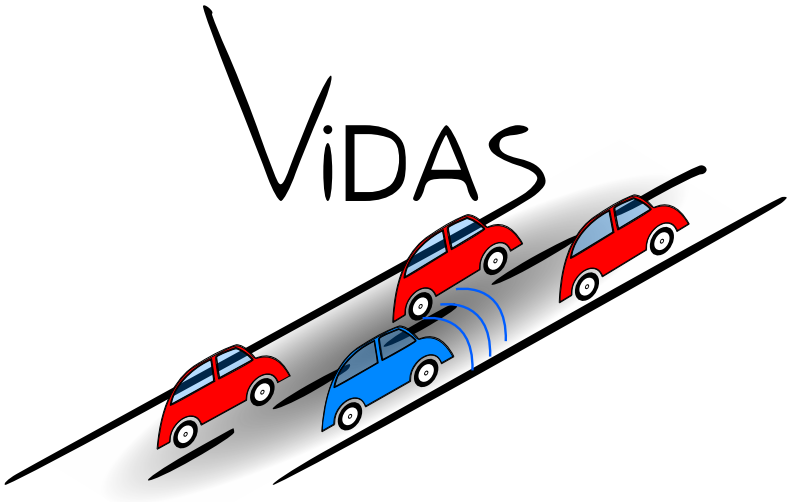


Projektgruppe **ViDAs**
Carl von Ossietzky Universität Oldenburg
Department für Informatik
2009-2010

Endbericht

Version 1.0
30. November 2010

Jianyu Bao, Peter Battram, Alex Enkelmann, Andreas Gabel,
Jens Heyen, Thilo Koepke, Christoph Lässche, Sven Sieverding

Inhaltsverzeichnis

1 Einleitung **7**
- 1.1 Aufgabenstellung . . . 7
- 1.2 Vorgaben . . . 7
- 1.3 Fachbegriffe und Abkürzungen . . . 8
- 1.4 Rechtliches . . . 8
- 1.5 Struktur dieses Dokuments . . . 8

2 Projektorganisation **9**
- 2.1 Vorgehensmodell . . . 9
- 2.2 Gruppenorganisation . . . 12
  - 2.2.1 Zeitplanung . . . 13
  - 2.2.2 Meilensteine . . . 13

3 Analyse und Definition **17**
- 3.1 Anforderungsdefinition . . . 17
  - 3.1.1 Vorgehensweise . . . 17
  - 3.1.2 Überblick über die Anforderungsdefinition . . . 18
- 3.2 Architektur- und Schnittstellendokument . . . 24
  - 3.2.1 Vorgehensweise . . . 25
  - 3.2.2 Überblick über das Architektur- und Schnittstellendokument . . . 25

4 Implementierung **31**
- 4.1 SILAB-Fahrsimulator . . . 31
  - 4.1.1 Aufbau des SILAB-Fahrsimulators . . . 31
  - 4.1.2 Aufbau und Funktionsweise einer DPU . . . 32
  - 4.1.3 Erweiterung des Simulators . . . 33
- 4.2 ViDAs-Umweltschnittstelle . . . 34
  - 4.2.1 Sensorik . . . 35
  - 4.2.2 Benutzerschnittstelle . . . 41
- 4.3 ViDAs-Komm . . . 43
  - 4.3.1 Nachrichten . . . 45
  - 4.3.2 High-Level-Architecture-Schnittstelle . . . 45
  - 4.3.3 DPU-Einbettung . . . 48
- 4.4 ViDAs-Kern . . . 51
  - 4.4.1 ACC-Submodul . . . 51
  - 4.4.2 LCA-Submodul . . . 62

4.4.3 Multiplexer . . . 69

5 Testen **71**
5.1 Testdatenbank . . . 71
5.2 Testplanung und -durchführung . . . 73
5.3 Automatisches Testen . . . 81
5.3.1 Gründe für das automatische Testen . . . 81
5.3.2 Erstellung von Testszenarien . . . 81

6 Machbarkeitsstudie **87**
6.1 Motivation . . . 87
6.2 Gegebene Hardwareplattform . . . 88
6.3 Arbeitsfluss . . . 89
6.3.1 Übersicht . . . 90
6.3.2 Vom Matlab-Modell zum Maschinencode . . . 92
6.3.3 Vom FPGA zum System-on-Chip . . . 96
6.4 Ausführungszeitanalyse . . . 103
6.4.1 Messbasierte Ausführungszeitbestimmung durch Code-Einbringung . . . 104
6.4.2 Statische Ausführungszeitanalyse . . . 107
6.5 CAN-Kommunikation . . . 109
6.5.1 Einleitung . . . 109
6.5.2 Grundlagen . . . 110
6.5.3 Vorüberlegungen . . . 110
6.5.4 Schedule . . . 113
6.5.5 Float-Werte . . . 113
6.5.6 Vorgehensweise . . . 113
6.5.7 Benutzte Komponenten . . . 114
6.6 Auswertung der Ausführungszeitanalyse . . . 115
6.6.1 Überprüfung der Hardwareplattform . . . 115
6.6.2 Annotation der gewonnenen Zeiten im Modell . . . 119

7 Ergebnisse und Fazit **123**
7.1 Ergebnisse . . . 123
7.1.1 Aufgabenstellung . . . 123
7.1.2 Ziele über die Aufgabenstellung hinaus . . . 123
7.1.3 Erweiterungsmöglichkeiten . . . 124
7.2 Fazit . . . 124
7.2.1 Gruppenklima . . . 124
7.2.2 Gruppenorganisation . . . 124
7.2.3 Kommunikation . . . 125
7.2.4 Gruppenräume . . . 126
7.2.5 Erfahrungen . . . 127
7.2.6 Danksagungen . . . 127

A Anhang **129**

A.1 Bilder 129
A.2 Projektplan-Export 168
A.3 Testplan-Export 177
A.4 FMEA-Export 184

B. Abbildungsverzeichnis **196**

C. Tabellenverzeichnis **199**

D Konstanten **200**

E. Glossar **201**

F. Literaturverzeichnis **204**

1 Einleitung

Das vorliegende Dokument stellt den Abschlussbericht der Projektgruppe „Virtual Driver Assistant“ (*ViDAs*) dar. Im Zeitraum vom Wintersemester 2009/2010 bis zum Wintersemester 2010/2011 wurde von dieser Projektgruppe ein *Fahrerassistenzsystem* entwickelt, welches den *Fahrer* sowohl beim Einfädeln auf die Autobahn als auch beim Fahrspurwechsel unterstützt. Neben diesem „Lane Change Assistant“ (*LCA*) ist ebenso ein „Adaptive Cruise Control“ (*ACC*) entwickelt worden, das eine durch den Fahrer eingestellte Geschwindigkeit hält und gegebenenfalls diese der Fahrsituation anpasst.

1.1 Aufgabenstellung

Die Aufgabenstellung der Projektgruppe bestand darin, ein Fahrerassistenzsystem für einen PKW der Mittelklasse zu entwickeln, welches auf einer vierspurigen Autobahn mit Auf- und Abfahrten funktionieren muss.

Das Fahrerassistenzsystem musste zum Einen ein LCA enthalten, der beim Spurwechsel und auch beim Einfädeln auf die Autobahn unterstützt. Dabei wird der Fahrer darauf hingewiesen, wann er die Fahrspur wechseln, beziehungsweise auf die Autobahn auffahren kann. Dazu muss sowohl der Verkehr nach vorn, als auch der rückwärtige Verkehr beobachtet und daraus eine Prognose, ob ein Fahrspurwechsel möglich ist, erstellt werden. Ebenso werden durchgezogene Linien erkannt und bei der Erstellung der Prognose berücksichtigt.

Zum Anderen sollte ein ACC implementiert werden. Dieses unterstützt den Fahrer beim Halten einer zuvor gewählten Geschwindigkeit unter Berücksichtigung der gesetzlich zulässigen Höchstgeschwindigkeit. Ebenso dient es dazu, den minimal nötigen *Sicherheitsabstand* einzuhalten. Dazu wird bei zu dichtem Auffahren die Geschwindigkeit des vorderen Fahrzeugs übernommen. Sobald kein Vorderfahrzeug mehr vorhanden ist, wird die Geschwindigkeit wieder auf die *Wunschgeschwindigkeit* des Fahrers geregelt. Falls auf der aktuellen Strecke eine Geschwindigkeitsbegrenzung gilt, so wird diese erkannt und eingehalten, wobei der Fahrer dennoch eine eigene, höhere Geschwindigkeit wählen kann. Dabei werden auch unterschiedliche Geschwindigkeitsbegrenzungen für beide Fahrspuren berücksichtigt.

1.2 Vorgaben

Anstelle eines realen Fahrzeugs sollte der Fahrsimulator *„SILAB“* verwendet werden. Um eine Umgebung mit Realitätsnähe zu schaffen, mussten real existierende Sensoren ausgewählt werden und mit ihren Eigenschaften innerhalb des SILAB-Fahrsimulators nachgebaut werden.

Desweiteren sollte eine Machbarkeitsstudie zur Portierung des Fahrerassistenzsystemes auf ein gegebenes FPGA-Board der Firma Xilinx (siehe Abschnitt 6.2) erstellt werden, um ein realitätsnahes System zu erhalten. Als Teil dieser Machbarkeitsstudie sollte ebenfalls ein Konzept zur CAN-Kommunikation zwischen Fahrerassistenzsystem und SILAB-Fahrsimulator erarbeitet werden.

1.3 Fachbegriffe und Abkürzungen

Alle in diesem Dokument enthaltenen Fachbegriffe und Abkürzungen sind im Glossar aufgeführt und werden dort erläutert. Die Begriffe sind kursiv gedruckt.

1.4 Rechtliches

Alle in diesem Dokument genannten Markennamen sind Warenzeichen der jeweiligen Eigentümer.

1.5 Struktur dieses Dokuments

Auf diese Einleitung folgt zunächst das Kapitel „Projektorganisation“, in dem das Vorgehensmodell und die Organisation der Projektgruppe beschrieben werden.

Darauf folgt das Kapitel „Analyse und Definition“, das die beiden vorherigen Dokumente der Projektgruppe noch einmal zusammenfasst.

Im Anschluss daran wird beschrieben, auf welche Weise und in welcher Simulationsumgebung das Fahrerassistenzsystem implementiert wurde.

Das Kapitel „Testen“ beschreibt die Testphase, in der das Fahrerassistenzsystem und alle zugehörigen Komponenten/Submodule auf korrekte Funktionalität überprüft wurden.

Nach der Vorstellung der Testphase folgt das Kapitel „Machbarkeitsstudie“, in dem die Ergebnisse der Prüfung, ob das Fahrerassistenzsystem auch auf externer Hardware lauffähig ist, dargestellt werden.

Am Ende des Kapitels befindet sich ein abschließendes Fazit, das die Erfahrungen der Gruppenmitglieder während der Projektlaufzeit zusammenfasst.

2 Projektorganisation

2.1 Vorgehensmodell

Abbildung 2.1 stellt das verwendete Vorgehensmodell dar. Hierbei handelte es sich um das V-Modell [V-M], da sich dieses durch das strukurierte Vorgehen und die ausgedehnten Testphasen besonders gut für die Entwicklung komplexer Systeme mit hohen Anforderungen an die Sicherheit eignet.

Das V-Modell wurde im Projekt durch „Concurrent Engineering“ erweitert, sodass Aufgaben auch parallel über mehrere Phasen des V-Modells bearbeitet wurden. Dieses Vorgehen birgt zwar das Risiko von Konflikten, die erst bei einer Überarbeitung behoben werden können, dafür werden Fehler früher entdeckt und die Auswirkungen sind geringer. So konnte sich z.B. ein Team, parallel zu den Entwurfsphasen, bereits mit neuen Techniken vertraut machen und Erfahrungen mit deren Umgang sammeln.

Der linke, absteigende Ast des V-Modells dient der Beschreibung und Entwicklung des Fahrerassistenzsystems. Durch diese unterschiedlichen Phasen gelang es der Projektgruppe, das zu entwickelnde System und dessen Umgebung zu verstehen und das Produkt daran optimal anzupassen. Gleichzeitig zur Entwicklung wurden die Testfälle für den Modultest, den Integrationstest und den abschließenden Systemtest erstellt.

Der rechte, aufsteigende Ast des V-Modells befasst sich nach der Implementierung mit der ausgiebigen Validierung des entwickelten Systems. Angefangen von Modultests, bei denen auf unterster Ebene auf Korrektheit geprüft wurde, über Integrationstests bis hin zum Systemtest, wurden alle Module nochmals vom Verantwortlichen getestet und ggf. korrigiert.

Projektbegleitend wurden sogenannte „transversale Prozesse“, zum Teil über die gesamte Projektlaufzeit, durchgeführt. Hierbei handelte es sich um das Projektmanagement und das Konfigurationsmanagement. Die Erstellung einer Testdatenbank sowie das Modellieren von Testumgebungen waren weitere transversale Prozesse, die während der Grobentwurfsphase begannen und bis zur Phase der Modultests abgeschlossen waren.

Das Concurrent Engineering wurde ab der Phase des Grobentwurfs eingesetzt. Während des Grobentwurfs war das System in mehrere Module aufgeteilt und deren Entwicklung von getrennten Teams über die Phasen Feinentwurf, Implementierung und Modultest fortgesetzt worden. Zur Phase der Integrationstests wurden die Module miteinander zu einem Gesamtsystem verbunden und getestet. Wurden dabei Probleme festgestellt, erfolgte ein Rücksprung an die entsprechende Stelle im V-Modell und es wurde nachgebessert.

Im Folgenden werden die durchgeführten Arbeiten der einzelnen Phasen genauer erläutert.

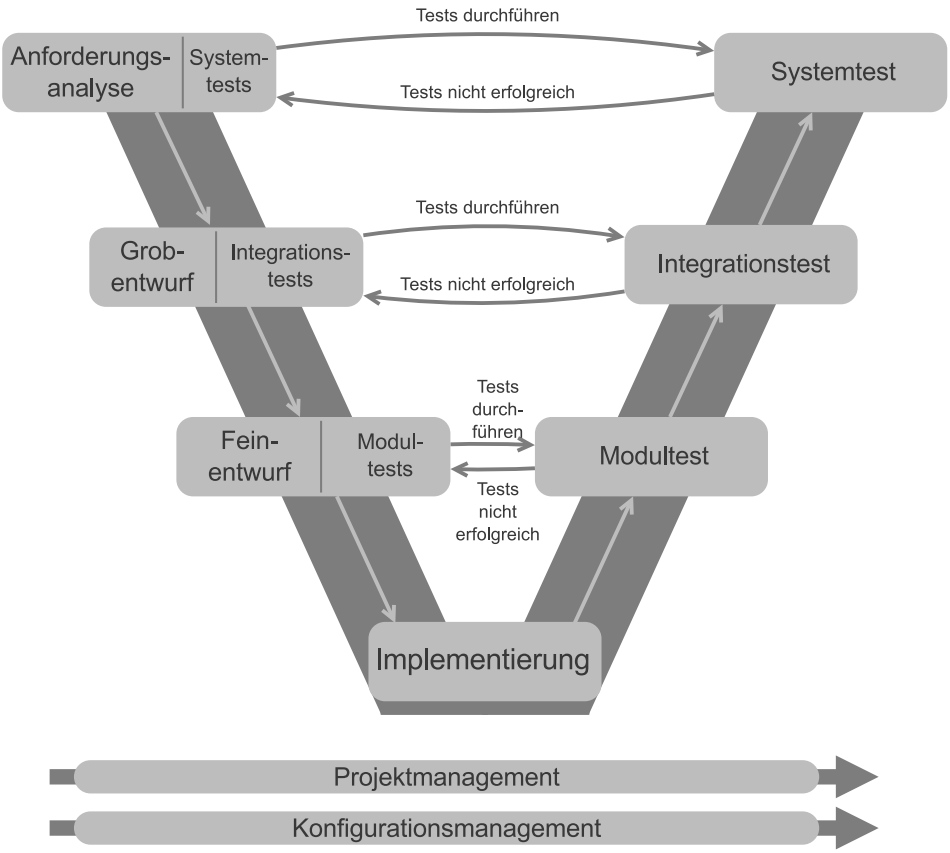


Abbildung 2.1: Vorgehensmodell: V-Modell

Anforderungsanalyse

In der Phase der Anforderungsanalyse wurden alle Anforderungen an das Fahrerassistenzsystem zusammengetragen. Diese Phase diente der Findung aller Funktionen, die das Fahrerassistenzsystem erfüllen musste und des besseren Verständnisses der Projektgruppe für dieses Themenumfeld. Die Erkenntnisse wurden in der „Anforderungsdefinition“ [Pro10a] festgehalten.

In diesem Dokument stehen die Anforderungen an das Verhalten des Systems, an das Ego-Fahrzeug und an die Umgebung, in der das Fahrerassistenzsystem betrieben werden soll (siehe [Pro10a], Seite 6 ff). Ebenso sind die Anforderungen an die Nutzerinteraktion und die funktionalen und nicht-funktionalen Anforderungen (siehe [Pro10a], Seite 18 ff) in diesem Dokument aufgeführt.

Daher wurde dieses Dokument auch als Grundlage verwendet, um sowohl die zu erbringende Leistung festzuhalten, als auch als Kommunikationsmittel zwischen den Mitgliedern der Projektgruppe und den Betreuern. Alle weiteren Entwicklungsschritte bauten auf dieser Anforderungsdefinition auf.

Parallel zur Phase der Anforderungsanalyse wurden die Tests für den Systemtest aufgestellt (siehe [Pro10a], Seite 54 ff).

Grobentwurf

Während des Grobentwurfs wurde die Architektur des Systems entwickelt. Dabei wurde es in drei Module aufgeteilt: ViDAs-Kern, ViDAs-Komm und ViDAs-Umweltschnittstelle. Diesen Modulen sind Funktionalitäten zugeordnet worden, die grundlegend in [Pro10c] auf Seite 3 ff beschrieben wurden. Außerdem wurde definiert, dass die Kommunikation des ViDAs-Kerns mit der ViDAs-Umweltschnittstelle ausschließlich über das ViDAs-Komm-Modul realisiert werden sollte.

Feinentwurf

In der Phase des Feinentwurfs wurde für jedes dieser drei Module ein Design erarbeitet und die gewünschte Funktionalität detailliert beschrieben, sodass in der anschließenden Implementierungsphase das Verhalten des Fahrerassistenzsystems modelliert werden konnte. Diese Beschreibungen sind ebenfalls in [Pro10c] auf Seite 4 ff festgehalten.

Zudem wurden die Kommunikationsschnittstelle zwischen den Modulen ausgearbeitet und ein Nachrichtenformat definiert. Alle für das Fahrerassistenzsystem benötigten Nachrichten wurden in [Pro10c] auf Seite 11 ff aufgelistet.

Implementierung

In dieser Phase wurde das in [Pro10c] und [Pro10a] beschriebene Verhalten des Fahrerassistenzsystems in ein Matlab-Modell umgesetzt. Der Fahrsimulator SILAB wurde um die benötigte Sensorik für das Ego-Fahrzeug erweitert und Anpassungen zur Einhaltung der Anforderungen an die Umwelt und das Ego-Fahrzeug wurden vorgenommen. Ebenso wurde mit der Machbarkeitsstudie für das Gesamtsystem begonnen. So wurde bereits die gegebene Hardware analysiert (siehe Abschnitt 6.2) und die Konfigurationsmöglichkeiten des FPGAs und des Codegenerators wurden untersucht (siehe Abschnitt 6.3).

Modultest

In der Phase der Modultests wurden einfache Tests auf unterster Ebene durchgeführt und so die Implementierung überprüft, um Fehler zu finden und zu beheben. Hierzu wurde für jedes ViDAs-Kern-Submodul eine eigene Testumgebung entwickelt, um ausführliche Tests leicht und wiederholbar durchführen zu können (siehe Abschnitt 5). Für die Sensorikerweiterungen des SILAB-Fahrsimulators wurden definierte Teststrecken erstellt und die von der Sensorik gelieferten Werte mit denen in der Teststrecke definierten Werten verglichen. Zum Testen des ViDAs-Komm-Moduls wurden eine Matlab und eine SILAB-Testumgebung entwickelt, von denen jeweils Werte versendet und empfangen werden können. Die Ergebnisse dieser Tests sind in der Testdatenbank (siehe Abschnitt 5.1) dokumentiert.

Integrationstest

Für die Integrationstests wurde das Modul ViDAs-Kern mit dem Modul ViDAs-Komm und das Modul ViDAs-Umweltschnittstelle mit dem Modul ViDAs-Komm verbunden und getestet. Es wurde überprüft, ob sich die Module untereinander wie gewünscht verhalten, also die Schnittstellen korrekt implementiert wurden. Hierzu wurden in beiden Fällen Daten an das Modul ViDAs-Komm angelegt und die Reaktion des Systems ausgewertet.

Systemtest

Beim Systemtest wurden alle Komponenten des Systems zu einem Gesamtsystem integriert und auf Fehler untersucht. Dabei wurde mit den in [Pro10a] auf Seite 65 ff beschriebenen Testszenarien auf die in [Pro10a] auf Seite 18 ff beschriebenen Funktionalitäten getestet. Zur Durchführung dieser Tests wurden automatisierte Fahrerskripte eingesetzt, siehe Abschnitt 5.3. Diese Testergebnisse wurden ebenfalls in der Testdatenbank festgehalten.

2.2 Gruppenorganisation

Die Gruppe setzte sich aus acht Gruppenmitgliedern zusammen, die gemeinsam die Anforderungsdefinition erarbeitet haben. Für das weitere Vorgehen nach der Fertigstellung der Anforderungsdefinition wurden die Gruppen Matlab, SILAB und Machbarkeitsstudie gebildet. Die Matlab-Gruppe entwickelte mit Hilfe von Matlab und dessen Toolboxen Simulink und Simulink/Stateflow das Fahrerassistenzsystem. Die Anforderungen an die Umwelt und das Fahrzeug wurden von der SILAB-Gruppe umgesetzt und in der Machbarkeitsstudie-Gruppe wurde die Machbarkeitsstudie für das Gesamtsystem erstellt. Innerhalb jeder dieser Gruppen wurde jeweils ein Ansprechpartner festgelegt, welcher die Koordination dieser Gruppe übernahm. In den wöchentlichen Statustreffen und in Rücksprache mit dem Projektleiter wurden Arbeitspakete für diese Untergruppen aufgestellt, welche mit Deadlines versehen an die einzelnen Personen verteilt wurden. Diese Arbeitspakete waren auch in den Protokollen der wöchentlichen Statustreffen nachzulesen.

Neben diesen Gruppen wurden ebenfalls noch Verantwortungsbereiche an einzelne Personen vergeben. So gab es innerhalb des Projektes einen Projektleiter, einen stellvertretenden Projektleiter, einen Test-Beauftragten, einen Webseiten-Beauftragten, einen Dokumenten-Beauftragten, einen Versionsmanagement-Beauftragten, einen SILAB-Beauftragten und einen Administrator. Die Aufgaben, die mit diesen Verantwortungsbereichen verbunden waren, sind die Folgenden:

- Der Projektleiter sowie der Stellvertreter waren für die Erstellung und Wartung des Projektplanes, sowie die Koordinierung der Gruppe verantwortlich.
- Der Test-Beauftragte war für die Erstellung und Wartung des Testplans und der Testdatenbank verantwortlich. Außerdem war er für die Koordinierung der Durchführung der Tests zuständig.
- Der Webseiten-Beauftragte erstellte und pflegte die Webseite der Projektgruppe.

- Der Dokumenten-Beauftragte war für die Erstellung von Dokumentenvorlagen verantwortlich und war der Ansprechpartner bei Problemen mit der Dokumentenerstellung.
- Der Versionsmanagement-Beauftragte war für die Einrichtung des Versionsmanagementsystem *git* zuständig und fungierte als Ansprechpartner bei Problemen mit dem Umgang mit git.
- Der SILAB-Beauftragte war für die Einrichtung und Konfiguration des SILAB-Fahrsimulators verantwortlich. Er war ebenso Ansprechpartner bei Problemen beim Umgang mit SILAB.
- Der Administrator war für die Pflege der Arbeitsrechner sowie für die Installation der, für die Projektgruppe benötigten, Werkzeuge auf den Rechnern zuständig.

2.2.1 Zeitplanung

Für die Zeitplanung des Projektes wurde das Werkzeug *GanttProject* eingesetzt. Der Ablauf des Projekts war durch das V-Modell bereits zu einem großen Teil vorgegeben und wurde im Detail ausgearbeitet. Dieser detailierte Projektplan ist im Anhang dieses Dokumentes zu finden und ist auf der Homepage der Projektgruppe sowie im Versionsmanagement git für alle Gruppenmitglieder zugänglich. Der Projektplan wurde in den wöchentlichen Gruppentreffen diskutiert und Arbeitspakete hieraus wurden mit Deadlines versehen an die Gruppenmitglieder verteilt. Die wöchentlichen Gruppentreffen wurden zudem von einem Gruppenmitglied moderiert und einem weiteren Gruppenmitglied protokolliert.

Protokolle dieser Gruppentreffen sind auf dem öffentlichen Teil der Homepage für jeden einzusehen. Vor den großen Meilensteinen aus Abschnitt 2.2.2 wurden Teilausschnitte des Projektplans grafisch aufgearbeitet und im Arbeitsraum der Projektgruppe aufgehangen. Ein solcher aufbereiteter Plan ist in Abbildung 2.2 zu sehen.

2.2.2 Meilensteine

Drei Meilensteine für die Projektplanung wurden durch die drei geplanten Reviews am 26. März 2010, 26. August 2010 und am 09. Dezember 2010 definiert und waren mit Ausnahme des dritten Review auch Stichtage für die Abgabedokumente. Der Endbericht wurde bereits am 30. November 2010 abgegeben. Der Inhalt und die Ergebnisse des Review 1 und Review 2, sowie der Inhalt des Review 3 sind im Folgenden beschrieben:

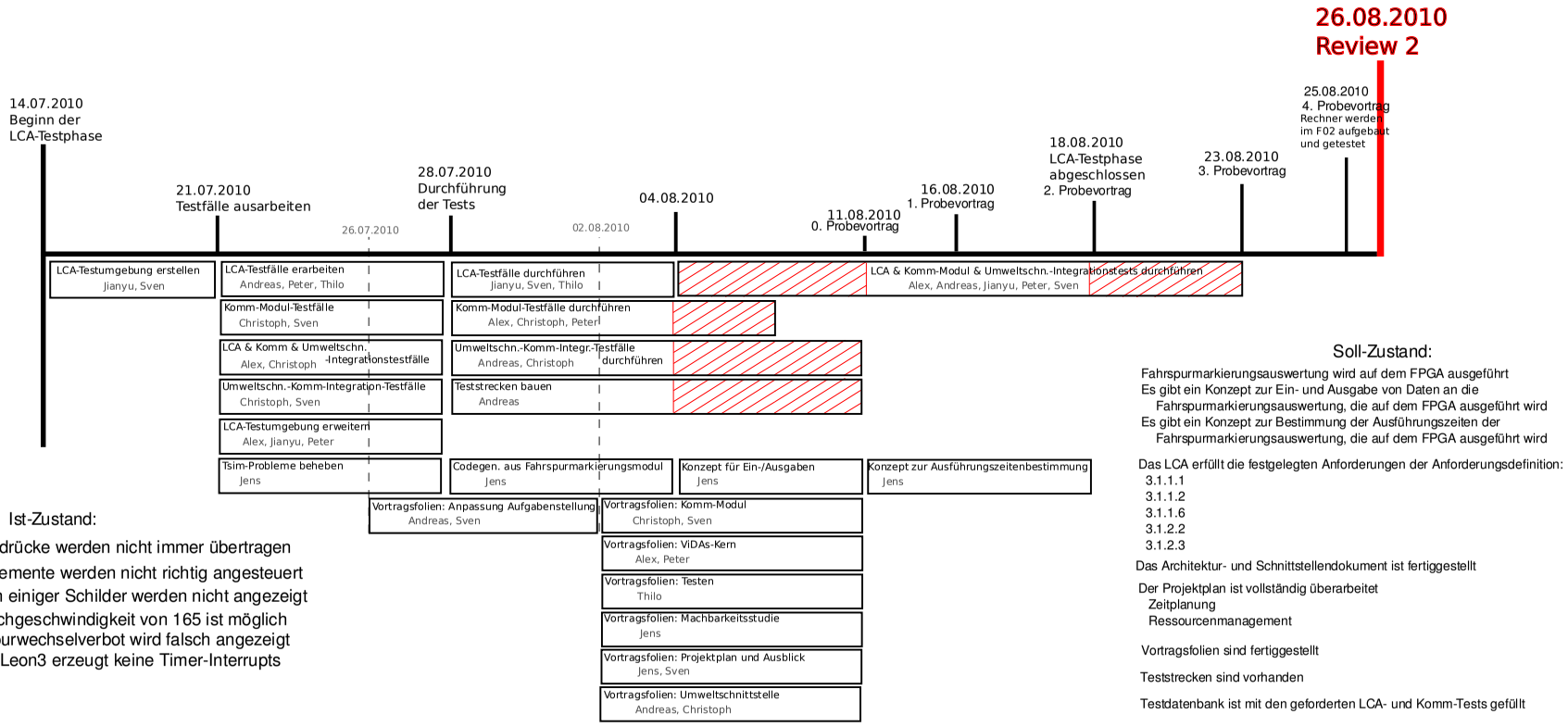
- **Review 1 am 26. März:**
  Auf dem ersten Review am 26. März 2010 wurde die Aufgabenstellung und die Einschränkungen, unter denen das spätere Fahrerassistenzsystem funktionieren sollte, vorgestellt. Desweiteren wurden die Anforderungen an das Fahrerassistenzsystem und die Testfälle präsentiert, mit denen diese Anforderungen überprüft wurden.

  Auch die geplante Architektur des Fahrerassistenzsystems wurde vorgestellt.

  Nach anschließender Diskussion wurde angemerkt, dass das Projekt zu umfangreich sei, um es im gegebenen Zeitraum zu erfüllen. So wurden nach diesem Review einige weitere

Einschränkungen getroffen, sodass die Aufgabe einen angemessen Umfang erhielt. Diese Änderungen sind in dem Dokument „Angepasste Aufgabenstellung“ [Pro10b] im Detail erläutert.

- **Review 2 am 26. August:**
  Auf dem zweiten Review am 26. August 2010 wurde diese angepasste Aufgabenstellung und die Implementierung des Prototypen präsentiert. Das Vorgehen zum Testen des Systems und die Testergebnisse der, bis zu diesem Zeitpunkt, durchgeführten Tests wurden erläutert.
- **Review 3 am 9. Dezember:**
  Das dritte Review am 9. Dezember 2010 war der Abnahmetest für das Fahrerassistenzsystem. Es wurde das Vorgehen und die Ergebnisse der Integrations- und Systemtests sowie die Ergebnisse der Machbarkeitsstudie präsentiert.



Ist-Zustand:

Tastendrücke werden nicht immer übertragen
Leuchtelemente werden nicht richtig angesteuert
Texturen einiger Schilder werden nicht angezeigt
Wunschgeschwindigkeit von 165 ist möglich
Fahrspurwechselverbot wird falsch angezeigt
Tsim-Leon3 erzeugt keine Timer-Interrupts

Soll-Zustand:

Fahrspurmarkierungsauswertung wird auf dem FPGA ausgeführt
Es gibt ein Konzept zur Ein- und Ausgabe von Daten an die Fahrspurmarkierungsauswertung, die auf dem FPGA ausgeführt wird
Es gibt ein Konzept zur Bestimmung der Ausführungszeiten der Fahrspurmarkierungsauswertung, die auf dem FPGA ausgeführt wird

Das LCA erfüllt die festgelegten Anforderungen der Anforderungsdefinition:
3.1.1.1
3.1.1.2
3.1.1.6
3.1.2.2
3.1.2.3

Das Architektur- und Schnittstellendokument ist fertiggestellt

Der Projektplan ist vollständig überarbeitet
Zeitplanung
Ressourcenmanagement

Vortragsfolien sind fertiggestellt

Teststrecken sind vorhanden

Testdatenbank ist mit den geforderten LCA- und Komm-Tests gefüllt

Abbildung 2.2: Grafisch aufbereiteter Teilausschnitt des Projektplans

3 Analyse und Definition

Dieses Kapitel gibt eine Übersicht über die vorhergehenden Dokumente und deren Entstehungsprozess. Dabei wird zum Einen auf die Anforderungsdefinition und zum Anderen auf das Architektur- und Schnittstellendokument eingegangen.

3.1 Anforderungsdefinition

Das V-Modell (siehe Abschnitt 2.1) sieht eine Anforderungsanalyse vor, die der Findung aller Funktionen, die das Fahrerassistenzsystem erfüllen muss, dient. Diese ist als Anforderungsdefinition [Pro10a] verfasst worden. Dieses Dokument wird in diesem Abschnitt genauer erläutert. Dabei wird sowohl auf die Vorgehensweise beim Verfassen, als auch auf den Inhalt eingegangen.

3.1.1 Vorgehensweise

Das Anforderungsdefinitions-Dokument basiert auf der vorgegebenen Aufgabenstellung (siehe Abschnitt 1.1).

Für das Dokument wurde zunächst eine Systembeschreibung erstellt. Diese gibt das durch die Aufgabenstellung definierte Fahrerassistenzsystem wieder. Dabei wurden vor allem die Teilsysteme für das ACC und das LCA berücksichtigt und für diese bereits die geplante Funktionsweise sowie die notwendigen Bedienelemente festgelegt.

Ein weiterer Teil des Dokumentes beschäftigt sich mit der Simulationsumgebung, welche durch die Aufgabenstellung vorgegeben war. Dazu wurden auch die durch den SILAB-Fahrsimulator gegebenen Bedingungen, zum Beispiel in Bezug auf die Realisierung von Sensoren, erarbeitet. Außerdem wurde die Umwelt definiert, das heißt der Aufbau der Fahrbahn, die Position von Verkehrsschildern und das Verhalten von Fremdfahrzeugen. Hierbei wurden auch einige Einschränkungen der Umwelt getroffen, um die Entwicklung aufgrund der beschränkten Projektzeit und der geringen Teilnehmerzahl zu vereinfachen.

Aufbauend auf dieser Systembeschreibung wurde eine Anforderungsspezifikation festgelegt. Diese legt die funktionalen und nichtfunktionalen Anforderungen an das zu entwickelnde Fahrerassistenzsystem fest. Ebenso wurden Anwendungsfälle entwickelt, die versuchen, alle möglichen Verhaltensweisen des Systems abzudecken.

Aus den Anforderungen und Anwendungsfällen wurden Testfälle erstellt, die versuchen sollen, alle Anforderungen und Anwendungsfälle zu testen. Diese wurden auch zu Testszenarien kombiniert, mit denen alle Testfälle kombiniert abgedeckt werden sollen. Diese Testszenarien wurden als Vorlage für die Tests in der Testdatenbank (siehe Kapitel 5) benutzt.

3.1.2 Überblick über die Anforderungsdefinition

Dieser Abschnitt gibt einen Überblick über die Anforderungsdefinition. Nähere Details können in diesem Dokument [Pro10a] nachgelesen werden.

3.1.2.1 Systembeschreibung

Das zu entwickelnde System ist ein Fahrerassistenzsystem, welches die Funktionen eines ACC und LCA umfasst. Für die grundlegende Funktionalität wurde sich an Systemen orientiert, die bereits auf dem Markt erhältlich sind. Details zur Funktionalität wurden dem Handbuch Fahrerassistenzsysteme [WHW09] entnommen und durch die ISO-Norm 15622 [ANS] (ACC) ergänzt. Zusätzlich sind einige neue Ideen der Projektgruppenteilnehmer mit in das Fahrerassistenzsystem eingeflossen.

ACC

Das ACC sorgt dafür, dass ein Sicherheitsabstand zum Vorderfahrzeug stets eingehalten wird. Dabei wird die zuvor vom Fahrer festgelegte Wunschgeschwindigkeit eingehalten, sofern dies nicht durch ein langsameres, vorausfahrendes Fremdfahrzeug verhindert wird. Ebenso führt eine Geschwindigkeitsbegrenzung, die vom Fahrerassistenzsystem mittels der Sensorik erkannt wurde, dazu, dass die Wunschgeschwindigkeit nicht erreicht werden kann.

LCA

Das LCA unterstützt den Fahrer beim Wechseln von Fahrspuren sowie beim Auf- und Abfahren auf bzw. von der Autobahn. Es greift aber nicht aktiv in die Steuerung des *Ego-Fahrzeugs* ein. Bei aktiviertem LCA kann der Fahrer über Lichtsignale an den beiden Außenspiegeln ablesen, ob ein Fahrspurwechsel bzw. ein Auf- oder Abfahren derzeit möglich ist. Dabei wird immer nur die Fahrspur direkt neben dem Fahrzeug beachtet. Ein grünes Signal bedeutet dabei, dass ein Fahrspurwechsel gefahrlos möglich ist. Ist das Signal rot, so ist ein Fahrspurwechsel derzeit nicht ohne Gefährdung der Verkehrssicherheit möglich oder aufgrund eines, durch die Sensorik erkannten, Überholverbotes nicht erlaubt.

Bedienelemente und Anzeigen

Die Wahl der Wunschgeschwindigkeit erfolgt über Bedienelemente, über die das ACC aktiviert werden kann. Ebenso gibt es Bedienelemente zum Ändern dieser Geschwindigkeit. Eine Geschwindigkeitswahlbedienelementkontrollleuchte signalisiert, dass ein Bedienelement zur Geschwindigkeitswahl aktiviert wurde. Ebenso erscheint die aktuelle Wunschgeschwindigkeit auf einer Anzeige. Durch erneute Wahl eines Geschwindigkeitswahlbedienelementes mit aktiver Geschwindigkeitswahlbedienelementkontrollleuchte kann das ACC wieder deaktiviert werden. Sobald eine andere Geschwindigkeit gewählt wird, ist diese aktuell und die Kontrollleuchte der zuvor gewählten Stufe erlischt. Dabei gibt es ein Bedienelement zur Übernahme der aktuellen Geschwindigkeit sowie drei Bedienelemente mit voreingestellten Geschwindigkeiten. Diese sind mit Standardwerten initialisiert, können aber bei stehendem Fahrzeug angepasst werden. Wird

eine Wunschgeschwindigkeit während der Fahrt angepasst, so wird diese Einstellung nicht gespeichert. Sollte die Geschwindigkeit des Fahrzeuges unter $30km/h$ sinken oder betätigt der Fahrer die Bremse, so wird das ACC deaktiviert.

Das LCA wird über ein Bedienelement aktiviert und deaktiviert, dessen Status durch eine Kontrollleuchte am Bedienelement angezeigt wird.

Als weitere Funktion des Gesamtsystems werden dem Fahrer die aktuell geltende Geschwindigkeitsbegrenzung und ein eventuell geltendes Überholverbot auf einer Anzeige dargestellt.

Simulator

Das entwickelte Fahrerassistenzsystem wird nicht in einem realen Fahrzeug, sondern innerhalb der Fahrsimulatorsoftware „SILAB“ (siehe Abschnitt 4.1) genutzt. Eine schematische Darstellung der Simulationsumgebung sowie deren Anbindung an das Fahrerassistenzsystem und einer Unterteilung in reale und virtuelle Komponenten ist in Abbildung 3.1 zu sehen. Zur Repräsentation des Ego-Fahrzeugs innerhalb der Simulationsumgebung wird ein Mittelklasse-Wagen vom Typ Skoda Octavia verwendet, der über definierte Maße verfügt und als Modell im Simulator vorhanden ist. Das Fahrverhalten dieses Fahrzeugs wurde experimentell mit Hilfe des Simulators ermittelt. Die notwendige Sensorik wurde in realistischer Weise innerhalb des Simulationssystems nachgebaut. Dabei sind die Sensoren perfekt, das heißt es gibt keine verrauschten Sensordaten.

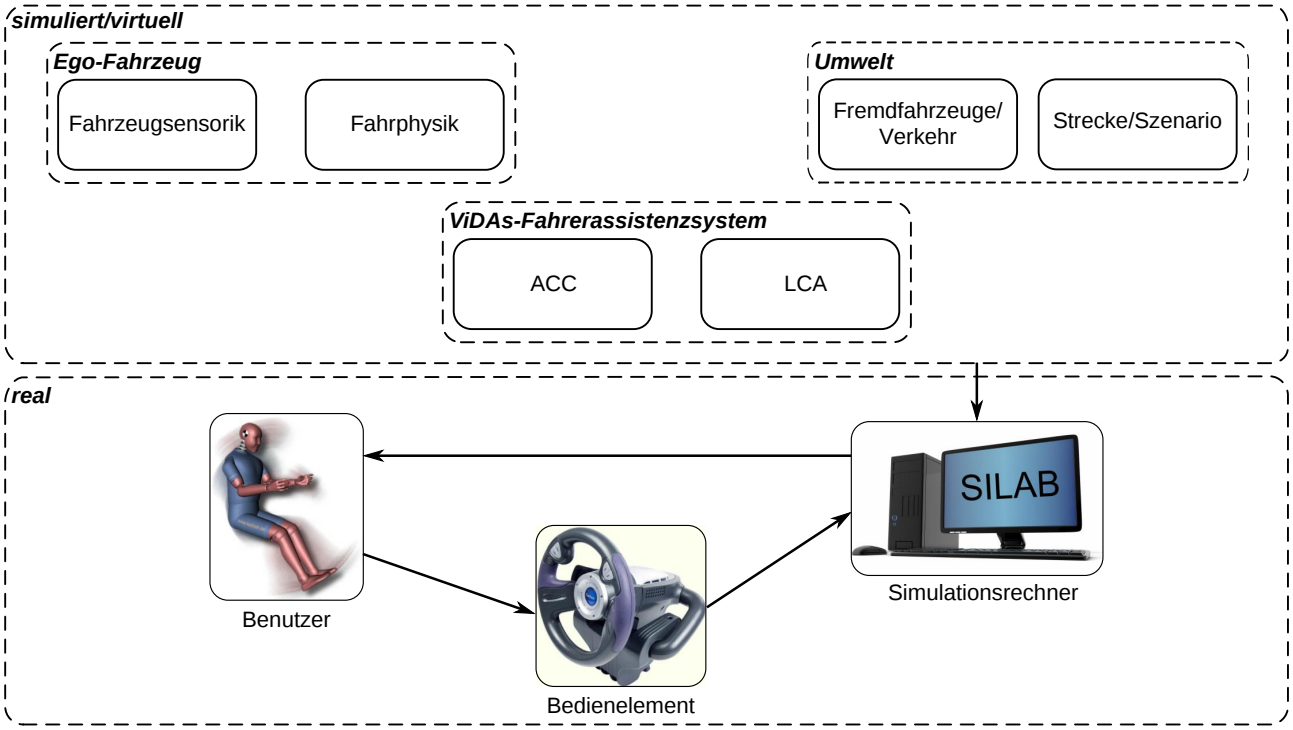


Abbildung 3.1: Schematische Darstellung der Simulationsumgebung und Anbindung an das Fahrerassistenzsystem

3.1.2.2 Umwelt

Bei der als Fahrumgebung simulierten Autobahn handelt es sich um eine deutsche, vierspurige, überregionale Bundesautobahn mit Stand- oder Beschleunigungs- beziehungsweise *Verzögerungsstreifen*, sowie den anschließenden Auf- und Abfahrten. Die Maße hierfür stammen aus den *Richtlinien für die Anlage von Autobahnen* [For08] sowie aus *Hinweise für das Anbringen von Verkehrszeichen und Verkehrseinrichtungen* [Gie90, Gie02].

Bemaßung

In den Richtlinien für die Anlage von Autobahnen [For08] ist der genutzte Autobahntyp unter der Bezeichnung RQ31 in der Entwurfsklasse EKA 1 A zu finden. Abbildung 3.2 gibt einen Überblick über die Fahrspur mit allen Maßangaben.

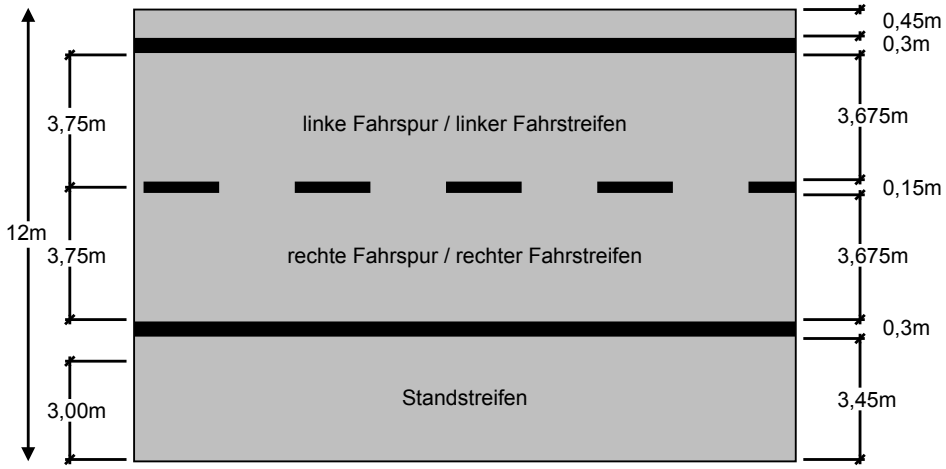


Abbildung 3.2: Breitenangaben zur Autobahn mit Standstreifen

Die Markierungen der Fahrstreifen entsprechen ebenfalls den Richtlinien. Abbildung 3.2 gibt auch hierzu einen Überblick.

Eine Übersicht über die Maße von Auf- und Abfahrten gibt Abbildung 3.3 oder Abbildung 3.4.

Abbildung 3.5 zeigt die Länge des *Beschleunigungsstreifens* an Anschlussstellen. Die Maße sind den Richtlinien für die Anlage von Autobahnen [For08] entnommen. Der Beschleunigungsstreifen wird dort als Einfahrttyp E1 (beziehungsweise Ausfahrtstyp A1) bezeichnet.

Auf der beschriebenen Autobahn kommen nur Kurven vor, die einen Radius von minimal 900 m aufweisen. Die Anwendung dieses Wertes kann Abbildung 3.6 entnommen werden. Dieser Wert entstammt den Richtlinien für die Anlage von Autobahnen [For08] für den verwendeten Autobahntyp RQ31 und der Entwurfsklasse EKA 1 A und ermöglicht Fahrer sowie Fahrerassistenzsystem noch eine ausreichende Sicht nach vorne und hinten, falls sich im Kurveninneren sichtbehindernde Gegenstände befinden.

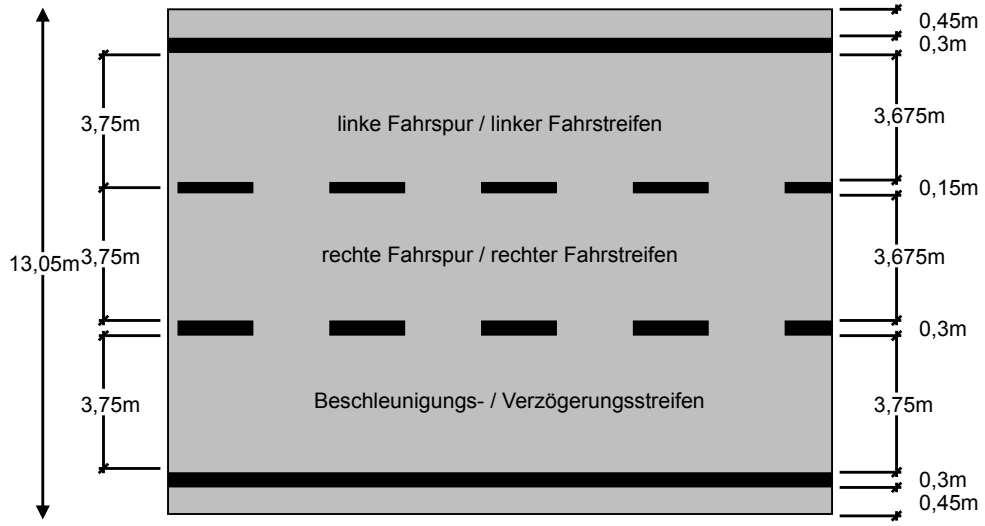


Abbildung 3.3: Breitenangaben zur Autobahn mit Beschleunigungs- oder Verzögerungsstreifen

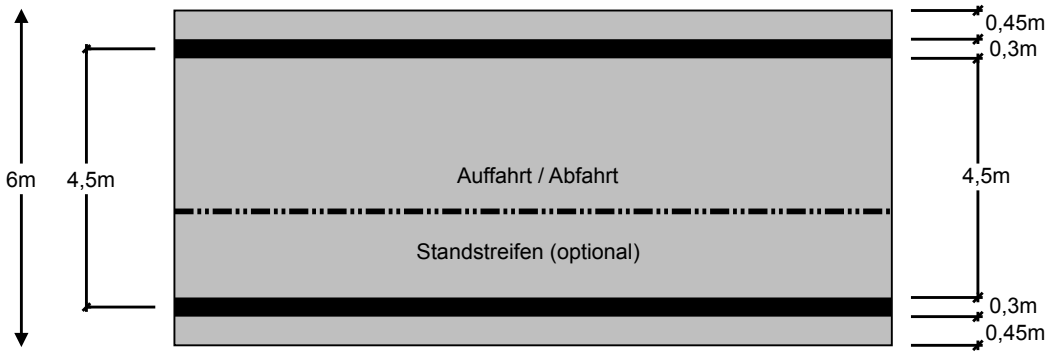


Abbildung 3.4: Breitenangaben zur Auf- beziehungsweise Abfahrt einer Autobahn

Beschilderung

In der betrachteten Umwelt kommen ausschließlich die im Folgenden vorgestellten Verkehrsschilder vor:

Das **Verkehrszeichen 274** erteilt ein Streckenverbot, das verbietet, schneller als mit einer bestimmten Geschwindigkeit zu fahren. Nach dem hier aufgezeigten Schild ist eine Höchstgeschwindigkeit von 60 km/h zulässig. Mögliche Begrenzungen der Höchstgeschwindigkeit werden immer in vollen *Zehnern* angegeben.

Das **Verkehrszeichen 278** hebt die Wirkung des, durch Zeichen 274 aufgestellten, Streckenverbots auf.

Das **Verkehrszeichen 276** kündigt ein Überholverbot für Kraftfahrzeuge aller Art an. Es verbietet, mehrspurige Kraftfahrzeuge und Krafträder mit Beiwagen zu überholen.

Das **Verkehrszeichen 280** hebt die Wirkung des, durch Zeichen 276 aufgestellten, Streckenverbots auf.

Das **Verkehrszeichen 282** hebt die Wirkung aller Streckenverbote auf.

Die Verkehrsschilder können nach den Hinweisen für das Anbringen von Verkehrszeichen und Verkehrseinrichtungen [Gie90, Gie02] seitlich der Fahrbahn und auch über der Fahrbahn angebracht werden. Abbildung 3.7 illustriert dieses. Allerdings war eine Positionierung über der Fahrbahn aufgrund von Beschränkungen des SILAB-Fahrsimulators nicht möglich.

Um die Geschwindigkeit auf der Autobahn herabzusetzen, werden sogenannte Geschwindigkeitsrichter verwendet. So wird die Beschränkung der Höchstgeschwindigkeit von „Keine Geschwindigkeitsbeschränkung“ zunächst mit einer Beschränkung der Geschwindigkeit auf 120 km/h eingeleitet. Anschließend erfolgt die weitere Abstufung auf die gewünschte Geschwindigkeit in 20 km/h-Stufen. Der Abstand zwischen den Schildern dieses Geschwindigkeitsrichters beträgt mindestens 200 m.

Annahmen

Um die Entwicklung des Systems zu vereinfachen, wurden einige Annahmen getroffen. Diese schließen einige Besonderheiten, die auf realen Autobahnen auftreten können, aus. Die folgenden Annahmen werden für das Fahrerassistenzsystem getroffen:

- Keine *Geisterfahrer*. Fahrzeuge, auch das Ego-Fahrzeug, folgen der *Linienführung* der Autobahn, d.h. Fahrzeuge fahren in der für die Fahrbahn vorgegebenen Fahrtrichtung.
- Keine Hindernisse mit einer geringeren Geschwindigkeit als 30 km/h befinden sich auf der Fahrbahn. Ebenfalls keine stehenden Fremdfahrzeuge, wie z.B. an einem Stauende, und auch keine fliegenden Hindernisse.
- Die Maximalgeschwindigkeit des Ego-Fahrzeugs beträgt 160 km/h, die Maximalgeschwindigkeit der Fremdfahrzeuge 200 km/h.
- Auf der Autobahn befindet sich keine Baustelle.
- Es wird nur bei Tageslicht gefahren.
- Auf der Autobahn gibt es weder Steigungen noch Gefälle im Straßenverlauf.
- Es herrschen klare Sichtverhältnisse und es wirken keine Wettereinflüsse auf das System.
- Verkehrsschilder werden nicht von anderen Objekten verdeckt.

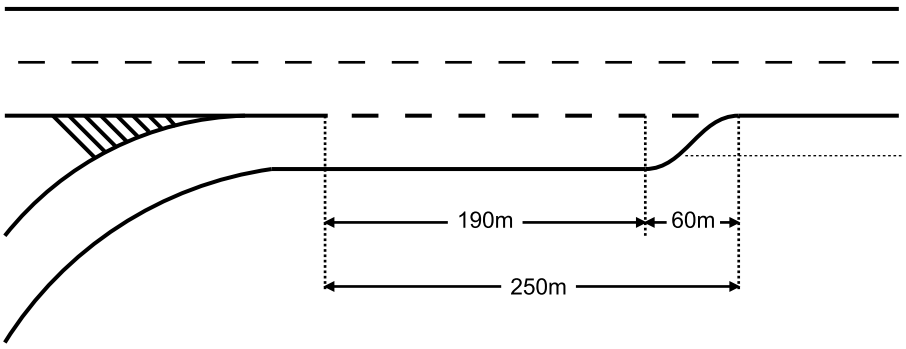


Abbildung 3.5: Längenangaben zu Beschleunigungs- beziehungsweise Verzögerungsstreifen

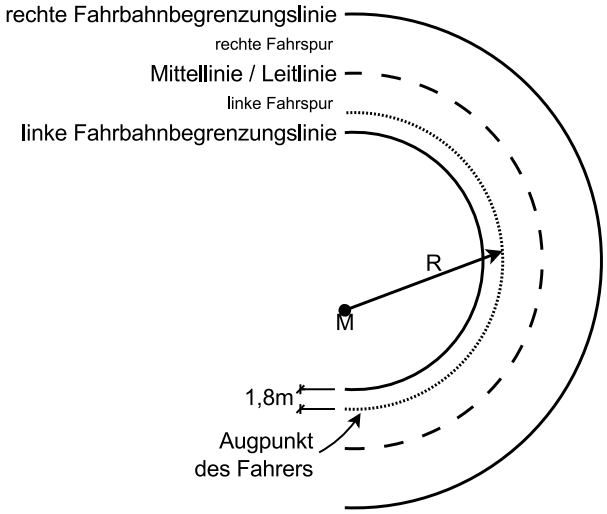


Abbildung 3.6: Angaben zu Kurvenradien

- Markierungen auf und neben der Fahrbahn sind klar erkennbar und sind weder verschmutzt noch abgenutzt. Selbiges gilt auch für die Beschilderung.
- Auf der Fahrbahn befinden sich nur das Ego-Fahrzeug sowie Motorräder, weitere PKW oder LKW.
- Die Fahrbahn ist mit den spezifizierten Markierungen ausgestattet.
- Die Schilderklasse 274 tritt nur in folgenden Ausführungen auf: 80 km/h, 90 km/h, 100 km/h, 110 km/h, 120 km/h und 130 km/h.
- Die Beschilderung der Autobahn ist ortsfest und wie in Abbildung 3.7 spezifiziert angebracht.

3.1.2.3 Anforderungsspezifikation

Aus den erarbeiteten Ergebnissen wurden funktionale und nichtfunktionale Anforderungen entwickelt, die im Kapitel 3 der Anforderungsdefinition [Pro10a] nachzulesen sind und hier nicht

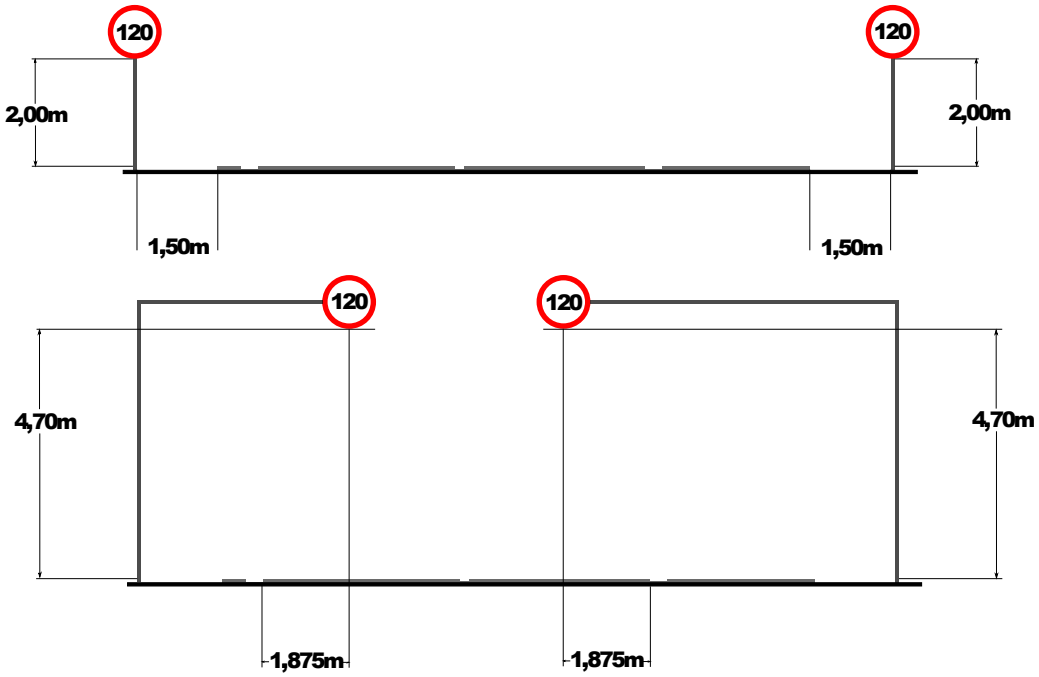


Abbildung 3.7: Schilderpositionierung

erneut aufgezählt werden.

Ebenso sind in diesem Kapitel Anwendungsfälle aufgeführt, die der Verdeutlichung der implementierten Funktionen dienen.

3.1.2.4 Testfälle und Testszenarien

Aus den Anforderungen und den Anwendungsfällen wurden Testfälle entwickelt, die in der Anforderungsdefinition [Pro10a] auf Seite 54 ff nachzulesen sind. Diese Testfälle sollen alle Funktionen und dabei auftretenen Fehler abdecken und prüfen.

Außerdem wurden durch Kombination der Testfälle Testszenarien entwickelt, die im gleichen Kapitel nachzulesen sind. Diese wurden entworfen, um sicher zu stellen, dass alle Funktionen korrekt funktionieren.

3.2 Architektur- und Schnittstellendokument

Das Architektur- und Schnittstellendokument [Pro10c] legt fest, aus welchen Teilkomponenten das Fahrerassistenzsystem aufgebaut ist und wie diese untereinander kommunizieren.

3.2.1 Vorgehensweise

Bereits beim Verfassen der Anforderungsdefinition wurde festgestellt, dass zwei Teilmodule für das Fahrerassistenzsystem entwickelt werden sollen. Diese sind das ACC und das LCA. Für das Architektur- und Schnittstellendokument wurden die Grenzen dieser Module genauer festgelegt. Ebenso wurde ein Obermodul definiert, das diese beiden Teilmodule enthält, der *ViDAs-Kern*.

Auch die Umweltschnittstelle als eigenes Modul wurde bereits in der Anforderungsdefinition beschrieben und im Architektur- und Schnittstellendukument als *ViDAs-Umweltschnittstelle* genauer in seinen Grenzen definiert.

Um die Kommunikation zwischen dem ViDAs-Kern und der ViDAs-Umweltschnittstelle zu realisieren, wurde ein weiteres Modul, das *ViDAs-Komm*-Modul, entwickelt.

Für das ViDAs-Komm-Modul wurden Nachrichten für die Kommunikation definiert, die in einem eigenen Abschnitt dargestellt werden. Über diese Nachrichten werden alle benötigte Daten übertragen.

3.2.2 Überblick über das Architektur- und Schnittstellendokument

Dieser Abschnitt soll einen Überblick über das Architektur- und Schnittstellendokument [Pro10c] vermitteln. Nähere Details können in diesem Dokument nachgelesen werden.

3.2.2.1 Architektur

Dieser Abschnitt beschreibt die aus der Anforderungsdefinition abgeleitete Architektur des ViDAs-Fahrerassistenzsystems (siehe Abbildung 3.8).

Module

Folgende Module wurden identifiziert:

- *ViDAs-Umweltschnittstelle*: In der ViDAs-Umweltschnittstelle ist die benötigte Sensorik des Ego-Fahrzeugs untergebracht. Zudem werden die Eingaben des Fahrers in diesem Modul entgegen genommen und Ausgaben des Systems an den Fahrer weitergegeben. Diese Funktionalität der ViDAs-Umweltschnittstelle ist in dem Fahrsimulator SILAB umgesetzt. Zusammen mit dem ViDAs-Komm-Modul ist die ViDAs-Umweltschnittstelle somit die Schnittstelle des ViDAs-Kerns mit dem Fahrzeug und dem Fahrer sowie der Umwelt des Fahrzeuges.
- *ViDAs-Kern*: Der ViDAs-Kern enthält die Logik und Regler, um die Funktionalitäten des ACC und des LCA zu realisieren sowie einen ViDAs-Kern-Komm-Adapter zur Kommunikation mit der ViDAs-Umweltschnittstelle. Somit bildet der ViDAs-Kern den Hauptteil des Fahrerassistenzsystems.
- *ViDAs-Komm*: Das ViDAs-Komm-Modul stellt die Kommunikation zwischen dem ViDAs-Kern und der ViDAs-Umweltschnittstelle her. Dieses Modul abstrahiert von der physikalischen Realisierung des Kommunikationkanals zwischen den beiden anderen Modulen.

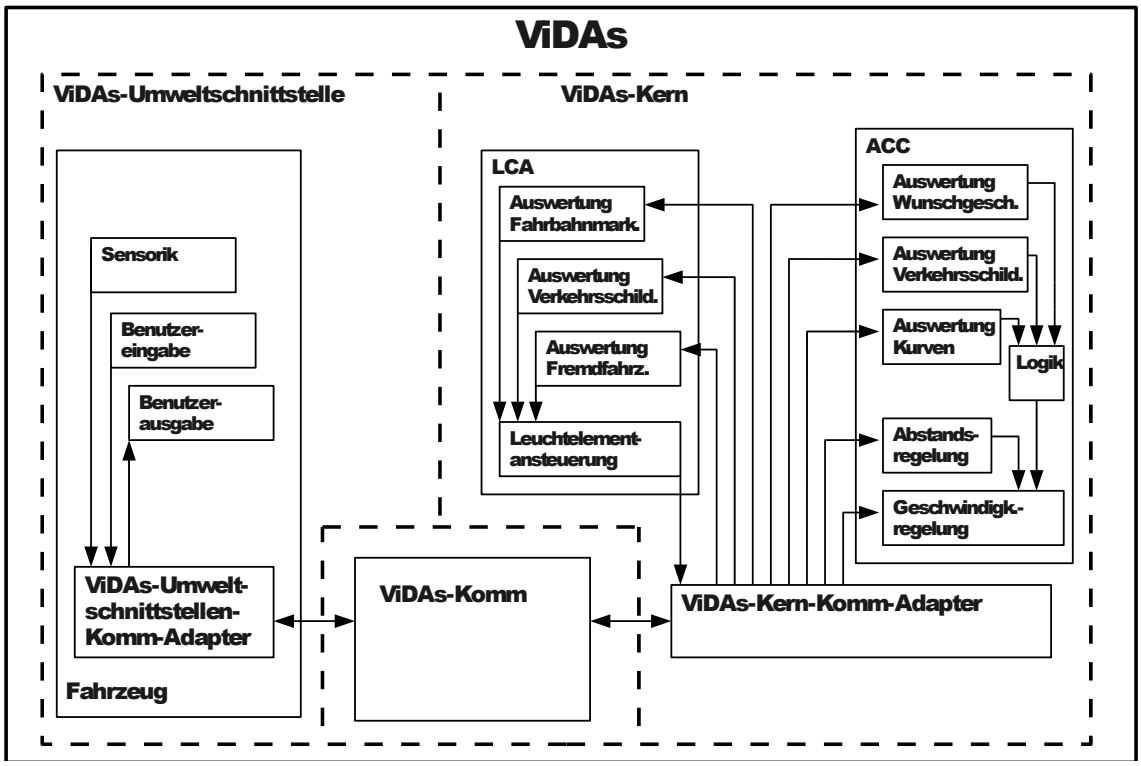


Abbildung 3.8: ViDAs Architektur

ViDAs-Umweltschnittstelle

Das Modul *ViDAs-Umweltschnittstelle* besteht aus weiteren Submodulen. Diese sind:

- *Sensorik*: Diese Komponente ist für die Erfassung der Umgebung des Fahrzeuges zuständig. Sie beinhaltet sämtliche, für das Fahrerassistenzsystem nötige, Sensorik (Lidar-Sensoren, Radar-Sensoren, Kamera, Geschwindigkeitssensor). Das Submodul unterteilt sich in je ein weiteres Submodul für jeden im Fahrzeug verbauten Sensor. Diese Submodule ermitteln anhand von konkreten Werten aus der Fahrdynamik des SILAB-Fahrsimulators oder auf Basis von aus dem Objektmodell des Simulators abgeleiteten Werten (Kamera, Radar-Sensoren, Lidar-Sensoren) die Sensordaten. Die Sensordaten werden an den ViDAs-Umweltschnittstellen-Komm-Adapter übergeben.
- *Benutzereingabe*: Das Submodul dient dazu, Eingaben des Benutzers entgegenzunehmen und zu verarbeiten. Dies sind einerseits Steuersignale für das Fahrzeug selbst, die vom Benutzer entweder per Lenkrad oder Tastatur eingegeben werden. Andererseits sind es auch die Bedienelemente für das ACC und LCA, also Bedienelemente zur Aktivierung der Funktionalitäten, Einstellung der Wunschgeschwindigkeit und Anpassung der Wunschgeschwindigkeit für das ACC. Diese zusätzlich zur Fahrzeugsteuerung notwendigen Bedienelemente werden dabei durch entsprechend belegte Tasten auf der Tastatur des Simulationsrechners realisiert.
- *Benutzerausgabe*: Dieses Submodul übernimmt die Ausgabe aller Informationen des Systems an den Benutzer. Dazu gehören die Anzeigen für den Aktivierungszustand des

Funktionalitäten	LCA	ACC
3.1.1.1	X	
3.1.1.2	X	
3.1.1.3		X
3.1.1.4		X
3.1.1.5		X
3.1.1.6	X	X
3.1.1.7		X
3.1.1.8		X
3.1.2.1		X
3.1.2.2		X
3.1.2.3	X	
3.1.2.4		X

Tabelle 3.3: Verteilung der Funktionalitäten aus der Anforderungsdefinition auf die Submodule des ViDAs-Kerns

ACC und LCA, die Anzeige für die eingestellte Wunschgeschwindigkeit, Anzeigen für Beschränkungen durch Verkehrsschilder (Überholverbot, Geschwindigkeitsbeschränkung) sowie die Statusleuchten des LCA, die anzeigen, ob ein Spurwechsel nach links oder rechts möglich ist. Die anzuzeigenden Informationen erhält das Submodul vom ViDAs-Umweltschnittstellen-Komm-Adpater und gibt sie über den Bildschirm des Simulationsrechners aus.

ViDAs-Kern

Das Modul *ViDAs-Kern* besteht aus den Submodulen *LCA* und *ACC*.

Das LCA wertet Fahrbahnmarkierungen, Fremdfahrzeuge und Ausfahrtsschilder aus und steuert entsprechend der erkannten Hindernisse und Fahrbahnsituationen die Leuchtelemente am Außenspiegel an (siehe Anforderungsdefinition, [Pro10a] Seite 7).

Das ACC wertet Fremdfahrzeuge und deren Geschwindigkeit sowie Geschwindigkeitsbegrenzungsschilder und Kurven aus, um das Fahrzeug entsprechend der Geschwindigkeitsbegrenzung, die durch eines dieser erkannten Hindernisse gegeben ist, und der eingestellten Wunschgeschwindigkeit zu beschleunigen bzw. abzubremsen (siehe Anforderungsdefinition [Pro10a], Seite 6 f).

Die Matrix in Tabelle 3.3 zeigt den Abgleich zwischen den Funktionalitäten aus der Anforderungsdefinition und den Komponenten im ViDAs-Kern-Modul. Die erste Spalte listet die Funktionalitäten auf. Die weiteren Spalten stellen die Komponenten LCA und ACC dar.

ViDAs-Komm

Das ViDAs-Komm-Modul dient der Verbindung des ViDAs-Kerns mit der ViDAs-Umweltschnittstelle.

Diese Verbindung wird auf zwei unterschiedliche Arten realisiert.

1. Die erste Art der Verbindung ist eine Kommunikation per „High Level Architecture“ (HLA). HLA verbindet mehrere Komponenten über eine „Run Time Infrastructure“ (RTI). Zu dieser verbinden sich alle Komponenten, um Daten mit anderen Komponenten austauschen zu können. Dieses ist in Abbildung 3.9 skizziert.

   Durch die Verwendung der HLA-Schnittstelle ist leichteres Debuggen möglich, da alle Daten mitgeschnitten werden können und während der Simulation die Zustandswechsel des Matlab/Simulink-Modells beobachtet werden können.

2. Die zweite Art ist die Kommunikation per CAN-Bus. Dieser Bus wird genutzt, da die Geschwindigkeit gegenüber der HLA-Schnittstelle höher ist und es sich bei dem CAN-Protokoll um einen Standard in der Automobilindustrie handelt. Aufgrund von Zeit- und Ressourcenmangel wird für das Fahrerassistenzsystem kein realer CAN-Bus genutzt, sondern es wurde als Teil der Machbarkeitsstudie ein Konzept zur CAN-Kommunikation entwickelt (siehe Abschnitt 6.5).

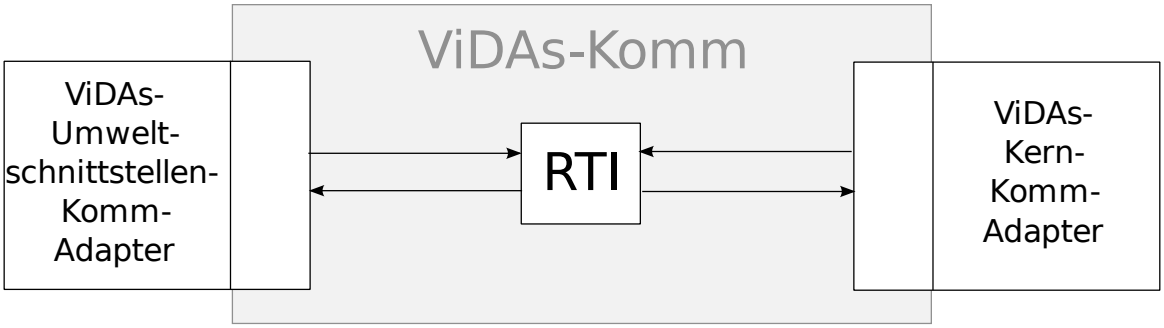


Abbildung 3.9: Funktionsweise der High Level Architecture

Genauere Details zur Funktionsweise des ViDAs-Komm-Moduls finden sich in Abschnitt 1.2 des Architektur- und Schnittstellendukuments [Pro10c].

Der Export des ViDAs-Kerns als *DPU*, wie in Abschnitt 4.3.3 beschrieben, wurde im Architektur- und Schnittstellendokument noch nicht erläutert.

3.2.2.2 Nachrichten

Die benutzen Nachrichten können im Kapitel 2 des Architektur- und Schnittstellendukuments [Pro10c] nachgelesen werden. Sie beginnen stets mit einem 15 Bit-langen Zeitstempel. Die weiteren Felder sind je nach Nachricht unterschiedlich definiert und haben unterschiedliche Längen.

Die maximale Länge einer Nachricht ist dabei inklusive Zeitstempel auf 32 Bit beschränkt, da dies eine Beschränkung der CAN-Implementierung des SILAB-Fahrsimulators darstellt (siehe dazu auch Abschnitt 6.5).

Diese Nachrichten werden für die Kommunikation per HLA-Schnittstelle (siehe Abschnitt 4.3.2) genutzt und sind ebenfalls für die Nutzung über den CAN-Bus (siehe Abschnitt 6.5) vorgesehen.

Feld	**Größe**	**Wert**	**Beschreibung**
ID	11 Bit	01000000101_2	ID für Warnsignal
Zeitstempel	15 Bit		Zeitstempel der Nachricht
Signaltyp	2 Bit	10_2 System deaktiviert 11_2 Starke Bremsung erforderlich 00_2 Warnsignal deaktivieren	Art des Warnsignals

Tabelle 3.5: Beispiel einer Nachricht aus dem Architektur- und Schnittstellendokument [Pro10c]. Dieses Beispiel Beschreibt die Nachricht `Warnsignal`.

Um die Nachrichten für die CAN-Kommunikation nutzen zu können, wurden die IDs bereits mit einer entsprechenden Priorisierung versehen.

In Tabelle 3.5 ist ein Beispiel für eine Nachricht aus dem Architektur- und Schnittstellendokument zu sehen.

4 Implementierung

4.1 SILAB-Fahrsimulator

Zur Realisierung der Projektaufgabe wurde unter anderem der Fahrsimulator *SILAB* benutzt. Diese Fahrsimulations-Software wird vom Würzburger Institut für Verkehrswissenschaften - GmbH [wiv] angeboten und unter Berücksichtigung von Kunden-Feedback und Kundenwünschen ständig weiterentwickelt. Im Rahmen der Projektgruppe wurde die Simulations-Software benutzt um das *Ego-Fahrzeug*, sowie dessen Umwelt (Strecke auf der das Fahrzeug fährt, Fremdfahrzeuge, usw.) zu simulieren. Zum besseren Verständnis der folgenden Kapitel werden daher zunächst einige der grundlegenden Konzepte des SILAB-Fahrsimulators erläutert.

4.1.1 Aufbau des SILAB-Fahrsimulators

Die Simulationssoftware SILAB ist modular aufgebaut. Sie besteht aus einem Hauptprogramm mit grafischer Oberfläche sowie einer Vielzahl einzelner Module, sogenannten „Data Processing Units“ (*DPUs*, siehe Abschnitt 4.1.2) die miteinander verschaltet sind. Jedes einzelne dieser Module übernimmt eine oder auch mehrere Funktionen. Diese Funktionen reichen von einfachen Rechenoperationen mit den Eingangsdaten und dem Anlegen des Ergebnisses an den Ausgang, bis hin zu sehr komplexen Funktionen, wie der Kapselung des Fahrzeugmodells oder der Simulation der Umwelt. Einige dieser Module sind zwingend notwendig für den Betrieb des SILAB-Fahrsimulators, andere hingegen sind optional oder auch für allgemeinere Aufgaben mehrfach verwendbar.

Die konkrete Verschaltung der einzelnen Teilmodule wird in einer Konfigurationsdatei definiert. Hier wird unter Anderem angegeben, welche Module im System existieren und auf welchem Rechner sie ausgeführt werden (die Simulation kann auch verteilt ausgeführt werden). Darüber hinaus kann eine Prioritätszahl angegeben werden, mit der die Ausführungsreihenfolge der DPUs beeinflusst wird. Zusätzlich wird festgelegt, welche Ein- und Ausgänge der einzelnen Module miteinander verbunden sind. Neben den Informationen über die DPUs werden in der Konfigurationsdatei der Aufbau der Strecke, Fremdfahrzeuge und deren Verhalten sowie sonstige Simulationsparameter definiert (siehe Abbildung 4.1). Es wird also für jedes Szenario eine separate Konfigurationsdatei benötigt. Die Konfigurationsdatei, welche vom Benutzer ausgewählt wurde, wird vom Hauptprogramm geladen, ausgewertet, die verwendeten DPUs geladen, das Szenario sowie Fremdfahrzeuge im Simulator erzeugt, initialisiert und die Simulation vorbereitet, so dass sie auf Knopfdruck vom Benutzer gestartet werden kann.

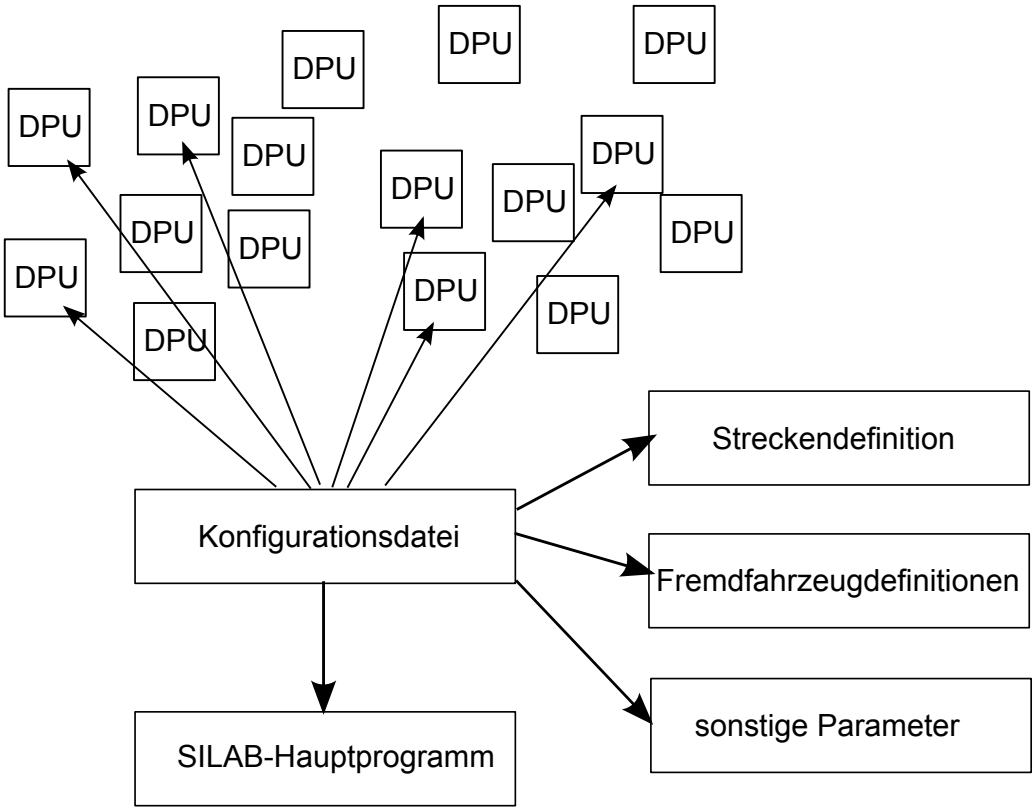


Abbildung 4.1: Aufbau des SILAB-Fahrsimulators

4.1.2 Aufbau und Funktionsweise einer DPU

Eine DPU ist ein einzelnes Verarbeitungsmodul innerhalb der Simulationssoftware SILAB. Sie liegt meist in kompilierter Form als *Dynamic-Link Library (DLL)* vor (für Ausnahmen hiervon siehe Abschnitt 4.1.3) und kann wie eine Blackbox, die über eine nach außen sichtbare Schnittstelle mit Ein- und Ausgängen verfügt, betrachtet werden. Die eigentliche Datenverarbeitung findet im Inneren der DPU statt und ist in der Regel nach außen hin nicht sichtbar.

Unabhängig davon auf welche Weise eine DPU implementiert wird (siehe Abschnitt 4.1.3), muss sie mindestens einen Basissatz von Funktionen implementieren, die dann zu verschiedenen Zeitpunkten vom Simulator aufgerufen werden. Diese Funktionen sind in Abbildung 4.2 im Inneren der DPU dargestellt.

Die Funktion `initialize` wird einmalig aufgerufen, sobald die DPU in den Simulator geladen wird. In dieser Funktion müssen die Ein- und Ausgänge der DPU spezifiziert werden.

Die Funktion `prepare` wird in der Setup-Phase der Simulation ausgeführt, also nachdem die vom Benutzer ausgewählte Konfigurationsdatei bereits geladen wurde, aber noch vor dem unmittelbaren Start der Simulation. Hier können zum Beispiel globale Variablen initialisiert oder Log-Files geöffnet werden.

Die Funktion `start` wird einmalig aufgerufen, sobald nach erfolgreicher Setup-Phase die Simulation über das SILAB-Hauptprogramm gestartet wird. Hier können beispielsweise Aktionen,

wie das Loggen der Startzeit der Simulation geschehen.

Die zentrale Funktion einer DPU ist die Funktion `trigger`, diese wird vom Simulator in regelmäßigen Zeitabständen (je nach eingestellter Taktrate für den Simulator in der Konfigurationsdatei) aufgerufen und als Parameter wird die aktuell verstrichene Zeit seit Simulationsstart in Millisekunden übergeben. Innerhalb dieser Funktion geschieht die eigentliche Datenverarbeitung und es werden neue Werte für die Ausgänge der DPU berechnet und angelegt.

Die Funktion `stop` bildet das Pendant zur Funktion start, sie wird einmalig beim Beenden der Simulation aufgerufen.

Als letztes die Methode `release`: Diese wird aufgerufen, unmittelbar bevor die DPU aus dem Simulator entladen wird. Hier können zum Beispiel noch geöffnete Log-Files geschlossen werden.

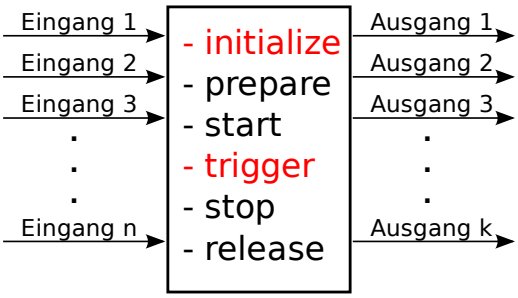


Abbildung 4.2: Grundlegender Aufbau einer DPU

4.1.3 Erweiterung des Simulators

Neben der Möglichkeit über die Konfigurationsdateien und den bereits mit SILAB ausgelieferten DPUs eine Vielzahl verschiedener Simulatorkonfigurationen zu erstellen, besteht noch die Möglichkeit, den Funktionsumfang des Simulators durch eigene DPUs zu erweitern. Für die Erweiterung des Fahrsimulators stehen dem Benutzer verschiedene Wege zur Auswahl.

Zunächst gibt es da die Möglichkeit, mehrere der bereits existierenden primitiven DPUs so miteinander zu verschalten, dass dabei das gewünschte Verhalten implementiert wird, zum Beispiel eine mathematische Funktion oder ein einfacher Regler.

Für Aufgaben, die sich nicht durch Verschaltung bereits existierender DPUs realisieren lassen, können neue DPUs erstellt werden. Hier gibt es drei Möglichkeiten wie diese vom Benutzer implementiert werden können:

Entwicklung einer DPU in Ruby

Der Benutzer kann eine DPU in der Skriptsprache Ruby [rub] schreiben. Es muss dabei darauf geachtet werden, dass die in Abschnitt 4.1.2 genannte Schnittstelle an vordefinierten Funktionen eingehalten wird. Darüber hinaus müssen neben dem eigentlichen Programmcode im Kopf des Skripts auch noch die Ein- und Ausgänge, die der DPU zur Verfügung stehen sollen, deklariert werden. Das Ruby Skript wird dann von einem in die Simulationssoftware eingebetteten Ruby-Interpreter zur Laufzeit des Simulators ausgeführt.

Vorteil dieser Methode ist, dass so schnell und auf einfache Weise DPUs erzeugt werden können, jedoch zu Lasten der Ausführungsgeschwindigkeit, da der Code nicht in kompilierter Form vorliegt, sondern zur Laufzeit interpretiert wird. In der Konfigurationsdatei werden in Ruby geschriebene DPUs über eine Wrapper-DPU (DPURubyScript) eingebunden, sie werden also programmintern genau wie kompilierte DPUs behandelt.

In Ruby geschriebene DPUs werden auch als *Ruby Programming Unit (RPU)* bezeichnet. Bis auf die Tatsache, dass die DPU in Ruby geschrieben wurde, hat diese Bezeichnung allerdings keine weitere Aussage.

Entwicklung einer DPU in C++

Des Weiteren können DPUs in C++ programmiert werden. Ein Wizard zum Erzeugen eines Projektes, welches bereits die notwendige Schnittstelle einer DPU implementiert (siehe 4.1.2) liegt der Simulationssoftware bei und hilft bei der schnellen Erzeugung des DPU Grundgerüstes. Aus dem C++ Projekt der DPU wird dann letztendlich eine DLL kompiliert, welche in den SILAB-Fahrsimulator eingebunden werden kann.

Vorteil dieser Variante gegenüber in Ruby geschriebenen DPUs ist, dass die Ausführungsgeschwindigkeit deutlich höher ist, da der Code bereits in kompilierter Form vorliegt. Darüber hinaus lassen sich in C++ auch Aufgaben lösen, welche die Möglichkeiten der Skriptsprache Ruby übersteigen. Allerdings muss die DPU bei jeder Änderung im Code neu kompiliert werden, dies entfällt bei der Ruby-DPU.

Generierung einer DPU aus einem Matlab-Modell

Die dritte Möglichkeit ist das Erzeugen einer DPU aus einem Matlab-Modell. Hierzu kann die gewünschte Funktionalität samt Ein- und Ausgängen direkt in Matlab modelliert werden, wobei einige Einschränkungen bei der Modellierung beachtet werden müssen, Details hierzu sind [Wür09] zu entnehmen. Aus dem Matlab-Modell kann mit einem speziellen Target C++-Code generiert werden, aus dem wiederum eine DLL kompiliert wird, die in den SILAB-Fahrsimulator eingebunden werden kann.

Der Vorteil dieser Variante liegt darin, dass die Funktionalität mittels Matlab grafisch modelliert werden kann, was üblicherweise intuitiver ist, als direkt den Programmcode zu erzeugen. Außerdem kann das Modell bereits vor der Überführung in eine DLL getestet und gedebugt werden, was für kürzere Test- und Debugzyklen sorgt. Besonders für die Entwicklung von Reglern ist diese Variante sehr gut geeignet, da hier auf die Möglichkeiten von Matlab/Simulink zum Reglerentwurf zurückgegriffen werden kann. Bei der Ausführungsgeschwindigkeit ist diese Variante mit der direkt in C++ geschriebenen DPU identisch, da bei beiden Varianten am Ende eine kompilierte DLL erzeugt wird.

4.2 ViDAs-Umweltschnittstelle

Die ViDAs-Umweltschnittstelle übernimmt die Anbindung des Fahrerassistenzsystems an die Umwelt. Sie ist, wie in Abbildung 4.3 zu sehen, in drei Teilsysteme gegliedert, die Benutzereingabe, Benutzerausgabe und Sensorik. Benutzerein- und ausgabe zusammen ergeben die

Benutzerschnittstelle des Fahrerassistenzsystems. Jedes dieser Teilsysteme besteht dabei aus mehreren DPUs(siehe Abschnitt 4.1.2), welche in Aufbau und Funktion in den folgenden Abschnitten genauer dargestellt werden.

Der Begriff Umwelt hat dabei im Zusammenhang mit der ViDAs-Umweltschnittstelle je nach Kontext eine unterschiedliche Bedeutung. Im Falle der Benutzerein- und ausgabe entspricht Umwelt dem Fahrer, der das System bedient oder über dessen Zustand informiert wird. Im Zusammenhang der Sensorik ist die Umwelt der Erfassungsbereich der Sensorik, dieser schließt das Ego-Fahrzeug, sowie die Strecke auf der sich dieses bewegt, die auf der Strecke befindlichen Verkehrszeichen und Fahrbahnmarkierungen, sowie Fremdfahrzeuge ein.

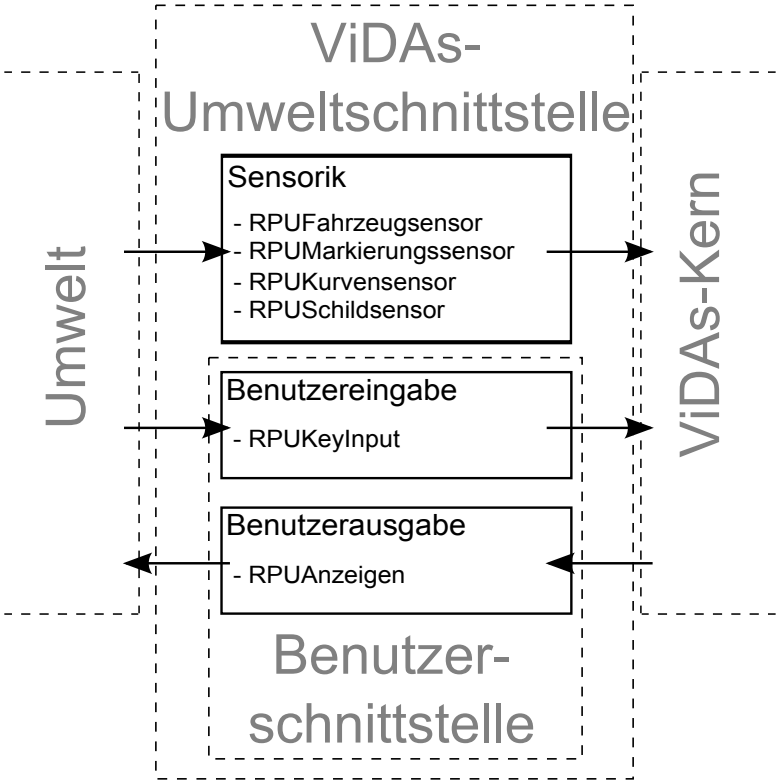


Abbildung 4.3: Übersicht der ViDAs-Umweltschnittstelle mit Teilsystemen und dazugehörigen DPUs

4.2.1 Sensorik

Das Sensorik-Teilsystem besteht, wie in Abbildung 4.4 skizziert, aus vier Radar- und zwei Lidar-Sensoren sowie einer Kamera. Die Funktionaliät der verschiedenen Sensoren ist in [Pro10a] genauer beschrieben. Die genannten Sensoren werden im SILAB-Fahrsimulator durch mehrere DPUs realisiert, die verschiedene Teilfunktionen übernehmen.

Es handelt sich bei keinem der Sensoren um einen realen Sensor. Stattdessen wurde in den Sensorik-DPUs das vereinfachte Verhalten und der Erfassungsbereich der realen Sensoren mit

Mitteln des SILAB-Fahrsimulators nachgebaut. Mit vereinfacht ist hier gemeint, dass Störeinflüsse, die auf einen realen Sensor wirken würden, nicht betrachtet werden, sondern von einer idealen Umwelt ausgegangen wird. Es treten also keine Effekte, wie zum Beispiel Sensorrauschen auf. Es wäre jedoch eine denkbare Erweiterung der Sensorik derartige Effekte einzubauen, um das System unter möglichst realistischen Bedingungen zu betreiben.

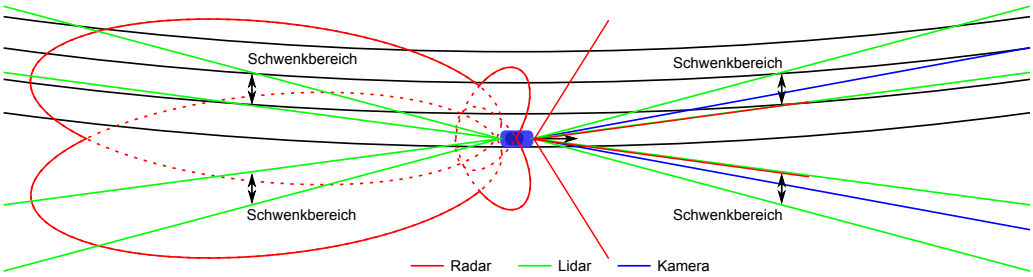


Abbildung 4.4: Am Fahrzeug montierte Sensorik

4.2.1.1 Radar- und Lidarsensoren

Die am Fahrzeug angebrachten Radar- und Lidarsensoren zur Erkennung von Fremdfahrzeugen werden durch das Modul `RPUFahrzeugsensor` realisiert. Die Ausgänge der DPU sind in Abbildung 4.5 zu sehen und entprechen dabei den Werten Abstand (in Meter) sowie Geschwindigkeit (in km/h) der sechs relevanten Fremdfahrzeuge (in Abbildung 4.6 als Nummer (2) bis (7) dargestellt) im Sensorbereich. Die Zahlen hinter den Ausgängen der DPU bilden dabei eine Zuordnung der Signale zu den in Abbildung 4.6 dargestellten und nummerierten relevanten Fremdfahrzeugen. Fahrzeuge außerhalb des Sensorbereiches werden nicht beachtet, auch wenn das mit dem Objektmodell des Fahrsimulators möglich wäre.

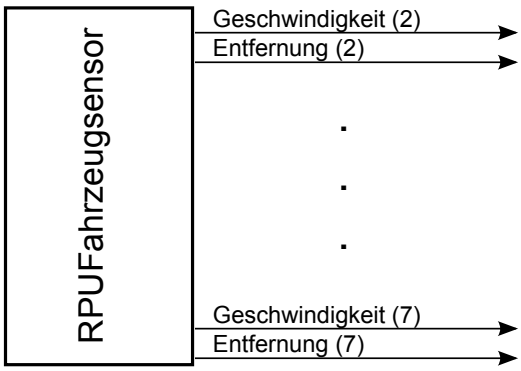


Abbildung 4.5: Ausgänge der RPUFahrzeugsensor

Die Arbeitsweise der `RPUFahrzeugsensor` ist wie folgt: Zunächst werden zwei Flächen um das Ego-Fahrzeug definiert, die eine Vereinfachung des vorderen und hinteren Sensorbereiches aus der Anforderungsdefinition darstellen. Dies wird in Abbildung 4.6, nicht maßstabsgetreu, durch die eliptischen Flächen angedeutet. Jetzt werden aus dem Objektmodell des Simulators zunächst

alle Fremdfahrzeuge ausgewählt, welche sich innerhalb der Sensorflächen befinden. Aus dieser Menge an Fremdfahrzeugen werden nun jene Fremdfahrzeuge entfernt, die nicht zu den relevanten Fremdfahrzeugen gehören, so dass am Ende maximal sechs Fremdfahrzeuge übrig bleiben, sofern sich an jeder in Abbildung 4.6 gezeigten, relevanten Position ein Fremdfahrzeug im Sensorbereich befindet. Für die verbliebenen relevanten Fremdfahrzeuge wird Geschwindigkeit und relative Entfernung zum Ego-Fahrzeug bestimmt und mit diesen Werten die entsprechenden Ausgänge der DPU beschaltet.

Dieser Vorgang wiederholt sich bei jedem Aufruf der Trigger-Methode der DPU, so dass an den Ausgängen immer aktuelle Werte über die Fremdfahrzeuge anliegen. Sollte an einer der relevanten Positionen kein Fremdfahrzeug im Sensorbereich sein, so werden die Ausgänge für diese Position auf den Wert '0' gesetzt. Die Entfernung zu Fremdfahrzeugen wird dabei von hinterer Stoßstange des Ego-Fahrzeugs zur vorderen Stoßstange des Fremdfahrzeugs gemessen, sofern es sich neben oder hinter dem Ego-Fahrzeug befindet. Befindet sich das Fremdfahrzeug dagegen vor dem Ego-Fahrzeug so wird die Entfernung von vorderer Stoßstange des Ego-Fahrzeugs bis zur hinteren Stoßstange des Fremdfahrzeugs verwendet.

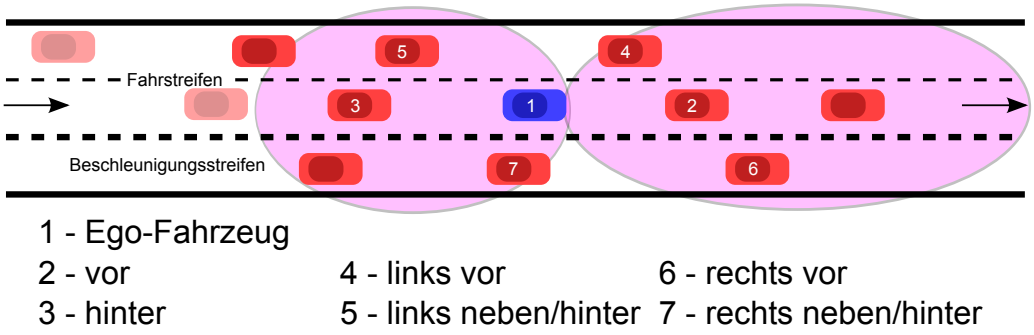


Abbildung 4.6: Relevante Fremdfahrzeuge im Sensorbereich

4.2.1.2 Kamera

Ursprünglich war geplant, neben der Fahrzeugsensorik noch eine Kamera mit nachgeschalteter Bildverarbeitung zu verwenden, um Kurven und Straßenschilder sowie Fahrbahnmarkierungen zu erkennen, um daraus abzuleiten, auf welcher Fahrspur sich das Ego-Fahrzeug befindet. Im Zuge der angepassten Aufgabenstellung [Pro10b] fiel die Ermittlung dieser benötigten Umgebungsdaten aus Bilddaten jedoch weg und die benötigten Informationen sollten direkt aus dem Fahrsimulator, bzw. dessen Objektmodell gewonnen werden.

Aus dieser Anpassung resultieren drei voneinander unabhängige Module, die je eine der Teilfunktionalitäten (Schildererkennung, Kurvenerkennung und Markierungserkennung) realisieren. Wichtig dabei war, dass die Ausgänge dieser Module mit denen der ursprünglich geplanten Bildverarbeitung übereinstimmen, so dass eventuell in einem weiteren Entwicklungsschritt die drei Module wieder durch eine Kamera und Bildverarbeitung ersetzt werden können. Die drei entstandenen Module werden im weiteren in Aufbau und Funktion genauer beschrieben.

RPUSchildsensor

Das Modul `RPUSchildsensor` dient dazu, Straßenschilder, die links und rechts der Fahrbahn angebracht sind, zu erkennen und auszuwerten. Die Ausgänge der DPU sind in Abbildung 4.7 dargestellt. Dies sind je ein Ausgang für die ID des erkannten Schildes (siehe [Pro10c] S. 18 f.) auf der linken und rechten Fahrbahnseite, sowie ein Ausgang für die Entfernung des Ego-Fahrzeuges zu diesen Schildern.

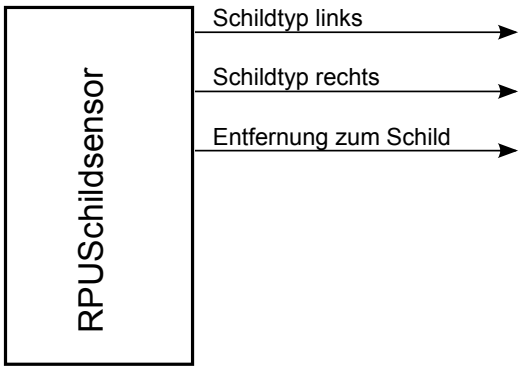


Abbildung 4.7: Ausgänge der RPUSchildsensor

Im Folgenden wird die Funktionsweise der DPU beschrieben: Es wird zunächst wie beim Fahrzeugsensor ein Abfragebereich definiert, welcher die Grenzen des Sichtbereiches der Kamera modelliert (200 Meter nach vorne, je 32 Meter nach links und 32 Meter nach rechts) [Pro10a]. In der Trigger-Methode werden nun bei jedem Aufruf die Objekte innerhalb dieses Abfragebereiches durchlaufen und geprüft, ob es sich dabei um ein Verkehrsschild handelt. Auf diese Weise werden die dem Ego-Fahrzeug am nächsten stehenden Schilder rechts und (sofern vorhanden) links der Fahrbahn ermittelt. Von diesen beiden Schildern wird nun der Schildtyp ermittelt und geprüft, ob dieser Teil der, in der Anforderungsdefinition [Pro10a] beschriebenen, relevanten Schilder für das System ist. Falls ja, wird die exakte Entfernung zum Ego-Fahrzeug ermittelt und am Ausgang angelegt. Der Schildtyp für das rechte und linke Schild wird in die zugehörige ID kodiert und am jeweiligen Ausgang angelegt.

RPUKurvensensor

Der Kurvensensor dient dazu festzustellen, ob sich vor dem Ego-Fahrzeug eine Kurve befindet, wie weit diese, bzw. ihre stärkste Krümmung noch vom Ego-Fahrzeug entfernt ist, in welche Richtung diese verläuft und wie stark ihre Krümmung ist (siehe Abbildung 4.8). Sie stellt dafür entsprechende Ausgänge zur Verfügung (siehe Abbildung 4.9).

Die Ermittlung der neuen Ausgabewerte erfolgt wie auch bei den anderen DPUs in der Trigger-Methode. Informationen über die Strecke lassen sich aus dem Objektmodell des SILAB-Fahrsimulators relativ zur Position des Ego-Fahrzeugs auslesen. Jede Kurve auf der Autobahn verfügt über eine festgelegte maximale Krümmung. Eine Kurve besteht nun immer aus drei Teilstücken, am Kurvenanfang und Kurvenende jeweils einem sogenannten Klothoiden (siehe Kapitel Umwelt in [Pro10a]) und dazwischen einem Teilstück mit der für die Kurve festgelegten Krümmung.

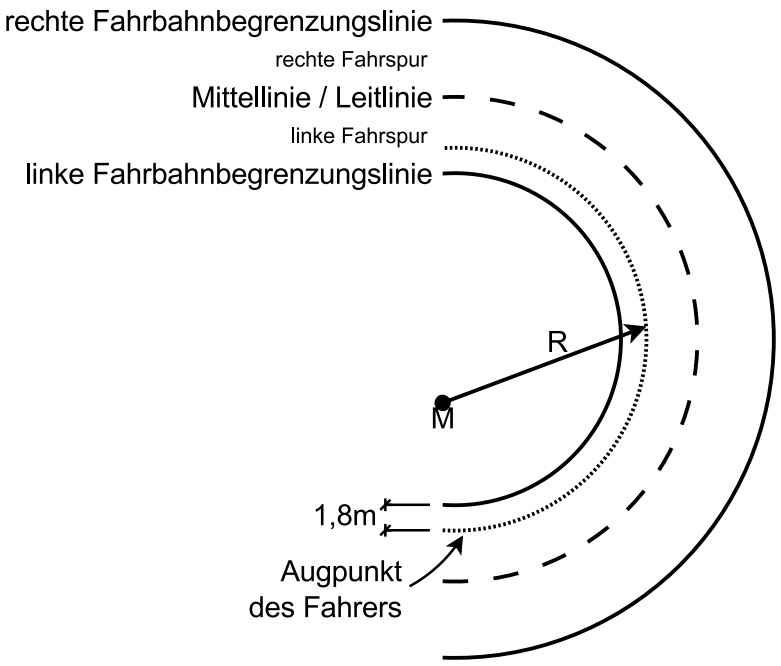


Abbildung 4.8: Angaben zu Kurvenradien

Die Klothoiden dienen dazu, die Krümmung langsam an die maximale Krümmung heranzuführen bzw. von dieser wieder auf eine Gerade ohne Krümmung zurückzuführen, so dass es einen sanften Übergang von Gerade zu Kurve und wieder zurück zur Geraden gibt. In der Triggermethode wird der zu prüfende Fahrbahnabschnitt in einer Schleife langsam immer weiter vom Ego-Fahrzeug weggeschoben, bis maximal 200 Meter vor das Fahrzeug (maximale Sichtweite der Kamera). Befindet sich in diesem Bereich keine Kurve, so wird für Krümmung und Entfernung jeweils der Wert '0' ausgegeben. Wird innerhalb der 200 Metern eine Kurve erkannt, so wird die Schleife so lange weiter durchlaufen, bis entweder der Beginn des konstanten Teilstücks der Kurve gefunden oder aber 200 Meter erreicht werden.

Im letzteren Fall wird am Ausgang eine Entfernung von 200 Metern und die Krümmung der Kurve in 200 Metern angegeben. Sollte beim Durchlaufen der Schleife jedoch der Bereich mit der konstanten Krümmung erreicht werden, so bricht die Schleife ab. Als Entfernung wird die ermittelte Entfernung zum Beginn des konstanten Abschnitts sowie die entsprechende Krümmung angelegt. Dies ist der für das Assistenzsystem relevante Wert, da bis zu diesem Punkt die Geschwindigkeit des Ego-Fahrzeugs eventuell vom ACC gedrosselt werden muss, um die Grenzen für die seitliche Beschleunigung des Ego-Fahrzeugs nicht zu überschreiten. Erreicht das Ego-Fahrzeug den Bereich mit der konstanten Krümmung, so wird so lange eine Entfernung von 0 Metern sowie die maximale Kurvenkrümmung am Ausgang angelegt, bis das Ego-Fahrzeug den Klothoiden am Ausgang der Kurve erreicht. Ab dort wird dann bis zum Kurvenende weiterhin eine Distanz von 0 Metern geliefert, jedoch der aktuell an der Position des Ego-Fahrzeugs herrschende Krümmungswert.

Nach Verlassen der Kurve haben wieder beide Ausgänge den Wert '0'. Der Ausgang für die Richtung der Kurve wird dabei jeweils entsprechend dem Vorzeichen der Kurvenkrümmung

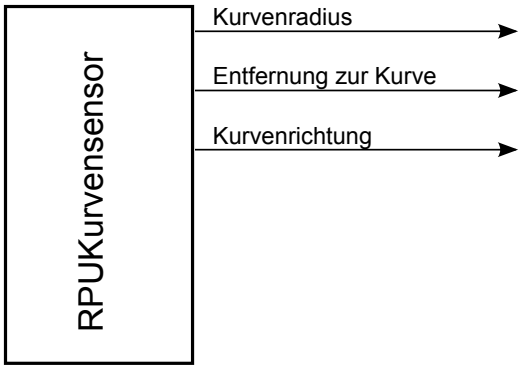


Abbildung 4.9: Ausgänge der RPUKurvensensor

bestimmt und auf den passenden Wert für eine Links- bzw. Rechtskurve gesetzt wie in [Pro10c] Seite 18 ff. definiert.

RPUMarkierungssensor

Diese DPU dient dazu, die Fahrspurmarkierungen links und rechts der Spur auf der sich das Ego-Fahrzeug befindet zu ermitteln. Aus diesen Informationen kann der ViDAs-Kern ableiten, auf welcher Fahrspur sich das Ego-Fahrzeug befindet und was für eine Art von Fahrspur rechts bzw. links neben dem Ego-Fahrzeug ist. Die Ausgänge der DPU sind in Abbildung 4.10 dargestellt. Dies sind je ein Ausgang für die ID der Spurmarkierung (kodiert gemäß [Pro10c] Seite 18 ff.) links und rechts vom Ego-Fahrzeug.

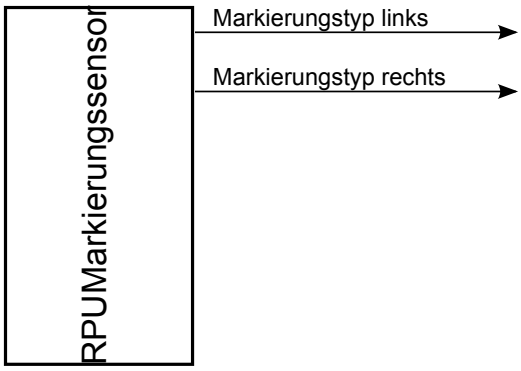


Abbildung 4.10: Ausgänge der RPUMarkierungssensor

Da die Spurmarkierungen keine eigenen Objekte sind, können sie auch nicht über das Objektmodell des Simulators abgefragt werden. Daher musste eine andere Lösung für diese Aufgabe gefunden werden. Dem Simulator ist bekannt, auf welcher Fahrspur sowie auf was für einem Streckenabschnitt sich das Ego-Fahrzeug derzeit befindet. Daher können diese Informationen in die benötigten Fahrspurmarkierungen überführt werden. Es wird geprüft, auf welcher Spur sich das Fahrzeug befindet und ob es sich neben einer Auffahrt, Abfahrt oder auf freier Strecke befindet und in Abhängigkeit davon die passende ID für die Fahrspurmarkierung links und rechts auf die Ausgänge der DPU gelegt.

4.2.2 Benutzerschnittstelle

Die Benutzerschnittstelle bildet die Anbindung des Fahrerassistenzsystems zum Benutzer. Sie besteht aus zwei Teilsystemen: Der Benutzereingabe, welche Eingaben des Benutzers entgegennimmt, auswertet und die aufbereiteten Eingabedaten an den ViDAs-Kern weiterleitet. Daneben die Benutzerausgabe, sie stellt Ausgaben des ViDAs-Kerns visuell, durch Leuchtelemente und Displays, für den Benutzer dar, so dass dieser über den aktuellen Systemzustand informiert ist. Eine mögliche Integration der Benutzerschnittstelle in ein reales Fahrzeug ist in Abbildung 4.11 skizziert. Auf die, in der Skizze dargestellten, Bedienelemente, Anzeigen und Leuchtelemente wird im Folgenden näher eingegangen.

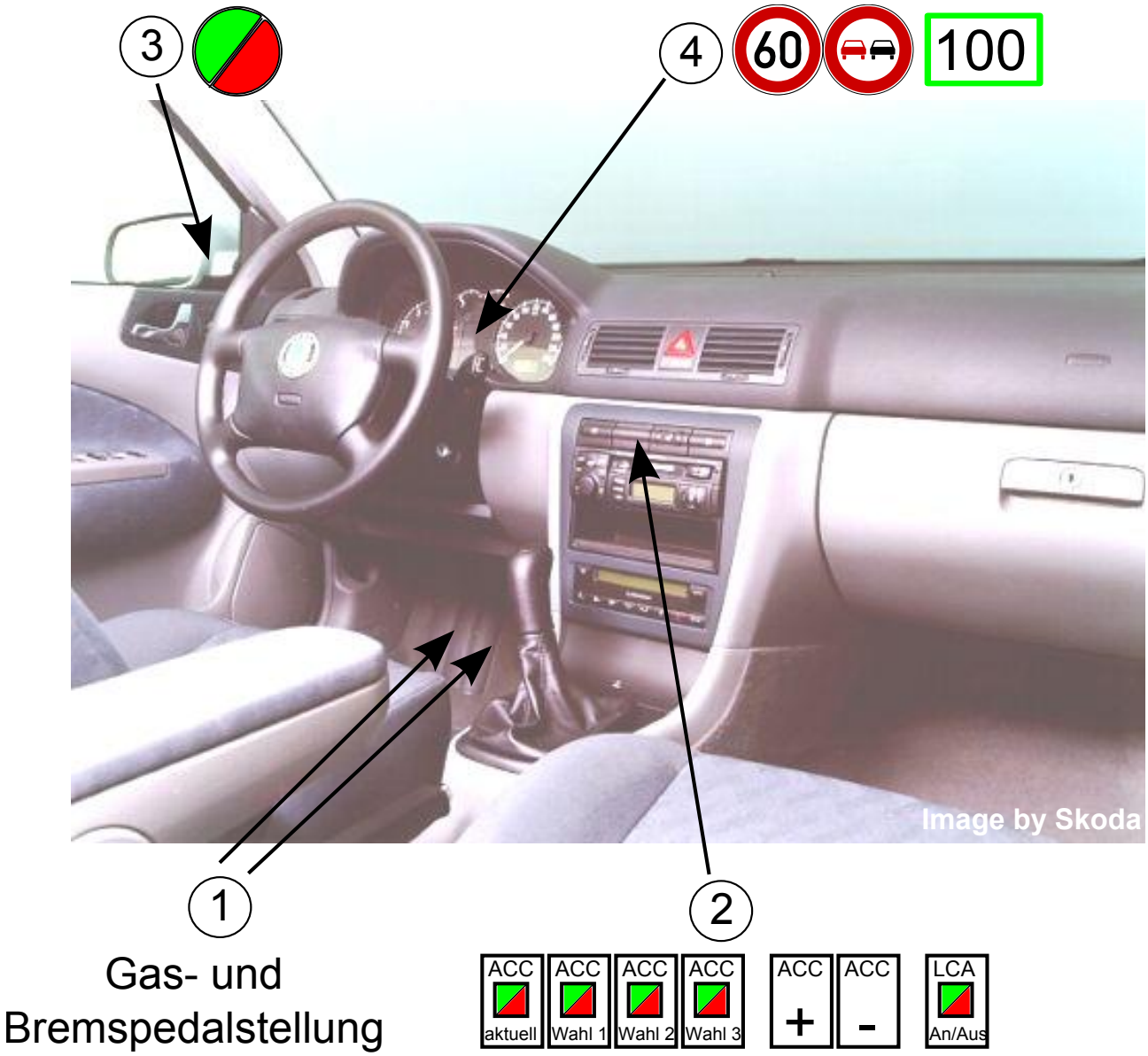


Abbildung 4.11: Übersicht der Benutzerschnittstelle

4.2.2.1 Benutzereingabe

Für die Bedienung des Fahrerassistenzsystems benötigt der Benutzer Bedienelemente. Das System muss für einige Bedienelemente zudem in der Lage sein zu unterscheiden, ob sie nur kurz, oder für länger als 3 Sekunden betätigt werden. Dies wird für die Programmierung der Geschwindigkeitswahlbedienelemente des ACC benötigt, welche in Abschnitt 4.4.1.1 beschrieben wird. Diese Bedienelemente sind in Abbildung 4.11 als Nummer (2) skizziert. Bei der für das Projekt verwendeten Simulator-Konfiguration wurden diese Bedienelemente auf die Funktionstasten der PC-Tastatur am Simulationsrechner gemapped (siehe Tabelle 4.1).

Taste	**Belegung**
F1	ACC Aktuelle Geschwindigkeit
F2	ACC Wahl 1 (Standard: 80 km/h)
F3	ACC Wahl 2 (Standard: 120 km/h)
F4	ACC Wahl 3 (Standard: 160 km/h)
F5	ACC Wunschgeschwindigkeit +5
F6	ACC Wunschgeschwindigkeit -5
F8	LCA An/Aus

Tabelle 4.1: Zuordnung der Bedienelemente zu Funktionstasten

Die Verarbeitung der Tastatureingaben übernimmt die DPU `RPUKeyInput`. Diese reagiert in ihrer Trigger-Methode auf Tastendrücke der Tastatur. Dazu wird geprüft, ob der Tastencode der gedrückten Taste einer der belegten Funktionstasten entspricht und ob die Taste bereits länger als 3 Sekunden gedrückt wird. Aus diesen Informationen wird ein entsprechendes Ausgabesignal erzeugt, über welches dem ViDAs-Kern mitgeteilt wird, ob und welches Bedienelement betätigt wurde. Neben diesen sieben Bedienelementen zählen auch noch das Gas- und Bremspedal (siehe Abbildung 4.11 Nummer (1)) zur Benutzereingabe, da deren Stellung an den ViDAs-Kern übermittelt wird und Einfluss auf das Systemverhalten hat.

Die Steuerung des Ego-Fahrzeugs selbst ist wahlweise über ein USB-Lenkrad oder die Tastatur möglich, je nachdem was am Simulationsrechner zur Verfügung steht. Die Anbindung dieser Eingabegeräte erfolgt über DPUs, die der Simulationssoftware beiliegen.

4.2.2.2 Benutzerausgabe

Die Benutzerausgabe besteht aus den Leuchtelementen an den Außenspiegeln des Ego-Fahrzeugs (wie in Abbildung 4.11 Nummer (3) für den linken Außenspiegel dargestellt, am rechten Außenspiegel existiert ebenfalls ein solches Leuchtelement), den Leuchtelementen an den Bedienelementen (Nummmer (2) in der Abbildung), sowie zwei Displays zur Verkehrsschildanzeige und einem Display für die aktuelle Wunschgeschwindigkeit (Nummer (4) in der Abbildung).

Da die Erstellung einer grafischen Benutzerausgabe nicht Teil der Projektaufgabe war, erfolgt die Benutzerausgabe mit den Mitteln, welche der Fahrsimulator zur Verfügung stellt. Hierzu gibt es die DPU `RPUAnzeigen`, welche allerdings über keine Logik verfügt, sondern lediglich die

für den Benutzer relevanten Systemausgaben (siehe Abbildung 4.12) bündelt, so dass sie über dieses Modul abrufbar sind. Die Ausgaben können in SILAB als Textfenster angezeigt werden, welche den konkret am Ausgang anliegenden Wert zeigen, ohne diesem jedoch eine spezielle Semantik zuzuweisen, d.h. der Benutzer muss wissen, welcher Ausgabewert welche Bedeutung hat. Mögliche Werte der einzelnen Signale und ihre Bedeutung können Tabelle 4.3 entnommen werden.

Abbildung 4.12: Ausgänge der RPUAnzeigen

Für Demonstrationszwecke und zur intuitiveren Bedienung wurde allerdings ein Java-Programm geschrieben, welches die reinen Zahlenwerte an den Ausgängen entsprechend auswertet und grafisch darstellt. Dieses Programm befindet sich jedoch noch im Alphastadium.

Für die Visualisierung der Strecke und Umgebung sowie der Fremdfahrzeuge wird die dem SILAB-Fahrsimulator beiligende Grafikengine verwendet.

4.3 ViDAs-Komm

Um die ViDAs-Umweltschnittstelle mit dem ViDAs-Kern zu verbinden, ist eine Kommunikationsschnittstelle notwendig. Diese muss dafür sorgen, dass alle notwendigen Sensordaten und Benutzereingaben des SILAB-Fahrsimulators an den ViDAs-Kern übertragen werden. Ebenso muss die Reaktion des Systems an den SILAB-Fahrsimulator übertragen werden, um das Fahrzeug zu steuern beziehungsweise Ausgaben anzuzeigen.

Dafür wurden Nachrichten definiert, in denen diese Daten verpackt werden (siehe Abschnitt 3.2.2.2). Diese Nachrichten können über verschiedene Übertragungsschnittstellen übertragen werden. Dazu wurde zum Einen die *High Level Architecture (HLA)* genutzt und zum Anderen der direkte Export des Matlab-Modells in eine SILAB-DPU. Diese Schnittstellen werden im Folgenden erläutert. Eine Kommunikation über die CAN-Schnittstelle wurde aus Zeitgründen nicht realisiert, dennoch wurde die Kommunikation hierfür vorbereitet. Vorarbeiten für die CAN-Kommunikation finden sich in der Machbarkeitsstudie (siehe Abschnitt 6.5).

Bei der Entwicklung der Kommunikationskomponenten wurde darauf geachtet, dass diese so entwickelt sind, dass die Kommunikation jederzeit durch ein anderes Protokoll ersetzt werden

Anzeige/Leuchtelement	**Wert**	**Bedeutung**
Verkehrsschild 1	0	keine Anzeige
Verkehrsschild 1	1	Schild Geschwindigkeitsbegrenzung 80 km/h anzeigen
Verkehrsschild 1	2	... 90 km/h ...
Verkehrsschild 1	3	... 100 km/h ...
Verkehrsschild 1	4	... 110 km/h ...
Verkehrsschild 1	5	... 120 km/h ...
Verkehrsschild 1	6	... 130 km/h ...
Verkehrsschild 2	0	keine Anzeige
Verkehrsschild 2	1	Überholverbotsschild anzeigen
Wunschgeschwindigkeit	0	ACC deaktiviert, Anzeige Wunschgeschwindigkeit deaktiviert
Wunschgeschwindigkeit	x	Wunschgeschwindigkeitsanzeige zeigt x km/h
Wechselmöglichkeit links	0	Leuchtelement aus (LCA deaktiviert)
Wechselmöglichkeit links	1	Leuchtelement leuchtet grün, Spurwechsel möglich
Wechselmöglichkeit links	2	Leuchtelement leuchtet rot, Spurwechsel nicht möglich
Wechselmöglichkeit rechts	*	analog zum Leuchtelement für die linke Seite
ACC aktuell Leuchtelement	0	Leuchtelement leuchtet rot, Bedienelement nicht aktiviert
ACC aktuell Leuchtelement	1	Leuchtelement leuchtet grün, Bedienelement aktiviert
ACC aktuell Leuchtelement	2	Leuchtelement blinkt dreimal rot, Bedienelement kann nicht aktiviert werden
LCA An/Aus Leuchtelement	*	analog zum Leuchtelement für das Bedienelement ACC aktuell
ACC Wahl 1 Leuchtelement	0	Leuchtelement leuchtet rot, Bedienelement nicht aktiviert
ACC Wahl 1 Leuchtelement	1	Leuchtelement leuchtet grün, Bedienelement aktiviert
ACC Wahl 1 Leuchtelement	2	Leuchtelement blinkt dreimal rot, Bedienelement kann nicht aktiviert werden
ACC Wahl 1 Leuchtelement	3	Leuchtelement blinkt grün, Bedienelement im Programmiermodus
ACC Wahl 2 Leuchtelement	*	analog zum Leuchtelement für das Bedienelement ACC Wahl 1
ACC Wahl 3 Leuchtelement	*	analog zum Leuchtelement für das Bedienelement ACC Wahl 1

Tabelle 4.3: Bedeutung der Werte von Ausgabesignalen

kann. Der schnittstellenunspezifische Teil der Implementierung stellt dabei jeweils einen Adapter dar, der an die Kommunikationsschnittstelle, das ViDAs-Komm-Modul, andockt (siehe Abbildung 4.13 für eine HLA-Implementierung).

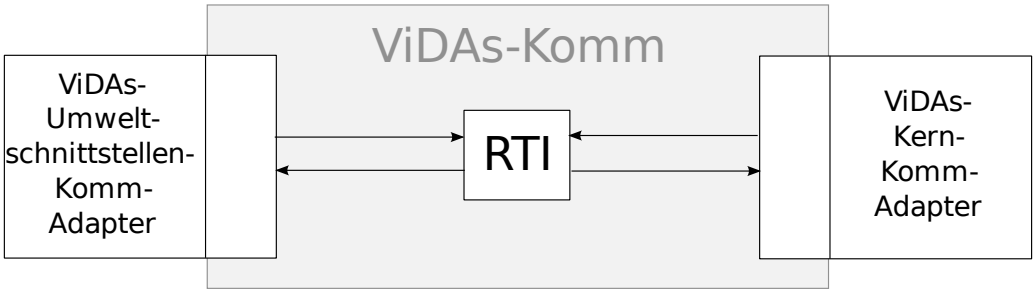


Abbildung 4.13: Schematische Darstellung des HLA-Komm-Moduls

4.3.1 Nachrichten

Für die Kommunikation zwischen den Modulen wurden Nachrichten festgelegt, in die die zu übertragenden Informationen verpackt werden. Diese wurden im Architektur- und Schnittstellendokument [Pro10c] in Kapitel 2 definiert. In diesem Dokument sind ebenso die hier genannten Punkte im Abschnitt 1.2 detaillierter erläutert.

Der Aufbau der Nachrichten wurde so gewählt, dass eine Kommunikation über eine CAN-Schnittstelle mit ihnen möglich ist. Aufgrund von Beschränkungen bei der CAN-Kommunikation über den SILAB-Simulator wurde die maximale Nachrichtenlänge auf 32 Bit beschränkt (siehe dazu auch Abschnitt 6.5).

Die ersten 15 Bit einer Nachricht sind für einen Zeitstempel reserviert. Dieser wird bei jedem Zeitschritt von beiden Komponenten erhöht und dient der Synchronisation dieser.

Die weiteren Felder sind je nach Nachricht definiert. Details hierzu sind dem Architektur- und Schnittstellendokument [Pro10c] ab Seite 11 zu entnehmen.

4.3.2 High-Level-Architecture-Schnittstelle

Die High Level Architecture ist eine Kommunikationsschnittstelle, die es ermöglicht, mehrere Teilsysteme über eine Netzwerkschnittstelle zu verbinden. Dabei werden Nachrichten in Form von fest definierten Objekten übertragen. Diese Objekte müssen zunächst definiert werden. Mittels dieser Definition wird den Kommunikationsteilnehmern mitgeteilt, welche Objekte es gibt. Sie muss auf allen Systemen identisch sein.

4.3.2.1 Kommunikation unter den Teilnehmern

Die Kommunikation findet über eine *Run-Time Infrastructure (RTI)* statt. Diese koordiniert die einzelnen Kommunikationsteilnehmer, die sich in *Federations* zusammen schließen. Dabei kann

die RTI, die eine Art Serverdienst darstellt, auf einem beliebigen Rechner ausgeführt werden, mit dem die Teilnehmer sich verbinden. Innerhalb einer Federation können alle Teilnehmer untereinander kommunizieren. Alle Objekte, die ein Teilnehmer versendet, werden von den anderen Teilnehmern empfangen. Dabei wird bei diesen durch die HLA-Bibliothek ein Callback aufgerufen, der das gesendete Objekt übergibt. Eine Übersicht über die Funktionsweise einer HLA-Kommunikation ist in Abbildung 4.14 zu sehen.

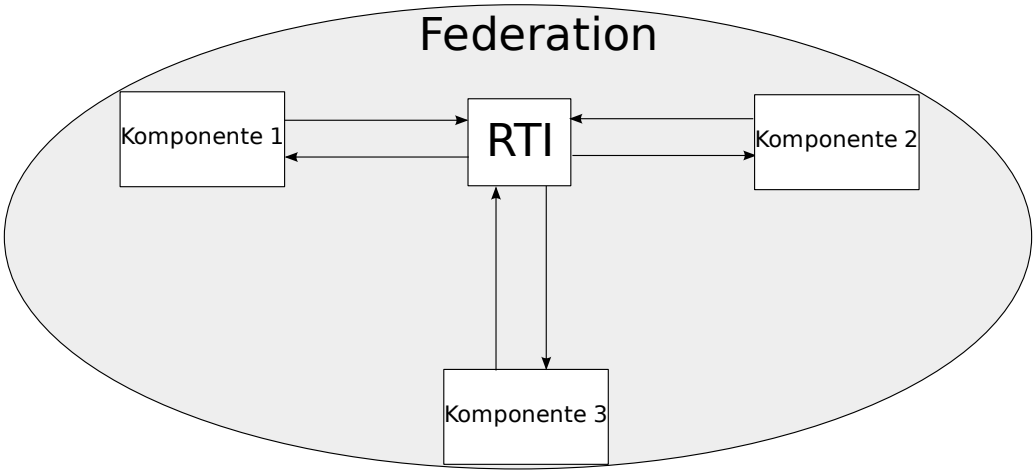


Abbildung 4.14: Schematische Darstellung der Funktionsweise der HLA-Kommunikation

4.3.2.2 Zeitsynchronisation

Ebenso lässt sich mittels HLA eine Zeitsynchronisation realisieren. Dazu gibt es die Möglichkeit, einen Master festzulegen, der den Takt vorgibt und auf den alle anderen Komponenten warten müssen, bevor sie einen weiteren Simulationsschritt durchführen dürfen. Desweiteren gibt es auch die Möglichkeit, allen Komponenten die Rolle des Masters zuzuweisen, sodass die teilnehmenden Komponenten erst dann den nächsten Simulationsschritt durchführen dürfen, wenn alle Komponenten den vorherigen Simulationsschritt abgeschlossen haben.

4.3.2.3 Aufzeichnen von Nachrichten

Bei der Kommunikation über die HLA-Schnittstelle ist es möglich, alle übertragenen Nachrichten anzusehen bzw. zu speichern. Dies erleichtert die Fehlersuche, da geprüft werden kann, ob die Nachricht korrekt übertragen wird. So kann der Fehler genauer lokalisiert werden. Es kann zum Beispiel ausgeschlossen werden, dass eine Komponente nicht korrekt sendet oder der Fehler auf die empfangende Komponente eingeengt werden.

4.3.2.4 Nutzung für ViDAs

Im konkreten System findet die Kommunikation zwischen dem SILAB-Fahrsimulator *SILAB* und dem Matlab-Modell des VIDAs-Kerns statt. Dazu wurde auf der SILAB-Seite eine DPU

(siehe Abschnitt 4.1.2) entwickelt, die eine Anbindung an eine RTI der Firma Pitch über eine C++-HLA-Bibliothek der gleichen Firma zur Verfügung stellt. Informationen über diese Produkte finden sich auf der Internetseite des Herstellers [pit]. Hierbei wurde auf eine bereits entwickelte Erweiterung für einen anderen Simulator zurück gegriffen, die für das IMoST-Projekt [OFF] entwickelt wurden. Sie wurde auf die Nutzung als SILAB-DPU angepasst und die für die Kommunikation erforderlichen Nachrichten wurden eingebaut.

Auf der Matlab-Seite wurde ebenfalls fertiger Code des IMoST-Projekts [OFF] genutzt. Dieser kommuniziert über die Matlab-HLA-Toolbox der Firma FORWARDSIM. Nähere Informationen hierzu finden sich auf der Produktinternetseite des Herstellers [for].

Der Vorteil dieser Kommunikationslösung ist, dass auf der Matlab-Seite zum Beispiel Zustandsübergänge in Stateflow direkt beobachten werden können. Außerdem kann während des Betriebs Code in m-Dateien angepasst werden, sodass kleine Korrekturen ohne viel Aufwand möglich sind. Desweiteren muss keine eigene Zeitsynchronisation entwickelt werden, da diese bereits von der HLA-Kommunikation unterstützt wird.

Die Zeitsynchronisierung ist so gewählt, dass beide Komponenten Master-Komponenten sind. Dadurch warten sie gegenseitig aufeinander und bleiben so stets synchron. Diese Wahl wurde daher getroffen, weil davon ausgegangen wurde, dass beide Komponenten durch die unterschiedlichen Simulatoren mit gleicher Frequenz aufgerufen werden würden (siehe dazu Abschnitt 4.3.2.5). Sollte es hier Schwankungen geben, so soll dennoch die Synchronität sicher gestellt werden, was durch diesen Aufbau möglich ist.

Da das Matlab-Modell und der Fahrsimulator dennoch unter Umständen nicht zwingend parallel laufen, muss sichergestellt werden, dass wichtige Signale bei der Gegenstelle angekommen sind. Daher wurde ein Handshake-Protokoll entwickelt. Dieses sorgt dafür, dass eine Nachricht so lange übertragen wird, bis die Gegenstelle gemeldet hat, dass diese Nachricht angekommen ist. Dies ist zum Beispiel beim Betätigen des Bremspedals wichtig, da das ACC sich in diesem Fall ausschalten muss. Für den Handshake wurden für die entsprechenden Signale neue Nachrichten eingebaut, die das Empfangen einer Nachricht dadurch bestätigen, dass deren Inhalt in die Gegenrichtung erneut gesendet wird, der Sender also seine gesendeten Daten empfängt. Ein Handshake wird für das Bremsen und das Betätigen von Bedienelementen genutzt.

4.3.2.5 Probleme mit der HLA-Kommunikation

Bei der Benutzung stellte sich allerdings heraus, dass es große Zeitprobleme bei der Verwendung von HLA gibt. So kann es vorkommen, dass die Übertragung einer Nachricht bis zu 600 Millisekunden benötigt. Dabei ist der Zeitfortschritt auf der Matlab-Seite deutlich langsamer, wodurch durch den Master-Status auch SILAB ausgebremst wird. Hierbei scheint es sich um ein Problem mit der Matlab-Bibliothek zu handeln. Wird die Menge der zu übertragenden Objekte auf nur zwei reduziert, so tritt keine Verzögerung mehr ein. Dieses Problem wurde durch den Hersteller bestätigt und es gibt derzeit keine konkrete Lösung dafür.

Durch dieses Verhalten treten bei der Verwendung von HLA große Probleme auf, da die Entscheidungslogik für das Gas- und Bremspedal sich im ViDAs-Kern befindet und so eine sehr

große Reaktionszeit beim Bremsen und Gasgeben entsteht. Ebenso hat der Geschwindigkeitsregler Probleme mit dem korrekten Regeln der Geschwindigkeit, wenn die Kommunikation über HLA stattfindet. Außerdem ist ein Testen mittels des automatisierten Testskripts (siehe Abschnitt 5.3) nicht möglich, da die Reaktionszeiten für ein deterministisches Verhalten und einen einfachen Regler für die Geschwindigkeit, wie er im Testskript zum Beschleunigen genutzt wird, zu groß sind.

4.3.3 DPU-Einbettung

Aufgrund der im vorherigen Abschnitt beschriebenen Zeitprobleme wurde eine anderen Lösung für die Kommunikation zwischen Simulator und Matlab erarbeitet. Da eine CAN-Implementierung zu zeitaufwendig gewesen wäre und es zunächst keine funktionierende Kommunikationsschnittstelle gegeben hätte, wurde überlegt, das Matlab-Modell als DPU in den Fahrsimulator einzubetten.

Es existiert bereits ein Target für den Real-Time Workshop, mit dem sich direkt aus Matlab-Modellen SILAB-DPUs erzeugen lassen. Dieses stand allerdings aus Lizenzgründen nicht zur Verfügung, weswegen eine eigene Löung erstellt wurde.

4.3.3.1 Entwicklung der DPU

Mittels des Real-Time Workshops ist es möglich, C- oder C++-Code aus dem Matlab-Modell des ViDAs-Kerns zu erzeugen (siehe Abschnitt 6.3.2). Dabei wurde das Modell so vorbereitet, dass alle Ein- und Ausgänge in bzw. aus einem Submodell, welches den eigentlichen ViDAs-Kern enthält, geführt wurden (siehe Abbildung 4.15). Anschließend wurde das Modell über den Real-Time Workshop aus Matlab als C++-Code exportiert. Durch die gebündelten Ein- und Ausgänge ist es möglich, diese im generieren Quelltext wieder zu finden.

Anschließend wurde ein SILAB-DPU-Wrapper entwickelt, in welchen der generierte Quelltext eingebetet werden kann, so dass aus dem Modell eine SILAB-DPU entsteht. Der Wrapper enthält dabei alle benötigten Ein- und Ausgänge des SILAB-Fahrsimulators, die an das Modell übergeben beziehungsweise von diesem gesetzt werden sollen. In dieser DPU werden in jedem Simulationsschritt alle Eingangsvariablen des Modells mit den Werten der Eingänge gefüllt und an die Ausgänge die Werte der Ausgangsvariablen gelegt. Dabei werden für Tastendrücke des Benutzers sowie für Leuchtelemente der Benutzerschnittstelle noch kleine Berechnungen durchgeführt, um kompatibel zur Schnittstelle zu sein. So wird die Nachricht für die Leuchtelemente an den Außenspiegeln und die für die Bedienelementkontrolleuchten, die jeweils die Information für alle Leuchtelemente enthalten, dekodiert und Ausgänge für die einzelnen Leuchtelemente aus der DPU geführt. Ebenso werden die Tastendrücke aus einem DPU-Eingang gelesen und die Tastencodes dekodiert, in die entsprechenden Tastendruck-Nachrichten umgewandelt und an den ViDAs-Kern weitergegeben. Nach dem Setzen der Werte wird die Funktion `rt_OneStep()`, die einen Simulationsschritt im Modell ausführt, aufgerufen.

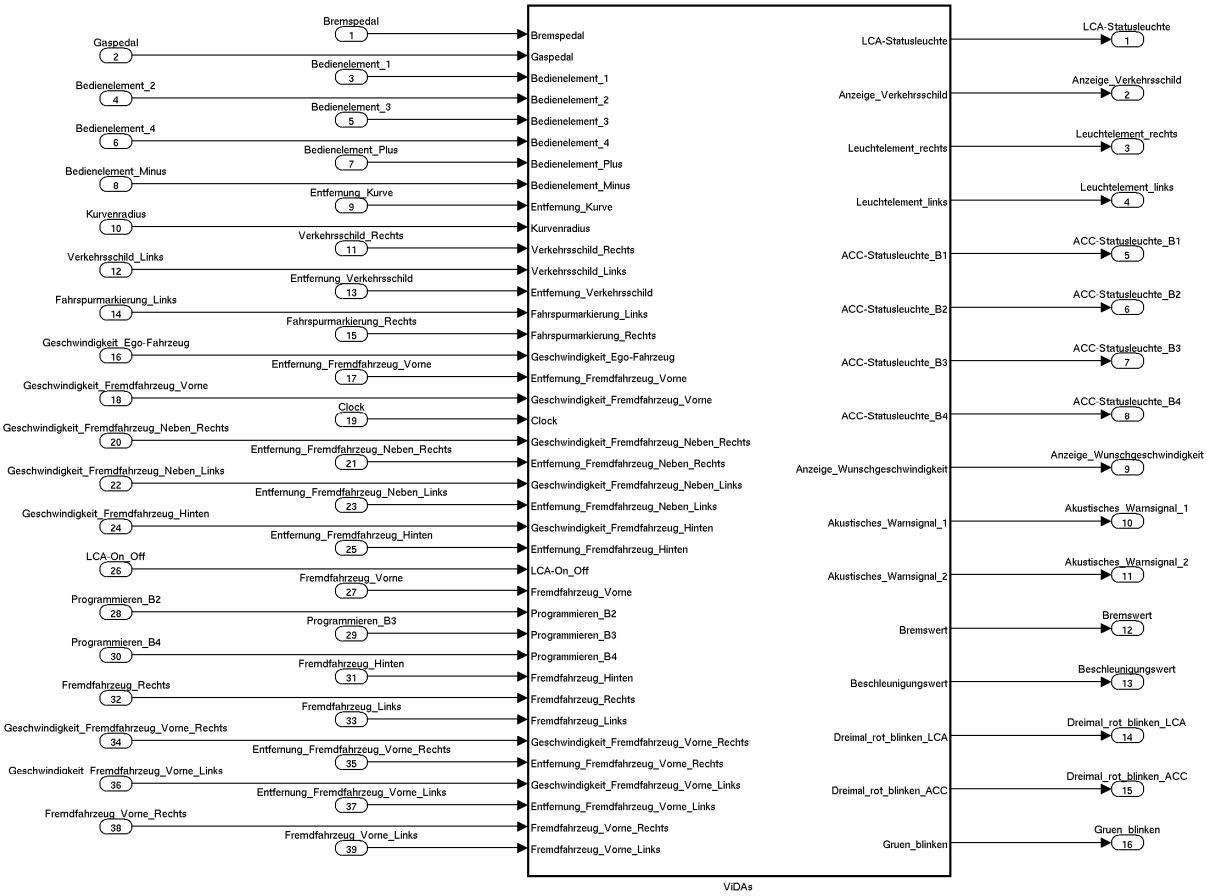


Abbildung 4.15: Ein- und Ausgänge des ViDAs-Kerns

4.3.3.2 Verhalten des Simulators

SILAB ist dabei so konfiguriert, dass alle 20 ms ein Simulationsschritt durchgeführt wird. Dadurch ist auch automatisch die Simulationsfrequenz des Modells festgelegt. In Tests hat sich herausgestellt, dass kürzere Simulationsintervalle des SILAB-Fahrsimulators nicht zuverlässig eingehalten werden können. Auch bei einem Intervall von 20 ms kann es dazu kommen, dass dieses vom SILAB-Fahrsimulator nicht eingehalten werden kann, wodurch auch das Modell teilweise in unregelmäßigen Schritten ausgeführt wird. In Tests hat sich allerdings gezeigt, dass solche Schwankungen nur selten vorkommen.

Eine Möglichkeit diesem Simulationsverhalten entgegenzuwirken, wäre es, das Modell in einem eigenen Thread in einem konstanten, kürzeren Intervall häufiger aufzurufen. Dabei würden die Simulationsdaten nur in einem SILAB-Simulationsschritt aktualisiert. Daher muss das Modell in diesem Fall einen prädiktiven Regler nutzen. Allerdings wurde diese Idee verworfen, da der entwickelte prädiktive Regler zum Zeitpunkt der DPU-Erstellung noch nicht hierfür nutzbar war. Sollte es zu einer Verzögerung der SILAB-Simulation kommen, würde das das Modell in einem Simulationsschritt, in dem der prädiktive Regler bereits neue Daten erwartet hätte, mit alten Daten arbeiten. Dadurch könnte es zu Problemen mit dem Regler kommen.

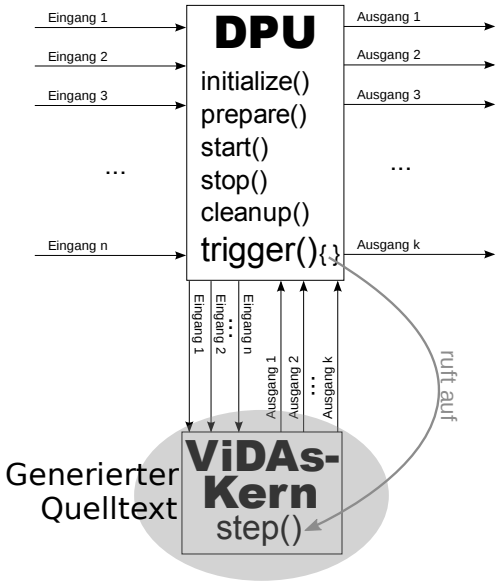


Abbildung 4.16: Schematische Darstellung der ViDAs-Kern-DPU

Der prädiktive Regler wurde dennoch implementiert, da die CAN-Kommunikation einen besseren Rahmen hierfür bieten und dieser Regler hier genutzt werden kann. In der endgültigen Version des Reglers werden keine Daten mehr in festen Schritten erwarten, sondern geprüft, ob der Wert am Eingang sich geändert hat. Mit diesem Regler wäre es möglich, die DPU noch zu erweitern, sodass oben beschriebenes Verhalten möglich wäre.

Da sämtliche DPUs durch den SILAB-Fahrsimulator in jedem Zeitschritt ausgeführt werden, ist, anders als bei der Übertragung per HLA, kein Handshake nötig. Die Wertänderungen werden in jedem Fall bemerkt und können so auch an das Modell weiter geleitet werden.

4.3.3.3 Vorbereitung des generierten Codes

Um die eingestellten Geschwindigkeiten der Geschwindigkeitswahlbedienelemente speichern zu können, wurde ein Patch entwickelt, der an den entsprechenden Stellen im Quelltext diesen leicht anpasst. Hier werden eigene Funktionen eingefügt, die die Werte in einer Textdatei speichern beziehungsweise diese aus dieser Datei wieder laden, da eine solche Funktion zwar in Matlab möglich ist, aber nicht von der Code-Generierung unterstützt wird. Dieser Patch muss nach jedem Exportieren angewendet werden. Sollten zu große Änderungen am Matlab-Modell stattgefunden haben, so kann es sein, dass der Patchvorgang nicht erfolgreich ist. In diesem Fall müssen die Änderungen manuell eingepflegt und hieraus ein neuer Patch generiert werden.

4.3.3.4 Erstellen einer neuen DPU

Die Vorgehensweise beim Erstellen einer neuen DPU ist zusammengefasst folgende:

a) Exportieren des Matlab-Modells mittels des Real-Time Workshops.

b) Patchen des generieren Quelltextes mittels eines Shell-Skripts.

c) Einfügen des generierten und gepatchten Quelltextes in den DPU-Wrapper.

d) Kompilieren der DPU.

Die Abbildung 4.16 skizziert den Aufbau der DPU und das Zusammenspiel mit dem generierten Code. Dabei stellt der obere Block den DPU-Wrapper dar, der mit dem SILAB-Fahrsimulator verbunden ist. Dieser setzt, sobald er vom SILAB-Fahrsimulator getriggert wurde, die Eingänge des unteren Blocks, welcher den generierten Code symbolisiert. Außerdem führt der Wrapper einen Simulationsschritt im Modell durch und leitet die Ausgaben wieder zurück an den SILAB-Fahrsimulator.

4.4 ViDAs-Kern

Der ViDAs-Kern stellt die höchste Übersichtsebene des Fahrerassistenzsystems ViDAs dar. Er beinhaltet die beiden Submodule `ACC` und `LCA`, in denen die jeweiligen Funktionalitäten des Fahrerassistenzsystems umgesetzt sind. Desweiteren beinhaltet der ViDAs-Kern einen `Multiplexer`, der eine Priorisierung der Stellsignale vom ViDAs-Kern bzw. vom Fahrer vornimmt und den `ViDAs-Kern-Komm-Adapter`, der die Anbindung an das `Komm-Modul` und somit an die ViDAs-Umweltschnittstelle übernimmt. Eine Übersicht über die Architektur des ViDAs-Kerns ist in Abbildung 4.17 zu sehen.

Die Submodule `ACC`, `LCA` und der `Multiplexer` werden in den folgenden, identisch aufgebauten Abschnitten in der Art und Weise vorgestellt, dass nach einer kurzen Einleitung, ein tabellarischer Überblick über die Eingänge und Ausgänge der einzelnen Submodule gegeben wird und danach eine detaillierte Beschreibung der Funktionsweise folgt. Der `ViDAs-Kern-Komm-Adapter` besitzt keinerlei Logik und dient lediglich zur Weitergabe der Daten an die Kommunikationsschnittstelle. Er wird hier nicht tiefergehend erläutert.

Um wiederholende Beschreibungen von Komponenten zu vermeiden, wird hier kurz auf die Systemclock und die Delay-Blöcke eingegangen. Die Systemclock wird von jedem Automaten zum Triggern benötigt. Die Delay-Blöcke dienen zur Erzeugung künstlicher Verzögerungen und führen somit zu einem realistischen Zeitverhalten des Fahrassistenzsystems (siehe 6.6.2).

4.4.1 ACC-Submodul

Das Submodul `ACC` beinhaltet die im Folgenden vorgestellten Funktionalitäten. Das `ACC` regelt die durch den Fahrer des Ego-Fahrzeugs eingestellte Wunschgeschwindigkeit innerhalb des Komfortbereichs (siehe [Pro10a], Seite 78). Dabei werden geltende Geschwindigkeitsbegrenzungen, die durch die Sensorik erfasst werden, berücksichtigt. Weiterhin reagiert das `ACC` auf Kurven, die eine laterale Beschleunigung außerhalb des Komfortbereichs verursachen. Desweiteren regelt das `ACC` den Abstand zwischen dem Ego-Fahrzeug und dem auf der selben Fahrspur vorausfahrenden Fremdfahrzeug auf den vorgegebenen minimalen Sicherheitsabstand (siehe [Pro10a], Seite 78f).

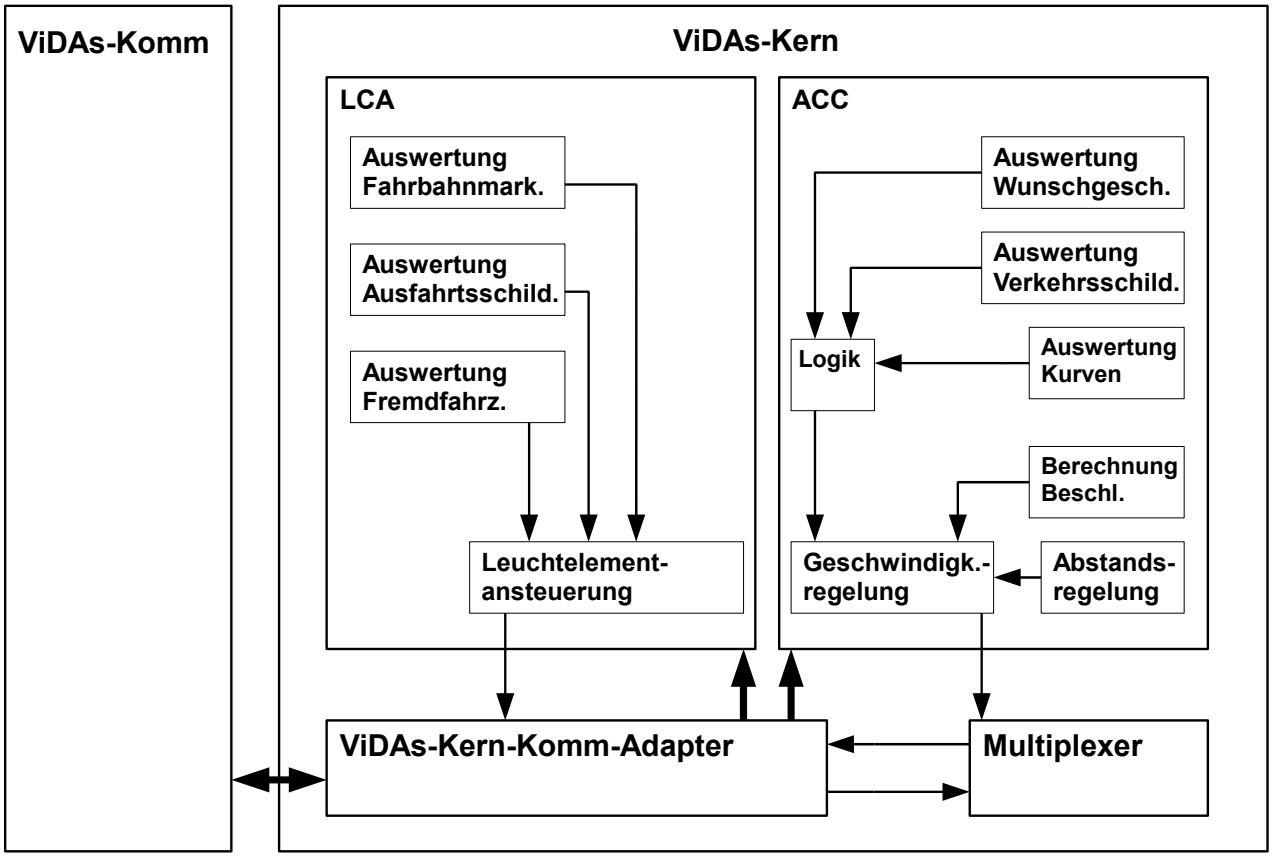


Abbildung 4.17: ViDAs-Kern Architektur

Zur Umsetzung der genannten Funtionalitäten werden die Submodule `Wunschgeschwindigkeitsauswertung`, `Schildauswertung`, `Kurvenauswertung`, `Geschwindigkeitslogik`, `Beschleunigungsberechnung`, `Abstandsregler` und `Geschwindigkeitsregler` benötigt (siehe Abbildung A.4). Diese werden in den folgenden Kapiteln vorgestellt.

4.4.1.1 Wunschgeschwindigkeitsauswertung

Die `Wunschgeschwindigkeitsauswertung` dient zur Auswertung der durch den Fahrer betätigten Geschwindigkeitswahlbedienelemente und zur Programmierung dieser.

Eingänge

Name	Datentyp	Wertebereich	Beschreibung
Bedienelement_Plus	boolean	$[0, 1]$	Event, das bei Betätigung des Bedienelements Plus ausgelöst wird.
Bedienelement_Minus	boolean	$[0, 1]$	Event, das bei Betätigung des Bedienelements Minus ausgelöst wird.

Bedienelement_1	boolean	[0, 1]	Event, das bei Betätigung des Bedienelements 1 ausgelöst wird.
Bedienelement_2	boolean	[0, 1]	Event, das bei Betätigung des Bedienelements 2 ausgelöst wird.
Bedienelement_3	boolean	[0, 1]	Event, das bei Betätigung des Bedienelements 3 ausgelöst wird.
Bedienelement_4	boolean	[0, 1]	Event, das bei Betätigung des Bedienelements 4 ausgelöst wird.
Bremspedal	boolean	[0, 1]	Event, das bei Betätigung des Bremspedals ausgelöst wird.
Aktuelle_Geschwindigkeit	uint8	[0, 160]	Aktuelle Geschwindigkeit des Ego-Fahrzeugs.
Bedienelement_2_3Sek	boolean	[0, 1]	Event, das bei 3 Sekunden langer Betätigung des Bedienelements 2 ausgelöst wird.
Bedienelement_3_3Sek	boolean	[0, 1]	Event, das bei 3 Sekunden langer Betätigung des Bedienelements 3 ausgelöst wird.
Bedienelement_4_3Sek	boolean	[0, 1]	Event, das bei 3 Sekunden langer Betätigung des Bedienelements 4 ausgelöst wird.
Fahrzeug_vorne	boolean	[0, 1]	Status des Sensorbereichs, der sich vor dem Ego-Fahrzeug auf der selben Fahrspur befindet. Sobald sich ein Fremdfahrzeug im Sensorbereich befindet, liegt eine '1' an.

Ausgänge

Name	**Datentyp**	**Wertebereich**	**Beschreibung**
L1	boolean	[0, 1]	Leuchtelement 1 aktiviert/deaktiviert.
L2	boolean	[0, 1]	Leuchtelement 2 aktiviert/deaktiviert.
L3	boolean	[0, 1]	Leuchtelement 3 aktiviert/deaktiviert.
L4	boolean	[0, 1]	Leuchtelement 4 aktiviert/deaktiviert.
Akustisches_Warnsignal	boolean	[0, 1]	Akustisches Warnsignal aktiviert/deaktiviert.
Geschwindigkeit	uint8	[0, 160]	Wunschgeschwindigkeit des Fahrers.
rot_blinken	boolean	[0, 1]	Modus 'rot blinken' des jeweiligen Leuchtelements aktiviert/deaktiviert.
gruen_blinken	boolean	[0, 1]	Modus 'grün blinken' des jeweiligen Leuchtelements aktiviert/deaktiviert.

Beschreibung

Das Submodul `Wunschgeschwindigkeitsauswertung` (siehe Abbildung A.5) besteht aus den beiden Automaten `Wunschgeschwindigkeit` und `Bedienelemente_programmieren` sowie zwei Funktionen zum Laden und Speichern der vordefinierten bzw. der neu programmierten Geschwindigkeiten der Bedienelemente `2-4`.

Bevor die Automaten mit der Auswertung beginnen können, müssen die Geschwindigkeiten der Bedienelemente `2-4` initialisiert werden. Dies übernimmt die Funktion `Bedienelemente_laden`, indem sie die Werte aus einer Textdatei ausliest und an die Eingänge der Automaten anlegt.

Der Automat `Wunschgeschwindigkeit` (siehe Abbildung A.6) ist für die Auswertung der durch den Fahrer betätigten Bedienelemente zuständig und startet im Subautomaten `Geschwindigkeitskontrolle` (siehe Abbildung A.6) im Zustand `Aus`. Hier werden initial die Geschwindigkeiten der Bedienelemente `2-4` geladen. Die Funktionsweise der Zustände `B1` bis `B4` ist grundsätzlich identisch und wird deshalb nur anhand des Zustands `B1` erläutert. Wenn im Zustand `Aus` das Bedienelement `1` bei einer Geschwindigkeit des Ego-Fahrzeugs größer oder gleich 30 km/h betätigt wird, aktiviert sich das `ACC`, steuert die Statusleuchte des Bedienelements an und setzt die Wunschgeschwindigkeit entsprechend der aktuellen Geschwindigkeit. In den anderen Zuständen `B2` bis `B4` wird die Wunschgeschwindigkeit jeweils auf die Geschwindigkeit gesetzt, mit der das Bedienelement belegt ist. Wenn im Zustand `B1` das Bedienelement `1` erneut betätigt wird, deaktiviert sich das `ACC`. Bei der Betätigung der Bedienelemente `2-4`, wechselt der Automat, unter der Voraussetzung, dass die Geschwindigkeit des Ego-Fahrzeugs immer noch größer oder gleich 30 km/h ist, in die jeweiligen Zustände `B2` bis `B4` und setzt die Wunschgeschwindigkeit auf den jeweiligen gespeicherten Wert. In Zustand `B1` befindlich, besteht die Möglichkeit, die Wunschgeschwindigkeit mittels der Bedienelemente `Plus` und `Minus` anzupassen. Hierbei wird der Wert der Wunschgeschwindigkeit pro Betätigung des Bedienelemets `Plus` oder `Minus` um eine vordefinierte Schrittweite erhöht bzw. verringert. Hierbei wird beachtet, dass die Wunschgeschwindigkeit nicht größer als 160 km/h und nicht kleiner als 30 km/h gewählt werden kann. Wenn in den Zuständen `B2` bis `B4` die Bedienelemente `Plus` oder `Minus` betätigt werden, wechselt der Automat in Zustand `B1`.

Wenn die Geschwindigkeit des Ego-Fahrzeugs bei aktivem `ACC` unter 30 km/h fällt und sich kein Fremdfahrzeug vor dem Ego-Fahrzeug befindet, deaktiviert sich das `ACC` und lässt ein Warnsignal ertönen, das durch den parallel laufenden Subautomaten `Warnen` (siehe Abbildung A.6) zeitlich gesteuert wird. Ein weiterer Grund zur Deaktivierung des aktivierten `ACCs` ist die Betätigung des Bremspedals durch den Fahrer.

Für den Fall, dass der Fahrer ein Geschwindigkeitswahlbedienelement bei einer Geschwindigkeit des Ego-Fahrzeugs zwischen 0 und 30 km/h betätigt, wird er mittels eines roten Blinkens des betätigten Bedienelements darauf hingewiesen, dass sich das `ACC` nicht aktiviert hat und der Automat Geschwindigkeitskontrolle sich demnach im Zustand `Aus` befindet.

Zur Programmierung der Geschwindigkeitswahlbedienelemente startet der Automat `Bedienelemente_program` (siehe Abbildung A.7) initial im Zustand `idle`. Dabei werden die gespeicherten Werte für die Geschwindigkeitswahlbedienelemente geladen. Sobald die Geschwindigkeit des Ego-Fahrzeugs '`0`' ist, wechselt der Automat in den Zustand `Stillstand`. Nachdem eines der Geschwindigkeitswahlbedienelemente für drei Sekunden betätigt wird und das Ego-

Fahrzeug sich immer noch im Zustand `Stillstand` befindet, wird in den entsprechenden Zustand gewechselt und das entsprechende Leuchtelement blinkt grün. In diesem Zustand ist es jetzt möglichi, die Wunschgeschwindigkeit für das entsprechende Geschwindigkeitswahlbedienelement anzupassen. Um diese Wunschgeschwindigkeit anzupassen, wird das Bedienelement `Plus` oder `Minus` betätigt. Es wird dabei berücksichtigt, dass die Wunschgeschwindigkeit im Bereich $[30, 160]km/h$ liegt, ansonsten wechselt der Automat in den Zustand 'Warn' und das entsprechende Leuchtelement blinkt dreimal rot. Nachdem die Wunschgeschwindigkeit angepasst wurde und dasselbe Geschwindigkeitswahlbedienelement erneut betätigt wird, wechselt der Automat wieder in den Initialzustand und es wird die angepasste Wunschgeschwindigkeit gespeichert. In dem Fall, dass ein anderes Geschwindigkeitswahlbedienelement für drei Sekunden betätigt wird, wechselt der Automat in den entsprechenden Zustand und speichert die angepasste Wunschgeschwindigkeit nicht. Daraufhin kann die Wunschgeschwindigkeit für das betätigte Geschwindigkeitswahlbedienelement angepasst werden. Sobald das Ego-Fahrzeug in Bewegung ist, wechselt der Automat in den Initialzustand und speichert die Änderungen der Wunschgeschwindigkeit nicht. In dem Fall, dass ein falsches Geschwindigkeitswahlbedienelement betätigt wird, wechselt der Automat in den Zustand 'Warn' und das entsprechende Leuchtelement blinkt dreimal rot. Danach wechselt der Automat wieder in den Zustand, aus dem er in den Zustand 'Warn' gewechselt ist.

Nach abgeschlossener Programmierung der Bedienelemente werden die neuen Geschwindigkeiten mittels der Funktion `Bedienelemente_speichern` in eine Textdatei gespeichert, aus der sie dann von der Funktion `Bedienelemente_laden` wieder ausgelesen werden können.

4.4.1.2 Schildauswertung

Die `Schildauswertung` dient zur Verarbeitung der durch die ViDAs-Umweltschnittstelle bereitgestellten Geschwindigkeitsbegrenzungen und Überholverbote.

Eingänge

Name	**Datentyp**	**Wertebereich**	**Beschreibung**
Verkehrsschild_links	uint8	$[0, 14]$	Typ des linken Verkehrsschilds. Angebracht am linken Fahrbahnrand oder oberhalb der linken Fahrspur.
Verkehrsschild_rechts	uint8	$[0, 14]$	Typ des rechten Verkehrsschilds. Angebracht am rechten Fahrbahnrand oder oberhalb der rechten Fahrspur.
Fahrspur	uint8	$[0, 3]$	Fahrspur, auf der sich das Ego-Fahrzeug befindet.
Entfernung_Schild	uint8	$[0, 200]$	Entfernung zum nächsten Verkehrsschild.
Aktuelle_Geschwindigkeit	uint8	$[0, 160]$	Aktuelle Geschwindigkeit des Ego-Fahrzeugs.

Ausgänge

Name	Datentyp	Wertebereich	Beschreibung
Geschwindigkeit	uint8	$[0, 160]$	Geschwindigkeit, die durch das Verkehrsschild vorgegeben wird.
Anzeige_Verkehrsschild	uint8	$[0, 6]$ $[9, 14]$	Anzeige der aktuell geltenden Verkehrsschilder.

Beschreibung

Das Submodul `Schildauswertung` (siehe Abbildung A.8) erhält von der ViDAs-Umweltschnittstelle Informationen über den Typ des linken bzw. rechten Verkehrsschilds. Da die ViDAs-Umweltschnittstelle diese Informationen nur für den Zeitraum, in dem das Ego-Fahrzeug sich zwischen 200 und 0 Metern Entfernung zum Schild befindet, am Eingang anlegt, müssen die Schilder solange gespeichert werden, bis ein neues Schild erkannt wird. Dies geschieht mittels der Blöcke 2-5 und den Speicherbausteinen (Block 6-9)(siehe Abbildung A.8). Sobald ein neues Schild erkannt wird, schaltet der Switch und legt das neue Schild in der Speicher. Ansonsten verbleibt das alte Schild im Speicher. Die Schilder werden dem Automaten `Auswertung_Schilder` (siehe Abbildung A.8) übergeben, wo mithilfe der Fahrspur, auf der sich das Ego-Fahrzeug befindet, das linke und rechte Verkehrsschild ausgewertet werden. Dies geschieht mittels zweier Subautomaten. Der Automat `Fahrspur_auswerten` (siehe Abbildung A.10) ermittelt mittels der Fahrspur, welches der beiden Verkehrsschilder, links oder rechts, für das Ego-Fahrzeug relevant ist. Der Automat `Rechnen` (siehe Abbildung A.10) setzt mithilfe des Typs des Verkehrsschilds die Variablen `Geschwindigkeit` und `Überholverbot`.

Parallel zur Auswertung der Schilder verwendet die `Schildauswertung` den Automaten `praediktive_ Streckenberechnung` (siehe Abbildung A.9). Dieser wird eingesetzt, um eine möglichst präzise Regelung zu gewährleisten. Er berechnet die Entfernung zum Schild für den Zeitraum zwischen zwei Aktualisierungen der Entfernung durch die ViDAs-Umweltschnittstelle eigenständig weiter. Er benutzt hierfür die aktuelle Geschwindigkeit und die Entfernung zum Schild. Der Automat funktioniert in der Art und Weise, dass solange selbstständig ein neuer Wert für die Entfernung berechnet und ausgegeben wird, bis die ViDAs-Umweltschnittstelle einen neuen Entfernungswert liefert. Wenn ein neuer Wert geliefert wird, wird dieser am Ausgang angelegt und an das Submodul `Auswertung_Entfernung` übergeben.

Damit das `ACC` exakt bei Erreichen des Verkehrsschilds die geforderte Geschwindigkeitsbegrenzung einhält, benötigt die `Schildauswertung` das Submodul `Auswertung_Entfernung` (siehe Abbildung A.11). Hier wird der genaue Zeitpunkt berechnet, zu dem die Geschwindigkeit, auf die das `ACC` verzögern soll, an den Geschwindigkeitsregler übergeben wird. Dazu werden die Ist- sowie die Sollgeschwindigkeit und die Entfernung zum Schild benötigt. Das Submodul `Entfernungsberechnung` (siehe Abbildung A.12) ermittelt anhand der Ist- und der Sollgeschwindigkeit die Entfernung zum Schild, bei der das Fahrzeug mit der Verzögerung beginnen muss. Sobald dieser Zeitpunkt erreicht ist (Block 7), setzt Block 9 den Ausgang `Geschwindigkeit` vom vorherigen Verkehrsschild auf das neue Schild. Die Speicherung der Schilder übernehmen die Data Store Blöcke 3-6.

Da die `Schildauswertung` dem Fahrer aktuell geltende Verkehrsschilder anzeigen soll, bekommt das Submodul zusätzlich noch die Information über ein Überholverbot. Die Speicherung des Überholverbots wird in diesem Fall vom Automaten `Speicher_Ueberholverbot` übernommen. Das Submodul `Umwandlung_Verkehrsschildtyp` (siehe Abbildung A.12) codiert die aktuell geltenden Schilder in eine Zahl und legt diese am Ausgang `Anzeige_Verkehrsschild` an.

4.4.1.3 Kurvenauswertung

Die `Kurvenauswertung` hat den Zweck, die von der ViDAs-Umweltschnittstelle bereitgestellten Informationen über die nächste Kurve auszuwerten und zu entscheiden, ob eine Verzögerung nötig ist, um im Rahmen des Komfortbereichs die maximal erlaubte laterale Beschleunigung nicht zu überschreiten.

Eingänge

Name	Datentyp	Wertebereich	Beschreibung
Aktuelle_Geschwindigkeit	uint8	$[0, 160]$	Aktuelle Geschwindigkeit des Ego-Fahrzeugs.
Entfernung_Kurve	uint8	$[0, 160]$	Entfernung zur nächsten Kurve
Kurvenradius	uint16	$[0, 1000]$	Kurvenradius der nächsten Kurve

Ausgänge

Name	Datentyp	Wertebereich	Beschreibung
Geschwindigkeit	uint8	$[0, 160]$	Geschwindigkeit, die durch den Kurvenradius der Kurve vorgegeben wird.

Beschreibung

Um zu ermitteln, mit welcher Geschwindigkeit die nächste Kurve maximal durchfahren werden darf, verwendet das Submodul `Kurvenauswertung` (siehe Abbildung A.13) den Automaten `Auswertung_Kurvenradien` (siehe Abbildung A.14). Hier wird für die spezifischen Kurvenradien jeweils der Ausgang `Geschwindigkeit` mit der maximalen Kurvengeschwindigkeit belegt.

Die Berechnung des Zeitpunkts, wann diese Geschwindigkeit an den Geschwindigkeitsregler übergeben wird, berechnet das Submodul `Auswertung_Entfernung` (siehe Abbildung A.15). Dieser berechnet anhand der Ist- und der Sollgeschwindigkeit den Zeitpunkt, wann das Fahrzeug mit dem Verzögern beginnen muss. Solange der Zeitpunkt noch nicht erreicht ist (Block 1), legt Block 3 '`0`' an den Ausgang `Geschwindigkeit`. Danach leitet Block 3 die Sollgeschwindigkeit an den Ausgang.

Die `Kurvenauswertung` verwendet analog zur `Schildauswertung` den Automaten `praediktive_`

`Streckenberechnung` (siehe Abbildung A.9) um die Präzision der Regelung zu erhöhen.

4.4.1.4 Geschwindigkeitslogik

Die `Geschwindigkeitslogik` entscheidet, welche der drei Sollgeschwindigkeiten an den Geschwindigkeitsregler übergeben wird.

Eingänge

Name	Datentyp	Wertebereich	Beschreibung
Wunschgeschwindigkeit	uint8	$[0, 160]$	Wunschgeschwindigkeit des Fahrers
Schildgeschwindigkeit	uint8	$[0, 160]$	Geschwindigkeit, die durch das Schild vorgegeben wird.
Kurvengeschwindigkeit	uint8	$[0, 160]$	Geschwindigkeit, die durch den Kurvenradius der Kurve vorgegeben wird.

Ausgänge

Name	Datentyp	Wertebereich	Beschreibung
Geschwindigkeit	uint8	$[0, 160]$	Geschwindigkeit, die die höchste Priorität hat.

Beschreibung

Das Submodul `Geschwindigkeitslogik` (siehe Abbildung A.17) besteht aus dem Automaten `Entscheider` (siehe Abbildung A.18), der einen der Eingänge `Wunschgeschwindigkeit`, `Schildgeschwindigkeit` oder `Kurvengeschwindigkeit` an den Ausgang `Geschwindigkeit` legt. Von den Eingängen, die ungleich '`0`' sind, wird der minimale am Ausgang angelegt.

4.4.1.5 Beschleunigungsberechnung

Die `Beschleunigungsberechnung` ermittelt mithilfe der Geschwindigkeit des Ego-Fahrzeugs dessen aktuelle Beschleunigung. Für den Fall, dass die Berechnungsrate des ViDAs-Kerns höher ist als die Aktualisierungsrate der eingehenden Daten, wird, um die Präzision der nachfolgenden Regelungen zu erhöhen, die Geschwindigkeit des Ego-Fahrzeugs im Zeitraum zwischen zwei Aktualisierungen mittels der Beschleunigung berechnet.

Eingänge

Name	Datentyp	Wertebereich	Beschreibung
v_ego_in	double	$[0, 161[$	Aktuelle Geschwindigkeit des Ego-Fahrzeugs

Ausgänge

Name	Datentyp	Wertebereich	Beschreibung
v_ego_out	uint8	$[0, 160]$	Aktuelle Geschwindigkeit des Ego-Fahrzeugs
a_ego	double	$[-11, 7]$	Aktuelle Beschleunigung des Ego-Fahrzeugs

Beschreibung

Um die aktuelle Beschleunigung des Ego-Fahrzeugs zu berechnen, benötigt der Automat `Berechnung_ Beschleunigung` (siehe Abbildung A.20) die aktuelle Geschwindigkeit des Ego-Fahrzeugs. Dabei wird die Geschwindigkeitsdifferenz zwischen zwei Takten gemessen und auf eine Sekunde hochgerechnet.

Um die Präzision der nachfolgenden Regelungen zu erhöhen, berechnet der Automat `praediktive_ Geschwindigkeitsberechnung` (siehe Abbildung A.21) die aktuelle Geschwindigkeit für den Zeitraum zwischen zwei Aktualisierungen der Geschwindigkeit durch die ViDAs-Umweltschnittstelle eigenständig weiter und stellt diese den Reglern zur Verfügung. Dieser benutzt hierfür die aktuelle Geschwindigkeit und die aktuelle Beschleunigung des Ego-Fahrzeugs. Der Automat funktioniert in der Art und Weise, dass solange selbstständig ein neuer Wert für die Geschwindigkeit berechnet und ausgegeben wird, bis die ViDAs-Umweltschnittstelle einen neuen Wert liefert. Wenn ein neuer Wert geliefert wird, wird dieser an den Ausgang des Automaten angelegt.

4.4.1.6 Abstandsregler

Der `Abstandsregler` hat die Aufgabe, den Abstand zwischen dem Ego-Fahrzeug und dem auf der selben Fahrspur wie das Ego-Fahrzeug vorausfahrenden Fremdfahrzeug auf den minimalen Sicherheitsabstand zu regeln. Weiterhin ermittelt der `Abstandsregler` einen Gefahrenfaktor, mit dem der `Geschwindigkeitsregler` bei Bedarf eine Notfallbremsung einleitet.

Eingänge

Name	Datentyp	Wertebereich	Beschreibung
Aktuelle_Geschwindigkeit	uint8	$[0, 160]$	Aktuelle Geschwindigkeit des Ego-Fahrzeugs
Abstand_Fremdfahrzeug	uint8	$[0, 200]$	Abstand zwischen dem Ego-Fahrzeug und dem, auf der selben Fahrspur wie das Ego-Fahrzeug, vorausfahrenden Fremdfahrzeug.
Geschwindigkeit_-Fremdfahrzeug	uint8	$[0, 160]$	Geschwindigkeit des, auf der selben Fahrspur wie das Ego-Fahrzeug, vorausfahrenden Fremdfahrzeugs.
Fahrzeug_vorne	boolean	$[0, 1]$	Zeigt an, ob sich ein Fremdfahrzeug, auf der selben Fahrspur wie das Ego-Fahrzeug, im Sensorbereich befindet.

Ausgänge

Name	Datentyp	Wertebereich	Beschreibung
Beschleunigung	double	$[-55, 90]$	Beschleunigung, die benötigt wird, um den Abstand zum vorausfahrenden Fremdfahrzeug zu regeln.
Gefahrenfaktor	double	$[0, \infty]$	Faktor, der besagt, wann eine Notfallbremsung benötigt wird.

Beschreibung

Um die Beschleunigung zu berechnen, die nötig ist, den minimalen Sicherheitsabstand zum vorausfahrenden Fremdfahrzeug einzuhalten, verwendet das Submodul `Abstandsregler` (siehe Abbildung A.22) die aktuelle Geschwindigkeit des Ego-Fahrzeugs sowie die Geschwindigkeit, den Abstand und die Information über die Anwesenheit des Fremdfahrzeugs. Der `Abstandsregler` ist durch einen PID-Regler realisiert. Mittels der Blöcke 4-5 werden die Einheiten der beiden Geschwindigkeiten in m/s umgewandelt. Block 6 stellt den minimalen Sicherheitsabstand, in diesem Fall den 2-Sekunden-Abstand, zum vorausfahrenden Fremdfahrzeug dar. Die Differenz-Blöcke 7-8 sorgen dafür, dass der Regler die Geschwindigkeit des Ego-Fahrzeugs der Geschwindigkeit des Fremdfahrzeugs und den Abstand zwischen Ego- und Fremdfahrzeug dem minimalen Sicherheitsabstand anpasst. Blöcke 9-11 stellen den Proportional-, Integral- und Differentialanteil des PID-Reglers dar. Block 12 sorgt dafür, dass der Integralanteil aufintegriert wird. Die Summe der drei Anteile (Block 13) ergibt die benötigte Beschleunigung.

Neben der Berechnung der Beschleunigung stellt der `Abstandsregler` mithilfe der Funktion `Gefahrenbremsung` einen Gefahrenfaktor zur Verfügung, mit dem der `Geschwindigkeitsregler`

bei einer Überschreitung eines Schwellwertes einen möglichen Auffahrunfall durch eine Notfallbremsung verhindert. Die Formel zur Berechnung des Gefahrenfaktors ist folgende:

$$faktor = (v\_ego - v\_fremd) \cdot (\frac{1}{abstand - ((\frac{v\_fremd}{3.6}) \cdot 2)})$$

Somit ist der Faktor abhängig von der Differenz der beiden Fahrzeuggeschwindigkeiten sowie der Differenz zwischen dem Abstand der beiden Fahrzeuge und dem Sicherheitsabstand.

4.4.1.7 Geschwindigkeitsregler

Der `Geschwindigkeitsregler` hat die Aufgabe die Ist-Geschwindigkeit des Ego-Fahrzeugs der Soll-Geschwindigkeit, vorgegeben durch Fahrer, Verkehrsschilder oder Kurven, im Rahmen der Begrenzungen des Komfortbereichs anzupassen. Ausnahmen dieser Begrenzungen werden im Fall einer Notfallbremsung gemacht, bei der der Fahrer durch ein akustisches Warnsignal auf eine Notfallbremsung hingewiesen wird. Damit die ViDAs-Umweltschnittstelle die benötigte Beschleunigung umsetzen kann, wird diese in Werte für die Bremspedal- und Gaspedalstellung umgewandelt und an die ViDAs-Umweltschnittstelle übergeben.

Eingänge

Name	**Datentyp**	**Wertebereich**	**Beschreibung**
Aktuelle_Geschwindigkeit	uint8	$[0, 160]$	Aktuelle Geschwindigkeit des Ego-Fahrzeugs
Soll_Geschwindigkeit	uint8	$[0, 160]$	Geschwindigkeit, auf die der Geschwindigkeitsregler einregeln soll.
Beschleunigung_-Abstandsregelung	double	$[0, 0]$	Von der Abstandsregelung ermittelte Beschleunigung, die benötigt wird, um den Abstand zum vorausfahrenden Fremdfahrzeug zu regeln.
Wunschgeschwindigkeit_-Fahrer	uint8	$[0, 160]$	Zeigt den Status des `ACC` an. Sobald ungleich '`0`' ist das `ACC` aktiviert.
Faktor	double	$[0, \infty]$	Faktor, der besagt, wann eine Notfallbremsung benötigt wird.
Aktuelle_Beschleunigung	double	$[-11, 7]$	Aktuelle Beschleunigung des Ego-Fahrzeugs

Ausgänge

Name	Datentyp	Wertebereich	Beschreibung
Bremspedal	single	$[0, 10]$	Bremspedalstellung, die an das Ego-Fahrzeug übergeben wird.
Gaspedal	single	$[0, 1]$	Gaspedalstellung, die an das Ego-Fahrzeug übergeben wird.
Warnsignal	boolean	$[0, 1]$	Warnsignal, das ertönt, solange eine Notfallbremsung stattfindet.

Beschreibung

Das Submodul `Geschwindigkeitsregler` (siehe Abbildung A.24) ermittelt aus der Differenz der Ist- und der Sollgeschwindigkeit (Block 1) und dem Regelungsfaktor (Block 2) die benötigte Beschleunigung zur Anpassung der Ist- an die Sollgeschwindigkeit. Daraufhin wird diese Beschleunigung mit der des `Abstandsreglers` verglichen und die höherpriorisierte Beschleunigung (Block 3) weitergeleitet.

Um Beschleunigungsänderungen im Rahmen des Komfortbereichs zu gewährleisten, wird das Submodul `AntiRuck` (siehe Abbildung A.25) zwischengeschaltet. Der Automat `AntiRuck` (siehe Abbildung A.26) sorgt dafür, dass die ausgehende Beschleunigung solange mit der vorgegebenen Schrittweite an die eingehende Beschleunigung angepasst wird, bis die beiden Beschleunigungen übereinstimmen oder das Maximum bzw. Minimum erreicht ist. Während das Minimum nur durch den Komfortbereich festgeschrieben wird, kann es beim Maximum neben dem Komfortbereich (Block 10) auch zu Begrenzung durch das Fahrzeug kommen. Die maximale realisierbare Beschleunigung des Fahrzeugs wird durch Block 9 berechnet.

Bevor die Beschleunigung aus dem Submodul `AntiRuck` an das Submodul `Beschleunigungsumrechnung` weitergegeben wird, wird entschieden, ob die Notfallbremsung zum Einsatz kommt und der Fahrer dementsprechend durch ein Warnsignal darauf hingewiesen werden muss. Wenn der Schwellwert in Block 11 überschritten wird, gibt Block 14 die Notfallbremsung mit -5 m/s (Block 12) frei.
Um die benötigte Beschleunigung umzusetzen, wird mittels der Verzögerung der Motorbremse (Block 8) entschieden, ob das Fahrzeug beschleunigen oder verzögern muss. Wenn die benötigte Beschleunigung größer ist als die Verzögerung der Motorbremse, muss das Fahrzeug beschleunigen. Daraufhin wird die Bremspedalstellung gleich '0' gesetzt und die benötigte Beschleunigung mithilfe des Blocks 21 in eine Gaspedalstellung umgerechnet und ausgegeben. Für den Fall, dass das Fahrzeug verzögern muss, wird die Gaspedalstellung genullt und die Beschleunigung mittels Block 19 in eine Bremspedalstellung umgewandelt und ausgegeben.

4.4.2 LCA-Submodul

Das Submodul `LCA` beinhaltet die im Folgenden vorgestellten Funktionalitäten. Das `LCA` steuert Signalleuchten für die Fahrspurwechselmöglichkeit des Ego-Fahrzeugs. Dabei werden die Fahr-

spurmarkierungen, Verkehrsschilder und die Fremdfahrzeuge berücksichtigt, um die Fahrspur, Ausfahrtmöglichkeit und Fahrspurwechselverbote zu ermitteln.

Zur Umsetzung der genannten Funktionalitäten werden die Submodule `Auswertung_Markierungen`, `Auswertung_ Ausfahrtsschilder`, `Auswertung_ Fremdfahrzeuge` und `Auswertung_Verbote` benötigt (siehe Abbildung A.28). Diese werden in den folgenden Abschnitten vorgestellt.

4.4.2.1 Auswertung Markierungen

Das Submodul `Auswertung_Markierungen` (siehe Abbildung A.29) ermittelt die Fahrspur, auf der sich das Ego-Fahrzeug befindet. Weiterhin werden Spurwechselverbote in Form von durchgezogenen Linien erkannt und es wird ausgewertet, ob sich das Ego-Fahrzeug neben einer Auf- bzw. Abfahrt befindet.

Eingänge

Name	**Datentyp**	**Wertebereich**	**Beschreibung**
Markierung_rechts	uint8	$[0, 4]$	Die Fahrspurmarkierung rechts des Ego-Fahrzeugs ist folgendermaßen kodiert: • 0: ungültige Fahrspurmarkierung • 1: 15cm gestrichelte Linie • 2: 30cm gestrichelte Linie • 3: 15cm durchgezogene Linie • 4: 30cm durchgezogene Linie
Markierung_links	uint8	$[0, 4]$	Die Fahrspurmarkierung links des Ego-Fahrzeugs ist analog zu Markierung_rechts kodiert

Ausgänge

Name	Datentyp	Wertebereich	Beschreibung
Fahrspur	uint8	$[0, 3]$	Die Fahrspur, auf der das Ego-Fahrzeug fährt, ist folgendermaßen kodiert: • 0: ungültige Fahrspur • 1: rechte Fahrspur • 2: linke Fahrspur • 3: Auf-/Abfahrt
Linienverbot	boolean	$[0, 1]$	Spurwechselverbot, das durch durchgezogene Linie entsteht
NebenAufAbfahrt	boolean	$[0, 1]$	Es wird der Wert '`1`' gesetzt, solange das Ego-Fahrzeug sich neben einer Auf- bzw. Abfahrt befindet

Beschreibung

Der Automat `Fahrspurauswertung` (siehe Abbildung A.30) startet initial im Zustand `Ungültig` bis die rechte und linke Fahrspurmarkierung ungleich '`0`', d.h. beides gültige Markierungen sind. Daraufhin wird die Transition entsprechend der passenden Markierungskombination genommen und `Fahrspur`, `Linienverbot` und `NebenAufAbfahrt` gesetzt.

4.4.2.2 Auswertung Ausfahrtsschilder

Anhand der erkannten Verkehrsschilder für die Ausfahrt und der Fahrspurmarkierung rechts vom Ego-Fahrzeug wertet das Submodul `Auswertung_Ausfahrtsschilder` (siehe Abbildung A.31) aus, ob sich das Ego-Fahrzeug neben einer Ausfahrt befindet.

Eingänge

Name	Datentyp	Wertebereich	Beschreibung
Verkehrsschild_rechts	uint8	[0, 14]	Verkehrsschild, das für die rechte Fahrspur gilt. Angebracht am rechten Fahrbahnrand oder oberhalb der rechten Fahrspur. Es werden nur folgende Schilder berücksichtigt: • 11: 100m Ausfahrt-Bake • 12: 200m Ausfahrt-Bake • 13: 300m Ausfahrt-Bake • 14: Ausfahrt-Schild
Markierung_rechts	uint8	[0, 4]	Fahrspurmarkierung rechts des Ego-Fahrzeugs
Entfernung_Schild	uint8	[0, 200]	Entfernung zum erkannten Verkehrsschild

Ausgänge

Name	Datentyp	Wertebereich	Beschreibung
Ausfahrt	boolean	[0, 1]	Es wird der Wert '1' gesetzt, solange sich das Ego-Fahrzeug neben einer Ausfahrt befindet

Beschreibung

Der Automat `Ausfahrtsschilderauswertung` (siehe Abbildung A.32) startet initial im Zustand `A` bis eine der Ausfahrts-Barken erkannt wird. Daraufhin wechselt der Automat in den Zustand `B` und bleibt in diesem Zustand, bis rechts vom Ego-Fahrzeug eine 30cm breite, gestrichelte Linie erkannt wird oder das Ego-Fahrzeug an einem Ausfahrts-Schild vorbeigefahren ist. Wird rechts vom Ego-Fahrzeug eine 30cm breite, gestrichelte Linie erkannt, wechselt der Automat in den Zustand `C`, legt am Ausgang für die `Ausfahrt` den Wert '1' an und bleibt in diesem Zustand, bis die Fahrspurmarkierung rechts vom Ego-Fahrzeug ungleich '2' ist, d.h. keine 30cm breite, gestrichelte Linie erkannt wird. Daraufhin wechselt der Automat in den Initialzustand `A` und setzt am Ausgang den Wert '0'.

4.4.2.3 Auswertung Fremdfahrzeuge

Das Submodul `Auswertung_Fremdfahrzeuge` (siehe Abbildung A.33) ermittelt die Fahrspurwechselverbote links und rechts unter Berücksichtigung der erkannten Fremdfahrzeuge.

Eingänge

Name	Datentyp	Wertebereich	Beschreibung
Fahrspur	uint8	$[0, 3]$	Die Fahrspur, auf der das Ego-Fahrzeug fährt
v_Egofahrzeug	uint8	$[0, 160]$	Geschwindigkeit des Ego-Fahrzeugs (km/h)
v_Fremdfahrzeug_neben_-rechts	uint8	$[0, 200]$	Geschwindigkeit des Fremdfahrzeugs rechts neben dem Ego-Fahrzeug (km/h)
s_Fremdfahrzeug_neben_-rechts	int16	$[-10, 200]$	Entfernung bis zum Fremdfahrzeug rechts neben dem Ego-Fahrzeug (m)
v_Fremdfahrzeug_neben_-links	uint8	$[0, 200]$	Geschwindigkeit des Fremdfahrzeugs links neben dem Ego-Fahrzeug (km/h)
s_Fremdfahrzeug_neben_-links	int16	$[-10, 200]$	Entfernung bis zum Fremdfahrzeug links neben dem Ego-Fahrzeug (m)
v_Fremdfahrzeug_hinten	uint8	$[0, 200]$	Geschwindigkeit des Fremdfahrzeugs hinter dem Ego-Fahrzeug (km/h)
s_Fremdfahrzeug_hinten	uint8	$[0, 200]$	Entfernung bis zum Fremdfahrzeug hinter dem Ego-Fahrzeug (m)
v_Fremdfahrzeug_vorne_-rechts	uint8	$[0, 200]$	Geschwindigkeit des Fremdfahrzeugs vor dem Ego-Fahrzeug rechts (km/h)
s_Fremdfahrzeug_vorne_-rechts	uint8	$[0, 200]$	Entfernung bis zum Fremdfahrzeug vor dem Ego-Fahrzeug rechts (m)
v_Fremdfahrzeug_vorne_-links	uint8	$[0, 200]$	Geschwindigkeit des Fremdfahrzeugs vor dem Ego-Fahrzeug links (km/h)
s_Fremdfahrzeug_vorne_-links	uint8	$[0, 200]$	Entfernung bis zum Fremdfahrzeug vor dem Ego-Fahrzeug links (m)
NebenAufAbfahrt	boolean	$[0, 1]$	Es wird der Wert '1' gesetzt, soweit das Ego-Fahrzeug sich neben einer Auf- bzw. Abfahrt befindet
Fahrzeug_neben _rechts	boolean	$[0, 1]$	Ein Fremdfahrzeug befindet sich rechts neben dem Ego-Fahrzeug
Fahrzeug_neben _links	boolean	$[0, 1]$	Ein Fremdfahrzeug befindet sich links neben dem Ego-Fahrzeug
Fahrzeug_hinten	boolean	$[0, 1]$	Ein Fremdfahrzeug befindet sich hinter dem Ego-Fahrzeug
Fahrzeug_vorne_rechts	boolean	$[0, 1]$	Ein Fremdfahrzeug befindet sich rechts vor dem Ego-Fahrzeug

Fahrzeug_vorne_rechts	boolean	[0, 1]	Ein Fremdfahrzeug befindet sich links vor dem Ego-Fahrzeug

Ausgänge

Name	Datentyp	Wertebereich	Beschreibung
Wechselverbot_links	boolean	[0, 1]	Ein Wechselverbot nach links liegt vor
Wechselverbot_rechts	boolean	[0, 1]	Ein Wechselverbot nach rechts liegt vor

Beschreibung

Der Automat `Fremdfahrzeugauswertung` (siehe Abbildung A.34) besteht aus dem Zustand `Verkehrauswertung`, in dem mit Hilfe der Fahrspurinformation des Ego-Fahrzeugs geprüft wird, ob sich Fremdfahrzeuge im Sicherheitsabstand des Ego-Fahrzeugs befinden. Zum Beispiel: Das Ego-Fahrzeug fährt auf der rechten Fahrspur (`Fahrspur`='1') und befindet sich nicht neben einer Auf- bzw. Abfahrt (`nebenaufabfahrt`='0'). Es wird geprüft, ob sich links neben dem Ego-Fahrzeug, hinter dem Ego-Fahrzeug und links vor dem Ego-Fahrzeug Fremdfahrzeuge befinden. Wenn an einer dieser Stellen ein Fremdfahrzeug fährt, wird zusätzlich geprüft, ob der Sicherheitsabstand zum Ego-Fahrzeug unterschritten ist.

4.4.2.4 Auswertung Verbote

Das Submodul `Auswertung_Verbote` (siehe Abbildung A.35) steuert die Leuchtelemente des LCA an. Dafür werden die aus Abschnitt 4.4.2.1 und Abschnitt 4.4.2.3 ermittelten Informationen über `Fahrspur`, `Linienverbot`, `Wechselverbot_links` und `Wechselverbot_rechts` sowie die Information, ob das Ego-Fahrzeug sich neben einer Ausfahrt befindet, berücksichtigt.

Eingänge

Name	Datentyp	Wertebereich	Beschreibung
Linienverbot	boolean	[0, 1]	Spurwechselverbot, aufgrund einer durchgezogenen Linie
Fahrspur	uint8	[0, 3]	Die Fahrspur auf der das Ego-Fahrzeug fährt
NebenAufAbfahrt	boolean	[0, 1]	Es wird der Wert '1' gesetzt, solange das Ego-Fahrzeug sich neben einer Auf- bzw. Abfahrt befindet
Ausfahrt	boolean	[0, 1]	Es wird der Wert '1' gesetzt, solange das Ego-Fahrzeug sich neben einer Ausfahrt befindet

Wechselverbot_links	boolean	[0, 1]	Ein Wechselverbot nach links liegt vor
Wechselverbot_rechts	boolean	[0, 1]	Ein Wechselverbot nach rechts liegt vor
Geschwindigkeit_-egofahrzeug	uint8	[0, 160]	Geschwindigkeit des Ego-Fahrzeugs (km/h)

Ausgänge

Name	Datentyp	Wertebereich	Beschreibung
Signalleuchte_links	uint8	[0, 2]	Die Anzeige einer Fahrspurwechselmöglichkeit auf die linke Fahrspur, ist folgendermaßen kodiert: • 0: LCA ist deaktiviert • 1: Fahrspurwechsel ist möglich • 2: Fahrspurwechsel ist nicht möglich
Signalleuchte_rechts	uint8	[0, 2]	Die Anzeige einer Fahrspurwechselmöglichkeit auf die rechte Fahrspur
Leuchtelement_LCA	boolean	[0, 1]	Anzeige über Aktivierung bzw. Deaktivierung des LCA
Leuchtelement_LCA_Blink	boolean	[0, 1]	Anzeige über Fehlermeldung bei der ungültigen Aktivierung des LCA

Beschreibung

Der Automat `Verbotauswertung` (siehe Abbildung A.36) startet initial im Zustand `LCA_Aus`. Bei der Aktivierung des LCA mit einer Geschwindigkeit des Ego-Fahrzeugs unter 60 km/h wechselt der Automat in den Zustand `Blink`. Daraufhin werden die beiden Ausgänge `Leuchtelement_LCA_Blink` und `Leuchtelement_LCA` auf '1' gesetzt. Nachdem das Leuchtelement für das LCA dreimal rot blinkt, wechselt der Automat wieder in den Initialzustand.

Wird das Bedienelement `LCA` bei einer Geschwindigkeit des Ego-Fahrzeugs von 60 km/h oder mehr betätigt, wechselt der Automat in den Zustand `LCA_An`. Dieser Zustand wertet die Informationen über die Fahrspur des Ego-Fahrzeugs und die bestehenden Linien- und Wechselverbote aus und steuert die LCA-Leuchtelemente entsprechend an. Hierbei wird auch berücksichtigt, ob sich das Ego-Fahrzeug neben einer Ausfahrt befindet. Sobald die Geschwindigkeit des Ego-Fahrzeugs unter 60 km/h sinkt oder das Bedienelement `LCA` erneut betätigt wird, wechselt der Automat wieder in den Initialzustand und setzt alle Ausgänge des Automaten auf den Wert '0'.

4.4.3 Multiplexer

Der Multiplexer dient zur Priorisierung der System- und Benutzereingaben.

Eingänge

Name	Datentyp	Wertebereich	Beschreibung
Fahrer_Brems	double	$[0, 10]$	Bremspedalstellung, die durch den Fahrer angelegt wird.
Fahrer_Gas	double	$[0, 1]$	Gaspedalstellung, die durch den Fahrer angelegt wird.
ACC_Brems	double	$[0, 10]$	Bremspedalstellung, die durch das `ACC` angelegt wird.
ACC_Brems	double	$[0, 1]$	Gaspedalstellung, die durch das `ACC` angelegt wird.

Ausgänge

Name	Datentyp	Wertebereich	Beschreibung
Gaspedalstellung	double	$[0, 1]$	Gaspedalstellung, die an die ViDAs-Umweltschnittstelle übergeben wird.
Bremspedalstellung	double	$[0, 10]$	Bremspedalstellung, die an die ViDAs-Umweltschnittstelle übergeben wird.
Bremspedal	double	$[0, 1]$	Zeigt an, ob das Bremspedal betätigt wird.

Beschreibung

Der Multiplexer (siehe Abbildung A.37) priorisiert die vier Eingänge von der höchsten zur niedrigsten Priorität in der folgenden Reihenfolge : `Fahrer_Brems`, `Fahrer_Gas`, (`ACC_Brems` und `ACC_Gas`). Die Bremspedalstellung, die durch den Fahrer angelegt wird, wird, sobald ungleich '`0`' (Block 4), an den Ausgang `Bremspedalstellung`, also an die ViDAs-Umweltschnittstelle, weitergegeben (Block 7) und verursacht, dass der Ausgang `Gaspedalstellung` genullt wird (Block 10). Weiterhin wird am Ausgang `Bremspedal` eine Betätigung des Bremspedals durch den Fahrer angezeigt (Block 9). Die nächstniedrigere Priorität hat der Eingang `Fahrer_Gas`. Sobald dieser größer '`0`' (Block 3) und der Eingang `Fahrer_Brems` gleich '`0`' sind, wird der Ausgang `Bremspedalstellung` genullt. Ob `Fahrer_Gas` am Ausgang `Gaspedalstellung` anliegt, hängt zusätzlich noch davon ab, ob `Fahrer_Gas` größer ist als `ACC_Gas` (Block 2). Die niedrigste Priorität besitzen die Eingänge `ACC_Brems` und `ACC_Gas`. `ACC_Brems` wird durch die Blöcke 5 und 7 an den Ausgang `Bremspedalstellung` weitergegeben, wenn an den Eingängen des Fahrers kein Wert größer '`0`' anliegt. Zusätzlich zu derselben Bedingung muss `ACC_Gas` größer sein

als `Fahrer_Gas` (Block 2), um am Ausgang `Gaspedalstellung` angelegt zu werden (Block 6). Die Aufgabe, zu vermeiden, dass an beiden Ausgängen Werte ungleich '`0`' anliegen, übernimmt das `ACC`, da dies jeweils einem Ausgang den Wert '`0`' zuweist.

5 Testen

Wie aus der bereits in Abschnitt 2.1 gezeigten Abbildung des V-Modells (siehe Abbildung 2.1) hervorgeht, folgt bei diesem Vorgehensmodell nach der Implementierungsphase die Testphase. Ebenfalls ist dort die Aufteilung der Testphase in die Modultests, Integrationstests und schließlich die Systemtests erkennbar.

Aufgrund des bevorstehenden zweiten Reviews wurde entschieden, dass diese strikte Trennung nicht für das gesamte Fahrerassistenzsystem eingehalten werden soll. Stattdessen wurde beschlossen, die Trennung für das ACC und das LCA separat einzuhalten. So war es möglich, bereits beim zweiten Review ein Submodul zu präsentieren, für das sowohl die Modul- als auch die Integrationstests vollständig durchgeführt wurden, statt für beide Submodule lediglich die Modultests durchgeführt zu haben. Genaueres dazu folgt im Abschnitt 5.2.

5.1 Testdatenbank

Um die erstellten Testfälle verwalten zu können, wurde ein Datenbanksystem verwendet, das mit Hilfe eines übersichtlichen Web-Interfaces allen Mitgliedern der Projektgruppe einfachen Zugriff ermöglichte. Zu diesem Zweck wurde das vorhandene System der ehemaligen Projektgruppe „easyDrive“ verwendet. Somit bestand ein solides Basissystem, das nach einigen Anpassungen und Ergänzungen vollkommen der Problemstellung dieser Projektgruppe gerecht wurde.

Die Datenbank selbst wird über eine MySQL-Datenbank mit drei Tabellen realisiert. Es gibt die Tabelle `creators`, in der die Namen der Projektgruppen-Mitglieder gespeichert sind, die Tabelle `testcase`, in der die Texte der Testfälle gespeichert sind, und die Tabelle `results`, in der sich die Ergebnisse zu den zugehörigen Testfällen befinden.

Das Web-Interface unterstützt die folgenden Funktionen:

- Testfälle und Untertestfälle anlegen und löschen
- Ergebnisse zu Testfällen und Untertestfällen anlegen und löschen
- Testfalltypen festlegen („Modul“, „Integration“, „System“)
- Testfälle als „erfolgreich“ oder „nicht erfolgreich“ markieren
- Anzeige des Status der Datenbank

Weiterhin besteht die Möglichkeit, automatisch einen Testbericht erzeugen zu lassen, der alle eingetragenen Testfälle inklusive ihrer Ergebnisse enthält.

Legende	
	Ergebnisse anzeigen
	Gültiges Ergebnis vorhanden
	Kein gültiges Ergebnis vorhanden
	Eintrag bearbeiten
	Neuen Untertestfall anlegen
	Testfall rekursiv kopieren
	Eintrag löschen
	Version erhöhen
TC	Zahl der Testfälle
RS	Zahl der Ergebnisse
PH	Zahl der Seitenaufrufe
E	Zahl der erfolgreichen Ergebnisse
NE	Zahl der nicht erfolgreichen Ergebnisse

(a) Legende des Webinterfaces

Name	TC	RS	PH
Alex Enkelmann	67	68	1867
Andreas Gabel	108	136	1610
Christoph Laesche	154	267	3104
Günter Ehmen	0	0	528
Jens Heyen	0	0	56
Jianyu Bao	8	92	925
Peter Battram	103	113	2096
Sven Sieverding	158	165	2485
Thilo Koepke	122	243	5075
Summe	**720**	**1084**	**17746**

Testarten/Ergebnisse			
Art	**TC**	**E**	**NE**
Modultests	503	503	0
Integrationstests	210	210	0
Systemtests	7	7	0

(b) Datenbankinhalt

Abbildung 5.1: Legende des Test-Datenbank-Webinterfaces und Übersicht des Datenbank-Inhalts

Aufgrund der vorhandenen Möglichkeit, zu einem Testfall Untertestfälle anzulegen, wurde beschlossen, die Testfälle in einer baumartigen Struktur anzulegen.

Um eine übersichtliche Struktur zu schaffen, wurde ein Meta-Testfall mit dem Titel „Das Fahrerassistenzsystem funktioniert“ angelegt. Dieser, nach dem Ziel der Projektgruppe benannte, Meta-Testfall beinhaltet alle anderen Testfälle und wurde erst als erfolgreich markiert, nachdem alle enthaltenen Testfälllle als erfolgreich markiert waren. Die Modul-, Integrations- und Systemtestfälle wurden vollständig als Untertestfälle dieses Testfalls angelegt. Wurden alle einem Testfall untergeordneten Testfälle erfolgreich durchgeführt, wurde der übergeordnete Testfall als erfolgreich markiert.

Die Legende des Web-Interfaces und die Statusanzeige der Test-Datenbank sind in Abbildung 5.1 dargestellt, ein kleiner Ausschnitt der Testdatenbank, auf dem die Struktur der Testfälle erkennbar ist, ist in Abbildung 5.2 zu sehen.

ID	Bezeichnung	Testtyp	Version	Referenz	Ergebnisse	Neuen Testfall anlegen
72	Das Fahrerassistenzsystem funktioniert	Modul	0.1		1	
74	ACC-Teilsystem funktioniert	Modul	0.3	ACC-Modul	1	
440	Abstandsregler funktioniert	Modul	0.2	ACC-Modul	1	
439	Geschwindigkeitslogik funktioniert	Modul	0.2	ACC-Modul	1	
537	Entscheidung Geschwindigkeit (1)	Modul	0.1	ACC-Modul	1	
538	Entscheidung Geschwindigkeit (1.1)	Modul	0.1	ACC-Modul	1	
539	Entscheidung Geschwindigkeit (2)	Modul	0.1	ACC-Modul	1	

Abbildung 5.2: Ausschnitt aus der Testdatenbank

Nachdem alle Testfälle erfolgreich durchgeführt worden waren, konnte auch der Meta-Testfall

„Das Fahrerassistenzsystem funktioniert“ als erfolgreich markiert werden.

Durch diese Form der Strukturierung ergibt sich ein geordneter und übersichtlicher Aufbau des automatisch generierten Testberichts.

5.2 Testplanung und -durchführung

Um die Testphase zu planen, wurde zusätzlich zum Projektplan ein separater Testplan erstellt. Wie auch schon bei der Erstellung des Projektplans wurde zu diesem Zweck die Software *GanttProject* verwendet. Beide Pläne sind im Anhang unter A.3 und A.2 zu finden.

Ein separater Plan der Testphase wurde angelegt, um eine bessere Übersicht zu schaffen. Es wäre ebenfalls möglich gewesen, diesen in den Testplan zu integrieren. Außerdem konnten die Gruppenmitglieder den einzelnen Vorgängen zugewiesen werden. Auf diese Weise wusste jedes Mitglied der Projektgruppe, welcher Aufgabe es zugeteilt war, und es konnte gleichzeitig darauf geachtet werden, dass keinem Mitglied eine zu hohe Arbeitsbelastung zugemutet wurde.

Um eine angemessene Arbeitsbelastung sicherzustellen, wurde der Testplan in Abstimmung mit dem Projektplan erstellt und gepflegt. Zur weiteren Vorbereitung der Testphase wurde außerdem, auf Anregung aus dem ersten Review, der sogenannte *Code of Practice for the Design and Evaluation of Advanced Driver Assistance Systems* [Res06] durchgearbeitet, sowie eine *Sicherheitsanalyse* [Pro10d] durchgeführt.

Code of Practice

Das von *response 3* herausgegebene Dokument zum Code of Practice wurde von einem Team aus Gruppenmitgliedern zur Vorbereitung der Erstellung von Testfällen vollständig durchgearbeitet.

Dabei stellte sich heraus, dass vieles dort Angesprochene bereits bei der Erstellung der vorläufigen Testfälle in der Anforderungsdefinition berücksichtigt worden war. Allerdings erschien der Projektgruppe das dort vorgestellte Verfahren zur Konstruktion von Situationen als sehr hilfreich. Das Verfahren schlägt vor, sich bei der Konstruktion von Situationen über folgende Fragestellungen Gedanken zu machen:

- Wo soll das System funktionieren?
- Wo könnte das System fälschlicherweise benutzt werden?
- Wo hat das System seine Grenzen?
- Welche Situationen könnten durch Fehlfunktionen zustande kommen?

Bei der Beantwortung entstand eine Großzahl an Situationen, allerdings waren davon nicht alle relevant für die Sicherheit des Fahrerassistenzsystems. Deshalb wurden anschließend die für die von der Projektgruppe festgelegten Anforderungen an das Fahrerassistenzsystem relevanten Situationen ausgewählt und zunächst für die Erstellung der Sicherheitsanalyse benutzt.

Sicherheitsanalyse

Bei der Sicherheitsanalyse entschied sich die Projektgruppe für die Durchführung einer *Fehlermöglichkeits- und Einflussanalyse* (FMEA), da diese Methode einen Standard in der Automobilindustrie darstellt und sowohl ihr Erstellungsaufwand als auch ihr Umfang im Vergleich zu anderen Methoden relativ gering sind.

Zwar wird bei dieser Methode ausdrücklich darauf hingewiesen, dass Fehlermöglichkeiten bei der Erstellung übersehen werden können, den Mitgliedern der Projektgruppe waren bei der Methodenauswahl allerdings die beiden zuvor genannten Vorteile wichtiger, da die Sicherheitsanalyse als transversaler Prozess neben der regulären Arbeit erstellt und gepflegt werden musste und somit im Hinblick auf die beschränkte Projektlaufzeit und die Größe der Projektgruppe Wert auf ein unkompliziertes Verfahren gelegt wurde.

Mit der Erstellung beschäftigte sich ein eigenes Team aus drei Gruppenmitgliedern. Im weiteren Verlauf der Projektgruppe wurde das Dokument regelmäßig von der gesamten Gruppe aktualisiert. Die möglichen Fehler wurden dabei für jede Funktion des Fahrerassistenzsystems konstruiert und anschließend nach folgenden Kriterien bewertet:

- Auftreten von Fehlern
- Bedeutung von Fehlern
- Entdeckung von Fehlern

Diese Kriterien wurden mit Zahlenwerten bewertet, wie in den Tabellen 5.1 und 5.2 dargestellt.

Zahlenwert	**Legende**
10/9	sehr hoch
8/7	hoch
6/5/4	mäßig
3/2	gering
1	sehr gering

Tabelle 5.1: Bewertung von Auftretenswahrscheinlichkeit und Bedeutung möglicher Fehler

Zahlenwert	**Legende**
10/9	sehr gering
8/7	gering
6/5/4	mäßig
3/2	hoch
1	sehr hoch

Tabelle 5.2: Bewertung der Entdeckungswahrscheinlichkeit möglicher Fehler

Aus dem Produkt der drei Zahlenwerte für die Auftretenswahrscheinlichkeit, die Bedeutung und die Entdeckungswahrscheinlichkeit eines möglichen Fehlers errechnet sich schließlich die

sogenannte *Risikoprioritätszahl* (RPZ), die Aufschluss über die Risikohöhe des betreffenden möglichen Fehlers gibt.

So lässt sich auch erklären, wieso die Skala bei der Entdeckung der Fehler umgekehrt eingeteilt ist als bei Auftreten und Bedeutung. Denn ein Fehler, der leicht zu entdecken ist (bei dem also die Entdeckung „sehr hoch“ ist), wirkt sich auf die Risikoprioritätszahl nicht so sehr aus, wie ein Fehler, der nur sehr schwer zu entdecken ist. Die FMEA half dann bei der Erstellung von Testfällen für das System. Die vollständige FMEA ist im Anhang unter A.4 zu finden.

Beginn der Testphase

Die eigentliche Testphase begann, wie dem Testplan im Anhang unter A.3 zu entnehmen ist, nach Beendigung der Implementierungsphase. Da zu diesem Zeitpunkt das zweite Review bevorstand, wurde der Entschluss gefasst, eines der beiden Submodule komplett durchzutesten. Das bedeutete, dass sowohl die Modultests, als auch die Integrationstests mit dem in Abschnitt 4.3 erläuterten ViDAs-Komm-Modul, durchgeführt worden sein sollten. Das fertig getestete und integrierte Submodul sollte dann im zweiten Review präsentiert werden.

Beginn der LCA-Testphase

Die Projektgruppe entschied sich für das LCA-Submodul, da der Zeitrahmen eingeschränkt war und das LCA weniger komplex ist als das ACC. Der Testplan sah vor, dass zunächst die Modultestfälle für das LCA-Submodul und das ViDAs-Komm-Modul entwickelt, möglichst präzise formuliert und anschließend in die Test-Datenbank eingetragen wurden.

Dies geschah in kleinen Teams aus Gruppenmitgliedern. Ein Team überlegte anhand der Anforderungen an das LCA-Submodul, welche Fehlfunktionen, Sonderfälle oder auch Benutzungsfehler auftreten könnten. Für das ViDAs-Komm-Modul ging ein weiteres Team analog vor. So wurde versucht, eine möglichst vollständige Testabdeckung zu erreichen. Die Testabdeckung des LCA-Submoduls ist in Tabelle 5.3 dargestellt.

Anforderung	**Testbericht-Kapitel**
Das System muss dem Fahrer beim Fahrspurwechsel assistieren.	1.592 - 1.651
Das System muss dem Fahrer beim Einfädeln auf die Autobahn assistieren.	1.593, 1.602, 1.603, 1.609, 1.613, 1.614, 1.620, 1.621, 1.622, 1.623
Das System weist den Fahrer darauf hin, ob ein Fahrspurwechsel möglich ist.	1.562 - 1.591

Tabelle 5.3: Abdeckung der LCA-Anforderungen mit Testfällen

Nach der Entwicklung der Testfälle wurden diese in neu zusammengestellten Teams durchgeführt. Dabei wurde versucht sicherzustellen, dass niemand selbst aufgestellte Testfälle durchführte, was allerdings aufgrund der sehr eingeschränkten Gruppengröße nicht immer umzusetzen war.

Während die LCA-Modultests durchgeführt wurden, beschäftigte sich parallel ein Team mit der Erstellung der Integrationstests von LCA-Submodul und ViDAs-Komm-Modul, die im Anschluss ebenfalls durchgeführt wurden.

Durchführung der LCA-Modultests

Für die Durchführung der Modultests des LCA-Matlab-Modells wurde eine Testumgebung in Matlab modelliert, in der an die Eingangsports des Modells Werte angelegt werden konnten, wodurch an den Ausgangsports die vom LCA-Submodul erzeugten Ausgangswerte anlagen, die automatisch in einer Textdatei protokolliert und gespeichert wurden. Die Werte für die Eingangsports wurden in Matrixform in eine m-Datei geschrieben. Die Werte für die Matrix konnten aus den Texten der Testfälle entnommen werden. Dabei entsprach jede Zeile der Matrix einem eigenen Testfall.

Die Werte der Matrix wurden zeilenweise an die Eingangsports des Modells angelegt und die daraus resultierenden Werte an den Ausgangsports wurden in eine Logdatei geschrieben.

So konnte das tatsächliche Ergebnis mit dem erwarteten Ergebnis abgeglichen werden. Der Ablauf dieser automatischen Matlab-Modelltests ist in Abbildung 5.3 schematisch dargestellt.

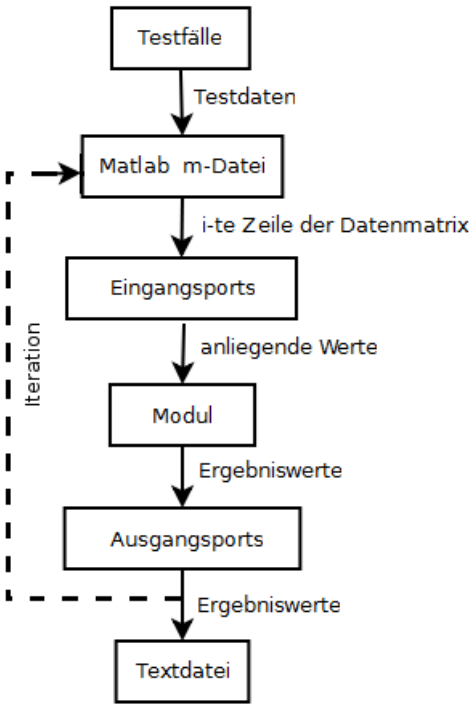


Abbildung 5.3: Prinzip der automatischen Matlab-Modelltests

Wenn die beiden Ergebnisse nicht übereinstimmten, wurde der Testfall als „nicht erfolgreich“ in die Testdatenbank eingetragen und dem verantwortlichen Gruppenmitglied dies mitgeteilt. Daraufhin untersuchte das verantwortliche Gruppenmitglied die Fehlerursache. Dabei wurde zunächst kontrolliert, ob das erwartete Ergebnis fehlerhaft formuliert war, ob bei der Aufstellung der Matrix ein Fehler passiert war oder ob die Testumgebung fehlerhaft modelliert wurde.

Wenn diese ersten Fehlerquellen ausgeschlossen werden konnten, wurde das Modell des LCA-Submoduls selbst untersucht und dort die Fehlerursache gesucht und behoben.

Im Anschluss daran wurden alle Testfälle erneut durchgeführt, um zu testen, ob nach einer Änderung am Modell vorher fehlgeschlagene Testfälle erfolgreich durchgeführt werden konnten und ob bereits zuvor erfolgreich durchgeführte Testfälle auch nach der Änderung noch erfolgreich durchgeführt werden können.

Beginn der ACC-Testphase

Im zweiten Teil der Testphase wurde für das ACC-Submodul das in der LCA-Testphase beschriebene Vorgehen erneut durchgeführt. Es wurden also zunächst Modultestfälle für das ACC-Submodul erstellt und parallel eine Testumgebung für die automatische Durchführung der Modultestfälle modelliert. Die Testüberdeckung für das ACC-Submodul ist in Tabelle 5.4 zu finden.

Anforderung	**Testbericht-Kapitel**
Der Sicherheitsabstand zum vorausfahrenden Fremdfahrzeug muss bei aktiviertem ACC stets groß genug sein.	1.684, 1.688, 1.689, 1.690
Das ACC muss erkannte Geschwindigkeitsbeschränkungen berücksichtigen.	1.6, 1.7, 1.12, 1.42 - 1.58, 1.687, 1.690
Das System muss stets die aktuelle Geschwindigkeitsbegrenzung kennen und wissen, ob es ein Überholverbot gibt.	1.42 - 1.58
Das ACC muss bis zu einer Geschwindigkeit des Ego-Fahrzeugs von $v_{ego,max}$ funktionieren.	779, 781, 782, 788
Das ACC muss ab einer Geschwindigkeit des Ego-Fahrzeugs von $v_{acc,min}$ aktivierbar sein.	1.60 - 1.71

Tabelle 5.4: Abdeckung der ACC-Anforderungen mit Testfällen

Durchführung der ACC-Modultests

Auch diese Modultestfälle wurden anschließend von einem eigenen Team, wie bereits in der LCA-Modultestphase beschrieben, mit Hilfe der erstellten Testumgebung und einer m-Datei mit Testfallmatrix durchgeführt. Während der Durchführung der Modultests beschäftigten sich zeitgleich weitere Teams aus Gruppenmitgliedern mit der Erstellung der Integrationstestfälle von ACC-Submodul und ViDAs-Komm-Modul, sowie mit der Erstellung einer Testumgebung für automatisierte Tests des ViDAs-Kerns. Da das ViDAs-Komm-Modul bereits in der LCA-Testphase erfolgreich getestet wurde, gab es in der ACC-Testphase hierfür nicht erneut gesonderte Tests, sondern es wurde als funktionierend vorausgesetzt.

Abschließend wurden auch bei diesen Tests die tatsächlichen Ergebnisse mit den erwarteten

Ergebnissen abgeglichen, gegebenenfalls die aufgetretenen Fehler gesucht und behoben und anschließend die Ergebnisse in die Test-Datenbank eingetragen.

Beginn der ViDAs-Kern-Testphase

Nachdem für das LCA-Submodul und das ACC-Submodul die Modul- und Integrationstests durchgeführt worden waren, konnten die Testfälle für den gesamten ViDAs-Kern aufgestellt werden. Dabei fiel auf, dass diese Testfälle sehr redundant zu den ACC- und LCA-Modultests waren, da die beiden Submodule weitestgehend autonom voneinander funktionieren und lediglich die Information der erkannten Fahrspur vom ACC an das LCA weitergegeben wurde. Deshalb wurde entschieden, nur einige repräsentative Testfälle aufzustellen und durchzuführen, da aufgrund der erfolgreich durchgeführten Modultests beider Submodule sichergestellt war, dass beide Submodule korrekt funktionieren. Diese repräsentativen Testfälle betrafen fahrspurrelevante Verkehrsschilder, sowie den korrekten Betrieb beider Submodule simultan. Genaueres zu den Tests des ViDAs-Kerns kann dem Testbericht [Pro10e] in den Kapiteln 1.691-1.713 entnommen werden.

Test der Kommunikation zwischen ViDAs-Kern und ViDAs-Umweltschnittstelle

Da die Kommunikation zwischen MATLAB und SILAB zunächst mit Hilfe des ViDAs-Komm-Moduls und der bereits in Abschnitt 4.3.2 vorgestellten HLA-Schnittstelle realisiert wurde, wurde diese HLA-Schnittstelle ebenfalls verwendet, um die Kommunikation zwischen ViDAs-Kern und ViDAs-Umweltschnittstelle zu testen.

Dieses Vorgehen bot sich an, da man auf diese Weise sehen konnte, wie die gesendete Nachricht aufgebaut war, und wie sie beim Empfänger ankam.

Zu diesem Zweck wurde eine Testumgebung entwickelt, die auf der einen Seite des ViDAs-Komm-Moduls aus einem eigenen Matlab-Modell bestand, das jedoch die m-Dateien des ViDAs-Kern-Komm-Adapters verwendete.

Der Sinn des eigens zu diesem Zweck erstellten Modells lag darin, selbst Werte an die Ausgänge anlegen zu können. Empfangene Nachrichten wurden direkt in Matlab ausgegeben.

Auf der anderen Seite des ViDAs-Komm-Moduls befand sich der SILAB-Fahrsimulator. Dieser wurde für die Kommunikationstests so konfiguriert, dass Nachrichten, die gesendet oder empfangen wurden, entsprechend gekennzeichnet in eine Datei geschrieben wurden, da eine direkte Ausgabe in SILAB aus technischen Gründen nicht möglich ist.

So konnten zum Einen auf beiden Seiten Nachrichten definiert werden, die versendet werden sollten. Zum Anderen konnten empfangene und gesendete Nachrichten auf ihre Gleichheit überprüft werden.

Auf diese Weise konnte sehr einfach erkannt werden, ob die Kommunikation korrekt funktionierte. Das Prinzip der Testumgebung ist in Abbildung 5.4 verdeutlicht. Diese Testumgebung wurde ebenfalls dazu verwendet, die Integrationstests zwischen LCA-Submodul, ViDAs-Komm-Modul und ViDAs-Umweltschnittstelle sowie zwischen ACC-Submodul, ViDAs-Komm-Modul und ViDAs-Umweltschnittstelle durchzuführen. Zu diesem Zweck wurde das Matlab-Modell der Testumgebung durch das Modell des LCA-Submoduls beziehungsweise des ACC-Submoduls ersetzt.

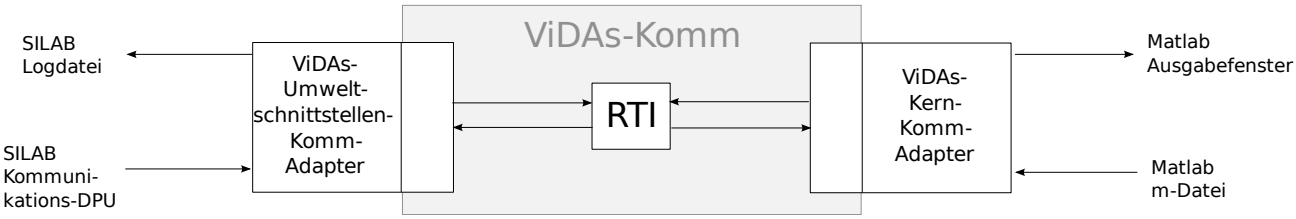


Abbildung 5.4: Prinzip der Kommunikationstests

Im Verlauf des Projekts traten jedoch Synchronisationsprobleme zwischen Matlab und SILAB auf (siehe Abschnitt 4.3.2.5), so dass das Fahrerassistenzsystem auf Eingaben des Benutzers nur verspätet reagierte.

Deshalb wurde beschlossen, die HLA-Kommunikation zu ersetzen und das Fahrerassistenzsystem vollständig als SILAB-DPU zu realisieren. Näheres hierzu kann in Abschnitt 4.3.3 nachgelesen werden.

Aufgrund dieser Entscheidung stellte anschließend lediglich eine Funktion im Programmcode der DPU die Kommunikation dar. Da sie aber nach der Umstellung dennoch erneut getestet werden musste, um nach wie vor die korrekte Funktionsweise zu gewährleisten, entschied sich die Projektgruppe dafür, die erneuten Tests anhand eines Code-Reviews durchzuführen.

Der Fragenkatalog und die Bewertungen sind in Tabelle 5.5 zu finden.

Dazu schaute sich ein Team aus drei Projektgruppenmitgliedern, die den Code nicht erstellt hatten, den Programmcode an und beurteilten anhand des erarbeiteten Fragenkatalogs, ob der Code den Anforderungen entsprach. Da es sich nur um einen kleinen Teil des gesamten Codes handelte, der untersucht werden musste, fiel der Fragenkatalog entsprechend kurz aus.

Kriterium	**Ja**	**Nein**
Sind undefinierte oder unbenutzte Variablen vorhanden?		X
Wird allen Variablen der richtige Wert zugewiesen?	X	
Sind Tippfehler im Code enthalten?		X
Sind den Bedienelementen die richtigen Tasten zugeordnet?	X	
Wurden vorhandene Typecasts korrekt durchgeführt?	X	
Ist die Ansteuerung der Leuchtelemente richtig?	X	
Könnte eine Endlosschleife auftreten?		X
Löst das Programm das Problem?	X	
Ist die Problemlösung effizient umgesetzt?	X	
Deckt das Programm alle nötigen Fälle ab?	X	
Stilrichtlinien: Wurden die Einrückungen eingehalten?	X	
Stilrichtlinien: Eine Anweisung pro Zeile?	X	

Tabelle 5.5: Protokoll des Code-Reviews

Mit Hilfe des Code-Reviews konnten einige kleine Fehler erkannt und anschließend erfolgreich behoben werden, die zum Großteil jedoch auf kleinere, nachträgliche Änderungen des Matlab-Modells zurückzuführen waren.

Anschließend wurde in der Test-Datenbank ein zusätzlicher Testfall angelegt und die Auswertung des Code-Reviews als Ergebnistext eingetragen. Dieser Testfall ist im Testbericht [Pro10e] in Abschnitt 1.427 zu finden.

Test der Sensorik

Um die in Abschnitt 4.2.1 erklärte Sensorik des Ego-Fahrzeugs zu testen, wurden eigene Teststrecken erstellt, wie in Abschnitt 5.3.2.2 beschrieben. Zum Teil waren diese mit stillstehenden Fremdfahrzeugen ausgestattet, um die Entfernungserkennung zu überprüfen und zum Teil waren Strecken mit beweglichen Fremdfahrzeugen vorhanden, um relevante Daten der Sensorik in eine Log-Datei schreiben zu können, die hinterher ausgewertet werden konnte. Auf diese Weise konnte festgestellt werden, ob die Sensoren wie gewünscht funktionieren.

Test des Fahrerassistenzsystems

Im letzten Teil der Testphase wurde das gesamte Fahrerassistenzsystem auf seine korrekte Funktionsweise geprüft.

Dazu wurden sieben Testszenarien entwickelt, die jeweils möglichst viele Testfälle vereinten. Welche Testfälle in welchem Testszenario vereint wurden, ist in Tabelle 5.6 dargestellt.

Szenario	**Testbericht-Kapitel**
Testszenario 1	1.3, 1.22, 1.59, 1.141, 1.173, 1.171, 1.174, 1.184, 1.350, 1.351, 1.427, 1.429, 1.667
Testszenario 2	1.173, 1.184, 1.427, 1.556, 1.562, 1.596, 1.601, 1.608, 1.612, 1.667
Testszenario 3	1.3, 1.22, 1.173, 1.184, 1.427, 1.556, 1.562, 1.596, 1.601, 1.608
Testszenario 4	1.3, 1.4, 1.22, 1.23, 1.42, 1.59, 1.141, 1.184, 1.427, 1.562, 1.592
Testszenario 5	1.3, 1.4, 1.22, 1.59, 1.141, 1.184, 1.427
Testszenario 6	1.3, 1.22, 1.184, 1.427, 1.556, 1.562 1.596, 1.601, 1.608, 1.612
Testszenario 7	1.3, 1.22, 1.59, 1.141, 1.184, 1.427, 1.530, 1.556, 1.562, 1.592, 1.596, 1.601, 1.608, 1.612

Tabelle 5.6: Abdeckung der Testfälle in den Szenarien

Zu diesen Testszenarien wurden entsprechende Fahrtstrecken in SILAB erstellt, auf denen die Szenarien durchgeführt wurden.

Zur Durchführung der Szenarien wurde zudem für jedes Szenario ein automatisches Fahrerskript geschrieben, welches das Fahrzeug selbstständig durch das jeweilige Testszenario steuern konn-

te. Eine genauere Erklärung dieser automatischen Fahrerskripte ist im Kapitel 5.3 zu finden. Anschließend wurden dann bei den Szenariodurchführungen relevante Werte für die Auswertung der Szenarios geloggt und in eine Textdatei geschrieben.

Der Inhalt dieser Logdatei wurde mit dem erwarteten Ergebnis abgeglichen und als Ergebnis für das jeweilige Szenario in die Test-Datenbank eingetragen.

Abschließend wurden sowohl für den in Abschnitt 4.4.1.6 beschriebenen Abstands- als auch den in Abschnitt 4.4.1.7 beschriebenen Geschwindigkeitsregler Stabilitätstests durchgeführt. Die Tests für die in den Abschnitten 4.4.1.2 und 4.4.1.5 beschriebenen prädiktiven Regler wurden begonnen, konnten aber nicht mehr innerhalb der Projektlaufzeit abgeschlossen werden.

Auch die Testfälle für die Regler sowie die zugehörigen Ergebnisse sind im Testbericht [Pro10e] zu finden.

5.3 Automatisches Testen

5.3.1 Gründe für das automatische Testen

Das automatische Testsystem (siehe Abschnitt 5.3.2) wurde entwickelt, um Testszenarien reproduzierbar durchführen zu können. Dies ist notwendig, da Tests immer unter den gleichen Bedingungen ausgeführt werden müssen. Ein erneutes Durchführen ist z.B. nach einer Fehlerkorrektur notwendig. Wird dabei das Fahrzeug von einem menschlichen Fahrer gesteuert, so ist die genaue Reproduktion der ursprünglichen Fahrt nicht möglich. Es könnte zu Problemen beim genauen Einhalten der Geschwindigkeit kommen oder Fahrspurwechsel könnten zu leicht unterschiedlichen Zeitpunkten durchgeführt werden. Somit herrschen andere Bedingungen für den Test. Hierdurch kann es zu fehlerhaften Testergebnissen kommen.

Diese Probleme wurden durch automatische Tests gelöst.

Ebenso ist es beim manuellen Fahren schwer, die genauen Ausgaben nachzuvollziehen, wenn diese nicht in eine Datei geschrieben werden, da diese evtl. nur für einen kurzen Moment sichtbar sind und sich danach wieder ändern können. Daher wurde auch eine Log-Funktion entwickelt, mittes der die Ausgaben festgehalten werden können.

5.3.2 Erstellung von Testszenarien

Für das automatische Testen wurde ein eigenes Testsystem entwickelt. Dieses ist in einer SILAB-DPU (siehe Abschnitt 4.1.2) implementiert. Im Folgenden wird diese DPU als Testskript bezeichnet. Dabei gibt es bereits einen vorgefertigten Rahmen in Form von Funktionen, mit denen das Fahrzeug automatisiert gesteuert werden kann. Es muss nur das Verhalten des Fahrzeuges mittels des Skripts nachgebildet werden, wie es in Abschnitt 5.3.2.1 genauer beschrieben wird.

Ebenso wurde eine Möglichkeit entwickelt, auf einfache Weise Teststrecken zu erstellen. Dies ist notwendig, da die Konfigurationsdateien des SILAB-Fahrsimulators eine zu große Auswahl

an Modellierungsmöglichkeiten für Strecken bieten. Dadurch ist der Umfang der für eine Teststrecke zu erstellenden SILAB-Konfigurationsdatei sehr groß. Bedingt durch die Komplexität der SILAB-Konfigurationsdatei steigt ebenfalls die Fehleranfälligkeit beim Erstellen von Teststrecken.

Daher wurde ein eigenes Konfigurationsformat entwickelt, welches die Streckenerstellung vereinfacht und aus dem automatisiert SILAB-Konfigurationsdateien erzeugt werden können. Da nur wenige der verfügbaren Elemente für Teststrecken benötigt wurden, mussten auch nur für diese Befehle implementiert werden. Einen Überblick über die verfügbaren Elemente und deren Benutzung gibt Abschnitt 5.3.2.2.

Zur Auswertung der Testfahrten wird eine Logdatei erzeugt, deren Aufbau im Abschnitt 5.3.2.3 genauer erläutert wird.

5.3.2.1 Einfügen eines Tests in das Testskript

Wie bereits in Abschnitt 4.1.2 beschrieben, wird in jedem Simulationsschritt die Funktion `trigger()` durch den Fahrsimulator aufgerufen. Um eine neue Testfahrt des Ego-Fahrzeugs mit Hilfe des Testskriptes durchzuführen, muss zunächst eine neue Funktion entwickelt werden, die aus dieser Funktion aufgerufen wird.

Dabei wird als Parameter die aktuelle Simulationszeit übergeben. Über diesen Parameter kann bestimmt werden, wie lange eine Aktion durchgeführt wird. Dazu kann die Differenz der aktuellen Simulationszeit zur Simulationszeit zu Beginn der Aktion genutzt werden. So kann z.B. einige Millisekunden lang eine Aktion ausgeführt werden und, sobald diese Zeit abgelaufen ist, eine andere.

Folgende Möglichkeiten der Steuerung des Ego-Fahrzeugs wurden implementiert, um das Erstellen von automatisierten Testfahrten zu vereinfachen:

Spur halten

Mittels der `stay()`-Funktion kann das Ego-Fahrzeug der aktuellen Spur folgen. Dies wird z.B. für Kurven benötigt.

Solange das Fahrzeug der Spur folgen soll, muss in jedem Simulationsschritt `stay()` aufgerufen werden.

Fahrspurwechsel

Die Funktion `lane_change()` führt einen Spurwechsel aus. Dabei erwartet die Funktion einen Parameter für die Richtung, in die der Spurwechsel durchgeführt werden soll. Dieser ist '`1`' für einen Wechsel auf die rechte Spur und '`2`' für einen Wechsel auf die linke Spur.

Die Funktion muss in jedem Simulationsschritt aufgerufen werden, bis der Spurwechsel komplett vollzogen wurde.

Beschleunigen und Bremsen

Die Steuerung der Geschwindigkeit erfolgt über die `accelerate()`-Funktion. Sie erwartet als Parameter die gewünschte Geschwindigkeit in km/h.

Ist dieser Wert größer als die aktuelle Geschwindigkeit, so beschleunigt die Funktion das Ego-Fahrzeug. Falls der Wert kleiner ist, setzt die Funktion die Beschleunigung auf den Wert '`0`', sodass das Ego-Fahrzeug ausrollt, bis die aktuelle Geschwindigkeit der gewünschten Geschwindigkeit entspricht.

Die Funktion muss in jedem Simulationsschritt aufgerufen werden, in dem eine Geschwindigkeit erreicht bzw. gehalten werden soll.

Ein Betätigen des Bremspedals ist in der Funktion nicht vorgesehen. Ist diese gewünscht, z.B. um zu testen, ob das ACC beim Betätigen der Bremse deaktiviert wird, muss die globale Variable `brakePedal` mindestens für einen Simulationsschritt auf einen gültigen Wert gesetzt werden. Danach kann der Wert wieder zurück auf '`0`' gesetzt werden.

Bedienelementbetätigung

Auch Bedienelemente können innerhalb des Testskripts betätigt werden. Hierzu dient die Funktion `tastendruck()`, die zwei Parameter erwartet.

Der erste Parameter bestimmt, welche Taste gedrückt werden soll. Gültige, ganzzahlige Werte liegen dabei im Bereich von '`1`' bis '`8`'. Diese entsprechen den F-Tasten der PC-Tastatur des Simulationsrechners, deren Funktionen in Tabelle 4.1 nachgelesen werden können.

Der zweite Parameter ist ein boolescher Wert. Ist er auf '`true`' gesetzt, so wird ein langes Betätigen des Bedienelements simuliert. Andernfalls wird das Bedienelement nur kurz betätigt.

5.3.2.2 Erstellen von Teststrecken

Zum einfachen Erstellen von Strecken in SILAB wurde ein eigenes Streckenerstellungs-System entwickelt. Um mit diesem eine Strecke zu entwerfen, muss eine Datei mit der Endung `.scn` angelegt werden. In dieser können die einzelnen Elemente der Strecke mittels den folgenden Befehlen eingefügt werden:

- `Gerade(<name>, <länge>)`:
  Erzeugt eine Gerade mit dem Namen `<name>` und einer Länge von `<länge>` Metern.
- `GeradeVerbot(<name>, <länge>)`:
  Erzeugt eine Gerade mit Spurwechselverbot mit dem Namen `<name>` und einer Länge von `<länge>` Metern.
- `Kurve(<name>, <länge>, <radius>)`:
  Erzeugt eine Kurve mit dem Namen `<name>`, einer Länge von `<länge>` Metern und einem Radius von `<radius>` Metern.
- `GeradeVorAuffahrt(<name>, <länge>)`:
  Erzeugt eine Gerade mit dem Namen `<name>` und einer Länge von `<länge>` Metern. Dabei ist die Standspur eine Auffahrt mit durchgezogener Linie auf der linken Seite.

- `StartAuffahrt(<name>, <länge>)`:
  Erzeugt eine Gerade mit dem Namen `<name>` und einer Länge von `<länge>` Metern. Dabei ist die Standspur eine Auffahrt mit gestrichelter Linie auf der linken Seite.
- `Abfahrt(<name>)`:
  Erzeugt eine Ab- und Auffahrt mit dem Namen `<name>`. Aus technischen Gründen könnten auf diesem Abschnitt keine anderen Objekte plaziert werden.

Dabei werden die Elemente in der Reihenfolge ihres Auftretens in der Strecken-Datei miteinander verknüpft. Zu beachten ist, dass die Namen der Elemente eindeutig sein müssen.

Ebenso können an der Strecke Verkehrsschilder positioniert werden. Dieses geschieht über den Befehl `Schild()`.

- `Schild(<abschnitt>, <typ>, <position>, [<typ_rechts>])`

Als Parameter wird dabei der Name des Streckenabschnitts, auf dem das Schild stehen soll, der Typ des Schild und die Position in Metern seit Beginn des Streckenabschnitts übergeben. Optional kann ein weiterer Parameter übergeben werden, mit dem ein anderer Schildtyp für die linke Fahrspur festgelegt werden kann.

Für das Erstellen und Steuern von Fremdfahrzeugen wurden ebenfalls Befehle entwickelt. Dies sind die folgenden:

- `Fahrzeug(<name>, <abschnitt>, <position>, <spur>, <geschwindigkeit>, <aktivierungspunkt>)`:
  Erzeug ein Fremdfahrzeug mir dem Namen `<name>`, welches im Streckenabschnitt mit dem Namen `<abschnitt>` `<position>` Meter nach Beginn der Strecke auf der Spur `<spur>` aufgesetzt wird, wobei '1' für die rechte und '2' für die linke Fahrspur stehen. Es hat dabei die Geschwindigkeit `<geschwindigkeit>` km/h und wird aktiviert, sobald das Ego-Fahrzeug `<aktivierungspunkt>` Meter auf dem Steckenabschnitt gefahren ist.
- `Geschwindigkeitsaenderung(<fahrzeug>, <abschnitt>, <position>, <geschwindigkeit>, <beschleunigung>, <abbremsung>)`:
  Veranlasst das Fremdfahrzeug mit dem Namen `<fahrzeug>`, die Geschwindigkeit zu ändern, sobald das Ego-Fahrzeug auf dem Streckenabschnitt `<abschnitt>` `<position>` Meter gefahren ist. Die neue Geschwindigkeit ist dabei `<geschwindigkeit>` km/h. Falls beschleunigt wird, erfolgt dies mit `<beschleunigung>` m/s. Falls abgebremst wird, erfolgt dies mit `<abbremsung>` m/s.
- `Spurwechsel(<fahrzeug>, <abschnitt>, <position>, <spur>, <strecke>)`:
  Sorgt dafür, dass das Fremdfahrzeug mit dem Namen `<fahrzeug>` einen Spurwechsel auf die Spur `<spur>` durchführt, sobald das Ego-Fahrzeug `<position>` Meter auf dem Streckenabschnitt `<abschnitt>` gefahren ist. Dabei benötigt der Spurwechsel `<strecke>` Meter.
- `Deaktivieren(<fahrzeug>, <abschnitt>, <position>)`:
  Deaktiviert ein Fremdfahrzeug, sobald das Ego-Fahrzeug `<position>` Meter auf dem Streckenabschnitt `<abschnitt>` gefahren ist. Ist das Fahrzeug deaktiviert, so verschwindet es von der Strecke.

Wurde eine solche Datei erzeugt, kann mit Hilfe eines Makefiles automatisiert aus der `.scn`-Datei eine `.cfg`-Datei erzeugt werden, die in SILAB geöffnet werden kann. Dabei wird auch automatisch das Testskript, welches im vorherigen Abschnitt beschrieben wurde, geladen.

5.3.2.3 Erstellung von Logdateien

Während der Simulation werden einige Daten in eine Logdatei geschrieben. Diese sind:

- Die aktuelle Simulationszeit
- Der Status der Leuchtelemente
- Die Wunschgeschwindigkeit
- Erkannte Verkehrsschilder und deren Abstand zum Ego-Fahrzeug
- Die aktuelle Geschwindigkeit
- Sensordaten zur Kurve (Abstand zum Ego-Fahrzeug und Radius)
- Sensordaten zu Fremdfahrzeugen

Um die zu sichernde Datenmenge für die Logdatei gering zu halten, wird vor dem Sichern geprüft, ob sich die Werte, abgesehen von der Simulationszeit, im Vergleich zum vorherigen Simulationsschritt geändert haben. Nur wenn sich mindestens ein Wert geändert hat, werden die Daten des Simulationsschritts in die Datei geschrieben.

Um genauer zu untersuchende Stellen, z.B. für explizite Testfälle, später in der Logdatei wieder zu finden, kann mit der Funktion `logtestnr()` ein numerischer Wert für den aktuellen Test in die Datei geschrieben werden. Ebenso werden Betätigungen von Bedienelementen in die Logdatei geschrieben.

Mit Hilfe dieser Marker sowie der Verfolgung der Werteänderungen können in der Logdatei die relevanten Stellen gefunden werden. Anhand der hier gespeicherten Werte kann geprüft werden, ob sie den an dieser Stelle erwarteten Werten entsprechen.

5.3.2.4 Durchführung eines automatischen Tests

Abbildung 5.5 zeigt eine Übersicht über den Ablauf des automatischen Testens.

Zunächst muss, wie in Abschnitt 5.3.2.2 beschrieben, eine Strecke erstellt werden. Aus dieser kann daraufhin automatisch eine SILAB-Konfigurationsdatei generiert werden.

Parallel dazu muss das Testskript erweitert werden. Dieses ist in Abschnitt 5.3.2.1 detailliert beschrieben.

Die Kombination aus der aus der Strecken-Datei generierten Konfigurationsdatei und des Testskripts ermöglicht das automatische Ausführen von Tests im SILAB-Simulator. Dazu muss lediglich diese generierte Konfigurationsdatei geladen werden.

Als Resultat der Simulation wird eine Log-Datei erzeugt, die daraufhin ausgewertet werden kann.

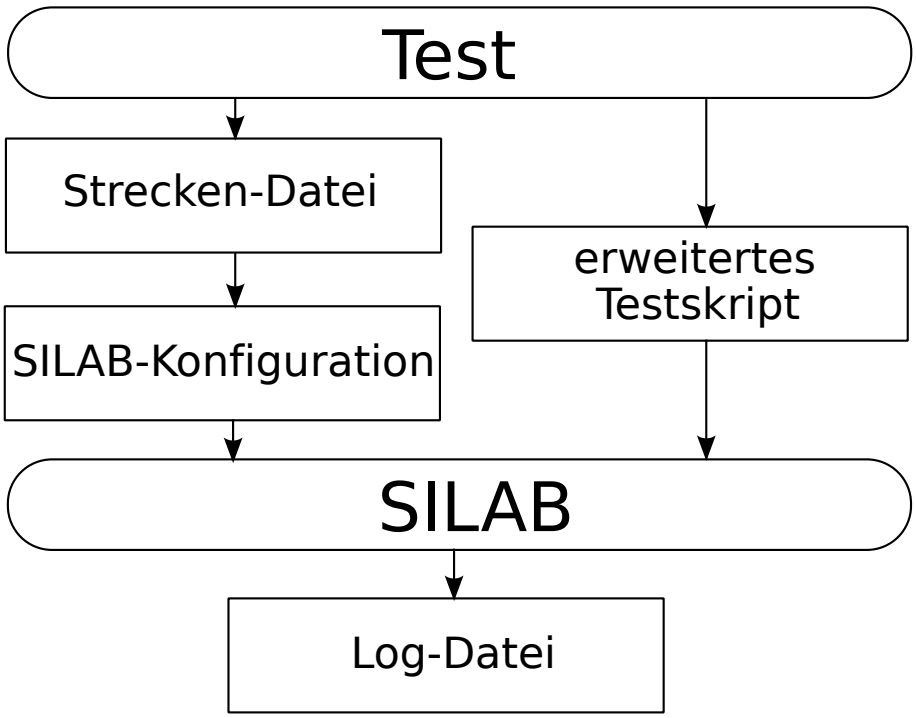


Abbildung 5.5: Vorgehen bei automatischen Tests

6 Machbarkeitsstudie

6.1 Motivation

In der Machbarkeitsstudie wird geprüft, ob die gegebene Hardwareplattform angemessen dimensioniert ist, um das entwickelte Fahrerassistenzsystem ViDAs nach einer möglichen Portierung auf dieser ausführen zu können. Aufgrund von Personal- und Zeitmangel wurde nach dem ersten Review die Aufgabenstellung angepasst, sodass statt der Portierung und Ausführung des Systems auf der Hardwareplattform lediglich die Machbarkeit überprüft wurde.

Das Fahrerassistenzsystem interagiert mit der physikalischen Umgebung und greift direkt in das Längsverhalten und damit in die Dynamik des Fahrzeugs ein (ACC-Funktionalität). Somit agiert das System in einer Umgebung, in der enge zeitliche Fristen gelten, die unter allen Umständen eingehalten werden müssen.

Das Nichteinhalten einer Frist kann schweren Schaden für Fahrer, Insassen und Fahrzeug bedeuten, falls z.B. ein Bremsbefehl zu spät berechnet und übermittelt wurde. Daher muss gezeigt werden, dass die verwendete Hardwareplattform so dimensioniert ist, dass unter allen Umständen alle Fristen eingehalten werden und so jederzeit das korrekte zeitliche Zusammenspiel garantiert ist.

Dies kann nur gezeigt werden, in dem die Ausführungszeit im schlechtesten Falle (worst-case execution-time, *kurz:* WCET) ermittelt und zusammen mit der Nachrichtenlaufzeit betrachtet wird. Aufgrund der hohen Komplexität der verwendeten Hardwareplattform (7-stufige Pipeline, Caches, Floating-Point-Unit) und der großen Anzahl an möglichen Eingangsbelegungen ist es sehr aufwendig und schwierig, die WCET, selbst für ein System von noch eher geringerer Komplexität als dem vorliegenden ViDAs, zu ermitteln.

Da es sich bei dem einjährigen Projekt um eine studentische Arbeitsgruppe handelt, wird lediglich eine prototypische Portierung des Fahrerassistenzsystems auf einen, als System-on-Chip konfigurierten, FPGA vorgenommen und das Vorgehen dokumentiert.

Für ein Subsystem des entwickelten ViDAs-Systems wird dann exemplarisch die durchschnittliche Ausführungszeit auf dem FPGA/SoC gemessen sowie durch eine statische Analyse anhand des Maschinencodes, der Beschreibung und des Timingmodells des eingesetzten Prozessors, exemplarisch die WCET des Subsystems errechnet.
Die Ermittlung der Nachrichtenlaufzeiten erfolgt durch Abbildung der erforderlichen Nachrichten des ViDAs-Systems auf einen CAN-Bus und anschließendem statischen Scheduling der Kommunikation anhand des CAN-Protokolls.

Die ermittelten Ausführungszeiten (durchschnittlich und worst-case) sowie die Nachrichtenlaufzeiten werden wiederum ins Matlab-Modell integriert, so dass dem Modell ein realisti-

scheres Zeitverhalten aufgezwungen wird. Dadurch können die Auswirkungen unterschiedlicher Ausführungs- und Signallaufzeiten betrachtet werden und das zeitliche Zusammenspiel unterschiedlicher Komponenten überprüft werden.

Im Anschluss an diese Motivation wird in Abschnitt 6.2 zunächst die gegebene Hardwareplattform beschrieben. Abschnitt 6.3 erläutert zunächst das Konzept bei der Portierung des ViDAs-Systems. Es folgt dann die detailierte Beschreibung, wie aus einem Matlab-Modell Maschinencode erzeugt wird, was bei der Konfiguration eines FPGAs als System-on-Chip beachtet werden muss und schliesst mit der Ausführung des erzeugten Maschinencodes auf der erstellten Hardwareplattform. Abschnitt 6.4 widmet sich der Ausführungszeitanalyse und geht besonders auf einen *messbasierten* Ansatz ein, gefolgt von der Beschreibung der *statischen Analyse* der WCET. Abschnitt 6.5 befasst sich mit der Kommunikation über den CAN-Bus und definiert u.A. die Nachrichten sowie deren Übertragungszeiten. Die Ergebnisse der Betrachtung der Hardwareplattform und der Ausführungszeiten werden in Abschnitt 6.6 vorgestellt.

6.2 Gegebene Hardwareplattform

Die gegebene Hardwareplattform besteht aus einem FPGA-Board (*field-programmable gate array*) der Firma *Xilinx* [Xil], das als System-on-Chip konfiguriert ist. Bei dem FPGA-Board handelt es sich um ein Xilinx Virtex5-ML509-Evaluations-Board, das mit einem LX110T-FPGA bestückt ist. Das System-on-Chip wurde mit der *GRLIB*-Bibliothek der Firma *Aeroflex Gaisler* [Aer] erstellt.

Bei der GRLIB-Bibliothek handelt es sich um eine Bibliothek aus verschiedenen IP-Core-Paketen, aus denen Systems-on-Chip erstellt werden können. Die einzelnen Pakete sind in der Hardwarebeschreibungssprache VHDL geschrieben und somit unabhängig von ASIC- bzw. FPGA-Board-Anbietern einsetzbar. Neben vielen anderen Paketen verfügt die Bibliothek über einen sparc-zertifizierten Leon3-Prozessor, der in einem System mehrfach instanziiert werden kann und so auch Mehr-Prozessor-Systeme möglich sind.

Neben allen nötigen Skripten zur Zusammenstellung und Konfiguration eines Systems sind Beispieldesigns und Konfigurationsdateien für diverse FPGA-Boards enthalten. Da die Bibliothek unter der GNU GPL Lizenz vertrieben wird und die Quellen eingesehen werden können, eignet sich die GRLIB hervorragend für Forschungsprojekte. Für den kommerziellen Einsatz sind ebenfalls entsprechende Lizenzen erhältlich.

Bei dem verwendeten System handelt es sich um das mitgelieferte Beispieldesign für das eingesetzte FPGA-Board, die Architektur ist in Abbildung 6.1 zu sehen. Der Leon3-Prozessor ist mit einem Instruktions- und Datencache von 2x8kb bzw. 1x8kb ausgestattet, wurde jedoch um eine Floating-Point-Unit mit Pipelining erweitert.
Als Systembus wird der AMBA AHB/APB eingesetzt, der über eine AHB/APB-Bridge die Timer und die UART mit dem Prozessor und dem Speicher inkl. Memory Controller verbindet. Die verwendete Debug-Schnittstelle unterstützt neben serieller Kommunikation auch JTAG und Ethernet.
Gepunktet dargestellt ist die Schnittstelle mit dem CAN-Bus, über den die Sensorik, Aktorik

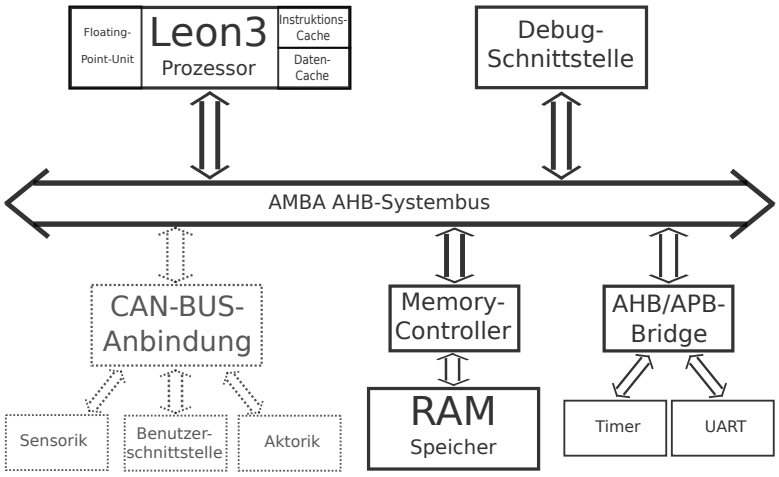


Abbildung 6.1: Architektur des verwendeten System-on-Chip

sowie die Benutzerschnittstelle angesprochen werden. Aus zeitlichen Gründen konnte die CAN-Bus-Schnittstelle jedoch nicht fertiggestellt werden.
Auf die Erstellung des verwendeten Systems wird in Abschnitt 6.3.3 im Detail eingegangen.

Technische Daten

- Leon3-Prozessor
  - 80 MHz Taktfrequenz
  - Integer-Unit mit 7-stufiger Pipepline
  - Floating-Point-Unit mit Pipeline
  - Instruktions- und Daten-Cache (2x8kb,LRU/1x8kb)
  - Memory Management Unit
- 256MB DDR2-Speicher
- AMBA AHB/APB on-Chip Systembus
- AHB/APB-Bridge
- Debug-Schnittstelle (Serial, JTAG, Ethernet)
- 2x Timer mit 8bit Prescaler, 32bit Timerwert

6.3 Arbeitsfluss

Dieses Kapitel vermittelt zunächst in Abschnitt 6.3.1 einen Überblick über die Arbeiten der Machbarkeitsstudie. In Abschnitt 6.3.2 wird dann im Detail auf die Generierung von Quellcode aus einem Matlab-Modell und der Anbindung des Quellcodes an die Umgebung eingegangen.

Abschnitt 6.3.3 widmet sich der Erstellung und Konfiguration eines FPGAs als System-on-Chip, das als Hardwareplattform genutzt wird. Die Arbeiten dieser beiden Abschnitte sind in Abbildung 6.2 dargestellt.
Für die, auf die hier beschriebenen Arbeiten aufbauende, Bestimmung der Ausführungszeiten des generierten Maschinencodes sei an dieser Stelle auf Kapitel 6.4 verwiesen. Dort wird im Detail erläutert, wie die durchschnittliche Ausführungszeit eines Moduls messbasiert auf der realen Hardwareplattform (Abschnitt 6.4.1) bzw. durch statische Analyse des Maschinencodes die WCET (Abschnitt 6.4.2) ermittelt wird und wie diese Ausführungszeiten im Matlab-Modell annotiert werden.

6.3.1 Übersicht

Um das Fahrerassistenzsystem ViDAs auf der in Kapitel 6.2 beschriebenen Hardwareplattform ausführen zu können, muss aus dem Matlab-Modell auf der Modell-Ebene zunächst Quellcode generiert werden. Da das Urpsrungsmodell nach der Code-Generierung nicht mehr in der Matlab-Simulationsumgebung benutzt wird, müssen die Ein- und Ausgänge des Modells zuvor durch *S-Function-Blöcke* gekapselt werden (siehe Abbildung 6.2 *Modell-Ebene*), da diese sonst nicht vom Quellcode aus zugreifbar sind. Bei den Ein- und Ausgängen handelt es sich um Skalare, die ihre Werte auf Quell-/Maschinencode-Ebene über ein gemeinsames Array mit der Umgebung ausgetauschen.

Die Code-Generierung erfolgt unabhängig von der Hardwareplattform, so dass der aus dem Modell erzeugte Code an die Umgebung, in der das Modell ausgeführt werden soll, angebunden werden muss. Für den Datenaustausch zwischen Modell und Umgebung wird ein gemeinsames Array verwendet, aus dem das Modell neue Werte liest und in das es die Ergebnisse schreibt. Hierfür müssen entsprechende get/set-Methoden implementiert werden, die dann mit den Ein-/Ausgängen des Modells verknüpft werden. Von der Umgebung werden neue Werte, z.B. Sensordaten, über den CAN-Bus gesendet bzw. Ergebnisse vom Modell über den CAN-Bus empfangen. Das bedeutet, dass zuden oben genannten get/set-Methoden eine CAN-Bus-Schnittstelle implementiert werden muss, die die Werte zwischen dem Array und der Umgebung über den CAN-Bus kommuniziert.

Für die Kompilierung des generierten Quellcodes und der Anbindung an die Umgebung in Maschinencode wird der Cross-Compiler *sparc-elf-gcc* des *BCC (Bare-C Cross-Compiler System for LEON)* verwendet.

Als Hardwareplattform wird ein FPGA verwendet werden, auf dem ein System-on-Chip synthetisiert wird, das aus der GRLIB-Bibliothek zusammengestellt ist (siehe Kapitel 6.2). Anschließend wird die Werkzeug-Suite *Xilinx ISE* des FPGA-Herstellers benötigt, um das System-on-Chip aus der vorliegenden Hardwarebeschreibung zu synthetisieren und auf dem FPGA zu implementieren (siehe Abbildung 6.2, *Hardware-Beschreibungs-Ebene*). Nach der Konfiguration des FPGAs als System-on-Chip kann der zuvor erstellte Maschinencode darauf ausgeführt werden.

Die Ausführungszeitanalyse des Fahrerassistenzsystems ViDAs wird messbasiert auf der Hardwareplattform auf der System-Ebene und durch eine statische Analyse des Maschinencodes auf

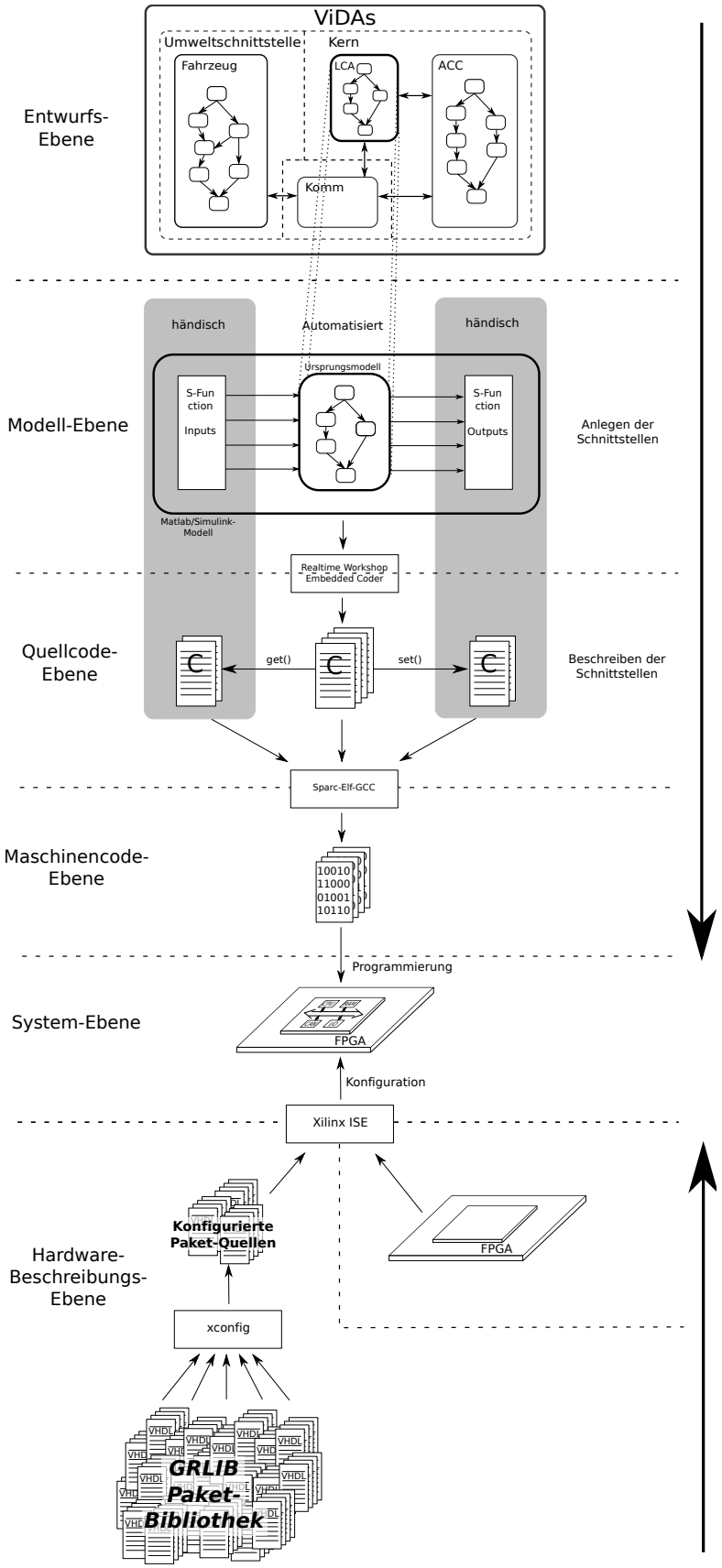


Abbildung 6.2: Schematische Darstellung der Arbeitsabläufe

der Quellcode- bzw. Maschinencode-Ebene durchgeführt. Das messbasierte Verfahren fügt vor und nach der Ausführung des Modells zusätzliche Befehle in den Quellcode ein. Mit Hilfe von Timern der Hardwareplattform wird so die durchschnittliche Ausführungszeit des Fahrerassistenzsystems auf System-Ebene ermittelt.

Bei der statischen Ausführungszeitanalyse wird der Kontrollflussgraph des Assistenzsystems untersucht und mit Hilfe eines Zeitmodells und der Konfiguration des verwendeten Prozessors die WCET ermittelt. Mit der WCET soll zusätzlich zur durchschnittlichen Ausführungszeit eine sichere obere Grenze angegeben werden.

Zur Bestimmung der Nachrichtenlaufzeiten werden zunächst die benötigten Nachrichten definiert (siehe Kapitel 6.5) und gemäß des CAN-Bus-Protokolls ein statisches Scheduling durchgeführt. Anhand des Schedules können dann die Nachrichtenlaufzeiten ermittelt werden.

Die ermittelten Ausführungszeiten zusammen mit den Nachrichtenlaufzeiten werden dann wiederum auf der Modell-Ebene in das Matlab-Modell integriert, um diesem ein realistischeres Zeitverhalten aufzuzwingen. Dazu werden Berechnungen künstlich durch sog. *Transport-Delay*-Blöcke verzögert. Details werden in Abschnitt 6.6.2 beschrieben.

6.3.2 Vom Matlab-Modell zum Maschinencode

Für die Generierung von C/C++-Code aus einem Matlab-Modell wird der *Real-Time Workshop* (kurz RTW) verwendet, dessen Konfiguration im Folgenden erläutert wird. Zunächst soll jedoch beschrieben werden, wie das Modell für die Kommunikation mit der Umgebung auf der Hardwareplattform, auf der das Modell ausgeführt werden soll, erweitert werden muss.

6.3.2.1 Kommunikation mit der Umgebung

Um die Ein- und Ausgänge des Modells im erzeugten Quellcode zugreifbar zu machen, werden zwei *S-Function*-Blöcke verwendet (siehe Abbildung 6.2 *Modell-Ebene*). S-Function-Blöcke sind spezielle Funktionsblöcke der Matlab-Bibliothek, in denen beliebiges Verhalten implementiert werden kann. Die S-Function-Blöcke weisen den Ein-/Ausgängen des Modells, bei denen es sich um Skalare vom Datentyp `boolean`, `uint8`, `int16` oder `double` handelt, einen eindeutigen Namen und eine entsprechende get/set-Methode zu.

Über diese get/set-Methoden kann das Modell auf Quellcode-Ebene auf ein Array zugreifen, das die Kommunikationsschnittstelle des Modells mit der Umgebung darstellt. Aus diesem Array erhält das Modell über die get-Methoden neue Eingabewerte und schreibt seine Ausgaben mithilfe der set-Methoden auch dorthin zurück.
Auf der anderen Seite des Arrays ist eine Schnittstelle, z.B. zu einem CAN-Bus, implementiert. Die Schnittstelle liest neue Sensorwerte vom CAN-Bus und stellt sie dem Modell über das Array zur Verfügung. Neue Ausgaben des Modells überträgt die Schnittstelle aus dem Array auf den CAN-Bus und leitet diese somit an die Umgebung, z.B. die Aktorik, weiter.

Sowohl die Implementierung der get/set-Methoden für das Modell als auch die Zuweisung der eindeutigen Namen und der entsprechenden Methoden zu den Ein-/Ausgängen gesche-

hen händisch (siehe Listing 6.2 und 6.1). Gleiches gilt für das Array zum Datenaustausch und der Schnittstelle zwischen Array und Umgebung.

S-Function-Blöcke müssen auf zwei Ebenen beschrieben werden. Zum Einen werden in einer .c-Datei die Ein- bzw. Ausgänge des Funktions-Blocks und sein Verhalten während der Simulation mit Matlab definiert. Zum Anderen erfolgt in einer .tlc-Datei (target language compiler) die Zuweisung der Ein-/Ausgänge zu den eindeutigen Namen und den get/set-Methoden. Die Beschreibung der S-Function-Blöcke geschieht auf der Modell-Ebene, damit die Zuweisungen während der Generierung des Quellcodes integriert werden können.
Die Implementierung der get/set-Methoden sowie der Schnittstelle zwischen dem Array und der Umgebung erfolgt auf der Quellcode-Ebene (siehe Abbildung 6.2).

Listing 6.1 zeigt in einer beispielhaften .tlc-Datei die Zuweisung der Modellausgänge (`LibBlockInputSignal(..)`)mit den Namen `geschwindigkeit` und `anzeige_verkehrsschild` sowie die Verknüpfungen mit den entsprechenden set-Methoden `setGeschwindigkeit` bzw. `setAnzeige_Verkehrsschild`, bei denen die Ausgänge als Parameter dienen.

Listing 6.1: Ausschnitt einer .tlc-Datei

```
...
  %assign geschwindigkeit = LibBlockInputSignal(0, "", "", 0)
  %assign anzeige_verkehrsschild = LibBlockInputSignal(1, "", "", 0)

  setGeschwindigkeit(%<geschwindigkeit>);
  setAnzeige_Verkehrsschild(%<anzeige_verkehrsschild>);
...
```

In Listing 6.2 ist das Array zum Datenaustausch, die beiden set-Methoden aus Listing 6.1 und eine Funktion zur Erzeugung des Takt-Ereignisses für Matlab zu sehen.

Listing 6.2: Ausschnitt aus einer .c-Datei

```
  #include <stdbool.h>

  /* Array zum Datenaustausch */
  unsigned short ioputs[7];

  /* Erzeugung des Matlab-Takt-Ereignisses */
  bool clock_toggle(bool clock_in)
  {
          return !clock_in;
  }

  void setGeschwindigkeit(unsigned short val)
  {
          ioputs[5] = val;
  }

  void setAnzeige_Verkehrsschild(unsigned short val)
  {
          ioputs[6] = val;
  }
...
```

An dieser Stelle muss erwähnt werden, dass an den Takteingang des Matlab-Modells mittels der S-Function-Blöcke ein *boolesches*-Signal angeschlossen werden muss, da in einem Modell ohne Taktung, auch wenn es über mehrere Berechnungsstufen verfügt, kein interner Zustand gespeichert wird. Folglich würde bei einem mehrmaligen Aufruf des Modells stets nur der erste Berechnungsschritt ausgeführt werden.

Es muss an dieser Stelle zwischen der Clock der Hardware, also dem Takt des FPGA-Boards, der Rate mit der das Modell aufgerufen wird (step()-Funktion, siehe Abschnitt 6.3.2.3), und dem Takteingang des Matlab-Modells unterschieden werden. Bei den Signal am Takteingang des Matlab-Modells handelt es sich viel mehr um ein Ereignis, das vom Modell benötigt wird, um einen neuen Berechnungsschritt beginnen zu können.

6.3.2.2 Konfiguration des Real-Time Workshop

Bei der Konfiguration des Real-Time Workshops muss bei den grundlegenden Einstellungen (siehe Abbildung 6.3) zunächst ein *System target file* ausgewählt werden. Dieses definiert die wesentlichen Merkmale der Hardwareplattform, für die Quellcode erzeugt werden soll. Da es sich in diesem Projekt um eine eingebettete Hardwareplattform handelt, wird Plattform `ert.tlc Real-Time Workshop Embedded Coder` ausgewählt. Möglich sind an dieser Stelle auch z.B. `Autosar`, `grt.tlc Generic Real-Time Target`, `grt.tlc` mit dynamischer Speicher-Allokation und neben Weiteren auch die automatische Erstellung eines `Visual C/C++ Project Makefile`. Als Programmiersprache wird `C` verwendet (hier ist ebenfalls `C++` möglich).
Wie bereits weiter oben erwähnt, muss der erzeugte Quellcode um eigene Methoden zur Anbindung an die Umgebung erweitert werden, sodass die Einstellung `Generate code only` gewählt wird.

Beim Menüpunkt für die Interface-Einstellungen (siehe Abbildung 6.4) muss die Unterstützung für `floating-point numbers` und `continuous time` aktiviert sein, da das Fahrerassistenzsystem ViDAs Regler enthält, die sowohl Fließkomma-Zahlen nutzen als auch auf einem kontinuierlichem Zeitmodell beruhen.
Als `Data exchange Interface` muss `None` gewählt werden, da die Kommunikationsschnittstelle zwischen Modell und Umgebung händisch hergestellt wird.

6.3.2.3 Ergebnis der Code-Generierung

Die Code-Generierung kann nun über die Menüleiste `Tools` → `Real-time Workshop` → `Build Model` gestartet werden. Im Matlab-Hauptfenster sind die detailierten Ausgaben der Code-Generierung zu sehen.

Nach erfolgreicher Durchführung der Code-Generierung
`### Successful completion of Real-Time Workshop build procedure for model: ...`
liegen die Dateien mit dem erzeugten Quellcode in einem Unterordner mit der Endung `<modellname>_ert_rtw`. Die wichtigsten Dateien sind:

- `<modellname>.c`

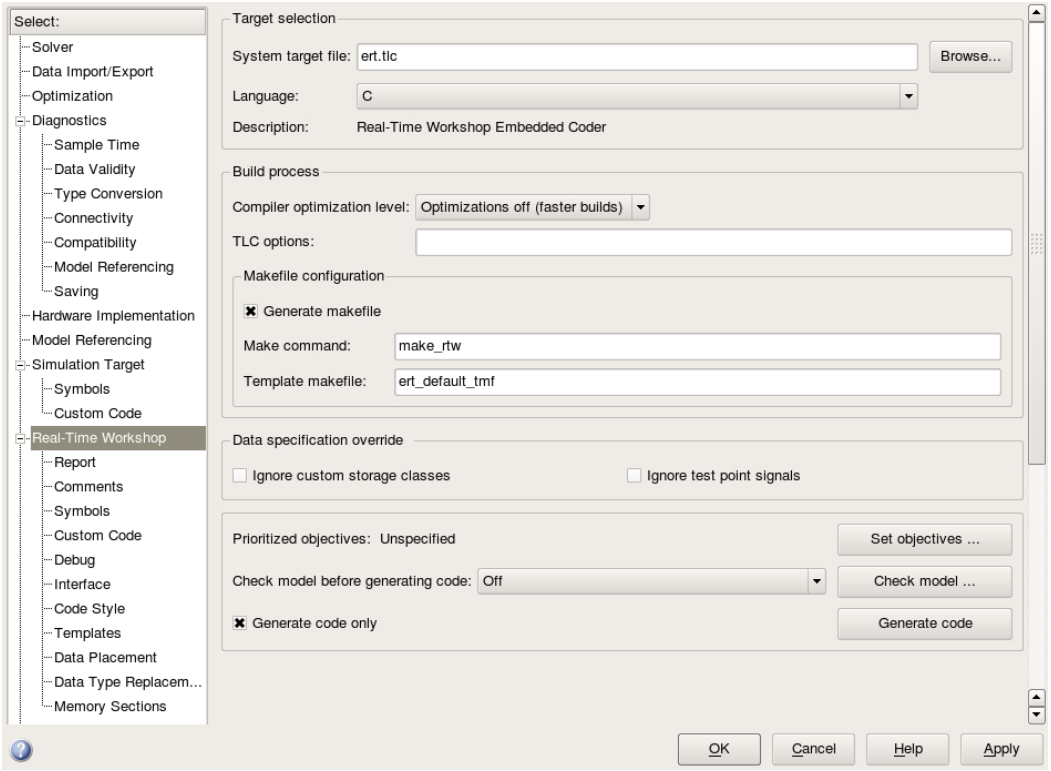


Abbildung 6.3: Grundlegende Einstellungen des *Real-Time Workshop*

- `<modellname>.h`
- `ert_main.c`

Die beiden ersten Dateien enthalten hierbei das Modell, die `ert_main.c` beinhaltet alle nötigen Methodenaufrufe des Modells:

- `<modellname>_initialize()`
- `<modellname>_terminate()`
- `<modellname>_step()`

zum Initialisieren und Beenden des Modells.
Um das Modell einen Matlab-Takt voranschreiten zu lassen, muss die `step()`-Methode aufgerufen werden. Für einen bestimmten Modelltakt muss die step()-Methode mit einem Timer o.Ä. verbunden werden.
Zusammen mit den Methoden zur Kommunikation des Modells mit der Umgebung ist die Code-Generierung abgeschlossen.

Für die Kompilierung des Quellcodes in Maschinencode für die Hardwareplattform wird die *Bare-C Cross-Compiler System for LEON (BCC)* verwendet, da es sich bei dem System-on-Chip um eine Sparc-Architektur handelt. Das Kommando lautet:

```
$ sparc-elf-gcc *.c -o <modellname>.elf -lm
```

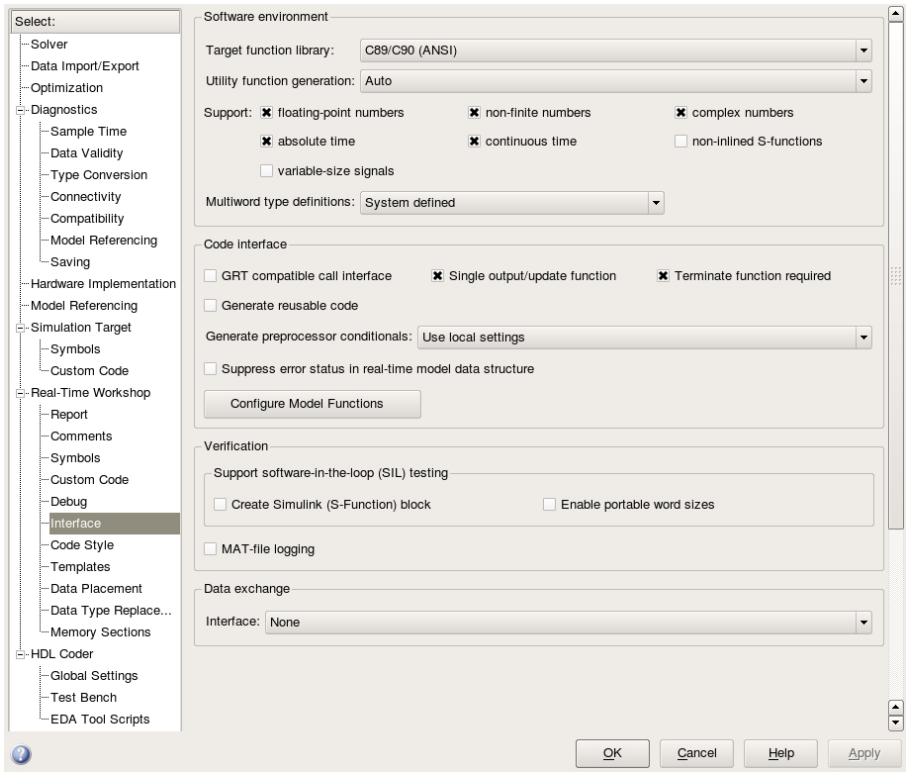


Abbildung 6.4: Interface-Einstellungen des *Real-Time Workshop*

6.3.3 Vom FPGA zum System-on-Chip

In diesem Kapitel wird erläutert, wie das in Kapitel 6.2 beschriebene System-on-Chip aus der *GRLIB*-Bibliothek erstellt wird und dieses auf dem verwendeten Xilinx-Virtex5-ML509-Evluations-FPGA-Board implementiert wird, um eine lauffähige Hardwareplattform zu erhalten (siehe Abbildung 6.2 *Hardware-Beschreibungs-Ebene*).

Als Grundlage für ein neues System-on-Chip wird eine Kopie des Unterordners `leon3-xilinx-ml509` im Ordner `designs` der GRLIB genommen und in der ersten Zeile des `Makefiles` der Pfad zur GRLIB angepasst. Alles andere bleibt unverändert.

Listing 6.3: Anpassung des Pfads zur GRLIB

```
GRLIB=/Pfad/zu/grlib-gpl-1.0.22-b4095
TOP=leon3mp
BOARD=xilinx-ml509-xc5vlx110t
include $(GRLIB)/boards/$(BOARD)/Makefile.inc
DEVICE=$(PART)-$(PACKAGE)$(SPEED)
...
```

`$ make help` gibt dann eine Übersicht über die möglichen make-Ziele, Listing 6.4 zeigt die für Xilinx-FPGAs relevanten make-Ziele.

Listing 6.4: Auswahl der möglichen make-Ziele für Xilinx-FPGAs

```
interactive targets:

make ise-launch           : start ISE project navigator for XST project
make ise-launch-synp      : start ISE project navigator for synplify project
make xgrlib               : start grlib GUI

batch targets:

make ise              : synthesize and place&route with Xilinx ISE
make ise-map          : synthesize design using Xilinx XST
make ise-prec         : synthesize with precision, place&route with Xilinx ISE
make ise-synp         : synthesize with synplify, place&route with Xilinx ISE
make synplify         : synthesize design using synplify
make scripts          : generate compile scripts only
make clean            : remove all temporary files except scripts
make distclean        : remove all temporary files
```

`$ make xconfig` startet das grafische Menü zur Auswahl und Konfiguration der Systemkomponenten (siehe Abbildung 6.5). Über die einzelnen Tasten können gezielt Teile des Systems konfiguriert werden. Bei der Erstellung eines neuen Designs empfiehlt es sich, mit der Konfiguration der Synthese (`Syntheses`) zu beginnen und die Einstellungen des gesamten Systems zu kontrollieren. Hierbei erweist sich als praktisch, dass dem Benutzer alle Einstellungen gezeigt werden, in dem dieser stets nur auf `Next` klickt.

Abbildung 6.6 zeigt die Menüs für den Prozessor, die Floating-Point-Unit sowie des Cache-Systems. In der vorliegenden Hardwareplattform wird als Floating-Point-Unit die `GRFPU` verwendet, die sich durch Pipelining auszeichnet. Die `GRFPU-LITE` hingegen verzichtet auf die Pipeline. Wird keine Floating-Point-Unit eingesetzt, muss beim Kompilieren des Quellcodes der Schalter `-msoft-float` verwendet werden (siehe 6.3.2 Abschnitt *Ergebnis der Code-Generierung*), so dass die Operationen, die eine Floating-Point-Unit benötigen, auf die zur Verfügung stehenden Operationen abgebildet werden.

Das verwendete Cache-System besteht sowohl aus einem 2x8 kbyte großem Instruktions-Cache mit der Ersetzungsstrategie `LRU` als auch aus einem 8 kbyte großem Daten-Cache.

Wie bereits oben erwähnt, sind in der GRLIB-Bibliothek alle notwendigen Skripte enthalten. Nach der Konfiguration muss das System von der Werkzeug-Suite *Xilinx ISE* synthetisiert und

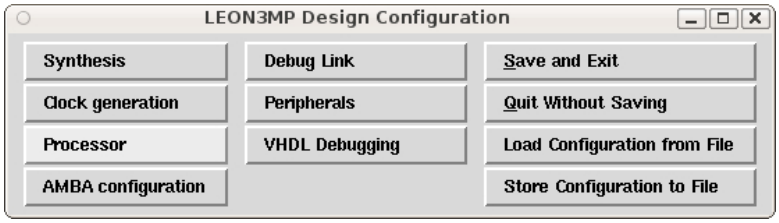


Abbildung 6.5: Hauptfenster des grafischen Konfigurationsmenüs `xconfig`

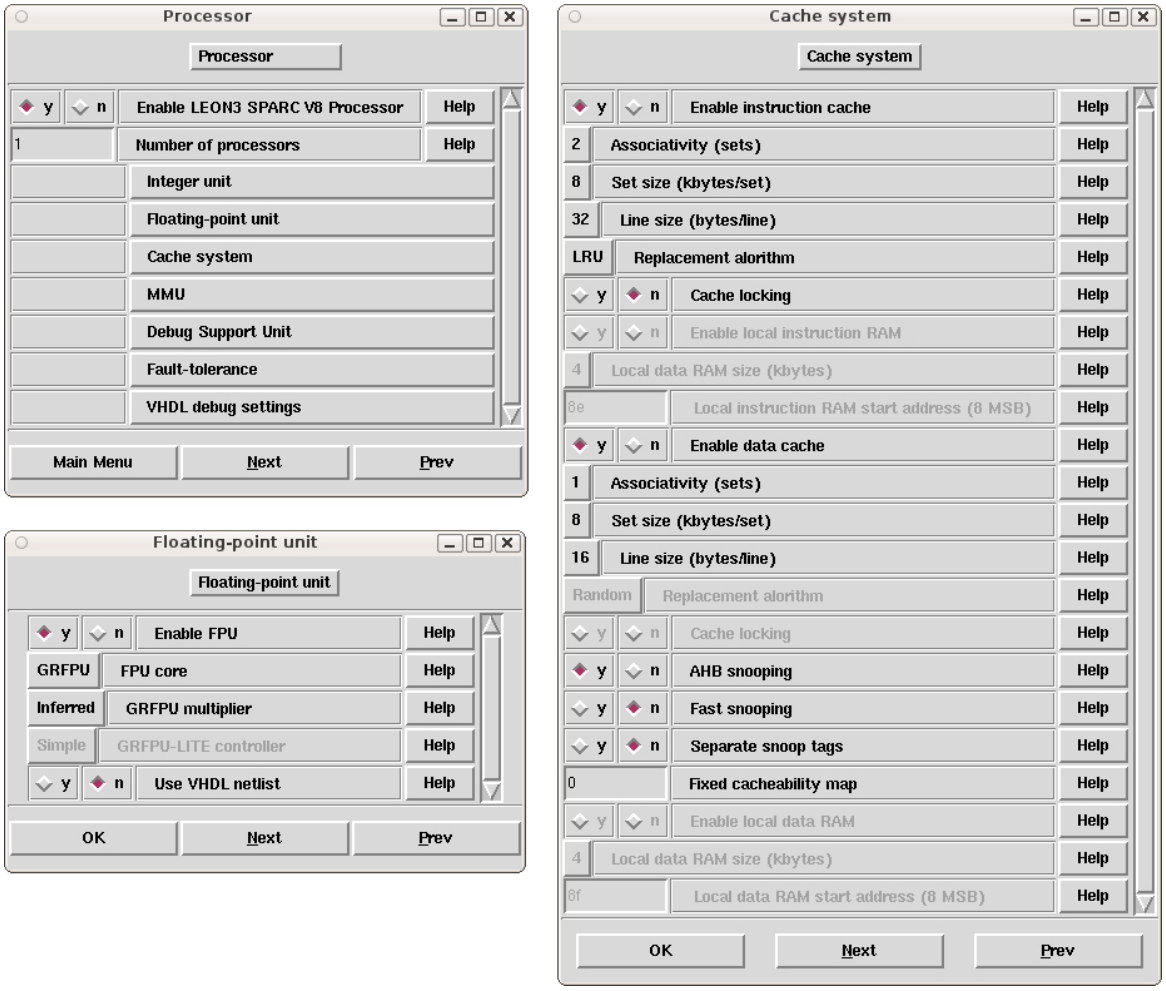


Abbildung 6.6: Konfigurationsmenü des Prozessors, der Floating-Point-Unit und des Cache-Systems

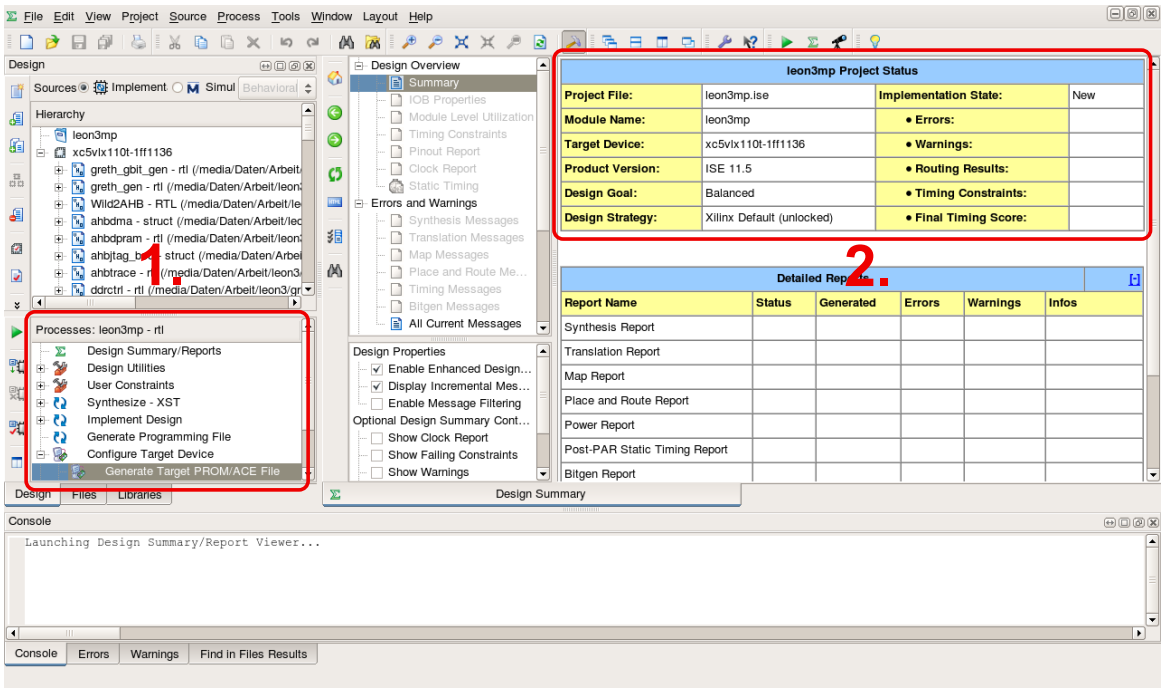


Abbildung 6.7: Xilinx ISE

implementiert werden, da ein Xilinx-FPGA-Board verwendet wird. Die Prozesse können mit grafischer Oberfläche durchgeführt werden oder in einer Konsolenvariante gestartet werden:

`$ make ise-map` startet die Synthese mit Xilinx XST in der Konsole.
`$ make ise` startet die Synthese und Implementierung in der Konsole.
`$ make ise-launch` startet mit grafischer Programmoberfläche.

Für weitere make-Ziele sei auf die Dokumentation des Herstellers verwiesen. *GRLIB IP Library User's Manual* (`GRLIB-Ordner/doc/grlib.pdf`) vermittelt einen schnellen Einstieg in die Nutzung der Skripte.
Im Folgenden wird der Gebrauch der grafischen Oberfläche von Xilinx-ISE erläutert.

Abbildung 6.7 zeigt das Xilinx-ISE-Hauptfenster. Da keine Einstellungen nötig sind, kann der Benutzer alle nötigen Prozessschritte bis zur Erzeugung des Bitstroms zur Konfiguration des FPGAs per Doppelklick auf `Generate Target PROM/ACE File` (1.) starten.
Nun wird das System zunächst synthetisiert, dann implementiert und die *Bitstrom*-Datei (*.bit*) zur Konfiguration des FPGAs erzeugt. Anschließend wird aus dem Bitstrom eine *SysAce*-Datei erzeugt (*.ace*), die auf eine CompactFlash-Karte kopiert wird und von der aus der FPGA dann konfiguriert wird.
Das SysAce-Format bietet den Vorteil, dass mehrere (beim vorliegenden FPGA-Board bis zu acht) Konfigurationen gleichzeitig auf der CompactFlash-Karte gespeichert werden können und so ein Test oder Vergleich von verschiedenen Konfigurationen einfach möglich ist. Um eine andere Konfiguration für den FPGA zu laden, muss hierzu lediglich die Konfigurationsadresse per Dip-Schalter geändert und das FPGA-Board neu gestartet werden. Im Folgenden werden die Schritte zur Erstellung einer SysAce-Datei erläutert.

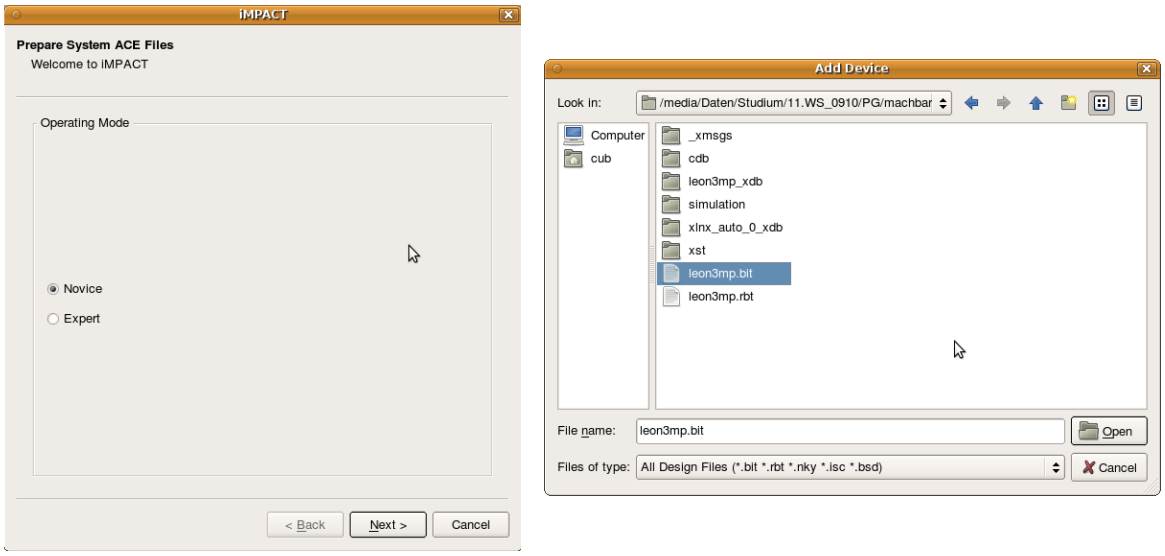


Abbildung 6.8: ISE Impact

Zuvor sei darauf hingewiesen, dass nach Abschluss aller Prozessschritte alle zeitlichen Beschränkungen (*Timing Constraints*) eingehalten worden sein müssen (siehe *Timing Constraints (2.)* unter *Project Status)*. Konnten zeitliche Beschränkungen nicht eingehalten werden, wird dies nur als Warnung, und nicht als Fehler, vermerkt, sodass die Implementierung nicht abgebrochen wurde.
Die Nichteinhaltung kann zur Folge haben, dass zwar der FPGA erfolgreich konfiguriert werden kann, jedoch kein gültiger Takt anliegt und keine Operationen ausgeführt werden.

Abbildung 6.8 *links* zeigt das erste Fenster des Assistenten zur Erstellung einer SysAce-Datei. Bevor zweimal auf `Next` geklickt wird, muss `Novice` ausgewählt sein.
Nun kann ein Name sowie der Speicherort für die Datei angegeben werden. `Next`.
Der Konfiguration muss eine Konfigurations-Adresse (Configuration Address) zugewiesen werden und es kann ein kurzer Name vergeben werden. `Next`. `Finish`. `OK`.
Im folgenden Fenster (siehe Abbildung 6.8 rechts) muss nun der zuvor erzeugte Bitstrom ausgewählt werden (Dateiendung `.bit`). `Open`. `No`.
Im Hauptfenster von *ISE Impact* wird nun über das Menü `Operations` → `Generate File...` und anschließendem Klick auf `OK` die Dateierstellung gestartet.

Sobald die Erstellung erfolgreichg abgeschlossen ist, kann das Programm geschlossen werden. An dem zuvor angegebenen Speicherort befinden sich ein entsprechender Ordner sowie eine Datei namens `xilinx.sys`. Beides muss nun auf eine CompactFlash-Karte kopiert werden. Anschließend kann der FPGA gestartet und von der CompactFlash-Karte konfiguriert werden.

Sollte die gewünschte Konfiguration nicht geladen werden, muss unter Umständen der Dip-Schalter für die Konfigurationsadresse und den Konfigurationsmodus angepasst werden (Standardbelegung: `00010101` startet von Konfigurationsadresse 0). Bei dem verwendeten FPGA-Board befindet sich der Dip-Schalter oben links auf dem Board, siehe Abbildung 6.9. Die Belegung des Dip-Schalters ist in Tabelle 6.1 aufgelistet.

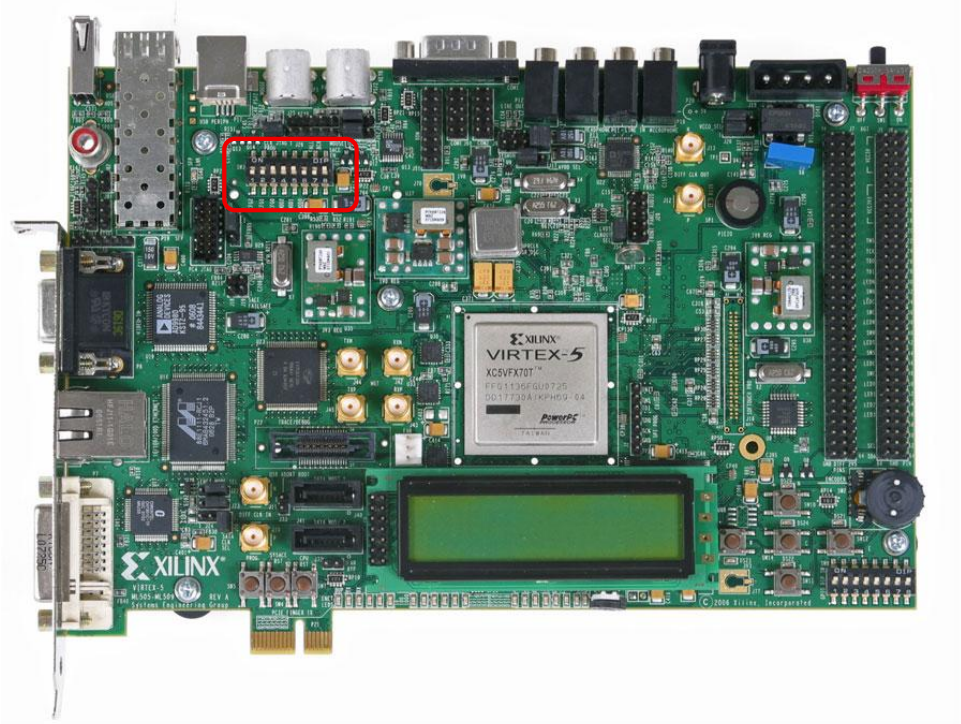


Abbildung 6.9: Xilinx ML509 FPGA-Board: Dip-Schalter der Konfigurationsadresse/-modus

Schalter	**Funktion**
1 - 3	Konfigurationsadresse[2]..[0].
4 - 6	Konfigurationsmodus[2]..[0].
7	Nicht belegt.
8	System ACE Konfiguration (An=Aktiviert, Aus=Deaktiviert).

Tabelle 6.1: Konfigurationsadresse/-modus [Xil09]

Für das Laden und Ausführen eines Programms auf dem nun als System-on-Chip konfigurierten FPGA wird das Debug- und Monitoring-Programm *grmon-eval* benötigt (nicht in der GRLIB-Bibliothek enthalten, kann aber beim Hersteller der Bibliothek *Aeroflex Gaisler* heruntergeladen werden[Aer]). Um die JTAG-Schnittstelle des FPGA-Boards für die Verbindung zu nutzen, muss der Schalter `-xilusb` angegeben werden. Zudem empfiehlt sich der Schalter `-u`, um die Ausgaben in die Konsole vom grmon-eval umzuleiten.
Zum Laden eines Programms wird nun folgender Befehl eingegeben:
`$ grlib> load <dateiname>.elf`
Zum Starten des Programms muss nun noch
`$ grlib> run`
eingegeben werden. Listing 6.5 zeigt einen Beispiellauf.

Listing 6.5: Ausführung eines Programms mit `grmon-eval`

```
$ cat hello.c
#include <stdio.h>

main() {
  printf("Hello World\n");
}

$ sparc-elf-gcc hello.c -o hello.elf
$ grmon-eval -xilusb -u

 GRMON LEON debug monitor v1.1.42 evaluation version

 Copyright (C) 2004-2008 Aeroflex Gaisler - all rights reserved.
 For latest updates, go to http://www.gaisler.com/
 Comments or bug-reports to support@gaisler.com

 This evaluation version will expire on 28/3/2011
 Xilinx cable: Cable type/rev : 0x3
 JTAG chain: xc5vlx110t xccace xc95144xl xcf32p xcf32p

 Device ID: : 0x509
 GRLIB build version: 4095

 initialising ...................
 detected frequency:  80 MHz

 Component                            Vendor
 LEON3 SPARC V8 Processor             Gaisler Research
 AHB Debug UART                       Gaisler Research
 AHB Debug JTAG TAP                   Gaisler Research
 SVGA Controller                      Gaisler Research
 GR Ethernet MAC                      Gaisler Research
 DDR2 Controller                      Gaisler Research
 AHB/APB Bridge                       Gaisler Research
 LEON3 Debug Support Unit             Gaisler Research
 LEON2 Memory Controller              European Space Agency
 System ACE I/F Controller            Gaisler Research
 Generic APB UART                     Gaisler Research
 Multi-processor Interrupt Ctrl       Gaisler Research
 Modular Timer Unit                   Gaisler Research
 PS/2 interface                       Gaisler Research
 PS/2 interface                       Gaisler Research
 General purpose I/O port             Gaisler Research
 AMBA Wrapper for OC I2C-master       Gaisler Research
 AMBA Wrapper for OC I2C-master       Gaisler Research
 AHB status register                  Gaisler Research

 Use command 'info sys' to print a detailed report of attached cores

grlib> load hello.elf
section: .text at 0x40000000, size 21296 bytes
section: .data at 0x40005330, size 2744 bytes
total size: 24040 bytes (963.2 kbit/s)
read 159 symbols
entry point: 0x40000000
grlib> run
Hello World

Program exited normally.
grlib> quit
Closing  Xilinx cable
$
```

6.4 Ausführungszeitanalyse

Das entwickelte Fahrerassistenzsystem ViDAs wird in einem Fahrzeug eingesetzt und greift durch die ACC-Funktionalität direkt in das Verhalten des Fahrzeugs ein, indem es beschleunigt oder abbremst. Durch die LCA-Funktionalität beeinflusst das System zusätzlich den Fahrer in seiner Entscheidung, einen Spurwechsel durchzuführen. Sollte ein Bremseingriff zu spät erfolgen oder der Fahrer nicht rechtzeitig vor einem Fremdfahrzeug gewarnt werden, so kann es zu kritischen Verkehrssituationen kommen.

Deshalb sind dem System enge zeitliche Fristen für die Reaktion auferlegt, die unter allen Umständen eingehalten werden müssen. Sollte eine Frist nicht eingehalten werden, so drohen schwerer Schaden für Fahrer, Insassen und Fahrzeug.

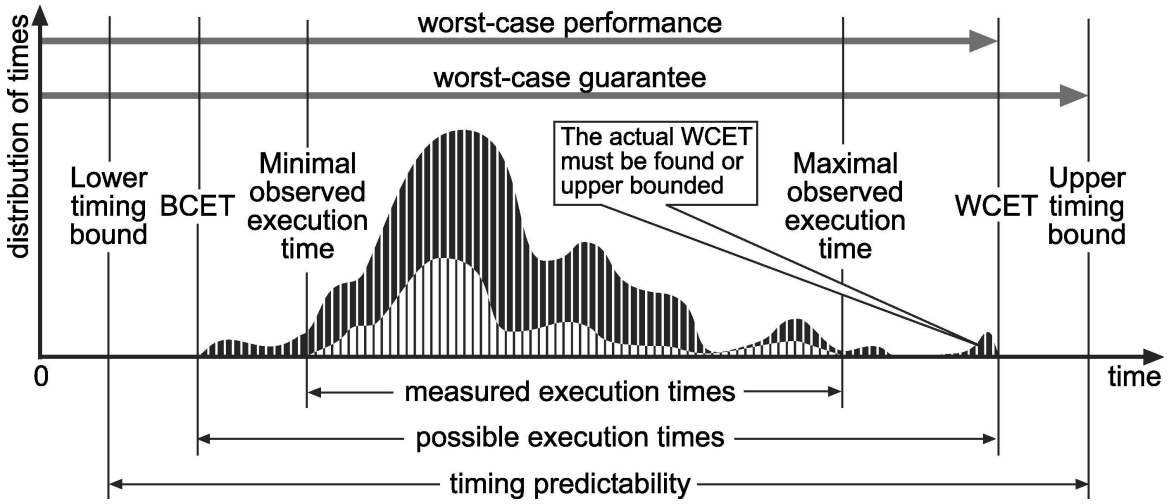


Abbildung 6.10: Bezeichnungen von Ausführungszeiten. Die untere helle Kurve repräsentiert die Untermenge von gemessenen Ausführungszeiten. Das Minimum/Maximum wird als minimale/maximale gemessene Ausführungszeit bezeichnet. Die dunkle, umschließende Kurve stellt alle Ausführungszeiten dar. Das Minimum/Maximum wird als beste/schlechteste Ausführungszeit (best-/worst-case execution times, BCET,WCET) bezeichnet [WEE+08].

Um garantieren zu können, dass alle Fristen eingehalten werden, muss die WCET (worst-case execution-time) ermittelt und betrachtet werden. Die Bestimmung der WCET ist aufgrund von komplexer Hardware mit Pipelining, Caches, FPU, etc. sehr schwierig.

Für das vorliegende System sollen zwei Ansätze zur Bestimmung der Ausführungszeit erläutert werden, deren Resultate in Abbildung 6.10 dargestellt sind.
Die helle untere Kurve repräsentiert die gemessenen Ausführungszeiten, die durch zufällig oder auch systematisch generierte Eingangsdaten erreicht werden.

Bei der Generierung der Eingangsdaten kann jedoch nicht garantiert werden, dass der worst-case enthalten und somit bei der Messung mit abgedeckt ist. Folglich kann auch nicht garantiert werden, dass die maximale gemessene Ausführungszeit der WCET entspricht.

Das gemessene Maximum liegt unterhalb der WCET und ist in Abbildung 6.10 mit `maximal observed execution time` bezeichnet.

Der zweite Ansatz führt eine statische Analyse des Programms durch. Die Analyse ist unabhängig von Eingangsdaten und untersucht die Ausführungszeit anhand des Kontrollflusses für alle Cache- und Pipelinezustände. Mit Hilfe eines Timingmodells des Prozessors wird die obere Schranke für die Ausführungszeit berechnet und gleichzeitig der WCET-Pfad bestimmt. Die statische Analyse betrachtet folglich alle möglichen Ausführungspfade, ist jedoch ebenfalls sehr aufwendig und benötigt detailiertes Wissen über die Implementierung. Zudem handelt es sich um pessimistische Abschätzungen, sodass die garantierte obere Schranke der Ausführungszeit oberhalb der WCET liegt, in Abbildung 6.10 mit `upper timing bound` bezeichnet.

Die WCET des Systems liegt folglich "irgendwo" zwischen der maximalen gemessenen Ausführungszeit und der garantierten oberen Schranke der Ausführungszeit der statischen Analyse.

Im Folgenden soll nun zunächst gezeigt werden, wie die Ausführungszeit eines Systems messbasiert ermittelt werden kann. Im Anschluss folgt dann die Beschreibung der statischen Analyse zur Ermittlung einer garantierten oberen Schranke der Ausführungszeit.

6.4.1 Messbasierte Ausführungszeitbestimmung durch Code-Einbringung

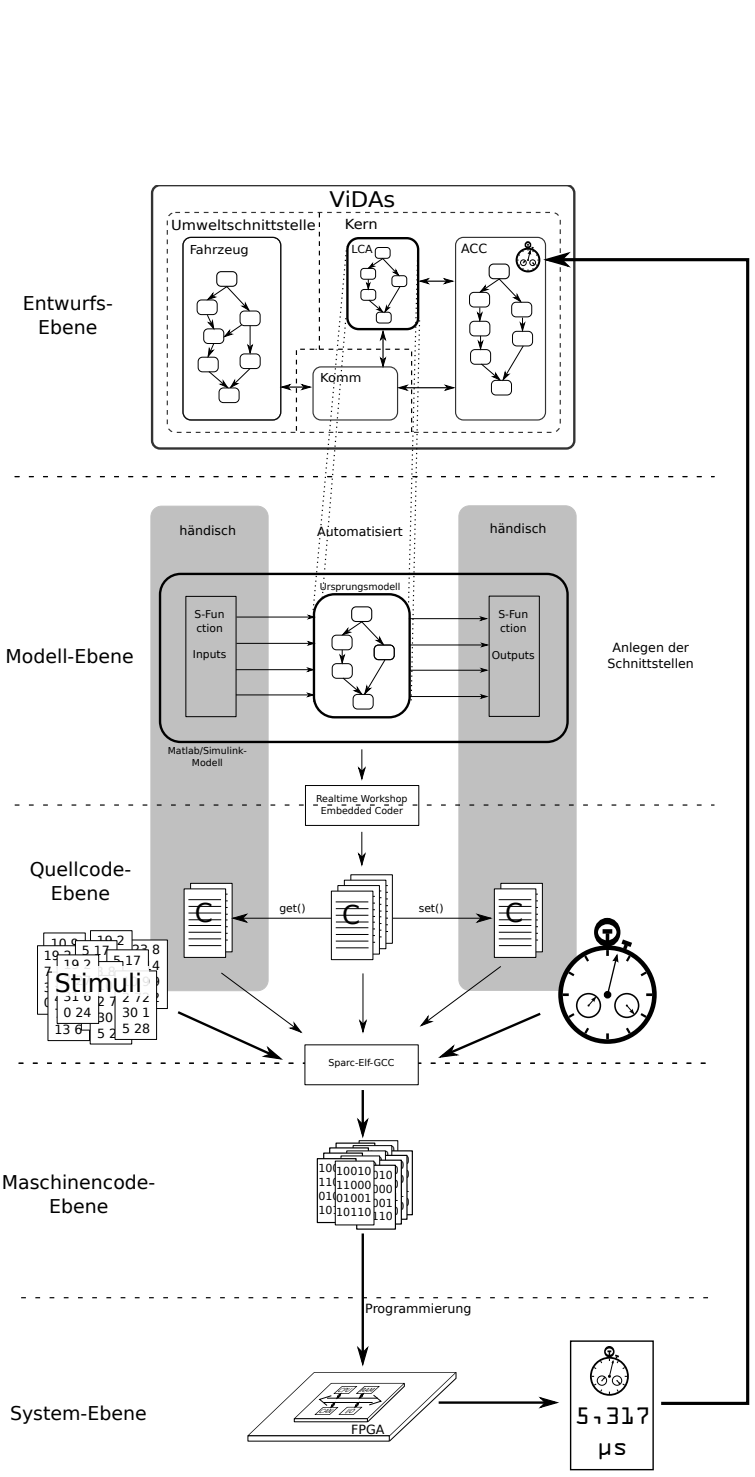
In diesem Abschnitt wird ein messbasierter Ansatz vorgestellt, der durch einbringen von zusätzlichen Code die Ausführungszeit bestimmt. Dazu wird um das zu messende Modell eine Messumgebung erzeugt, die dem Modell Stimuli bereitstellt und das Modell aufruft. Vor und nach dem Aufruf des Modells wird die Zeit gespeichert, anschließend die Ausführungszeit errechnet und dann ausgegeben.

6.4.1.1 Vorgehensweise

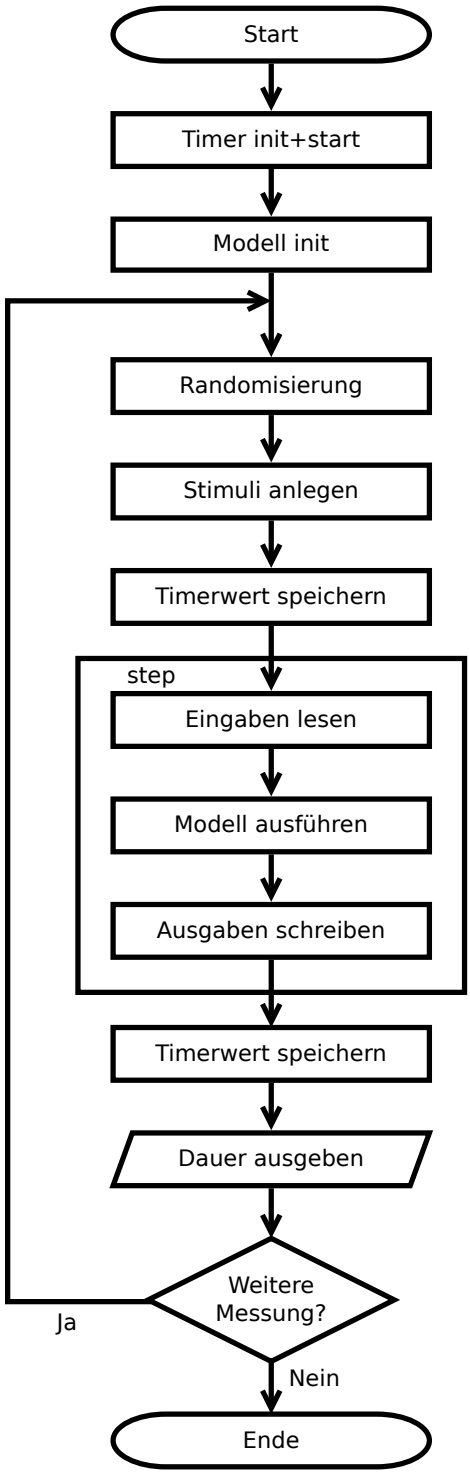
Der messbasierte Ansatz zur Bestimmung der Ausführungszeit setzt auf der Quellcode-Ebene an und ergänzt den bereits vorhandenen Quellcode aus Abschnitt 6.3 um weitere Funktionen, siehe Abbildung 6.11(a). Für die Messung wird ein Timer verwendet, der von der Hardwareplattform bereitgestellt wird. Abbildung 6.11(b) zeigt detailiert den Ablauf der Messung, der nun im Folgenden erläutert wird.

Um den Timer zu Initialisieren und zu Starten, müssen die entsprechenden Kontrollregister benutzt werden. Da für die Messung kein Timer sondern eine Hardware-Uhr benötigt wird, wird der Timer so konfiguriert, dass er dauerhaft läuft und sich automatisch neustartet, sobald die Zeit abgelaufen ist. Anschließend wird das Modell durch Aufruf der Methode `<modellname>_initialize()` initialisiert, die vom Real-Time Workshop erzeugt wurde (siehe Abschnitt 6.3.2).

Vor jeder Messung wird ein Programm ausgeführt, um den Cache- und Pipelinezustand zu randomisieren. Anderenfalls könnten sich die Messungen gegenseitig, z.B. durch vorgeladene Caches oder Pipelines, beeinflussen. Bei dem späteren Betrieb des Systems wird zwischen zwei Aufrufen ebenfalls der Systemzustand durch andere Prozesse verändert.



(a) Schematische Darstellung des Arbeitsablaufs mit integrierter Messung



(b) Ablauf beim Messen

Abbildung 6.11: Grafiken zur Verdeutlichung des Vorgehens beim Messen

Die für die Messung benötigten Stimuli werden als Matrix in die Messumgebung integriert und zeilenweise in ein Array kopiert. Aus diesem Array liest das Modell per get-Methoden neue Eingabewerte und schreibt per set-Methoden die Ergebnisse zurück (siehe Abschnitt 6.3.2).

Die Messung wird gestartet, indem der Wert des Timers ausgelesen und zwischengespeichert wird. Das Modell wird durch Aufrufen der `<modellname>_step()`-Methode für einen Matlab-Takt ausgeführt. Die step-Methode wurde ebenfalls vom Real-Time Workshop erzeugt. Das Modell ist so aufgebaut, dass ein Aufruf der step-Methode ausreicht, um aus den Stimuli am Eingang die Ergebnisse für den Ausgang zu berechnen und bereitzustellen. Bei einem Aufruf werden zuerst die neuen Eingabedaten aus dem Array durch die get-Methoden geladen, dann die Berechnungen des Modells durchgeführt und schließlich die Werte an den Ausgängen in das Array geschrieben.

Nun wird die Messung, in dem erneut die Zeit vom Timer ausgelesen wird und die Ausführungszeit durch Differenzbildung ausgerechnet und ausgegeben, beendeta. Das Ergebnis bleibt auch bei einem Timerüberlauf korrekt, da die Timerwerte als unsigned behandelt werden und somit keine negativen Werte möglich sind.

Der nächste Messzyklus mit anderen Stimuli kann nun mit der Randomisierung des Caches und der Pipeline beginnen.

6.4.1.2 Einfluss durch Messung

Das oben beschriebene Vorgehen des Messens birgt den Nachteil, dass das System verändert und somit beeinflusst wird. So sind zusätzliche Befehle für den Kontrollfluss des Messablaufs notwendig und nehmen Einfluss auf die Messgenauigkeit.
Bei allen eingebrachten Befehlen, die zusätzlich zum Modell ausgeführt werden, wird der Zustand des Systems verändert. Dies hat veränderte Cache-Inhalte und andere Daten in der Pipeline zur Folge, so dass z.B. Cache-Misses auftreten, die ohne den Einfluss der Messung nicht vorkämen.

Ein Teil der zeitlichen Beeinflussung kann bestimmt werden, in dem die Messung nach dem oben beschriebenen Vorgehen durchgeführt wird, jedoch werden zwischen den beiden Zeitstempeln keinerlei Operationen ausgeführt. Dadurch wird ermittelt, wie viel Zeit das speichern der Zeitstempel benötigt. Auf diese Weise kann ein Teil des Einflusses aus den Messergebnissen herausgerechnet werden.

6.4.1.3 Eingabestimuli

Bei der Generierung von Eingabestimuli sollte beachtet werden, dass nur Werte aus den definierten Wertebereichen erzeugt werden. Werden längere Stimuli-Folgen verwendet, so sollte es sich um realistische Stimuli-Folgen handeln, die dem Verhalten der Umwelt entsprechen. Bei zufällig generierten Stimuli kann es zu Folgen kommen, die eine zu hohe Anzahl an Zustandswechseln auslöst und somit die durchschnittliche Ausführungszeit pro Stimulivektor verfälscht.

Beinhaltet das System einen Regler, bei dem die Ausgangsgröße wieder als Eingangsgröße dient und bei dem von der Umwelt eine Reaktion auf die Regelung erwartet wird, so können die

Stimuli nicht zuvor statisch berechnet werden. In diesem Falle wird für die Messung ein Modell der Umwelt benötigt, um für den Regler valide Eingangswerte in Form einer valider Reaktion der Umwelt berechnen zu können.

6.4.2 Statische Ausführungszeitanalyse

Im Gegensatz zum oben vorgestellen messbasierten Ansatz wird hier nun die Bestimmnug der Ausführungszeit durch stzatische Analyse eines Programms vorgestellt. Durch die statische Analyse lassen sich sichere Abschätzungen für alle Cache- und Pipelinezustände eines Programmpunkts berechnen, ohne Einflussnahme durch Messinstrumentierung und ohne Ausführung des Programms. Zudem werden keine Eingabestimuli benötigt.

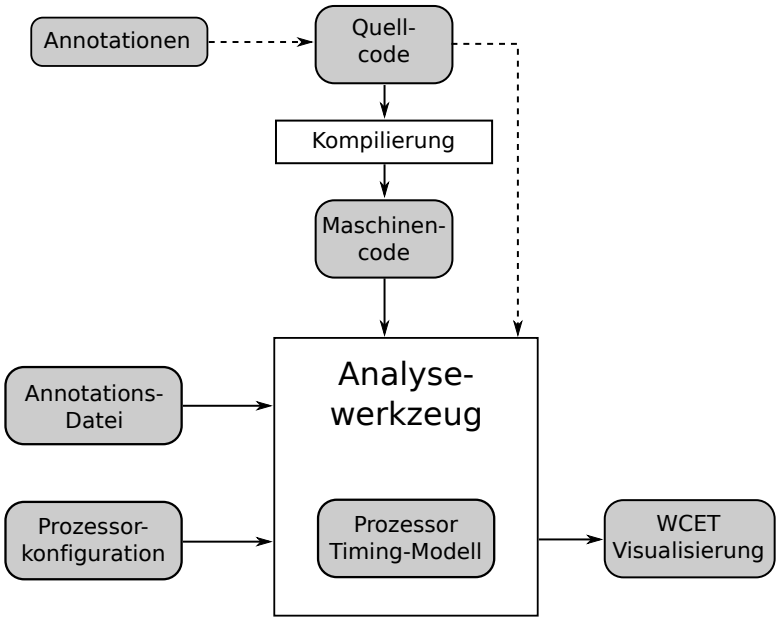


Abbildung 6.12: Teile einer statischen Analyse

Im weiteren wird eine statische Analyse anhand des Werkzeugs `aiT` der Firma `AbsInt` [Abs] erläutert. Abbildung 6.13 veranschaulicht die Phasen des Werkzeugs.

Bei diesem Vorgehen (siehe Abbildung 6.12) können bereits im Quellcode Angaben zur Schleifenobergrenzen oder Rekursionstiefen in Form von speziellen Kommentaren gemacht werden. Der Quellcode wird dann mit Debuginformationen für die Hardwareplattform kompiliert. Der Maschinencode dient dem Analyse-Werkzeug zusammen mit der Prozessorkonfiguration als Eingabe. Damit der Benutzer auch nach der Kompilierung noch Annotationen einfügen kann, ohne den Quellcode neu kompilieren zu müssen, wird zudem noch eine Annotationsdatei eingelesen. Anhand eines Timing-Modells des verwendeten Prozessors berechnet das Werkzeug nun die WCET sowie den WCET-Pfad.

6.4.2.1 Erzeugen des Kontrollflussgraphen

Bei der Erzeugung des Kontrollflussgraphen wird der kompilierte Maschinencode analysiert und anhand der enthaltenen Instruktionen der Kontrollfluss des Programms rekonstruiert. Hierzu muss die Hardwareplattform, für die das Programm kompiliert wurde, bekannt sein, damit auch plattformspezifische Befehle erkannt werden können. In einem zweiten Schritt werden Informationen zu Schleifendurchläufen und oberen Schranken bei Rekursionen annotiert, die der Benutzer in einer *ais*-Datei spezifiziert hat. Zudem werden Sprungadressen von Unterprogrammaufrufen und Verzweigungen ermittelt.

Der Kontrollfluss und die bereits gesammelten Informationen werden in der menschenlesbaren *Control Flow Representation Language* (CRL) auf Assemblerebene gespeichert. Diese Repräsentationsform eignet sich besonders gut für weitere Analysen. Kann das Werkzeug keine Werte ermitteln, so muss der Benutzer die benötigten Daten spezifizieren.

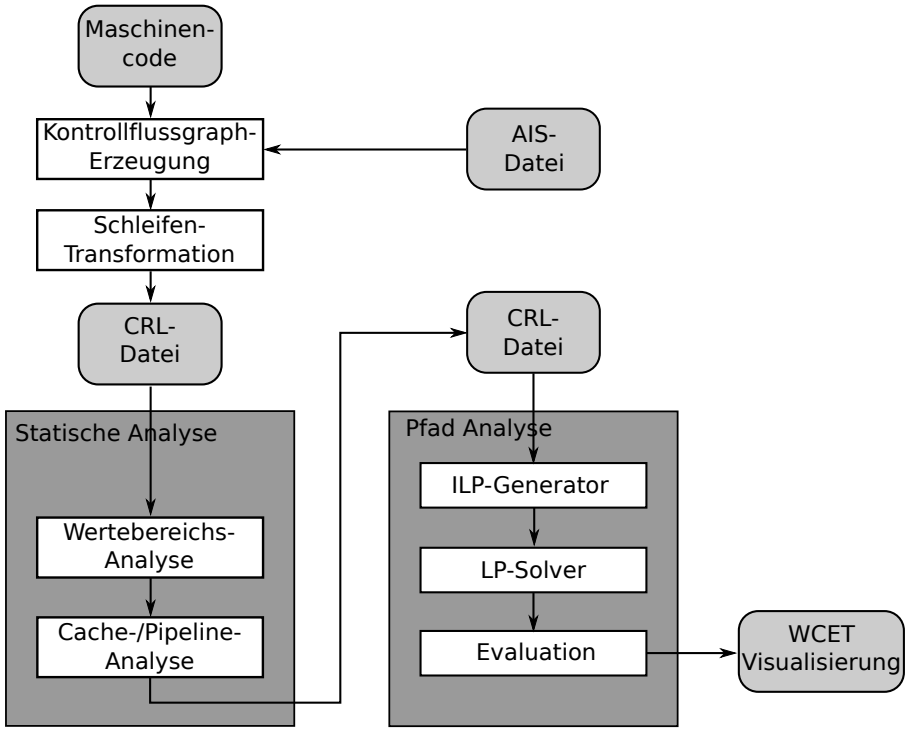


Abbildung 6.13: Phasen der WCET-Berechnung des verwendeten aiT-Werkzeugs von AbsInt [FH04]

6.4.2.2 Wertebereichsanalyse

Die Wertebereichsanalyse berechnet Wertebereiche für Register sowie Adressbereiche für Speicherzugriffe, damit indirekte Speicherzugriffe gefunden werden. Dies erhöht die Genauigkeit der Cache-Analyse und dient der Bestimmung von Schleifengrenzen. Eine exakte Berechnung der Werte ist nur selten möglich, daher werden Bereiche mit unteren und oberen Schranken bestimmt, in denen der korrekte Wert garantiert liegt.

6.4.2.3 Schleifengrenzenanalyse

Da der Großteil eines Programmablaufs in Schleifen stattfindet, ist das Bekanntsein der Schleifengrenzen ist für die Bestimmung der WCET sehr wichtig. Besonders die ersten Schleifendurchläufe haben einen großen Einfluss auf die WCET, da sich der Cache-Inhalt zunächst stabilisieren muss, wofür besonders viele Speicherzurgriffe notwendig sind. In den meisten Fällen muss die maximale Anzahl von Schleifendurchläufen jedoch vom Benutzer angegeben werden, da eine automatische Erkennung nur gelingt, wenn die Zählvariable in einem Register liegt.

6.4.2.4 Cache-Analyse

Bei der Cache-Analyse wird ermittelt, ob ein Cache-Zugriff erfolgreich ist, oder ob ein Zugriff auf den Speicher notwendig wird. Kann ein Cache-Zugriff nicht klassifiziert werden, so kann der Benutzer eine Standardklassifikation vorgeben.

6.4.2.5 Pipeline-Analyse

Die Pipeline-Analyse bildet das Verhalten der Pipeline nach, um Ausführungszeiten für Befehlssequenzen (Basisblöcke) zu bestimmen. Dafür wird der Pipelinezustand betrachtet, besonders wichtig sind hierbei die aktuelle Belegung von Ressourcen sowie die Klassifikation von Speicherzugriffen (siehe Cache-Analyse). Die benötigten Zyklen werden für jeden Basisblock für jeden möglichen Ausführungskontext berechnet und annotiert.

6.4.2.6 Pfad-Analyse

Die Pfad-Analyse berechnet nun für das vorliegende Programm den WCET-Pfad anhand der zuvor errechneten Zyklenzahlen der Pipeline-Analyse unter Einbezug des entsprechenden Ausführungskontextes. Dafür wird ein Integer Linear Programming Constraintsystem aufgestellt und der Pfad mit der maximalen Laufzeit durch Lösen der Constraints ermittelt.

6.5 CAN-Kommunikation

6.5.1 Einleitung

Teil der Aufgabenstellung ist es, ein praxisnahes System zu erstellen. Daher ist eine Kommunikation über einen CAN-Bus zwischen dem SILAB-Fahrsimulator und dem FPGA-Board vorgesehen.

In diesem Teil des Dokumentes wird ein Konzept zur Realisierung der CAN-Kommunikation vorgeschlagen.

Für eine Kommunikation über den CAN-Bus zwischen der ViDAs-Umweltschnittstelle und dem ViDAs-Kern ist zunächst eine Untersuchung der zu übertragenden Daten notwendig. Dabei muss vor allem auf die Anzahl, die Größe und dadurch auch die Übertragungsdauer geachtet werden. Gegebenfalls müssen dann Nachrichten gekürzt beziehungsweise in mehrere Nachrichten aufgeteilt werden um per CAN übertragbar zu sein.

6.5.2 Grundlagen

Beim *Controller Area Network* (CAN) handelt es sich um ein in der Industrie, vor allem im Automobilbereich, weit verbreitetes Bussystem zum Austausch von Nachrichten zwischen verschiedenen Komponenten.

Feld	**Größe**
Start-of-frame	1 Bit
Identifier	11 Bit
Remote transmission request (RTR)	1 Bit
Identifier extension bit (IDE)	1 Bit
Reserved bit (r0)	1 Bit
Data length code (DLC)	4 Bit
Data field	0 bis 64 Bit
CRC	15 Bit
CRC delimiter	1 Bit
ACK slot	1 Bit
ACK delimiter	1 Bit
End-of-frame (EOF)	7 Bit
Intermission	3 Bit

Tabelle 6.2: Aufbau einer CAN-Nachricht

Tabelle 6.2 stellt den Aufbau einer CAN-Nachricht im *base frame format* dar. Die dabei für die Kommunikation zwischen dem Simulator und dem FPGA-Board wichtigen Felder sind der *Identifier*, mit dem festgestellt wird, um welche Nachricht es sich handelt und durch den die Priorität festgelegt wird, und das *Data Field*, das die übertragenen Daten enthält. Die anderen Felder werden automatisch von den jeweiligen Frameworks verwaltet. Alle Felder vor dem *Data Field* stellen dabei den *Header* und alle Felder danach den *Footer* dar.

6.5.3 Vorüberlegungen

Die Nachrichtenpakete im Architektur- und Schnittstellendokument sind auf eine maximale Nachrichtenlänge von 32 Bit angepasst worden, da die CAN-Schnittstelle des SILAB-Fahrsimulators keine längeren CAN-Nachrichten unterstützt. Hierdurch wurde die Kamera-Nachricht in drei Nachrichten unterteilt, an den restlichen Nachrichten wurden keine Veränderungen vorgenommen. Die hieraus resultierenden Größen des Datenfeldes sind in Tabelle 6.3 abzulesen. Da durch

das *Data length code*-Feld im CAN-Header die Datengröße byteweise angegeben wird, sind die Datenfelder auf volle Bytes aufgerundet.

Nachrichten an ViDAs-Kern		**Nachrichten an ViDAs-Umweltschnittstelle**	
Nachricht	**Länge in Bit**	**Nachricht**	**Länge in Bit**
Gaspedal	32	Bremswert	32
Bremspedal	32	Warnsignal	24
Geschwindigkeit	24	Beschleunigungswert	32
Sensor vorne	32	Wechselmoeglichkeit	24
Sensor links	32	Leuchtelemente	24
Sensor rechts	32	Verkehrsschildanzeige	24
Sensor hinten	32	Geschwindigkeits-wahlanzeige	24
Sensor vorne links	32		
Sensor vorne rechts	32		
Kamera Schildererkennung	32		
Kamera Markierungserken-nung	24		
Kamera Kurvenerkennung	32		
Bedienelemente	24		
Gesamt Daten	*392*		*184*
Footer/Header	*611*		*329*
Gesamt	**1003**		**513**

Tabelle 6.3: Nachrichtenlängen

Desweiteren beträgt die Größe der Header- und Footer-Informationen einer CAN-Nachricht 47 Bit und ist zu jedem Datenfeld zu addieren. Das hieraus resultierende zusätzliche Datenaufkommen ist ebenfalls Tabelle 6.3 zu entnehmen. Somit ergibt sich ein Gesamtdatenaufkommen von 1516 Bit pro Sendeperiode, da in einer Sendeperiode jede Nachricht nur einmal versendet wird. Als Sendeperiode wurden 20 ms gewählt, weil die dynamische Simulationsschrittweite des SILAB-Fahrsimulators im Mittel 20 ms beträgt und somit ein häufigeres Senden nicht notwendig ist.

Zur Ermittlung einer geeigneten Übertragungsrate wurden die relevanten Übertragungsraten des CAN-Busses untersucht. In Tabelle 6.4 wurden empfohlene Übertragungsraten zur Nutzung eines CAN-Busses aufgetragen.

Für den Anschluss eines CAN-Busses an den gegebenen FPGA gibt es zwei Möglichkeiten. Zum Einen wird die ECU um einen entsprechende CAN-Komponente aus der GRLIB erweitert und der CAN-Bus über freie Pins angeschlossen. Hierzu ist ein Adapter notwendig, der eine Anpassung der Spannungsniveaus des FPGA-Boards sowie des CAN-Busses vornimmt. Dieser Adapter ist von Detlef Janssen entwickelt worden und liegt zur Benutzung bereit.

Die zweite Möglichkeit ist ein Adapter (CAN232 [Law]) , der an einer seriellen Schnittstelle

(RS232) eine CAN-Bus-Anbindung bereitstellt. Der Adapter wird über die serielle Schnittstelle des FPGA-Boards angeschlossen und über das RS232-Protokoll angesprochen. Bei dieser Möglichkeit ist jedoch eine maximale Übertragungsgeschwindigkeit des CAN-Busses von 125kbit möglich, da die serielle Schnittstelle des FPGA-Boards maximal mit 115.200Baud betrieben werden kann. Aufgrund des Overheads des CAN-Protokolls ergibt sich die Übertragungsgeschwindigkeit des CAN-Busses zu 125kbit.

Sofern eine Übertragungsgeschwindigkeit von 125kbit/s für das vorliegende System ausreicht, wird die zweite Möglichkeit favorisiert, da die ECU nicht aufwendig um einen CAN-IP-Core erweitert werden muss.

Geschwindigkeit	**Senden ViDAs-Kern (ms)**	**Senden ViDAs-Umweltschnittstellen (ms)**
1 Mbit/s	0,513	1,003
500 kbit/s	1,026	2,006
250 kbit/s	2,052	4,012
125 kbit/s	4,104	8,024
100 kbit/s	5,130	10,030
50 kbit/s	10,26	20,060
20 kbit/s	25,65	50,15
10 kbit/s	51,3	100,3

Tabelle 6.4: Übertragungsdauer von allen Nachrichten einer Periode

Der Zeitstempel der Nachrichten wird genutzt, um sicher zu stellen, dass nur Nachrichten der aktuellen Simulationsperiode verarbeitet werden. Dieser wird in jeder Simulationsperiode bei beiden Kommunikationsteilnehmern um eins erhöht und nach Erreichen des Maximalwertes wieder auf null gesetzt. Die erste Simulationsperiode wird durch eine Status-Nachricht, siehe Tabelle 6.5, makiert. Diese wird vom SILAB-Fahrsimulator mit dem Code *START* (siehe Tabelle 6.6) in dessen ersten Simulationszeitschritt gesendet.

Feld	**Größe**	**Wert**	**Beschreibung**
ID	11 Bit	00011001000	ID für Status-Nachricht
Zeitstempel	15 Bit		Zeitstempel der Nachricht
Statuscode	2 Bit	1_{10} bis 3_{10}	Statuscode (siehe Tabelle 6.6)

Tabelle 6.5: Status-Nachricht

Über die Zeitstempel in jeder Nachricht wird sicher gestellt, dass die Systeme synchron arbeiten. Stellt eines der Systeme fest, dass der vom anderen System empfangene Zeitstempel nicht mit dem aktuellen Simulationsschritt übereinstimmt oder dass in einem Simulationsschritt Nachrichten mit verschiedenen Zeitstempeln empfangen wurden, so müssen die Systeme neu synchronisiert werden. Die empfangenen Nachrichten dieser Periode werden verworfen und die Gegenstelle muss hierauf reagieren (siehe 6.5.6.1). Die ViDAs-Umweltschnittstelle stellt hierbei den Master dar und verschickt in diesem Fall eine Status-Nachricht, die in Tabelle 6.5

zu sehen ist. In dieser Nachricht ist der aktuelle Simuationszeitschritt des Masters als Zeitstempel angegeben und gilt dann ab diesem Zeitpunkt als aktueller Simulationszeitschritt für beide Systeme. Stellt der ViDAs-Kern fest, dass der empfangene Zeitstempel nicht mit dem eigenen Wert überein stimmt, so muss dieser eine Status-Nachricht mit dem Code *SYNC_REQ* an die ViDAs-Umweltschnittstelle senden, die diese dazu auffordert, die Status-Nachricht mit dem Code *SYNC* zur Synchronisation zu schicken. Die Codes sind der Tabelle 6.6 zu entnehmen.

Name	**Code**	**Bedeutung**
START	1_{10}	Simulator hat die Simulation gestartet.
SYNC_REQ	2_{10}	Die ViDAs-Umweltschnittstelle wird zur Synchronisation aufgefordert.
SYNC	3_{10}	Der Zeitstempel der Nachricht muss als Simulationszeitschritt übernommen werden.

Tabelle 6.6: Status-Codes der Status-Nachricht

6.5.4 Schedule

Zur Übertragung der Nachrichtenpakete aus dem Architektur- und Schnittstellendokument über den CAN-Bus, ist ein Schedule erforderlich. Hierbei wird eine Periodenlänge von 20 ms festgelegt, da der SILAB-Fahrsimulator eine mittlere Schrittweite von 20 ms besitzt. Der Takt des ViDAs-Kerns ist mit 2 ms festgelegt worden.

Die ViDAs-Umweltschnittstelle sendet die Sensordaten im Zeitslot zwischen 0 ms und 2,5 ms und der ViDAs-Kern antwortet im Zeitslot zwischen 17,5 ms und 20 ms, mit den errechneten Werten. Dieser Ablauf ist in Abbildung 6.14 skizziert dargestellt.

6.5.5 Float-Werte

Für Nachrichten, die das Gas- bzw Bremspedal betreffen, wird eine Nachricht mit 16 Bit Float-Wert verschickt. Da in den Sprachen C und C++ keine solchen Werte im sog. „Half precision floating-point format“ vorgesehen sind, müssen die Werte hier anders codiert werden. Da die Vorkommastelle nie den Wert 10 übersteigt, kann der Wert als 16 Bit Integer übertragen werden, dessen erste beide Zahlen im Dezimalformat die Vorkomma- und die verbleibenen drei die Nachkommastelle darstellen. Es bleibt so eine ausreichende Genauigkeit vorhanden. Aus dem Wert $9,2356$ wird so der Integer 9235, aus dem Wert $10,0000$ wird der Integer 10000.

6.5.6 Vorgehensweise

6.5.6.1 DPU für SILAB

Zunächst wurde überlegt, die CAN-Funktionalitäten des Simulators zu nutzen. Hierfür sind bereits spezielle DPUs vorhanden, die über eine selber zu entwickelnde DPU genutzt werden

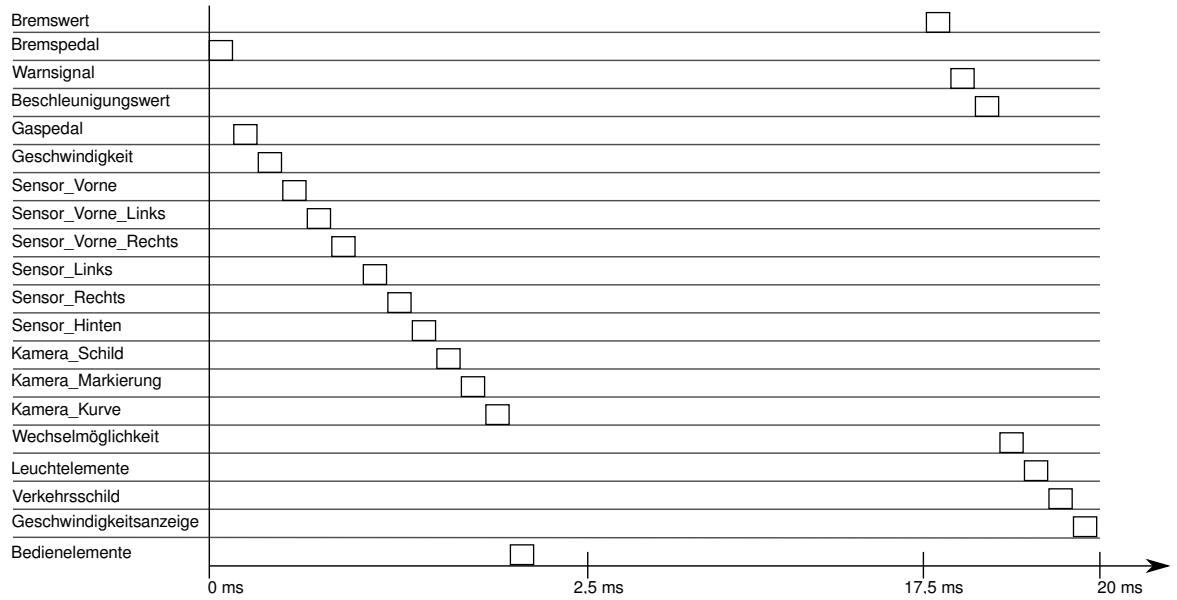


Abbildung 6.14: Can-Schedule

können. Für jeden Nachrichtentyp muss dabei eine eigene Nachrichten-DPU erzeugt werden, die in jedem Simulationsschritt des SILAB-Fahrsimulators mit Daten befüllt wird bzw. aus der Daten gelesen werden. Dabei ist die Länge des Datenbereiches auf 32 Bit beschränkt.

Allerdings ist das Intervall zwischen den Simulationsschritten des SILAB-Fahrsimulators nicht konstant, sondern schwankt zwischen acht und 25 Millisekunden. Daher kann das zuvor beschriebene Scheduling mit dieser Lösung nicht durchgeführt werden.

Es wurde eine zum Simulator kompatible CAN-Hardware angeschafft (siehe Abschnitt 6.5.7.1), die ebenfalls über eine API verfügt, über die sie aus C-Programmen heraus angesprochen werden kann. Da aufgrund des variablen Intervals kein korrektes Scheduling vorgenommen werden kann, soll die API direkt genutzt werden. Dazu wird eine DPU für den SILAB-Fahrsimulator entwickelt, die für die Kommunikation über die CAN-Hardware zuständig ist. In dieser DPU wird das Senden und Empfangen in einem Thread realisiert, so dass das Scheduling eingehalten werden kann und nicht von den Simulationsschritten des SILAB-Fahrsimulators abhängig ist.

6.5.7 Benutzte Komponenten

6.5.7.1 CAN-Realisierung für SILAB

Der Fahrsimulator SILAB unterstützt die Anbindung an einen CAN-Bus. Dabei wird nur eine kleine Auswahl von CAN-Hardware unterstützt. Eine der unterstützten PCI-Karten ist die Karte *PCAN-PCI* der Firma *Peak*.

Da, wie bereits erwähnt, diese Karte allerdings nicht über die SILAB-DPU angesprochen werden soll, wird der mitgelieferte Treiber mitsammt der dazugehörigen API genutzt. Dies erlaubt, eine eigene DPU mit dieser API zu entwickeln. Dadurch können die Scheduling-Zeiten genauer

eingehalten werden. Wie gut die Zeiten eingehalten werden können, muss sich in den ersten Tests zeigen, da die Software nicht exklusiv auf dem System läuft und deswegen durch andere laufende Software unterbrochen werden könnte.

Aus zeitlichen Gründen konnte eine Realisierung auf Seiten des FPGA bzw. der ECU nicht mehr durchgeführt werden.

6.6 Auswertung der Ausführungszeitanalyse

Dieser Abschnitt erläutert im ersten Teil die Überprüfung der gegebenen Hardwareplattform auf ausreichende Dimensionierung und präsentiert die dazugehörigen Ergebnisse in Form eines Schedules. Der zweite Teil erläutert die Annotation der Ausführungszeiten in Form von Verzögerungsblöcken, um das Modell um ein realistischeres Zeitverhalten zu erweitern.

6.6.1 Überprüfung der Hardwareplattform

Die Überprüfung der Hardwareplattform benötigt die WCET des ViDAs-Kerns, für die eine garantierte obere Schranke per statischer Analyse ermittelt werden kann. Da eine statische Analyse den Zeitrahmen des Projekts verlassen würde, werden im Folgenden auf Basis der durchschnittlichen Ausführungszeit des ACC-Submoduls `Schilderauswertung` die durchschnittlichen Ausführungszeiten der übrigen Submodule errechnet (siehe Abschnitt 6.6.1.2). Aus den durchschnittlichen Ausführungszeiten aller Submodule wird die durchschnittliche Ausführungszeit des ViDAs-Kerns berechnet und anhand des Verhältnisses der WCET zur durchschnittlichen Ausführungszeit der Schilderauswertung eine WCET für den ViDAs-Kern errechnet. Mithilfe der WCET des ViDAs-Kerns und den Übertragungszeiten der CAN-Kommunikation sowie weiteren Einschränkungen wird dann in Abschnitt 6.6.1.3 ein Scheduling der ECU versucht.

6.6.1.1 Ausführungszeit der Schilderauswertung

Die messbasierte Bestimmung der Ausführungszeit des ACC-Submoduls `Schilderauswertung` nach Abschnitt 6.4.1 ergibt bei etwa 3300 verwendeten Stimulivektoren die Verteilung in Abbildung 6.15. Die Stimulivektoren wurden bei der Durchfühung eines Testszenarios zur Schildererkennung vom SILAB-Fahrsimulator protkolliert.

Aufgetragen sind die benötigte Ausführungszeit gegen die absolute Häufigkeit, mit der dieser Wert auftritt. Bereits herausgerechnet ist der bestimmbare Overhead durch die Messungen (siehe Abschnitt 6.4.1.2), der etwa $0,5\mu s$ beträgt.

Die statistische Auswertung der gemessenen Ausführungszeiten ergibt folgende Werte:

- *Minimum* : $141,26\mu s$ *Maximum* : $152,14\mu s$
- *Arithmetisches Mittel* : $144,36\mu s$ **Median:** $144,19\mu s$ *Modus* : $144,38\mu s$

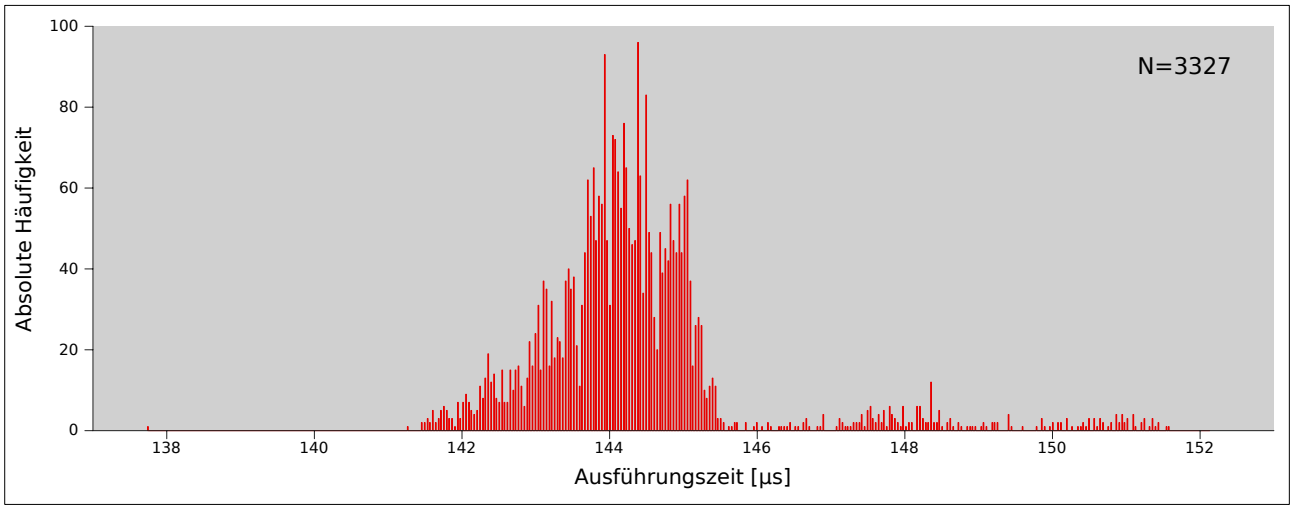


Abbildung 6.15: Häufigkeitsverteilung der gemessenen Ausführungszeiten für das ACC-Submodul `Schilderauswertung` bei 80MHz

Aufgrund seiner höheren Robustheit gegenüber Ausreißern im Vergleich zum arithmetischen Mittel wird für die weitere Betrachtung der durch den *Median* bestimmte Wert als durchschnittliche Ausführungszeit gewählt. Somit ergibt sich die *durchschnittliche Ausführungszeit* für die Schilderauswertung zu $144,19\mu s$.

Die statische Analyse mit dem Werkzeug `aiT` berechnet die WCET auf $177\mu s$ und liegt damit etwa 23% über der gemessenen durchschnittlichen Ausführungszeit. Dies lässt sich damit erklären, dass zum einen die durchschnittliche Ausführungszeit gemessen wurde und zum anderen, dass die Ergebnisse einer statischen Analyse pessimistisch sind.

6.6.1.2 Berechnung der weiteren Ausführungszeiten

Anhand der durchschnittlichen Ausführungszeit für die `Schilderauswertung` werden nun für alle übrigen Submodule des ACC sowie des LCA die durchschnittlichen Ausführungszeiten errechnet.

Hierzu wird eine Metrik entwickelt, die eine Bewertung eines Submoduls ermöglicht. Die Bewertung bezieht sich bei Statecharts auf die Anzahl der Zustände sowie die Anzahl der Transitionen. Zudem wird jede Transition anhand der verwendeten arithmetischen und logischen Operationen der Vorbedingungen sowie die darauf folgenden ausgeführten Befehle der Transition bewertet. Bei den verwendeten Simulinkkomponenten wird ebenfalls die Anzahl der verwendeten Blöcke sowie die dahinterstehenden arithmetischen und logischen Operationen betrachtet und bewertet.

Die Metrik wird zunächst auf das Submodul Schilderauswertung angewendet und eine Bewertung durchgeführt. Aus der Bewertung der Schilderauswertung und der Bewertung eines Submoduls wird nun ein Faktor errechnet, und dieser mit der Ausführungsgzeit der Schilderauswertung multipliziert, um eine verhältnismäßige Ausführungszeit zu erhalten.

<table>
<tr><td colspan="5">ViDAs-Kern: 1181μs</td></tr>
<tr><td rowspan="8">Input: 50μs</td><td>ACC: 837μs</td><td>LCA: 209μs</td><td>Multiplexer: 11μs</td><td rowspan="8">Output: 30μs</td></tr>
<tr><td>Wunschgeschwindigkeits-auswertung: 154μs</td><td>Markierungsauswertung: 44μs</td><td>Beschleunigung: 44μs</td></tr>
<tr><td>Schilderauswertung: 144μs</td><td>Ausfahrtsschilderauswertung: 22μs</td><td rowspan="6"></td></tr>
<tr><td>Kurvenauswertung: 55μs</td><td>Fremdfahrzeugauswertung: 66μs</td></tr>
<tr><td>Abstandsregelung: 77μs</td><td>Verbotsauswertung: 77μs</td></tr>
<tr><td>Geschwindigkeitslogik: 22μs</td><td rowspan="2"></td></tr>
<tr><td>Geschwindigkeitsregelung: 385μs</td></tr>
</table>

Tabelle 6.7: Errechnete durchschnittliche Ausführungszeiten

Da in der Machbarkeitsstudie lediglich exemplarisch gezeigt wird, wie die gegebene Hardwareplattform geprüft werden kann, für die weiteren Schritte aber weitere Ausführungszeiten bestimmt werden müssen, ist die Bestimmung der durchschnittlichen Ausführungszeit nach dem erläuterten Verfahren an dieser Stelle vertretbar.

Tabelle 6.7 zeigt die berechneten Ausführungszeiten. Für den ViDAs-Kern ergibt sich eine errechnete durchschnittliche Ausführungszeit von 1181μs. Anhand der durchschnittlichen Ausführungszeit und des Verhältnisses der WCET zur durchschnittlichen Ausführungszeit der Schilderauswertung wird für die WCET des ViDAs-Kerns 1500μs angenommen.

6.6.1.3 Scheduling der Hardwareplattform

Im Folgenden wird die ausreichende Dimensionierung der Hardwareplattform überprüft und ein Schedule für die maximale Auslastung dieser erstellt. Die Überprüfung und das Scheduling betrifft folgende Tasks:

- Neue Eingangsdaten empfangen
- Aufruf des Fahrerassistenzsystems
- Ergebnisse senden

Zudem müssen folgende Bedingungen eingehalten werden:

- Die Reaktionszeit des Systems t_{reakt} darf maximal $50ms$ betragen (siehe Konstantentabelle D).
- In Abschnitt 6.5 wird die Kommunikation des Fahrerassistenzsystems mit dem Simulationsrechner über CAN-Protokoll definiert. Aufgrund der Nachrichtenlängen benötigt das Senden von Daten an das Fahrerassistenzsystem mindestens $1,003ms$, das Senden der errechneten Ergebnisse an die Simulationsumgebung mindestens $0,513ms$ bei einer

Übertragungsgeschwindigkeit von $1Mbit/s$ (siehe Tabelle 6.3). Bei geringeren Übertragungsraten ($500kbit/s$, $250kbit/s$, $125kbit/s$, etc) verdoppelt sich die Übertragungszeit jeweils.

- Für das Fahrerassistenzsystem werden im vorherigen Abschnitt eine WCET von $1,5ms$ berechnet. Zudem reicht ein Aufruf des Systems, um aus neuen Eingabewerten die korrekten Ergebnisse zu berechnen und zum Versenden bereitzustellen.
- Die Periode des Schedules `sollte` $20ms$ bei Verwendung eines nicht echtzeitfähigen Betriebssystems auf dem Simulationsrechner (hier: Windows XP) nicht unterschreiten, da ansonsten ungleiche Periodendauern auftreten.

Anhand der gegebenen Bedingungen lässt sich feststellen, dass die gegebene Hardwareplattform ab einer Übertragungsgeschwindigkeit des CAN-Bus von $50kbit$ ausreicht:

$$Neue\ Daten\ empfangen + Berechnung + Ergebnisse\ senden \leq 50ms$$

$$20,06ms + 1,5ms + 10,26ms = 31,82ms \leq 50ms$$

Um die gegebene Hardwareplattform maximal zu nutzen, wird im folgenden ein Schedule vorgestellt, der mit der Einschränkung durch den gegebenen Simulationsrechner erstellt wurde (siehe Abbildung 6.16).

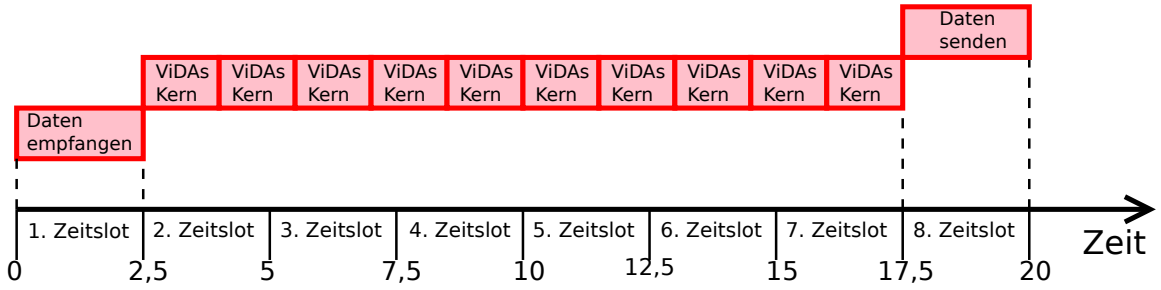


Abbildung 6.16: Schedule für die ECU

Die Periode des Schedules beträgt $20ms$ und ist in acht Zeitslots der Dauer $2,5ms$ unterteilt, d.h., dass alle $2,5ms$ ein Timerinterrupt ausgelöst wird, um die nächste Task zu starten. Im ersten Zeitslot werden die Daten über den CAN-Bus empfangen, beginnend mit dem ersten Timerinterrupt. Als Übertragungsgeschwindigkeit wird $500kbit/s$ (Übertragungsdauer $2,006ms$) gewählt.

Durch den zweiten Timerinterrupt wird die Ausführung des Fahrerassistenzsystems gestartet. Um die maximale Auflösung während einer Scheduleperiode zu erhalten, wird das System nun zehnmal nacheinander ausgeführt. Der prädiktive Regler berechnet bei jedem Aufruf des Systems die im ersten Zeitslot empfangenden Werte um jeweils $2ms$ weiter. So kann eine interpolierte Auflösung von $2ms$ erreicht werden, obwohl nur alle $20ms$ neue Daten vorliegen. Obwohl zwischen zwei Systemaufrufen maximal $1,5ms$ vergehen, rechnet der prädiktive Regler $2ms$

voraus, da er zeitunabhängig konstruiert wurde und die verstrichene Zeit als Parameter eingestellt werden kann. Aufgrund unterschiedlicher Perioden des Systems und des Timerinterrupts werden die nächsten vier Interrupts ignoriert und das System von sich selbst aufgerufen.

Beim achten Timerinterrupt wird dann das Senden der berechneten Ergebnisse durchgeführt, dies dauert $1,026ms$. Folglich werden die Timerinterrupts lediglich zum Initiieren des Empfangens und Sendens der Daten benötigt sowie zum einmaligen Aufruf des ViDAs-Kerns.

Für die CAN-Kommunikation ist auch eine Übertragungsgeschwindigkeit von 1Mbit möglich. Eine so hohe Übertragungsgeschwindigkeit hat jedoch keinen gravierenden Mehrwert, da der ViDAs-Kern lediglich zweimal zusätzlich zwischen Empfangen neuer Daten und Senden der berechneten Ergebnisse aufgerufen würde und bereits mit einer interpolierten Auflösung von 2ms arbeitet. Dieser Wert liegt bereits weit über den geforderten Werten.

Damit ist gezeigt, dass die gegebene Hardwareplattform ausreicht, um das Fahrerassistenzsystem auszuführen und alle Fristen stets eingehalten werden.

Um die gegebene Hardwareplattform optimal zu nutzen, sollte eine Übertragungsgeschwindigkeit des CAN-Busses von $500kbit/s$ gewählt werden und der beschriebene Schedule verwendet werden.

6.6.2 Annotation der gewonnenen Zeiten im Modell

Bei der Simulation eines Modells mit Matlab wird ein zweidimensionales Zeitmodell verwendet. Die eine Dimension ist hierbei die Zeit, die aus Sicht des Modells zwischen zwei Simulationsschritten vergeht und die vom Benutzer in den Optionen eingestellt werden kann. Die zweite Dimension ist die reale Zeit, die während der Berechnungen der Simulation vergeht. Über dieses Zeitmodell wird von der Ausführungszeit der Berechnungen zwischen zwei Simulationsschritten abstrahiert, sodass sich der Entwickler voll auf die Funktionalität des Modells konzentrieren kann. So dauert die Ausführung von einer Zuweisung zwischen zwei Simulationsschritten aus Modellsicht genau so lange wie die Ausführung von 10.000 Zuweisungen.

Um diese Lücke zwischen Simulation des Modells in Matlab und der Ausführung auf einer Hardwareplattform zu verkleinern, sollen die Ausführungszeiten der Submodule gezielt verlägert werden, indem an den Ausgängen der Submodule sog. *Transport-Delay*-Blöcke eingebaut werden. Ein Transport-Delay-Block leitet das Signal am Eingang erst nach n-Simulationsschritten an den Ausgang weiter. Auf diese Weise kann dem Modell ein realistischeres Zeitverhalten aufgezwungen werden.

In Abschnitt 6.6.1.2 wurden bereits für die Erstellung eines Schedules die durchschnittlichen Ausführungszeiten aller Submodule anhand der gemessenen durchschnittlichen Ausführungszeit der Schilderauswertung berechnet. Diese Ausführungszeiten spiegeln das Verhältnis der Ausführungszeit eines Submoduls zur Ausführungszeit der anderen Submodule wieder. Die Übertragung dieser Verhältnismäßigkeiten in das Matlab-Modell wird im Folgenden beschrieben.

6.6.2.1 Berechnung der Verzögerungszeiten

In Tabelle 6.7 sind die berechneten Ausführungszeiten für die einzelnen Submodule dargestellt. Würden die Werte aus der Tabelle direkt in die Verzögerungsblöcke übernommen, würde das Modell zwar sehr exakt das Zeitverhalten nachbilden, dies würde jedoch einen sehr hohen Simulationsaufwand bedeuten, da z.B. die Markierungsauswertung um über 40 Simulationsschritte verzögert würde.

Um den Mehraufwand bei der Simulation so gering wie möglich zu halten, sollen die Ausführungszeiten heruntergerechnet werden, um relative Ausführungszeiten zu erhalten. Hierzu wird der größte gemeinsame Teiler aller Ausführungszeiten berechnet und die Ausführungszeiten durch diesen dividiert. Größter gemeinsamer Teiler ist 11. Die Ausführungszeit der Schilderauswertung ist nicht glatt durch 11 teilbar, sodass das Ergebnis auf 13 abgerundet wird. Die relativen Ausführungszeiten sind in Tabelle 6.8 dargestellt.

Da alle Submodule des Modells innerhalb eines Simulationsschrittes neue Werte berechnen können, die Ausführungszeit somit *einen* Simulationsschritt beträgt, muss vor der Annotation im Modell von jedem Wert noch 1 subtrahiert werden. Damit hat der Multiplexer eine zusätzliche Verzögerung von 0 und eine Ausführungszeit von 1 Simulationsschritt.

ViDAs-Kern		
ACC	**LCA**	Multiplexer: 1
Wunschgeschwindigkeitsauswertung: 14	Markierungsauswertung: 4	Beschleunigung: 4
Schilderauswertung: 13	Ausfahrtsschilderauswertung: 2	
Kurvenauswertung: 5	Fremdfahrzeugauswertung: 6	
Abstandsregelung: 7	Verbotsauswertung: 7	
Geschwindigkeitslogik: 2		
Geschwindigkeitsregelung: 35		

Tabelle 6.8: Verzögerungswerte für die Annotation im Matlab-Modell

6.6.2.2 Annotation im ViDAs-Kern

Im Verlauf der Machbarkeitsstudie wurde festgestellt, dass alle Submodule des ViDAs-Kerns innerhalb eines Aufrufs für neue Daten am Eingang gültige Werte berechnen und an den Ausgang anlegen. Abhängigkeiten zwischen Submodulen im Modell werden vom Real-Time Workshop bei der Code-Generierung durch eine entsprechende Ausführungsreihenfolge aufgelöst, sodass es im ViDAs-Kern keine Unterschiede bei der Ausführungszeit gibt, bezogen auf die Anzahl der Aufrufe des Modells. Der Einsatz von Transport-Delay-Blöcken mit einer Verzögerung ungleich null würde somit nur die Ausführungszeit eines Aufrufs verändern.

Auf diese Weise kann jedoch überprüft werden, wie sich das Fahrerassistenzsystem verhält, falls die Nachrichtenübertragung fehlschlägt. Dazu muss der ViDAs-Kern lediglich um mehr als

eine Periodendauer der Nachrichtenübertragung verzögert werden, so dass der Zeitpunkt zum Senden oder Empfangen der Nachrichten "verpasst" wird.

Ansonsten ist die Annotation von Ausführungszeiten für den ViDAs-Kern nicht sinnvoll.

7 Ergebnisse und Fazit

Dieses Kapitel gibt einen Überblick über die Ergebnisse der Projektgruppe. Außerdem ist ein Fazit über die Projektgruppe, welches auch persönliche Eindrücke der Projektgruppenmitglieder widergibt, enthalten.

7.1 Ergebnisse

7.1.1 Aufgabenstellung

In diesem Abschnitt werden die erarbeiteten Ergebnisse mit der angepassten Aufgabenstellung und den Anforderungen aus der Anforderungsdefinition verglichen und somit kontrolliert, inwieweit die Aufgabenstellung erfüllt worden ist.

Die gestellte Aufgabe (siehe Abschnitt 1.1) wurde erfolgreich gelöst. Es wurde ein Fahrerassistenzsystem entwickelt, welches den Fahrer mit einem ACC und einem LCA bei der Fahrt unterstützt. Dieses System wurde erfolgreich getestet und an den SILAB-Fahrsimulator angebunden, sodass ein simuliertes Benutzen des Systems möglich ist.

Für die Tests wurden eigene Systeme entwickelt, mit denen einzelnen die Module und das Gesamtsystem getestet wurden. Außerdem wurde ein System für automatische Testfahrten entwickelt, mit denen ganze Szenarien getestet wurden. Ebenso liegt eine Sicherheitsanalyse der Anforderungen vor, die anhand des Code of Practice durchgeführt worden ist. Diese enthält eine FMEA, die zur Abschätzung der Risiken und Vorbeugung vor Fehlern behilflich ist.

Desweiteren wurde eine Machbarkeitsstudie durchgeführt und die notwendigen Schritte einer Portierung und Ausführung des Fahrerassistenzsystems auf einen FPGA dokumentiert. In diesem Dokument wurde ebenfalls untersucht, ob eine Kommunikation zwischen dem ViDAs-Fahrsimulator und dem Fahrerassistenzsystem über den CAN-Bus möglich ist. Zudem wurde erläutert, wie die Ausführungszeit des Systems bestimmt und die Hardware auf ausreichende Dimensionierung überprüft werden kann.

7.1.2 Ziele über die Aufgabenstellung hinaus

In der Aufgabenstellung war für die CAN-Kommunikation nur die Durchführung einer Machbarkeitsstudie vorgesehen. Dennoch wurde begonnen, diese zu implementieren. Es wurde eine DPU für SILAB entwickelt, welche die Kommunikation über den CAN-Bus unterstützt. Diese wurde allerdings aufgrund Zeitmangels nur kurz getestet. Ebenso wurde eine CAN-Implementierung

für die FPGA-Hardware begonnen, die allerdings aus den gleichen Gründen nicht getestet wurde.

7.1.3 Erweiterungsmöglichkeiten

Eine mögliche Erweiterung des entwickelten Systems ist die Fertigstellung und das Testen der CAN-Kommunikation. Damit verbunden ist auch die Fertigstellung der Portierung auf das FPGA-Board. Ausführliche Vorarbeiten hierfür sind bereits in der Machbarkeitsstudie (siehe Kapitel 6) geschehen.

Ebenso könnte, wie in der ursprünglichen Aufgabenstellung vorgesehen, eine Bilderkennung implementiert werden. Diese würde die bisher durch ein SILAB-Fahrsimulator-Modul realisierte Schildererkennung ersetzen. Durch den modularen Aufbau des Systems könnte das derzeit genutzte Modul jederzeit durch ein neues Modul ersetzt werden.

7.2 Fazit

7.2.1 Gruppenklima

Besonders zu Beginn der Projektgruppe gab es Probleme, da teilweise Aufgaben nicht erfüllt, Fristen nicht eingehalten und Treffen verpasst wurden. Daher wurden Sanktionen in Form von Geldzahlungen eingeführt. Dieses hat die Disziplin der Gruppe gesteigert und Fristen und Termine wurden häufiger eingehalten. Die Strafen wurden in eine eigens eingerichtete Projektgruppenkasse eingezahlt.

Um das Gruppenklima zu fördern, wurden gemeinsame Treffen außerhalb der Projektgruppenzeit organisiert. So gab es z.B. mehrere Treffen zum Grillen oder Wii-Spielen, sowie ein Treffen zum Bowling.

Dabei wurden diese Treffen zum Teil aus der Projektgruppenkasse bezahlt.

Insgesamt herrschte ein gutes Gruppenklima, auch wenn es gelegentlich zu Meinungsverschiedenheiten kam.

7.2.2 Gruppenorganisation

Die Organisation der Projektgruppe war eine neue Erfahrung und daher eine große Herausforderung. Es wurde zwar eine spezielle Software zur Projektorganisation verwendet, die die Organisation vereinfachte, dennoch stellte sich sehr bald heraus, dass es ein komplexes Unterfangen war, den Projektzeitraum sinnvoll in Phasen zu unterteilen, passende Zeitslots für einzelne Aufgaben zu definieren und die Projektgruppemitglieder passend auf die einzelnen Vorgänge zu verteilen. Dafür mussten die Kompetenzen aller Gruppenmitglieder gefunden werden, um die Arbeitskräfte effizient ausnutzen zu können.

Als dann für die Testphase ein separater Plan erstellt wurde, der mit dem bereits bestehenden Projektplan abgestimmt werden musste, stieg der Organisationsaufwand weiter an, da die begrenzten Ressourcen zum Teil auf zwei Projektpläne verteilt werden mussten, aber dennoch nicht überlastet werden durften.

Aber nicht nur die Erstellung der Projektpläne gestaltete sich schwierig, auch einen wöchentlichen Termin für die Statustreffen zu finden war nicht einfach. Es mussten acht Leute mit unterschiedlichen Vorlesungs-/Übungsterminen, zum Teil relativ langen Anreisewegen und Nebenjobs an einem gemeinsamen Termin Zeit haben.

Weiterhin wurde bereits sehr früh in der Projektgruppe deutlich, dass der gesamte Gruppenfortschritt aufgehalten wurde, sobald ein Gruppenmitglied seine Aufgabe nicht zeitgemäß erledigte. Das ging zum Teil so weit, dass die Projektplanung verändert werden musste, um die zusätzliche Arbeit wieder im Plan unterzubringen.

7.2.3 Kommunikation

Die Kommunikation unter den Gruppenteilnehmern war sehr gut.

Da es wöchentliche Treffen gab, konnten in diesen viele Probleme geklärt werden und auch Anregungen und Verbesserungsvorschläge eingebracht werden. Ein weiteres Gruppentreffen pro Woche hätte für noch mehr Rückkopplung zwischen den Projektgruppenteilnehmern gesorgt. Allerdings ist es, wie bereits in Abschnitt 7.2.2 erwähnt, sehr schwer, bei größeren Gruppen einen gemeinsamen Termin für ein weiteres Treffen zu finden.

Durch die Einführung von Präsenzzeiten war es möglich, Gruppenmitglieder zu festen Zeiten in den Projektgruppenräumen anzutreffen. Jedes Mitglied hatte mindestens eine Zeit, zu der es anwesend sein musste. So war es auf einfache Weise möglich, sich persönlich zu treffen, ohne vorher einen Termin festlegen zu müssen. Ebenso konnte auf diese Weise der Arbeitsfortschritt genauer beobachtet werden.

Die weitere Kommunkation erfolgte hauptsächlich per E-Mail. Dies hatte den Vorteil, dass evtl. aufgetragene Arbeitsaufträge genau definiert und jederzeit wieder durch den Bearbeiter nachgeschaut werden konnten.

Für E-Mails, die die gesamte Projektgruppe betrafen, gab es einen E-Mail-Verteiler. Dieser Verteiler wurde häufig genutzt, um die gesamte Projektgruppe über Änderungen, Protokolle oder Aufgaben zu informieren.

Um einen Ansporn zu schaffen, noch mehr zu kommunizieren, wurde ein weiterer E-Mail-Verteiler eingerichtet. E-Mails, die über diesen Verteiler verschickt wurden, wurden nur den Projektgruppenmitgliedern und nicht den Betreuern zugestellt.

Da einige Projektgruppenteilnehmer auch einen Instant Messenger besitzen, erfolgte die Kommunikation teilweise auch über diesen. Ebenso wurde telefoniert, um komplexe Sachzusammenhänge erklären zu können.

Für die einfache Verfügbarkeit der stets aktuellsten Dokumente sorgte ein automatisches System, welches jedes Dokument in der jeweils aktuellsten Version in einem internen, passwortgeschützten Bereich der Webseite zur Verfügung stellte. Dadurch konnte jederzeit einfach auf

den aktuellsten Stand der Dokumente zurückgegriffen werden. Ebenso wurden auf der Webseite andere für die Projektgruppe wichtige Dateien zur Verfügung gestellt.

Die Bearbeitung der Dokumente, der Modelle und der Software erfolgte unter Verwendung eines Versionsmanagementsystems. Hierfür wurde `git` [git] benutzt. Dieses sorgte für sehr viel Flexibilität, da durch das Konzept von lokalen Kopien, die nicht direkt mit den Daten auf dem Server „gemergt“ werden, weniger Konflikte bei gleichzeitiger Bearbeitung von Dokumenten auftraten. Ebenso sorgte das Konzept der „Branches“ dafür, dass Dokumentation und Entwicklung streng getrennt waren. Außerdem konnten durch das Erstellen temporärer Branches experimentelle Entwicklungen getestet werden, ohne dass sie auf die eigentliche Entwicklung in einem anderen Branch Einfluss nahmen.

Vor den wichtigen Meilensteinen wurde eine Erreichbarkeit am Wochenende vereinbart, damit schnelle Nachfragen bei Problemen möglich waren.

7.2.4 Gruppenräume

Dass der Projektgruppe, zumindest die meiste Zeit lang, fast ein gesamtes Gebäude zur Verfügung stand, wirkte sich sehr positiv auf das Arbeitsklima aus. Hier war genug Freiraum für alle Mitglieder vorhanden und es konnten problemlos Unterlagen deponiert werden, sodass am Folgetag die Arbeit ohne großen Aufwand an der gleichen Stelle wieder fortgesetzt werden konnte. Ebenso wurde so ein zentraler Treffpunkt geschaffen, an dem Kleingruppentreffen oder aber auch Sitzungen statt finden konnten. Dabei musste nicht auf andere Nutzer Rücksicht genommen werden und auch kein Raum zuvor reserviert werden, was zu einer sehr hohen Flexibilität führte.

Dadurch, dass auch einige ehemalige Büros mit schließbarer Tür vorhanden waren, war auch ein ungestörtes Arbeiten möglich, wenn andere Teilnehmer in Gespräche vertieft waren.

Leider kam es gelegentlich zu Ausfällen der Internetverbindung. Diese machten sich sehr bemerkbar, da so auch z.B. die Test-Datenbank oder das Versionsmanagementsystem nicht mehr nutzbar waren. Ebenso konnte Matlab in diesem Fall nicht genutzt werden, da die Verbindung zum Lizenzserver fehl schlug. Solche Ausfälle haben die Produktivität sehr eingeschränkt.

Desweiteren wurde kurzzeitig eine andere Abteilung des OFFIS in das Gebäude mit den Räumen der Projektgruppe ausgelagert. Dies führte dazu, dass Räume besetzt waren, die zuvor genutzt werden konnten, wodurch die Produktivität der Projektgruppe zu dieser Zeit leicht absank.

Zwischenzeitlich standen den Teilnehmern außerdem zu wenig Arbeitsrechner zur Verfügung, sodass nicht alle Projektgruppenteilnehmer zeitgleich arbeiten konnten. Dies konnte dadurch ausgeglichen werden, dass einige Projektgruppenteilnehmer an ihrem privaten Laptop arbeiteten. Vor allem der Mangel an Arbeitsrechnern mit installiertem SILAB-Fahrsimulator machte sich beim Erstellen der Testskripte bemerkbar, da hier mehrere Teilgruppen ihre Entwicklungen testen wollten, es aber nicht gleichzeitig konnten. Dies hing allerdings auch damit zusammen, dass einer der beiden SILAB-Rechner zwischenzeitlich an ein Projektgruppenmitglied verliehen war, damit dieser zu Hause mit dem SILAB-Fahrsimulator arbeiten konne.

Hier konnte keine konkrete Zeitplanung erfolgen, da der Bedarf teilweise spontan entstand und außerdem, wie bereits in Abschnitt 7.2.2 beschrieben, eine Terminfindung nicht einfach war.

Leider kam es während der Projektlaufzeit auch zwei mal dazu, dass Arbeitsrechner aufgrund von Hardwaredefekten ausfielen. Hierdurch kam es zu kleinen Verzögerungen, da die Hardware zunächst ersetzt werden musste.

7.2.5 Erfahrungen

Trotz des teilweise hohen Zeitaufwandes hat die Projektgruppe dennoch viel Spaß bereitet. Vor allem das gemeinsame Arbeiten und das Erfolgserlebnis bei funktionierenden Komponenten waren sehr motivierend.

Während der Projektgruppenlaufzeit wurden neue Kompetenzen erworben. Durch die Gruppenarbeit konnte ebenso die Teamfähigkeit verbessert werden. Auch wurde die Fähigkeit erworben, Arbeits- und Zeitaufwand für Aufgaben besser einzuschätzen.

Diese Erfahrungen können sich für das spätere Berufsleben als wichtig herausstellen.

7.2.6 Danksagungen

Ebenso hat es sich als sehr hilfreich erwiesen, dass die Projektgruppe in Kooperation mit dem OFFIS angeboten wurde. Hier gibt es bereits einige erfahrene Mitarbeiter, die sich mit Teilgebieten des zu entwickelnden Systems beschäftigt haben. Bei Problemen fand man immer ein offenes Ohr und einen Ansprechpartner im OFFIS. Häufig wurde sofort geholfen oder mindestens auf einen Kollegen verwiesen, der weiterhelfen konnte.

Besonders sei an dieser Stelle auf Lars Weber verwiesen, der sehr bei der Nutzung des SILAB-Fahrsimulators geholfen hat. Die Systemadministratoren des OFFIS haben uns ebenfalls tatkräftig unterstützt und uns bei der Verwaltung und Wartung der Arbeitsrechner geholfen.

Auch die Mitarbeiter der Universität Oldenburg haben uns sehr unterstützt. Die HLA-Nutzung wäre ohne die Hilfe von Stefan Puch nicht möglich gewesen. Detlef Janßen hat bei der Codegenerierung und der CAN-Kommunikation geholfen und uns auch die entsprechende CAN-Hardware entwickelt. Die Regler-Stabilitätstest waren nur dank der Unterstützung durch Jens Oehlerking möglich. Henning Jost danken wir besonders für das Korrektur lesen dieses Dokuments.

Nicht zuletzt gebührt den Betreuern, Günter Ehmen und Kim Grüttner, ein großer Dank, da sie eine sehr große Unterstützung für die Projektgruppe waren und ein so gutes Gelingen ohne sie nicht möglich gewesen wäre.

A Anhang

A.1 Bilder

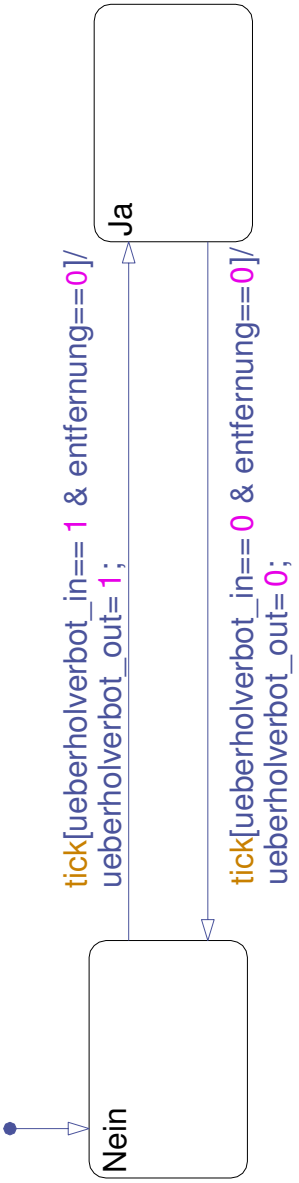


Abbildung A.1: Automat Speicher_Überholverbot

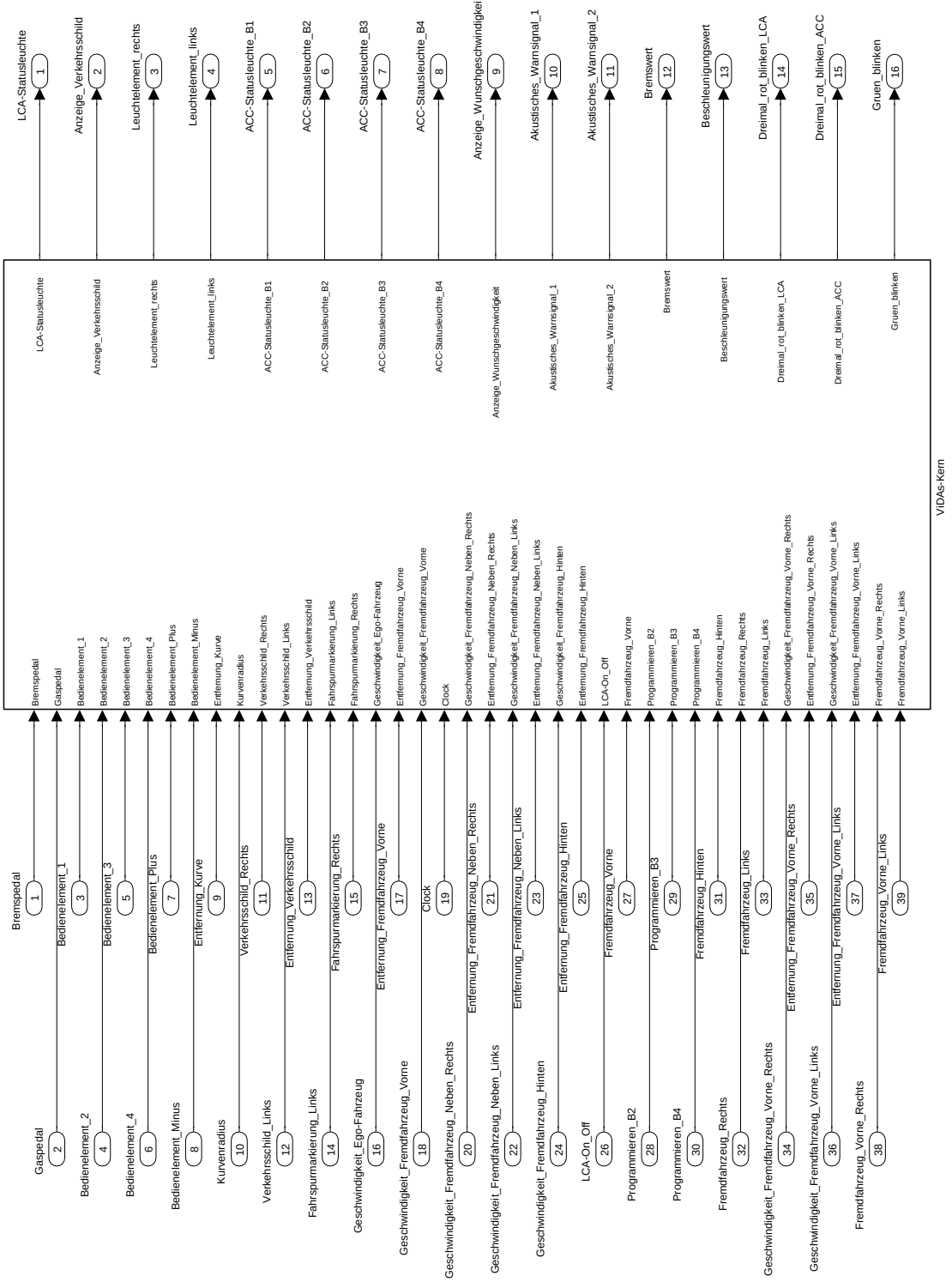


Abbildung A.2: ViDAs-Kern

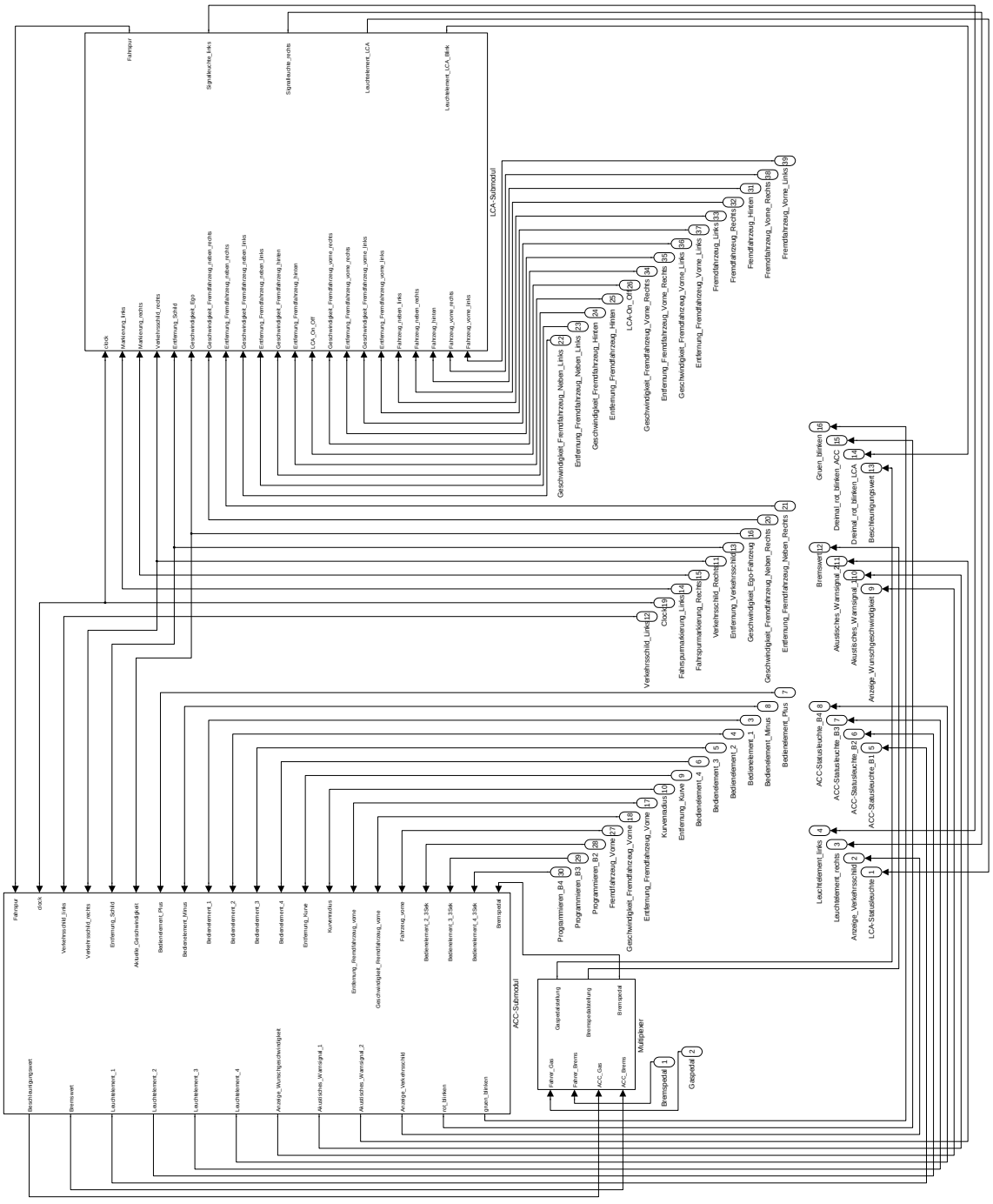


Abbildung A.3: Übersicht ViDAs-Kern

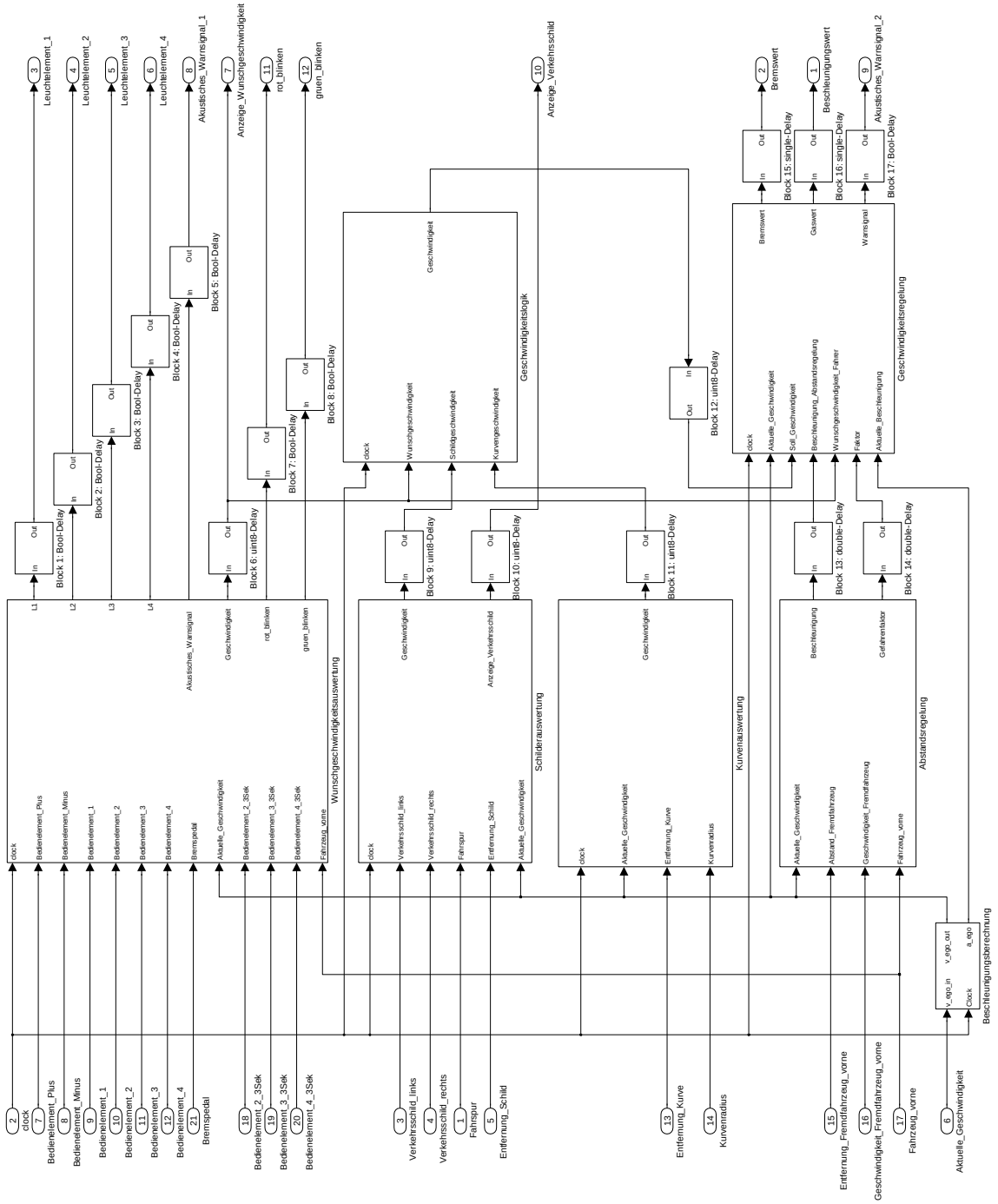


Abbildung A.4: Übersicht ACC

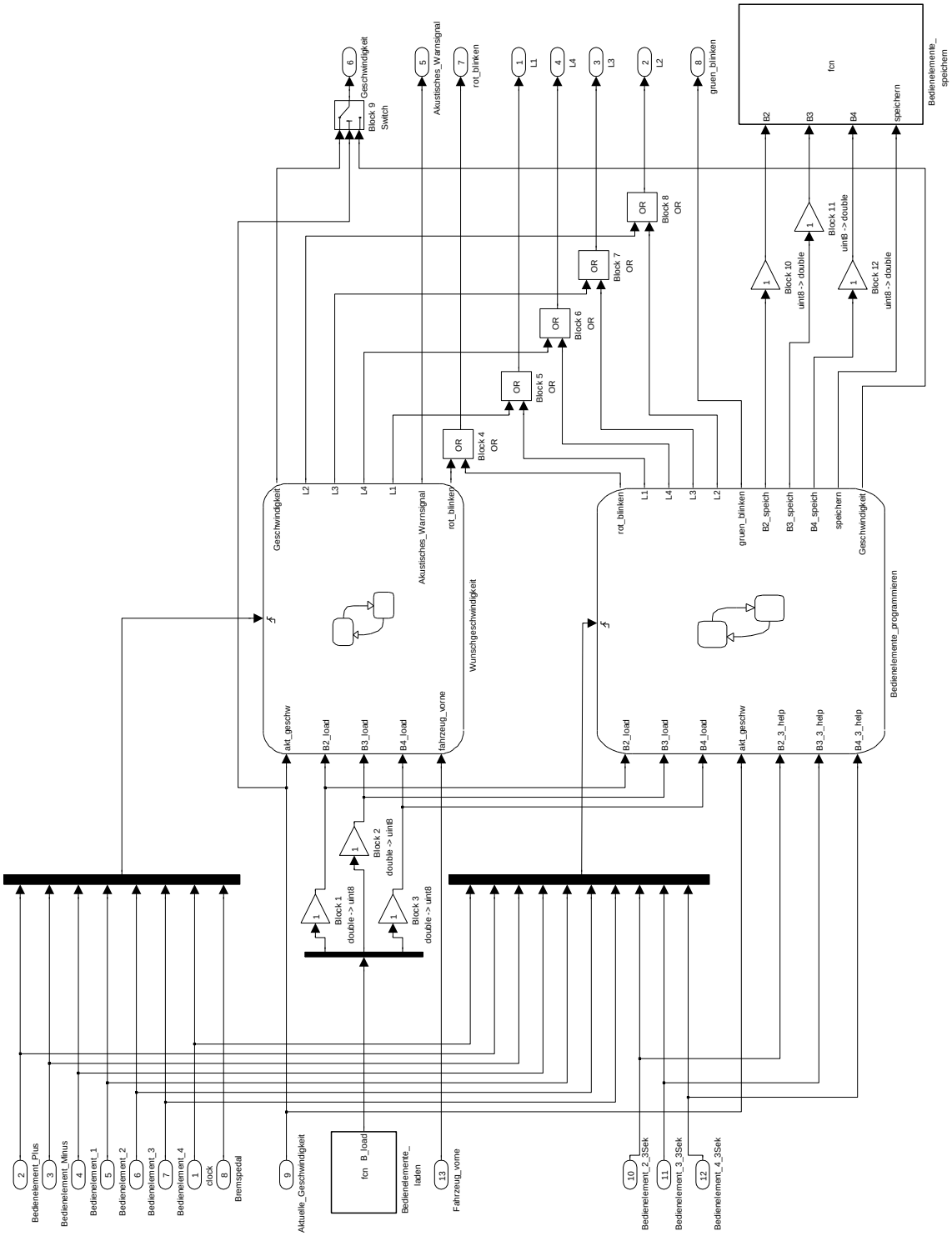


Abbildung A.5: Submodul Wunschgeschwindigkeitsauswertung

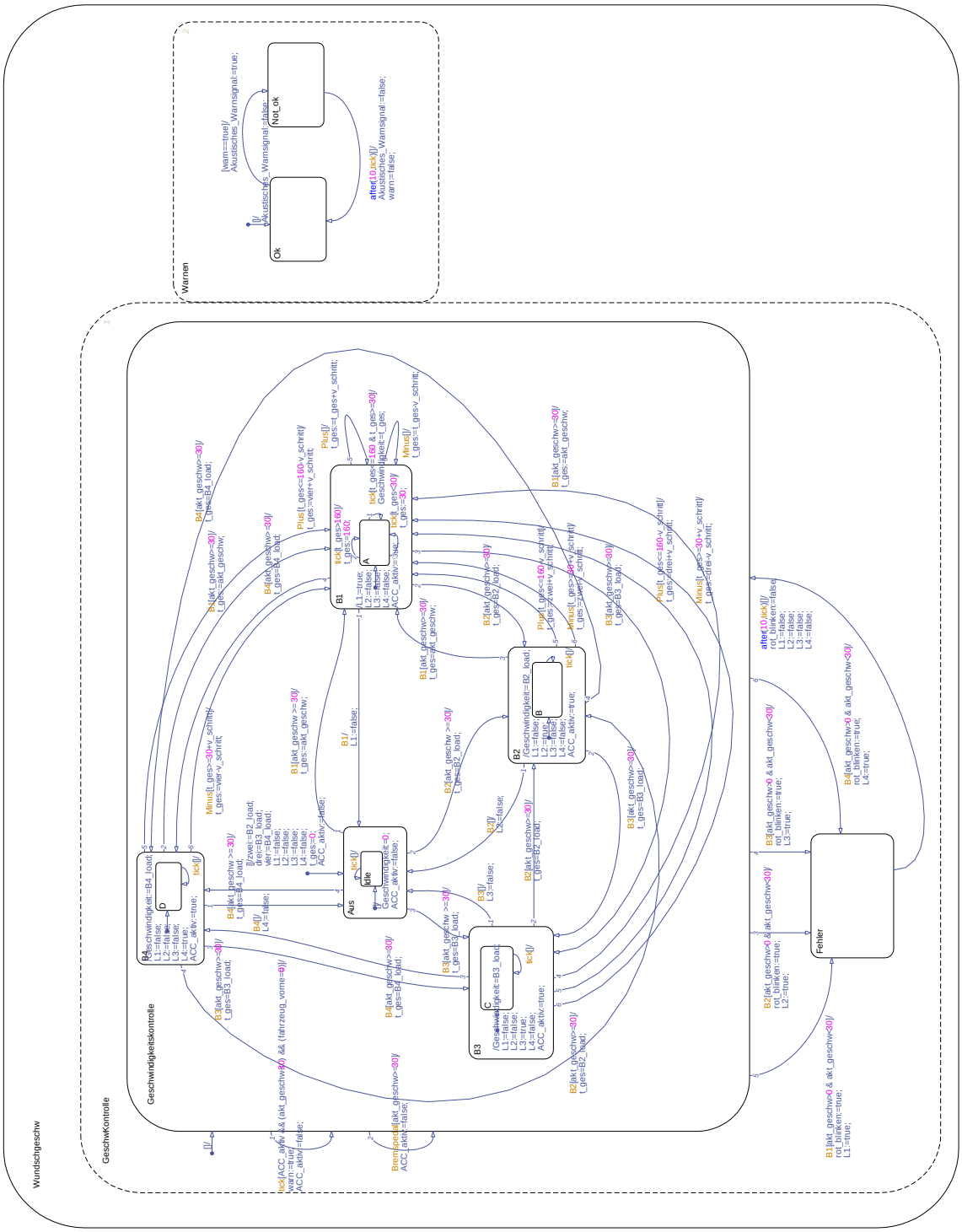


Abbildung A.6: Automat Wunschgeschwindigkeit

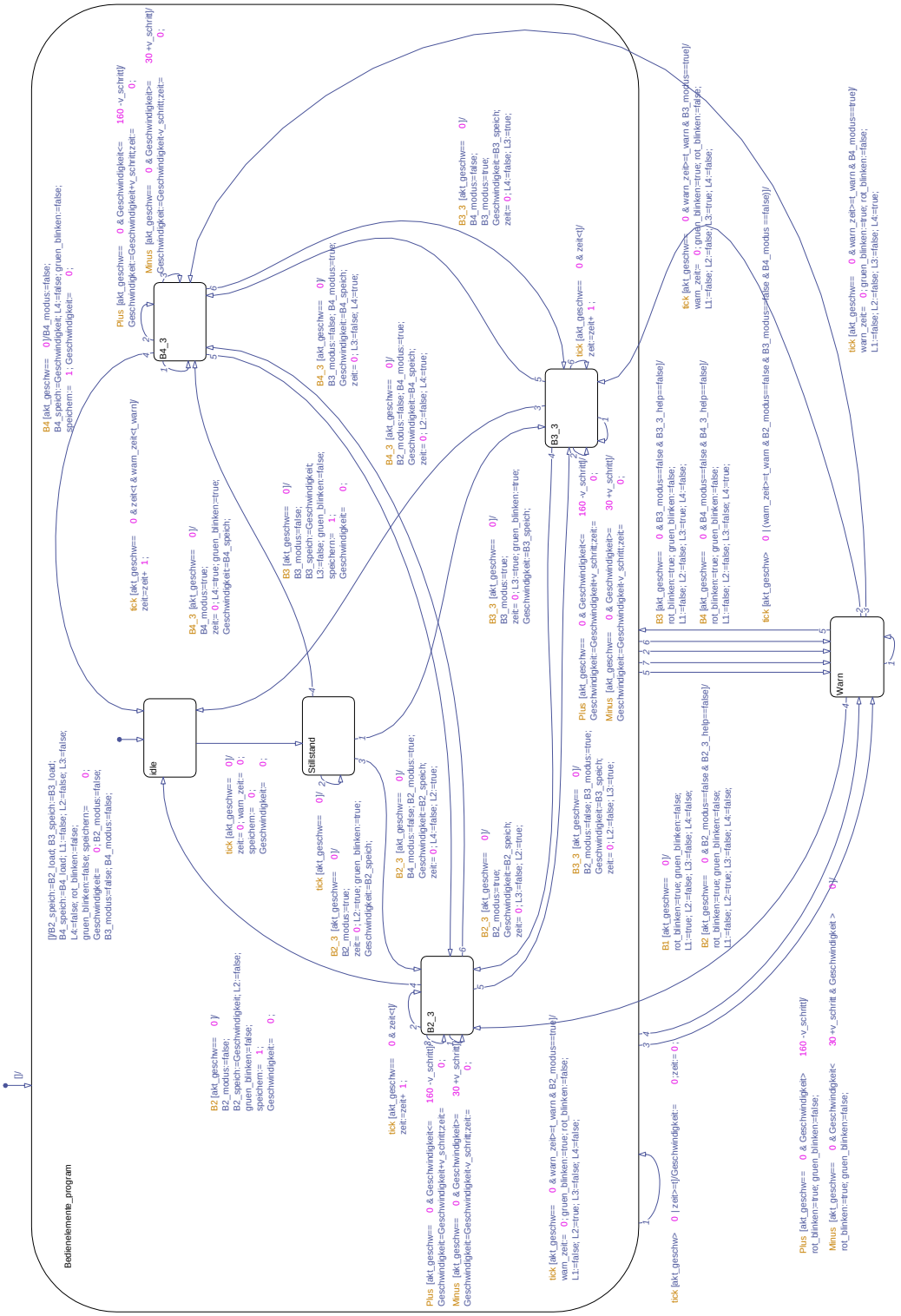


Abbildung A.7: Automat Bedienelemente_programmieren

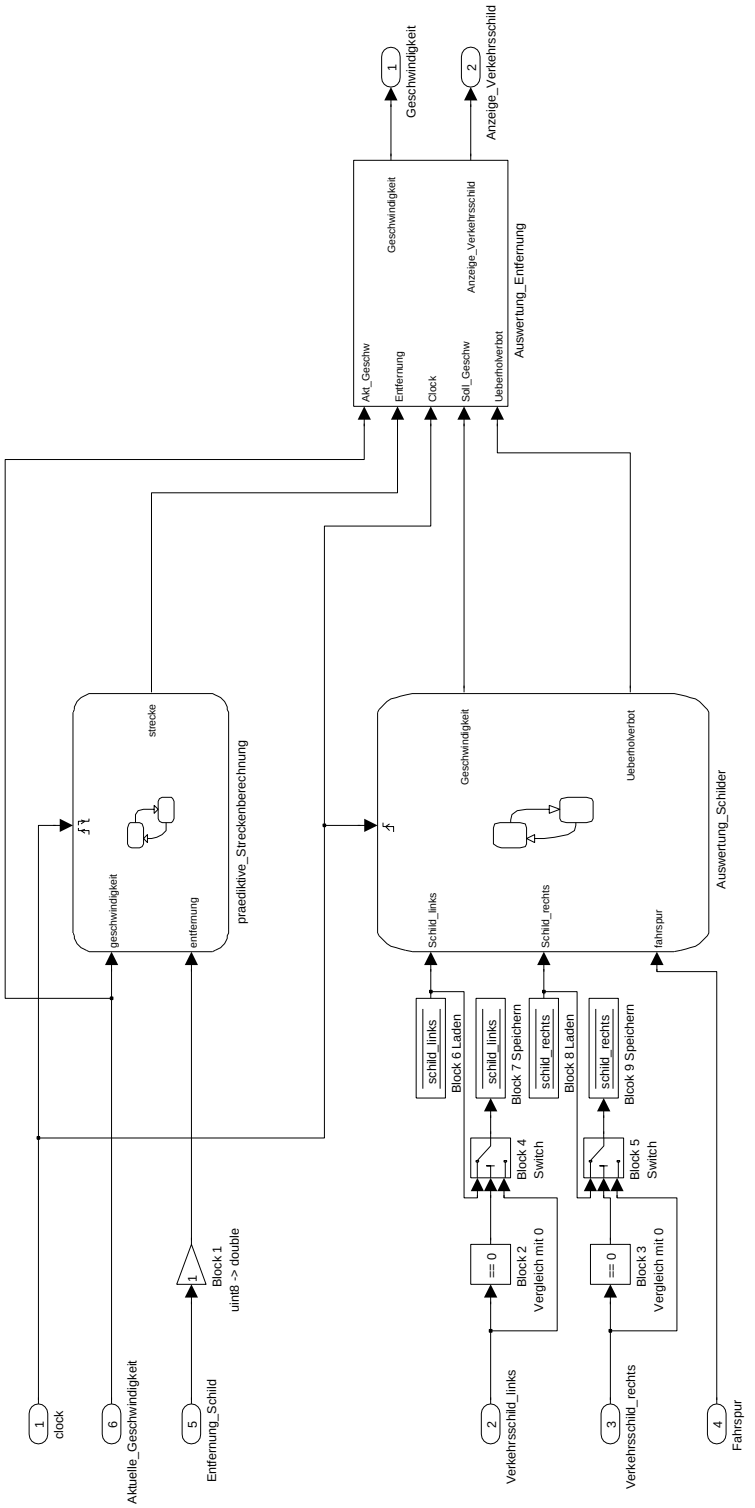


Abbildung A.8: Submodul Schildauswertung

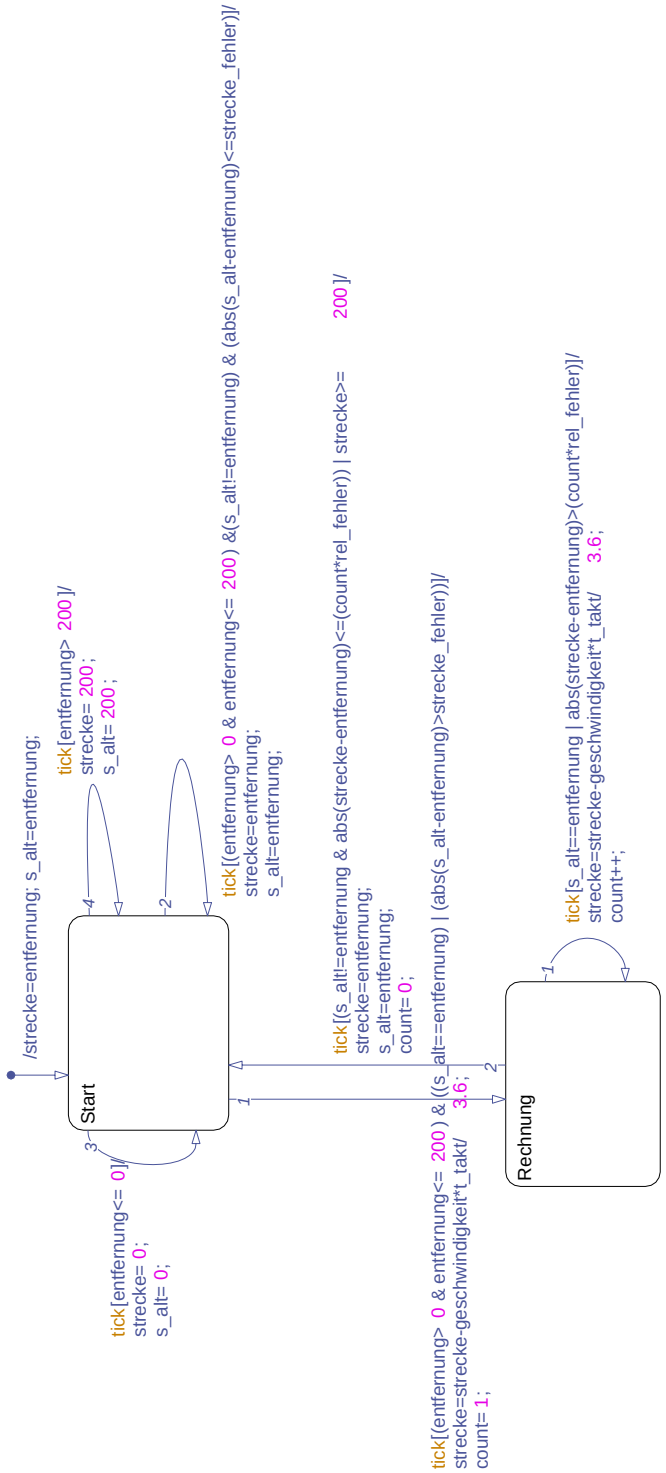


Abbildung A.9: Automat prädiktive_Streckenberechnung

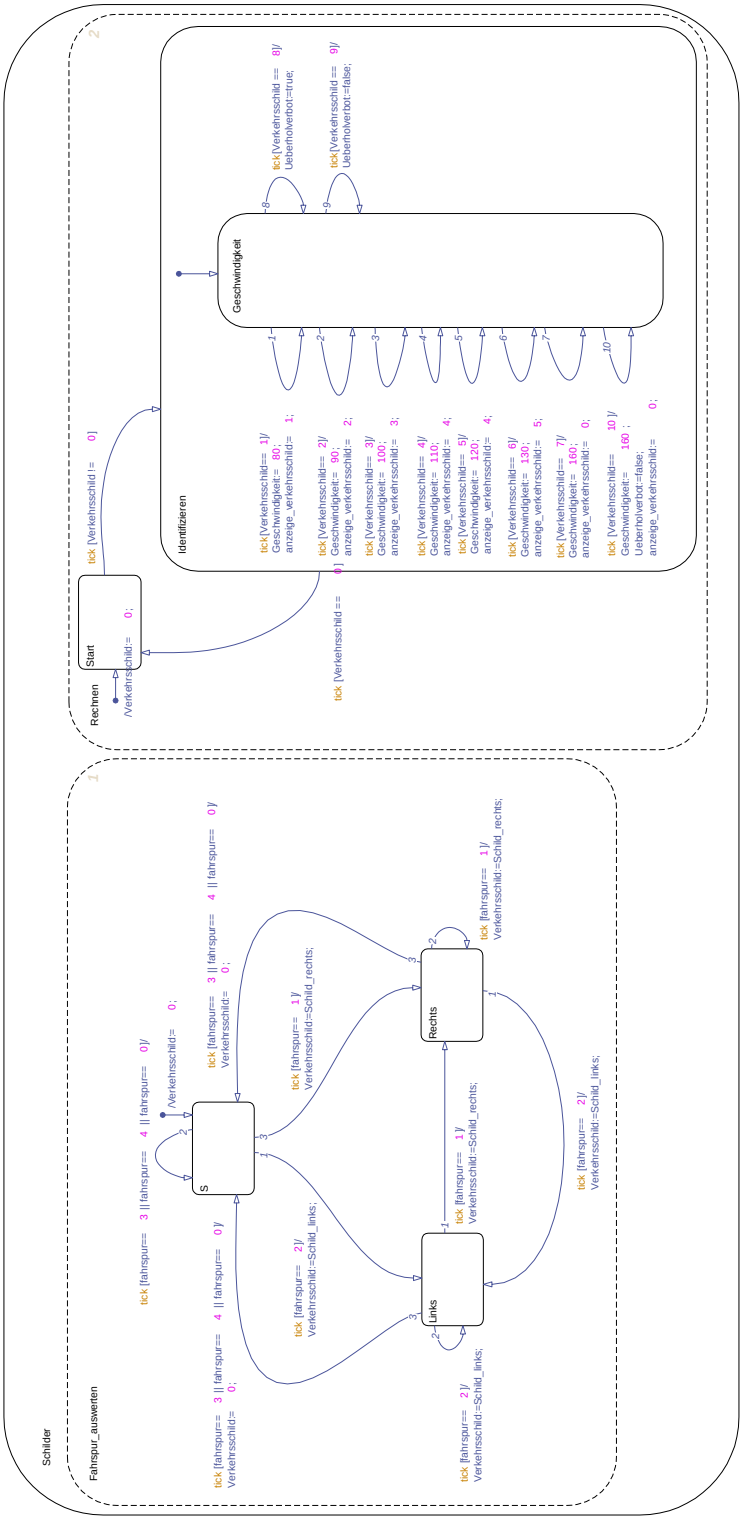


Abbildung A.10: Automat Auswertung_Schilder

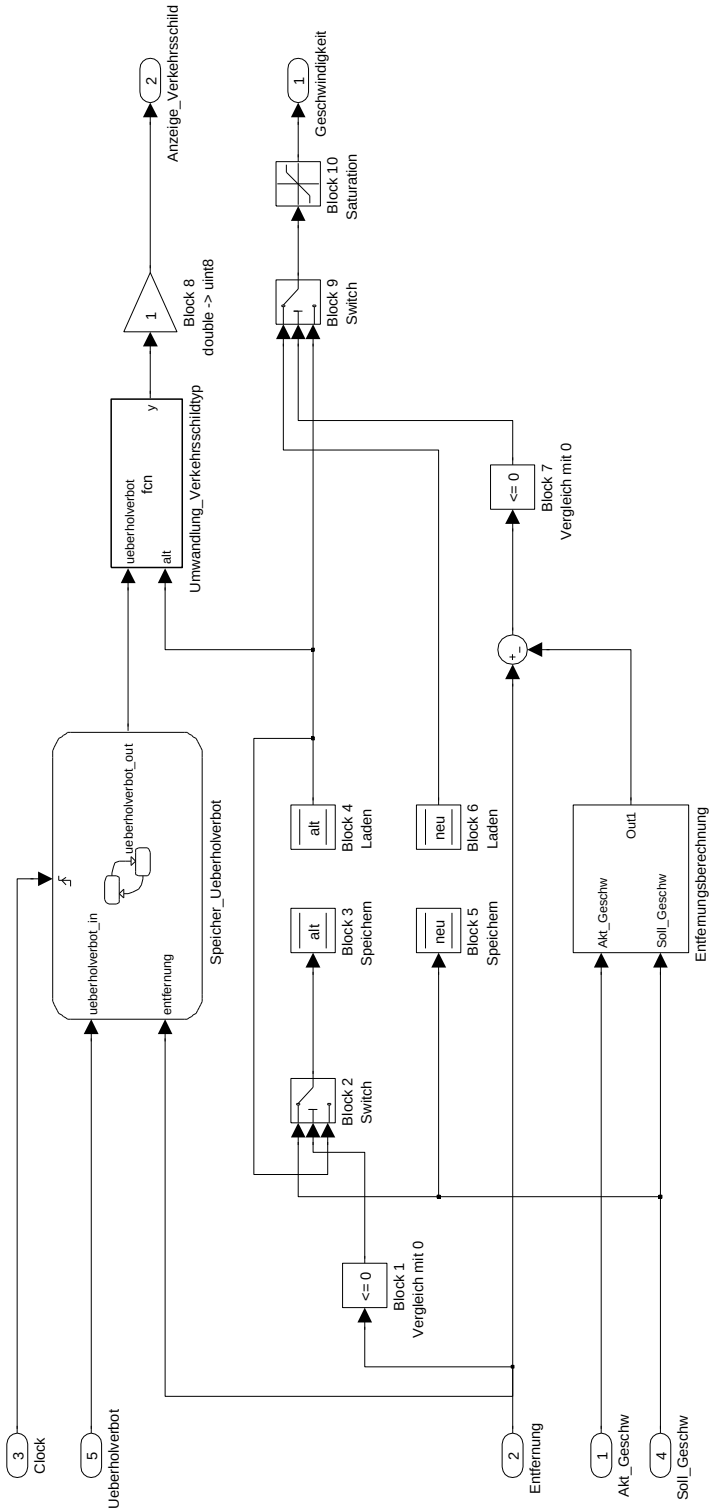


Abbildung A.11: Submodul Auswertung_Entfernung

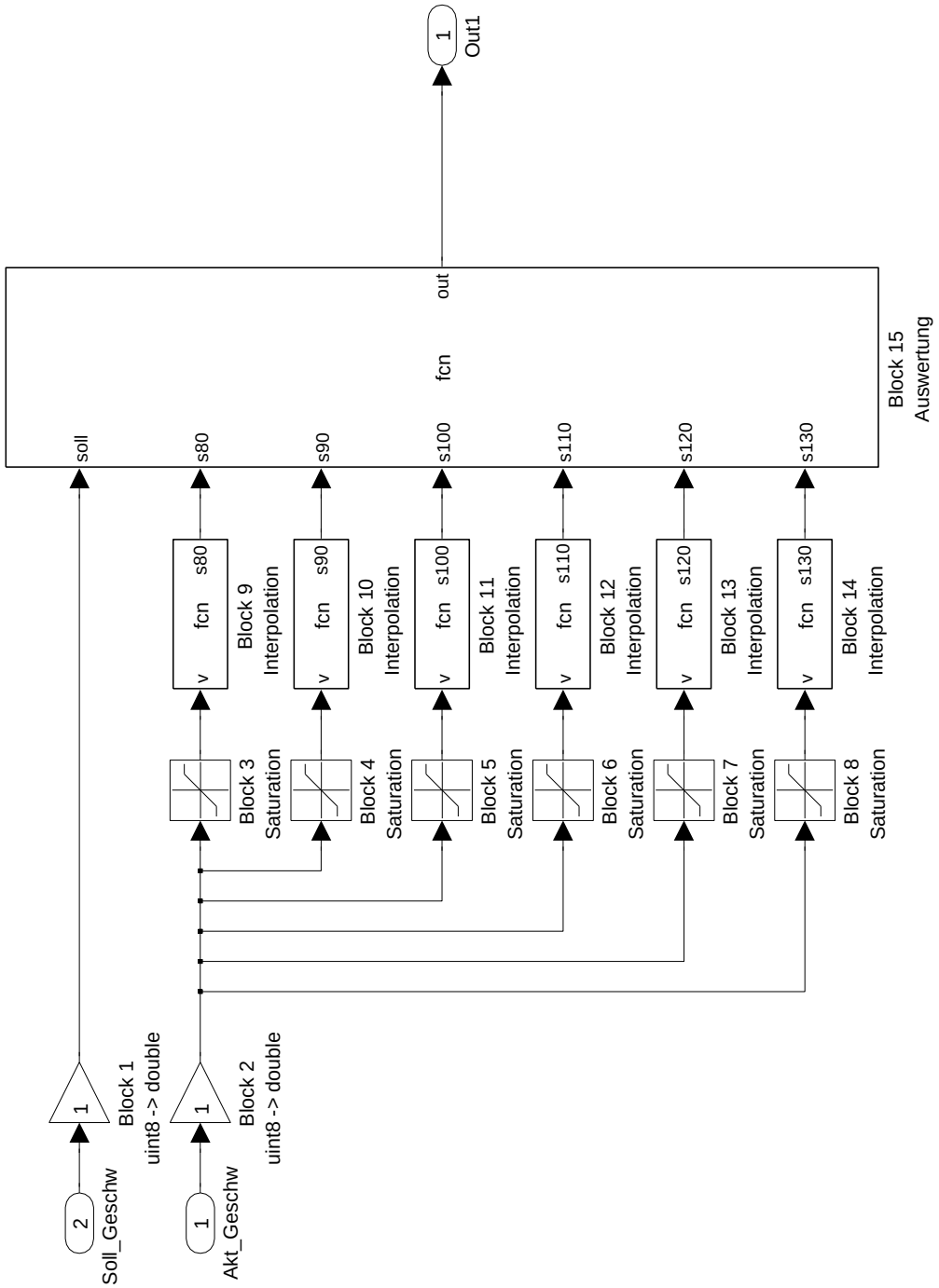


Abbildung A.12: Submodul Entfernungsberechnung

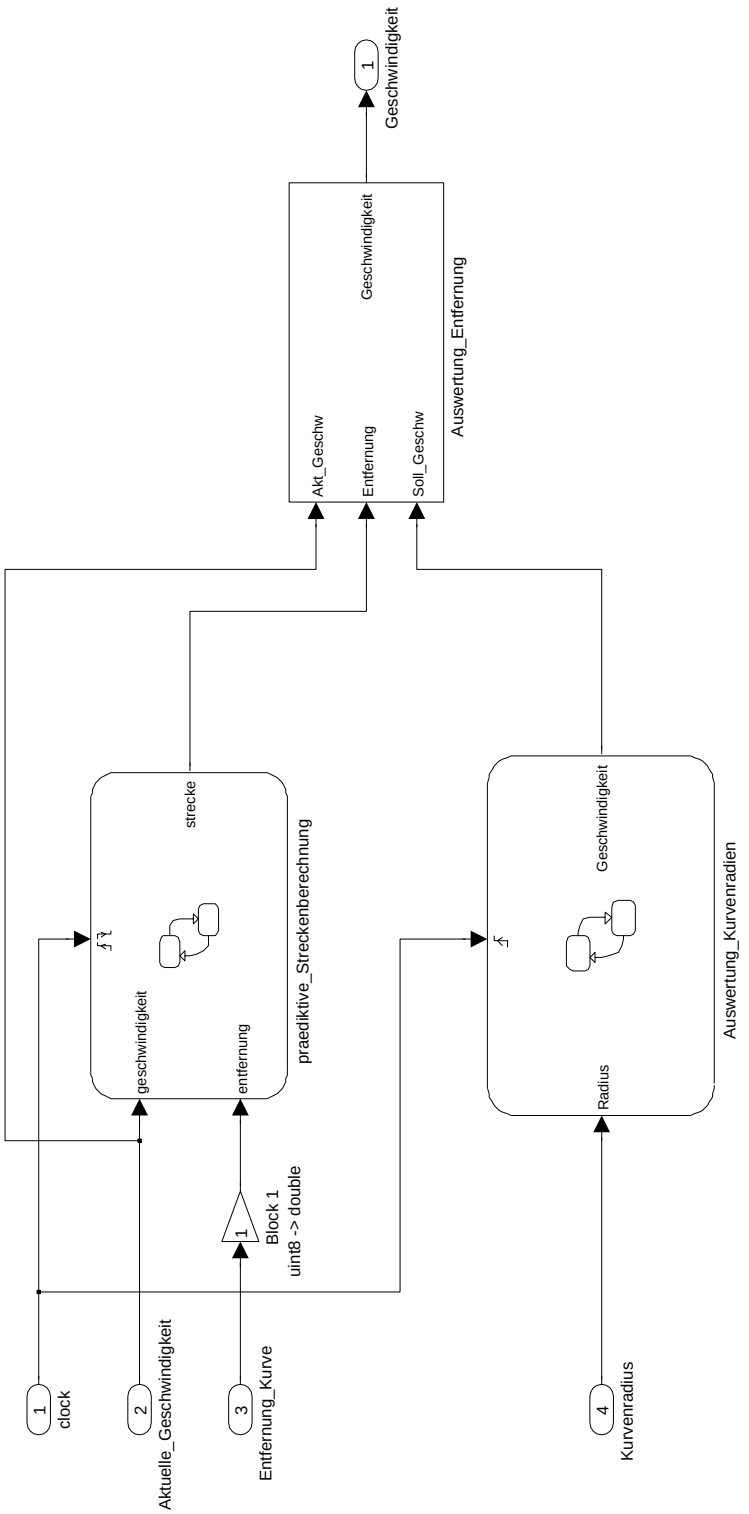


Abbildung A.13: Submodul Kurvenauswertung

Abbildung A.14: Automat Auswertung_Kurvenradien

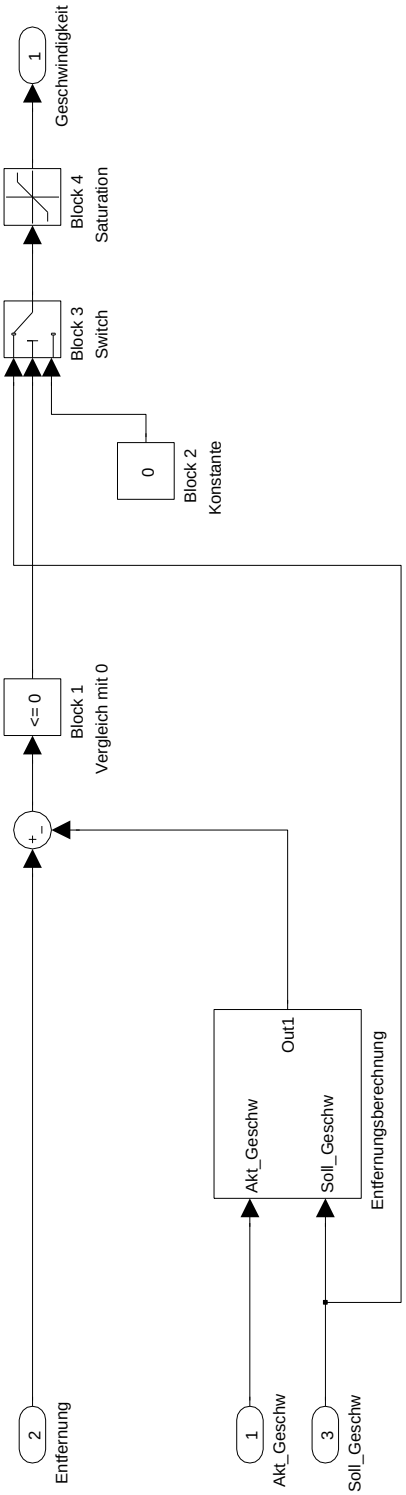


Abbildung A.15: Submodul Auswertung_Entfernung

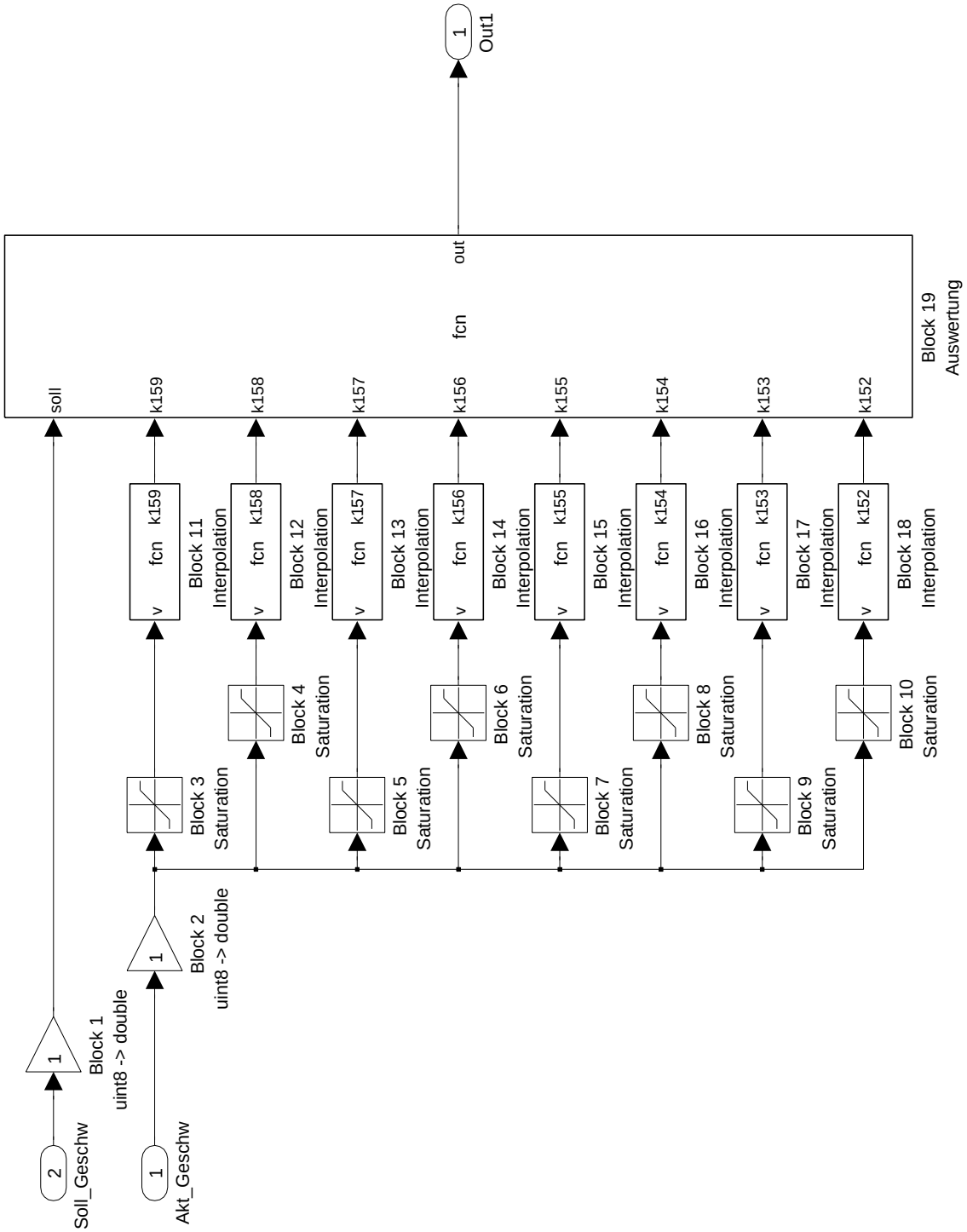


Abbildung A.16: Submodul Entfernungsberechnung

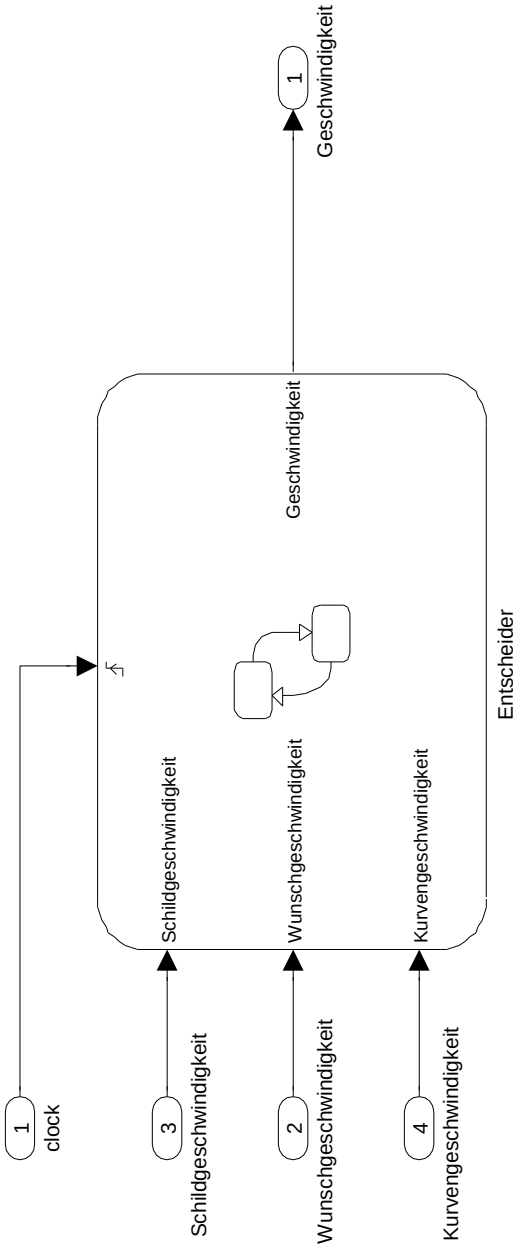


Abbildung A.17: Submodul Geschwindigkeitslogik

Abbildung A.18: Automat Entscheider

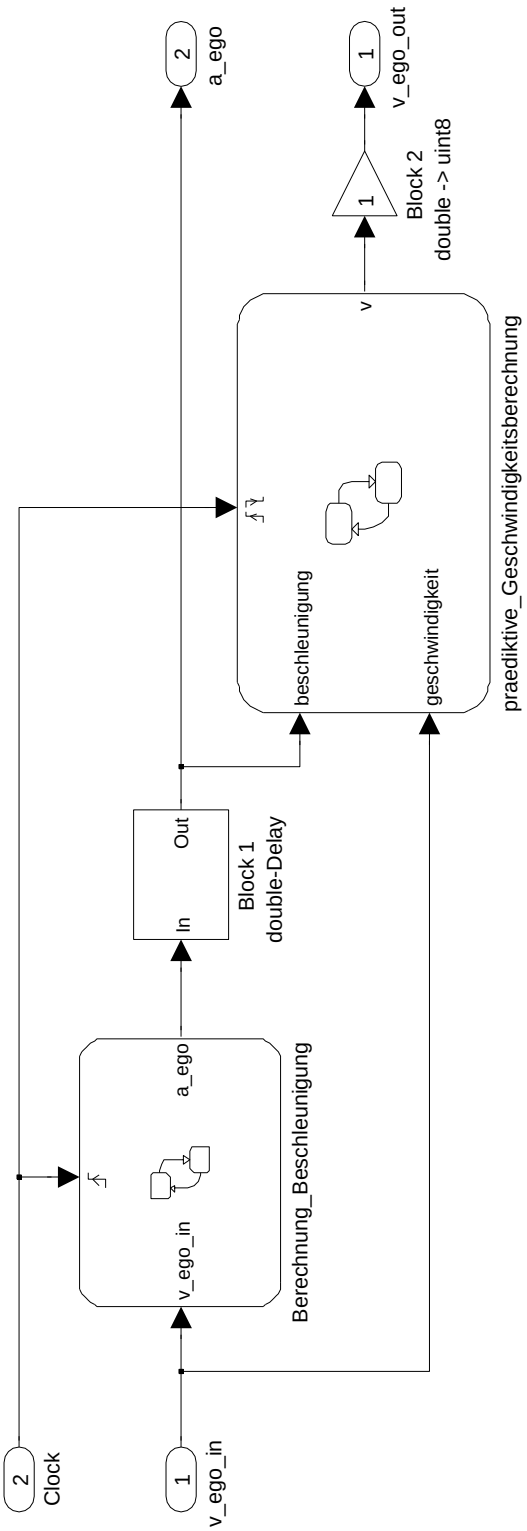


Abbildung A.19: Submodul Beschleunigungsberechnung

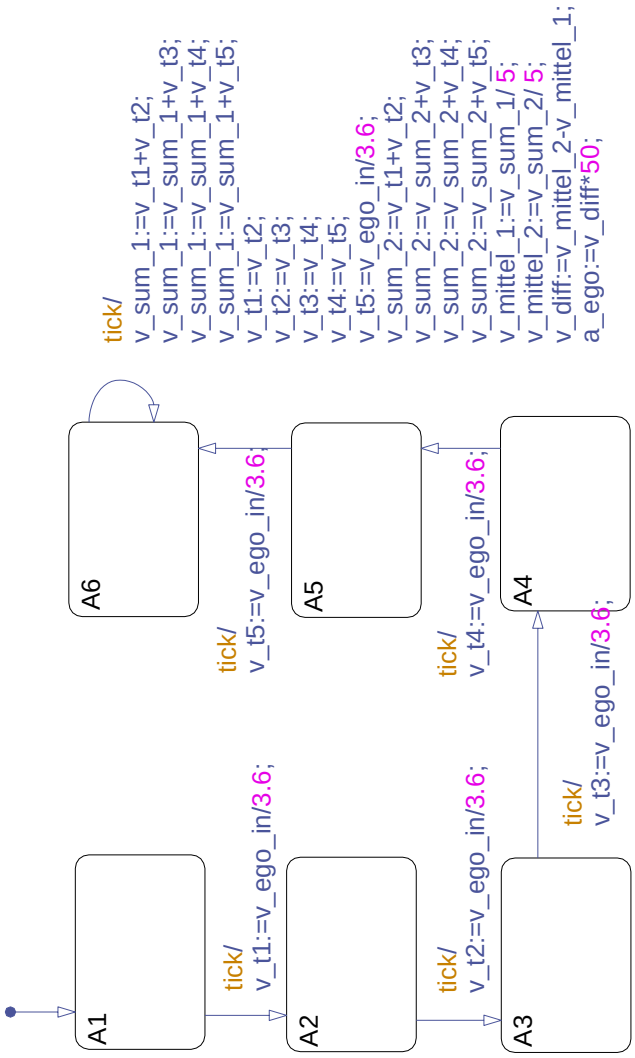


Abbildung A.20: Automat Berechnung_Beschleunigung

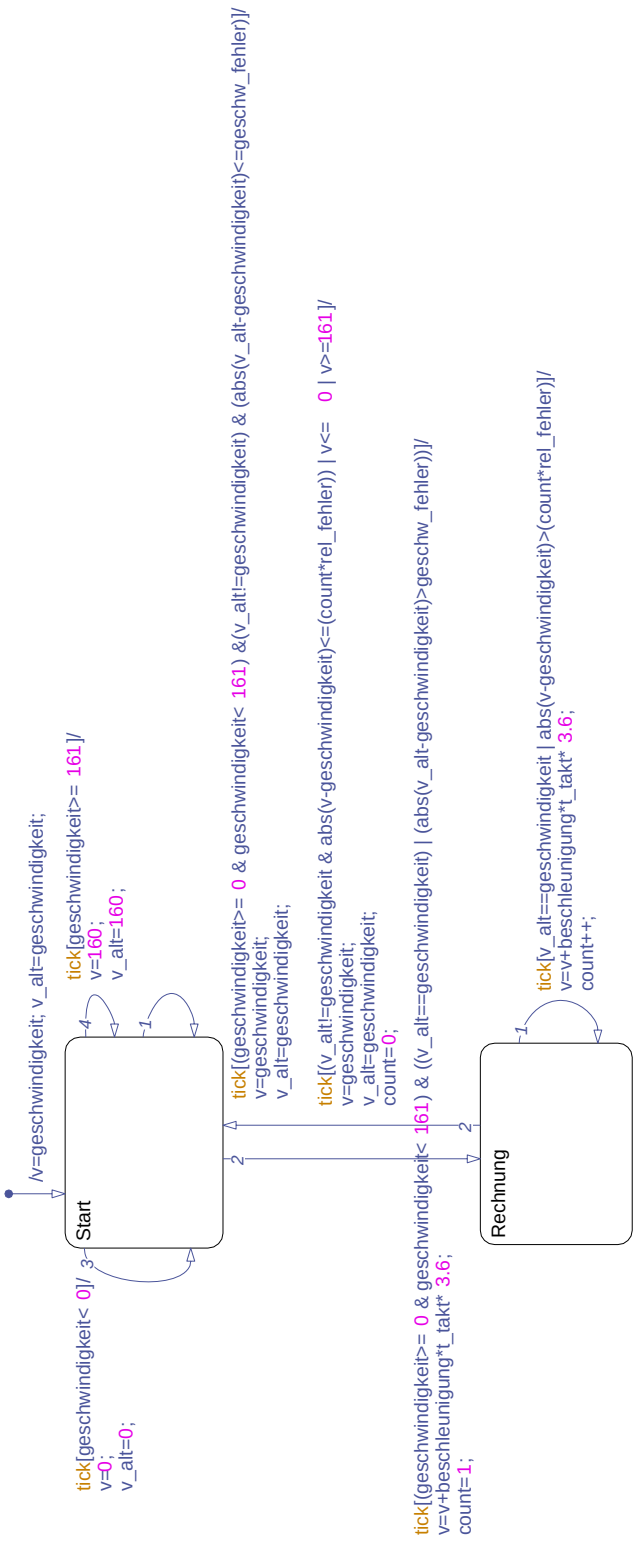


Abbildung A.21: Automat prädiktive_Geschwindigkeitsberechnung

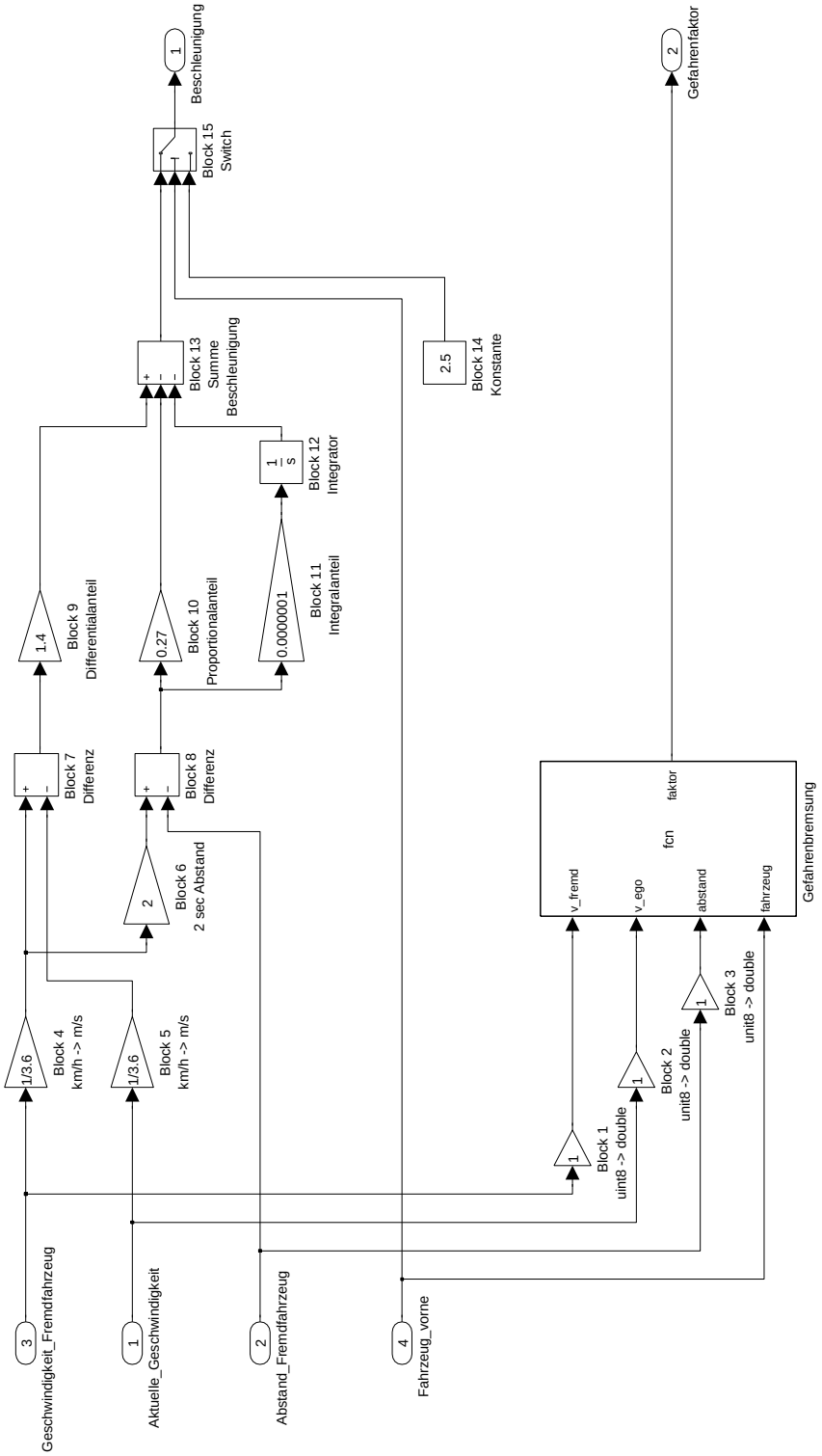


Abbildung A.22: Submodul Abstandsregler

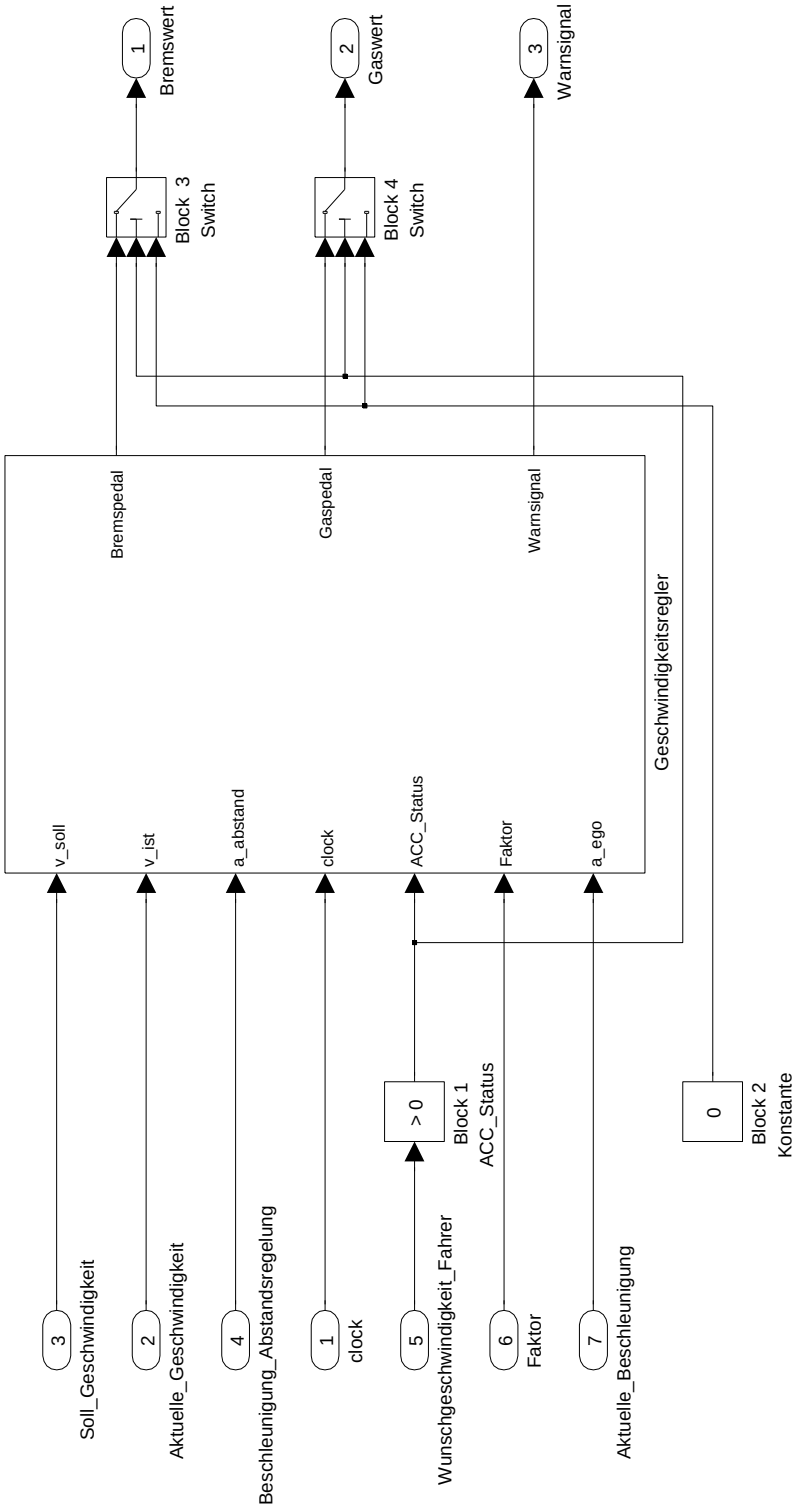


Abbildung A.23: Submodul Geschwindigkeitsregler

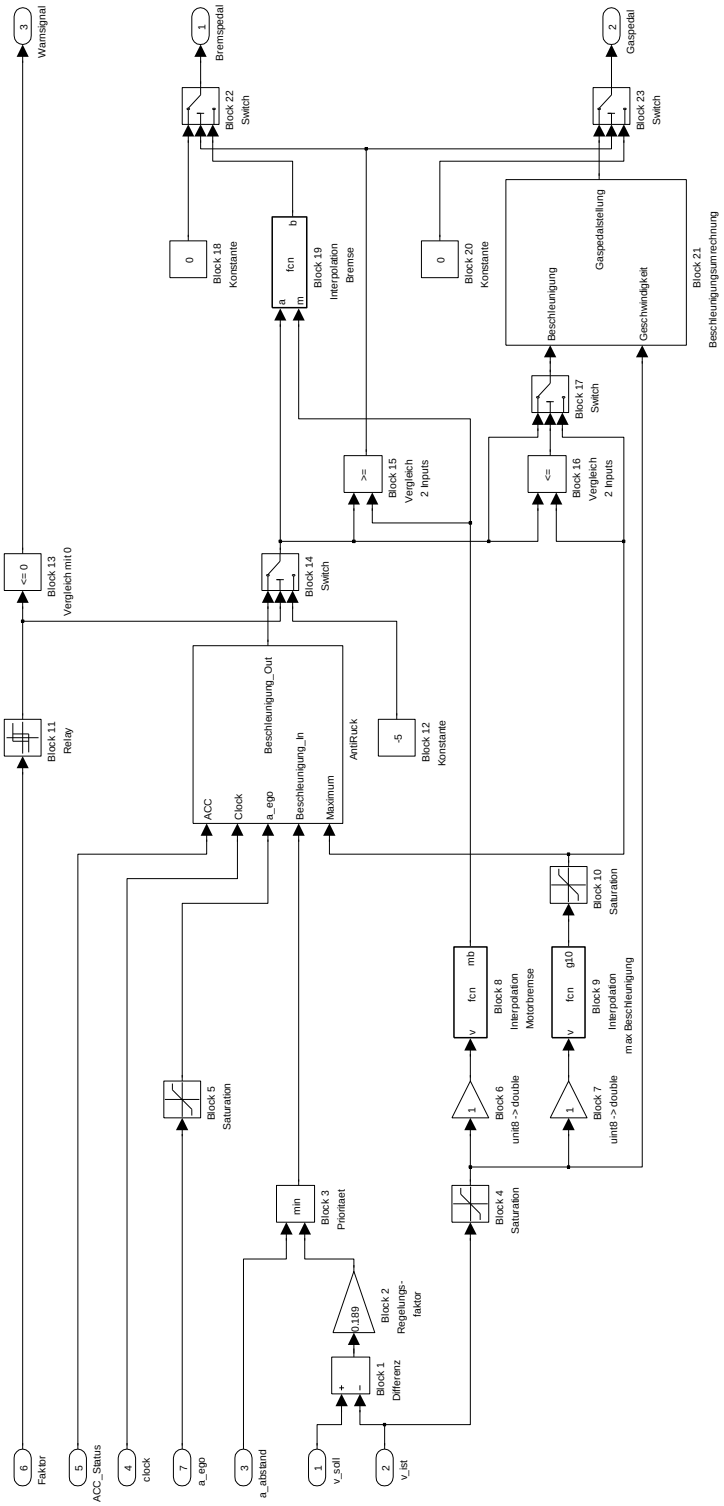


Abbildung A.24: Submodul Geschwindigkeitsregler

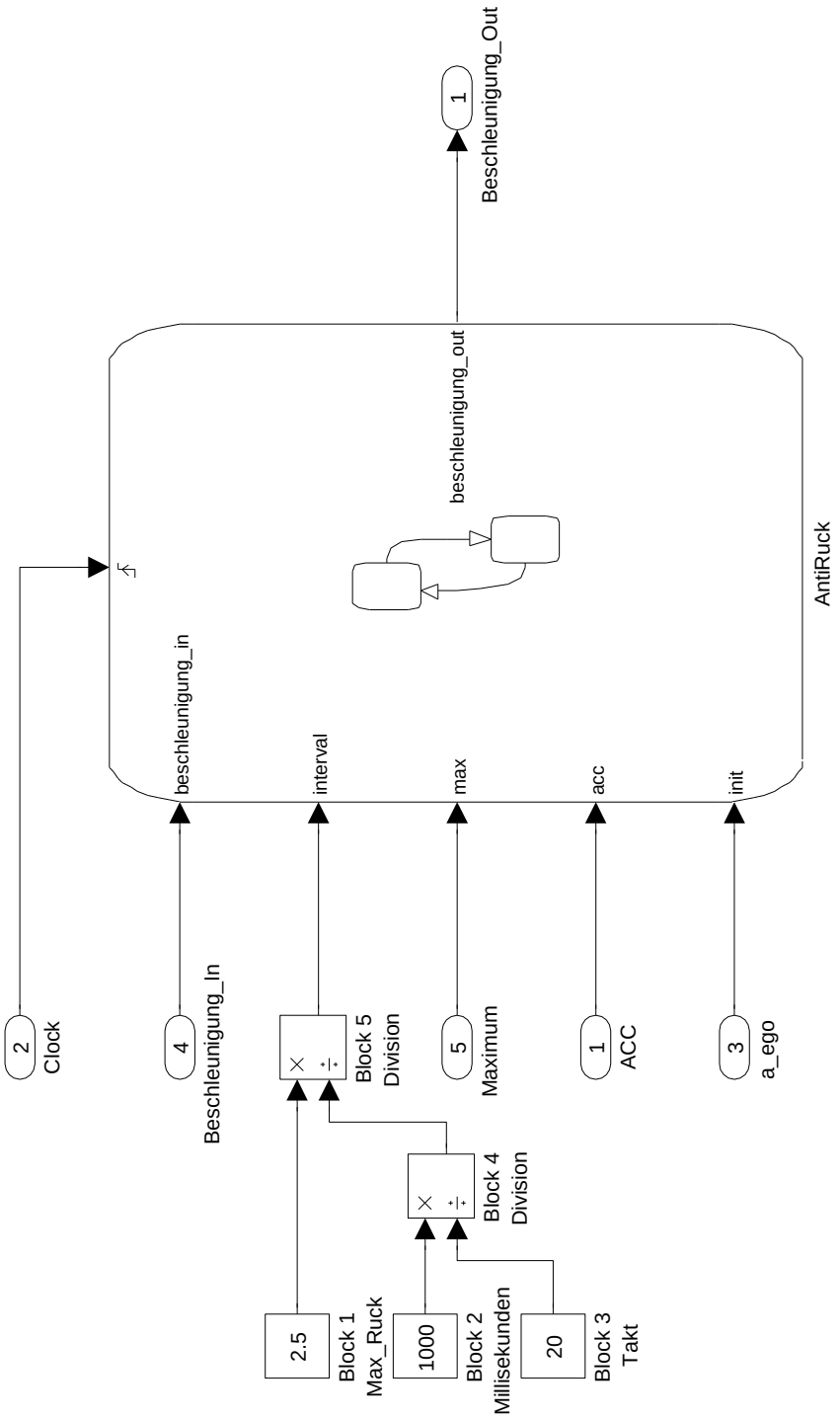


Abbildung A.25: Submodul AntiRuck

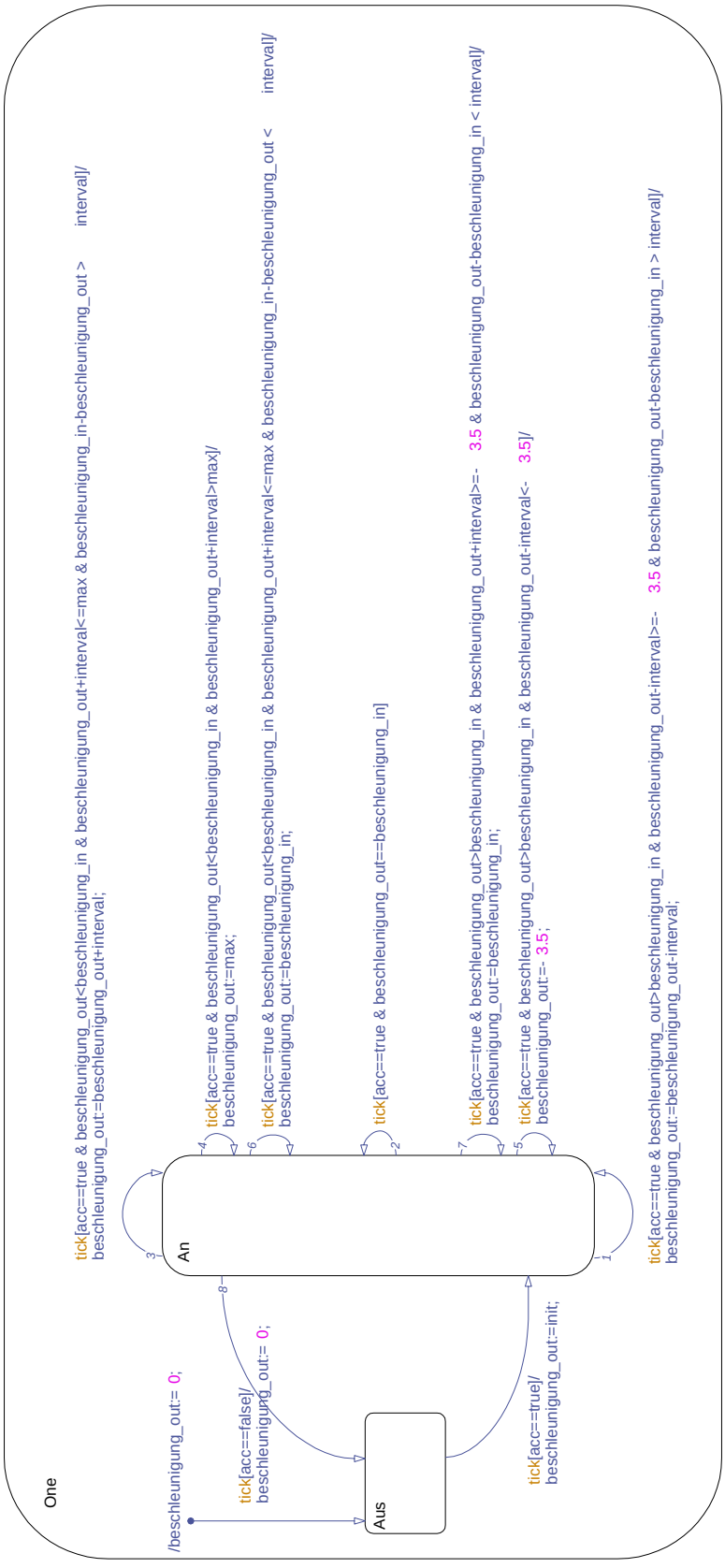


Abbildung A.26: Automat AntiRuck

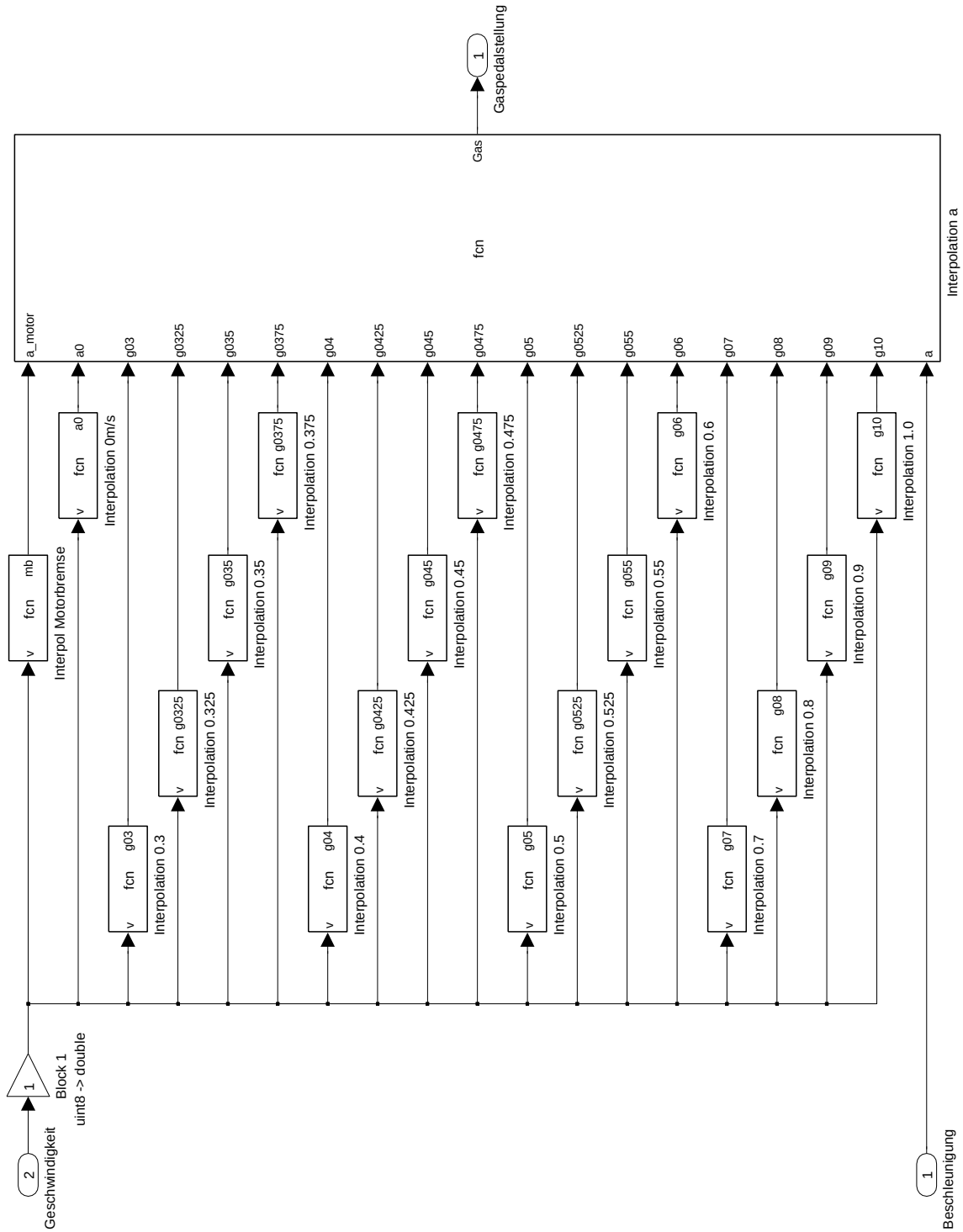


Abbildung A.27: Submodul Beschleunigungumrechnung

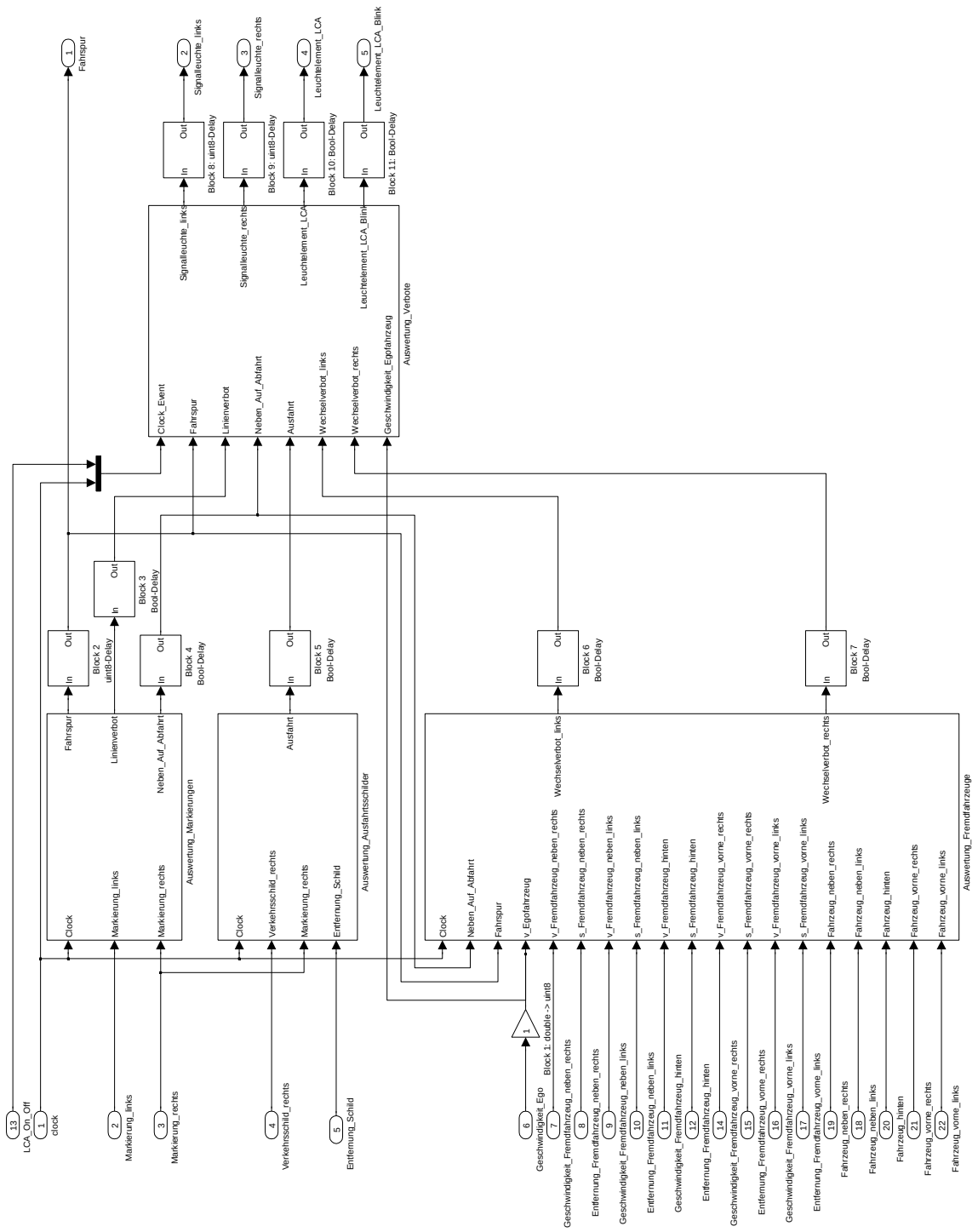


Abbildung A.28: Übersicht LCA

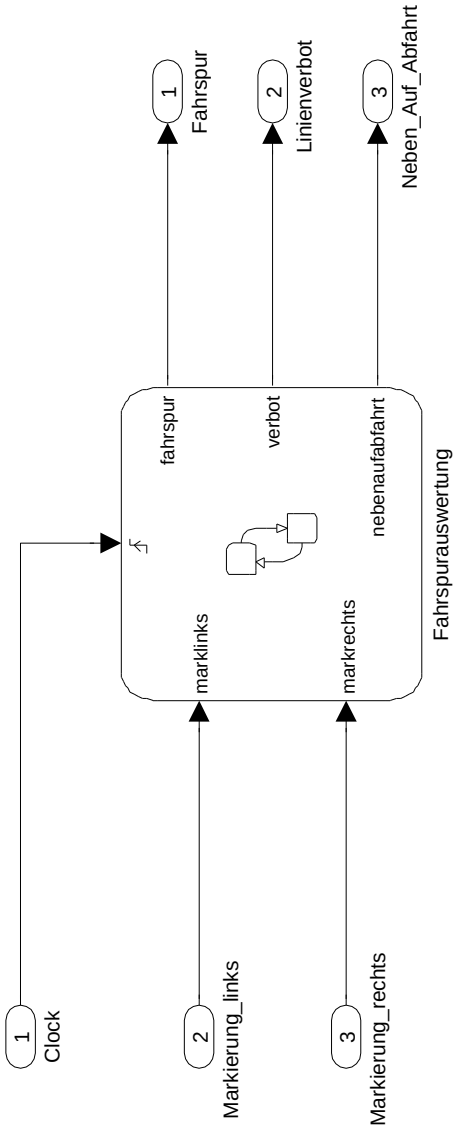


Abbildung A.29: Submodul Auswertung_Markierungen

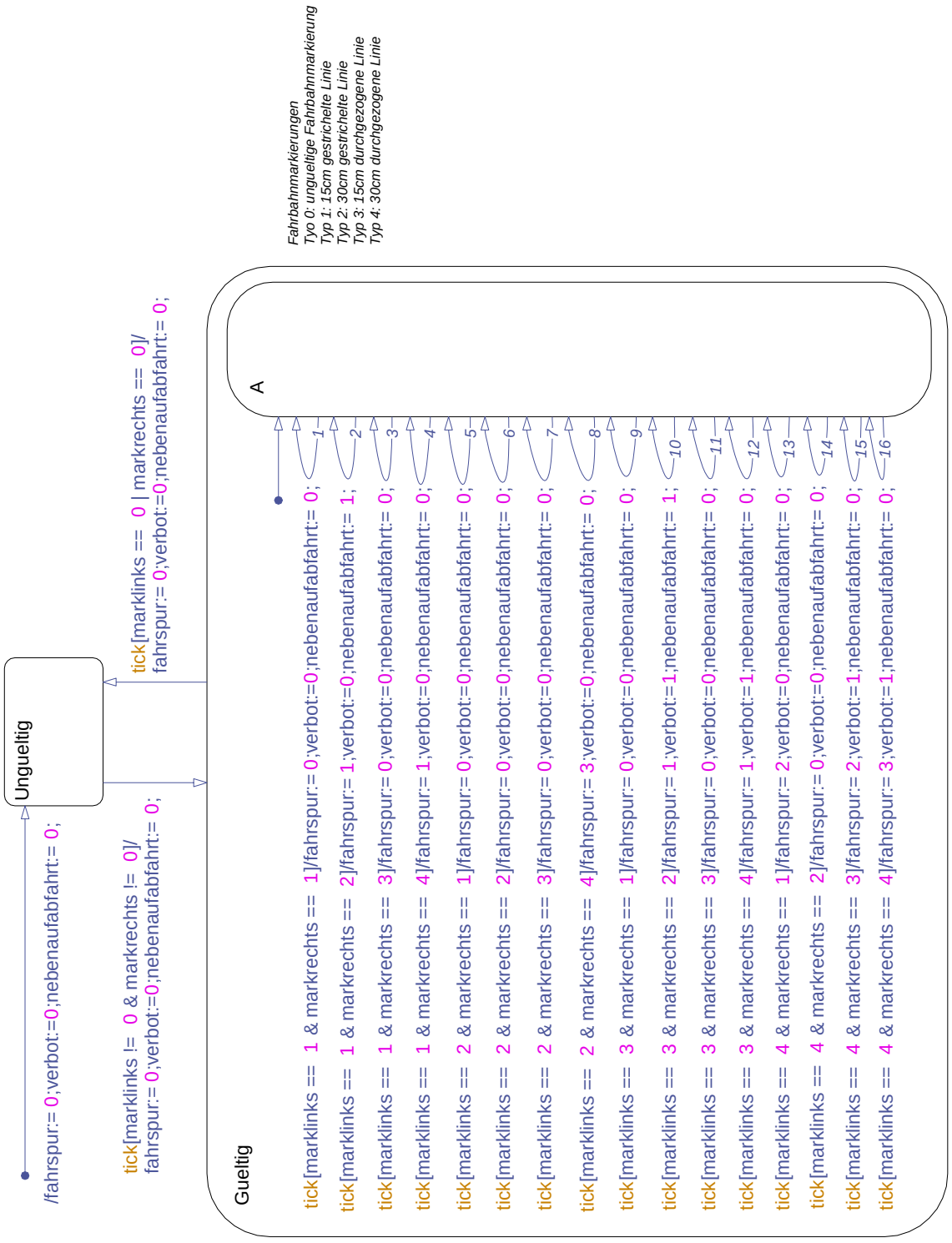


Abbildung A.30: Automat Fahrspurauswertung

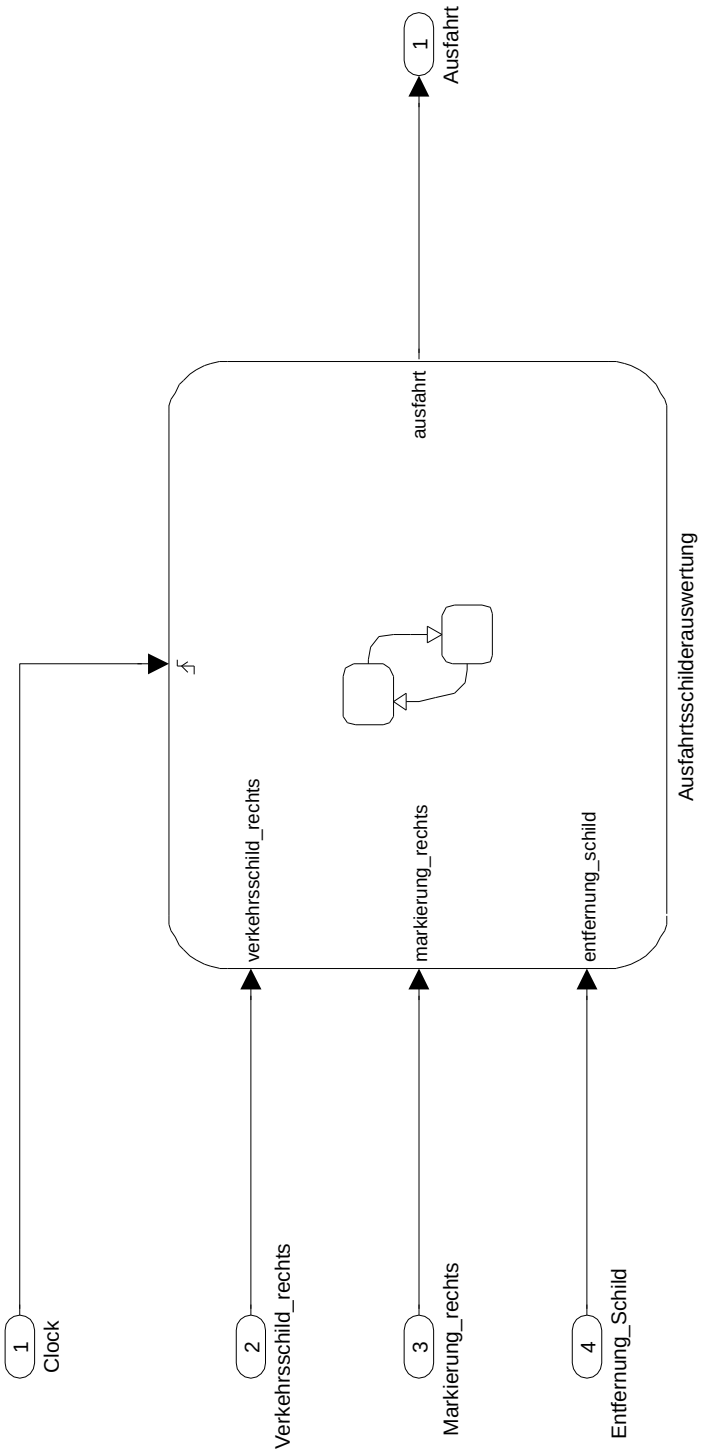


Abbildung A.31: Submodul Auswertung_Ausfahrtsschilder

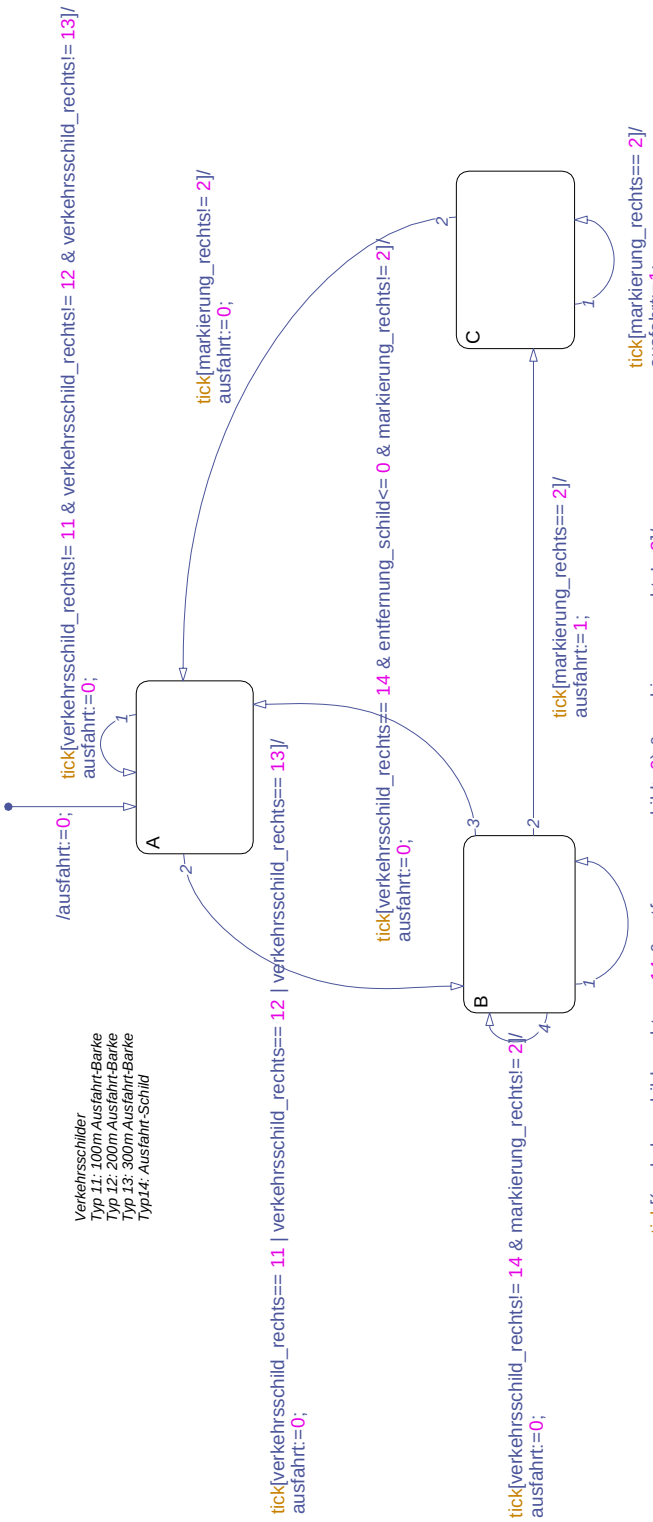


Abbildung A.32: Automat Ausfahrtsschilderauswertung

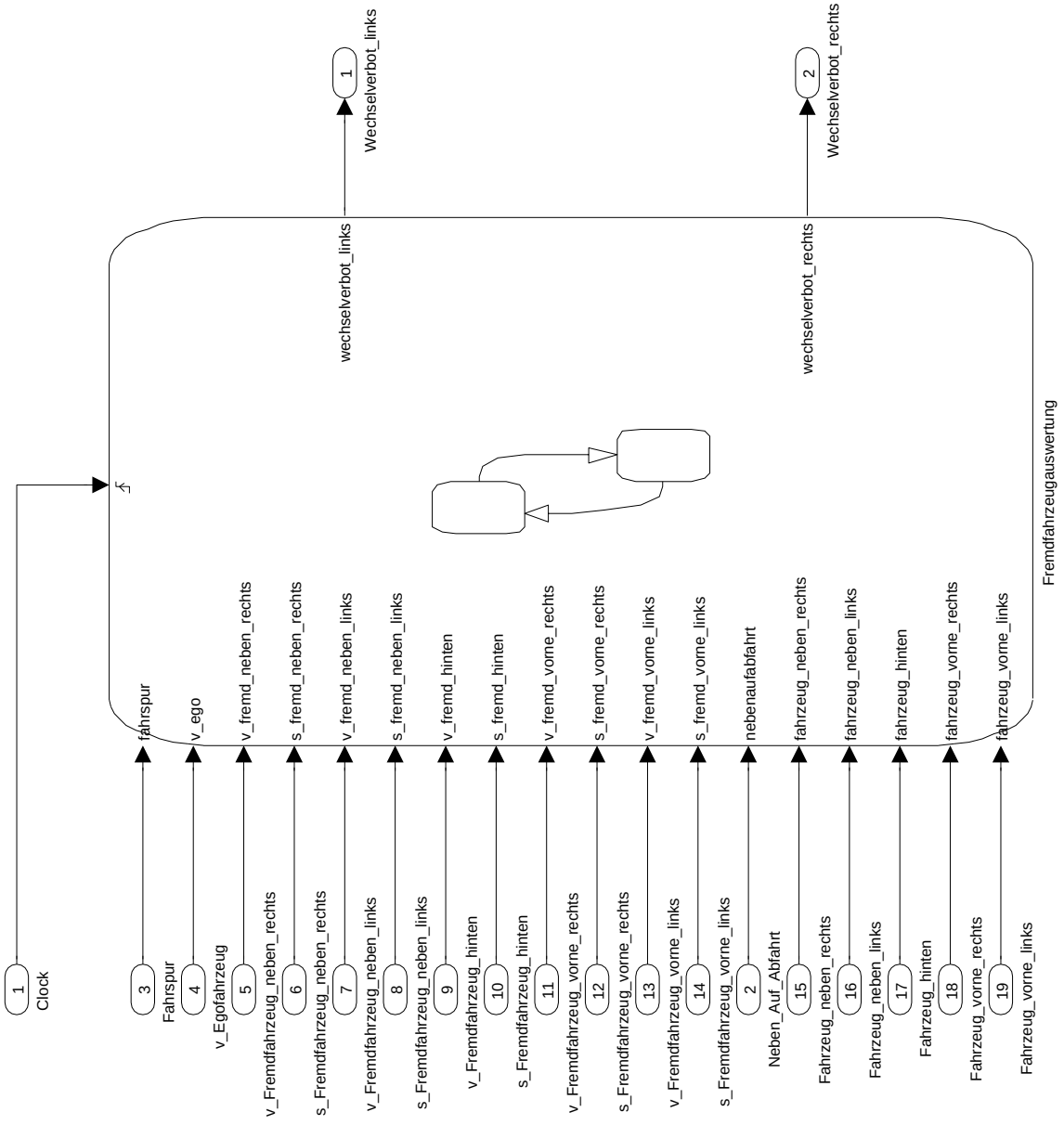


Abbildung A.33: Submodul Auswertung_Fremdfahrzeuge

Abbildung A.34: Automat Fremdfahrzeugauswertung

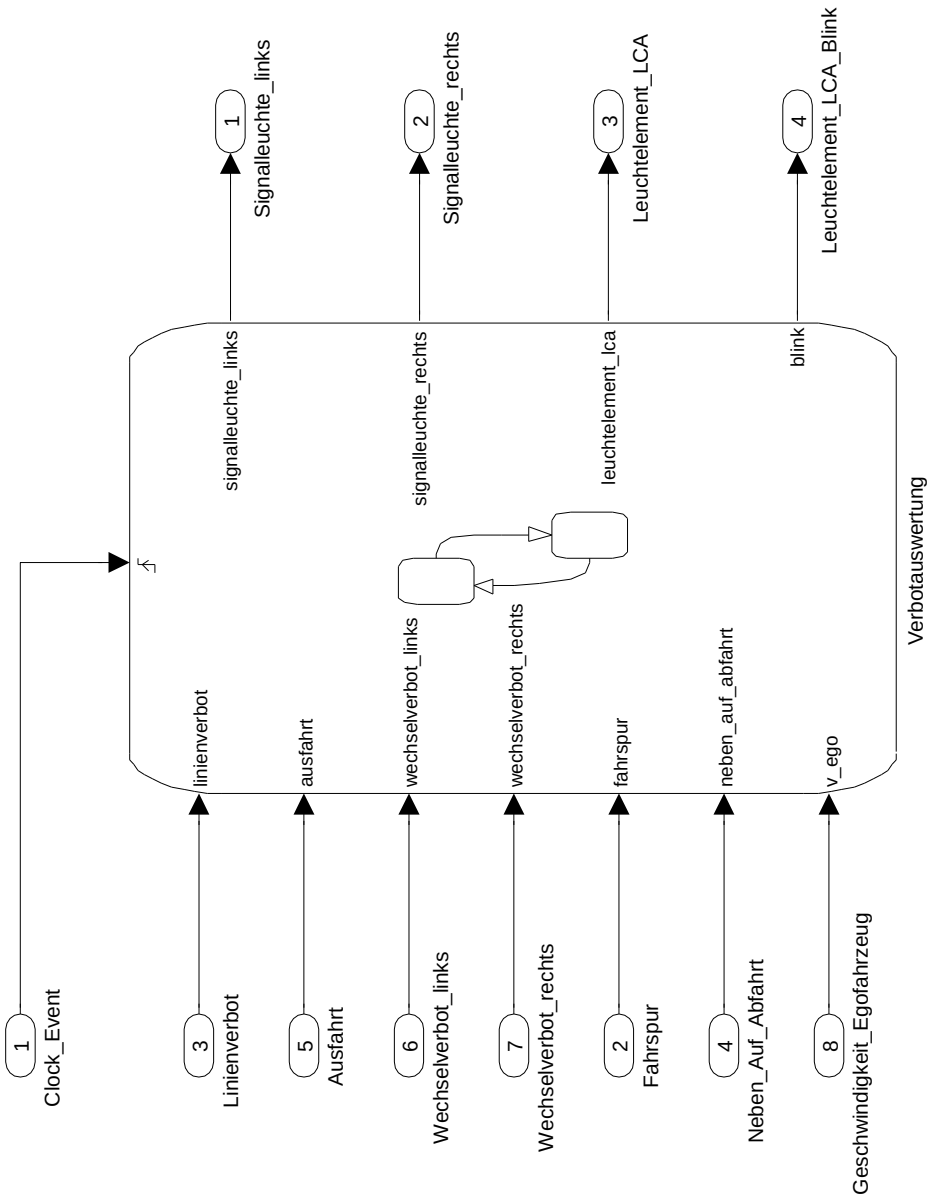


Abbildung A.35: Submodul Auswertung_Verbote

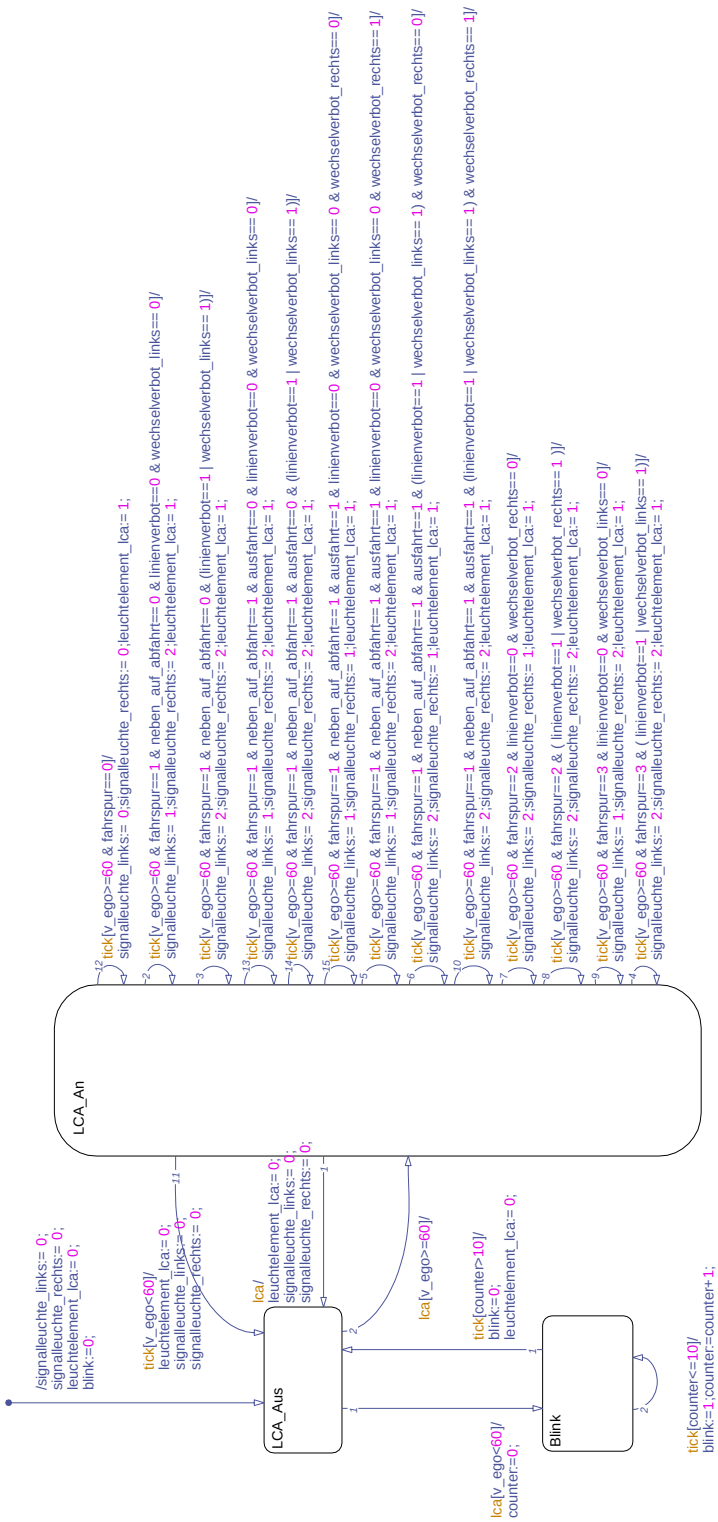


Abbildung A.36: Automat Verbotauswertung

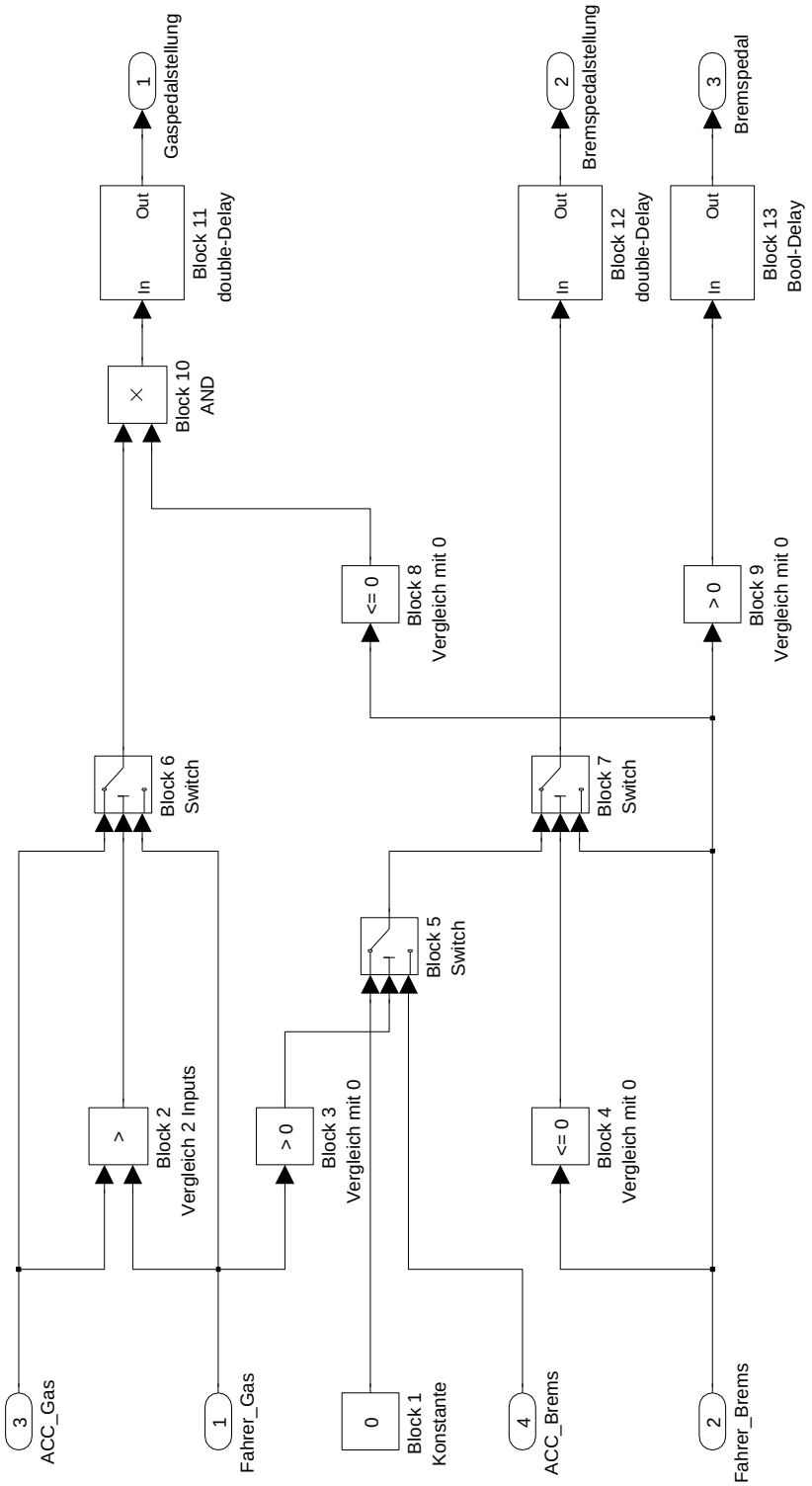


Abbildung A.37: Übersicht Multiplexer

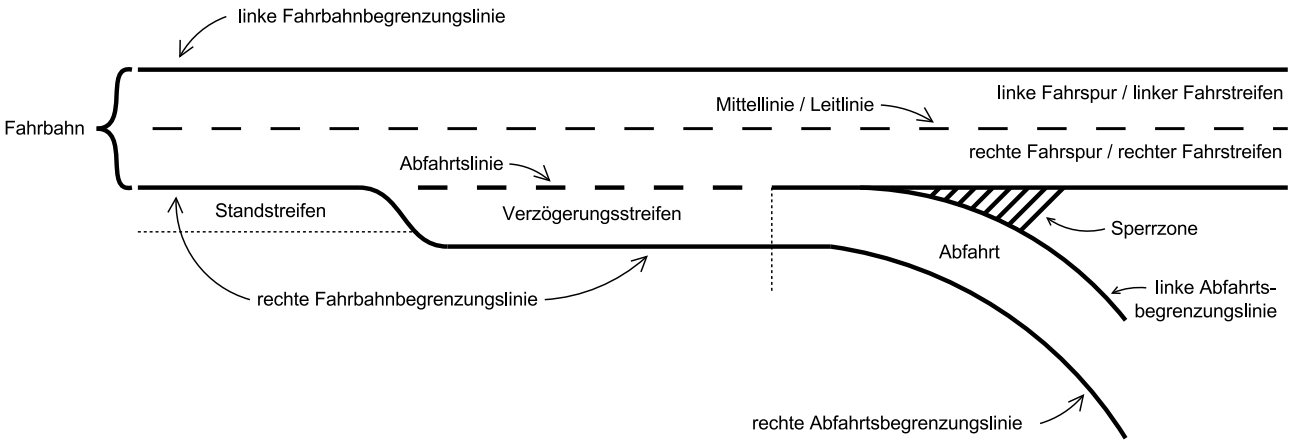


Abbildung A.38: Begriffskonvention: Autobahn mit Abfahrt

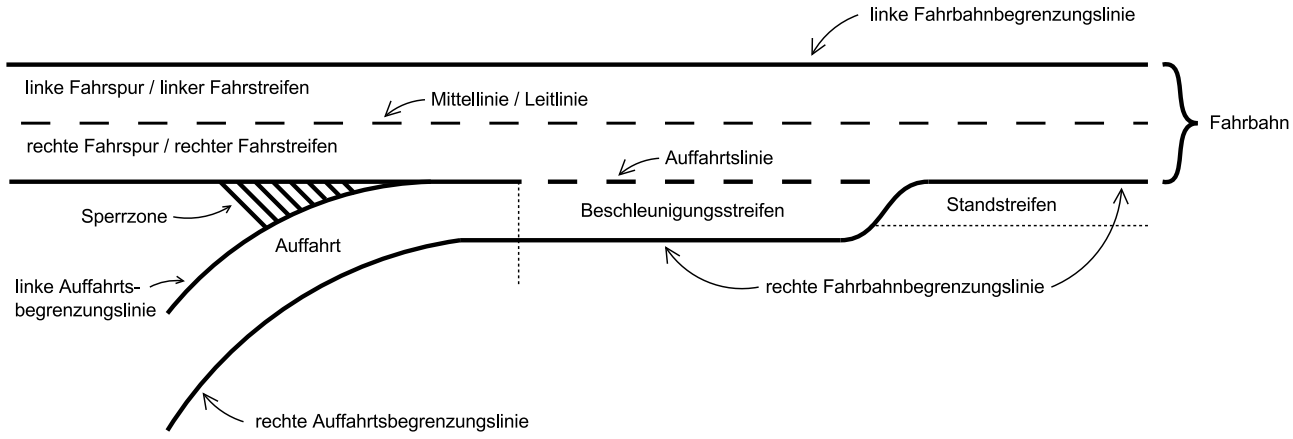


Abbildung A.39: Begriffskonvention: Autobahn mit Auffahrt

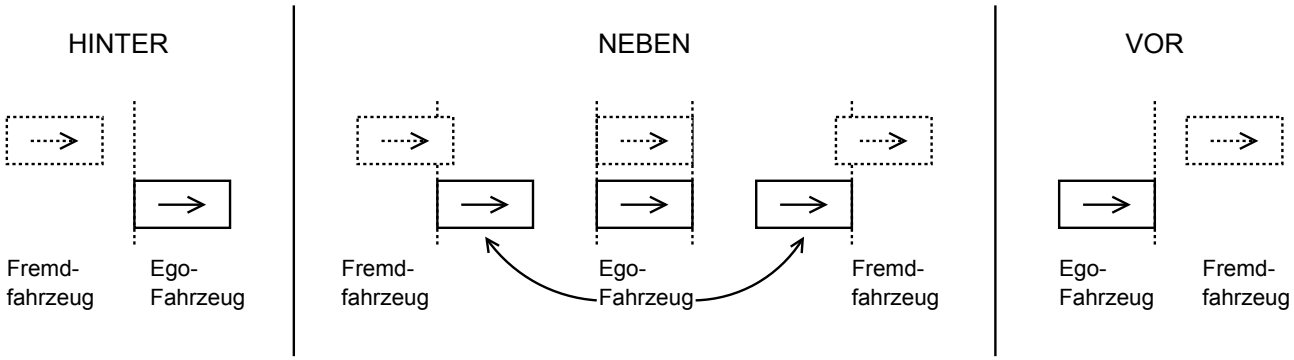


Abbildung A.40: Begriffskonvention: hinter, neben, vor

A.2 Projektplan-Export

GanttProject Report

Project : ViDAs

Start : 12/9/09

End : 12/10/10

Organization :

Web Link : http://vidas.informatik.uni-oldenburg.de

Description :

Der Auftrag:

Im Zeitraum vom Wintersemester 2009/2010 bis Wintersemester 2010/2011 ist ein Fahrerassistenzsystem zu entwickeln. Zum Einen ist die Funktionalität eines "Lane Change Assistant" (LCA), das den Fahrer beim Auffahren auf die Autobahn umzusetzen. Zum Anderen die Funktionalität eines "Adaptive Cruise Control" (ACC), das den Fahrer beim Halten einer eingestellten Wunschgeschwindigkeit unter Berücksichtigung des Sicherheitsabstandes zu vorrausfahrenden Fremdfahrzeugen, assistieren soll. Das ACC soll geltende Geschwindigkeitsbegrenzungen berücksichtigen und darf diese nicht überschreiten.

Die Umsetzung:

Das Fahrerassistenzsystem soll in einer virtuellen Umgebung eingesetzt werden, die durch den Fahrsimulator SILAB simuliert wird.

Date : Dec 8, 2010 3:01:44 PM

ViDAs
12/9/09-12/10/10

Tasks List

Name	Begin date	End date	Resources
Anforderungsdefinition	12/9/09	3/5/10	
Anforderungsdefinition 0.1	12/9/09	1/23/10	Dirk Wacker Sven Sieverding Jens Heyen Thilo Koepke Jianyu Bao Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche
Sensorrecherche	12/9/09	12/19/09	Dirk Wacker Alex Enkelmann
Streckeninformationen sammeln	12/9/09	12/19/09	Sven Sieverding Jens Heyen
PKW Eigenschaften	12/9/09	1/9/10	Andreas Gabel
Testfaelle bestimmen	12/9/09	12/19/09	Thilo Koepke Peter Battram
Systementwurf	12/9/09	12/19/09	Jianyu Bao Christoph Laesche
Umwelteigenschaften	12/21/09	1/9/10	Sven Sieverding Jens Heyen
Sensoranforderungen	12/21/09	1/9/10	Dirk Wacker Alex Enkelmann
Testszenarien und Anwendungsfaelle	12/21/09	1/9/10	Thilo Koepke Peter Battram
Nicht-/Funktionale Anforderungen	12/21/09	1/9/10	
Anwendungsfaelle ausfuehren	1/11/10	1/16/10	
Nicht-/funktionale Anfordrungen erweitern	1/11/10	1/23/10	Dirk Wacker Jianyu Bao Alex Enkelmann
Fehlerfaelle bestimmen	1/18/10	1/23/10	Thilo Koepke Peter Battram
Einleitung schreiben	1/11/10	1/23/10	Christoph Laesche
Entwicklungsprozess	1/11/10	1/23/10	Sven Sieverding Jens Heyen
Anforderungsdefinition V0.1	1/25/10	2/4/10	
Korrektur Anfdef 0.1	1/27/10	2/9/10	
Anwendungsfaelle ueberarbeiten	1/27/10	2/9/10	Dirk Wacker Jens Heyen Thilo Koepke Alex Enkelmann
Testfaelle ueberarbeiten	1/27/10	2/9/10	Sven Sieverding Jianyu Bao Andreas Gabel Peter Battram Christoph Laesche
Anforderungsdefinition V0.5	2/9/10	2/10/10	
Korrektur Anfdef 0.5	2/10/10	3/5/10	Dirk Wacker Sven Sieverding Jens Heyen Thilo Koepke Jianyu Bao

Page 2

ViDAs
12/9/09-12/10/10

			Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche
Anforderungsdefinition V1.0	3/5/10	3/6/10	
Planung des Review 1	3/8/10	3/26/10	
Review 1	3/26/10	3/27/10	
Entwurf	3/8/10	4/21/10	Dirk Wacker Sven Sieverding Jens Heyen Thilo Koepke Jianyu Bao Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche
Schnittstellen definieren	3/8/10	4/21/10	Dirk Wacker Sven Sieverding Jens Heyen Thilo Koepke Jianyu Bao Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche
Grobentwurf	3/8/10	3/17/10	Sven Sieverding Jens Heyen Thilo Koepke Jianyu Bao Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche
Feinentwurf	3/17/10	4/21/10	Sven Sieverding Jens Heyen Thilo Koepke Jianyu Bao Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche
Architektur- und Schnittstellendokument V0.1	4/21/10	4/22/10	Sven Sieverding Jens Heyen Thilo Koepke Jianyu Bao Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche
Implementierung	4/23/10	7/14/10	
ViDAs-Kern	4/23/10	7/14/10	
ACC-Submodul	4/23/10	5/26/10	Sven Sieverding Jianyu Bao
LCA-Submodul	4/23/10	5/26/10	Thilo Koepke Peter Battram
ACC + LCA implementiert	5/26/10	5/27/10	
Reglerentwurf	4/23/10	7/14/10	Peter Battram Alex Enkelmann

Page 3

ViDAs
12/9/09-12/10/10

Kern-Komm-Modul	5/26/10	6/30/10	Sven Sieverding
Komm-Modul implementiert	6/30/10	7/1/10	
Komm_Modultest	6/30/10	7/14/10	Sven Sieverding Christoph Laesche
Integration ACC+LCA	5/26/10	6/2/10	Sven Sieverding Peter Battram
Vorbereitung Testumgebung	7/7/10	7/14/10	Jianyu Bao
ViDAs-Umweltschnittstelle	4/23/10	7/14/10	
Umweltschnittstelle Komm	4/23/10	6/30/10	Christoph Laesche
Sensorik/Aktorik	4/23/10	6/9/10	Andreas Gabel
Sensorik, Aktorik implementiert	6/9/10	6/10/10	
Streckenbau	6/9/10	7/14/10	Andreas Gabel
Automatisiertes Fahren	5/26/10	6/23/10	Thilo Koepke Jianyu Bao
Automatisierte Fahrdatenauswertung	6/23/10	7/7/10	Jianyu Bao
Automatisiertes Fahren funktioniert	7/7/10	7/8/10	
Machbarkeitsstudie	4/23/10	11/30/10	
Einarbeitung	4/23/10	5/15/10	Jens Heyen
Designflow	5/19/10	5/26/10	Jens Heyen
Planung	5/26/10	6/2/10	Jens Heyen
Hardware-Anforderungen	5/17/10	6/2/10	Jens Heyen
Hardwareanforderungen festgelegt	6/2/10	6/3/10	
Code-Generierung	6/2/10	7/28/10	Jens Heyen
Code-Generierung funktioniert	7/28/10	7/29/10	
Laufzeitmessung	7/28/10	10/20/10	Jens Heyen
Konzept CAN-Kommunikation	9/29/10	10/20/10	Sven Sieverding Christoph Laesche
Dokument vollenden	10/20/10	11/30/10	Jens Heyen
SILAB einrichten & einarbeiten	1/18/10	4/23/10	Andreas Gabel
Testdatenbank	2/15/10	7/14/10	
Einarbeiten und Einrichten	2/15/10	4/23/10	Thilo Koepke
Testphase Vorbereitungen	6/23/10	7/14/10	Thilo Koepke
Erster Prototyp	7/14/10	7/15/10	
Vorbereitung Review 2	7/26/10	8/26/10	
Vortragsfolien: Neue Aufgabenstellung	7/26/10	7/31/10	Sven Sieverding Andreas Gabel
Vortragsfolien: ViDAs-Kern	8/2/10	8/11/10	Peter Battram
Vortragsfolien: Komm-Modul	8/2/10	8/11/10	Sven Sieverding Christoph Laesche
Vortragsfolien: Testen	8/2/10	8/11/10	Thilo Koepke
Vortragsfolien: Umweltschnittstelle	8/2/10	8/11/10	Andreas Gabel Christoph Laesche
Vortragsfolien: Ausblick	8/2/10	8/11/10	Sven Sieverding Alex Enkelmann
Vortrag ueben	8/11/10	8/26/10	Thilo Koepke Andreas Gabel Peter Battram Alex Enkelmann
Review 2	8/26/10	8/27/10	
Testphase	7/14/10	11/3/10	
LCA-Testphase	7/15/10	8/26/10	Sven Sieverding Thilo Koepke Jianyu Bao Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche
Integrationstests durchführen	7/14/10	7/31/10	Thilo Koepke

Page 4

			Peter Battram Alex Enkelmann Christoph Laesche
Gesamtintegrationstests	8/11/10	8/26/10	Sven Sieverding Christoph Laesche
ACC-Testphase	8/26/10	9/29/10	Sven Sieverding Thilo Koepke Jianyu Bao Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche
ACC-Tests abgeschlossen	9/29/10	9/30/10	
System-Testphase	9/29/10	11/3/10	Sven Sieverding Thilo Koepke Jianyu Bao Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche
Tests abgeschlossen	11/3/10	11/4/10	
Endbericht	11/1/10	11/30/10	Sven Sieverding Thilo Koepke Jianyu Bao Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche
Grobstruktur anlegen	11/1/10	11/6/10	
Endbericht schreiben	11/8/10	11/24/10	
Endbericht korrigieren	11/24/10	11/30/10	
Endvortrag vorbereiten	11/24/10	12/1/10	Sven Sieverding Jens Heyen Thilo Koepke Jianyu Bao Peter Battram Christoph Laesche
ViDAs fertig	11/30/10	12/1/10	Sven Sieverding Jens Heyen Thilo Koepke Jianyu Bao Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche
Endvortrag üben	12/1/10	12/9/10	Sven Sieverding Jens Heyen Jianyu Bao Christoph Laesche
Endvortrag	12/9/10	12/10/10	Sven Sieverding Jens Heyen Thilo Koepke Jianyu Bao Andreas Gabel Peter Battram Alex Enkelmann Christoph Laesche

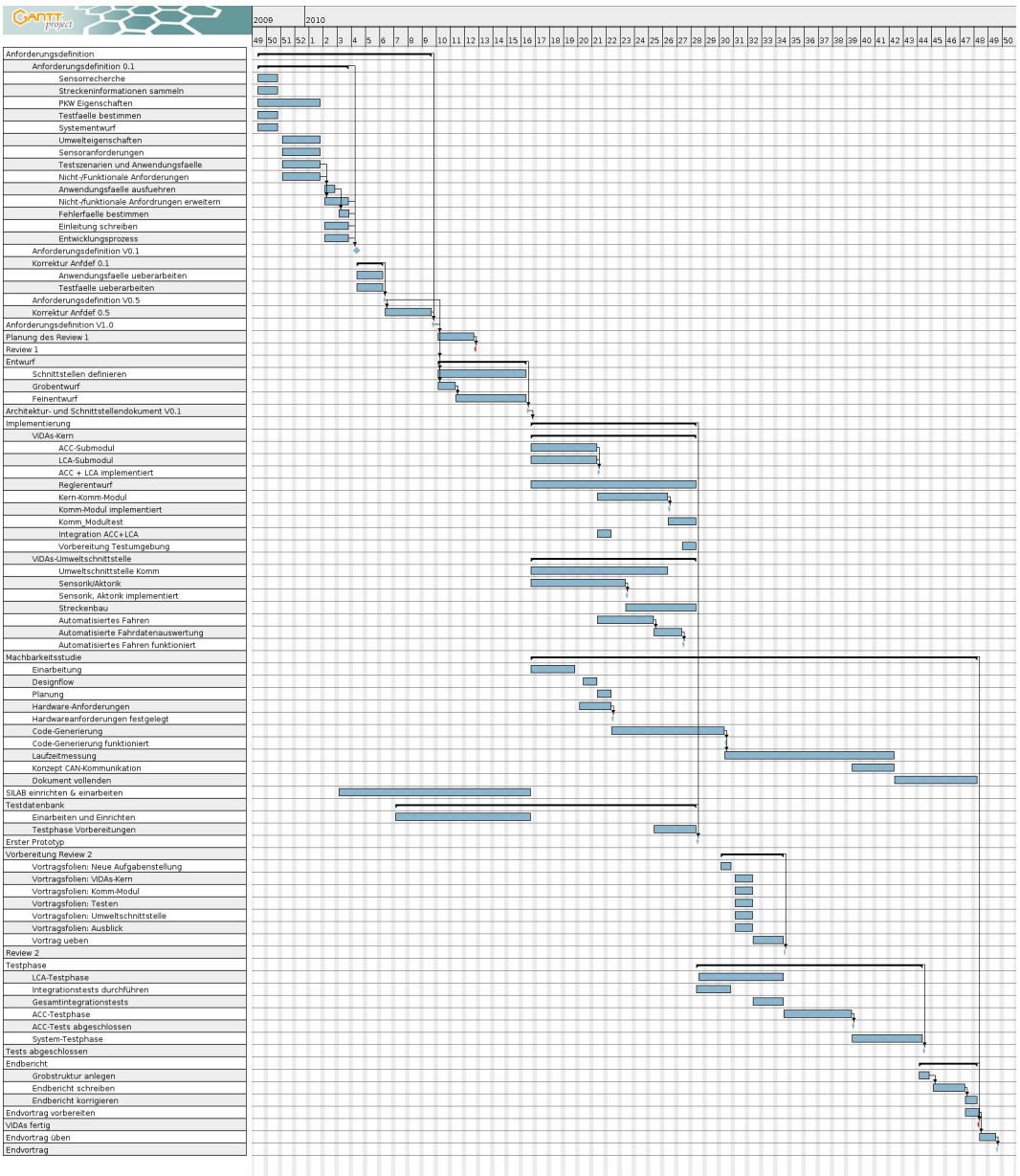
ViDAs
12/9/09-12/10/10

Resources List

Name	Default role
Sven Sieverding	Projektleiter
Christoph Laesche	Admin
Jens Heyen	Co-Manager
Alex Enkelmann	Dokumentenbeauftragter
Andreas Gabel	Silabbeauftragter
Thilo Koepke	Testbeauftragter
Peter Battram	Websitenbeauftragter
Jianyu Bao	Versionsmanagement
Dirk Wacker	Versionsmanagement

ViDAs
12/9/09-12/10/10

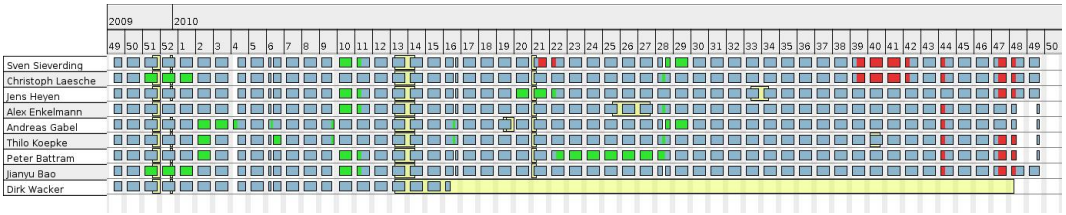
Gantt Chart



Page 7

ViDAs
12/9/09-12/10/10

Resources Chart



Page 8

A.3 Testplan-Export

GanttProject-Bericht

Projekt : ViDAs-Testplan

Anfang : 14.07.10

Ende : 04.11.10

Organisation : ViDAs

WWW-Link : http://hurrikan.informatik.uni-oldenburg.de:9890/test/index.php

Beschreibung :

Dieser Plan wurde angefertigt, um die verschiedenen Abschnitte der Testphase detailliert planen, darstellen und den einzelnen Vorgängen Ressourcen zuweisen zu können. Auf diese Weise wusste immer jeder Teilnehmer der Projektgruppe, was er zu tun hatte. Um eine Überlastung der Gruppenteilnehmer zu verhindern, wurde dieser Plan nur in Abstimmung mit dem ViDAs-Projektplan erstellt und erweitert. Aus diesem Grund können eventuell auftretende Lücken in der Ressourcenauslastung auch mit Hilfe des ViDAs-Porjektplans erklärt werden.

Date : 05.12.2010 18:00:45

ViDAs-Testplan
14.07.10-04.11.10

Vorgangsliste

Vorgang	Anfang	Ende	Ressource
Testphase	14.07.10	03.11.10	
LCA-Testphase	14.07.10	25.08.10	
Teststrecken bauen	14.07.10	28.07.10	Andreas Gabel
LCA-Testumgebung erstellen	14.07.10	21.07.10	Jianyu Bao Sven Sieverding
Testfälle erarbeiten	21.07.10	28.07.10	
LCA-Testfälle erarbeiten	21.07.10	28.07.10	Peter Battram Thilo Koepke
Komm-Testfälle erarbeiten	21.07.10	28.07.10	Christoph Lässche Sven Sieverding
Umweltschnittstelle-Testfälle erarbeiten	21.07.10	28.07.10	Andreas Gabel
Komm+Umweltschnittstelle-Testfälle erarbeiten	21.07.10	28.07.10	Christoph Lässche Sven Sieverding
LCA+Komm+Umweltschnittstelle-Testfälle erarbeiten	21.07.10	28.07.10	Andreas Gabel Christoph Lässche
LCA-Testumgebung erweitern	21.07.10	28.07.10	Peter Battram Jianyu Bao Alex Enkelmann
Testfälle und -umgebung fertig	28.07.10	29.07.10	Andreas Gabel Peter Battram Thilo Koepke Jianyu Bao Alex Enkelmann Christoph Lässche Sven Sieverding
Modultests	29.07.10	06.08.10	
LCA-Testfälle durchführen	29.07.10	04.08.10	Peter Battram Thilo Koepke Sven Sieverding
Komm-Testfälle durchführen	29.07.10	06.08.10	Jianyu Bao Alex Enkelmann Christoph Lässche
Umweltschnittstelle-Testfälle durchführen	29.07.10	04.08.10	Andreas Gabel Christoph Lässche
Modultests abgeschlossen	06.08.10	07.08.10	
Integrationstests	06.08.10	25.08.10	
Komm+Umweltschnittstelle-Testfälle durchführen	06.08.10	11.08.10	Andreas Gabel Christoph Lässche
LCA+Komm+Umweltschnittstelle-Testfälle durchführen	11.08.10	25.08.10	Jianyu Bao Christoph Lässche Sven Sieverding
LCA Testphase abgeschlossen	25.08.10	26.08.10	
ACC-Testphase	25.08.10	29.09.10	
ACC-Implementierung abschliessen	25.08.10	01.09.10	
Bekannte Fehler beheben	25.08.10	01.09.10	Andreas Gabel Peter Battram Sven Sieverding
Tasten programmierbar	25.08.10	01.09.10	Jianyu Bao Alex Enkelmann
ACC-Testumgebung erstellen	25.08.10	01.09.10	Peter Battram Christoph Lässche
Teststrecken bauen	01.09.10	15.09.10	Andreas Gabel

Page 2

ViDAs-Testplan
14.07.10-04.11.10

			Christoph Lässche
Testfälle erarbeiten	25.08.10	15.09.10	
ACC-Modultestfälle erarbeiten	25.08.10	01.09.10	Thilo Koepke Christoph Lässche Sven Sieverding
ACC & Umwelt-Testfälle erarbeiten	01.09.10	15.09.10	Peter Battram Thilo Koepke Jianyu Bao
ACC in Kern-Testfälle erarbeiten	01.09.10	15.09.10	Thilo Koepke Alex Enkelmann Sven Sieverding
Modultests	01.09.10	15.09.10	
ACC-Modultestfälle durchführen	01.09.10	15.09.10	Peter Battram Sven Sieverding
Integrationstests	15.09.10	29.09.10	
ACC & Umwelt-Testfälle durchführen	15.09.10	29.09.10	Jianyu Bao Christoph Lässche Sven Sieverding
ACC in Kern-Testfälle durchführen	15.09.10	22.09.10	Andreas Gabel Peter Battram Thilo Koepke Alex Enkelmann
Ausgaben verschönern	22.09.10	29.09.10	Andreas Gabel
Prädiktiver Regler	22.09.10	29.09.10	Peter Battram Alex Enkelmann
Test-Doku-Texte schreiben	22.09.10	29.09.10	Thilo Koepke
ACC-Testphase abgeschlossen	29.09.10	30.09.10	
ViDAs-Kern als DPU in SILAB einbetten	29.09.10	06.10.10	Christoph Lässche Sven Sieverding
DPU Integrationstests Strecken bauen	06.10.10	13.10.10	Christoph Lässche Sven Sieverding
System-Testphase	29.09.10	03.11.10	
Testszenarien erarbeiten	29.09.10	13.10.10	
Testszenarien erarbeiten 1 Szenarien 1 bis 4 aus der Anforderungsdefinition	29.09.10	13.10.10	Peter Battram Thilo Koepke
Testszenarien erarbeiten 2 Szenarien 5 bis 7 aus der Anforderungsdefinition	29.09.10	13.10.10	Andreas Gabel Jianyu Bao Alex Enkelmann
Testszenarien durchführen	13.10.10	03.11.10	
Kern & umwelt-Testszenarien durchführen	13.10.10	03.11.10	Andreas Gabel Peter Battram Thilo Koepke Jianyu Bao Alex Enkelmann Christoph Lässche Sven Sieverding
Testphase abgeschlossen	03.11.10	04.11.10	

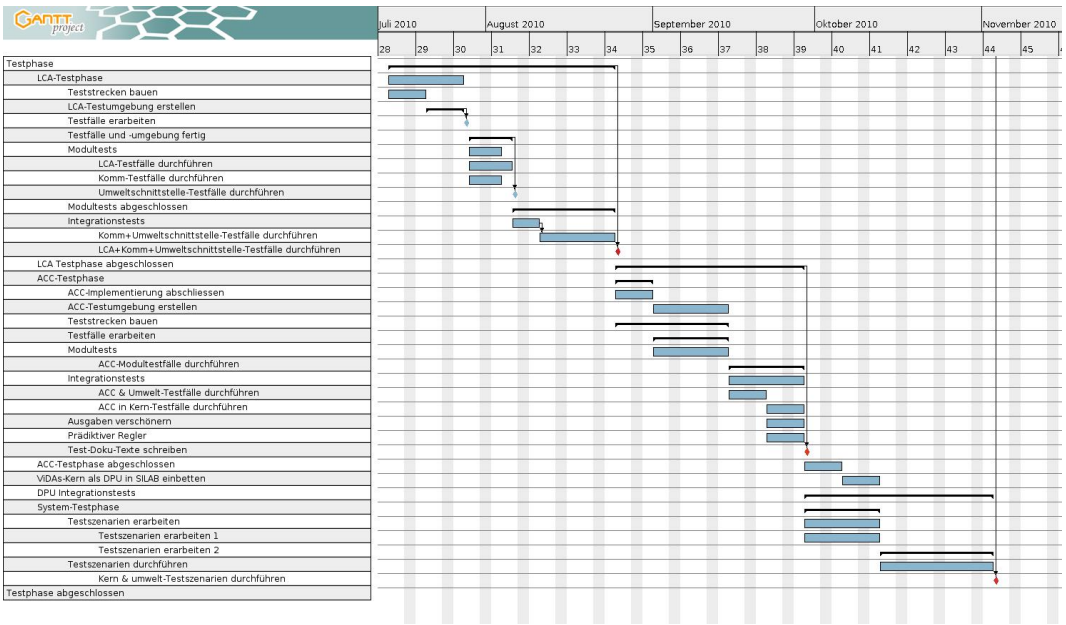
Page 3

Ressourcenliste

Ressource	Rolle
Thilo Koepke	Testbeauftragter
Peter Battram	Tester
Andreas Gabel	Tester
Alex Enkelmann	Tester
Sven Sieverding	Tester
Christoph Lässche	Tester
Jianyu Bao	Tester

ViDAs-Testplan
14.07.10-04.11.10

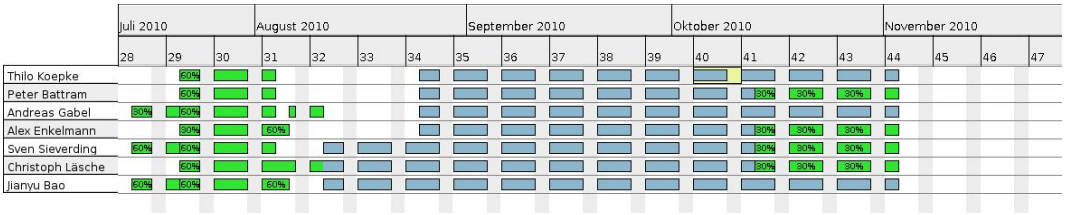
Gantt-Diagramm



Page 5

ViDAs-Testplan
14.07.10-04.11.10

Ressourcendiagramm



A.4 FMEA-Export

ViDAs — *Fehlermöglichkeits- und Einflußanalyse*

Lieferanten:	Abteilungen:	K-FMEA:	Teilename: /	erstellt durch: PG_ViDAs	Team:
		P-FMEA:	Teilenummer: /	geändert durch:	
		Erstellungsdatum: 5.5.2010	Modell/System/Typ:	Änderungsstand:	
		letzte Änderung: 08.12.10			

Nr.	Bau-gruppe / Schnitt-stelle / Pro-zess	Funktion	mögliche Fehler	mögliche Folgen	mögliche Ursachen	derzeitige Abstell-maßnahmen	Auftreten	Bedeutung	Entdeckung	RPZ	empfohlene Abstell-maßnahmen	Verantwort-lich/ Termin	durchgeführte Abstell-maßnahmen	Auftreten	Bedeutung	Entdeckung	RPZ
1	LCA	Beim Fahrspurwechsel assistieren	Leuchtelemente inaktiv	Fahrer erhält keine Unterstützung beim Fahrspurwechsel	Leuchtelemente kaputt, Kommunikationsfeh-ler	Keine	1	1	2	2	Ausführliche Tests der Kommunikations-komponente	Christoph, Sven, Thilo 06.08.2010	Ausführliche Tests der Kommunikation	1	1	2	2
2	LCA	Beim Fahrspurwechsel assistieren	LCA Komplettausfall	Fahrer erhält keine Unterstützung beim Fahrspurwechsel	Hardwaredefekt, Sensoren defekt, Berechnungsfehler	Keine	1	1	2	2	Ausführliche Tests des LCA-Moduls	Peter, Sven, Thilo 03.08.2010	Ausführliche Tests des LCA-Moduls	1	1	2	2
3	LCA	Beim Fahrspurwechsel assistieren	Falsche Anzeige der Spurwechselmög-lichkeit	Unerlaubter Fahrspurwechsel	Berechnungsfehler, Sensoren defekt	Keine	2	5	2	20	Ausführliche Tests des LCA-Moduls	Peter, Sven, Thilo 03.08.2010	Ausführliche Tests des LCA-Moduls	1	5	2	10
4	LCA	LCA ist aktivierbar ab v_lca,min	LCA kann nicht aktiviert werden bei v_ego >= v_lca,min	LCA inaktiv, Fahrer erhält keine Unterstützung beim Fahrspurwechsel	Bedienelement defekt, Geschwindigkeits-sensor defekt, Berechnungsfehler, Kommunikationsfeh-ler	Keine	4	6	1	24	Ausführliche Tests des LCA-Moduls, ausführliche Tests der Kommunikations-komponente	Christoph, Sven, Thilo, Peter 06.08.2010	Ausführliche Tests des LCA-Moduls, ausführliche Tests der Kommunikation	2	6	1	12
5	LCA	LCA ist aktivierbar ab v_lca,min	LCA bleibt aktiv bei v_ego < v_lca,min	LCA Fehlerhaft aktiv, kann falsche Informationen anzeigen	Geschwindigkeits-sensor defekt, Berechnungsfehler, Kommunikationsfeh-ler	Keine	3	4	4	48	Ausführliche Tests des LCA-Moduls, ausführliche Tests der Kommunikations-komponente	Christoph, Sven, Thilo, Peter 06.08.2010	Ausführliche Tests des LCA-Moduls, ausführliche Tests der Kommunikation	2	4	4	32
6	LCA	LCA ist aktivierbar ab v_lca,min	LCA Aktivierung wird nicht angezeigt	Fehlerhafte Informationen über LCA Status werden angezeigt	Kontrolleuchtele-ment defekt, Berechnungsfehler, Kommunikationsfeh-ler	Keine	3	4	3	36	Ausführliche Tests des LCA-Moduls, ausführliche Tests der Kommunikations-komponente	Peter, Sven, Thilo 03.08.2010	Ausführliche Tests des LCA-Moduls, ausführliche Tests der Kommunikation	2	4	3	24

Seite 1 von 12

	LCA	LCA deaktivieren	LCA bleibt aktiv	LCA ist fehlerhaft aktiv	Bedienelement defekt, Berechnungsfehler, Kommunikationsfeh-ler	Keine					Ausführliche Tests des LCA-Moduls, ausführliche Tests der Kommunikations-komponente	Christoph, Sven, Thilo, Peter 06.08.2010	Ausführliche Tests des LCA-Moduls, ausführliche Tests der Kommunikation				
							3	4	3	36				2	4	3	24
7																	
	ACC	Sicherheitsab-stand einhalten	zu geringer Sicherheitsabstand	Zu dichtes Auffahren, Auffahrunfall	Fehlerhafte Schilderkennung, fehlerhafte Abstandsmessung, fehlerhafte Abstandsberech-nung	Keine	6	9	3	162	Ausführliche Tests des Abstandsreglers	Peter, Thilo 15.09.2010	Ausführliche Tests des Abstandreglers	2	9	3	54
8																	
	ACC	Sicherheitsab-stand einhalten	zu großer Sicherheitsabstand	Unerwünschtes Abbremsen des Fahrzeugs	Fehlerhafte Schilderkennung, fehlerhafte Abstandsmessung, fehlerhafte Abstandsberech-nung	Keine	6	1	3	18	Ausführliche Tests des Abstandreglers	Peter, Thilo 15.09.2010	Ausführliche Tests des Abstandreglers	2	1	3	6
9																	
	ACC	Erkannte Geschwindigkeits-begrenzungen berücksichtigen	Geschwindigkeits-begrenzung nicht eingehalten	Zu schnelles Fahren	Fehlerhafte Schilderkennung, fehlerhafte Abstandsmessung zum Schild, fehlerhafte Berechnung der Bremskraft	Keine	6	10	4	240	Ausführliche Tests des Geschwindigkeits-reglers	Peter, Thilo 15.09.2010	Ausführliche Tests des Geschwindigkeits-reglers	2	10	4	80
10																	
	ACC	Das ACC funktioniert von v_acc,min bis v_ego,max, sonst schaltet sich das ACC aus und alarmiert den Fahrer	ACC kann nicht aktiviert werden bei v_acc,min <= v_ego <= v_ego,max	ACC inaktiv, Fahrer erhält keine Unterstützung von ACC	Ein oder mehrere Bedienelemente defekt, Geschwindigkeits-sensor defekt, Berechnungsfehler, Kommunikationsfeh-ler	Keine	4	6	2	48	Ausführliche Tests des ACC-Moduls	Peter, Thilo 15.09.2010	Ausführliche Tests des ACC-Moduls, ausführliche Tests der Kommunikation	2	6	2	24
11																	
	ACC	Das ACC funktioniert von v_acc,min bis v_ego,max, sonst schaltet sich das ACC aus und alarmiert den Fahrer	ACC bleibt aktiv bei v_ego < v_acc,min	ACC ist fehlerhaft aktiv, falsch assistiert, Auffahrunfall	Geschwindigkeits-sensor defekt, Berechnungsfehler, Kommunikationsfeh-ler	Keine	3	8	3	72	Ausführliche Tests des ACC-Moduls	Peter, Thilo 15.09.2010	Ausführliche Tests des ACC-Moduls, ausführliche Tests der Kommunikation	2	8	3	48
12																	

13	ACC	Das ACC funktioniert von v_acc,min bis v_ego,max, sonst schaltet sich das ACC aus und alarmiert den Fahrer	ACC bleibt aktiv bei v_ego > v_ego,max	ACC ist fehlerhaft aktiv, falsch assistiert, Auffahrunfall	Geschwindigkeits-sensor defekt, Berechnungsfehler, Kommunikationsfeh-ler	Keine	3	10	3	90	Ausführliche Tests des ACC-Moduls	Peter, Thilo 15.09.2010	Ausführliche Tests des ACC-Moduls, ausführliche Tests der Kommunikation	2	10	3	60
14	ACC	Das ACC funktioniert von v_acc,min bis v_ego,max, sonst schaltet sich das ACC aus und alarmiert den Fahrer	ACC schaltet sich bei v_ego < v_acc,min oder v_ego > v_ego,max aus, alarmiert den Fahrer darüber nicht	Der Fahrer merkt nicht, dass ACC sich ausgeschaltet hat, Auffahrunfall	Alarmelement defekt, Berechnungsfehler, Kommunikationsfeh-ler	Keine	2	10	2	40	Ausführliche Tests des ACC-Moduls	Peter, Thilo 15.09.2010	Ausführliche Tests des ACC-Moduls, ausführliche Tests der Kommunikation	1	10	2	20
15	ACC	ACC deaktivieren	ACC bleibt aktiv	ACC ist fehlerhaft aktiv, Auffahrunfall	Ein oder mehrere Bedienelemente defekt, Bremspedal defekt, Berechnungsfehler, Kommunikationsfeh-ler	Keine	3	10	2	60	Ausführliche Tests des ACC-Moduls	Peter, Thilo 15.09.2010	Ausführliche Tests des ACC-Moduls, ausführliche Tests der Kommunikation	2	10	2	40
16	ACC	Bei Beschleunigung bleibt ACC aktiv	ACC schaltet sich aus	ACC ist fehlerhaft inaktiv, keine Unterstützung für Fahrer	Berechnungsfehler, Kommunikationsfeh-ler	Keine	1	4	3	12	Ausführliche Tests des ACC-Moduls	Peter, Thilo 15.09.2010	Ausführliche Tests des ACC-Moduls, ausführliche Tests der Kommunikation	1	4	3	12
17	ACC	Beim stärkeren Bremsen als Fahrkomfort schaltet sich das ACC aus und alarmiert den Fahrer	ACC bleibt aktiv	ACC ist fehlerhaft aktiv, Auffahrunfall	Bremspedal defekt, Berechnungsfehler, Kommunikationsfeh-ler	Keine	3	10	3	90	Ausführliche Tests des ACC-Moduls	Peter, Thilo 15.09.2010	Ausführliche Tests des ACC-Moduls, ausführliche Tests der Kommunikation	2	10	3	60
18	ACC	Beim stärkeren Bremsen als Fahrkomfort schaltet sich das ACC aus und alarmiert den Fahrer	ACC schaltet sich aus, alarmiert den Fahrer darüber nicht	Der Fahrer merkt nicht, dass ACC sich ausgeschaltet hat, Auffahrunfall	Alarmelement defekt, Berechnungsfehler, Kommunikationsfeh-ler	Keine	2	10	2	40	Ausführliche Tests des ACC-Moduls	Peter, Thilo 15.09.2010	Ausführliche Tests des ACC-Moduls, ausführliche Tests der Kommunikation	1	10	2	20

Seite 3 von 12

19	ACC	aktuelle Geschwindigkeitsbegrenzungen anzeigen	aktuelle Geschwindigleitsbegrenzungen werden nicht angezeigt	Fahrer sieht nicht auf Display aktuelle Geschwindigkeitsbegrenzungen	Display defekt, Berechnungsfehler, Kommunikationsfehler	Keine	3	3	2	18	Ausführliche Tests des ACC-Moduls	Peter, Thilo 15.09.2010	Ausführliche Tests des ACC-Moduls, ausführliche Tests der Kommunikation	2	3	2	12
20	ACC	aktuelle Geschwindigkeitsbegrenzungen anzeigen	aktuelle Geschwindigleitsbegrenzungen werden falsch angezeigt	Fahrer erhält falsche Information, zu schnelles oder zu langsames Fahren	Display defekt, Berechnungsfehler, Kommunikationsfehler	Keine	3	5	3	45	Ausführliche Tests des ACC-Moduls	Peter, Thilo 15.09.2010	Ausführliche Tests des ACC-Moduls, ausführliche Tests der Kommunikation	2	5	3	30
21	Sensorik	Schilder erkennen	Geschwindigkeitsbegrenzung nicht erkannt	Geschwindigkeitsbeschränkung wird nicht eingehalten, Fahrer erhält falsche Informationen im Display angezeigt	Kameraausfall, Übertragungsfehler, Fehler in der Objekterkennung	Keine	7	7	4	196	Ausführliche Tests der Kommunikationskomponente, Übertragungsprotokoll überprüfen (z.B. Model-checking)	Christoph, Sven, Thilo 06.08.2010	Ausführliche Tests der Kommunikation	2	7	4	56
22	Sensorik	Schilder erkennen	Falsche Geschwindigkeitsbegrenzung erkannt	Unerwünschtes herabsetzen der Geschwindigkeit, Überschreitung der geltenden Geschwindigkeitsbegrenzung, Fahrer erhält falsche Informationen im Display angezeigt	Fehler in der Objekterkennung, Übertragungsfehler	Keine	6	7	4	168	Ausführliche Tests der Kommunikationskomponente, Übertragungsprotokoll überprüfen (z.B. Model-checking)	Christoph, Sven, Thilo 06.08.2010	Ausführliche Tests der Kommunikation	3	7	4	84
23	Sensorik	Schilder erkennen	Überholverbot nicht erkannt	Unzulässiges Anzeigen einer gültigen Fahrspurwechselmöglichkeit, Fahrer erhält falsche Informationen im Display angezeigt	Kameraausfall, Übertragungsfehler, Fehler in der Objekterkennung	Keine	6	7	4	168	Ausführliche Tests der Kommunikationskomponente, Übertragungsprotokoll überprüfen (z.B. Model-checking)	Christoph, Sven, Thilo 06.08.2010	Ausführliche Tests der Kommunikation	3	7	4	84
24	Sensorik	Schilder erkennen	Aufhebung eines Überholverbotes nicht erkannt	Fehlerhafte Anzeige eines Spurwechselverbotes, Fahrer erhält falsche Informationen im Display angezeigt	Kameraausfall, Übertragungsfehler, Fehler in der Objekterkennung	Keine	6	7	4	168	Ausführliche Tests der Kommunikationskomponente, Übertragungsprotokoll überprüfen (z.B. Model-checking)	Christoph, Sven, Thilo 06.08.2010	Ausführliche Tests der Kommunikation	3	7	4	84

25	Sensorik	Schilder erkennen	Aufhebung einer Geschwindigkeitsbegrenzung nicht erkannt	Fahrzeug wird nicht wieder auf die eingestellte Wunschgeschwindigkeit beschleunigt, Fahrer erhält falsche Informationen im Display angezeigt	Kameraausfall, Übertragungsfehler, Fehler in der Objekterkennung	Keine	6	7	4	168	Ausführliche Tests der Kommunikations-komponente, Übertragungsproto-koll überprüfen (z.B. Model-checking)	Christoph, Sven, Thilo 06.08.2010	Ausführliche Tests der Kommunikation	2	7	4	56
26	Sensorik	Schilder erkennen	Aufhebung aller Streckenverbote nicht erkannt	Fahrzeug wird nicht wieder auf die eingestellte Wunschgeschwin-digkeit beschleu-nigt, Fahrer erhält falsche Informa-tionen im Display angezeigt	Kameraausfall, Übertragungsfehler, Fehler in der Objekterkennung	Keine	6	7	4	168	Ausführliche Tests der Kommunikations-komponente, Übertragungsproto-koll überprüfen (z.B. Model-checking)	Christoph, Sven, Thilo 06.08.2010	Ausführliche Tests der Kommunikation	3	7	4	84
27	Sensorik	Vorrausfahrende Fahrzeuge erkennen	Komplettausfall	das vorausfahrende Fahrzeug wird nicht erkannt	Ausfall des Lidarsensors, Kommunikationsfeh-ler, Berechnungsfehler	Keine	3	10	1	30	Übertragungsproto-koll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikations-komponente	Christoph, Sven, Thilo 06.08.2010	Ausführliche Tests der Kommunikation	2	10	1	20
28	Sensorik	Vorrausfahrende Fahrzeuge erkennen	falsche Erkennung	das vorausfahrende Fahrzeug kann nicht richtig erkannt	Störung des Lidarsensors, Berechnungs- oder Übertragungsfehler	Keine	5	8	4	160	Übertragungsproto-koll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikations-komponente	Christoph, Sven, Thilo 06.08.2010	Ausführliche Tests der Kommunikation	3	8	4	96
29	Sensorik	Vorrausfahrende Fahrzeuge erkennen	schwache Erkennungsfunktion	kleiner Erkennungsbereich, zu später zu erkennen	Störung des Lidarsensors, Berechnungs- oder Übertragungsfehler	Keine	4	8	5	160	Übertragungsproto-koll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikations-komponente	Christoph, Sven, Thilo 06.08.2010	Ausführliche Tests der Kommunikation	2	8	5	80

30	Sensorik	Geschwindigkeit des vorausfahrenden Fahrzeug und den Geschwindigkeits-unterschied bestimmen	Komplettausfall	Geschwindigkeit und deren Unterschied gar nicht zu bestimmen	Ausfall des Lidar- oder Geschwindigkeits-sensors, Kommunikationsfeh-ler, Berechnungsfehler	Keine	3	10	1	30	Übertragungsproto-koll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikations-komponente, Ausführliche Tests des Geschwindigkeits-reglers	Christoph, Sven, Thilo, Peter 15.09.2010	Ausführliche Tests der Kommunikation, Ausführliche Tests des Geschwindigkeits-reglers	2	10	1	20
31	Sensorik	Geschwindigkeit des vorausfahrenden Fahrzeugs und den Geschwindigkeits-unterschied bestimmen	falsche Erkennung der Geschwindigkeit	Nichteinhalten des Sicherheitsabstan-des, Auffahrunfall	Störung des Lidarsensors, Berechnungs- oder Übertragungsfehler	Keine	5	8	4	160	Übertragungsproto-koll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikations-komponente, Ausführliche Tests des Geschwindigkeits-reglers	Christoph, Sven, Thilo, Peter 15.09.2010	Ausführliche Tests der Kommunikation, Ausführliche Tests des Geschwindigkeits-reglers	3	8	4	96
32	Sensorik	Geschwindigkeit des vorausfahrenden Fahrzeugs und den Geschwindigkeits-unterschied bestimmen	falsche Erkennung des Geschwindigkeitsunte rschieds	Nichteinhalten des Sicherheitsabstan-des, Auffahrunfall	Störung des Lidar- oder Geschwindigkeits-sensors, Berechnungs- oder Übertragungsfehler	Keine	5	8	4	160	Übertragungsproto-koll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikations-komponente, Ausführliche Tests des Geschwindigkeits-reglers	Christoph, Sven, Thilo, Peter 15.09.2010	Ausführliche Tests der Kommunikation, Ausführliche Tests des Geschwindigkeits-reglers	3	8	4	96
33	Sensorik	Geschwindigkeit des vorausfahrenden Fahrzeug und den Geschwindigkeits-unterschied bestimmen	schwache Erkennungsfunktion	kleiner Erkennungsbereich, zu späte Bestimmung	Störung des Lidar- oder Geschwindigkeits-sensors, Berechnungs- oder Übertragungsfehler	Keine	4	8	5	160	Übertragungsproto-koll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikations-komponente, Ausführliche Tests des Geschwindigkeits-reglers	Christoph, Sven, Thilo, Peter 15.09.2010	Ausführliche Tests der Kommunikation, Ausführliche Tests des Geschwindigkeits-reglers	3	8	4	96

34	Sensorik	Fahrzeug hinter dem Ego-Fahrzeug erkennen	Komplettausfall	Unfall beim Spurwechsel, Fahrer erhält falsche Informationen	Ausfall des Radarsensors oder des Lidarsensors, Berechnungsfehler, Kommunikationsfehler	Keine	3	10	1	30	Übertragungsprotokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikationskomponente, Berechnungskomponenten optimieren	Christoph, Sven, Thilo, Peter 06.08.2010	Übertragungsprotokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikation, Ausführliche Tests des LCA-Moduls	2	10	1	20
35	Sensorik	Fahrzeug hinter dem Ego-Fahrzeug erkennen	falsche oder nicht rechtzeitige Erkennung	Unfall beim Spurwechsel, Fahrer erhält falsche Informationen	Störung des Radar- oder Lidarsensors, Berechnungs- oder Übertragungsfehler	Keine	5	5	4	100	Übertragungsprotokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikationskomponente, Berechnungskomponenten optimieren	Christoph, Sven, Thilo, Peter 06.08.2010	Übertragungsprotokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikation, Ausführliche Tests des LCA-Moduls	3	5	4	60
36	Sensorik	Fahrzeug neben dem Ego-fahrzeug erkennen	Komplettausfall	Unfall beim Spurwechsel, Fahrer erhält falsche Informationen	Ausfall des Radarsensors, Kommunikationsfehler, Berechnungsfehler	Keine	3	10	1	30	Übertragungsprotokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikationskomponente, Berechnungskomponenten optimieren	Christoph, Sven, Thilo, Peter 06.08.2010	Übertragungsprotokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikation, Ausführliche Tests des LCA-Moduls	2	10	1	20
37	Sensorik	Fahrzeug neben dem Ego-fahrzeug erkennen	falsche Erkennung	Unfall beim Spurwechsel, Fahrer erhält falsche Informationen	Störung des Radarsensors, Berechnungs- oder Übertragungsfehler	Keine	5	5	4	100	Übertragungsprotokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikationskomponente, Berechnungskomponenten optimieren	Christoph, Sven, Thilo, Peter 06.08.2010	Übertragungsprotokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikation, Ausführliche Tests des LCA-Moduls	3	5	4	60

38	Sensorik	Fahrzeug neben dem Ego-fahrzeug erkennen	schwache Erkennungsfunktion	kleiner Erkennungsbereich, zu späte Bestimmung	Störung des Radarsensors, Berechnungs- oder Übertragungsfehler	Keine	4	5	5	100	Übertragungsproto-koll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikations-komponente, Berechnungskom-ponenten optimieren	Christoph, Sven, Thilo, Peter 06.08.2010	Übertragungspro-tokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikation, Ausführliche Tests des LCA-Moduls	2	5	5	50
39	Sensorik	Geschwindigkeit des hinteren Fahrzeugs und den Geschwindigkeits-unterschied bestimmen	Komplettausfall	Geschwindigkeit und Unterschied können nicht bestimmt werden	Ausfall des Radarsensors, Ausfall des Geschwindigkeits-sensors, Berechnugsfehler, Übertragungsfehler	Keine	3	10	1	30	Übertragungsproto-koll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikations-komponente, Berechnungskom-ponenten optimieren	Christoph, Sven, Thilo, Peter 06.08.2010	Übertragungspro-tokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikation, Ausführliche Tests des LCA-Moduls	2	10	1	20
40	Sensorik	Geschwindigkeit des hinteren Fahrzeugs und den Geschwindigkeits-unterschied bestimmen	falsche Erkennung der Geschwindigkeit des hinteren Fahrzeugs	Geschwindigkeit und Unterschied können nicht bestimmt werden	Störung des Radar- oder Lidarsensors, Berechnungs- oder Übertragungsfehler	Keine	5	8	4	160	Übertragungsproto-koll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikations-komponente, Berechnungskom-ponenten optimieren	Christoph, Sven, Thilo, Peter 06.08.2010	Übertragungspro-tokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikation, Ausführliche Tests des LCA-Moduls	3	8	4	96
41	Sensorik	Geschwindigkeit des hinteren Fahrzeugs und den Geschwindigkeits-unterschied bestimmen	falsche Erkennung des Geschwindig-keitsunterschieds des hinteren Fahrzeugs	Falsche Bestimmung des Geschwindigkeitsun-terschieds	Störung des Radar-, Geschwindigkeits- oder Lidarsensors, Berechnungs- oder Übertragungsfehler	Keine	5	8	4	160	Übertragungsproto-koll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikations-komponente, Berechnungskom-ponenten optimieren	Christoph, Sven, Thilo, Peter 06.08.2010	Übertragungspro-tokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikation, Ausführliche Tests des LCA-Moduls	3	8	4	96

42	Sensorik	Geschwindigkeit des hinteren Fahrzeugs und den Geschwindigkeits-unterschied bestimmen	schwache Erkennungsfunktion	kleiner Erkennungsbereich, zu späte Bestimmung	Störung des Radar-, Geschwindigkeits- oder Lidarsensors, Berechnungs- oder Übertragungsfehler	Keine	4	8	5	160	Übertragungsproto-koll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikations-komponente, Berechnungskom-ponenten optimieren	Christoph, Sven, Thilo, Peter 06.08.2010	Übertragungspro-tokoll überprüfen (z.B. Model-checking), Ausführliche Tests der Kommunikation, Ausführliche Tests des LCA-Moduls	2	8	5	80
43	Aktorik	Bremsen	Komplettausfall	Auffahrunfall, Sach- und Personenschäden	Bremsleitung mechanisch defekt	Keine	2	10	1	20	Keine						0
44	Aktorik	Bremsen	Verminderte Leistung	Auffahrunfall, Sach- und Personenschäden	Abgenutzte Bremsbeläge, Bremskraftverstärker defekt, Loch in der Bremsleitung, verstopfte Bremsleitung	Keine	5	8	2	80	Keine						0
45	Aktorik	Bremsen	Bremsen lösen sich nicht mehr	Anfahren wird erschwert/ist nicht mehr möglich, Störung des Verkehrsflusses	Überhitzte Bremsbeläge	Keine	3	8	2	48	Keine						0
46	Aktorik	Beschleunigen	Komplettausfall	Anfahren nicht mehr möglich, Störung des Verkehrsflusses	Tank leer, Motor abgewürgt/ausge-gangen, Benzinschlauch gerissen	Keine	2	10	1	20	Häufiger zur Tankstelle fahren, Reservekanister füllen						0
47	Aktorik	Beschleunigen	Verminderte Leistung	Anfahren wird erschwert, Störung des Verkehrsflusses	Benzinschlauch verstopft/löchrig, Kraftstofffilter verstopft, Zündkerze/-kabel defekt	Keine	4	8	2	64	Ersatzzündkerzen im Kofferraum						0
48	Aktorik	Beschleunigen	permanentes Beschleunigen	Auffahrunfall, Sach- und Personenschäden	Drosselklappe verklemmt	Keine	5	8	2	80	Keine						0

Seite 9 von 12

Bewertung Automobil

Produkt-FMEA							
Bedeutung		**Auftretens-wahrscheinlichkeit**	**Fehleranteil in ppm**		**Entdeckungs-wahrscheinlichkeit**	**Sicherheit Nachweis-verfahren**	
sehr hoch		**sehr hoch**			**sehr gering**		
Sicherheitsrisiko, Nichterfüllung gesetzlicher Vorschriften, Liegenbleiber	10 / 9	sehr häufiges Auftreten der Fehlerursache, unbrauchbares, ungeeignetes Konstruktionskonzept	500.000 / 100.000	10 / 9	Entdecken der aufgetretenen Fehlerursache ist unwahrscheinlich, Zuverlässigkeit der Konstruktionsauslegung wurde nicht oder kann nicht nachgewiesen werden, Nachweisverfahren sind gering	90%	10 / 9
hoch		**hoch**			**gering**		
Funktionsfähigkeit des Fahrzeuges stark eingeschränkt, sofortiger Werkstattaufenthalt zwingend erforderlich, Funktionseinschränkung wichtiger Teilsysteme	8 / 7	Fehlerursache tritt wiederholt auf, problematische , unausgereifete Konstruktion	50.000 / 10.000	8 / 7	Entdecken der aufgetretenen Fehlerursache weniger wahrscheinlich, Zuverlässigkeit der Konstruktionsauslegung kann wahrscheinlich nicht nachgewiesen werden, Nachweisverfahren sind unsicher	98%	8 / 7
mäßig		**mäßig**			**mäßig**		
Funktionsfähigkeit des Fahrzeuges eingeschränkt, sofortiger Werkstattaufenthalt nicht zwingend erforderlich, Funktionseinschränkung wichtigen Bedien- und Komfortsysteme	6 / 5 / 4	Gelegentliche auftretende Fehlerursache, geeignete, im Reifegrad fortgeschrittene Konstruktion	5.000 / 1.000 / 500	6 / 5 / 4	Entdecken der aufgetretenen Fehlerursache wahrscheinlich, Zuverlässigkeit der Konstruktionsauslegung könnte vielleichtnachgewiesen werden, Nachweisverfahren sind relativ sicher	99,70%	6 / 5 / 4
gering		**gering**			**hoch**		
geringe Funktionsfähigkeit des Fahrzeuges, Beseitigung beim nächsten Wertstattaufenthalt, Funktionseinschränkung von Bedien- und Komfortsysteme	3 / 2	Auftreten der Fehlerursache ist gering, bewährte konstruktive Auslegung	100 / 50	3 / 2	Entdecken der aufgetretenen Fehlerursache ist sehr wahrscheinlich, durch mehrere Nachweisverfahren bestätigt	99,90%	3 / 2
sehr gering		**sehr gering**			**sehr hoch**		
Sehr geringe Funktionsbeeinträchtigung, nur vom Fachpersonal erkennbar	1	Auftreten der Fehlerursache ist unwahrscheinlich	1	1	Aufgetretene Fehlerursache wird sicher entdeckt	99,99%	1

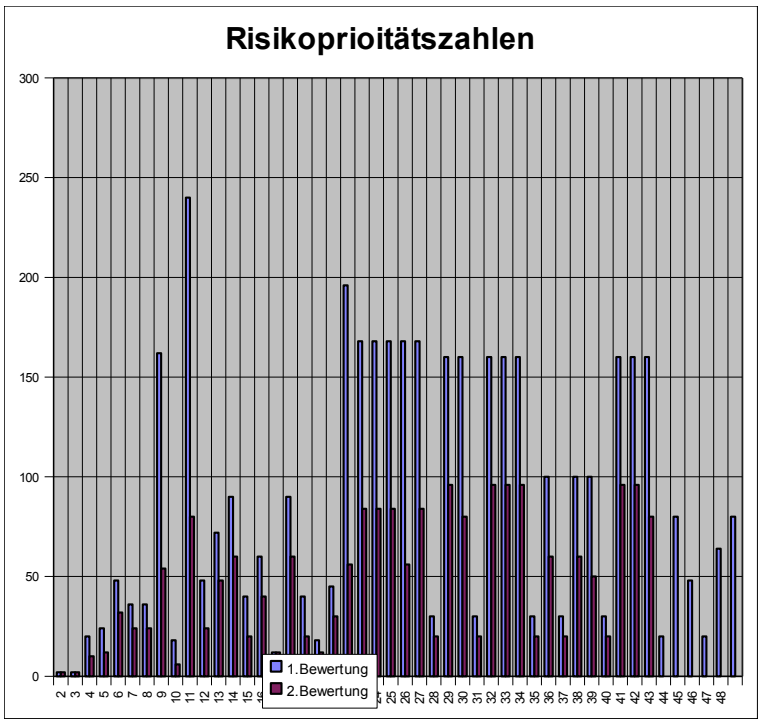
Seite 10

Bewertung Automobil

Prozess-FMEA							
Bedeutung		**Auftretens-wahrscheinlichkeit**	**Fehleranteil in ppm**		**Entdeckungs-wahrscheinlichkeit**	**Sicherheit Nachweis-verfahren**	
sehr hoch		**sehr hoch**			**sehr gering**		
Sicherheitsrisiko, Nichterfüllung gesetzlicher Vorschriften, Liegenbleiber	10 / 9	sehr häufiges Auftreten der Fehlerursache, unbrauchbarer, ungeeigneter Prozess	500.000 / 100.000	10 / 9	Entdecken der aufgetretenen Fehlerursache ist unwahrscheinlich, die Fehlerursache kann nicht geprüft werden	90%	10 / 9
hoch		**hoch**			**gering**		
Funktionsfähigkeit des Fahrzeuges stark eingeschränkt, sofortiger Werkstattaufenthalt zwingend erforderlich, Funktionseinschränkung wichtiger Teilsysteme	8 / 7	Fehlerursache tritt wiederholt auf, ungenauer Prozess	50.000 / 10.000	8 / 7	Entdecken der aufgetretenen Fehlerursache weniger wahrscheinlich, wahrscheinlich nicht zu entdeckende Fehlerursache, unsichere Prüfungen	98%	8 / 7
mäßig		**mäßig**			**mäßig**		
Funktionsfähigkeit des Fahrzeuges eingeschränkt, sofortiger Werkstattaufenthalt nicht zwingend erforderlich, Funktionseinschränkung wichtigen Bedien- und Komfortsysteme	6 / 5 / 4	Gelegentliche auftretende Fehlerursache, weiniger genauer Prozess	5.000 / 1.000 / 500	6 / 5 / 4	Entdecken der aufgetretenen Fehlerursache wahrscheinlich, Prüfungen sind relativ sicher	99,70%	6 / 5 / 4
gering		**gering**			**hoch**		
geringe Funktionsfähigkeit des Fahrzeuges, Beseitigung beim nächsten Wertstattaufenthalt, Funktionseinschränkung von Bedien- und Komfortsysteme	3 / 2	Auftreten der Fehlerursache ist gering, genauer Prozess	100 / 50	3 / 2	Entdecken der aufgetretenen Fehlerursache ist sehr wahrscheinlich, Prüfungen sind sicher, z.B. mehrere voneinander unabhängige Prüfungen	99,90%	3 / 2
sehr gering		**sehr gering**			**sehr hoch**		
Sehr geringe Funktionsbeeinträchtigung, nur vom Fachpersonal erkennbar	1	Auftreten der Fehlerursache ist unwahrscheinlich	1	1	Aufgetretene Fehlerursache wird sicher entdeckt	99,99%	1

Auswertung

	1.Bewertung	2.Bewertung
1	2	2
2	2	2
3	20	10
4	24	12
5	48	32
6	36	24
7	36	24
8	162	54
9	18	6
10	240	80
11	48	24
12	72	48
13	90	60
14	40	20
15	60	40
16	12	12
17	90	60
18	40	20
19	18	12
20	45	30
21	196	56
22	168	84
23	168	84
24	168	84
25	168	56
26	168	84
27	30	20
28	160	96
29	160	80
30	30	20
31	160	96
32	160	96
33	160	96
34	30	20
35	100	60
36	30	20
37	100	60
38	100	50
39	30	20
40	160	96
41	160	96
42	160	80
43	20	0
44	80	0
45	48	0
46	20	0
47	64	0
48	80	0



Maximal Risiko 1.Bewertung	240
Maximal Risiko 2.Bewertung	96

Seite 12

Abbildungsverzeichnis

2.1 Vorgehensmodell: V-Modell . . . 10
2.2 Grafisch aufbereiteter Teilausschnitt des Projektplans . . . 15

3.1 Schematische Darstellung der Simulationsumgebung . . . 19
3.2 Breitenangaben zur Autobahn mit Standstreifen . . . 20
3.3 Breitenangaben zur Autobahn mit Beschleunigungs- oder Verzögerungsstreifen . 21
3.4 Breitenangaben zur Auf- beziehungsweise Abfahrt einer Autobahn . . . 21
3.5 Längenangaben zu Beschleunigungs- beziehungsweise Verzögerungsstreifen . . . 23
3.6 Angaben zu Kurvenradien . . . 23
3.7 Schilderpositionierung . . . 24
3.8 ViDAs Architektur . . . 26
3.9 Funktionsweise der High Level Architecture . . . 28

4.1 Aufbau des SILAB-Fahrsimulators . . . 32
4.2 Grundlegender Aufbau einer DPU . . . 33
4.3 Übersicht der ViDAs-Umweltschnittstelle . . . 35
4.4 Am Fahrzeug montierte Sensorik . . . 36
4.5 Ausgänge der RPUFahrzeugsensor . . . 36
4.6 Relevante Fremdfahrzeuge im Sensorbereich . . . 37
4.7 Ausgänge der RPUSchildsensor . . . 38
4.8 Angaben zu Kurvenradien . . . 39
4.9 Ausgänge der RPUKurvensensor . . . 40
4.10 Ausgänge der RPUMarkierungssensor . . . 40
4.11 Übersicht der Benutzerschnittstelle . . . 41
4.12 Ausgänge der RPUAnzeigen . . . 43
4.13 Schematische Darstellung des HLA-Komm-Moduls . . . 45
4.14 Schematische Darstellung der Funktionsweise der HLA-Kommunikation . . . 46
4.15 Ein- und Ausgänge des ViDAs-Kerns . . . 49
4.16 Schematische Darstellung der ViDAs-Kern-DPU . . . 50
4.17 ViDAs-Kern Architektur . . . 52

5.1 Legende Test-Datenbank-Webinterface/Übersicht Datenbank-Inhalt . . . 72
5.2 Ausschnitt aus der Testdatenbank . . . 72
5.3 Prinzip der automatischen Matlab-Modelltests . . . 76
5.4 Prinzip der Kommunikationstests . . . 79
5.5 Vorgehen bei automatischen Tests . . . 86

6.1 Architektur des verwendeten System-on-Chip . . . 89
6.2 Schematische Darstellung der Arbeitsabläufe . . . 91
6.3 Grundlegende Einstellungen des *Real-Time Workshop* . . . 95
6.4 Interface-Einstellungen des *Real-Time Workshop* . . . 96
6.5 Hauptfenster des grafischen Konfigurationsmenüs `xconfig` . . . 97
6.6 Konfigurationsmenü des Prozessors, der FPU und des Cache-Systems . . . 98
6.7 Xilinx ISE . . . 99
6.8 ISE Impact . . . 100
6.9 Xilinx ML509 FPGA-Board: Dip-Schalter der Konfigurationsadresse/-modus . . . 101
6.10 Bezeichnungen von Ausführungszeiten . . . 103
6.11 Grafiken zur Verdeutlichung des Vorgehens beim Messen . . . 105
6.12 Teile einer statischen Analyse . . . 107
6.13 Phasen der WCET-Berechnung mit aiT . . . 108
6.14 Can-Schedule . . . 114
6.15 Häufigkeitsverteilung der gemessenen Ausführungszeiten . . . 116
6.16 Schedule für die ECU . . . 118

A.1 Automat Speicher_Überholverbot . . . 129
A.2 ViDAs-Kern . . . 130
A.3 Übersicht ViDAs-Kern . . . 131
A.4 Übersicht ACC . . . 132
A.5 Submodul Wunschgeschwindigkeitsauswertung . . . 133
A.6 Automat Wunschgeschwindigkeit . . . 134
A.7 Automat Bedienelemente_programmieren . . . 135
A.8 Submodul Schildauswertung . . . 136
A.9 Automat prädiktive_Streckenberechnung . . . 137
A.10 Automat Auswertung_Schilder . . . 138
A.11 Submodul Auswertung_Entfernung . . . 139
A.12 Submodul Entfernungsberechnung . . . 140
A.13 Submodul Kurvenauswertung . . . 141
A.14 Automat Auswertung_Kurvenradien . . . 142
A.15 Submodul Auswertung_Entfernung . . . 143
A.16 Submodul Entfernungsberechnung . . . 144
A.17 Submodul Geschwindigkeitslogik . . . 145
A.18 Automat Entscheider . . . 146
A.19 Submodul Beschleunigungsberechnung . . . 147
A.20 Automat Berechnung_Beschleunigung . . . 148
A.21 Automat prädiktive_Geschwindigkeitsberechnung . . . 149
A.22 Submodul Abstandsregler . . . 150
A.23 Submodul Geschwindigkeitsregler . . . 151
A.24 Submodul Geschwindigkeitsregler . . . 152
A.25 Submodul AntiRuck . . . 153
A.26 Automat AntiRuck . . . 154
A.27 Submodul Beschleunigungumrechnung . . . 155

A.28 Übersicht LCA 156
A.29 Submodul Auswertung_Markierungen 157
A.30 Automat Fahrspurauswertung 158
A.31 Submodul Auswertung_Ausfahrtsschilder 159
A.32 Automat Ausfahrtsschilderauswertung 160
A.33 Submodul Auswertung_Fremdfahrzeuge 161
A.34 Automat Fremdfahrzeugauswertung 162
A.35 Submodul Auswertung_Verbote 163
A.36 Automat Verbotauswertung 164
A.37 Übersicht Multiplexer 165
A.38 Begriffskonvention: Autobahn mit Abfahrt 166
A.39 Begriffskonvention: Autobahn mit Auffahrt 166
A.40 Begriffskonvention: hinter, neben, vor 167

Tabellenverzeichnis

3.3 Verteilung der Funktionalitäten auf die Submodule des ViDAs-Kerns 27
3.5 Beispiel einer Nachricht 29

4.1 Zuordnung der Bedienelemente zu Funktionstasten 42
4.3 Bedeutung der Werte von Ausgabesignalen 44

5.1 Bewertung von Auftretenswahrscheinlichkeit und Bedeutung möglicher Fehler . . 74
5.2 Bewertung der Entdeckungswahrscheinlichkeit möglicher Fehler 74
5.3 Abdeckung der LCA-Anforderungen mit Testfällen 75
5.4 Abdeckung der ACC-Anforderungen mit Testfällen 77
5.5 Protokoll des Code-Reviews 79
5.6 Abdeckung der Testfälle in den Szenarien 80

6.1 Konfigurationsadresse/-modus 101
6.2 Aufbau einer CAN-Nachricht 110
6.3 Nachrichtenlängen 111
6.4 Übertragungsdauer von allen Nachrichten einer Periode 112
6.5 Status-Nachricht 112
6.6 Status-Codes der Status-Nachricht 113
6.7 Errechnete durchschnittliche Ausführungszeiten 117
6.8 Verzögerungswerte für die Annotation im Matlab-Modell 120

D Konstanten

Dieser Abschnitt enthält eine Übersicht über die im Dokument verwendeten Konstanten.

Konstante	Beschreibung	Wert
$a_{acc,max}$	Maximale Beschleunigung des ACC innerhalb des Komfortbereichs	$2,5\frac{m}{s^2}$
$a_{acc,mnb}$	Maximale, negative Beschleunigung des ACC ausserhalb des Komfortbereichs	$5\frac{m}{s^2}$
a_{acc,max_lat}	Maximale laterale Beschleunigung des ACC	$2,0\frac{m}{s^2}$
$a_{acc,min}$	Minimale Beschleunigung des ACC	$-3,5\frac{m}{s^2}$
$d_{linie\_fb,fahrbahn}$	Abstand der Fahrbahnbegrenzungslinie vom Fahrbahnrand	$0,45m$
$d_{schild,linie\_fb}$	Abstand der Schilder über der Fahrbahn von der Fahrbahnbegrenzungslinie	$1,875m$
$d_{schild,seite}$	Abstand der seitlichen Schilder vom Fahrbahnrand	$1,5m$
$d_{seite,min}$	Minimaler, seitlicher Abstand zwischen Fahrzeugen	$1,0m$
$j_{acc,max}$	Maximaler Ruck des ACC	$3,5\frac{m}{s^3}$
$j_{acc,min}$	Minimaler Ruck des ACC	$-2,5\frac{m}{s^3}$
r_{min}	Minimaler Kurvenradius	$900m$
$s_{hinten,max}$	Maximale Reichweite der Abstandssensorik nach hinten	$120m$
$s_{hinten,min}$	Minimale Reichweite der Abstandssensorik nach hinten	$15m$
$s_{lim,max}$	Maximale Reichweite der Sensorik zur Verkehrsschilderkennung	$200m$
$s_{lim,seite}$	Maximale Reichweite der Sensorik zur Verkehrsschilderkennung zur Seite	$32m$
$s_{seite,max}$	Maximale Reichweite der Abstandssensorik zur Seite	$5,8m$
$s_{seite,min}$	Minimale Reichweite der Abstandssensorik zur Seite	$1,0m$
$s_{spur,max}$	Maximale Reichweite der Sensorik zur Fahrspurerkennung nach vorne	$200m$
$s_{spur,seite}$	Maximale Reichweite der Sensorik zur Fahrspurerkennung zur Seite	$30m$
$s_{vorne,max}$	Maximale Reichweite der Abstandssensorik nach vorne	$200m$
$s_{vorne,min}$	Minimale Reichweite der Abstandssensorik nach vorne	$15m$
t_{reakt}	Obere Schranke für Reaktionszeit des Systems auf Sensordaten	$50ms$
t_{luecke}	Einzuhaltende Zeitlücke zwischen zwei Fahrzeugen (Zwei-Sekunden-Abstand)	$2s$
v_{acc,max_kurve}	Maximale Geschwindigkeit des ACC in Kurven	$152,74\frac{km}{h}$
$v_{acc,min}$	Minimale Geschwindigkeit des Ego-Fahrzeugs (ACC)	$30\frac{km}{h}$
$v_{acc,schrittweite}$	Schrittweite der Einstellungen beim Programmieren der individuellen Geschwindigkeitswahl-Bedienelemente	$5\frac{km}{h}$
$v_{ego,max}$	Maximale Geschwindigkeit des Ego-Fahrzeugs	$160\frac{km}{h}$
$v_{fremd,max}$	Maximale Geschwindigkeit von Fremdfahrzeugen	$200\frac{km}{h}$
$v_{fremd,min}$	Minimale Geschwindigkeit von Fremdfahrzeugen	$30\frac{km}{h}$
$v_{lca,min}$	Minimale Geschwindigkeit des Ego-Fahrzeugs (LCA)	$60\frac{km}{h}$
v_{lim,min_init}	Minimale, initiale Geschwindigkeitsbegrenzung	$120\frac{km}{h}$
$w_{fahrbahn,max}$	Maximale Breite der Fahrbahn (incl. Beschleunigungsstreifen)	$13,05m$
w_{ego}	Breite des Ego-Fahrzeugs	$2m$
w_{linie_aa}	Breite der Auf- und Abfahrtslinie	$0,3m$
w_{linie_fb}	Breite der Fahrbahnbegrenzungslinie	$0,3m$
w_{linie_l}	Breite der Leitlinie	$0,15m$
w_{schild}	Höhe und Breite der Verkehrsschilder	$0,9m$
w_{spur}	Breite einer Fahrspur	$3,75m$

E. Glossar

ACC

Adaptive Cruise Control. Auch: Abstandsregeltempomat, Distronic (Mercedes-Benz). Bietet die Funktionalität eines gewöhnlichen Tempomats und hält zusätzlich den Sicherheitsabstand zum Vorderfahrzeug ein.

Beschleunigungsstreifen

Auch: Beschleunigungsspur. Siehe Abbildung A.38 oder A.39.

Code of Practice for the Design and Evaluation of Advanced Driver Assistance Systems

Leitfaden zur Erstellung und zum Test von Fahrerassistenzsystemen

DPU

Data Processing Unit. Ein Teilmodul der Fahrsimulationssoftware SILAB, dass über Ein- und Ausgänge verfügt und eine Funktionalität kapselt.

Dynamic-Link Library (DLL)

Dynamic Link Library. Eine dynamische Bibliotheks-Datei, welche kompilierten Programmcode enthält.

Ego-Fahrzeug

Bezeichnet das Fahrzeug, in dem sich das Fahrerassistenzsystem befindet bzw. das Fahrzeug, dass vom Benutzer gesteuert wird. Gegensatz: Fremdfahrzeug.

Fahrer

Als Fahrer wird im Allgemeinen die Person bezeichnet, die ein Fahrzeug führt. Innerhalb dieses Dokuments bezieht sich die Bezeichnung Fahrer jedoch im Speziellen auf die Person, die das Ego-Fahrzeugs führt.

Fahrerassistenzsystem

Ein Fahrerassistenzsystem ist eine elektronische Zusatzkomponente in einem modernen Fahrzeug (hier im speziellen in einem PKW), die den Fahrer in bestimmten Situationen unterstützt. Die Unterstützung zielt hauptsächlich auf einen Zugewinn an Sicherheit und Fahrkomfort.

Federation

Eine Federation ist ein Zusammenschluss von *HLA*-Kommunikationsteilnehmern, die über eine *RTI* kommunizieren können.

Fehlermöglichkeits- und Einflussanalyse

Methode zur Durchführung einer Sicherheitsanalyse

Geisterfahrer

Als Geisterfahrer bzw. Falschfahrer wird im Straßenverkehr einen Fahrer bezeichnet, der eine Autobahn oder sonstige Straße mit getrennten Richtungsfahrbahnen entgegen der vorgeschriebenen Fahrtrichtung befährt.

High Level Architecture (HLA)

High-Level Architecture ist eine Kommunikationsschnittstelle für Komponenten auf verschiedenen Rechnern. Die Kommunikation erfolgt per Ethernet. Es gibt Bibliotheken für verscheidene Programmiersprachen.

LCA

Lane Change Assistant. Auch: Fahrspurwechselassistant, SideAssist (Audi,VW). Warnt bei einem Überholmanöver vor herrannahenden oder im toten Winkel befindlichen Fremdfahrzeugen.

Linienführung

Verlauf einer Straße durch die Landschaft. Von konkreten Maßen der Straße wird abstrahiert.

response 3

Ein Teilprojekt des aus EU-Mitteln gefördeten und von der Daimler-Chrysler AG koordinierten Projekts „Preventive and Active Safety Applications“ (preVENT)

Risikoprioritätszahl

Gibt in einer Sicherheitsanalyse die Höhe des Sicherheitsrisikos an. Wird aus dem Produkt der Bewertungen von Auftretenswahrscheinlichkeit, Bedeutung und Entdeckungswahrscheinlichkeit eines möglichen Systemfehlers berechnet.

Ruby Programming Unit (RPU)

Ruby Programming Unit. Bezeichnung für eine in der Skriptsprache Ruby geschriebene DPU.

Run-Time Infrastructure (RTI)

Die Run-Time Infrastructure ist ein Serverdienst, der die Kommunikation zwischen den Teilnehmern einer *Federation* koordiniert und für die Zeitsynchronisierung zuständig ist.

Sicherheitsabstand

Sicherheitsabstand bezeichnet im Straßenverkehr den Abstand, den Verkehrtsteilnehmer seitlich und nach vorne zu anderen Verkehrsteilnehmern einhalten sollten um Gefahrsituationen zu vermeiden. Der Sicherheitsabstand nach vorne zum Beispiel, muss so großsein, dass bei einem plötzlichen Abbremsen des Vorderfahrzeugs noch ein rechtzeitiges Bremsen des eigenen Fahrzeugs möglich ist, ohne dass es zu einem Auffahren kommt.

Sicherheitsanalyse

Vorgehen, um in Sicherheitskritischen Systemen Fehlermöglichkeiten zu entdecken, deren Auftretenswahrscheinlichkeit, Bedeutung für die Sicherheit des Systems und Entdeckungswahrscheinlichkeit zu bewerten, und daraus ein Sicherheitsrisiko abzuleiten.

SILAB

Simulation Laboratory. Ein Fahrsimulationssystem vom Würzburger Institut für Verkehrswissenschaften.

Verzögerungsstreifen

Auch: Verzögerungsspur. Siehe Abbildung A.38 oder A.39.

ViDAs

Virtual Driver Assistant

ViDAs-Komm

Das ViDAs-Komm-Modul stellt die Kommunikation zwischen dem ViDAs-Kern und der ViDAs-Umweltschnittstelle her. Dieses Modul abstrahiert von der physikalischen Realisierung des Kommunikationkanals zwischen den beiden anderen Modulen.

Wunschgeschwindigkeit

Geschwindigkeit, die der Fahrer mit dem Ego-Fahrzeug fahren möchte.

F. Literaturverzeichnis

[Abs] AbsInt Angewandte Informatik GmbH: *aiT Worst-Case Execution Time Analyzers*. http://www.absint.com/ait/

[Aer] Aeroflex Gaisler AB: *Aeroflex-Gaisler*. http://www.gaisler.com/

[ANS] ANSI American National Standards Institute: *ISO 15622 Transport information and control systems - Adaptive Cruise Control Systems - Performance requirements and test procedures*. 2002

[FH04] Ferdinand, Christian ; Heckmann, Reinhold: aiT: worst case execution time prediction by static program analysis. In: *IFIP Congress Topical Sessions*, 377-384

[for] *ForwardSim HLA Toolbox*. http://www.forwardsim.com/en/prod_hlat.php

[For08] Forschungsgesellschaft für Straßen- und Verkehrswesen, Arbeitsgruppe Straßenentwurf: *Richtlinien für die Anlage von Autobahnen*. Ausgabe 2008. Juni 2008

[Gie90] Giesa, Siegfried: *Hinweise für das Anbringen von Verkehrszeichen und Verkehrseinrichtungen*. 8. Auflage. Kirschbaum Verlag, 1990

[Gie02] Giesa, Siegfried: *Hinweise für das Anbringen von Verkehrszeichen und Verkehrseinrichtungen*. 12. Auflage. Kirschbaum Verlag, 2002

[git] *About Git*. http://git-scm.com/about

[Law] Lawicel AB: *CAN232 - The Original CAN to RS232 converter*. http://www.can232.com/

[OFF] OFFIS e.V.: *IMoST-Projekt*. http://www.offis.de/en/offis_in_portrait/structure/projects/view/detail/status/imost.html

[pit] *Pitch pRTI*. http://www.pitch.se/products/prti

[Pro10a] Projektgruppe ViDAs: *Anforderungsdefinition*. März 2010. http://www.vidas.informatik.uni-oldenburg.de

[Pro10b] Projektgruppe ViDAs: *Angepasste Aufgabenstellung*. April 2010. http://www.vidas.informatik.uni-oldenburg.de

[Pro10c] Projektgruppe ViDAs: *Architektur- und Schnittstellendokument*. August 2010. http://www.vidas.informatik.uni-oldenburg.de

[Pro10d] Projektgruppe ViDAs: *Fehlermöglichkeits- und Einflussanalyse*. November 2010

[Pro10e] Projektgruppe ViDAs: *Testbericht*. November 2010. http://www.vidas.informatik.uni-oldenburg.de

[Res06] Response 3: *Code of Practice for the Design and Evaluation of Advanced Driver Assistance Systems.* Version number 3.0. Oktober 2006. http://ip-prevent.org/en/public_documents/deliverables/d112_code_of%_practice_for_the_design_and_evaluation_of_adas.htm

[rub] *Programmiersprache Ruby.* http://www.ruby-lang.org/de/

[V-M] *V-Modell XT.* http://www.v-modell-xt.de/

[WEE+08] Wilhelm, Reinhard ; Engblom, Jakob ; Ermedahl, Andreas ; Holsti, Niklas ; Thesing, Stephan ; Whalley, David ; Bernat, Guillem ; Ferdinand, Christian ; Heckmann, Reinhold ; Mitra, Tulika ; Mueller, Frank ; Puaut, Isabelle ; Puschner, Peter ; Staschulat, Jan ; Stenström, Per: The worst-case execution-time problem–overview of methods and survey of tools. In: *ACM Trans. Embed. Comput. Syst.* 7 (2008), May, 36:1–36:53. http://dx.doi.org/http://doi.acm.org/10.1145/1347375.1347389. – DOI http://doi.acm.org/10.1145/1347375.1347389. – ISSN 1539–9087

[WHW09] Winner, Hermann ; Hakuli, Stephan ; Wolf, Gabriele: *Handbuch Fahrerassistenzsysteme.* Vieweg+Teubner Verlag / GWV Fachverlage GmbH, Wiesbaden http://ebooks.ub.uni-muenchen.de/17963/

[wiv] *Würzburger Institut für Verkehrswissenschaften GmbH.* http://www.wivw.de

[Wür09] Würzburger Institut für Verkehrswissenschaften GmbH: *Dokumentation der Fahrsimulationssoftware SILAB.* Version 2.5. Januar 2009

[Xil] Xilinx, Inc.: *FPGA and CPLD Solutions from Xilinx, Inc.* http://www.xilinx.com/

[Xil09] Xilinx: *ML507 Evaluation Platform User Guide.* http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf. Version: 3.1.1, Oktober 2009