet die Projektgruppe.
Um die erarbeiteten Ergebnisse zu präsentieren und das weitere Vorgehen abzustimmen, fanden wöchentliche Treffen statt. Die Inhalte dieser Treffen wurden schriftlich festgehalten und den Teilnehmern im Wiki in Form von Protokollen zur Verfügung gestellt. Die Moderation und Vorbereitung der Treffen fand durch die Studenten statt.

2.2 Infrastruktur

Zum Zwecke der Präsentation unserer Ergebnisse wurde ein Internetauftritt eingerichtet. Dieser ist unter `http://ess.cs.tu-dortmund.de/Teaching/PGs/autolab` zu finden.

Um eine interne Arbeitsplattform (zum Beispiel für Zwischenergebnisse, offene Fragen, etc.) für die Beteiligten der Projektgruppe zur Verfügung zu stellen, ist ein Wiki eingerichtet worden. Die Kommunikation der Teilnehmer und Betreuer untereinander wurde durch eine PG-Mailingliste erleichtert. Des weiteren wurde das Versionsverwaltungssystem Subversion eingeführt. Durch dieses Hilfsmittel ist es möglich, dass mehrere Personen gleichzeitig an ein und demselben Dokument (z.B. Quelltext) arbeiten können. Die Versionierung erfolgt in einem zentralen Projektarchiv. Die im Laufe der Arbeit entstehenden Änderungen der Inhalte werden dann in diesem Projektarchiv zusammengefügt und allen Teilnehmern angezeigt.

Den Teilnehmern der Projektgruppe wurde seitens des Lehrstuhls ein eigenes Labor mit mehreren Sun Rays, einem Linux-Notebook und einem Windowssystem zur Verfügung gestellt. In diesem Labor fanden ebenfalls die Hardware- und Softwarearbeiten statt. Das Labor bietet aufgrund

der bisher exklusiven Nutzung durch die Projektgruppe ebenfalls die Möglichkeit, den von uns entwickelten Demonstrator und die Steuerungs- und Messumgebung dauerhaft aufzubauen.

Abbildung 2.1: Die Arbeitsplätze im Labor

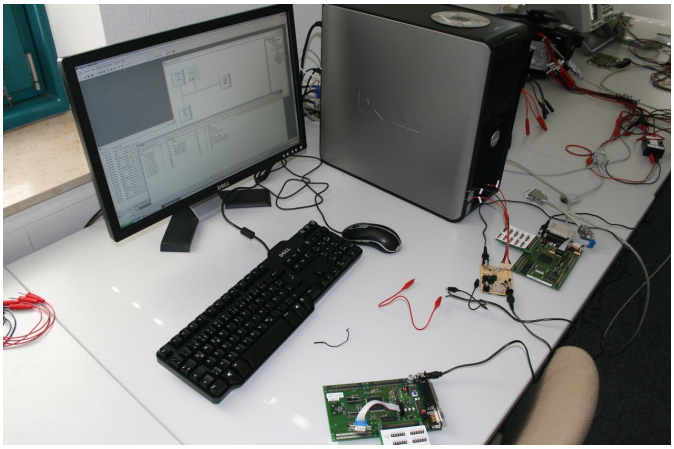


Abbildung 2.2: Der Windowsrechner

2.3 Zeitlicher Ablauf

Die Projektgruppe hat zu Beginn des Wintersemesters 2007/2008 ihre Arbeit aufgenommen. Begonnen wurde mit einer Seminarphase in Form einer Exkursion zum *„Naturfreundehaus am Hülsberg“* in Wuppertal-Cronenberg. Ziel dieser Seminarfahrt war es, das Kennenlernen der Teilnehmer untereinander zu fördern und gemeinschaftlich einen Einstieg in die Thematik zu finden. Dieser Einstieg erfolgte in Form von Vorträgen über verschiedene Aspekte, die sich aus der Zielformulierung der Projektgruppe ergaben. Eine Übersicht über die Seminarthemen findet sich in Kapitel 3. Der Rest des ersten Monats wurde zur Vorbereitung der Projektarbeit und organisatorischen Entscheidungen genutzt. In der ersten Gruppensitzung wurde die Gruppe in mehrere Kleingruppen unterteilt. Diese Teilgruppen befassten sich unter anderem mit dem Entwurf der Strommessung- und versorgung sowie der Einarbeitung in die zur Verfügung stehende Software wie z.B. CANoe. Die übrigen Teilnehmer nahmen sich weiterer organisatorischer Aufgaben und des Projektmanagements an.

Im November wurden grundsätzliche Entscheidungen über die zu verwendende Hardware getroffen. Als Folge dieser Entscheidungen wurden zwei Steuergeräte bestellt. Zum Thema Stromversorgung wurden die vorhandenen Möglichkeiten untersucht und beschlossen, ein den Anforderungen genügendes Netzteil zu kaufen. Ferner wurde ein Zeitplan in Anlehnung an das V-Modell entwickelt (siehe Kapitel 2.5) und gemäß diesem als erster Schritt die umfassende Anforderungsaufnahme festgelegt. Hierzu wurden drei Gruppen gebildet, die für ihr jeweiliges Thema die Anforderungen sowie die jeweiligen Prioritäten festgelegt haben. Diese Teilgruppen beschäftigten sich mit dem Fahrzeugnetz, der Steuerumgebung und der Messumgebung. Diese Arbeiten mündeten im Anforderungsdokument (siehe Kapitel 4). Außerdem standen parallel dazu Überlegungen zur Ausgestaltung des Fahrzeugnetzes auf dem Plan.

Im Dezember konnte die Software CANoe durch verschiedene Experimente an unsere Bedürfnisse angepasst und als Visualisierungswerkzeug für unser Netz vorbereitet werden. Außerdem wurden die Messknoten genauer technisch geplant. Für das Fahrzeugnetz wurden vier Entwürfe entwickelt, welche die von Audi und Kostal zugesagte Hardware berücksichtigten und gemäß des Anforderungsdokumentes evaluiert wurden. Auch wurde die Projektplanung noch einmal kritisch hinterfragt und ein Wechsel auf ein iteratives Projektmanagement beschlossen. Da der alte Plan das mittlerweile notwendig gewordene, ausführliche technische Analysieren der bereitgestellten Hardware nicht vorsah, wurde ein neuer Zeitplan erstellt. Die neuen analytischen Aufgaben haben infolgedessen den Großteil der Teammitglieder umfassend eingespannt. Die Aktoren wurden auseinandergebaut und dokumentiert, Kabel und Stecker wurden angefertigt, erste Gehversuche mit der CAN-Karte und CANoe durchgeführt, sowie Testpads zum Ansprechen der Flash-Schnittstellen der Steuergeräte lokalisiert und nachgemessen. Parallel dazu wurde ausgehend vom gewählten Fahrzeugnetzentwurf je ein logischer und technischer Systementwurf erstellt, der Demonstrator (ein Minimalziel der PG) geplant und das Thema globale Zeit weiter bearbeitet. Im weiteren Verlauf wurde nun zur „Halbzeit“ des ersten Semesters der erreichte Status Quo erfasst und festgelegt, welche Ziele wir im ersten Prototypen zum Ende des ersten Zyklus (entsprechend dem Ende des ersten PG-Semesters) verwirklicht haben wollten. In der Woche vor Beginn der zweiwöchigen Weihnachts- und Neujahrpause wurde die erste Vorlage für den Zwischenbericht erstellt. Ferner wurde die Geschwindigkeit der TriCore-Boards durch Experimente bestimmt. Der FADC konnte in Betrieb genommen, ein Blinker-Relais beschafft und der Scheinwerfermotor zur Steuerung der Scheinwerfer in Bewegung gesetzt werden.

Im ersten Monat des Jahres 2008 konnte der CAN-Treiber prinzipiell verstanden und (noch ohne ProOSEK) zum Laufen gebracht werden. Bei ProOSEK konnten leider noch keine großen Fortschritte erreicht werden, da es galt, vielfältige, wechselhafte Fehlermeldungen auszuwerten. Bei der Kostal-Hardware wurde beim Gangwahlschalter und Dachelement die Schnittstelle zum Flashen vorbereitet. Erste Versuche auf Register zuzugreifen und zu Flashen waren jedoch noch nicht erfolgreich. Auch bei der Lenksäule wurde versucht, das Flashen vorzubereiten. Allerdings gab es Fehler beim Installieren der Flashersoftware. In den folgenden beiden Wochen kamen einige Teilaufgaben, wie z.B. das Analysieren der Kostal-Hardware oder das Einbinden des ProOsek, auf Grund vielschichtiger Probleme zunächst massiv ins Stocken, konnten dann aber doch noch gelöst werden. Auch im Bereich Stromversorgung gab es Probleme, da die vorhandenen Anschlüsse zu dünnwandig und daher nicht geeignet waren. Bei der Kostal-Lenksäule konnte keine Möglichkeit gefunden werden, den Mikrocontroller aufzuwecken oder zu flashen, daher wurde dieses Bauteil zunächst aus dem ersten Prototypen gestrichen. Auch beim Dachelement und beim Gangwahlschalter verlief das Flashen nicht erfolgreich, da lediglich Fehlermeldungen auftraten. Zwar konnten die Module nun geweckt werden, doch wurden keine CAN-Nachrichten empfangen. Allerdings gelang es nun, die Datenspeicher auszulesen. Große Probleme machten die Watchdog-Chips, wodurch etwa das mittlerweile erfolgreich gestartete Flashen unvermittelt abbrach (siehe Kapitel 7.3.3).

Im Februar 2008 konnten dann Erfolge erzielt und die Rückschläge der beiden Vorwochen teils ausgebessert werden. Es wurden Möglichkeiten gefunden, die Module wachzuhalten, sie zu flashen und die Kommunikation zwischen Mikrocontroller und Watchdog-Chip erfolgreich mit einem Oszilloskop zu untersuchen. Die Scheinwerfer-Gruppe konnte mittels CANoe rudimentär das Kurvenlicht implementieren; die CAN-Treiber-Gruppe brachte den Treiber auf beiden Boards zum Einsatz; infolgedessen konnten erfolgreich Nachrichten empfangen und versendet werden. ProOSEK funktionierte allerdings noch nicht fehlerfrei und wurde zunächst nicht weiterverfolgt. Die Blinker konnten mit dem Relais in Betrieb genommen werden und die Stromversorgungsgruppe konnte die neuen Kabel für das Netzteil anfertigen und einsetzen. Ferner konnte die Platine zur Strommessung fertig gelötet werden. Auch der Zwischenbericht machte Fortschritte. Folglich fiel das Fazit in diesem Monat deutlich positiver aus als zuvor. Parallel zu den genannten Aufgaben wurde die Projektplanung stetig überprüft, die Exkursionen geplant und die laufende Dokumentation in SVN und Wiki gepflegt.

Mit Abschluss des ersten PG-Semesters konnte der erste Prototyp (siehe Kapitel 7) fertig gestellt werden.

Um einen Einblick in die aktuelle Forschung und Entwicklung der in dem Projekt enthaltenen Themengebiete zu bekommen, wurde eine Exkursion zur Embedded World (26.-28.02.2008, Messe Nürnberg), zur Firma Elektrobit in Erlangen sowie zur Audi AG in Ingolstadt durchgeführt. Ferner wurde die Niederlassung der Firma KOSTAL in Dortmund besichtigt. Für diese Firmenbesuche wurde ein kurzer Vortrag erstellt, um die Arbeiten der Projektgruppe vorstellen zu können.

Nach der vorlesungsfreien Zeit konnte die Projektgruppe in das zweite Semester starten.

Im April wurde zunächst die weitere organisatorische Ausrichtung der PG-Arbeit diskutiert. Dabei wurde festgesetzt, dass es einen weiteren Prototyp sowie ein abschließendes Endprodukt geben soll. Ausgehend von diesen Entscheidungen wurde der zeitliche Ablauf und damit insbesondere die Zeitpunkte der Fertigstellung der einzelnen Produktstufen festgelegt: Da die PG auf

dem jährlich stattfindenden Campusfest vorgestellt werden sollte, wurde die Deadline für den zweiten Prototyp kurz vor den Veranstaltungstag gelegt. Das Endprodukt sollte bis zum Ende des Sommersemesters fertiggestellt werden. Die Hauptaufgaben des ersten Monats des neuen Semesters waren die Umsetzung der Messumgebung, die Integration weiterer Hardware, wie z.B. dem Gangwahlschalter und dem Dachelement. Bei der Umsetzung der Messumgebung lag das Hauptaugenmerk darin, Experimente mit der Strommessplatine durchzuführen. Im Laufe dieser Experimente galt es, verschiedene Fehlerquellen, die zu nicht ganz korrekten Messwerten führten (z.B. Fehler in der Verstärkung), zu identifizieren und zu beseitigen. Bei der Hardware war es zunächst notwendig, die Watchdog-Probleme in den Griff zu bekommen. Dies bereitete zunächst noch einige Schwierigkeiten, welche aber im weiteren Verlauf erfolgreich bewältigt werden konnten. Nachdem dieses Problem gelöst werden konnte, ging es darum, die LEDs und die Hall-Sensoren genauer zu analysieren und zu testen. Nachdem es ermöglicht wurde, den Gangwahlschalter und das Dachelement zu programmieren, wurde ein Treiber für die Kommunikation über den CAN-Bus benötigt. Versuche, einen frei verfügbaren CAN-Treiber in das Projekt einzubinden, verliefen nicht erfolgreich, da dieser nicht komplett für unsere Bedürfnisse ausgerichtet und zum Teil fehlerhaft war. Daraufhin wurde ein Treiber in Eigenleistung erstellt und erfolgreich eingebunden. Zusätzlich konnte die Dokumentation der Hardwarekomponenten weiter vorangetrieben werden. Ferner wurde beschlossen, ein auf der Embedded-World-Messe vorgestelltes variables Programmiermodul anzuschaffen und in das Projekt einzugliedern.

Um die Ziele des Demonstrators für den zweiten Prototyp zu erreichen, wurde festgelegt, dass die Open-Source-Rennsimulation „TORCS“ verwendet werden soll. Dies erforderte, die Rennsimulation an unsere Anforderungen anzupassen, damit die von uns vorbereitete Hardware in die Simulation eingegliedert werden konnte. Um diese Simulation sinnvoll verwenden zu können, wurde ein PC-Lenkrad beschafft und mittels eigens hierfür konstruierter Verkabelung an die TriCore-Boards angeschlossen. Parallel zu diesen Arbeiten konnte das ProOsek erfolgreich eingebunden werden. Von nun an konnten CAN-Nachrichten versendet werden. Als nächster Schritt wurde die Einbindung der Scheinwerfer-Software in Angriff genommen und auf einen ereignisgesteuerten Ansatz umgestellt. Auch konnte der zweite Scheinwerfer eingebunden werden. Im Software-Bereich wurde der von uns benötigte Funktionsumfang abgesteckt und das Nachrichtenformat festgelegt.

Im Folgemonat wurden zunächst die Vorbereitungen für das Campusfest und damit für den ersten öffentlichen Auftritt getroffen. Dies erforderte die Bildung eines Organisationskomitees, welches den Stand (Tische, Plakate, Aufbau, etc.) planen sollte und weitere organisatorische Angelegenheiten, wie in etwa das Erstellen eines Schichtplans, erledigen sollte. Auf seiten der Hardwaregruppen wurden die im Vormonat aufgenommenen Arbeiten weitergeführt und weitere Probleme gelöst. So konnten z.B. die ersten CAN-Nachrichten verschickt werden. Die Software des Gangwahlschalters konnte angepasst werden, damit das Modul auf Nachrichten anderer Steuergeräte reagieren konnte; auch konnten die Tasten des Dachelements in Betrieb genommen werden. Im Bereich der Strommessung wurde die Software für den Scheinwerfer fertig gestellt und erfolgreich getestet. Die Messdaten konnten online über CAN gesendet werden oder alternativ gesammelt und en bloc abgefragt werden. Für die „TORCS“-Rennsimulation wurde ein Mensch-Maschine-Interface geplant, welches unter anderem vorsah, die Steuerung per Lenkrad zu ermöglichen, wobei einige Feinjustierungen vorgenommen werden mussten. Des weiteren wurde der Gangwahlschalter integriert. Um die Simulation auf dem Campusfest vorstellen zu können, wurde die Software auf ein eigenes Notebook portiert.

Im Juni wurden die letzten Details für das Campusfest vorbereitet, so wurden die Plakate ge-

druckt, T-Shirts angefertigt und Absprachen mit benachbarten Gruppen getroffen. Die „TORCS“-Simulation wurde weiter angepasst, getestet und in den Demonstrator eingebunden. Die Applikationssoftware wurde erfolgreich in OSEK portiert; allerdings stellte sich das Problem, dass die Software noch nicht in den FLASH-Speicher geschrieben werden konnte; parallel dazu konnte die Buslast auf einen akzeptablen Wert reduziert werden. Kurz vor dem Stichtag konnte eine erfolgreiche Generalprobe des zweiten Prototyp durchgeführt werden, bei der einige kleinere Schwierigkeiten (z.B. unpassende Lenksensibilität) zu Tage traten, die bis zur Vorstellung behoben werden konnten. Nach dem erfolgreichen Auftritt beim Campusfest wurde das Projekt leider zeitlich etwas aus der Bahn geworfen. Das eingesetzte Netzteil fiel schon während des Fests technisch bedingt aus, so dass kurzfristig Ersatz beschafft werden musste. Zusätzlich schlich sich beim Gangwahlschalter ein Kurzschluss ein, so dass die Arbeiten auf einem zweiten, bisher noch nicht eingesetzten Gangwahlschalter von neuem beginnen mussten. Zusätzlich wurde ein Abgleich der im ersten Semester formulierten Minimalziele und Anforderungen mit dem bis dato erreichten Stand der Dinge durchgeführt. Um die eingesetzten TriCore-Boards ein wenig besser schützen zu können, wurden Gehäuse für die Boards entworfen und gebaut.

Im Juli und teilweise in der vorlesungsfreien Zeit im August wurden die letzten Arbeiten auf dem Weg zur Fertigstellung des Endprodukts vorgenommen. Da nun zusätzliche TriCore-Boards zur Verfügung standen, war es möglich, jeden Aktor an ein eigenes Mess- und Steuerboard anzuschließen. Ferner konnten die Arbeiten am Dachelement vorangetrieben werden. Da die Einbindung des Dachelements einen Low-Speed-CAN-Bus erforderte, wurde ein TC 1796-Board als Gateway umfunktioniert. Des weiteren wurde die Steuer- und Messumgebung weiter verbessert und in ProOSEK eingebettet. Die Software wurde soweit modifiziert, dass sie in den Flash-Speicher geschrieben werden konnte. Um die Messwerte genauer spezifizieren zu können, wurde ein Konverter-Tool entwickelt. Infolgedessen konnten die Werte zeitgenau dargestellt werden. Parallel zu diesen Arbeiten wurde eine Präsentation erstellt, um das Vorgehen und die Arbeit der Projektgruppe im abschließenden Fachgespräch, das am 25. August 2008 stattfand, angemessen repräsentieren zu können.

2.4 Besuch von Schülergruppen

Im Laufe des zweiten Semesters wurde das Projekt mehreren Schülergruppen gymnasialer Oberstufen vorgestellt. Hierbei wurde zunächst eine kurze Einführung in Form eines Vortrags vermittelt.

Inhalt dieses Vortrags war eine kurze Ausführung über die Motivation, Projektgruppen während des Informatikstudiums an der Technischen Universität Dortmund durchzuführen. Nach dieser eher allgemeinen Ausführung wurden die speziellen Ziele sowie die eigentliche Vorgehens- und Arbeitsweise innerhalb der Projektgruppe verdeutlicht. Anschließend wurden die einzelnen Bestandteile des aktuellen Prototyps näher vorgestellt. Als Abschluss wurde der aktuelle Stand der Entwicklung sowie ein Ausblick auf noch zu erledigende Arbeiten vermittelt.

Nach diesem Vortrag hatten die Schüler die Möglichkeit, den ersten Prototyp „live“ in Aktion zu erleben.

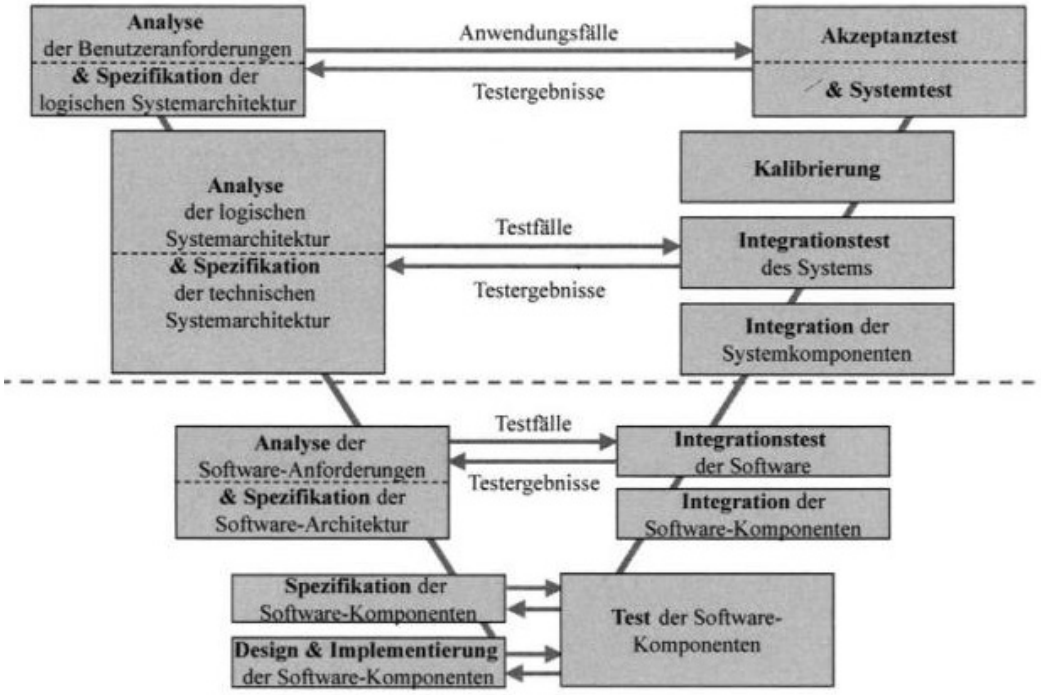


Abbildung 2.3: V-Modell zur Prozessbeschreibung

2.5 Projektplanung/-management

Die Entwicklung eines Laboraufbaus erfüllt alle wesentlichen Merkmale eines Projektes wie etwa eindeutige Aufgabenstellung, definierbare Ziele und zeitliche Befristung. Daher ist eine Projektplanung gemäß den Regeln des Projektmanagements sinnvoll.

Zunächst sind die Projektziele zu unterscheiden. Dies sind Terminziele, Kostenziele, sowie Qualitätsziele. Für alle drei Arten sollte also eine eigene Zielplanung erfolgen. Wesentliche Elemente des Projektmanagements sind dabei die Zerlegung in Teilprobleme, deren Finalisierung mit dem Erreichen definierter Meilensteine erfolgt. Hierbei sollte eine umfassende Planung und Verfolgung der benötigten und verfügbaren Ressourcen erfolgen, um die Kostenziele des Gesamtprojektes stets zu gewährleisten. Auch ist eine Terminplanung und die terminliche Koordination verschiedener beteiligter Gruppen bzw. Abteilungen notwendig.

Für unsere Projektgruppe bedeutet dies, dass sich die ersten verbindlichen Ziele aus den Minimalzielen des Projektgruppenantrages ergeben. Ausgehend von den im Vorbereitungsseminar erarbeiteten Kenntnissen über Prozesse der Fahrzeug- und Fahrzeugsystementwicklung wurde der Beschluss gefasst, sich an das verbreitete V-Modell (siehe Abbildung 2.3) zu halten. Hierauf aufbauend wurde ein erster Zeitplan erstellt (siehe Tabelle 2.1).

Aufgrund der Tatsache, dass ein Großteil unserer Arbeit darin bestand, die richtigen Belegungen

Woche	Inhalt	Entspricht im V-Modell
1	Einstieg, PG-Fahrt	
2	Ideensammlung	
3	Vorbereitung	
4 / 5	Anforderungen an Versuchsaufbau und Messumgebung	Analyse der Benutzeranforderungen
6 / 7	Entwurf von Versuchsaufbau und Messumgebung erstellen	Spezifikation und Analyse der logischen Systemarchitektur
8 / 9	Abstimmung der Entwürfe und Hardwareauswahl	Spezifikation der technischen Systemarchitektur
10 / 11	Auswahl Standardsoftware, Test dieser, Test mit gewählten Hardwarekomponenten	Spezifikation der Software-Komponenten und Test dieser, Integrationstest der Software
12 / 13	Inbetriebnahme des Versuchsaufbaus, (erste Gehversuche mit eigenen Testprogrammen)	Integration der Systemkomponenten, Integrationstest des Systems
SoSe 1	Rekapitulation des WS	
SoSe 2-6	Implementierung der entworfenen Steuer- und Messumgebung in Hard- und Software	Integration der Software- und Systemkomponenten, Integrationstests
SoSe 2-7	Entwurf Demonstrator	Analyse und Spezifikation der Software
SoSe 8-11	Implementierung des Demonstrators	Implementierung und Integration der Software-Komponenten sowie zugehörige Tests
SoSe 12	Einsatz Demonstrator	Integration der Systemkomponenten
SoSe 10-14	Abschlussbericht	

Tabelle 2.1: Erster Entwurf eines Zeitplans

der Anschlüsse der einzelnen Hardwarekomponenten herauszufinden, passte das eigentliche V-Modell nicht mehr zu unserer Herangehensweise. Wir entschieden uns daher dafür, eine andere Form des V-Modells, das V-Modell XT in der Variante inkrementell mit Vorspezifikation zu nutzen. Ein Vorteil dieses Vorgehens ist, dass der Gesamtaufwand leichter planbar ist, als bei einem rein inkrementellen Vorgehen. Des weiteren liegt bei mehreren Inkrementen, in unserem Fall sind dies zwei, der Schwerpunkt ab dem zweiten Inkrement auf der Realisierung. Hierdurch wird die Struktur des Projekts vereinfacht. Das erste Inkrement, der erste Prototyp, beinhaltet nur eine abgespeckte Version des Demonstrators, ist aber ansonsten vollkommen funktionsfähig. In diesem Inkrement steckt ein Großteil der Vorarbeit, die nötig war, um die Hardware erstmals anzuschließen und in Betrieb zu nehmen. Während der Erstellung des Prototyps wurde sowohl die Entwicklungsumgebung installiert und in Betrieb genommen, als auch Testmethoden etabliert, um Referenzwerte gegenüber einem lauffähigen System zu haben. Das zweite Inkrement ist das fertige Endprodukt.

Die Umgestaltung des Zeitplans nahmen zwei PG-Teilnehmer vor, die zwischenzeitlich durch den Besuch einer Spezialvorlesung vertiefte Kenntnisse im Projektmanagement erworben haben.

2.6 Teameinteilung

Schon in den Zielbeschreibungen und in der Projektplanung wurde deutlich, dass sich aus den Arbeiten in der Projektgruppe mehrere Schwerpunkte ergeben würden. Neben der Inbetriebnahme des Versuchsaufbaus war ein weiterer Schwerpunkt in der Implementierung der Steuer- und Messumgebung zu sehen. Die Ergebnisse dieser beiden Schwerpunkte sollten schlussendlich in der Entwicklung und Inbetriebnahme eines Demonstrators münden.

Ausgehend von den Zielbeschreibungen und den Ergebnissen der Projektplanung, wurden diese Schwerpunkte in einzelne Teilziele aufgeteilt. Um diese Teilziele zu erreichen, wurden von den zwölf studentischen Teilnehmern mehrere Teams gebildet.

Eine Grobeinteilung konnte in die folgenden Hauptteams vorgenommen werden, wobei diese erneut in kleinere Teilteams aufgeteilt wurden.

- Team Fahrzeugnetz:
  Zu Beginn sollten die einzelnen Komponenten des Fahrzeugnetzes in den Teams hinsichtlich der gegebenen Funktionalitäten analysiert werden. Darauf aufbauend sollte die Inbetriebnahme der für den ersten Prototyp notwendigen Fahrzeugkomponenten sowie die Vernetzung der Steuergeräte mittels eines selbst entwickelten CAN-Bussystems erfolgen.
- Team Steuerungs- und Messumgebung:
  In einer ersten Planungsphase sollte die Umgebung in Hard- und Software entworfen werden. Dieser Planungsphase sollte eine Implementierungsphase folgen, in welcher die Hard- und die Software sowohl beschafft als auch an die Anforderungen der Steuer- und Messumgebung angepasst werden sollte.

Sich ergebende Überschneidungen innerhalb einzelner Arbeitsbereiche erforderten eine gute Koordination und Kooperation zwischen den Teams.

Kapitel 3

Seminar

Zur Vorbereitung und um das Wissen der Teilnehmer der Projektgruppe auf den gleichen Stand zu bringen, wurden in einer Seminarphase zu Beginn des Wintersemesters verschiedene Einzelvorträge zu Teilgebieten der Thematik von den Studenten erarbeitet und gehalten.

Um einen thematischen Überblick zu vermitteln, folgt eine Kurzbeschreibung der Ausarbeitungen zu den einzelnen Vorträgen. Die Ausarbeitungen sowie die Folien selbst sind auf der Webseite der Projektgruppe (http://ess.cs.tu-dortmund.de/Teaching/PGs/autolab/seminarthemen.html) abrufbar.

3.1 Verteilte Echtzeitsysteme

Echtzeitsysteme sind Systeme, bei denen ein Ergebnis innerhalb eines vorher fest definierten Zeitintervalls garantiert berechnet werden muss. Verteilte Echtzeitsysteme bestehen aus mehreren miteinander verbundenen Komponenten und erfüllen beispielsweise Steuer- und Kontrollaufgaben in Fahrzeugen. Grundsätzlich müssen Echtzeitsysteme ihre Operationen innerhalb einer vorher festgelegten Zeit (Deadline) abschließen. Kann die Verletzung einer Deadline kompensiert werden, handelt es sich um ein weiches Echtzeitsystem. Bei einem harten Echtzeitsystem hat das Nichteinhalten der Deadline ein sofortiges Versagen des Systems zur Folge. Dies könnte z.B. eine verspätetes Auslösen eines Airbags sein. Zur Vermeidung von Fehlern in verteilten Echtzeitsystemen, die durch einen Komponentenausfall oder durch Komponentenfehler entstehen können, bieten sich zwei verschiedene Techniken an: Mit dem Fail-Silent-Knoten und der Triple-Modular-Redundanz können falsche Berechnungen eines Knotens maskiert oder der Ausfall eines Knotens kompensiert werden.

Um ein verteiltes System mit hoher Vorhersagbarkeit zu erhalten, bietet sich eine Time-Triggered-Architektur an.

Quellen: [Kop01, Mar03]

3.2 Zeit- und ereignisgesteuerte Echtzeitsysteme

In Abhängigkeit von den Auslösemechanismen für den Start der Kommunikations- und Verarbeitungsaktivitäten in einem Computersystem existieren unterschiedliche Ansätze zum Design von Echtzeitsystemen: In einem ereignisgesteuerten Echtzeitsystem wird ein Verarbeitungsprozess als eine Konsequenz des Auftretens eines signifikanten Ereignisses (Termination eines Tasks, Empfang einer Nachricht oder ein externer Interrupt) initiiert. Solch ein signifikantes Ereignis ist eine Zustandsänderung, die eine Bearbeitung durch das Steuerungssystem erfordert. In einem zeitgesteuerten System hingegen werden alle Aktivitäten periodisch zu vorbestimmten Zeitpunkten gestartet. Die Implementierung einer gegebenen externen Spezifikation durch ein zeitgesteuertes System verlangt eine sehr detaillierte Entwicklungsphase, in welcher die maximale Ausführungszeit aller zeitkritischen Programme festgelegt und die Ausführungspläne für alle Betriebsmodi erstellt werden müssen. Als Resultat dieses detaillierten Planungsaufwands ist das Zeitverhalten zeitgesteuerter Systeme gut vorhersagbar. Bei Implementierung eines ereignisgesteuerten Systems ist dieser Aufwand nicht nötig. Allerdings gibt es bei diesem Ansatz keine zeitlichen Abkapselungen der Tasks, wodurch das zeitliche Verhalten - im Vergleich zu zeitgesteuerten Systemen - nicht so gut vorhersagbar ist. Dies hat zur Folge, dass das Testen der ereignisgesteuerten Systeme mit einem höheren Aufwand verbunden ist. Folglich ist dieser Ansatz für große Systeme nicht so gut geeignet wie der zeitgesteuerte Ansatz. Aus Sicht der Ressourcennutzung hingegen ist der ereignisgesteuerte Ansatz in vielen Fällen dem zeitgesteuerten Ansatz überlegen.

Versuche, die Vorteile beider Ansätze zu kombinieren, sind von großem Forschungsinteresse. Eine mögliche Lösung ist es, den ereignisgesteuerten Entwurf für die Knoten des Computersystems und den zeitgesteuerten Entwurf für die Kommunikation der Knoten untereinander anzuwenden; dies bietet den Vorteil, dass das System eine zeitliche Abkapselung zur Verfügung stellen würde und sehr gut auf der Architekturebene testbar wäre (zeitgesteuerter Ansatz) sowie eine relativ große Flexibilität des Schedulings und der Ressourcenausnutzung innerhalb der Knoten gegeben wäre (ereignisgesteuerter Ansatz).

Quellen: [Kop91, Kop93, Kop97, Oet97, H.W05, Wil06]

3.3 CAN

Die heutigen Automobile beinhalten immer mehr Elektronik. Egal ob Klein- oder Oberklassewagen, in jedem Auto ist eine Vielzahl von elektronischen Geräten verbaut. Diese Geräte lassen sich grob in die Klassen Multimediakomponente (z.B. CD-Spieler, Freisprecheinrichtung etc.) und Steuergeräte (z.B. ABS, ESP etc.) einteilen. Da immer mehr Elektronikelemente in die heutigen Autos eingebaut werden, haben Bussysteme eine große Bedeutung in der Automobilindustrie. Somit auch der CAN-Bus. Der CAN-Bus wird sowohl zur Vernetzung von Multimediakomponenten als auch zur Vernetzung von Steuergeräten eingesetzt. Der CAN-Bus besitzt eine besonders gute Fehlererkennung und eine hohe Geschwindigkeit (max. 1 MBit), wodurch er besonders bei der Vernetzung von sicherheitskritschen Systemen verbaut wird. Da der CAN-Bus einen ereignisorientierten Ansatz verfolgt, was für sicherheitsrelevante Systeme nicht von Vorteil ist, wurde auch eine Erweiterung, der „Time-Triggered-CAN“ (TTCAN) entwickelt. Der TTCAN vereinigt alle Vorteile des normalen CAN mit der Eigenschaft des Echtzeitverhaltens.

Der CAN-Bus ist letztendlich nicht mehr aus der Automobilindustrie wegzudenken. Durch seinen Einsatz wurden sowohl die Kosten als auch das Gewicht der heutigen Autos sehr reduziert. Des weiteren führen seine Ausfallsicherheit und seine sehr gute Fehlererkennung zur Anwendung in lebensrettenden Systemen, wie z.B. das ABS oder ESP.

Quellen: [Ets02]

3.4 FlexRay

Auf Grund der immer weiter steigenden Anzahl an Mikrocontrollern einerseits und der damit verbundenen Kommunikationskomplexität sowie den immer komplexer werdenden Aufgaben andererseits reichen die älteren Bussysteme wie CAN und LIN nicht mehr aus. FlexRay bietet eine deutlich größere Bandbreite von bis zu 10 Mbit pro Kanal und ist damit auch für zukünftige Aufgaben gut ausgestattet. Das FlexRay-Konsortium[1] entwickelt den Standard fortwährend weiter. So ist zum Beispiel der Einsatz von Lichtwellenleitern bereits angedacht. FlexRay bietet eine innovative Struktur der Kommunikation. Das System ist echtzeitfähig und trotzdem in der Lage, für Komponenten geringerer Priorität auf Ereignisse zu reagieren. Es ist möglich, beliebige Topologien miteinander zu verbinden. Die Knoten des Systems sind größtenteils unabhängig, so dass Ausfälle des Gesamtsystems minimiert werden.

FlexRay ist das Bussystem für die zukünftig immer komplexer werdenden Aufgaben im Automobilbereich. Eine Verdrängung von CAN und LIN ist jedoch nicht absehbar, da die Kosten immer noch verhältnismäßig hoch sind.

3.5 LIN

LIN dient als Subbus zur kostengünstigen Vernetzung von Steuergeräten im Auto. Einsatz findet LIN im Bereich der einfachen Sensor- und Aktoranwendungen. Für diesen kostensensiblen Bereich kommt der CAN-Bus nicht in Frage, da die Bandbreite und Vielseitigkeit hier nicht benötigt wird und auch die Kosten für die Vernetzung mit diesem Bus zu hoch sind. LIN hat sich heute als Subbus für einfache Sensor- und Aktoranwendungen in der Automobilindustrie etabliert und wird von vielen Automobilherstellern eingesetzt. Um die Kosten für die einzelnen Steuergeräte niedrig zu halten, basiert LIN auf den weit verbreiteten UART-Interfaces. Dies ist eine serielle Schnittstelle, die bereits in den meisten Mikrocontrollern integriert ist. Der LIN-Bus wird für sicherheitsunkritische Komponenten im Auto eingesetzt. In den Anwendungsbereich fallen Komfortfunktionen, wie zum Beispiel Fensterheber, Zentralverriegelung, elektrische Spiegelverstellung, elektrische Sitzverstellung, Regensensor, Lichtsensor, elektrisches Schiebedach oder Steuerung für Klimaanlage.

Quellen: [lin06, WZ07]

[1] Quelle: Das FlexRay Konsortium. URL: `http://www.flexray.com` [Stand: 13.02.2008]

3.6 OSEK/OS

OSEK-OS wurde vom OSEK/VDX-Konsortium entwickelt und ist ein Betriebssystem für Steuergeräte im Automobilbereich. Der Standard wurde entwickelt, um wiederkehrende Kosten und Zeit für die Entwicklung von Steuergerätesoftware einzusparen. Ziel war es, Portabilität und Wiederverwendbarkeit zu gewährleisten. Diese Ziele sollten erreicht werden, indem anwendungs- und hardwareunabhängige Interfaces bereitgestellt wurden. Die Architektur sollte durch Konfigurierbarkeit und Skalierbarkeit effizient auf die Anwendung abgestimmt werden.

OSEK-OS ist der momentane Standard im Bereich der Betriebssysteme für automotive Steuergeräte.

Quelle: [vdib]

3.7 OSEK/NM

Die Vernetzung von Steuergeräten/ECUs in den heutigen Fahrzeugen nimmt immer mehr zu. Standardisierte Schnittstellen und Protokolle sind unabdingbare Voraussetzung für die Funktionsfähigkeit dieser komplexen Netze. In einem Kfz arbeiten sehr viele Steuergeräte sehr eng miteinander zusammen. Das Motormanagement ist ein Beispiel, bei dem ständig neue Messwerte von Sensoren (TPS, O2, MAT, etc.) erfasst und weiterverarbeitet werden müssen, um das in der aktuellen Situation benötigte Kraftstoff-Sauerstoff-Gemisch bestimmen zu können. Bei einem komplexen Netz besteht die Gefahr, dass es durch gegenseitige Beeinflussungen der einzelnen Knoten zu Fehlfunktionen kommt. Dies muss unbedingt vermieden werden. Eine dedizierte Netzwerk-Management-Komponente hat die Aufgabe, den Status (bspw. zur Zeit aktive Knoten) eines Netzes zu überwachen. Unkoordinierte Ab- und Anmeldungen der Knoten von bzw. bei dem Netz werden durch das Führen von „Verhandlungen“ realisiert. Dies ist ein Mechanismus, der verhindern soll, dass sich Knoten, die noch für Berechnungen benötigt werden, plötzlich in den Sleep-Modus versetzen. Ein weiteres Ziel des Netzwerk-Managements ist die Einsparung von Kosten und Entwicklungszeit, sowie die Portabilität durch den Einsatz vorgefertigter Module. Das OSEK/VDX[2] Network Management (kurz: NM) ist eine solche Netzwerk-Management-Komponente, welche bereits erfolgreich von verschiedenen Kfz-Herstellern in ihren Fahrzeugen eingesetzt wird.

Quellen: [vdia, vdi00, For07]

3.8 OSEK/COM

OSEK/COM ist eine vom OSEK-VDX-Konsortium entwickelte, signalbasierte Kommunikationsumgebung für Anwendungssoftware für automotive Steuergeräte. Es beschreibt die Schnittstellen und das Verhalten für die Kommunikation sowohl innerhalb einer elektronischen Steuereinheit

[2] OSEK/VDX-Portal. URL: http://www.osek-vdx.org [Stand: 11.02.2008]

(interne Kommunikation) als auch zwischen vernetzten Knoten im Fahrzeug (externe Kommunikation). Die Spezifikation beschreibt dabei das Verhalten innerhalb einer elektronischen Steuereinheit (ECU), in der sogenannten Interaction Layer. Es wird vorausgesetzt, dass OSEK/-COM zusammen mit einem OSEK-konformen Betriebssystem verwendet wird. Für andere Betriebssysteme definiert die Spezifikation Anforderungen für den Betrieb von OSEK/COM. Die Kommunikation in OSEK/COM erfolgt über Nachrichtenobjekte. Bei interner Kommunikation wird eine Nachricht direkt vom sendenden Nachrichtenobjekt (SMO) zum empfangenden Nachrichtenobjekt (RMO) übertragen. Bei externer Kommunikation benutzt die Interaction Layer Dienste der unter ihr liegenden Schichten. Über I-PDUs (Interaction Layer Protocol Data Units) werden zu sendende Nachricht vom sendenden Nachrichtenobjekt an die darunter liegende Schicht übergeben, während zu empfangende Nachrichten von der unteren Schicht ebenfalls über I-PDUs an das empfangende Nachrichtenobjekt übergeben werden. Die Schnittstelle der Anwendung zur Interaction Layer ist für interne und externe Kommunikation die gleiche.

Quellen: [vdi04]

3.9 OSEKtime

In der Automobilbranche zeichnet sich der Trend ab, mechanische und hydraulische Kopplungen durch fehlertolerante elektronische Systeme zu ersetzen. Beispiele hierfür sind Break-by-Wire (die vollelektronische Bremse) und Drive-by-Wire (Lenkung). Diese X-by-Wire-Anwendungen bestehen aus mehreren Steuergeräten, die ein zuverlässiges, fehlertolerantes und koordiniertes Handeln in hoher zeitlicher Synchronität erfordern. OSEK/VDX-OS hat sich für die typischen Anforderungen im Innenraum und Antriebsstrang bewährt, doch den sicherheitskritischen X-by-Wire-Anwendungen ist das ereignisgesteuerte OSEK nicht gewachsen. Für diesen speziellen Einsatzzweck wurde OSEKtime spezifiziert. Das time-triggered OSEK/OS erzielt Determinismus auf Ebene des Betriebssystems. Die zugehörige fehlertolerante Kommunikationsschicht FTCom erzeugt ein deterministisches Zeitverhalten bei der Kommunikation zwischen den einzelnen Steuergeräten. Dabei spielt die zur Verfügung gestellte globale Zeit eine große Rolle. Gemeinsam erfüllen OSEKtime und FTCom die von den sicherheitskritischen X-by-Wire-Anwendungen gestellten Anforderungen.

Quellen: [Hom05, WZ07, ose05, ose01, ftc01, Gal, TMG, Schb, FF, Scha]

3.10 AUTOSAR

Heutzutage werden schon in Klein- und Mittelklassewagen dutzende elektronische Geräte (vornehmliche Steuergeräte für Fahrzeugfunktionen) verbaut. In der Ober- und Luxusklasse werden sogar bis zu 70 Steuergeräte verbaut. Die Hersteller wollen gerade durch die Zunahme dieser durch die Steuergeräte implementierten Software einen Mehrwert des Autos und damit für das Unternehmen schaffen. Doch die Steuergeräte haben eine Vielzahl an verschiedenen Eigenschaften, so dass für jeden Mikrocontroller die Software erneut angepasst werden muss. Damit aber nicht jeder Code für beliebig viele Mikrocontroller programmiert und gewartet werden muss, gibt

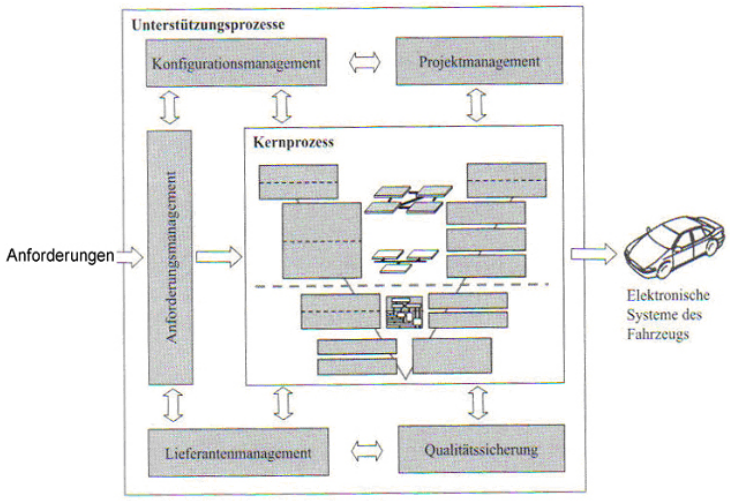


Abbildung 3.1: Prozesse in der automativen Systementwicklung

es spezielle Standards, wie z.B. OSEK und AUTOSAR. AUTOSAR[3] ermöglicht es, die Anwendungssoftware unabhängig von dem speziellen Mikrocontroller zu entwickeln. AUTOSAR stellt auf mehreren Schichten Treiber zur Verfügung, die die Hardware von der Software entkoppeln. Somit kann die Software dann auf jedem anderen Mikrocontroller, der ebenfalls AUTOSAR als Betriebssystem einsetzt, wiederverwendet werden, ohne an den Mikrocontroller angepasst zu werden. AUTOSAR gibt es in verschiedenen Varianten, „den Scalability Classes“, damit nur die wirklich benötigten Features als Betriebssystem auf dem entsprechenden Mikrocontroller laufen.

AUTOSAR gibt es erst seit kurzem und gehört somit noch zur „Bleeding Edge of Development“. Seine Ansätze, wie der Speicherschutz, führen jedoch konsequent die Vision verteilter Anwendungen auf den Steuergeräten, sowie die Nutzung eines Steuergerätes für mehrere Aufgaben (z.B. Redundanzen zur Erreichung von erhöhter Ausfallsicherheit) fort.

3.11 Entwicklungsprozesse

Bei der Betrachtung der Entwicklung von elektronischen Systemen sind neben dem eigentlichen Entwicklungsprozess (Kernprozess) eine Reihe von Unterstützungsprozessen relevant (siehe Abbildung 3.1). Diese sind weitgehend unabhängig von der Softwareentwicklung und können daher für Entwicklungsprojekte jedweder Art meist in gleicher oder ähnlicher Art und Weise eingesetzt werden. Der Kernprozess kann durch mehrere Modell beschrieben werden. Häufig benutzt wird das V-Modell. Es bietet den großen Vorteil, einen durchgängigen Entwicklungsprozess zu modellieren, der deshalb notwendig ist, weil so die Wechselwirkungen zwischen der Entwicklung von Software, Elektronik und des Gesamtsystems am besten eingebunden werden. Außerdem sind hierbei die Maßnahmen der Qualitätssicherung bereits mit eingebunden. Im so dargestellten Kernprozess zur Entwicklung von elektronischen Systemen und Software werden folgende acht

[3]Quelle: AUTOmotive OPEN SYSTEM ARchitecture. URL: http://www.autosar.com [Stand: 15.07.2008]

Schritte unterschieden:

- Analyse der Benutzeranforderungen und Spezifikation der logischen Systemarchitektur
- Analyse der Software-Anforderungen und Spezifikation der Software-Architektur
- Spezifikation der Software-Komponenten
- Design, Implementierung und Test der Software-Komponenten
- Integration der Software-Komponenten und Integrationstest der Software
- Integration der Systemkomponenten und Integrationstest des Systems
- Kalibrierung
- Systemtest und Akzeptanztest

Quellen: [JS06, Beu06]

3.12 Test und Diagnose

Das Versagen von sicherheitskritischen Funktionen kann zu schweren Unfällen führen. An die Sicherheit und Zuverlässigkeit von Fahrzeugfunktionen, wie zum Beispiel der Bremsen, werden daher hohe Anforderungen gestellt. Sowohl die Methode des Testens als auch die Diagnose bezeichnen Maßnahmen zur Sicherstellung dieser Anforderungen und zur Identifikation von Fehlern. *Testen* ist eine Methode zur Identifikation von Fehlern während der Produktion. Eine besondere Bedeutung hat es somit im Rahmen der Qualitätssicherung. Es soll garantieren, dass das Produkt die gestellten Anforderungen erfüllt. Im Gegensatz zur Diagnose werden hier gezielt Spezifikations- und Implementierungsfehler nachgewiesen, die im Rahmen der Integration verschiedener Komponenten entstehen. Eine weitere Aufgabe des Testens ist es, möglichst frühzeitig Fehler, wie zum Beispiel Spezifikationsfehler, nachzuweisen, um so letztlich eine Beschleunigung des Entwicklungsprozesses und demzufolge Kostenersparnis zu bewirken. Für die frühzeitige Identifikation von Fehlern gibt es daher im Automotive-Bereich spezielle Verfahren: *Model in the loop*, *Rapid Prototyping*, *Hardware in the Loop* und *Software in the Loop*.

Die *Diagnosefunktion* dient besonders der Realisierung der Überwachung sicherheitsrelevanter Systeme sowie in diesem Zusammenhang Werkstätten, um bei Fehlverhalten des Fahrzeugs eine gezielte Fehlersuche zu ermöglichen.

Quellen: [JS06, WZ07]

Kapitel 4

Anforderungen an das Projekt

Ausgehend von den Zielen und den damit verbundenen typischen Anwendungsszenarien sind die Anwendungsfälle *Demonstration*, *Einsatz in der Lehre*, *Einsatz in der externen Forschung*, *Einsatz in der internen Forschung in Dortmund* und *Wartung* für die Realisierung des Projekts relevant. Diese Anwendungsfälle werden im Folgenden näher spezifiziert (siehe 4.1). Anschließend werden aus diesen Anwendungsfällen die Anforderungen für das Projekt abgeleitet (4.2).

4.1 Anwendungsfälle

4.1.1 Demonstration

Das zu entwickelnde System soll interessierten Besuchern, z. B. am Tag der offenen Tür des Lehrstuhls oder während des Campusfests, präsentiert werden. Dabei können die Besucher einen Einblick in die Funktionsweise des (Teil-) Systems bekommen und wichtige Funktionen nachvollziehen. Dies geschieht mithilfe einer Vorführanwendung, die (unter Aufsicht) von Mitarbeitern des Lehrstuhls bedient werden kann. Die Besucher können sowohl eine grafische Rückmeldung am PC als auch eine optische Rückmeldung des Aufbaus erhalten. Dies kann in Form eines Fahrsimulators geschehen.

4.1.2 Lehre

Das System soll in der Lehre eingesetzt werden. Es ist vorgesehen, dass Studenten das System für praktische Experimente und vorlesungsbegleitende Übungen in Arbeitsgruppen nutzen werden. Mitarbeiter des Lehrstuhls leiten bzw. unterstützen die Arbeit der Studenten.

Hierbei werden mehrere Studenten gleichzeitig am System arbeiten und in begrenztem Umfang auch eigene Software für die Steuergeräte schreiben.

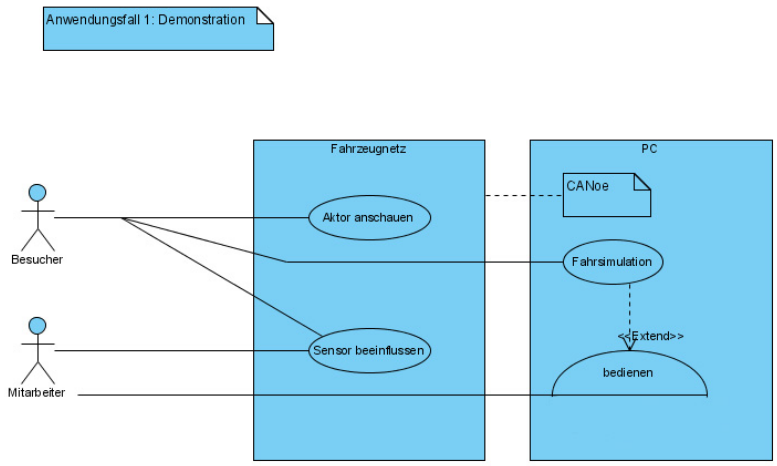


Abbildung 4.1: Anwendungsfalldiagramm Demonstration

4.1.3 Externe Forschung (remote)

Externe Wissenschaftler sollen in der Lage sein, das System unabhängig von ihrem Aufenthaltsort für ihre Forschung nutzen zu können. Dafür werden sie das Internet benutzen und sollen möglichst viele der Vorort-Möglichkeiten aus der Entfernung nutzen können. Dies soll möglichst automatisiert ablaufen, damit nicht stets jemand in Dortmund dafür abgestellt werden muss, das System zu überwachen oder gar Informationen über den Systemzustand weiterzugeben.

4.1.4 Interne Forschung (in Dortmund)

Wissenschaftler der Technischen Universität Dortmund sollen das System für ihre Forschung nutzen können. Diese Forschung umfasst beispielsweise die Betrachtung der Lastverteilung im Fahrzeugnetz, alternative Fahrzeugtopologien oder Messung und Analyse des Stromverbrauchs in Abhängigkeit von der Konfiguration.

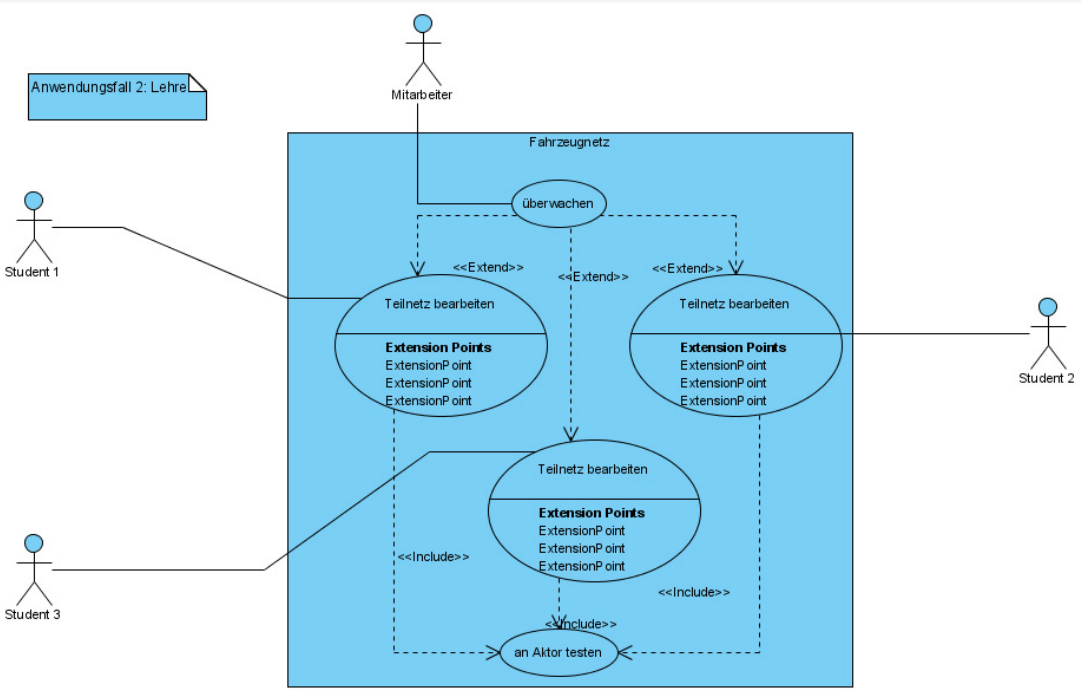


Abbildung 4.2: Anwendungsfalldiagramm Lehre

4.1.5 Wartung

Wartungstechnikern soll der Zugang zu Systemkomponenten ermöglicht werden, um das System bei Fehlern warten zu können.

4.2 Anforderungen

Aus den oben angegebenen Anwendungsfällen ergeben sich die Projekt-Anforderungen, die in drei Kategorien unterteilt werden können: Anforderungen an das Fahrzeugnetz, Anforderungen an die Steuerumgebung und ihre Benutzerschnittstelle sowie Anforderungen an die Messumgebung und ihre Benutzerschnittstelle. Jede Anforderung ist mit einer eindeutigen Identifikationsbezeichnung und einer Priorität [1-3; wobei 1 der höchsten Priorität entspricht] gekennzeichnet.

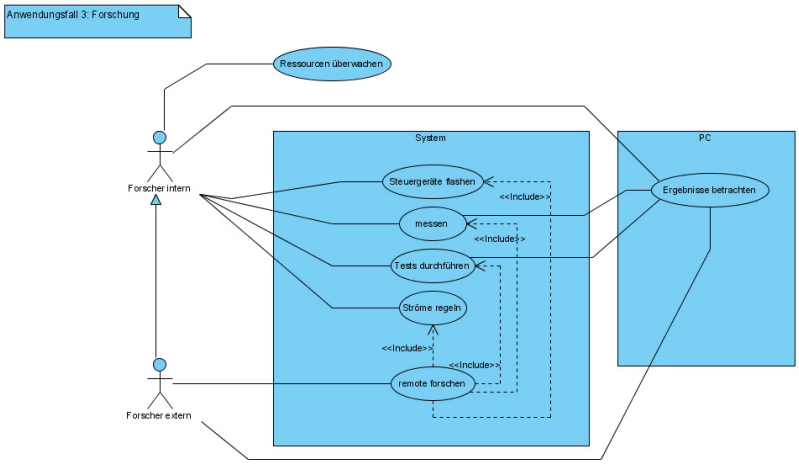


Abbildung 4.3: Anwendungsfalldiagramm Forschung

4.2.1 Anforderungen an das Fahrzeugnetz

4.2.1.1 Allgemeine Anforderungen

FA01 Vorführbarkeit **Priorität 2**

Der Anwendungsfall 4.1.1 liefert die Anforderung, dass das System (oder ein Teilsystem) ohne große Umbaumaßnahmen vorführbar ist.

FA02 Modularität **Priorität 1**

Das Fahrzeugnetz soll in verschiedene, eigenständig funktionierende Teilnetze zerlegbar sein. Dabei soll es für den Anwendungsfall 4.1.1 einzelne Teilnetze geben, die portabel sind und auf Veranstaltungen, wie etwa dem Tag der offenen Tür, vorgeführt werden können. Aus dem Anwendungsfall 4.1.2 folgt auch die Anforderung, dass Teilsysteme einzeln an PC-Arbeitsplätze angeschlossen werden können.

FA03 Fehlertoleranz **Priorität 2**

Das System soll die Besucher beeindrucken und daher eine gewisse Fehlertoleranz besitzen, damit bei Vorführungen (Anwendungsfall 4.1.1) leichte Pannen nicht zu einem PR-Desaster werden und Studierende (Anwendungsfall 4.1.2) keine fatalen Fehler hervorrufen können.

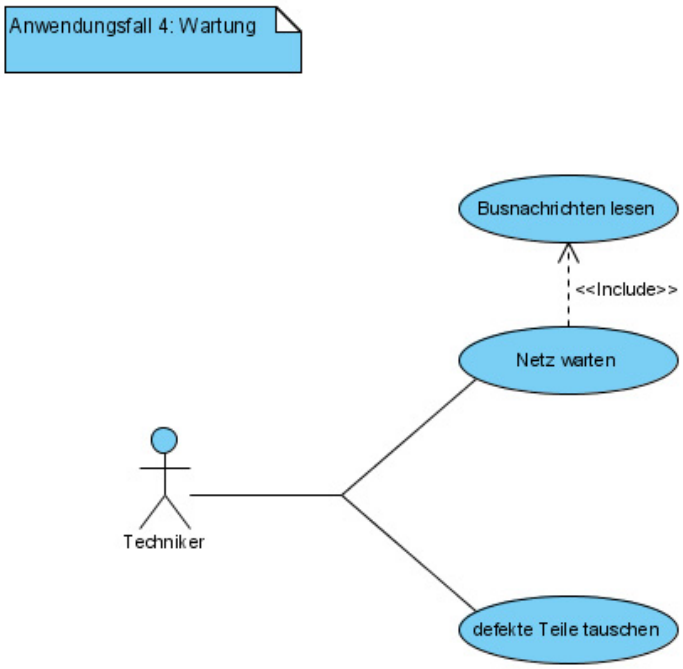


Abbildung 4.4: Anwendungsfalldiagramm Wartung

FA04 Skalierbarkeit **Priorität 2**

Das System soll skalierbar sein, um später weitere Arbeitsplätze oder Komponenten nachzurüsten. Daher sollten die Steuergeräte multifunktional sein.

FA05 Lehrbetrieb **Priorität 2**

Der Anwendungsfall 4.1.2 und der Anwendungsfall 4.1.4 erfordern, dass das System schnell und möglichst einfach zwischen Lehr- und Forschungsbetrieb umgeschaltet werden kann, damit Kapazitäten der Mitarbeiter und des Systems nicht für zahlreiche Umschaltvorgänge verbraucht werden.

FA06 Automatisierte Testabläufe **Priorität 3**

Die Unterstützung automatischer Testabläufe durch das System wäre wunschenswert.

FA07 Realitätsnähe **Priorität 1**

Damit die am System erfolgte Forschung Nutzen bringt, muss das System nah an der Realität aktueller Fahrzeugnetzwerke sein.

FA08 Kosten **Priorität 2**

Der Systemaufbau sowie der nachfolgende Betrieb soll in Bezug auf finanzielle Kosten und aufzubringende Arbeitsleistung möglichst günstig erfolgen.

FA09 Erfüllung von Standards **Priorität 2**

Es sollten aktuelle Industriestandards erfüllt werden und es sollte möglich sein, auch zukünftige Standards durch nicht-erhebliche Maßnahmen zu erfüllen und moderne Technologien zu nutzen. Dafür ist es nötig, das System möglich flexibel zu gestalten. Dies folgt auch aus der Anforderung, dass es möglich sein soll, verschiedene Forschungsprojekte anzugehen, sich also nicht nur an einem speziellen Forschungsziel auszurichten. Dies folgt aus dem Anwendungsfall 4.1.4.

FA10 Größe des Aufbaus **Priorität 1**

Der Aufbau muss in den vorhandenen Raum passen.

FA11 Austausch defekter Teile **Priorität 2**

Der Anwendungsfall 4.1.5 bringt die Anforderung, dass defekte Teile leicht austauschbar sein sollen.

FA12 CAN **Priorität 1**

Da entsprechende Hardware vorhanden ist, muss CAN als primärer Bus benutzt werden. Es kann zusätzlich auch LIN eingesetzt werden.

4.2.1.2 Anforderungen an die Steuergeräte

FS01 Verhalten der Sensoren **Priorität 1**

Der Anwendungsfall 4.1.1 liefert die Anforderung, dass die Sensoren, so zum Beispiel der Neigungssensor, ausgelöst oder simuliert werden können.

FS02 Programmierung der Steuergeräte **Priorität 1**

Die Beschreibung des Anwendungsfalls 4.1.2 liefert, dass die meisten Steuergeräte individuell, parallel und möglichst schnell flashbar sein sollen.

FS03 Stromversorgung der Aktoren **Priorität 2**

Aus dem Anwendungsfall 4.1.2 folgt ferner, dass es möglich ist, die Stromversorgung auf modularen Betrieb umstellen zu können. Die Aktoren müssen somit individuell mit Energie versorgt werden können.

FS04 Notwendige Stromversorgung Lehre **Priorität 1**

Bezugnehmend auf Anwendungsfall 4.1.2, soll die Stromversorgung der Steuergeräte an- und abgeschaltet werden können.

FS05 Weitere Stromversorgung Lehre **Priorität 3**

Des weiteren wäre es schön, wenn die Stromversorgung der Steuergeräte regelbar wäre (Anwendungsfall 4.1.2).

FS06 Remote-Steuerung der Steuergeräte **Priorität 2**

Um dem Anwendungsfall 4.1.3 gerecht zu werden, sollten die meisten Steuergeräte ferngesteuert geflasht werden können.

FS07 Remote-Steuerung der Sensoren **Priorität 2**

Wie im Anwendungsfall 4.1.3 beschrieben, sollten die Sensoren ferngesteuert ausgelöst oder simuliert werden können.

FS08 Remote-Steuerung der Stromversorgung **Priorität 3**

Es wäre schön, wenn die Regelung der Stromversorgung der Steuergeräte ferngesteuert vorgenommen werden könnte, so zum Beispiel das An- und Abschalten des Stroms durch IP-Unterstützung; es sollte die ferngesteuerte Simulation eines Stromausfalls möglich sein.

Für die Umsetzung des Szenarios (Anwendungsfall 4.1.3) muss sichergestellt sein, dass sicherheitskritische Grenzen für Strom, Spannung etc. eingehalten werden. Dies muss auch bei Fehlsteuerung Bestand haben. Ferner muss ein Feuermelder im Raum vorhanden sein.

FS09 Anzahl der Steuergeräte **Priorität 1**

Um den Anwendungsfall 4.1.4 zu realisieren, müssen mindestens 10 Steuergeräte möglich sein; die Stromversorgung muss 5A bereitstellen können.

4.2.1.3 Anforderungen an die Benutzerschnittstelle

FB01 Grafische Rückmeldung **Priorität 2**

Wie im Anwendungsfall 4.1.1 bereits beschrieben, muss grafisch sichtbar sein, dass sich etwas tut. Infolgedessen muss das Fahrzeugnetz an einen PC angeschlossen werden. Folglich ist ein PC mit einer Schnittstellenkarte für den gewählten Bus und entsprechender Software notwendig (die bereits beschaffte Lösung mit CANoe muss benutzt werden).

FB02 Aufbau **Priorität 2**

Aus dem Anwendungsfall 4.1.1 ergibt sich weiterhin, dass Reaktionen des Systems auch im Aufbau sichtbar sein sollen. Folglich sind Komponenten mit sichtbaren Funktionen notwendig, zum Beispiel ein Scheinwerfer, ein Dachelement oder ein Gangwahlschalter. Die Komponenten sollen auch aus einiger Entfernung sichtbar sein. Ferner soll die reale Funktionalität der Komponenten sichtbar sein (Beispiel: der Scheinwerfer geht wirklich an und nicht „nur“ ein Licht am PC)

FB03 Demonstrator **Priorität 3**

Hinsichtlich des Anwendungsfalls 4.1.1 wäre es schön, wenn das Auto „virtuell“ gefahren werden könnte. Hier ist eine Anbindung an eine Open Source Fahrsimulation (siehe SB11) denkbar.

FB04 Externe Sicht auf den Aufbau **Priorität 2**

Ausgehend vom Anwendungsfall 4.1.3 sollte das System einen fest definierten Aufbau haben, der zu bestimmten Zeiten gegeben ist und von außen, zum Beispiel mittels einer Webcam, betrachtet werden kann. Dies ermöglicht ein visuelles Feedback für die externen Wissenschaftler.

FB05 Zugriff auf den Bus **Priorität 1**

Der Anwendungsfall 4.1.5 bringt die Anforderung an einen einfachen Auslesezugang mit sich. Somit wird es Wartungstechnikern erleichtert, den Bus auslesen zu können.

FB06 Nachrichtenstruktur **Priorität 1**

Ferner liefert der Anwendungsfall 4.1.5 die Anforderung, dass die über den Bus laufenden Nachrichten klar definiert sein müssen.

FB07 Nachweisaufbau **Priorität 1**

Es muss ein einfacher Nachweis-/Testaufbau vorhanden sein, der die Funktionstüchtigkeit des Fahrzeugnetzes und des Messaufbaus belegt.

4.2.2 Anforderungen an die Steuerumgebung und ihre Benutzerschnittstelle

4.2.2.1 Allgemeine Anforderungen

SA01 Skalierbarkeit **Priorität 1**

Das System muss in Module gegliedert sein, um das Erweitern des Systems um weitere Komponenten problemlos zu ermöglichen. Außerdem muss das System in mehrere unabhängige Teilsysteme gekapselt werden können.

SA02 Echtzeit **Priorität 1**

Es muss gewährleistet sein, dass zur Laufzeit getätigte Eingaben in Echtzeit verarbeitet werden.

SA03 Verfügbarkeit **Priorität 1**

Es muss eine möglichst hohe Verfügbarkeit des Systems sichergestellt werden.

SA04 Robustheit **Priorität 1**

Die Steuerumgebung muss robust gegenüber Bedienungsfehlern sein, um Schäden am System zu vermeiden.

SA05 Kosten **Priorität 1**

Der Arbeitsaufwand und die Kosten für die Steuerumgebung dürfen den Rahmen unserer Möglichkeiten nicht sprengen und müssen im richtigen Verhältnis zum Aufwand und den Kosten der anderen Aufgaben stehen.

SA06 Ausmaße **Priorität 1**

Die Steuerumgebung muss mit den anderen Systemteilen in unserem Labor aufbaubar sein.

SA07 Reaktive Simulation **Priorität 3**

Es soll eine reaktive Simulation, also Reaktion auf I/O-Signale in Echtzeit, möglich sein.

4.2.2.2 Anforderungen an die Steuerknoten

SK01 Simulation von nicht vorhandener Hardware **Priorität 1**

Komponenten, die nicht physikalisch vorhanden sind aber vom System benötigt werden, können simuliert werden. Dabei müssen simulierte Komponenten, die das System beeinflussen (z.B. Sensoren), gültige Eingaben für vorhandene Komponenten produzieren. Andererseits ist es erforderlich, dass simulierte Komponenten, die vom System Einfluss nehmen (z.B. Getriebe, Motor), Eingaben vorhandener Komponenten verarbeiten und entsprechend darauf reagieren.

SK02 Simulation des Fahrzeugbetriebs **Priorität 1**

Für reale Randbedingungen muss der Fahrzeugbetrieb simuliert werden. Um das Verhalten des Systems beim Betrieb des Fahrzeugs beobachten zu können, müssen zum einen reale Funktionen simuliert werden können. Dazu gehört beispielsweise die Simulation der Fahrt. Zum anderen müssen unterschiedliche Umweltverhältnisse simuliert werden können, um das Verhalten des Systems bei verschiedenen Umgebungseinflüssen testen zu können. Dazu gehören beispielsweise die Beschaffenheit der Straße (z.B. Kurve, Steigung) sowie das Wetter (z.B. Helligkeit, Regen).

SK03 Flashen und Resetten von Komponenten **Priorität 1**

Für alle Anwendungsfälle ergibt sich, dass man einzelne oder mehrere Komponenten gleichzeitig flashen oder zurücksetzen können muss. Dazu müssen die Steuerknoten Zugriff auf den Programmspeicher der Komponenten haben.

SK04 Testen von Aktoren **Priorität 3**

Aus dem Anwendungsfall 4.1.5 ergibt sich, dass die Möglichkeit, einzelne Systemkomponenten zu testen, wünschenswert wäre. Daher wäre es schön, wenn sich die Aktoren zum Testen über CAN-Nachrichten vom Steuerknoten ansprechen ließen.

4.2.2.3 Anforderungen an die Benutzerschnittstelle

SB01 Physikalische Eingabe **Priorität 1**

Für die Anwendungsfälle 4.1.1, 4.1.2 und 4.1.4 ergibt sich die Anforderung, dass das System oder Teile des Systems bei der lokalen Benutzung durch die Betätigung von Schaltern oder anderen physikalischen Eingabegeräten steuerbar sein muss.

SB02 Grafische Eingabe **Priorität 2**

Das System soll per Software über eine grafische Schnittstelle steuerbar sein. Dabei muss die Eingabe sowohl über grafische Bedienoberflächen als auch über die direkte Eingabe physikalischer Werte möglich sein.

SB03 Betätigung physikalischer Schalter durch Software **Priorität 1**

Besonders durch den Anwendungsfall 4.1.3, aber auch durch die anderen Anwendungsfälle ergibt sich, dass das Betätigen physikalischer Schalter über die Software möglich sein muss. Ein wichtiges Beispiel dafür ist etwa das Ein-/Ausschalten des Systems oder von Teilen des Systems.

SB04 Flashen und Resetten von Komponenten **Priorität 1**

Aus allen Anwendungsfällen folgt die Anforderung, dass die Benutzerschnittstelle eine (komfortable) Möglichkeit zur Verfügung stellt, das Flashen und Resetten mehrerer Komponenten gleichzeitig durchzuführen.

SB05 Benutzung durch mehrere Gruppen **Priorität 1**

Wie im Anwendungsfall 4.1.2 beschrieben, müssen mehrere Gruppen gleichzeitig an verschiedenen Teilen des Systems arbeiten können. Daher muss die Software so konfigurierbar sein, dass sie nur ein festgelegtes Teilsystem steuert. Andere Teilsysteme dürfen nicht beeinflusst werden.

SB06 Remote-Zugriff **Priorität 2**

Wie im Anwendungsfall 4.1.3 beschrieben, soll der Fernzugriff auf das System ermöglicht werden. Daraus folgt die Anforderung, dass die Steuerung des Systems über das Internet möglich sein soll. Bei dieser Remote-Bedienung sollte die Zugriffssicherheit gewährleistet sein.

SB07 Eingeschränkte Benutzerrechte **Priorität 2**

Für die Anwendungsfälle 4.1.2 und 4.1.3 sollte der Zugriff auf das System eingeschränkt sein, um zu verhindern, dass das System durch fehlerhafte Bedienung Schaden nimmt. Daher sollte beim Einsatz in der Lehre und beim Remote-Zugriff der Zugriff auf sicherheitskritische Funktionen und Eigenschaften des Systems eingeschränkt werden können.

SB08 Festlegen von Simulationsabläufen **Priorität 2**

Aus dem Anwendungsfall 4.1.1 ergibt sich, dass es möglich sein soll, Simulationsabläufe festzulegen und wiederholt ausführen zu lassen. Dies ist auch für die Anwendungsfälle 4.1.2, 4.1.3 und 4.1.4 sinnvoll. Diese Abläufe sollten zum einen durch das Aufzeichnen zur Laufzeit vorgenommener Eingaben und zum anderen durch Festlegen von Eingabefolgen bereits vor der Laufzeit möglich sein. Die definierten Simulationsabläufe sollten jederzeit abrufbar sein und wiedergegeben können.

SB09 Test von Aktoren **Priorität 3**

Aus dem Anwendungsfall 4.1.5 ergibt sich, dass die Möglichkeit, einzelne Systemkomponenten zu testen, wünschenswert wäre. Daher wäre es schön, wenn die grafische Schnittstelle eine Möglichkeit zum Testen einzelner Komponenten zur Verfügung stellen würde.

SB10 Konsolenbasierte Nutzerschnittstelle **Priorität 1**

Es ist eine konsolenbasierte Benutzerschnittstelle notwendig, die zur Programmierung des Systems vorgesehen ist. Sie soll auch als Basis für eventuelle grafische Erweiterungen und des remote Zugangs dienen.

SB11 Fahrsimulation **Priorität 3**

Es wäre schön, wenn Benutzer am PC eine angebundene Open Source-Fahrsimulation spielen können, durch die die Funktionen des Aufbaus und die Funktionalität der Nachrichtenübertragung im System auch ohne technische Kenntnisse greifbar werden.

4.2.3 Anforderungen an die Messumgebung und ihre Benutzerschnittstelle

4.2.3.1 Allgemeine Anforderungen

MA01 Skalierbarkeit **Priorität 1**

Die Messumgebung sollte so aufgebaut werden, dass die Experimentierplattform in Zukunft um weitere Steuergeräte und Aktoren erweitert werden kann.

MA02 Modularität **Priorität 1**

Der modulare Charakter der Experimentierplattform soll bei der Messumgebung ebenfalls umgesetzt werden. Studenten können so in Arbeitsgruppen getrennt an einzelnen Modulen arbeiten (Anwendungsfall 4.1.2). Auch für Demonstrationen lassen sich speziell ausgewählte Module vorführen (Anwendungsfall 4.1.1).

MA03 Übersichtlichkeit **Priorität 2**

Die Anzahl der benötigten Mess- und Steuerboards sollte möglichst gering gehalten werden, um die Fehlerquellen beim Auf- und Abbau nach dem Transport für Demonstrationen zu reduzieren (Anwendungsfall 4.1.1).

MA04 Kosten **Priorität 2**

Sowohl die Kosten für die Hard- und Software als auch die benötigte Arbeitszeit sollten sich im Rahmen halten.

MA05 Schnittstelle zum PC **Priorität 1**

Es wird eine Anbindung an den PC benötigt, um Messwerte an den Rechner zu übertragen und somit die Darstellung, Protokollierung und Auswertung zu ermöglichen. Eine ausreichende Bandbreite wird dafür vorausgesetzt.

MA06 Remote-Messungen **Priorität 3**

Remote-Messungen sollten realisierbar sein, um anderen Universitäten die Nutzung der Experimentierplattform zu ermöglichen (Anwendungsfall 4.1.3).

MA07 Stromversorgung **Priorität 1**

Die Messgeräte sollen über eine vom Fahrzeugnetz getrennte Stromversorgung verfügen, da im Stromnetz der Versuchsplattform unterschiedliche Energieversorgungszustände simuliert werden sollen. Aufgrund der Remote-Messungen muss auch die Stromversorgung für die Messumgebung abschaltbar sein, z. B. über IP-Steckdosen (Anwendungsfall 4.1.3).

MA08 I/O-Pins **Priorität 1**

Die im aufzubauenden Fahrzeugnetz verwendeten Mikrocontroller-Boards besitzen I/O-Pins, an die beispielsweise externe Taster, Schalter, LEDs, etc. angeschlossen werden können. Diese I/O-Pins sollen von der Messumgebung überwacht werden, um Reaktionen der Boards, beispielsweise aufgrund von ansteigendem Strombedarf, auswerten zu können.

4.2.3.2 Anforderungen an die Messknoten

MK01 Exakte Messungen **Priorität 1**

Der fließende Strom der einzelnen Steuergeräte soll möglichst exakt gemessen werden. Die Genauigkeit sollte hoch genug sein, um den Stromverbrauch von einzelnen Tasks zu bestimmen. Aus bekannten Forschungsprojekten ergeben sich dabei eine Abtastung von mindestens 250kS/s verteilt auf 16 Kanäle und eine Auflösung von 10, 15 und 16 Bit.

MK02 Galvanische Trennung **Priorität 1**

Der Versuchsaufbau soll nicht von der Messumgebung beeinflusst werden. Die galvanische Trennung schützt zusätzlich die Messumgebung und den Versuchsaufbau gegenseitig vor Kurzschlüssen oder Spannungsspitzen.

MK03 Buskommunikation **Priorität 1**

Die Kommunikation auf dem Bus soll von der Messumgebung beobachtet werden, um die Busauslastung bestimmen zu können.

MK04 Sendende Kommunikationsteilnehmer **Priorität 1**

Die Messumgebung soll die Sendeaktivitäten eines Knotens erfassen.

MK05 Busübergreifende Kommunikation **Priorität 2**

Die Knoten des Fahrzeugnetzes können an verschiedenen Bussen angeschlossen sein. Sind bei der Kommunikation auf den Bussen auch die Teilnehmer bekannt, kann bestimmt werden, welche Knoten häufig oder nur selten miteinander kommunizieren und ggf. auch, ob viel Kommunikation busübergreifend stattfindet.

MK06 Empfangende Kommunikationsteilnehmer **Priorität 3**

Die Messumgebung soll die Empfänger von versandten Nachrichten protokollieren.

MK07 Zeitstempel **Priorität 1**

Um die Messwerte der einzelnen Messknoten auswerten zu können, wird ein Zeitstempel für die Strommessung benötigt. Der Zeitpunkt, zu dem ein bestimmter Knoten mit anderen Knoten auf dem Bus kommuniziert, soll von der Messumgebung ebenfalls genau festgehalten werden. Dadurch soll bestimmt werden, ob die Kommunikation zu einer veränderten Stromaufnahme des Knotens führt. Der Zeitpunkt ist wichtig, um einen Abgleich mit der Messung der Stromaufnahme zu einem bestimmten Zeitpunkt zu ermöglichen.

4.2.3.3 Anforderungen an die Benutzerschnittstelle

MB01 Messwerte aufzeichnen **Priorität 1**

Messwerte sollen mit einem Zeitstempel protokolliert und abgespeichert werden.

MB02 Messwerte in Echtzeit **Priorität 2**

Zusätzlich zu Anforderung MB01 sollen dem Benutzer die aktuellen Messwerte über die Schnittstelle ohne Verzögerung präsentiert werden.

MB03 Grafische Darstellung von Messwerten **Priorität 2**

Damit der Benutzer Veränderungen der Messwerte erkennt, sowie eine bessere zeitliche Anschauung der Messwerte bekommt, müssen diese grafisch aufbereitet werden. Dies soll mittels einer übersichtlichen und leicht zu verstehenden Darstellung erreicht werden (z. B. Koordinatensystem, Balkendarstellung, Darstellung über den zeitlichen Verlauf, etc.).

MB04 Benutzerschnittstelle für Remote-Zugriff **Priorität 3**

Für die Realisierung des Anwendungsfalls 4.1.3 wird eine Remote-Benutzerschnittstelle benötigt.

MB05 Benutzerschnittstelle für Vorführzwecke **Priorität 3**

Die Benutzerschnittstelle für Vorführzwecke sollte im Umfang reduziert und grafisch weiter aufbereitet werden, z. B. Darstellung der Geschwindigkeit durch einen virtuellen Tacho (Anwendungsfall 4.1.1). Dies kann über eine Fahrsimulation (siehe SB11) erfolgen.

MB06 Auswertung von protokollierten Messungen **Priorität 2**

Auf den protokollierten Messdaten sollen Berechnungen und Vergleiche stattfinden können.

Kapitel 5

Grundlagen

In diesem Kapitel werden die Hardware- und Softwarekomponenten, die für die Realisierung des Projekts AutoLab genutzt wurden, vorgestellt.

5.1 Hardware-Komponenten

5.1.1 CAN-Bus

Eine Herausforderung bei der selbst zu entwickelnden Hardware war die Erstellung einer Verkabelung für den CAN-Bus, welche zum einen möglichst flexibel, aber auch robust und einfach einzusetzen ist. Da der CAN-Bus Signale mit Frequenzen von bis zu 1 MHz über den Bus schickt, haben wir uns auf der Suche nach einem passendem Kabel für Shielded Twisted Pair Kabel entschieden. Diese Variante der Netzwerkkabel überträgt Signale störfrei noch bei 100Mhz und mehr. Dadurch und durch die Tatsache, dass wir für den CAN-Bus nur 2 der 8 Adern im Netzwerkkabel benutzt haben, ist unsere Bus-Verkabelung generell durchaus auch auf andere Bussysteme wie LIN oder Flexray erweiterbar. Der Kabelaufwand würde bei dieser Zusammenlegung der Signale dann stark reduziert, da ja nur das vorhandene Bus(-kabel) genutzt werden müsste.

Berechnungen ergaben, dass bei den verwendeten Geschwindigkeiten Kabellängen von ca. 30 cm zwischen Bus und Busteilnehmer die Signale noch nicht stören. Durch diesen Umstand konnten wir Y-Stecker-ähnliche Kabel löten, bei denen das lange Kabel jeweils der Bus, das kurze die Zugangsleitung zum Busteilnehmer ist. Der große Vorteil liegt hierbei natürlich bei der Verkabelung, die ganz ohne spezielle Hubs oder Verteiler auskommt.

Zusätzlich zu den verwendeten Kabeleigenschaften ist der CAN-Bus an sich mit seiner Übertragungsart sehr robust, so dass wir bei umfangreichen Tests der konfektionierten Kabel keine Beeinträchtigung feststellen konnten.

5.1.2 TriBoards

Die Firma Infineon stellt 32-Bit-Mikrocontroller, bzw. deren Kerne, unter der Produktbezeichnung TriCore her. Diese Mikrocontrollerfamilie findet besonders im Automotive-Bereich Einsatz, da die Mikrocontroller zusätzliche DSP-Funktionalität aufweisen.

Von Infineon werden neben den reinen Mikrocontrollern auch Evaluations-Boards unter dem Namen TriCore-Board vertrieben. Aufgebaut sind die Boards auf 100mm x 160mm EURO-Boards. Es gibt diese TriBoards mit TriCore TC176x- und TC179x -Mikrocontroller. Die TriBoards werden mit einer Spannung zwischen 5,5 und 60 V DC betrieben.

Zur Ausstattung der TriCore-Mikrocontroller gehören neben den üblichen Timern und I/O-Pins unter anderem A/D-Wandler, CAN-Schnittstellen und RS-232-Schnittstellen. Alle Pins der TriCores sind über 80-polige Stift-/Buchsenleisten nach außen geführt. Zwei der CAN-Schnittstellen sind zusätzlich über jeweils eine 2x5-polige Stiftleiste erreichbar. Das Board weist eine 9-polige Sub-D Schnittstelle für die erste RS-232-Schnittstelle auf, sowie eine 25-polige Sub-D-Schnittstelle für das OSCD (On Chip Debug System) auf.

TriBoard TC1766

Das TriBoard TC1766 zeichnet sich durch die folgende Konfiguration aus:

- CPU:
  - TC1766
- Takt:
  - 80 MHz Systemtakt (15 MHz Quarz)
- Schnittstellen:
  - 2x RS-232 (1x 9-pol. Sub-D, 1x BERG10)
  - 2x CAN (2x BERG10 für CAN 0,1)
  - 2x OCDS (1x BERG16 & 25-pol. Sub-D, 1x SAMTEC QSH-030-01-F-D-A)
  - 4x 80-pin-Anschlüsse (männlich) mit allen I/O-Signalen
  - 4x 80-pin-Anschlüsse (weiblich) mit allen I/O-Signalen

Zur Peripherie des TriCore TC1766 gehören folgende Module:

- Systemtimer
- General Purpose Timer Array V4 (GPTA)
- Asynchrone Serielle Schnittstelle (ASC0 und ASC1)
- Synchrone Seriellel Schnittstelle (SSC0 und SSC1)

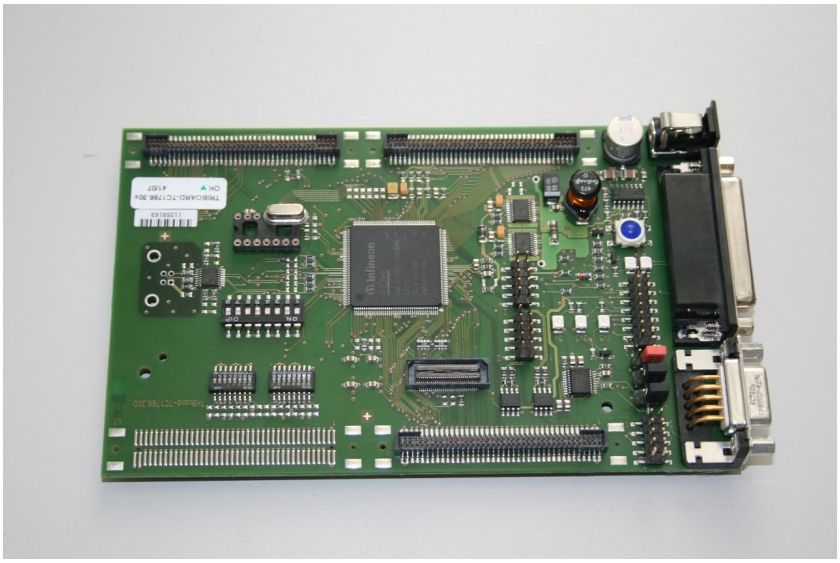


Abbildung 5.1: TriCore TC1766

- MultiCAN (CAN0 und CAN1)
- Micro Second Bus Schnittstelle (MSC)
- Micro Link Serial Bus Schnittstelle (MLI0 und MLI1)
- A/D-Wandler (ADC0 und FADC) mit 36 analogen Eingängen

TriBoard TC1796

Das TriBoard TC1796 zeichnet sich durch die folgende Konfiguration aus:

- CPU:
  - TC1796
- Speicher:
  - 4 MB Burst-Flash
  - 1 MB asynchrones SRAM
- Takt:
  - 150 MHz Systemtakt (20 MHz Quarz)
- Schnittstellen:
  - 2x RS-232 (1x 9-pol. Sub-D, 1x BERG10)

- 4x CAN (2x BERG10 für CAN 0,1)
- 2x OCDS (1x BERG16 & 25-pol. Sub-D, 1x SAMTEC QSH-030-01-F-D-A)
- 4x 80-pin-Anschlüsse (männlich) mit allen I/O-Signalen
- 4x 80-pin-Anschlüsse (weiblich) mit allen I/O-Signalen

Zur Peripherie des TriCore TC1796 gehören folgende Module:

- Systemtimer
- General Purpose Timer Array V4 (GPTA0, GPTA1 und LTCA2)
- Asynchrone Serielle Schnittstelle (ASC0 und ASC1)
- Synchrone Serielle Schnittstelle (SSC0 und SSC1)
- MultiCAN (CAN0, CAN1, CAN2 und CAN3)
- Micro Second Bus Schnittstelle (MSC0 und MSC1)
- Micro Link Serial Bus Schnittstelle (MLI0 und MLI1)
- A/D-Wandler (ADC0, ADC1 und FADC) mit 44 analogen Eingängen

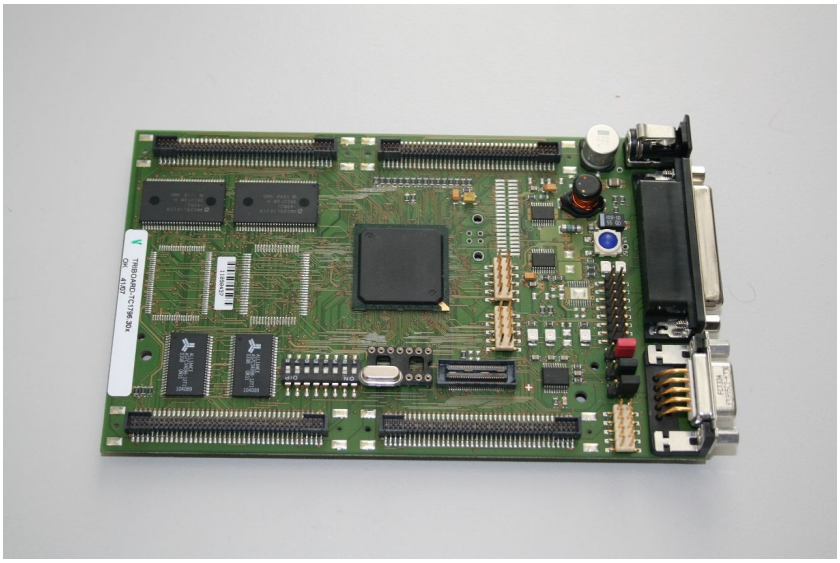


Abbildung 5.2: TriCore TC1796

5.1.3 Scheinwerfer

Die mit 12V betriebenen Scheinwerfer kommunizieren über CAN mit einer Übertragungsrate von 500kBaud. Ein Scheinwerfer hat die folgenden funktionellen Bestandteile:

- Reflektor für Abblend- und Fernlicht
- Blinker
- LED-Lichtleiste für Tagfahrlicht

Der Anschlussstecker enthält folgende Anschlüsse:

- CAN-Anschlüsse
- Versorgung AFS-Modul
- Masse für AFS-Modul
- Versorgung Abblendlicht
- Masse für Abblendlicht
- Versorgung Blinker
- Versorgung für Standlicht
- Versorgung für Tagfahrlicht
- Masse für Stand-, Tagfahr-, Blink- und Fernlicht

Über CAN-Nachrichten lassen sich folgende Funktionen steuern:

- Soll-Position des Lichtkegels vertikal (AFS-Funktion)
- Soll-Position des Lichtkegels horizontal für Kurvenlicht (LWR-Funktion)

Explizit nicht über CAN ansteuern lassen sich als folgende Funktionen

- Abblendlicht ein / aus
- Blinker
- Fernlicht
- Dimmung des Tagfahrlichts

Hier muss die Versorgungsleitung selbst geschalten werden. Die Verwendung eines Relais ist notwendig, um den Blinker über CAN zu schalten.

Abbildung 5.3: Der Scheinwerfer

5.1.4 Sensorcluster

Der mit 12V betriebene Sensorcluster kommuniziert über CAN mit einer Übertragungsrate von 500kBaud. Im realen Betrieb bei Audi liefert der Cluster unter anderem die Informationen für das ESP-System. Wir nutzten die Informationen hingegen zur Ausrichtung des Scheinwerfers. Am Cluster selbst sind die Beschleunigungsachsen beschriftet (x=Längs (vor/zurück); Y=Quer (links/rechts); Z=Hoch). Innerhalb des Clusters befinden sich 2 Sensoren, die die folgenden Daten versenden:

- Beschleunigung entlang der Fahrzeugrichtung (X-Achse) in G
- Beschleunigung quer zur Fahrzeugrichtung (Y-Achse) in G
- Winkelbeschleunigung in rad/s^2 (Von beiden Sensoren, also redundant vorhanden)
- Drehrate entlang der Z-Achse in rad/s (Von beiden Sensoren, also redundant vorhanden)
- Zusätzliche Botschaftszähler und Checksummen

5.1.5 Dachelement

Das Dachelement, welches als Gateway für die Kommunikation zwischen Fahrzeugknoten und lokalen Subsystemen dient, vereint eine Vielzahl von Einzelkomponenten in einem Modul.

Sensorsignale, wie z.B. EC-Spiegel, werden in dem Dachelement verarbeitet. Des weiteren erfüllt das Dachelement folgende Funktionen:

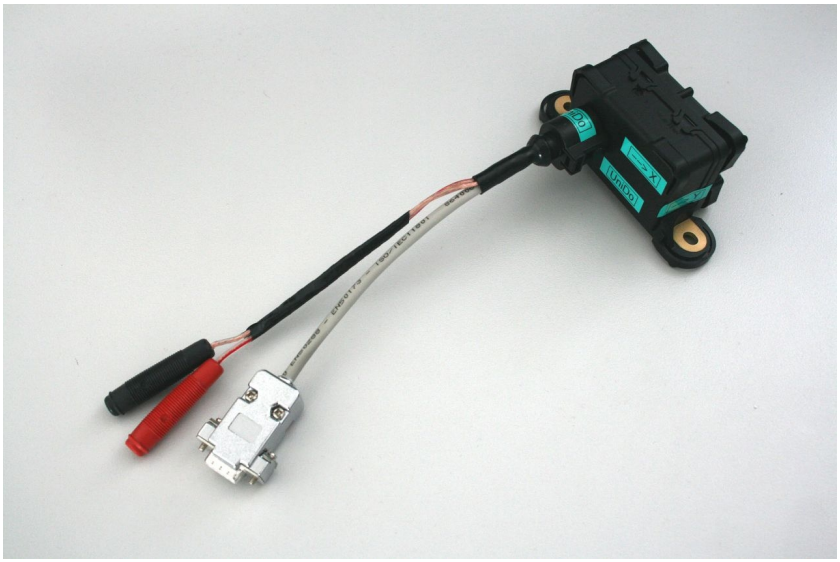


Abbildung 5.4: Der Sensorcluster

- Ansteuerung des Schiebe-/Hebedachs
- blendefreies Lese- und Ambientelicht
- Integration der Innenraumüberwachung
- Auswertung der Scheibensensorik

Das Dachelement besitzt vier verschiedene Tasten (siehe Abbildung 5.5). Über die linke und rechte Taste können die beiden äußeren Leselampen und über die mittlere Taste das Ambientelicht eingeschaltet werden.

Durch das Drücken der Taste im Zentrum des Bedienpanels wird das Hebedach angesteuert. Gleichzeitig fungiert diese Taste als Schieber, mit dem das Schiebedach, je nachdem, in welche Richtung der Schieber bewegt wird, geöffnet oder geschlossen werden kann.

Die hier beschriebenen Funktionen werden alle über den Freescale MC9S12DB128 angesteuert. Der Mikrocontroller wird über einen speziellen Baustein (SBC: System Basis Chip), welcher sich auf dem Bord befindet, in einen Wach-/Schlafzustand versetzt.

Zum Flashen des Controllers wird die BDM-Schnittstelle des inDart-One-Programmiergerätes verwendet.

Außerdem gibt es an dem Dachelement noch vier Schnittstellen, wodurch es mit Subsystemen verbunden werden kann. Um das Dachelement in Betrieb nehmen zu können, ist eine Mindestverbindung (12V, GND, CAN) über Modul 1 notwendig.

Abbildung 5.5: Dachelement

5.1.6 Gangwahlschalter

Von der Firma Kostal wurde uns ein Gangwahlschalter aus dem BMW X5 zur Verfügung gestellt.

Der Gangwahlschalter hat zwei Modi: zum einen die Automatik-Stellung und zum anderen den Tiptronic-Modus. Um in den Tiptronic-Modus zu gelangen, wird der Schalthebel nach links in eine zweite Schaltgasse geführt. In dem Automatik-Modus kann zwischen dem Fahrmodus, der Leerlaufstellung und dem Rückwärtsgang gewählt werden.

Des weiteren verfügt der Gangwahlschalter über drei Knöpfe. Über den P-Knopf kann der Parkmodus des Fahrzeugs aktiviert werden. Der Sport-Knopf ermöglicht dem Fahrer einen sportlicheren Fahrmodus. Der unbeschriftete Knopf an der linken Seite des Schalthebels löst die Arretierung des Rückwärtsgangs.

In dem Gangwahlschalter ist der Mikrocontroller MC9S12DG128 von Freescale verbaut. Der Mikrocontroller verfügt über eine 16-Bit-CPU und hat folgende Eigenschaften:

- 128 KB Flash EEPROM
- 2 KB EEPROM
- 8 KB RAM
- 2 SCI-Schnittstellen
- 2 SPI-Schnittstellen

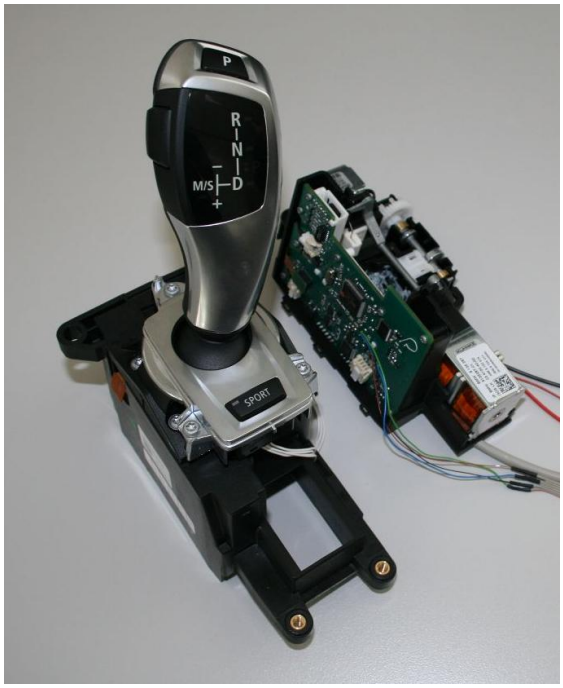


Abbildung 5.6: Gangwahlschalter

- max. 25 MHz Busfrequenz
- 2 CAN-Schnittstellen (1 MBit pro Sekunde)
- 2 8-Kanal A/D-Wandler mit 10 Bit Auflösung
- 5V Versorgungsspannung

Der Mikrocontroller kann über die BDM-Schnittstelle des inDart-One geflasht und debuggt werden.

Der Gangwahlschalter verfügt über einen 6-poligen Anschluss, an dem die 12V Versorgungsspannung und der CAN-Bus angeschlossen werden.

5.1.7 Schaltermodul Lenksäule

Allgemeine Beschreibung

Das Schaltermodul Lenksäule (SMLS) der Firma Audi besitzt einen modularen Aufbau und besteht aus folgenden Komponenten:

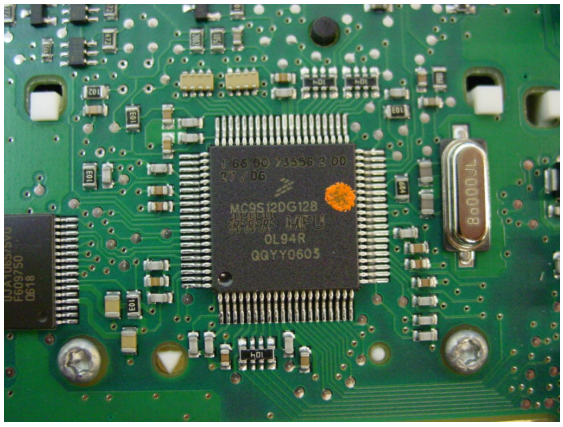


Abbildung 5.7: Mikrocontroller des Gangwahlschalters

- Lenksäulenelektronik J527 zur Signalumsetzung und Verarbeitung des *Antriebs- und Komfort-CAN-Bus*
- Blinkerschalter
- Wischerschalter mit Intervall-Potenziometer
- Lenkwinkelsensor mit Wickelfeder für den Fahrerairbag

Optional kann ein separater Lenkstockschalter zur Bedienung der Geschwindigkeitsregelanlage angeschlossen werden.

Des weiteren erfasst die Lenksäulenelektronik die Signale des Zündanlassschalters. Diese Signale werden über herkömmliche Leitungen an die Lenksäulenelektronik geleitet. Die Schaltpositionen des Zündschlosses werden von der Elektronik verarbeitet und dem *Komfort-CAN-Bus* zur Verfügung gestellt. Zusätzlich werden diese Informationen dem CAN-Bus-System *Antrieb* zur weiteren Verarbeitung bereitgestellt.

Die Erkennung der jeweiligen Schalterstellung der einzelnen Komponenten geschieht über eine Spannungscodierung anhand unterschiedlicher Widerstandswerte. Diese Werte sind in Abhängigkeit der jeweiligen Stellung definiert.

Die Lenksäulenelektronik analysiert diese Stellungsinformationen und leitet diese über den *Komfort-CAN-Bus* an das zentrale Steuergerät weiter.

Die Bestimmung des aktuellen Lenkwinkels geschieht über optische Elemente in der Lenksäulenelektronik und wird über den *Antriebs-CAN-Bus* an das Steuergerät zur weiteren Verarbeitung geleitet.

Abbildung 5.8: Standardlenksäulenmodul

Das Steuergerät für die Lenksäulenelektronik J527

Das Steuergerät für die Lenksäulenelektronik J527 ist das zentrale Element des Gesamtsystems *Schaltermodul Lenksäule*. Diese Elektronik verarbeitet die von den einzelnen Komponenten erfassten Informationen und leitet diese über den *Komfort-* oder den *Antriebs-CAN-Bus* an das zuständige Steuergerät weiter.

Das Steuergerät besitzt einen Freescale MC68HC908AZ60ACFU Mikrocontroller mit den folgenden technischen Daten:

- 8,4 Mhz interne Bus-Frequenz
- 60 KB Flash Electrically Erasable Read-Only Memory (FLASH)
- FLASH Data Security
- 1 KB On-Chip Electrically Erasable Programmable Read-Only Memory mit Security Option (EEPROM)
- 2 KB On-Chip RAM
- Serial Peripheral Interface Module (SPI)
- Serial Communications Interface Module (SCI)
- 8-Bit, 15-Kanal Analog-zu-Digital Converter (ADC-15)

Zum Flashen des Controllers wird die MON08-Schnittstelle des inDart-One Programmiergerätes verwendet.

5.2 Betriebssystem ProOSEK

ProOSEK ist eine von Elektrobit entwickelte OSEK-Variante mit beiliegendem Konfigurator. Der Konfigurator ermöglicht es, leicht Komponenten, wie zum Beispiel Tasks oder Counter, und Ablaufstrukturen für das Betriebssystem zu erstellen. Nachdem man das Betriebssystem, die jeweiligen Tasks und ggf. weitere Komponenten konfiguriert hat, werden Templates erzeugt. Diese Dateien müssen dann mit dem gewünschten, auf die Situation und Funktion angepassten C-Code gefüllt und anschließend compiliert werden.

Die entstandenen .elf-Dateien können auf die vorhandenen TriBoards gespielt und dort ausgeführt werden.

Anhand der folgenden Grafiken werden wichtige Konfigurationen, die uns bei den ersten Anwendungen des Konfigurators aufgefallen sind, erläutert. Darunter sind auch einige Einstellungen, die essentiell für ein funktionierendes ProOSEK sind. Die Screenshots wurden anhand unseres ProOSEKs für die Fahrzeugfunktionen erstellt:

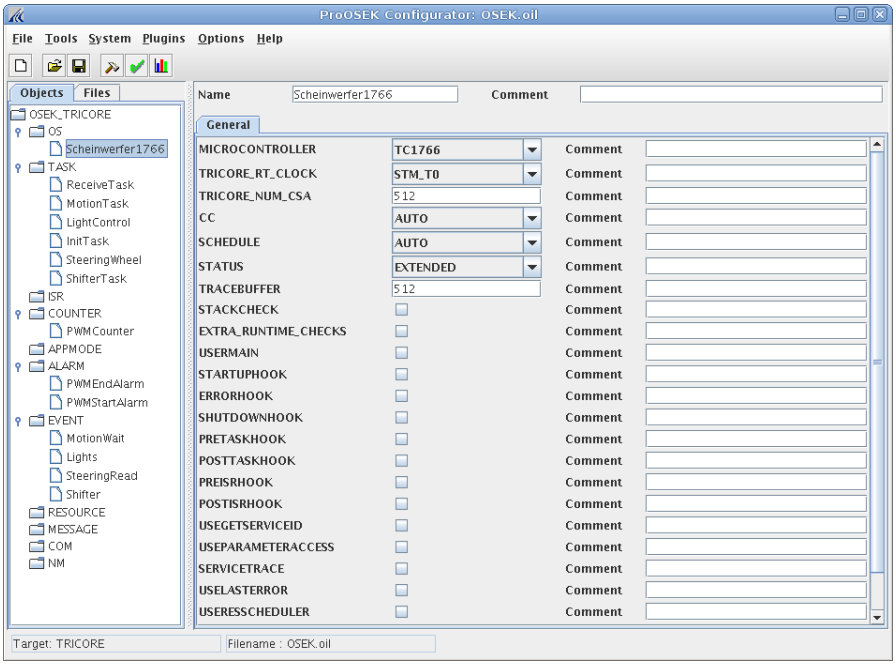


Abbildung 5.9: Konfigurator 1/3

Im ersten Screenshot sind die Grundeinstellungen für das Betriebssystem im OS-Teil des Konfigurators dargestellt:

- Mikrocontroller: Einstellen des Mikrocontroller-Typs des Boards, bei unserem Konfigurator entweder TC1796 oder TC1766.
- TRICORE RT Clock: Aktivierung der Hardware-Counter auf dem Board, um sie im Betriebssystem verwenden zu können.

- Tricore Num CSA: Einstellen der Größe der Context Save Area des Betriebssystems. Hier muss dringend ein vernünftiger Wert eingetragen werden. Ohne Wert baut der Konfigurator das Betriebssystem, ohne eine Fehlermeldung auszugeben. Dieses läuft jedoch direkt nach Start in eine Trap, was wiederum schwer auf diese Einstellung zurückzuführen ist. Mit dem Wert 512 haben wir das OS ohne Probleme betreiben können.
- Tracebuffer: Hier hat sich ebenfalls ein Wert von 512 bewährt.

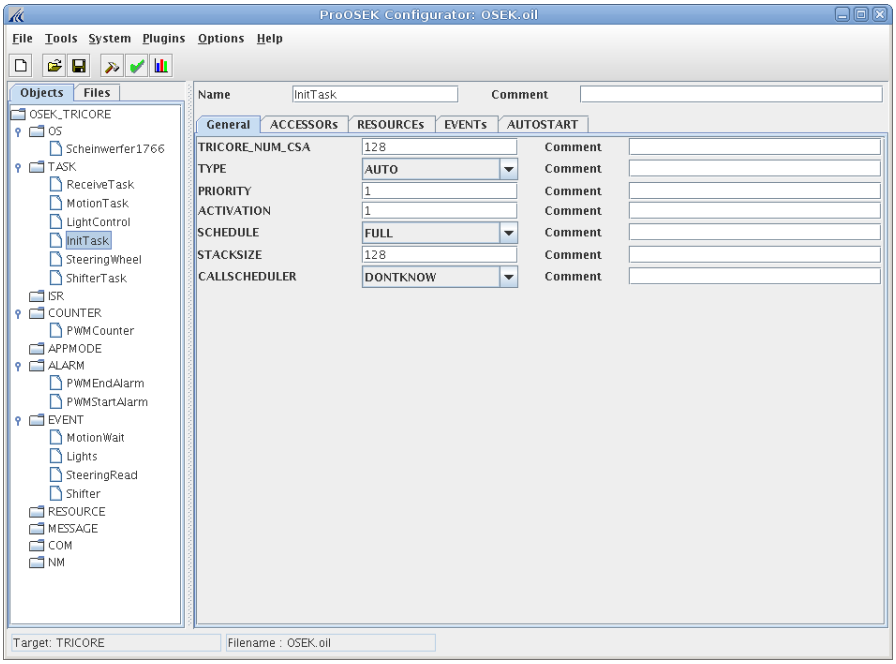


Abbildung 5.10: Konfigurator 2/3

Im nächsten Screenshot werden die Einstellungen für einen Task des OSEK deutlich:

- TRICORE NUM CSA: Wie beim OS-Teil beschrieben, für ProOSEK eine wichtige Einstellung. Ein Wert von 128 hat sich bewährt.
- Stack: Hier hat sich auch ein Wert von 128 bewährt.
- Autostart: Hier kann man einen Task automatisch vom Betriebssystem starten lassen.
- Events/Ressourcen: Hier stellt man die Events oder Ressourcen ein, die der Task benutzt. Diese müssen zuerst im Events/Ressourcen-Menü erstellt werden. Wichtig bei den Events ist, dass bei den Masken jeweils genau ein Bit gesetzt ist d.h. folgende Masken sind beispielsweise erlaubt: 0x01, 0x02, 0x04 etc.

Im dritten Screenshot wird ein Counter eingerichtet.

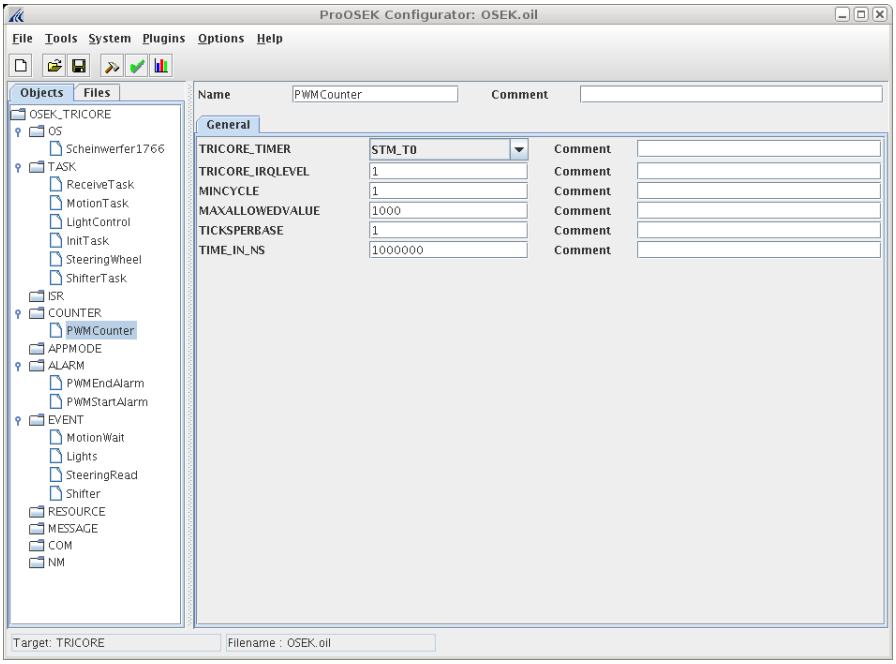


Abbildung 5.11: Konfigurator 3/3

- TRICORE TIMER: Hier gibt es die Auswahl zwischen einem UserCounter, den der User manuell im Programm hochzählt, oder einem beim OS aktivierten Hardware-Timer, der von der Hardware hochgezählt wird.
- Time in NS: Zeit in Nanosekunden, die vergeht, bevor ein Tick passiert.
- Max Value: Tick-Zeit, bei dem der Counter wieder auf 0 zurückgesetzt wird.
- Min Cycle: Minimum Cycle Time für die Alarme, die an diesem Counter hängen.
- Tickspersbase: Unbelegte Variable, die vom User irgendwie benutzt werden kann.

5.3 Werkzeuge

5.3.1 CANoe

CANoe ist ein Werkzeug der Firma *Vector* zur Entwicklung, zum Test und zur Analyse von Netzwerken und Steuergeräten. Aus der Bandbreite der unterstützten Bussysteme war für unser Projekt nur CAN von Bedeutung, so dass wir die entsprechenden Lizenzen für die Softwareversion CANoe 7.0 (educational) erworben haben.

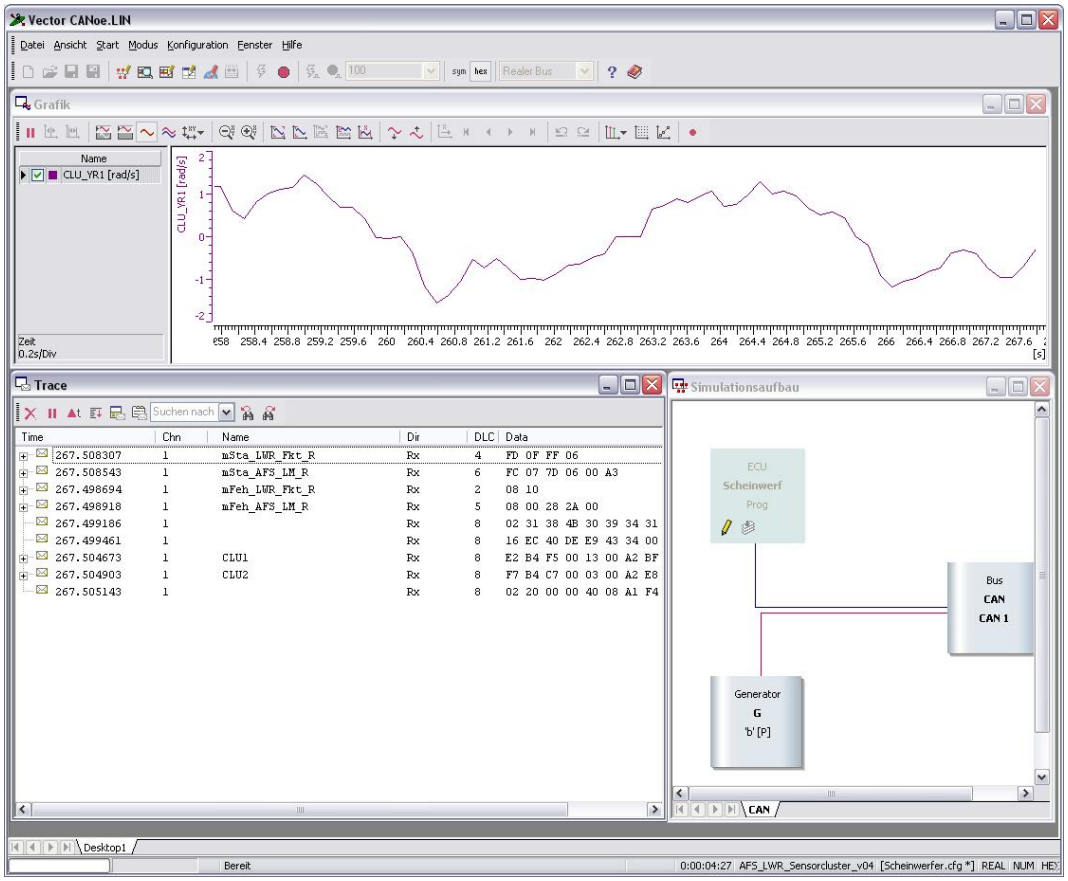


Abbildung 5.12: Unsere Anwendung von CANoe 7.0

Simulationsaufbau

Als Grundlage dient in CANoe ein Simulationsaufbau. Dieser zeigt den CAN-Bus sowie daran angeschlossene Komponenten, die als Knoten dargestellt werden. Am Anfang des Entwicklungsprozesses kann eine Simulation der Komponenten erfolgen. Mittels der CANoe-spezifischen, C-ähnlichen Programmiersprache *CAPL* werden die zukünftigen Verhaltensweisen der Knoten programmiert, so dass sie von CANoe simuliert werden können. Ein Simulationsknoten kann in der frühen Phase der Entwicklung die Aufgaben übernehmen, die zukünftig etwa durch ein real existierendes Steuergerät ausgeführt werden. Es ist sowohl eine komplette Simulation als auch eine Restbussimulation möglich. Um noch schneller Funktionalitäten simulieren zu können, bietet CANoe den sogenannten *Generator-Block*, mit dem man eine bestimmte Nachricht über den Bus schicken kann. Dabei können verschiedene Trigger-Bedingungen verwendet werden.

Datenbanken

Zur Verwaltung und Interpretation der Nachrichten, die über den Bus übermittelt werden, arbeitet CANoe auf Datenbanken, die vom Benutzer zu erstellen sind. Sie enthalten unter anderem den Aufbau der Botschaften mit den darin enthaltenden Signalen, so dass diese von CANoe ausgewertet werden können.

Messaufbau

Um Auswertungen vorzunehmen, gibt es in CANoe einen weiteren wichtigen Bestandteil, der sich *Messaufbau* nennt. Über den Messaufbau können Auswertungswerkzeuge symbolisch an den Bus angeschlossen werden.

Dazu zählen

- Trace-Fenster: Es werden die Aktivitäten auf dem Bus gezeigt und protokolliert. Dabei können die Nachrichten aufgeschlüsselt werden.
- Grafik-Fenster: Die Signale der CAN-Nachrichten können grafisch dargestellt werden, um Verläufe zu verfolgen.
- Logging- und Replay-Block: Die Busaktivitäten können mit Hilfe des Logging-Blocks aufgezeichnet werden. So kann man sie im Nachhinein genauer betrachten, aber auch mit Hilfe des Replay-Blocks erneut online auf den Bus senden. Eine Alternative ist es, in den sogenannten *Offline-Modus* zu wechseln. Dabei liegt nur ein simulierter Bus vor, der als Eingabe das Log verwenden kann. Zu unterscheiden sind diese beiden Funktionen von der Makrofunktion, mit der man (online) mitgezeichnete Aktivitäten auf den Panels und Diagnose-Konsolen wiedergeben kann.
- Weitere Statistik-Werkzeuge, auf die nicht weiter eingegangen wird.

Weitere Funktionalitäten von CANoe

In diesem kurzen Überblick sind nur einige der für uns interessanten Funktionen CANoes grob umrissen. Für weiterführende Informationen sei auf die CANoe-Hilfe verwiesen.

Unsere Einsatzgebiete

Im Folgenden seien die essentiellen Einsatzgebiete vorgestellt, in denen wir CANoe verwendeten.

Frühzeitiger Komponententest Durch die Funktionalitäten des Trace-Fensters sowie des Generator-Blocks und der simulierten Knoten konnten wir frühzeitig unsere Komponenten testen. Dies war z.B. beim Sensorcluster der Fall. Nachrichten des Sensorclusters konnten wir ad hoc im Trace-Fenster sowie grafisch anzeigen lassen. Auch andere Geräte, wie die Scheinwerfer oder TriBoards, konnten wir so mit einfachen Mitteln auf erste Funktionsfähigkeit prüfen. Dabei haben wir den Geräten mit Hilfe des Generator-Blocks oder eines simulierten Knotens entsprechende Nachrichten zugesandt.

Frühzeitige Simulation Durch simulierte Knoten konnten wir erste Funktionalitäten mit Hilfe von CANoe simulieren. So geschehen etwa bei den Scheinwerfern. Hier setzten wir das Kurvenlicht zunächst mit CANoe um.

Restbussimulation Nicht vorhandene Komponenten simulierten wir mit CANoe. So haben wir etwa den Blinkerhebel simuliert, indem wir ein CANoe-Panel erstellten. Bei Betätigung eines Buttons wurde die notwendige CAN-Nachricht an das TriBoard gesandt, welches den Blinkvorgang steuerte.

5.3.2 inDart-One

Der inDart-One ist ein leistungsstarkes Werkzeug zum Programmieren und zum Auffinden, Diagnostizieren und Beheben von Fehlern (Debuggen). Dabei unterstützt der inDart-One die in den von uns verwendeten Hardware-Komponenten eingebauten Freescale Mikrocontroller MC68HC908 (Schaltermodul Lenksäule) und MC9S12 (Dachelement und Gangwahlschalter).

Abbildung 5.13: Frontansicht inDart-One

Die Verbindung zu den Steuergeräten erfolgt entweder über eine MON08- oder BDM-Schnittstelle (siehe Abbildung 5.14) gemäß der Freescale-Spezifikationen.

Abbildung 5.14: Rückansicht inDart-One mit MON08- und BDM-Schnittstelle

Der Anschluss an den PC und die Stromversorgung erfolgt mittels USB.

Um eine Mehrfachprogrammierung realisieren zu können, besteht die Möglichkeit, bis zu 32 inDart-One Geräte am gleichen PC anzuschließen (durch USB-Hubs). Für diesen Zweck ist das MultiBlaze-Werkzeug in das Programmiergerät integriert.

Das Programmiergerät kann vollständig über die enthaltene IPL-One-Programmierbibliothek kontrolliert werden. Diese Bibliothek ist eine DLL, welche alle low-level-Funktionen beinhaltet, um das Gerät einzustellen und die Befehle auszuführen. Die Bibliothek enthält in C geschriebene Funktionen.

Der inDart-One-Programmierer wurde von uns zusammen mit dem CodeWarrior-Entwicklungstool der Firma Metrowerks benutzt (nähere Informationen über das Entwicklungstool finden sich im Abschnitt 5.3.3).

5.3.3 Metrowerks CodeWarrior

Das Entwicklungstool CodeWarrior ist eine integrierte Entwicklungsumgebung (IDE) der Firma Metrowerks. Diese Umgebung ermöglicht es, Software für die Programmierung der von uns verwendeten Steuergeräte zu erstellen, zu flashen und zu debuggen.

Das Tool enthält einen Editor, einen Assembler, einen C-Compiler und einen Debugger.

Technische Details:

- Metrowerks Integrierte Entwicklungsumgebung (IDE)
- Projektmanager mit grafischer Oberfläche (GUI)
- kontextsensitiver Editor
- Optimierender C/C++ Compiler
- ANSI C Compiler
- C++ Compiler nach ANSI/ISO Standard
- Standalone Assembler

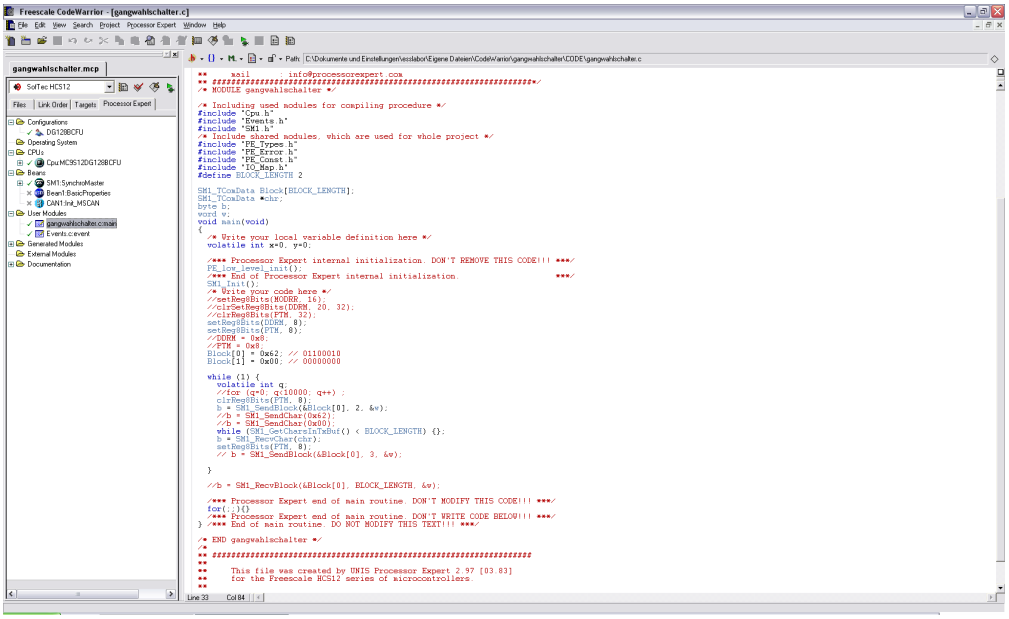


Abbildung 5.15: Screenshot CodeWarrior

- Source- oder Low-Level-Debugger mit Fenster-basierter Arbeitsumgebung

Die CodeWarrior-Entwicklungsumgebung kann mittels des modularen Aufbaus speziell an die Bedürfnisse der von uns verwendeten Freescale-Mikrocontroller MC68HC908 (Schaltermodul Lenksäule) und MC9S12 (Dachelement und Gangwahlschalter) angepasst werden.

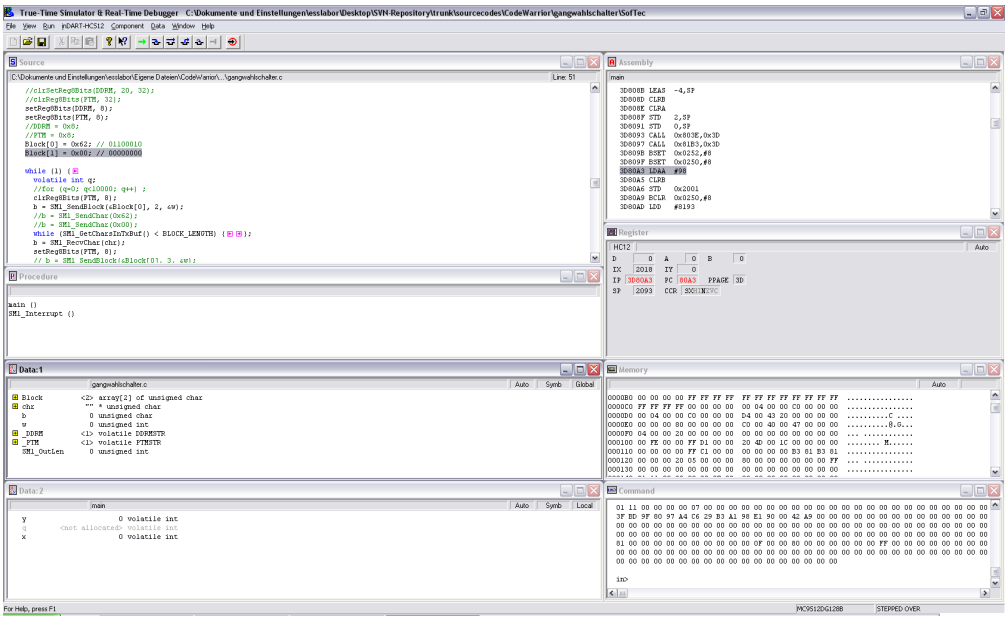


Abbildung 5.16: Screenshot CodeWarrior (Debugger)

Kapitel 6

Erste Entwürfe

Bei unseren ersten Entwürfen ging es darum, verschiedene Netze mit unseren vorhandenen Komponenten zu entwickeln, die verschiedene Funktionalitäten eines Autos implementieren können. Bei Sensoren, die wir noch nicht in Hardware besaßen, haben wir angenommen, dass sie im PC simuliert werden. Bei unseren Entwürfen haben wir zwischen einer logischen- und technischen Systemstruktur unterschieden. Bei der technischen Struktur haben wir versucht, uns nach den Bussen wie sie im Auto vorliegen zu richten. Dies ist der High-Speed-CAN für den Antriebsstrang und der Low-Speed-CAN für den Karosseriestrang.

Welche Teile bei uns an welchen Strang gehörten, und welche Entwürfe wir gemacht haben, zeigen die nachfolgenden Grafiken.

Logische Struktur der Getriebesteuerung

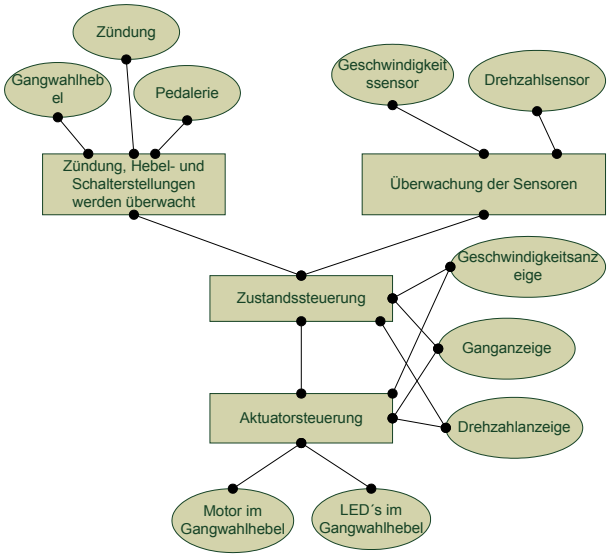


Abbildung 6.1: logische Struktur Getriebesteuerung

Technische Struktur der Getriebesteuerung

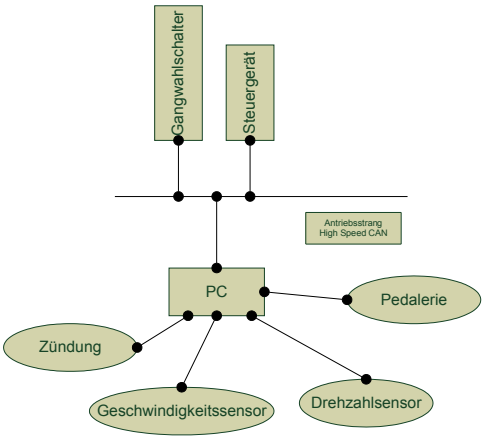


Abbildung 6.2: technische Struktur Getriebesteuerung

Folgende Funktionen wären auf der obigen Struktur (Abbildung 6.2) denkbar:

1. Beim Umschalten von Automatik auf manuell reagiert das Getriebe
2. Bei Automatik muss der Motor ab bestimmter Drehzahl schalten
3. Beim manuellen Schalten, schaltet das Getriebe den Gang
4. Bei Automatik sollte „Kick-Down“ unterstützt werden
5. Auch bei manuellem Betrieb sollten Drehzahlgrenzen eingehalten werden
6. Sport-Knopf erhöht Drehzahlgrenzen
7. Bei Vorwärtsfahrt kein Schalten in den Rückwärtsgang möglich
8. P-Modus kann nur verlassen werden, wenn Bremse getreten und Zündung an

Logische Struktur der Lichtsteuerung

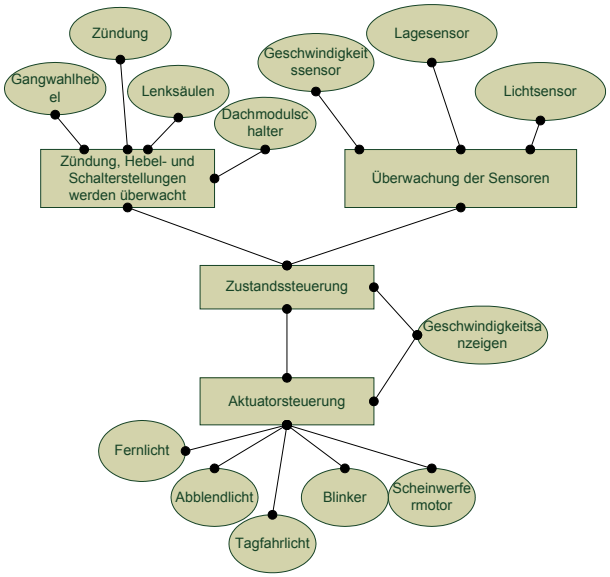


Abbildung 6.3: logische Struktur Lichtsteuerung

Lichtsteuerung mittels Lenksäule

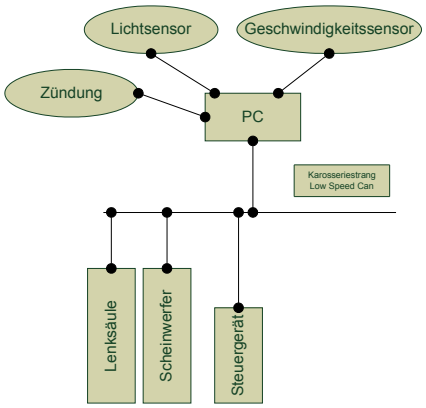


Abbildung 6.4: technische Struktur Lichtsteuerung mittels Lenksäule

Folgende Funktionen wären auf der obigen Struktur (Abbildung 6.4) denkbar:

1. Linker Hebel der Lenksäule steuert Fernlicht/Lichthupe
2. Bei Schalten der Zündung geht das Tagfahrlicht an
3. Wenn bestimmtes Lichtlevel unterschritten, wird Abblendlicht angeschaltet

Lichtsteuerung mittels Dachelement

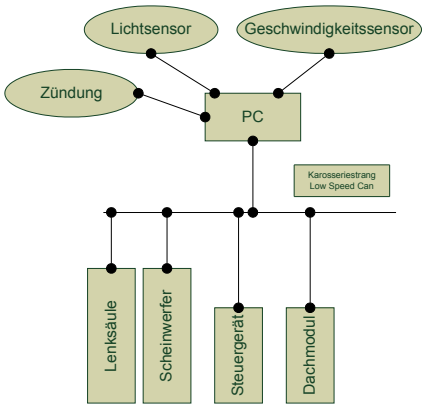


Abbildung 6.5: technische Struktur Lichtsteuerung mittels Dachelement

Folgende Funktionen wären auf der obigen Struktur (Abbildung 6.5) denkbar:

1. Durch Betätigen der Blinker wird der Blinker angeschaltet und das Tagfahrlicht gedimmt
2. Wenn der Schalter im Dachelement gedrückt ist, geht das Warnblinklicht an
3. Wenn das Warnblinklicht an ist, können die Blinker nicht mehr gesetzt werden

Lichtsteuerung mittels Lagesensor

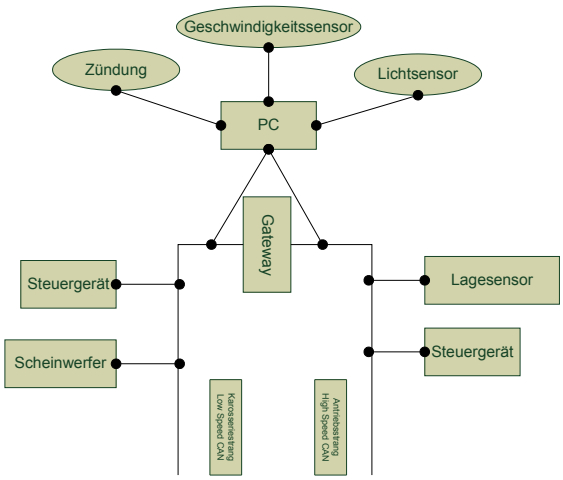


Abbildung 6.6: technische Struktur Lichtsteuerung mittels Lagesensor

Folgende Funktionen wären auf der obigen Struktur (Abbildung 6.6) denkbar:

1. Lagesensor steuert Höheneinstellung des Scheinwerfers
2. Lagesensor und Geschwindigkeitssensor steuern das Kurvenlicht
3. Wenn der Lagesensor eine Kurve erkennt, wird das Fernlicht abgeschaltet

Lichtsteuerung mittels Gangwahlschalter

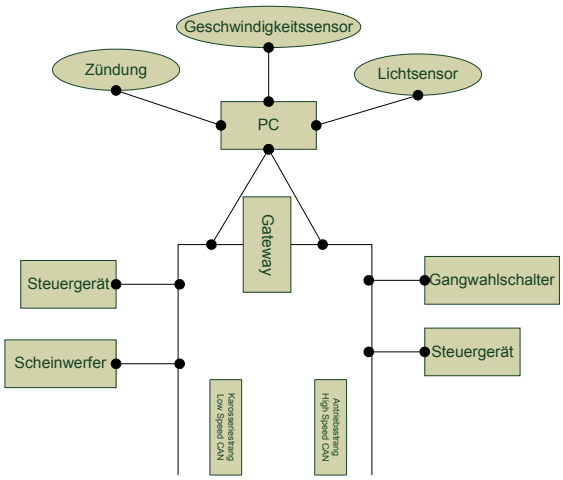


Abbildung 6.7: technische Struktur Lichtsteuerung mittels Gangwahlschalter

Folgende Funktionen wären auf der obigen Struktur (Abbildung 6.7) denkbar:

1. Wenn der Ganghebel auf D/R/manuell steht, wird das Tagfahrlicht angeschaltet
2. Wenn der Ganghebel auf D/R/manuell steht und ein bestimmtes Lichtlevel unterschritten ist, wird das Abblendlicht angeschaltet

kompletter Aufbau Lichtsteuerung

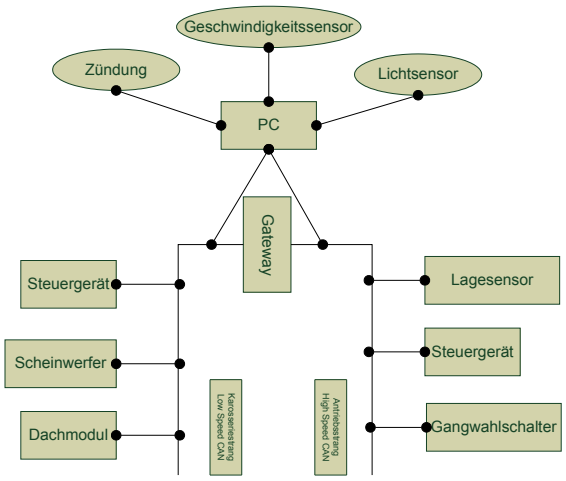


Abbildung 6.8: technische Struktur kompletter Aufbau Lichtsteuerung

Die obige Grafik stellt den kompletten möglichen Aufbau der Lichtsteuerung dar, somit sind alle oben genannten Funktionen möglich.

Kapitel 7

Erster Prototyp

7.1 Überblick

Ausgehend von unseren Ergebnissen der Projektplanung wurde beschlossen, dass das Prototyping in mehrere Phasen unterteilt werden sollte.

Als Resultat der ersten Phase sollte ein erster Prototyp mit zunächst vereinfachter Funktionalität erstellt werden. Die Abbildung 7.1 zeigt den konzeptionellen Aufbau unseres Prototyps, der im folgenden kurz erläutert wird.

Der Prototyp kann in zwei Hauptbestandteile kategorisiert werden:

1. das Fahrzeugnetz
2. die Steuer- und Messumgebung

Die bisherigen Komponenten der Fahrzeugumgebung sind der Scheinwerfer (5.1.3) und der Sensorcluster (5.1.4), welche mittels eines CAN-Bus Systems (*CAN 1*) miteinander verbunden sind. Zweck dieses Aufbaus ist es, den Scheinwerfer anhand der vom Sensorcluster ermittelten Daten zu steuern. Somit kann zum Beispiel das Kurvenlicht des Autos realisiert werden.

Die Realisierung geschieht wahlweise mittels PC-Unterstützung (Implementierung unter Zuhilfenahme des CANoe-Systems) oder direkt durch das TriCore-Board TC 1766 (5.1.2). Für diesen Zweck ist die Implementierung der Funktionalität in Software nötig (Steuersoftware und CAN-Treiber, um die Kommunikation auf dem CAN-Bus zu ermöglichen).

Ein weiteres CAN-Bus-System (*CAN 2*) sorgt für die Anbindung des zweiten TriCore-Boards TC 1796 (5.1.2) an den PC. Dieses Evaluationsboard dient unter anderem als Messschnittstelle für die von uns durchgeführten Messungen (zum Beispiel die Messung des Stromverbrauchs des TriCore-Boards TC 1766): Die Daten werden mittels Softwareunterstützung erhoben, mit

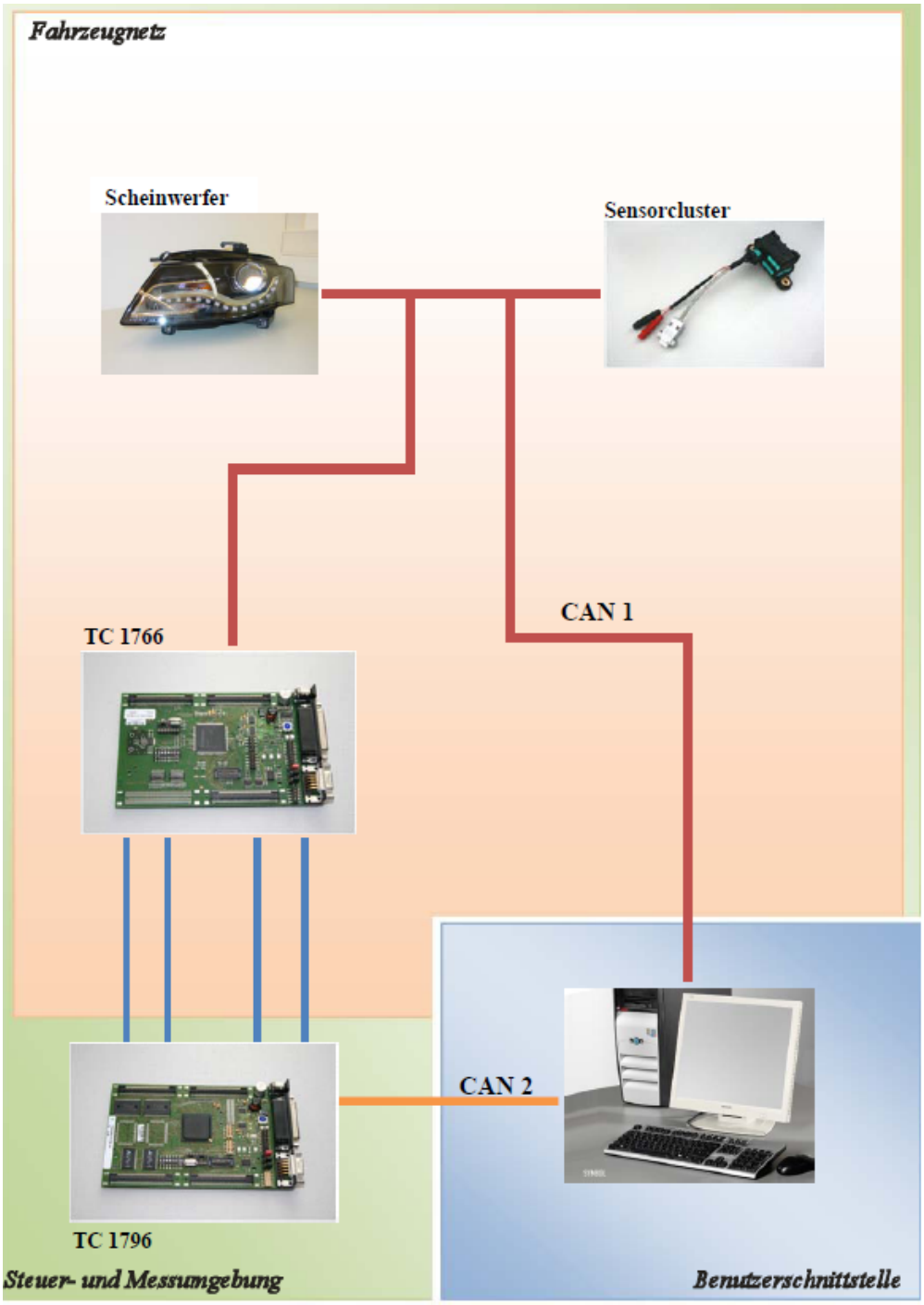


Abbildung 7.1: Konzeption des ersten Prototyps

Hilfe eines Analog/Digital-Wandlers (A/D-Wandler) in digitale Werte umgewandelt und zur Auswertung und weiteren Verarbeitung an den PC übermittelt. Des weiteren dient dieses Board dazu, die Steuerung des Fahrzeugnetzboards (TC 1766) zu übernehmen.

Weiterhin haben wir für unseren Prototyp Message-IDs für die CAN-Botschaften definiert, damit dies in der Software einheitlich umgesetzt wird. Die CAN-Botschaften sind in der folgenden Tabelle beschrieben. Dabei steht ein x in der Spalte-ID für durch die Hardware vorgegebene IDs.

ID	Teil	Funktion	Daten
x	Scheinwerfer	Soll	Siehe Dokumentation
x	Scheinwerfer	Soll	Siehe Dokumentation
x	Scheinwerfer	Status	Siehe Dokumentation
x	Scheinwerfer	Status	Siehe Dokumentation
x	Scheinwerfer	Status	Siehe Dokumentation
x	Scheinwerfer	Status	Siehe Dokumentation
x	Lagesensor	Status	Siehe Dokumentation
x	Lagesensor	Status	Siehe Dokumentation
20*	Gangwahlschalter		
21*	Dachelement		
210	Dachelement	Soll	3 Bit Licht an/aus für drei Lampen
210	Dachelement	Status	[-2,3] Schiebeschalter Stellungen, 3 Bit für Taster, 3 Bit für Licht
40*	TC1796		
400	TC1796	Soll	1 Bit Strom an/aus, 1 Bit Reset
401	TC1796	Status	Stromverbrauch 10 Bit Genauigkeit
41*	TC1766		
410	TC1766	Soll	Digitale I/O Signale setzen
411	TC1766	Status	Digitale I/O Signale abrufen
3*	Fehlermessages		
x	Scheinwerfer		
x	Scheinwerfer		
38*	Gangwahlschalter		
37*	Dachelement		
35*	TC1796		
34*	TC1766		

Tabelle 7.1: CAN-Botschaften

7.2 Einzelne Arbeiten

7.2.1 Verkabelung

Die Verkabelung, vor allem die Stromversorgung, muss vorher gut geplant werden, da hier Fehler im Betrieb fatale Folgen haben können. Das zur Verfügung stehende Labornetzgerät schaltet sich erst bei Strömen oberhalb von 50 A ab. Die Verkabelung wurde daher so ausgelegt, dass dieses Maximum an Strom fließen kann, ohne Kabelbrände oder ähnliches zu verursachen.

Die meisten Kabel sind mit $1,5mm^2$ dicken Kabeln konfektioniert worden. Beim Scheinwerferanschluss musste z.B. die Fahrzeugmasse sogar mit $2,5mm^2$ angeschlossen werden. Die Kabel halten dadurch pro Kabel Ströme von bis zu 16 A aus, pro Bananenstecker kann ein Strom von bis zu 30 A fließen.

7.2.2 Aktoranbindung

Damit man die Scheinwerfer-Aktoren wie das Fernlicht oder das Tagfahrlicht schalten kann, mussten wir eine Lösung finden, die Stromversorgung für den Aktor unterbrechen zu können. Zu diesem Zweck haben wir eine Transistorschaltung entworfen, die logikkompatibel ist. Dies hat den Vorteil, dass wir die Steuerung der Transistoren mithilfe eines Steuerboards wie dem TC1766 steuern können.

Die Schaltung besteht hauptsächlich aus 2 Transistoren, einem (Power-) Feldeffekttransistor und einem „normalen kleinen“ Transistor, der dem Powertransistor vorgelagert ist. Der kleine Transistor hat lediglich die Aufgabe, den Powertransistor logikkompatibel zu steuern. In ersten Entwürfen mit nur einem Transistor hat dies nicht ganz geklappt, da der Transistor erst kurz oberhalb von 3,5 V anfing aufzuregeln und Strom durchfließen zu lassen.

Aus diesem Grund haben wir einen passenden Vortransistor gewählt, um auch bei 0 - 0,8 V am Steuerungspin des Steuergeräts 0 V am Ausgang des Powertransistors (und bei größer 2,8 V die volle Spannung von 13,8 V) anzulegen. Der entsprechende Verbraucher (z.B. das Tagfahrlicht) kann dadurch seinen Strom vom Netzteil ziehen. Die Schaltung wurde in einem Kästchen mit Bananensteckerbuchsen eingebaut. Damit ist dieser Teil komplett in die bisherige Verkabelung integrierbar.

Für die Verbindung zum Triboard (siehe Kapitel 5.1.2) haben wir ebenfalls Stecker konfektionieren müssen. Diese mussten auf der einen Seite einen Bananenstecker und auf der anderen Seite eine Buchse für einzelne Pinne des TC 1766 haben. Softwareseitig musste das auf dem Steuerboard laufende Programm so angepasst werden, dass es nach dem Empfangen einer bestimmten CAN-Nachricht einen Pin auf logisch High oder logisch Low legt. Nach den Anpassungen und der passenden Verkabelung, können nun das Blinklicht, das Tagfahrlicht und das Fernlicht entsprechend per CAN-Botschaft ein und ausgeschaltet werden.

Der Powertransistor ist für entsprechend hohe Ströme ausgelegt, wurde jedoch nur bei max. 2A betrieben. Höhere Ströme verursachen natürlich auch eine höhere Abwärme, so dass hier zu

überlegen wäre, ob und wie weit der Transistor zu kühlen ist.

7.2.3 Sensorik

Die Realisierung einer Messumgebung war eines der primären Ziele des Projektes. Der Begriff „Sensorik“ umfasst hier vor allem das Erfassen von analogen als auch digitalen Signalen. Durch den Einsatz der bereits in Abschnitt 5.1.2 vorgestellten TriBoards, standen uns bis zu 44 A/D-Wandler und bis zu 10 Ports mit jeweils zwischen 4 und 16 Pins zur Verfügung, die als Ein- oder Ausgabepin konfiguriert werden können.

Durch die Messumgebung sollte zum einen die Stromaufnahme eines angeschlossenen Steuergerätes gemessen werden, und zum anderen sollten Pegeländerungen an digitalen Eingangspins überwacht werden. Eingangspins des TriBoards konnten so beispielsweise mit einem Steuergerät verbunden werden, um Statusänderungen überwachen zu können.

Ein konkretes Beispiel ist die Überwachung einer Ausgangsleitung zur Ansteuerung eines Aktors. Um die Reaktion eines Steuergerätes auf eine CAN-Nachricht überwachen zu können, wird in diesem Fall überprüft, ob der angeschlossene Aktor tatsächlich mit Regelungssignalen (Stromversorgung eines Elektromotors, Schalten einer Beleuchtungseinrichtung, usw.) versorgt wird.

Strommessung

Um einen Strom messen zu können, kann der Spannungsabfall an einem Widerstand gemessen werden, der in der Versorgungsleitung des entsprechenden Gerätes sitzt. Der Spannungsabfall an diesem Widerstand berechnet sich nach dem Ohm'schen Gesetz zu:

$$U = I * R$$

Der eingesetzte Widerstand ist üblicherweise sehr gering ($< 1\ \Omega$) und wird auch als „Nebenwiderstand“ oder „Shunt“ bezeichnet.

Für den ersten Prototyp wurde zunächst eine maximale Stromaufnahme von 500 mA angenommen. Der verwendete Shunt sollte 0,1 Ω besitzen. Nach dem Ohm'schen Gesetz war demnach mit einem Spannungsabfall von

$$U = 0,5\text{A} * 0,1\Omega = 0{,}05\ \text{Volt}$$

zu rechnen.

Die A/D-Wandler eines TriCore-Mikrocontrollers arbeiten im Bereich von 0 ... 3,3 Volt und mit einer Auflösung von 10 Bit (im Fall des FADC - Fast Analog/Digital Converter). Um die volle Auflösung des A/D-Wandlers nutzen zu können, musste eine Lösung gefunden werden, um die am Shunt abfallende Spannung von 0 ... 0,05 Volt auf den Spannungsbereich 0 ... 3,3 Volt verstärken zu können.

Die benötigte Verstärkung (engl. Gain) berechnet sich hierbei einfach nach:

$$\text{Gain} = \frac{3,3\text{Volt}}{0,05\text{Volt}} = 66$$

Da der Shunt in der Versorgungsleitung eines Gerätes sitzt und daher mit keinem Anschluss auf Massepotential liegt, lag der Einsatz eines Instrumentenverstärkers nahe.

In einem Instrumentenverstärker werden extrem hochohmige OPVs (Operationsverstärker, auch OpAmp genannt) eingesetzt. Solche Verstärker besitzen Differenzeingänge und einen unipolaren Ausgang.

Nach einiger Recherche fiel die Wahl auf den INA114. Dieser Instrumentenverstärker besitzt 8 Anschlüsse:

- 2x Versorgungsspannung (V^+ & V^-)
- 2x Messeingang (V_{IN}^+ & V_{IN}^-)
- 1x Ausgangsspannung
- 1x Spannungsreferenz (hier: Massepotential)
- 2x Verstärkungseinstellung (R_G)

Die Verstärkungseinstellung erfolgt durch einen externen Widerstand. Die Formel zur Berechnung des erforderlichen Widerstandes lautet:

$$G = 1 + \frac{50\ k\Omega}{R_G}$$

Nach Umstellen der Formel und Einsetzen der Werte (angestrebte Verstärkung war 66), bekommt man den erforderlichen Widerstand für unsere Schaltung:

$$R_G = \frac{50\ k\Omega}{65} = 769{,}23\Omega$$

Der nächste lieferbare Widerstandswert aus der Reihe E12 / E24 ist 820 Ω. Setzt man diesen Wert ein, erhält man die für unsere Schaltung erreichte Verstärkung von:

$$G = 1 + \frac{50\ k\Omega}{820\Omega} = 61{,}975$$

Der nächste Schritt war der Entwurf der eigentlichen Schaltung. In Abbildung 7.2 sind die oben beschriebenen Elemente zu erkennen. Da die Platine möglicherweise später an andere zu messende Stromstärken angepasst werden können sollte, wurde R_{Gain} (der Widerstand zur Verstärkungseinstellung) über K6 (2x6 pol. Stiftleiste) angeschlossen. Auf diese Weise kann die aktuell benötigte Verstärkung per Jumper ausgewählt werden. Zum Schutz des A/D-Wandlers auf dem TriBoard vor Über-/Unterspannung wurde der Ausgang des Instrumentenverstärkers über Shottky-Dioden mit der Referenzmasse $V_{SSFAREF}$ und der Referenzspannung $V_{DDFAREF}$ des A/D-Wandlers verbunden (FA bezieht sich hier auf den FADC des TriCores). Der Widerstand R3 dient dem Schutz der Dioden vor Überstrom.

Die fertig aufgebaute Platine ist in Abbildung 7.3 zu sehen.

Zusätzlich wurde ein Relais und die notwendigen Schalttransistoren auf dieser Platine verbaut, um die zu messende Schaltung komplett von der Stromversorgung trennen zu können.

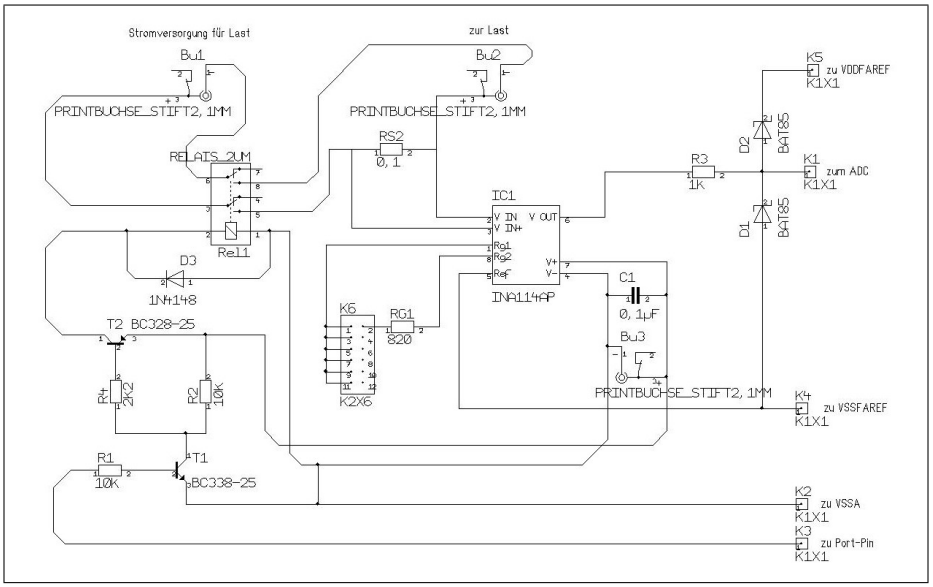


Abbildung 7.2: Stromlaufplan der Strommessplatine

Die einzelnen Anschlüsse sind im folgenden aufgeführt:

- K2: Masse des TriBoard
- K3: Port-Pin zur Ansteuerung des Relais
- K1: Anschluss an Analogeingang des TriCore FADC
- K4: Anschluss an Referenzmasse des TriCore FADC
- K5: Anschluss an Referenzspannung des TriCore FADC

Digitale Eingangspegel messen

Jeden Port-Pin eines TriCores kann man als Ein- oder Ausgang konfigurieren. Die Messung des an einem Eingangspin anliegenden Signalpegels erfolgt durch Abfragen des entsprechenden Portregisters.

Messergebnisse auswerten

Alle Messergebnisse, d.h. Ergebnisse von A/D-Wandlungen und Port-Pin-Abfragen, wurden regelmäßig in CAN-Nachrichten an den PC gesendet. Auf dem PC lief CANoe als Anwendung, das die empfangenen Daten graphisch darstellte.

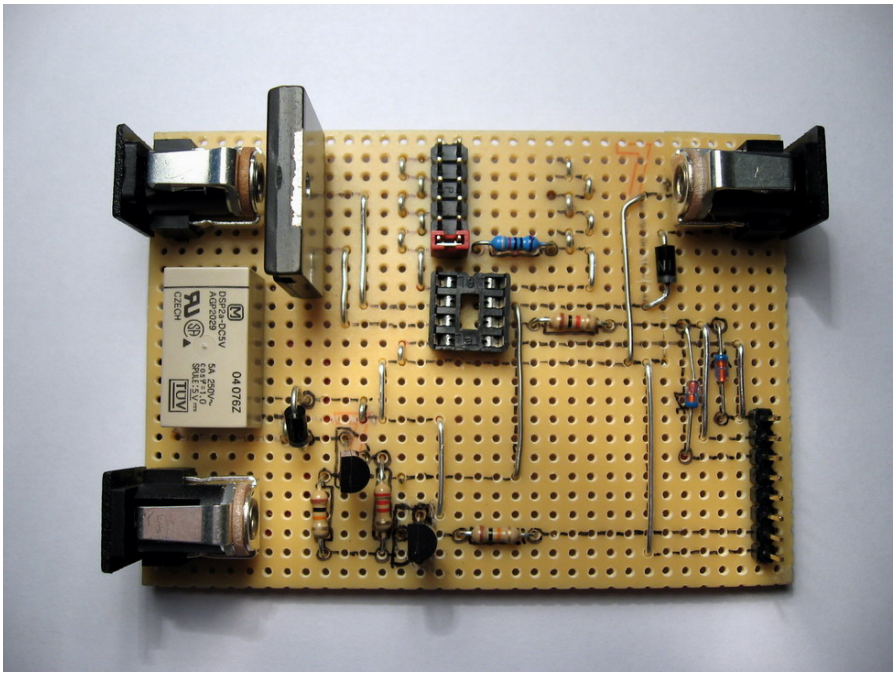


Abbildung 7.3: Fertig aufgebaute Strommessplatine

7.2.4 Software

Realisierung des Kurvenlichts

Das TriCore-Board TC1766 übernahm im Rahmen unseres ersten Prototypen die Aufgabe, Status-Botschaften des Sensorclusters entgegenzunehmen, auszuwerten und Soll-Botschaften an den Scheinwerfer zu senden. Die Soll-Botschaften an den Scheinwerfer mussten kontinuierlich gesendet werden, da sich der Scheinwerfer bei Ausbleiben von Soll-Botschaften in die Ausgangsstellung zurückstellt. Um eine potentielle Kurvenfahrt zu erkennen, nutzten wir ein Signal des Sensorclusters, welches die Drehrate um die Z-Achse wiedergibt. Bei Überschreitung eines bestimmten Schwellwerts sandte das TriBoard in den kontinuierlichen Soll-Botschaften ein Signal, welches dem Scheinwerfer dazu veranlasste, sich in die Richtung auszurichten, in die der Sensorcluster gedreht wurde. Dabei wurde der Scheinwerfer um einen festgelegten Wert (Winkel) verstellt, der noch unabhängig von dem genauen Wert der Drehrate war. Bei nur kurzfristiger Überschreitung des Schwellwerts hätte der Scheinwerfer nur ruckartig in eine Richtung gezuckt. Um dies zu verhindern, haben wir bei Überschreitung des Schwellwerts diesen Wert für eine gewisse Zeitspanne $t < 0,5s$ weiterverwendet. So erreichten wir sanfte Bewegungen des Scheinwerfers.

Messsoftware auf TriBoard TC1796

Da das TriBoard TC1796 mit mehr analogen Eingängen und digitalen Ein-/Ausgängen ausgestattet ist als das TriBoard TC1766, wurde es für die aufzubauende Messumgebung verwendet.

Bereits in Abschnitt „7.2.3 Sensorik“ wurde der Aufbau einer Messplatine beschrieben. Diese Messplatine lieferte eine Spannung, die proportional zur Stromaufnahme eines angeschlossenen Steuergerätes war.

Die Software auf dem TriBoard TC1796 hat folgende Aufgaben:

- Messung auf externen Trigger starten (CAN-Nachricht)
- Stromversorgung des angeschlossenen Steuergerätes herstellen (Relais ansteuern)
- analoge Spannung messen und in digitalen Wert (10 Bit) umwandeln
- digitale Pegel messen
- Messergebnisse in CAN-Nachrichten an CANoe senden

Zur Realisierung der Messsoftware sind folgende Module des TriBoards TC1796 notwendig bzw. zu konfigurieren:

- FADC - Fast Analog Digital Converter (ein besonders schneller Analog/Digital-Wandler) zur Messung von analogen Signalen (bspw. Spannungsabfall am Shunt)
- CAN-Modul (dazu wird der CAN-Treiber (siehe Kapitel 7.2.5) benutzt)
- I/O-Port-Pins als Ausgang (zur Steuerung des Relais)
- ~~I/O-Port-Pins als Eingang (zur Messung von digitalen Pegeln)~~

Der letzte Punkt wurde auf die nächste Projektiteration verschoben. Als Alternative wurden die A/D-Wandler zur Messung digitaler Pegel benutzt.

Für das Empfangen und Versenden von CAN-Nachrichten konnte der vorhandene CAN-Treiber in unsere Software integriert werden.

Die Konfiguration des FADC und der I/O-Pins wurde in Header-Files ausgelagert, die von uns geschrieben wurden.

Der FADC des TriCore TC1796 wurde so konfiguriert, dass eine A/D-Wandlung regelmäßig von einem Timer angestoßen wurde. Das Ergebnis einer A/D-Wandlung wurde periodisch abgefragt und in einer CAN-Nachricht versendet.

Ein I/O-Pin des TriBoard wurde als Ausgang konfiguriert, um das Relais der Strommessplatine ansteuern zu können.

Die PC-Software CANoe war ein weiterer wichtiger Bestandteil der gesamten Messumgebung. Die Software auf dem TriBoard TC1796 und CANoe arbeiteten eng zusammen, indem Steuer- und Statusnachrichten ausgetauscht wurden.

Der Programmablauf sah folgendermaßen aus:

- Zum Starten der Messung:
  - CANoe sendet Steuernachricht zum Starten einer Messung
  - TriBoard setzt Pegel des Port-Pin für das Relais auf „high“ → Relais schließt; zu messendes Steuergerät wird mit Strom versorgt
  - TriBoard fragt regelmäßig das Ergebnis einer A/D-Wandlung ab
  - Ergebnis wir in Statusnachricht versendet
  - CANoe gibt Statusnachricht aus
- Zum Stoppen der Messung:
  - CANoe sendet Steuernachricht zum Stoppen der Messung
  - TriBoard setzt Pegel des Port-Pin für das Relais auf „low“ → Relais öffnet; zu messendes Steuergerät wird von der Stromversorgung getrennt

7.2.5 CAN-Bus-Treiber

Die TriBoards (siehe Kapitel 5.1.2) besitzen CAN-Schnittstellen, die wir zum Versenden und Empfangen von Steuerbefehlen und Messwerten verwenden wollten. Für das TriBoard TC1796 wurde bereits ein entsprechender Treiber im Rahmen einer Studienarbeit erstellt, den wir freundlicherweise benutzen durften. Der Implementierungsaufwand in Bezug auf CAN-Nachrichten durch uns wurde dadurch auf ein Minimum reduziert.

Die Verwendung des Treibers ist sehr einfach und mit wenigen Schritten in ein Projekt einzubinden. Unbedingt zu beachten sind allerdings die Einstellungen der Timings (vgl. Abschnitt 7.3.1, CAN-Bus-Timing).

7.3 Aufgetretene Probleme und deren Lösungen

7.3.1 CAN-Bus-Timing

Beim Einsatz des uns zur Verfügung gestellten CAN-Treibers für das TriBoard TC1796 kam es aufgrund eines nicht exakt eingestellten Taktes zu Problemen.

Dieses Problem äußerte sich durch anhaltende Bit-Fehler, die von CANoe gemeldet wurden. Die Überprüfung des Quellcodes brachte leider keine Abhilfe, da alle Einstellungen logisch und korrekt erschienen.

Mit Hilfe eines Oszilloskopes konnten wir den Fehler allerdings sehr schnell eingrenzen und die Timingeinstellungen als Ursache ausmachen. Ein erneutes Betrachten der entsprechenden Stellen im Quellcode ermöglichte uns die notwendigen Änderungen vorzunehmen.

Zu beachten sind der Systemtakt, der interne VCO (Voltage Controlled Oscillator) und der eingesetzte Quarz. Die betreffenden Einstellungen müssen exakt aufeinander abgestimmt sein.

Sind diese Einstellungen korrekt, wird das CAN-Modul stabil mit vollem oder halbem Systemtakt versorgt (einstellbar). Zur Einstellung der Baudrate der CAN-Schnittstelle wird ein Baudratenvorteiler (BRP - Baud Rate Prescaler) benutzt. Dieser muss entsprechend der gewünschten Baudrate gewählt werden.

7.3.2 Analyse der Hardwarekomponenten

Allgemein

Bei der Untersuchung der Kostal-Hardware erwies es sich als großes Problem, dass wir keinerlei Dokumentation zu den vorliegenden Komponenten hatten. Daher mussten wir mit großem Aufwand die Hardware analysieren, um Pin- und Steckerbelegungen auf den Boards sowie die Anschlussmöglichkeiten für das Programmiergerät herauszufinden.

Da die Platinen der Geräte teilweise mehrlagig sind, konnten wir die Leiterbahnen nicht mit bloßem Auge verfolgen, sondern mussten eventuelle Verbindungen mit einem Multimeter ermitteln. Dabei musste geraten werden, wo eine ins Innere der Platine verschwindende Leiterbahn wieder an der Oberfläche heraustritt. Dies erforderte ein hohes Maß an Geduld und manchmal auch etwas Glück.

Da es sich bei der vorliegenden Hardware nicht um Entwicklungsplatinen handelt, sondern um Serienprodukte, mussten wir die Anschlüsse für das Programmiergerät selbst anlöten. Als erstes Hindernis erwies sich dabei das Identifizieren der Kontakte zum Flashen auf dem Board. Nachdem diese gefunden waren, ergab sich die nächste Schwierigkeit: die sehr kleine und dichte Bauform der Platinen. Diese setzte für das Löten großes Geschick voraus.

Auch beim Messen von Signalen mit einem Oszilloskop erwies sich diese Bauform als Hürde, da man mit den Messkontakten die Signale nicht von den Mikrocontrollerpins und auch nur schlecht von den Durchkontaktierungen abgreifen konnte.

Schaltermodul Lenksäule

Die Analyse des Schaltermoduls Lenksäule (SMLS) erwies sich als sehr problematisch.

Die für den Anschluss der Programmierschnittstelle MON08 erforderlichen Verbindungspunkte ließen sich nur bedingt identifizieren, da die Leiterbahnen beziehungsweise Signalwege nicht mit bloßem Auge oder per Widerstandsmessung nachvollzogen werden konnten. Somit konnte im bisherigen Verlauf der Projektgruppe keine zufriedenstellende Lösung für die Flash-Programmierung

gefunden werden.

Versuche, die Komponente direkt über ein selbst gebautes Kabel an einen der von dem Modul bereitgestellten zwei CAN-Bussysteme (Antriebs- oder Komfort-CAN) anzuschließen und mittels CANoe anzusprechen respektive das Verhalten zu simulieren, scheiterten daran, dass weder der Antriebs- noch der Komfort-CAN der Lenksäule dauerhaft reproduzierbare Signale senden bzw. empfangen konnte.

All diese Hindernisse haben uns dazu bewegt, die Komponente Schaltermodul Lenksäule nicht in den ersten Prototyp aufzunehmen.

7.3.3 System Basis Chip

Jedes der drei Hardwaremodule von Kostal lieferte Fehler beim Versuch den Flashspeicher über die BDM-Schnittstelle auszulesen oder zu beschreiben. Der Grund dafür lag darin, dass sich die jeweiligen Mikrocontroller in einem Schlafmodus befanden. Verantwortlich dafür waren die jeweiligen System Basis Chips, die den Energiezustand der jeweiligen Mikrocontroller steuerten.

Der System Basis Chip UJA1065 von Philips ist im Gangwahlschalter verbaut und verantwortlich für das Aufwecken des Freescale MC9S12DG128 Mikrocontrollers. Zunächst konnte der Freescale Controller durch Anschließen von 12V an den PT-Wecken-Pin des 6-poligen Steckers wach gehalten werden. Dies ermöglichte das Auslesen des Programmspeichers unter Verwendung der Entwicklungsumgebung CodeWarrior von Metrowerks. Beim Versuch den Programmspeicher zu flashen trat eine Störung in der BDM-Kommunikation auf. Daraus resultierte ein Mikrocontroller mit beschädigtem Programmspeicher. Die Folge war, dass sich das Modul selbst durch Anlegen der 12V am PT-Wecken-Pin nicht aufwecken ließ. Ein ausgiebiges Studium der UJA1065-Handbücher ergab, dass durch Anlegen von 6V an dem TEST-Pin des UJA1065 der Mikrocontroller aufgeweckt werden konnte. Diese vorübergehende Lösung ermöglichte es, den Gangwahlschalter mit eigener Software zu flashen und zu debuggen.

Der System Basis Chip L4969 (SO20) von STMicroelectronics ist im Dachelement verbaut und zuständig für den Energiezustand des Freescale MC9S12DB128 Mikrocontrollers. Durch Drücken der mittleren Taste, die ebenfalls das Ambientelicht steuert, ließ sich das Dachelement am Leben halten. Anders als erwartet war es mit der Software des inDart-One nur möglich, den Datenspeicher oder einen sehr kleinen Bereich des Programmspeichers auszulesen. Jeder Versuch, einen größeren Programmspeicherbereich auszulesen, endete mit einer Fehlermeldung. Unter Verwendung des CodeWarriors tauchten diese Schwierigkeiten zunächst nicht auf. Wie beim Gangwahlschalter wurde auch die Software auf dem Mikrocontroller des Dachelements beim Flashversuch beschädigt. Im Folgenden ließ sich auch der Freescale MC9S12DB128 nicht mehr aufwecken. Laut Handbuch des L4969 mussten die Pins NRES und V1 verbunden werden, damit der Freescale Mikrocontroller nach dem Einschalten nicht in den Schlafzustand überführt werden konnte. Diese vorübergehende Lösung ermöglichte es, das Dachelement mit eigener Software zu flashen und zu debuggen.

Beide System Basis Chips, UJA1065 und L4969 (SO20), verfügen über einen CAN-Bus, der zum Aufwecken der Mikrocontroller benutzt werden kann. Leider war dies in keinem Laborversuch gelungen.

Eine mögliche Lösung wird es sein, hardwareseitig die benötigten Verbindungen an den Pins der jeweiligen System Basis Chips fest zu verdrahten. Beim Dachelement könnten die Pins NRES und V1 mit Hilfe eines Schalters verbunden werden. Diese Lösung birgt jedoch das Risiko, die Hardwaremodule von Kostal zu beschädigen. So ist bereits der TEST-Pin am UJA1065 System Basis Chip abgebrochen.

Aus diesem Grund wird eine softwareseitige Lösung angestrebt. Beide System Basis Chips verfügen über eine serielle Schnittstelle (SPI - Serial Peripheral Interface). Beim Start der Hardwaremodule wecken die beiden System Basis Chips ihren jeweiligen Mikrocontroller auf und warten ein vorgegebenes Zeitintervall auf eine SPI-Nachricht. Durch die SPI-Kommandos ist es möglich, den jeweiligen System Basis Chip zu deaktivieren oder zumindest zum Erhalten des Wachzustandes zu bewegen. Erste Versuche in diese Richtung wurden bereits unternommen.

Beim Lenksäulenmodul ist es nicht gelungen, einen System Basis Chip ausfindig zu machen. Selbst die für den Anschluss der Programmierschnittstelle MON08 erforderlichen Verbindungspunkte ließen sich nur bedingt identifizieren.

7.4 Stand der Realisierung

Nachdem die Zielbestimmungen für den ersten Prototyp festgelegt wurden, und die damit verbundenen Forschungsarbeiten weitesgehend abgeschlossen waren, erfolgte die Implementierung des Prototyps.

Während dieser Umsetzung traten die im vorherigen Abschnitt erwähnten und zum Teil sehr zeitaufwändigen Hindernisse auf, die trotzdem zum größten Teil bewältigt werden konnten.

Der Prototyp besteht wie vorgesehen aus den Teilbereichen Fahrzeugnetz und Steuerungs- bzw. Messumgebung.

Um diese beiden Bereiche physisch miteinander verbinden zu können, wurden zwei CAN-Bussysteme entwickelt. Das erste (CAN1) verbindet die Fahrzeugkomponenten Scheinwerfer und Sensorcluster untereinander und bietet des weiteren eine Verbindung zum TriCore-Board TC 1766, welches als Steuerzentrale für dieses Teilnetz fungiert. Optional ist die Simulation dieses Netzes auch noch über den PC mittels CANoe möglich. Auf dem TriBoard TC 1766 befindet sich die von uns geschriebene Steuerungssoftware, mit welcher es möglich ist, die angestrebte Demonstration einer Kurvenfahrt mit automatischer Scheinwerferausrichtung zu realisieren.

Das zweite CAN-Bussystem (CAN2) sorgt für die Anbindung des zweiten TriCore-Boards (TC1796), das für die Steuerungs- und Messumgebung verantwortlich ist, an den PC.

Mit Hilfe dieses Boards und der selbst entworfenen Strommessplatine können Messungen durchgeführt werden. Diese Messungen können zum einen die Stromaufnahme eines angeschlossenen Steuergeräts zum anderen aber auch Pegeländerungen an digitalen Eingangspins, zum Beispiel bei Statusänderungen eines Steuergeräts, umfassen. Nach einer gegebenenfalls notwendigen Umwandlung analoger in digitale Werte können diese zur weiteren Auswertung, Verarbeitung und Visualisierung an den PC weitergeleitet werden.

Zusätzlich zu dieser Messfunktionalität dient das TriCore-Board TC 1796 zur Steuerung des angeschlossenen Fahrzeugnetzboards TC 1766.

Um die Kommunikation zwischen den TriCore-Boards TC 1766 und TC1796 und den angeschlossenen Steuergeräten zu gewährleisten, wurde ein CAN-Treiber, der bereits im Rahmen einer Studienarbeit entwickelt wurde, an die boardspezifischen Anforderungen angepasst.

Kapitel 8

Zweiter Prototyp

8.1 Überblick

Die zweite Phase des Prototypings umfasste die Erstellung eines weiteren Prototyps, bevor das Endprodukt fertig gestellt werden konnte.

Der zweite Prototyp kann zunächst ebenfalls in die zwei Hauptbestandteile Fahrzeugnetz und Steuer- bzw. Messumgebung kategorisiert werden. Da diese Ausbaustufe der Öffentlichkeit auf dem Campusfest präsentiert werden sollte, wurde der Demonstrator bzw. die Benutzerschnittstelle stark verbessert, indem den Besuchern die Möglichkeit gegeben wurde, die Funktionalität des Aufbaus und somit das Zusammenspiel der einzelnen Komponenten live erleben zu können. Hierfür wurde die Open-Source-Rennsimulation TORCS[1] an unsere Bedürfnisse angepasst und in den Demonstrator eingegliedert. Um diese Rennsimulation spielen zu können, wurde ein handelsübliches PC-Lenkrad (siehe Kapitel 8.2.4) als Grundlage beschafft und so modifiziert, dass Eingaben wie zum Beispiel Betätigen der Knöpfe und das eigentliche Lenken in CAN-Nachrichten umgewandelt werden und an die entsprechenden Kommunikationspartner weitergeleitet werden konnten.

Das Fahrzeugnetz konnte um einige neue Funktionalitäten erweitert werden. So ist es gelungen, den Gangwahlschalter (siehe Kapitel 5.1.6) in den Aufbau einzubauen und die hierfür benötigte Software zu implementieren. Somit wurde es ermöglicht, die Rennsimulation um die Features manuelle und automatische Gangschaltung zu erweitern. Die Gänge der Rennwagen konnten somit „per Hand“ oder automatisch geschaltet werden. Die Interaktivität der Rennsimulation wurde zusätzlich mittels der Scheinwerfer (siehe Kapitel 5.1.3) erhöht. Diese reagierten zum Beispiel auf eine Kurvenfahrt des Rennwagens, indem das Licht automatisch an die Straßenverhältnisse angepasst werden konnte. Des weiteren wurde das Blinklicht in den Rennablauf eingegliedert, wodurch die Simulation erneut realistischer gestaltet werden konnte.

Die Kommunikation der einzelnen Komponenten erfolgte nach wie vor über den schon im ersten

[1] http://torcs.sourceforge.net/

Prototyp eingesetzten CAN1-Bus. Die Kommunikation der Steuer- und Messumgebung mit der Benutzerschnittstelle erfolgte mit dem schon im ersten Prototyp bewährten CAN2-Bus.

Des weiteren konnte die Steuer- und Messumgebung weiter verfeinert, getestet und in ProOSEK eingebettet werden.

Der resultierende Aufbau ist in der folgenden grafischen Übersicht zusammengefasst (siehe Abbildung 8.1). Hierbei sind den einzelnen Komponenten die jeweiligen Steuer- und Statusnachrichten zugeordnet, wodurch eine grobe Übersicht über den zustandekommenden Nachrichtenaustausch vermittelt wird.

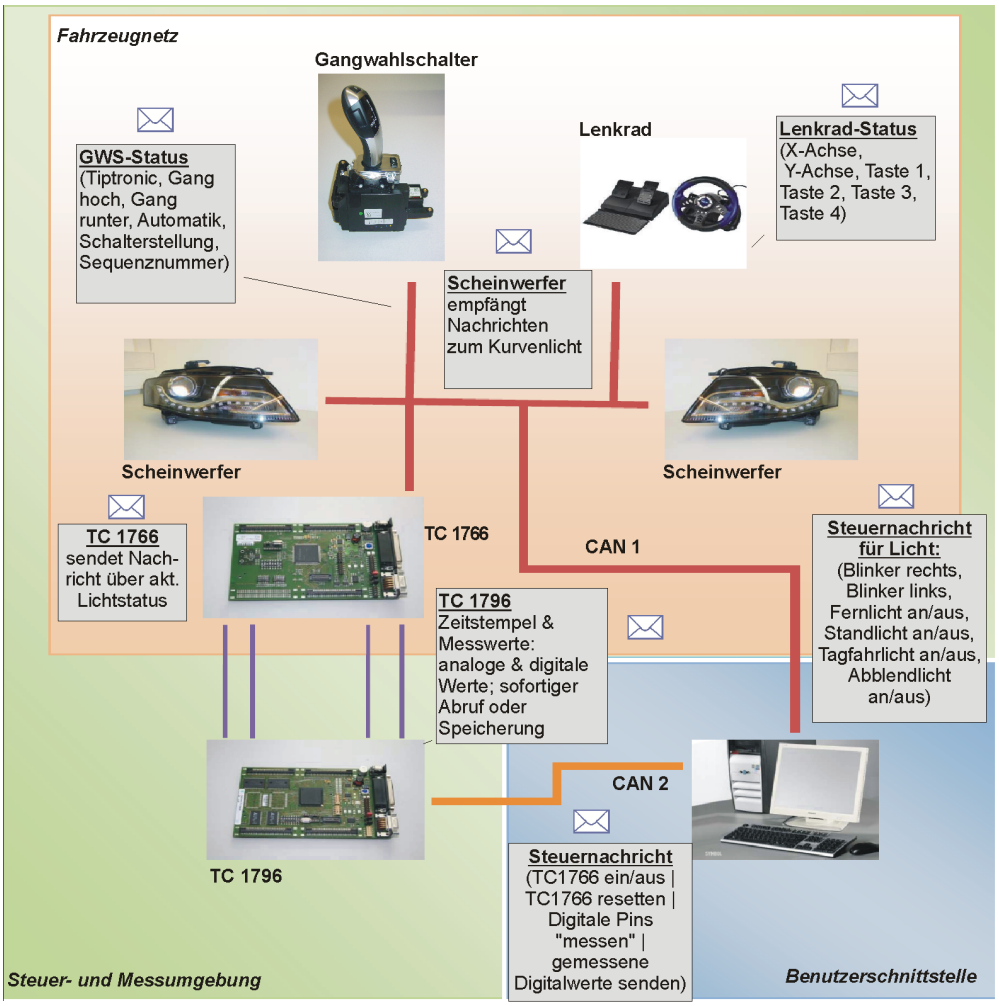


Abbildung 8.1: Konzeption des zweiten Prototyps

8.2 Einzelne Arbeiten

8.2.1 Verkabelung

Um die Verkabelung des gesamten Versuchaufbaus zu gewährleisten, mussten zum Teil große Leitungswege überbrückt werden. Hierbei fiel auf, dass vor allem die Scheinwerfer ein wahres Bündel von Leitungen benötigten. Dieses Bündel würde auf dem Campusfest eine wesentlich längere Strecke zurücklegen müssen. Da die Standard-Labormesskabel max. nur 1m lang sind, mussten diese stets ineinander gesteckt werden, wollte man eine ausreichende Kabellänge für das Campusfest erreichen.

Parallel zur Planung des Standes auf dem Campusfest entstand der erste Gedanke, die Steuerkabel alle in einem 10 m langen Kabel unterzubringen, um so die Kabelanzahl drastisch zu reduzieren. Dies schlug jedoch leider fehl. Obwohl extra ein 20 adriges Kabel gewählt wurde, welches so konfektioniert wurde, dass stets eine Masseleitung neben einer Steuerleitung aufgelegt war, funktionierte die Signalübertragung nicht. Mit zehn ineinandergesteckten Messkabeln (Querschnitt $1,5mm^2$) funktionierte die Signalübertragung jedoch tadellos. Der zweite Gedanke, elf Kabel á 10 Meter zu konfektionieren, scheiterte in der Umsetzung auch an der Tatsache, dass der Spannungsabfall über die Leitungslänge bei dem gewählten Querschnitt von $1,5mm^2$ zu groß war. Je nach Scheinwerfer resultierte dies entweder in einem ein- oder ausgeschalteten Zustand, vermutlich je nachdem, welche elektrotechnischen Eigenschaften die im Scheinwerfer verbaute Elektronik aufwies. Da in der Zwischenzeit das Campusfest wesentlich näher gerückt war, wurden die Kabelwege nochmal neu und genauer ausgerechnet und ein Kabelweg von nur noch ca. 5 Metern ermittelt. Dieser wurde dann mit Messkabel á 1 Meter, die ineinander gesteckt waren, hergestellt.

8.2.2 Sensorik

Strommessplatine

Die Strommessplatine, die schon für den ersten Prototypen geplant und aufgebaut wurde, hatte leider zunächst einen kleineren, nicht sofort ersichtlichen Fehler. Dieser Fehler lag zum einen an einer fehlerhaften Beschaltung und zum anderen an einer fehlenden Referenzmasse an den Eingängen des Instrumentenverstärker-ICs INA114. Im folgenden Abschnitt erfolgt eine kurze Beschreibung des Fehlers und der vorgenommenen Veränderungen an der Schaltung.

Fehlerbeschreibung

Der an einem Shunt gemessene Spannungsabfall wurde an die Eingänge des Instrumentenverstärkers gelegt. Über einen externen Widerstand konnte der Verstärkungsfaktor festgelegt werden. Am Ausgang konnte nun die verstärkte Spannung gemessen und von einem angeschlossenen TriBoard ausgewertet werden. Die Ausgangsspannung zeigte allerdings ein nicht nachzuvollziehendes Verhalten. Die Ausgangsspannung bewegte sich im Bereich von wenigen mV,

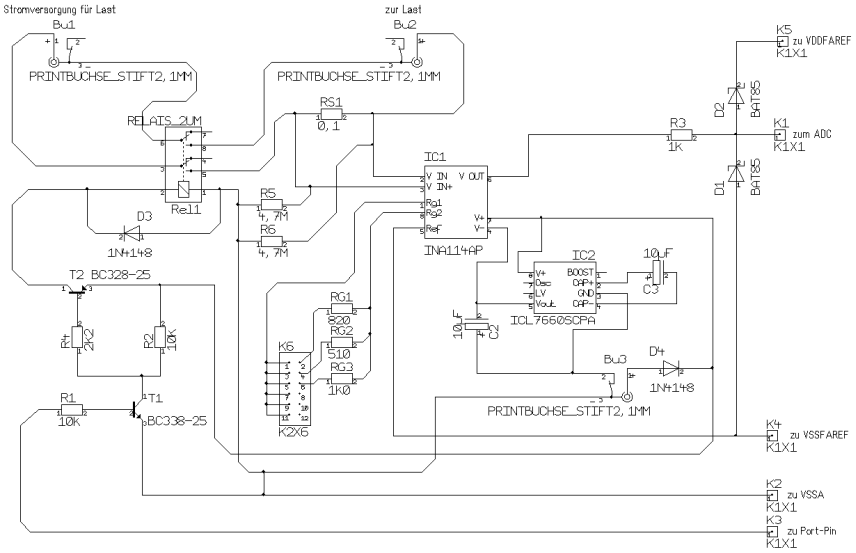


Abbildung 8.2: Stromlaufplan der Strommessplatine, 2. Prototyp

obwohl bereits die Eingangsspannung teilweise höher lag. Bei einem Verstärkungsfaktor von ca. 62 (dieser Wert ergab sich aufgrund von Vorüberlegungen, die bereits beim ersten Prototypen angestellt worden waren, vgl. hierzu den Abschnitt zur Strommessung 7.2.3) war demnach eine wesentlich höhere Ausgangsspannung zu erwarten.

Nach erneutem Konsultieren des Datenblattes ergab sich ein erster Hinweis auf eine fehlerhafte Beschaltung. Der interne Aufbau des Instrumentenverstärkers INA114 besteht im Grunde aus Operationsverstärkern (für eine exakte Beschreibung sei hier auf das Datenblatt verwiesen). Operationsverstärker benötigen, bis auf wenige Ausnahmen, eine symmetrische Spannungsversorgung, d.h. eine positive und eine negative Versorgungsspannung. Dies wurde beim ersten Prototypen nicht beachtet und nun korrigiert.

Die Erzeugung einer negativen Spannung wurde mit Hilfe einer sog. Ladungspumpe realisiert, dem IC ICL7660. Diese Änderung brachte leider keine Verbesserung des Verhaltens. Nun folgte die Suche in diversen Foren nach vergleichbaren Schaltungen und Problemen, wobei wir auf eine fehlende Referenzmasse aufmerksam wurden.

Die Strommessplatine wurde so aufgebaut, dass eine high-side current-Messung durchgeführt wurde, d.h. es wurde der Spannungsabfall an einem Shunt gemessen, der sich in der positiven Versorgungsspannungsleitung des entsprechenden Verbrauchers befand. Diese Art der Beschaltung der Eingänge des Instrumentenverstärkers führt dazu, dass keiner der beiden Eingänge eine Verbindung zu einem definierten Massepotential hat. Um dies zu beheben, wurden Widerstände im MΩ-Bereich eingesetzt, welche die Eingänge des Instrumentenverstärkers mit der Masse verbinden. Nach diesen Veränderungen konnte nun das gewünschte Verhalten der Strommessplatine erreicht werden. Im der Abbildung 8.2 befindet sich der angepasste Stromlaufplan mit allen hier beschriebenen Veränderungen.

8.2.3 Software

Fahrzeugfunktion - Scheinwerfer

Die Scheinwerfer, die uns zur Verfügung gestellt wurden, haben eigene Steuergeräte verbaut, die allerdings nur für die Steuerung der Servos für das Kurvenlicht zuständig sind. Es wird dabei auf CAN-Nachrichten gelauscht, deren Inhalt angibt, in welche Richtung (angegeben in Schritten) der Lichtkegel leuchten soll. Der Lichtkegel kann dabei in vier Richtungen verstellt werden:

- Kurvenlicht: rechts / links
- Leuchtweitenregulierung: oben / unten

Die eigentlichen Leuchtmittel wie Glühbirnen, LEDs und Xenon-Lampen werden direkt mit Strom versorgt, ohne dass dafür ein Steuerboard vorhanden ist.

Da uns mit den TriBoards leistungsfähige Mikrocontrollerboards zur Verfügung standen, die auch die CAN-Nachrichten für das Kurvenlicht bereitstellen konnten, sollte eines dieser TriBoards die komplette Steuerung der beiden Scheinwerfer übernehmen.

Für die zu entwickelnde Software wurden dazu folgende Vorüberlegungen angestellt:

- Das TriBoard sollte CAN-Nachrichten erhalten, die festlegen, welche Lampen ein/ausgeschaltet werden sollen (Abblendlicht, Blinker, usw.).
- Um die Lampen anzusteuern sollten, I/O-Pins benutzt werden.
- I/O-Pins können nur wenig Strom liefern, daher musste der Hauptstrom geschaltet werden (siehe Transistorkästchen).

Zunächst wurde ein Nachrichtenformat festgelegt, das die notwendigen Informationen zur Steuerung der Lampen beinhaltet, sowie eine noch nicht belegte CAN-ID gewählt.

CAN-ID: 500h

Bit	7	6	5	4	3	2	1	0
Bedeutung	-	-	Abblend-licht	Tagfahr-licht	Stand-licht	Fern-licht	Blinker rechts	Blinker links

Tabelle 8.1: Aufbau der CAN-Nachricht zur Steuerung der Scheinwerfer

Die I/O-Pins des TriBoards mussten je nach angeschlossener Funktion einen Pegel halten oder ein alternierendes Signal ausgeben. Die Hauptscheinwerfer benötigen natürlich ein kontinuierliches Signal, ebenso der Spiegel für das Fernlicht und das Tagfahrlicht.

Ein Richtungsanzeiger/Blinker wird hingegen naturgemäß in einem bestimmten Rhythmus ein- und ausgeschaltet. Für Deutschland ist dabei laut StVZO §54 eine Blinkfrequenz von 60/min ±30 einzuhalten.

Eine Besonderheit ist bei dem Tagfahrlicht/Standlicht zu beachten. Diese Leuchteinheit besteht aus LEDs, die beide Funktionen übernehmen.

- Wird das Tagfahrlicht eingeschaltet, so werden die LEDs kontinuierlich mit Spannung versorgt.
- Wird das Standlicht eingeschaltet, so werden die LEDs mit einem PWM-Signal versorgt.

Durch die Ansteuerung mit einem PWM-Signal (Pulsweitenmodulation) können die LEDs gedimmt werden. Da LEDs sehr flink sind, ist es notwendig die Versorgungsspannung mit einer hohen Frequenz ein- und auszuschalten. Durch Veränderung des Verhältnisses von On- zu Offzeit kann damit eine LED in bestimmten Grenzen gedimmt werden, da das menschliche Auge zu träge ist, um das ein- und ausschalten wahrzunehmen. Wird die Frequenz und das Verhältnis falsch gewählt, kann es allerdings zum Flackern kommen.

Fahrzeugfunktion - Lenkrad

Um den zweiten Prototypen auch für Präsentationszwecke zu nutzen, haben wir ein Gaming-Lenkrad (siehe Kapitel 8.2.4) zur Verfügung gestellt bekommen.

Bei einer genauen Prüfung des im Lenkrad befindlichen Potentiometers hat sich ergeben, dass wir asymmetrische Werte für die Lenkradpositionen erhalten. Zudem hat sich herausgestellt, dass TORCS nur Werte zwischen -100 und 100 verarbeiten kann.

Für die Steuerung der Scheinwerfer gab es folgende Vorüberlegungen:

- Unterteilung des Lenkradwinkels in Sektoren von -100 bis +100
- Filtern von häufigen Peaks
- Hebel nach oben/Taste 2B: rechter Blinker an/aus
- Hebel nach unten/Taste 2A: linker Blinker an/aus
- Hebel nach hinten: Abblendlicht an/aus
- Hebel nach vorne: Fernlicht an/aus

Als erstes haben wir die Lenkwinkel des Lenkrads zu beiden Seiten so angepasst, dass wir zwei symmetrische Bereiche haben. Zudem haben wir die Bereiche in Schritte von -100 bis +100 unterteilt. Damit die Lenkung möglichst stabil ist, haben wir einen Bereich von -3 bis 5 als Geradeausfahrt festgelegt. Da wir beim Testen der Software einige ungewollte Ausschläge der Werte beobachtet haben, was auf die schon etwas ältere Hardware zurückzuführen ist, haben wir die häufigsten Peaks ausgefiltert.

Letztendlich haben wir die CAN-Nachricht entworfen, die den Status des Lenkrads übermittelt: CAN-ID: 220h

Byte	4	3	2	1	0
Bedeutung	Tasten	X-Achse	-	Y-Achse	-

Tabelle 8.2: Aufbau der CAN-Nachricht zum Senden des Lenkradstatus

Bei dieser Nachricht ist zu beachten, dass die Tasten dieselben Signale wie der Hebel am Lenkrad senden. Die X-Achse gibt in dieser Nachricht die Werte für den Lenkwinkel und die Y-Achse die Werte für die Bremse und das Gaspedal an.

Mit dem Festlegen der Funktionen des Lenkrads haben wir auch die CAN-Nachricht entworfen, welche an das TriBoard zur Scheinwerfersteuerung gesendet wird:
CAN-ID: 501h

Bit	5	4	3	2	1	0
Bedeutung	Abblend-licht	Tagfahr-licht	Stand-licht	Fern-licht	Blinker rechts	Blinker links

Tabelle 8.3: Aufbau der CAN-Nachricht zur Steuerung der Scheinwerfer mit dem Lenkrad

Im Endeffekt wurde hier die Scheinwerfer-Status-Nachricht genommen, um die Kommandos des Lenkrads ergänzt und dann an das Steuerboard für die Scheinwerfer gesendet.

Einbettung in OSEK

Mess- & Steuerumgebung

Neben der Realisierung der eigentlichen Fahrzeugfunktionen war auch die Realisierung einer Mess- und Steuerumgebung ein wesentlicher Bestandteil der PG.

Grundsätzlich sollte es möglich sein, zwei verschiedene Arten von Messungen vorzunehmen:

- digitale Pegel
- analoge Signale

Die Messung digitaler Pegel soll der Überwachung von I/O-Pins eines Steuergerätes (oder eines Mikrocontrollerboards) dienen. Wird bspw. der I/O-Pin eines Fahrzeugfunktionsboards auf high-Pegel geschaltet, so kann die Mess- und Steuerumgebung dies detektieren.

Um die digitalen Pins zu schützen (Ausgang auf Ausgang geschaltet), sollten die ADCs (A/D-Wandler) benutzt werden, da diese einen sehr hohen Eingangswiderstand haben und daher unempfindlicher sind.

Analoge Signale treten in unserem Aufbau nur im Zusammenhang mit der Strommessplatine auf. Hier wird die Stromversorgung eines Verbrauchers (bis ca. 2A) gemessen, indem der Spannungsabfall an einem Shunt betrachtet wird.

Die Messung der Stromaufnahme soll Aufschluss darüber geben, ob bspw. das Aktivieren/Deaktivieren von Peripherie auf einem Mikrocontrollerboard oder die Ausführung bestimmter Algorithmen/Funktionen zu einem merklichen Anstieg/Abfall der Stromaufnahme führt.

Dazu sind einige Punkte zu beachten, die in der Mess- und Steuerumgebung umgesetzt werden müssen:

- Messung mit hoher Auflösung, um leichte Schwankungen erkennen zu können
- Messung in kurzer zeitlicher Abfolge, um kurzzeitige Veränderungen erkennen zu können

Diese beiden Punkte führen dazu, dass eine relativ große Menge an Daten anfällt. Die FADCs (Fast Analog Digital Converter) auf einem TriBoard arbeiten mit einer Auflösung von 10 Bit. Orientiert man sich an aktuell verfügbaren Datenloggern, so sind Sampleraten von bis zu 250kS/sec. üblich.

Dies führt zu 2,5 MBit an Daten pro Sekunde. Da die Daten über den CAN-Bus an einen PC gesendet werden sollen, ist es notwendig, die Datenmenge pro Sekunde einzuschränken. Da die Stromaufnahme nur zu bestimmten Zeitpunkten, bei denen eine merkliche Veränderung in der Stromaufnahme anzunehmen ist, interessiert, ist es sinnvoll, Messungen zu speichern und später abzurufen. Das Abrufen der Messdaten kann dabei mit langsamer Geschwindigkeit erfolgen. Um auch schon während einer Messung Daten auswerten zu können, soll ein Anteil der Samples sofort gesendet werden.

Die Speicherung der Messdaten macht natürlich erforderlich, dass Zeitstempel verwendet werden. Diese Zeitstempel ermöglichen es, die später ausgelesenen Werte mit bestimmten Ereignissen (Aufrufen einer Methode, Ansteuern eines I/O-Pins, Einschalten der ADCs, etc.) in Bezug zu setzen.

Die verwendeten Zeitstempel werden als 32-Bit Werte realisiert, und ähnlich einem Counter verwendet. Beim Start der Software wird der Zeitstempel mit 0 initialisiert und danach ständig weiter hochgezählt (bei einem Überlauf wird wieder bei 0 gestartet). Eine Messung erfolgt dabei immer zu einem eindeutigen „Zählerstand“, und kann bei der Auswertung als Referenz verwendet werden.

Neben der Messung von Pegeln/Signalen eines angeschlossenen Steuergerätes/Mikrocontrollerboards soll eine Steuerung dieses Boards möglich sein. Diese Steuerung soll sich zunächst auf das Ein-/Ausschalten und die Möglichkeit eines Resets beschränken.

Ein TriBoard war auch hier die Grundlage für die Erstellung der beschriebenen Mess- und Steuerumgebung, die über CAN-Nachrichten gesteuert werden soll.

Für die zu entwickelnde Software wurden dazu folgende Vorüberlegungen angestellt (siehe Kapitel 4, Anforderungen an das Projekt):

- Das Messen digitaler Pegel muss möglich sein.
- Das Messen analoger Signale muss möglich sein.

- Das TriBoard sollte CAN-Nachrichten erhalten, die festlegen, welche Funktion ausgeführt wird (Strom messen, Daten senden,...).

Zunächst wurde ein Nachrichtenformat festgelegt, das die notwendigen Informationen zur Steuerung der Mess- und Steuerumgebung enthält, sowie eine noch nicht belegte CAN-ID gewählt.
CAN-ID: 400h

Bit	7	6	5	4	3	2	1	0
Bedeutung	-	-	-	-	Messwerte senden	Strom messen	Reset	Ein/Aus

Tabelle 8.4: Aufbau der CAN-Nachricht zur Steuerung der Mess- und Steuerumgebung

Die Daten, die von der Mess- und Steuerumgebung versendet werden, mussten ebenfalls ein eindeutiges Nachrichtenformat erhalten:

Stromaufnahme
CAN-ID: 401h (On-the-fly während einer Messung)
CAN-ID: 700h (gespeicherte Werte)

Byte	0	1	2	3	4	5	6	7
Bedeutung	Zeit-stempel	Zeit-stempel	Zeit-stempel	Zeit-stempel	Messwert Bits 9-8	Messwert Bits 7-0	-	-

Tabelle 8.5: Aufbau der CAN-Nachricht von Analogen Messdaten

Digitale Pegel
CAN-ID: 402h

Byte	0	1	2	3	4	5	6	7
Bedeutung	Zeit-stempel	Zeit-stempel	Zeit-stempel	Zeit-stempel	Messwerte	-	-	-

Tabelle 8.6: Aufbau der CAN-Nachricht von Digitalen Messdaten

Das Byte für Messwerte ist dabei folgendermaßen aufgebaut:

In der Tabelle 8.7 beziehen sich die Angaben der Form „ANxx“ auf den entsprechenden Eingang des ADCs. Diese Bezeichnung wird ebenfalls auf den Erweiterungsboards verwendet.

TORCS

Als Demonstrator für unsere Arbeit war es unser Ziel den Aufbau in eine Fahrsimulation zu integrieren und diese damit zu steuern. Zu diesem Zweck bot sich das Open-Source-Projekt TORCS an. Der Fortschritt der Projektgruppe ermöglichte es zum Zeitpunkt der Demonstration

Bit	7	6	5	4	3	2	1	0
Bedeutung	-	AN11	AN10	AN9	AN8	AN7	AN2	AN0

Tabelle 8.7: Aufbau des Byte für die Messwerte in der CAN-Nachricht von digitalen Messdaten

sowohl den Gangwahlschalter, als auch die Scheinwerfer in den Aufbau zu integrieren. Sämtliche Funktionen des Gangwahlschalters konnten dabei zu Verarbeitung in der Rennsimulation genutzt werden. Auch beim Scheinwerfer konnten wir die meisten Funktionen integrieren: Blinker, Standlicht, Tagfahrlicht, Abblendlicht, Fernlicht, sowie Kurvenlicht abhängig vom Lenkwinkel. Das in Kapitel 8.2.4 beschriebene Gaming-Lenkrad diente dabei als Ersatz der nicht einsetzbaren Lenksäule.

TORCS selbst lief auf einem Laptop. Der Quellcode des Spiels wurde so modifiziert, dass es mittels eines selbstgebauten USB-Adapters auf dem CAN-Bus zugreifen konnte. So war es möglich, die Daten direkt vom CAN-Bus auszulesen und zu verarbeiten. Für TORCS relevant waren dabei eine Nachricht vom Gangwahlschalter sowie eine Nachricht vom TriCore-Board des Lenkrads. Die Gangwahlschalter-Nachricht enthielt Informationen über den jeweiligen Modus (Automatik: Drive, Neutral, Parken, Tiptronic), die Nachricht des Lenkrads enthielt hingegen die Stellung des Lenkrads und der Pedalerie sowie den Status der Blinker, des Abblendlichts und des Fernlichts. Neben den empfangenen Nachrichten sendete TORCS selbst eine Status-Nachricht. Diese enthielt die Geschwindigkeit des Fahrzeugs zur Verarbeitung in anderen Fahrzeugkomponenten (z.B. im Gangwahlschalter).

Im Quellcode selbst bedurfte es einiger Suche, um eine Stelle zu finden, an der wir diese Informationen ins Spiel einfließen lassen konnten. Letztendlich sind wir auf die Datei human.cpp gestoßen, in der unter anderem die Fahrzeugsteuerung implementiert war. In eine Initialisierungsmethode, in der beispielsweise die Joystickinitialisierung untergebracht war, fügten wir die Initialisierung des USB-CAN-Adapters ein. Außerdem wurde die USB-Verbindung gestartet und ein Nachrichtenfilter gesetzt, damit nur für das Spiel bestimmte Daten vom CAN-Bus gelesen und in die Queue aufgenommen werden. Die Steuerung des Spiels selber fand hauptsächlich in der Methode „common_drive“ statt. Dort werden unter anderem die Tastatur-, Maus- und Joystickeingaben abgefragt. In dieser Methode haben wir das Auslesen des USB-CAN-Adapters implementiert. Anhand der Nachrichten wurde dann überprüft, was zu tun ist. So konnte im Spiel das Licht des Fahrzeugs ein- und ausgeschaltet werden, geschaltet werden und natürlich das Fahrzeug gesteuert werden. All diese Methoden wurden in der Methode common_drive umgesetzt. Dort haben wir das Spiel auch dahingehend erweitert, dass während des Rennens zwischen dem Automatik- und dem Tiptronicmodus hin- und hergeschaltet werden kann. In der Originalversion ist die Wahl zwischen diesen beiden Modi nur vor einem Rennen über das Menü vorgesehen. Von der Eventloop von TORCS erfolgt der Aufruf einer der beiden Methoden „drive_at“ und „drive_mt“, die ebenfalls in der Datei human.cpp zu finden sind. Welche Methode aufgerufen wird, hängt davon ab, ob die manuelle Schaltung oder die Automatikschaltung ausgewählt wurden. „drive_at“ steht dabei für „drive automatic transition“, „drive_mt“ entsprechend für „drive manual transition“. Wurde nun während des Spiels umgeschaltet, wurde von uns eine Variable gesetzt, die den ausgewählten Schaltmodus speichert. Beim Aufruf der Methoden durch die Eventloop, wird nun gleich zu Beginn überprüft, ob die Methode zum gewünschten, in der Variable gespeicherten Spielmodus passt. Falls dies nicht der Fall ist, wird die Methode entsprechend gewechselt. Während die manuelle Schaltung von uns selbst implementiert wurde, wurde die Automatikschaltung von TORCS übernommen. Eine besondere Herausforderung war die Lenkbarkeit mit

dem Lenkrad. Eine 1:1-Umsetzung der empfangenen Signale hätte zu einer äußerst empfindlichen Lenkung geführt, was das Spiel besonders bei höheren Geschwindigkeiten beinahe unspielbar gemacht hätte. Daher war es erforderlich, die Lenkwirkung an die Geschwindigkeit des Fahrzeugs zu koppeln. Außerdem haben wir vorherige Lenkungswerte in die Berechnung der Lenkwirkung einbezogen, um eine allzu ruckartige Lenkung zu vermeiden und die Lenkung möglichst flüssig zu gestalten. Darüber hinaus hatte das alte Gaming-Lenkrad ein relativ schlechte Mechanik, so dass zum Beispiel der Nullzustand des Lenkrads nicht immer den selben Wert hatte. Diese falschen Werte hieß es zu ignorieren, so dass zumindest eine Geradeaus-Fahrt halbwegs möglich war. Es brauchte einige Tests, bis wir ein zufriedenstellendes Ergebnis erreicht hatten.

8.2.4 Hardware

System Basis Chip des Gangwahlschalters

Der System Basis Chip des Gangwahlschalters dient als Basis-Chip auf der Platine. Über ihn werden Bauteile mit der benötigten Spannung versorgt, er dient als Transceiver für CAN-Botschaften und über ihn kann weitere Peripherie aktiviert werden. Konfigurieren lässt sich der System Basis Chip über 16 Bit lange SPI-Nachrichten, die vom Mikrocontroller an den System Basis Chip gesendet werden. Mit diesen SPI-Nachrichten können die Werte in den Registern zur Konfiguration des System Basis Chips geändert werden.

Unmittelbar nach dem Anlegen der Versorgungsspannung an den Gangwahlschalter schaltet der System Basis Chip die Versorgungsspannung V1 für den Mikrocontroller ab, falls der System Basis Chip nicht mit einer der nachfolgenden Methoden zuvor in den Software Development Mode versetzt wurde.

1. Software Development Mode per “Hardware”
Diese Methode muss genutzt werden, falls fehlerhafte Software auf den Gangwahlschalter geflasht wurde und für den Mikrocontroller keine Spannung mehr über V1 ausgegeben wird. In diesem Fall wird die Leiterbahn zwischen dem Test-Pin (Pin 16) des System Basis Chips und der unmittelbar folgenden Durchkontaktierung aufgetrennt. Dieser Schritt ist notwendig, da der Test-Pin direkt mit der Masse der BDM-Schnittstelle verbunden ist und an den Test-Pin 6V angelegt werden müssen. Das Auftrennen der Leiterbahn ist bereits bei allen aktuell vorhandenen Gangwahlschaltern durchgeführt. Um nun 6V an den Test-Pin anzulegen, muss folgendermaßen vorgegangen werden. An die Hauptstromversorgung (Bananenstecker) des Gangwahlschalter werden 13,8V angeschlossen (Netzteil noch nicht anschalten!). Von einem weiteren Netzteil mit 6V wird die Masse mit der Masse des 13,8V-Netzteils verbunden. Nun kann der Plus-Pol des 6V-Netzteils mit dem Test-Pin verbunden werden, z.B. über eine Messspitze eines Multimeters. Zuerst wird das 6V-Netzteil und anschließend das 13,8V-Netzteil angeschaltet. Jetzt wird der Mikrocontroller mit Strom versorgt und es kann neue Software auf den Gangwahlschalter geflasht werden.

2. Software Development Mode per “Software”
Um den System Basis Chip per Software in den Software Development Mode zu bringen, ist es nötig, zu Beginn der Main-Methode eine entsprechende SPI-Nachricht zu versenden. Diese Nachricht sollte direkt als erstes nach den CodeWarrior Methoden *PE_low_level_init* und *SM1_Init*

gesendet werden. So lange diese Nachricht immer zu Beginn korrekt gesendet wird, muss der Gangwahlschalter nicht über die erste Methode aufgeweckt werden.

Um den CAN-Transceiver des System Basis Chips zu aktivieren, müssen zwei SPI-Nachrichten gesendet werden. Mit der ersten SPI-Nachricht wird der System Basis Chip in den Normal Mode versetzt (Achtung, dieser Mode ist unabhängig vom zuvor genannten Software Development Mode!) und mit der zweiten wird der CAN-Transceiver aktiviert. Ohne die Aktivierung des CAN-Transceivers können CAN-Nachrichten weder vom Mikrocontroller versendet noch empfangen werden.

Des weiteren verfügt der System Basis Chip über einen Enable-Pin um weitere Peripherie zu aktivieren. Für die volle Funktionstüchtigkeit des Gangwahlschalters muss der Enable-Ausgang auf High gesetzt werden, was wiederum über eine entsprechende SPI-Nachricht geschieht.

Funktionalität des Gangwahlschalters

Der Gangwahlschalter konnte erstmals im zweiten Prototypen in Betrieb genommen werden. Dort wurde er in eine Demonstratorumgebung integriert. Teil dieser Umgebung war ebenfalls das frei zur Verfügung stehende Spiel "TORCS". Mit Hilfe des Gangwahlschalters kann das virtuelle Fahrzeug gesteuert werden. Man hat die Wahl zwischen dem Automatikmodus oder dem Tiptronic-Modus. Bei der Programmierung haben wir uns an der Funktionalität des Gangwahlschalters des BMW X5 orientiert.

Im Automatikmodus hat man die Möglichkeit zwischen den Modi R, N, P und D zu wählen. Nach dem Einschalten des Gangwahlschalter befindet sich dieser standardmäßig im P-Modus. Um aus diesem heraus zu gelangen, muss neben dem gleichzeitigen Treten der Bremse die unbeschriftete Taste seitlich am Gangwahlschalter gedrückt werden. Diese muss ebenfalls betätigt werden, wenn man in den R-Modus wechseln möchte. Es kann allerdings nur in den Tiptronic-Modus gewechselt werden, wenn man sich im D-Modus befindet. In den P-Modus gelangt man, indem die P-Taste gedrückt wird.

Zusätzlich besitzt der Gangwahlschalter eine Nachtbeleuchtung und einen Sportknopf. Die Nachtbeleuchtung wird aktiviert, sobald die Scheinwerfer eingeschaltet worden sind. Das TriCore-Board (TC 1766) versendet dazu über den CAN-Bus eine CAN-Nachricht, welche Informationen über den aktuellen Lichtstatus enthält. Diese spezielle CAN-Nachricht besitzt die CAN-ID 501. Empfängt der Gangwahlschalter diese Nachricht, dann reagiert er entsprechend darauf. Auch der Gangwahlschalter versendet über den Bus CAN-Nachrichten, welche mit der CAN-ID 201 versehen sind. Diese enthalten Informationen über den aktuellen Status, wie Tiptronic, Gang hoch, Gang runter, Automatik, Schalterstellung, Sportknopf und Sequenznummer. Die Sequenznummer wird benötigt, um bei eventuell verloren gegangenen CAN-Nachrichten den Schaltvorgang trotzdem sicher an die anderen Steuergeräte zu übertragen. Dazu wird die entsprechende CAN-Nachricht jedes getätigten Schaltvorgangs mehrfach mit der gleichen Sequenznummer gesendet. So kann trotz verlorener CAN-Nachrichten genau ein Schaltvorgang von den Busteilnehmern erkannt werden. Die Funktionalität des Sportknopfes haben wir so realisiert, dass beim Drücken von diesem das Fernlicht der Scheinwerfer aktiviert wird sowie der Blinker links eingeschaltet wird.

Gaming-Lenkrad Messadapter

Für den zweiten Prototypen wurde ein Ersatz für das Lenksäulenmodul benötigt, da die Rennsimulation „TORCS“ (siehe Abschnitt 8.2.3) mit dem vorhanden Hardwareaufbau gesteuert werden sollte. Ein zur Verfügung gestelltes PC-Gaming-Lenkrad hat dabei diese Rolle übernommen.

Ein TriBoard wurde auch in diesem Fall wieder als Steuerboard eingesetzt, um Informationen des Lenkrads (Lenkwinkel, Pedalstellung, Tasten) in CAN-Nachrichten umzuwandeln, und so die Verbindung zum Rest des Aufbaus herzustellen. Außerdem haben wir auf diesem Board die Steuerung des Fern-, Abblend- und Kurvenlichts und der Blinker realisiert.

Da das erwähnte PC-Gaming Lenkrad bereits etwas älter war, hatte es den Vorteil, dass es noch für den Anschluss an den PC-Gameport vorgesehen war. Dieses inzwischen völlig von modernen PCs verschwundene Interface, besticht durch die extrem simpel aufgebaute externe Hardware in Form von Joysticks, Steuerhörnern oder Lenkrädern.

Alle Joysticks dieses Standards benutzen 100 kΩ Widerstände/Potentiometer zur „Kodierung“ von gedrückten Tasten oder Joystickbewegungen/Lenkwinkeln. Der Anschluss erfolgt dabei über eine 15-polige SUB-D Verbindung, dessen Pinbelegung in Tabelle 8.8 aufgeführt ist. Die auf-

Pin	Bedeutung
1	+ 5 V DC
2	Joystick 1, erster Button
3	Joystick 1, X-Position
4	Masse
5	Masse
6	Joystick 1, Y-Position
7	Joystick 1, zweiter Button
8	+ 5 V DC
9	+ 5 V DC
10	Joystick 2, erster Button
11	Joystick 2, X-Position
12	Masse
13	Joystick 2, Y-Position
14	Joystick 2, zweiter Button
15	+ 5 V DC

Tabelle 8.8: Pinbelegung des PC-GamePorts

geführten Spannungen von + 5 V DC beziehen sich dabei auf die vom PC gelieferte Spannung. Da diese jedoch lediglich über Widerstände/Potentiometer geführt wird, kann sie als unkritisch angesehen werden und bedeutet ein weiterer Vorteil, da die TriBoards nur 3,3 V zur Verfügung stellen.

Eine genauere Untersuchung der Lenkrad-Hardware ergab, dass die Potentiometer für Lenkrad und Pedale nicht wie üblich mit dem Schleifkontakt zwischen 0 und 100kΩ beschaltet waren, sondern als Ganzes einen veränderlichen Widerstand darstellten.

Die Tasten werden beim Betätigen standardmäßig auf Masse gezogen und liegen offen im un-

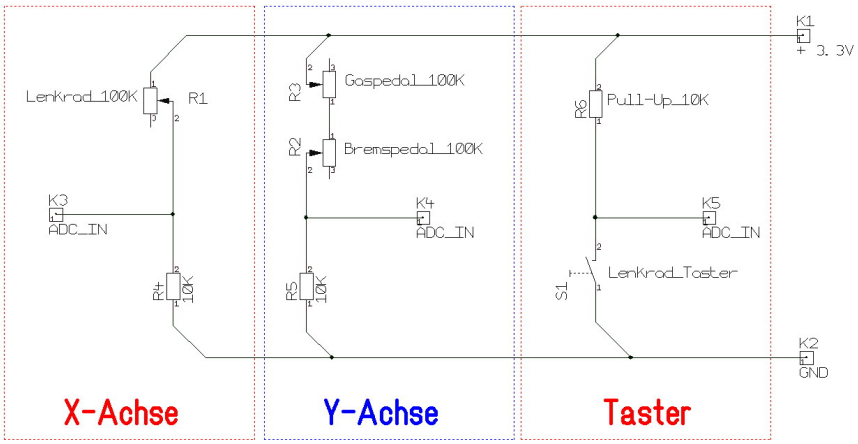


Abbildung 8.3: interne Beschaltung des PC-Gaming Lenkrads inkl. Vorüberlegungen zum Anschluss an ein TriBoard

betätigten Fall.

Anschluss an ein TriBoard

Für den Anschluss des Lenkrads sollten auf dem TriBoard die ADCs sowohl für die analogen Signale des Lenkrads und der Pedale als auch für die Tasten verwendet werden.

Die Tasten sind relativ einfach an ein TriBoard anzuschließen. Beim Drücken einer Taste liegt der entsprechende ADC-Eingang auf Masse, und der ADC liefert den Wert 0. Um auch im unbetätigten Zustand einer Taste einen definierten Wert zu erhalten, wurden die ADC-Eingänge jeweils per Pull-Up Widerstand auf Betriebsspannung (+ 3,3 V) gehalten.

Die intern für Lenkrad und Pedale verbauten Potentiometer wurden jeweils mit einem weiteren Widerstand zu einem Spannungsteiler erweitert. Die dort abfallende Spannung konnte dann in Abhängigkeit von Lenkradstellung bzw. Pedalstellung ausgewertet werden.

Die interne Beschaltung für Lenkrad, Pedale und Tasten, sowie die Vorüberlegungen zum Anschluss an ein TriBoard sind in Abbildung 8.3 dargestellt.

Ein Problem war in diesem Zusammenhang die nicht symmetrische Ausgangsspannung für Lenkwinkel und Pedalstellung. Diese Unzulänglichkeit der Hardware sollte allerdings per Software behoben werden.

Die o.a. Vorüberlegungen führten zu einer kleinen Adapterplatine, die die notwendigen Widerstände trug. Eine 15-pol. SUB-D-Buchse ermöglichte den problemlosen Anschluss des PC-Gaming-Lenkrads oder auch eines beliebigen anderen Standard-PC-Joysticks, der für den Gameport-Anschluss vorgesehen ist. Über eine Pfostenbuchse konnte diese Adapterplatine direkt auf eine Erweiterungsplatine eines TriBoards aufgesteckt werden. Der Stromlaufplan für die Adapterpla-

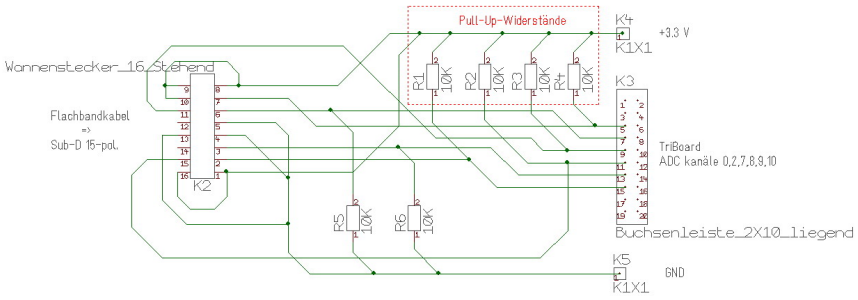


Abbildung 8.4: Stromlaufplan der Adapterplatine zum Anschluss eines PC-Joysticks an das TriBoard

tine ist in Abbildung 8.4 dargestellt.

8.2.5 CAN-Bus-Treiber

CAN-Treiber für den Gangwahlschalter

Der Gangwahlschalter muss über den CAN-Bus mit dem Fahrzeugnetz kommunizieren können. Der Freescale Mikrocontroller MC9S12DG128 verfügt über das Highspeed CAN-Modul MSCAN12, das den Versand und Empfang von CAN-Nachrichten ermöglicht. Das von Freescale sehr gut dokumentierte [msc04, msc05] Modul gestattete die Implementierung der CAN-Bus-Kommunikation in übersichtlichen 60 Codezeilen. Da der Quellcode des Gangwahlschalters ausführlich dokumentiert ist, soll an dieser Stelle nur das grundsätzliche Vorgehen erläutert werden.

Als erstes muss das MSCAN-Modul aktiviert werden. Dazu wir das Modul in den Initialisierungsmodus überführt. Hier werden durch Schreiben in entsprechende Register der Taktgeber, die Timings und ID-Filter eingestellt. Das Aktivieren des MSCAN-Moduls und das Setzen des Taktgebers auf den Oszillatortakt wird durch Schreiben in das MSCAN Control 1 Register realisiert. Der CAN-Bus des Fahrzeugnetzes wird mit einer Baudrate von 500 Kbps betrieben. Damit der Gangwahlschalter ebenfalls mit dieser Baudrate kommuniziert, muss im MSCAN Bus Timing Register 0 der Baudratenvorteiler (Prescaler) entsprechend eingestellt werden. Dieser berechnet sich durch folgende Formel:

$$\text{Bit Time} = \frac{\text{Prescale value}}{f_{CANCLK}} * (1 + \text{Time Segment 1} + \text{Time Segment 2})$$

Im Fall des Gangwahlschalter ergibt sich die Gleichung:

$$\frac{1}{500\ \text{kHz}} = \frac{1}{8\ \text{MHz}} * (1 + 11 + 4)$$

Als nächstes werden vier 16-Bit-ID-Filter definiert. Ein Filter besteht aus zwei Teilen. Die Maske bestimmt welche der 16 Bits ausgewertet werden müssen. Der Akzeptanzfilter stellt die eigentliche

16-Bit-ID dar. Zum Abschluss der Initialisierung wird das Modul durch Schreiben in das MSCAN Control Register 0 in den Betriebsmodus versetzt.

Das CAN-Modul enthält drei Sendepuffer. Aus den Registern kann ein leerer Puffer ermittelt und zum nächsten Versand markiert werden. Nach der Angabe der ID und der zu versendenden Nachricht selbst kann die Übertragung beginnen. Ob die Übertragung erfolgreich war, kann ebenfalls aus den Registern des CAN-Moduls ausgelesen werden.

Um viele Nachrichten in sehr kurzer Zeit empfangen zu können, verfügt das MSCAN12-Modul über fünf Empfangspuffer, die über eine FIFO Struktur verfügen. Das Receive Buffer Full Flag (RFLG) ist gesetzt, wenn mindestens einer der fünf Puffer belegt ist. Es wird immer die zuerst angekommene CAN-Nachricht ausgelesen. Um nach dem Lesen den Empfangspuffer wieder freizugeben, wird das RFLG gesetzt. Falls keine weitere Nachricht vorliegt, bleibt das RFLG auf null. Ist aber ein weiterer Puffer gefüllt, setzt das CAN-Modul das RFLG sofort wieder auf 1.

Zu beachten ist, dass das MSCAN12 Modul nicht direkt an die CAN-High und CAN-Low-Leitungen des CAN-Busses angeschlossen werden, sondern einen Transceiver benötigen. Diese Funktion übernimmt der System Basis Chip des Gangwahlschalter.

8.3 Stand der Realisierung

Der zweite Prototyp implementiert bereits große Teile der von uns gewünschten Funktionalitäten. Jedoch erkennt man an einigen Stellen, dass besonders auf Funktionen geachtet wurde, die für eine Demonstration der Projektgruppe nötig waren. Somit fehlen noch einige Feinheiten, besonders in der Steuer- und Messumgebung.
Der Gangwahlschalter konnte als erste Hardwarekomponente mit selbst geschriebener Software integriert werden. Diese ist sogar schon flashbar. Zusammen mit dem Gaming-Lenkrad kann die Fahrzeugsimulation TORCS gesteuert werden, wobei die Scheinwerfer dabei zum Beispiel mit Kurvenlicht reagieren. Außerdem ist für die Mess- und Steuerumgebung die Software schon entwickelt und funktionsfähig.
Es muss an folgenden Bereichen weiter gearbeitet werden:

- Komponenten: Möglichst viele der uns zur Verfügung stehenden Komponenten sollen in den Aufbau integriert werden. Dazu muss an den noch verbliebenen Komponenten weiter gearbeitet werden. Zur Verfügung stehen uns derzeit noch: die Fahrzeugtür, das Lenksäulenmodul (könnte Gaming-Lenkrad ersetzen) sowie das Dachmodul (würde Gateway-Knoten benötigen, da ein Low-Speed-CAN erforderlich ist)
- Software: Bis auf die Messsoftware läuft die komplette Software bereits auf einem Osek-System. Bisweilen handelt es sich allerdings um ein einziges Osek-System. Hier ist es das Ziel, dass die einzelnen Hardware-Komponenten ihre eigenen Steuer-Boards bekommen, um so möglichst modular einsetzbar zu sein. Dazu muss die Software aufgeteilt werden. Darüber hinaus muss die Messsoftware noch für Osek angepasst werden.
- Steuer- und Messumgebung: Um alle Hardware-Komponenten steuern bzw. deren Verhalten messen zu können, sind weitere TriCore-Boards notwendig, die in den Aufbau integriert werden müssen.

- Messdaten visualisieren: Die von den Messboards aufgezeichneten Daten sollen visualisiert werden können. Da wir uns dazu entschlossen haben, CANoe zur Visualisierung zu nutzen, bedarf es eines Konverters. Die Messboards senden auf Anfrage all ihre Daten über den Bus, sodass CANoe sie mitloggen kann. Dieses Logfile soll so konvertiert werden, dass ein Logfile entsteht, in welchem die aufgezeichneten Daten an den richtigen Stellen eingefügt wurden.
- Flashen: Bislang kann nur der Gangwahlschalter geflasht werden. Dies soll aber auch etwa für die TriCore-Boards, über die die Fahrzeugfunktionen realisiert sind, möglich sein.
- Brettaufbau: Um einen Brettaufbau zu realisieren, muss dieser geplant werden. Alternativ könnte zunächst eine abgeschwächte Form eines solchen Aufbaus realisiert werden, in dem zumindest alle Boards in Gehäusen untergebracht sind. Für beide Fälle sind allerdings weitere Kabel notwendig, die erstellt werden müssen.
- Fernsteuerung: Die Steuer- und Messumgebung ist bereits darauf ausgelegt, Geräte an und ausschalten zu können. Wie die Steuerboards selbst geschaltet werden können und wie zum Beispiel das CANoe-PC-System ansprechbar ist, muss weiter geplant werden.

Kapitel 9

Erster öffentlicher Auftritt

Anlässlich des jährlich stattfindenden Studieninformationstags und des damit verbundenen Campusfests wurde das Projekt zum ersten Mal einer breiten Öffentlichkeit präsentiert.

Bevor der Schritt in die Öffentlichkeit getan werden konnte, mussten einige organisatorische Dinge, wie zum Beispiel die Planung eines Ausstellungsstands und die Einteilung in betreuende Teams vorbereitet werden.

9.1 Vorbereitung und Organisation

Zuerst wurde ein Aufbau für den Prototypen entworfen.

Da die Realisierung des Prototyps den Einsatz der Rennsimulation „TORCS“ umfasste, sollte ein möglichst realistisches Fahrgefühl vermittelt werden. Dazu wurde uns von der Firma KOSTAL freundlicherweise ein echter Fahrersitz für die Dauer des Campusfests zur Verfügung gestellt. Dieser Sitz wurde auf eine Holzkonstruktion montiert, um eine geeignete Sitzhöhe in Verbindung mit dem zum Prototyp gehörenden Lenkrad zu erreichen. Nachdem diese Sitzhöhe fest stand, wurde eine stabile Holzkonstruktion für den Gangwahlschalter angefertigt und neben dem Fahrersitz verschraubt.

Zusätzlich zu diesem Aufbau wurden die Voraussetzungen für eine visuelle Benutzerschnittstelle in Form einer Beamer-Präsentation des Rennspiels geschaffen. Hierfür wurde eine Leinwand und ein Beamer zur Verfügung gestellt. Somit konnte die Reaktion auf die Lenk- und Schaltbewegungen bestens beobachtet werden.

Des weiteren wurde der eigentliche Hardwareaufbau, bestehend aus dem Netzteil zur Stromversorgung, der CAN-Bus-Konstruktion, den beiden Scheinwerfern, den beiden Tricore-Boards, dem CANoe-Rechner, dem Linux-Rechner für die Rennsimulation sowie umfangreicher Verkabelung zur Verfügung gestellt.

Um den Besuchern einen Überblick über die Arbeiten der Projektgruppe geben zu können, wurden zwei Plakate mit der Motivation, den Zielen der PG, dem Vorgehen und der momentanen Realisierung entworfen.

Für die Betreuung des Ausstellungsstands wurden Teams gebildet, die jeweils in zweistündigen Schichten den Aufbau betreuten und die Fragen der Besucher beantworteten. Um die PG-Teilnehmer direkt erkennen zu können, wurden T-Shirts mit einem selbstentworfenen Logo angefertigt.

9.2 Bericht über das Campusfest

Das Campusfest hat am 14. Juni 2008 stattgefunden. Das Besucherinteresse schwankte. Es gab vereinzelte Firmenkontakte und zeitweise waren bis zu 20 Besucher am Stand. Allerdings waren tagsüber viele Besucher jüngere Kinder in Begleitung ihrer Eltern (meist Angehörige der Universität). Somit war unsere Präsentation, die auf die Zielgruppe der älteren Schüler und angehenden Informatikstudenten ausgelegt war, hier nicht optimal und es konnte nur wenig inhaltlich über das Projekt vermittelt werden. Gegen Ende des Tages waren dann eher Informatikstudenten, die zuvor andere Stände betreut hatten, an unserem Stand.

Das Campusfest war insgesamt für uns ein guter Erfolg: Die Software hat gehalten und auf aufgetretene Probleme wurde flexibel reagiert. Da der Aufbau zuvor nie solange im Dauerbetrieb war, konnten wir unsere Entwurfsentscheidungen somit live bestätigen. Leider kam es aber zu einem Defekt des Netzteils, wobei nicht klar ist, ob dies nicht schon beim Transport zum Audimax passiert ist. Daher konnten teilweise nicht alle Funktionen gezeigt werden. Daneben gab es ein paar Probleme, welche bei der Vorbereitung zukünftiger Veranstaltungen beachtet werden sollten, um den Erfolg zu maximieren:

- Die Plakate standen zu tief im Raum.
- Auch der Aufbau stand zu tief im Raum, weshalb viele Gäste zunächst dachten, das Spiel sei unser Produkt.
- Beim Standdienst fehlten Rollenverteilungen. Es wäre sinnvoll gewesen jeweils eine Person zu benennen, die Besucher anspricht.
- Es gab keine Flyer für interessierte Gäste und auch keine Visitenkarten um Kontakte zu halten.
- Kekse oder Gummibärchen könnten helfen, um Leute näher heran zu holen, die nur aus der Entfernung zugeguckt haben.
- Teilweise war die Gruppe unterbesetzt und konnte mit nur zwei Leuten den Andrang nicht bedienen.
- Der Aufbau war zeitlich knapp. Zwar wurden wir fast pünktlich um 10 Uhr damit fertig, allgemein gewünscht war von der Campusfestleitung aber, dass dies schon um 9:30 der Fall ist.

- Auch beim Abbau waren wir langsamer als alle anderen. Auch hier wurde die geplante Stunde eingehalten, doch schon um 16:10 waren fast alle anderen Gruppen weg und der Abstellraum zugesperrt.
- Ein Schild in der oberen Etage, welches den Weg nach unten zeigt, wäre hilfreich, da es für Ortsunkundige nicht ersichtlich war, wo die Treppe eigentlich ist.
- Die Triboards sollten auf dem Tisch befestigt werden, damit sie nicht herunter rutschen können.

9.3 Fazit

Unser Auftritt auf dem Campusfest ist gut gelungen. Der vorab geplante zweite Prototyp konnte realisiert und demonstriert werden. Die direkte Zielgruppe war weniger vertreten als erhofft, aber es waren trotzdem viele Besucher am Stand. Es sind einige Probleme aufgetreten, die in der Zukunft vermieden werden sollten, aber der erfolgreiche Umgang mit diesen bestätigte, dass unsere Anforderungen nach Vorführbarkeit und Flexibilität erfüllt worden sind.

technische universität dortmund

Fakultät für Informatik

ESS AG Embedded System Software

PG 522 : AUTOLAB

Eine Experimentierplattform für automotive Softwareentwicklung

MOTIVATION

Moderne Autos sind **verteilte, eingebettete Rechnersysteme** auf Rädern.

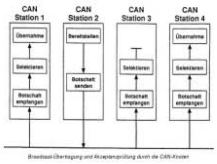
Ohne den Einsatz von **Software** können moderne Fahrzeugfunktionen nicht realisiert werden.

Die Software läuft auf elektronischen Modulen (**Steuergeräten**), die die Steuerung und Regelung übernehmen.

Kernstück der Steuergeräte sind **Mikrocontroller,** die die Verarbeitungsaufgaben durchführen.

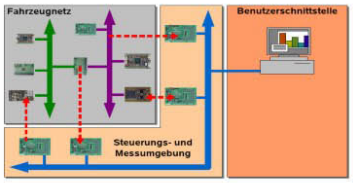
Verbunden werden diese Steuergeräte über verschiedene **Bussysteme**, wie z.B. CAN, LIN oder Flexay.

Die Kommunikation der Steuergeräte erfolgt über **Nachrichtenaustausch.**



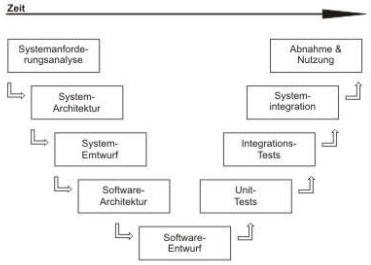
ZIELE DER PG

- Entwurf eines **Fahrzeugelektronik-Versuchsaufbaus** mit einem CAN-basierten Karosserienetzwerk als Grundlage
-
-
- Entwurf einer **Steuerungs- und Messumgebung** um das Programmieren der Hardware sowie das Erfassen des **Energieverbrauchs**, der



VORGEHEN

Orientierung am **V-Modell** als grundlegende Projektmanagment-Struktur



Betreuer der PG 522 Autolab: Prof. Dr.-Ing. Olaf Spinczyk, Horst Schirmeier, Jochen Streicher
Teilnehmer: Alexander Berger, Oliver Botschkowski, Stephan Braun, Sabrina Hecke, Florian Hohnsbehn, Christian Horn, Gregor Kaleta, Boris Konrad, Sebastian Kosch, Matthias Meier, Robert Neue und Thomas Romanek

Abbildung 9.1: Plakat 1 für das Campusfest

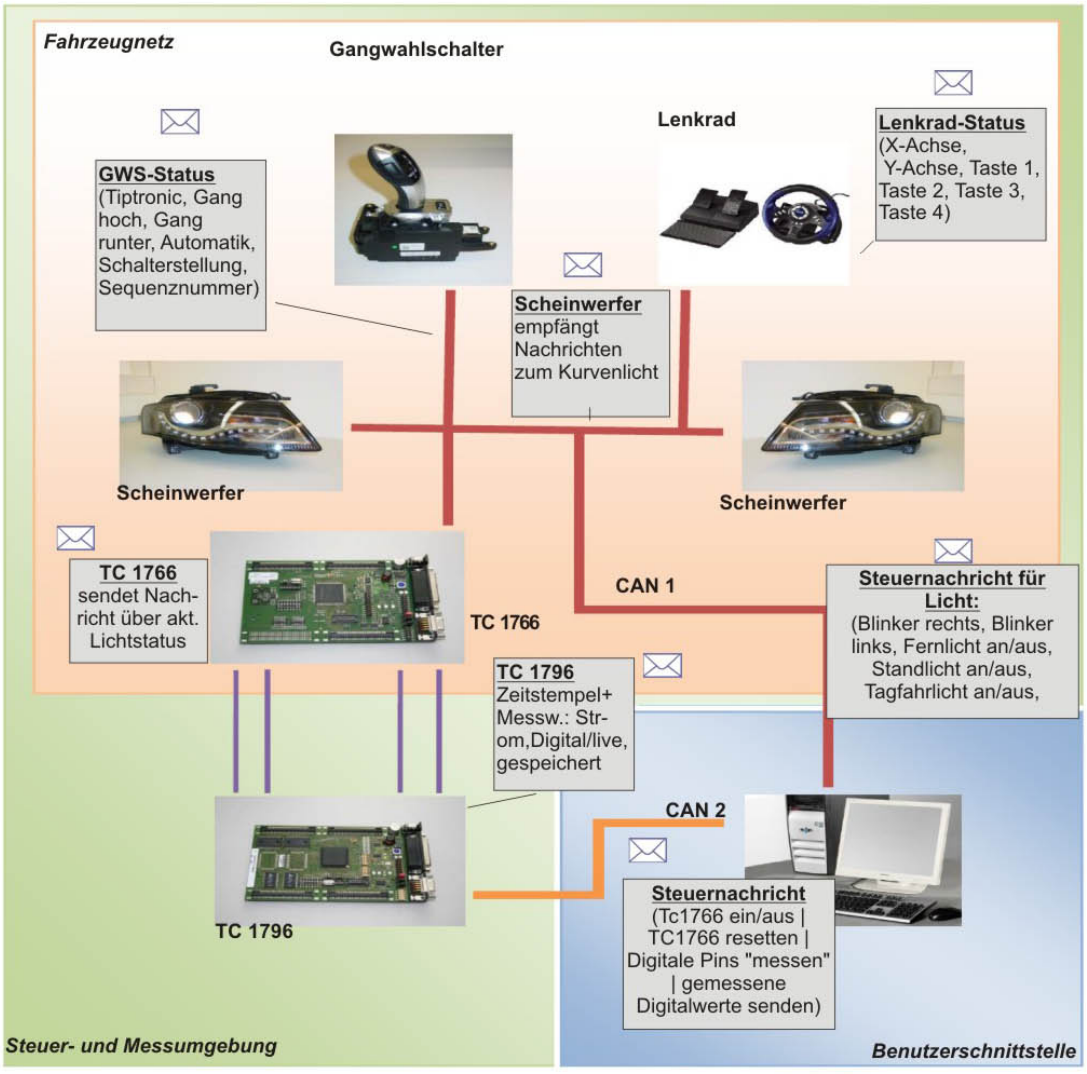
technische universität dortmund

Fakultät für Informatik

ESS AG Embedded System Software

PG 522 : AUTOLAB

REALISIERUNG



Mit freundlicher Unterstützung von Audi KOSTAL EB infineon

Betreuer der PG 522 Autolab: Prof. Dr.-Ing. Olaf Spinczyk, Horst Schirmeier, Jochen Streicher
Teilnehmer: Alexander Berger, Oliver Botschkowski, Stephan Braun, Sabrina Hecke, Florian Hohnsbehn, Christian Horn, Gregor Kaleta, Boris Konrad, Sebastian Kosch, Matthias Meier, Robert Neue und Thomas Romanek

Abbildung 9.2: Plakat 2 für das Campusfest

Kapitel 10

Endprodukt

10.1 Überblick

Nach der Fertigstellung des zweiten Prototyps und der Präsentation auf dem Campusfests blieben noch etwa sechs Wochen, um die dritte Entwicklungsiteration durchzuführen und ein Endprodukt zu entwerfen, spezifizieren und zu bauen.

Dieses basiert natürlich stark auf dem zweiten Prototypen, allerdings wird der durch die langfristig geplante Vorführung bedingte Fokus auf dem Anwendungsfall Demonstration abgeschwächt. Stattdessen wurde die bisher nicht vollumfänglich integrierte Messumgebung gestärkt. Die Gliederung in Fahrzeugnetz, Steuer- und Messumgebung sowie dem PC als Benutzerschnittstelle wurde beibehalten, allerdings unter klarerer Trennung der einzelnen Ebenen. Da nun als zusätzliche Hardware weitere TriCore-Boards verfügbar waren, wurde im Endprodukt auch die Zuordnung je eines Mess- und Steuerboards pro Aktor umgesetzt. Die weitergeführte Arbeit am Dachelement ermöglicht die Einbindung dieses zusätzlichen Elements. Da dieses über einen Low-Speed-CAN sendet, unser bisheriger Bus aber auf High-Speed-CAN basierte, musste ein TC 1796-Board als Gateway programmiert und eingesetzt werden, an welches das Dachelement über eine dritte CAN-Verbindung in das Gesamtsystem integriert werden kann. Die Kommunikation der einzelnen Aktoren erfolgt nach wie vor über den schon im ersten Prototyp eingesetzten CAN1-Bus. Die Kommunikation der Steuer- und Messumgebung mit der Benutzerschnittstelle erfolgt ebenso wie zuvor mit dem schon bewährten CAN2-Bus.

Des weiteren konnte die Steuer- und Messumgebung weiter verfeinert, getestet und in ProOSEK eingebettet werden. Statt nur im RAM zu laufen, wurde die Software so weiterentwickelt, dass sie im Flash-Speicher der Boards angesiedelt werden kann und diese so dauerhaft in Betrieb genommen werden können. Die Ausgabe der Messwerte wurde genauer spezifiziert. Die so mit CANoe aufgezeichneten Werte werden mit der Logging-Funktion in eine Log-Datei geschrieben. Hierzu wurde ein Konverter-Tool programmiert, welches diese Log-Files einlesen, die Messwerte darin lokalisieren und ihre Zeitstempel gemäß der Zeitskala auf der Benutzerschnittstelle (also die Zeit von CANoe) anpassen kann. Die Ausgabe ist ein konvertiertes Logfile, welches wiederum in CANoe eingelesen kann, so dass die Messwerte so zeitgenau dargestellt werden.

Der resultierende Aufbau ist in der folgenden grafischen Übersicht zusammengefasst (siehe Abbildung 10.1).

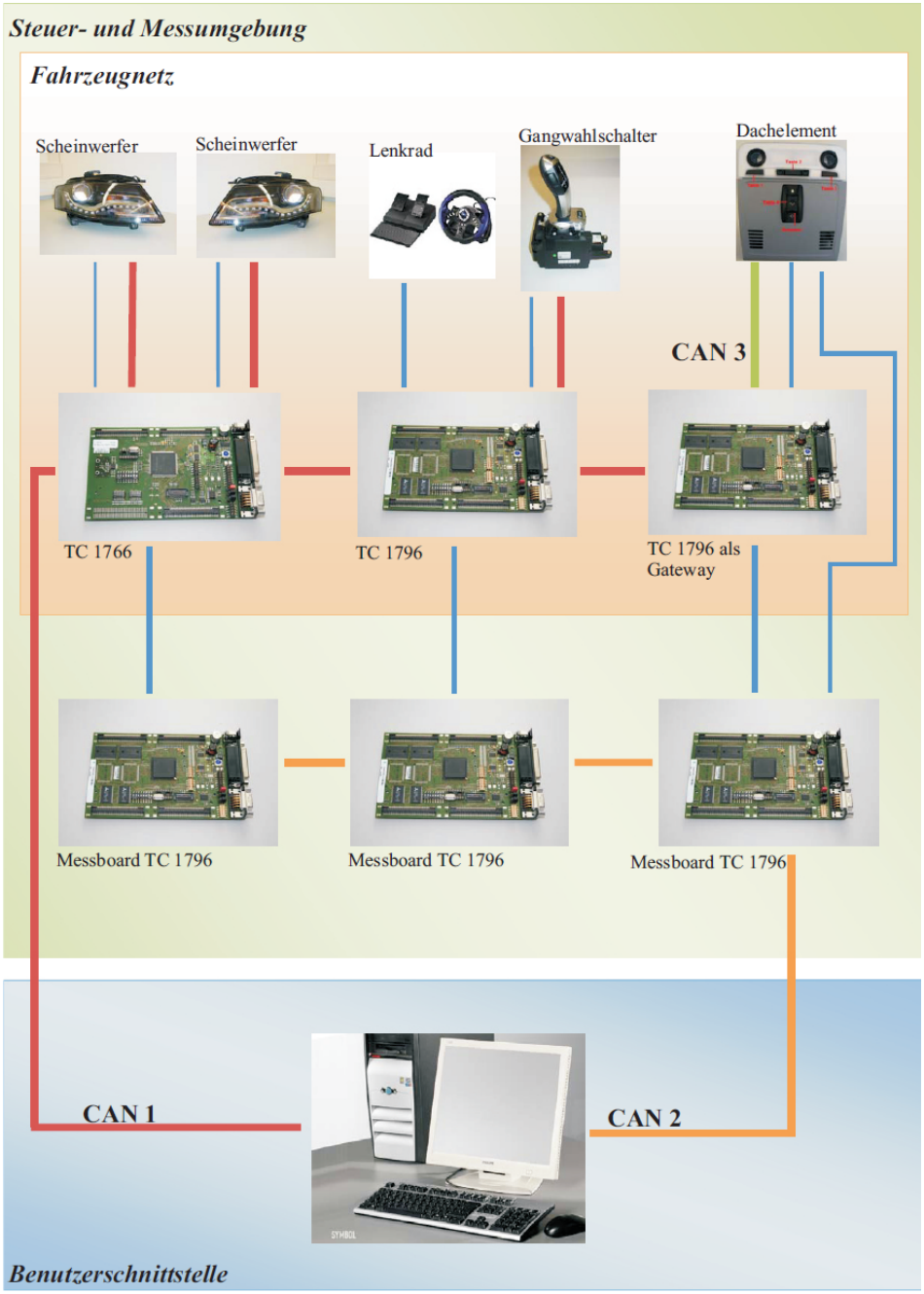


Abbildung 10.1: Konzeption des Endprodukts

10.2 Einzelne Arbeiten

10.2.1 Aktoranbindung

Gatewayknoten für den Betrieb des Dachelements

Das Dachelement besitzt lediglich einen Low-Speed CAN-Tansceiver mit einer maximalen Übertragungsrate von 125 Kbps. Alle anderen CAN-Teilnehmer verfügen über einen High-Speed CAN-Transceiver und werden mit einer Übertragungsrate von 500 Kbps betrieben.

Im Endprodukt wird sowohl ein High-Speed- als auch ein Low-Speed-CAN verwendet. Um eine Kommunikation zwischen den beiden Bussen zu ermöglichen, wurde ein Gatewayknoten auf Basis des Softec/Freescale DEMO9S12XEP100 Boards erstellt. Der Freescale 9S12XEP100-Mikrocontroller verfügt über vier MSCAN Module, so dass die bereits implementierten CAN-Treiber des Gangwahlschalters und Dachelements weitgehend übernommen werden konnten. Die ersten beiden CAN-Module verfügen bereits über einen onboard High-Speed CAN-Transceiver, den Philips/NXP TJA1041. Die übrigen zwei CAN-Module sind für den Anschluss von externen CAN-Transceivern vorgesehen. So wurde an das dritte MSCAN-Modul der Low-Speed CAN-Tranceiver TJA1054 angeschlossen.

Die Software ist sehr einfach gehalten. Sie beobachtet beide CAN-Busse und leitet die entsprechenden Pakete, die den Filter passieren, auf den jeweils anderen CAN-Bus weiter.

Statt des DEMO9S12XEP100 Boards von Freescale war ein TC 1796 als Gatewaykonten vorgesehen. Auf Grund von Zeitmangel und der Einfachheit der CAN-Treiber für den Freescale Mikrocontroller haben wir uns kurzfristig für das DEMO9S12XEP100 Board entschieden. Wie Sie im Kapitel 10.3.1 nachlesen können, ist das Vorgehen beim Einsatz des TC 1796 analog.

10.2.2 Software

Verteilte Fahrzeugfunktionen mit OSEK

Von den Funktionalitäten her unterscheidet sich die endgültige Softwarewareversion im Endprodukt zur Steuerung der Fahrzeugfunktionen kaum von der des zweiten Prototypen. Es wurden lediglich einige feine Veränderungen durchgeführt und der Sportknopf des Gangwahlschalters eingebunden. Letzter schaltet den linken Blinker und das Fernlicht bei Betätigung ein.

Die große Veränderung liegt darin, dass wir die Software gesplittet haben, damit jedes Gerät sein eigenes Steuergerät erhält. Das Lenkrad wurde mittels einer Erweiterungsplatine an ein TC1796 angeschlossen. Die Scheinwerfer werden weiterhin vom TC1766 gesteuert. Eine Kommunikation mit dem Dachelement ist optional über ein weiteres TC1796 als Gateway möglich.

Da wir nun verteilt auf mehreren Boards arbeiten, muss auf jedem einzelnen Board ein OSEK laufen. Somit benötigt jedes Board einen ReceiveTask, der wie schon in der Vorgängerversion

auf dem Bus lauscht und das Verhalten maßgeblich kontrolliert. Die Nachrichten vom Lenkrad müssen nun als CAN-Nachrichten verpackt und verschickt werden. Somit ergeben sich leichte Umstrukturierungen im Code.

Das System ist somit deutlich variabler und mit wenig Overhead an die jeweilige Situation beziehungsweise den jeweiligen Aufbau anpassbar.

Mess- & Steuerumgebung mit ProOSEK

Im Endprodukt sollte die Software auf den TriBoards vollständig mit ProOSEK realisiert werden. Bis auf die Software für die Mess- und Steuerumgebung war dieses Ziel bereits mit dem zweiten Prototypen erreicht worden.

Die Funktion der Software wurde bereits im Kapitel 8.2.3 erläutert. An dieser Stelle soll nur kurz auf die Vorteile eingegangen werden, die sich durch den Einsatz von ProOSEK ergeben.

Die Messung von Stromaufnahme und digitalen Pegeln sollte zeitlich hochgenau möglich sein. Mit dem bisherigen Ansatz, den Wert des Systemtimers abzufragen, und in Abhängigkeit davon bestimmte Funktionen auszuführen, konnte man keine besonders hohe Genauigkeit erreichen. ProOSEK bietet die Möglichkeit, Counter/Timer zu erstellen, die mit einer festgelegten Periode zählen. Zudem sind Alarme möglich, die auf diese Counter/Timer bei erreichen eines bestimmten Counter-/Timer-Wertes reagieren und beispielsweise Tasks aufrufen, Events auslösen oder sogenannte Alarm-Callback-Routinen aufrufen. In unserem Fall schien uns die Verwendung der Callback-Routinen am sinnvollsten. So wird beim Auslösen eines Alarms in der entsprechenden Routine eine Alarm-Variable gesetzt. Wie beziehungsweise wo diese Variable ausgewertet wird, wird in der folgenden Übersicht über die Software beschrieben.

Während des Programmablaufes wird von einem Haupt-Task ständig auf eingehende CAN-Nachrichten (auf ID 400h) gehorcht – die Steuernachricht für die Mess- und Steuersoftware. Dieser Haupt-Task erzeugt regelmäßig Events, die das eigentliche Betriebssystem ProOSEK dazu auffordert einen Taskwechsel durchzuführen.

Folgende Tasks sind vorhanden:

- Ein Haupt-Task, der die CAN-Steuer-Nachrichten empfängt.
- Ein Task zur Steuerung der Versorgungsspannung (ein/aus) und des Resetsignals.
- Ein Task zur Messung digitaler Pegel.
- Ein Task zur Messung analoger Signale (Stromaufnahme).
- Ein Task zum Senden aufgezeichneter analoger Messwerte.

Der **Task zur Steuerung der Versorgungsspannung und des Resetsignals** wird dabei unabhängig von evtl. eingegangenen Nachrichten aufgerufen. Dieser Task setzt entsprechend der Steuernachricht die I/O-Pins.

Der **Task zur Messung digitaler Pegel** wird nur ausgerufen, wenn zwei Bedingungen erfüllt sind:

1. die Steuernachricht signalisiert das Einschalten des Boards
2. eine Alarm-Variable besitzt einen bestimmten Wert

Eine getrennte Aktivierung zur Messung digitaler Pegel ist damit nicht notwendig.

Punkt 2 erfordert eine kurze Beschreibung, da hier nun auch der o.g. Alarm wichtig ist: Wurde ein Alarm ausgelöst, so setzt die entsprechende Callback-Routine eine Alarm-Variable. Nur wenn zusätzlich zum gesetzten Bit in der Steuernachricht auch diese Variable einen bestimmten Wert hat, wird der Task aufgerufen. Der **Task zur Messung analoger Signale** wird nur in Abhängigkeit einer Alarm-Variable aufgerufen. Die Überprüfung, ob eine Messung tatsächlich durchzuführen ist, geschieht im Task selbst. Dieses Vorgehen ist notwendig, um auch Aktionen, die beim Abschalten einer Messung durchgeführt werden, im Code integrieren zu können. Der **Task zum Senden aufgezeichneter Werte** wird wieder aufgerufen, wenn durch die Steuernachricht ein Senden der Signale angefordert wird und eine Alarm-Variable gesetzt ist.

Mit Hilfe der Alarme ist es möglich, bestimmte Aktionen in genau festgelegten zeitlichen Abständen auszuführen. Zum einen wird dadurch sichergestellt, dass die Messungen in einem exakten zeitlichen Raster durchgeführt werden, zum anderen kann die Last auf dem CAN-Bus konstant gehalten werden, da die erzeugte Buslast vom Intervall des benutzten Alarms abhängt (das Senden von Nachrichten erfolgt ebenfalls nur bei gesetzter Alarm-Variable).

Visualisierung von Messdaten

Von den Messboards erfasste Messwerte werden nur selektiv auf den Bus übertragen. Um alle aufgezeichneten Messewerte zu erhalten und in CANoe zu visualisieren bedarf es eines Umwegs. Mittels eines Logging-Knotens im Messaufbau werden Nachrichten auf dem Bus aufgezeichnet. Werden alle Messwerte bei dem Board angefragt, so sendet dieses die komplette Aufzeichnung in einzelnen Nachrichten, wobei jede Nachricht mit einem Zeitstempel versehen ist. Das Logfile mit diesen Nachrichten wird mit einem Konverter erfasst. Dieser liest die Zeitstempel aus den Nachrichten aus und rechnet ihn in die Zeitskala des Logs um. Als Ausgabe wird ein neues Logfile erstellt, bei dem die Messnachrichten an der richtigen Stelle einsortiert worden sind. Dieses kann dann wiederum mittels der Replay- oder Offlinefunktion von CANoe eingelesen und visualisiert werden.

Die Messung wird gestartet oder gestoppt wenn das Messboard eine entsprechende Nachricht über den CAN-Bus erhält. Sobald die Messung gestoppt wird, sendet das Messboard eine Nachricht mit der aktuellen Systemzeit des Boards. Um die interne Uhr des Boards mit der CANoe-Zeit zu synchronisieren, wird die Ankunftszeit dieser Nachricht in CANoe mit dem Zeitpunkt des Versendens des Stoppsignals verglichen. Hierbei gibt es eine geringe Zeitdifferenz, welche in der Laufzeit auf dem Bus begründet ist. Standardmäßig wird nun die Hälfte dieser Differenz, also die Zeit die eine Nachricht über dem Bus ungefähr gebraucht hat, von den umgerechneten Zeitstempeln abgezogen. Dadurch werden diese Messwerte zum möglichst exakten Zeitpunkt der korrespondierende Ereignisse eingegliedert. Da die Buslatenz nicht konstant ist, kann die angenommene

Differenz ungenau sein. Der Konverter kann daher einen Offset als Parameter bekommen um dies auszugleichen. Erkennen kann man eine Ungenauigkeit etwa durch ein Referenzsignal. Wenn dessen umgerechneter Zeitstempel nicht mit dem Absendezeitpunkt deckungsgleich ist, gab es einen Fehler. In diesem Fall wird der Konverer mit dem gleichen Ursprungslogfile erneut aufgerufen, bekommt aber mittels des Flags „-o“ einen Offset mit, der diesen Fehler ausgleicht. Die Eingabe erfolgt dabei als Double in Sekunden, für einen Offset von 0,23ms also beispielsweise mit „-o 0.00023“. Wichtig ist hierbei noch, dass der Umrechnungsfaktor der Boardzeit beachtet wird. Eine Boardzeiteinheit kann je nach Einstellung der für uns relevanten Modi entweder 0.8 Milli- oder 1 Mikrosekunde entsprechen. Standardmäßig wird von 0.8 Millisekunden ausgegangen, ein anderer Umrechnungsfaktor kann über den Paramter „-f“ übergeben weden. Ebenfalls in der Software des Messboards variieren kann die für Nachrichten mit Messwerten verwendete ID. Hierfür gibt es das Flag „-id“. In der Ausgabedatei bekommen die konvertierten Nachrichten die ID 702.

Auch die Angabe von Quell- und Zieldatei erfolgt über Parameter. Hierbei ist „-s“ das Flag für Source (Quelldatei) und „-t“ das Flag für Target (Zieldatei). Die übliche Dateiendung von CANoe-Logfiles ist „.asc“, diese enthalten aber nur Text (ASCII).

10.2.3 Hardware

System Basis Chip des Dachelements

Der System Basis Chip ist eines der wichtigsten Bauteile des Dachelementes, denn dieser versetzt den Mikrocontroller in einen Wach-/Schlafzustand. Gleichzeitig werden über diesen CAN-Nachrichten versendet beziehungsweise empfangen.

Der System Basis Chip kann über SPI-Nachrichten konfiguriert werden. Bevor der SBC allerdings geflasht werden kann, muss er in einen Wachzustand versetzt werden. Dies erreicht man unter anderem, indem die Resetleitung nahe des Mikrocontrollers aufgetrennt wird. Zusätzlich ist es notwendig, eine Flanke oder eine beliebige Nachricht über den CAN-Bus zu senden.

Alternativ kann der System Basis Chip auch softwareseitig aufgeweckt werden. Dazu muss eine spezielle SPI-Nachricht vom Mikrocontroller aus versendet werden:

Programm Enable (Tabelle 10.1)

Bit	7	6	5	4	3	2	1	0
Bedeutung	CG1	CG0	PGEN	ADJ4	ADJ3	ADJ2	ADJ1	ADJ40

Tabelle 10.1: RCADJ RC-Oscillator Adjust Register

Diese Nachricht aktiviert durch ein entsprechendes Setzen des ”RCADJ RC-Oscillator Adjust Registers” den ”Programm Enable” Mode.

Defaultmässig müssen drei weitere SPI-Nachrichten verschickt werden, die der Konfiguration des System Basis Chips dienen. Dies gilt auch für den Fall, dass der System Basis Chip hardwareseitig

aufgeweckt worden ist.

1. **CRC-Prüfung deaktivieren (Tabelle 10.2):**

Bit	7	6	5	4	3	2	1	0
Bedeutung	NCRC	STAT	WNDF	STF	OTF	UCF	WAKE	NPOR

Tabelle 10.2: SYS System Status Register

2. **Disable Window Watchdog (Tabelle 10.3):**

Bit	7	6	5	4	3	2	1	0
Bedeutung	WDEN	WND	SWT1	SWT0	WDT3	WDT2	WDT1	WDT0

Tabelle 10.3: WDC Watchdog Control Register

3. **CAN in Operation Mode (Tabelle 10.4):**

Bit	7	6	5	4	3	2	1	0
Bedeutung	ACT	TXEN	RES	RES	RES	LP2	LP1	LP20

Tabelle 10.4: CTCR CAN-Tranceiver Control Register

Wichtig ist, dass alle SPI-Nachrichten gleich nach der *PE_low_level_init* Methode initialisiert und gesendet werden, weil so sichergestellt wird, dass das Dachelement korrekt konfiguriert wird.

Möchte man das Dachelement in Betrieb nehmen, muss vorher die Resetleitung wieder geschlossen werden. Erst dann können die CAN-Nachrichten vernünftig versendet werden.

Funktionalität des Dachelements

Da die Funktionalität des Dachelements bereits im Kapitel Grundlagen beschrieben worden ist, soll an dieser Stelle nur erwähnt werden, welche dieser Funktionalitäten im Endprodukt programmiertechnisch umgesetzt werden konnten.

Sowohl das Ambientelicht als auch die Leselampen können nun über die entsprechenden Tasten ein- und ausgeschaltet werden. Zusätzlich besitzt das Dachelement, genau wie der Gangwahlschalter, eine Nachtbeleuchtung. Es empfängt vom TC 1766 CAN-Botschaften mit der ID 501, welche Informationen über den aktuellen Lichtstatus enthalten, und reagiert entsprechend darauf, indem die Nachtbeleuchtung ein- oder ausgeschaltet wird. Das Dachelement liest über den CAN-Bus den Status des Gangwahlschalters (ID 200). Wird der Parkmodus betreten, schaltet das Dachelement das Ambientelicht an. Beim Verlassen des Parkmodus oder einen Knopfdruck wird das Ambientelicht ausgeschaltet.

10.2.4 CAN-Bus-Treiber

CAN-Treiber für das Dachelement

Das Dachelement verfügt über das gleiche High-Speed MSCAN12-Modul wie der Gangwahlschalter. Da die Implementierungen sich bis auf die Angabe eines Baudratenvorteilers von „5“ und der unterschiedlichen IDs absolut gleichen, wird hierauf nicht weiter eingegangen.

Zu beachten ist jedoch, dass der System Basis Chip des Dachelements nur über einen Low-Speed CAN-Transceiver verfügt. Aus diesem Grund wurde das Timing auf eine Baudrate von 125 Kbps angepasst.

10.3 Aufgetretene Probleme und deren Lösungen

10.3.1 CAN-Bus des Dachelements

Bei der Kommunikation des Dachelements mit CANoe und dem Gateway (TC 1796) kommt es zu Schwierigkeiten. Der Grund liegt in den Unterscheiden zwischen dem CAN-Low-Speed-Physical-Layer (ISO 11898-3) und dem CAN-High-Speed-Physical-Layer (ISO 11898-2). Als zwei grundlegende Unterschiede wären zu nennen:

1. Die Spannungsdifferenz zwischen der CAN-High-Leitung und der CAN-Low-Leitung beträgt bei Low-Speed CAN 5 Volt, bei High-Speed CAN aber lediglich 2 Volt.
2. Bei High-Speed CAN wird zur Terminierung an beiden Enden des Busses ein Widerstand von 120 Ohm zwischen die CAN-High-Leitung und die CAN-Low-Leitung geschaltet. Bei Low-Speed CAN findet die Terminierung jeweils zwischen CAN-Low und RTL (CAN-Low-Termination-Source) sowie zwischen CAN-High und RTH (CAN-High-Termination-Source) des jeweiligen Knotens statt.

Sowohl die PCI-CANoe-Karte als auch die Triboards verfügen nur über High-Speed CAN-Transceiver. Somit ist eine direkte und fehlerfreie Kommunikation mit dem Dachelement nicht möglich. Zur Lösung dieses Problems könnte ein CAN-Konverter eingesetzt werden. Dieser enthält einen Low-Speed als auch einen High-Speed CAN-Transceiver. Eine andere Lösung stellt das Triboard TC 1796 selbst zur Verfügung. Es enthält ein CAN-Modul, das an einen externen CAN-Transceiver angeschlossen werden kann. Wählt man an dieser Stelle einen Low-Speed CAN-Transceiver, so ist eine fehlerfreie Kommunikation mit dem Dachelement gewährleistet. Aus Zeitgründen und der Einfachheit des CAN-Treibers für den Freescale Mikrocontroller haben wir uns kurzfristig für das DEMO9S12XEP100 Board entschieden. Näheres ist im Kapitel 10.2.1 nachzulesen.

Für die Kommunikation der Dachmodule über einen standardkonformen Low-Speed CAN-Bus müssen zwei Maßnahmen getroffen werden:

1. Zum Flashen musste die Reset-Leitung zum Mikrocontroller aufgetrennt werden. Sobald kein Programmiergerät an der BDM-Schnittstelle angeschlossen war, wurde die Reset-Leitung nicht auf +5 V gehalten und der Mikrocontroller wurde durchgehend resettet. Damit das Dachelement auch ohne Programmiergerät betreiben werden kann, muss der BDM-Stecker des Dachelements mit einem selbstgelöteten Adapter versehen werden. Dieser verbindet die Reset-Leitung über einen 4,7 kOhm Widerstand mit VDD.
2. Das Dachelement wird serienmäßig mit einem relativ hochohmig abgeschlossenen CAN-Transceiver ausgeliefert. Das verursacht jedoch in kleinen Bussen Schwierigkeiten. Um eine fehlerfreie CAN-Kommunikation zu gewährleisten, ist es notwendig, die Terminierung des System Basis Chips zu verändern. Dazu wurden die entsprechenden hochohmigen Widerstände mit niederohmigen Widerständen parallel geschaltet.

Als Alternative könnte ein Dummysteuergerät zusätzlich an den CAN-Bus angeschlossen werden. Dieses Steuergerät muss niederohmig terminiert sein und der Mikrocontroller im Reset gehalten werden. Wir haben ein Dachelement zu einem Dummysteuergerät umfunktioniert, indem wir die entsprechenden hochohmigen Widerstände mit niederohmigen Widerständen parallel geschaltet und die Reset-Leitung auf Masse gezogen haben.

Es ist uns gelungen unter bestimmten Voraussetzungen dem Dachelement zu ermöglichen CAN-Botschaften, die vom High-Speed CAN-Transceiver des Triboards versendet werden, zu empfangen.

1. Wie schon erwähnt, muss die Terminierung des System Basis Chips verändert werden.
2. Die CAN-Leitung zum Gateway muss relativ kurz sein. Am besten man schließt den CAN-Stecker des Dachelements direkt an das Triboard an.
3. Die BDM-Schnittstelle muss mit dem Programmiergerät verbunden sein.

Dem Dachelement ist es auf diese Weise weiterhin nicht möglich, selbst fehlerfrei CAN-Botschaften zu versenden. Diese Konfiguration wird nicht mehr angewendet, da auf Basis des DEMO9S12XEP100 Boards ein vollfunktionsfähiger Gatewayknoten zur Verfügung steht.

10.3.2 Messsoftware in OSEK

Als wir angefangen haben, die Messsoftware zu schreiben, haben wir uns nur wenige Gedanken über die Ausführung des Codes in Osek gemacht. Das hat sich im weiteren Verlauf als Nachteil herausgestellt. Bei den ersten Tests nach der Einbettung der funktionsfähigen Software in OSEK lief die Software immer in die Trap 1. Da die Software ohne OSEK jedoch fehlerfrei funktionierte, haben wir den Fehler nicht sofort erkannt. Um das Problem zu lösen, ließen wir das Programm schrittweise ablaufen (Breakpoint gesetzt und mit “step” einzelne Anweisungen ausgeführt), um uns den Problem zu nähern.

Wie sich herausgestellt hat, lag das Problem darin, dass der Ringbuffer, den wir zum Abspeichern der Messwerte angelegt haben, Speicher überschrieben hat, der von OSEK benutzt wird. Der

Ringbuffer hat sich ursprünglich an der Position „0xa1000000“ befunden. Um die Sigtrap zu beheben haben wir die Variable _rbvar im Linkerscript „ramresident-gnu.ldscript“ hinzugefügt. Diese Variable befindet sich nun direkt hinter dem OSEK Code. Die Speicheradresse der Variable _rbvar dient nun dazu, die Startadresse für den Ringbuffer festzulegen.

Nach diesen Modifizierungen läuft die Messsoftware in OSEK ohne weitere Probleme.

10.3.3 Flashen der Triboards

Unter anderem gehörte das Flashen der Triboards zu den Minimalzielen der PG. Einerseits sollte es möglich sein, seinen Code permanent auf dem Board abzulegen und somit unabhängig von der Rechnerumgebung sein zu können, andererseits stellt dies auch eine Grundlage für die Remotearbeit dar.

Unsere Software, die sich im RAM problemlos ausführen ließ, war jedoch nicht ohne weiteres im Flash zu verankern. Zum einen mussten die Linkerscripte entsprechend angepasst werden; auf der anderen Seite benötigten wir eine Möglichkeit, die Daten in den Flash zu schreiben.

Nach einer rudimentären Anpassung der Linkerscripte haben wir versucht, entsprechend der FAQ das kompilierte .elf-File in ein Binary umzuwandeln. Durch unnötige Abstände der Sektionen wurden die Binärdateien jedoch immer mehr als 800 MB groß und passten nicht auf den Speicher der Triboards. Auch ein direktes Flashen der erzeugten .elf-Files, mittels Flashload, scheiterte.

Wir haben anschließend mit einem Loader, den wir von der Universität Erlangen bekommen haben, gearbeitet. Mittels tricore-insight haben wir die Software in den RAM kopiert und von dort aus auf die .elf-Files zugegriffen, die wir in den Flash schreiben wollten. Hierbei ist insbesondere zu erwähnen, dass das Programm keine Pfadexpansion beinhaltet. Die Pfade müssen also komplett von Hand eingegeben werden, damit das entsprechende .elf-File in den Flash geschrieben werden kann.

Nach etlichem Herumprobieren haben wir nun auch eine Möglichkeit gefunden, die Daten permanent im Flash abzulegen. Auf Grund mehrerer Probleme ist es uns allerdings noch nicht gelungen, die Programme dort auch auszuführen. Es gab verschiedene Komplikationen, wie zum Beispiel Zugriffe auf geschützte Speicherbereiche. Das Problem der Ausführung besteht weiterhin.

10.4 Stand der Realisierung

Das Ziel der Pojektgruppe war es, einen Brettaufbau als Experimentierplattform für automobile Softwareentwicklung zu schaffen. Dieses Ziel wurde erreicht, wobei die Ausprägung im Folgenden zusammenfassend dargestellt wird.

Unser Endprodukt integriert die Komponenten Scheinwerfer, Gangwahlschalter, Dachelement, Gaming-Lenkrad (siehe 10.1). Diese Komponenten sind komplett integriert, und es werden nahezu sämtliche Funktionalitäten, die von den Geräten geboten werden, genutzt. Hierbei ist zu bemerken, dass teilweise Komponenten redundant vorhanden sind.

- Vom Dachelement sind drei Exemplare vorhanden, die alle vollständig integriert sind. Die einzelnen Elemente unterscheiden sich dabei nur in Kleinigkeiten voneinander.
- Der Gangwahlschalter ist in doppelter Ausführung vorhanden, wobei auch hier beide vollständig integriert worden sind. Einer der beiden hat als zusätzliche Ausstattung einen Sportknopf, sowie Spulen, die eine Blockade-Möglichkeit schaffen. Dadurch kann man beispielsweise den Wechsel in die Tiptronic aus einem anderen Modus als den Drive-Modus verhindern.
- Die Scheinwerfer sind ein Paar von Scheinwerfern mit einer linken und einer rechten Version. Beide empfangen und versenden unterschiedliche Nachrichten-IDs.

Um eine Kommunikation zum Dachelement zu ermöglichen, welches einen Low-Speed-CAN benötigt, ist ein Gatewayknoten eingebunden. Dieser ist bislang durch Software auf einem TC1796-Board realisiert und nutzt nicht die hardware-unterstützte Gateway-Funktion des Boards. Dies kann in der Zukunft verbessert werden.

Eine Komponente, die zwar vollständig integriert ist, aber im Endprodukt nicht auftaucht, ist der Sensorcluster. Diesen haben wir bereits im ersten und zweiten Prototypen genutzt, um das Kurvenlicht zu realisieren. Im Endprodukt übernimmt dessen Steuerung das Lenkrad, so dass der Sensorcluster nicht aufgenommen wurde. Er bleibt aber weiterhin funktionsfähig. Ein weiteres Vorgehen im Bezug auf den Sensorcluster könnte sein, dessen gesendete Werte besser umzurechnen, sodass genauere Aussagen über die Position machbar sind.

Eine weitere Komponenten im Endprodukt ist das PC-System. Auf diesem betreiben wir CANoe, um Restbussimulationen durchzuführen, Nachrichten zu visualisieren und ähnliches (Vgl. 5.3.1). Somit wird auch die Steuer- und Messumgebung durch CANoe realisiert. Die einzelnen Messboards, welche den Steuergeräten und Steuerboards vorgeschaltet sind, sind über einen separaten CAN-Bus mit dem PC verbunden. Die komplette Steuerung erfolgt über CAN-Nachrichten und auch die aufgezeichneten Messdaten werden auf diesem Wege übertragen. Es gibt zwei Möglichkeiten der Messung: Zum einen werden Messdaten live während der Messung an das PC-System übertragen, und zum anderen gibt es die Möglichkeit, dass die Boards Messdaten aufzeichnen und dann auf Anfrage übertragen. So kann eine höhere Auflösung als während der Live-Übertragung ermöglicht werden. Ein Problem hierbei ist unter Umständen die internen Zeiten der Messboards mit der Zeit von CANoe zu realisieren. Unsere Lösung hierfür versucht, die Uhren bestmöglich zu synchronisieren, wobei manuell nachstellt werden kann, falls man anhand eines Referenzsignals Fehler erkennt (Vergleiche 10.2.2). Ob sich diese Methode in der Praxis bewährt muss sich in der Zukunft zeigen. Verbesserungspotential birgt hier eine Taktleitung.

Neben diesem PC-System gibt es ein weiteres in Form eines Laptops. Um TORCS als Fahrsimulation zu nutzen, muss der Laptop in den Aufbau integriert werden (Vergleiche 8.2.3).

Komponenten, die von uns nicht integriert wurden, sind die folgenden:

- Lenksäulenmodul: Wegen der Komplexität des Moduls konnten wir es im Rahmen dieser Projektgruppe nicht integrieren. Auch haben wir es kaum erforschen können. Unsere Erkentnisse sind lediglich die in 5.1.7 genannten. Auf lange Sicht wäre es schön, wenn das Lenksäulenmodul, plus zuätzliche Konstruktionen, das Gaming-Lenkrad ablösen würde.
- Fahrzeugtür: Die uns zur Verfügung stehende Fahrzeugtür mit dem Steuergerät für die

elektrischen Fensterheber, konnten wir aus Zeitgründen noch nicht integrieren. Sie stellt eine interessante mögliche Erweiterung unseres Brettaufbaus dar.

- Dachelement: Passend zu unserem Dochmodul haben wir kurz vor Ende der Projektgruppe ein Dachelement erhalten, das durch das Dachmodul gesteuert werden kann. Da wir uns nicht mehr mit dem Dachelement befassen konnten, ist auch dies eine Erweiterung die noch durchgeführt werden kann.

Nicht vollständig gelöst ist das Flash-Problem. Mittlerweile ist es zwar möglich, die TriCore-Boards zu flashen, jedoch ist die Anwendung nicht ausführbar (Vgl. 10.3.3).

Neben den nicht integrierten Komponenten birgt auch die Funktionalität der Fernsteuerung Potential zur weiteren Bearbeitung. Das Flashen der Steuergeräte soll beispielsweise auch aus der Ferne möglich sein. Da ein großer Teil der Steuerung von dem PC-System aus möglich ist, muss eine Möglichkeit geschaffen werden, das PC-System aus der Ferne zu steuern. Um auch ein komplettes Ein- und Ausschalten der Mess- und Steuerboards zu ermöglichen, haben wir bereits eine IP-Steckdose eingeplant. Diese Anschaffung wurde jedoch noch nicht getätigt. Auch ist der Messaufbau noch nicht beobachtbar, was etwa durch eine Webcam im Labor handhabbar wäre.

Um dem Ziel eines Brettaufbaus nahe zu kommen, gibt es Gehäuse für alle Boards. Der Fortschritt der Integration der Hardware-Komponenten und das fehlende Wissen darüber, welche Hardware-Komponenten in naher Zukunft hinzu kommen, haben es uns noch nicht erlaubt einen echten Brettaufbau, auf einer Holz- oder Metallkonstruktion, zu planen.

Kapitel 11

Abgleich der Anforderungen mit dem Endprodukt

Da wir das beschriebene iterative Vorgehen gewählt haben, gab es durch getroffene Entwurfsentscheidungen für die einzelnen Prototypen auch Änderungen bei der erneuten Anforderungserfassung. Hierbei wurde darauf verzichtet, jeweils ein komplett neues Anforderungsdokument zu erstellen, stattdessen wurden von den neuen Entwürfen nicht mehr erfüllte Anforderungen erhoben und in der Gruppe besprochen. Daraus resultierend gab es die folgenden Änderungen an den ursprünglichen Anforderungen.

11.0.1 Priorität 1

- Die Software CANoe dient bei uns sowohl als Benutzerschnittstelle zum CAN-Bus des Fahrzeugnetzes als auch zur Steuer- und Messumgebung. Dadurch war es gleich möglich, die Funktionen einer konsolenbasierten Schnittstelle zu nutzen und dabei mit dem grafischen Benutzerinterface zu arbeiten. Deshalb wurde die Anforderung, eine konsolenbasierte Benutzerschnittstelle zu entwickeln (SB10), verworfen. Sie hätte keinen Zugewinn mehr dargestellt und hat auf der ursprünglichen Annahme beruht, eine Benutzerschnittstelle selbst zu programmieren und darauf aufbauend ein grafisches Interface laufen lassen zu wollen.
- Die galvanische Trennung (MK02) von Messumgebung und Versuchsaufbau basierte auf Vorüberlegungen zur Messung, die später zu Gunsten der getroffenen Entwurfsentscheidungen (siehe Kapitel 7.2.3) verworfen wurden. Dadurch wurden auch die Überlegungen zur galvanischen Trennung irrelevant und die Anforderung gestrichen.
- Die Kapselung in unabhängige Teilsysteme (SA01) wurde durch verschiedene Aktoren realisiert. Allerdings besitzen wir z.B. nur ein Netzgerät und nur einen Windows-PC mit CAN-Bus-Anschluss und CANoe, so dass momentan ein vollständig getrennter paralleler Betrieb nicht möglich ist. Der Aufbau erfüllt aber die Anforderung, es müsste nur entsprechend zusätzliche Hardware bereit gestellt werden.

11.0.2 Priorität 2

- Vorab wurde der Anwendungsfall Remote-Nutzung diskutiert. Darauf aufbauend wurden einige Anforderungen erfasst, die hierfür notwendig sind (z.B. FS06, SB06, SB07). Da dieser Anwendungsfall selbst weniger wichtig war, wurde angesichts der verbliebenen Zeit nach Fertigstellung des zweiten Prototypen beschlossen, diese Funktionalität auch im Endprodukt nicht zu realisieren, sondern nur vorzubereiten (etwa durch Analyse geeigneter IP-Steckdosen). Damit entfielen auch Anforderungen an das System, die sich auf diese externe Zugriffsmöglichkeiten ergeben hatten.
- Eingeschränkt gilt dies auch für die Anforderung (SB07) die sich auf Zugriffseinschränkungen bezieht. Hier wurden im System keine besonderen Maßnahmen eingebaut um dies zu erfüllen, die Anforderungen allerdings indirekt dadurch abgedeckt, dass es Teilsysteme gibt, die auch durch fehlerhafte Bedienung keinen Schaden nehmen und die daher z.B. auf Vorführungen eingesetzt werden können.

11.0.3 Minimalziele

Die in Kapitel 1.3.2 genannten Minimalziele waren bereits in die Anforderungserfassung mit eingegangen und werden durch das Endprodukt erfüllt:

- Erstellung eines Konzeptes - Im ersten Semester wurde Fahrzeugnetz und Steuer-/Messumgebung konzeptioniert und geplant. Für die weiteren Prototypen wurde, wo nötig, das Konzept ergänzt. Siehe Kapitel 4 und 6.
- Inbetriebnahme eines Versuchsaufbaus - Insgesamt haben wir drei Protoypen erstellt, in denen die vorhandenen Steuergeräte (siehe Kapitel 5.1) mit einander vernetzt und in Betrieb genommen wurden. Siehe Kapitel 7,8 und 10. Die dabei verwendeten Entwicklungswerkzeuge und Standardtools sind in Kapitel 5.3 beschrieben.
- Steuerungs- und Messumgebung: Die entworfene Umgebung soll in Hard- und Software implementiert werden. - Auch dies wurde von uns in den drei Prototypen schrittweise umgesetzt. Die geplanten Messungen konnte ab dem zweiten Prototypen durchgeführt werden und im Endprodukt auch wie gewünscht über CAN übertragen und am PC visualisiert werden. Siehe insbesondere Kapitel 7.2.3 und 8.2.2
- Demonstrator - mittels einer geeigneten Demonstrationsanwendung sollte die Funktionstüchtigkeit demonstriert werden. Unter zu Hilfenahme der Fahrtsimulation TORCS haben wir dies umgesetzt und auf dem Campusfest öffentlich vorgeführt. Siehe Kapitel 9.
- Wissenschaftliche Dokumentation - Auch die notwendige Dokumentationen wurden von uns erstellt. Der Zwischenbericht wurde nach dem ersten Semester erstellt und der Abschlussbericht mit diesem Dokument vorgelegt.

Kapitel 12

Lessons learned

Im Verlauf der Projektgruppe sind wir immer wieder auf einige Probleme gestoßen, die man mit mehr Erfahrung in der Durchführung eines Projektes hätte vermeiden können. Das Kapitel Lessons learned soll nun dazu dienen, allgemeine Fehler und Probleme der Projektgruppe aufzuführen. Diese "Lektionen" sollen uns (den Projektgruppenteilnehmern) sowie eventuell nachfolgenden Projektgruppen dazu dienen, allgemeine Fehler während der Durchführung und Planung eines Projektes zu vermeiden. Zudem zeigt dieses Kapitel, dass wir neben unserem erworbenen technischen Verständnis auch viel hinsichtlich der Durchführung und Organisation eines Projektes gelernt haben, was sich auf beliebige andere Projekte übertragen lässt.

12.1 Projektmanagement

12.1.1 Abschätzung der Arbeitspakete

Die Abschätzung der Arbeitspakete hat während der ersten Phase unserer PG gut funktioniert. In dieser Phase wurden die Anforderungsdokumente geschrieben und viel organisatorische Arbeit (wie zum Beispiel: Hardwarebeschaffung, SVN und Wiki aufsetzen, usw.) erledigt. In dieser Phase ist es zu keinen nennenswerten Verzögerungen gekommen, wodurch die Arbeitspakete wie vorgesehen abgearbeitet werden konnten.

Als wir jedoch angefangen haben, die Hardware in Betrieb zu nehmen, erwies sich die Abschätzung der einzelnen Arbeitspakete als nicht mehr angemessen. Teilweise sind Hardwarefehler aufgetreten, die uns sehr viel Zeit gekostet haben (siehe 7.3.3, 10.3.3). Durch eine bessere Einschätzung dieses Arbeitsaufwands wäre auch eine bessere Zeiteinteilung möglich gewesen. So aber ist es dazu gekommen, dass die eine oder andere Gruppe länger für ihre Arbeiten benötigt hat als es geplant war. Dabei haben wir gelernt, dass man bei der Abschätzung von Arbeitspaketen und dem damit verbundenen Aufwand auch Rückschläge und Verzögerungen beachten und schon am Anfang des Projekts Alternativen hätte berücksichtigen sollen. Des weiteren haben wir gemerkt, dass wir das Gantt-Diagramm, welches wir am Anfang der PG erstellt haben, am besten in

regelmäßigen Abständen hätten aktualisieren sollen. Durch ein aktuelles Diagramm wäre uns jederzeit klar gewesen, ob wir unsere zeitliche Planung einhalten können oder nicht. Ferner wären uns auch wichtige, noch zu realisierende Punkte aufgefallen, welche wir bei unserem Vorgehen erst kurz vor der Deadline bemerkt haben.

Die Planung der Arbeitspakete haben wir in unseren wöchentlichen PG-Sitzungen vorgenommen. Eine Alternative wäre gewesen, die Pakete in einem mehrere Wochen umfassenden Arbeitsplan festzuhalten, welcher bei auftretenden Verzögerungen aktualisiert worden wäre. Ein solcher Arbeitsplan hätte eine genauere Planung der Deadline der einzelnen Pakete ermöglicht. Ohne diese grobkörnigere Planung haben wir die Arbeitspakete von Woche zu Woche neu eingeschätzt oder einfach weitergeführt. Dies ist in manchen Fällen allerdings auch ein Vorteil gewesen, da manche Arbeitspakete auch schneller erledigt werden konnten als geplant. So konnten wir die frei gewordenen Kapazitäten schnell auf andere Arbeitspakete aufteilen.

12.1.2 Planung

Für die Planung unseres Projekts haben wir zu Beginn viel Zeit und Arbeit investiert. Wir haben uns Gedanken über die verschiedenen Anwendungsfälle und Anforderungen gemacht und außerdem die Funktionen, die unbedingt implementiert werden sollten, diskutiert. Diese Investition hat sich sehr bezahlt gemacht. Allein dadurch, dass wir den verschiedenen Funktionen Prioritäten verliehen haben, hatten wir die Flexibilität, bei Zeitknappheit niedriger priorisierte Funktionen zu streichen. Das Ziel unserer Planung, dass wir einmalig viel Arbeit und Sorgfalt in die Planung investieren und danach nur noch im äußersten Notfall diese ändern, ist aufgegangen. Im Rückblick haben wir hier gelernt, dass eine sorgfältige Planung, auch wenn diese sehr zeitaufwändig ist, die solide Grundlage eines jeden Projekts ist.

12.1.3 Risikomanagement

Ein bewusstes Risikomanagent hat es im Verlauf unserer PG nicht gegeben. Wenn wir zurückblicken, wäre ein explizites Risikomanagement sehr sinnvoll gewesen. Aufgrund nicht dokumentierter Hardware, mangels Expertenwissens oder einfach wegen Defekten sind einige Arbeitspakete immer wieder ins Stocken geraten und wurden dadurch nur langsamer bearbeitet oder mussten sogar aufgeschoben werden. Ein auf diese Probleme ausgerichtetes Risikomanagement wäre vorteilhaft gewesen. Somit hätte man bei der Abschätzung der Arbeitspakete auch das Risiko einer Verzögerung abschätzen könnten und hieraus den Einfluss und die Konsequenzen für das gesamte Projekt ableiten können. Außerdem hätte man alternative Aufgaben festlegen können, die im Falle eines technischen Defekts hätten erledigt werden können. An dieser Stelle sei jedoch gesagt, dass sich die einzelnen Gruppen in einem solchen Fall einer alternativen Aufgabe gewidmet haben. Durch ein Risikomanagement hätten sich diese Aufgaben jedoch besser auf das Projekt abstimmen lassen können. Eine weitere Aufgabe des Risikomanagements wäre es gewesen, schon vorher Lösungsstrategien zu entwickeln, damit man im Falle eines Defekts, beim Ausfall eines Projektteilnehmers, einer verspäteten Lieferung oder mangelnder Dokumentation sofort auf diese Strategien zurückgreifen kann, ohne weitere Zeit zu verlieren. Bei einigen Problemen mussten wir erst das weitere Vorgehen in den wöchentlichen Sitzungen beraten, wodurch uns Zeit verloren gegangen ist. In allen Fällen haben wir sofort reagiert und alternative Strategien entwickelt, was

allerdings manchmal zu einem größeren Arbeits- und auch Zeitaufwand geführt hat. Letztendlich haben wir während der PG gelernt, dass viele unvorhersehbare Dinge passieren können. Auch ist es sinnvoll, wie oben ausgeführt, wenn man schon vor dem tatsächlichen Auftreten von Problemen auf eine Lösungstrategie zurückgreifen kann. Auf jeden Fall sollte man Arbeitspakete im Falle einer festen Deadline (wie z.B. dem Campusfest) nicht zu knapp planen, damit man immer noch Spielraum für unvorhergesehene Änderungen bzw. Ereignisse hat.

12.1.4 Rollen in der PG

Eine explizite, statische Rollenverteilung bzw. Aufteilung von Zuständigkeiten und Verantwortungsbereichen hat bei uns nicht stattgefunden, da dies in einer Gruppe von Studenten auch sehr schwierig erscheint. Implizit hat jedoch jeder einmal eine verschiedene Rolle eingenommen wie z.B. die des Projektleiters durch das Leiten der PG-Sitzungen (jede Woche hat ein anderer Student die Sitzungen vorbereit und geleitet), die des Kunden durch das Erstellen der Anforderungen oder die des Entwicklers durch das Implementieren seiner Aufgabe. Durch diese flexible Verteilung von Rollen hat jedes Projektgruppenmitglied einen Einblick in alle für ein solches Projekt typischen Rollen bekommen. Diese Flexibilität hat uns wesentliche Eindrücke vermittelt und die eigenen Stärken und Schwächen in den verschiedenen Bereichen aufgezeigt, was bei einer statischen Rollenverteilung nicht möglich gewesen wäre. Dadurch, dass wir mehrmals verschiedene Rollen besetzt haben, haben wir auch die Möglichkeit gehabt, uns auf vielen Gebieten zu verbessern. Aus unserer Sicht ist es allgemein für derartige Projektgruppen empfehlenswert, diese flexible Rollenzuteilung beizubehalten. So ist es für jeden Teilnehmer möglich, die Sicht auf ein Projekt aus den verschiedenen Perspektiven zu haben und Erfahrung in allen Bereichen der Projektdurchführung zu sammeln.

12.2 SVN

Während der Projektgruppe wurde in einem SVN-Repository entwickelt. Leider haben wir dies erst gegen Mitte des zweiten Semesters konsequent durchgezogen, so dass sich vorher mehrmals Probleme durch veralteten Code ergaben. Vor allem während der Portierung der Software zur Steuerung der Fahrzeugfunktionen auf OSEK erwies sich dies als schwierig. Auf der einen Seite wurde noch an den Funktionen der Software gearbeitet, während eine andere Arbeitsgruppe versuchte den aktuellen Stand zu portieren.

Veralteter Code im SVN beziehungsweise eine für Außenstehende unklare Gliederung innerhalb des Home-Verzeichnisses haben dafür gesorgt, dass wir einige Arbeitstage damit vergeudet haben, veralteten Code zu portieren, der überhaupt keine Relevanz mehr besaß.

Ferner ist jedoch auch zu bedenken, das Code, der sich kompilieren lässt, nicht unbedingt etwas sinnvolles tut. Hierbei ergaben sich durch die Unterschiede der CAN-Treiber für das TC1766 und das TC1796 immer wieder Probleme, da kompilierter aber nicht lauffähiger Code im SVN zu finden war.

Ein SVN ist eine sinnvolle Einrichtung, die bei richtigem Gebrauch vieles erleichtern kann. Benutzt man diese Struktur jedoch nicht konsequent, so ergeben sich hieraus eher Nachteile. Au-

ßerdem ist unbedingt zu beachten, dass bei einem Kopiervorgang identischer Daten für mehrere Projekte innerhalb des SVNs, die .svn-Ordner zu löschen sind, damit die Dateien auch an der neuen Stelle versioniert werden können.

Kapitel 13

Zusammenfassung und Ausblick

Das Ergebnis der einjährigen Arbeit in der PG war unser Endprodukt (10.1). Für die Projektgruppe wurden uns einige Fahrzeugkomponenten zur Verfügung gestellt, darunter zum Beispiel ein Paar Scheinwerfer (5.1.3) und ein Gangwahlschalter (5.1.6). Für das Endprodukt wurden diese Komponenten von uns selbst programmiert, in Betrieb genommen und über einen CAN-Bus verbunden. Zusätzlich wurde dieses Fahrzeugnetz an ein PC-System angeschlossen, auf dem die Software CANoe (5.3.1) lief. Damit konnten einzelne Komponenten gesteuert sowie Busaktivitäten gemessen und visualisiert werden. Zur Demonstration unserer Arbeit wurde zudem ein Teil des Fahrzeugnetzes in die Open-Source-Fahrsimulation TORCS 8.2.3 integriert, so dass das Spiel mit unseren Fahrzeugkomponenten gesteuert werden konnte. Dass die Projektgruppe daher im Hinblick auf die gesteckten Ziele erfolgreich war, wurde bereits in Kapitel 11 erklärt. Im folgenden wird nun erläutert, welchen Gewinn die Projektgruppe für die Teilnehmer gebracht hat.

Die Projektgruppe AutoLab war für alle Teilnehmer das erste Projekt in dieser Größenordnung und über einen solchen Zeitraum. Ein einjähriges, praxisorientiertes Projekt mit zwölf Teilnehmern war sicher für alle eine neue Erfahrung. Genau dort lag auch eine große Herausforderung, denn die Planung und Organisation eines solchen Projekts war für uns alle Neuland. Auch für unser Betreuerteam war es die erste derartige Projektgruppe. Besonders im ersten Semester mussten wir feststellen, dass nicht immer alles so lief, wie wir es uns vorgestellt hatten, und mussten ein ums andere mal unsere Planungen korrigieren. Im zweiten Semester zeigte sich dann, dass wir aus unseren Fehlern gelernt hatten, und es lief deutlich besser. Die gleich zu Beginn des Semesters für jede Woche festgelegten Ziele und Meilensteine konnten in den meisten Fällen eingehalten werden. Hervorzuheben ist, dass die Organisation nicht vom Betreuerteam vorgegeben wurde, sondern weitestgehend, mit der nötigen Unterstützung, den Teilnehmern überlassen wurde – eine Aufgabe, die sicher vielen im späteren Berufsleben wieder begegnen wird. Zu den organisatorischen Aufgaben eines jeden einzelnen gehörten auch das Protokollieren und die Leitung der wöchentlichen Besprechungen.

Wie bei jeder Arbeit in einer Gruppe war auch in der Projektgruppe AutoLab das Teamwork von großer Bedeutung. Sicher hat jeder der Teilnehmer schon vorher einmal in einem Team gearbeitet, doch bei einem so umfangreichen Projekt war es doch etwas anderes. Um die Ar-

beit aufzuteilen, wurden Arbeitsgruppen gebildet. Da sich im Verlauf der Projektgruppe die Aufgabenfelder änderten, wechselte auch die Gruppeneinteilung einige Male, so dass jeder PG-Teilnehmer mit jedem Teilbereich in Berührung gekommen. Darüberhinaus mussten sich die PG-Mitglieder dadurch einige Male auf neue Teampartner einstellen und vor allem auch immer wieder in unterschiedliche, neue Themengebiete einarbeiten. Dies ist eine Fähigkeit, die auch im späteren Berufsleben verlangt werden wird.

Neben der eigentlichen Zusammenarbeit in den Teilgruppen wurde im Laufe der PG auch ein weiterer wichtiger Aspekt der Teamarbeit deutlich: Die Notwendigkeit des transparenten Arbeitens. Bei einem solchen Projekt ist es sehr wichtig, dass jeder seine persönlichen Arbeiten und Erkenntnisse für alle ersichtlich und nachvollziehbar protokolliert. Dies ist zum einen wichtig, damit immer alle Mitglieder der Gruppe auf dem aktuellen Stand sind und nicht auf der Grundlage eines überholten Kenntnisstands arbeiten. Zum anderen ist dies wichtig, damit die Projektgruppe für gewisse Arbeiten nicht von einzelnen Mitgliedern abhängig ist. Anfangs ist dies in der Projektgruppe unterschätzt worden, so dass beispielsweise die TriBoards (5.1.2) nur in Anwesenheit der zuständigen Gruppe genutzt werden konnte. In der Folge wurde dieser Fehler dann vermieden, und es traten diesbezüglich keine Probleme mehr auf. Als Hilfsmittel wurden unter anderem die Hilfsmittel Wiki und SVN (2.2) verwendet. Dieses Vorgehen ist auch in der Praxis üblich.

Auch fachlich konnten viele Erkenntnisse und Erfahrungen gewonnen werden. Die Arbeiten in der Projektgruppe waren sehr verschiedenartig. Sie reichten unter anderem von Hardwareuntersuchungen und -basteleien über Programmierung etwa der Mikrocontroller bis hin zur Anwendung von professioneller Software, die auch in der Industrie eingesetzt wird (zum Beispiel 5.3.1 CANoe). All diese Aufgaben zeichneten sich durch ihren sehr hohen Praxisbezug aus, der in konventionellen Lehrveranstaltungen nicht oder nur minimal vorhanden und in dieser Form auch gar nicht möglich ist.

Abschließend kann man sagen, dass die Projektgruppe auch für uns Teilnehmer ein großer Erfolg war, aus der wir viele wertvolle Erfahrungen mitnehmen können. Die einjährige Arbeit in Teams und Teilgruppen, das Vorgehen nach in der Industrie verbreiteten Methoden und der Umgang mit modernen Hard- und Softwarekomponenten, die aktuellen Industriestandards entsprechen, gaben einen sehr guten Einblick in das Berufsleben, das uns in Zukunft erwarten könnte. Diese Faktoren machten die Projektgruppe zu einer Lehrveranstaltung, die eine ideale Ergänzung zu dem theoretischen Wissen darstellt, welches in konventionellen Lehrveranstaltungen vermittelt wird.

Aufgrund des Erfolges der Projektgruppe ist es verständlich, dass sowohl die Arbeit am AutoLab als auch das Prinzip der Projektgruppe in diesem Bereich fortgesetzt werden. In den folgenden zwei Semestern wird die Projektgruppe 522 AutoLab durch die Projektgruppe 533 CoaCh beerbt, die sich mit neuen Steuergeräte-Architekturen im Automobilbereich beschäftigt. Zudem wird die Arbeit der PG 522 in einem SchoolLab des deutschen Zentrums für Luft- und Raumfahrttechnik (DLR) an der TU Dortmund genutzt werden, das von einem der PG-Teilnehmer betreut werden wird.

Literaturverzeichnis

[Beu06] Beucher, Ottmar: MATLAB und Simulink. Pearson Studium, 2006.

[Bro06] Broy, Manfred: Challenges in automotive software engineering. In Leon J. Osterweil, H.Dieter Rombach, May Lou Soffa, ACM, 2006.

[DC02] Daimler-Chrysler: Der neue Maybach. ATZ/MTZ Sonderheft, page 125, 2002.

[Dud04] Dudenhöffer, Ferdinand: Studie: Ausfallsicherheit der Fahrzeugelektronik. Automotive, 2004.

[Ets02] Etschberger, Konrad: Controller Area Network. Carl Hanser Verlag, 2002.

[FF] Felix Fastnacht, Dr. Jochen Schoof: OSEKtime - die ersten praktischen Erfahrungen mit dem neuen Standard. http://joscho.de/PUB/ei2002b.pdf.

[For07] Form, Prof. Dr.-Ing. Thomas: Folien zur Vorlesung "Datenbussysteme in Straßenfahrzeugen" an der TU Braunschweig. http://www.ifr.ing.tu-bs.de/lehre/downloads/skripte/skript_dbf_38.pdf, 2007. [Stand: 15.10.2007].

[ftc01] OSEK/VDX fault tolerant communication. Version 1.0. http://portal.osek-vdx.org/files/pdf/specs/ftcom10.pdf, July 2001.

[Gal] Galla, Thomas M.: TimeCore - Software Komponenten und Entwurfswerkzeuge für sicherheitsrelevante verteilte Applikationen im Automobil. http://members.chello.at/thomasmgalla/papers/2004-04-24_Paper_Elektronik_Automotive.pdf.

[GK97] G. Kiczales, J. Lamping; A. Mendhekar; C. Maeda; C. Lopes; J.-M. Loingtier; J. Irwin: Aspectoriented programming. In M. Aksit and S. Matsuoka, editors, Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP '97), volume 1241 of Lecture Notes in Com- puter Science, pages 220-242, 1997.

[Hom05] Homann, Matthias: OSEK; Betriebssystem-Standard für Automative und Embedded Systems. mitp-Verlag, Bonn, 2005.

[H.W05] H.Wörn, U.Brinkschulte: Echtzeitsysteme. Springer, 2005.

[JS06] J. Schäuffele, T. Zurawka: Automotive Software Engineering. ATZ/MTZ Fachbuch, 2006.

[Kop91] KOPETZ, HERMANN: Event-Triggered versus Time-Triggered Real-Time Systems. Proc. Int. Workshop on Operating Systems of the 90s and Beyond Lecture Notes in Computer Science Vol.563, pages 87-101, 1991.

[Kop93] KOPETZ, HERMANN: Should Responsive Systems be Event-Triggered or Time-Triggered? IEICE TRANS. INF. and SYST., Vol. E76-D, NO.11, 1993.

[Kop97] KOPETZ, HERMANN: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, 1997.

[Kop01] KOPETZ, HERMANN: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, 2001.

[lin06] LIN Specification Package Rev. 2.1. http://lin-subbus.org/, 2006.

[LMN04] LINDA M. NORTHROP, PAUL C. CLEMENTS: An introduction to software product lines. In Robert L. Nord, editor, SPLC, volume 3154 of Lecture Notes in Computer Science, page 322, 2004.

[Mar03] MARWEDEL, PETER: Embedded System Design. Kluwer Academic Publishers, 2003.

[MR06] MARTIN RAPPL, ALEXANDER PRETSCHNER; CHRISTIAN SALZMANN; THOMAS STAUNER: 3rd international workshop on software engineering for automotive systems -seas 2006. In ICSE ´06: Proceeding of the 28th international conference on Software engineering, pages 1034, New York, NY, USA, 2006.

[msc04] MSCAN Block Guide V02.15. http://www.freescale.com/files/microcontrollers/doc/ref_manual/S12MSCANV2.pdf, Juli 2004.

[msc05] Using MSCAN on the HCS12 Family. http://www.freescale.com/files/microcontrollers/doc/app_note/AN3034.pdf, September 2005.

[Oet97] OETKEN, M.: Seminar Echtzeitbetriebssystemkerne. Oetken, 1997.

[OS03] OLAF SPINCZYK, WOLFGANG SCHRÖDER-PREIKSCHAT; DANILO BEUCHE; HOLGER PAPAJEWSKI: PURE/OSEK - Eine aspektorientierte Betriebssystemfamilie für Kraftfahrzeuge. In Proceedings of the GI Workshop on Automotive SW Engineering and Concepts, pages 330-334, Frankfurt/Main, Germany, 2003.

[OS05] OLAF SPINCZYK, DANIEL LOHMANN; MATTHIAS URBAN: Advances in AOP with AspectC++. In Hamido Fujita and Mohamed Mejri, editors, New Trends in Software Methodologies, Tools and Techniques (SoMeT '05), number 129 in Frontiers in Artificial Intelligence and Applications, pages33-53, Tokyo,Japan, 2005.

[OS06] OLAF SPINCZYK, HOLGER PAPAJEWSKI: Using feature models for product derivation. In Proceedings of the 10th Software Product Line Conference (SPLC 06), Baltimore, Maryland, 2006.

[ose01] OSEK/VDX time triggered operating system. Version 1.0. http://portal.osek-vdx.org/files/pdf/specs/ttos10.pdf, July 2001.

[ose05] OSEK OS. Version 2.2.3. http://portal.osek-vdx.org/files/pdf/specs/os223.pdf, February 2005.

[Scha] SCHOOF, DR. JOCHEN: OSEKtime - Betriebssystem-Standard für X-by-Wire. http://www.joscho.de/PUB/BSsymp03.pdf.

[Schb] SCHOOF, DR. JOCHEN: OSEKtime - Standard für zeitgesteuerte Betriebssysteme. http://joscho.de/PUB/ei2001.pdf.

[TMG] THOMAS M. GALLA, JOCHEN OLIG: OSEKtime - Eine Softwareplattform für sicherheitsrelevante verteilte Applikationen im Automobil. http://members.chello.at/thomasmgalla/papers/2003-08-21_Paper.pdf.

[vdia] OSEK-VDX, NM Spezifikation Version 2.5.3. http://portal.osek-vdx.org/files/pdf/specs/nm253.pdf. [Stand: 11.02.2008].

[vdib] OSEK-VDX, Operating System Specification 2.2.3. http://portal.osek-vdx.org/files/pdf/specs/os223.pdf. [Stand: 13.02.2008].

[vdi00] OSEK-VDX, open systems in automotive networks. VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik. Düsseldorf: VDI-Verlag, 2000.

[vdi04] OSEK/VDC Communication. Version 3.0.3. http://portal.osek-vdx.org/files/pdf/specs/osekcom303.pdf, July 2004.

[Wil06] WILLIAMS, R.: Real time System Development. Elsevier, 2006.

[WZ07] WERNER ZIMMERMANN, RALF SCHMIDGALL: Bussysteme in der Fahrzeugtechnik; Protokolle und Standards. Vieweg, 2007.

Abbildungsverzeichnis

Index